TECHNISCHE UNIVERSITÄT
CHEMNITZ

Strategische Planung technischer Kapazität in komplexen Produktionssystemen: mathematische Optimierung grafischer Modelle mit der Software AURELIE

Dissertation
zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

Herr Dipl.-Inf. Christian Andreas Hochmuth, MBA
geboren am 18. Mai 1984 in Chemnitz

Fakultät für Informatik
an der Technischen Universität Chemnitz

Gutachter:	Prof. Dr.-Ing. Martin Gaedke
	Prof. em. Dr. oec. Dr. rer. nat. habil. Peter Köchel
	Prof. Dr.-Ing. Jörg Lässig
Tag der Verteidigung:	27. Februar 2020

Bibliografische Information

Hochmuth, Christian Andreas. 2020. »Strategische Planung technischer Kapazität in komplexen Produktionssystemen: mathematische Optimierung grafischer Modelle mit der Software AURELIE« Dissertation, Technische Universität Chemnitz, Fakultät für Informatik.

231 Seiten, 70 Abbildungen, 12 Tabellen, 19 Algorithmen, 216 Quellen.

Für Alexandra und Lasse

Vorwort

Die vorliegende Dissertation ist an der Fakultät für Informatik der Technischen Universität Chemnitz entstanden. Im Mittelpunkt dieser Arbeit steht das Grundkonzept der Software AURELIE, die im Zuge der Promotion bei der Bosch Rexroth AG entwickelt und eingeführt wurde. Die Veröffentlichung markiert den Schlusspunkt eines langen Prozesses und bietet Gelegenheit für einen Rückblick. Der Natur einer externen Promotion entsprechend haben mich Personen aus Forschung und Praxis auf diesem Weg begleitet.

Zunächst möchte ich meine Förderer aus dem universitären Umfeld hervorheben. Ihre wissenschaftliche Anleitung half mir, die in der Dissertation vorgestellten Ideen und Konzepte in publikationsgerechter Form zu strukturieren. Ich danke Prof. Dr.-Ing. Martin Gaedke für wertvolle Hinweise und für die Bereitschaft, die Betreuung der Arbeit zu übernehmen. Ebenso gilt mein Dank Prof. em. Dr. oec. Dr. rer. nat. habil. Peter Köchel, der mein Promotionsvorhaben von Beginn an fachlich und persönlich unterstützt hat. Des Weiteren danke ich Prof. Dr.-Ing. Jörg Lässig für die Begutachtung und für die lehrreiche Zusammenarbeit bei der Erstellung von Veröffentlichungen.

Ein wichtiges Merkmal dieser Arbeit ist, dass der entwickelte Lösungsansatz erfolgreich in der Praxis umgesetzt werden konnte. Hieran haben meine Betreuer auf der Seite der Bosch Rexroth AG sowie die Fachexperten im Kollegenkreis großen Anteil. In diesem Zusammenhang danke ich Dr.-Ing. Michael Sauter und Dr.-Ing. Bernd A. Müller für ihre Mentorenschaft und das Öffnen vieler verschlossener Türen. Alessandro Battino und Torsten Wassum danke ich für ihre Unterstützung bei der Spezifikation, dem Test und der Einführung der Software. Lukas Hahmann gilt mein Dank für seinen Einsatz im Rahmen von Anwenderschulungen und der technischen Dokumentation der Algorithmen.

Darüber hinaus sei allen in der gebotenen Kürze nicht genannten Kollegen, Familienmitgliedern und Freunden mein tiefer Dank versichert.

Düsseldorf, im Mai 2020 Christian A. Hochmuth

Referat

Aktuelle Entwicklungen erfordern die Flexibilisierung von Produktionssystemen, wodurch die Komplexität insbesondere in der variantenreichen Serienfertigung steigt. Als Folge bestehen erhebliche Herausforderungen darin, die technische Kapazität in solchen komplexen Produktionssystemen effizient, transparent und flexibel zu planen. Um diese Herausforderungen zu bewältigen, wurde im Zusammenhang mit dieser Arbeit die Software AURELIE entwickelt und erfolgreich an den Standorten der Bosch Rexroth AG eingeführt. Im Folgenden wird das Grundkonzept der Software dargelegt, welches sich aus den kalkulatorischen Grundlagen und den entwickelten Kernalgorithmen zusammensetzt.

Im Fokus stehen aus strategischer Sicht die maximalen Kapazitäten, die minimalen Investitionen zur Herstellung der geplanten Stückzahlen und die optimale Auslastung der Produktionsanlagen. Den Rahmen für die Optimierung der Planungsziele bildet ein System von Wertströmen, welches alle Prozesse zur Fertigung und Montage von Produkten an einem Standort umfasst. Die Komplexität resultiert aus der Struktur solcher Systeme, beruhend auf der rekursiven Verknüpfung von Prozessschritten und der mehrfachen Belegung von Produktionsanlagen. Diese Verknüpfungen und die begrenzte Auslastung der Produktionsanlagen führen in der Planung zu wechselseitigen Abhängigkeiten.

Um die angesprochenen Abhängigkeiten zu berücksichtigen, werden in Unternehmen Ansätze verfolgt, die sich in ein Spektrum zwischen zwei unterschiedlichen Fällen einordnen lassen: Im ersten Fall werden einfache Modelle erstellt, die nur sehr begrenzt die Komplexität abbilden, jedoch von allen Beteiligten im Planungsprozess verstanden werden. Im zweiten, gegenteiligen Fall werden in aufwändiger Arbeit umfangreiche Modelle entwickelt, die zwar der Komplexität Rechnung tragen, allerdings nicht verständlich sind. Der Grund dafür ist, dass dem Planer die Aufgabe aufgebürdet wird, die Abhängigkeiten in mathematische Ausdrücke zu überführen. Als Konsequenz ist in der Praxis zu beobachten, dass sich Vollständigkeit und Verständlichkeit der Modelle ausschließen.

Zur Lösung dieses Zielkonflikts wird ein softwaregestützter Workflow vorgeschlagen, welcher auf der neuen Software AURELIE beruht. Im ersten Schritt erstellt der Planer mit Hilfe der Software ein grafisches Modell, das nicht nur verständlich ist, sondern auch das System in seiner Komplexität vollständig widerspiegelt. Im zweiten, automatisch ablaufenden Schritt validiert die Software das grafische Modell und transformiert dieses in ein mathematisches Modell. Im dritten, ebenso automatisierten Schritt optimiert die Software das mathematische Modell, worauf der Planer das Modell anpasst und der Workflow von Neuem beginnt. Das Ergebnis dieser Teilautomatisierung ist eine signifikante Steigerung der Effizienz, Transparenz und Flexibilität im Planungsprozess.

Im Zuge einer Systemanalyse werden zuerst Struktur und Schnittstelle eines Systems von Wertströmen beschrieben, wofür ein spezifischer mathematischer Formalismus erarbeitet wird. Im Anschluss werden die kalkulatorischen Grundlagen dargelegt und die Anforderungen an eine Software abgeleitet, welche sich zur Modellierung und Optimierung eines solchen Systems eignet. Mit der Bewertung des Stands der Technik folgt der Beleg, dass zum Zeitpunkt der Betrachtung keine geeignete Software existierte. Als ein wesentlicher Beitrag neben der Systemanalyse werden danach die Kernalgorithmen zur grafischen Modellierung, Modelltransformation und mathematischen Optimierung beschrieben.

Abstract

Recent developments require a transition to more flexible production systems, which causes higher complexity, especially in the case of series production with a great number of variants. As a result, the efficient, transparent and flexible planning of the technical capacity in such complex production systems represents considerable challenges. In order to overcome these challenges, the software AURELIE was developed in the context of writing this dissertation and successfully implemented at the locations of Bosch Rexroth AG. In the following, the fundamental concept of the software is described, which comprises the foundations with respect to calculations and the developed core algorithms.

From a strategic perspective, the focus lies on the maximal capacities, the minimal investments which are required to produce the planned quantities and the optimal utilization of the production facilities. The scope for optimizing the planning objectives is defined by a system of value streams, which encompasses all processes for the manufacturing and assembly of products at a specific location. The complexity results from the structure of such systems, which is based on the recursive combination of process steps and the multiple allocation of production facilities. These combinations and the limited utilization of the production facilities lead to interdependencies in the planning.

To account for the aforementioned interdependencies, companies follow approaches which can be classified along a spectrum between two distinct cases: In the first case, simple models are being created, which are very limited regarding the mapping of the complexity, but which are understood by all participants involved in the planning process. In the second, contrary case, extensive efforts are made to develop comprehensive models, which take the complexity into account, but which are not understandable. The underlying reason is that the task to transform the interdependencies into mathematical expressions is imposed upon the planner. As a consequence, it can be observed in practice that completeness and understandability of the models are mutually exclusive.

With the aim to solve this conflict of objectives, a software-based workflow is proposed, which is supported by the newly developed software AURELIE. In the first step, the planner is aided by the software in creating a graphical model, which is not only understandable, but also reflects the system completely with respect to its complexity. In the second step, which is executed automatically, the software validates the graphical model and transforms it into a mathematical model. In the third step, which is automated as well, the software optimizes the mathematical model, whereupon the planner adjusts the model and the workflow starts from the beginning. The outcome of this partial automation is a significant improvement of efficiency, transparency and flexibility in the planning process.

In the course of a system analysis, firstly the structure and the interface of a system of value streams are described, utilizing a specifically developed mathematical formalism. Subsequently, the foundations of the calculation are explained and the requirements are derived which a software must fulfill in order to be suitable for modeling and optimization of such a system. By evaluating the state of the art, it is then proven that at the time of the review a suitable software did not exist. As a major contribution in addition to the system analysis, after that the core algorithms for graphical modeling, model transformation and mathematical optimization are described.

Inhaltsverzeichnis

Abbildungsverzeichnis

70 Abbildungen (und Unterabbildungen)

Tabellenverzeichnis

12 Tabellen

Algorithmenverzeichnis

19 Algorithmen

1 Einführung

Im Fokus dieser Arbeit steht ein neuer Ansatz zur strategischen Planung der technischen Kapazität in komplexen Produktionssystemen. Dieser Ansatz, welcher auf der Weiterentwicklung von Methoden der grafischen Modellierung und der mathematischen Optimierung basiert, wurde in der Software AURELIE umgesetzt. Im Verlauf der Arbeit wird das Grundkonzept der Software vorgestellt, welches die Grundlagen zur Kalkulation der Planungsziele und die Kernalgorithmen einschließt. In Abschnitt 1.1 wird die Ausgangssituation dargelegt, worauf in Abschnitt 1.2 die Problembeschreibung und die Einordnung der Dissertation folgen. Danach wird in Abschnitt 1.3 der Lösungsansatz skizziert, von welchem in Abschnitt 1.4 die Forschungsfragen und der Aufbau der Arbeit abgeleitet werden.

1.1 Ausgangssituation: Potenziale in der Planung

Seit einigen Jahren wird der Wettbewerb zwischen produzierenden Unternehmen durch die Individualisierung der Produkte und die Verkürzung der Produktlebenszyklen bestimmt. Unternehmen sehen sich mit der Notwendigkeit konfrontiert, ihre Prozesse von der Produktentwicklung, der Beschaffung und der Produktion bis zum Versand und dem Aftersales zu beschleunigen. Hinzu kommen neue Entwicklungen, die unter den Begriffen Digitalisierung, Industrie 4.0 und Industrial Internet zusammengefasst werden können. Die Begriffe stehen für den Einsatz von Algorithmen, welche die Möglichkeit eröffnen, große Datenmengen zu verarbeiten und Entscheidungsprozesse zu automatisieren. Als Folge ist in Unternehmen eine Flexibilisierung aller Prozesse festzustellen.

Die angesprochene Flexibilisierung besitzt Auswirkungen auf die Produktion und in noch größerem Maße auf die variantenreiche Serienfertigung. Allgemein wird die Serienfertigung von der Einzelfertigung dadurch abgegrenzt, dass größere Stückzahlen jeweils einer Produktvariante hergestellt werden. Die Variantenvielfalt bezieht sich zum einen darauf, dass zur Herstellung verschiedener Varianten unterschiedliche Prozessschritte ausgeführt werden. Zum anderen ist es häufig möglich, dasselbe Produkt mit Hilfe verschiedener Prozessschritte zu fertigen oder zu montieren. Darüber hinaus werden die Produktionsanlagen in vielen Fällen mehrfach belegt, um Prozessschritte zur Herstellung eines oder mehrerer Produkte auszuführen. Das Ergebnis dieser Variantenvielfalt ist eine erhebliche Komplexität, welche durch die genannten Trends noch zusätzlich gesteigert wird.

Diese Komplexität stellt die Planung, Steuerung und Ausführung der Prozesse in der Produktion vor immense Herausforderungen. In der vorliegenden Arbeit wird die strategische Planung der technischen Kapazität thematisiert und eine Lösung zur Bewältigung dieser Herausforderungen entwickelt. Die Planungsziele bestehen in der Bestimmung der maximal erreichbaren Kapazitäten, der minimal notwendigen Investitionen und der optimalen

Auslastung aller Produktionsanlagen. Der Bezugsrahmen ist jeweils durch ein System gegeben, welches alle Wertströme zur Fertigung und Montage der Produkte an einem Standort zusammenfasst. Die Komplexität spiegelt sich in der Struktur des Systems wider und resultiert aus der rekursiven Verknüpfung von Prozessschritten sowie der mehrfachen Nutzung vieler Produktionsanlagen. Diese Aspekte äußern sich in wechselseitigen Abhängigkeiten zwischen den Elementen entsprechender Systeme von Wertströmen.

Zur Erreichung der formulierten Planungsziele ist es notwendig, die existierenden Abhängigkeiten in der Planung mathematisch zu berücksichtigen. Hierzu werden in der Praxis meist Tabellenkalkulationen verwendet, wobei die Ansätze jeweils als Zwischenstufen zweier gegensätzlicher Fälle aufgefasst werden können. Im ersten Fall werden einfache Modelle erstellt, welche die Abhängigkeiten nur unzulänglich abbilden, jedoch den Vorteil bieten, dass die Modelle von allen Beteiligten verstanden werden. Im zweiten, entgegengesetzten Fall werden umfangreiche Modelle entwickelt, die zwar der Komplexität Rechnung tragen, aber nicht verständlich sind. Dieser Widerspruch ist dadurch begründet, dass es in der Verantwortung des Planers liegt, die Abhängigkeiten durch mathematische Ausdrücke geeignet abzubilden. Als Konsequenz verbleiben in allen Fällen Potenziale in Bezug auf die Effizienz, die Transparenz und die Flexibilität in der Planung.

1.2 Problembeschreibung und Einordnung der Dissertation

Problembeschreibung. Untersucht wird die strategische Planung technischer Kapazität in komplexen Produktionssystemen, insbesondere in der variantenreichen Serienfertigung. Das Problem besteht darin, Modelle und Algorithmen zu entwickeln, welche die Modellierung entsprechender Systeme von Wertströmen und die Optimierung der Modelle gemäß den Planungszielen ermöglichen. Hierbei ist die Herausforderung, die Systeme in ihrer gegebenen Komplexität vollständig abzubilden und gleichzeitig durch verständliche Modelle zu repräsentieren. Der Begriff der Verständlichkeit bezieht sich darauf, dass es den Planern möglich sein soll, die Elemente eines Systems und deren Beziehungen auf einfache Weise abzubilden. Das bedeutet, dass die Planer nicht gezwungen sein sollen, die wechselseitigen Abhängigkeiten zwischen den Elementen in mathematische Ausdrücke zu überführen. Die Entlastung von dieser Aufgabe verspricht die Schöpfung der zuvor genannten Potenziale: die Steigerung der Effizienz, Transparenz und Flexibilität.

Um die Planer entsprechend von der Erstellung mathematischer Modelle zu befreien, wird das Konzept eines softwaregestützten Workflows entwickelt. Der Workflow basiert auf der neuen Software AURELIE, deren Einsatzzweck darin besteht, komplexe Produktionssysteme grafisch zu modellieren und die Modelle mathematisch zu optimieren. Die Software wurde im Dialog mit Fachexperten und Führungskräften der Bosch Rexroth AG vom Autor dieser Arbeit eigenständig entwickelt. Nach Abschluss der Entwicklung beschloss die Leitung des Unternehmens, diese zum Standard für die strategische Planung der technischen Kapazität zu erklären und in der Folge im Unternehmen einzuführen. Aktuell wird die Software im weltweiten Produktionsnetzwerk an mehr als 40 Standorten auf regelmäßiger Basis eingesetzt. Wie die Planer bestätigen, konnten dadurch die Effizienz, Transparenz und Flexibilität in der Planung signifikant gesteigert werden.

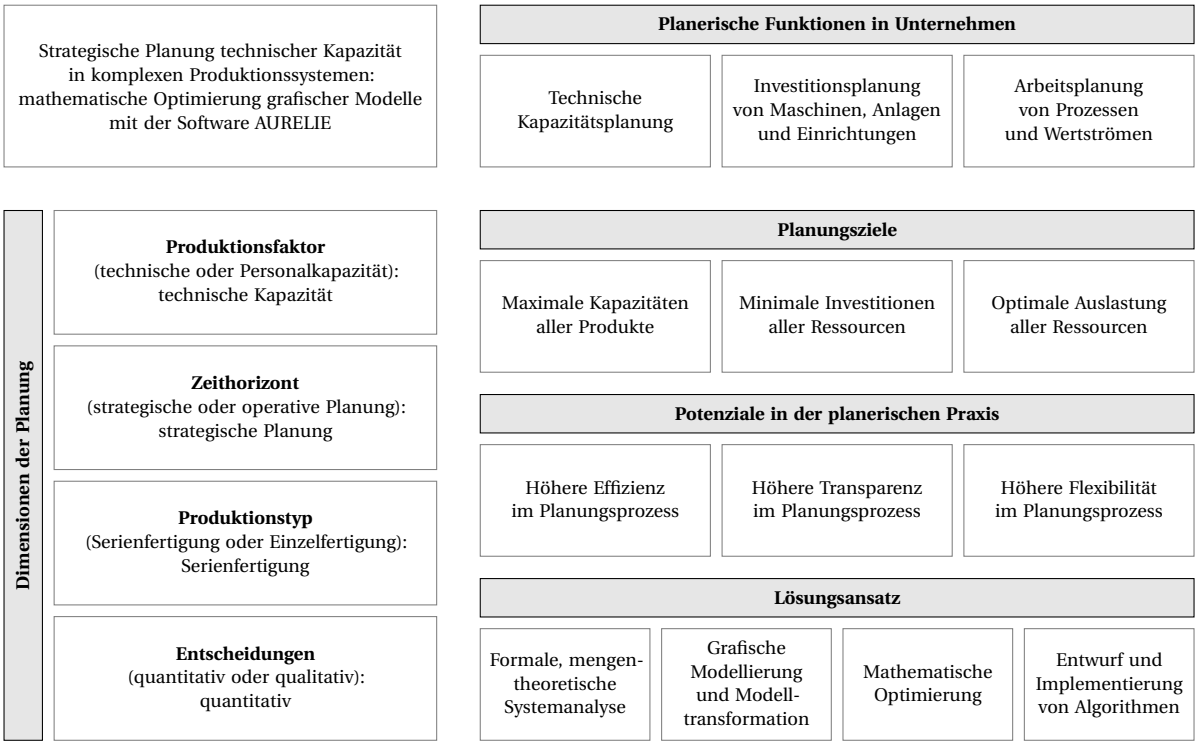


Abbildung 1.1: Praktische und methodische Einordnung der Dissertation. Die schematische Darstellung charakterisiert das Thema der Arbeit mit Blick auf die betriebswirtschaftliche Anwendung und den erarbeiteten Lösungsansatz. Im Mittelpunkt steht die Produktion in Unternehmen und hierbei die strategische Planung technischer Kapazität gemäß definierten Zielen. Durch die Software AURELIE werden Effizienz, Transparenz und Flexibilität gesteigert, beruhend auf neuen Algorithmen zur grafischen Modellierung und mathematischen Optimierung.

Einordnung der Dissertation. Bevor der Lösungsansatz beschrieben wird, soll die Arbeit, wie in Abbildung 1.1 dargestellt, praktisch und methodisch eingeordnet werden. Die dort aufgeführten und im Folgenden genannten Begriffe werden im Hauptteil näher erläutert und, wann immer hilfreich, formal eindeutig definiert. Sie werden an dieser Stelle vorweggenommen, um das Vorgehen und die Ergebnisse der Arbeit zu umreißen.

Dimensionen der Planung: Der begrenzende Produktionsfaktor betrifft die *technische Kapazität*, die insbesondere von der Personalkapazität zu unterscheiden ist. Betrachtet werden die Stückzahlen, welche mit den jeweils gegebenen Produktionsanlagen erreichbar sind, wobei die Verfügbarkeit des nötigen Personals vorausgesetzt wird. Der Fokus liegt dabei auf der *strategischen Planung*, welche sich von der operativen Planung durch ihren langfristigen Zeithorizont unterscheidet. Es wird der Produktionstyp einer *Serienfertigung* angenommen, woraus folgt, dass gegenüber einer Einzelfertigung größere Stückzahlen hergestellt werden. Das Ziel ist die Unterstützung *quantitativer Entscheidungen*, die anders als qualitative Entscheidungen auf numerischen Informationen basieren. Indem der Planer Wertströme, Ressourcen usw. festlegt, bildet dieser qualitative Randbedingungen ab, welche den Rahmen zur Automatisierung der Entscheidungen bilden.

Planerische Funktionen in Unternehmen: Als Ergebnis der Arbeit wird das Grundkonzept einer Software entwickelt, die mehrere Planungsfunktionen unterstützt. Zu diesen zählt die *technische Kapazitätsplanung*, welche die Stückzahlen ermittelt, die durch Eigenproduktion erreichbar sind. Diese werden mit dem erwarteten Bedarf verglichen, um die Stückzahlen festzulegen, die durch Eigenproduktion hergestellt werden sollen. Danach folgt im Planungsprozess die *Investitionsplanung*, welche die Investitionen ermittelt, die zur Herstellung der geplanten Stückzahlen benötigt werden. Die *Arbeitsplanung* schafft die Grundlagen für die beiden vorherigen Planungsfunktionen, indem Prozesse und Wertströme erfasst und geplant werden. Hierbei wird die Entscheidung unterstützt, wie die geplanten Stückzahlen der Produkte auf die Prozessschritte in den Wertströmen zu verteilen sind.

Planungsziele: In den genannten Planungsfunktionen werden unterschiedliche Planungsziele verfolgt, die in der Arbeit durch die Formulierung von Anforderungen konkretisiert werden. Das zuerst zu bestimmende Planungsziel betrifft die *maximalen Kapazitäten* zur Herstellung der gegebenen Produkte. Das zweite Planungsziel, welches nach der Festlegung der geplanten Stückzahlen betrachtet wird, besteht in der Ermittlung der *minimalen Investitionen*, die bei bestmöglicher Nutzung aller Ressourcen notwendig sind. Unter Verwendung des Ergebnisses ist zuletzt das dritte Planungsziel, die *optimale Auslastung* aller verfügbaren und zusätzlich zu installierenden Ressourcen, gesucht.

Potenziale in der planerischen Praxis: Mangels der Möglichkeit, die Komplexität realer Produktionssysteme in der Planung zu berücksichtigen, bleiben Potenziale unausgeschöpft. Diese Potenziale beziehen sich darauf, die Effizienz, die Transparenz und die Flexibilität im Planungsprozess zu steigern. Eine Erhöhung der *Effizienz* bewirkt eine geringere Bindung wichtiger Ressourcen, wobei dies insbesondere die notwendige Arbeitszeit des Planers betrifft. Durch eine Steigerung der *Transparenz* wird das Risiko von Fehlern reduziert, während die Nachvollziehbarkeit der erzielten Ergebnisse zunimmt. Eine Erhöhung der *Flexibilität* beschreibt die Fähigkeit, schneller ein spezifisches Modell für einen Standort anzulegen und fortlaufend anzupassen. Da die Realisierung dieser Potenziale nur schwer messbar ist, werden Anforderungen formuliert und deren Erfüllung bewertet.

Lösungsansatz: In dieser Arbeit wird ein softwaregestützter Workflow vorgeschlagen, welcher durch eine weitgehende Automatisierung die Komplexität für den Planer reduziert. Mit einer mengentheoretischen *Systemanalyse* wird zuerst das Ziel verfolgt, eine eindeutige Beschreibung für Systeme von Wertströmen zu erarbeiten. Die Lösung basiert auf der *grafischen Modellierung* des jeweils gegebenen Produktionssystems und der *Transformation* des erstellten grafischen Modells in ein geeignetes mathematisches Modell. Im Anschluss daran folgt die *mathematische Optimierung* des durch Transformation erzeugten mathematischen Modells gemäß den definierten Planungszielen. Zu diesem Zweck werden *Algorithmen* entworfen und implementiert, welche den kalkulatorischen Kern der Software AURELIE bilden. Der Lösungsansatz wird im Folgenden näher beschrieben.

Es wird der Bogen von einem betriebswirtschaftlichen Problem zu einer Lösung gespannt, welche auf Methoden der Informatik zurückgreift und diese weiterentwickelt. Entsprechend enthält die Dissertation Beiträge für Forschung und planerische Praxis.

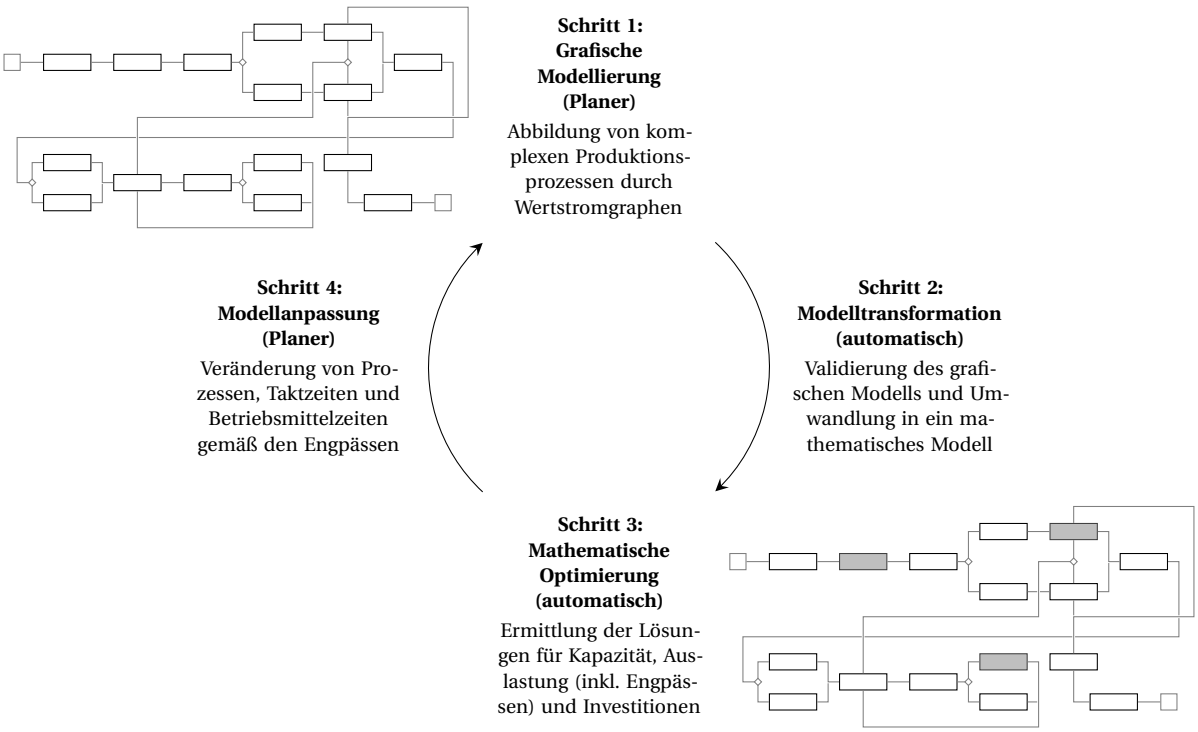


Abbildung 1.2: Softwaregestützter Workflow für die Planung. Die strategische Planung technischer Kapazität verläuft mit der Software AURELIE teilautomatisiert in vier Schritten. Nach der grafischen Modellierung eines Systems von Wertströmen wird das grafische Modell automatisch validiert, transformiert und mathematisch optimiert. Daraufhin analysiert der Planer die ggf. verbleibenden Engpässe in den Wertströmen und überarbeitet die Prämissen der Planung.

1.3 Lösungsansatz: softwaregestützter Workflow

Im Folgenden soll der Lösungsansatz näher beschrieben werden, welcher auf einem softwaregestützten Workflow basiert. Dessen Ablauf gliedert sich in vier Schritte, die entweder teilweise oder vollständig automatisiert werden, wie in Abbildung 1.2 dargestellt ist. Schritte, welche der Planer ausführt, um quantitative Informationen abzubilden und qualitative Randbedingungen zu berücksichtigen, werden teilweise automatisiert. Diese Teilautomatisierung bezieht sich darauf, dass der Planer bei der Modellerstellung durch die Software unterstützt wird. Sobald das Modell vorliegt, werden die nachfolgenden Schritte zur Transformation und Optimierung des Modells automatisch ausgeführt.

Der erste Schritt ist jeweils die *grafische Modellierung* des Produktionssystems, bei welcher der Planer durch die Benutzeroberfläche der Software unterstützt wird. Zur Erstellung des Modells bildet der Planer die Wertströme mit allen nötigen quantitativen Informationen ab, einschließlich Taktzeiten aller Prozessschritte und Betriebsmittelzeiten aller Produktionsanlagen. Dabei beachtet der Planer qualitative Randbedingungen, wie z. B. den vorgegebenen Ablauf der Fertigung und Montage, durch entsprechende Verknüpfung der Prozessschritte. Dem schließt sich der zweite Schritt an, die automatische *Transformation* des grafischen Modells in ein mathematisches Modell, welches sich zur Optimierung eignet. Hervorzuheben

ist hierbei, dass alle wechselseitigen Abhängigkeiten zwischen den Elementen des Systems berücksichtigt werden. Mit dem mathematischen Modell ist es möglich, Zielfunktionen und Nebenbedingungen zur Verfolgung der Planungsziele zu formulieren.

Damit folgt der dritte, ebenso automatisierte Schritt, die *mathematische Optimierung* des umgewandelten Modells. Nach Bestimmung der optimalen Lösungen für die Planungsziele werden die Ergebnisse dargestellt und aufbereitet, worauf der Planer die ggf. verbleibenden Engpässe analysiert. An diese Analyse knüpft der vierte Schritt an, die *Anpassung* der Prämissen des Modells durch den Planer, wie z. B. der Schichtmodelle der Produktionsanlagen. Der Zyklus schließt sich, indem der Planer den ersten Schritt wiederholt und die veränderten Prämissen im grafischen Modell umsetzt. Die automatisierte Modelltransformation befreit den Planer von der Aufgabe, die Komplexität des Produktionssystems in einem mathematischen Modell abzubilden. Durch Algorithmen zur Modellierung und Optimierung werden die Effizienz, die Transparenz und die Flexibilität gesteigert.

1.4 Forschungsfragen und Aufbau der Arbeit

Übergeordnete Forschungsfragen. Die nachfolgenden übergeordneten Forschungsfragen betreffen Formalismen, Modelle und Algorithmen, die zur Umsetzung des softwaregestützten Workflows benötigt werden. Dabei wird vorausgesetzt, dass die Ergebnisse in dieser Form noch nicht existieren und entsprechend Entwicklungsarbeit zu leisten ist. Der Nachweis wird im weiteren Verlauf der Arbeit an der jeweiligen Stelle erbracht.

A Wie lassen sich Systeme von Wertströmen formal eindeutig beschreiben, und welche Anforderungen folgen daraus für die Modellierung und die Optimierung?
B Inwieweit erfüllt aktuell verfügbare, in Unternehmen einsetzbare Software diese Anforderungen, und worin besteht der Entwicklungsbedarf?
C Wie kann ein gegebenes System grafisch modelliert werden, und wie lauten geeignete Algorithmen zur Validierung und Transformation in ein mathematisches Modell?
D Welche Ansätze zur Optimierung sind geeignet, um die Eindeutigkeit der Ergebnisse zu gewährleisten, und wie lassen sich diese in Algorithmen überführen?

Aufbau der Arbeit und detaillierte Forschungsfragen. Der Aufbau der vorliegenden Arbeit ist durch den softwaregestützten Workflow und die Schaffung der hierzu erforderlichen Voraussetzungen bestimmt. Zunächst wird eine Systemanalyse durchgeführt, um auf dieser Grundlage verfügbare Lösungen zu bewerten und eine neue Lösung in Gestalt der Software AURELIE zu entwickeln. Diese Entwicklung gliedert sich in zwei Lösungsschritte, die Modellierung und die Optimierung, entsprechend dem Workflow und den Komponenten der Software. Es folgt ein Überblick der einzelnen Kapitel des Hauptteils und der jeweils adressierten, detaillierten Forschungsfragen.

Kapitel 2, Lösungsvorbereitung: Systemanalyse. Durch die Systemanalyse werden die notwendigen Vorarbeiten zur Bewertung existierender Lösungen und zur Entwicklung einer neuen Lösung erbracht. Zu Beginn wird der Kontext des betrachteten Problems eingegrenzt, worauf die Beschreibung der Struktur und der Schnittstelle eines Systems von Wertströmen

folgen. Danach werden die Grundlagen zur Kalkulation von Kapazität, Auslastung und Investitionen dargelegt, bezüglich ihrer Anwendbarkeit geprüft und erweitert. Abschließend werden die Anforderungen an eine Software zur Unterstützung der Planung im gegebenen Kontext kategorisiert und formuliert. Die essenziellen Anforderungen konkretisieren in ihrer Gesamtheit das Problem, das es in dieser Arbeit zu lösen gilt.

A.1 Wie ist der Kontext der vorliegenden Arbeit, die strategische Planung der technischen Kapazität in der Serienfertigung, abzugrenzen?
A.2 Wie kann die Struktur eines gegebenen Systems von Wertströmen aus Sicht der Planung im Kontext dieser Arbeit formal eindeutig beschrieben werden?
A.3 Welche Eingaben und Ausgaben werden an der Systemschnittstelle ausgetauscht, und wie lassen sich die definierten Planungsziele beschreiben?
A.4 Welche Grundlagen existieren, um Kapazität, Auslastung und Investitionen zu kalkulieren, welche Voraussetzungen gelten, und inwieweit werden diese erfüllt?
A.5 Wie lassen sich die Grundlagen erweitern, um die Struktur realer Produktionssysteme in der Planung vollständig zu berücksichtigen?
A.6 Wie werden Anforderungen an eine geeignete Software kategorisiert, und wie lauten die essenziellen Anforderungen im gegebenen Fall?

Kapitel 3, Stand der Technik. Mit dem Vorliegen eindeutig definierter Anforderungen ist es möglich, existierende Software im Hinblick auf deren Erfüllung zu untersuchen. Zuerst werden Softwaretypen ausgewählt, wobei die grundsätzlich zu erfüllenden Anforderungen und die Eignung für einen produktiven Einsatz in Unternehmen betrachtet werden. Danach folgen die Bewertung der Softwaretypen für den gegebenen Einsatzzweck und, basierend auf den Erkenntnissen, die Zusammenfassung des Entwicklungsbedarfs.

B.1 Welche Typen von Softwareanwendungen sind für die Unterstützung der strategischen Planung technischer Kapazität in Unternehmen einsetzbar?
B.2 Bis zu welchem Grad erfüllen repräsentative, den Stand der Technik widerspiegelnde Vertreter der ausgewählten Softwaretypen die formulierten Anforderungen?
B.3 Existieren Anforderungen, die von keinem Vertreter der ausgewählten Softwaretypen erfüllt werden, obgleich diese als essenziell kategorisiert wurden?
B.4 Welcher Bedarf, eine neue Software entsprechend den formulierten Anforderungen zu entwickeln, verbleibt als Ergebnis der Bewertung?

Kapitel 4, Lösungsschritt I: grafische Modellierung und Modelltransformation. Durch den ersten Lösungsschritt wird der identifizierte Entwicklungsbedarf im Hinblick auf die Modellierung eines Systems von Wertströmen erfüllt. Hierzu wird von den Anforderungen eine geeignete grafische Notation abgeleitet, mit welcher die Abhängigkeiten zwischen den Elementen eines Systems dargestellt werden können. Daraufhin wird ein mathematisches Modell beschrieben, welches dem grafischen Modell entspricht und sich zur mathematischen Optimierung eignet. Ergänzt wird die Beschreibung durch die Entwicklung neuer Algorithmen zur Validierung eines grafischen Modells und zur Transformation in ein mathematisches Modell. Zuletzt richtet sich der Blick auf die Umsetzung in der Software AURELIE, und hierbei auf die Benutzerführung und die Anforderungserfüllung.

C.1 Wie kann der Begriff des Graphen aus der Literatur erweitert werden, um Wertströme im Kontext der vorliegenden Arbeit vollständig und eindeutig zu beschreiben?
C.2 Wie lautet ein geeigneter Algorithmus zur Validierung eines grafischen Modells, welches aus entsprechenden Wertstromgraphen zusammengesetzt ist?
C.3 Wie kann ein mathematisches Modell definiert werden, welches sich zur Optimierung unter Verwendung exakter Verfahren eignet?
C.4 Wie lautet ein geeigneter Algorithmus zur automatischen Transformation eines grafischen Modells in ein jeweils entsprechendes mathematisches Modell?
C.5 Wie lassen sich die Algorithmen in einer Software umsetzen, und inwieweit werden dadurch die zuvor formulierten Anforderungen erfüllt?

Kapitel 5, Lösungsschritt II: mathematische Optimierung. Mit dem zweiten Lösungsschritt wird der verbleibende Entwicklungsbedarf im Hinblick auf die Optimierung eines Systems von Wertströmen erfüllt. Anknüpfend an das vorangegangene Kapitel wird das mathematische Modell eines gegebenen Systems von Wertströmen verarbeitet. Auf der Grundlage des mathematischen Modells werden Zielfunktionen formalisiert, die durch Nebenbedingungen beschränkt werden und iterativ zu minimieren oder zu maximieren sind. Hierfür werden neue Algorithmen entwickelt, sodass als Ergebnis die Planungsziele gemäß den vorgegebenen Anforderungen erreicht werden. Den Abschluss bildet die Umsetzung in der Software AURELIE, wobei die Benutzerführung, die Erfüllung der Anforderungen, Erweiterungen und die Validierung der Ergebnisse betrachtet werden.

D.1 Welche Ansätze sind aus theoretischer und praktischer Sicht denkbar, um die Eindeutigkeit der Optimierungsergebnisse sicherzustellen?
D.2 Wie können die definierten Planungsziele gemäß den jeweils gewählten Ansätzen in formaler Notation beschrieben werden?
D.3 Wie lauten jeweils geeignete Algorithmen zur entsprechenden Maximierung der Kapazitäten, Minimierung der Investitionen und Optimierung der Auslastung?
D.4 Kann die Korrektheit der entwickelten Algorithmen formal nachgewiesen werden, falls diese keine Grundlage in der Literatur besitzen?
D.5 Wie lassen sich die Algorithmen umsetzen, inwieweit werden die formulierten Anforderungen erfüllt, und wie können die Ergebnisse validiert werden?

2 Lösungsvorbereitung: Systemanalyse

Gegenstand dieser Arbeit ist die strategische Planung der technischen Kapazität in der Serienfertigung. Im Hinblick auf die Planungsziele gilt es, die erreichbaren Kapazitäten zu maximieren, die notwendigen Investitionen zu minimieren und die Auslastung aller Maschinen, Anlagen und Einrichtungen zu optimieren. Den Bezugsrahmen hierfür bildet ein Produktionssystem aus sogenannten Wertströmen, das alle Prozesse zur Fertigung und Montage von Produkten an einem Produktionsstandort zusammenfasst. Aufgrund der unüberschaubaren Zahl von Möglichkeiten, die einzelnen Prozessschritte im Wertstrom eines Produkts zu verknüpfen, ist die Erreichung der Planungsziele eine Aufgabe von großer Komplexität. Des Weiteren ist vielen Prozessschritten dieselbe Produktionsanlage zugeordnet, wodurch sich die maximalen Stückzahlen jeweils gegenseitig beschränken. Als Beispiel dient ein ausgewählter Standort der Bosch Rexroth AG, an dem 119 Produkte in 1795 Prozessschritten hergestellt werden. Eine einzelne Produktionsanlage wird in diesem Beispiel von 67 Prozessschritten beansprucht. Entwicklungen des Marktes wie die Individualisierung der Produkte und die Verkürzung der Produktlebenszyklen erfordern eine weitere Flexibilisierung des Produktionsablaufs. Die Konsequenz ist eine nochmalige Steigerung der Komplexität in der Planung solcher Produktionssysteme.

In dem vorliegenden Kapitel wird mit einer Systemanalyse der Grundstein zur Bewältigung der oben geschilderten Herausforderungen gelegt. Der Zweck der Systemanalyse ist es, ein grundsätzliches Verständnis für Systeme von Wertströmen und für die strategische Planung der Produktion gemäß den definierten Zielen zu schaffen. In dem Zuge wird ein Formalismus zur eindeutigen Beschreibung der Struktur solcher Systeme entwickelt, die Kalkulation der Planungsziele erläutert und als Ergebnis die Komplexität reduziert. Der Verein Deutscher Ingenieure e. V. (VDI) definiert den Begriff der Systemanalyse übereinstimmend wie folgt (Richtlinie VDI 3633:1996-11, S. 17):

> »Systematische Untersuchung eines Systems hinsichtlich aller Systemdaten, Systemelemente und deren Wirkungen aufeinander. Mit Hilfe der Systemanalyse wird die Komplexität des Systems entsprechend den Untersuchungszielen durch sinnfällige Zergliederung in seine Elemente aufgelöst (Strukturierung).«

Die Systemanalyse endet mit der Formulierung von Anforderungen, die eine geeignete Software erfüllen muss, um ein geplantes System zu modellieren und das Modell abhängig vom Planungsziel zu optimieren. Die Anforderungen beschreiben das letztlich in dieser Arbeit zu lösende Problem, und folglich wird auf sie im weiteren Verlauf wiederholt Bezug genommen. Zunächst dienen sie in Kapitel 3 dem Nachweis, dass eine solche Software zum Zeitpunkt der Betrachtung nicht existierte. In Kapitel 4 und 5 werden mit ihrer Hilfe die Entscheidungen im Entwurf der Software AURELIE begründet, die im Rahmen der Arbeit entwickelt wurde und erstmals alle der formulierten Anforderungen erfüllt. Für die

Entwicklung von AURELIE war die Systemanalyse und insbesondere die Formulierung der zu erfüllenden Anforderungen ein entscheidender, die Lösung vorbereitender Schritt. Dessen Bedeutung wird durch den Grundsatz des Software Engineering bestätigt, dass die Entwicklung neuer Software insbesondere dann Erfolg verspricht, wenn die wichtigsten Anforderungen systematisch ermittelt und festgehalten werden (siehe u. a. Standish 1995, 2011, Yeo 2002, Legris und Collerette 2006, Brocke et al. 2009, Hochmuth 2011).

Das Kapitel ist folgendermaßen untergliedert. In Abschnitt 2.1 wird zuerst der Kontext des betrachteten Problems eingegrenzt. Abschnitt 2.2 widmet sich der Entwicklung eines mathematischen Formalismus zur Beschreibung der Systemstruktur, welcher auf der rekursiven Verknüpfung von einzelnen Prozessschritten eines Produkts zu Prozessen und Wertströmen basiert. In Abschnitt 2.3 folgt darauf die Systemschnittstelle, mittels derer Material und Informationen zwischen einem System und seiner Umwelt ausgetauscht werden können. Damit sind die Voraussetzungen erfüllt, um in Abschnitt 2.4 und 2.5 die Kalkulation von Kapazität, Auslastung und Investitionen im einfachen Fall eines einzelnen Prozessschritts bzw. im allgemeinen, komplexeren Fall verknüpfter Prozessschritte zu erläutern. Abschnitt 2.6 fasst die Anforderungen in Bezug auf die Modellierung eines geplanten Systems von Wertströmen und die Optimierung des Modells zusammen.

2.1 Kontext: strategische Planung technischer Kapazität in der Serienfertigung

Die Planung der Produktion umfasst eine Vielzahl von Aktivitäten, deren gemeinsames Ziel es ist, durch vorausschauende Betrachtung die Voraussetzungen zur wirtschaftlichen Ausführung aller Abläufe zu schaffen. Diese Aktivitäten unterscheiden sich u. a. im Hinblick darauf, ob zum einen die Auslastung von Anlagen oder zum anderen der Einsatz von Mitarbeitern geplant werden soll. Weiterhin ist entscheidend, wie weit die Planung in die Zukunft reicht und welche Größenordnung die Stückzahlen annehmen. Daraus folgen unterschiedliche Anforderungen an eine Software zur Untersützung der Planungsaktivitäten. Um die Bedingungen zu beschreiben, unter denen die Ergebnisse dieser Arbeit in der Praxis anwendbar sind, muss deshalb der Kontext des untersuchten Problems eindeutig eingegrenzt werden. Wie in Abbildung 2.1 schematisch dargestellt ist, werden zu diesem Zweck drei Teilgebiete der Produktionsplanung herausgegriffen, welche den Raum der Aktivitäten in der Planung jeweils entlang einer Dimension einschränken:

(1) technische Kapazität vs. Personalkapazität (gemäß Produktionsfaktor),
(2) strategische Planung vs. operative Planung (gemäß Zeithorizont),
(3) Serienfertigung (Serienfabrikation) vs. Einzelfertigung (gemäß Produktionstyp).

Technische Kapazität vs. Personalkapazität. Der Fokus liegt in dieser Arbeit auf der Produktion von verkaufsfähigen Gütern in einem Unternehmen (in Abgrenzung zu anderen betriebswirtschaftlichen Funktionen wie z. B. der Produktentwicklung, dem Einkauf und dem Vertrieb) und diesbezüglich auf der Planung der technischen Kapazität. Diese unterscheidet sich von der Planung der Personalkapazität grundsätzlich im Hinblick auf den

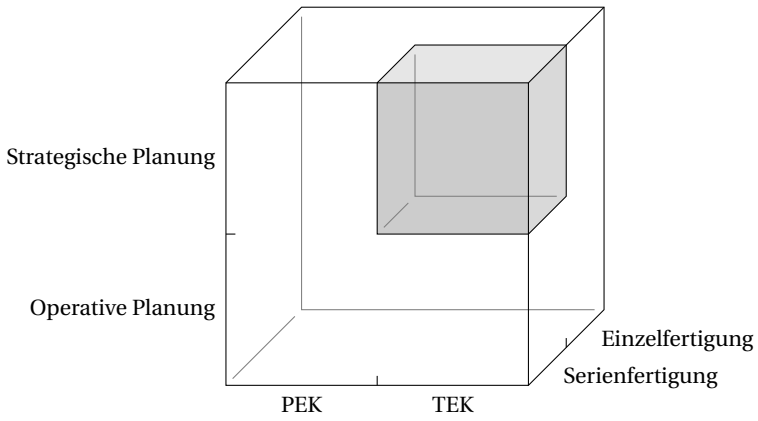


Abbildung 2.1: Kontext des betrachteten Problems (Planungsdimensionen). Festlegung auf drei Teilgebiete der Produktionsplanung entlang je einer Dimension: technische Kapazität vs. Personalkapazität (TEK bzw. PEK, gemäß Produktionsfaktor), strategische Planung vs. operative Planung (gemäß Zeithorizont) und Serienfertigung vs. Einzelfertigung (gemäß Produktionstyp).

jeweils begrenzenden Produktionsfaktor (Produktionsanlagen bzw. Produktionsmitarbeiter), welcher die maximal erreichbaren Stückzahlen bedingt:

(a) *technische Kapazität* (TEK): maximale Stückzahlen bestimmt durch die Zahl, Funktion und den Nutzungsgrad der Produktionsanlagen;
(b) *Personalkapazität* (PEK): maximale Stückzahlen bestimmt durch die Zahl, Qualifikation, Verfügbarkeit und die Leistung der Produktionsmitarbeiter.

In der vorliegenden Arbeit wird von einer zumindest teilweise automatisierten Produktion ausgegangen. Das heißt, die Abläufe erfordern ggf. den Einsatz von Personal, in jedem Fall aber den Einsatz von Produktionsanlagen, im weiteren Verlauf der Arbeit zusammengefasst als Maschinen, Anlagen und Einrichtungen (MAEs). Im Allgemeinen ist es umso wichtiger, alle zur Verfügung stehenden MAEs effektiv einzusetzen, je höher der Grad der Automatisierung in der Produktion ist. Der Grund dafür ist, dass neben den Betriebskosten (variable Kosten) bedingt durch die Abschreibung des Vermögenswertes der MAEs unabhängig von deren Nutzung periodische Kosten entstehen (Fixkosten). Die Kosten für Mitarbeiter in der Produktion werden dagegen v. a. durch die Anwesenheitszeit bestimmt, die mit Hilfe von Instrumenten wie Arbeitszeitkonten, Versetzungen und Verleihungen von Mitarbeitern sowie durch die Anpassung der Schichtmodelle vergleichsweise flexibel an die jeweils angefragten Stückzahlen angepasst werden kann (im Wesentlichen variable Kosten).

Da in einer teilautomatisierten Produktion Personal zum Betrieb von MAEs notwendig ist, existieren Abhängigkeiten zwischen TEK und PEK. Der Personalbedarf, die erforderliche Anwesenheitszeit von Mitarbeitern zur Steuerung der Anlagen, folgt näherungsweise proportional der erwarteten Auslastung. Des Weiteren können Investitionen in Produktionsanlagen mit höherem Automatisierungsgrad den Personalbedarf reduzieren. Wird das eingesetzte Personal verringert, bewirkt dies eine einmalige Steigerung der Produktivität bezogen auf die Anwesenheitszeit der Mitarbeiter. Zur kontinuierlichen Steigerung der Produktivität muss die Automatisierung um ein Produktivitätsmanagement ergänzt werden (siehe u. a. Sauter und von Killisch-Horn 2010, Grotepaß et al. 2014).

Strategische Planung vs. operative Planung. Im Schlaglicht der nachfolgenden Untersuchungen steht die strategische Planung, die mit Bezug auf den Zeithorizont der Aktivitäten gemessen ab dem Zeitpunkt der Betrachtung von der operativen Planung zu unterscheiden ist. Hierzu wird ein Schwellwert von einem Jahr festgelegt, welcher das Verständnis des Begriffs in der unternehmerischen Praxis bei der Bosch Rexroth AG widerspiegelt. Entsprechend werden die beiden Begriffe wie folgt voneinander abgegrenzt:

(a) *strategische*, d. h. langfristige Planung: alle Planungsaktivitäten mit einem Zeithorizont ab dem Betrachtungszeitpunkt von mehr als einem Jahr;
(b) *operative*, d. h. kurz- und mittelfristige Planung: alle Planungsaktivitäten mit einem Zeithorizont ab dem Betrachtungszeitpunkt von höchstens einem Jahr.

Im Mittelpunkt der strategischen Planung der TEK, wie sie in der Arbeit verstanden wird, stehen die Kapazitäten zur Herstellung von Produkten, die daraus folgende Auslastung aller MAEs und die Investitionen in MAEs, die zur Fertigung und Montage der geplanten Stückzahlen erforderlich sind. Es werden drei Planungsziele verfolgt:

maximale Kapazitäten: maximale Produktstückzahlen, wobei die Bedingung gelten muss, dass die Auslastung keiner MAE einen Wert von 100 Prozent überschreitet;
minimale Investitionen: minimaler Bedarf zusätzlicher MAEs zur Fertigung und Montage aller Produkte in den geplanten Stückzahlen;
optimale Auslastung: optimale Verteilung der geplanten Stückzahlen aller Produkte unter Nutzung der verfügbaren MAEs.

Festzuhalten ist an dieser Stelle, dass die zu erläuternden Planungsziele als Funktionen der Stückzahlen und der Auslastung aller MAEs darstellbar sind. Die Stückzahlen sind je nach Planungsziel gesucht oder gegeben. Die Auslastung der MAEs wird im langfristigen Mittel des Planungsintervalls (i. d. R. Quartal, Halbjahr oder Jahr) bestimmt und ist abhängig von der Struktur des geplanten Systems, den Taktzeiten der Prozessschritte, den Betriebsmittelzeiten der genutzten MAEs und den Stückzahlen der Produkte.

Serienfertigung vs. Einzelfertigung. Die im Rahmen der Arbeit getroffenen Annahmen und die davon abgeleiteten Ergebnisse gelten in erster Linie für den Produktionstyp der Serienfertigung, welcher von der Einzelfertigung zu unterscheiden ist. Laut dem Verband für Arbeitsgestaltung, Betriebsorganisation und Unternehmensentwicklung (REFA), einer führenden Organisation im Bereich des Industrial Engineering, liegt eine Serienfertigung vor, wenn »gleichartige Gegenstände in größeren Stückzahlen« hergestellt werden (Hammer 1997, S. 178, dort unter dem Begriff der Serienfabrikation aufgeführt). In diesem Fall ist es zur strategischen Planung der TEK zulässig, für jeden Prozessschritt eines Produkts eine Taktzeit als Sollzeit vorzugeben. Dagegen ist das im Fall einer Einzelfertigung im Allgemeinen nicht möglich, da jeweils nur eine begrenzte Stückzahl mit der gleichen Taktzeit gefertigt bzw. montiert wird. Entsprechend beruht die Abgrenzung der zwei genannten Produktionstypen auf einer bestimmten Stückzahl einzelner Produktvarianten in einem gegebenen Zeitintervall, wofür in der Literatur verschiedene Werte genannt werden. Folgende monatliche Zahlen sollen als ein Anhaltspunkt dienen (siehe Schlegel 2002, S. 19):

(a) *Serienfertigung* (Mittel-, Großserien- und Massenfertigung): $\geqslant 20$ Stück/Monat
(b) *Einzelfertigung* (einschließlich Kleinserienfertigung): < 20 Stück/Monat

Besondere Anforderungen an die Planung stellt der Fall einer variantenreichen Serienfertigung, wie er häufig bei der Bosch Rexroth AG anzutreffen ist. Der Begriff der Variantenvielfalt bezieht sich in dem Zusammenhang sowohl auf die Produkte als auch auf die Prozessschritte. Zum einen muss zwischen verschiedenen Prozessschritten unterschieden werden, um die Varianten der Produkte zu fertigen und zu montieren. Zum anderen besteht in vielen Fällen die Möglichkeit, zur Herstellung eines Produkts zwischen verschiedenen Prozessschritten frei zu wählen. Die daraus folgenden Bedingungen bzw. Freiheitsgrade werden im Verlauf des Kapitels durch die rekursive Verknüpfung von Prozessschritten zu Prozessen formalisiert. Ein weiteres hervorzuhebendes Merkmal der variantenreichen Serienfertigung, die Überlagerung der Prozessschritte, betrifft die Nutzung der MAEs. Darunter wird in dieser Arbeit verstanden, dass MAEs im Planungsintervall von mehreren Prozessschritten beansprucht werden. Wenn für deren Auslastung ein Höchstwert vorgegeben ist, dann beschränken sich die erreichbaren Stückzahlen der jeweiligen Prozessschritte gegenseitig.

All diese Merkmale eines Produktionssystems steigern die Komplexität und müssen in der Planung der TEK berücksichtigt werden. Zuerst wird die Struktur solcher Systeme analysiert, welche aus Prozessschritten, Prozessen und Wertströmen besteht, bevor die Systemschnittstelle und die Grundlagen der Kalkulation folgen.

2.2 Systemstruktur: rekursive Zusammensetzung von Wertströmen

Ein System ist gemäß VDI eine abgegrenzte Anordnung von Komponenten, die miteinander in einer Beziehung stehen (siehe Richtlinie VDI 3633:1996-11, S. 17). Diese Definition spiegelt das allgemeine Verständnis in Theorie und Praxis wider (siehe u. a. Hammer 1997, S. 187, Ehrlenspiel und Meerkamm 2013, S. 21 ff) und wird jeweils nach Kontext der Untersuchung abgewandelt oder konkretisiert (für eine Diskussion siehe Backlund 2000). Entsprechend wird der Begriff des Systems auch in dieser Arbeit verwendet. Um eine Verwechslung mit den Komponenten zu vermeiden, aus denen ein Produkt laut dessen Stückliste besteht, wird im Folgenden allerdings nicht von *Komponenten* eines Systems, sondern als rein begriffliche Abweichung von den *Elementen* eines Systems gesprochen.

In diesem Abschnitt soll die Struktur eines Systems sogenannter Wertströme analysiert und mit Hilfe mathematischer Definitionen eindeutig wiedergegeben werden. Auf elementarer Ebene basiert die Systemstruktur auf Prozessschritten zur Fertigung und Montage der gegebenen Produkte. Diese werden durch Verkettung sequenzieller, alternativer und/oder selektiver Verknüpfungen rekursiv zu Prozessen und Wertströmen verbunden.

Zur exakten Formulierung der mathematischen Zusammenhänge werden die nachfolgenden Symbole eingeführt. Sie besitzen eine spezifische, z. T. kontextabhängige Bedeutung und ergänzen die in der Literatur verbreiteten, im entsprechenden Verzeichnis aufgeführten Symbolklassen, Operatoren und Standardsymbole:

$\mathrm{AL}(c_1, c_2)$	alternative Verknüpfung von $c_{1/2} \in \mathrm{VMg}(P)$, $\mathrm{AL}(c_1, c_2) \in \mathrm{VMg}(P)$
c	Operand und/oder Ergebnis einer Verknüpfung, $c \in \mathrm{VMg}(P)$, ggf. mit Index 1, 2 usw.
c^*	Element maximaler Ordnung einer Teilmenge C, $c^* \in C$
C	Teilmenge einer Verknüpfungsmenge, beschreibt einen geplanten Ablauf zur Fertigung und Montage, $C \subset \mathrm{VMg}(P)$, z. B. $\{\mathrm{PS}\,1, \mathrm{PS}\,2, \mathrm{PS}\,3, \mathrm{AL}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{SQ}(\mathrm{AL}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{PS}\,3)\}$
i, j	natürliche Zahl, jeweils verwendet mit kontextabhängiger Bedeutung, $i, j \in \mathbb{N}$ oder $\mathbb{N}_0$
n	Zahl durch Verkettung zu verknüpfender Operanden c_i, $i \in \{1, \ldots, n\}$, $n \in \mathbb{N}$
P	Menge der Prozessschritte eines Produkts, z. B. $\{\mathrm{PS}\,1, \mathrm{PS}\,2, \ldots\}$
q	fixe Quote einer selektiven Verknüpfung, $q \in (0, 1)$, ggf. mit Index 1, 2 usw.
q'	resultierende Quote verketteter selektiver Verknüpfungen, $q' \in (0, 1)$, ggf. mit Index 1, 2 usw.
$\mathrm{SL}(c_1, c_2, q)$	selektive Verknüpfung von $c_{1/2} \in \mathrm{VMg}(P)$ mit einer fixen Quote $q \in (0, 1)$, $\mathrm{SL}(c_1, c_2, q) \in \mathrm{VMg}(P)$
$\mathrm{SQ}(c_1, c_2)$	sequenzielle Verknüpfung von $c_{1/2} \in \mathrm{VMg}(P)$, $\mathrm{SQ}(c_1, c_2) \in \mathrm{VMg}(P)$
$\mathrm{VMg}(P)$	Verknüpfungsmenge von P, z. B. $\{\mathrm{PS}\,1, \mathrm{PS}\,2, \mathrm{PS}\,3, \ldots, \varnothing, \mathrm{SQ}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{AL}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{SL}(\mathrm{PS}\,1, \mathrm{PS}\,2, q), \ldots, \mathrm{SQ}(\mathrm{SQ}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{PS}\,3), \ldots : q \in (0, 1)\}$
$\mathrm{VMg}(P, i)$	Verknüpfungsmenge von P einer Ordnung $i \in \mathbb{N}_0$, z. B. $\mathrm{VMg}(P, 0) = \{\mathrm{PS}\,1, \mathrm{PS}\,2, \ldots, \varnothing\}$, $\mathrm{VMg}(P, 1) = \{\mathrm{SQ}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{AL}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{SL}(\mathrm{PS}\,1, \mathrm{PS}\,2, q), \ldots : q \in (0, 1)\}$ usw.

2.2.1 Prozessschritte, Stückzahlverteilung und Verknüpfungstypen

Begriff des Prozessschritts. Ein Vorgang, mit gleicher Bedeutung in der Literatur auch Arbeitsvorgang oder Arbeitsgang genannt, ist laut REFA definiert als eine Arbeit, die einem bestimmten Zweck dient und stets am selben Ort, in derselben Zeit, mit denselben Personen und denselben Arbeitsmitteln ausgeführt wird (siehe Hammer 1997, S. 41 f). Das heißt, nicht nur die Eigenschaften und der Nutzungsgrad der Produktionsanlagen bestimmen die geplante Zeitdauer eines Vorgangs. Darüber hinaus müssen die Verfügbarkeit, die Qualifikation und die Leistung der einzusetzenden Produktionsmitarbeiter in der operativen und strategischen Planung der Kapazität berücksichtigt werden.

Da allerdings im Kontext dieser Arbeit die Planung der TEK betrachtet wird, soll vorausgesetzt werden, dass Mitarbeiter mit der geforderten Qualifikation stets in ausreichender Zahl zur Verfügung stehen. Um diese grundlegende Annahme für die Kalkulation von Kapazität, Auslastung und Investitionen festzuhalten, wird für die weitere Analyse in Anlehnung an die obige Definition des Vorgangs der Begriff des Prozessschritts geprägt:

Definition 2.1: Prozessschritt. Ein Prozessschritt ist ein mindestens teilweise maschineller Vorgang zur Fertigung und Montage eines Produkts, der unter idealen Voraussetzungen im Hinblick auf die nötigen Arbeitspersonen stets in derselben Zeit sowie mit demselben Ergebnis ausgeführt wird und in entsprechender Weise geplant werden kann. Jedem Prozessschritt ist eine bestimmte Maschine, Anlage oder Einrichtung zugeordnet, die im Planungsintervall auch anderen Prozessschritten zugewiesen sein kann, aber zu jedem Zeitpunkt nur für einen einzigen Prozessschritt genutzt werden darf.

Der damit definierte Begriff des Prozessschritts ist von der sogenannten Prozesszeit abgeleitet. Laut REFA (Hammer 1997, S. 160) bezeichnet die Prozesszeit die »Nutzungszeit von Betriebsmitteln [MAEs], die von diesen, aber nicht von der Arbeitsperson beeinflusst wird«. Im Folgenden wird von Prozessschritten und Prozessen gesprochen, um dadurch zu betonen, dass maßgeblich die MAEs die Nutzungszeit bestimmen. Mit Blick auf den Zeitanteil,

der von der Arbeitsperson abhängig ist, wird dagegen stets von idealen Voraussetzungen ausgegangen. Das heißt, zur Planung der TEK wird immer die kürzestmögliche Nutzungszeit angenommen, die mit der jeweils gegebenen MAE erzielt werden kann.

Zu maschinell unterstützten Vorgängen, d. h. zu Prozessschritten im Verständnis der oben stehenden Definition, zählen neben nicht wertschöpfenden Tätigkeiten wie dem Messen und Prüfen außerdem alle Fertigungsverfahren. Gemäß der Norm DIN 8580:2003-09 werden diese in die folgenden Hauptgruppen untergliedert:

(1) Urformen (generative Fertigungsverfahren wie z. B. Gießen, Sintern und Pasten),
(2) Umformen (z. B. Walzen, Gesenkschmieden, Pressen, Tiefziehen und Biegen),
(3) Trennen (Zerspanen wie z. B. Sägen, Hobeln, Fräsen, Bohren, Drehen und Schleifen, außerdem Schneiden, Funkenerodieren und Demontieren),
(4) Fügen (z. B. Schweißen, Löten, Kleben, Montieren, Nieten und Schrauben),
(5) Beschichten (z. B. Lackieren, Galvanisieren, Pulverbeschichten und Feuerverzinken),
(6) Ändern von Stoffeigenschaften (z. B. Härten und Glühen).

Verknüpfung von Prozessschritten. Jeder Vorgang ist laut REFA Teil einer zu leistenden Gesamtarbeit (siehe Hammer 1997, S. 41), und entsprechend wird auch jeder Prozessschritt mit anderen Prozessschritten zu einem oder mehreren Prozessen verknüpft. Es stellt sich die Frage, welche Typen der Verknüpfung hierbei unterschieden werden müssen. Um diese zu beantworten, wurde vor Erstellung der Arbeit eine Reihe von Workshops mit erfahrenen Fachexperten und verantwortlichen Führungskräften der Bosch Rexroth AG durchgeführt. Zur Diskussion stand dabei die zukünftige Planung der Kapazität, der erwarteten Auslastung und der erforderlichen Investitionen mit strategischem Zeithorizont, wie im ersten Abschnitt des Kapitels festgehalten. Als Ergebnis von darauf basierenden, eigenen Überlegungen werden die drei Typen der sequenziellen, alternativen und selektiven Verknüpfung eingeführt. Zunächst sollen sie formal beschrieben werden, um in den nachfolgenden Abschnitten ihre Bedeutung durch ausgewählte Beispiele zu veranschaulichen.

Durch die geeignete Kombination von Verknüpfungen der verschiedenen Typen ist es möglich, Abläufe zur Fertigung und Montage eines Produkts zum Zweck der strategischen Planung der TEK eindeutig zu erfassen, wie Abbildung 2.2 für ein einfaches Beispiel zeigt. Jeder mögliche Fall, wie komplex er auch zuerst erscheinen mag, kann auf eine verkettete Verknüpfung von einzelnen Prozessschritten zurückgeführt und als solche beschrieben werden. Zum Beleg dieser Aussage wird ein mathematischer, auf dem Prinzip der Rekursion aufbauender Formalismus entwickelt und mengentheoretisch definiert.

Der Ablauf der Rekursion ist folgender: Zunächst wird ein Prozessschritt zur Fertigung und Montage eines Produkts mit einem zweiten Prozessschritt oder mit einem Bypass sequenziell, alternativ oder selektiv verknüpft (Rekursionsanfang). Ein Bypass bezeichnet in dem Zusammenhang einen Ablaufabschnitt, den ein Teil der Produktstückzahl anstelle von einem oder mehreren Prozessschritten ohne Zeitverzögerung durchläuft. Danach wird das Ergebnis der Verknüpfung in der gleichen Weise wieder mit einem Prozessschritt, einem Bypass oder mit dem Ergebnis einer anderen Verknüpfung geeignet verbunden (Rekursionsschritt). Der zuletzt genannte Schritt wird wiederholt, bis alle Prozessschritte gemäß dem geplanten Ablauf miteinander verknüpft sind.

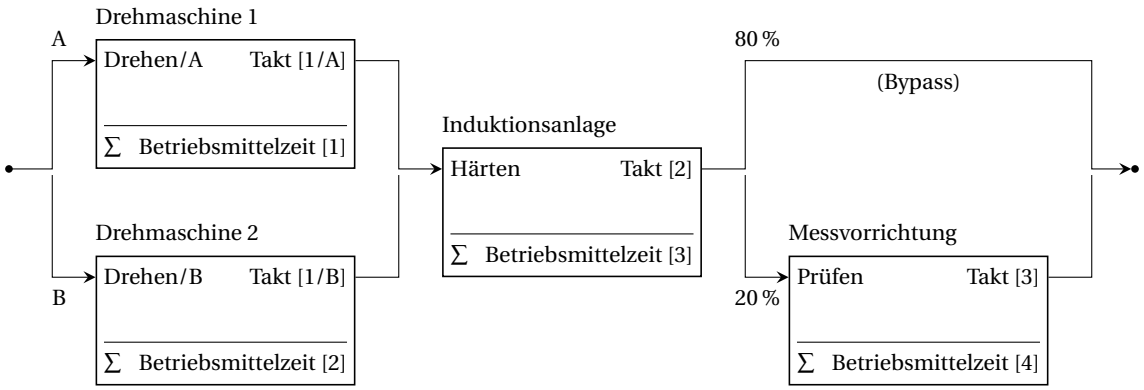


Abbildung 2.2: Verknüpfung von Prozessschritten zu Prozessen (Beispiel). Im gezeigten Beispiel sind vier Prozessschritte mit je einer MAE vorgegeben, von denen jedes Stück des Produkts nacheinander zwei bzw. zu einem fix vorgegebenen Anteil drei durchläuft (sequenzielle Verknüpfung, 'Drehen', 'Härten' bzw. 'Prüfen'). Für den ersten Prozessschritt existieren zwei frei wählbare Möglichkeiten (alternative Verknüpfung, 'Drehen/A' und 'B'). Den dritten Prozessschritt ('Prüfen') durchlaufen stets genau 20 Prozent der Stückzahl (selektive Verknüpfung). Dies entspricht vier möglichen Folgen von Prozessschritten für jedes Stück des Produkts, d. h. vier Prozessen.

Im Weiteren werden die definitorischen Grundlagen geschaffen, um die Ergebnisse rekursiv verketteter Verknüpfungen zu beschreiben. Wie ein Beispiel zum Abschluss verdeutlichen soll, ist jeder geplante Ablauf eindeutig als eine endliche Teilmenge der unendlichen Menge aller möglichen Ergebnisse darstellbar. Zunächst werden die Prozessschritte eines gegebenen Produkts zu einer Menge P zusammengefasst. Von dieser wird wiederum die Verknüpfungsmenge $\mathrm{VMg}(P)$ abgeleitet. Die Menge enthält neben allen Prozessschritten des Produkts aus der zuvor genannten Menge P einen Bypass und die Ergebnisse von allen darauf basierenden rekursiven Verknüpfungen, wobei eine unendliche Zahl von Rekursionsschritten zulässig, d. h. die Rekursionstiefe nicht beschränkt ist. Damit ist es nun möglich, die drei zu definierenden Typen von Verknüpfungen formal als Abbildungen von Elementen zu Ergebnissen innerhalb einer Menge auszudrücken. Hierzu wird jedem möglichen Paar verknüpfter Elemente $c_{1/2}$ aus der Menge $\mathrm{VMg}(P)$, welche als Operanden bezeichnet werden, ein Ergebnis c derselben Menge zugeordnet. Das entspricht folgender Zuordnungsvorschrift für den Fall einer sequenziellen (SQ) bzw. alternativen (AL) Verknüpfung:

$$\begin{aligned} \mathrm{SQ,AL}\colon \mathrm{VMg}(P) \times \mathrm{VMg}(P) &\to \mathrm{VMg}(P)\,, \\ (c_1, c_2) &\mapsto c\,. \end{aligned} \tag{2.1}$$

Im Fall einer selektiven Verknüpfung (SL) werden die Operanden $c_{1/2}$ um einen Parameter q im Intervall $(0,1)$ erweitert, dessen Rolle zu einem späteren Zeitpunkt dargelegt werden soll. Die entsprechende Zuordnungsvorschrift für den dritten Verknüpfungstyp lautet analog zu oben stehender Gleichung (2.1) in formaler Notation:

$$\begin{aligned} \mathrm{SL}\colon \mathrm{VMg}(P) \times \mathrm{VMg}(P) \times (0,1) &\to \mathrm{VMg}(P)\,, \\ (c_1, c_2, q) &\mapsto c\,. \end{aligned} \tag{2.2}$$

Es wird deutlich, dass Verknüpfungen zweier Elemente der Menge $\mathrm{VMg}(P)$ in jedem Fall wieder zu einem Element derselben Menge führen. Gesucht ist eine geeignete, rekursive Definition, um diese Menge eindeutig zu beschreiben. Hierzu werden in einem Zwischenschritt die zu Beginn der Rekursion vorgegebenen und die nach jedem Rekursionsschritt erzeugten Elemente jeweils zusammengefasst. Die daraus resultierenden, zueinander disjunkten Mengen werden Verknüpfungsmengen $\mathrm{VMg}(P,i)$ der Ordnungen i genannt, wobei eine Ordnung gleich null den Rekursionsanfang und Ordnungen einer natürlichen Zahl größer als null den jeweiligen Rekursionsschritt bezeichnen.

Die Menge $\mathrm{VMg}(P,0)$ enthält die Prozessschritte der Menge P sowie einen Bypass. Demgegenüber werden die Elemente der Mengen $\mathrm{VMg}(P,i)$ aller natürlichen Ordnungen i größer als null durch Verknüpfungen zweier Operanden $c_{1/2}$ gebildet. Die Operanden stammen jeweils aus einer Menge $\mathrm{VMg}(P,i_{1/2})$ einer kleineren Ordnung $i_{1/2}$. Das heißt, es handelt sich um vorgegebene Elemente oder Ergebnisse vorheriger Rekursionsschritte. Zusätzlich muss mindestens einer der Operanden ein Element der Menge $\mathrm{VMg}(P,i-1)$, d. h. ein Ergebnis des unmittelbar vorhergehenden Rekursionsschritts sein. Durch Ergänzung dieser Bedingung spiegelt die Ordnung die Zahl der Rekursionsschritte wider, die ausgeführt werden müssen, um die Elemente zu erzeugen. Damit lautet die Definition der Verknüpfungsmenge $\mathrm{VMg}(P,i)$ einer allgemeinen Ordnung i wie folgt.

Definition 2.2: Verknüpfungsmenge VMg(P,i) einer Ordnung i. Gegeben sei die Menge P der Prozessschritte eines Produkts. Die Verknüpfungsmenge $\mathrm{VMg}(P,0)$ der Ordnung null enthält die Prozessschritte der Menge P und einen Bypass, der formal durch das Symbol der leeren Menge repräsentiert wird. Dies entspricht dem folgenden Ausdruck:

$$i=0\colon \qquad \mathrm{VMg}(P,0)=P\cup\{\varnothing\}. \tag{2.3a}$$

Die Verknüpfungsmenge $\mathrm{VMg}(P,i)$ einer gegebenen natürlichen Ordnung i größer als null enthält die Ergebnisse der sequenziellen, alternativen und selektiven Verknüpfungen aller Paare von Operanden $c_{1/2}$ der Verknüpfungsmengen $\mathrm{VMg}(P,i_{1/2})$. Dabei gelten die Bedingungen, dass die Ordnungen $i_{1/2}$ kleiner als i und mindestens eine der zwei Ordnungen gleich $i-1$ sein muss. Das heißt, zumindest ein Operand muss ein Element aus der Verknüpfungsmenge $\mathrm{VMg}(P,i-1)$ sein. Formal ausgedrückt gilt:

$$\begin{aligned} i\in\mathbb{N}\colon \qquad \mathrm{VMg}(P,i)=\Big\{c\colon\ & \exists i_1,i_2\in\{0,\dots,i-1\}\Big(\big(i_1=i-1\vee i_2=i-1\big)\wedge \\ & \exists c_1\in\mathrm{VMg}(P,i_1)\ \exists c_2\in\mathrm{VMg}(P,i_2)\ \exists q\in(0,1) \\ & \qquad \big(c=\mathrm{SQ}(c_1,c_2)\vee c=\mathrm{AL}(c_1,c_2)\vee c=\mathrm{SL}(c_1,c_2,q)\big)\Big)\Big\}. \end{aligned} \tag{2.3b}$$

Unter Verwendung der Definition gilt es nun, den eingangs geprägten Begriff der Verknüpfungsmenge $\mathrm{VMg}(P)$ unabhängig von einer festgelegten, einschränkenden Ordnung zu definieren. Wie zuvor erläutert, enthält die Menge alle zu Beginn gegebenen Elemente und außerdem alle Ergebnisse, die durch Verknüpfungen von zwei Operanden derselben Menge in den darauffolgenden Rekursionsschritten erzeugt werden. Damit verbleibt schließlich nur noch, die disjunkten Mengen aller Ordnungen zu vereinigen.

Definition 2.3: Verknüpfungsmenge VMg(P). Es sei die Menge P der Prozessschritte eines Produkts gegeben. Die Verknüpfungsmenge $\mathrm{VMg}(P)$ folgt aus der disjunkten Vereinigung der Verknüpfungsmengen $\mathrm{VMg}(P,i)$ aller natürlichen Ordnungen i größer als oder gleich null, wie in Gleichung (2.3a, 2.3b) definiert. Dementsprechend gilt:

$$\mathrm{VMg}(P) = \dot{\bigcup_{i\in\mathbb{N}_0}} \mathrm{VMg}(P,i)\,. \tag{2.4}$$

Der in Gleichung (2.4) angegebene Ausdruck beschreibt die Ergebnisse aller möglichen Verknüpfungen basierend auf den Prozessschritten einer Menge P. Um den geplanten Ablauf zur Fertigung und Montage eines Produkts zu erfassen, werden die einzelnen Prozessschritte gemäß dem vorgegebenen Plan auf eindeutige Weise miteinander verbunden. Die Menge dieser Verknüpfungen bezeichnet formal eine endliche Teilmenge C der unendlichen Verknüpfungsmenge $\mathrm{VMg}(P)$. Folglich lässt sich jeder mögliche Ablauf, wie eingangs schon vorweggenommen, durch eine solche Teilmenge beschreiben.

Jedes Element der Menge C gehört zur Verknüpfungsmenge $\mathrm{VMg}(P,i)$ einer Ordnung i, welche die Zahl der Rekursionsschritte widerspiegelt, die notwendig sind, um das jeweilige Element zu erzeugen. Für das Verständnis der weiteren Ausführungen ist insbesondere das sogenannte Element c^* maximaler Ordnung in einer Teilmenge C von Bedeutung. Dieses Element geht als Ergebnis aus dem letzten Rekursionsschritt hervor und wird mit Bezug auf die Menge $\mathrm{VMg}(P,i)$ wie folgt definiert.

Definition 2.4: Element c^* maximaler Ordnung. Gegeben seien die Menge P der Prozessschritte eines Produkts und eine Teilmenge C der Verknüpfungsmenge $\mathrm{VMg}(P)$, wie in Gleichung (2.4) definiert. Jedes Element der Menge C gehört entsprechend Gleichung (2.3a, 2.3b) zur Verknüpfungsmenge $\mathrm{VMg}(P,i)$ genau einer Ordnung i. Bezug nehmend darauf ist das Element c^* der maximalen Ordnung in der Menge C durch die notwendige Bedingung charakterisiert, dass innerhalb dieser Menge kein anderes Element zur Verknüpfungsmenge $\mathrm{VMg}(P,j)$ einer Ordnung j gehört, die größer ist als die Ordnung i.

Im allgemeinen Fall verknüpfter Prozessschritte ist das Element c^* maximaler Ordnung das Ergebnis derjenigen Verknüpfung, die ggf. nach Verkettung weiterer Verknüpfungen alle anderen Elemente verbindet. Im einfachen Fall eines einzelnen Prozessschritts oder Bypasses werden keine Elemente verknüpft, die Rekursion endet bereits vor ihrem Beginn, und das Element c^* bezeichnet diesen Prozessschritt bzw. Bypass. In einer endlichen Teilmenge C der Verknüpfungsmenge $\mathrm{VMg}(P)$, die einen geplanten, d. h. insbesondere einen lückenlosen Ablauf beschreibt, existiert stets genau ein solches Element. Der Grund dafür ist, dass jedes Paar von Elementen einer solchen Menge in einer bestimmten Beziehung zueinander steht. Entweder ist eines der beiden Elemente das Ergebnis einer ggf. verketteten Verknüpfung mit dem anderen Element als einem Operanden, oder die Elemente werden im Rekursionsverlauf miteinander verknüpft. Im ersten Fall besitzen die zwei Elemente jeweils verschiedene Ordnungen. Im zweiten Fall muss die Menge C ein weiteres, drittes Element enthalten, welches dem Ergebnis der Verknüpfung entspricht und dessen Ordnung laut Definition 2.2 größer als jene beider Elemente ist.

Verteilung der Produktstückzahl. Die Definitionen bilden in ihrer Gesamtheit einen Formalismus für die Struktur eines Systems von Wertströmen. Dieser ermöglicht es, jeden geplanten Ablauf rekursiv aus Prozessschritten einer Menge P zusammenzusetzen und eindeutig als eine endliche Teilmenge C der Verknüpfungsmenge $\mathrm{VMg}(P)$ auszudrücken. Darauf aufbauend wird im Folgenden dargelegt, wie jedem Prozessschritt ein konstanter oder variabler Anteil der Stückzahl zugewiesen wird. Die Verteilung bestimmt, in welchem Ausmaß die jeweils zugeordneten MAEs genutzt werden und ist entsprechend Grundlage für die Kalkulation von Kapazität, Auslastung und Investitionen.

Der Stückzahlanteil eines Prozessschritts folgt aus zwei Prämissen: (1) Die Stückzahl des Produkts durchläuft das Element c^* maximaler Ordnung in der je nach Planungsziel gesuchten oder gegebenen, unverminderten Höhe; (2) für jede Verknüpfung zweier Operanden gilt, dass der Anteil der Stückzahl, welcher durch das Ergebnis verläuft, nach bestimmten Bedingungen auf die Operanden verteilt und in dem Zuge ggf. reduziert wird. Das heißt, in der umgekehrten Reihenfolge, in der sich die Prozessschritte des Produkts im Rekursionsverlauf zum Element c^* maximaler Ordnung fügen, wird die Stückzahl von diesem Element an die Prozessschritte weitergegeben. Die Bedingungen der Verteilung sind vom Typ der Verknüpfung und ggf. einem zusätzlichen Parameter q abhängig. Es folgen die Definitionen und die Anwendungsfälle der drei Verknüpfungstypen.

Definition 2.5: Sequenzielle Verknüpfung $\mathrm{SQ}(c_1, c_2)$. Gegeben sei die Menge P der Prozessschritte eines Produkts. Von einem Element c der Verknüpfungsmenge $\mathrm{VMg}(P)$ sei bekannt, dass dieses Element, wie in Gleichung (2.1) festgehalten, als Ergebnis aus der sequenziellen Verknüpfung $\mathrm{SQ}(c_1, c_2)$ zweier Operanden $c_{1/2}$ derselben Menge folgt. In einem solchen Fall gilt die Bedingung, dass die Stückzahlen, die jeweils durch diese drei Elemente verlaufen, immer den gleichen Wert annehmen müssen.

Bemerkung: Die Definition trifft bewusst keine Aussage über die Reihenfolge der verknüpften Elemente $c_{1/2}$ im Ablauf der Fertigung und Montage. Diese ist zwar entscheidend für die allgemeine Planung, Steuerung und Ausführung des Ablaufs, aber nicht für die Kalkulation der mittleren Auslastung genutzter MAEs im Planungsintervall.

Der Anwendungsfall sequenzieller Verknüpfungen sind Prozessschritte in der Fertigung und Montage eines Produkts, die laut Plan nacheinander ausgeführt werden müssen. Zu Beispielen zählen Prüfvorgänge nach einer Bearbeitung und Stufen der Wertschöpfung in der Produktion, die mit derselben oder verschiedenen MAEs ausgeführt werden.

Im Fall einer alternativen oder einer selektiven Verknüpfung wird die Stückzahl dagegen nicht unverändert weitergegeben. Stattdessen wird sie gemäß einem variablen bzw. fix vorgegebenen Verhältnis auf die miteinander verknüpften Elemente verteilt.

Definition 2.6: Alternative Verknüpfung $\mathrm{AL}(c_1, c_2)$. Gegeben sei die Menge P der Prozessschritte eines Produkts. Von einem Element c der Verknüpfungsmenge $\mathrm{VMg}(P)$ sei bekannt, dass dieses Element, wie in Gleichung (2.1) festgehalten, als Ergebnis aus der alternativen Verknüpfung $\mathrm{AL}(c_1, c_2)$ zweier Operanden $c_{1/2}$ derselben Menge folgt. Diesen Operanden sei je eine nichtnegative reelle Variable zugeordnet, die dem Stückzahlanteil entsprechen soll, der durch das jeweilige Element verläuft. In dem Fall gilt die Bedingung, dass die Summe

dieser Variablen, die als Prozessstückzahlen bezeichnet werden, immer gleich der Stückzahl sein muss, welche das Ergebnis c der Verknüpfung durchläuft.

Das heißt, jedem alternativ verknüpften Prozessschritt ist zur Planung der TEK eine nichtnegative reelle Prozessstückzahl zuzuweisen, die ausdrückt, welcher Anteil der Stückzahl des Produkts den jeweiligen Abschnitt durchläuft. Den Anwendungsfall bilden Prozessschritte, die sich bezüglich Taktzeiten, nachfolgender Prozessschritte und/oder MAEs unterscheiden, aber gegeneinander ausgetauscht werden können. Darunter wird verstanden, dass die Stückzahl frei unter ihnen verteilt werden kann, ohne das Ergebnis, d. h. das gefertigte oder montierte Produkt, zu beeinflussen. Beispiele hierfür sind die parallele Fertigung mit mehreren MAEs oder eine teilautomatisierte Bearbeitung, die bei Bedarf anstelle einer vollständig automatisierten Bearbeitung mit einer manuellen MAE ausgeführt wird.

Im Fall einer selektiven Verknüpfung wird die Stückzahl in ähnlicher Weise auf die jeweils miteinander verbundenen Elemente verteilt. Allerdings ist die Verteilung in dem Fall nicht variabel, sondern an einen Parameter q gekoppelt.

Definition 2.7: Selektive Verknüpfung SL(c_1, c_2, q). Gegeben sei die Menge P der Prozessschritte eines Produkts. Von einem Element c der Verknüpfungsmenge VMg(P) sei bekannt, dass dieses Element, wie in Gleichung (2.2) festgehalten, als Ergebnis aus einer selektiven Verknüpfung SL(c_1, c_2, q) zweier Operanden $c_{1/2}$ derselben Menge folgt. Dabei repräsentiert der Parameter q eine fix vorgegebene Quote im Intervall $(0, 1)$. In dem Fall gilt die Bedingung, dass der Stückzahlanteil, welcher jeweils die Elemente c_1 und c_2 durchläuft, im Verhältnis zur Stückzahl, die durch das Ergebnis c verläuft, immer gleich dem Wert der Quote q bzw. dem verbleibenden Wert $1 - q$ sein muss.

Der erste Anwendungsfall selektiver Verknüpfungen betrifft die Trennung von einzelnen Prozessschritten für Varianten eines Produkts. Beispielsweise werden zur Herstellung einer Standardvariante und einer Sondervariante häufig Prozessschritte ausgeführt, die sich in Bezug auf die Taktzeiten, nachfolgende Prozessschritte im Wertstrom des Produkts und/oder zugeordnete MAEs voneinander unterscheiden (indem z. B. je nach Variante eine Standardmontagelinie bzw. eine Sondermontagelinie genutzt wird). Der zweite Anwendungsfall bezieht sich auf notwendige, aber nicht wertschöpfende Tätigkeiten, die für einen fixen relativen Anteil der Stückzahl ausgeführt werden (z. B. eine Prüfung von 20 Prozent, wie in Abbildung 2.2 auf Seite 16 dargestellt). In entsprechenden Beispielen werden die betreffenden Prozessschritte selektiv mit einem Bypass verknüpft, sodass der verbleibende Anteil der Stückzahl (im gerade genannten Beispiel 80 Prozent) an den verknüpften Prozessschritten vorbeigeführt wird, ohne dass eine andere MAE genutzt wird.

Verknüpfung von n Operanden. Jede Verknüpfung verbindet unabhängig von ihrem Typ zwei Operanden aus der Menge VMg(P). Indem das Ergebnis einer Verknüpfung in der gleichen Weise mit einem weiteren Element aus dieser Menge verbunden wird, ist es möglich, die Definitionen der Verknüpfungstypen auf eine beliebige Zahl n von mehr als zwei Operanden c_i zu erweitern. Um n Elemente zu verknüpfen, werden hierzu $n - 1$ Verknüpfungen desselben Typs nacheinander verkettet. Zuerst wird das Ergebnis, welches aus der Verknüpfung der zwei Elemente $c_{1/2}$ hervorgeht, mit dem Element c_3 verbunden, danach

das Ergebnis mit dem Element c_4 usw. Die dadurch beschriebene Verkettung führt im Fall sequenzieller Verknüpfungen zu dem folgenden Ausdruck:

$$c = \mathrm{SQ}\left[\mathrm{SQ}\Big(\ldots \mathrm{SQ}\big(\mathrm{SQ}\,(c_1, c_2), c_3\big), \ldots\Big), c_n\right]. \tag{2.5}$$

Analog zu Gleichung (2.5) lautet der zugehörige Ausdruck zur Verkettung einer beliebigen Zahl von Verknüpfungen im alternativen Fall:

$$c = \mathrm{AL}\left[\mathrm{AL}\Big(\ldots \mathrm{AL}\big(\mathrm{AL}\,(c_1, c_2), c_3\big), \ldots\Big), c_n\right]. \tag{2.6}$$

Im Fall selektiver Verknüpfungen ist es dagegen nicht möglich, die Definition in einer vergleichbar einfachen Weise auf n Operanden zu erweitern. Der Grund ist, dass die Quoten selektiver Verknüpfungen multipliziert werden müssen, wenn letztere miteinander verkettet werden. Nimmt man für jede dieser $n-1$ Verknüpfungen eine Quote q an, ist der Stückzahlanteil, der letztlich durch das erste Element verläuft, gleich dem Wert q^{n-1}. Deshalb wird im Weiteren unterschieden zwischen den fixen Quoten q_i auf der einen Seite, die je einer Verknüpfung als Parameter zugeordnet werden, und den resultierenden Quoten q'_i auf der anderen Seite, welche jeweils dem Stückzahlanteil des Elements c_i entsprechen sollen und als Ergebnis aus der Verkettung aller Verknüpfungen hervorgehen.

Für jeden Operanden c_1 bis c_{n-1} sei je eine resultierende Quote q'_i im Intervall $(0,1)$ gegeben, wobei analog zur Definition für den Fall zweier Operanden Folgendes gefordert wird: Im Verhältnis zu der Stückzahl, die durch das Ergebnis c verläuft, muss der Stückzahlanteil der Elemente c_1 bis c_{n-1} jeweils immer gleich dem Wert der resultierenden Quote q'_i mit entsprechendem Index i und der Stückzahlanteil des Elements c_n immer gleich dem zu eins verbleibenden Wert sein. Da negative Stückzahlen ausgeschlossen werden sollen, wird als zusätzliche Einschränkung vorausgesetzt, dass die Summe der resultierenden Quoten q'_1 bis q'_{n-1} kleiner als eins ist. Gesucht sind die fixen Quoten q_i, um alle n Elemente selektiv zu verknüpfen. Um diese zu bestimmen, gilt es zuallererst, das Ergebnis c der Verkettung in Abhängigkeit von den fixen Quoten q_i zu formulieren:

$$c = \mathrm{SL}\left[\mathrm{SL}\Big(\ldots \mathrm{SL}\big(\mathrm{SL}\,(c_1, c_2, q_1), c_3, q_2\big), \ldots\Big), c_n, q_{n-1}\right]. \tag{2.7}$$

Auf dieser Basis ist im nächsten Schritt ein Ausdruck für die fixen Quoten q_i der einzelnen Verknüpfungen abhängig von den resultierenden Quoten q'_i gesucht. Um den Anteil der Stückzahl zu bestimmen, welcher letztendlich das erste Element durchläuft, muss die Stückzahl gemäß den verketteten selektiven Verknüpfungen mit den Quoten q_1 bis q_{n-1} multipliziert werden. Für das zweite Element gilt als Folge der Verknüpfung mit dem ersten Element, dass die insgesamt gegebene Stückzahl in diesem Fall nicht mit der Quote q_1, sondern mit dem Faktor $1-q_1$ und zusätzlich wieder mit allen Quoten q_2 bis q_{n-1} multipliziert werden muss. Nach der gleichen Überlegung gilt für das dritte Element, dass die Stückzahl als Konsequenz der Verknüpfung mit den zwei vorigen Elementen mit dem Faktor $1-q_2$ sowie mit den Quoten q_3 bis q_{n-1} von allen jeweils nachfolgenden verketteten Verknüpfungen

zu multiplizieren ist. Führt man dieses Vorgehen bis zum Element c_{n-1} fort, dann folgt für die resultierenden Quoten q'_1 bis q'_{n-1} der unten stehende Ausdruck:

$$i \in \{1,\dots,n-1\}: \qquad q'_i = \begin{cases} \prod_{j=1}^{n-1} q_j\,, & \text{falls } \; i = 1\,, \\ \left(1-q_{i-1}\right) \prod_{j=i}^{n-1} q_j\,, & \text{falls } \; i \in \{2,\dots,n-1\}\,. \end{cases} \tag{2.8}$$

Die Auflösung von Gleichung (2.8) nach q_i liefert die gesuchte Lösung. Somit ist es möglich, mehr als zwei Operanden wie in Gleichung (2.7) selektiv zu verknüpfen und die Stückzahl wie gefordert zu verteilen, indem die fixen Quoten q_i gemäß folgender Gleichung auf Basis gegebener, resultierender Quoten q'_i festgelegt werden:

$$i \in \{1,\dots,n-1\}: \qquad q_i = \begin{cases} 1 - \dfrac{q'_{i+1}}{\sum_{j=1}^{i+1} q'_j}\,, & \text{falls } \; i \in \{1,\dots,n-2\}\,, \\ \sum_{j=1}^{n-1} q'_j\,, & \text{falls } \; i = n-1\,. \end{cases} \tag{2.9}$$

In aller Kürze, die geboten ist, um zu den kalkulatorischen Grundlagen zu führen, sollen die Schritte eines Induktionsbeweises wiedergegeben und auf formale Weise gezeigt werden, weshalb unter Voraussetzung von Gleichung (2.8) ebenso Gleichung (2.9) gilt.

Beweis. Aus Gleichung (2.8) folgt durch die Umstellung des Quotienten q'_2/q'_1 unmittelbar, dass die zu zeigende Gleichung (2.9) für q_1 Gültigkeit besitzt (Induktionsanfang):

$$i = 1: \qquad \frac{q'_2}{q'_1} = \frac{1-q_1}{q_1}\,. \tag{2.10a}$$

Daraus folgt:

$$q_1 = \frac{q'_1}{q'_1+q'_2} = 1 - \frac{q'_2}{q'_1+q'_2} = 1 - \frac{q'_{i+1}}{\sum_{j=1}^{i+1} q'_j}\,. \tag{2.10b}$$

Gleichung (2.10b) ist das Ergebnis einer geeigneten Umformung von Gleichung (2.10a) und entspricht, wie leicht zu erkennen ist, Gleichung (2.9) für den Fall von q_1.

Setzt man Gleichung (2.9) für q_{i-1} voraus, folgt aus Gleichung (2.8) durch Umstellung des Quotienten q'_{i+1}/q'_i für alle folgenden Quoten q_i außer q_{n-1} (Induktionsschritt):

$$i \in \{2,\dots,n-2\}: \qquad \frac{q'_{i+1}}{q'_i} = \frac{1-q_i}{\left(1-q_{i-1}\right) q_i}\,, \;\; \text{wobei} \;\; q_{i-1} = 1 - \frac{q'_i}{\sum_{j=1}^{i} q'_j}\,. \tag{2.11a}$$

Daraus folgt:

$$q_i = \frac{\sum_{j=1}^{i} q'_j}{\sum_{j=1}^{i+1} q'_j} = 1 - \frac{q'_{i+1}}{\sum_{j=1}^{i+1} q'_j}\,. \tag{2.11b}$$

Umstellen des Ausdrucks in Gleichung (2.8) für q'_{n-1} nach q_{n-1} führt unter der Voraussetzung, dass Gleichung (2.9) für q_{n-2} gültig ist, schließlich zu dem Ergebnis:

$$i = n-1: \qquad q'_{n-1} = (1-q_{n-2})\,q_{n-1}\,, \;\text{ wobei }\; q_{n-2} = 1 - \frac{q'_{n-1}}{\sum_{j=1}^{n-1} q'_j}\,. \tag{2.12a}$$

Daraus folgt:

$$q_{n-1} = \sum_{j=1}^{n-1} q'_j\,. \tag{2.12b}$$

Das heißt, Gleichung (2.9) resultiert für alle q_i, $i \in \{1,\ldots,n-1\}$, aus Gleichung (2.8). □

Illustratives Beispiel. Zum besseren Verständnis soll das Beispiel in Abbildung 2.2 auf Seite 16 unter Verwendung der definierten Begriffe auf formale Weise beschrieben werden. Der geplante Ablauf zur Fertigung und Montage des Produkts basiert in dem gegebenen Fall auf einer Menge P, welche die folgenden vier Prozessschritte enthält:

$$P = \{\text{‘Drehen/A’}, \text{‘Drehen/B’}, \text{‘Härten’}, \text{‘Prüfen’}\}\,. \tag{2.13}$$

Von dieser wird wiederum eine unendliche Verknüpfungsmenge $\mathrm{VMg}(P)$ abgeleitet, welche nach Gleichung (2.4) die Prozessschritte der Menge P, einen Bypass und die Ergebnisse aller möglichen, im Verlauf der Rekursion erzeugten Verknüpfungen zusammenfasst. Die Menge $\mathrm{VMg}(P)$ lässt sich in dem Beispiel wie folgt darstellen (Verknüpfungen zweier Operanden, die jeweils denselben Prozessschritt bezeichnen, werden nicht aufgeführt, auch wenn diese laut Definition 2.3 nicht ausgeschlossen sind):

$$\begin{aligned}\mathrm{VMg}(P) = \big\{&\text{‘Drehen/A’}, \text{‘Drehen/B’}, \text{‘Härten’}, \text{‘Prüfen’}, \varnothing,\\ &\mathrm{SQ}(\text{‘Drehen/A’}, \text{‘Drehen/B’}), \mathrm{AL}(\text{‘Drehen/A’}, \text{‘Drehen/B’}),\\ &\mathrm{SL}(\text{‘Drehen/A’}, \text{‘Drehen/B’}, q), \ldots,\\ &\mathrm{SQ}(\mathrm{SQ}(\text{‘Drehen/A’}, \text{‘Drehen/B’}), \text{‘Härten’}), \ldots\colon\, q \in (0,1)\big\}\,.\end{aligned} \tag{2.14}$$

Jeder Ablauf lässt sich eindeutig als eine endliche Teilmenge C ausdrücken, welche die Prozessschritte, ggf. einen Bypass und die Ergebnisse aller gemäß Plan zu berücksichtigenden Verknüpfungen enthält. Die Menge C ist in diesem Beispiel wie folgt definiert:

$$\begin{aligned}C = \big\{&\text{‘Drehen/A’}, \text{‘Drehen/B’}, \text{‘Härten’}, \text{‘Prüfen’}, \varnothing, c_1, c_2, c_3, c_4\big\},\\ c_1 &= \mathrm{AL}(\text{‘Drehen/A’}, \text{‘Drehen/B’})\,,\\ c_2 &= \mathrm{SQ}(c_1, \text{‘Härten’})\,,\\ c_3 &= \mathrm{SL}(\varnothing, \text{‘Prüfen’}, 0{,}8)\,,\\ c_4 &= \mathrm{SQ}(c_2, c_3)\,.\end{aligned} \tag{2.15}$$

In dem hier gegebenen Fall bezeichnet das Element c_4 zugleich das Element c^* maximaler Ordnung. Einsetzen des Elements c_1 in den Ausdruck für c_2 sowie des daraus resultierenden Ausdrucks und des Elements c_3 in den Ausdruck für c_4 liefert für letzteres einen nichtrekursiven, geschlossenen Ausdruck, der nachfolgend angegeben ist:

$$c_4 = \mathrm{SQ}\Big[\mathrm{SQ}\big(\mathrm{AL}(\text{‘Drehen/A’}, \text{‘Drehen/B’}), \text{‘Härten’}\big), \mathrm{SL}(\varnothing, \text{‘Prüfen’}, 0{,}8)\Big]\,. \tag{2.16}$$

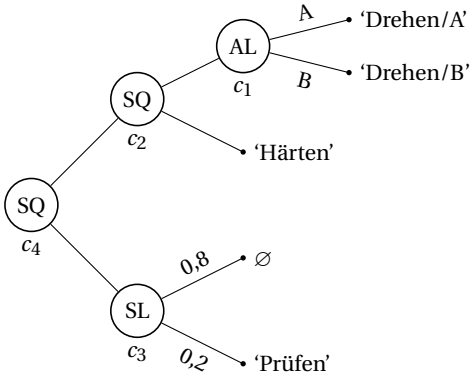


Abbildung 2.3: Ausdrucksbaum einer Verknüpfung von Prozessschritten (Beispiel). Die Blätter des Baums umfassen neben den Prozessschritten des Produkts einen Bypass, der in formalem Zusammenhang als leere Menge dargestellt wird. Der Baum als Ganzes betrachtet und jeder Teilbaum repräsentieren jeweils eine sequenzielle (SQ), alternative (AL) oder selektive (SL) Verknüpfung. Die inneren Knoten des Baums entsprechen den Ergebnissen dieser Verknüpfungen (c_1 bis c_4). Unter den Blättern und den inneren Knoten bezeichnet die Wurzel das Element c^* maximaler Ordnung (c_4).

Abbildung 2.3 stellt den Ausdruck grafisch dar, jedoch nicht als Folge von durchlaufenen Prozessschritten, sondern als Baum bestehend aus rekursiv verknüpften Operanden.

Zuletzt soll beschrieben werden, wie sich die Verkettung der Verknüpfungen in diesem Beispiel auf die Verteilung der Stückzahl des Produkts und damit auf die Auslastung der zugeordneten Produktionsanlagen auswirkt. Die je nach Planungsziel entweder gesuchte oder gegebene Stückzahl durchläuft das Element c^* maximaler Ordnung, in diesem Fall das Element c_4. Gemäß den sequenziellen Verknüpfungen SQ(c_2, c_3) und SQ$(c_1,$ 'Härten'$)$, aus welchen das Element c_4 bzw. c_2 resultiert, wird die Stückzahl in unveränderter Höhe an die Elemente c_2 und c_3 bzw. c_1 und den Prozessschritt 'Härten' weitergegeben. Als Folge der alternativen Verknüpfung AL('Drehen/A', 'Drehen/B'), deren Ergebnis das Element c_1 ist, verteilt sich die Stückzahl zu variablen Anteilen auf die Prozessschritte 'Drehen/A' und 'Drehen/B'. Bedingt durch die selektive Verknüpfung SL$(\varnothing,$ 'Prüfen', 0,8), aus welcher das Element c_3 folgt, durchläuft ein relativer Anteil der Stückzahl gleich einem Wert von 20 Prozent den letztlich verbleibenden Prozessschritt 'Prüfen'.

Bemerkung: Die eingeführte Notation beschreibt, wie Prozessschritte rekursiv miteinander verknüpft werden. Um auf formale Weise darzustellen, welche MAE einem Prozessschritt jeweils zugeordnet ist, wird die betreffende MAE im weiteren Verlauf in eckigen Klammern nach dem Prozessschritt ergänzt (z. B. 'Drehen/A' ['Drehmaschine 1']).

2.2.2 Prozesse und Wertströme

Begriff des Prozesses. Basierend auf dem Begriff des Prozessschritts und der rekursiven Verknüpfung von Prozessschritten eines Produkts soll im Kontext der vorliegenden Arbeit nachfolgend der Begriff des Prozesses definiert werden. Eine entsprechende Festlegung ist erforderlich, da in der Literatur neben der technischen eine unternehmerische Sicht existiert. Aus technischer Sicht beschreibt ein Prozess die Gesamtheit aller in einem System aufeinander einwirkenden Vorgänge, welche Materie, Energie oder Information umformen, transportieren oder speichern (Definition laut REFA, siehe Hammer 1997, S. 158). Aus unternehmerischer Sicht wird unter einem Prozess eine Tätigkeit verstanden, welche sich über funktionale Grenzen hinweg von einem zum anderen Ende in einem Unternehmen erstreckt und dazu dient, Werte für den Kunden zu schaffen (siehe Hammer 2010, S. 4). Unter

Zuhilfenahme der vorher eingeführten Begriffe wird die nachfolgende Definition formuliert, die einer Synthese der beiden Sichtweisen gleichkommt:

Definition 2.8: Prozess. Ein Prozess ist definiert als eine Folge verknüpfter Prozessschritte zur Fertigung und Montage eines Produkts. Durch sequenzielle Verknüpfungen werden Prozessschritte, die nacheinander ausgeführt werden sollen, zu einem Prozess verbunden; alternative und selektive Verknüpfungen erzeugen für variable bzw. fixe Anteile der Stückzahl verschiedene Folgen von Prozessschritten, d. h. verschiedene Prozesse.

Verfolgt man den Weg jedes Stücks durch Fertigung und Montage, dann ist ein Prozess das Ergebnis aller Entscheidungen, welcher der jeweils folgenden Prozessschritte am Punkt einer alternativen oder selektiven Verknüpfung auszuführen ist. Im Fall einer alternativen Verknüpfung kann diese Entscheidung unabhängig für jedes Stück getroffen werden. Dagegen muss im Fall einer selektiven Verknüpfung eine fix vorgegebene Quote in Bezug auf die jeweilige Stückzahl an diesem Punkt erfüllt werden.

Begriff des Wertstroms. Der Wertstrom ist eine Metapher, um die Menge aller Aktivitäten zur Schöpfung eines Wertes vom Standpunkt des Kunden zu beschreiben, in diesem Kontext die Prozesse zur Fertigung und Montage eines Produkts. In der Literatur existiert eine Vielzahl unterschiedlicher Definitionen, wobei als Übereinstimmung festzustellen ist, dass sie alle wertschöpfenden Aktivitäten und nicht wertschöpfenden Aktivitäten im Zuge der Leistungserstellung zusammenfassen (siehe u. a. Rother und Shook 1999, S. 3, Erlach 2010, S. 8 f, Singh et al. 2011, S. 799 f). Um die Erzeugung von intellektuellen Werten explizit einzubeziehen, definieren Rother und Shook (1999, S. 3) einen Wertstrom bestehend aus einem sogenannten Produktionsfluss und Entwicklungsfluss:

> »A value stream is all the actions (both value added and non-value added) currently required to bring a product through the main flows essential to every product: (1) the production flow from raw material into the arms of the customer, and (2) the design flow from concept to launch.«

Im Mittelpunkt der vorliegenden Arbeit steht der erste Teil der Definition, d. h. der Produktionsfluss. Wie bei Rother und Shook (1999) wird nicht die Wertschöpfungskette über die organisationalen Grenzen von Unternehmen hinweg betrachtet, sondern die Tür-zu-Tür-Produktion im eigenen Unternehmen untersucht. Die Aktivitäten in der Produktion lassen sich in die Bewegung von Material und die Planung und Steuerung untergliedern. Gemäß dieser Sichtweise beschreibt der Materialfluss den Weg jedes Stücks durch Fertigung und Montage zwischen Arbeitsplätzen und Lagerorten. Der Informationsfluss dient der Planung und Steuerung des Ablaufs, wobei der Zeithorizont vom jeweiligen Kontext abhängig ist (siehe Rother und Shook 1999, S. 4, Erlach 2010, S. 8, Singh et al. 2011, S. 799 f).

Der Begriff des Wertstroms wird oft im Zusammenhang des Wertstrommanagements (engl. *Value Stream Management*) gebraucht, einer Methode zur Aufnahme, Gestaltung und Planung von Wertströmen in der Produktion. Das allgemeine Ziel dieser Methode ist die Reduktion der Verschwendung, einem Begriff für vermeidbare nicht wertschöpfende Tätigkeiten. Dazu werden der Materialfluss (Arbeitsplätze und Bestände, Materialbereitstellung

usw.) und der Informationsfluss (Übermittlung von Daten, Losgrößen, Stückzahlen aller Varianten usw.) entsprechend ausgerichtet (für Konzepte, Modelle und Fallstudien wird auf die zuvor genannte Literatur verwiesen).

Statt auf die methodischen Details einzugehen, ist an dieser Stelle festzuhalten, dass der allgemein gebrauchte Wertstrombegriff sämtliche Aspekte von Produktentwicklung und Produktion zusammenfasst. Dieser Umstand führt zu einer großen Zahl verschiedener Darstellungsformen, die sich aufgrund zweier Punkte nur eingeschränkt für die strategische Planung der TEK eignen. Erstens erfordern solche Darstellungen oftmals genaue Informationen zum Produktionsablauf, die ggf. in der Phase der operativen Planung, aber nicht in der Phase der strategischen Planung vorliegen. Diese Aussage gilt insbesondere für Produkte und Prozesse, die zum Zeitpunkt der Planerstellung noch nicht existieren und zu denen keine Erfahrungswerte vorliegen. Als Beispiele seien die Steuerung der Prozesse und die Verwaltung der Bestände genannt, die noch nicht im Detail feststehen, wenn erste Überlegungen zur Errichtung eines Produktionsstandorts angestellt werden.

Zweitens werden wichtige Informationen in vielen Fällen zum Zweck der Vereinfachung zusammengefasst, wodurch diese verloren gehen. Beispielsweise werden MAEs ähnlicher Funktion häufig zu einer Gruppe vereinigt (siehe Erlach 2010, S. 8). Ebenso beschränkt sich die Abbildung von alternativen und selektiven Verknüpfungen einzelner Prozessschritte zumeist auf einfache Szenarien (siehe Erlach 2010, S. 81). Um die Auslastung kalkulieren zu können, müssen alle Prozessschritte einbezogen werden, denen eine MAE zugeordnet ist, jedoch wird dies nur selten dargestellt. Letztlich führt der in seinem Wesen mehrdeutige Begriff des Wertstroms zu Darstellungen, die einem anderen Zweck dienen und sich für eine Anwendung in diesem Kontext nicht eignen. Statt aber nur eine neue Darstellungsform zu ergänzen, muss der Begriff selbst konkretisiert werden.

Hierzu wird eine eindeutige Definition formuliert, die auf den Prozessschritten eines Produkts und deren verschiedenartiger, rekursiver Verknüpfung basiert. Das Konzept des Materialflusses und Informationsflusses wird dabei übernommen und auf die Struktur eines Systems übertragen, wie sie durch den eingeführten Formalismus beschrieben wird. Die entsprechende, in dieser Arbeit fortan gültige Definition lautet wie folgt:

Definition 2.9: Wertstrom. Ein Wertstrom ist definiert als die Gesamtheit aller Prozesse, welche aus der planmäßigen, rekursiven Verknüpfung der Prozessschritte zur Fertigung und Montage genau eines Produkts resultieren. Der Materialfluss beschreibt die möglichen Wege jedes Stücks des jeweiligen Produkts durch die Produktion, d. h. die Abfolge der Maschinen, Anlagen und Einrichtungen, die nacheinander durchlaufen werden, um aus allen Komponenten (materielle Eingabe) das Produkt (materielle Ausgabe) zu fertigen bzw. zu montieren. Zur Planung des Wertstroms ermöglicht es der Informationsfluss, die Stückzahlen des Produkts sowie aller alternativ verknüpften Prozessschritte (Informationseingabe) vorzugeben und die Auslastung (Informationsausgabe) zurückzugeben.

Wie auch der Begriff des Wertstroms werden die untergeordneten Begriffe des Materialflusses und Informationsflusses durch diese Definition präzisiert. Laut REFA verkettet der Materialfluss allgemein alle Vorgänge, die dem Zweck dienen, Güter zu gewinnen, zu bearbeiten, zu verarbeiten und zu verteilen (siehe Hammer 1997, S. 135). Mit Bezug auf die weiter oben definierten Begriffe werden darunter die sequenziell, alternativ und/oder selek-

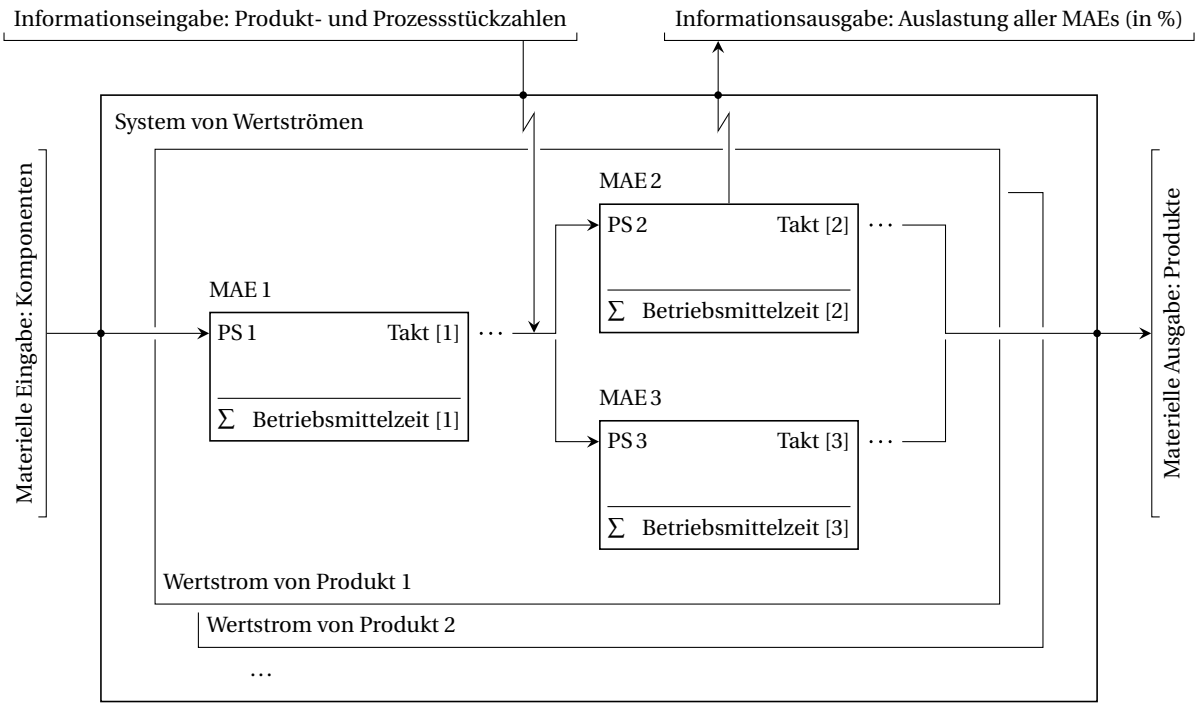


Abbildung 2.4: System von Wertströmen (Ebenen, Elemente und Ressourcen). Ein Wertstrom fasst die Prozesse eines gegebenen Produkts zusammen. Prozesse resultieren aus einem oder mehreren verknüpften Prozessschritten (PS), denen je eine MAE zugeordnet ist. Den Bezugsrahmen der Planung bildet ein System, dessen Struktur aus Ebenen (Wertströme), Elementen (Prozessschritte/Prozesse) und Ressourcen (MAEs) zusammengesetzt ist. Durch eine Schnittstelle werden die Ein- und Ausgaben des Materialflusses und Informationsflusses mit der Umwelt ausgetauscht.

tiv verknüpften Prozessschritte zur Herstellung des Produkts und die Zuordnung von MAEs zu diesen Prozessschritten verstanden. Der Informationsfluss bezieht sich im Allgemeinen auf die Erfassung, Sammlung, Verarbeitung, Speicherung und die Verteilung von Daten zur Planung und zur Steuerung (siehe Erlach 2010, S. 6). Im Kontext dieser Arbeit betrifft dies zum einen die Vorgabe einer Stückzahl für jedes Produkt und für jeden Operanden einer alternativen Verknüpfung im Wertstrom des Produkts, zum anderen die Ermittlung der daraus resultierenden, mittleren Auslastung aller MAEs.

System von Wertströmen. Bevor im Anschluss die Eingaben und Ausgaben des Materialflusses und des Informationsflusses detailliert erläutert werden, müssen die zuvor eingeführten Begriffe in Beziehung zueinander gesetzt werden. Wertströme fassen im Verständnis dieser Arbeit sämtliche Prozesse eines gegebenen Produkts zusammen (siehe Definition 2.9). Prozesse resultieren ihrerseits aus einem oder mehreren Prozessschritten, die durch sequenzielle, alternative und/oder selektive Verknüpfungen miteinander verbunden sind (siehe Definition 2.2–2.8). Zur Ausführung eines Prozessschritts ist jeweils eine MAE erforderlich, welche dem betreffenden Prozessschritt zugeordnet ist und deren Auslastung mit fortdauernder Nutzung ansteigt. MAEs können im Planungsintervall mehreren Prozessschritten

zugewiesen sein, dürfen aber zu jedem beliebigen Zeitpunkt nur für einen einzigen Prozessschritt genutzt werden (siehe Definition 2.1). Um zu einem Ergebnis in Bezug auf die Auslastung zu gelangen, müssen daher alle Wertströme, die dieselben MAEs beanspruchen, in einer Weise geplant werden, welche die zwischen ihnen bestehenden Abhängigkeiten vollständig berücksichtigt. Das heißt für die Planung im hier gegebenen Kontext, dass im allgemeinen Fall alle Wertströme an einem Produktionsstandort in einem System vereinigt werden müssen. Dessen Struktur lässt sich in Ebenen, Elemente und Ressourcen untergliedern, wobei die weiter oben definierten, spezifischen Ausdrücke diesen allgemeinen Begriffen folgendermaßen zugeordnet werden können:

(1) *Ebenen* (Wertströme): Gleich aufeinanderliegenden Ebenen überlagern sich Wertströme von Produkten, deren Herstellung dieselben MAEs erfordert.
(2) *Elemente* (Prozessschritte/Prozesse): Das System besteht aus nicht weiter zu teilenden Prozessschritten, die rekursiv zu Prozessen verknüpft werden.
(3) *Ressourcen* (MAEs): Mit ihrer endlichen Betriebsmittelzeit repräsentieren MAEs begrenzte Ressourcen, die durch Nutzung zunehmend ausgeschöpft werden.

Über eine Schnittstelle, welche dem Austausch der Ein- und Ausgaben des Materialflusses und des Informationsflusses der eingeschlossenen Wertströme dient, steht das jeweilige System in Verbindung mit seiner Umwelt. Abbildung 2.4 illustriert schematisch die Struktur eines solchen Systems sowie dessen Schnittstelle.

2.3 Systemschnittstelle: Funktionen der Eingaben und Ausgaben

Ein System von Wertströmen, wie es in dieser Arbeit verstanden wird, ist ein sogenanntes offenes System. Das heißt, es steht mit seiner Umwelt durch den kontinuierlichen Austausch festgelegter Ein- und Ausgangsgrößen in Verbindung (in der Terminologie der Informatik Ein- bzw. Ausgaben genannt). Die allgemeine Funktion eines offenen Systems ist durch die eindeutige Zuordnung von Ausgaben zu Eingaben gekennzeichnet (siehe Ehrlenspiel und Meerkamm 2013, S. 24). In diesem Abschnitt soll dargelegt werden, welche Ein- und Ausgaben ein System von Wertströmen aus Sicht der strategischen Planung der TEK entlang des Materialflusses und des Informationsflusses austauscht. Die Aussagen basieren jeweils auf der betriebswirtschaftlichen Praxis bei der Bosch Rexroth AG.

2.3.1 Materialfluss: Bereitstellung von Komponenten für Produkte

In Abschnitt 2.2 wurde der Begriff des Wertstroms eingeführt, um den Ablauf zur Fertigung und Montage eines Produkts durch rekursive Verknüpfung von Prozessschritten im Materialfluss zu beschreiben (siehe Definition 2.9). Die materiellen Ausgaben eines Systems von Wertströmen sind folglich die fertigen Produkte und die Eingaben die jeweils benötigten Komponenten, wie in Abbildung 2.4 dargestellt ist.

Die Begriffe von Komponente und Produkt werden wie folgt definiert. Gemäß der Norm DIN 199-1:2002-03 legt die Strukturstückliste die Zuordnung von allen Teilen und Gruppen

zu einem gegebenen Erzeugnis fest. Ein Teil ist entsprechend dieser Norm ein »Gegenstand, für dessen weitere Aufgliederung [...] kein Bedürfnis besteht« (DIN 199-1:2002-03, S. 15), und eine Gruppe ist ein »aus zwei oder mehr Teilen und/oder Gruppen niedrigerer Ordnung bestehender Gegenstand« (DIN 199-1:2002-03, S. 12). Sowohl Teile als auch Gruppen werden in der vorliegenden Arbeit als *Komponenten* bezeichnet. Das heißt, in der gleichen Weise, wie Gruppen Teile und untergeordnete Gruppen enthalten, können solche Komponenten ggf. aus anderen Komponenten zusammengesetzt sein.

Ein Erzeugnis ist gemäß der obigen Norm ein »durch Produktion entstandener gebrauchsfähiger bzw. verkaufsfähiger Gegenstand« (DIN 199-1:2002-03, S. 11). Unter dem Begriff des *Produkts* wird die Definition im Folgenden auf einen Gegenstand erweitert, der wie ein Erzeugnis durch Produktion entstanden ist, im Unterschied zu diesem aber nicht notwendigerweise gebrauchsfähig oder verkaufsfähig sein muss. Stattdessen kann ein Produkt auch allein zu dem Zweck hergestellt werden, um in anderen, übergeordneten Produkten verarbeitet zu werden. Gemäß diesem allgemeinen Verständnis ist jede Komponente, die nicht nur zugekauft, sondern zumindest z. T. im eigenen Unternehmen gefertigt bzw. montiert wird, ebenso ein Produkt. Um diesen Zusammenhang im Schema eines Systems darzustellen, wie es Abbildung 2.4 zeigt, müssen die Wertströme von untergeordneten und übergeordneten Produkten miteinander verbunden werden.

Dem schließt sich die Frage an, wie die herzustellende Stückzahl eines Produkts ausgehend von der Kundennachfrage ermittelt wird. Dies führt zur sogenannten *Stücklistenauflösung*, die in der Norm DIN 199-5:1981-10 festgelegt ist und zwischen dem Primärbedarf und dem Sekundärbedarf eines Produkts unterscheidet. Der *Primärbedarf* ist die Eingangsgröße für die nachfolgende Berechnung und beschreibt die Menge, die letztendlich für den Verkauf an den Kunden bestimmt ist. Demgegenüber bezeichnet der *Sekundärbedarf* die Menge eines Produkts, welche zur Montage übergeordneter Produkte benötigt wird. Die Strukturstückliste definiert, aus welchen Komponenten ein Produkt auf jeder Strukturstufe besteht und welche Menge jeder Komponente zur Herstellung eines Stücks des Produkts erforderlich ist. Um den Sekundärbedarf einer Komponente zu ermitteln, muss der Bedarf aller übergeordneten Produkte jeweils mit der Menge aus der Strukturstückliste des Produkts, dem sogenannten Stücklistenfaktor, multipliziert und im Anschluss addiert werden (siehe DIN 199-5:1981-10, S. 3 f). Unter der *Produktstückzahl* wird im Weiteren die Menge verstanden, die zur Deckung des gesamten Bedarfs aus Primärbedarf und Sekundärbedarf dient.

In Bezug auf die strategische Planung der TEK ist anzumerken, dass im Allgemeinen nur solche Produkte einzubeziehen sind, die unter Nutzung von MAEs durch Produktion in Wertströmen des eigenen Unternehmens bereitgestellt werden. Produkte, die in vollem Umfang zugekauft werden, müssen nicht in der Planung Berücksichtigung finden. Begrenzt aber eine solche Komponente die Kapazität eines Produkts, ist es zweckmäßig, den Zulieferer als Wertstrom mit einer MAE und entsprechender Betriebsmittelzeit in die Planung aufzunehmen. Der Begriff des Produkts bezieht sich weiterhin i. d. R. nicht auf eine einzelne, spezifische Variante. Vielmehr werden in der Phase der strategischen Planung der TEK Varianten und Produkte soweit möglich zu Produktfamilien zusammengefasst, woraus an den Standorten der Bosch Rexroth AG typischerweise 50 bis 100 voneinander zu unterscheidende Produkte resultieren. Damit wird das Ziel verfolgt, den Planungsaufwand bei gleichzeitig akzeptabler Genauigkeit gemessen am Einsatzzweck zu minimieren.

2.3.2 Informationsfluss: Planung der Produktion

Der Informationsfluss eines Wertstroms dient der Planung und der Steuerung der Produktion, wobei der Fokus im Kontext dieser Arbeit, wie zu Eingang des Kapitels dargelegt, auf der strategischen Planung der TEK liegt (siehe Abschnitt 2.1). Analog zum Materialfluss tauscht ein System von Wertströmen entlang des Informationsflusses Ein- und Ausgaben aus. Wie zuvor erläutert, ist die Struktur eines solchen Systems durch die rekursive Verknüpfung von Prozessschritten bestimmt (siehe Abschnitt 2.2). Die Informationseingaben sind somit neben den Produktstückzahlen die Prozessstückzahlen, die allen Operanden alternativer Verknüpfungen zugeordnet werden und festlegen, welcher variable Anteil der Stückzahl eines Produkts durch das jeweilige Element verläuft (siehe Definition 2.6). Werden Elemente sequenziell oder selektiv verknüpft, ist die Verteilung der Produktstückzahl fix vorgegeben (siehe Definition 2.5 bzw. 2.7). Im Folgenden wird eine Lösung der Prozessstückzahlen, d. h. eine mögliche Verteilung, als gültig bezeichnet, wenn diese sämtliche Bedingungen erfüllt, die in den Definitionen der Verknüpfungstypen genannt werden. Aus den festgelegten Stückzahlen resultiert die Auslastung der genutzten MAEs im Planungszeitraum, die als Informationsausgabe von dem geplanten System zurückgegeben wird.

Konkret werden zur strategischen Planung der TEK, wie sie in der vorliegenden Arbeit definiert ist, drei Planungsziele verfolgt. Gesucht sind (1) die maximalen Kapazitäten, (2) die minimalen Investitionen und (3) die optimale Auslastung (siehe Abschnitt 2.1). Diese drei Planungsziele stehen bezüglich der Reihenfolge ihrer Betrachtung in einer prozessualen Abhängigkeit und z. T. in einer mathematischen Abhängigkeit. Aus prozessualer Sicht gilt es im ersten Schritt, die maximalen Kapazitäten zu ermitteln, um sie mit dem erwarteten Bedarf zu vergleichen und über Eigenproduktion und Zukauf zu entscheiden. Unter Berücksichtigung weiterer, unternehmerischer Faktoren, wie beispielsweise den Kosten, dem verfügbaren Personal und zu schützenden intellektuellen Werten, werden auf dieser Grundlage die geplanten Stückzahlen aller selbst herzustellenden Produkte festgelegt. Im zweiten Schritt müssen die minimalen Investitionen in MAEs bestimmt werden, die zur Fertigung bzw. Montage der Stückzahlen benötigt werden. Ist die Zahl existierender und zusätzlich zur Verfügung stehender MAEs bekannt, muss im dritten Schritt untersucht werden, wie die Stückzahlen der Produkte auf die einzelnen Prozessschritte zu verteilen sind, um die daraus resultierende Auslastung aller MAEs zu optimieren.

Da wie angesprochen unternehmerische, in ihrer Natur qualitative Faktoren in die Planung einbezogen werden müssen, ist aus mathematischer Sicht die Maximierung der Kapazitäten in Bezug auf die jeweiligen Ein- und Ausgangsgrößen unabhängig von den anderen zwei Planungszielen. Dagegen fließen die ermittelten minimalen Investitionen als Eingangsgröße unmittelbar in die Optimierung der Auslastung aller genutzten MAEs ein. Im Folgenden werden die drei Planungsziele mit Hilfe von Zielfunktionen und Nebenbedingungen auf Basis der Ein- und Ausgaben des Informationsflusses beschrieben.

Maximale Kapazitäten. Um vom strategischen Standpunkt über Produktion und Zukauf zu entscheiden, ist zuerst die Frage von Bedeutung, welche Kapazitäten in einem Unternehmen für die Produktion zur Verfügung stehen. Unter der betrachteten TEK eines Produkts wird die maximale Stückzahl des Produkts verstanden, die unter der Bedingung gefertigt bzw.

montiert werden kann, dass die daraus folgende Auslastung keiner MAE einen Wert von eins, d. h. 100 Prozent der Betriebsmittelzeit, überschreitet.

Die TEK wird nicht nur durch die Auslastung der MAEs beschränkt, die zur Fertigung und Montage des Produkts selbst genutzt werden. Zudem müssen alle Komponenten, die laut Strukturstückliste benötigt werden, zur Deckung des Sekundärbedarfs mindestens in dieser Menge verfügbar sein. Wenn die Stückzahl einer Komponente genau dem Sekundärbedarf entspricht und es in den Grenzen der zulässigen Auslastung der MAEs nicht möglich ist, die Stückzahl dieser Komponente weiter zu steigern, dann gilt das auch für die Stückzahlen aller übergeordneten Produkte, welche die Komponente enthalten. Das heißt, in dem Fall ist auch die Kapazität für die Produkte erschöpft. Das trifft selbst dann zu, wenn die Auslastung all derjenigen MAEs, die zur Fertigung und Montage dieser Produkte genutzt werden, noch nicht den Höchstwert von eins erreicht.

Dementsprechend ist das erste Planungsziel, die Produktstückzahlen unter drei Bedingungen zu maximieren: (1) Die Lösung muss gültig sein (s. o.); (2) die Auslastung keiner MAE darf einen Wert von eins überschreiten; (3) die Stückzahlen der erforderlichen Komponenten müssen mindestens dem Sekundärbedarf aller Produkte entsprechen.

Bemerkung: Wie in Abschnitt 2.2 erläutert wurde, müssen zur Planung die Wertströme all derjenigen Produkte in einem System vereinigt werden, die dieselben MAEs beanspruchen und als Folge einander bedingen. Für die Suche maximaler Kapazitäten gilt insbesondere, dass die TEK eines Produkts ggf. erhöht werden kann, indem die TEK anderer Produkte innerhalb eines gegebenen Systems verringert wird. Das ist genau dann möglich, wenn dieselben MAEs die maximalen Stückzahlen zweier Produkte begrenzen. Für diesen Fall muss eine Strategie zur Auswahl einer Lösung festgelegt werden.

Minimale Investitionen. Falls die geplanten Produktstückzahlen die TEK übersteigen und alle Maßnahmen zur Erhöhung der Stückzahl mit verfügbaren Mitteln (z. B. durch die Verkürzung von Taktzeiten oder die Einführung von zusätzlichen Schichten) in der Planung ausgeschöpft wurden, verbleiben allein Investitionen in neue MAEs. Diese gilt es zu minimieren, indem alle verfügbaren MAEs möglichst effizient genutzt werden.

Der Umfang der Investitionen wird wie folgt durch die Überlastung der MAEs bestimmt. Überschreitet die Auslastung u einer MAE einen Wert von eins, müssen $\lceil u \rceil - 1$ neue MAEs desselben Typs installiert werden. Das heißt, überlastete MAEs werden durch Investitionen gedanklich vervielfacht. Wenn Prozessschritte im Wertstrom eines Produkts alternativ verknüpft sind und ihnen verschiedene MAEs zugeordnet werden, dann muss der Planer festlegen, welche MAE im Bedarfsfall überlastet werden soll.

Das zweite Planungsziel ist somit, den Betrag zu minimieren, um welchen die Auslastung von MAEs einen Wert von eins überschreitet. Die Stückzahlen aller Produkte einschließlich der erforderlichen Komponenten werden vom Planer vorgegeben. Einzige Bedingung ist, dass die ermittelte Lösung, wie weiter oben erläutert, gültig sein muss.

Bemerkung: In dem Fall, dass eine Gruppe von MAEs alternativ verknüpften Prozessschritten eines oder mehrerer Produkte zugeordnet ist, legt der Planer für jedes der Produkte wie erwähnt jeweils diejenige MAE fest, die bei Bedarf überlastet werden soll. Jedoch ist es ggf.

möglich, die Überlastung einer der MAEs zu reduzieren, indem eine zweite MAE aus der Gruppe in umso größerem Maß für andere Produkte genutzt und deren Überlastung erhöht wird. Dann ist für das Produkt der ersten MAE ein größerer Anteil der summierten Betriebsmittelzeit aller anderen MAEs in der Gruppe verfügbar, und als Folge wird die Überlastung der MAE reduziert. Da in diesem Fall mehrere Lösungen existieren, muss eine Strategie zur Auswahl einer Lösung festgelegt werden (vgl. maximale Kapazitäten).

Optimale Auslastung. Zuletzt stellt sich in der Planung die Frage, welche Prozessschritte ausgeführt werden sollen, um die Produkte unter Nutzung aller im Planungszeitraum verfügbaren MAEs zu fertigen bzw. zu montieren. Es gilt demnach, die Prozessstückzahlen festzulegen, um die geplanten Stückzahlen aller Produkte in den Wertströmen zu verteilen und die resultierende Auslastung der jeweils zugeordneten MAEs zu optimieren. Aus theoretischer Sicht erscheint es zunächst naheliegend, Werte zu wählen, welche den erwarteten Ressourcenbedarf minimieren, beispielsweise im Hinblick auf Betriebskosten und Energieverbrauch. Zu diesem Zweck müssten allerdings frühzeitig in der Planung z. T. noch vor der Installation und Inbetriebnahme von MAEs Daten zum Verbrauch im laufenden Betrieb vorliegen. Diese Daten sind in der Praxis zum gegebenen Zeitpunkt im Allgemeinen nicht in der geforderten Genauigkeit verfügbar. Das gilt insbesondere dann, wenn Wertströme neuer Produkte oder neue Prozesse geplant werden.

Aus diesem Grund wird folgendes Vorgehen vorgeschlagen, das im Zuge dieser Arbeit zusammen mit der Software AURELIE erfolgreich bei der Bosch Rexroth AG eingeführt wurde und in der Planung angewandt wird. Grundlage ist eine Priorisierung der alternativ verknüpften Prozessschritte und der jeweils zugeordneten Prozessstückzahlen in den Wertströmen aller gegebenen Produkte. Die Prioritäten werden von der Abteilung für Fertigungsplanung definiert, um die unternehmerischen Zielvorgaben umzusetzen, welche von der Leitung des Standorts und übergeordneten Instanzen festgelegt werden. Diese Zielvorgaben betreffen zumeist die Variabilität, Qualität, Geschwindigkeit und Wirtschaftlichkeit der Produktionsprozesse (siehe Erlach 2010, S. 14). Die optimale Auslastung ist genau dann erreicht, wenn die Verteilung der Stückzahlen die Prioritäten vollständig berücksichtigt.

Auf dieser Basis ist das dritte Planungsziel, die Auslastung aller MAEs gemäß einer Iterationsvorschrift zu maximieren. Analog zum vorigen Planungsziel werden die Stückzahlen aller Produkte einschließlich der Komponenten vorgegeben. Zunächst ist eine initiale Lösung gesucht, welche wie oben erläutert gültig sein muss. Im ersten Iterationsschritt wird die Prozessstückzahl mit der Priorität eins maximiert. In den darauffolgenden Iterationsschritten wird das Maximum der Prozessstückzahl mit jeweils aufsteigender Priorität bestimmt, wobei die Teillösungen aller vorherigen Iterationsschritte Nebenbedingungen darstellen. Die Iteration endet mit der vorletzten Priorität, da der Wert der letzten Prozessstückzahl nach Maximierung aller vorhergehenden eindeutig bestimmt ist.

Dabei existieren für die initiale Lösung und für die Teillösungen aller Iterationsschritte zwei Bedingungen: (1) Die Lösung muss gültig sein (s. o.); (2) die Auslastung keiner MAE darf einen Höchstwert von eins überschreiten, wobei dieser Wert um den jeweils minimalen Betrag zu erhöhen ist, sodass für die gegebenen Produktstückzahlen eine gültige Lösung existiert (minimale Investitionen als Eingangsgröße).

Bemerkung: Da die Stückzahlen der Produkte einschließlich erforderlicher Komponenten im Fall der minimalen Investitionen und der optimalen Auslastung vom Planer selbst vorgegeben werden, ist es nicht notwendig, die Verfügbarkeit der Komponenten wie im Fall der maximalen Kapazitäten als eine Bedingung zu definieren. Es wird stattdessen vorausgesetzt, dass in einem vorbereitenden Schritt die Stückliste aufgelöst wird, um den Sekundärbedarf zu ermitteln und die Stückzahlen bedarfsgerecht festzulegen.

Zur Verfolgung der Planungsziele gilt es, das geplante System zu modellieren und das resultierende Modell gemäß den obigen, in natürlicher Sprache beschriebenen Zielfunktionen zu optimieren. Um die mathematischen Grundlagen zu schaffen, soll im nächsten Abschnitt die Kalkulation für den einfachen Fall eines Prozessschritts erläutert werden.

2.4 Grundlagen der Kalkulation: einfacher Fall eines Prozessschritts

Zunächst werden in diesem Abschnitt die Zeitgrößen definiert, die im Kontext der Arbeit als kalkulatorische Grundlage für die Planung der TEK dienen: die Taktzeiten der Prozessschritte, der Plan-Nutzungsgrad und die Betriebsmittelzeiten der jeweils genutzten MAEs. Danach soll erläutert werden, wie die Kapazität, die Auslastung und die Investitionen im einfachen Fall eines einzelnen Prozessschritts zu bestimmen sind, um zum Abschluss anhand repräsentativer Zahlen und eines gedanklichen, praxisnahen Beispiels zu prüfen, ob die hierfür formulierten Voraussetzungen im Allgemeinen erfüllt sind.

Zur exakten Formulierung der Aussagen in diesem und den sich anschließenden Abschnitten werden folgende Symbole mit eigener Bedeutung eingeführt:

r	MAE, z. B. Bearbeitungszentrum, $r \in R$, ggf. mit Index 1, 2 usw.
R	Menge aller verfügbaren MAEs, z. B. {'Bearbeitungszentrum', 'Drehmaschine', …}
T_{BM}	Betriebsmittelzeit, $T_{BM} \in \mathbb{R}_{>0}$, ggf. mit Index 1, 2 usw.
t_{eff}	effektive, d. h. im langfristigen Mittel erreichbare Taktzeit, $t_{eff} \in \mathbb{R}_{>0}$
t_{opt}	optimale, d. h. im besten Fall erreichbare Taktzeit, $t_{opt} \in \mathbb{R}_{>0}$
$u_r(y)$	Auslastung der MAE r für eine gegebene Produktstückzahl y, $u_r(y) \in \mathbb{R}_{\geqslant 0}$
y	Stückzahl des aktuell betrachteten Produkts, $y \in \mathbb{R}_{\geqslant 0}$
y_{max}	maximale Produktstückzahl (TEK), $y_{max} \in \mathbb{R}_{\geqslant 0}$
η_{plan}	Plan-Nutzungsgrad, festgelegt als Zielvorgabe, $\eta_{plan} \in (0, 1]$

2.4.1 Taktzeiten, Nutzungsgrad und Betriebsmittelzeit

Optimale Taktzeit t_{opt}. Die optimale Taktzeit t_{opt} eines Prozessschritts bezeichnet die »vorgegebene Zeit zwischen Entnahme von Teil zu Teil [nicht im Verständnis von Komponenten, sondern von Stück zu Stück] bei störungsfreiem Betrieb und optimiertem Ablauf« (Bosch-interne Regelung, zum Begriff der Taktzeit siehe auch Hammer 1997, S. 189). Die Definition schließt einen von der MAE abhängigen, d. h. unbeeinflussbaren und einen von der Arbeitsperson abhängigen, d. h. beeinflussbaren Zeitanteil ein. Da in dieser Arbeit aber die Planung der TEK im Fokus steht, d. h. die Planung der maximalen durch MAEs bedingten Stückzahlen,

werden für den Zeitanteil, den nicht die MAE, sondern die Arbeitsperson bestimmt, ideale Voraussetzungen angenommen (siehe Abschnitt 2.2, Begriff des Prozessschritts). Entsprechend wird immer die kürzestmögliche, im langfristigen Mittel mit der jeweils gegebenen MAE erzielbare Taktzeit für die Planung der TEK verwendet.

Im Folgenden soll zum besseren Verständnis ein Kurzüberblick zu den Zeitanteilen gegeben werden, aus welchen sich die Nutzungszeit einer MAE zusammensetzt. Die Nutzungszeit ist definiert als die Summe aller Sollzeiten, in denen eine MAE »planmäßig genutzt oder zur Hauptnutzung vorbereitet wird« (alle Definitionen gemäß Bosch-interner Regelung, siehe auch Hammer 1997, S. 212 f, REFA 2012, S. 43, 59). Sie wird untergliedert in die planmäßig von Menschen beeinflussbare und die unbeeinflussbare Nutzungszeit. Zusätzlich wird mit Blick auf den zu erfüllenden Zweck zwischen der Hauptnutzungszeit und der Nebennutzungszeit unterschieden. Die beeinflussbare (bzw. unbeeinflussbare) Hauptnutzungszeit beschreibt die Summe all derjenigen Sollzeiten, in denen eine MAE mit (bzw. ohne) »menschlicher Beeinflussung wertschöpfend arbeitet« (z. B. Bohren mit manueller bzw. mit automatischer Vorschubregelung). Daneben umfasst die beeinflussbare (bzw. unbeeinflussbare) Nebennutzungszeit alle Sollzeiten, in denen eine MAE mit (bzw. ohne) »menschlicher Beeinflussung planmäßig zur Hauptnutzung vorbereitet wird« (z. B. durch das manuelle Einspannen eines Werkstücks bzw. das automatische, robotergestützte Beladen der MAE).

Plan-Nutzungsgrad η_{plan}. Der Plan-Nutzungsgrad η_{plan} einer MAE stellt in der unternehmerischen Praxis der Bosch Rexroth AG eine Zielvorgabe des Geschäftsbereichs[1] dar, welche dazu dient, die Anzahl der erforderlichen Betriebsmittel, in diesem Kontext der MAEs, und die TEK gemäß dem Plan zu ermitteln (Bosch-interne Regelung, zum Begriff des Nutzungsgrads siehe auch REFA 2012, S. 60). Er spiegelt im langfristigen Mittel den relativen Zeitanteil des störungsfreien, optimalen Betriebs bezogen auf die Betriebsmittelzeit einer MAE wider (vgl. physikalischen Wirkungsgrad, i. d. R. bezeichnet mit dem Symbol η). In diesem Verständnis wird der Begriff in der vorliegenden Arbeit verwendet.

Verluste, welche den Plan-Nutzungsgrad η_{plan} reduzieren, werden wie folgt untergliedert (siehe Praxisliteratur zu *Total Productive Maintenance* und *Overall Equipment Effectiveness*, Nakajima 1988 sowie u. a. Hartmann 2013, May und Schimek 2015, Koch 2016):

(1) geplante Stillstände (z. B. Wartung und Instandhaltung),
(2) Verfügbarkeitsverluste (z. B. Rüsten, große Störungen),
(3) Leistungsverluste (z. B. Taktzeitverluste, Kleinstörungen),
(4) Qualitätsverluste (Ausschuss und Nacharbeit).

Für die Planung der TEK wird angenommen, dass alle Potenziale zur Vermeidung und Verminderung von Verlusten ausgeschöpft werden. Hintergrund ist, dass die maximalen Stückzahlen gesucht sind, welche allein durch die eingesetzten MAEs begrenzt werden. Es gilt, die Investitionen zu minimieren, die sich mit den verfügbaren MAEs unter keinen Umständen vermeiden lassen, und die Auslastung der MAEs durch Verteilung der Stückzahlen zu optimieren. Entsprechend wird für den Plan-Nutzungsgrad η_{plan} gemäß Bosch-interner

[1] Zum Zeitpunkt der Entwicklung gliederte sich die Bosch Rexroth AG in drei Geschäftsbereiche: Mobile Applikationen, Industrielle Applikationen und Erneuerbare Energien.

Regelung grundsätzlich ein Wert von 80 Prozent vorausgesetzt. In einer hochmechanisierten Fertigung gilt ein noch größerer Wert von 85 Prozent, selbst wenn dieser in der Praxis oftmals nur schwer zu erreichen ist. Erfahrungsgemäß akzeptiert die Geschäftsleitung zur Rechtfertigung von Investitionen nur in Ausnahmefällen einen geringeren Wert, und zwar ausschließlich unter der Bedingung, dass eine Reduktion der verbleibenden Verluste nicht möglich oder nicht wirtschaftlich umsetzbar ist (betrifft z. B. Ausschuss im Ablauf der Montage, welcher durch fehlerhafte zugekaufte Komponenten verursacht wird).

Effektive Taktzeit t_{eff}. Zur Verkürzung der Notation wird mit Blick auf die nachfolgenden Beispiele die effektive Taktzeit t_{eff} eingeführt. Diese Taktzeit wird jeweils einem Prozessschritt zugeordnet und ist analog zur zuvor eingeführten optimalen Taktzeit t_{opt} als Zeit zwischen der Entnahme von Stück zu Stück definiert. Im Unterschied zu dieser bezieht sie sich nicht auf den störungsfreien, optimalen Betrieb, sondern schließt alle Verluste ein und gilt im langfristigen Mittel. Mit Hilfe des Plan-Nutzungsgrads η_{plan}, welcher das Verhältnis der störungsfreien, optimalen Zeit zur gesamten Betriebsmittelzeit einer MAE und entsprechend das Verhältnis der zwei Taktzeiten zueinander ausdrückt, kann folgende Gleichung zur Bestimmung der effektiven Taktzeit t_{eff} angegeben werden:

$$t_{eff} = \frac{t_{opt}}{\eta_{plan}}. \tag{2.17}$$

Betriebsmittelzeit T_{BM}. Die Betriebsmittelzeit T_{BM} beschreibt die »Zeit, in der das Betriebsmittel [im gegebenen Kontext die MAE] zweckentsprechend zur Verfügung steht« (Bosch-interne Regelung, mit identischem Wortlaut außerdem definiert durch REFA, siehe Hammer 1997, S. 59 f). Neben der Belegungszeit, in welcher dieser Zweck durch die Herstellung von Produkten erfüllt wird, schließt die Betriebsmittelzeit T_{BM} auch die Brachzeiten außerhalb des störungsfreien, optimalen Betriebs ein (siehe Plan-Nutzungsgrad η_{plan}, Verluste in der Produktion). Die Zeit ist vom Schichtmodell abhängig, welches auf den Arbeitstagen im Kalenderjahr basiert, und wird im Planungsintervall zusammengefasst (meist quartalsweise, halbjährlich oder jährlich, siehe Abschnitt 2.1, strategische Planung).

Um die TEK zu bestimmen, wird grundsätzlich vorausgesetzt, dass sämtliche MAEs gemäß einem Schichtmodell genutzt werden, welches die maximal mögliche Schichtzeit an jedem Arbeitstag (entspricht i. d. R. drei Schichten je Arbeitstag zu acht Stunden je Schicht) und die maximale Zahl an Arbeitstagen im Planungsintervall (ggf. einschließlich Wochenendarbeit) vorsieht. Folglich ist es zur Planung der TEK möglich, für die meisten MAEs dieselbe Betriebsmittelzeit T_{BM} vorauszusetzen. Im Einzelfall kann jedoch der Betrieb einer MAE in einem solchen Schichtmodell technisch oder organisatorisch unwirtschaftlich sein. Im Allgemeinen muss deshalb von einer Betriebsmittelzeit T_{BM} ausgegangen werden, deren Wert von der betrachteten MAE abhängig ist. Um dies zu berücksichtigen, wird jeder MAE r_1, r_2 usw. einer Menge R, welche jeweils die Gesamtheit aller am Standort verfügbaren und im Planungszeitraum zu installierenden MAEs bezeichnet, eine eigene Betriebsmittelzeit $T_{BM,1}$, $T_{BM,2}$ usw. mit entsprechendem Index 1, 2 usw. zugeordnet.[2]

[2] Aktuell ist bei der Bosch Rexroth AG die Software AURELIE in der Version 1.2 im Einsatz, welche die Vorgabe eines Standardschichtmodells und eines erweiterten Schichtmodells für alle MAEs erlaubt.

Schichtmodell	Wochentage	Arbeitstage, pro Jahr	Arbeitstage, pro Quartal	Schichten, pro Arbeitstag	Schichtdauer, in h (Mittelwert)	Betriebsmittelzeit, in h
5-S	Mo–Fr	249	62,25	1	8,0	489
10-S	Mo–Fr	249	62,25	2	8,0	996
15-S	Mo–Fr	249	62,25	3	7,5	1401
18-S	Mo–Sa	299	74,75	3	7,5	1682
21-S	Mo–So	365	91,25	3	7,5	2053

Tabelle 2.1: Betriebsmittelzeiten nach Schichtmodellen (Beispielzahlen). Zentrale Vorgaben zur strategischen Planung der TEK in Quartalsintervallen, gemäß Bosch-interner Regelung im Fall manueller oder teilautomatisierter Fertigung (Schichtmodelle 5-S, 18-S und 21-S ergänzt, wie bei der Bosch Rexroth AG in der Praxis zumeist verwendet). Mit Ausnahme des Schichtmodells 21-S, welches an jedem Tag des Kalenderjahres einen Betrieb in drei Schichten vorsieht, ist in der Zahl der Arbeitstage bereits berücksichtigt, dass einige Feiertage auf Wochentage fallen.

In Tabelle 2.1 sind Vorgaben für die strategische Planung der TEK in Quartalsintervallen aufgeführt, die bei der Bosch Rexroth AG grundsätzlich für alle Standorte gelten. Typischerweise legt die Leitung eines Geschäftsbereichs, vertreten durch die Zentralabteilung für Fertigungskoordination, ein Standardschichtmodell und zusätzlich ein erweitertes Schichtmodell fest (zumeist 15-S bzw. 18-S, siehe oben stehende Tabelle), die jeder Standort im jeweiligen Verantwortungsbereich zur Erstellung der Planung verwenden muss.

2.4.2 Kapazität, Auslastung und Investitionen

Voraussetzungen. Nachdem die grundlegenden Zeitgrößen der Planung eingeführt wurden, soll im nächsten Schritt die Kalkulation von Kapazität, Auslastung und Investitionen im einfachen Fall erläutert werden. Dieser Fall ist dadurch charakterisiert, dass die folgenden zwei Voraussetzungen gelten: (1) Zur Fertigung und Montage eines betrachteten Produkts ist nur ein einziger Prozessschritt vorgesehen; (2) die hierzu erforderliche MAE r wird nicht zur Ausführung weiterer Prozessschritte genutzt. Das heißt, unter Berücksichtigung des jeweils gültigen Plan-Nutzungsgrads η_{plan} steht die MAE r innerhalb der gesamten Betriebsmittelzeit T_{BM} ohne Unterbrechung zur Verfügung, um diesen einen Prozessschritt mit einer vorgegebenen effektiven Taktzeit t_{eff} auszuführen.

Technische Kapazität $y_{\max}$. Im Folgenden soll die TEK, d. h. die maximale Stückzahl eines Produkts, formal mit dem Symbol $y_{\max}$ bezeichnet werden. Da die Kalkulation im Rahmen der strategischen Planung auf dem langfristigen Mittel basiert, kann die TEK $y_{\max}$ im oben beschriebenen Fall mit akzeptabler Genauigkeit durch das Verhältnis zweier Zeitgrößen bestimmt werden: der Betriebsmittelzeit T_{BM} der genutzten MAE r und der effektiven Taktzeit t_{eff} des auszuführenden Prozessschritts. Diese Berechnungsvorschrift entspricht im Ergebnis den Vorgaben für die Planung, die bei der Bosch Rexroth AG angewandt werden (Bosch-interne Regelung, TEK im Planungsintervall). Demnach gilt:

$$y_{\max} = \frac{T_{BM}}{t_{eff}}. \tag{2.18}$$

Auslastung $u_r(y)$. Gesucht ist eine Gleichung, um die Auslastung $u_r(y)$ der MAE r abhängig von einer gegebenen, nichtnegativen reellen Produktstückzahl y zu bestimmen. Die Lösung beruht darauf, dass die Auslastung $u_r(y)$ gemäß der hier betrachteten, ausschließlich technisch bedingten Kapazität einen Wert von eins annimmt, wenn die Produktstückzahl y der TEK $y_{\max}$ entspricht (siehe Abschnitt 2.1, strategische Planung). Wird zusätzlich ein direkter, proportionaler Zusammenhang zwischen der Funktion $u_r(y)$ und ihrem Argument y angenommen, d. h. ein stetiges Anwachsen der Auslastung mit steigender Stückzahl des Produkts vorausgesetzt, dann folgt daraus die Gleichung:

$$u_r(y) = \frac{y}{y_{\max}}. \tag{2.19}$$

Einsetzen von Gleichung (2.18) in Gleichung (2.19) führt zu folgendem Ausdruck, welcher die Auslastung $u_r(y)$ bezeichnet und von der Produktstückzahl y abhängig ist. Dieser Ausdruck dient für die Kalkulation als Grundlage und ist fallweise zu erweitern:

$$u_r(y) = \frac{t_{\text{eff}}\, y}{T_{\text{BM}}}. \tag{2.20}$$

Die Annahme eines proportionalen, d. h. kontinuierlichen Zusammenhangs ist zulässig, da Informationen zum diskreten Ablauf von Fertigung und Montage (insbesondere zur Reihenfolge der Aufträge, zu Losgrößen sowie zu geplanten und ungeplanten Stillständen wie z. B. Wartung bzw. Störungen) in der Phase der strategischen Planung nicht oder nur in ungenügender Genauigkeit vorliegen (siehe Abschnitt 2.2, Begriff des Wertstroms). Die Frage ist in diesem Kontext vielmehr, welchen Wert die Auslastung $u_r(y)$ im langfristigen Mittel des Planungsintervalls annimmt. Unter den formulierten Voraussetzungen des einfachen Falls eines Prozessschritts ist Gleichung (2.20) demzufolge gültig.

Investitionen. Wie in Abschnitt 2.3 erläutert, werden die notwendigen Investitionen zur Fertigung und Montage eines Produkts in einer gegebenen, geplanten Stückzahl von der Überlastung der jeweils genutzten MAEs abgeleitet (siehe Planungsziel der minimalen Investitionen). Überschreitet die Auslastung $u_r(y)$ einer MAE r einen Wert von eins, müssen entsprechend $\lceil u_r(y) \rceil - 1$ neue MAEs desselben Typs installiert werden, um das Produkt in der geplanten Stückzahl y zu fertigen bzw. zu montieren. Das heißt, bei Bedarf werden all diejenigen MAEs, die eine Überlastung erfahren, gedanklich vervielfacht.

Wenn mehrere Prozessschritte eines Produkts unter Nutzung von verschiedenen MAEs alternativ miteinander verknüpft werden, dann legt der Planer wie zuvor beschrieben fest, welche von diesen MAEs überlastet und als Ergebnis vervielfacht werden soll, sobald die Produktstückzahl y die TEK $y_{\max}$ übersteigt.

Prüfung der Voraussetzungen. Zu Beginn des Abschnitts wurden die Voraussetzungen genannt, welche den einfachen Fall eines einzigen Prozessschritts kennzeichnen und unter denen die obigen Gleichungen ohne weitere Anpassung gelten. Um die Begründung dafür zu erbringen, warum diese Gleichungen fallweise erweitert werden müssen, sollen die Voraussetzungen anhand von Zahlen aus der Praxis und eines gedanklichen Beispiels

Standort*	Region	Produktgruppe*	Planung für	Erstellt am	Produkte	PS	MAEs	max Produkte/ MAE	max PS/ MAE
Standort 1	EMEA	PG 1, Mobile Applikationen	Q4 2013	08.2012	36	391	**268**	14	14
Standort 2	EMEA	PG 2, Mobile Applikationen	Q1 2013	09.2012	57	520	**397**	4	4
Standort 3/1	AMER	PG 1, Mobile Applikationen	Q3 2013	09.2011	31	253	151	10	10
Standort 3/2	AMER	PG 3, Industrielle Applikationen	Q1 2013	09.2012	27	182	35	15	15
Standort 4	EMEA	PG 2, Mobile Applikationen	Q4 2012	09.2012	**62**	490	**167**	13	13
Standort 5	EMEA	PG 2, Mobile Applikationen	Q4 2012	09.2012	43	212	162	5	5
Standort 6/1	EMEA	PG 4, Mobile Applikationen	Q4 2013	08.2012	55	299	72	18	22
Standort 6/2	EMEA	PG 5, Erneuerbare Energien	Q4 2012	06.2012	53	**730**	54	**28**	**28**
Standort 7	APAC	PG 2, Mobile Applikationen	Q1 2014	09.2013	38	505	148	12	12
Standort 8	AMER	PG 4, Mobile Applikationen	Q4 2013	08.2012	33	84	33	4	4
Standort 9/1	EMEA	PG 6, Industrielle Applikationen	Q3 2013	04.2013	**152**	**1404**	60	**46**	**46**
Standort 9/2	EMEA	PG 5, Erneuerbare Energien	Q4 2012	10.2012	**119**	**1795**	87	**67**	**67**
Mittelwert					59	572	136	20	20

*anonymisiert

Tabelle 2.2: Produkte, Prozessschritte und Produktionsanlagen (Beispielzahlen). Die Tabelle zeigt eine Auswahl von Planungen für Produktgruppen (PG) an neun verschiedenen Standorten der Bosch Rexroth AG. Der Zeithorizont erstreckt sich in allen Fällen mindestens auf das Jahr der Erstellung. Betrachtet wird das Quartal mit der maximalen Zahl an Prozessschritten (PS). Diese wird verglichen mit der Zahl genutzter MAEs sowie mit der maximalen Zahl von Produkten und Prozessschritten, denen dieselbe MAE zugeordnet ist (max Produkte/MAE bzw. PS/MAE).

überprüft werden. In der Praxis ist i. d. R. nicht nur ein einziger Prozessschritt auszuführen, um ein Produkt zu fertigen und zu montieren, sondern eine Folge vieler verschiedener Prozessschritte. Für variable und fix vorgegebene Anteile der Stückzahl eines Produkts werden zudem oftmals unterschiedliche Folgen von Prozessschritten geplant. Die notwendigen MAEs werden zumeist für mehrere Prozessschritte eines oder mehrerer Produkte genutzt. Diese Merkmale einer variantenreichen Serienfertigung müssen in der Kalkulation ohne unzulässige Vereinfachungen berücksichtigt werden (siehe Abschnitt 2.1).

Tabelle 2.2 zeigt hierzu repräsentative Zahlen geplanter Systeme von Wertströmen. Dargestellt ist eine Auswahl von Planungen der Bosch Rexroth AG für verschiedene Produktgruppen an weltweit verteilten Produktionsstandorten. Zwar wurde in Abschnitt 2.2 erläutert, dass alle Wertströme an einem Standort in einem System vereinigt werden müssen, und im Allgemeinen gilt dies auch nach wie vor. Wenn aber zur Herstellung zweier Produktgruppen an einem Standort jeweils verschiedene MAEs genutzt werden, ist es möglich, die Produktgruppen unabhängig voneinander zu betrachten. Aus Sicht der Planung handelt es sich um voneinander getrennte Standorte, auch wenn sie sich am selben geografischen Ort befinden. In den aufgeführten Fällen werden durchschnittlich 59 Produkte gefertigt und montiert, wobei ein Produkt i. d. R. mehrere Varianten zusammenfasst. Eine MAE wird im Mittel aller Standorte für bis zu 20 Prozessschritte genutzt, im Fall der Produktgruppen an Standort 9/1 und 9/2 jeweils für bis zu 46 bzw. 67 Prozessschritte.

Entsprechend müssen Gleichung (2.18) und (2.20) zur Bestimmung der TEK $y_{\max}$ bzw. der Auslastung $u_r(y)$ fallweise erweitert werden. Im Allgemeinen ist es nicht möglich, die Produkte an einem Standort unabhängig voneinander unter Berücksichtigung nur einer MAE und nur eines Prozessschritts zu betrachten. Vielmehr müssen alle Abhängigkeiten

innerhalb eines Produktionssystems in die Kalkulation einbezogen werden. Zum einen bestehen diese zwischen verknüpften Prozessschritten eines Produkts, zum anderen zwischen Produkten, die sich dieselben MAEs teilen. Wie sich die Abhängigkeiten auswirken, soll nun an einem gedanklichen Beispiel erläutert werden.

Ein gegebenes Produkt wird in einem einzigen Prozessschritt mit einer Fräsmaschine r_1 gefertigt, entsprechend einer Komplettbearbeitung. Entgegen den eingangs formulierten Voraussetzungen werden die MAE r_1 wie auch eine Drehmaschine r_2 für mehrere Produkte genutzt. Im einfachen Fall eines einzigen Prozessschritts ist die TEK y_{max} das Ergebnis der optimalen Taktzeit t_{opt} und des Plan-Nutzungsgrads η_{plan}, die wie definiert zur effektiven Taktzeit t_{eff} zusammengefasst werden, sowie der Betriebsmittelzeit $T_{BM,1}$. Im allgemeinen Fall ist nicht vorauszusetzen, dass die MAE r_1 in der gesamten Betriebsmittelzeit $T_{BM,1}$ zur Verfügung steht. Stattdessen muss der von jedem Prozessschritt beanspruchte Zeitanteil berücksichtigt werden. Es soll untersucht werden, wodurch es möglich ist, die TEK y_{max} des Produkts bei vollständiger Auslastung der MAE r_1 noch weiter zu steigern.

Dabei wird angenommen, dass der Plan-Nutzungsgrad η_{plan} nicht zu erhöhen ist, da die Verluste in der Produktion nicht reduziert werden können. Genauso sei die Betriebsmittelzeit $T_{BM,1}$ unveränderlich vorgegeben, da weder weitere Schichten eingeführt noch Schichten verlängert werden sollen. Als Konsequenz verbleiben in dem Beispiel nur zwei Wege, um die TEK y_{max} des gegebenen Produkts zu steigern:

(1) Der für das Produkt verfügbare Zeitanteil der Betriebsmittelzeit $T_{BM,1}$ wird vergrößert. Dies entspricht einer höheren Priorisierung des Produkts gegenüber allen anderen Produkten, deren Bearbeitung laut Plan die Nutzung der MAE r_1 erfordert.

(2) Die effektive Taktzeit t_{eff} wird reduziert, indem mit der MAE r_1 anstelle der Komplettbearbeitung nur eine Fertigbearbeitung ausgeführt wird. In diesem Fall muss zuvor eine zusätzliche Vorbearbeitung mit der MAE r_2 erfolgen.

Im ersten Fall wird der Zeitanteil der Betriebsmittelzeit $T_{BM,1}$, der jeweils für andere Produkte zur Verfügung steht, reduziert. Als Folge der Abhängigkeiten, die durch die Verknüpfungen der Prozessschritte entstehen, gilt dies im zweiten Fall analog für die Betriebsmittelzeit $T_{BM,2}$. Soll für das Produkt mit der MAE r_1 nur eine Fertigbearbeitung ausgeführt werden, muss es zuerst eine Vorbearbeitung durchlaufen, wofür die MAE r_2 genutzt wird. Dies entspricht der alternativen Verknüpfung von zwei Prozessschritten, einer Komplettbearbeitung und einer Fertigbearbeitung mit der MAE r_1, wobei der letztere sequenziell mit einem dritten Prozessschritt, einer Vorbearbeitung mit der MAE r_2, verbunden ist. Als Folge dessen sinkt der verfügbare Zeitanteil der Betriebsmittelzeit $T_{BM,2}$ für alle anderen Produkte. Zusammenfassend kann die TEK y_{max} des Produkts zwar gesteigert werden, jedoch führt das in beiden Fällen zu einer Reduktion der TEK mindestens eines anderen Produkts.

Das Beispiel zeigt, dass die festgehaltenen Voraussetzungen schon in sehr einfachen Fällen nicht erfüllt werden. Die Kalkulation von Kapazität, Auslastung und Investitionen muss entsprechend erweitert werden, wobei die Abhängigkeiten im jeweils vorliegendem Fall zu berücksichtigen sind. Im nächsten Abschnitt soll erläutert werden, zu welchen Ergebnissen diese Erweiterung im allgemeinen Fall mehrerer miteinander verknüpfter Prozessschritte

eines Produkts führt. Die Erkenntnisse daraus können unmittelbar auf den Fall mehrerer Produkte übertragen werden. Hierzu müssen die Prozessschritte verschiedener Produkte alternativ oder selektiv verknüpft werden, je nachdem, ob die Stückzahlen unabhängig voneinander sein bzw. in einem fixen Verhältnis stehen sollen.

2.5 Erweiterung der Kalkulation: allgemeiner Fall verknüpfter Prozessschritte

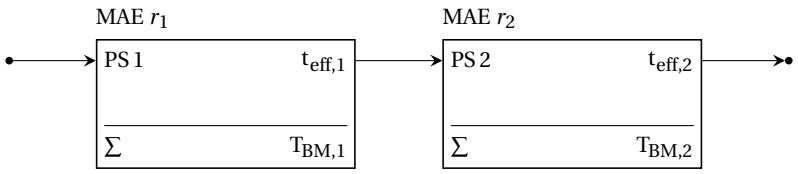
In diesem Abschnitt soll an zuerst einfachen, danach immer komplexeren Beispielen veranschaulicht werden, wie die Kalkulation von Kapazität, Auslastung und Investitionen im allgemeinen Fall verknüpfter Prozessschritte eines Produkts auf geeignete Weise zu erweitern ist. Der grafischen Illustration der Beispiele dienen Schemata, welche die mathematischen, in ihrem Wesen formalen Beschreibungen visuell unterstützen sollen und in den nachfolgenden Kapiteln zu einem grafischen Modell vereinfacht werden.

Um die Aussagen in diesem Abschnitt exakt zu formulieren, werden als Ergänzung zur Notation aus Abschnitt 2.4 folgende Symbole eingeführt:

m	Zahl der Prozessstückzahlen x_α alternativ verknüpfter Prozessschritte, $m \in \mathbb{N}$
$t_{\text{eff},i}$	effektive Taktzeit mit Index $i \in \mathbb{N}$ eines sequenziell verknüpften Prozessschritts, $t_{\text{eff},i} \in \mathbb{R}_{>0}$
$t_{\text{eff},i\alpha}$	effektive Taktzeit mit zweiteiligem Index i/α, $i \in \mathbb{N}, \alpha \in \{\text{'A'}, \text{'B'}, \ldots\}$ eines sequenziell, alternativ und/oder selektiv verknüpften Prozessschritts, $t_{\text{eff},i\alpha} \in \mathbb{R}_{>0}$
$u_r(\mathbf{x})$	Auslastung der MAE r für einen gegebenen Vektor $\mathbf{x}$ von Prozessstückzahlen, $u_r(\mathbf{x}) \in \mathbb{R}_{\geqslant 0}$
$\mathbf{x}$	Vektor $[x_{\alpha_1} \ \ldots \ x_{\alpha_m}]^\top$ der Prozessstückzahlen x_α, $\mathbf{x} \in \mathbb{R}^m_{\geqslant 0}$
$\mathbf{x}^*$	Vektor $[x^*_{\alpha_1} \ \ldots \ x^*_{\alpha_m}]^\top$, optimale Lösung gemäß gegebenem Planungsziel, $\mathbf{x}^* \in \mathbb{R}^m_{\geqslant 0}$
x_α	Prozessstückzahl mit Index $\alpha \in \{\text{'A'}, \text{'B'}, \ldots\}$ alternativ verknüpfter Prozessschritte, $x_\alpha \in \mathbb{R}_{\geqslant 0}$
x^*_α	optimale Teillösung gemäß gegebenem Planungsziel, $x^*_\alpha \in \mathbb{R}_{\geqslant 0}$
α	Index ggf. verketteter alternativer und/oder selektiver Verknüpfungen, $\alpha \in \{\text{'A'}, \text{'B'}, \ldots\}$

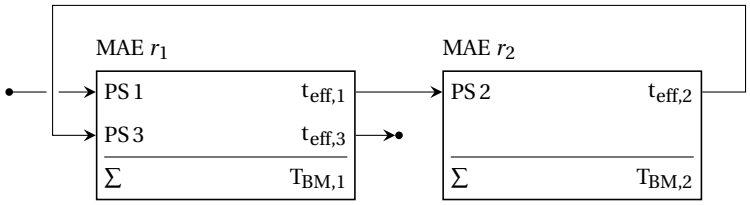
Die Auslastung $u_r(y)$ bzw. $u_r(\mathbf{x})$ einer MAE r ist abhängig von der Produktstückzahl y und deren Verteilung auf die verknüpften Prozessschritte im Wertstrom des Produkts. Wie in Abschnitt 2.2 erläutert, werden die Bedingungen hierfür durch den Typ der Verknüpfung bestimmt. Im Fall sequenzieller und selektiver Verknüpfungen ist die Verteilung der Produktstückzahl y fix vorgegeben (siehe Definition 2.5 bzw. 2.7). Im Gegensatz dazu ist sie im Fall alternativer Verknüpfungen frei wählbar und wird im weiteren Verlauf der Arbeit durch den Vektor $\mathbf{x}$ der Prozessstückzahlen repräsentiert (siehe Definition 2.6). Die Komplexität der Kalkulation steigt insbesondere dann, wenn es zur Beschreibung des Ablaufs notwendig ist, Verknüpfungen verschiedener Typen zu kombinieren. Die TEK $y_{\max}$ bezeichnet stets das Maximum der Produktstückzahl y, wobei beschränkend gilt, dass die Auslastung $u_r(y)$ bzw. $u_r(\mathbf{x})$ keiner MAE r den Höchstwert von eins überschreiten darf. Gesucht ist in den nachfolgenden Beispielen dieses Abschnitts jeweils ein geschlossener mathematischer Ausdruck für die Auslastung $u_r(y)$ bzw. $u_r(\mathbf{x})$ jeder MAE r und, zu ermitteln durch Einsetzen und zielgerichtetes Umformen des Ergebnisses, für die TEK $y_{\max}$.

Die notwendigen Investitionen werden wie im einfachen Fall eines Prozessschritts durch die Auslastung, genauer durch die Überlastung, bestimmt. Überschreitet die Auslastung u_r



(a) Beispiel SQ1

(b) Beispiel SQ2



(c) Beispiel SQ3

Abbildung 2.5: Schemata sequenzieller Verknüpfungen (Beispiel SQ1, SQ2 und SQ3). In Beispiel SQ1 ist eine sequenzielle Verknüpfung zweier Prozessschritte (PS) gegeben, denen je eine MAE r_1 bzw. r_2 zugeordnet ist. Beispiel SQ2 stellt im Hinblick auf den Ablauf die gleiche Verknüpfung dar, wobei für die Prozessschritte dieselbe MAE r genutzt wird. In Beispiel SQ3 liegt eine Kombination der vorigen Beispiele SQ1 und SQ2 vor, wie sie in der Praxis häufig anzutreffen ist.

einer MAE r einen Wert von eins, müssen zusätzlich $\lceil u_r \rceil - 1$ MAEs desselben Typs beschafft werden, um das Produkt in der geplanten Stückzahl zu fertigen bzw. zu montieren. Da dieser direkte Zusammenhang für jedes der nachfolgenden Beispiele in der gleichen Weise gilt, wird auf den entsprechenden Ausdruck verzichtet. Statt unzweckmäßiger Wiederholungen sollen vielmehr die jeweiligen Besonderheiten hinsichtlich der Bestimmung der Auslastung $u_r(y)$ bzw. $u_r(\mathbf{x})$ und der TEK $y_{\max}$ aufgezeigt und verallgemeinert werden, sodass deutlich wird, wie die Systemstruktur die Kalkulation bedingt.

2.5.1 Sequenzielle Verknüpfung

Mit Hilfe von sequenziellen Verknüpfungen werden gemäß Definition 2.5 verschiedene Prozessschritte eines Produkts miteinander verbunden, die stets von der gesamten Stückzahl durchlaufen werden müssen. Solche Prozessschritte werden im geplanten Ablauf der Fertigung und Montage typischerweise nacheinander ausgeführt, wobei ihnen entweder jeweils eine eigene oder dieselbe MAE zugeordnet ist. Im Folgenden soll die Kalkulation der resultierenden Auslastung $u_r(y)$ und der TEK $y_{\max}$ für diese beiden Fälle und für ihre Kombination an je einem Beispiel veranschaulicht werden.

Beispiel SQ1 SQ (PS 1 [r_1], PS 2 [r_2]) (formale Notation, siehe Abschnitt 2.2)

Die Fertigung eines Produkts teilt sich in eine Vorbearbeitung und eine darauffolgende Fertigbearbeitung, wie in Abbildung 2.5(a) dargestellt. Für die Vorbearbeitung (Prozessschritt 1, effektive Taktzeit $t_{eff,1}$) wird eine Drehmaschine r_1 (Betriebsmittelzeit $T_{BM,1}$) genutzt, welche ein Mitarbeiter in einem teilautomatisierten Ablauf steuert. Dagegen erfolgt die Fertigbearbeitung (Prozessschritt 2, effektive Taktzeit $t_{eff,2}$) vollständig automatisiert mit einem Bearbeitungszentrum (BAZ) r_2 (Betriebsmittelzeit $T_{BM,2}$).

Aus der Beschreibung geht hervor, dass in dem Beispiel eine sequenzielle Verknüpfung zweier Prozessschritte mit je einer MAE $r_{1/2}$ vorliegt. Das heißt, die beiden Prozessschritte werden von der gesamten Produktstückzahl y durchlaufen (siehe Definition 2.5), und jede MAE $r_{1/2}$ wird nur für einen einzigen Prozessschritt genutzt. In einem solchen Fall kann die Auslastung $u_{r_{1/2}}(y)$ gemäß Gleichung (2.20) wie folgt bestimmt werden:

$$u_{r_{1/2}}(y) = \frac{t_{eff,1/2}\, y}{T_{BM,1/2}}. \tag{2.21}$$

Erreicht die Auslastung $u_{r_{1/2}}(y)$ einer MAE $r_{1/2}$ einen Wert von eins, ist es nicht möglich, die Produktstückzahl y weiter zu erhöhen, unabhängig von der Auslastung $u_{r_{2/1}}(y)$ der jeweils verbleibenden MAE $r_{2/1}$. Die TEK y_{max} bezeichnet die maximale Produktstückzahl y, welche die Bedingung erfüllt, dass die Auslastung $u_{r_{1/2}}(y)$ keiner der MAEs $r_{1/2}$ einen Wert von eins übersteigt. Formal lautet diese Definition:

$$y_{max} = \max\left\{y\colon\ y \geqslant 0 \wedge \forall r \in \{r_1, r_2\}\ u_r(y) \leqslant 1\right\}. \tag{2.22}$$

Löst man Gleichung (2.22) mit Hilfe der Auslastung $u_{r_{1/2}}(y)$ aus Gleichung (2.21) auf, führt das zu folgendem Ausdruck, welcher der TEK y_{max} entspricht:

$$y_{max} = \min\left\{\frac{T_{BM,1}}{t_{eff,1}}, \frac{T_{BM,2}}{t_{eff,2}}\right\}. \tag{2.23}$$

In dem Beispiel werden die Drehmaschine r_1 und das BAZ r_2 für jeweils einen sequenziell verknüpften Prozessschritt genutzt. Die TEK y_{max} wird unter diesen Bedingungen durch diejenige der zwei MAEs $r_{1/2}$ bestimmt, deren Auslastung $u_{r_{1/2}}(y)$ zuerst einen Wert von eins erreicht, wenn die Produktstückzahl y stetig erhöht wird. Im nächsten Beispiel werden wieder zwei Prozessschritte sequenziell verknüpft, wobei diesen im Unterschied zum gerade betrachteten Beispiel dieselbe MAE zugeordnet ist.

Beispiel SQ2 SQ (PS 1 [r], PS 2 [r])

Wie in Beispiel SQ1 gliedert sich die Fertigung eines Produkts in eine Vorbearbeitung und eine Fertigbearbeitung, dargestellt in Abbildung 2.5(b). Das BAZ r (Betriebsmittelzeit T_{BM}) wird gegenüber dem vorherigen Beispiel nicht mehr nur für die Fertigbearbeitung (Prozessschritt 2, effektive Taktzeit $t_{eff,2}$) verwendet, sondern auch für die nun ebenfalls automatisierte Vorbearbeitung (Prozessschritt 1, effektive Taktzeit $t_{eff,1}$) genutzt.

Dabei ist zu berücksichtigen, dass es Definition 2.1 verbietet, mit ein und derselben MAE zu einem Zeitpunkt mehr als einen Prozessschritt auszuführen. Um in einem solchen Fall

die Auslastung $u_r(y)$ als Ergebnis einer vorgegebenen Produktstückzahl y zu bestimmen, müssen aus diesem Grund die effektiven Taktzeiten $t_{eff,1/2}$ addiert werden. Eingesetzt in Gleichung (2.20) folgt daraus der unten angegebene Ausdruck:

$$u_r(y) = \frac{(t_{eff,1} + t_{eff,2})\, y}{T_{BM}}. \tag{2.24}$$

Die TEK y_{max} ist definiert als die maximale Produktstückzahl y, wobei die Auslastung $u_r(y)$ einen Wert von eins nicht überschreiten darf. Formal gilt:

$$y_{max} = \max\{y:\ y \geqslant 0 \wedge u_r(y) \leqslant 1\}. \tag{2.25}$$

Indem man Gleichung (2.25) mit Hilfe der Auslastung $u_r(y)$ aus Gleichung (2.24) auflöst, kann für die gesuchte TEK y_{max} folgender Ausdruck gefunden werden:

$$y_{max} = \frac{T_{BM}}{t_{eff,1} + t_{eff,2}}. \tag{2.26}$$

Das Beispiel soll verdeutlichen, dass die effektiven Taktzeiten $t_{eff,1/2}$ addiert werden müssen, um die Auslastung $u_r(y)$ und die TEK y_{max} zu bestimmen, wenn im jeweils gegebenen Fall den sequenziell verknüpften Prozessschritten eines Produkts dieselbe MAE zugeordnet ist. In der Praxis, wie auch an Standorten der Bosch Rexroth AG, ist oftmals eine Kombination der Beispiele SQ1 und SQ2 vorzufinden. Spezielle Produktionsanlagen und Messvorrichtungen werden im Ablauf der Fertigung und Montage wiederholt für dasselbe Produkt genutzt. Einen solchen Fall beschreibt auch das folgende, dritte Beispiel.

Beispiel SQ3 SQ(SQ(PS 1 [r_1], PS 2 [r_2]), PS 3 [r_1])

Um ein Produkt zu fertigen, finden nacheinander eine erste Messung (Eingangsprüfung), eine Bearbeitung und zum Abschluss eine zweite Messung (Ausgangsprüfung) statt, wie in Abbildung 2.5(c) dargestellt. Die beiden Messvorgänge (Prozessschritt 1 und 3, effektive Taktzeit $t_{eff,1}$ bzw. $t_{eff,3}$) erfordern jeweils den wiederholten Einsatz der Messvorrichtung r_1 (Betriebsmittelzeit $T_{BM,1}$). Für die Bearbeitung (Prozessschritt 2, effektive Taktzeit $t_{eff,2}$) wird zwischen den Messungen ein BAZ r_2 (Betriebsmittelzeit $T_{BM,2}$) verwendet.

Gemäß der Beschreibung ist in dem Beispiel eine sequenzielle Verknüpfung zweier Prozessschritte gegeben. Das Ergebnis, welches aus dieser Verknüpfung folgt, ist sequenziell mit einem dritten Prozessschritt verbunden. Dem ersten und dritten Prozessschritt ist dieselbe MAE r_1 zugeordnet, dem zweiten Prozessschritt dagegen die MAE r_2. Analog zu Beispiel SQ2 kann die resultierende Auslastung $u_{r_1}(y)$ der MAE r_1 bestimmt werden, indem in Gleichung (2.20) die Summe der effektiven Taktzeiten $t_{eff,1/3}$ eingesetzt wird:

$$u_{r_1}(y) = \frac{(t_{eff,1} + t_{eff,3})\, y}{T_{BM,1}}. \tag{2.27a}$$

Da die MAE r_2 ausschließlich für die Bearbeitung und damit nur für einen einzigen Prozessschritt genutzt wird, gilt für die Auslastung $u_{r_2}(y)$ gemäß Gleichung (2.20):

$$u_{r_2}(y) = \frac{t_{eff,2}\, y}{T_{BM,2}}. \tag{2.27b}$$

Die TEK $y_{\max}$ ist analog zu Beispiel SQ1 definiert als die maximale Produktstückzahl y unter der Bedingung, dass die Auslastung $u_{r_{1/2}}(y)$ keiner der MAEs $r_{1/2}$ einen Wert von eins überschreitet. Formal lautet die Definition entsprechend:

$$y_{\max} = \max\left\{y\colon\ y \geqslant 0 \wedge \forall r \in \{r_1, r_2\}\ u_r(y) \leqslant 1\right\}. \tag{2.28}$$

Mit Gleichung (2.27a, 2.27b) kann durch Einsetzen der Auslastung $u_{r_{1/2}}(y)$ folgender Ausdruck gefunden werden, welcher nach Gleichung (2.28) der TEK $y_{\max}$ entspricht:

$$y_{\max} = \min\left\{\frac{\mathrm{T_{BM,1}}}{\mathrm{t_{eff,1}} + \mathrm{t_{eff,3}}}, \frac{\mathrm{T_{BM,2}}}{\mathrm{t_{eff,2}}}\right\}. \tag{2.29}$$

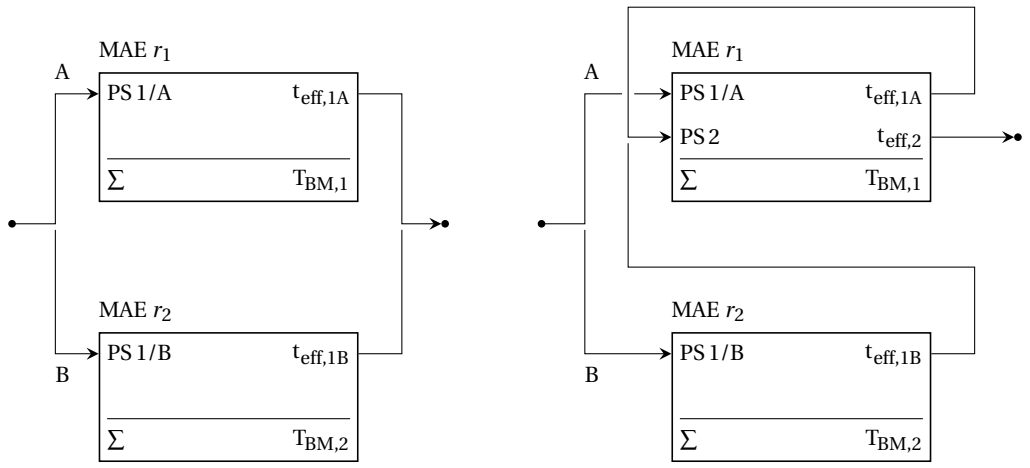
Die Beispiele zeigen, dass die Produktstückzahl y im Fall sequenzieller Verknüpfungen nicht zu steigern, d. h. die TEK $y_{\max}$ erreicht ist, sobald die Auslastung $u_r(y)$ mindestens einer MAE r einen Wert von eins annimmt. Ist zwei Prozessschritten dieselbe MAE zugeordnet, muss dabei die Überlagerung dieser Prozessschritte berücksichtigt werden.

Statt einschränkender Bedingungen, welche die Kapazität begrenzen, werden im Fall einer alternativen Verknüpfung zusätzliche Freiheitsgrade geschaffen. Wie in Abschnitt 2.2 erläutert, kann die Stückzahl auf variable, nichtnegative reelle Prozessstückzahlen verteilt werden, wobei jedem Operanden einer alternativen Verknüpfung jeweils eine Prozessstückzahl zugewiesen wird. Im nächsten Teil des Abschnitts soll die Kalkulation der Auslastung und der Kapazität in drei Beispielen beleuchtet werden.

2.5.2 Alternative Verknüpfung

Durch alternative Verknüpfungen werden gemäß Definition 2.6 solche Prozessschritte eines Produkts verbunden, die im geplanten Ablauf von einem variablen Anteil der Stückzahl durchlaufen werden. Zum besseren Verständnis der folgenden Beispiele wird eine Indexnotation für Prozessschritte und Taktzeiten eingeführt. Ist ein Prozessschritt alternativ und/oder selektiv verknüpft, soll es möglich sein, von der Bezeichnung auf seine Position unter allen Operanden der verketteten Verknüpfungen zu schließen. Hierzu wird jedem alternativ und/oder selektiv verknüpften Prozessschritt ein Index i/α zugeordnet, der aus zwei Teilen besteht (z. B. 1/A, 2/B usw.). Der Teilindex i mit dem Wert 1, 2 usw. bezeichnet die Position des jeweiligen Prozessschritts unter allen sequenziellen Verknüpfungen (liegt keine solche vor, wird ein Wert von eins verwendet). Der Teilindex α mit dem Wert A, B usw. bezeichnet die Position unter allen alternativen und selektiven Verknüpfungen.

Da alle Prozessschritte i/α mit gleichem Index α (z. B. 1/A, 2/A usw.) sequenziell verknüpft sind, muss durch die Prozessschritte jeweils die gleiche Stückzahl verlaufen. Diese Stückzahl wird gemäß Definition 2.6 mit dem Begriff der Prozessstückzahl und dem Symbol x_α bezeichnet. Entsprechend repräsentiert der Vektor $\mathbf{x}$ mit den Koordinaten x_α die Verteilung der Produktstückzahl y im Wertstrom des Produkts. Es folgen drei Beispiele alternativer Verknüpfungen, welche die Kalkulation der Auslastung $u_r(\mathbf{x})$ und der TEK $y_{\max}$ unter Verwendung der eingeführten Indexnotation veranschaulichen.



(a) Beispiel AL1 (b) Beispiel AL2

Abbildung 2.6: Schemata alternativer Verknüpfungen (Beispiel AL1 und AL2). Beispiel AL1 beschreibt die alternative Verknüpfung zweier Prozessschritte (PS), denen je eine MAE r_1 bzw. r_2 zugeordnet ist. Beispiel AL2 erweitert Beispiel AL1, indem die beiden Prozessschritte sequenziell mit einem dritten Prozessschritt verbunden werden, der jeweils im Anschluss ausgeführt werden muss. Für diesen zusätzlichen Prozessschritt wird wie für den ersten die MAE r_1 benötigt.

Beispiel AL1 AL(PS 1/A[r_1], PS 1/B[r_2])

Die Fertigung eines Produkts verläuft im Zuge einer Komplettbearbeitung in einem einzigen Bearbeitungsschritt, wie in Abbildung 2.6(a) dargestellt. Hierfür existieren grundsätzlich zwei verschiedene Möglichkeiten, wobei das Ergebnis mit Blick auf das Produkt jeweils das gleiche ist. Die Bearbeitung erfolgt entweder vollständig automatisiert (Prozessschritt 1/A, effektive Taktzeit $t_{eff,1A}$) mit einem BAZ r_1 (Betriebsmittelzeit $T_{BM,1}$) oder teilautomatisiert (Prozessschritt 1/B, effektive Taktzeit $t_{eff,1B}$) unter Inanspruchnahme einer zusätzlich verfügbaren, manuell zu bedienenden Drehmaschine r_2 (Betriebsmittelzeit $T_{BM,2}$).

Gegeben ist somit eine alternative Verknüpfung zweier Prozessschritte 1/A und 1/B, denen jeweils eine der zwei MAEs $r_{1/2}$ zugeordnet ist. Die Produktstückzahl y kann in einem Fall wie in diesem Beispiel variabel auf zwei Prozessstückzahlen x_A und x_B (Prozessschritt 1/A bzw. 1/B) verteilt werden. Deren Summe muss allerdings stets gleich der Produktstückzahl y sein, sodass formal gilt (siehe Definition 2.6):

$$y = x_A + x_B\,. \tag{2.30}$$

Die Prozessstückzahlen x_A und x_B durchlaufen unabhängig voneinander das BAZ r_1 bzw. die Drehmaschine r_2. Um die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der beiden MAEs $r_{1/2}$ zu bestimmen, muss Gleichung (2.20) wie folgt für einen Vektor $\mathbf{x} = [x_A\ x_B]^\top$ erweitert werden:

$$u_{r_1}(\mathbf{x}) = \frac{t_{eff,1A}\,x_A}{T_{BM,1}}\,, \qquad u_{r_2}(\mathbf{x}) = \frac{t_{eff,1B}\,x_B}{T_{BM,2}}\,. \tag{2.31a, 2.31b}$$

Die TEK y_{max} bezeichnet wie im bisherigen Verlauf die maximale Produktstückzahl y, wobei die resultierende Auslastung $u_{r_{1/2}}(\mathbf{x})$ keiner der zwei MAEs $r_{1/2}$ einen Wert von eins

überschreiten darf. Die Prozessstückzahlen x_A und x_B können einen beliebigen reellen Wert größer als oder gleich null annehmen. Gemäß Gleichung (2.30) muss ihre Summe jedoch der Produktstückzahl y entsprechen. Drückt man diesen Zusammenhang in formaler Schreibweise aus, lautet die Definition der TEK y_{max}:

$$y_{max} = \max\left\{x_A + x_B \colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2\}\ u_r(\mathbf{x}) \leqslant 1\right\}. \tag{2.32}$$

Um hierfür einen geschlossenen Ausdruck zu finden, wird die TEK y_{max} als eine Funktion von zwei optimalen Teillösungen x_A^* und x_B^* dargestellt. Die Teillösungen sollen dadurch definiert sein, dass die Produktstückzahl y der TEK y_{max} entspricht, wenn x_A^* und x_B^* als Summanden in Gleichung (2.30) eingesetzt werden. Einfacherweise folgt daraus:

$$y_{max} = x_A^* + x_B^*. \tag{2.33}$$

Die zwei Prozessstückzahlen x_A und x_B sind gemäß Gleichung (2.31a, 2.31b) allein durch die Bedingung beschränkt, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ je einer der MAEs $r_{1/2}$ kleiner als oder gleich eins sein muss, wobei dies im Fall von x_A die MAE r_1 und im Fall von x_B die MAE r_2 betrifft. Eine Bedingung, die zu einer Abhängigkeit zwischen x_A und x_B führt, existiert in diesem Beispiel dagegen nicht. Die gesuchte TEK y_{max} ist definiert als das Maximum der Produktstückzahl y und entspricht der Summe der zwei optimalen Teillösungen x_A^* und x_B^*, wie in Gleichung (2.32) bzw. (2.33) angegeben. Da sich die beiden Summanden, wie soeben erläutert, nicht gegenseitig begrenzen, müssen diese unabhängig voneinander jeweils den größten Wert annehmen, der nach obiger Bedingung in Bezug auf die Auslastung $u_{r_{1/2}}(\mathbf{x})$ zulässig ist. Das heißt in formaler Notation:

$$x_A^* = \max\left\{x_A \colon \mathbf{x} \geqslant \mathbf{0} \wedge u_{r_1}(\mathbf{x}) \leqslant 1\right\}, \tag{2.34a}$$

$$x_B^* = \max\left\{x_B \colon \mathbf{x} \geqslant \mathbf{0} \wedge u_{r_2}(\mathbf{x}) \leqslant 1\right\}. \tag{2.34b}$$

Gleichung (2.31a, 2.31b) führt jeweils zu folgendem Ausdruck für die optimale Teillösung x_A^* bzw. x_B^*, welcher der Definition gemäß Gleichung (2.34a, 2.34b) entspricht:

$$x_A^* = \frac{T_{BM,1}}{t_{eff,1A}}, \qquad x_B^* = \frac{T_{BM,2}}{t_{eff,1B}}. \tag{2.35a, 2.35b}$$

Einsetzen der Ergebnisse aus Gleichung (2.35a, 2.35b) in Gleichung (2.33) führt zu dem gesuchten Ausdruck für die TEK y_{max}, der in diesem Beispiel lautet:

$$y_{max} = \frac{T_{BM,1}}{t_{eff,1A}} + \frac{T_{BM,2}}{t_{eff,1B}}. \tag{2.36}$$

Da die MAEs $r_{1/2}$ verschiedenen, alternativ miteinander verknüpften Prozessschritten zugeordnet sind, ist im vorliegenden Beispiel die TEK y_{max} zu bestimmen, indem die maximalen Stückzahlen, welche durch diese Prozessschritte verlaufen, addiert werden. Anders formuliert, durch unabhängige Maximierung der Prozessstückzahlen wird gleichermaßen deren Summe maximiert. Beschränkend gilt wie zuvor die Bedingung, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ keiner der MAEs $r_{1/2}$ einen Wert von eins übersteigen darf.

Bisher erschien die Kalkulation intuitiv, jedoch ist in vielen Fällen eine Kombination von Verknüpfungen verschiedener Typen erforderlich, um Produktionsabläufe aus der Praxis abzubilden. Als Folge wächst unweigerlich die Komplexität. Das nächste Beispiel soll diese Beobachtung an dem immer noch vergleichsweise einfachen Fall einer alternativen und einer sequenziellen Verknüpfung belegen.

Beispiel AL2 SQ(AL(PS 1/A $[r_1]$, PS 1/B $[r_2]$), PS 2 $[r_1]$)

Die Fertigung eines Produkts ist analog zu Beispiel SQ1 in eine Vorbearbeitung und eine Fertigbearbeitung gegliedert, wie in Abbildung 2.6(b) dargestellt. Für die Vorbearbeitung existieren grundsätzlich wie in Beispiel AL1 zwei verschiedene Möglichkeiten, welche sich in Bezug auf das Ergebnis nicht unterscheiden. Die Vorbearbeitung erfolgt entweder vollständig automatisiert (Prozessschritt 1/A, effektive Taktzeit $t_{eff,1A}$) mit einem BAZ r_1 (Betriebsmittelzeit $T_{BM,1}$) oder teilautomatisiert (Prozessschritt 1/B, effektive Taktzeit $t_{eff,1B}$) mit einer Drehmaschine r_2 (Betriebsmittelzeit $T_{BM,2}$). Dagegen erfordert die Fertigbearbeitung (Prozessschritt 2, effektive Taktzeit $t_{eff,2}$) in jedem Fall die Nutzung des BAZ r_1, d. h. unabhängig davon, welche der beiden MAEs zuvor zur Vorbearbeitung eingesetzt wurde.

In dem gegebenen Beispiel liegt eine alternative Verknüpfung zweier Prozessschritte 1/A und 1/B zur Vorbearbeitung mit je einer der MAEs $r_{1/2}$ vor. Das Ergebnis der Verknüpfung ist sequenziell mit einem Prozessschritt 2 zur Fertigbearbeitung verbunden, welchem die MAE r_1 zugeordnet ist. Folglich ist es möglich, die Produktstückzahl y variabel auf zwei Prozessstückzahlen x_A (Prozessschritt 1/A und 2) und x_B (Prozessschritt 1/B und 2) zu verteilen. Wie in Beispiel AL1 gilt die Bedingung, dass deren Summe der Produktstückzahl y entsprechen muss (siehe Definition 2.6). Das heißt formal:

$$y = x_A + x_B\,. \tag{2.37}$$

Das BAZ r_1 soll wie beschrieben sowohl für die Vorbearbeitung (Prozessschritt 1/A) als auch für die Fertigbearbeitung (Prozessschritt 2) genutzt werden. Als Folge wird es von den Prozessstückzahlen x_A (Prozessschritt 1/A und 2) und x_B (Prozessschritt 2) durchlaufen. Um die Auslastung $u_{r_1}(\mathbf{x})$ zu bestimmen, muss die Überlagerung der Prozessschritte berücksichtigt werden. Gleichung (2.20) ist entsprechend für einen Vektor $\mathbf{x} = [x_A\ x_B]^\top$ zu erweitern, indem die Stückzahlen mit den Taktzeiten gewichtet werden:

$$u_{r_1}(\mathbf{x}) = \frac{(t_{eff,1A} + t_{eff,2})\,x_A + t_{eff,2}\,x_B}{T_{BM,1}}\,. \tag{2.38a}$$

Im Gegensatz dazu wird die Drehmaschine r_2 allein für die Vorbearbeitung genutzt und von der Prozessstückzahl x_B durchlaufen (Prozessschritt 1/B). Die Auslastung $u_{r_2}(\mathbf{x})$ ist in diesem Fall auf einfache Weise zu bestimmen, indem die Stückzahl und die zugehörige Taktzeit in Gleichung (2.20) eingesetzt werden, sodass entsprechend gilt:

$$u_{r_2}(\mathbf{x}) = \frac{t_{eff,1B}\,x_B}{T_{BM,2}}\,. \tag{2.38b}$$

Die TEK $y_{\max}$ bezeichnet die maximale Produktstückzahl y, wobei wie in Beispiel AL1 die Bedingung erfüllt sein muss, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ keiner der zwei MAEs $r_{1/2}$ einen Wert von eins übersteigt. Entsprechend lautet die formale Definition:

$$y_{\max} = \max\left\{x_{\mathrm{A}} + x_{\mathrm{B}} \colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2\}\ u_r(\mathbf{x}) \leqslant 1\right\}. \tag{2.39}$$

In der Praxis finden Vorbearbeitung und Fertigbearbeitung oft bevorzugt mit dem vollautomatisierten BAZ r_1 statt (Prozessschritt 1/A bzw. 2). Sobald aber die Auslastung $u_{r_1}(\mathbf{x})$ einen Wert von eins erreicht, erfolgt die Vorbearbeitung für einen Teil x_{B} der Produktstückzahl y mit der Drehmaschine r_2 (Prozessschritt 1/B), um Kapazität zur Fertigbearbeitung mit dem BAZ r_1 (Prozessschritt 2) zu schaffen. Der verbleibende Stückzahlanteil x_{A} durchläuft zur Vorbearbeitung weiter das BAZ r_1 (Prozessschritt 1/A). Gesucht ist die TEK $y_{\max}$, die erzielt werden kann, indem die Produktstückzahl y optimal auf die zwei Prozessstückzahlen x_{A} und x_{B} verteilt wird. Das Ergebnis lautet wie folgt.

Theorem 2.1: Technische Kapazität $y_{\max}$ für Beispiel AL2. Im Ablauf der Fertigung und Montage seien die Prozessschritte 1/A, 1/B und 2 (effektive Taktzeit $\mathrm{t_{eff,1A}}$, $\mathrm{t_{eff,1B}}$ bzw. $\mathrm{t_{eff,2}}$) unter Nutzung von zwei MAEs $r_{1/2}$ (Betriebsmittelzeit $\mathrm{T_{BM,1/2}}$) alternativ und sequenziell verknüpft, wie in Abbildung 2.6(b) dargestellt. In dem Fall gilt für die TEK $y_{\max}$:

$$y_{\max} = \begin{cases} \dfrac{\mathrm{T_{BM,1}}}{\mathrm{t_{eff,2}}}, & \text{falls } \dfrac{\mathrm{T_{BM,1}}}{\mathrm{t_{eff,2}}} \leqslant \dfrac{\mathrm{T_{BM,2}}}{\mathrm{t_{eff,1B}}}, \quad \text{(a)} \\ \dfrac{\mathrm{T_{BM,1}} + \dfrac{\mathrm{t_{eff,1A}}\,\mathrm{T_{BM,2}}}{\mathrm{t_{eff,1B}}}}{\mathrm{t_{eff,1A}} + \mathrm{t_{eff,2}}}, & \text{sonst.} \qquad \text{(b)} \end{cases} \tag{2.40}$$

Interpretation. Bevor der Beweis des Theorems folgt, soll das Verständnis dafür geschaffen werden, wie sich die Terme in Gleichung (2.40) zusammensetzen. Die TEK $y_{\max}$ ist durch eine Fallunterscheidung bestimmt, die auf einem Vergleich der Quotienten $\mathrm{T_{BM,1}}/\mathrm{t_{eff,2}}$ und $\mathrm{T_{BM,2}}/\mathrm{t_{eff,1B}}$ beruht. Diese zwei Ausdrücke bezeichnen jeweils das Maximum der Produktstückzahl y unter der Bedingung, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ einen Wert von eins nicht übersteigt, wobei nur ein Prozessschritt berücksichtigt wird. Der erste Ausdruck betrifft entsprechend die Fertigbearbeitung mit der MAE r_1 (Prozessschritt 2), der zweite die Vorbearbeitung mit der MAE r_2 (Prozessschritt 1/B). Hintergrund für den Vergleich dieser beiden Ausdrücke ist folgender: Da die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der zwei MAEs $r_{1/2}$ nach oben beschränkt ist, kann der maximale Wert der Produktstückzahl y erzielt werden, indem zuerst ausschließlich diese zwei Prozessschritte ausgeführt werden.

Im Fall (a) von Gleichung (2.40) ist der Quotient $\mathrm{T_{BM,1}}/\mathrm{t_{eff,2}}$ kleiner als oder gleich dem Quotienten $\mathrm{T_{BM,2}}/\mathrm{t_{eff,1B}}$. Das heißt, werden nur die zwei Prozessschritte betrachtet und wird die Produktstückzahl y stetig erhöht, dann erreicht die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der MAE r_1 zuerst oder zusammen mit der MAE r_2 einen Wert von eins. Da aber zur Fertigbearbeitung im geplanten Ablauf immer die MAE r_1 genutzt werden muss, kann die Produktstückzahl y nicht weiter gesteigert werden. Entsprechend folgt die TEK $y_{\max}$ in dem gegebenen Fall unmittelbar aus der Betriebsmittelzeit $\mathrm{T_{BM,1}}$ und der effektiven Taktzeit $\mathrm{t_{eff,2}}$.

Im Fall (b) der oben aufgeführten Gleichung ist der Quotient $T_{BM,1}/t_{eff,2}$ größer als der Quotient $T_{BM,2}/t_{eff,1B}$. Das heißt, werden wie im vorigen Fall nur die zwei genannten Prozessschritte betrachtet und wird die Produktstückzahl y stetig gesteigert, dann erreicht die Auslastung $u_{r_2}(\mathbf{x})$ der MAE r_2 zuerst einen Wert von eins. Allerdings ist es dieses Mal möglich, die Produktstückzahl y weiter zu erhöhen, indem die MAE r_1 nicht mehr nur zur Fertigbearbeitung, sondern auch zur Vorbearbeitung genutzt wird. Die Kapazität, welche für die Vorbearbeitung mit der MAE r_2 zur Verfügung steht, wird hierzu gedanklich der MAE r_1 hinzugefügt. Entsprechend werden die Betriebsmittelzeiten $T_{BM,1/2}$ addiert. Die effektiven Taktzeiten $t_{eff,1A}$ und $t_{eff,1B}$ stimmen aber nicht notwendigerweise überein, weshalb die Betriebsmittelzeit $T_{BM,2}$ vor Bildung der Summe mit dem Faktor $t_{eff,1A}/t_{eff,1B}$ multipliziert werden muss. Dann kann für die Vorbearbeitung unabhängig von der jeweils genutzten MAE $r_{1/2}$ die effektive Taktzeit $t_{eff,1A}$ angenommen werden. Da jedes Stück des Produkts immer beide Bearbeitungsschritte durchläuft, müssen die effektiven Taktzeiten $t_{eff,1A}$ und $t_{eff,2}$ addiert werden. Fasst man all das zusammen, ist die TEK y_{max} in diesem Fall das Ergebnis einer Betriebsmittelzeit $T_{BM,1} + t_{eff,1A}\, T_{BM,2}/t_{eff,1B}$ und einer effektiven Taktzeit $t_{eff,1A} + t_{eff,2}$, wie in Gleichung (2.40) weiter oben angegeben.

Beweis. Wie schon in Beispiel AL1 wird die TEK y_{max} als eine Funktion zweier optimaler Teillösungen x_A^* und x_B^* ausgedrückt. Optimal heißt in diesem Zusammenhang, dass die Produktstückzahl y, welche gemäß Gleichung (2.37) der Summe aus x_A und x_B entspricht, mit der gesuchten TEK y_{max} übereinstimmt, wenn x_A^* bzw. x_B^* als Summanden eingesetzt werden. Beschreibt man diese Bedingung formal, dann folgt daraus:

$$y_{max} = x_A^* + x_B^* . \tag{2.41}$$

Für alle möglichen Lösungen $\mathbf{x} = [x_A\ x_B]^\top$ gilt die Bedingung, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ keiner der zwei MAEs $r_{1/2}$, die nach Gleichung (2.38a, 2.38b) aus einem Paar von x_A und x_B folgt, einen Wert größer als eins annehmen darf. Um einen Ausdruck für die TEK y_{max} zu finden, der Gleichung (2.39) erfüllt, ist aus der Menge aller Lösungen $\mathbf{x}$, die nach obiger Bedingung gültig sind, die optimale Lösung $\mathbf{x}^* = [x_A^*\ x_B^*]^\top$ gesucht. Laut Gleichung (2.41) ist diese dadurch gekennzeichnet, dass die Summe der optimalen Teillösungen x_A^* und x_B^* wie angegeben der TEK y_{max} entspricht. Gemäß Definition der TEK y_{max} darf keine andere gültige Lösung $\mathbf{x}$ existieren, nach welcher die Produktstückzahl y den zugehörigen Wert gemäß der optimalen Lösung $\mathbf{x}^*$, d. h. die TEK y_{max}, übersteigt.

Die Prozessstückzahl x_A im Allgemeinen und die optimale Teillösung x_A^* im Besonderen sind allein durch die Bedingung beschränkt, dass die Auslastung $u_{r_1}(\mathbf{x}^*)$ der MAE r_1 kleiner als oder gleich eins sein muss. Wird angenommen, dass die Auslastung $u_{r_1}(\mathbf{x}^*)$ kleiner als eins ist, dann existiert eine weitere gültige Lösung $\mathbf{x}$, nach welcher die Prozessstückzahl x_A größer ist als die optimale Teillösung x_A^*, die Werte für x_B und x_B^* gleich sind und als Folge die Produktstückzahl y die TEK y_{max} übersteigt. Da dies aber laut Definition der TEK y_{max} ausgeschlossen ist, liegt ein Widerspruch vor. Dementsprechend muss die Annahme falsch, ihre Negation wahr und die Auslastung $u_{r_1}(\mathbf{x}^*)$ gleich eins sein. Unter dieser Voraussetzung gilt nach Umstellung von Gleichung (2.38a) für die optimale Teillösung x_A^*:

$$x_A^* = \frac{T_{BM,1} - t_{eff,2}\, x_B^*}{t_{eff,1A} + t_{eff,2}} . \tag{2.42}$$

Indem Gleichung (2.42) in Gleichung (2.41) eingesetzt wird, ist die TEK $y_{\max}$ anschließend als Funktion der optimalen Teillösung x_{B}^{*} darstellbar:

$$y_{\max} = \frac{\mathrm{T_{BM,1}} - \mathrm{t_{eff,2}}\, x_{\mathrm{B}}^{*}}{\mathrm{t_{eff,1A}} + \mathrm{t_{eff,2}}} + x_{\mathrm{B}}^{*} = \frac{\mathrm{T_{BM,1}} + \mathrm{t_{eff,1A}}\, x_{\mathrm{B}}^{*}}{\mathrm{t_{eff,1A}} + \mathrm{t_{eff,2}}}\,. \tag{2.43}$$

Die Zeitgrößen seien positiv gegeben. Unter allen möglichen Lösungen $\mathbf{x}$, für welche die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der MAE r_1 gleich eins und jene der MAE r_2 nicht größer als eins ist, muss die Teillösung x_{B}^{*} der optimalen Lösung $\mathbf{x}^{*}$ maximal sein. Der Grund dafür ist, dass die Annahme des Gegenteils wie zuvor zu einem Widerspruch führt. Wird entsprechend angenommen, dass eine Lösung $\mathbf{x}$ existiert, welche die zwei genannten Bedingungen in Bezug auf die Auslastung $u_{r_{1/2}}(\mathbf{x})$ erfüllt und nach welcher die Prozessstückzahl x_{B} größer ist als die Teillösung x_{B}^{*}, dann übersteigt die TEK $y_{\max}$ für x_{B} laut Gleichung (2.43) den Wert für x_{B}^{*}. Somit gilt für die optimale Teillösung x_{B}^{*} allgemein:

$$x_{\mathrm{B}}^{*} = \max\left\{x_{\mathrm{B}} \colon \mathbf{x} \geqslant \mathbf{0} \wedge u_{r_1}(\mathbf{x}) = 1 \wedge u_{r_2}(\mathbf{x}) \leqslant 1\right\}. \tag{2.44}$$

Mit Hilfe der Auslastung $u_{r_{1/2}}(\mathbf{x})$ aus Gleichung (2.38a, 2.38b) ist es möglich, einen Ausdruck für die optimale Teillösung x_{B}^{*} zu bestimmen, welcher Gleichung (2.44) erfüllt:

$$x_{\mathrm{B}}^{*} = \min\left\{\frac{\mathrm{T_{BM,1}}}{\mathrm{t_{eff,2}}}, \frac{\mathrm{T_{BM,2}}}{\mathrm{t_{eff,1B}}}\right\}. \tag{2.45}$$

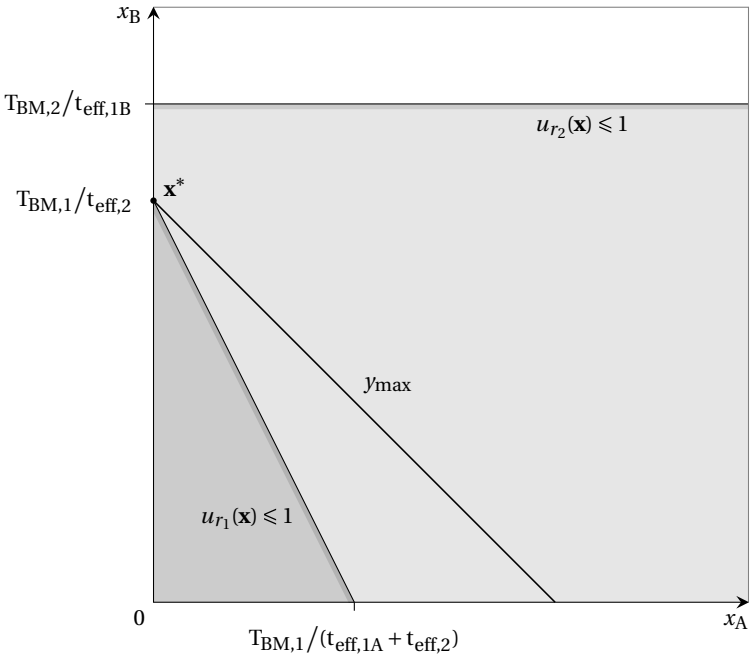
Durch Umformulierung von Gleichung (2.45) kann dieser Ausdruck im nächsten Schritt als Ergebnis einer Fallunterscheidung dargestellt werden:

$$x_{\mathrm{B}}^{*} = \begin{cases} \dfrac{\mathrm{T_{BM,1}}}{\mathrm{t_{eff,2}}}, & \text{falls } \dfrac{\mathrm{T_{BM,1}}}{\mathrm{t_{eff,2}}} \leqslant \dfrac{\mathrm{T_{BM,2}}}{\mathrm{t_{eff,1B}}}, \\ \dfrac{\mathrm{T_{BM,2}}}{\mathrm{t_{eff,1B}}}, & \text{sonst.} \end{cases} \tag{2.46}$$

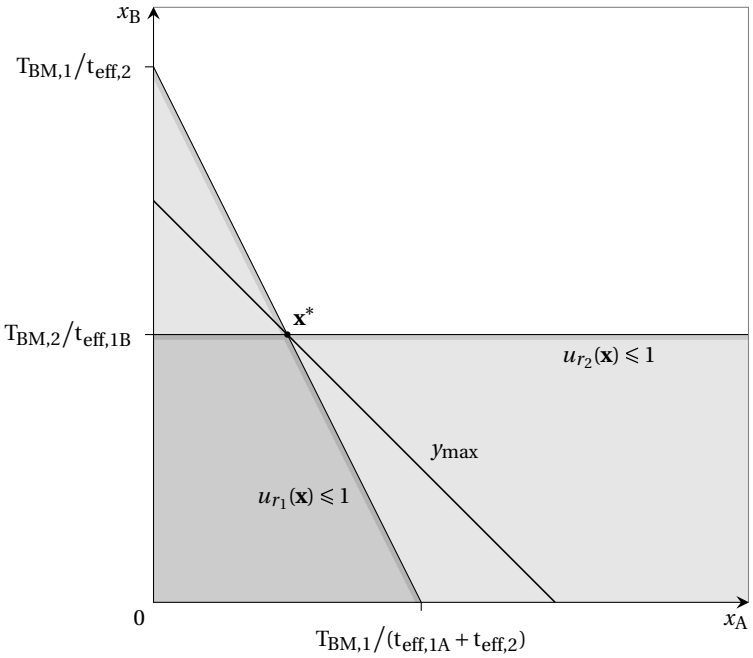
Einsetzen des Teilergebnisses aus Gleichung (2.46) in Gleichung (2.43) führt schließlich zu dem gesuchten Ausdruck für die TEK $y_{\max}$ in diesem Beispiel:

$$y_{\max} = \begin{cases} \dfrac{\mathrm{T_{BM,1}} + \dfrac{\mathrm{t_{eff,1A}}\,\mathrm{T_{BM,1}}}{\mathrm{t_{eff,2}}}}{\mathrm{t_{eff,1A}} + \mathrm{t_{eff,2}}}, & \text{falls } \dfrac{\mathrm{T_{BM,1}}}{\mathrm{t_{eff,2}}} \leqslant \dfrac{\mathrm{T_{BM,2}}}{\mathrm{t_{eff,1B}}}, \\ \dfrac{\mathrm{T_{BM,1}} + \dfrac{\mathrm{t_{eff,1A}}\,\mathrm{T_{BM,2}}}{\mathrm{t_{eff,1B}}}}{\mathrm{t_{eff,1A}} + \mathrm{t_{eff,2}}}, & \text{sonst.} \end{cases} \tag{2.47}$$

Nach Umformung entspricht Gleichung (2.47) der zu beweisenden Gleichung (2.40). □



(a) Fall $T_{BM,1}/t_{eff,2} \leqslant T_{BM,2}/t_{eff,1B}$



(b) Fall $T_{BM,1}/t_{eff,2} > T_{BM,2}/t_{eff,1B}$

Abbildung 2.7: Grafische Lösung einer alternativen Verknüpfung (Beispiel AL2). Im zweidimensionalen Fall werden die Bedingungen, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ nicht größer als eins sein darf, jeweils durch Geraden repräsentiert, welche die Ortsvektoren aller gültigen Lösungen $\mathbf{x}$ begrenzen. Gesucht ist die TEK y_{max}, die als Gerade mit dem Anstieg $\Delta x_B/\Delta x_A = -1$ durch den Ortsvektor der optimalen Lösung $\mathbf{x}^*$ verläuft. Im Fall (a) beschreibt die Lösung $\mathbf{x}^*$ den Schnittpunkt der Geraden $u_{r_1}(\mathbf{x}) = 1$ und $x_A = 0$, im Fall (b) den Schnittpunkt der zwei Geraden $u_{r_{1/2}}(\mathbf{x}) = 1$.

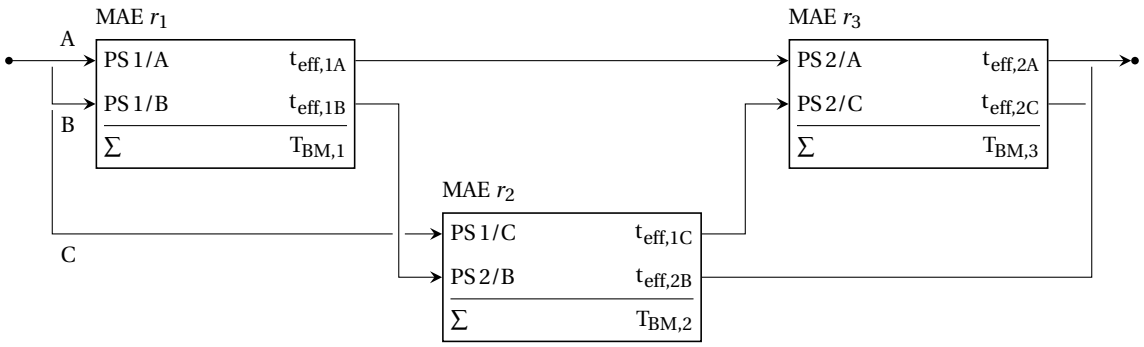


Abbildung 2.8: Schema einer alternativen Verknüpfung (Beispiel AL3). Beispiel AL3 beschreibt den Fall einer alternativen Verknüpfung dreier Prozessschritte (PS) 1/A, 1/B und 1/C. Jeder von diesen ist sequenziell mit einem weiteren Prozessschritt 2/A, 2/B bzw. 2/C verbunden. Die MAEs r_1, r_2 und r_3 werden für die Prozessschritte 1/A und 1/B, 1/C und 2/B bzw. 2/A und 2/C genutzt, d. h. für jeweils zwei verschiedene, verkettet miteinander verknüpfte Prozessschritte.

Grafische Lösung. Abbildung 2.7(a) und 2.7(b) stellen die TEK $y_{\max}$ als optimale Lösung eines linearen, zweidimensionalen Problems fallweise grafisch dar. Wie definiert seien die vorgegebenen Zeitgrößen positiv. Für jede der MAEs $r_{1/2}$ muss die Bedingung gelten, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ einen Wert von eins nicht überschreitet. Diese zwei Bedingungen werden im allgemeinen, mehrdimensionalen Fall durch Hyperebenen und im gegebenen, zweidimensionalen Fall des Beispiels durch Geraden repräsentiert, welche den Raum $\mathbb{R}^2_{\geqslant 0}$ aller möglichen Paare $(x_\mathrm{A}, x_\mathrm{B})$ der nichtnegativen reellen Prozessstückzahlen x_A und x_B jeweils in einen abgeschlossenen, gültigen Halbraum $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_{1/2}}(\mathbf{x}) \leqslant 1\}$ und in einen offenen, ungültigen Halbraum $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_{1/2}}(\mathbf{x}) > 1\}$ teilen.

Die konvexe Schnittmenge der gültigen Halbräume, im allgemeinen Fall repräsentiert durch ein Polytop und im vorliegenden, zweidimensionalen Fall durch ein Polygon, bildet die Menge aller gültigen Lösungen $\mathbf{x}$. Die Betragssummennorm der eingeschlossenen Ortsvektoren dieser Lösungen, bestimmt durch die Summe der Koordinaten x_A und x_B, entspricht jeweils der Produktstückzahl y. Gesucht ist demzufolge der Ortsvektor, welcher die Betragssummennorm maximiert und damit zur optimalen Lösung $\mathbf{x}^*$ führt. Durch diesen verläuft schließlich die TEK $y_{\max}$ als Gerade mit dem Anstieg $\Delta x_\mathrm{B} / \Delta x_\mathrm{A} = -1$.

Der erste Halbraum $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_1}(\mathbf{x}) \leqslant 1\}$ entspricht im Fall (a) von Gleichung (2.40) einer Teilmenge des zweiten Halbraums $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_2}(\mathbf{x}) \leqslant 1\}$, weshalb $\mathbf{x}^*$ den Schnittpunkt der Geraden $u_{r_1}(\mathbf{x}) = 1$ mit der Koordinatenachse x_B bezeichnet. Demgegenüber stellt im Fall (b) keiner der Halbräume $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_{1/2}}(\mathbf{x}) \leqslant 1\}$ eine Teilmenge des jeweils anderen dar, und folglich beschreibt $\mathbf{x}^*$ den Schnittpunkt der beiden Geraden $u_{r_{1/2}}(\mathbf{x}) = 1$.

In Beispiel AL2 kann die TEK $y_{\max}$ bestimmt werden, indem zwei Fälle in Abhängigkeit von den vorgegebenen Zeiten unterschieden werden. Im allgemeinen Fall gilt es jedoch, ein lineares, mehrdimensionales Problem zu lösen, wie das nächste, komplexere Beispiel einer verketteten Verknüpfung veranschaulichen soll.

Beispiel AL3

$$\mathrm{AL}\,(\mathrm{AL}\,(\mathrm{SQ}\,(\mathrm{PS}\,1/\mathrm{A}\,[r_1], \mathrm{PS}\,2/\mathrm{A}\,[r_3]), \mathrm{SQ}\,(\mathrm{PS}\,1/\mathrm{B}\,[r_1], \mathrm{PS}\,2/\mathrm{B}\,[r_2])),\ \mathrm{SQ}\,(\mathrm{PS}\,1/\mathrm{C}\,[r_2], \mathrm{PS}\,2/\mathrm{C}\,[r_3]))$$

Analog zu Beispiel SQ1 ist die Fertigung eines Produkts in eine Vorbearbeitung und eine Fertigbearbeitung gegliedert, wie in Abbildung 2.8 dargestellt. Hierfür stehen drei MAEs mit unterschiedlichen Eigenschaften und daraus folgender Eignung zur Verfügung, wobei folgende Bedingungen gelten: Die MAE r_1 (Betriebsmittelzeit $\mathrm{T_{BM,1}}$) kann nur zur Vorbearbeitung eingesetzt werden. Dagegen erlaubt die MAE r_2 (Betriebsmittelzeit $\mathrm{T_{BM,2}}$) sowohl die Ausführung der Vorbearbeitung als auch der Fertigbearbeitung. Die MAE r_3 (Betriebsmittelzeit $\mathrm{T_{BM,3}}$) eignet sich nur zur Fertigbearbeitung. Einschränkend gilt außerdem, dass nacheinander immer zwei verschiedene MAEs zum Einsatz kommen müssen, um Verluste durch Umrüsten der MAEs weitestgehend zu vermeiden.

Gemäß diesen Bedingungen wählt jedes Stück des Produkts einen von drei Prozessen, jeweils entsprechend einem Paar der MAEs $r_{1/3}$, $r_{1/2}$ oder $r_{2/3}$, welche das Stück im Ablauf der Fertigung durchläuft. Das Beispiel beschreibt demnach eine verkettete alternative Verknüpfung dreier Prozessschritte, welche der Vorbearbeitung dienen und im Folgenden mit der Abkürzung 1/A (r_1, effektive Taktzeit $\mathrm{t_{eff,1A}}$), 1/B (r_1, $\mathrm{t_{eff,1B}}$) und 1/C (r_2, $\mathrm{t_{eff,1C}}$) bezeichnet werden sollen. Von diesen drei Prozessschritten ist jeder einzelne sequenziell mit einem weiteren Prozessschritt zur Fertigbearbeitung verknüpft, abgekürzt mit 2/A (r_3, effektive Taktzeit $\mathrm{t_{eff,2A}}$), 2/B (r_2, $\mathrm{t_{eff,2B}}$) bzw. 2/C (r_3, $\mathrm{t_{eff,2C}}$). Die Produktstückzahl y kann variabel auf drei Prozessstückzahlen x_α, $\alpha \in \{\text{'A'}, \text{'B'}, \text{'C'}\}$, verteilt werden, die jeweils den zwei sequenziell verknüpften Prozessschritten $1/\alpha$ und $2/\alpha$ zugeordnet sind. Wie zuvor in Beispiel AL1 und AL2 muss dabei die Bedingung erfüllt sein, dass deren Summe der Produktstückzahl y entspricht, sodass gilt (siehe Definition 2.6):

$$y = x_\mathrm{A} + x_\mathrm{B} + x_\mathrm{C}\,. \tag{2.48}$$

Jede der drei MAEs $r \in \{r_1, r_2, r_3\}$ wird wie zuvor beschrieben zur Ausführung von zwei verschiedenen Prozessschritten genutzt. Analog zu Beispiel AL2 muss die Überlagerung dieser Prozessschritte berücksichtigt werden, um die Auslastung $u_r(\mathbf{x})$ aller MAEs r zu bestimmen. Gleichung (2.20) ist entsprechend für einen Vektor $\mathbf{x} = [x_\mathrm{A}\ x_\mathrm{B}\ x_\mathrm{C}]^\top$ zu erweitern, indem die drei Prozessstückzahlen x_α, $\alpha \in \{\text{'A'}, \text{'B'}, \text{'C'}\}$, mit den jeweils zugeordneten Taktzeiten gewichtet werden. Für die einzelnen MAEs r folgt daraus in diesem Beispiel:

$$u_{r_1}(\mathbf{x}) = \frac{\mathrm{t_{eff,1A}}\,x_\mathrm{A} + \mathrm{t_{eff,1B}}\,x_\mathrm{B}}{\mathrm{T_{BM,1}}}\,, \tag{2.49a}$$

$$u_{r_2}(\mathbf{x}) = \frac{\mathrm{t_{eff,1C}}\,x_\mathrm{C} + \mathrm{t_{eff,2B}}\,x_\mathrm{B}}{\mathrm{T_{BM,2}}}\,, \tag{2.49b}$$

$$u_{r_3}(\mathbf{x}) = \frac{\mathrm{t_{eff,2A}}\,x_\mathrm{A} + \mathrm{t_{eff,2C}}\,x_\mathrm{C}}{\mathrm{T_{BM,3}}}\,. \tag{2.49c}$$

Die TEK y_max ist analog zu den vorangegangenen Beispielen definiert als die maximale Produktstückzahl y, wobei in dem betrachteten Beispiel die Bedingung erfüllt sein muss,

dass die Auslastung $u_r(\mathbf{x})$ keiner der drei MAEs $r \in \{r_1, r_2, r_3\}$ einen Wert von eins übersteigt. In formaler Schreibweise lautet die zugehörige Vorschrift:

$$y_{\max} = \max\left\{x_\mathrm{A} + x_\mathrm{B} + x_\mathrm{C} \colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2, r_3\}\ u_r(\mathbf{x}) \leqslant 1\right\}. \tag{2.50}$$

Wie in Beispiel AL1 und AL2 wird die gesuchte TEK $y_{\max}$ zum Finden einer Lösung als eine Funktion optimaler Teillösungen x_α^*, $\alpha \in \{\text{'A'}, \text{'B'}, \text{'C'}\}$, dargestellt. Das heißt in diesem Fall, dass die Produktstückzahl y der TEK $y_{\max}$ entsprechen muss, wenn die drei Teillösungen x_α^* als Summanden in Gleichung (2.48) einsetzt werden. Demnach gilt formal:

$$y_{\max} = x_\mathrm{A}^* + x_\mathrm{B}^* + x_\mathrm{C}^*. \tag{2.51}$$

Dem Verständnis des Folgenden dient ein Rückblick. Um in Beispiel AL2 eine Lösung für die letztlich gesuchte TEK $y_{\max}$ zu finden, wurde in einem ersten Schritt das Maximum der Prozessstückzahl x_B ermittelt. Hierbei galt die Bedingung, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ nicht den vorgegebenen maximalen Wert von eins überschreiten durfte, wie schon im Verlauf dieses Kapitels erläutert. Daraus folgte die Teillösung x_B^*, deren Einsetzen es im Anschluss daran erlaubte, einen geschlossenen Ausdruck für die TEK $y_{\max}$ zu ermitteln. In dem Zuge wurde implizit die Teillösung x_A^* bestimmt, die aus der Maximierung der Prozessstückzahl x_A resultiert, wobei zusätzlich zur obigen Bedingung in Bezug auf die Auslastung $u_{r_{1/2}}(\mathbf{x})$ eine Nebenbedingung erfüllt sein musste. Die Nebenbedingung besagte, dass die im ersten Schritt maximierte Prozessstückzahl x_B ihren Wert behalten muss. Das führt zu der Idee, den Prozessstückzahlen Prioritäten zuzuordnen und sie gemäß dieser Priorisierung iterativ zu maximieren, wobei zur Beibehaltung der Werte, die in allen vorigen Schritten gefunden wurden, in jedem Schritt Nebenbedingungen hinzugefügt werden. Es stellt sich die Frage, ob das Planungsziel der maximalen Kapazitäten im Allgemeinen erreichbar ist, indem eine solche Heuristik zur iterativen Maximierung angewandt wird.

Zur Beantwortung dieser Frage richtet sich der Blick auf Beispiel AL3. Wie oben erläutert, wird im ersten Iterationsschritt das Maximum der Prozessstückzahl mit der Priorität eins unter der Bedingung gesucht, dass im Fall keiner MAE $r \in \{r_1, r_2, r_3\}$ die Auslastung $u_r(\mathbf{x})$ einen Wert von eins überschreitet. In den darauffolgenden Iterationsschritten wird jeweils das Maximum der nächsten Prozessstückzahl bestimmt, mit der gleichen Bedingung in Bezug auf die Auslastung $u_r(\mathbf{x})$ und allen Maxima aus den vorherigen Iterationsschritten als zu erfüllenden Nebenbedingungen. Tabelle 2.3 stellt die Ergebnisse einer iterativen Maximierung der Prozessstückzahlen x_A, x_B und x_C in allen sechs Permutationen der gesuchten TEK $y_{\max}$ und der zugrundeliegenden optimalen Lösung $\mathbf{x}^*$ gegenüber. Es wird deutlich, dass die maximal erreichbare Stückzahl des Produkts durch die Anwendung einer Heuristik wie der beschriebenen selbst im einfachsten Fall jeweils gleicher effektiver Taktzeiten und gleicher Betriebsmittelzeiten nicht gefunden wird.

Das gilt analog für das Planungsziel der minimalen Investitionen. Hierzu soll angenommen werden, dass die geplante Stückzahl genau der TEK $y_{\max}$ entspricht. Erhöht man nun stetig die Produktstückzahl y, indem die Prozessstückzahlen x_A, x_B und x_C iterativ maximiert werden, dann überschreitet die Auslastung $u_r(\mathbf{x})$ einer MAE r einen Wert von eins, obwohl eine gültige Lösung existiert, die keine Überlastung und somit keine Investitionen erfordert.

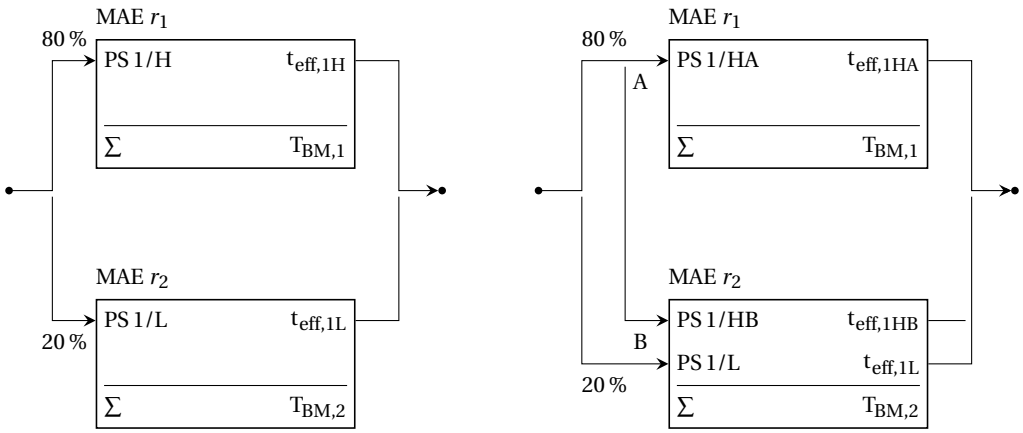
Zielfunktion(en)	Stückzahlverteilung				Auslastung		
	x_A	x_B	x_C	y	$u_{r_1}(\mathbf{x})$	$u_{r_2}(\mathbf{x})$	$u_{r_3}(\mathbf{x})$
(1a) Iterative Maximierung von x_A, x_B, x_C (oder x_A, x_C, x_B)							
$x_A^* = \max\{x_A\colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2, r_3\}\ u_r(\mathbf{x}) \leqslant 1\}$	**600**	0	0	600	1,0	0,0	1,0
$x_B^* = \max\{x_B\colon [\text{s. } x_A^*] \ldots \wedge x_A = x_A^*\}$	600	**0**	0	600	1,0	0,0	1,0
$x_C^* = \max\{x_C\colon [\text{s. } x_A^*] \ldots \wedge x_A = x_A^* \wedge x_B = x_B^*\}$	600	0	**0**	**600**	1,0	0,0	1,0
(1b) Iterative Maximierung von x_B, x_C, x_A (oder x_B, x_A, x_C)							
$x_B^* = \max\{x_B\colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2, r_3\}\ u_r(\mathbf{x}) \leqslant 1\}$	0	**600**	0	600	1,0	1,0	0,0
$x_C^* = \max\{x_C\colon [\text{s. } x_B^*] \ldots \wedge x_B = x_B^*\}$	0	600	**0**	600	1,0	1,0	0,0
$x_A^* = \max\{x_A\colon [\text{s. } x_B^*] \ldots \wedge x_B = x_B^* \wedge x_C = x_C^*\}$	**0**	600	0	**600**	1,0	1,0	0,0
(1c) Iterative Maximierung von x_C, x_A, x_B (oder x_C, x_B, x_A)							
$x_C^* = \max\{x_C\colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2, r_3\}\ u_r(\mathbf{x}) \leqslant 1\}$	0	0	**600**	600	0,0	1,0	1,0
$x_A^* = \max\{x_A\colon [\text{s. } x_C^*] \ldots \wedge x_C = x_C^*\}$	**0**	0	600	600	0,0	1,0	1,0
$x_B^* = \max\{x_B\colon [\text{s. } x_C^*] \ldots \wedge x_C = x_C^* \wedge x_A = x_A^*\}$	0	**0**	600	**600**	0,0	1,0	1,0
(2) TEK $y_{\max}$ und optimale Lösung $\mathbf{x}^* = [x_A^*\ x_B^*\ x_C^*]^\top$							
$y_{\max} = \max\{x_A + x_B + x_C\colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2, r_3\}\ u_r(\mathbf{x}) \leqslant 1\}$	**300**	**300**	**300**	**900**	1,0	1,0	1,0

Tabelle 2.3: Lösungen einer iterativen Heuristik und optimale Lösung. Betrachtet wird das Beispiel AL3, wobei effektive Taktzeiten von 2,5 Stunden und Betriebsmittelzeiten von 1500 Stunden gegeben seien. Unter dieser Annahme sind die Ergebnisse einer iterativen Maximierung der Prozessstückzahlen x_A, x_B und x_C in jeder möglichen Reihenfolge aufgeführt (1a–c). Die Ergebnisse sind der optimalen Lösung $\mathbf{x}^*$ gegenübergestellt (2). Daraus geht hervor, dass die Heuristik die gesuchte TEK $y_{\max}$ unabhängig von der Reihenfolge nicht erreicht.

Daraus folgt, dass im allgemeinen Fall von mehr als zwei Prozessstückzahlen anstelle einer solchen Heuristik ein Verfahren der linearen Optimierung angewandt werden muss, um die optimale Lösung zu bestimmen (z. B. Simplex-Verfahren, Innere-Punkte-Verfahren oder Ellipsoidmethode, für eine Einführung siehe Korte und Vygen 2012).

Zusammenfassend schaffen alternative Verknüpfungen Freiheitsgrade zur Verteilung der Stückzahl des Produkts. Diese Freiheitsgrade werden formal durch nichtnegative reelle Variablen, sogenannte Prozessstückzahlen, repräsentiert. Um die optimale Lösung zu bestimmen, genügt es in Beispiel AL1 und AL2, jeweils zwei Fälle zu unterscheiden. Dass dies im Allgemeinen jedoch nicht hinreichend ist, belegt Beispiel AL3. Wie die Zahlen in Tabelle 2.2 auf Seite 38 verdeutlichen, werden an einem typischen Produktionsstandort mitunter mehr als 100 Produkte in mehr als 1000 Prozessschritten hergestellt, wobei bestimmte Produktionsanlagen mehrfach durchlaufen werden. In dem gleichem Maße wächst die Zahl der Freiheitsgrade und der wechselseitigen Abhängigkeiten.

Im Anschluss folgen zwei Beispiele selektiver Verknüpfungen, welche sowohl Eigenschaften alternativer als auch sequenzieller Verknüpfungen besitzen. Danach stehen die notwendigen Werkzeuge zur Verfügung, um jeden in der Praxis möglichen Ablauf zur Fertigung und Montage eines Produkts in diesem Kontext zu beschreiben und basierend darauf Kapazität, Auslastung und Investitionen zu kalkulieren.



(a) Beispiel SL1 (b) Beispiel SL2

Abbildung 2.9: Schemata selektiver Verknüpfungen (Beispiel SL1 und SL2). Beispiel SL1 beschreibt die selektive Verknüpfung zweier Prozessschritte (PS), die für 80 bzw. 20 Prozent der Stückzahl ausgeführt werden und denen je eine MAE r_1 bzw. r_2 zugeordnet ist. Beispiel SL2 erweitert Beispiel SL1, indem für die vorigen 80 Prozent wahlweise ein alternativ verknüpfter Prozessschritt ausgeführt wird. In diesem Fall wird wie für die übrigen 20 Prozent der Stückzahl die MAE r_2 genutzt.

2.5.3 Selektive Verknüpfung

Es folgen zwei Beispiele einer selektiven Verknüpfung, des zuletzt verbleibenden Verknüpfungstyps in dieser Arbeit. Gemäß Definition 2.7 wird die Stückzahl wie im Fall alternativer Verknüpfungen auf die jeweiligen Prozessschritte verteilt. Das Verhältnis der Stückzahlanteile ist allerdings im Unterschied zu alternativen Verknüpfungen nicht variabel, sondern durch eine Quote q im Intervall $(0,1)$ fix vorgegeben (z. B. 80 Prozent, d. h. mit einem Wert von 0,8). Um wahlfrei ausführbare Prozessschritte in Verbindung mit einzelnen Varianten eines Produkts zu berücksichtigen, werden alternative und selektive Verknüpfungen nacheinander verkettet. Nach einem einführenden Beispiel soll ein komplexeres beschrieben werden, welches einen solchen Fall darstellt.

Beispiel SL1 SL(PS 1/H[r_1], PS 1/L[r_2], 0,8)

Im Ablauf der Montage eines Produkts werden eine Standardvariante (mit einer Quote von 80 Prozent) und eine Sondervariante (verbleibende Stückzahl entsprechend 20 Prozent) unterschieden, wie in Abbildung 2.9(a) dargestellt. Bedingt durch ihre Konstruktion müssen die zwei Varianten mit Hilfe verschiedener Vorrichtungen montiert werden. Aus diesem Grund ist die Montage der Standardvariante (Prozessschritt 1/H, effektive Taktzeit $t_{eff,1H}$) nur unter Verwendung einer eigens eingerichteten Standardmontagelinie r_1 (Betriebsmittelzeit $T_{BM,1}$) möglich, und für die Montage der Sondervariante (Prozessschritt 1/L, effektive Taktzeit $t_{eff,1L}$) muss eine Sondermontagelinie r_2 (Betriebsmittelzeit $T_{BM,2}$) genutzt werden. Dieser Fall fand sich im Zuge der Erstellung dieser Arbeit an vielen Standorten der Bosch Rexroth AG wieder, oft erweitert um zusätzliche Varianten.

Das Beispiel beruht auf einer selektiven Verknüpfung zweier Prozessschritte 1/H und 1/L zur Montage jeweils einer Variante des Produkts, wobei eine Quote q mit einem Wert von 0,8 gegeben ist. Die Produktstückzahl y wird auf die Stückzahlen $q\,y$ und $(1-q)\,y$ verteilt, welche der Standardvariante bzw. Sondervariante (Prozessschritt 1/H bzw. 1/L) entsprechen (siehe Definition 2.7). Wie im Fall einer alternativen Verknüpfung ist die Summe der Stückzahlanteile, die jeweils durch die Prozessschritte verlaufen, gleich der Produktstückzahl y. Ihr Verhältnis ist jedoch nicht variabel, sondern durch die vorgegebene Quote q festgelegt. In Bezug auf die Auslastung $u_{r_{1/2}}(y)$ entspricht der Fall einer sequenziellen Verknüpfung von zwei Prozessschritten mit der effektiven Taktzeit $q\,\mathrm{t_{eff,1H}}$ bzw. $(1-q)\,\mathrm{t_{eff,1L}}$, wobei die Produktstückzahl jeden von diesen vollständig durchlaufen muss. Da den Prozessschritten jeweils eine der zwei MAEs $r_{1/2}$ zugeordnet ist, kann die Auslastung $u_{r_{1/2}}(y)$ wie folgt durch Einsetzen in Gleichung (2.20) bestimmt werden:

$$u_{r_1}(y) = \frac{q\,\mathrm{t_{eff,1H}}\,y}{\mathrm{T_{BM,1}}}, \qquad u_{r_2}(y) = \frac{(1-q)\,\mathrm{t_{eff,1L}}\,y}{\mathrm{T_{BM,2}}}. \tag{2.52a, 2.52b}$$

Wie in den Beispielen SQ1 und SQ3, in denen jeweils eine sequenzielle Verknüpfung von Prozessschritten mit zwei MAEs $r_{1/2}$ gegeben ist, bezeichnet die TEK $y_{\max}$ die maximale Produktstückzahl y unter der Bedingung, dass die Auslastung $u_{r_{1/2}}(y)$ einen Wert von eins nicht überschreitet. Drückt man dies in formaler Schreibweise aus, heißt das:

$$y_{\max} = \max\left\{y\colon\ y \geqslant 0 \wedge \forall r \in \{r_1, r_2\}\ u_r(y) \leqslant 1\right\}. \tag{2.53}$$

Einsetzen der Auslastung $u_{r_{1/2}}(y)$ aus Gleichung (2.52a, 2.52b) führt zu dem gesuchten Ausdruck für die TEK $y_{\max}$, welcher Gleichung (2.53) entspricht:

$$y_{\max} = \min\left\{\frac{\mathrm{T_{BM,1}}}{q\,\mathrm{t_{eff,1H}}}, \frac{\mathrm{T_{BM,2}}}{(1-q)\,\mathrm{t_{eff,1L}}}\right\}. \tag{2.54}$$

Das Beispiel zeigt, dass eine selektive Verknüpfung vergleichbar mit einer alternativen jedem Prozessschritt einen Anteil der Stückzahl zuordnet. Das Verhältnis ist im Gegensatz zu alternativen Verknüpfungen fix vorgegeben. Gleich einer sequenziellen Verknüpfung kann die Produktstückzahl y nicht weiter erhöht werden, sobald die Auslastung $u_r(y)$ von mindestens einer der genutzten MAEs r einen Wert von eins erreicht.

Die Komplexität wächst, indem selektive und alternative Verknüpfungen nacheinander verkettet werden, wie das letzte Beispiel belegen soll.

Beispiel SL2 SL(AL(PS 1/HA $[r_1]$, PS 1/HB $[r_2]$), PS 1/L $[r_2]$, 0,8)

Die Grundlage bildet das zuvor beschriebene Beispiel SL1. Unterschieden werden folglich abermals eine Standardvariante (mit einer Quote von 80 Prozent) und eine Sondervariante (Stückzahl entsprechend 20 Prozent), wie in Abbildung 2.9(b) dargestellt. Analog zum vorigen Fall erfolgt die Montage der Standardvariante (Prozessschritt 1/HA, effektive Taktzeit $\mathrm{t_{eff,1HA}}$) mit einer Standardmontagelinie r_1 (Betriebsmittelzeit $\mathrm{T_{BM,1}}$) und die Montage der Sondervariante (Prozessschritt 1/L, effektive Taktzeit $\mathrm{t_{eff,1L}}$) unter Einsatz einer Sonder-

montagelinie r_2 (Betriebsmittelzeit $T_{BM,2}$). Als Erweiterung gegenüber Beispiel SL1 ist es darüber hinaus möglich, die Standardvariante mit der Sondermontagelinie r_2 zu montieren (Prozessschritt 1/HB, effektive Taktzeit $t_{eff,1HB}$), um deren Kapazität auszuschöpfen. Der Grund ist, dass die Sondermontagelinie r_2 nicht auf eine bestimmte Variante eingerichtet ist und auch die Montage der Standardvariante erlaubt. Im Vergleich zur Standardmontagelinie r_1 verlängert sich in der Praxis jedoch zumeist die Taktzeit.

Aus der obigen Beschreibung kann geschlossen werden, dass in dem Beispiel eine Verkettung von Verknüpfungen vorliegt. Gegeben ist eine alternative Verknüpfung von zwei Prozessschritten 1/HA und 1/HB, welche die Montage der Standardvariante mit jeweils einer der MAEs $r_{1/2}$ repräsentieren. Das Ergebnis ist selektiv mit einem Prozessschritt 1/L zur Montage der Sondervariante verbunden, dem wie oben beschrieben die MAE r_2 zugeordnet ist. Für die selektive Verknüpfung ist wie in Beispiel SL1 eine Quote mit einem Wert von 0,8 festgelegt. Die Produktstückzahl y wird als Folge auf die Stückzahlen $q\,y$ und $(1-q)\,y$ verteilt, welche der Standardvariante (Prozessschritt 1/HA oder 1/HB) bzw. der Sondervariante (Prozessschritt 1/L) entsprechen (siehe Definition 2.7). Gemäß der alternativen Verknüpfung kann die Stückzahl $q\,y$ variabel auf zwei Prozessstückzahlen x_{HA} und x_{HB} (Prozessschritt 1/HA bzw. 1/HB) verteilt werden. Analog zu den vorherigen Beispielen in diesem Abschnitt muss dabei die Bedingung erfüllt sein, dass deren Summe gleich der Stückzahl $q\,y$ ist (siehe Definition 2.6). Damit gilt formal:

$$y = \frac{x_{HA} + x_{HB}}{q}\,. \tag{2.55}$$

Gleichung (2.55) führt zu einer wichtigen Erkenntnis: Sind die Prozessstückzahlen x_{HA} und x_{HB} der Standardvariante des Produkts (Prozessschritt 1/HA bzw. 1/HB) gegeben, muss die Stückzahl $(1-q)\,y$ der Sondervariante (Prozessschritt 1/L) bedingt durch die fix vorgegebene Quote q immer gleich dem Ausdruck $(1-q)\,(x_{HA}+x_{HB})/q$ sein.

Es folgt die Herleitung der Gleichungen für die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der zwei MAEs $r_{1/2}$ abhängig von einem Vektor $\mathbf{x} = [x_{HA}\ x_{HB}]^\top$. Über die Standardmontagelinie r_1 ist bekannt, dass diese nur für die Montage der Standardvariante in der Stückzahl x_{HA} genutzt wird (Prozessschritt 1/HA). Entsprechend gilt nach Einsetzen in Gleichung (2.20):

$$u_{r_1}(\mathbf{x}) = \frac{t_{eff,1HA}\,x_{HA}}{T_{BM,1}}\,. \tag{2.56a}$$

Die Sondermontagelinie r_2 wird von beiden Varianten beansprucht. Zum einen dient sie der Montage der Sondervariante in der Stückzahl $(1-q)\,y$ (Prozessschritt 1/L), wie erläutert übereinstimmend mit $(1-q)\,(x_{HA}+x_{HB})/q$. Zum anderen wird sie zur Montage der Standardvariante in der Stückzahl x_{HB} eingesetzt (Prozessschritt 1/HB). Um die resultierende Auslastung $u_{r_2}(\mathbf{x})$ zu bestimmen, muss die Überlagerung dieser Prozessschritte berücksichtigt werden, indem Gleichung (2.20) wie folgt erweitert wird:

$$u_{r_2}(\mathbf{x}) = \frac{\dfrac{(1-q)\,t_{eff,1L}}{q}\left[x_{HA}+x_{HB}\right] + t_{eff,1HB}\,x_{HB}}{T_{BM,2}}\,. \tag{2.56b}$$

Durch Umformung von Gleichung (2.56b) ist es möglich, die Auslastung $u_{r_2}(\mathbf{x})$ der MAE r_2 als eine Funktion zweier Argumente x_{HA} und x_{HB} darzustellen, die jeweils mit einem Koeffizienten multipliziert und addiert werden (nützlich für den folgenden Beweis):

$$u_{r_2}(\mathbf{x}) = \frac{\dfrac{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}{q}\,x_{\mathrm{HA}} + \left\lceil \mathrm{t}_{\mathrm{eff,1HB}} + \dfrac{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}{q} \right\rceil x_{\mathrm{HB}}}{\mathrm{T}_{\mathrm{BM},2}}. \tag{2.56c}$$

Die TEK $y_{\max}$ bezeichnet analog zu den Beispielen AL1 und AL2, welche die Verteilung auf zwei Prozessstückzahlen beschreiben, das Maximum der Produktstückzahl y. Dabei muss wie zuvor die Bedingung erfüllt sein, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der MAEs $r_{1/2}$ jeweils nicht größer als eins ist. In formaler Schreibweise heißt das:

$$y_{\max} = \max\left\{ \frac{x_{\mathrm{HA}} + x_{\mathrm{HB}}}{q} : \mathbf{x} \geqslant \mathbf{0} \wedge \forall r \in \{r_1, r_2\}\ u_r(\mathbf{x}) \leqslant 1 \right\}. \tag{2.57}$$

In der Praxis werden die Montagelinien oft für verschiedene Varianten genutzt. Das heißt, die Montage der Standardvariante erfolgt mit der Standardmontagelinie r_1 und die Montage der Sondervariante, da in diesem Fall nicht anders möglich, mit der Sondermontagelinie r_2 (Prozessschritt 1/HA bzw. 1/L). Sobald aber die Auslastung $u_{r_1}(\mathbf{x})$ der MAE r_1 einen Wert von eins erreicht, wird ein bestimmter Teil x_{HB} der Stückzahl $q\,y$ der Standardvariante mit der MAE r_2 (Prozessschritt 1/HB) montiert, um deren Kapazität zu nutzen. Voraussetzung ist, dass die Auslastung $u_{r_2}(\mathbf{x})$ der MAE r_2 noch nicht einem Wert von eins entspricht. Der verbleibende Stückzahlanteil x_{HA} durchläuft unverändert die MAE r_1 (Prozessschritt 1/HA). Unter den gegebenen Bedingungen ist die TEK $y_{\max}$ gesucht, die erzielt werden kann, indem die Stückzahl $q\,y$ der Standardvariante optimal auf die zwei Prozessstückzahlen x_{HA} und x_{HB} verteilt wird. Das führt zu folgendem Ergebnis.

Theorem 2.2: Technische Kapazität $y_{\max}$ für Beispiel SL2. Im Ablauf der Fertigung und Montage seien die Prozessschritte 1/HA, 1/HB und 2 (effektive Taktzeit $\mathrm{t}_{\mathrm{eff,1HA}}$, $\mathrm{t}_{\mathrm{eff,1HB}}$ bzw. $\mathrm{t}_{\mathrm{eff,1L}}$) unter Nutzung von zwei MAEs $r_{1/2}$ (Betriebsmittelzeit $\mathrm{T}_{\mathrm{BM},1/2}$) alternativ und selektiv verknüpft, wie in Abbildung 2.9(b) dargestellt. Für die selektive Verknüpfung sei eine fixe Quote q im Intervall $(0,1)$ gegeben. In dem Fall gilt für die TEK $y_{\max}$:

$$y_{\max} = \begin{cases} \dfrac{\mathrm{T}_{\mathrm{BM},2}}{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}, & \text{falls } \dfrac{\mathrm{T}_{\mathrm{BM},2}}{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}} \leqslant \dfrac{\mathrm{T}_{\mathrm{BM},1}}{q\,\mathrm{t}_{\mathrm{eff,1HA}}}, \quad \text{(a)} \\ \dfrac{\dfrac{\mathrm{t}_{\mathrm{eff,1HB}}\,\mathrm{T}_{\mathrm{BM},1}}{\mathrm{t}_{\mathrm{eff,1HA}}} + \mathrm{T}_{\mathrm{BM},2}}{q\,\mathrm{t}_{\mathrm{eff,1HB}} + (1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}, & \text{sonst.} \quad \text{(b)} \end{cases} \tag{2.58}$$

Interpretation. Vor dem formalen Beweis soll wieder das Verständnis dafür geschaffen werden, wie sich die Terme in Gleichung (2.58) zusammensetzen. Die TEK $y_{\max}$ ist analog zu Beispiel AL2 durch eine Fallunterscheidung bestimmt, die auf einem Vergleich der beiden

Ausdrücke $T_{BM,2}/\left[(1-q)\,t_{eff,1L}\right]$ und $T_{BM,1}/(q\,t_{eff,1HA})$ basiert. Diese beschreiben jeweils das Maximum der Produktstückzahl y, wobei die Auslastung $u_{r_{1/2}}(\mathbf{x})$ einen Wert von eins nicht überschreiten darf und nur ein Prozessschritt betrachtet wird. Der erste Ausdruck betrifft die Montage der Sondervariante mit der MAE r_2 (Prozessschritt 1/L), der zweite die Montage der Standardvariante mit der MAE r_1 (Prozessschritt 1/HA). Der Grund für den Vergleich der zwei Ausdrücke ist der folgende: Da die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der MAEs $r_{1/2}$ nach oben begrenzt ist, kann der maximale Wert der Produktstückzahl y erreicht werden, wenn zunächst nur diese beiden Prozessschritte ausgeführt werden.

Im Fall (a) von Gleichung (2.58) ist der Ausdruck $T_{BM,2}/\left[(1-q)\,t_{eff,1L}\right]$ kleiner als oder gleich dem Ausdruck $T_{BM,1}/(q\,t_{eff,1HA})$. Das heißt, wenn ausschließlich die zwei genannten Prozessschritte berücksichtigt werden und die Produktstückzahl y stetig gesteigert wird, dann erreicht die Auslastung $u_{r_{2/1}}(\mathbf{x})$ der MAE r_2 zuerst oder zusammen mit jener der MAE r_1 einen Wert von eins. Da aber zur Montage der Sondervariante stets die MAE r_2 genutzt und bezüglich der Varianten die Quote q erfüllt werden muss, kann die Produktstückzahl y in diesem Fall nicht weiter erhöht werden. Demnach folgt die TEK y_{max} unmittelbar aus der Betriebsmittelzeit $T_{BM,2}$ und einer effektiven Taktzeit $(1-q)\,t_{eff,1HB}$.

Im Fall (b) der oben aufgeführten Gleichung ist der Ausdruck $T_{BM,2}/\left[(1-q)\,t_{eff,1L}\right]$ größer als der Ausdruck $T_{BM,1}/(q\,t_{eff,1HA})$. Das heißt, werden wie in dem vorigen Fall nur die zwei Prozessschritte berücksichtigt und wird die Produktstückzahl y stetig erhöht, dann erreicht die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der MAE r_1 zuerst einen Wert von eins. In diesem Fall ist es jedoch möglich, die Produktstückzahl y weiter zu steigern, indem die MAE r_2 nicht mehr nur zur Montage der Sondervariante genutzt wird, sondern auch eingesetzt wird, um die Standardvariante zu montieren. Aus diesem Grund wird die Kapazität, welche für die Montage der Standardvariante mit der MAE r_1 bereitsteht, gedanklich der MAE r_2 hinzugefügt. Folglich werden die Betriebsmittelzeiten $T_{BM,1/2}$ addiert. Da sich die effektiven Taktzeiten $t_{eff,1HB}$ und $t_{eff,1HA}$ aber im Allgemeinen unterscheiden, muss die Betriebsmittelzeit $T_{BM,1}$ zuvor mit dem Faktor $t_{eff,1HB}/t_{eff,1HA}$ multipliziert werden. Danach kann unabhängig von der MAE $r_{1/2}$ für die Montage der Standardvariante die effektive Taktzeit $t_{eff,1HB}$ vorausgesetzt werden. Bedingt durch das fix vorgegebene Verhältnis, in welchem die Stückzahlen der Varianten zueinander stehen, müssen die effektiven Taktzeiten $t_{eff,1HB}$ und $t_{eff,1L}$ mit der Quote q bzw. mit dem Faktor $1-q$ multipliziert und addiert werden. Zusammenfassend bestimmen in dem betrachteten Fall eine Betriebsmittelzeit $t_{eff,1HB}\,T_{BM,1}/t_{eff,1HA} + T_{BM,2}$ und eine effektive Taktzeit $q\,t_{eff,1HB} + (1-q)\,t_{eff,1L}$ die TEK y_{max}, entsprechend den verwendeten Faktoren und Summanden in der oben stehenden Gleichung (2.58).

Beweis. Um die Beweisidee aus Beispiel AL2 fortzuführen, wird die TEK y_{max} als eine Funktion zweier optimaler Teillösungen x^*_{HA} und x^*_{HB} sowie eines dritten Arguments, der fix vorgegebenen Quote q, ausgedrückt. In diesem Zusammenhang bedeutet optimal, dass die Produktstückzahl y, die gemäß Gleichung (2.55) dem Ausdruck $(x_{HA} + x_{HB})/q$ entspricht, mit der gesuchten TEK y_{max} übereinstimmt, wenn x^*_{HA} bzw. x^*_{HB} anstelle der Prozessstückzahlen eingesetzt werden. Formal ausgedrückt lautet diese Bedingung:

$$y_{max} = \frac{x^*_{HA} + x^*_{HB}}{q}\,. \tag{2.59}$$

Alle Lösungen $\mathbf{x} = [x_{\mathrm{HA}}\ x_{\mathrm{HB}}]^\top$ müssen die Bedingung erfüllen, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ keiner der beiden MAEs $r_{1/2}$, die nach Gleichung (2.56a, 2.56c) aus einem Paar von x_{HA} und x_{HB} folgt, einen Wert größer als eins annimmt. Um einen Ausdruck für die TEK $y_{\max}$ zu finden, welcher Gleichung (2.57) erfüllt, ist unter allen Lösungen $\mathbf{x}$, die gemäß obiger Bedingung gültig sind, die optimale Lösung $\mathbf{x}^* = [x^*_{\mathrm{HA}}\ x^*_{\mathrm{HB}}]^\top$ gesucht. Laut Gleichung (2.59) ist diese dadurch gekennzeichnet, dass der Ausdruck $(x^*_{\mathrm{HA}} + x^*_{\mathrm{HB}})/q$ mit den optimalen Teillösungen x^*_{HA} und x^*_{HB} der TEK $y_{\max}$ entspricht. Die Definition der TEK $y_{\max}$ verlangt, dass keine andere gültige Lösung $\mathbf{x}$ existiert, nach welcher die Produktstückzahl y den zugehörigen Wert der optimalen Lösung $\mathbf{x}^*$, d. h. die TEK $y_{\max}$, übersteigt.

Die Prozessstückzahl x_{HB} und insbesondere die optimale Teillösung x^*_{HB} sind allein durch die Bedingung beschränkt, dass die Auslastung $u_{r_2}(\mathbf{x})$ der MAE r_2 kleiner als oder gleich eins sein muss. Wenn angenommen wird, dass die Auslastung $u_{r_2}(\mathbf{x}^*)$ kleiner als eins ist, dann existiert eine gültige Lösung $\mathbf{x}$, nach welcher die Prozessstückzahl x_{HA} der optimalen Teillösung x^*_{HA} entspricht, der Wert für x_{HB} größer ist als x^*_{HB} und die Produktstückzahl y als Folge die TEK $y_{\max}$ übersteigt. Da dies aber die Definition der TEK $y_{\max}$ verbietet, liegt ein Widerspruch vor. Daher muss die Annahme falsch, die Negation der Annahme wahr und die Auslastung $u_{r_2}(\mathbf{x}^*)$ gleich eins sein. Nach Umstellung von Gleichung (2.56c) gilt für die optimale Teillösung x^*_{HB} unter dieser Voraussetzung (der Ausdruck wird nicht vereinfacht, da dies für den folgenden Schritt nützlich ist):

$$x^*_{\mathrm{HB}} = \frac{\mathrm{T}_{\mathrm{BM},2} - \dfrac{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}{q}\,x^*_{\mathrm{HA}}}{\mathrm{t}_{\mathrm{eff,1HB}} + \dfrac{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}{q}}\,. \tag{2.60}$$

Durch Einsetzen von Gleichung (2.60) in Gleichung (2.59) ist es im nächsten Schritt möglich, die TEK $y_{\max}$ als Funktion der optimalen Teillösung x^*_{HA} darzustellen:

$$y_{\max} = \frac{x^*_{\mathrm{HA}}}{q} + \frac{\mathrm{T}_{\mathrm{BM},2} - \dfrac{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}{q}\,x^*_{\mathrm{HA}}}{q\left[\mathrm{t}_{\mathrm{eff,1HB}} + \dfrac{(1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}{q}\right]} = \frac{\mathrm{t}_{\mathrm{eff,1HB}}\,x^*_{\mathrm{HA}} + \mathrm{T}_{\mathrm{BM},2}}{q\,\mathrm{t}_{\mathrm{eff,1HB}} + (1-q)\,\mathrm{t}_{\mathrm{eff,1L}}}\,. \tag{2.61}$$

Die Zeitgrößen seien positiv und die Quote q im Intervall $(0,1)$ gegeben. Betrachtet man alle möglichen Lösungen $\mathbf{x}$, für welche die Auslastung $u_{r_{1/2}}(\mathbf{x})$ der MAE r_1 nicht größer als eins und jene der MAE r_2 gleich eins ist, muss die Teillösung x^*_{HA} der optimalen Lösung $\mathbf{x}^*$ maximal sein. Der Grund hierfür ist wieder, dass die Annahme des umgekehrten Falls zu einem Widerspruch führt. Wird zu diesem Zweck angenommen, dass eine Lösung $\mathbf{x}$ existiert, welche die zwei zuvor genannten Bedingungen in Bezug auf die Auslastung $u_{r_{1/2}}(\mathbf{x})$ erfüllt und nach welcher die Prozessstückzahl x_{HA} größer ist als die Teillösung x^*_{HA}, dann übersteigt die TEK $y_{\max}$ für x_{HA} laut Gleichung (2.61) den Wert für x^*_{HA}. Allgemein ausgedrückt gilt dementsprechend für die optimale Teillösung x^*_{HA}:

$$x^*_{\mathrm{HA}} = \max\left\{x_{\mathrm{HA}} \colon \mathbf{x} \geqslant \mathbf{0} \wedge u_{r_1}(\mathbf{x}) \leqslant 1 \wedge u_{r_2}(\mathbf{x}) = 1\right\}. \tag{2.62}$$

Einsetzen der Auslastung $u_{r_{1/2}}(\mathbf{x})$ aus Gleichung (2.56a, 2.56c) führt zu folgendem Ausdruck für die optimale Teillösung x^*_{HA}, welcher die vorige Gleichung (2.62) erfüllt:

$$x^*_{\mathrm{HA}} = \min\left\{ \frac{\mathrm{T}_{\mathrm{BM},1}}{\mathrm{t}_{\mathrm{eff},1\mathrm{HA}}}, \frac{q\,\mathrm{T}_{\mathrm{BM},2}}{(1-q)\,\mathrm{t}_{\mathrm{eff},1\mathrm{L}}} \right\}. \tag{2.63}$$

Durch Umformulierung von Gleichung (2.63) ist die optimale Teillösung x^*_{HA} als Ergebnis einer Fallunterscheidung darstellbar:

$$x^*_{\mathrm{HA}} = \begin{cases} \dfrac{q\,\mathrm{T}_{\mathrm{BM},2}}{(1-q)\,\mathrm{t}_{\mathrm{eff},1\mathrm{L}}}, & \text{falls } \dfrac{\mathrm{T}_{\mathrm{BM},2}}{(1-q)\,\mathrm{t}_{\mathrm{eff},1\mathrm{L}}} \leqslant \dfrac{\mathrm{T}_{\mathrm{BM},1}}{q\,\mathrm{t}_{\mathrm{eff},1\mathrm{HA}}}, \\ \dfrac{\mathrm{T}_{\mathrm{BM},1}}{\mathrm{t}_{\mathrm{eff},1\mathrm{HA}}}, & \text{sonst.} \end{cases} \tag{2.64}$$

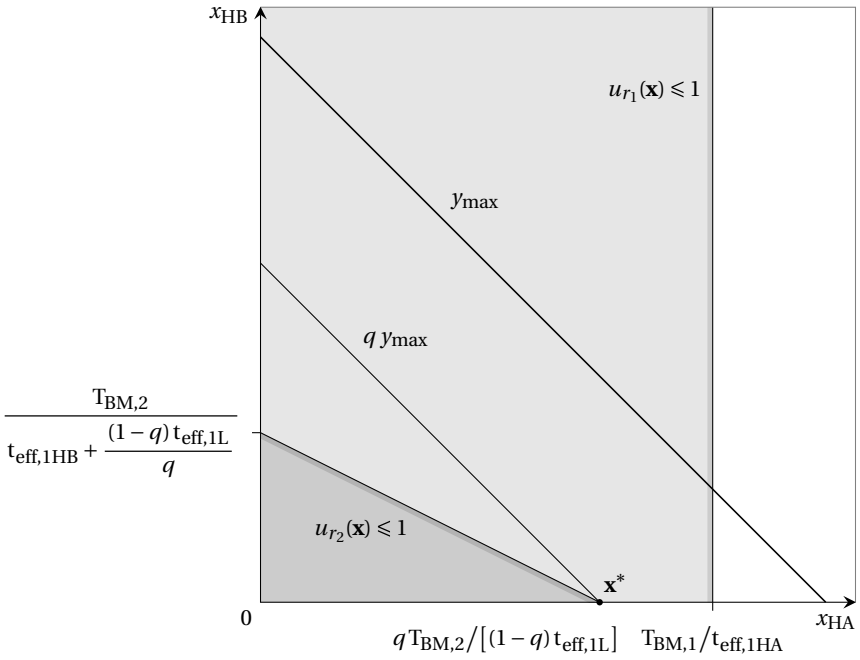
Setzt man das Teilergebnis aus Gleichung (2.64) in Gleichung (2.61) ein, folgt daraus der gesuchte Ausdruck für die TEK $y_{\max}$ in dem gegebenen Beispiel:

$$y_{\max} = \begin{cases} \dfrac{\dfrac{q\,\mathrm{t}_{\mathrm{eff},1\mathrm{HB}}\,\mathrm{T}_{\mathrm{BM},2}}{(1-q)\,\mathrm{t}_{\mathrm{eff},1\mathrm{L}}} + \mathrm{T}_{\mathrm{BM},2}}{q\,\mathrm{t}_{\mathrm{eff},1\mathrm{HB}} + (1-q)\,\mathrm{t}_{\mathrm{eff},1\mathrm{L}}}, & \text{falls } \dfrac{\mathrm{T}_{\mathrm{BM},2}}{(1-q)\,\mathrm{t}_{\mathrm{eff},1\mathrm{L}}} \leqslant \dfrac{\mathrm{T}_{\mathrm{BM},1}}{q\,\mathrm{t}_{\mathrm{eff},1\mathrm{HA}}}, \\ \dfrac{\dfrac{\mathrm{t}_{\mathrm{eff},1\mathrm{HB}}\,\mathrm{T}_{\mathrm{BM},1}}{\mathrm{t}_{\mathrm{eff},1\mathrm{HA}}} + \mathrm{T}_{\mathrm{BM},2}}{q\,\mathrm{t}_{\mathrm{eff},1\mathrm{HB}} + (1-q)\,\mathrm{t}_{\mathrm{eff},1\mathrm{L}}}, & \text{sonst.} \end{cases} \tag{2.65}$$

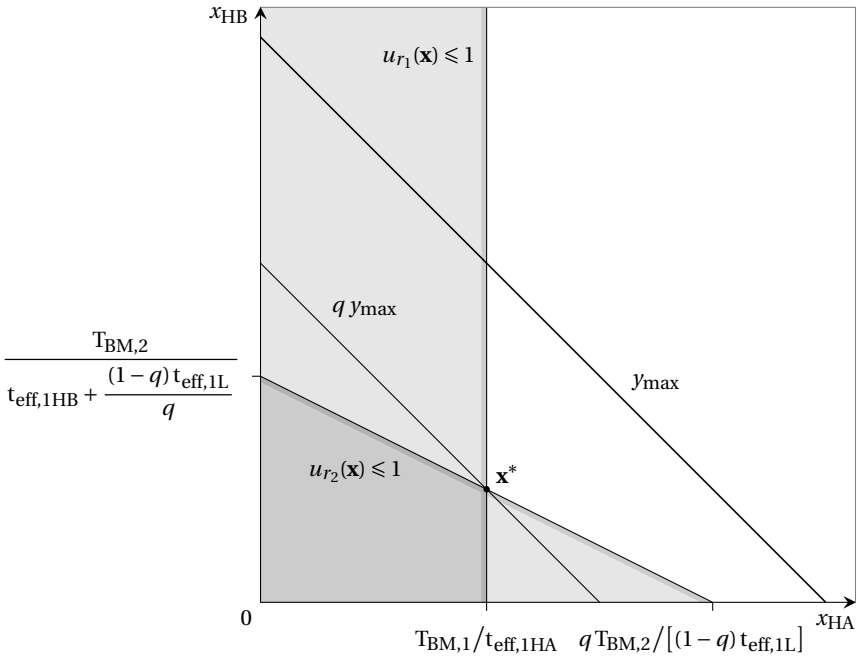
Gleichung (2.65) entspricht nach Umformung der zu zeigenden Gleichung (2.58). □

Grafische Lösung. Wie in Beispiel AL2 stellen Abbildung 2.10(a) und 2.10(b) die TEK $y_{\max}$ als optimale Lösung eines linearen, zweidimensionalen Problems fallweise grafisch dar. Ihrer Definition entsprechend seien die Zeitgrößen positiv und die Quote q der selektiven Verknüpfung im Intervall $(0, 1)$ gegeben. Für jede der MAEs $r_{1/2}$ muss die Bedingung gelten, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ nicht größer als eins ist. Die beiden Bedingungen werden im gegebenen, zweidimensionalen Fall des Beispiels durch Geraden repräsentiert, welche den Raum $\mathbb{R}^2_{\geqslant 0}$ aller Paare $(x_{\mathrm{HA}}, x_{\mathrm{HB}})$ der nichtnegativen reellen Prozessstückzahlen x_{HA} und x_{HB} jeweils in einen abgeschlossenen, gültigen Halbraum $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_{1/2}}(\mathbf{x}) \leqslant 1\}$ und in einen offenen, ungültigen Halbraum $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_{1/2}}(\mathbf{x}) > 1\}$ teilen.

Die konvexe Schnittmenge der gültigen Halbräume, dargestellt als ein Polygon, bildet abermals die Menge der gültigen Lösungen $\mathbf{x}$. Die Betragssummennorm der Ortsvektoren dieser Lösungen entspricht nun allerdings nicht der Produktstückzahl y, sondern jeweils der Stückzahl $q\,y$ der Standardvariante. Da die Quote q eine Konstante darstellt, ist wie in Beispiel AL2 derjenige Ortsvektor gesucht, welcher die Betragssummennorm maximiert und zur optimalen Lösung $\mathbf{x}^*$ führt. Durch diesen verläuft die maximale Stückzahl $q\,y_{\max}$ der Standardvariante als Gerade mit dem Anstieg $\Delta x_{\mathrm{HB}} / \Delta x_{\mathrm{HA}} = -1$. Parallel dazu, skaliert um den konstanten Faktor $1/q$, verläuft die TEK $y_{\max}$.



(a) Fall $T_{BM,2}/[(1-q)\,t_{eff,1L}] \leqslant T_{BM,1}/(q\,t_{eff,1HA})$



(b) Fall $T_{BM,2}/[(1-q)\,t_{eff,1L}] > T_{BM,1}/(q\,t_{eff,1HA})$

Abbildung 2.10: Grafische Lösung einer selektiven Verknüpfung (Beispiel SL2). Die Bedingungen, dass die Auslastung $u_{r_{1/2}}(\mathbf{x})$ nicht größer als eins sein darf, werden jeweils durch Geraden repräsentiert, welche die Ortsvektoren aller gültigen Lösungen $\mathbf{x}$ begrenzen. Durch den Ortsvektor der optimalen Lösung $\mathbf{x}^*$ führt die Stückzahl $q\,y_{max}$ als Gerade mit dem Anstieg $\Delta x_{HB}/\Delta x_{HA} = -1$. Parallel dazu verläuft die TEK y_{max}, skaliert um den Faktor $1/q$. Im Fall (a) bezeichnet die Lösung $\mathbf{x}^*$ den Schnittpunkt der Geraden $u_{r_2}(\mathbf{x}) = 1$ und $x_{HB} = 0$, im Fall (b) den Schnittpunkt der zwei Geraden $u_{r_{1/2}}(\mathbf{x}) = 1$.

Im Fall (a) von Gleichung (2.58) entspricht der zweite Halbraum $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_2}(\mathbf{x}) \leqslant 1\}$ einer Teilmenge des ersten Halbraums $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_1}(\mathbf{x}) \leqslant 1\}$, weshalb $\mathbf{x}^*$ den Schnittpunkt der Geraden $u_{r_2}(\mathbf{x}) = 1$ mit der Koordinatenachse x_{HA} beschreibt. Im Fall (b) ist dagegen keiner der Halbräume $\{\mathbf{x} \in \mathbb{R}^2_{\geqslant 0} \colon u_{r_{1/2}}(\mathbf{x}) \leqslant 1\}$ eine Teilmenge des jeweils anderen, und als Folge dessen bezeichnet $\mathbf{x}^*$ den Schnittpunkt der beiden Geraden $u_{r_{1/2}}(\mathbf{x}) = 1$.

2.6 Anforderungen in Bezug auf die Modellierung und die Optimierung

Grundlage jeder Softwareentwicklung ist eine formale oder zumindest informelle Definition von Anforderungen. Diese Aussage gilt gleichermaßen für das lineare Wasserfallmodell und für Varianten der agilen, iterativen Softwareentwicklung (siehe u. a. Cohn 2004). Der Abschnitt fasst die Ergebnisse aus diesem Kapitel zusammen, indem Anforderungen an eine geeignete Software zur Unterstützung der Planung in diesem Kontext kategorisiert und formuliert werden (siehe Abschnitt 2.1). Eine solche Software soll es erlauben, ein geplantes System von Wertströmen zu modellieren und gemäß den Planungszielen zu optimieren (siehe Abschnitt 2.2 und 2.3). Die Kalkulation von Kapazität, Auslastung und Investitionen muss im einfachen Fall eines Prozessschritts und im allgemeinen Fall mehrerer verknüpfter Prozessschritte den erläuterten Vorgaben der Planung entsprechen und auf unzulässige Vereinfachungen verzichten (siehe Abschnitt 2.4 und 2.5).

2.6.1 Kategorisierung möglicher Anforderungen

Anforderungen zur Beschreibung neuer oder geänderter Leistungsmerkmale werden nach ihrer Wichtigkeit unterschieden. Hintergrund ist, dass die zur Verfügung stehenden Ressourcen zielgerichtet den wichtigsten Anforderungen zugeordnet werden sollen. Die technische Norm ISO/IEC/IEEE 29148:2011(E) definiert zu diesem Zweck die folgenden drei Kategorien (S. 67, vorherige, sinngemäß gleiche Fassung siehe IEEE Std 830-1998, S. 7):

(1) *Essenziell* (engl. *essential*): Anforderungen, welche das neue/geänderte System (eine zu entwickelnde Software) erfüllen *soll* (engl. *shall*). Für jede essenzielle Anforderung sollte erläutert werden, welche Auswirkungen erwartet werden müssen, wenn die betreffende Anforderung nicht umgesetzt wird.

(2) *Wünschenswert* (engl. *desirable*): Anforderungen, welche das neue/geänderte System erfüllen *sollte* (engl. *should*). Wünschenswerte Anforderungen sollten priorisiert werden, und für jede von ihnen sollte der Grund erläutert werden, warum es sich jeweils um eine wünschenswerte Anforderung handelt.

(3) *Optional* (engl. *optional*): Anforderungen, welche das neue/geänderte System erfüllen *könnte* (engl. *might*). Wie zuvor schon wünschenswerte sollten auch optionale Anforderungen priorisiert werden, und es sollte jeweils erläutert werden, aus welchem Grund diese als optional eingeschätzt werden.

Das Ziel dieser Arbeit ist die Beantwortung von grundsätzlichen mathematischen und algorithmischen Fragen, welche den Einsatz einer Software zur strategischen Planung der TEK in der Serienfertigung betreffen. Die Betrachtung beschränkt sich auf die essenziellen Anforderungen in Bezug auf die Modellierung und die Optimierung. Weitere, in der Praxis wichtige Anforderungen (wie z. B. solche, die sich auf die Benutzerschnittstelle beziehen) werden nicht diskutiert, da dies den gegebenen Rahmen sprengen würde. Das Gleiche gilt für alle wünschenswerten und optionalen Anforderungen.

Für das Verständnis ist es nützlich, die Kategorie der essenziellen Anforderungen weiter zu untergliedern. Dafür werden zwei Typen eingeführt: (1a) *Basisanforderungen* müssen erfüllt sein, um ein geplantes System von Wertströmen vollständig zu modellieren und das Modell wie definiert zu optimieren; (1b) *Leistungsanforderungen* stellen zusätzlich sicher, dass die Planungsziele effizient, transparent und flexibel erreicht werden.

Die Erfüllung der Leistungsanforderungen unterscheidet Software, welche sich für den praktischen Einsatz in einem Unternehmen eignet, von solcher, deren Einsatz theoretisch möglich, aber nicht wirtschaftlich ist. Die Erfahrungen bei der Bosch Rexroth AG münden in der Erkenntnis, dass die Einführung neuer Software zur Unterstützung der Planung nur dann gelingt, wenn deren Nutzung wirksam dazu beiträgt, die Effizienz, Transparenz und Flexibilität in der Planung zu steigern (vgl. Karlsen et al. 2005). Diese drei Begriffe werden in der vorliegenden Arbeit wie folgt gebraucht:

Effizienz: In diesem Kontext verstanden als die Zahl erstellter Modelle im Verhältnis zur Zeit, welche ein Planer für deren Erstellung benötigt. Je effizienter die Planung erfolgt, desto weniger Ressourcen werden im Verlauf der Planung gebunden. Im gegebenen Fall betrifft das insbesondere die Arbeitszeit des Planers.

Transparenz: In diesem Kontext verstanden als die intuitive Abbildung eines geplanten Systems durch ein Modell. Je transparenter die Planung erfolgt, desto geringer ist die Wahrscheinlichkeit von Fehlern und desto leichter sind die erzielten Ergebnisse gegenüber Mitarbeitern und der Leitung des Unternehmens zu erklären.

Flexibilität: In diesem Kontext verstanden als die Fähigkeit, ein Modell zu erstellen und zu adaptieren, wenn sich die Produkte/Prozesse ändern. Je flexibler die Planung erfolgt, desto schneller ist ein Modell für einen Standort mit spezifischen Produkten/Prozessen erstmalig anzulegen und fortlaufend anzupassen.

Da die Volatilität der Kundennachfrage im Allgemeinen steigt, müssen Pläne in immer kürzeren Zyklen aktualisiert werden. Als Folge gewinnen die drei Kriterien an Bedeutung, an erster Stelle die Flexibilität. Abgesehen von der Effizienz ist die Erfüllung der anderen Kriterien allerdings nur schwierig zu messen. Neben Basisanforderungen werden deshalb im Folgenden überprüfbare Leistungsanforderungen formuliert, deren Erfüllung letztlich zu höherer Effizienz, Transparenz und/oder Flexibilität führt.

2.6.2 Formulierung der essenziellen Anforderungen

Die nachfolgenden Basisanforderungen und Leistungsanforderungen, jeweils bezeichnet als A[1–14] bzw. A[1–14]$^+$, beschreiben eine Software, die eine vollständige und gleichzeitig effiziente, transparente und flexible Planung ermöglicht. Diese Software besteht aus einem Modellierer und einem Optimierer. Der Modellierer soll es ermöglichen, ein System von Wertströmen durch ein geeignetes grafisches Modell abzubilden und das grafische Modell danach automatisch in ein mathematisches umzuwandeln. Auf dieser Grundlage sollen anschließend mit Hilfe des Optimierers die Planungsziele erreicht werden.

Basisanforderung A1: Superposition aller Wertströme. Die Software soll es erlauben, alle Wertströme des Systems an einem Produktionsstandort in einem einzigen Modell zusammenzufassen (siehe Abschnitt 2.2). Dabei soll berücksichtigt werden, dass Prozessschritten in verschiedenen Wertströmen des Systems je nach Anwendungsfall dieselben MAEs zugeordnet sein können, diese aber zu jedem beliebigen Zeitpunkt nur für einen einzigen Prozessschritt genutzt werden dürfen (siehe Definition 2.1).

Erläuterung: Wie in Abschnitt 2.2 beschrieben, müssen Wertströme in einem System vereinigt werden, um die Abhängigkeiten zu berücksichtigen, welche durch die mehrfache Zuordnung von MAEs verursacht werden. Daher muss auch ein Modell, das zur Abbildung des Systems dienen soll, wie gefordert all diese Wertströme einschließen.

Basisanforderung A2: Systemstruktur. Um die Planung insbesondere im allgemeinen Fall verknüpfter Prozessschritte nicht unzulässig zu vereinfachen, soll das Modell die Struktur des abgebildeten Systems widerspiegeln. Dies schließt die folgenden Ebenen, Elemente und Ressourcen des jeweils gegebenen Systems ein (siehe Abschnitt 2.2):

(1) *Ebenen* (Wertströme): rekursiv zusammengesetzt aus Prozessschritten/Prozessen und jeweils einem Produkt zugeordnet (siehe Definition 2.9);
(2) *Elemente* (Prozessschritte/Prozesse): im Wertstrom eines Produkts ggf. rekursiv sequenziell, alternativ und/oder selektiv verknüpft (siehe Definition 2.1–2.8);
(3) *Ressourcen* (MAEs): ggf. mehreren Prozessschritten in einem oder verschiedenen Wertströmen zugeordnet (siehe Definition 2.1, Basisanforderung A1).

Zu Prozessschritten/Prozessen und MAEs: In einem System, in dem mehrere Prozesse eines Produkts dieselbe MAE durchlaufen, existieren die folgenden zwei Anwendungsfälle, deren Modellierung mit der Software gleichermaßen möglich sein soll:

(a) Die Prozesse werden vor dem Durchlaufen der jeweiligen MAE *vereinigt*. Das heißt, im Wertstrom des Produkts wird nur ein einziger Prozessschritt unter Nutzung dieser MAE definiert. Die Taktzeiten und alle nachfolgenden Prozessschritte stimmen in dem Fall überein (siehe Beispiel AL2, Prozessschritt 2).

(b) Die Prozesse werden *nicht vereinigt*. Das heißt, es werden mehrere Prozessschritte unter Nutzung der MAE definiert. Abhängig vom geplanten Ablauf unterscheiden sich die Taktzeiten (siehe Beispiel SL2, Prozessschritt 1/HB und 1/L) und/oder die nachfolgenden Prozessschritte (siehe Beispiel AL3, Prozessschritt 1/A und 1/B).

Erläuterung: Die zuvor erwähnten Abhängigkeiten sind durch Prozessschritte verschiedener Produkte begründet, denen ein und dieselbe MAE zugeordnet ist. Darüber hinaus existieren Abhängigkeiten zwischen Prozessschritten eines Produkts, die miteinander verknüpft sind. Um ein System vollständig zu modellieren, müssen die entsprechenden Bedingungen berücksichtigt werden. Das betrifft insbesondere die Zuordnung von Ausgaben zu Eingaben des Informationsflusses, wie im Folgenden festgehalten wird.

Basisanforderung A3: Systemschnittstelle. Die Ein- und Ausgaben des Modells sollen jeweils den Ein- bzw. Ausgaben des Informationsflusses an der Schnittstelle des abgebildeten Systems entsprechen (siehe Abschnitt 2.3). Das heißt, nach Vorgabe der Produktstückzahlen und der Prozessstückzahlen soll das Modell die daraus resultierende Auslastung aller MAEs zurückgeben. Die Zuordnung von Ausgaben zu Eingaben soll im einfachen Fall eines einzelnen Prozessschritts und im allgemeinen Fall mehrerer verknüpfter Prozessschritte mit der beschriebenen Kalkulation übereinstimmen (siehe Abschnitt 2.4 bzw. 2.5).

Erläuterung: Die Schnittstelle eines Systems erfüllt den Zweck, mit der Umwelt einerseits Material und andererseits Informationen auszutauschen, wie in Abschnitt 2.3 erläutert wurde. Durch die Abbildung mit einem geeigneten Modell, welches den Eingaben des Informationsflusses die gleichen Ausgaben zuordnet, ist es möglich, das System zur Planung durch das Modell zu ersetzen. Der Anwender[3] bzw. der Optimierer als Komponente der Software tauscht hierzu die Ein-/Ausgaben mit dem Modell aus, um das Modell stellvertretend für das System gemäß den Planungszielen zu optimieren.

Basisanforderung A4: sequenzielle Verknüpfung. Bei der Zuordnung von Ausgaben zu Eingaben (siehe Basisanforderung A3) soll das Modell die Bedingungen berücksichtigen, die für alle *sequenziell* verknüpften Prozessschritte in dem abgebildeten System gelten, wobei Verknüpfungen ggf. verkettet sind. Je nach Anwendungsfall liegen Verkettungen von Verknüpfungen verschiedener Typen vor (siehe Abschnitt 2.2, Definition 2.5).

Erläuterung: Der allgemeine Fall mehrerer verknüpfter Prozessschritte umfasst sequenzielle, alternative und selektive, ggf. verkettete Verknüpfungen. Die vorherige Basisanforderung A4 und die zwei nachfolgenden detaillieren in diesem Sinne die weiter oben aufgeführte Basisanforderung A3. Ist ein Prozessschritt sequenziell verknüpft, wird dessen Stückzahl nicht nur durch die jeweils zugeordnete MAE begrenzt, sondern zusätzlich auch durch diejenigen MAEs, welche den verknüpften Prozessschritten zugeordnet sind.

Basisanforderung A5: alternative Verknüpfung. Bei der Zuordnung von Ausgaben zu Eingaben (siehe Basisanforderung A3) soll das Modell die Bedingungen berücksichtigen, die für alle *alternativ* verknüpften Prozessschritte in dem abgebildeten System gelten, wobei Verknüpfungen ggf. verkettet sind (vgl. Basisanforderung A4). Als eine Eingabe des Modells soll jedem Operanden einer alternativen Verknüpfung eine nichtnegative reelle Prozessstückzahl zugeordnet werden, welche den Stückzahlanteil beschreibt, der durch den Operanden verläuft (siehe Abschnitt 2.2, Definition 2.6).

[3] Der Anwender ist in erster Linie der Planer, welcher die technischen Kapazitäten, die notwendigen Investitionen und die erwartete Auslastung der Produktionsanlagen ermittelt. Die organisatorische Funktion ist bei der Bosch Rexroth AG vom betrachteten Standort abhängig.

Erläuterung: Die nichtnegativen reellen Prozessstückzahlen sind laut der oben genannten Definition der alternativen Verknüpfung Variablen. Zusammen mit den Stückzahlen, die für alle Produkte vorgegeben werden, bilden sie die Eingaben des Modells und spannen einen unendlichen Suchraum auf (siehe Basisanforderung A3).

Basisanforderung A6: selektive Verknüpfung. Bei der Zuordnung von Ausgaben zu Eingaben (siehe Basisanforderung A3) soll das Modell die Bedingungen berücksichtigen, die für alle *selektiv* verknüpften Prozessschritte in dem abgebildeten System gelten, wobei Verknüpfungen ggf. verkettet sind (vgl. Basisanforderung A4). Als Parameter des Modells soll jeder selektiven Verknüpfung eine Quote zugeordnet werden, welche den relativen Stückzahlanteil des ersten Operanden festlegt. Der danach verbleibende Stückzahlanteil soll durch den zweiten Operanden verlaufen (siehe Abschnitt 2.2, Definition 2.7).

Erläuterung: Wie im Fall einer sequenziellen Verknüpfung (siehe Basisanforderung A4) wird die Stückzahl eines Prozessschritts durch weitere Bedingungen begrenzt, falls dieser mit anderen selektiv verknüpft ist. Um die Anforderungen an die Modellierung abzuschließen, wird nun die Darstellung des Modells betrachtet.

Leistungsanforderung A7$^+$: grafische Modellierung. Die Software soll es erlauben, das gegebene System durch ein *grafisches* Modell abzubilden. Zu diesem Zweck soll dem Anwender eine Menge von Symbolen zur Verfügung stehen. Durch die geeignete Kombination dieser Symbole soll es möglich sein, die Struktur des Systems wie zuvor beschrieben eindeutig abzubilden (siehe Abschnitt 2.2, Basisanforderung A2).

Erläuterung: Um die Vorteile aufzuzeigen, die ein grafischer Modellierungsansatz bietet, werden folgende grundsätzliche Möglichkeiten der Modellierung unterschieden:

(1) textuell (Programmiersprachen, z. B. Java, oder Beschreibungssprachen, z. B. XML),
(2) tabellarisch (Software zur Erstellung von Tabellenkalkulationen, z. B. Microsoft Excel),
(3) formularbasiert (Software mit Formulareingabemasken, z. B. SAP ERP),
(4) grafisch (Software zur Erstellung von Vektorgrafiken, z. B. Microsoft Visio).

Bei der Bosch Rexroth AG wurden in der Vergangenheit Tabellenkalkulationen genutzt, um die strategische Planung der TEK zu unterstützen. Die offenkundigen Schwächen von Tabellenkalkulationen in der Praxis, die auch textuellen und formularbasierten Ansätzen immanent sind, führten zur Anforderung, Systeme grafisch zu modellieren. Es soll auf Basis der Erfahrungen bei der Bosch Rexroth AG begründet werden, warum eine grafische Modellierung demgegenüber effizienter, transparenter und flexibler ist.

Höhere Effizienz grafischer Modellierung: Als Ergebnis der rekursiven Verknüpfung von Prozessschritten ist die Struktur eines Systems von Wertströmen mit einem Netzwerk vergleichbar. Tabellenkalkulationen erlauben jedoch nur eine Modellierung in einem zweidimensionalen Raster. Dieser Gegensatz führt in Bezug auf ihre Nutzung zu folgender Beobachtung: Entweder beschreiben sie das System zwar vollständig, aber auf Kosten der Verständlichkeit, oder sie vereinfachen es, um es verständlich darstellen zu können. Ein grafisches Modell ist nicht nur vollständig, sondern auch verständlich

darstellbar (siehe Ko et al. 2009, S. 754, 759). Das verkürzt die Zeit zur Modellierung und führt zu einer Steigerung der Effizienz in der Planung, sowohl bei der Erstellung als auch bei der Weiterentwicklung eines Modells.

Höhere Transparenz grafischer Modellierung: Wie oben erläutert, führt die rekursive Verknüpfung der Prozessschritte in einem geplanten, abzubildenden System zu einer netzwerkartigen Struktur. Tabellenkalkulationen verbergen die Logik zur Abbildung dieser Struktur in Funktionen und Bezügen zwischen einzelnen Zellen, ggf. erweitert um Programmcode (z. B. Microsoft Excel VBA). Ihre Nutzung birgt daher Fehlerquellen (siehe u. a. Powell et al. 2008, 2009). Ein geeignetes grafisches Modell bringt Muster, Lücken und Engpässe in einem Prozess zum Vorschein (siehe Ko et al. 2009, S. 759), wodurch eine Steigerung der Transparenz in der Planung erreicht wird.

Höhere Flexibilität grafischer Modellierung: Verglichen mit Tabellenkalkulationen ist es deutlich einfacher, ein grafisches Modell zu erstellen und zu adaptieren, sollten sich Produkte/Prozesse ändern. Voraussetzung ist, dass der Anwender die Bedeutung der Symbole und die Möglichkeiten für deren Kombination kennt. Dann gelingt es schneller, spezifische Produkte/Prozesse eines Standorts zu berücksichtigen, um ein Modell anzulegen oder anzupassen. Das gilt insbesondere auch dann, wenn ein anderer Anwender das zu ändernde Modell erstellt hat. Daraus folgt eine kürzere Zeit zur Umsetzung von Änderungen und damit eine Steigerung der Flexibilität.

Ein System grafisch modellieren zu können, genügt jedoch noch nicht. Zudem muss sichergestellt sein, dass der Anwender die zu verwendenden Symbole überblickt und versteht. In Bezug auf die Symbolmenge wird aus diesem Grund Folgendes gefordert.

Leistungsanforderung A8$^+$: minimale Symbolmenge. Betrachtet wird die Zahl der verschiedenen Symbole, die benötigt werden, um ein gegebenes System grafisch zu modellieren (siehe Leistungsanforderung A7$^+$). Diese Zahl soll *minimal* sein, wobei es aber weiterhin möglich sein muss, die Elemente des Systems mit Hilfe der Symbole oder durch deren geeignete Kombination eindeutig voneinander zu unterscheiden.

Erläuterung: Um eine minimale Zahl von Symbolen zu erreichen und gleichzeitig Eindeutigkeit zu gewährleisten, folgt daraus, dass Prozessschritte, MAEs sowie sequenzielle, alternative und selektive Verknüpfungen durch je ein Symbol repräsentiert werden müssen. Die Anforderung entspricht dem inhaltlichen Minimalprinzip ordnungsmäßiger Modellvisualisierung von Deelmann und Loos (2004, S. 290) und ist wie folgt zu begründen.

Die Steigerung der Effizienz, Transparenz und Flexibilität ist umso größer, je geringer die Komplexität des grafischen Modells ist. Diese kann reduziert werden, indem Elemente und Beziehungen zwischen Elementen, die zur Erfüllung der Anforderungen abgebildet werden müssen, möglichst zusammengefasst werden. Jeder der entstehenden Kategorien wird ein Symbol zugeordnet, woraus eine minimale Symbolmenge resultiert, welche dem Zweck des Modells entspricht. Beispielsweise wird mit dem grafischen Standard Business Process Model and Notation (BPMN) das Ziel verfolgt, durch Minimierung der Zahl solcher Kategorien das Verständnis zu fördern (siehe OMG 2013, S. 25).

Wird die Menge der Symbole wie beschrieben gebildet, enthält diese nur solche, die aus den Basisanforderungen A1–A6 abgeleitet werden können. Alle anderen Symbole stellen Informationen dar, die zur Planung in diesem Kontext nicht benötigt werden und in vielen Fällen auch nicht vorliegen. Müsste der Anwender diese zur Erstellung eines grafischen Modells verwenden, dann wäre er gezwungen, Annahmen zu treffen, um die Optima der Planungsziele zu suchen. Ein Modell auf der Grundlage einer minimalen Symbolmenge definiert stattdessen Freiheitsgrade (siehe Basisanforderung A5). Als Ergebnis kann es mit einem geeigneten Optimierer gekoppelt werden, um nicht Annahmen treffen zu müssen, sondern zum Finden der Optima ein effizientes Verfahren einzusetzen.

Im nächsten Schritt gilt es, die Basisanforderungen A2–A6, die sich auf die mathematische Modellierung des Systems beziehen, mit den Leistungsanforderungen A7$^+$ und A8$^+$ zu verbinden, welche die Darstellung des grafischen Modells betreffen.

Leistungsanforderung A9$^+$: Modelltransformation. Die Software soll das grafische, vom Anwender erstellte Modell des gegebenen Systems automatisch in ein mathematisches Modell umwandeln, welches für die nachfolgende Optimierung geeignet ist. Da an dieser Stelle lediglich gefordert wird, dass das Modell den definierten Eingaben die jeweils entsprechenden Ausgaben zuordnet (siehe Basisanforderung A3), schließt die Formulierung im erweiterten Sinne auch Simulationsmodelle ein.

Erläuterung: Für die Optimierung besteht eine Grundvoraussetzung darin, dass das System durch ein mathematisches Modell beschrieben wird. Der grafische Ansatz ist daher nur dann effizienter, transparenter und flexibler als die Alternativen, wenn der Anwender neben dem grafischen Modell nicht zusätzlich auch ein mathematisches erstellen muss.

Es folgen die Anforderungen in Bezug auf die Optimierung des Modells. Die variablen Prozessstückzahlen bilden die Freiheitsgrade des Modells und damit die Dimensionen des Suchraums (siehe Basisanforderung A5). Die möglichen Lösungen, repräsentiert durch Ortsvektoren in diesem Suchraum, ordnen jeder MAE einen Wert für deren Auslastung zu (siehe Basisanforderung A3). Eine Lösung ist gültig, wenn sie sämtliche Bedingungen der drei Verknüpfungstypen erfüllt (siehe Basisanforderung A4–A6). Für die Planungsziele ist jeweils unter allen gültigen Lösungen die optimale gesucht.

Basisanforderung A10: Maximierung der Kapazitäten. Die Software soll auf Grundlage eines vom Anwender erstellten Modells die TEK, d. h. die maximalen Stückzahlen aller Produkte, zurückgeben. Es existieren drei Bedingungen: (1) Die Lösung muss wie oben erläutert gültig sein; (2) die Auslastung keiner MAE darf einen Wert von eins überschreiten; (3) die Stückzahlen aller erforderlichen Komponenten müssen mindestens dem Sekundärbedarf entsprechen (siehe Abschnitt 2.3, Planungsziel der maximalen Kapazitäten). Das Ergebnis soll in jedem Fall eindeutig sein. Das heißt, wenn mehrere Lösungen existieren, von denen keine die anderen dominiert, soll eine Lösung ausgewählt werden. Eine Lösung dominiert eine andere, wenn die erste Lösung die zweite im Hinblick auf die TEK aller Produkte mindestens erreicht und im Fall eines oder mehrerer Produkte übertrifft. Die Strategie, welche die Auswahl einer nicht dominierten Lösung bestimmt, soll vor Beginn der Optimierung feststehen, um stets zum selben Ergebnis zu gelangen.

Erläuterung: Eine Lösung dominiert eine zweite, wenn die Stückzahl der ersten Lösung für kein Produkt kleiner und für mindestens ein Produkt größer als die Stückzahl der zweiten Lösung ist. Die gesuchte optimale Lösung ist dadurch gekennzeichnet, dass sie gültig ist, alle Bedingungen gemäß dem Planungsziel erfüllt und nicht dominiert wird. Das Ergebnis der Optimierung kann abhängig vom gegebenen Fall eine unendliche Menge nicht dominierter Lösungen sein, wie in Abschnitt 2.3 angemerkt wurde. Diese Menge wird im mehrdimensionalen Suchraum durch die Teilmenge der Ortsvektoren repräsentiert, die auf dem Rand eines konvexen Polytops liegen. Ein solches, mehrdeutiges Ergebnis eignet sich nicht als Entscheidungsgrundlage für die Planung in Unternehmen. Um in jedem Fall ein eindeutiges Ergebnis zu erhalten, soll daher eine Strategie zur Auswahl einer Lösung festgelegt werden (z. B. durch die Vorgabe eines Verhältnisses für die TEK aller Produkte).

Basisanforderung A11: Minimierung der Investitionen. Die Software soll die minimale Überlastung aller MAEs zurückgeben, die erforderlich ist, um alle Produkte zu fertigen und zu montieren. Dabei bezeichnet der Begriff der Überlastung den Betrag, um welchen die Auslastung einen Wert von eins überschreitet. Der Anwender soll die Stückzahlen der Produkte vorgeben können, einschließlich aller Komponenten. Einzige Bedingung ist, dass die Lösung gültig sein muss (siehe Abschnitt 2.3, Planungsziel der minimalen Investitionen). Wie oben soll das Ergebnis in jedem Fall eindeutig sein. Werden Prozessschritte mit verschiedenen MAEs alternativ verknüpft, soll der Anwender diejenige festlegen können, die bei Bedarf überlastet wird. Existieren mehrere nicht dominierte Lösungen, soll eine von diesen ausgewählt werden. Eine Lösung dominiert eine andere, wenn die erste Lösung die zweite bezüglich der Überlastung aller MAEs nicht überschreitet und im Fall mindestens einer MAE unterschreitet. Die Strategie, nach welcher die Auswahl erfolgt, soll vor Beginn der Optimierung feststehen (vgl. Basisanforderung A10).

Erläuterung: Wie im Fall der maximalen Kapazitäten kann das Ergebnis eine unendliche Menge nicht dominierter Lösungen sein (siehe Abschnitt 2.3). Das gilt genau dann, wenn alternativ verknüpften Prozessschritten verschiedener Wertströme dieselben MAEs zugeordnet sind (siehe Basisanforderung A1). Folglich ist eine Strategie zur Auswahl einer Lösung festzulegen (z. B. durch den Ausgleich der Überlastung zwischen MAEs).

Basisanforderung A12: Optimierung der Auslastung. Die Software soll die Auslastung aller verfügbaren MAEs der folgenden Iterationsvorschrift entsprechend maximieren und das Ergebnis zurückgeben. Grundlage ist eine Priorisierung der alternativ verknüpften Prozessschritte in den Wertströmen, d. h. der Prozessstückzahlen. Der Anwender soll die Prioritäten vorgeben können, wie auch die Stückzahlen der Produkte einschließlich aller erforderlichen Komponenten. Zunächst ist eine initiale Lösung gesucht. Ausgehend von dieser soll im ersten Iterationsschritt die Prozessstückzahl mit der Priorität eins maximiert werden. Danach soll in jedem Iterationsschritt das Maximum der Prozessstückzahl mit aufsteigender Priorität bestimmt werden, wobei die Teillösungen voriger Iterationsschritte Nebenbedingungen darstellen. Es existieren zwei Bedingungen: (1) Die Lösung bzw. Teillösung muss gültig sein; (2) die Auslastung keiner MAE darf einen Höchstwert von eins überschreiten, wobei dieser Wert um die jeweils minimale Überlastung (Ergebnis aus Basisanforderung A11) zu erhöhen ist (siehe Abschnitt 2.3, Planungsziel der optimalen Auslastung).

Erläuterung: Die Anforderung beschreibt das Planungsziel nicht als geschlossene Zielfunktion, sondern als eine Iterationsvorschrift, nach welcher alle Prozessstückzahlen maximiert werden. Da die Priorisierung alle Prozessstückzahlen, d. h. sämtliche Freiheitsgrade des Modells, einschließt (siehe Basisanforderung A5), ist das Ergebnis in jedem Fall eindeutig. Wie in Abschnitt 2.3 ausgeführt wurde, kann die Iteration mit der vorletzten Priorität enden, da danach auch der Wert der letzten Prozessstückzahl eindeutig bestimmt ist.

Basisanforderung A13: globales Optimum. Bezug nehmend auf die drei Planungsziele (siehe Basisanforderung A10–A12) soll sichergestellt sein, dass die Software das *globale* Optimum zurückgibt. Das heißt, es wird kein *lokales* Optimum akzeptiert, welches nur in einer Umgebung des Suchraums die optimale Lösung darstellt.

Erläuterung: Auf Grundlage der Ergebnisse wird u. a. über Eigenproduktion, Zukauf und Investitionen entschieden. Die Unternehmensleitung muss in die Lage versetzt werden, die bestmögliche Entscheidung unter vollständiger Berücksichtigung aller verfügbaren Informationen zu treffen. Das heißt, für die Planungsziele dürfen keine besseren Lösungen als die gefundenen existieren. Um die besten Lösungen zu bestimmen, ist es unmöglich, den stetigen Suchraum vollständig zu erfassen. Heuristiken und Metaheuristiken (wie z. B. Hill Climbing, Simulated Annealing und genetische Algorithmen) sind nicht geeignet, da sie allenfalls gute Lösungen in vertretbarer Zeit ermitteln. Damit verbleiben exakte, mathematische Verfahren der linearen Optimierung (Simplex-Verfahren o. a.).

Leistungsanforderung A14$^+$: lineare obere Laufzeitschranke. Für eine repräsentative Auswahl realer Systeme (siehe Tabelle 2.2 auf Seite 38) soll die Laufzeit der Optimierung im Verhältnis zur gegebenen Zahl der Prozessschritte nicht größer als 0,15 Sekunden je Prozessschritt sein. Die Zeitmessung stoppt, sobald die Optima aller drei Planungsziele gefunden wurden (siehe Basisanforderung A10–A12). Die Anforderung soll auf einem Testsystem geprüft werden, dessen Leistung dem folgenden entspricht: Microsoft Windows 7, 4 GB RAM, CPU Intel Core i5-2520M, vier Kerne mit 2,50 GHz.[4]

Erläuterung: Die Optimierung erfolgt auf Basis eines Modells, welches der Anwender für das jeweils geplante System erstellt. Nach Prüfung der dabei erzielten Ergebnisse passt der Anwender das Modell an, um die TEK weiter zu steigern und die nötigen Investitionen weiter zu reduzieren. Die Anpassung betrifft nicht die definierten Freiheitsgrade, sondern die grundsätzliche Struktur des Modells, indem der Anwender Prozessschritte/Prozesse, Taktzeiten und Betriebsmittelzeiten ändert. Danach ist das Modell wieder im Hinblick auf seine Freiheitsgrade zu optimieren. Um diesen iterativen Workflow effizient zu gestalten, muss gewährleistet sein, dass die globalen Optima der Planungsziele in einer akzeptablen Laufzeit gefunden werden. Da das zu optimierende Modell die Laufzeit des Optimierers beeinflusst, wird eine repräsentative Auswahl von Planungen vorgegeben, anhand derer geprüft werden soll, ob die Anforderung erfüllt ist. Die Laufzeit muss umso kürzer sein, je geringer die Komplexität des Modells ist, da erwartet wird, dass der Anwender das Modell v. a. zu Beginn in kurzen Intervallen anpasst und optimiert. Daraus folgt eine lineare Schranke für die Laufzeit im Verhältnis zur Komplexität des Modells.

[4] Typisches Rechnersystem eines Planers der Bosch Rexroth AG zum Zeitpunkt der Erstellung

Ein aussagekräftiger Indikator, um die angesprochene Komplexität eines Modells zu quantifizieren, ist die Zahl der Prozessschritte. Diese berücksichtigt sowohl die Zahl der Freiheitsgrade, d. h. der Variablen, als auch die Zahl der Nebenbedingungen. Wenn der Anwender Prozessschritte anlegt, ist für die Freiheitsgrade der Verknüpfungstyp entscheidend. Wird ein Prozessschritt alternativ verknüpft, muss eine Prozessstückzahl eingeführt werden, um den variablen Stückzahlanteil zu beschreiben, welcher diesen Prozessschritt durchläuft. Dagegen wird die Zahl der Nebenbedingungen dadurch bestimmt, ob die zugeordnete MAE allein für diesen Prozessschritt oder gleichzeitig auch für andere genutzt wird. Im ersten Fall muss eine neue Nebenbedingung eingeführt werden, im zweiten Fall muss eine existierende erweitert werden. Als Folge steigen mit der Zahl der Prozessschritte näherungsweise linear auch die Zahlen der Freiheitsgrade und Nebenbedingungen. Im Allgemeinen umfassen Modelle in der Praxis nicht mehr als 2000 Prozessschritte, wie die Zahlen in Tabelle 2.2 zeigen. Für den Anwender wird eine Laufzeit von bis zu fünf Minuten als akzeptabel angenommen, woraus 0,15 Sekunden je Prozessschritt folgen.

Im Mittelpunkt des zurückliegenden Kapitels stand die Analyse und Beschreibung von Systemen, die sich aus Wertströmen zur Fertigung und Montage von Produkten an einem Produktionsstandort zusammensetzen. Den Kontext für die Betrachtung bildete die strategische Planung der technischen Kapazität in der (variantenreichen) Serienfertigung (siehe Abschnitt 2.1). Zum Abschluss liegen folgende Ergebnisse vor:

(1) *Struktur* und *Schnittstelle* eines Systems auf der Grundlage eindeutiger Definitionen für Prozessschritte sowie für deren verschiedenartige, rekursive Verknüpfung zu Prozessen und Wertströmen (siehe Abschnitt 2.2 und 2.3);

(2) *Kalkulation* der Planungsziele für den einfachen Fall eines Prozessschritts und für den allgemeinen Fall verknüpfter Prozessschritte eines Produkts (bedingt die Komplexität und ist übertragbar auf mehrere Produkte, siehe Abschnitt 2.4 und 2.5);

(3) Kategorisierung und Formulierung der essenziellen *Anforderungen* zur Modellierung eines Systems und zur Optimierung des Modells, um die Planungsziele effizient, transparent und flexibel zu erreichen (siehe Abschnitt 2.6).

Damit wurden die Voraussetzungen geschaffen, um den Stand der Technik zu bewerten und die Kernalgorithmen der Software AURELIE zu erläutern, die erstmals alle der formulierten Anforderungen erfüllt. Insbesondere erlaubt es diese, ein gegebenes System vollständig durch ein grafisches, formal eindeutiges Modell abzubilden, in ein mathematisches Modell umzuwandeln und innerhalb der vorgegebenen Laufzeit zu optimieren.

3 Stand der Technik

Ziel dieses Kapitels ist die Bewertung der aktuell verfügbaren Software zur Unterstützung der strategischen Planung technischer Kapazität. Betrachtet werden insbesondere die Möglichkeiten von Software zur Modellierung eines Systems von Wertströmen an einem Standort und zur Optimierung des Modells gemäß den definierten Planungszielen. Hierfür wurden in Kapitel 2 die Grundlagen im Hinblick auf die Struktur und die Schnittstelle des Systems sowie die Kalkulation von Kapazität, Auslastung und Investitionen erarbeitet. Die diesbezüglich zu erfüllenden essenziellen Anforderungen, die in Abschnitt 2.6 kategorisiert und formuliert wurden, bilden den Maßstab für die Bewertung.

Dieses Vorgehen, basierend auf der Formulierung von zu erfüllenden Anforderungen gefolgt von einer Bewertung der verfügbaren Software, entspricht der empfohlenen Praxis für das Projektmanagement zur Einführung neuer Informationssysteme (siehe Ammenwerth und Haux 2005, S. 207 ff, für Fallbeispiele). Falls keine geeignete Standardsoftware existiert, muss auf Grundlage der festgelegten Anforderungen eine Individualsoftware entwickelt und eingeführt werden (siehe Mertens et al. 2012, S. 137, Schlick 2001, S. 201).

Eine vollständige Bewertung jeglicher kommerziell und frei verfügbarer Software ist jedoch vor dem Hintergrund des unerschöpflichen Angebots unmöglich. Das Ziel besteht vielmehr darin, im Rahmen dieser Arbeit eine möglichst umfassende Bewertung durchzuführen. Zunächst werden Typen von Software auf Basis vergleichbarer Möglichkeiten gebildet und anhand definierter Kriterien ausgewählt. Da aber auch das verbleibende Angebot unüberschaubar groß ist, wird jeder Softwaretyp an einem Beispiel evaluiert, welches den Stand der Technik widerspiegelt. Es wird angenommen, dass Software eines Typs im Hinblick auf die Erfüllung der Anforderungen insgesamt bestenfalls vergleichbar, aber nicht besser als das gewählte Beispiel ist. Unter dieser Voraussetzung ist die Bewertung des Beispiels hinreichend auf Software desselben Typs übertragbar.

In Abschnitt 3.1 werden zwei Kriterien definiert, um Softwaretypen für die Bewertung auszuwählen: Es wird gefordert, dass die Software grundsätzlich geeignet ist und in Unternehmen produktiv im Planungsprozess eingesetzt werden kann. Die Anwendung der beiden Kriterien führt zur Auswahl der folgenden vier Typen und Beispiele:

(1) *Software zur Erstellung von Tabellenkalkulationen*,
Bsp. Microsoft Excel, in Abschnitt 3.2;

(2) *Software zur Materialflusssimulation*,
Bsp. Siemens Plant Simulation, in Abschnitt 3.3;

(3) *Software für Supply Chain Management*,
Bsp. SAP APO Supply Network Planning, in Abschnitt 3.4;

(4) *Software zur Prozessmodellierung*,
Bsp. BPMN mit idealem Interpreter und Optimierer, in Abschnitt 3.5.

In Abschnitt 3.6 folgt das Fazit: Da keine Software existiert, die alle Anforderungen erfüllt, wurde im Rahmen der vorliegenden Arbeit die Software AURELIE entwickelt und mit Erfolg weltweit bei der Bosch Rexroth AG eingeführt.

3.1 Auswahl zu evaluierender Softwaretypen

In diesem Abschnitt erfolgt die Auswahl der Softwaretypen, die jeweils an einem für den Stand der Technik charakteristischen Beispiel evaluiert werden sollen. Wie zu Beginn erläutert, werden zwei Kriterien festgelegt, um zu dieser Auswahl zu gelangen: Die Software muss erstens grundsätzlich geeignet sein, d. h. die vollständige Modellierung eines Systems gemäß den Basisanforderungen A1–A6 ermöglichen. Zweitens muss sie auf allen Hierachieebenen in den Planungsabteilungen eines Unternehmens wie z. B. der Bosch Rexroth AG mit Blick auf die Qualifikation der Mitarbeiter produktiv einsetzbar sein.

Ein Beispiel aus der Praxis verdeutlicht, dass die Planung ein arbeitsteiliger, sich über das gesamte Unternehmen erstreckender Prozess ist, und unterstreicht dadurch die Bedeutung des zweiten Kriteriums: Bei der Bosch Rexroth AG ist die strategische Planung der TEK im April 2014 in der Verantwortung von insgesamt 146 Mitarbeitern, die weltweit an 34 Standorten arbeiten. Jeder Standort ist einem von zwölf Erzeugnisgebieten zugeordnet, wobei die Produktionsanlagen auch zur Herstellung von Produkten anderer Erzeugnisgebiete genutzt werden. Die Modellerstellung ist eine Aufgabe der Abteilungen für Fertigungsausführung, Fertigungsplanung oder Zeitwirtschaft des Standorts, in einzelnen Fällen auch der Assistenz der Werkleitung. Periodisch, i. d. R. zu Quartalsbeginn, übermitteln die Planer das Modell zur Prüfung an die Zentralbereichsabteilung des jeweiligen Erzeugnisgebiets.

Diese Arbeitsteilung führt zu dem zweiten Kriterium, dass Software, die für eine Bewertung in Frage kommt, von Planern aller Hierarchieebenen in Unternehmen produktiv einsetzbar sein muss. Zum einen besitzen die Mitarbeiter, die Anwender dieser Software sein sollen, zumeist eine Qualifikation im Bereich der Produktion oder der Betriebswirtschaft, selten der Mathematik und Informatik. Zum anderen erfordert der fortlaufende Wechsel der Verantwortlichkeiten, dass die Komplexität der Software möglichst gering und die daraus folgende Einarbeitungszeit möglichst kurz sein soll.

Ausschluss von Software zur Wertstromaufnahme

In Abschnitt 2.2 wurde der Begriff des Wertstroms eingeführt. Wie an dieser Stelle erläutert, ist es das Ziel des Wertstrommanagements, den Materialfluss und den Informationsfluss durch die Aufnahme, das Design und die Planung von Wertströmen auszulegen und zu verbessern (siehe u. a. Erlach 2010, Rother und Shook 1999, Singh et al. 2011). Bei jedem dieser Schritte besteht eine wesentliche Aufgabe des Planers darin, Zeichnungen der aktuellen und der geplanten Wertströme anzulegen. Dabei kann er auf Programme zur Erzeugung von Vektorgrafiken zurückgreifen, die geeignete Symbole zur Verfügung stellen, und auf Symbolbibliotheken, die solche Programme erweitern. Der Zweck von Software dieser Art ist die grafische Erfassung von Wertströmen und die Analyse, welche der ausgeführten Tätigkeiten nicht wertschöpfend sind. Darüber hinaus ermöglicht es solche Software, geplante

Name	Informationen*
Bluespring LeanView 6.1[b]	http://www.bluespring.com/value-stream-mapping
BreezeTree FlowBreeze 4.3[a]	http://www.breezetree.com/flowcharting-software/
ConceptDraw PRO 10, VSM Solution	http://www.conceptdraw.com/solution-park/business-↵ value-stream-mapping/
Creately, VSM Tool	http://creately.com/value-stream-mapping-tool
Edraw Max 8.4, VSM Solution	http://www.edrawsoft.com/valuestreammapping.php
eVSM 9[b]	http://evsm.com/
iFakt VSM	http://www.ifakt.de/software/vsm/
iGrafx FlowCharter	http://www.igrafx.com/gl/products/process-modeling-↵ analysis/flowcharter
Lean Enterprise Institute, VSM Icons[a]	http://www.lean.org/common/display/?o=866
Leanpilot 1.8.5	http://www.leanpilot.com/
Lucidchart, VSM Shapes and Templates	http://www.lucidchart.com/pages/examples/value-↵ stream-mapping
Paul Herber, VSM Shapes Stencil and Template[b]	http://www.paulherber.co.uk/visio-value-stream-map/
QI Macros, VSM Template[a]	http://www.qimacros.com/quality-tools/value-stream-↵ map/
SmartDraw 2016, Lean Diagrams	http://www.smartdraw.com/value-stream-map/
Systems2win, VSM Template[a]	http://www.systems2win.com/solutions/value-stream-↵ mapping-template.htm

[a,b]Symbolbibliothek, erweitert (a) Microsoft Excel bzw. (b) Microsoft Visio *abgerufen am 4. Januar 2017

Tabelle 3.1: Software zur grafischen Wertstrommodellierung. Es existiert eine Vielzahl von frei verfügbaren und kommerziellen Softwarelösungen, die genutzt werden können, um Wertströme grafisch zu erfassen. Oft erweitern sie Standardsoftware wie z. B. Microsoft Excel und Visio. Zwar erleichtert solche Software die Beschreibung von Wertströmen, jedoch ist es i. d. R. nicht möglich, die Beziehungen zwischen den Elementen eines gegebenen Systems vollständig abzubilden.

Wertströme zu skizzieren, um aufzuzeigen, wie nicht wertschöpfende Tätigkeiten zukünftig reduziert werden können. Der Planer verknüpft die bereitgestellten Symbole, sodass diese in ihrer Kombination die zuvor genannten Informationen widerspiegeln. Tabelle 3.1 bietet einen Überblick zu entsprechenden Softwarelösungen.

Zur Veranschaulichung zeigt Abbildung 3.1 das exemplarische Ergebnis einer Wertstromaufnahme. Verglichen mit einer Erstellung von freier Hand erleichtert die Anordnung und Verknüpfung von Symbolen die grafische Modellierung eines Systems. Für die Planung, d. h. für die Optimierung des erstellten Modells gemäß den Planungszielen, ist solche Software allerdings nicht geeignet, wie im Folgenden begründet wird.

Kritisch ist festzustellen, dass die Symbole in vielen Fällen Informationen repräsentieren, die ggf. in der Phase der operativen Planung, aber nicht in der Phase der strategischen Planung bekannt sind. Als Beispiele seien die Reihenfolge der Aufträge zwischen den Arbeitsplätzen (z. B. FIFO oder Push), die Verwaltung der Bestände von halbfertigen und fertig bearbeiteten Produkten (z. B. Puffer und Lager) und die Zuführung von Komponenten (z. B. durch einen Milkrun) genannt. Da aber die Symbole zur Abbildung solcher Informationen nicht zwingend verwendet werden müssen, ist dies noch nicht hinreichend, um den Softwaretyp auszuschließen. Auch dass sich entsprechende Software typischerweise einzig dafür eignet, ein System grafisch zu modellieren, aber nicht, die Auslastung der MAEs zu

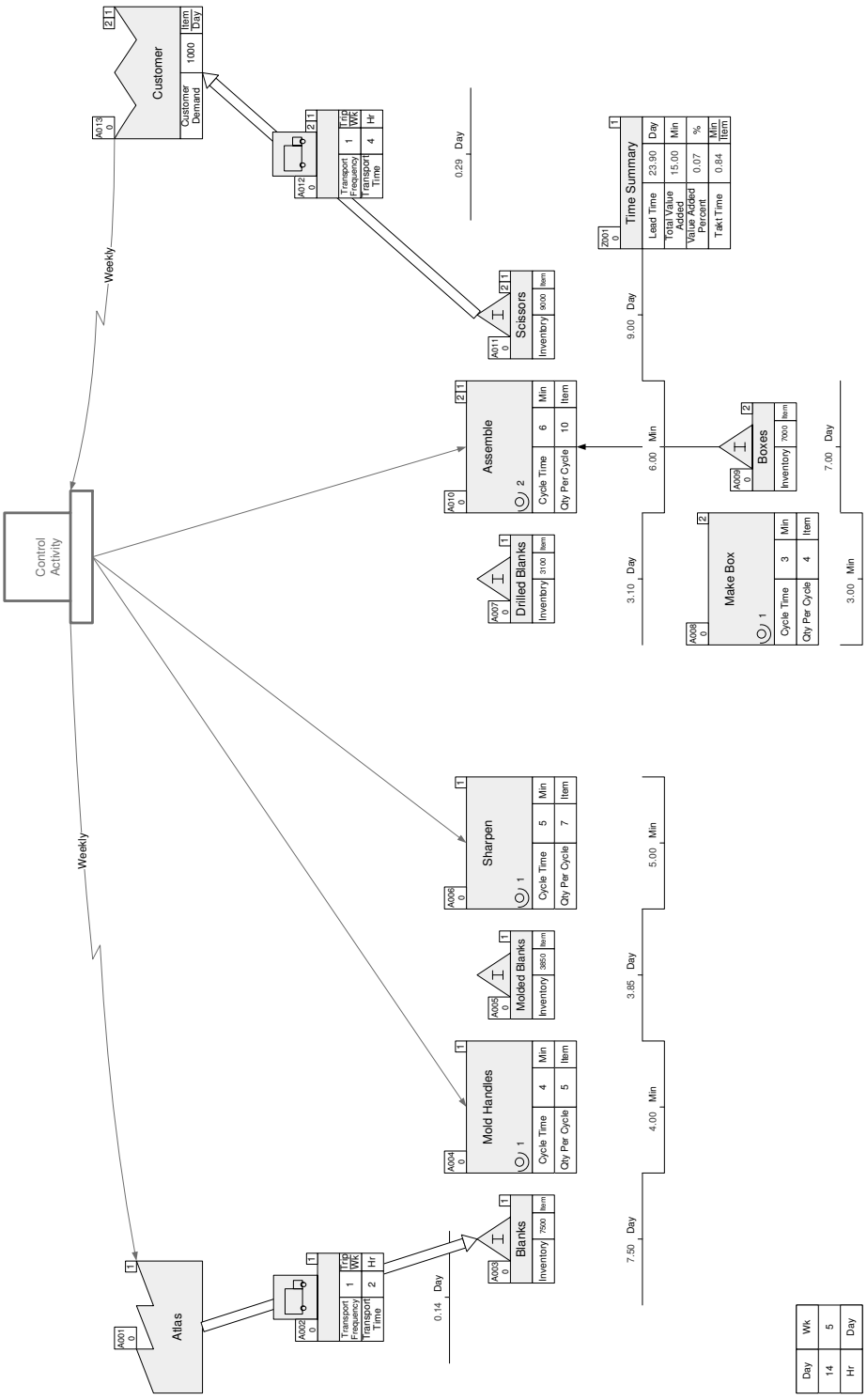


Abbildung 3.1: Ergebnis einer Wertstromaufnahme (Beispiel). Die Symbole beschreiben einen Wertstrom in Bezug auf den Materialfluss (Transport, Prozessschritte und Bestände von Komponenten sowie halbfertigen und fertig bearbeiteten Produkten) und den Informationsfluss (elektronische Datenübermittlung und Steueranweisungen). Das vorliegende Beispiel wurde mit der Software eVSM in der Version 7 erstellt (Quelle und Erläuterung siehe eVSM 2015).

kalkulieren, ist noch kein Ausschlussgrund. Sofern ein grafisches Modell alle notwendigen Informationen bereitstellt, ist es denkbar, einen Interpreter einzusetzen, welcher das grafische Modell in ein analytisches oder Simulationsmodell umwandelt. Auf dieser Grundlage kann anschließend die Auslastung der MAEs ermittelt werden.

Es existieren jedoch zwei Defizite, die im Allgemeinen verhindern, Software zur Wertstromaufnahme für die strategische Planung der TEK einzusetzen. Zum einen ist die Verkettung von alternativen und selektiven Verknüpfungen mit den gebotenen, einfachen Symbolen nur begrenzt darstellbar (siehe Beispiel SL2 in Abschnitt 2.5). Zum anderen kann mit Hilfe solcher Symbole nicht der Anwendungsfall beschrieben werden, dass eine MAE für unterschiedliche Prozessschritte eines Produkts genutzt wird (siehe Beispiel AL2 und AL3 im genannten Abschnitt). Als Konsequenz ist es nicht möglich, Prozessschritte zu erfassen, denen zwar dieselbe MAE zugeordnet ist, die sich aber bezüglich Taktzeiten und/oder nachfolgender Prozessschritte unterscheiden (siehe Beschreibung der zwei Anwendungsfälle im Zusammenhang mit Basisanforderung A2). Allgemein ist deshalb festzuhalten, dass die Beziehungen zwischen den Elementen eines Systems nicht vollständig abgebildet werden können. Die Basisanforderungen A1–A6 sind somit nicht erfüllt, weshalb Software zur Wertstromaufnahme für die Bewertung keine Berücksichtigung findet.

Bemerkung: Damit ist Software ausgeschlossen, die sich auf das Zeichnen von Wertströmen mit Hilfe einfacher Symbole beschränkt. Software, die sich zur Modellierung komplexer Prozesse unter Berücksichtigung aller Beziehungen zwischen den Elementen eines Systems eignet, wird im Weiteren berücksichtigt.

Ausschluss algebraischer Modellierungssprachen

Software zur Wertstromaufnahme stehen grundsätzlich sogenannte algebraische Modellierungssprachen (AMLs) gegenüber, welche die Formulierung und Lösung mathematischer Optimierungsprobleme erlauben. Statt ausführlicher Erläuterungen soll aufgrund ihrer Verbreitung in der Disziplin der Operations Research und der Verfügbarkeit vieler Fallbeispiele der Verweis auf die Literatur genügen (siehe u. a. Kallrath 2004, 2012). Einige der wichtigsten Vertreter sind mitsamt Quellen in Tabelle 3.2 genannt.

AMLs zeichnen sich dadurch aus, dass sie es gestatten, einmal erstellte Modelle schnell und flexibel umzuformulieren (siehe Kallrath und Maindl 2006, S. 39). Zudem sind sie zur Lösung von Problemen mit einer großen Zahl von Variablen und Bedingungen geeignet. Ihre Syntax orientiert sich dabei an der allgemein üblichen mathematischen Notation (Mengen, Ausdrücke usw. usf.), weshalb sie eine Qualifikation im Fachgebiet der mathematischen Modellierung voraussetzen. Sie verlangen vom Anwender, das Problem von der Praxis in eine abstrakte, algebraische Form zu übersetzen. Für außergewöhnliche Aufgaben von größerer Tragweite wie die Suche nach einem neuen Produktionsstandort kann die Berufung eines Kreises qualifizierter Experten zweckmäßig sein. Wie am Beispiel der Bosch Rexroth AG erläutert, ist die strategische Planung der TEK jedoch eine wiederholte, auf eine große Zahl von Mitarbeitern aller Hierarchieebenen verteilte Aufgabe. Die erforderliche Qualifikation kann aus diesem Grund im Allgemeinen nicht vorausgesetzt werden. Damit kommen AMLs für eine Bewertung im Rahmen dieses Kapitels, ungeachtet der Anerkennung ihrer zweifellos gegebenen Leistungsfähigkeit, nicht in Betracht.

Name	Quellen und weiterführende Informationen
AIMMS	Bisschop und Roelofs (2004), Roelofs und Bisschop (2016a,b)
AMPL	Fourer et al. (1990, 2003, 2004), Fourer und Gay (2003)
APMonitor	Hedengren (2008), Hedengren et al. (2014), Hedengren und Eaton (2015)
ASCEND	Piela et al. (1991, 1992), Krishnan et al. (1993), Bhargava et al. (1994)
GAMS	Meeraus (1976), Brooke et al. (1988), Bussieck und Meeraus (2004), GAMS (2016)
GNU MathProg	Eleyat et al. (2011), Pryor und Chinneck (2011), Makhorin (2016a,b)
LINGO	Schrage (1986, 2006), Cunningham und Schrage (2004), LINDO (2016)
LPL	Hürlimann und Kohlas (1988), Hürlimann (1999, 2004, 2016)
MINOPT	Rojnuckarin und Floudas (1994), Schweiger und Floudas (1998a,b, 2004)
Mosel	Colombani und Heipcke (2002), Colombani et al. (2004), Guéret et al. (2002), Heipcke (2012)
MPL	Maximal Software (2002, 2017), Kristjansson und Lee (2004)
MPSX/370 und OSL	Slate und Spielberg (1978), Wilson und Rudin (1992), Spielberg (2004)
NOP-2	Neumaier (1997), Schichl et al. (2001), Schichl und Neumaier (2004)
OMNI	Haverly Systems (1976), Kuip (1993), Haverly (2001, 2004)
OPL	Van Hentenryck et al. (1999, 2004), IBM (2016a, 2016b)
PCOMP	Dobmann et al. (1995), Liepelt und Schittkowski (2000), Schittkowski (2004)
Pyomo	Hart et al. (2011, 2012), Mason (2013), DeCarolis et al. (2010)
SAMPL	Valente et al. (2001, 2008, 2009), Valente (2011)
TOMLAB	Holmström (1999, 2008), Holmström und Edvall (2004), Holmström et al. (2010)

Tabelle 3.2: Liste in der Praxis verwendeter algebraischer Modellierungssprachen. Im akademischen Umfeld der Operations Research und zunehmend auch von firmeneigenen Forschungsabteilungen werden AMLs eingesetzt, um mathematische Optimierungsprobleme zu lösen. Hierzu existiert eine große Zahl verschiedener Sprachen und Softwareumgebungen, die sich in Bezug auf die gebotenen Möglichkeiten zur Modellierung und Optimierung unterscheiden.

Bewertung von vier Softwaretypen

Software zur grafischen, aber unvollständigen Modellierung und Sprachen zur abstrakten, algebraischen Modellierung müssen somit ausgeschlossen werden. Unter Berücksichtigung der zwei festgelegten Kriterien, der Möglichkeit zur vollständigen Abbildung eines gegebenen Systems und zum produktiven Einsatz in Unternehmen, verbleiben die folgenden vier Typen verfügbarer und für die planerische Praxis relevanter Software:

(1) *Software zur Erstellung von Tabellenkalkulationen* (Bsp. Microsoft Excel): in Unternehmen weithin eingesetzte Software zur Strukturierung und Lösung mathematischer Probleme mit begrenzten Möglichkeiten der Optimierung;

(2) *Software zur Materialflusssimulation* (Bsp. Siemens Plant Simulation): leistungsfähige, kommerzielle Software zur stochastischen Modellierung, diskreten ereignisorientierten Simulation und Optimierung von Prozessen in Produktion und Logistik;

(3) *Software für Supply Chain Management* (Bsp. SAP APO SNP): betriebswirtschaftliche Standardsoftware zur Modellierung und mathematischen Optimierung von Problemen der Ressourcenbelegung in Logistiknetzwerken;

(4) *Software zur Prozessmodellierung* (Bsp. BPMN): allgemein verfügbare Software auf Basis grafischer Standards zur Abbildung von Geschäftsprozessen, bewertet in Kombination mit einem hypothetischen idealen Interpreter und Optimierer.

Index	Anforderung	Tabellen-kalkulationen (Microsoft Excel)	Materialfluss-simulation (Siemens Plant Simulation)	Supply Chain Management (SAP APO SNP)	Prozess-modellierung (BPMN)
A1	Superposition aller Wertströme	+	+	+	+
A2	Systemstruktur	+	+	+	+
A3	Systemschnittstelle	+	+	+	+*
A4	Sequenzielle Verknüpfung	+	+	+	+
A5	Alternative Verknüpfung	+	+	+	+
A6	Selektive Verknüpfung	+	+	+	+
A7+	**Grafische Modellierung**	−	±	−	±
A8+	**Minimale Symbolmenge**	−	−	−	−
A9+	**Modelltransformation**	−	+	−	±*
A10	Maximierung der Kapazitäten	−	−	±	±*
A11	Minimierung der Investitionen	−	−	±	±*
A12	Optimierung der Auslastung	−	−	±	±*
A13	Globales Optimum	−	−	+	±*
A14+	**Lineare obere Laufzeitschranke**	±	−	+	±*

− nicht erfüllt ± teilweise erfüllt + erfüllt *unter Annahme eines hypothetischen idealen Interpreters und Optimierers

Tabelle 3.3: Bewertung ausgewählter Beispiele von Softwaretypen. Aus der Übersicht geht hervor, dass keines der Beispiele allen definierten Anforderungen genügt. Einzeln betrachtet erfüllt kein Softwaretyp in vollem Umfang die Leistungsanforderungen A7+ und A8+ zur grafischen Modellierung und die Basisanforderungen A10–A12 zur mathematischen Optimierung.

Die Aufzählung kann keinen Anspruch auf Vollständigkeit erheben. Wie eingangs dargelegt ist das Ziel vielmehr, eine möglichst umfassende Bewertung der verfügbaren Software im Rahmen der Arbeit zu erreichen. Eine Abgrenzung der vier Typen ist zudem nicht trennscharf möglich. Dies betrifft u. a. Software wie Siemens Plant Simulation, die zur Simulation des Materialflusses vom Zulieferer bis zum Kunden geeignet ist und damit zur Lösung von Fragen des Supply Chain Management beiträgt. Im Folgenden werden die Beispiele stellvertretend für den Stand der Technik in Bezug auf die Anforderungen aus Abschnitt 2.6 evaluiert. Tabelle 3.3 fasst die Ergebnisse zusammen.

3.2 Software zur Erstellung von Tabellenkalkulationen

Software zur Erstellung von Tabellenkalkulationen unterstützt die tabellarische Eingabe, die formelbasierte Verarbeitung sowie die tabellarische und die grafische Auswertung von numerischen Daten. Die Struktur der Daten ist dabei durch Zellen in Zeilen und Spalten vorgegeben. Mit Hilfe sogenannter Zellbezüge ist es möglich, Werte mit Formelvariablen in mathematischen Funktionen zu verknüpfen. Nach einer Änderung der Daten in verknüpften Zellen werden die Ergebnisse in verknüpfenden Zellen automatisch aktualisiert. Durch verschachtelte Verknüpfung von Eingaben zu Zwischenergebnissen sind umfangreiche Berechnungen möglich (siehe u. a. Bourg 2009, Guerrero 2010, Winston 2011).

Entsprechende Software ist in größeren Unternehmen, insbesondere in administrativen Funktionen, für alle Mitarbeiter allgemein verfügbar. Fundierte Kenntnisse im Umgang mit

der Software stellen oft eine Einstellungsvoraussetzung dar. Als Folge waren Tabellenkalkulationen vor der Einführung von AURELIE bei der Bosch Rexroth AG für die strategische Planung der TEK der am häufigsten anzutreffende Softwaretyp.

3.2.1 Beispiel: Microsoft Excel

Die Bewertung erfolgt am Beispiel des Industriestandards Microsoft Excel. Vor dem Hintergrund der ubiquitären Verbreitung in Unternehmen und der Tatsache, dass die eingangs formulierten, allgemeinen Aussagen alle Informationen für eine Bewertung bereitstellen, wird für Details auf die genannten Quellen verwiesen. Aktuell ist Microsoft Excel als Bestandteil des Pakets Microsoft Office 2016 verfügbar.[1]

Die Modellierung eines gegebenen Systems von Wertströmen erfolgt auf Basis fix definierter Prämissen, insbesondere der Zuordnung der Produktionsanlagen zu Prozessschritten/Prozessen und Produkten. Für die Abbildung dieser Beziehungen stehen dem Anwender Formeln und Zellbezüge zur Verfügung. Deren zielgerichtete Nutzung erfordert zum einen, das System in seiner Komplexität vollständig zu erfassen, und zum anderen, über das mathematische Verständnis zu verfügen, die Struktur des Systems mitsamt den Verknüpfungen aller Prozessschritte in eine Tabellenkalkulation zu überführen.

3.2.2 Erfüllungsgrad der Anforderungen

Erfüllungsgrad der Basisanforderungen A1–A6
Superposition aller Wertströme, Systemstruktur, Systemschnittstelle sowie sequenzielle, alternative und selektive Verknüpfung

Grundsätzlich ist es möglich, ein System, das aus Wertströmen verschiedener Produkte zusammengesetzt ist (A1), durch eine Tabellenkalkulation abzubilden. Es können Formeln und Zellbezüge genutzt werden, um die Systemstruktur (A2) und die Systemschnittstelle (A3) unter Berücksichtigung aller sequenziellen, alternativen und selektiven Verknüpfungen zu modellieren (A4–A6). Die Basisanforderungen A1–A6 sind somit *erfüllt*.

In der Praxis ist die Vollständigkeit eines solchen Modells allerdings vom fachlichen Hintergrund und Zeitaufwand des Erstellers abhängig. Bei der Bosch Rexroth AG reichte das Spektrum von einfachen Tabellenkalkulationen, die Produktionsanlagen eines Typs zusammenfassen (z. B. einer Technologie wie Fräsen oder Drehen), zu ungleich aufwändigeren Modellen, die einen Wertstrom als verkettetes Bediensystem beschreiben, um die Bestände durch die mittlere Warteschlangenlänge zu approximieren.

Erfüllungsgrad der Leistungsanforderungen $A7^+$–$A9^+$
Grafische Modellierung, minimale Symbolmenge und Modelltransformation

Die Modellierung eines gegebenen Systems ist mit einer Tabellenkalkulation naturgemäß tabellarischer, nicht grafischer Natur. Entsprechend existiert keine minimale Symbolmenge,

[1] Microsoft Office: Excel. Abgerufen am 22. Januar 2018. http://office.microsoft.com/de-de/excel/.

und die Umwandlung des grafischen Modells in ein analytisches oder Simulationsmodell entfällt. Die Leistungsanforderungen A7$^+$–A9$^+$ sind folglich *nicht erfüllt*.

Erfüllungsgrad der Basisanforderungen A10–A13
Maximierung der Kapazitäten, Minimierung der Investitionen, Optimierung der Auslastung und globales Optimum

Mit der Zielwertsuche steht in Microsoft Excel ein Werkzeug zur Verfügung, um einfache Probleme abhängig von einer veränderlichen Zelle/Variable zu lösen. Die Software ist durch einen Solver erweiterbar, welcher die Lösung von linearen und nichtlinearen Problemen mit bis zu 200 veränderlichen Zellen/Variablen erlaubt.[2] In der Praxis erfordert die Abbildung von Systemen jedoch z. T. mehr als 1000 Prozessstückzahlen, um die Verteilung der Produktstückzahlen auf alternativ verknüpfte Prozessschritte zu beschreiben. Folglich verhindert die Beschränkung der Variablenzahl im Allgemeinen, das jeweils gegebene System vollständig abzubilden. Die Basisanforderungen A10–A13 sind damit *nicht erfüllt*.

Bei der Bosch Rexroth AG war festzustellen, dass der Solver für die Planung der TEK nicht eingesetzt wurde. Stattdessen legten die Anwender selbst fest, zu welchem fixen Anteil der Produktstückzahl alternativ verknüpfte Prozessschritte durchlaufen werden sollen. Damit war es ihre Verantwortung, eine optimale Verteilung der Produktstückzahlen in Bezug auf Kapazitäten, Investitionen und Auslastung festzulegen.

Erfüllungsgrad der Leistungsanforderung A14$^+$
Lineare obere Laufzeitschranke

Wie zuvor erläutert, ist eine Optimierung gemäß den Planungszielen mit Microsoft Excel im Allgemeinen nicht möglich. Daher kann keine Laufzeit zur Ermittlung der globalen Optima gemessen werden. Statt einer automatisierten Optimierung muss der Anwender fixe Werte für die Freiheitsgrade des Modells festlegen. Da allerdings die Ergebnisse der Kalkulation augenblicklich aktualisiert werden, sobald der Anwender die vorgegebenen Werte verändert, gilt die Leistungsanforderung A14$^+$ zumindest als *teilweise erfüllt*.

Ergebnis der Bewertung

Positiv ist zunächst festzustellen, dass Microsoft Excel die Basisanforderungen A1–A6 erfüllt und zur Modellierung eines Systems von Wertströmen geeignet ist. Des Weiteren ist Software dieses Typs für Mitarbeiter in Planungsfunktionen i. d. R. zugänglich. Von den Anwendern können fundierte Kenntnisse vorausgesetzt werden.

Negativ ist zu bemerken, dass Software zur Erstellung von Tabellenkalkulationen wie Microsoft Excel naturgemäß die tabellarische, aber nicht die grafische Modellierung eines Systems von Wertströmen unterstützt. Die Leistungsanforderungen A7$^+$–A9$^+$ sind dementsprechend nicht erfüllt. Problematisch ist insbesondere, dass die komplexe Logik zur Abbildung von Verknüpfungen zwischen Prozessschritten in Formeln und Zellbezügen versteckt

[2] Microsoft Support: Solver Limits for Constraints and Adjustable Cells. Abgerufen am 21. April 2014. http://support.microsoft.com/kb/75714/en-us.

und nicht geeignet visualisiert wird. In der Anwendung führt dies zu reduzierter Effizienz, Transparenz und Flexibilität. Belege hierfür liefern Studien zu Fehlerrisiken bei der Erstellung von Tabellenkalkulationen (siehe u. a. Powell et al. 2008, 2009). In der Praxis bestimmen daher die Analysefähigkeit, das Verständnis und die Achtsamkeit des Anwenders die Vollständigkeit eines Modells in besonderem Maße. Im Hinblick auf die Optimierung gemäß den definierten Planungszielen erfüllt Microsoft Excel nicht die Basisanforderungen A10–A13 und nur teilweise die Leistungsanforderung A14$^+$. Zusammenfassend ist festzustellen, dass Software zur Erstellung von Tabellenkalkulationen, wie am Beispiel von Microsoft Excel evaluiert, für die Planung in diesem Kontext ungeeignet ist.

3.3 Software zur Materialflusssimulation

Die Simulation eines experimentierbaren Modells bietet immer dann Vorteile, wenn Experimente an einem realen System nicht oder zumindest nicht wirtschaftlich realisiert werden können, wie dies oft in der Produktion und der Logistik der Fall ist. Eine Simulationsstudie verläuft üblicherweise in den folgenden Schritten (siehe Eley 2012, S. 17 f):

(1) Festlegung der Aufgaben und Ziele,
(2) Erhebung erforderlicher Daten,
(3) Implementierung des Simulationsmodells,
(4) Verifikation und Validation des Modells,
(5) Durchführung und Auswertung von Experimenten,
(6) Übertragung der Ergebnisse auf die Realität.

Software zur Simulation des Materialflusses unterstützt in den meisten Fällen mindestens die Schritte (3), (4) und (5), um ein System von Wertströmen einschließlich der Dynamik seiner Prozesse abzubilden und Entscheidungen zur Steigerung seiner Leistung abzuleiten (für eine Übersicht siehe u. a. Kühn 2006, Semini et al. 2006).

Technische Grundlage ist in aller Regel das Modellierungsmuster gemäß der diskreten ereignisorientierten Simulation (DES). Das Verhalten des Systems wird dabei als Folge von Ereignissen abgebildet, die zu bestimmten Zeitpunkten auftreten, den Systemzustand verändern und Folgeereignisse erzeugen. Zwischen zwei aufeinanderfolgenden Ereignissen ist der Zustand des Systems unverändert. Durch die Definition von Zufallsvariablen können stochastische Einflussgrößen berücksichtigt werden (z. B. die Maschinenverfügbarkeit, die Zeit zwischen zwei Aufträgen und die Bearbeitungszeit). Im Vergleich zu analytischen Modellen, welche die Ausgaben des Systems durch geschlossene mathematische Ausdrücke der Eingaben approximieren, müssen weniger vereinfachende Annahmen getroffen werden. Die Grundlagen der DES, insbesondere der Modellierung und Analyse, sind in der Literatur beschrieben (siehe u. a. Banks et al. 2010, Fishman 2001, Law 2014).

Zur Optimierung eines Simulationsmodells ist im Gegensatz zu analytischen Modellen kein exaktes, mathematisches Verfahren anwendbar. Der Grund ist, dass kein geschlossener Ausdruck für die stochastische, zu optimierende Zielfunktion vorliegt. Den Eingaben werden scheinbar zufällige Ausgaben zugeordnet, abhängig von der Realisierung der Zufallsvariablen im jeweiligen Simulationslauf. Stattdessen muss ein geeignetes Verfahren der

simulationsbasierten Optimierung (SO) genutzt werden, wofür zwei verschiedene Ansätze existieren: die nichtrekursive und die rekursive SO (siehe Pflug 1996, S. 9 ff).

Im Fall der nichtrekursiven SO wird im ersten Schritt eine deterministische Zielfunktion auf Basis einer großen Zahl von Simulationsläufen approximiert. Im zweiten Schritt kann diese mit Hilfe eines exakten Verfahrens optimiert werden. Allerdings steigt die nötige Zahl der Simulationsläufe, um die Zielfunktion genau genug zu approximieren, für komplexe Probleme schnell und ist schwer vorauszusagen. Der Ansatz wurde unter dem Begriff der retrospektiven SO eingeführt (siehe Healy und Schruben 1991).

Gemäß der rekursiven SO wird das Simulationsmodell an eine Heuristik oder Metaheuristik wie z. B. einen genetischen Algorithmus (GA) oder einen Partikelschwarmoptimierer (PSO) gekoppelt. In jedem Rekursionsschritt werden die Ergebnisse der Simulationsläufe des vorigen Rekursionsschritts bewertet, um neue Lösungen zu suchen und zu deren Bewertung Simulationsläufe zu starten. Die Rekursion endet, sobald ein zuvor festgelegtes Abbruchkriterium erfüllt ist. Dieser Ansatz, der auch als prospektive SO bezeichnet wird, ist insbesondere für komplexe Probleme geeignet. Vorteilhaft ist, dass i. d. R. eine gute Lösung in akzeptabler Zeit gefunden wird, wenn auch nicht das globale Optimum.

Die grundsätzlichen Verfahren wurden bereits vor längerer Zeit eingeführt und profitieren von der rasant wachsenden Leistung moderner Rechnersysteme. Für weitere Informationen wird auf die Literatur verwiesen (siehe u. a. Andradóttir 1998, Carson und Maria 1997, Fu 2002, Fu et al. 2005, Ólafsson und Kim 2002, Köchel 2009).

3.3.1 Beispiel: Siemens Plant Simulation

Software zur Materialflusssimulation wird am Beispiel von Siemens Plant Simulation bewertet, einer leistungsfähigen, in der Industrie weit verbreiteten Software zur DES von Prozessen in Produktion und Logistik. Sie ermöglicht u. a. die Simulation eines Modells in verschiedenen Szenarien, die Visualisierung des Verlaufs und der Ergebnisse der Simulation, die Analyse des Zusammenhangs zwischen den Eingaben und Ausgaben eines Modells mit Hilfe eines künstlichen neuronalen Netzes und die Optimierung eines Modells durch einen GA. Als Folge ihrer Verbreitung ist in der Literatur eine Vielzahl von Anwendungen, Fallbeispielen und Musterlösungen zu finden (siehe u. a. Bangsow 2010, Eley 2012).

Falls Einflussgrößen eines Systems unbekannt sind, kann der Anwender diese, wie zuvor beschrieben, durch Zufallsvariablen ersetzen (siehe Eley 2012, S. 18 ff). Um optimale Werte für die Informationseingaben eines Systems zu suchen, kann das Modell an Heuristiken oder einen GA gekoppelt werden. Ein Beispiel ist die Suche nach einer Auftragsreihenfolge, welche die Rüstzeiten minimiert (siehe Eley 2012, S. 271 ff). Für die folgende Bewertung wird stellvertretend der Einsatz eines GA untersucht, um das erstellte Modell rekursiv simulationsbasiert zu optimieren (siehe oben stehende Erläuterung).

Siemens Plant Simulation bietet dem Anwender eine Bibliothek erweiterbarer Bausteine an, um auf grafische Weise Simulationsmodelle zu erstellen. Dabei handelt es sich um Klassen im Sinne der objektorientierten Programmierung, von denen verschiedene Objekte erzeugt werden können, um abhängig vom gegebenen Anwendungsfall bestimmte Eigenschaften der übergeordneten Klasse zu übernehmen oder zu überschreiben. Eine Kategorisierung dieser Bausteine lautet wie folgt (siehe Eley 2012, S. 35 ff):

(1) bewegliche Elemente (im Folgenden abgekürzt als BEs),
(2) Bausteine des Materialflusses,
(3) Bausteine des Informationsflusses,
(4) Ressourcen,
(5) Bausteine der Oberfläche.

BEs bilden Aufträge und andere physische Objekte ab (wie z. B. Werkstücke, Transportbehälter und Fahrzeuge). Sie werden i. d. R. durch eine Quelle erzeugt, im Materialfluss bewegt und durch eine Senke wieder vernichtet (siehe Eley 2012, S. 35).

Bausteine des Materialflusses dienen der Abbildung von Maschinen (z. B. Einzelstationen und Parallelstationen), Lagern sowie von Fördertechnik zum Zweck des physischen Transports (z. B. Wege und Förderstrecken). Darüber hinaus zählen zu dieser Klasse Quellen und Senken, die wie zuvor beschrieben den Beginn bzw. das Ende des Materialflusses repräsentieren. Logische Verknüpfungen (z. B. Kanten und Flusssteuerungen) verbinden die Bausteine miteinander (siehe Eley 2012, S. 35 f).

Bausteine des Informationsflusses schließen u. a. Variablen, Datenstrukturen (z. B. Tabellen, Listen und Warteschlangen), Methoden und Schnittstellen ein. Variablen bezeichnen Daten wie z. B. Zahlenwerte oder Zeichenketten, wogegen komplexere Daten in Datenstrukturen organisiert werden. Durch die Programmierung von Methoden mit SimTalk, einer Skriptsprache, kann der Anwender die Logik zur Steuerung des Informationsflusses entsprechend dem gegebenen System anpassen. Schnittstellen ermöglichen den Datenaustausch mit externen Programmen und Datenquellen (siehe Eley 2012, S. 36).

Ressourcen sind im Verständnis von Siemens Plant Simulation Bausteine, die von anderen Bausteinen eines Modells wie z. B. Maschinen genutzt werden. Zu Beispielen gehören Arbeitsplätze, Gruppen von Mitarbeitern (sogenannte Werkerpools) und der Schichtkalender zur Vorgabe des Schichtmodells (siehe Eley 2012, S. 36).

Bausteine der Oberfläche visualisieren schließlich den Zustand des Simulationsmodells zu einem gegebenen Zeitpunkt (z. B. Diagramme und Displays) und erlauben es dem Anwender, mittels Eingaben den weiteren Verlauf der Simulation zu steuern (z. B. über Dialoge, siehe Eley 2012, S. 37). Im Folgenden soll bewertet werden, wie die Bausteine von Siemens Plant Simulation eingesetzt werden können, um ein System von Wertströmen zu modellieren und gemäß den Planungszielen zu optimieren.

3.3.2 Erfüllungsgrad der Anforderungen

Erfüllungsgrad der Basisanforderungen A1–A6
Superposition aller Wertströme, Systemstruktur, Systemschnittstelle sowie sequenzielle, alternative und selektive Verknüpfung

Siemens Plant Simulation bietet die Möglichkeit, ein System aus Wertströmen verschiedener Produkte zu modellieren, zu simulieren und zu optimieren (A1). Mit den Grundfunktionen der Software und deren Erweiterung durch eigene Methoden stehen Werkzeuge zur Verfügung, um die Systemstruktur auf geeignete Weise durch ein Modell abzubilden (A2). Die Auslastung genutzter Maschinen kann im langfristigen Mittel durch Simulation bestimmt

werden (A3). Falls verschiedene Maschinen nacheinander oder wahlweise für die Bearbeitung eines Produkts genutzt werden sollen, verbindet der Anwender diese durch Kanten bzw. Flusssteuerungen. Dadurch sind sequenzielle, alternative und selektive Verknüpfungen von Prozessschritten abbildbar (A4–A6). Weiterführende Informationen zur Modellierung mit Siemens Plant Simulation sind in der eingangs genannten Literatur detailliert beschrieben. Die Basisanforderungen A1–A6 sind, um dies zusammenzufassen, *erfüllt*.

Erfüllungsgrad der Leistungsanforderung A7$^+$
Grafische Modellierung

Um ein System von Wertströmen mit Siemens Plant Simulation durch ein Simulationsmodell abzubilden, erzeugt der Anwender im ersten Schritt eine Reihe grafischer Bausteine des Materialflusses. Hierzu zählen insbesondere Quellen, Senken, zumeist mehrere Maschinen und ggf. Flusssteuerungen. Die Bausteine verbindet er durch Kanten, die als Pfeile dargestellt werden und logische Verknüpfungen zur Definition des Weges der Produkte von Maschine zu Maschine repräsentieren. Um komplexere Beziehungen zwischen den Elementen des Systems (z. B. alternative Verknüpfungen von Prozessschritten) abzubilden, muss der Anwender im zweiten Schritt das Modell um weitere, nicht grafische Bausteine des Informationsflusses erweitern (z. B. um Methoden und Variablen). Da somit nicht die Gesamtheit der notwendigen Informationen des Systems grafisch modelliert werden kann, ist die Leistungsanforderung A7$^+$ nur *teilweise erfüllt*.

Im Folgenden wird das genannte Beispiel, die Abbildung einer alternativen Verknüpfung von Prozessschritten, näher erläutert. Der Anwender muss zunächst einen Baustein für eine Flusssteuerung erzeugen und diesen an der Stelle zwischen den Maschinen durch Kanten einbinden, an der sich der weitere Weg der BEs unterscheidet. Entweder vereinfacht er das System, indem er eine fixe oder regelbasierte Zuordnung der BEs zu den nachfolgenden Maschinen voraussetzt, oder er erweitert die Flusssteuerung durch die Programmierung einer Methode mit SimTalk. In diesem Fall muss der Anwender Variablen für alle verknüpften Prozessschritte definieren, welche die Prozessstückzahlen, d. h. die Informationseingaben des Modells, bezeichnen. Mit Bezug auf diese Variablen muss er eine Methode programmieren, welche die BEs den verfügbaren Maschinen zuordnet, sodass deren Verteilung im Mittel den vorgegebenen Werten der Variablen entspricht. Danach kann das Modell an einen GA gekoppelt werden, um optimale Werte für die Variablen zu suchen.

Erfüllungsgrad der Leistungsanforderung A8$^+$
Minimale Symbolmenge

Die Software eignet sich v. a. zur detaillierten Beschreibung und Analyse von realen, komplexen Systemen. Der Grund hierfür ist, dass bei der Erstellung eines Simulationsmodells gegenüber einem analytischen Modell weniger vereinfachende Annahmen getroffen werden müssen, um das System zu beschreiben. Statt die Auslastung der Maschinen nur im langfristigen Mittel zu kalkulieren, werden die Bewegungen von jedem BE eines Produkts nachverfolgt. Zudem stehen grafische Bausteine des Materialflusses zur Verfügung, die sehr detaillierte Informationen zum Ablauf von Produktion und Logistik wie z. B. Fördertechnik

und Transportmittel abbilden. Diese Genauigkeit ist in der Phase der operativen Planung, aber nicht in der Phase der strategischen Planung vorteilhaft, da die entsprechenden Informationen zu diesem Zeitpunkt nicht vorliegen. Wie bei Software zur Wertstromaufnahme ist die Verfügbarkeit zusätzlicher Symbole jedoch kein Nachteil, sofern der Anwender nicht zwingend von ihnen Gebrauch machen muss.

Mit Hilfe der grafischen Symbole ist allerdings nicht der Anwendungsfall abbildbar, dass eine MAE für verschiedene Prozessschritte genutzt wird (siehe Abschnitt 2.5, Beispiel AL2, AL3 und SL2, vgl. Software zur Wertstromaufnahme in Abschnitt 3.1). Nach der visuellen Darstellung des Modells werden Prozesse eines Produkts, welche dieselbe MAE durchlaufen, an diesem Punkt vereinigt. Nur mit grafischen Symbolen ist es nicht möglich, Prozessschritte zu definieren, denen dieselbe MAE zugeordnet ist, die sich aber hinsichtlich ihrer Taktzeiten und/oder nachfolgender Prozessschritte unterscheiden. Unter Zuhilfenahme der Begriffe von Siemens Plant Simulation heißt das: Wird ein einzelnes BE im Verlauf der Simulation mit einer Maschine bearbeitet, dann sind sowohl die Taktzeit als auch der weitere Weg des BE stets unabhängig von seinem Weg bis zu diesem Punkt. Um dies zu ändern, verbleibt dem Anwender nur die Programmierung einer Methode mit SimTalk. Mit einer solchen Methode kann er bewirken, dass der Weg eines BE einbezogen wird, um zu entscheiden, mit welcher Taktzeit und mit welchen Maschinen die weitere Bearbeitung erfolgen soll. Da aber die Definition der minimalen Symbolmenge verlangt, diese Unterscheidung mit grafischen Symbolen zu treffen, ist die Leistungsanforderung A8$^+$ *nicht erfüllt.*

Erfüllungsgrad der Leistungsanforderung A9$^+$
Modelltransformation

Das vom Anwender erstellte grafische Modell wird automatisch in ein Simulationsmodell umgewandelt, das wie erläutert zum Zweck der Optimierung an einen GA gekoppelt werden kann. Die Leistungsanforderung A9$^+$ ist somit *erfüllt.*

Erfüllungsgrad der Basisanforderungen A10–A12
Maximierung der Kapazitäten, Minimierung der Investitionen und Optimierung der Auslastung

Um das Modell zu optimieren, muss der Anwender, wie in der Bewertung zur grafischen Modellierung ausgeführt, eigene Methoden und Variablen zur Bezeichnung der Prozessstückzahlen von alternativen Verknüpfungen definieren. Unter dieser Voraussetzung bietet Siemens Plant Simulation die Möglichkeit, das erstellte Modell an einen GA zu koppeln, um die optimalen Werte der Variablen zu suchen. Praktisch sind dem Vorgehen allerdings Grenzen gesetzt. So muss es gemäß der Basisanforderung A1 möglich sein, die Wertströme aller Produkte eines Standorts in einem Modell und, daraus folgend, alle Variablen in einem Suchvektor zusammenzufassen. Dagegen ist die Zahl der Dimensionen des Suchraums, d. h. der Variablen, bei Verwendung eines GA oder eines vergleichbaren Optimierungsverfahrens typischerweise auf nicht mehr als 100 beschränkt. Zudem steigt die Zahl der in einem Optimierungszyklus zu evaluierenden Lösungen, im Fall eines GA Individuen einer Population genannt, mit der Zahl der Variablen, wodurch die Laufzeit schnell wächst. Eine

simultane Suche der Optima aller Variablen ist deshalb nicht realisierbar. Stattdessen muss der Anwender einzelne Wertströme losgelöst voneinander modellieren oder das System vereinfachen, wodurch die tatsächlichen Optima im Allgemeinen nicht gefunden werden. Die Basisanforderungen A10–A12 sind daher *nicht erfüllt*.

Erfüllungsgrad der Basisanforderung A13
Globales Optimum

Die Anwendung von Metaheuristiken wie z. B. GAs oder PSO ist immer dann zweckmäßig, wenn eine exakte Lösung aufgrund der Komplexität des zu lösenden Problems nicht effizient zu ermitteln ist. Sie zeichnen sich dadurch aus, dass sie für anderweitig unlösbare Probleme in akzeptabler Laufzeit i. d. R. eine gute Lösung finden. Bei dieser handelt es sich jedoch zumeist um ein lokales Optimum, nicht um das gesuchte globale Optimum. Als Konsequenz ist die Basisanforderung A13 *nicht erfüllt*.

Erfüllungsgrad der Leistungsanforderung A14$^+$
Lineare obere Laufzeitschranke

Da die globalen Optima gesucht sind, stattdessen aber in vielen Fällen auch nach langer Suche lediglich ein lokales Optimum gefunden wird, ist es nicht möglich, eine Laufzeit zur Erreichung der globalen Optima zu garantieren. Zudem ist die Laufzeit der Simulation und damit der rekursiven SO umso größer, je genauer die Ergebnisse bestimmt werden sollen. Folglich muss die Leistungsanforderung A14$^+$ als *nicht erfüllt* gelten.

Ergebnis der Bewertung

Positiv ist festzuhalten, dass Siemens Plant Simulation grundsätzlich geeignet ist, Produktionssysteme großer Komplexität, die Wertströme verschiedener Produkte zusammenfassen, sehr detailliert abzubilden. Dementsprechend erfüllt die Software auch die Basisanforderungen A1–A6. Kritisch ist anzumerken, dass dieses Maß an Genauigkeit zwar in der Phase der operativen Planung zweckmäßig sein kann, aber in der Phase der strategischen Planung weder notwendig noch umsetzbar ist, da die entsprechenden Informationen nicht verfügbar sind. Die Software bietet zudem die Möglichkeit, ein System grafisch zu modellieren, wenn auch mit den erläuterten Einschränkungen. Das vom Anwender erstellte grafische Modell wird automatisch in ein Simulationsmodell umgewandelt, welches mit einem GA optimiert werden kann, womit die Leistungsanforderung A9$^+$ erfüllt ist.

Negativ ist demgegenüber zu sagen, dass sich die Software nicht dazu eignet, reale Systeme Hunderter, teils Tausender Prozessschritte effizient abzubilden. Der Grund ist zum einen, dass nicht alle Elemente und Beziehungen zwischen den Elementen in einem System grafisch modelliert werden können. Stattdessen erfordert deren Abbildung die Programmierung von Methoden. Die Basisanforderungen A7$^+$ und A8$^+$ sind somit nur teilweise bzw. nicht erfüllt. Infolgedessen gestaltet es sich in der Praxis sehr schwierig, Verknüpfungen von Prozessschritten abzubilden, wie sie im vorigen Kapitel beschrieben wurden. Zum anderen ist es durch Kopplung des Simulationsmodells an eine Metaheuristik wie einen GA im Allgemeinen nicht möglich, alle Freiheitsgrade eines Systems zu optimieren, um die globalen Optima

gemäß den Planungszielen zu finden. Damit sind auch die Basisanforderungen A10–A13 und die Leistungsanforderung A14+ nicht erfüllt. Für die strategische Planung der TEK ist Software zur Materialflusssimulation, im Rahmen dieses Kapitels evaluiert am Beispiel von Siemens Plant Simulation, daher letztlich ungeeignet. Ihre Stärke ist vielmehr die detaillierte, operative Planung von einzelnen Wertströmen bestehender Systeme.

3.4 Software für Supply Chain Management

Der moderne Begriff des Supply Chain Management (SCM) wurde zuerst von Oliver und Webber (1992) verwendet. Er umfasst die Planung und Steuerung sämtlicher Aktivitäten innerhalb der Lieferkette (engl. *Supply Chain*) von Zulieferern zu Kunden. Die Definition des Council of Supply Chain Management Professionals, einer führenden Expertengruppe, lautet entsprechend wie folgt (CSCMP 2015):

> »[...] encompasses the planning and management of all activities involved in sourcing and procurement, conversion, and all Logistics Management activities. Importantly, it also includes coordination and collaboration with channel partners, which can be suppliers, intermediaries, third-party service providers, and customers.«

Es existiert eine Vielzahl von Definitionen, wobei diese im Kern i. d. R. der angegebenen entsprechen (siehe u. a. Gibson et al. 2005). Da die Produktion zunehmend in globalen Netzwerken erfolgt, erfährt SCM immer größere Bedeutung (zur Einführung siehe Blanchard 2010, Chopra und Meindl 2013, Jacoby 2009, Simchi-Levi et al. 2008, für Software zur Unterstützung Stadtler und Kilger 2008 sowie zum Forschungsstand Chen und Paulraj 2004, Halldorsson et al. 2007, Kouvelis et al. 2006, Melo et al. 2009, Storey et al. 2006).

Software für SCM wird, wie schon in Abschnitt 3.1 angesprochen, auch als Überbegriff für solche verwendet, die eine Simulation des Materialflusses in Produktion und Logistik auf technischer Grundlage der DES und SO erlaubt (zur Abgrenzung von Logistik und SCM siehe Larson und Halldorsson 2004). Daneben umfasst sie auch Software, die zur Planung von Beschaffung, Produktion, Logistik und Distribution eingesetzt wird und auf mathematischer Modellierung und Optimierung basiert. Im Gegensatz zum vorigen Softwaretyp existiert dementsprechend kein einzelnes dominierendes Lösungsverfahren. Den zu evaluierenden Softwaretyp kennzeichnet vielmehr seine Anwendung für SCM gemäß obiger Definition. Im Folgenden bezeichnet der Begriff daher solche Software, die eine Planung und Optimierung der Lieferkette auf einer spezifischen Betrachtungsebene ermöglicht, unabhängig davon, welches Lösungsverfahren zum Einsatz kommt.

Zu den wichtigsten Beispielen zählen (siehe Meyr et al. 2008):

(1) AspenTech aspenONE Supply Chain Management[3],
(2) JDA (vormals i2 Technologies) Solutions[4],

[3] AspenTech: Products, aspenONE Supply Chain Management. Abgerufen am 5. Januar 2017. http://www.aspentech.com/products/aspenONE-Supply-Chain-Management/.
[4] JDA: Solutions. Abgerufen am 5. Januar 2017. http://jda.com/solutions.

(3) Oracle JD Edwards EnterpriseOne Supply Chain Planning[5],
(4) SAP Supply Chain Management[6].

3.4.1 Beispiel: SAP APO Supply Network Planning

Die Bewertung des Softwaretyps erfolgt am Beispiel SAP APO (Advanced Planning and Optimization), einer Komponente der kommerziell verfügbaren Software SAP SCM (wie oben Abkürzung für Supply Chain Management) von SAP, dem führenden Anbieter betriebswirtschaftlicher Standardsoftware. SAP APO erweitert ein ERP-System (Enterprise Resource Planning) um Funktionen zur Planung und Optimierung von Beschaffung, Produktion, Logistik und Disposition (für eine Einführung zu SAP APO und Praxisbeispiele siehe u. a. Balla und Layer 2007, Bartsch und Bickenbach 2002, Dickersbach 2009, Kallrath und Maindl 2006, Stadtler und Kilger 2008 sowie Stadtler et al. 2012). Hierbei stehen drei Module zur Verfügung, die sich nach Betrachtungsebene, Einsatzzweck und Lösungsverfahren unterscheiden (siehe Kallrath und Maindl 2006, S. 10 f; alle Informationen in diesem Abschnitt beziehen sich wie ebenda auf SAP APO in der Version 4.1; die Software wurde seither weiterentwickelt, wobei die Aussagen wie unten stehend bemerkt nach wie vor gelten):

(1) *Supply Network Planning (SNP):* mittel- und langfristige Planung von Einkauf, Produktion und Distribution in einem Netzwerk aus mehreren Standorten – unter Verwendung von linearer Programmierung (LP), Mixed-Integer Linear Programming (MILP) und Heuristiken/Constraint Propagation;

(2) *Production Planning and Detailed Scheduling (PP/DS):* kurzfristige Produktions- und Feinplanung an einem einzelnen Standort – unter Verwendung von Heuristiken/Constraint Propagation und evolutionären Algorithmen;

(3) *Transportation Planning and Vehicle Scheduling (TP/VS):* Planung von Transporten für Aufträge und Lieferungen einschließlich Zuordnung von Transportmitteln – unter Verwendung von evolutionären Algorithmen.

Für die Bewertung wird im Folgenden das Modul SNP und dabei insbesondere die Lösung mittels LP und MILP betrachtet. Hierfür existieren zwei Gründe: Erstens liegt der Fokus im Kontext dieser Arbeit auf der strategischen, d. h. der mittel- bis langfristigen Planung. Zweitens stellen in der obigen Übersicht allein LP und MILP exakte, mathematische Optimierungsverfahren dar und garantieren als solche das Finden der globalen Optima gemäß Basisanforderung A13. Anzumerken ist dabei, dass SAP APO SNP die Planung der Produktion in einem Netzwerk unterstützt, dessen Standorte auch Zulieferer und Kunden einschließen können. Die folgende Bewertung beschränkt sich jedoch darauf, inwieweit diese Software für den Zweck geeignet ist, ein System von Wertströmen an einem Produktionsstandort zu modellieren und zu optimieren, wie im vorigen Kapitel dargelegt wurde.

[5] Oracle: JD Edwards EnterpriseOne. Abgerufen am 5. Januar 2017. http://www.oracle.com/us/products/applications/jd-edwards-enterpriseone/.
[6] SAP: Supply Chain Management. Abgerufen am 5. Januar 2017. http://www.sap.com/germany/product/scm.html.

SAP APO SNP ermöglicht die Planung und Optimierung der Beschaffung, Produktion, Distribution, des Transports und der Bestände. In einem Vorverarbeitungsschritt erstellt ein Generator basierend auf Daten aus SAP ERP[7] und/oder SAP APO ein Modell. Diese Daten sind zum einen Stammdaten, insbesondere Standorte, Transportbeziehungen, Produkte, Ressourcen und Prozesse, und zum anderen Bewegungsdaten wie z. B. der Prognosebedarf und Kundenaufträge. Entsprechend der Annahme eines kontinuierlichen Flusses, die auch in Abschnitt 2.4 zugrunde gelegt wurde, beziehen sich die Stückzahlen auf festgelegte Zeitintervalle. Sobald die Optimierung des Modells abgeschlossen ist, werden die Ergebnisse in Bewegungsdaten umgewandelt, welche der Anwender in SAP APO und ggf. in SAP ERP auswerten kann (siehe Kallrath und Maindl 2006, S. 11 f). Datenstrukturen, die zur Erfüllung der Anforderungen genutzt werden müssen, werden im Folgenden eingeführt und in aller Kürze erläutert. Das Ergebnis der Bewertung von SAP APO SNP kann weitgehend auf aktuell am Markt verfügbare SCM-Software übertragen werden.

Bemerkung: Laut Informationen der SAP AG[8] ist PP/DS ab dem zweiten Quartal 2016 ein integraler Bestandteil der neuen Version des ERP-Systems S/4HANA. SNP geht langfristig im Nachfolger SAP IBP auf, der zusätzliche Funktionen wie z. B. Prognoseverfahren bietet. Die in vielen Unternehmen eingesetzte Software SAP APO wird jedoch parallel weiterentwickelt und profitiert von Innovationen wie u. a. der HANA-Datenbank. Für die Bewertung ist dabei wichtig, dass das Datenmodell erhalten bleibt.

3.4.2 Erfüllungsgrad der Anforderungen

Erfüllungsgrad der Basisanforderungen A1–A3
Superposition aller Wertströme, Systemstruktur und Systemschnittstelle

Wie eingangs zur Einführung erläutert, erzeugt ein Generator auf Basis von Stammdaten und Bewegungsdaten aus SAP ERP und/oder SAP APO ein Modell. Das erstellte Modell bildet das gegebene System ab und wird anschließend durch den Einsatz von LP und/oder MILP optimiert. Um die Auslastung der MAEs im Hinblick auf entstehende Kosten zu bewerten oder zu begrenzen, definiert der Anwender Ressourcen. Diese repräsentieren Betriebsmittel, die für Aktivitäten zur Deckung des Kundenbedarfs wie z. B. der Produktion benötigt werden (siehe Kallrath und Maindl 2006, S. 13 f, 59 ff). Mit Blick auf die Erfüllung der Anforderungen erlaubt es damit SAP APO SNP, alle Aktivitäten in der Produktion, d. h. sämtliche Wertströme eines Systems an einem Produktionsstandort, einschließlich aller erforderlichen Ressourcen zur Optimierung in einem Modell zusammenzufassen (A1).

Im Weiteren wird die Abbildung der Systemstruktur und der Systemschnittstelle durch das Modell betrachtet. Der Anwender ordnet den Produkten mit Hilfe noch zu erläuternder Datenstrukturen Ressourcen zu, wobei eine Ressource im Planungszeitraum für mehrere Produkte genutzt werden kann. Um die Auslastung durch Bedingungen zu begrenzen, kann der Anwender für jede Ressource eine Kapazität festlegen. Die Struktur des Systems wird

[7] Die Aussagen gelten für SAP ERP und dessen Nachfolger SAP S/4HANA gleichermaßen. Um die Ausführungen abzukürzen, wird im Folgenden nur von SAP ERP gesprochen.

[8] Vortrag von Bernhard Lokowandt, SAP AG, Product Owner für Materials Planning and Procurement, »Produktionsplanung in SAP S/4HANA«, SAP Education Logistics Event, Walldorf, 8. Oktober 2015.

damit in geeigneter Weise durch das Modell wiedergegeben (A2). Zur Berücksichtigung der Auslastung definiert der Anwender eine Kostenfunktion, die auf Basis vorgegebener Produktstückzahlen ermittelt wird (siehe Kallrath und Maindl 2006, S. 19). Auf dieser Basis minimiert der Optimierer die Kostenfunktion, um zu entscheiden, welche Ressourcen genutzt werden sollen. Die Eingaben des Modells bilden somit die Stückzahlen in jedem Prozessschritt und entsprechen den definierten Eingaben des Informationsflusses an der Systemschnittstelle. Das Gleiche gilt für die Ausgaben, indem das Modell die resultierende Auslastung der MAEs an den Optimierer zurückgibt (A3). Außerhalb der Software sind die Ein- und Ausgaben des Modells, z. B. zur Optimierung mit einem externen Lösungsverfahren, nicht sichtbar (siehe Kallrath und Maindl 2006, S. 39). Für die betrachtete Anwendung bietet das jedoch keinen Anlass zur Kritik, da die Modellierung und die Optimierung innerhalb von SAP APO SNP erfolgen. Die Basisanforderungen A1–A3 sind daher *erfüllt*.

Inwieweit bei der Abbildung der Systemstruktur die verschiedenen Typen zur Verknüpfung von Prozessschritten berücksichtigt werden, wird nun im Anschluss betrachtet.

Erfüllungsgrad der Basisanforderungen A4–A6
Sequenzielle, alternative und selektive Verknüpfung

Das von SAP APO SNP erzeugte Modell basiert auf Datenstrukturen, die den Zweck erfüllen, die Produktstücklisten zu definieren, die Aktivitäten in der Produktion (Prozessschritte) miteinander zu verknüpfen und die erforderlichen Ressourcen (MAEs) zuzuordnen. Bei diesen Datenstrukturen handelt es sich um sogenannte Production Process Models (PPM) oder, ab SAP APO Version 4.1, Production Data Structures (PDS). Im einfachsten Fall definiert der Anwender, dass die Aktivitäten nacheinander ausgeführt werden sollen, gleichbedeutend mit einer sequenziellen Verknüpfung der Prozessschritte (A4). Zur Abbildung alternativer Verknüpfungen kann der Anwender festlegen, dass eine Aktivität für einen bestimmten Anteil der Produktstückzahl auszuführen ist (A5). Darüber hinaus ist es möglich, Zeitanteile zur Ausführung einzelner Aktivitäten zu definieren, um selektive Verknüpfungen abzubilden (A6). Diese Zeitanteile entsprechen fix vorgegebenen Quoten, wobei Taktzeiten bereits berücksichtigt sind (siehe Kallrath und Maindl 2006, S. 14, 51, 61 ff, 109 f, 114). Die Basisanforderungen A4–A6 sind somit, wie schon die vorigen, *erfüllt*.

Erfüllungsgrad der Leistungsanforderungen $A7^+$–$A9^+$
Grafische Modellierung, minimale Symbolmenge und Modelltransformation

Wie oben beschrieben, pflegt der Anwender bestimmte Datenstrukturen, ggf. PPM oder PDS, um die Aktivitäten zur Produktion, d. h. die Prozessschritte in jedem Wertstrom, zu definieren (siehe Kallrath und Maindl 2006, S. 61 ff). Die Eingabe der Daten erfolgt hierbei über Formulare, die erreicht werden können, indem der Anwender bestimmte Transaktionen aufruft. Eine grafische Modellierung der Wertströme bietet die Software dagegen nicht. Folglich existiert, analog zur Bewertung von Microsoft Excel, keine minimale Symbolmenge, und die automatische Umwandlung eines grafischen Modells in ein analytisches oder Simulationsmodell entfällt. Stattdessen erzeugt der Generator auf Basis der Formulareingaben

des Anwenders ein analytisches Modell, das sich für eine Optimierung mittels LP und MILP eignet. Damit sind die Leistungsanforderungen A7$^+$–A9$^+$ *nicht erfüllt.*

Bemerkung: SAP ERP bietet die Möglichkeit, Arbeitspläne, d. h. Prozesse aus verknüpften Prozessschritten einschließlich zugeordneter MAEs, zu visualisieren. Die Erfassung und Überarbeitung von Arbeitsplänen erfolgt jedoch auch weiterhin mit Hilfe von Transaktionen und der formularbasierten Eingabe aller hierzu erforderlichen Daten. Eine grafische Modellierung der Wertströme, wie sie im vorherigen Kapitel beschrieben wurden, unterstützen zum aktuellen Zeitpunkt weder SAP ERP noch der Nachfolger S/4HANA.

Erfüllungsgrad der Basisanforderung A10
Maximierung der Kapazitäten

Voraussetzung für die Optimierung des erstellten Modells mit SAP APO SNP ist, wie angesprochen, die Definition einer Kostenfunktion. Neben der Planung tatsächlicher Kosten eignet sich eine solche Kostenfunktion grundsätzlich dazu, die Erreichung allgemeiner Planungsziele zu bewerten und ggf. widersprüchliche Planungsziele in Beziehung zueinander zu setzen (siehe Kallrath und Maindl 2006, S. 19). Dieses Konzept ist, zumindest theoretisch, auf die Maximierung der verfügbaren Kapazitäten übertragbar. Hierzu ordnet der Anwender jedem Produkt einen Erlöskoeffizienten zu, welcher nicht dem Umsatz durch Verkauf eines Stücks des Produkts entspricht, sondern als Gewichtungsfaktor dient. Auf dieser Basis kann der Solver der Software genutzt werden, um den Gesamterlös zu maximieren, resultierend aus der gewichteten Summe der Produktstückzahlen mit den definierten Erlöskoeffizienten (Anwendungsbeispiel siehe Kallrath und Maindl 2006, S. 198).

Die gefundene Lösung muss allerdings nicht immer eindeutig sein. Das ist insbesondere dann der Fall, wenn Prozessschritten verschiedener Produkte dieselbe MAE zugeordnet ist, wie dies in der Praxis häufig vorkommt. Begrenzt die Auslastung der MAE die als reellwertig vorausgesetzten Stückzahlen der Produkte und ist der Fall gegeben, dass die kumulierten Erlöse für jedes Produkt bei identischer Auslastung der MAE übereinstimmen, dann existieren unendlich viele Lösungen, die zum maximalen Gesamterlös führen (siehe Erläuterung zur Basisanforderung A10 in Abschnitt 2.6). Gemäß einer Lösung ist der kumulierte Erlös für ein beliebiges Produkt ggf. größer als derjenige für ein zweites Produkt, gemäß einer anderen Lösung kann es sich umgekehrt verhalten, wogegen der Gesamterlös einschließlich aller Produkte im Fall beider Lösungen gleich ist.

Um die Anforderungen im Hinblick auf die Optimierung zu erfüllen, muss es möglich sein, zu Beginn eine Strategie festzulegen, welche im beschriebenen, mehrdeutigen Fall die Auswahl einer Lösung bestimmt. Im Folgenden werden diesbezüglich drei Alternativen diskutiert, die unabhängig von SAP APO SNP im Zusammenhang mit der Maximierung oder Minimierung einer Kostenfunktion bestehen. Die Grundannahme ist hierbei, dass es wie im Fall von SAP APO SNP nur möglich ist, einen einzigen Optimierungslauf auszuführen. Ein iteratives Vorgehen, das auf der automatisierten Ausführung aufeinanderfolgender Optimierungsläufe basiert, scheidet aus. Das heißt, die Optimierung der Kostenfunktion muss in einem Schritt zu einer eindeutigen Lösung führen. Die Strategie zur Auswahl der Lösung muss abgebildet werden können, indem der Anwender die Koeffizienten der Kostenfunktion geeignet festlegt und ggf. zusätzliche Nebenbedingungen formuliert.

Alternative A: Definition von Schranken

Der Anwender könnte für die Stückzahl jedes Produkts eine untere Schranke und/oder eine obere Schranke definieren, um die Menge der optimalen Lösungen zu begrenzen. Für jede Schranke könnte er einen fixen Wert oder eine Funktion der Stückzahlen anderer Produkte vorgeben. Im Allgemeinen ist nicht vorauszusetzen, dass innerhalb beliebiger Schranken genau eine Lösung existiert, die gemäß dem Gesamterlös maximal ist. Vielmehr sind die Schranken, welche der Anwender vorgeben müsste, um die optimale Lösung eindeutig zu bestimmen, von den Parametern des Modells abhängig. Folglich müsste der Anwender zunächst durch eine Analyse des Modells oder durch wiederholte Optimierung solche Schranken suchen, die genau eine optimale Lösung begrenzen, um anschließend den Gesamterlös zu maximieren. Das Vorgehen widerspräche der Prämisse, das Ergebnis in einem einzigen Schritt zu ermitteln, weshalb Alternative A ausscheidet.

Alternative B: Festlegung fixer Verhältnisse zwischen Stückzahlen

Werden die Stückzahlen zweier Produkte durch die Auslastung derselben MAE beschränkt, könnte der Anwender für das Verhältnis der Stückzahlen einen fixen Wert festlegen. Bei dieser Alternative handelt es sich um einen Spezialfall der vorherigen, da für jede Stückzahl eine untere und eine entsprechende obere Schranke definiert werden. Der Spezialfall wird explizit genannt, da die Überlegung nahe liegt, die Verhältnisse der geplanten Stückzahlen zur eindeutigen Bestimmung einer optimalen Lösung anzuwenden. Dadurch kann jedoch nicht in jedem Fall eine optimale Lösung gefunden werden, da keine Lösung, welche den Gesamterlös maximiert, diesen Verhältnissen entsprechen muss. Analog zu Alternative A müsste der Anwender schrittweise die Kostenfunktion optimieren, um ein Verhältnis zu ermitteln, mit welchem die Menge optimaler Lösungen auf eine Lösung begrenzt werden kann. Alternative B wird deshalb mit derselben Begründung nicht weiterverfolgt.

Alternative C: Priorisierung durch Erlöskoeffizienten

Denkbar wäre darüber hinaus, dass der Anwender mit der Kostenfunktion eine Priorisierung abbildet und für jedes Produkt einen Erlöskoeffizienten definiert, welcher mit der Priorität steigt. Im Zuge der Maximierung des Gesamterlöses würden als Folge die Stückzahlen all derjenigen Produkte gesteigert, denen eine höhere Priorität zugewiesen wurde. Hierzu müsste der Anwender die Erlöskoeffizienten in einer Weise festlegen, sodass Folgendes gilt: Kann die Stückzahl eines Produkts gesteigert werden, ohne die Stückzahlen von Produkten höherer Priorität zu reduzieren, muss dadurch auch der Gesamterlös wachsen. Dies muss unabhängig davon gelten, in welchem Maße hierfür die Stückzahlen von Produkten niedrigerer Priorität reduziert werden müssen. Im Fall komplexer Modelle ist es jedoch schwierig, geeignete Erlöskoeffizienten zu bestimmen. Des Weiteren ist die resultierende Kostenfunktion aufgrund numerischer Instabilitäten oft nicht lösbar, wie nachfolgend gezeigt wird. Daher ist auch Alternative C nur begrenzt anwendbar.

Da alle der aufgeführten Alternativen verworfen werden oder nicht in jedem Fall anwendbar sind, gilt die Basisanforderung A10 allenfalls als *teilweise erfüllt*.

Produkt	Taktzeit $t_{eff,1/2}$, in min									Erlöskoeffizient, in €/Stück
	MAE 1	MAE 2	MAE 3	MAE 4	MAE 5	MAE 6	MAE 7	MAE 8	MAE 9	
PRD 1	1,0									$1{,}00 = 10^0$
PRD 2	10,0	1,0								$10{,}01 > 10^1$
PRD 3		10,0	1,0							$100{,}11 > 10^2$
PRD 4			10,0	1,0						$1001{,}11 > 10^3$
PRD 5				10,0	1,0					$10\,011{,}11 > 10^4$
PRD 6					10,0	1,0				$100\,111{,}11 > 10^5$
PRD 7						10,0	1,0			$1\,001\,111{,}11 > 10^6$
PRD 8							10,0	1,0		$10\,011\,111{,}11 > 10^7$
PRD 9								10,0	1,0	$100\,111\,111{,}11 > 10^8$
PRD 10									10,0	$1\,001\,111\,111{,}11 > 10^9$

Hinweis: Ist keine Taktzeit angegeben, wird das jeweilige Produkt nicht mit dieser MAE bearbeitet.

Tabelle 3.4: Ermittlung der Kapazitäten mit einer Kostenfunktion (Beispiel). Das Beispiel beschreibt die Bearbeitung von zehn Produkten (PRD) unter Nutzung von neun MAEs. Alle Produkte außer dem ersten und dem letzten durchlaufen Prozessschritte mit zwei MAEs. Jede MAE wird für zwei Produkte genutzt. Die Taktzeiten seien wie angegeben definiert, und es sei das Ziel, die Kapazitäten durch Maximierung einer Kostenfunktion zu ermitteln. Um die Produkte aufsteigend zu priorisieren, müssen exponentiell wachsende Erlöskoeffizienten definiert werden.

Als Ergänzung wird an einem Beispiel erläutert, worin die Schwierigkeit besteht, gemäß Alternative C geeignete Erlöskoeffizienten festzulegen. Wie in Tabelle 3.4 dargestellt, seien zehn Produkte und neun MAEs gegeben. Für jedes Produkt seien ein bzw. zwei Prozessschritte mit einer bestimmten Taktzeit und jeweils einer MAE auszuführen. Die Produkte seien entsprechend ihrer Reihenfolge aufsteigend zu priorisieren. Das heißt, bei gleicher Auslastung der MAEs sollen durch Maximierung der Kostenfunktion stets die Stückzahlen höher priorisierter Produkte gesteigert werden. Der Anwender muss nun Erlöskoeffizienten bestimmen, sodass die Stückzahl eines Produkts gegenüber allen vorherigen priorisiert wird. Durch die Abhängigkeiten und die Taktzeiten muss der Erlöskoeffizient eines beliebigen Produkts mehr als zehnmal größer als der Wert des jeweils vorangegangenen sein. Andernfalls würde der Gesamterlös wachsen oder gleich bleiben, wenn bei gleicher Auslastung der MAEs die Stückzahl eines Produkts reduziert und die Stückzahl eines Produkts niedrigerer Priorität um die zehnfache Differenz gesteigert wird. Bei der Suche geeigneter Erlöskoeffizienten treten zwei Probleme auf: Erstens ist die Suche sehr aufwändig, da alle Prozessschritte, Verknüpfungen und Taktzeiten berücksichtigt werden müssen. Zweitens wachsen die Erlöskoeffizienten im ungünstigsten Fall exponentiell, sodass die Kostenfunktion als Folge numerischer Instabilitäten nicht mehr lösbar ist.

In Bezug auf das erste Problem verdeutlicht das Beispiel, dass alle Abhängigkeiten eines Modells in die Ermittlung geeigneter Werte einbezogen werden müssen. Die Komplexität erfährt eine zusätzliche Steigerung, falls Prozessschritte selektiv miteinander verknüpft sind. In diesem Fall muss der Anwender das Modell dahingehend prüfen, ob die Bearbeitung von zwei Produkten mit derselben MAE erfolgt und dem Produkt mit der höheren Priorität eine größere Quote zugeordnet ist. Umso höher muss der Erlöskoeffizient für dieses Produkt sein, da bei gegebener Auslastung und gleichen Taktzeiten eine größere Stückzahl des niedriger

priorisierten Produkts möglich ist. Bei alledem ist entscheidend, welche MAE die Kapazitäten begrenzt. Da diese aber oftmals unbekannt ist, muss der Anwender das maximale Verhältnis der Taktzeiten zweier Produkte suchen, unter Berücksichtigung der Quoten selektiver Verknüpfungen. Gemäß diesem Verhältnis muss der Anwender die Erlöskoeffizienten festlegen, sodass die Produkte wie vorgegeben priorisiert werden.

Das exponentielle Wachstum der Erlöskoeffizienten führt zum zweiten Problem. Eine Kostenfunktion ist mit Hilfe von LP oder MILP nur dann exakt lösbar, wenn das Verhältnis ihrer Koeffizienten zueinander nicht größer als 10^{10} ist.[9] Ist das aber der Fall, wie im beschriebenen Beispiel, besteht die Gefahr numerischer Instabilitäten. Unter dem Begriff werden Ungenauigkeiten durch Rundungsfehler verstanden, die darauf zurückzuführen sind, dass die verfügbare Zahl von Kommastellen für Gleitkommawerte endlich ist. Die Auswirkung solcher Fehler steigt, wenn Berechnungen mit Werten unterschiedlicher Größenordnungen durchgeführt werden. Die Fehler können sich im Verlauf der Optimierung weiter verstärken, sodass keine Lösung gefunden wird. Selbst wenn der Anwender den immensen Aufwand akzeptiert, geeignete Erlöskoeffizienten zu ermitteln, ist es möglich, dass die Maximierung der Kostenfunktion nicht zu einer Lösung führt.

Schlussendlich soll belegt werden, dass das Beispiel die Komplexität realer Systeme widerspiegelt. In entsprechenden Systemen werden deutlich mehr Produkte mit einer ungleich größeren Zahl von MAEs hergestellt, wie Tabelle 2.2 auf Seite 38 zeigt (im Mittel 59 Produkte unter Nutzung von 136 MAEs). In den Wertströmen der Produkte werden meist mehrere Prozessschritte verknüpft (im Mittel 9,7 Prozessschritte je Produkt). Die MAEs werden zudem nicht nur zur Ausführung von Prozessschritten eines Produkts, sondern verschiedener Produkte genutzt (siehe genannte Tabelle, maximal 67 Produkte je MAE).

Weiter wird angenommen, dass die Taktzeiten der Prozessschritte zweier Produkte mit derselben MAE in einem Verhältnis stehen, welches dem Faktor zehn entspricht. Das gleiche Verhältnis wird für die Taktzeiten zweier Prozessschritte eines Produkts mit verschiedenen MAEs vorausgesetzt. In der Praxis treten Taktzeiten in noch größeren Verhältnissen auf, wie die Werte in Tabelle 3.5 verdeutlichen. Eine Härterei wird z. B. für Produkte mit Taktzeiten verschiedener Größenordnungen genutzt. Betrachtet man die Fertigung eines Produkts, unterscheiden sich oft die Taktzeiten zur automatisierten und manuellen Bearbeitung. Zwar führt in dem Beispiel jede andere Priorisierung der Produkte zu einem geringeren Wachstum der Erlöskoeffizienten. Da aber die Prioritäten eine Prämisse für die Planung darstellen, muss hierbei vom ungünstigsten Fall ausgegangen werden. Damit lässt sich zusammenfassen, dass das Beispiel aussagekräftig ist, um die Probleme aufzuzeigen, Kapazitäten mit Hilfe einer Kostenfunktion zu maximieren.

[9] IBM ILOG CPLEX Optimization Studio 12.6.2: Diagnosing Performance Problems, Numeric Difficulties. Intern von SAP APO SNP verwendeter Solver, Benutzerhandbuch. Abgerufen am 26. Januar 2018. http://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.6.2/ilog.odms.cplex.help/CPLEX/UsrMan/topics/cont_optim/simplex/20_num_difficulty.html.

Standort*	Region	Taktzeiten (zwei Produkte, eine MAE)			Taktzeiten (ein Produkt, zwei MAEs)		
		max $t_{eff,1} : t_{eff,2}$	$t_{eff,1}$, in min	$t_{eff,2}$, in min	max $t_{eff,1} : t_{eff,2}$	$t_{eff,1}$, in min	$t_{eff,2}$, in min
Standort 1	EMEA	31,3	5,95	0,19	**114,5**	21,76	0,19
Standort 2	EMEA	5,6	5,94	1,06	23,1	6,23	0,27
Standort 3/1	AMER	1,8	9,00	5,00	10,0	15,00	1,50
Standort 3/2	AMER	7,9	55,00	7,00	28,3	85,00	3,00
Standort 4	EMEA	**100,0**	10,00	0,10	**100,0**	10,00	0,10
Standort 5	EMEA	6,9	6,90	1,00	27,7	36,00	1,30
Standort 6/1	EMEA	4,0	4,33	1,08	16,1	2,15	0,13
Standort 6/2	EMEA	11,8	811,00	69,00	31,7	1320,00	41,70
Standort 7	APAC	**62,8**	15,70	0,25	70,3	17,57	0,25
Standort 8	AMER	3,0	1,44	0,48	16,7	8,33	0,50
Standort 9/1	EMEA	**42,4**	93,20	2,20	63,8	185,00	2,90
Standort 9/2	EMEA	28,9	1530,00	53,00	**660,0**	660,00	1,00
Mittelwert		25,5			96,8		

*anonymisiert

Tabelle 3.5: Größenordnungen von Taktzeiten (Beispielzahlen). Es werden die Planungen der Bosch Rexroth AG aus Tabelle 2.2 auf Seite 38 betrachtet. Die Tabelle zeigt zum einen das maximale Verhältnis der Taktzeiten von zwei Prozessschritten, die zur Bearbeitung verschiedener Produkte dienen und denen dieselbe MAE zugeordnet ist. Zum anderen ist dieses Verhältnis für zwei Prozessschritte eines Produkts angegeben, deren Ausführung jeweils verschiedene MAEs erfordert.

Erfüllungsgrad der Basisanforderung A11
Minimierung der Investitionen

Um auf der Grundlage geplanter Produktstückzahlen die minimalen Investitionen zu bestimmen, ist es analog zur Maximierung der Kapazitäten möglich, eine Kostenfunktion zu definieren. Voraussetzung hierfür ist, dass die Überlastung der verfügbaren MAEs mit Hilfe spezifischer Kosten bewertet wird. Mit SAP APO SNP ist es möglich, Kosten zur Nutzung von Ressourcen in zusätzlichen Schichten festzulegen (*Production Resource Capacity Expansion Cost*, siehe Kallrath und Maindl 2006, S. 19). Im Anschluss daran werden die Gesamtkosten zur Herstellung der geplanten Produktstückzahlen minimiert.

Jedoch ist die Minimierung einer Kostenfunktion auch in diesem Fall nur begrenzt anwendbar. Der Grund liegt darin, dass die Überlastung einer MAE ggf. reduziert werden kann, indem die Überlastung anderer MAEs erhöht wird. Das heißt, es können unendlich viele Lösungen existieren, welche die Kostenfunktion minimieren. Daher wird auch im Hinblick auf die Minimierung der Investitionen gefordert, dass nach einer festgelegten Strategie eine optimale Lösung ausgewählt wird. Von den möglichen Alternativen, die im Zusammenhang mit der Maximierung der Kapazitäten diskutiert wurden, kommt nur Alternative C in Betracht. Diese ist jedoch aus den gleichen Gründen, der Schwierigkeit, die Kostenfaktoren zu bestimmen, und daneben ihrem exponentiellen Wachstum, im Allgemeinen nicht praktikabel. Die Basisanforderung A11 ist ebenso nur *teilweise erfüllt*.

Erfüllungsgrad der Basisanforderung A12
Optimierung der Auslastung

Der Ansatz, eine Kostenfunktion zu minimieren, kann auch auf die Optimierung der Auslastung angewandt werden. Zu diesem Zweck werden die Prioritäten alternativ verknüpfter Prozessschritte als Priorisierung der zugeordneten MAEs interpretiert. Dabei wird jedoch bewusst vernachlässigt, dass MAEs i. d. R. für verschiedene alternativ verknüpfte Prozessschritte und verschiedene Produkte genutzt werden. Da SAP APO SNP nicht die Möglichkeit bietet, eine Zielfunktion in mehreren Iterationsschritten zu optimieren, wird wieder die zuvor diskutierte Alternative C aufgegriffen. Der Anwender legt in diesem Fall spezifische Kosten für die Unterauslastung jeder Ressource fest (*Production Resource Capacity Under-Utilization Penalty*, siehe Kallrath und Maindl 2006, S. 19), wobei die Kosten umso größer sein müssen, je höher die Priorität der MAE ist. Darauf folgt die Minimierung der Gesamtkosten, um die geplanten Produktstückzahlen herzustellen.

Analog zur Maximierung der Kapazitäten und der Minimierung der Investitionen kann jedoch eine Priorisierung mit Hilfe einer Kostenfunktion nur begrenzt abgebildet werden. Die Kostenfaktoren sind schwierig zu bestimmen und wachsen aufgrund der großen Zahl genutzter MAEs in einem Maße, dass numerische Instabilitäten das Finden einer Lösung verhindern. Auch die Basisanforderung A12 ist daher nur *teilweise erfüllt*.

Erfüllungsgrad der Basisanforderung A13
Globales Optimum

SAP APO SNP nutzt intern den Solver IBM ILOG CPLEX[10]. Dieser wendet zur Lösung von Optimierungsproblemen exakte, mathematische Verfahren an (LP und MILP). Dadurch ist gewährleistet, dass in jedem Fall das globale Optimum gefunden wird, sofern eine Lösung existiert. Die Basisanforderung A13 ist somit *erfüllt*.

Erfüllungsgrad der Leistungsanforderung A14$^+$
Lineare obere Laufzeitschranke

Die Erfüllung der Anforderung kann nicht unmittelbar geprüft werden, da die Bedingung modellabhängig formuliert ist und das Modell nicht offen vorliegt. Allerdings greift SAP APO SNP intern zur Lösung auf IBM ILOG CPLEX und damit auf kommerzielle Software zurück, die zusammen mit FICO Xpress[11] als führend auf dem Gebiet der mathematischen Optimierung gilt (siehe Kallrath und Maindl 2006, S. 39). Wie in den nachfolgenden Kapiteln ausgeführt wird, ist es bereits mit frei verfügbarer Software zur linearen Optimierung möglich, die Bedingung in Bezug auf die Laufzeit zu erfüllen. Unter der Voraussetzung, dass das Modell die Systemstruktur vollständig abbildet und folglich den gleichen Detaillierungsgrad aufweist, gilt die Leistungsanforderung A14$^+$ deshalb als *erfüllt*.

[10] IBM Analytics: CPLEX Optimizer. Abgerufen am 26. Januar 2018. http://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer/.
[11] FICO® Xpress Optimization. Abgerufen am 26. Januar 2018. http://www.fico.com/en/products/fico-xpress-optimization/.

Ergebnis der Bewertung

Positiv ist zuallererst anzumerken, dass SAP APO SNP die Basisanforderungen A1–A6, welche die Modellierung betreffen, vollständig erfüllt. Mit den bereitgestellten Datenstrukturen können die Elemente eines gegebenen Systems von Wertströmen und alle Beziehungen zwischen diesen Elementen abgebildet werden. Da zur Optimierung exakte, mathematische Verfahren Anwendung finden, ist zudem sichergestellt, dass die globalen Optima innerhalb der definierten linearen oberen Laufzeitschranke gefunden werden. Dementsprechend erfüllt SAP APO SNP die Basisanforderung A13 bzw. die Leistungsanforderung A14$^+$. Zusätzlich bietet die Software die Möglichkeit, die Planung auf der Grundlage aktueller Daten aus dem ERP-System zu optimieren und die Ergebnisse zurück an das ERP-System zu übergeben (Echtzeit-Integration, siehe Kallrath und Maindl 2006, S. 9 f).

Negativ ist dagegen, dass eine grafische Modellierung der Wertströme nicht unterstützt wird, wie es die Leistungsanforderungen A7$^+$–A9$^+$ verlangen. Des Weiteren werden im Hinblick auf die Optimierung des Modells die Basisanforderungen A10–A12 nur teilweise erfüllt. Hintergrund ist, dass eine definierte Kostenfunktion nur in einem einzigen Schritt optimiert werden kann und die Abbildung der Planungsziele mit Hilfe einer solchen Kostenfunktion in der Praxis mit deutlichen Einschränkungen verbunden ist. Zudem wird vom Anwender Expertenwissen im Bereich mathematischer Modellierung vorausgesetzt, das notwendig ist, um ein geeignetes Lösungsverfahren auszuwählen und die Optimierungsergebnisse zu interpretieren (siehe Kallrath und Maindl 2006, S. 102). Daher ist zu bezweifeln, dass ein unternehmensweiter Einsatz der Software zur strategischen Planung der TEK die Effizienz im Planungsprozess steigert. Die Nutzung von SAP APO SNP erscheint weniger für diesen Zweck, sondern vielmehr als Erweiterung eines ERP-Systems zur Planung und Optimierung in Unternehmensnetzwerken lohnenswert.

3.5 Software zur Prozessmodellierung

Die Modellierung von Geschäftsprozessen, in der Literatur oftmals auch unter der englischen Bezeichnung *Business Process Modeling* (BPM) zu finden, bildet die Grundlage zur Planung und Steuerung der Abläufe in Unternehmen. Der Begriff ist laut van der Aalst (2004a, S. 248 ff), einem bekannten Vertreter des Fachgebiets, wie folgt definiert:

> »Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information.«

Laut dem *Handbook on Business Process Modeling* ist BPM die Fortführung von zwei Strömungen (siehe Hammer 2010, S. 3 f): (1) der statistischen Prozesssteuerung, die erstmals den Prozess in den Vordergrund stellt, diesen quantitativ im Hinblick auf Abweichungen und Kennzahlen bewertet und kontinuierlich im Detail verbessert (siehe Deming 1953, Shewhart 1986), und (2) dem Business Process Reengineering (BPR), welches den Prozess mit dem Ziel der vollständigen Umgestaltung neu definiert als Arbeit, die sich durch das Unternehmen erstreckt und Werte für den Kunden schafft (siehe Hammer 1990, Hammer und Champy

1993). Für Details soll der Verweis auf die Literatur genügen (siehe u. a. van der Aalst 2004a,b, 2013, vom Brocke und Rosemann 2010, Havey 2005, Ko 2009, Ko et al. 2009).

Im Kontext von BPM existiert eine Vielzahl sich oft überschneidender Beschreibungssprachen und Notationen (für eine Übersicht siehe Ko et al. 2009). Viele Standards unterstützen zumindest einen Schritt des Lebenszyklus von BPM (siehe Ko et al. 2009, S. 751). Dieser läuft nach van der Aalst et al. (2003, S. 5) in den Schritten der Prozessgestaltung, der Systemkonfiguration, der Prozessausführung und der Diagnose ab. Die verfügbaren Standards lassen sich wie folgt kategorisieren (siehe Ko et al. 2009, S. 751 ff):

(1) grafische Standards (*Graphical Standards*),
(2) Ausführungsstandards (*Execution Standards*),
(3) Austauschstandards (*Interchange Standards*),
(4) Diagnosestandards (*Diagnosis Standards*).

Im weiteren Verlauf wird die Prozessmodellierung mit Hilfe eines grafischen Standards betrachtet und auf Prozesse in der Produktion übertragen. Die Grundlage bilden Prozesse im Verständnis eines Systems von Wertströmen, wie im vorigen Kapitel beschrieben. Beispiele wichtiger grafischer Standards sind UML AD (Unified Modeling Language Activity Diagrams, siehe OMG 2015), EPC (Event-Driven Process Chains, siehe Keller et al. 1992, Nüttgens und Rump 2002), YAWL (Yet Another Workflow Language, auch zur Prozessausführung geeignet, siehe van der Aalst et al. 2004, ter Hofstede et al. 2010) und BPMN.

3.5.1 Beispiel: BPMN mit idealem Interpreter und Optimierer

Unter allen grafischen Standards für die Prozessmodellierung ist BPMN (Business Process Model and Notation) in der Praxis der dominierende (siehe Ko et al. 2009, S. 756, Recker 2010, S. 182). Zudem ist BPMN einer derjenigen grafischen Standards, die sich am besten für die Integration, den Austausch und die Ausführung der modellierten Prozesse eignen (siehe Ko et al. 2009, S. 754). Diese Punkte führen zur Entscheidung, BPMN als Grundlage für die folgende Bewertung zu verwenden. Der Standard wird durch die Object Management Group (OMG) definiert und liegt aktuell (Stand Januar 2018) in der Version 2.0.2 vor (Spezifikation siehe OMG 2013). Für eine Einführung zu BPMN wird auf die einschlägige Literatur verwiesen (siehe u. a. Allweyer 2010, Freund und Rücker 2012, Silver 2011).

Zum Zeitpunkt der Bewertung existieren 64 Softwareanwendungen[12], die BPMN implementieren. Um die Ergebnisse auf möglichst viele Vertreter des Softwaretyps übertragen zu können, erfolgt die Bewertung nicht für eine spezifische Software zur Prozessmodellierung. Stattdessen wird von einer Softwareanwendung ausgegangen, welche die theoretischen Möglichkeiten von BPMN maximal ausschöpft.

Um einen Prozess zu modellieren und zu implementieren, wird dieser zunächst mit Hilfe einer grafischen Notation (z. B. in BPMN) beschrieben. Anschließend wird das grafische Modell in einen Ausführungsstandard wie z. B. WS-BPEL (Web Services Business Process Execution Language) übersetzt, einem Standard der Organization for the Advancement of

[12] OMG BPMN: Implementers. Abgerufen am 27. Januar 2018. http://www.bpmn.org/.

Structured Information Standards (Spezifikation siehe OASIS 2007). Danach ist es möglich, den Prozess zu testen, anzupassen und auszuführen (siehe Ko 2009, S. 17). Bei der Übersetzung der Prozesse von einem grafischen in einen Ausführungsstandard treten in der Praxis jedoch Schwierigkeiten auf. Diese Schwierigkeiten resultieren aus konzeptionellen Unterschieden der Standards (siehe Ko et al. 2009, S. 757 am Beispiel von BPMN und WS-BPEL sowie Koskela und Haajanen 2007, Recker und Mendling 2006, 2007).

Dessen ungeachtet ist dieses Vorgehen zumindest theoretisch auf das vorliegende Problem transferierbar. Es ist vorstellbar, ein System von Wertströmen einschließlich aller Prozesse in der Produktion zunächst in BPMN abzubilden und anschließend durch einen Interpreter zu übersetzen. Das Ziel der Übersetzung ist dabei nicht die Ausführung der Prozesse, sondern die Optimierung des Modells gemäß den definierten Planungszielen. Dementsprechend ist das Zielformat kein Ausführungsstandard wie z. B. WS-BPEL, sondern ein geeignetes mathematisches Modell (vgl. SAP APO SNP).

Zur Übersetzung von BPMN existieren verschiedene Interpreter, die jedoch unterschiedlichen Limitationen unterworfen sind. Der Grund ist die mangelnde formale Eindeutigkeit der grafischen Notation von BPMN (siehe Ko et al. 2009, S. 759). Um Software zur Prozessmodellierung unabhängig von der Leistung des Interpreters zu bewerten, wird deshalb ein hypothetischer idealer Interpreter und Optimierer angenommen. Durch die Bewertung am Beispiel von BPMN mit einem solchen Interpreter und Optimierer ist es möglich, (1) BPMN zur grafischen Modellierung auszuschließen, falls die Anforderungen unzureichend erfüllt werden, und (2) Lösungsideen für eine neue Entwicklung zu diskutieren, die sich zur grafischen Modellierung an BPMN orientiert oder bewusst von BPMN abweicht.

In Bezug auf die grafischen Symbole definiert BPMN in der Version 2.0.2 fünf grundlegende Kategorien von Elementen (siehe OMG 2013, S. 25 f):

(1) *Flow Objects* (*Events*, *Activities* und *Gateways*),
(2) *Data* (*Data Objects*, *Data Inputs*, *Data Outputs* und *Data Stores*),
(3) *Connecting Objects* (mit weiteren Unterkategorien),
(4) *Swimlanes* (*Pools* und *Lanes*),
(5) *Artifacts*.

Die sogenannten *Flow Objects* sind die wichtigsten Elemente, da sie den Ablauf eines Prozesses definieren. Elemente des Typs *Data* modellieren den Datenfluss. *Connecting Objects* verbinden *Flow Objects* untereinander oder verknüpfen diese mit weiteren Informationen. *Swimlanes* gruppieren Elemente eines Prozesses. *Artifacts* fügen einem Modell zusätzliche, unstrukturierte Informationen über den Prozess hinzu (siehe OMG 2013, S. 25 f).

BPMN stellt damit alle grafischen Elemente zur Verfügung, um allgemeine Prozesse in Unternehmen zu beschreiben. Abbildung 3.2 illustriert die Verknüpfung dieser Elemente am Beispiel eines Versandprozesses. Im Kontext der vorliegenden Arbeit wird die Beschreibung von Prozessen in der Produktion betrachtet. Hierzu müssen die Elemente um quantitative Informationen für die Kalkulation der Auslastung erweitert werden (z. B. Betriebsmittelzeiten der genutzten MAEs und Taktzeiten der Prozessschritte).

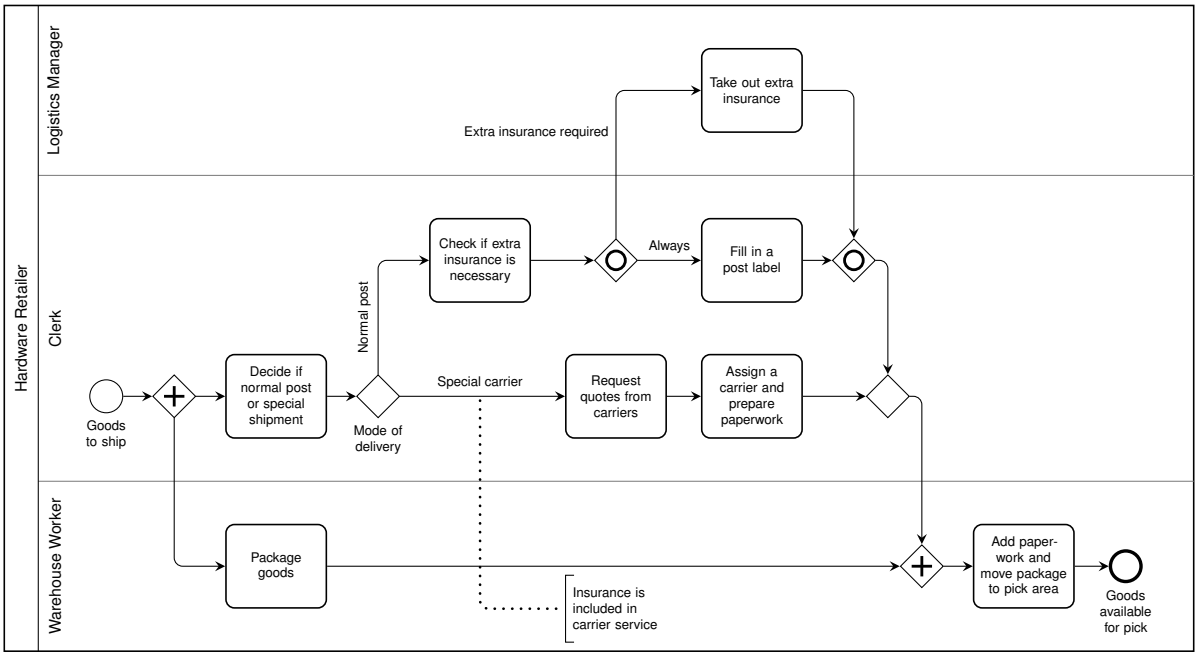


Abbildung 3.2: Modellierung eines einfachen Prozesses mit BPMN (Beispiel). Die Abbildung beschreibt den Versandprozess eines Eisenwarenhändlers. Durch die Einführung jeweils verschiedener *Lanes* werden die Verantwortlichkeiten der handelnden Personen abgegrenzt. *Gateways* definieren Verzweigungen und Vereinigungen, die unabhängig voneinander oder bedingt auszuführende Prozessabschnitte kennzeichnen (Quelle und Erläuterung siehe OMG 2010).

3.5.2 Erfüllungsgrad der Anforderungen

Erfüllungsgrad der Basisanforderung A1
Superposition aller Wertströme

BPMN definiert eine Vielzahl von Klassen und Elementen, welche die Liste im vorigen Abschnitt erweitern. Die Klasse *Resource* fasst Objekte zusammen, die von *Activities* referenziert werden können (siehe OMG 2013, S. 93 f, 152 ff). Unter einer Referenz wird im Verständnis der Objektorientierung der Verweis eines Objekts auf ein anderes Objekt verstanden. In diesem Fall handelt es sich um den Verweis einer *Activity* auf eine *Resource*, die zur Ausführung der *Activity* erforderlich ist. Dabei können verschiedene *Activities* dieselbe *Resource* referenzieren. Auf diese Weise ist es möglich, die Zuordnung von MAEs zu Prozessschritten abzubilden. In Bezug auf die Software zur Prozessmodellierung wird angenommen, dass alle Wertströme des abzubildenden Systems in einem Modell zusammengefasst werden können. Unter dieser Annahme ist die Basisanforderung A1 *erfüllt*.

Erfüllungsgrad der Basisanforderung A2
Systemstruktur

Mit den zur Verfügung stehenden Elementen von BPMN können auch komplexe Systeme in allen erforderlichen Details abgebildet werden. Werden die Elemente geeignet verbunden,

entspricht die Struktur des Modells dem abzubildenden System im Hinblick auf Wertströme, Prozessschritte/Prozesse und MAEs. Die Basisanforderung A2 ist daher *erfüllt*.

Im Folgenden soll bewertet werden, inwieweit hierbei die Besonderheiten eines Systems von Wertströmen berücksichtigt werden können. Dazu zählen die unterschiedlichen Typen der Verknüpfung von Prozessschritten und die Anwendungsfälle der Vereinigung von Prozessen, die jeweils ein und dieselbe MAE durchlaufen.

Erfüllungsgrad der Basisanforderung A3
Systemschnittstelle

Mit BPMN können *Data Inputs* und *Data Outputs* definiert werden, um zu kennzeichnen, an welchen Stellen Informationen zwischen dem System und der Umwelt ausgetauscht werden (siehe OMG 2013, S. 25 f, 210 ff). Um das Modell zu optimieren, muss auf der Grundlage vorgegebener Produktstückzahlen und Prozessstückzahlen die resultierende Auslastung aller MAEs bestimmt werden. Die Zuordnung entsprechender Ausgaben zu den Eingaben des Informationsflusses ist abhängig vom eingesetzten Interpreter. Unter Annahme eines hypothetischen idealen Interpreters ist die Basisanforderung A3 *erfüllt*.

Erfüllungsgrad der Basisanforderungen A4–A6
Sequenzielle, alternative und selektive Verknüpfung

Sequence Flows bezeichnen Elemente, die zur Gruppe der *Connecting Objects* gehören (siehe OMG 2013, S. 26 f, 95 ff). Sie erlauben es, *Events*, *Activities* und *Gateways* zu verknüpfen, und werden als gerichtete Pfeile dargestellt. Dadurch kann ausgedrückt werden, dass die verknüpften Elemente in einer definierten Abfolge durchlaufen werden, wie Abbildung 3.2 anhand eines Beispiels zeigt. Auf diese Weise ist es möglich, sequenzielle Verknüpfungen in einem Modell abzubilden (A4). *Gateways* beschreiben Elemente, an denen *Sequence Flows* vereinigt oder verzweigt werden (*Merging* und *Branching*, siehe S. 27, 286 ff, 434 ff OMG 2013). Mit Hilfe von *Gateways* können komplexe Bedingungen definiert werden, die bestimmen, welcher der nachfolgenden *Sequence Flows* fortgesetzt wird. Insbesondere können dadurch alternative und selektive Verknüpfungen in einem Modell wiedergegeben werden (A5 bzw. A6). Die Basisanforderungen A4–A6 sind somit *erfüllt*.

Erfüllungsgrad der Leistungsanforderung A7$^+$
Grafische Modellierung

BPMN ist zwar ein grafischer Standard zur Prozessmodellierung, unterstützt aber nicht in vollständigem Umfang die grafische Darstellung aller Beziehungen zwischen den Elementen eines Modells. Dieser Punkt ist insbesondere für Referenzen von *Activities* zu *Resources* von Bedeutung, da verschiedenen Prozessschritten je nach Anwendungsfall in der Praxis dieselbe MAE zugeordnet sein kann (siehe Superposition). Referenzen zur Zuordnung von *Resources* werden in BPMN auf Basis von XML (Extensible Markup Language) definiert und nicht durch grafische Symbole repräsentiert (siehe OMG 2013, S. 169 ff).

Um die Zuordnung von *Resources* zu visualisieren, ist es darüber hinaus möglich, *Activities* grafisch in *Lanes* zu gruppieren. In der Literatur wird diese Möglichkeit oft verwendet, um

Rollen von Personen oder Abteilungen abzugrenzen. In diesem Kontext ist sie theoretisch auch auf die Zuordnung von MAEs zu Prozessschritten anwendbar. Eine *Lane* bezeichnet die horizontale oder vertikale Partition eines grafischen Modells entlang der gesamten Breite bzw. Tiefe des Prozesses. Sie dient der Gruppierung sämtlicher *Activities*, die in Bezug auf ein bestimmtes Merkmal übereinstimmen (siehe OMG 2013, S. 304 ff). Entsprechend können *Lanes* im vorliegenden Fall genutzt werden, um alle *Activities* zusammenzufassen, denen dieselbe *Resource* zugeordnet ist. Allerdings eignet sich die Gruppierung von *Activities* mit Hilfe von *Lanes* nicht für große Zahlen von Prozessschritten und MAEs. Aus diesem Grund ist die Leistungsanforderung A7$^+$ nur *teilweise erfüllt*.

Diesem Ansatz wird eine Idee gegenübergestellt, die eine höhere Transparenz mit Blick auf die Beziehungen zwischen den Elementen verspricht: Nicht Prozessschritte, sondern MAEs werden durch Knoten repräsentiert und durch Kanten verknüpft, wie es die Beispielschemata im vorigen Kapitel zeigen. Prozessschritte werden durch eingehende Kanten an einer MAE visualisiert, die je nach Anwendungsfall getrennt oder vereinigt werden. Um den Weg jedes Stücks intuitiv abzubilden, werden die MAEs räumlich frei angeordnet, nicht notwendigerweise horizontal oder vertikal gruppiert. Dadurch ist eine grafische Darstellung für eine große Zahl an Prozessschritten und MAEs möglich. Auf der Grundlage dieser Idee wird im folgenden Kapitel eine neue, eigenständige grafische Notation entwickelt.

Erfüllungsgrad der Leistungsanforderung A8$^+$
Minimale Symbolmenge

Da die Elemente von BPMN nicht formal definiert und z. T. mehrdeutig sind, kann nicht von einer minimalen Symbolmenge gesprochen werden (siehe Dijkman et al. 2008, S. 1281 ff, Ko et al. 2009, S. 759, van Nuffel et al. 2009, S. 115 ff, Wohed et al. 2006, S. 161 ff). Als Konsequenz ist die Leistungsanforderung A8$^+$ *nicht erfüllt*.

Erfüllungsgrad der Leistungsanforderung A9$^+$
Modelltransformation

Selbst in dem Fall, dass von einem idealen Interpreter ausgegangen wird, ist die Umwandlung eines grafischen Modells in ein mathematisches Modell als Folge der nicht formalen, mehrdeutigen Notation von BPMN nicht immer vollständig möglich (siehe minimale Symbolmenge, vgl. Umwandlung von BPMN in WS-BPEL). Notwendig ist eine Formalisierung der Notation, indem die verfügbaren Symbole eingeschränkt und Konventionen zur Verknüpfung der Symbole definiert werden. Die Leistungsanforderung A9$^+$ ist daher auch unter der Annahme eines idealen Interpreters nur *teilweise erfüllt*.

Erfüllungsgrad der Basisanforderungen A10–A13
Maximierung der Kapazitäten, Minimierung der Investitionen, Optimierung der Auslastung und globales Optimum

Neben einem idealen Interpreter wird für die Bewertung ein ebenso hypothetischer Optimierer angenommen, welcher die Anforderungen bestmöglich erfüllt. Im vorliegenden Fall

muss dieser ein exaktes, mathematisches Verfahren wie z. B. LP und MILP anwenden (vgl. SAP APO SNP). Nur dann ist gewährleistet, dass für jede definierte Zielfunktion das globale Optimum gefunden wird. Voraussetzung für die Anwendung eines solchen Verfahrens ist allerdings, dass das Modell in mathematischer Form vorliegt. Die Erfüllung der Anforderungen in Bezug auf die Optimierung ist somit an die Erfüllung der Leistungsanforderung A9$^+$ geknüpft. Entsprechend sind die Basisanforderungen A10–A13 selbst unter der Annahme eines idealen Interpreters und Optimierers nur *teilweise erfüllt*.

Erfüllungsgrad der Leistungsanforderung A14$^+$
Lineare obere Laufzeitschranke

Es findet die gleiche Argumentation wie im vorigen Absatz Anwendung. Eine vollständige Transformation des grafischen Modells in ein mathematisches Modell bildet die Voraussetzung für die Nutzung eines exakten Optimierungsverfahrens, welches die lineare obere Schranke für die Laufzeit erfüllt. Die Leistungsanforderung A14$^+$ ist entsprechend auch unter der formulierten Annahme nur *teilweise erfüllt*.

Ergebnis der Bewertung

Positiv ist zusammenzufassen, dass BPMN geeignet ist, ein System von Wertströmen vollständig abzubilden. Eine Softwareanwendung, die zur Modellierung auf BPMN basiert, erfüllt damit die Basisanforderungen A1–A6. In Bezug auf die Abbildung der Systemschnittstelle gemäß Basisanforderung A3 wird dabei angenommen, dass die Softwareanwendung in Kombination mit einem hypothetischen idealen Interpreter und Optimierer eingesetzt wird. Ausgewählte Elemente von BPMN und das Konzept eines Interpreters und Optimierers werden in den folgenden Kapiteln für das vorliegende Problem adaptiert.

Negativ ist dagegen, dass die grafische Modellierung eines Systems nur mit Einschränkungen möglich ist, da nicht alle Beziehungen zwischen den Elementen des Systems grafisch abgebildet werden können. Auch wenn sich BPMN als weit verbreiteter Standard besonders zur Visualisierung von Geschäftsprozessen eignet, ist damit die Leistungsanforderung A7$^+$ nur teilweise erfüllt. Weiterhin basiert die grafische Modellierung mit BPMN nicht auf einer minimalen Symbolmenge, sondern auf nicht formal definierten, z. T. mehrdeutigen Symbolen. Die Leistungsanforderung A8$^+$ kann daher nicht als erfüllt gelten. Damit einhergehend ist nicht sichergestellt, dass das grafische Modell vollständig in ein mathematisches Modell umgewandelt werden kann, selbst wenn man die zuvor getroffene Annahme eines idealen Interpreters und Optimierers zugrunde legt. Die Leistungsanforderung A9$^+$ wie auch die Anforderungen A10–A14$^+$, welche die Optimierung des mathematischen Modells betreffen, sind aus diesem Grund nur teilweise als erfüllt anzusehen.

Um BPMN für die Modellierung von Wertströmen einzusetzen und auf dieser Grundlage die TEK strategisch zu planen, wäre es zuvor notwendig, die zur Verfügung stehenden Symbole einzuschränken. Zudem müssten Konventionen definiert werden, wie die verbleibenden Symbole zu verknüpfen sind und welche zusätzlichen Informationen abgebildet werden müssen, um die Auslastung kalkulieren zu können. Jedoch stellen solche Konventionen eine Fehlerquelle dar, da nicht garantiert ist, dass die Anwender diese berücksichtigen.

Zweckmäßiger ist es, eine Software zu entwickeln, welche diese Konventionen auf Basis einer formal eindeutigen, grafischen Notation automatisch umsetzt.

3.6 Fazit: Bedarf nach einer neuen Entwicklung

In Abschnitt 2.6 des vorigen Kapitels wurde eine Systemanalyse durchgeführt, die in der Formulierung der Anforderungen A1–A14$^+$ mündete. In diesem Kapitel erfolgte eine Bewertung kommerzieller und frei verfügbarer Software in Bezug auf die Erfüllung dieser Anforderungen. Da in der Praxis ein unüberschaubares Angebot in Frage kommender Software existiert, wurden zunächst Softwaretypen auf der Grundlage von zwei Kriterien ausgewählt: Die Software muss sich erstens grundsätzlich für die strategische Planung der TEK eignen, indem sie die Basisanforderungen A1–A6 erfüllt. Zweitens muss sie mit Blick auf die Qualifikation der Anwender und den arbeitsteiligen Planungsprozess produktiv in Unternehmen eingesetzt werden können. Auf dieser Basis wurden vier Softwaretypen identifiziert: Software zur Erstellung von Tabellenkalkulationen, zur Materialflusssimulation, für das Supply Chain Management und schließlich Software zur Prozessmodellierung. Jeder dieser Softwaretypen wurde an einem charakteristischen Beispiel bewertet, wodurch sichergestellt ist, dass die getroffenen Aussagen weitgehend übertragbar sind.

Das Ergebnis der Bewertung lautet zusammengefasst, dass keiner der Softwaretypen alle formulierten Anforderungen erfüllt, wie Tabelle 3.3 auf Seite 81 zeigt. Software zur Erstellung von Tabellenkalkulationen wie Microsoft Excel ist zwar zur Abbildung eines Systems von Wertströmen geeignet, jedoch nicht zu dessen grafischer Modellierung und zur Optimierung des Modells gemäß den definierten Planungszielen. Demgegenüber bietet Software zur Materialflusssimulation wie Siemens Plant Simulation zwar die Möglichkeit, ein grafisches Modell zu erstellen, ist aber zur Optimierung in diesem Kontext ebenso wenig geeignet. Software für das Supply Chain Management wie SAP APO SNP verfügt wiederum i. d. R. über die Möglichkeit zur exakten, mathematischen Optimierung, erreicht die Optima der gegebenen Planungsziele aber auch nur teilweise und bietet dem Anwender keinen grafischen Zugang zur Erstellung des Modells. Zuletzt wurde Software zur Prozessmodellierung auf Basis von BPMN und unter der Annahme eines idealen Interpreters und Optimierers bewertet. Selbst unter dieser Annahme deckt der Softwaretyp nicht alle Anforderungen ab, da die grafische Notation unvollständig, nicht formalisiert und z. T. mehrdeutig ist.

Hervorzuheben ist dabei, dass keiner der Softwaretypen die Leistungsanforderung A7$^+$ zur grafischen Modellierung und die Basisanforderungen A10–A12 bezüglich der Optimierung des Modells vollständig erfüllt. Die Leistungsanforderung A8$^+$, welche die Erstellung eines grafischen Modells auf Basis einer minimalen Symbolmenge beschreibt, erfüllt keiner der Softwaretypen zudem auch nur teilweise. Es besteht als Konsequenz ein begründeter Bedarf nach einer Software für die strategische Planung der TEK, welche die folgenden drei Punkte umsetzt: (1) die Software soll die Möglichkeit bieten, ein System von Wertströmen grafisch vollständig abzubilden, wobei das grafische Modell auf einer minimalen, formal eindeutigen Symbolmenge basieren soll; (2) sie soll dieses Modell in ein mathematisches Modell umwandeln können, und (3) sie soll das mathematische Modell gemäß den vorgegebenen Planungszielen optimieren können. Insbesondere der erste Punkt wird im Zusammenhang

mit den Unzulänglichkeiten grafischer Standards in der Literatur bestätigt. Folgendes Zitat bezieht sich auf BPMN in einer Version, die diesbezüglich nur begrenzt weiterentwickelt wurde, und dient als akzentuierender Beleg (Ko et al. 2009, S. 783): »Graphical standards are easily interpreted by business analysts but lack computational formalisms.«

Diese Forschungslücke schließt die Software AURELIE, die im Rahmen der vorliegenden Arbeit entwickelt wurde und die genannten Punkte vollständig umsetzt. Die betriebswirtschaftliche Relevanz der Entwicklung wird dadurch unterstrichen, dass die Software im Jahr 2012 weltweit an den Standorten der Bosch Rexroth AG mit signifikant messbarem Erfolg eingeführt wurde. Als Ergebnis wurde die Zeit für die strategische Planung der TEK abhängig vom Standort um 50 bis 75 Prozent reduziert. Das heißt, die Effizienz im Planungsprozess, gemessen am Ergebnis der Planung in Relation zur eingesetzten Zeit, wurde um 100 bis 300 Prozent gesteigert. Die Ideen, Modelle und die Kernalgorithmen zur Modellierung und Optimierung werden in den nun folgenden Kapiteln erläutert.

4 Lösungsschritt I: grafische Modellierung und Modelltransformation

Mit der Entwicklung der Software AURELIE wurde das Ziel verfolgt, den im vorigen Kapitel identifizierten Entwicklungsbedarf zu decken. Entsprechend erlaubt es die Software, ein grafisches Modell für ein System von Wertströmen zu erstellen, automatisch in ein mathematisches Modell zu transformieren und gemäß den definierten Planungszielen zu optimieren. Damit erfüllt AURELIE erstmals die formulierten Anforderungen an eine geeignete Software zur strategischen Planung der TEK. In diesem Kapitel sollen die Grundgedanken des ersten Lösungsschrittes, der grafischen Modellierung und der Modelltransformation, begründet werden. Die Transformation eines grafischen Modells in ein mathematisches, das mit Hilfe exakter Verfahren optimiert werden kann, ist einer der wichtigsten Beiträge der vorliegenden Arbeit. Um die gewonnenen Erkenntnisse wiederzugeben, werden die entwickelten Kernalgorithmen zur Validierung und Transformation erläutert und kurz gefasst in natürlicher Sprache beschrieben. Zur Vertiefung und zum Beleg der getroffenen Aussagen sind die ausführlichen Fassungen der Algorithmen in formalem, kommentiertem Pseudocode einschließlich aller Datenstrukturen im Anhang aufgeführt.

Grundlage für die folgenden Betrachtungen ist der Begriff eines Modells, das ein abstrahiertes und vereinfachtes Abbild der Realität darstellt (siehe Bungartz et al. 2009, S. 5, Vajna et al. 2009, S. 98, Pohl und Rupp 2011). Durch die Abbildung eines Systems von Wertströmen mit einem geeigneten Modell können die Prozesse in der Produktion geplant und gestaltet werden, bevor sie realisiert werden. Dies entspricht dem Zweck eines Modells, wie ihn der VDI definiert (Richtlinie VDI 3633:1996-11, S. 9):

> »Ein Modell ist eine vereinfachte Nachbildung eines existierenden oder gedachten Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System. […] Es wird genutzt, um eine […] Aufgabe zu lösen, deren Durchführung […] am Original nicht möglich oder zu aufwendig wäre.«

Das Kapitel ist folgendermaßen aufgebaut: In Abschnitt 4.1 werden zunächst die notwendigen Grundlagen der Graphentheorie und der Komplexitätstheorie eingeführt. Anschließend wird in Abschnitt 4.2 die grafische Modellierung eines gegebenen Systems von Wertströmen mit Hilfe von Wertstromgraphen thematisiert. Nachdem der Anwender ein solches grafisches Modell erstellt hat, muss dieses validiert und transformiert werden, um ein entsprechendes mathematisches Modell zu erzeugen. Die Validierung und die Transformation werden in den folgenden Abschnitten 4.3 bzw. 4.4 erläutert. Danach wird in Abschnitt 4.5

aus praktischer Sicht beleuchtet, wie die vorgestellten Konzepte in der Software AURELIE umgesetzt wurden. Abschließend wird in Abschnitt 4.6 diskutiert und letztlich bestätigt, dass damit das Entwicklungsziel erreicht wird.

4.1 Kurzeinführung: Graphentheorie und Komplexität

Die Graphentheorie ist ein Teilgebiet der Informatik, das sich der abstrakten Beschreibung von mathematischen, insbesondere diskreten Problemen widmet. Naturgemäß bildet die Graphentheorie den Rahmen für die grafische Modellierung eines Systems von Wertströmen. Für die Beschreibung von Algorithmen wird außerdem die Komplexitätstheorie benötigt, welche die Werkzeuge bereitstellt, um das asymptotische Wachstum der Laufzeit und des Speicherplatzbedarfs zu charakterisieren. Um die nötigen Voraussetzungen für die Betrachtungen in diesem Kapitel zu schaffen, folgt hierzu jeweils eine Kurzeinführung.

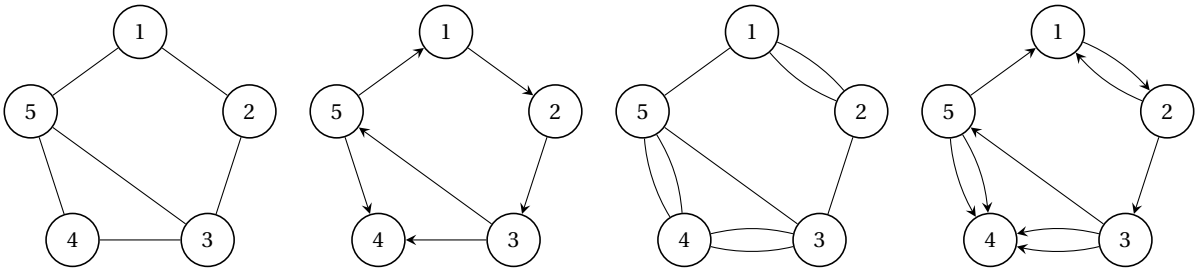
4.1.1 Graphentheorie

Bevor sich der Blick auf die Modellerweiterungen richtet, die im Zuge dieser Arbeit eingeführt werden, sollen die notwendigen graphentheoretischen Begriffe erläutert werden, wie sie in der Literatur Verwendung finden. In diesem Zusammenhang werden die nachfolgenden Symbole definiert und im weiteren Verlauf konkretisiert.

$d_G^-(v)$, $d_G^+(v)$	Eingangsgrad bzw. Ausgangsgrad des Knotens v, $d_G^-(v), d_G^+(v) \in \mathbb{N}_0$
E	Menge der Kanten (engl. *edge*, Plural *edges*) eines Graphen G
G	endlicher Graph mit Knoten V und Kanten E, definiert als Paar $G = (V, E)$
v, v'	aktuell bzw. als Nächstes zu besuchender Knoten, $v, v' \in V$
V	Menge der Knoten (engl. *vertex*, Plural *vertices*) eines Graphen G
σ	allgemeiner, zuerst zu besuchender Knoten, $\sigma \in V$

Begriff des Graphen. In der diskreten Mathematik werden Graphen verwendet, um Beziehungen zwischen gleichartigen Elementen eines Systems zu beschreiben. Formal repräsentiert ein Graph G eine Struktur aus einer Knotenmenge V und einer Kantenmenge E, welche die Knoten verbinden (siehe Deo 2004, S. 1 f, Diestel 2006, S. 2, Tutte 2001, S. 1 f). Entsprechend wird ein Graph zumeist als ein Paar (V, E) definiert. Ein typischer Anwendungsfall sind Graphen, in denen der kürzeste Weg zur Verbindung zweier festgelegter Knoten gesucht ist. Ähnlich der Fragestellung in dieser Arbeit werden Graphen weiterhin eingesetzt, um den maximalen Fluss zwischen zwei Knoten zu ermitteln, wobei die Kapazität der Kanten jeweils vorgegeben ist. Zu entsprechenden Verfahren zählen der Ford-Fulkerson-Algorithmus (siehe Elias et al. 1956, Ford und Fulkerson 1956, 1957) wie auch dessen zahlreiche Weiterentwicklungen (siehe u. a. Dinic 1970, Edmonds und Karp 1972, Goldberg und Tarjan 1988, Korte und Vygen 2012, Suhl und Mellouli 2009). Eine Adaption der noch vorzustellenden linearen Optimierung zur Maximierung des Flusses in einem Graphen ist der Netzwerksimplex (siehe Bazaraa et al. 2010, Hamacher und Klamroth 2006, Orlin 1997).

In der Darstellung werden für Knoten meist einfache Symbole gewählt, die um eine Bezeichnung des Knotens ergänzt werden. Eine Kante wird durch eine beliebig verlaufende



(a) Ungerichteter Graph ohne Mehrfachkanten (b) Gerichteter Graph ohne Mehrfachkanten (c) Ungerichteter Graph mit Mehrfachkanten (d) Gerichteter Graph mit Mehrfachkanten

Abbildung 4.1: Kategorisierung von unterschiedlichen Typen endlicher Graphen. Ein Graph repräsentiert eine Struktur, die aus einer Knotenmenge und einer Kantenmenge zusammengesetzt ist, wobei die Kanten die Knoten verbinden. In gerichteten Graphen ist im Unterschied zu ungerichteten entscheidend, in welcher Reihenfolge zwei Knoten durch eine Kante verbunden werden. In Graphen mit Mehrfachkanten existieren mehrere identische Kanten.

Linie zwischen den zu verbindenden Knoten repräsentiert, wobei die Linien je nach Graphentyp ggf. als Pfeile zur Anzeige einer Richtung dargestellt werden. Im Folgenden werden die allgemein üblichen Typen von Graphen unterschieden und deren Eigenschaften beschrieben. Für eine weitergehende Einführung zur Graphentheorie wird auf die Literatur verwiesen (siehe u. a. Deo 2004, Diestel 2006, Turau 2009, Werners 2013).

Je nach Einsatzzweck werden verschiedene Typen endlicher Graphen unterschieden, von denen die häufigsten in Abbildung 4.1 an einem einfachen Beispiel veranschaulicht werden. In diesem Zusammenhang bezieht sich der Begriff der Endlichkeit darauf, dass die Knotenmenge V und die Kantenmenge E eine endliche Zahl von Elementen enthalten (siehe Tutte 2001, S. 1 f). Die Graphentypen sind dadurch gekennzeichnet, welche Eigenschaften eine Kante zur Verbindung von Knoten besitzt und wie viele Knoten eine Kante jeweils verbindet. Somit können die Graphentypen durch die Definition der Kantenmenge E voneinander abgegrenzt werden, wie Tabelle 4.1 zeigt.

In gerichteten Graphen, auch als *Digraphen* bezeichnet, ist die Reihenfolge entscheidend, in welcher eine Kante von einem Knoten zum nächsten führt, wogegen in ungerichteten Graphen keine solche Reihenfolge definiert ist. Folglich repräsentiert eine Kante in gerichteten Graphen ein geordnetes Paar (v, v') zweier Knoten v und v'. Dagegen ist eine Kante in ungerichteten Graphen als eine Menge $\{v, v'\}$ darstellbar (siehe Turau 2009, S. 20, Diestel 2006, Werners 2013). Zudem können in Graphen mehrere identische Kanten, sogenannte Mehrfachkanten, existieren. In solchen Fällen wird auch von *Multigraphen* gesprochen (siehe Deo 2004, S. 2, Diestel 2006, S. 30, Illik 2009, S. 125, Turau 2009, S. 20). In *Hypergraphen* können Kanten mehr als zwei Knoten verbinden, weshalb sich eine Kante als eine Teilmenge der Knotenmenge V auffassen lässt (siehe Diestel 2006, S. 28).

Die Definition der Kantenmenge E in einem Graphen mit Mehrfachkanten ist somit zum einen davon abhängig, ob dessen Kanten gerichtet oder ungerichtet sind. Sofern Mehrfachkanten erlaubt sind, muss zum anderen berücksichtigt werden, ob diese als eigenständige mathematische Objekte unterschieden oder zusammengefasst werden sollen. Im einfachen Fall werden Mehrfachkanten nicht unterschieden, und es ist nur die Zahl der Kanten ent-

Graphentyp	Definition in natürlicher Sprache bzw. formaler Notation
(a) Ungerichtet, ohne Mehrfachkanten	Teilmenge aller zweielementigen Teilmengen der Knotenmenge V $E \subseteq \{\{v, v'\} \colon v, v' \in V\}$
(b) Gerichtet, ohne Mehrfachkanten	Teilmenge aller Paare aus dem kartesischen Produkt $V \times V$ der Knotenmenge V $E \subseteq V \times V$
(c) Ungerichtet, mit Mehrfachkanten*	Multimenge über der Menge aller zweielementigen Teilmengen der Knotenmenge V $E \colon \{\{v, v'\} \colon v, v' \in V\} \to \mathbb{N}_0$
(d) Gerichtet, mit Mehrfachkanten*	Multimenge über dem kartesischen Produkt $V \times V$ der Knotenmenge V $E \colon V \times V \to \mathbb{N}_0$
(e) Hypergraph	Teilmenge der Potenzmenge $\mathcal{P}(V)$ über der Knotenmenge V $E \subseteq \mathcal{P}(V)$

*Hinweis: Die Definition gilt für den Fall, dass Mehrfachkanten zusammengefasst werden.

Tabelle 4.1: Definitionen der Kantenmenge abhängig vom Graphentyp. Die in Abbildung 4.1 eingeführten Graphentypen unterscheiden sich im Hinblick darauf, wie die Kantenmenge definiert ist. Im ungerichteten Fall ist eine Kante eine Menge zweier Knoten, im gerichteten Fall ein geordnetes Paar. Wenn Mehrfachkanten erlaubt sind und nicht unterschieden werden, ist die Kantenmenge eine Abbildung solcher Mengen bzw. Paare auf die natürlichen Zahlen. In Hypergraphen ist jede Kante eine Teilmenge aller Knoten, da Kanten beliebig viele Knoten verbinden.

scheidend, die zwischen zwei Knoten existieren. Unter dieser Voraussetzung bezeichnet die Kantenmenge E eine Abbildung aller möglichen Kanten auf die natürlichen Zahlen. Alternativ wird jede Mehrfachkante als eigenes Objekt betrachtet. Um dies formal auszudrücken, müssen Hilfsfunktionen definiert werden, die jeweils den Startknoten und den Endknoten einer Kante zurückgeben (siehe Diestel 2006, S. 28).

Zur Speicherung von Graphen existieren verschiedene Möglichkeiten. Beispielsweise kann für jeden Knoten eine Adjazenzliste definiert werden, welche in ungerichteten Graphen die Nachbarn und in gerichteten Graphen die Nachfolger eines Knotens enthält. Eine weitere Möglichkeit ist die Speicherung in einer Adjazenzmatrix, deren Zeilen und Spalten jeweils einen Knoten repräsentieren. Wenn der Wert in einer Zeile und einer Spalte ungleich null ist, drückt dies aus, dass zwischen zwei Knoten eine Kante existiert. In einer Inzidenzmatrix entsprechen die Zeilen den Knoten und die Spalten den Kanten, wobei das Vorzeichen der Werte die Richtung angibt (siehe Cormen et al. 2001, S. 599 f, Illik 2009, S. 227 ff).

Traversierung, Quellen und Senken. Unter der *Traversierung* eines Graphen wird eine Suche verstanden, in deren Zuge alle Knoten und Kanten beschritten werden (siehe u. a. Cormen et al. 2010, Hochstättler 2010, Krumke und Noltemeier 2012, Nebel 2012, Turau 2009). Abhängig vom Startknoten und den Kanten in einem Graphen können nicht immer alle Knoten erreicht werden. In diesem Zusammenhang wird von Quellen und Senken gesprochen. Eine *Quelle* ist ein Knoten, zu dem keine Kanten führen, und eine *Senke* ein Knoten, an dem keine ausgehenden Kanten anliegen. Zur Definition der Begriffe ist es zweckmäßig, den Eingangsgrad $d_G^-(v)$ und den Ausgangsgrad $d_G^+(v)$ eines Knotens v einzuführen, die jeweils die Zahl eingehender bzw. ausgehender Kanten des Knotens bezeichnen. Auf dieser Basis lässt sich eine Quelle als ein Knoten mit einem Eingangsgrad von null und eine Senke analog als ein Knoten mit einem Ausgangsgrad von null definieren.

Die Information, welche Knoten und Kanten im Verlauf einer Suche beschritten werden, kann unterschiedlich repräsentiert werden. In der Literatur wird unter einem Kantenzug oft, aber nicht einheitlich eine alternierende Folge von Knoten und Kanten verstanden, die einen solchen Verlauf wiedergeben (siehe Diestel 2005, S. 10, Ray 2013, S. 14). Der Begriff wird in dieser Arbeit für eine einfache Folge von Kanten gebraucht, da die jeweiligen Knoten durch die Definition der Kanten eindeutig bestimmt sind. Ein Weg ist im Allgemeinen ein Kantenzug, in dem sich kein Knoten wiederholt (siehe Ray 2013, S. 15). Da in dieser Arbeit kein unendlicher Zyklus vorliegen muss, wenn derselbe Knoten wiederholt besucht wird, ist es zweckmäßig, von dieser Definition abzuweichen. Im Folgenden ist ein Weg daher ein Kantenzug, in welchem sich keine Kante wiederholt.

Breitensuche vs. Tiefensuche. Zur Suche in allgemeinen Graphen existieren zwei Verfahren, die nachfolgend beschrieben werden: die Breitensuche und die Tiefensuche. Die beiden Verfahren unterscheiden sich in Bezug auf die Reihenfolge, in welcher die Knoten V eines Graphen G entdeckt und besucht werden. Den Ausgangspunkt bildet jeweils ein allgemeiner Knoten σ aus der Knotenmenge V. Jedes dieser zwei Verfahren kann prinzipiell rekursiv oder nichtrekursiv implementiert werden. Die entsprechenden Implementierungen sind jeweils durch die Algorithmen A.1.1–A.1.4 im Anhang wiedergegeben.

Durch Ausführung einer *Breitensuche* werden immer erst alle Knoten besucht, die ausgehend vom aktuellen Knoten unmittelbar erreicht werden können (siehe Cormen et al. 2010, S. 603 ff, Krumke und Noltemeier 2012, S. 161). Die Bedingung bezieht sich darauf, dass jeweils eine Kante vom aktuellen Knoten zu ebendiesen Knoten existiert. Nachdem alle dieser Knoten besucht wurden, setzt die Suche mit deren Nachfolgern fort.

Entsprechend kann zur Speicherung der noch zu besuchenden Knoten eine Warteschlange verwendet werden. Eine Warteschlange bezeichnet eine linear verkettete Liste, die nach dem FIFO-Prinzip verwaltet wird. Das heißt, zuerst hinzugefügte Elemente werden zuerst zurückgegeben. Im Fall der Breitensuche kann die Datenstruktur nicht durch den rekursiven Aufruf einer Hilfsprozedur ersetzt werden, da die Verschachtelung der Aufrufe dem LIFO-Prinzip entspricht. Daher handelt es sich bei der rekursiven Implementierung der Breitensuche in Algorithmus A.1.2 nicht um eine echte rekursive Variante. Stattdessen wird die Schleife der nichtrekursiven Implementierung in Algorithmus A.1.1 durch den wiederholten Aufruf einer Hilfsprozedur ersetzt, wobei die Warteschlange weiterhin zur Verwaltung der zu besuchenden Knoten genutzt und als Argument übergeben wird.

Die *Tiefensuche* ist von der Breitensuche dadurch abzugrenzen, dass zunächst immer erst alle nachfolgenden Knoten besucht werden (siehe Cormen et al. 2010, S. 613 ff, Turau 2009, S. 94 ff, Krumke und Noltemeier 2012, S. 147 ff). Ausgehend vom aktuellen Knoten werden wieder alle unmittelbar erreichbaren Knoten ermittelt. Nachdem jeweils einer von diesen Knoten besucht wurde, werden all dessen Nachfolger besucht, bevor die Suche mit den verbleibenden unmittelbar erreichbaren Knoten fortsetzt.

In der nichtrekursiven Implementierung der Tiefensuche, wie sie in Algorithmus A.1.3 wiedergegeben ist, wird zur Speicherung der als Nächstes zu besuchenden Knoten ein Stapel verwendet. Ein Stapel ist eine linear verkettete Liste, die im Gegensatz zu einer Warteschlange nach dem LIFO-Prinzip verwaltet wird. Das heißt, zuerst werden die zuletzt

hinzugefügten Elemente zurückgegeben. In der rekursiven Implementierung kann auf eine Datenstruktur zur Verwaltung der Knoten verzichtet werden, da es durch die Verschachtelung der Prozeduraufrufe möglich ist, die Knoten in der erforderlichen Reihenfolge als Argumente zu übergeben. Die entsprechende Variante ist in Algorithmus A.1.4 wiedergegeben. In der Praxis wird zumeist die nichtrekursive Variante bevorzugt, da diese u. a. den wachsenden Speicherplatzbedarf des Prozedurstapels vermeidet.

Der Nachteil, dass eine Tiefensuche nicht terminiert, sofern stets ein nächster Knoten existiert, kann wie folgt eliminiert werden: In einem iterativen Prozess wird die Tiefensuche wiederholt vom selben Knoten ausgehend begonnen, wobei die maximale Tiefe in jedem Iterationsschritt wächst. Diese Variante wird als iterative Tiefensuche bezeichnet.

4.1.2 Komplexität von Algorithmen

Ein Algorithmus ist eine wohldefinierte Berechnungsvorschrift zur Umwandlung von Eingaben in Ausgaben, bestehend aus einer Folge von Rechenschritten (siehe Cormen et al. 2010, S. 5). Der Zweck eines Algorithmus ist die Lösung eines Rechenproblems, wofür dieser eine bestimmte Laufzeit und Speicherplatz benötigt. Um die Leistung von Algorithmen im Hinblick auf die letztgenannten Kriterien zu vergleichen und hierbei die Größe des Problems zu berücksichtigen, wird die Komplexität eines Algorithmus untersucht.

Zur Beschreibung der Laufzeit und des Speicherplatzbedarfs eines Algorithmus werden zwei reellwertige Funktionen definiert. Diese sind im Zusammenhang mit den Komplexitätsklassen relevant, die nachfolgend erläutert werden.

c	in diesem Kapitel verwendet als konstanter, positiver reeller Faktor, $c \in \mathbb{R}_{>0}$
n	Größe zu verarbeitender Eingabedaten eines Algorithmus, $n \in \mathbb{N}_{\geqslant 0}$
$f(n)$	asymptotische, obere oder untere Schranke für Laufzeit oder Speicherplatzbedarf, $f(n) \in \mathbb{R}_{\geqslant 0}$
$g(n)$	Funktion zur Beschreibung von Laufzeit oder Speicherplatzbedarf, $g(n) \in \mathbb{R}_{\geqslant 0}$

Komplexität als asymptotisches Maß. Eine unvermeidliche Einflussgröße bezogen auf die Laufzeit eines Algorithmus ist u. a. die Geschwindigkeit des genutzten Systems zur Ausführung der Rechenschritte. Der beanspruchte Speicherplatz ist neben den verwendeten Datenstrukturen von der Speicherverwaltung des ausführenden Systems abhängig. Durch die Beschreibung der Komplexität eines Algorithmus ist es möglich, diese Einflüsse zu eliminieren, indem die Laufzeit und der Speicherplatzbedarf nicht als absolute Werte gemessen werden. Stattdessen wird das asymptotische Wachstum ermittelt, das mit steigender Größe der Eingabedaten zu erwarten ist. Die Komplexität entspricht somit einem Grenzwert, welchen die Laufzeit bzw. der Speicherplatzbedarf bezogen auf die Größe der Eingabedaten nicht unterschreitet oder überschreitet (siehe Mehlhorn und Sanders 2008, S. 20).

Da es von der Fragestellung abhängig ist, ob eine obere oder untere Schranke gefordert ist, werden mehrere Komplexitätsklassen unterschieden, die in Tabelle 4.2 aufgeführt sind. Unter diesen ist die obere Schranke $\mathrm{O}(f(n))$ in der Praxis diejenige mit der größten Relevanz, weshalb auch von der O-Notation gesprochen wird (siehe u. a. Cormen et al. 2010, Mehlhorn und Sanders 2008, Ottmann und Widmayer 2012). Hintergrund ist, dass zumeist die Laufzeit oder der Speicherplatzbedarf im ungünstigsten Fall gefragt sind. Die als Argument übergebene Funktion $f(n)$ beschreibt das Wachstum der Schranke für eine Funktion $g(n)$, welche

Notation	Veranschaulichung	Formale Definition
$g(n) \in \mathrm{O}(f(n))$	$g(n)$ wächst *nicht schneller* als $f(n)$	$\mathrm{O}(f(n)) = \{g(n) : \exists c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geqslant n_0\ \ g(n) \leqslant c\, f(n)\}$
$g(n) \in \Omega(f(n))$	$g(n)$ wächst *nicht langsamer* als $f(n)$	$\Omega(f(n)) = \{g(n) : \exists c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geqslant n_0\ \ g(n) \geqslant c\, f(n)\}$
$g(n) \in \Theta(f(n))$	$g(n)$ wächst *genauso schnell* wie $f(n)$	$\Theta(f(n)) = \mathrm{O}(f(n)) \cap \Omega(f(n))$
$g(n) \in \mathrm{o}(f(n))$	$g(n)$ wächst *echt langsamer* als $f(n)$	$\mathrm{o}(f(n)) = \{g(n) : \forall c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geqslant n_0\ \ g(n) \leqslant c\, f(n)\}$
$g(n) \in \omega(f(n))$	$g(n)$ wächst *echt schneller* als $f(n)$	$\omega(f(n)) = \{g(n) : \forall c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geqslant n_0\ \ g(n) \geqslant c\, f(n)\}$

Tabelle 4.2: Definitionen unterschiedlicher Komplexitätsklassen. Die Komplexität eines Algorithmus bezieht sich i. d. R. auf die Laufzeit oder den Speicherplatzbedarf und beschreibt das asymptotische Wachstum abhängig von der Größe n der Eingabedaten. Hierfür werden die obigen Symbole verwendet, die jeweils unteren oder oberen Schranken entsprechen. Die Zugehörigkeit einer Funktion $g(n)$ zu einer Komplexitätsklasse drückt aus, dass diese Funktion innerhalb der Schranke wächst, welche durch die Funktion $f(n)$ definiert ist (siehe Mehlhorn und Sanders 2008, S. 20).

der Komplexitätsklasse $\mathrm{O}(f(n))$ zugeordnet wird. In dem Fall, dass die Laufzeit mit dem Quadrat der Eingabedaten wächst und eine entsprechende Schranke nicht überschreitet, gehört die Laufzeitfunktion $g(n)$ zur Komplexitätsklasse $\mathrm{O}(n^2)$. Ein solches Wachstum der Laufzeit ist für einfache Sortierverfahren typisch. Weitere Beispiele für Komplexitätsklassen sind $\mathrm{O}(\log n)$, $\mathrm{O}(n)$, $\mathrm{O}(n \log n)$ und $\mathrm{O}(2^n)$, die jeweils ein logarithmisches, lineares, quasilineares bzw. exponentielles Wachstum beschreiben. Ein Algorithmus mit logarithmischem Wachstum der Laufzeit ist z. B. die binäre Suche. Fortgeschrittene Sortierverfahren wie z. B. *Mergesort* besitzen ein quasilineares Wachstum der Laufzeit.

Bei der Bestimmung der Komplexitätsklasse eines Algorithmus ist zu beachten, dass konstante Faktoren vernachlässigt werden. Das heißt, eine Funktion $g(n)$, die zur Klasse $\mathrm{O}(c\,n)$ mit einem konstanten Faktor c größer als null gehört, ist stets auch der Klasse $\mathrm{O}(n)$ zugehörig. Des Weiteren wird im Zusammenhang mit einer oberen Schranke $\mathrm{O}(f(n))$ in Summen immer der Summand mit dem höchsten Grad betrachtet. Daneben existieren weitere Regeln, die in der Literatur beschrieben sind (siehe u. a. Mehlhorn und Sanders 2008).

Komplexität von Breitensuche und Tiefensuche. Sowohl Breitensuche als auch Tiefensuche besitzen eine Laufzeit von $\mathrm{O}(|V| + |E|)$ und einen Speicherplatzbedarf von $\mathrm{O}(|V|)$, wobei $|V|$ die Zahl der Knoten und $|E|$ die Zahl der Kanten bezeichnet (siehe Cormen et al. 2010, S. 603 ff, 613 ff, Hochstättler 2010, S. 59). Der Grund dafür ist, dass im ungünstigsten Fall alle Knoten V und Kanten E des gegebenen Graphen G besucht werden müssen. Dabei wird vorausgesetzt, dass zur Bestimmung nachfolgender Knoten und Kanten eine Datenstruktur eingesetzt wird, die einen Zugriff innerhalb einer Laufzeit von $\mathrm{O}(1)$ ermöglicht (z. B. eine Adjazenzmatrix). Der Speicherplatzbedarf schließt die Speicherung des Graphen selbst nicht mit ein, da diese von der gewählten Datenstruktur abhängig ist.

Beim Vergleich beider Suchverfahren ist zudem zu berücksichtigen, dass die Tiefensuche nicht vollständig ist, falls der Graph unendlich oder zyklisch ist. In einem solchen Graphen werden nicht alle Knoten besucht, selbst wenn es möglich ist, diese ausgehend vom Startknoten zu erreichen. Nur durch die iterative Tiefensuche ist sichergestellt, dass jeder erreichbare Knoten besucht wird. Die Breitensuche ist dagegen stets vollständig, sofern jeder Knoten

eine endliche Zahl von Nachfolgern besitzt und der Zielknoten erreichbar ist. Das gilt selbst dann, wenn der zu durchsuchende Graph unendlich ist.

Ein weiterer Faktor beim Vergleich der beiden Suchverfahren ist deren Eignung, die optimale Lösung zu ermitteln. Darunter wird verstanden, dass der gefundene Weg von einem Startknoten zu einem Zielknoten dem jeweils kürzestmöglichen Weg entspricht. Die Breitensuche ist optimal, sofern die Kanten keine Gewichte tragen. Demgegenüber garantiert die Tiefensuche nicht, den kürzesten Weg zu einem Knoten zu ermitteln.

In Bezug auf die Wertstromgraphen, die zur Modellierung von Systemen in den folgenden Abschnitten beschrieben werden, sind diese Punkte nicht relevant. Hintergrund ist, dass die Wertstromgraphen vollständig traversiert werden müssen und somit stets alle Knoten und alle Kanten zu besuchen sind. Einziges Kriterium zur Unterscheidung der Suchverfahren ist in diesem Kontext die Reihenfolge, in welcher die Knoten und Kanten besucht werden. Im Hinblick darauf wird in der vorliegenden Arbeit die Entscheidung für die Tiefensuche getroffen, da immer erst ein vollständiger Weg vom aktuellen Knoten bis zur Senke gesucht werden muss, bevor andere Wege fortgesetzt werden.

4.2 Modellierung eines Systems durch Wertstromgraphen

Um Wertströme in der Produktion auf eine Weise abzubilden, die zur strategischen Planung der TEK geeignet ist, muss der Begriff des Graphen aus der Literatur erweitert werden. Die Grundlagen aus dem ersten Abschnitt, die sich auf die formale Definition eines Graphen auf Basis einer Knotenmenge und einer Kantenmenge beziehen, sowie die grundsätzlichen Suchverfahren finden weiterhin Anwendung. Jedoch müssen zum einen spezifische Typen von Knoten unterschieden werden, um Ressourcen und Verzweigungen in den Wertströmen abzubilden. Zum anderen muss berücksichtigt werden, ob sich Kanten an einem Knoten vereinigen oder der weitere Weg nach dem Knoten je nach Anwendungsfall getrennt verläuft. Diese Erweiterungen des Standardbegriffs eines Graphen sind durch die Anforderungen aus Abschnitt 2.6 begründet, wie im Folgenden erläutert wird.

Zur formalen Beschreibung dieser Konzepte werden die folgenden Symbole eingeführt oder erweitert. Daneben wird auf die bereits eingeführten Symbole aus den vorherigen Kapiteln verwiesen, wie z. B. auf die Menge R der verfügbaren und zu installierenden MAEs, die in Abschnitt 2.4 eingeführt wurde.

b, b'	Index zur Bezeichnung eines Produkts, $b, b' \in \{1, \ldots, n\}$, ggf. mit Index 1, 2 usw.
$D_{\mathrm{AL},b}$, $D_{\mathrm{SL},b}$	Menge aller alternativen bzw. selektiven Flusspunkte von $G_{\mathrm{WS},b}$, $D_{\mathrm{AL},b}$ bzw. $D_{\mathrm{SL},b} \subset V_{\mathrm{WS},b}$
$d^{+}_{\max}$	maximaler Ausgangsgrad aller Flusspunkte $D_{\mathrm{AL},b}$ und $D_{\mathrm{SL},b}$
e, e'	aktuell bzw. als Nächstes zu besuchende Kante, $e, e' \in E_{\mathrm{WS},b}$, ggf. mit Index 1, 2 usw.
$E_{\mathrm{WS},b}$	Kantenmenge von $G_{\mathrm{WS},b}$, $E_{\mathrm{WS},b} \subset \{(v, j, v', i') : v, v' \in V_{\mathrm{WS},b} \wedge j, i' \in \mathbb{N}\}$
$G_{\mathrm{WS},b}$	Wertstromgraph des Produkts b, definiert als Paar $G_{\mathrm{WS},b} = (V_{\mathrm{WS},b}, E_{\mathrm{WS},b})$
$\{G_{\mathrm{WS},b}\}$	grafisches Modell eines Systems von Wertströmen, Abkürzung für $\{G_{\mathrm{WS},b} : b \in \{1, \ldots, n\}\}$
i, i'	natürliche Zahl, Eingang eines Knotens v bzw. v', $i, i' \in \mathbb{N}$, ggf. mit Index 1, 2 usw.
j, j'	s. o., Ausgang eines Knotens v bzw. v', $j, j' \in \mathbb{N}$, ggf. mit Index 1, 2 usw.
m, n	Zahl der Prozessstückzahlen x_a bzw. der Produktstückzahlen y_b, $m, n \in \mathbb{N}$
v, v'	aktueller bzw. nächster Knoten, im weiteren Verlauf $v, v' \in V_{\mathrm{WS},b}$, ggf. mit Index 1, 2 usw.

$V_{\mathrm{WS},b}$	Knotenmenge von $G_{\mathrm{WS},b}$, $V_{\mathrm{WS},b} = R \cup D_{\mathrm{AL},b} \cup D_{\mathrm{SL},b} \cup \{\sigma_b, \tau_b\}$
σ_b, τ_b	Quelle bzw. Senke, kennzeichnen Beginn bzw. Ende eines Wertstroms, $\sigma_b, \tau_b \in V_{\mathrm{WS},b}$

Ein grafisches Modell spiegelt in seiner Struktur das abzubildende System der gegebenen Wertströme wider. Zunächst soll diese übergeordnete Modellstruktur beschrieben werden, bevor die Eigenschaften der Modellelemente folgen.

4.2.1 Grafische Modellstruktur: Knoten und Kanten

Mit Hilfe der Software AURELIE bildet der Anwender im vorgegebenen Planungszeitraum jeweils einen Wertstrom für jedes Produkt ab. Falls sich Prozesse verändern, ist es möglich, den Planungszeitraum zu teilen, wodurch die eindeutige Beziehung zwischen Produkten und Wertströmen erhalten bleibt. Entsprechend ist das erstellte grafische Modell $\{G_{\mathrm{WS},b}\}$ formal als Menge aller Wertstromgraphen $G_{\mathrm{WS},b}$ abbildbar, wobei jeweils ein Wertstromgraph für jedes Produkt b definiert ist. Analog zum Standardbegriff ist ein Wertstromgraph aus einer Knotenmenge $V_{\mathrm{WS},b}$ und einer Kantenmenge $E_{\mathrm{WS},b}$ zusammengesetzt.

Die nachfolgenden Vereinbarungen in Bezug auf das grafische Modell sind das Ergebnis ausführlicher Tests mit Pilotanwendern an Standorten der Bosch Rexroth AG. Die Teilnehmer haben durchgängig bestätigt, dass die Darstellung der Modellelemente und der Beziehungen zwischen diesen verständlich und intuitiv ist.

Knotenmenge $V_{\mathrm{WS},b}$. Aus dem Index in der formalen Bezeichnung der Knotenmenge $V_{\mathrm{WS},b}$ geht hervor, dass die Menge von einem gegebenen Produkt b abhängig ist. Hintergrund ist, dass spezifische Knoten der Abbildung des Materialflusses dienen und sich dieser je nach Produkt unterscheidet. Zu solchen Knoten zählen die Quelle σ_b und die Senke τ_b zur Kennzeichnung des Beginns bzw. des Endes eines Wertstroms sowie die alternativen und die selektiven Flusspunkte $D_{\mathrm{AL},b}$ bzw. $D_{\mathrm{SL},b}$ für entsprechende Verknüpfungen. Diese Knoten sind der Knotenmenge $V_{\mathrm{WS},b}$ genau eines Produkts b zugehörig.

Ressourcen bilden verfügbare und zu installierende MAEs ab, die von verschiedenen, möglicherweise allen Produkten genutzt werden. Aus diesem Grund ist die Menge R der Ressourcen nicht genau einem Produkt zugeordnet, sondern Teilmenge der Knotenmenge $V_{\mathrm{WS},b}$ jedes einzelnen Produkts. Die Knotenmenge $V_{\mathrm{WS},b}$ umfasst somit neben den produktspezifischen Mengen alle Knoten der Menge R. Auf diese Weise wird die Basisanforderung A1 umgesetzt, die besagt, dass das Modell die Wertströme aller Produkte unter Berücksichtigung der Ressourcenzuordnung in sich vereinigen soll.

Zusammenfassend ist die Knotenmenge $V_{\mathrm{WS},b}$ durch die Vereinigung der spezifischen Mengen des gegebenen Produkts b und der Ressourcen R definiert, wie eingangs in der Liste der Symbole angegeben. Die Knotentypen sind in Abbildung 4.2 aufgeführt und werden erläutert, nachdem der Begriff der Kante in diesem Kontext konkretisiert wurde.

Kantenmenge $E_{\mathrm{WS},b}$. In der Kurzeinführung zu diesem Kapitel wurde dargelegt, dass sich Graphentypen anhand der Definition der Kantenmenge unterscheiden lassen. Da in dieser Arbeit der Begriff des Wertstromgraphen $G_{\mathrm{WS},b}$ geprägt wird, muss auch eine entsprechende Definition der Kantenmenge $E_{\mathrm{WS},b}$ eines Produkts folgen. Der Index in der Bezeichnung der

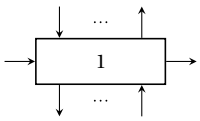
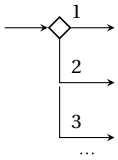
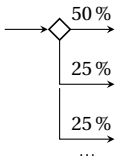
Formale Bezeichnung	Grafisches Symbol	Bedeutung und Eigenschaften
(a) Quelle σ_b		Beginn des Wertstroms eines Produkts. Besitzt keinen Eingang und genau einen Ausgang.
(b) Senke τ_b		Ende des Wertstroms eines Produkts. Besitzt im Gegensatz zur Quelle genau einen Eingang und keinen Ausgang.
(c) Ressourcen R		Maschinen, Anlagen und Einrichtungen. Besitzen beliebig viele Eingänge, denen jeweils der gegenüberliegende Ausgang zugeordnet ist. Für jede Ressource ist im Planungsintervall eine Betriebsmittelzeit T_{BM} definiert. Jeder Eingang entspricht einem Prozessschritt mit einer effektiven Taktzeit t_{eff}.
(d) Alternative Flusspunkte $D_{AL,b}$		Alternative Verknüpfungen von Prozessschritten und ggf. verketteten Verknüpfungen im Wertstrom eines Produkts. Besitzen jeweils genau einen Eingang und beliebig viele Ausgänge. Die Ausgänge sind nach ihrer Priorität sortiert.
(e) Selektive Flusspunkte $D_{SL,b}$		Selektive Verknüpfungen im Wertstrom eines Produkts. Besitzen jeweils genau einen Eingang und beliebig viele Ausgänge. Jedem Ausgang ist eine fixe Quote größer als null und kleiner als oder gleich 100 Prozent zugeordnet.

Abbildung 4.2: Bezeichnungen, Symbole und Bedeutungen der Knotentypen. Dargestellt sind die eingeführten Knotentypen zur grafischen Modellierung von Wertströmen. Quellen und Senken existieren jeweils genau einmal in jedem Wertstrom. Ressourcen können in Wertströmen verschiedener Produkte verknüpft werden, wobei an zueinander gehörigen Eingängen und Ausgängen Kanten aus demselben Wertstrom anliegen müssen. Flusspunkte sind dem Wertstrom genau eines Produkts zugeordnet und verbinden nur Kanten in diesem Wertstrom.

Menge weist darauf hin, dass sich die Menge je nach Produkt unterscheidet. Der Grund ist wie im Fall der Knotenmenge $V_{WS,b}$, dass Kanten den Materialfluss eines Produkts abbilden und jeweils nur ein Produkt b betreffen. Folglich müssen sowohl die Knotenmenge $V_{WS,b}$ als auch die Kantenmenge $E_{WS,b}$ für jedes Produkt b definiert werden.

Ein Wertstromgraph ist im Verständnis dieser Arbeit zunächst ein endlicher, gerichteter Graph mit Mehrfachkanten. Da entscheidend ist, welche Ressourcen nacheinander durchlaufen werden, muss in Bezug auf Kanten zwischen zwei Knoten berücksichtigt werden, in welcher Reihenfolge sie diese Knoten verbinden. Weiter müssen Mehrfachkanten erlaubt sein, da sich Kanten, die zu einem Knoten führen, nicht notwendigerweise an diesem Punkt vereinigen müssen. Stattdessen werden sie je nach Anwendungsfall weiter unterschieden, wie nachfolgend erläutert werden soll.

Kanten führen von einem Knoten zum nächsten und bilden auf diese Weise sequenzielle Verknüpfungen ab. Zudem werden Kanten genutzt, um zur Erfüllung der Basisanforderung A2 zwei Anwendungsfälle zu unterscheiden: Erstens können sich zwei Prozesse an einer MAE vereinigen, sodass derselbe Prozessschritt mit derselben Taktzeit ausgeführt wird und sich alle nachfolgenden Prozessschritte gleichen. Zweitens muss der Anwendungsfall

abgebildet werden können, dass die Prozesse nicht vereinigt werden, um verschiedene Prozessschritte mit unterschiedlichen Taktzeiten und/oder nachfolgenden Prozessschritten auszuführen. Daraus folgt, dass die Information, von welchem Knoten eine Kante zum nächsten Knoten führte, auch weiterhin wichtig ist, nachdem der nächste Knoten beschritten wurde. Nur unter Berücksichtigung dieser Information kann abhängig vom vorhergehenden Knoten eine jeweils andere nachfolgende Kante gewählt werden.

Um dies umzusetzen, wird das Konzept eines Eingangs und eines Ausgangs an einem Knoten eingeführt. Diese Eingänge und Ausgänge werden jeweils durch einen natürlichen Index i bzw. j repräsentiert und bezeichnen den Punkt, an dem eine Kante in einen Knoten mündet bzw. einen Knoten verlässt. Indem zwei Kanten zum selben Eingang an einer Ressource führen, wird derselbe Prozessschritt ausgeführt, und der weitere Weg im Wertstrom unterscheidet sich nicht. Führen die Kanten zu verschiedenen Eingängen, werden diese nicht vereinigt, und der Weg unterscheidet sich nach Verlassen der Ressource. Eine Kante e ist somit nicht als ein geordnetes Paar (v, v') zweier Knoten definiert, wie dies in allgemeinen gerichteten Graphen der Fall ist. Stattdessen wird den Knoten jeweils die Information über den Ausgang j bzw. Eingang i' hinzugefügt, sodass eine Kante e durch ein Tupel (v, j, v', i') repräsentiert werden kann. Die Definition besagt, dass die gegebene Kante e den Knoten v am Ausgang j verlässt und zum Eingang i' des nächsten Knotens v' führt.

Hinzuzufügen ist, dass beliebig viele Kanten zu einem Eingang an einem Knoten führen können, worauf sie an diesem Punkt vereinigt werden. An einem Ausgang eines Knotens kann jedoch nur eine ausgehende Kante anliegen. Demzufolge entspricht jede ausgehende Kante an einem Knoten einem eigenen Ausgang. Die Zuordnung von Ausgängen zu Eingängen und damit von ausgehenden zu eingehenden Kanten an diesem Knoten wird durch den Knotentyp bestimmt, wie nachfolgend beschrieben wird.

4.2.2 Modellelemente: Quellen, Senken, Ressourcen und Flusspunkte

Die Menge $V_{\mathrm{WS},b}$ der Knoten eines Wertstromgraphen $G_{\mathrm{WS},b}$ repräsentiert den Materialfluss eines Produkts. Dies schließt alle Betriebsmittel ein, die im Ablauf der Fertigung und Montage benötigt werden. Zudem ist für jeden Knoten eine eindeutige Zuordnung der Ausgänge zu den Eingängen definiert, welche der Anwender durch die Vorgabe von Prioritäten und Quoten beeinflussen kann. Auf diese Weise bestimmen die Knoten nicht nur den Materialfluss, sondern auch die Stückzahlverteilung und damit den Informationsfluss (siehe Erläuterung zum Begriff des Wertstroms in der Systemanalyse, Abschnitt 2.2).

Quellen σ_b und Senken τ_b. Allgemein ist eine Quelle als ein Knoten definiert, dessen Eingangsgrad gleich null ist. Dementsprechend ist die Quelle σ_b eines Wertstromgraphen $G_{\mathrm{WS},b}$ im Kontext dieser Arbeit der einzige Knoten aus der Menge $V_{\mathrm{WS},b}$, an dem keine eingehende Kante anliegen darf. Ihr Gegenstück ist die Senke τ_b, die ebenfalls entsprechend ihrer allgemeinen Definition einen Ausgangsgrad von null besitzt. Das heißt, an der Senke τ_b darf analog zur Quelle σ_b keine ausgehende Kante anliegen.

Die Symbole der beiden Knotentypen sind in Abbildung 4.2(a) bzw. 4.2(b) dargestellt. Mit Blick auf die Bedeutung der Knoten repräsentiert die Quelle σ_b den Beginn eines Wertstroms,

die Senke τ_b analog dazu dessen Ende. Entsprechend enthält die Knotenmenge $V_{\mathrm{WS},b}$ eines Wertstromgraphen $G_{\mathrm{WS},b}$ genau eine Quelle und eine Senke. In Bezug auf den Materialfluss bilden die beiden Knoten die Schnittstelle des Systems mit der Umwelt, wobei Komponenten an der Quelle aufgenommen und Produkte an der Senke bereitgestellt werden.

Ein Unterschied in der Definition der Knotentypen bezieht sich auf die Zahl der jeweils ausgehenden und eingehenden Kanten von Quellen bzw. Senken. Da der Weg in einem Wertstromgraphen $G_{\mathrm{WS},b}$ beendet ist, sobald die Senke τ_b erreicht wird, ist es unerheblich, welche Kante zu dieser Senke führte. Es sind daher mehrere eingehende Kanten an einer Senke σ_b erlaubt, und es wird lediglich gefordert, dass diese in denselben Eingang münden. Dagegen darf nur eine einzige Kante am Ausgang einer Quelle σ_b anliegen, da andernfalls der Weg zum ersten Knoten nicht eindeutig bestimmt ist.

Ressourcen *R*. Wie bereits angesprochen, repräsentieren Ressourcen MAEs, die von allen Produkten genutzt werden können, weshalb sie als eine Teilmenge zur Knotenmenge $V_{\mathrm{WS},b}$ jedes Produkts b gehören. Betrachtet man in einem Wertstromgraphen $G_{\mathrm{WS},b}$ eine Kante, die zu einem Eingang an einer Ressource führt, dann entspricht dieser Eingang einem Prozessschritt, welchem die jeweilige MAE zugeordnet ist. Daraus folgt, dass sich der Begriff des Eingangs an einem Knoten auch im Fall von Ressourcen stets auf genau einen Wertstromgraphen $G_{\mathrm{WS},b}$ bezieht. An einem Eingang i einer Ressource können in verschiedenen Wertstromgraphen $G_{\mathrm{WS},b}$ unterschiedliche Kanten anliegen, und entsprechend repräsentieren die Eingänge unterschiedliche Prozessschritte in den Wertströmen.

Einem gegebenen Eingang i einer Ressource ist eindeutig genau ein Ausgang j zugeordnet. Hierbei handelt es sich um denjenigen Ausgang, dessen Index j dem Index i des Eingangs entspricht. In der visuellen Darstellung ist dies der Ausgang, welcher dem jeweiligen Eingang genau gegenüberliegt, wie in Abbildung 4.2(c) dargestellt ist. Sofern das grafische Modell $\{G_{\mathrm{WS},b}\}$ gültig ist, besitzt eine Ressource in jedem Wertstromgraphen $G_{\mathrm{WS},b}$ als Folge die gleiche Zahl von Eingängen und Ausgängen. Durch die eindeutige, unveränderliche Zuordnung jeweils eines Ausgangs zu jedem Eingang einer Ressource kann unterschieden werden, welche Kante nach Verlassen der Ressource weiterverfolgt wird. Es ist stets die Kante am Ausgang der Ressource gegenüber von demjenigen Eingang, zu welchem die vorhergehende Kante führte. Abbildung 4.3 veranschaulicht das Konzept anhand der Beispiele sequenzieller Verknüpfungen aus Abschnitt 2.5.

Mit Hilfe dieser Festlegung und der entsprechenden Definition des grafischen Modells ist es möglich, die beschriebenen Anwendungsfälle der Basisanforderung A2 grafisch auf intuitive Weise abzubilden. Insbesondere kann die Zuordnung einer MAE zu einem Prozessschritt zeichnerisch wiedergegeben werden. Darüber hinaus wird durch die grafische Darstellung deutlich, dass eine Ressource für verschiedene Prozessschritte eines Produkts genutzt wird und welche nachfolgenden Prozessschritte ausgeführt werden.

Bemerkung: Die obigen Erläuterungen beziehen sich darauf, wie der Anwender durch Verknüpfung von Ressourcen den Materialfluss eines Produkts abbildet. Daneben bietet die Benutzeroberfläche der Software AURELIE dem Anwender die Möglichkeit, für alle Ressourcen eine Betriebsmittelzeit $\mathrm{T_{BM}}$ und für jeden Eingang an einer Ressource in einem Wertstromgraphen $G_{\mathrm{WS},b}$ eine effektive Taktzeit $\mathrm{t_{eff}}$ vorzugeben. Damit liegen alle notwen-

digen Informationen vor, um im langfristigen Mittel die Auslastung der jeweiligen MAE zu kalkulieren, wie es die Basisanforderung A3 verlangt.

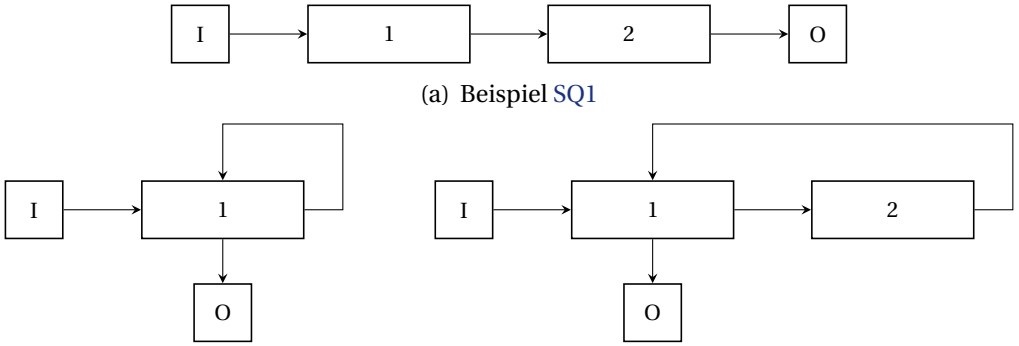
Alternative und selektive Flusspunkte $D_{\mathrm{AL},b}$ bzw. $D_{\mathrm{SL},b}$. Flusspunkte entsprechen im Gegensatz zu Ressourcen nicht physischen Objekten, sondern stellen Verzweigungen in einem Wertstrom dar. Zusammen mit Kanten, die sequenzielle Verknüpfungen abbilden, werden alternative und selektive Flusspunkte genutzt, um alternative bzw. selektive Verknüpfungen zu modellieren. Hierbei entsprechen die Knoten, zu denen die ausgehenden Kanten an einem Flusspunkt führen, den jeweils zu verknüpfenden Operanden.

Um dieses Konzept zu veranschaulichen: Wenn an einem alternativen Flusspunkt eine bestimmte Zahl ausgehender Kanten anliegt und die Kanten zu Ressourcen führen, dann wird auf diese Weise eine alternative Verknüpfung von Prozessschritten abgebildet. Die verknüpften Prozessschritte werden, wie zuvor erläutert, durch die Eingänge an den jeweiligen Ressourcen repräsentiert. Das heißt, allein durch die Verknüpfung der Knoten mit Hilfe von Kanten definiert der Anwender, dass Prozessschritte in einem Wertstrom alternativ bzw. selektiv miteinander verbunden sein sollen.

Da alternative und selektive Flusspunkte unterschiedliche Verknüpfungstypen repräsentieren, muss der Anwender jeweils spezifische Informationen an den ausgehenden Kanten hinzufügen. Im Fall alternativer Flusspunkte sind dies die Prioritäten der nachfolgenden Prozessschritte, wie in Abbildung 4.2(d) dargestellt. Durch diese Priorisierung ist es möglich, für die Herstellung geplanter Stückzahlen ein eindeutiges Ergebnis in Bezug auf die Auslastung der genutzten MAEs zu bestimmen. Gleichzeitig trägt die Priorisierung eine weitere Information: Folgt man an jedem alternativen Flusspunkt stets nur der ersten ausgehenden Kante, dann beschreibt der Weg die Prozessschritte, die zu einer Überlastung der MAEs führen dürfen, sofern die Kapazität nicht zur Herstellung der geplanten Produktstückzahlen ausreicht. Dadurch kann der Anwender festlegen, welche MAEs im Bedarfsfall überlastet werden sollen. Um die Nutzung alternativer Flusspunkte zu illustrieren, sind die Beispiele alternativer Verknüpfungen aus Abschnitt 2.5 in Abbildung 4.4 und 4.5 dargestellt. Abbildung 4.6 zeigt zusätzlich ein komplexeres Beispiel eines Wertstroms.

Im Fall selektiver Flusspunkte definiert der Anwender an jeder ausgehenden Kante einen positiven Wert von höchstens 100 Prozent, wie Abbildung 4.2(e) zeigt. Diese fix vorgegebene Quote entspricht dem relativen Anteil der Stückzahl, welche den Flusspunkt am jeweiligen Ausgang verlässt, im Verhältnis zur Stückzahl, die an dessen Eingang anliegt. Abbildung 4.7 veranschaulicht, wie die Beispiele selektiver Verknüpfungen aus Abschnitt 2.5 in die grafische Notation zur Modellierung von Wertströmen übertragen werden. Abbildung 4.8 stellt ein komplexeres Beispiel eines Wertstroms aus der Praxis dar, in welchem Flusspunkte verschiedener Typen miteinander verkettet werden.

Bemerkung: Die Definitionen 2.6 und 2.7 der alternativen bzw. selektiven Verknüpfung aus Abschnitt 2.2 beziehen sich jeweils auf genau zwei Operanden. Zwar wäre es theoretisch möglich, mehrere Flusspunkte mit jeweils zwei Ausgängen nacheinander zu verketten und auf diese Weise Verknüpfungen mit einer größeren Zahl von Operanden abzubilden. Um aber den Aufwand zur grafischen Modellierung zu reduzieren, ist eine beliebige, aber endliche

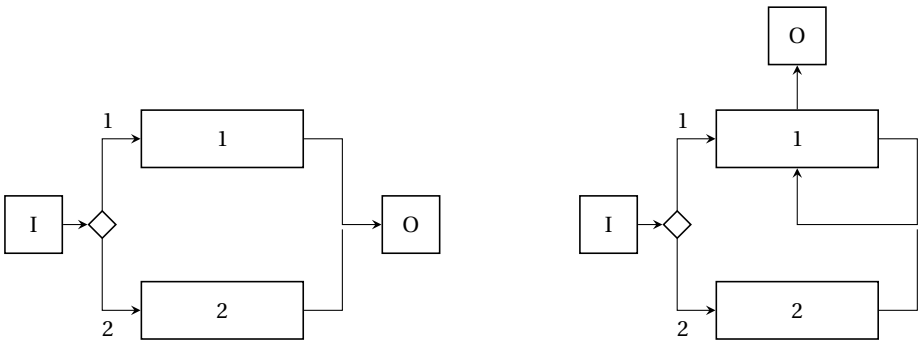


(a) Beispiel SQ1

(b) Beispiel SQ2

(c) Beispiel SQ3

Abbildung 4.3: Modellierung sequenzieller Verknüpfungen (Beispiel SQ1, SQ2 und SQ3). Die Abbildungen zeigen die Übertragung der Beispiele SQ1, SQ2 und SQ3 in die Notation zur grafischen Modellierung von Wertströmen. Das zweite und dritte Beispiel illustrieren, wie Eingängen an Ressourcen der jeweils gegenüberliegende Ausgang zugeordnet wird. Dadurch ist es im vorliegenden Fall möglich, den weiteren Ablauf abhängig vom gewählten Eingang festzulegen.



(a) Beispiel AL1

(b) Beispiel AL2

Abbildung 4.4: Modellierung alternativer Verknüpfungen (Beispiel AL1 und AL2). Die Abbildungen geben die Beispiele AL1 und AL2 in der eingeführten Notation wieder. Zur grafischen Modellierung der Verknüpfungen wird jeweils ein alternativer Flusspunkt genutzt. Die zugehörigen Prioritäten der verknüpften Prozessschritte sind an den ausgehenden Kanten ergänzt. In Beispiel AL2 werden analog zu den Beispielen sequenzieller Verknüpfungen verschiedene Eingänge an Ressourcen verbunden, um die nachfolgenden Prozessschritte fallweise zu unterscheiden.

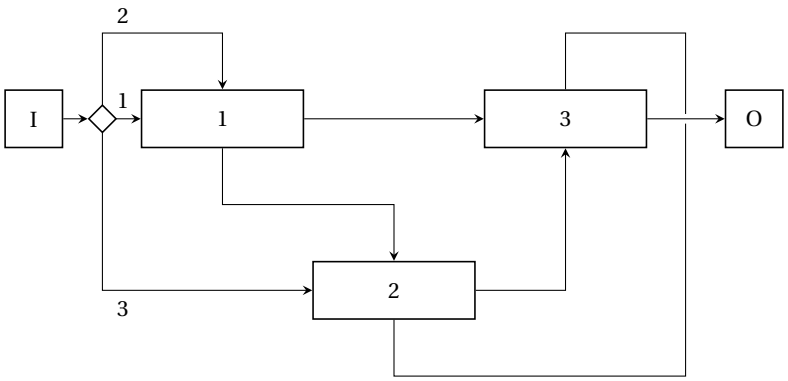


Abbildung 4.5: Modellierung einer alternativen Verknüpfung (Beispiel AL3). Die Abbildung verdeutlicht die Einfachheit, mit welcher komplexere Wertströme wie jener aus Beispiel AL3 grafisch modelliert werden können. Im vorliegenden Fall wird hierzu ein alternativer Flusspunkt mit drei ausgehenden Kanten definiert. Zudem führen die anliegenden Kanten an jeder Ressource jeweils zu zwei unterschiedlichen Eingängen. Abhängig vom gewählten Eingang der ersten Ressource folgt der zweite Prozessschritt unter Nutzung einer anderen Ressource.

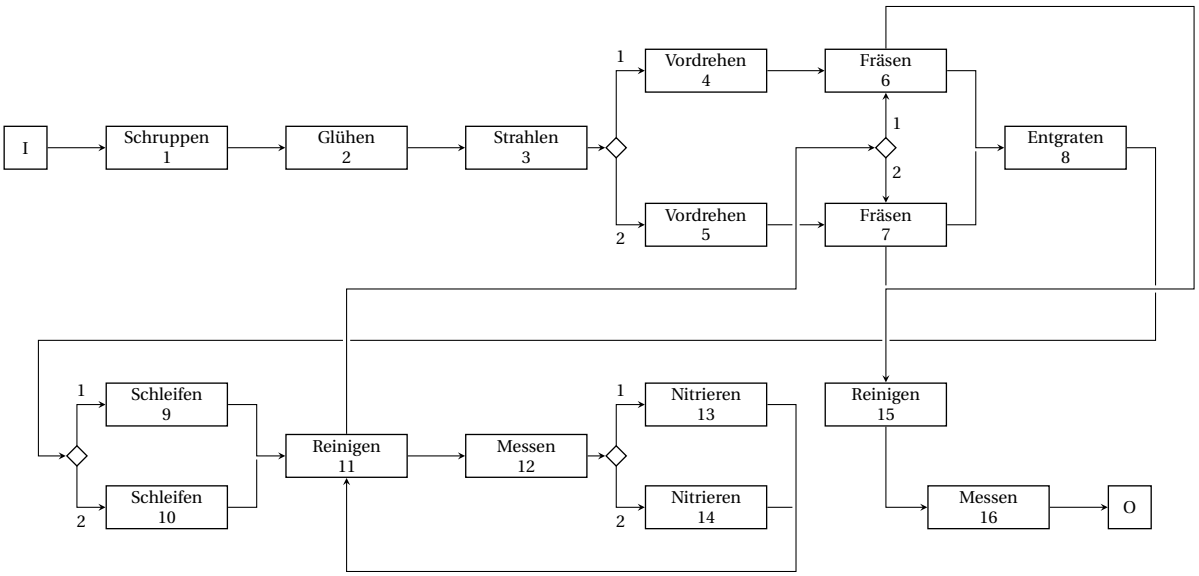
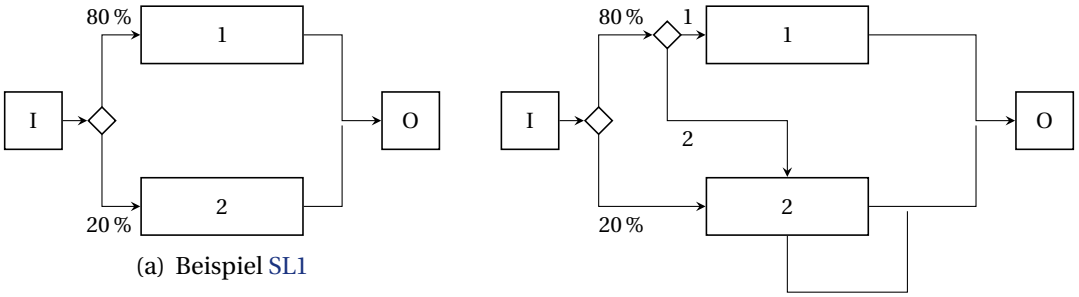


Abbildung 4.6: Wertstromgraph für die Fertigung einer Komponente (Praxisbeispiel). Die Modellierung eines realen Wertstroms, wie ihn die Abbildung zeigt, erfordert eine größere Zahl von Knoten und Kanten. Grundlage ist der Ablauf zur Fertigung einer Komponente für ein Produkt an einem Standort der Bosch Rexroth AG. Das Beispiel illustriert einige Möglichkeiten der eingeführten Notation zur Kombination von Elementen. Hierzu zählen die Verkettung von sequenziellen und alternativen Verknüpfungen sowie die Unterscheidung von Eingängen an Ressourcen.



(a) Beispiel SL1

(b) Beispiel SL2

Abbildung 4.7: Modellierung selektiver Verknüpfungen (Beispiel SL1 und SL2). Wie die Abbildungen verdeutlichen, werden selektive Verknüpfungen analog zu alternativen Verknüpfungen durch entsprechende Flusspunkte modelliert. In Beispiel SL2 werden darüber hinaus zwei Verknüpfungen miteinander verkettet. Entsprechend führt eine ausgehende Kante des ersten Flusspunkts zum Eingang des zweiten Flusspunkts. Durch die Nutzung verschiedener Eingänge an der zweiten Ressource ist es möglich, unterschiedliche Taktzeiten je Prozessschritt zu definieren.

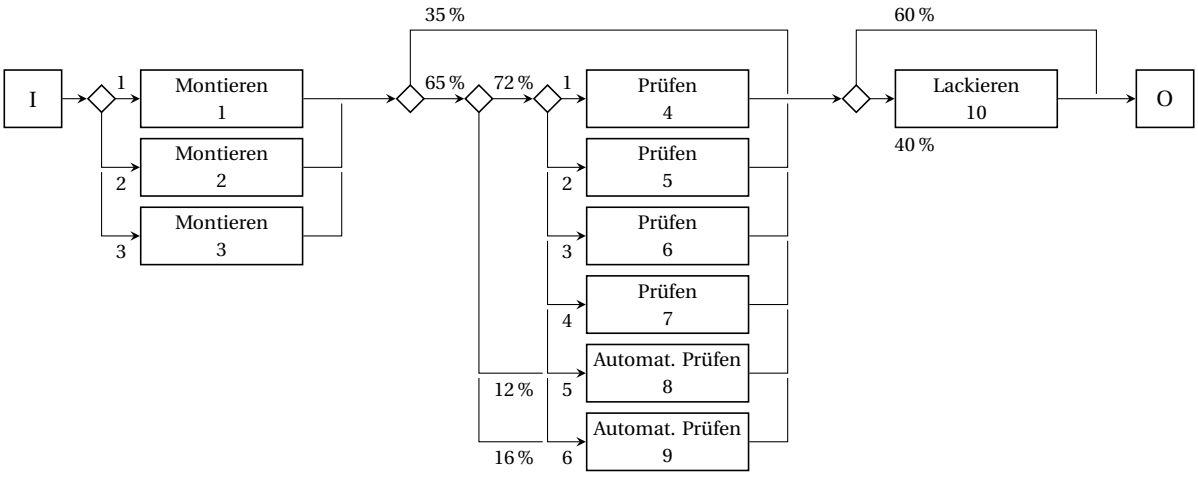


Abbildung 4.8: Wertstromgraph für die Montage eines Produkts (Praxisbeispiel). In der Praxis werden insbesondere zur Montage eines Produkts oftmals verschiedene Varianten betrachtet. Der abgebildete Wertstrom basiert auf drei Prozessschritten, die jeweils ein Stückzahlanteil durchläuft, und spiegelt die Montage eines Produkts an einem Standort der Bosch Rexroth AG wider. Durch die Verkettung von Verknüpfungen und die entsprechende Verbindung mehrerer Flusspunkte können komplexe Bedingungen zur Nutzung der Ressourcen abgebildet werden.

Zahl von ausgehenden Kanten erlaubt. Die Definitionen der Verknüpfungstypen werden dementsprechend auf eine größere Zahl von Operanden erweitert.

Kombination der Modellelemente. Mit Hilfe der oben beschriebenen Knoten und Kanten sowie den Regeln, wie diese miteinander zu kombinieren sind, ist es möglich, komplexe Wertströme auf intuitive Weise zu modellieren. Die Einführung der Software AURELIE bei der Bosch Rexroth AG hat gezeigt, dass die Anwender die grafische Notation nutzen, um eigenständig Lösungen zur Abbildung sich wiederholender Ablaufabschnitte in der Produktion zu entwickeln. Im Zuge der Einführung wurden diese Fragmente grafischer Modelle, in Anlehnung an Entwurfsmuster im Software Engineering auch Wertstrommuster genannt, extrahiert und allen Anwendern bereitgestellt (siehe Hahmann 2012). Damit wurde zum einen das Ziel verfolgt, das Verständnis der grafischen Notation zu fördern. Zum anderen sollte die Modellierung von Systemen beschleunigt werden, indem Wertstrommuster zu Wertströmen zusammengesetzt werden. Abbildung 4.9 zeigt Beispiele für Wertstrommuster zur optionalen Ausführung eines Arbeitsinhalts mit Hilfe einer anderen MAE, von denen eines dem Beispiel AL2 aus Abschnitt 2.5 entspricht.

4.3 Validierung eines grafischen Modells

Bestimmte Regeln in Bezug auf die grafische Notation dürfen während der Modellerstellung zu keinem Zeitpunkt verletzt werden. Hierzu zählt beispielsweise, dass Quellen keinen Eingang und nur einen Ausgang, Senken umgekehrt nur einen Eingang und keinen Ausgang und Flusspunkte nur einen Eingang besitzen dürfen. Andere Regeln müssen im Verlauf der Erstellung eines grafischen Modells verletzt werden können, da sie einen Zwischenzustand des Modells darstellen. Beispielsweise kann an einem Eingang einer Ressource eine Kante anliegen, die zugehörige am gegenüberliegenden Ausgang aber fehlen, da der Anwender die Modellierung noch nicht abgeschlossen hat. Daraus folgt, dass eine geeignete Software bereits mit Hilfe der Benutzeroberfläche bestimmte Regeln umsetzen kann. Es verbleiben jedoch Regeln, die vor der Umwandlung des grafischen Modells in ein mathematisches Modell geprüft werden müssen. Die Prüfung dieser zweiten Gruppe von Regeln, die sogenannte Validierung eines grafischen Modells $\{G_{WS,b}\}$, wird im Folgenden beschrieben.

4.3.1 Ziel, Grundidee und Datenstrukturen

Im Rahmen der Validierung wird geprüft, ob ein gegebenes grafisches Modell $\{G_{WS,b}\}$ in ein mathematisches Modell umgewandelt werden kann. Dieser Schritt erfolgt stets unmittelbar vor der Transformation des grafischen Modells. Der Ablauf orientiert sich grundsätzlich an der Tiefensuche, ebenso wie die verwendeten Datenstrukturen.

Ziel und Vorbedingungen. Das Ziel der Validierung besteht darin, für jeden Wertstromgraphen $G_{WS,b}$ eines grafischen Modells $\{G_{WS,b}\}$ die Erfüllung der folgenden vier Bedingungen zu prüfen. Es genügt hierbei festzustellen, dass mindestens eine Bedingung im Fall eines

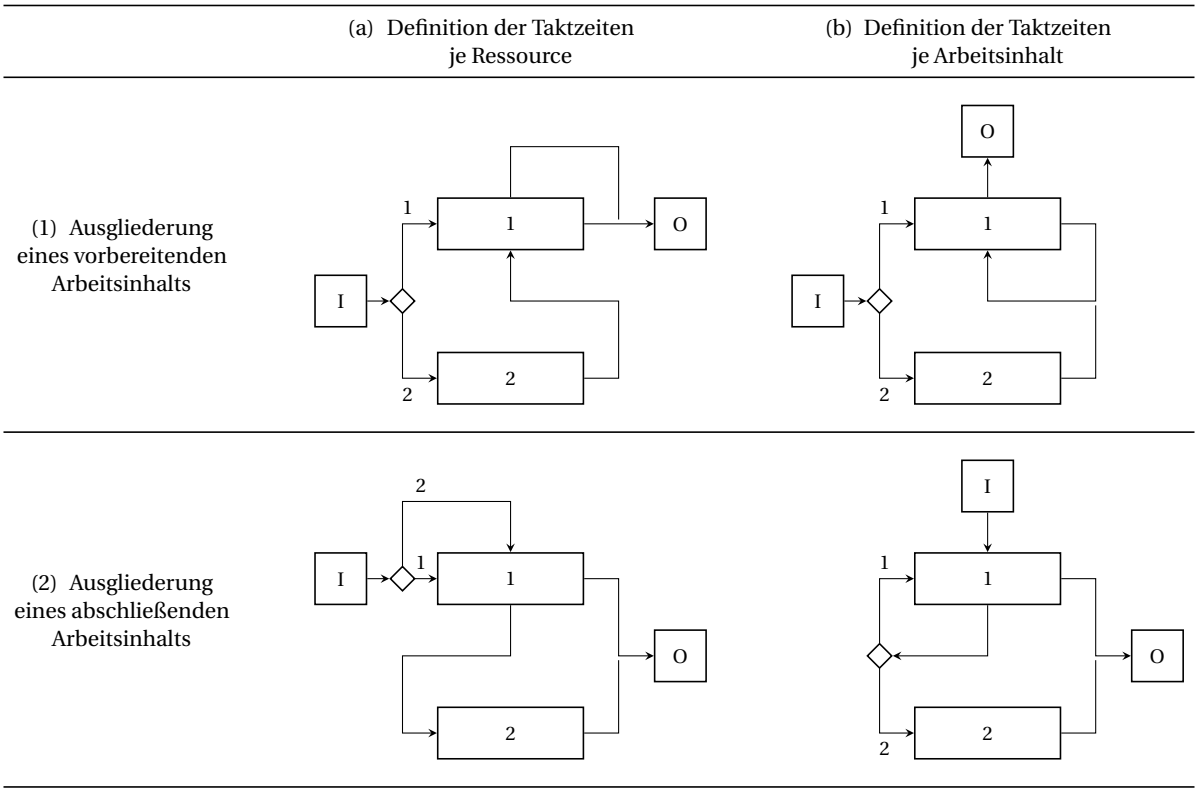


Abbildung 4.9: Muster in grafischen Modellen zur Ausgliederung eines Arbeitsinhalts. Wertstrommuster sind Modellfragmente zur Abbildung von Ablaufabschnitten, die in der Praxis wiederholt anzutreffen sind. Die Abbildung zeigt Möglichkeiten zur Modellierung eines Arbeitsinhalts, der bei Bedarf unter Nutzung einer anderen MAE ausgeführt werden kann. Abhängig davon, ob es sich um einen vorbereitenden oder abschließenden Arbeitsinhalt handelt und ob Taktzeiten einer Ressource zusammengefasst oder unterschieden werden sollen, existieren vier Fälle.

einzigen Wertstromgraphen verletzt ist, da in diesem Fall das grafische Modell als Ganzes ungültig ist. Die vier Bedingungen lauten:

B1 *Definierter Beginn jedes Wertstroms:* Am Ausgang der Quelle σ_b jedes Wertstromgraphen $G_{WS,b}$ muss genau eine Kante anliegen, die zu einem anderen Knoten führt. Das heißt, die Quelle σ_b muss einen definierten nachfolgenden Knoten besitzen, welcher durch die anliegende Kante erreicht wird.

B2 *Kein unendlicher Zyklus:* Folgt man ausgehend von der Quelle σ_b in jedem Wertstromgraphen $G_{WS,b}$ den nachfolgenden Kanten, darf keine Kante mehr als einmal besucht werden. Die vier möglichen Fälle solcher unendlichen Zyklen sind in Abbildung 4.10 zusammenfassend dargestellt.

B3 *Keine Kante ohne Nachfolger:* Zur Traversierung des Wertstromgraphen $G_{WS,b}$ muss jede Kante zur Senke τ_b weiterverfolgt werden können. Folglich muss jede Kante einen Nachfolger besitzen, sofern sie nicht in die Senke τ_b mündet. Ein Beispiel für eine Kante ohne Nachfolger ist in Abbildung 4.11(a) wiedergegeben.

B4 *Keine Kante ohne Vorgänger:* Jede Kante des Wertstromgraphen $G_{WS,b}$ muss ausgehend von der Quelle σ_b erreichbar sein. Das heißt, jede Kante muss einen Vorgänger besitzen, sofern sie nicht an der Quelle σ_b anliegt. Abbildung 4.11(b) zeigt ein entsprechendes Beispiel einer Kante ohne Vorgänger.

Dabei wird vorausgesetzt, dass das zu validierende grafische Modell $\{G_{WS,b}\}$ die folgenden Vorbedingungen erfüllt. Diese repräsentieren Regeln, deren Einhaltung bereits im Verlauf der Modellerstellung durch die Software geprüft werden kann:

V1 Jeder Knoten darf höchstens die *Zahl von Eingängen und Ausgängen* besitzen, welche der Knotentyp erlaubt (siehe Abbildung 4.2). Dies betrifft die Quelle σ_b und die Senke τ_b sowie die alternativen und selektiven Flusspunkte $D_{AL,b}$ bzw. $D_{SL,b}$.

V2 An jedem *Ausgang eines Knotens* darf jeweils *nur eine einzige Kante* anliegen. Im Fall der alternativen und selektiven Flusspunkte $D_{AL,b}$ bzw. $D_{SL,b}$ entspricht jede ausgehende Kante einem jeweils anderen Ausgang.

V3 Im Fall der *Ressourcen R* muss für jede eingehende Kante eine *effektive Taktzeit* t_{eff} größer als null definiert sein. Darüber hinaus muss für jede Ressource eine *Betriebsmittelzeit* T_{BM} gegeben sein, die ebenfalls größer als null ist.

V4 Im Fall der *alternativen Flusspunkte* $D_{AL,b}$ muss jeder ausgehenden Kante eine *Priorität* zugeordnet sein. Es wird im Folgenden vorausgesetzt, dass die Priorität jeweils dem Ausgang j entspricht, an welchem die Kante anliegt.

V5 Im Fall der *selektiven Flusspunkte* $D_{SL,b}$ muss jeder ausgehenden Kante eine *Quote* größer als null und kleiner als oder gleich 100 Prozent zugeordnet sein. Die Summe der Quoten eines selektiven Flusspunkts muss gleich 100 Prozent sein.

Grundidee zum Ablauf. Zum besseren Verständnis wird der Algorithmus in zwei Stufen vorgestellt. Zunächst soll der Ablauf der Traversierung aller Kanten eines grafischen Modells $\{G_{WS,b}\}$ durch einen ersten, einfachen Algorithmus 4.1 veranschaulicht werden. Der Algorithmus wird anschließend um Datenstrukturen und Anweisungen erweitert, um die Erfüllung der obigen Bedingungen zu prüfen. Diese Erweiterung führt zu Algorithmus 4.2, welcher dazu dient, ein grafisches Modell $\{G_{WS,b}\}$ zu validieren. Die Beschreibung erfolgt in diesem Kapitel anhand von Kurzfassungen in natürlicher Sprache. Die Langfassungen liegen in Gestalt des Algorithmus A.1.5 bzw. A.1.6 im Anhang vor, jeweils erstellt in formalem, kommentiertem Pseudocode. Im Anhang ist zudem eine vollständige Liste aller Objekte und Variablen sowie Funktionen und Prozeduren zu finden.

Zur Prüfung der oben aufgeführten Bedingungen müssen alle Wertstromgraphen $G_{WS,b}$ eines grafischen Modells $\{G_{WS,b}\}$ betrachtet werden. Die Prüfung endet, nachdem die entsprechende Schleife durchlaufen wurde oder ein Wertstromgraph $G_{WS,b}$ auch nur eine der oben stehenden Bedingungen verletzt. Im Zuge der Prüfung ist es erforderlich, jeden Wertstromgraphen $G_{WS,b}$ vollständig zu traversieren. Das heißt, alle Kanten $E_{WS,b}$ des jeweiligen Graphen müssen genau einmal beschritten werden. Da entscheidend ist, ob ausgehend von der Quelle σ_b eine Kante mehrmals besucht wird, orientiert sich der Algorithmus der

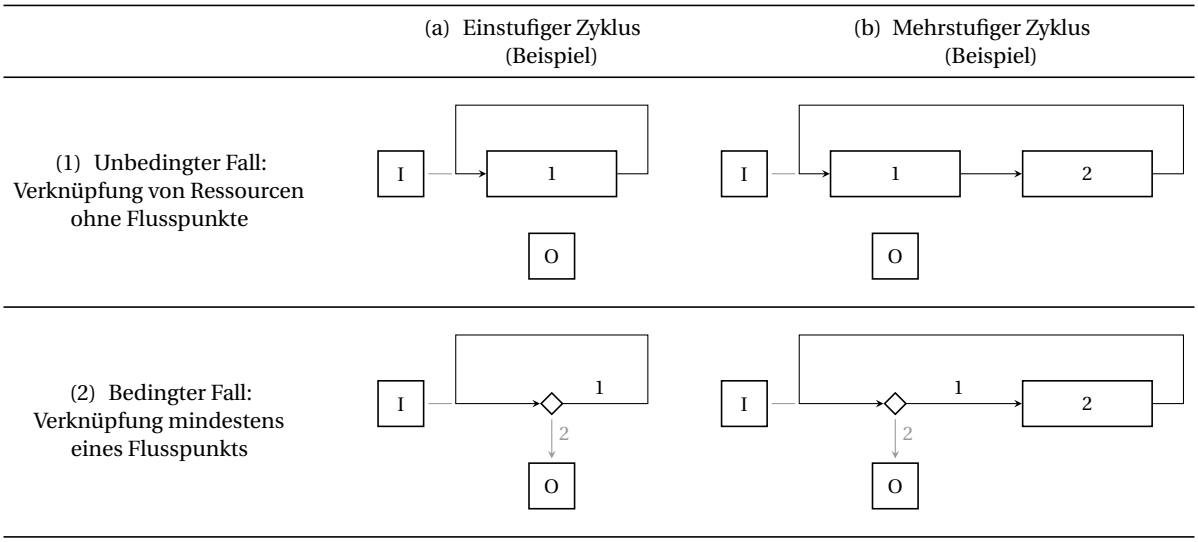
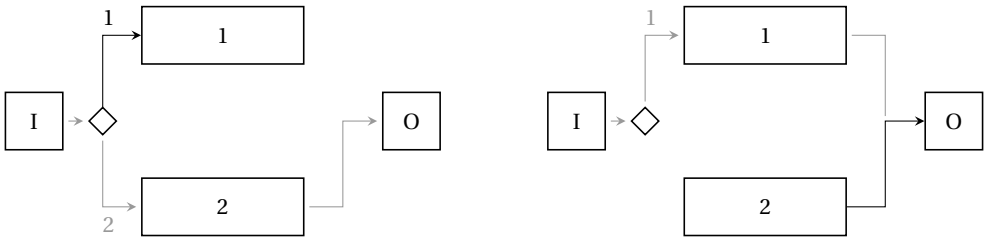


Abbildung 4.10: Kategorisierung unendlicher Zyklen in Wertstromgraphen. Im Zuge der Validierung wird sichergestellt, dass kein Wertstromgraph einen unendlichen Zyklus enthält. Unter dem Begriff ist ein Kantenzug zu verstehen, in welchem sich eine Kante wiederholt, sodass stets eine nachfolgende Kante existiert, aber die Senke niemals erreicht wird. Zyklen dieser Art können mehrstufig sein und durch Flusspunkte verlaufen, entsprechend den dargestellten vier Fällen.



(a) Kante ohne Nachfolger (Beispiel) (b) Kante ohne Vorgänger (Beispiel)

Abbildung 4.11: Kategorisierung nicht zulässiger Kanten in Wertstromgraphen. Neben unendlichen Zyklen muss ausgeschlossen werden, dass in Wertstromgraphen Kanten existieren, die nicht Teil eines Wegs von der Quelle bis zur Senke sind. Sofern kein unendlicher Zyklus vorliegt, betrifft dies alle Kanten, die keinen Nachfolger oder keinen Vorgänger besitzen und nicht an der Senke bzw. Quelle anliegen. Die zugehörigen zwei Fälle sind jeweils durch ein Beispiel veranschaulicht.

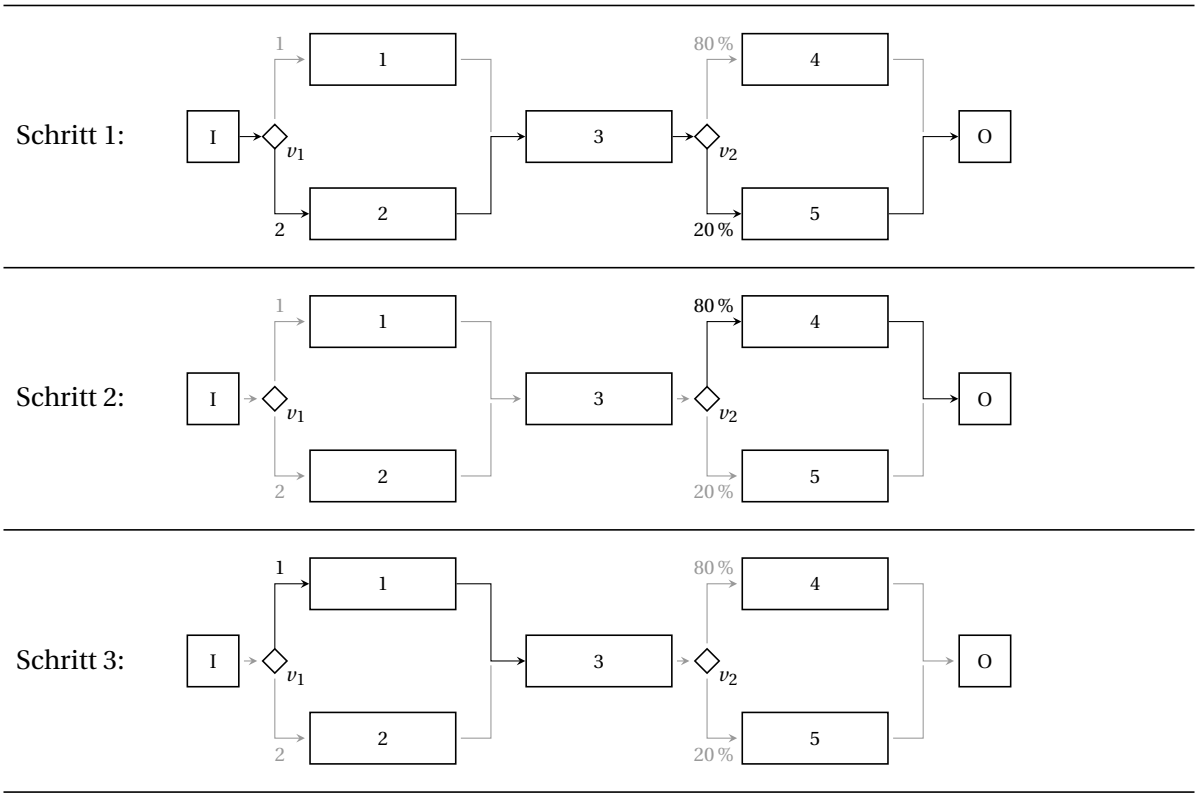


Abbildung 4.12: Schrittfolge der Validierung eines Wertstromgraphen (Beispiel). Grundlage ist ein einfacher exemplarischer Wertstromgraph mit zwei Flusspunkten. Wie für die Tiefensuche charakteristisch wird zuerst ein vollständiger Weg von der Quelle bis zur Senke gesucht, wobei jeweils der letzte Ausgang an jedem Flusspunkt weiterverfolgt wird. Danach wird die Kante am vorherigen Ausgang des zuletzt besuchten Flusspunkts betrachtet. Die Suche wird solange fortgesetzt, bis alle Kanten beschritten wurden, die ausgehend von der Quelle erreichbar sind.

Validierung an der Tiefensuche. Die Traversierung beginnt bei der Quelle σ_b und wird solange fortgesetzt, bis keine weiteren Kanten zu besuchen sind, wie Abbildung 4.12 an einem einfachen Beispiel zeigt. An alternativen und selektiven Flusspunkten werden alle ausgehenden Kanten beschritten, wobei immer erst eine Kante bis zur Senke τ_b weiterverfolgt wird, bevor die anderen Kanten besucht werden. Zur Prüfung der Bedingungen werden die nachfolgenden Datenstrukturen verwendet.

Datenstrukturen. Analog zur Tiefensuche kommt ein *Stapel* zum Einsatz, um die als Nächstes zu besuchenden Kanten zu verwalten. Wie in der Einführung erläutert, ist ein solcher *Stapel* eine linear verkettete Liste, die nach dem LIFO-Prinzip verwaltet wird. Das heißt, zuletzt hinzugefügte Elemente werden zuerst zurückgegeben. Dank dieser Eigenschaft eignet sich ein *Stapel* zur Ausführung einer Tiefensuche, da sich diese dadurch auszeichnet, dass zunächst immer erst alle nachfolgenden Kanten und Knoten besucht werden. Bei den Elementen des *Stapels* handelt es sich um Kanten und nicht, wie im Fall der Tiefensuche aus der Literatur gemäß Algorithmus A.1.3, um Knoten, da die Bedingungen die Kanten jedes Wert-

stromgraphen $G_{WS,b}$ betreffen. Weitere Datenstrukturen werden für den Algorithmus 4.1 zur Traversierung der Kanten nicht benötigt.

In dem darauf aufbauenden Algorithmus 4.2 zur Validierung eines gegebenen grafischen Modells $\{G_{WS,b}\}$ müssen zusätzliche Datenstrukturen verwendet werden, um die definierten Bedingungen zu prüfen. Zum einen wird mit Hilfe eines *Kantenzugs* die Folge von Kanten verwaltet, die in jedem Wertstromgraphen $G_{WS,b}$ ausgehend von der Quelle σ_b zur aktuellen Kante führen. Indem geprüft wird, ob die jeweils als Nächstes zu besuchende Kante bereits in dem *Kantenzug* enthalten ist, können unendliche Zyklen erkannt werden. Nachdem die Senke τ_b erreicht wurde, muss der *Kantenzug* von der letzten Kante beginnend reduziert werden, bis die letzte verbleibende Kante der Vorgänger der als Nächstes zu besuchenden Kante ist. Dadurch ist sichergestellt, dass der *Kantenzug* zu jedem Zeitpunkt die beschriebene Folge von Kanten von der Quelle σ_b bis zur aktuellen Kante repräsentiert. Da der *Kantenzug* jeweils an dessen Ende um eine Kante erweitert oder reduziert wird, handelt es sich wie im Fall des *Stapels* um eine nach dem LIFO-Prinzip verwaltete, linear verkettete Liste.

Um zu prüfen, ob letztendlich alle Kanten besucht wurden, genügt es, eine Menge aller noch nicht besuchten Kanten zu definieren. Diese wird mit der Menge $E_{WS,b}$ jedes Wertstromgraphen $G_{WS,b}$ initialisiert und bei jedem Besuch einer Kante um ebendiese Kante reduziert. Ist der *Stapel* leer und verbleiben keine weiteren zu besuchenden Kanten, muss entsprechend auch die *MengeUnbesuchterKanten* leer sein.

4.3.2 Beschreibung der Algorithmen

Zunächst soll der Ablauf des einführenden Algorithmus 4.1 erläutert werden, welcher die Traversierung aller Kanten eines grafischen Modells $\{G_{WS,b}\}$ beschreibt (Langfassung siehe Algorithmus A.1.5). Auf dieser Basis folgt die Erläuterung des Algorithmus 4.2 zur Validierung eines solchen grafischen Modells (Langfassung siehe Algorithmus A.1.6).

Algorithmus 4.1: Traversierung der Kanten eines grafischen Modells $\{G_{WS,b}\}$

Prozedur TraversiereKantenVonGrafischemModell(...)

In Zeile 01 ist die Hauptprozedur zur Traversierung der Kanten eines gegebenen grafischen Modells $\{G_{WS,b}\}$ deklariert. Gemäß der Definition dieser Prozedur werden nacheinander die Wertstromgraphen $G_{WS,b}$ aller Produkte b betrachtet. In dem Zuge werden alle Kanten $E_{WS,b}$ des jeweiligen Wertstromgraphen $G_{WS,b}$ als nicht entdeckt markiert, bevor die Hilfsprozedur zur Traversierung dieser Kanten aufgerufen wird.

Prozedur TraversiereKantenVonWSGraphen(...)

Die Definition der entsprechenden Hilfsprozedur folgt ab Zeile 06. Im ersten Schritt wird der *Stapel* als ein leeres Tupel initialisiert, wobei dessen Elemente jeweils eine Kante e bezeichnen. Daran anknüpfend wird geprüft, ob am Ausgang der Quelle σ_b eine Kante anliegt. Ist dies der Fall, wird der Kante zum nächsten Knoten v' und dessen Eingang i' gefolgt, um die entsprechende Kante ans Ende des *Stapels* zu fügen. Ab Zeile 13 wird in einer Schleife geprüft, ob der *Stapel* Elemente enthält und somit weitere Kanten zu besuchen sind. Wurde

```
     Eingabe: grafisches Modell {G_WS,b} aus Wertstromgraphen G_WS,b für alle Produkte b
     Ausgabe: markierte Kanten E_WS,b
01:  Prozedur TraversiereKantenVonGrafischemModell({G_WS,b})
02:      für alle Produkte b tue
...          für alle Kanten E_WS,b tue initialisiere die Kante als nicht entdeckt
05:          TraversiereKantenVonWSGraphen(G_WS,b, b)

06:  Prozedur TraversiereKantenVonWSGraphen(G_WS,b, b)
07:      initialisiere Stapel als leeres Tupel
08:      wenn an der Quelle σ_b eine Kante anliegt dann
...          folge dieser Kante e', um sie ans Ende des Stapels zu fügen
13:      solange Stapel nicht leer ist tue
14:          entferne aktuelle Kante e vom Ende des Stapels
15:          wenn Kante e bisher noch nicht entdeckt wurde dann
16:              markiere Kante e als entdeckt
...              TraversiereKantenAusRessource(...) bzw. TraversiereKantenAusFlusspunkt(...)

22:  Prozedur TraversiereKantenAusRessource(G_WS,b, Stapel, v, i)
23:      betrachte nur Ausgang j des Knotens v gegenüber von Eingang i
24:      wenn an Ausgang j eine Kante anliegt dann
...          folge dieser Kante e', um sie ans Ende des Stapels zu fügen

29:  Prozedur TraversiereKantenAusFlusspunkt(G_WS,b, Stapel, v)
30:      für alle Kanten an Ausgängen j tue
...          folge dieser Kante e', um sie ans Ende des Stapels zu fügen
```

Algorithmus 4.1: Traversierung der Kanten eines grafischen Modells $\{G_{WS,b}\}$ (Kurzfassung).

zu Beginn an der Quelle σ_b eine Kante gefunden, ist diese Bedingung erfüllt. Im Anschluss daran wird eine Kante e vom Ende des *Stapels* entfernt, um diese im nächsten Schritt zu betrachten. Falls die aktuelle Kante e noch nicht entdeckt wurde, wird sie entsprechend als entdeckt markiert, und es wird der Knoten v' ermittelt, zu welchem die Kante e führt. Sofern es sich bei dem Knoten v' um eine Ressource handelt, wird die Hilfsprozedur zur Traversierung solcher Kanten aufgerufen, die an Ausgängen einer Ressource anliegen. Ist der Knoten v' jedoch ein alternativer oder selektiver Flusspunkt, dann wird die Hilfsprozedur für entsprechende Knoten aufgerufen. Wenn aber der Knoten v' die Senke τ_b bezeichnet, dann wird die Kante e nicht weiterverfolgt. Falls der *Stapel* keine weiteren Elemente enthält, wird die Schleife daraufhin verlassen.

Prozedur TraversiereKantenAusRessource(...)

Die Hilfsprozedur ab Zeile 22 beschreibt, welche Kante nach einer Ressource weiterverfolgt werden soll, wenn eine gegebene Kante an einem Eingang i dieser Ressource anliegt. Wie zuvor erläutert, wird der Ausgang j der Ressource betrachtet, welcher dem Eingang i gegenüberliegt und in Bezug auf den Index diesem Eingang i entspricht. Sofern am Ausgang j eine Kante anliegt, wird diese Kante e' zum Knoten v' und dem Eingang i' an diesem

Knoten weiterverfolgt. Um die ausgehende Kante e' als Nächstes zu besuchen, wird sie ans Ende des *Stapels* gefügt und danach die Hilfsprozedur verlassen.

Prozedur TraversiereKantenAusFlusspunkt(...)

In Zeile 29 ist die umso kürzere Hilfsprozedur definiert, die beschreibt, welche Kanten nach einem alternativen oder selektiven Flusspunkt weiterverfolgt werden sollen. Einfacherweise werden die Kanten e' an allen Ausgängen j nacheinander zu den jeweils nächsten Knoten v' und Eingängen i' weiterverfolgt, um sie ans Ende des *Stapels* zu fügen. Auf diese Weise wird die Kante, die am letzten Ausgang anliegt, zuerst besucht. Die Reihenfolge, in welcher die ausgehenden Kanten eines Flusspunkts betrachtet werden, besitzt im Fall der Validierung eines grafischen Modells $\{G_{WS,b}\}$ noch keine Bedeutung.

Algorithmus 4.2: Validierung eines grafischen Modells $\{G_{WS,b}\}$

Auf dieser Basis folgt nun die Beschreibung von Algorithmus 4.2, der Validierung eines grafischen Modells $\{G_{WS,b}\}$ (Langfassung siehe Algorithmus A.1.6). Viele Abschnitte basieren auf dem vorangegangenen, weshalb sie kürzer gefasst wiedergegeben werden.

Funktion ValidiereGrafischesModell(...)

In der Hauptfunktion, die in Zeile 01 deklariert ist, werden wie zuvor alle Wertstromgraphen $G_{WS,b}$ des grafischen Modells $\{G_{WS,b}\}$ betrachtet. Sobald die Validierung eines dieser Wertstromgraphen negativ ausfällt, wird ein entsprechendes Ergebnis zurückgegeben, und die Hauptfunktion wird vorzeitig verlassen. Nur wenn die Wertstromgraphen $G_{WS,b}$ aller Produkte b die definierten Bedingungen erfüllen, wird die Schleife bis zum letzten Produkt b durchlaufen und ein positives Ergebnis zurückgegeben.

Funktion ValidiereWSGraphen(...)

Ab Zeile 06 folgt die Definition der Hilfsfunktion zur Validierung eines Wertstromgraphen $G_{WS,b}$. Zunächst wird der *Stapel* wieder als leeres Tupel initialisiert, ebenso wie der *Kantenzug*, welcher den Weg von der Quelle σ_b bis zur aktuellen Kante wiedergibt. Anschließend werden alle Kanten als nicht besucht markiert. Liegt an der Quelle σ_b keine Kante an, wird die Validierung des Wertstromgraphen $G_{WS,b}$ bereits an diesem Punkt negativ beendet. Sonst wird die entsprechende Kante e' ans Ende des *Stapels* gefügt. Damit wird in Zeile 16 die Schleife betreten, wobei die Schleifenbedingung an dieser Stelle erfüllt sein muss und daher im Gegensatz zum vorigen Algorithmus erst am Ende geprüft wird.

Zuerst wird die aktuelle Kante e_2 vom Ende des *Stapels* entfernt. Enthält der *Kantenzug* mindestens eine Kante und wird folglich ein begonnener Weg von der Quelle σ_b fortgesetzt, wird die Kante e_1 an dessen Ende ermittelt. Solange die aktuelle Kante e_2 vom *Stapel* kein Nachfolger der Kante e_1 vom *Kantenzug* ist, wird der *Kantenzug* reduziert, danach dessen letzte Kante e_1 bestimmt und die Bedingung erneut geprüft. Als Ergebnis wird der *Kantenzug* zurückverfolgt, bis dieser den Weg von der Quelle σ_b bis zur aktuellen Kante e_2 widerspiegelt. Diese Schrittfolge muss jeweils ausgeführt werden, nachdem ein vollständiger Weg von der Quelle σ_b bis zur Senke τ_b beschritten wurde oder wenn nach der betrachteten Kante keine

Eingabe: grafisches Modell $\{G_{WS,b}\}$ aus Wertstromgraphen $G_{WS,b}$ für alle Produkte b
Ausgabe: `wahr`, wenn das grafische Modell $\{G_{WS,b}\}$ gültig ist, sonst `falsch`

```
01: Funktion ValidiereGrafischesModell({G_WS,b})
02:     für alle Produkte b tue
03:         wenn nicht ValidiereWSGraphen(G_WS,b, b) gilt dann
04:             liefere falsch zurück
05:     liefere wahr zurück

06: Funktion ValidiereWSGraphen(G_WS,b, b)
...     initialisiere Stapel und Kantenzug als leere Tupel
09:     für alle Kanten E_WS,b tue initialisiere die Kante als nicht besucht
10:     wenn an der Quelle σ_b keine Kante anliegt dann
11:         liefere falsch zurück
...     folge der Kante e' an der Quelle σ_b, um sie ans Ende des Stapels zu fügen
16:     wiederhole
...         entferne aktuelle Kante e_2 vom Ende des Stapels
19:         wenn Kantenzug nicht leer ist dann
...             ermittle zuletzt besuchte Kante e_1 vom Ende des Kantenzugs
22:             solange Kante e_2 nicht unmittelbar auf Kante e_1 folgt tue
...                 entferne Kante und ermittle zuletzt besuchte Kante e_1 vom Ende des Kantenzugs
26:         wenn Kante e_2 bereits im Kantenzug enthalten ist dann
27:             liefere falsch zurück
28:         wenn Kante e_2 bisher noch nicht besucht wurde dann
29:             markiere Kante e_2 als besucht und füge sie ans Ende des Kantenzugs
...             wenn nicht ValidiereKantenAusRessource(...) bzw.
                        nicht ValidiereKantenAusFlusspunkt(...) gilt dann
...                 liefere falsch zurück
37:     solange Stapel nicht leer ist
38:     wenn nicht alle Kanten E_WS,b besucht wurden dann
39:         liefere falsch zurück
40:     liefere wahr zurück

41: Funktion ValidiereKantenAusRessource(G_WS,b, Stapel, v, i)
42:     betrachte nur Ausgang j des Knotens v gegenüber von Eingang i
43:     wenn an Ausgang j keine Kante anliegt dann
44:         liefere falsch zurück
...     folge der Kante e' an Ausgang j, um sie ans Ende des Stapels zu fügen
49:     liefere wahr zurück

50: Funktion ValidiereKantenAusFlusspunkt(G_WS,b, Stapel, v)
...     wenn an keinem Ausgang eine Kante anliegt dann
53:         liefere falsch zurück
54:     für alle Kanten an Ausgängen j tue
...         folge dieser Kante e', um sie ans Ende des Stapels zu fügen
59:     liefere wahr zurück
```

Algorithmus 4.2: Validierung eines grafischen Modells $\{G_{WS,b}\}$ (Kurzfassung).

Kanten mehr folgen, die es noch zu besuchen gilt (siehe Schritt 1 in Abbildung 4.12). Danach müssen alle verbleibenden Kanten beschritten werden, die an Flusspunkten anliegen, die auf diesem Wege besucht wurden (siehe Schritt 2 und 3 in Abbildung 4.12).

Ist die aktuelle Kante e_2 gemäß der Prüfung in Zeile 26 in dem reduzierten *Kantenzug* enthalten, muss ein unendlicher Zyklus vorliegen, und die Hilfsfunktion wird mit negativem Ergebnis verlassen. Wenn das aber nicht der Fall ist und die Kante e_2 noch nicht besucht wurde, dann wird diese entsprechend markiert und ans Ende des *Kantenzugs* gefügt. Analog zum vorherigen Algorithmus werden im Anschluss daran unterschiedliche Hilfsfunktionen aufgerufen, um abhängig vom Typ des Knotens v_2', zu welchem die Kante e_2 führt, zu verzweigen. Sobald der *Stapel* keine Elemente mehr enthält, wird gemäß der Bedingung in Zeile 37 die Schleife verlassen, und es wird geprüft, ob alle Kanten $E_{WS,b}$ des Wertstromgraphen $G_{WS,b}$ besucht wurden. Ist dies nicht der Fall, muss mindestens eine Kante ohne Vorgänger existieren, die nicht an der Quelle σ_b anliegt. Danach wird die Hauptfunktion verlassen und das Ergebnis dieser Prüfung zurückgegeben.

Funktion ValidiereKantenAusRessource(...)

In Zeile 41 ist die Hilfsfunktion zur Validierung aller ausgehenden Kanten an einem Eingang i einer Ressource deklariert. Wieder wird ausschließlich derjenige Ausgang j betrachtet, welcher dem Eingang i der Ressource gegenüberliegt. Wenn an Ausgang j keine Kante anliegt, wird die Hilfsfunktion mit negativem Ergebnis beendet. In diesem Fall besitzt die Kante, die zum vorherigen Eingang i der Ressource führt, keinen Nachfolger. Andernfalls wird die jeweilige Kante e' ans Ende des *Stapels* gefügt und somit als Nächstes besucht.

Funktion ValidiereKantenAusFlusspunkt(...)

Die Hilfsfunktion zur Validierung aller Kanten an Ausgängen alternativer und selektiver Flusspunkte folgt ab Zeile 50. Liegen keine Kanten an, wird die Hilfsfunktion verlassen und ein negatives Ergebnis zurückgegeben. In diesem Fall besitzt die eingehende Kante keinen Nachfolger. Sonst werden wie schon zuvor die Kanten e' an allen Ausgängen j ans Ende des *Stapels* gefügt, um sie als Nächstes zu besuchen.

4.3.3 Beweis der Zeitkomplexität

Gesucht ist nun die Komplexität der zwei Algorithmen. Der Fokus liegt auf einer oberen Schranke für die asymptotisch wachsende Laufzeit. Da hierzu die Größe der Datenstrukturen betrachtet wird, kann damit auch auf den Speicherplatzbedarf geschlossen werden. Gegeben sei ein grafisches Modell $\{G_{WS,b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{WS,b}$, deren Kanten zu traversieren sind bzw. welches zu validieren ist.

Algorithmus 4.1: Traversierung der Kanten eines grafischen Modells $\{G_{WS,b}\}$

Der einführende Algorithmus 4.1 entspricht einer Tiefensuche, in deren Verlauf die Wertstromgraphen $G_{WS,b}$ vollständig traversiert werden. Im Unterschied zu einer klassischen Tiefensuche werden nicht die Knoten, sondern die Kanten $E_{WS,b}$ jedes Wertstromgraphen $G_{WS,b}$ in einem *Stapel* verwaltet. Bezüglich der Laufzeit des Algorithmus gilt:

Theorem 4.1: Zeitkomplexität von Algorithmus 4.1. Es sei ein grafisches Modell $\{G_{\mathrm{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\mathrm{WS},b}$ gegeben, welche die Wertströme eines Systems zur Herstellung einer entsprechenden Zahl von Produkten abbilden. Der Algorithmus zur Traversierung aller Kanten des Modells besitzt in der angegebenen Implementierung eine Laufzeit von $\mathrm{O}\left(d^{+}_{\max} \sum_{b=1}^{n} |E_{\mathrm{WS},b}|\right)$.

Beweis. Im Verlauf der Traversierung wird jede Kante e höchstens einmal entdeckt, da sie unmittelbar danach entsprechend markiert wird. Nachdem sie entdeckt wurde, werden all ihre Nachfolger e' ans Ende des *Stapels* gefügt. Wenn die aktuelle Kante in einen Eingang i eines Knotens v mündet, dann bezeichnet ein solcher Nachfolger eine Kante, die an einem zugeordneten Ausgang j desselben Knotens anliegt. Die entsprechende Zuordnung von Ausgängen j zu Eingängen i ist vom Typ des Knotens v abhängig.

Im Fall einer Ressource wird nur die Kante weiterverfolgt, die am Ausgang j gegenüber von Eingang i anliegt, sofern diese existiert. Bezeichnet der Knoten v eine Ressource, besitzt die eingehende Kante daher unabhängig vom gewählten Eingang i höchstens einen Nachfolger. Im Fall eines alternativen oder selektiven Flusspunkts werden alle Kanten betrachtet, die an den Ausgängen des Knotens anliegen. Diesbezüglich wird angenommen, dass der Ausgangsgrad der Flusspunkte einen gegebenen Wert $d^{+}_{\max}$ nicht überschreitet. Unter dieser Annahme werden für jeden Wertstromgraphen $G_{\mathrm{WS},b}$ nicht mehr als $d^{+}_{\max}|E_{\mathrm{WS},b}|$ Kanten in den *Stapel* aufgenommen. Da jeweils genau eine Kante aus dem *Stapel* entfernt wird, muss die Schleife entsprechend oft durchlaufen werden. Weiter wird angenommen, dass die Zahl $\sum_{b=1}^{n} |E_{\mathrm{WS},b}|$ der Kanten mindestens der Zahl n der Produkte entspricht.

Weiterhin wird vorausgesetzt, dass die Bestimmung nachfolgender Kanten e' an einem Knoten v mit einer Laufzeit von $\mathrm{O}(1)$ möglich ist. Dabei wird unterstellt, dass eine geeignete Datenstruktur zur Repräsentation der Kanten genutzt wird (wie z. B. eine Adjazenzmatrix). Die gleiche Voraussetzung gilt für das Hinzufügen und Entfernen von Elementen des *Stapels* sowie für die Bestimmung, ob eine Kante bereits entdeckt wurde, und die entsprechende Markierung der Kante. Daraus resultieren für die Zeilen im Rumpf der Prozeduren jeweils die folgenden Laufzeiten (ohne aufgerufene Hilfsprozeduren):

02:	$\mathrm{O}(n)$	13 … 15:	$\mathrm{O}\left(d^{+}_{\max} \sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$
03 … 04:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$	16 … 21:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$
05:	$\mathrm{O}(n)$	23 … 28:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$
07 … 12:	$\mathrm{O}(n)$	30 … 34:	$\mathrm{O}\left(d^{+}_{\max} \sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$

Die Gesamtlaufzeit entspricht der Summe der angegebenen Laufzeiten für jede Zeile. Da hierbei kleinere Summanden und konstante Faktoren vernachlässigt werden, ist die gesuchte obere Schranke gleich dem zu zeigenden Ausdruck $\mathrm{O}\left(d^{+}_{\max} \sum_{b=1}^{n} |E_{\mathrm{WS},b}|\right)$. □

In grafischen Modellen realer Wertströme besitzen Flusspunkte i. d. R. nur einen einstelligen Ausgangsgrad. Zudem skaliert zwar die Zahl der Knoten und Kanten mit der Größe des Modells, jedoch wächst in der Praxis typischerweise nicht die Zahl ausgehender Kanten der einzelnen Flusspunkte. Folglich kann der maximale Ausgangsgrad $d^{+}_{\max}$ der Flusspunkte als Konstante aufgefasst werden. Unter dieser Annahme ist es möglich, für die Laufzeit von Algorithmus 4.1 die Schranke $\mathrm{O}\left(\sum_{b=1}^{n} |E_{\mathrm{WS},b}|\right)$ anzugeben.

Algorithmus 4.2: Validierung eines grafischen Modells $\{G_{WS,b}\}$

Der Algorithmus 4.2 zur Validierung eines grafischen Modells $\{G_{WS,b}\}$ erweitert den zuvor beschriebenen, einführenden Algorithmus 4.1. Wie dieser beruht er auf einer Tiefensuche zur vollständigen Traversierung der Wertstromgraphen $G_{WS,b}$. Neben einem *Stapel* werden ein *Kantenzug* und eine Menge genutzt, um die als Nächstes zu besuchenden bzw. bereits besuchten Kanten zu verwalten. Im Hinblick auf die Laufzeit gilt:

Theorem 4.2: Zeitkomplexität von Algorithmus 4.2. Es sei ein grafisches Modell $\{G_{WS,b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{WS,b}$ gegeben, welche die Wertströme eines Systems zur Herstellung einer entsprechenden Zahl von Produkten abbilden. Der Algorithmus zur Validierung des Modells besitzt in der angegebenen Implementierung eine Laufzeit von $O\left(d^+_{max} \sum_{b=1}^{n} |E_{WS,b}|\right)$.

Beweis. Jede Kante e eines Wertstromgraphen $G_{WS,b}$ wird im Verlauf des Algorithmus höchstens einmal besucht. Danach werden alle Nachfolger e' ans Ende des *Stapels* gefügt. Betrachtet man eine Kante, die zu einem Eingang i eines Knotens v führt, dann ist ein Nachfolger eine Kante, die an einem zugeordneten Ausgang j des Knotens anliegt. Die Zuordnung von Ausgängen j zu Eingängen i ist eine Eigenschaft des Knotentyps.

Falls die aktuelle Kante zu einer Ressource führt, dann wird nur die Kante am gegenüberliegenden Ausgang j betrachtet. Mündet die Kante in einen alternativen oder selektiven Flusspunkt, werden die Kanten an allen Ausgängen weiterverfolgt. Hierbei gilt die Annahme, dass kein Flusspunkt einen Ausgangsgrad besitzt, welcher den Wert d^+_{max} überschreitet. Somit werden in jedem Wertstromgraphen $G_{WS,b}$ nicht mehr als $d^+_{max}|E_{WS,b}|$ Kanten ans Ende des *Stapels* gefügt. Entsprechend oft wird die äußere Schleife durchlaufen, da in jedem Durchlauf genau eine Kante vom *Stapel* entfernt wird.

In dem Moment ihrer Entdeckung wird eine Kante ans Ende des *Kantenzugs* gefügt. Da jede Kante nur ein einziges Mal entdeckt werden kann, folgt daraus, dass in jedem Wertstromgraphen $G_{WS,b}$ insgesamt höchstens $|E_{WS,b}|$ Kanten in den *Kantenzug* aufgenommen werden. Diese Zahl bildet damit eine obere Schranke für die Zahl der Durchläufe der inneren Schleife, in welcher jeweils genau eine Kante vom *Kantenzug* entfernt wird.

Für die Bestimmung nachfolgender Kanten e' sowie für die Operationen des *Stapels* und des *Kantenzugs* gelten die gleichen Voraussetzungen, wie sie im Beweis für Algorithmus 4.1 formuliert wurden. Daraus resultieren jeweils die folgenden Laufzeiten für die Zeilen im Rumpf der Funktionen (ohne aufgerufene Hilfsprozeduren):

02 ... 04:	$O(n)$	37:	$O\left(d^+_{max} \sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$
05:	$O(1)$	38 ... 40:	$O(n)$
07 ... 15:	$O(n)$	42 ... 49:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$
16 ... 22:	$O\left(d^+_{max} \sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$	51 ... 53:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$
23 ... 25:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$	54 ... 58:	$O\left(d^+_{max} \sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$
26 ... 28:	$O\left(d^+_{max} \sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$	59:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$
29 ... 36:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$		

Da bei der Bildung der Summe kleinere Summanden und konstante Faktoren vernachlässigt werden, ist die gesuchte obere Schranke gleich $O\left(d^+_{max} \sum_{b=1}^{n} |E_{WS,b}|\right)$. □

Mit Blick auf Modelle aus der Praxis ist wie im Fall von Algorithmus 4.1 anzumerken, dass die maximale Zahl ausgehender Kanten $d^{+}_{\max}$ der einzelnen Flusspunkte nicht mit der Größe des Modells skaliert und somit eine Konstante darstellt. Demzufolge kann für die Laufzeit von Algorithmus 4.2 die obere Schranke $\mathrm{O}\left(\sum_{b=1}^{n} |E_{\mathrm{WS},b}|\right)$ angenommen werden.

Nach der Validierung eines gegebenen grafischen Modells $\{G_{\mathrm{WS},b}\}$ sind die Voraussetzungen erfüllt, um dieses in ein mathematisches Modell zu transformieren. Die Beschreibung der zugehörigen Algorithmen folgt im nächsten Abschnitt.

4.4 Transformation eines grafischen Modells in ein mathematisches Modell

Wie zu Beginn des Kapitels hervorgehoben, ist einer der wichtigsten Beiträge dieser Arbeit die Transformation eines grafischen Modells in ein mathematisches Modell, das sich zur Optimierung eignet. Ziel der Transformation ist es, die Elemente eines gegebenen Systems und die Beziehungen zwischen diesen Elementen in eine geeignete, insbesondere mathematische Form zu übersetzen. Sobald das Modell in dieser Form vorliegt, können Standardverfahren der mathematischen Optimierung genutzt werden, um die optimalen Werte gemäß den definierten Planungszielen zu suchen. In diesem Abschnitt soll beschrieben werden, wie die genannten Elemente und Beziehungen mathematisch repräsentiert werden und die Transformation eines grafischen Modells abläuft.

Um das resultierende mathematische Modell eindeutig zu beschreiben, werden die folgenden Symbole eingeführt. Darüber hinaus gelten alle anderen Symbole in der Bedeutung, wie in den vorhergehenden Abschnitten und im zugehörigen Verzeichnis angegeben. Aus Abschnitt 2.4 werden die Zeitgrößen zur Kalkulation der Auslastung aufgegriffen. Dies betrifft die Betriebsmittelzeit T_{BM} und die effektive Taktzeit t_{eff}.

a, a'	Index zur Bezeichnung der aktuellen bzw. am jeweils nächsten Eingang i oder Ausgang j anliegenden Prozessstückzahl x_a bzw. $x_{a'}$, $a, a' \in \{1,\dots,m\}$, ggf. mit Index 1, 2 usw.
N	Stufe in der Strukturstückliste eines Produkts b, $N \in \{1,\dots,n\}$
q, q'	Koeffizient der aktuellen bzw. nächsten Prozessstückzahl x_a bzw. $x_{a'}$, $q, q' \in [0,1]$
$\mathbf{x}$	Vektor $[x_1 \dots x_m]^\top$ an Eingängen i und Ausgängen j anliegender Prozessstückzahlen x_a, $\mathbf{x} \in \mathbb{R}^m_{\geqslant 0}$
x_a	Prozessstückzahl mit fortlaufendem Index $a \in \{1,\dots,m\}$, $x_a \in \mathbb{R}_{\geqslant 0}$
$\mathbf{y}$	Vektor $[y_1 \dots y_n]^\top$ der Stückzahlen y_b aller gegebenen Produkte b, $\mathbf{y} \in \mathbb{R}^n_{\geqslant 0}$
y_b	Stückzahl des Produkts $b \in \{1,\dots,n\}$, $y_b \in \mathbb{R}_{\geqslant 0}$

Matrizen eines mathematischen Modells ($\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\mathrm{Invest}}$) (jeweils mit einer allgemeinen, zuerst nicht näher bezeichneten Zeile i):

$\mathbf{P}$	Prozessmatrix,	$\forall b \in \{1,\dots,n\}, a \in \{1,\dots,m\}$	$\mathbf{P}[b, a] \in \{0,1\}$
$\mathbf{S}$	Produktstrukturmatrix,	$\forall b, b' \in \{1,\dots,n\}$	$\mathbf{S}[b, b'] \in \mathbb{R}_{\geqslant 0}$
$\mathbf{F}$	Flussmatrix,	$\forall i \in \{1,2,\dots\}, a \in \{1,\dots,m\}$	$\mathbf{F}[i, a] \in [-1,1]$
$\mathbf{L}$	Auslastungsmatrix,	$\forall i \in \{1,2,\dots\}, a \in \{1,\dots,m\}$	$\mathbf{L}[i, a] \in \mathbb{R}_{\geqslant 0}$
$\mathbf{L}_{\mathrm{Invest}}$	Investitionsmatrix,	$\forall i \in \{1,2,\dots\}, a \in \{1,\dots,m\}$	$\mathbf{L}_{\mathrm{Invest}}[i, a] \in \mathbb{R}_{\geqslant 0}$

Hilfsmatrizen (zur Bestimmung der oben stehenden Matrizen **F**, **L** und $\mathbf{L}_{\text{Invest}}$, jeweils mit Zeile i und j als Eingang bzw. Ausgang eines Knotens v):

$\mathbf{Q}_{\text{aus},bv}$ Ausgangsmatrix des Knotens v im Wertstromgraphen $G_{\text{WS},b}$ des Produkts b, $\forall\, j \in \{1,2,\ldots\}$, $a \in \{1,\ldots,m\}$ $\mathbf{Q}_{\text{aus},bv}[j,a] \in [0,1]$

$\{\mathbf{Q}_{\text{aus},bv}\}$ Menge der Ausgangsmatrizen $\mathbf{Q}_{\text{aus},bv}$ für alle Produkte b und alle alternativen Flusspunkte $D_{\text{AL},b}$, Abkürzung für $\{\mathbf{Q}_{\text{aus},bv} : b \in \{1,\ldots,n\} \wedge v \in D_{\text{AL},b}\}$

$\mathbf{Q}_{\text{ein},bv}$ Eingangsmatrix des Knotens v im Wertstromgraphen $G_{\text{WS},b}$ des Produkts b, $\forall\, i \in \{1,2,\ldots\}$, $a \in \{1,\ldots,m\}$ $\mathbf{Q}_{\text{ein},bv}[i,a] \in [0,1]$

$\{\mathbf{Q}_{\text{ein},bv}\}$ Menge der Eingangsmatrizen $\mathbf{Q}_{\text{ein},bv}$ für alle Produkte b und alle Knoten $V_{\text{WS},b}$ außer Quellen σ_b und Senken τ_b, Abkürzung für $\{\mathbf{Q}_{\text{ein},bv} : b \in \{1,\ldots,n\} \wedge v \in V_{\text{WS},b} \setminus \{\sigma_b,\tau_b\}\}$

U Vereinigungsmatrix, $\forall\, a', a \in \{1,\ldots,m\}$ $\mathbf{U}[a',a] \in [-1,1]$

Bevor der Algorithmus zur Transformation eines grafischen Modells folgt, soll zunächst dargelegt werden, wie die oben stehenden Matrizen genutzt werden, um ein abzubildendes System durch ein mathematisches Modell zu repräsentieren.

4.4.1 Mathematische Modellstruktur: Matrizen und Folgen

Ein mathematisches Modell, welches aus einem grafischen Modell $\{G_{\text{WS},b}\}$ hervorgeht, ist aus fünf Matrizen zusammengesetzt: der Prozessmatrix **P**, der Produktstrukturmatrix **S**, der Flussmatrix **F** und der Auslastungsmatrix **L** sowie der Investitionsmatrix $\mathbf{L}_{\text{Invest}}$. Die ersten vier Matrizen dienen der Abbildung von Bedingungen in Bezug auf Produkte und Prozesse. Die Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ wird eingeführt, um die Entscheidung des Anwenders abzubilden, welche Prozessschritte im Bedarfsfall zur Überlastung von Ressourcen führen dürfen. Somit kann ein solches mathematisches Modell als ein Tupel $(\mathbf{P},\mathbf{S},\mathbf{F},\mathbf{L},\mathbf{L}_{\text{Invest}})$ aufgefasst werden. Die Verteilung der Produktstückzahlen y_b wird durch Variablen, die Prozessstückzahlen x_a, abgebildet. Deren Priorisierung wird durch die *MaxFolge* wiedergegeben. Im Folgenden werden die Matrizen bzw. Folgen erläutert.

Variablen zur Abbildung der Stückzahlverteilung. In der Gesamtheit bestimmen die Matrizen die Verteilung der Produktstückzahlen in den Wertströmen der Produkte (vgl. Systemstruktur, Abschnitt 2.2). Dem liegt folgendes Konzept zugrunde: An jedem Eingang i und an jedem Ausgang j eines Knotens v liegt ein Anteil der Stückzahl y_b eines Produkts b an, welcher den Knoten an dem Eingang durchläuft bzw. an dem Ausgang verlässt. Der Stückzahlanteil folgt aus der Multiplikation einer Prozessstückzahl x_a mit einem Koeffizienten q im Intervall $[0,1]$, die diesem Eingang bzw. Ausgang zugeordnet sind. Die Prozessstückzahlen beschreiben im Verständnis von Abschnitt 2.5 nichtnegative reelle Variablen und werden durch einen fortlaufenden natürlichen Index a größer als null bezeichnet.

Am Ausgang der Quelle σ_b jedes Produkts wird eine neue Prozessstückzahl $x_{a'}$ eingeführt, die jeweils der Produktstückzahl y_b entspricht. Freiheitsgrade in Gestalt zusätzlicher Prozessstückzahlen entstehen durch alternative Verknüpfungen (siehe Definition 2.6). Mit jeder ausgehenden Kante an einem alternativen Flusspunkt wird entsprechend eine neue Prozessstückzahl $x_{a'}$ erzeugt. Im Unterschied dazu definiert eine selektive Verknüpfung ein fixes Verhältnis zwischen den Stückzahlanteilen, welche die Operanden der Verknüpfung durchlaufen (siehe Definition 2.7). Folglich liegt an jedem Ausgang eines selektiven Flusspunkts

ein fixer relativer Anteil der Stückzahl vom Eingang an, welcher die fix vorgegebene Quote am jeweiligen Ausgang widerspiegelt. Ressourcen bewirken keine Veränderung der Stückzahlen, weshalb der Stückzahlanteil am Eingang i einer Ressource dem gegenüberliegenden Ausgang j mit gleichem Index zugeordnet wird.

Damit verbleiben sequenzielle Verknüpfungen als einfachster Typ zur Verteilung der Stückzahl auf zwei Operanden (siehe Definition 2.5). Eine sequenzielle Verknüpfung wird durch eine Kante abgebildet, die stets zwei Knoten v und v' miteinander verbindet. Durch die Verbindung der Knoten wird der Stückzahlanteil vom Ausgang j des ersten Knotens v zum Eingang i' des zweiten Knotens v' übertragen, ohne diesen zu verändern.

Eine Besonderheit bilden Vereinigungen von Kanten am Eingang eines Knotens. Sofern es sich nicht um die Senke τ_b eines Wertstromgraphen $G_{\mathrm{WS},b}$ handelt, werden an solchen Punkten ebenfalls neue Prozessstückzahlen $x_{a'}$ eingeführt. Diese müssen jeweils mit der Summe der Stückzahlen an den vereinigten Kanten übereinstimmen und stellen folglich keine Freiheitsgrade dar. Die Einführung neuer Prozessstückzahlen an Vereinigungen ist notwendig, um jeder nachfolgenden Kante an dem jeweils betreffenden Knoten genau eine Prozessstückzahl x_a und einen Koeffizienten q zuzuordnen.

Bemerkung: Im weiteren Verlauf wird von Stückzahlen gesprochen, die an Eingängen, Ausgängen und Kanten anliegen. Diese beziehen sich stets auf Anteile der Stückzahl desselben Produkts, da die Begriffe eines Eingangs und eines Ausgangs eines Knotens nur innerhalb des Wertstromgraphen $G_{\mathrm{WS},b}$ eines Produkts b Gültigkeit besitzen.

Bedingungen in Bezug auf die Variablen. Um die unterschiedlichen Verknüpfungen und Vereinigungen abzubilden, sind Bedingungen erforderlich, welche die Werte der Prozessstückzahlen beschränken. Grundsätzlich muss die *Produktstruktur* berücksichtigt werden, die festlegt, welche Stückzahlen zur Herstellung von übergeordneten Produkten benötigt werden. Betrachtet man die Wertstromgraphen $G_{\mathrm{WS},b}$ der Produkte b, muss eine Zuordnung von den Prozessstückzahlen x_a am Ausgang der Quellen σ_b zu den Produktstückzahlen y_b definiert werden. An Vereinigungen und an alternativen Flusspunkten müssen die eingehenden und ausgehenden Stückzahlen insgesamt stets gleich sein, bezeichnet als *Flusserhalt*. Da Ressourcen eine endliche Betriebsmittelzeit besitzen, müssen die Stückzahlen an den Eingängen in die Berechnung der *Auslastung* einfließen.

Prozessmatrix P. Wie zuvor bereits angesprochen, muss eine Zuordnung zwischen dem Vektor $\mathbf{y} = [y_1 \ \dots \ y_n]^\top$ der Produktstückzahlen y_b und dem Vektor $\mathbf{x} = [x_1 \ \dots \ x_m]^\top$ der Prozessstückzahlen x_a hergestellt werden. Hierzu wird die Prozessmatrix $\mathbf{P}$ mit n Zeilen und m Spalten definiert. Eine Komponente $\mathbf{P}[b,a]$ in Zeile b und Spalte a ist genau dann gleich eins, wenn die Prozessstückzahl x_a am Ausgang der Quelle σ_b des Wertstromgraphen $G_{\mathrm{WS},b}$ von Produkt b anliegt. In diesem Fall entspricht die Prozessstückzahl x_a der Produktstückzahl y_b. In jedem anderen Fall existiert keine solche Zuordnung, und die Komponente $\mathbf{P}[b,a]$ der Matrix ist entsprechend gleich null. Der Vektor $\mathbf{y}$ der Produktstückzahlen y_b lässt sich nach dieser Festlegung als Ergebnis einer Matrixmultiplikation ausdrücken:

$$\mathbf{y} = \mathbf{P}\mathbf{x}. \tag{4.1}$$

Produktstrukturmatrix S. Die Matrix beschreibt die Zuordnung von Komponenten, d. h. von untergeordneten Produkten, zu übergeordneten Produkten und bildet damit die Strukturstückliste jedes Produkts ab. Zu diesem Zweck ist die Produktstrukturmatrix $\mathbf{S}$ als Matrix nichtnegativer reeller Komponenten in n Zeilen und n Spalten definiert. Die Komponente $\mathbf{S}[b, b']$ in Zeile b und Spalte b' drückt aus, dass ein Produkt b zu $\mathbf{S}[b, b']$ Stück in Produkt b' enthalten ist. Somit entspricht der Wert $\mathbf{S}[b, b']$ dem Stücklistenfaktor des gegebenen Produkts b im übergeordneten Produkt b'. Die Information ist wichtig, um den Sekundärbedarf an Produkten zur Herstellung von übergeordneten Produkten zu bestimmen. Dieser resultiert aus der Matrixmultiplikation $\mathbf{Sy}$ oder, nach Einsetzen der Prozessmatrix $\mathbf{P}$ aus Gleichung (4.1), dem Ausdruck $\mathbf{SPx}$. Daraus folgt die Bedingung in Bezug auf die *Produktstruktur*, die besagt, dass die Stückzahlen der Produkte $\mathbf{Px}$ nicht kleiner als die Werte sein dürfen, die zur Deckung des Sekundärbedarfs notwendig sind:

$$\textit{Produktstruktur:} \qquad (\mathbf{I} - \mathbf{S})\,\mathbf{Px} \geqslant \mathbf{0}\,. \tag{4.2}$$

Darüber hinaus können mit Hilfe der Produktstrukturmatrix $\mathbf{S}$ Zyklen in der Produktstruktur erkannt werden, wie nachfolgend gezeigt werden soll.

Theorem 4.3: Zyklische Produktstruktur. Gegeben sei eine Zahl n von mindestens zwei Produkten und eine Produktstrukturmatrix $\mathbf{S}$, wie oben definiert. Die Struktur eines Produkts b ist genau dann auf einer Stufe N zyklisch, wenn die Matrix $\mathbf{S}^N$ in Zeile b und Spalte b einen positiven Wert enthält. Hierbei bezeichnet die Stufe N eine natürliche Zahl größer als null und kleiner als oder gleich n. Das heißt, die Produktstruktur ist im Allgemeinen zyklisch, wenn diese Bedingung für mindestens ein Produkt b und eine Stufe N erfüllt ist.

Beweis. Im Fall einer Stufe N gleich eins ist das Theorem durch die Definition der Produktstrukturmatrix $\mathbf{S}$ erfüllt, sodass der Fall einer Stufe N größer als eins verbleibt. Zu diesem Zweck wird eine Folge von Produkten $(b_1, b_2, \ldots, b_N, b_1)$ angenommen. Das Produkt b_1 sei in b_2 enthalten, das Produkt b_2 in b_3 usw. usf. Die Folge wird bis zum Produkt b_N fortgesetzt, das wiederum im Produkt b_1 enthalten sein soll. Das heißt, die Produkte bilden einen Zyklus auf der Stufe N. Die Produktstruktur ist genau dann auf der Stufe N zyklisch, wenn mindestens eine solche Folge existiert. Formal lautet die Bedingung:

$$\begin{aligned} \exists b_2, \ldots, b_N \in \{1, \ldots, n\} \qquad & \mathbf{S}[b_1, b_2] > 0 \wedge \mathbf{S}[b_2, b_3] > 0 \wedge \ldots \\ & \wedge\, \mathbf{S}[b_{N-1}, b_N] > 0 \wedge \mathbf{S}[b_N, b_1] > 0\,. \end{aligned} \tag{4.3}$$

Da die Matrix $\mathbf{S}$ ausschließlich nichtnegative Komponenten enthält, liefert die Multiplikation der Komponenten genau dann ein positives Ergebnis, wenn jeder einzelne Wert größer als null ist. Auf dieser Basis lässt sich Gleichung (4.3) wie folgt umformulieren:

$$\exists b_2, \ldots, b_N \in \{1, \ldots, n\} \qquad \mathbf{S}[b_1, b_2]\,\mathbf{S}[b_2, b_3] \ldots \mathbf{S}[b_{N-1}, b_N]\,\mathbf{S}[b_N, b_1] > 0\,. \tag{4.4}$$

Ist eine Folge von Produkten $(b_1, b_2, \ldots, b_N, b_1)$ gegeben, welche die Bedingung für einen Zyklus erfüllen, dann ist der obige Ausdruck größer als null. In jedem anderen Fall ist der

Ausdruck gleich null. Daher muss genau dann eine solche Folge existieren, wenn die Summe über alle Folgen größer als null ist. Gleichung (4.4) ist somit äquivalent zu:

$$\sum_{b_N=1}^{n} \sum_{b_{N-1}=1}^{n} \dots \sum_{b_2=1}^{n} \left[\mathbf{S}[b_1, b_2]\, \mathbf{S}[b_2, b_3] \dots \mathbf{S}[b_{N-1}, b_N]\, \mathbf{S}[b_N, b_1] \right] > 0\,. \tag{4.5}$$

Durch geeignetes Setzen der Klammern lässt sich Gleichung (4.5) unter Berücksichtigung der Laufvariablen in jeder Summe wie folgt darstellen:

$$\sum_{b_N=1}^{n} \left[\sum_{b_{N-1}=1}^{n} \left[\dots \sum_{b_2=1}^{n} \left[\mathbf{S}[b_1, b_2]\ \mathbf{S}[b_2, b_3] \right] \dots \mathbf{S}[b_{N-1}, b_N] \right] \mathbf{S}[b_N, b_1] \right] > 0\,. \tag{4.6}$$

Wenn man die multiplizierten Ausdrücke $\mathbf{S}[b_1, b_2]\, \mathbf{S}[b_2, b_3]$ über alle möglichen Produkte b_2 aufaddiert, entspricht das Ergebnis der Komponente $\mathbf{S}^2[b_1, b_3]$ des Quadrats der Matrix $\mathbf{S}$ in Zeile b_1 und Spalte b_3. Eingesetzt in Gleichung (4.6) folgt daraus:

$$\sum_{b_N=1}^{n} \left[\sum_{b_{N-1}=1}^{n} \left[\dots \sum_{b_3=1}^{n} \left[\mathbf{S}^2[b_1, b_3]\, \mathbf{S}[b_3, b_4] \right] \dots \mathbf{S}[b_{N-1}, b_N] \right] \mathbf{S}[b_N, b_1] \right] > 0\,. \tag{4.7}$$

Analog lässt sich dies bis zur Potenz $N-2$ fortsetzen. Die Summen werden schrittweise von innen nach außen aufgelöst, sodass der folgende Ausdruck verbleibt:

$$\sum_{b_N=1}^{n} \left[\sum_{b_{N-1}=1}^{n} \left[\mathbf{S}^{N-2}[b_1, b_{N-1}]\, \mathbf{S}[b_{N-1}, b_N] \right] \mathbf{S}[b_N, b_1] \right] > 0\,. \tag{4.8}$$

Die Ersetzung der letzten zwei Summen liefert schließlich den gesuchten Ausdruck, welcher nur noch vom Produkt b_1 und der Stufe N abhängig ist:

$$\sum_{b_N=1}^{n} \left[\mathbf{S}^{N-1}[b_1, b_N]\, \mathbf{S}[b_N, b_1] \right] > 0\,. \tag{4.9}$$

Daraus folgt:

$$\mathbf{S}^N[b_1, b_1] > 0\,. \tag{4.10}$$

Das heißt, die Produktstruktur ist gemäß der Matrix $\mathbf{S}$ genau dann zyklisch, wenn mindestens ein Produkt b_1 auf einer Stufe N die zu zeigende Bedingung erfüllt. □

Vereinigungsmatrix U. Wenn sich im Wertstromgraphen $G_{\mathrm{WS},b}$ eines Produkts b Kanten an einem Eingang eines Knotens vereinigen, dann muss, wie oben erläutert, eine neue Prozessstückzahl $x_{a'}$ erzeugt werden. Diese Information wird durch die Vereinigungsmatrix $\mathbf{U}$ abgebildet, die als Matrix mit reellen Komponenten in m Zeilen und m Spalten definiert ist. Die Komponenten $\mathbf{U}[a', a]$ in den Zeilen a' und den Spalten a sind entweder gleich minus eins oder nicht negativ und höchstens gleich eins. Ist der Wert in einer Zeile a' und einer entsprechenden Spalte gleich minus eins, dann drückt dies aus, dass die Prozessstückzahl $x_{a'}$ als Folge sich vereinigender Kanten an einem Eingang eines Knotens erzeugt wird. Jede

Zeile, die nicht dem Nullvektor entspricht, repräsentiert somit eine Vereinigung von Kanten. Die Werte in der Zeile a' und den verbleibenden Spalten a geben jeweils den relativen Anteil der Prozessstückzahl x_a an, welcher insgesamt an den sich vereinigenden Kanten anliegt. Bezüglich der Notation wird für die Zeile a' die Kennzeichnung mit einem Akzent gewählt, da es sich um eine neue Prozessstückzahl $x_{a'}$ handelt.

Die Vereinigungsmatrix $\mathbf{U}$ wird als Hilfsmatrix genutzt, um die Komponenten der Flussmatrix $\mathbf{F}$ zu bestimmen. Der erste Teil des Flusserhalts, welcher unter Verwendung der Vereinigungsmatrix $\mathbf{U}$ formuliert wird, betrifft die Stückzahlen vor und nach Vereinigungen von Kanten. Demnach muss die Summe der Stückzahlen an sich vereinigenden Kanten stets mit der Stückzahl nach der Vereinigung übereinstimmen. Wird eine Prozessstückzahl $x_{a'}$ durch eine Vereinigung erzeugt, muss das Ergebnis der Matrixmultiplikation $\mathbf{U}[a', *]\,\mathbf{x}$ entsprechend gleich null sein, wobei der Ausdruck $\mathbf{U}[a', *]$ den Vektor an Zeile a' der Matrix bezeichnet. Betrachtet man alle Prozessstückzahlen $x_{a'}$, welche durch Vereinigungen erzeugt werden, muss daher für den Vektor $\mathbf{x}$ gelten:

$$\mathbf{U}\,\mathbf{x} = \mathbf{0}\,. \tag{4.11}$$

Eingangsmatrizen $\mathbf{Q}_{\mathbf{ein},bv}$ und Ausgangsmatrizen $\mathbf{Q}_{\mathbf{aus},bv}$. Die Matrizen bilden die Information ab, welche Anteile der Stückzahl y_b eines Produkts b an den Eingängen bzw. den Ausgängen eines Knotens v im Wertstromgraphen $G_{\mathrm{WS},b}$ anliegen. Hierzu sind die Eingangsmatrizen $\mathbf{Q}_{\mathrm{ein},bv}$ als Matrizen nichtnegativer reeller Komponenten in einer beliebigen Zahl von Zeilen und m Spalten definiert. Die Komponente $\mathbf{Q}_{\mathrm{ein},bv}[i,a]$ in Zeile i und Spalte a drückt dabei aus, dass der Anteil $\mathbf{Q}_{\mathrm{ein},bv}[i,a]\,x_a$ der Produktstückzahl y_b den Eingang i des Knotens v durchläuft. In den Wertstromgraphen $G_{\mathrm{WS},b}$ ist jedem Knoten v der Menge $V_{\mathrm{WS},b}$ außer der Quelle σ_b und der Senke τ_b eine solche Matrix zugeordnet. Jede Zeile i der Matrix entspricht genau einem Eingang des betreffenden Knotens v.

Analog dazu sind die Ausgangsmatrizen $\mathbf{Q}_{\mathrm{aus},bv}$ als Matrizen mit nichtnegativen reellen Komponenten in einer beliebigen Zahl von Zeilen und m Spalten definiert. In diesem Fall lautet die Festlegung, dass der Anteil $\mathbf{Q}_{\mathrm{aus},bv}[j,a]\,x_a$ der Produktstückzahl y_b den Ausgang j des Knotens v durchläuft. Das heißt, jede Zeile j entspricht genau einem Ausgang des jeweils bezeichneten Knotens v. Im Gegensatz zu den vorigen werden die Ausgangsmatrizen $\mathbf{Q}_{\mathrm{aus},bv}$ nur für die alternativen Flusspunkte $D_{\mathrm{AL},b}$ benötigt.

Der Stückzahlanteil eines Produkts b, welcher an einem Eingang i bzw. Ausgang j anliegt, lässt sich auch ohne explizite Definition des Index a der betreffenden Prozessstückzahl x_a bestimmen. Da die Komponenten der Matrizen in allen anderen Spalten definitionsgemäß gleich null sind, ist die Stückzahl an Eingang i bzw. an Ausgang j das Ergebnis der Matrixmultiplikation $\mathbf{Q}_{\mathrm{ein},bv}[i,*]\,\mathbf{x}$ bzw. $\mathbf{Q}_{\mathrm{aus},bv}[j,*]\,\mathbf{x}$. Dabei bezeichnet der Ausdruck $\mathbf{Q}_{\mathrm{ein},bv}[i,*]$ bzw. $\mathbf{Q}_{\mathrm{aus},bv}[j,*]$ den Vektor an Zeile i bzw. an Zeile j der jeweiligen Matrix.

Die Matrizen werden als Hilfsmatrizen benötigt, um die Flussmatrix $\mathbf{F}$, die Auslastungsmatrix $\mathbf{L}$ und die Investitionsmatrix $\mathbf{L}_{\mathrm{Invest}}$ zu bestimmen. Zunächst lässt sich mit ihrer Hilfe der zweite Teil der Bedingung des Flusserhalts formulieren, der sich auf die Stückzahlen an den Eingängen und Ausgängen der alternativen Flusspunkte $D_{\mathrm{AL},b}$ bezieht. Demnach muss die Stückzahl am Eingang solcher Knoten der Summe der Stückzahlen an den Ausgängen entsprechen. Indem man durch die Multiplikation mit einem vorangestellten transponierten

Einsvektor die Stückzahlen $\mathbf{Q}_{\text{aus},bv}[j,*]\,\mathbf{x}$ an allen Ausgängen j addiert, muss jeweils für alle alternativen Flusspunkte v in den Wertstromgraphen $G_{\text{WS},b}$ gelten:

$$\forall\, b \in \{1,\ldots,n\}\ \forall\, v \in D_{\text{AL},b} \qquad \left[\mathbf{Q}_{\text{ein},bv} - \mathbf{1}^{\top}\mathbf{Q}_{\text{aus},bv}\right]\mathbf{x} = \mathbf{0}\,. \tag{4.12}$$

Flussmatrix F. Die Vereinigungsmatrix **U**, die Mengen der Eingangsmatrizen $\{\mathbf{Q}_{\text{ein},bv}\}$ und die Mengen der Ausgangsmatrizen $\{\mathbf{Q}_{\text{aus},bv}\}$ werden zur Flussmatrix **F** zusammengeführt. Sowohl an Vereinigungen von Kanten als auch an alternativen Flusspunkten muss der Flusserhalt gelten, wie weiter oben in Gleichung (4.11) bzw. (4.12) definiert. Auf dieser Grundlage wird die Flussmatrix **F** gebildet, indem die Koeffizienten der einzelnen Bedingungen fortlaufend in jeweils einer neuen Zeile ergänzt werden. Jede Zeile der Matrix drückt folglich die Differenz der Stückzahlen vor und nach einer Vereinigung oder am Eingang und an den Ausgängen eines alternativen Flusspunkts aus. Der *Flusserhalt* ist genau dann gewährleistet, wenn für einen Vektor **x** von Prozessstückzahlen x_a gilt:

$$\textit{Flusserhalt:} \qquad \mathbf{F}\mathbf{x} = \mathbf{0}\,. \tag{4.13}$$

Auslastungsmatrix L und Investitionsmatrix $\mathbf{L}_{\text{Invest}}$. Mit den Eingangsmatrizen $\mathbf{Q}_{\text{ein},bv}$, welche die Stückzahlen an den Eingängen eines Knotens angeben, den effektiven Taktzeiten t_{eff} an jedem Eingang einer Ressource und der Betriebsmittelzeit T_{BM} ist es möglich, die Auslastung der genutzten Ressourcen zur Herstellung der Produktstückzahlen y_b ermitteln. Hierzu sind die Auslastungsmatrix **L** und die Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ als Matrizen mit nichtnegativen reellen Komponenten in einer beliebigen Zahl von Zeilen und m Spalten definiert. Jede Zeile i der Matrizen ist genau einer Ressource zugeordnet. Gemäß dieser Zuordnung entspricht das Ergebnis der Matrixmultiplikation $\mathbf{L}[i,*]\,\mathbf{x}$ bzw. $\mathbf{L}_{\text{Invest}}[i,*]\,\mathbf{x}$ der Auslastung $u_r(\mathbf{x})$ der Ressource an der gegebenen Zeile i. Hierbei bezeichnet der Ausdruck $\mathbf{L}[i,*]$ bzw. $\mathbf{L}_{\text{Invest}}[i,*]$ den Vektor an Zeile i der jeweiligen Matrix. Damit kann die Bedingung in Bezug auf die *Auslastung* formuliert werden, welche den Vektor **x** der Prozessstückzahlen x_a begrenzt. Die Schranke ist vom Planungsziel abhängig und im Fall der Suche maximaler Kapazitäten gleich eins, sodass formal gilt:

$$\textit{Auslastung:} \qquad \mathbf{L}\mathbf{x} \leqslant \mathbf{1}\,. \tag{4.14}$$

Die Auslastungsmatrix **L** unterscheidet sich von der Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ dahingehend, dass erstere sämtliche Eingänge an einer Ressource berücksichtigt. Letztere bezieht dagegen nur solche Eingänge ein, die nicht erreichbar sind, indem man an jedem alternativen Flusspunkt stets die erste ausgehende Kante weiterverfolgt. Hintergrund ist, dass all diejenigen Eingänge, welche nach den ersten ausgehenden Kanten folgen, Prozessschritte repräsentieren, die zu einer Überlastung von Ressourcen führen dürfen. Durch geeignete Priorisierung ausgehender Kanten an alternativen Flusspunkten legt der Anwender fest, welche Ressourcen im Bedarfsfall überlastet werden sollen. Anschaulich gesprochen, ist der Weg entlang der ersten ausgehenden Kanten alternativer Flusspunkte in einem Wertstromgraphen $G_{\text{WS},b}$ von der Quelle σ_b bis zur Senke τ_b der Weg, welcher immer dann genutzt werden soll, sofern die vorhandenen Kapazitäten nicht ausreichen.

Um dies umzusetzen, muss eine Bedingung formuliert werden, nach der nur diejenigen Prozessschritte genutzt werden sollen, welche eine Überlastung der jeweiligen Ressource bewirken dürfen. Die entsprechende Bedingung wird angewandt, um für die Minimierung der Überlastung eine initiale Lösung $\mathbf{x}$ der Prozessstückzahlen x_a zu bestimmen, und wird im nächsten Kapitel aufgegriffen. Diese lautet:

$$\mathbf{L}_{\text{Invest}}\,\mathbf{x} = \mathbf{0}\,. \tag{4.15}$$

***MaxFolge* der Prozessstückzahlen x_a.** Falls die Möglichkeit besteht, verschiedene Ressourcen zu nutzen, muss der Anwender eine Festlegung treffen, welche dieser Ressourcen bevorzugt ausgelastet werden sollen. Hierzu definiert der Anwender für alle ausgehenden Kanten alternativer Flusspunkte entsprechende Prioritäten. Damit ist jedoch nur eine Ordnung zwischen den Prozessstückzahlen hergestellt, die an den ausgehenden Kanten eines alternativen Flusspunkts anliegen. Um eine vollständige Ordnung aller Prozessstückzahlen zu definieren, wird zusätzlich die Information einbezogen, von welcher ausgehenden Kante eines alternativen Flusspunkts die nachfolgenden Knoten erreichbar sind. Wenn zwei Eingänge eines oder zweier verschiedener Knoten betrachtet werden, in beiden Fällen der Weg von der Quelle über denselben alternativen Flusspunkt führt und im ersten Fall eine ausgehende Kante mit höherer Priorität beschritten wird, dann muss die Prozessstückzahl am erstgenannten Eingang in der Sortierung vor jener am zweiten Eingang folgen. Diese Information kann gewonnen werden, indem ausgehend von der Quelle σ_b in jedem Wertstromgraphen $G_{\text{WS},b}$ eine Tiefensuche durchgeführt wird, in deren Verlauf die jeweils höher priorisierten Ausgänge alternativer Flusspunkte zuerst beschritten werden.

In der daraus resultierenden Sortierung werden die Indizes a der Prozessstückzahlen x_a in die *MaxFolge* $(a_1, a_2, \dots)$ eingefügt. Die Elemente der Folge bilden in ihrer Gesamtheit eine Teilmenge aller Indizes a. Die Sortierung wird genutzt, um die Prozessstückzahlen iterativ zu maximieren und dadurch die Auslastung gemäß dem entsprechenden Planungsziel zu optimieren, wie im nächsten Kapitel vertieft wird.

Durch die Transformation eines gegebenen grafischen Modells $\{G_{\text{WS},b}\}$ in ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ bleiben somit alle Elemente und die Beziehungen zwischen diesen Elementen erhalten. Die Matrizen drücken Bedingungen aus, welche den Vektor $\mathbf{x}$ der Prozessstückzahlen x_a beschränken. Dies betrifft die Zuordnung zu den Produktstückzahlen y_b sowie die Bedingungen in Bezug auf die Produktstruktur, den Flusserhalt und die Auslastung. Die *MaxFolge* entspricht der Reihenfolge, in welcher die Prozessstückzahlen x_a laut der Festlegung des Anwenders priorisiert werden müssen.

Illustratives Beispiel. Um die Definition der Matrizen und der *MaxFolge* zu veranschaulichen, sollen die Werte für das Beispiel in Abbildung 4.13 bestimmt werden. Gegeben ist ein grafisches Modell $\{G_{\text{WS},b}\}$ für zwei Produkte, deren Wertstromgraphen auf den Beispielen AL2 bzw. SL2 aus Abschnitt 2.5 basieren. Es ist das mathematische Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ gesucht, welches dem grafischen Modell entspricht. Aus der Zahl der Produkte folgt ein Vektor $\mathbf{y} = [y_1\ y_2]^\top$ von zwei Produktstückzahlen y_b. Nach Traversierung der Wertstromgraphen $G_{\text{WS},b}$ steht fest, dass ein Vektor $\mathbf{x} = [x_1\ \dots\ x_7]^\top$ von sieben Prozessstückzahlen x_a

nötig ist, um die Stückzahlverteilung in den Wertströmen abzubilden. Zunächst werden den Produktstückzahlen y_b die Prozessstückzahlen x_a zugeordnet, die im Wertstromgraphen $G_{\mathrm{WS},b}$ des jeweiligen Produkts b an der Quelle σ_b anliegen:

$$y_1 = x_1\,, \qquad y_2 = x_5\,. \tag{4.16a, 4.16b}$$

Die Zuordnung der Prozessstückzahlen x_a aus Gleichung (4.16a, 4.16b) lässt sich mit Hilfe einer entsprechenden Prozessmatrix $\mathbf{P}$ wie folgt ausdrücken:

$$\mathbf{y} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{x} = \mathbf{P}\mathbf{x}. \tag{4.17}$$

In dem Beispiel seien jeweils zwei Stück des ersten Produkts zur Herstellung des zweiten Produkts erforderlich. Zur Deckung des Sekundärbedarfs muss gelten:

$$y_1 \geqslant 2\, y_2\,. \tag{4.18}$$

Durch Formulierung einer geeigneten Produktstrukturmatrix $\mathbf{S}$ kann Gleichung (4.18) als Ergebnis einer Matrixmultiplikation dargestellt werden:

$$\mathbf{y} \geqslant \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix} \mathbf{y} = \mathbf{S}\mathbf{y}. \tag{4.19}$$

Damit richtet sich der Blick auf die Verteilung der Produktstückzahlen y_b. Die Transformation des grafischen Modells $\{G_{\mathrm{WS},b}\}$ führt zu der in Abbildung 4.13 dargestellten Zuordnung von Prozessstückzahlen x_a und Koeffizienten q zu den Eingängen und Ausgängen der Knoten $V_{\mathrm{WS},b}$. Die Eingangsmatrizen $\mathbf{Q}_{\mathrm{ein},1,v}$ und Ausgangsmatrizen $\mathbf{Q}_{\mathrm{aus},1,v}$ der Knoten v im Wertstromgraphen $G_{\mathrm{WS},1}$ des ersten Produkts lauten wie folgt:

$$\mathbf{Q}_{\mathrm{ein},1,v_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4.20}$$

$$\mathbf{Q}_{\mathrm{aus},1,v_1} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4.21}$$

$$\mathbf{Q}_{\mathrm{ein},1,r_1} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \tag{4.22}$$

$$\mathbf{Q}_{\mathrm{ein},1,r_2} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.23}$$

Die Eingangsmatrizen $\mathbf{Q}_{\mathrm{ein},2,v}$ und Ausgangsmatrizen $\mathbf{Q}_{\mathrm{aus},2,v}$ der Knoten v im Wertstromgraphen $G_{\mathrm{WS},2}$ des zweiten Produkts nehmen die folgenden Werte an:

$$\mathbf{Q}_{\mathrm{ein},2,v_2} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \tag{4.24}$$

$$\mathbf{Q}_{\mathrm{ein},2,v_3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0{,}8 & 0 & 0 \end{bmatrix}, \tag{4.25}$$

$$\mathbf{Q}_{\mathrm{aus},2,v_3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.26}$$

$$\mathbf{Q}_{\text{ein},2,r_3} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \tag{4.27}$$

$$\mathbf{Q}_{\text{ein},2,r_4} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0{,}2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.28}$$

Im Wertstromgraphen $G_{\text{WS},1}$ werden die beiden Kanten, die von den Ressourcen $r_{1/2}$ ausgehen, vor dem zweiten Eingang der Ressource r_1 vereinigt. Vor der Vereinigung liegen an den zwei Kanten die Prozessstückzahlen $x_{2/3}$ an, nach der Vereinigung vor dem Eingang die Prozessstückzahl x_4. Um den Flusserhalt zu gewährleisten, muss gelten:

$$x_2 + x_3 - x_4 = 0\,. \tag{4.29}$$

Die Bedingung in Gleichung (4.29) lässt sich durch geeignete Formulierung einer Vereinigungsmatrix $\mathbf{U}$ wie unten angegeben ausdrücken. Die Nullzeilen sind dadurch zu erklären, dass nach Erzeugung einer Prozessstückzahl $x_{a'}$ an einer Vereinigung von Kanten die Bedingung in der Zeile a' ergänzt wird. Auf diese Weise kann mit dem Index a' der Prozessstückzahl die zugehörige Zeile wiedergefunden werden. Damit gilt:

$$\mathbf{U}\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 & 0 \end{bmatrix} \mathbf{x} = \mathbf{0}\,. \tag{4.30}$$

Der Flusserhalt muss zudem an allen alternativen Flusspunkten $D_{\text{AL},b}$ gelten. Die Bedingungen sind für die Knoten $v_{1/3}$ nachfolgend angegeben. Die Indizes a der Prozessstückzahlen x_a und die Koeffizienten q stammen aus den obigen Eingangsmatrizen $\mathbf{Q}_{\text{ein},bv}$ und Ausgangsmatrizen $\mathbf{Q}_{\text{aus},bv}$. Diese führen zu folgendem Ergebnis:

$$x_1 - x_2 - x_3 = 0\,, \qquad 0{,}8\,x_5 - x_6 - x_7 = 0\,. \tag{4.31a, 4.31b}$$

Die Zusammenfassung von Gleichung (4.30) sowie von Gleichung (4.31a, 4.31b) liefert die Bedingung in Bezug auf den Flusserhalt. Diese lässt sich mit Hilfe einer entsprechenden Flussmatrix $\mathbf{F}$ in dem Beispiel folgendermaßen darstellen:

$$\mathbf{F}\mathbf{x} = \begin{bmatrix} 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}8 & -1 & -1 \end{bmatrix} \mathbf{x} = \mathbf{0}\,. \tag{4.32}$$

Die Stückzahlen werden durch die endliche Auslastung der MAEs begrenzt. Die Schranke ist vom Planungsziel abhängig und im Fall der Maximierung der Kapazitäten gleich eins. Bezüglich der effektiven Taktzeiten t_{eff} und Betriebsmittelzeiten T_{BM} werden Indizes gewählt, welche den Bezeichnungen in den Beispielen AL2 und SL2 aus Abschnitt 2.5 entsprechen. Damit gilt für die Auslastung $u_r(\mathbf{x})$ der vier MAEs jeweils die Bedingung:

$$u_{r_1}(\mathbf{x}) = \frac{t_{\text{eff},1A}}{T_{\text{BM},1}}\, x_2 + \frac{t_{\text{eff},2}}{T_{\text{BM},1}}\, x_4 \leqslant 1\,, \tag{4.33}$$

$$u_{r_2}(\mathbf{x}) = \frac{\mathrm{t_{eff,1B}}}{\mathrm{T_{BM,2}}}\, x_3 \leqslant 1\,, \tag{4.34}$$

$$u_{r_3}(\mathbf{x}) = \frac{\mathrm{t_{eff,3HA}}}{\mathrm{T_{BM,3}}}\, x_6 \leqslant 1\,, \tag{4.35}$$

$$u_{r_4}(\mathbf{x}) = 0{,}2\,\frac{\mathrm{t_{eff,3L}}}{\mathrm{T_{BM,4}}}\, x_5 + \frac{\mathrm{t_{eff,3HB}}}{\mathrm{T_{BM,4}}}\, x_7 \leqslant 1\,. \tag{4.36}$$

Die obigen Bedingungen aus Gleichung (4.33) bis (4.36) können wie die vorigen als Ergebnis einer Matrixmultiplikation dargestellt werden. Hierzu werden die effektiven Taktzeiten $\mathrm{t_{eff}}$ und die Stückzahlen an den Eingängen aller Ressourcen mit den jeweiligen Betriebsmittelzeiten $\mathrm{T_{BM}}$ in Beziehung gesetzt, um eine entsprechende Auslastungsmatrix $\mathbf{L}$ zu formulieren. Daraus folgt die Bedingung in Bezug auf die Auslastung:

$$\mathbf{L}\mathbf{x} = \begin{bmatrix} 0 & \frac{\mathrm{t_{eff,1A}}}{\mathrm{T_{BM,1}}} & 0 & \frac{\mathrm{t_{eff,2}}}{\mathrm{T_{BM,1}}} & 0 & 0 & 0 \\ 0 & 0 & \frac{\mathrm{t_{eff,1B}}}{\mathrm{T_{BM,2}}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\mathrm{t_{eff,3HA}}}{\mathrm{T_{BM,3}}} & 0 \\ 0 & 0 & 0 & 0 & 0{,}2\,\frac{\mathrm{t_{eff,3L}}}{\mathrm{T_{BM,4}}} & 0 & \frac{\mathrm{t_{eff,3HB}}}{\mathrm{T_{BM,4}}} \end{bmatrix} \mathbf{x} \leqslant \mathbf{1}\,. \tag{4.37}$$

Die Investitionsmatrix $\mathbf{L}_{\mathrm{Invest}}$ entspricht der Auslastungsmatrix $\mathbf{L}$ aus Gleichung (4.37), wobei Eingänge von Ressourcen, die nach den ersten Ausgängen alternativer Flusspunkte folgen, unberücksichtigt bleiben. Um nur Prozessschritte zu nutzen, die gemäß der Festlegung des Anwenders zu einer Überlastung von Ressourcen führen dürfen, muss der Ausdruck $\mathbf{L}_{\mathrm{Invest}}\,\mathbf{x}$ gleich null sein. In dem vorliegenden Beispiel lautet die Bedingung:

$$\mathbf{L}_{\mathrm{Invest}}\,\mathbf{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\mathrm{t_{eff,1B}}}{\mathrm{T_{BM,2}}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{\mathrm{t_{eff,3HB}}}{\mathrm{T_{BM,4}}} \end{bmatrix} \mathbf{x} = \mathbf{0}\,. \tag{4.38}$$

Abschließend wird die *MaxFolge* bestimmt. Sofern zuerst der Wertstromgraph $G_{\mathrm{WS},1}$ transformiert wird und alle Ausgänge der zwei alternativen Flusspunkte $v_{1/3}$ berücksichtigt werden, entspricht die *MaxFolge* der Folge $(2,3,6,7)$. Gemäß der Sortierung der Indizes a, welche die Priorisierung des Anwenders abbildet, müssen die jeweiligen Prozessstückzahlen x_a iterativ maximiert werden, um die Auslastung zu optimieren.

Nachtrag zur Eindeutigkeit der Priorisierung. An Ausgängen alternativer Flusspunkte entstehen Prozessstückzahlen, welche durch die Sortierung der ausgehenden Kanten priorisiert werden. Um eine Ordnung zwischen allen Prozessstückzahlen herzustellen, wird zusätzlich berücksichtigt, welche ausgehenden Kanten an alternativen Flusspunkten von der Quelle

Index	Definition	Stückzahlverteilung								Auslastung				
		x_1	x_2	x_3	x_4	x_5	x_6	x_7	y	$u_{r_1}(\mathbf{x})$	$u_{r_2}(\mathbf{x})$	$u_{r_3}(\mathbf{x})$	$u_{r_4}(\mathbf{x})$	$u_{r_5}(\mathbf{x})$
(1)	$\max\{x_2 : \mathbf{x} \geqslant \mathbf{0} \wedge \mathbf{Px} = \mathbf{y}_{\text{plan}} \wedge \mathbf{Fx} = \mathbf{0} \wedge \mathbf{Lx} \leqslant \mathbf{1}\}$	200	**100**	0	100	**0**	100	100	200	0,00	1,00	1,00	1,00	1,00
(2)		200	**75**	25	100	**25**	75	100	200	0,25	1,00	0,75	1,00	1,00
(3)	⋮	200	**50**	50	100	**50**	50	100	200	0,50	1,00	0,50	1,00	1,00
(4)		200	**25**	75	100	**75**	25	100	200	0,75	1,00	0,25	1,00	1,00
(5)	$\max\{x_5 : \mathbf{x} \geqslant \mathbf{0} \wedge \mathbf{Px} = \mathbf{y}_{\text{plan}} \wedge \mathbf{Fx} = \mathbf{0} \wedge \mathbf{Lx} \leqslant \mathbf{1}\}$	200	**0**	100	100	**100**	0	100	200	1,00	1,00	0,00	1,00	1,00

Tabelle 4.3: Lösungen im Fall mehrdeutiger Priorisierung (Beispiel). Dargestellt sind mögliche Lösungen für den Wertstromgraphen in Abbildung 4.14. Die Taktzeiten und Betriebsmittelzeiten seien in einer Weise vorgegeben, dass die Auslastung jeder Ressource bei einer Stückzahl von 100 den Wert eins erreicht. Die Lösungen (1) und (5) repräsentieren jeweils den Fall, dass die Prozessstückzahl x_2 bzw. x_5 maximal ist. Die Zwischenlösungen folgen daraus, dass die Steigerung einer der zwei Prozessstückzahlen die Reduktion der jeweils anderen erfordert.

bis zum aktuellen Knoten führen. Das Ergebnis liegt in Gestalt der *MaxFolge* vor, welche die Indizes a der Prozessstückzahlen x_a in der entsprechenden Reihenfolge enthält.

Es lassen sich jedoch Fälle konstruieren, in denen die definierte Ordnung nicht eindeutig ist. Hierzu müssen selektive und alternative Flusspunkte miteinander verkettet werden und die jeweils ausgehenden Kanten zu verschiedenen Eingängen derselben Ressource führen. Das heißt, in einem Wertstrom müssen verschiedene Prozessschritte, denen dieselbe MAE zugeordnet ist, durch rekursive Verkettung selektiv und alternativ miteinander verknüpft sein. Einen entsprechenden Fall zeigt Abbildung 4.14.

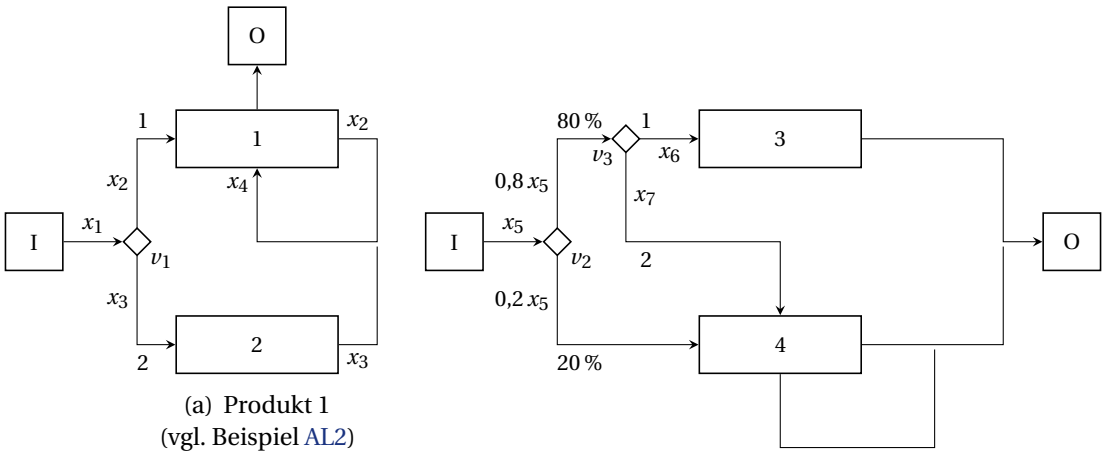
In einem solchen Fall ist die Ordnung zweier oder mehrerer Prozessstückzahlen davon abhängig, welche ausgehende Kante eines selektiven Flusspunkts zuerst weiterverfolgt wird. Diese Entscheidung beeinflusst wiederum die Auslastung der MAEs, wie Tabelle 4.3 für den gegebenen Fall zeigt. Jedoch wurde darauf verzichtet, das grafische Modell dahingehend zu erweitern, dass an ausgehenden Kanten selektiver Flusspunkte nicht nur Quoten, sondern auch Prioritäten ergänzt werden müssen. Hierfür existieren drei Gründe:

(1) Die Komplexität des Modells würde gesteigert werden, da den Anwendern die Notwendigkeit einer Priorisierung vermittelt werden müsste, obgleich die Stückzahlen an den ausgehenden Kanten durch die vorgegebenen Quoten unveränderlich sind.

(2) Solange die ausgehenden Kanten eines Knotens in einer Liste und nicht in einer Menge verwaltet werden, ist sichergestellt, dass stets dasselbe Ergebnis erzielt wird, auch wenn die Sortierung der Liste für den Anwender nicht sichtbar ist.

(3) Entsprechende Fälle verketteter Verknüpfungen sind im Zuge der Einführung der Software AURELIE bei der Bosch Rexroth AG nicht aufgetreten, auch nicht im Fall besonders komplexer Wertströme mit einer großen Zahl von Prozessschritten und MAEs.

Der letzte Punkt unterstreicht, dass es sich um eine rein theoretische Unvollständigkeit des grafischen Modells handelt, die in der Praxis keine Relevanz besitzt.



(a) Produkt 1 (vgl. Beispiel AL2)

(b) Produkt 2 (vgl. Beispiel SL2)

Abbildung 4.13: Variablen als Ergebnis einer Modelltransformation (Beispiel). Nachdem ein grafisches Modell validiert wurde, kann dieses in ein mathematisches Modell transformiert werden, das sich zur Optimierung eignet. Hierzu werden jedem Eingang und jedem Ausgang eines Knotens eine Prozessstückzahl und ein Koeffizient zugeordnet, die jeweils einem Anteil der Produktstückzahl entsprechen. Das Ergebnis dieser Zuordnung ist für ein exemplarisches grafisches Modell dargestellt, das aus je einem Wertstromgraphen für zwei Produkte zusammengesetzt ist.

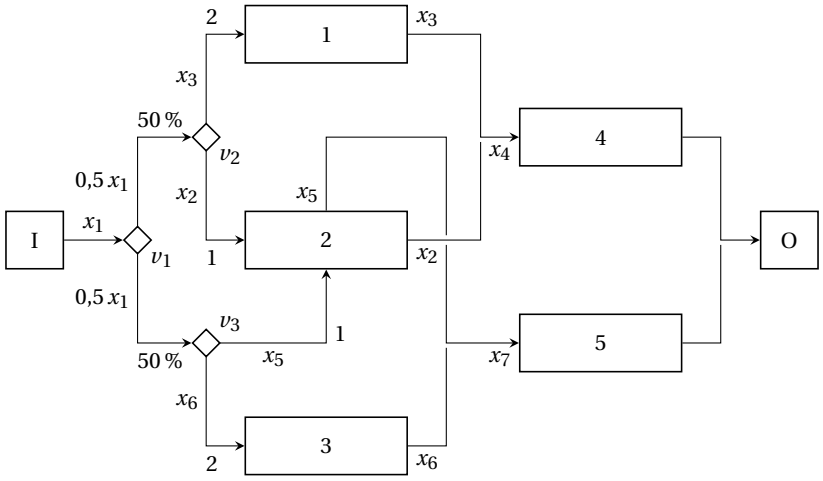


Abbildung 4.14: Mehrdeutige Priorisierung als Folge verketteter Verknüpfungen (Beispiel). Im Regelfall ist durch Priorisierung der Ausgänge alternativer Flusspunkte gewährleistet, dass eine eindeutige Lösung existiert, welche den geplanten Produktstückzahlen entspricht. Überlagern sich verkettet verknüpfte Prozessschritte, genügen die Prioritäten nicht immer, um eine Ordnung zwischen allen Prozessstückzahlen abzuleiten. Einen solchen, konstruierten Fall zeigt das Beispiel.

4.4.2 Ziel, Grundidee und Datenstrukturen

Das Ziel der Transformation besteht darin, ein vom Anwender erstelltes grafisches Modell $\{G_{WS,b}\}$ in ein entsprechendes mathematisches Modell $(\mathbf{P},\mathbf{S},\mathbf{F},\mathbf{L},\mathbf{L}_{Invest})$ umzuwandeln, das sich zur Optimierung eignet. Zudem soll die *MaxFolge* mit den Indizes a der zu priorisierenden Prozessstückzahlen x_a bestimmt werden. Danach ist es möglich, das mathematische Modell zu nutzen, um mittels exakter Verfahren die Optima der definierten Planungsziele zu suchen. Wie die zuvor erläuterte Validierung basiert auch die Transformation in ihrem Ablauf auf einer Tiefensuche in den Wertstromgraphen $G_{WS,b}$.

Ziel und Vorbedingungen. Konkret ist es das Ziel, die zu Beginn des Abschnitts eingeführten Matrizen und Folgen zu ermitteln: die Prozessmatrix **P**, die Flussmatrix **F**, die Auslastungsmatrix **L**, die Investitionsmatrix $\mathbf{L}_{Invest}$ und die *MaxFolge* der Prozessstückzahlen x_a. Die Produktstrukturmatrix **S** ist vorgegeben und wird folglich nicht durch den Algorithmus bestimmt. Zur Ausführung des Algorithmus existieren zwei Vorbedingungen:

V1 Es müssen sämtliche *Vorbedingungen* erfüllt sein, die bereits im Hinblick auf die *Validierung* für Algorithmus 4.2 galten. Dies betrifft sowohl die Verknüpfung der grafischen Modellelemente als auch die Ergänzung der Knoten um effektive Taktzeiten t_{eff}, Betriebsmittelzeiten T_{BM}, Prioritäten und Quoten.

V2 Das gegebene grafische Modell $\{G_{WS,b}\}$ muss vor der Transformation in ein mathematisches Modell $(\mathbf{P},\mathbf{S},\mathbf{F},\mathbf{L},\mathbf{L}_{Invest})$ *erfolgreich validiert* worden sein. Entsprechend wird vorausgesetzt, dass alle notwendigen Kanten zur Verbindung der Knoten vorliegen und die Wertstromgraphen $G_{WS,b}$ keine unendlichen Zyklen besitzen.

Grundidee zum Ablauf. Analog zu dem Vorgehen, welches bereits für die Validierung eines grafischen Modells $\{G_{WS,b}\}$ Anwendung fand, erfolgt die Beschreibung des Algorithmus für dessen Transformation in zwei Stufen. Zuerst soll an einem einführenden, einfachen Algorithmus 4.3 die Traversierung aller Knoten eines grafischen Modells $\{G_{WS,b}\}$ erläutert werden, wobei die gesuchten Matrizen und Folgen noch nicht bestimmt werden. Danach wird der Algorithmus 4.4 zur Transformation des grafischen Modells $\{G_{WS,b}\}$ vorgestellt. Hierzu wird der erstgenannte Algorithmus erweitert, um im Zuge der Traversierung alle notwendigen Informationen aus dem grafischen Modell $\{G_{WS,b}\}$ zu extrahieren. Wie im vorigen Abschnitt wird eine kurz gefasste Form der Beschreibung in natürlicher Sprache gewählt. Die Langfassungen sind einschließlich aller verwendeten Datenstrukturen als Algorithmus A.1.7 bzw. A.1.8 im Anhang zu finden. Diese sind in formalem, kommentiertem Pseudocode erstellt und geben den Ablauf detailliert wieder.

Zur Transformation eines gegebenen grafischen Modells $\{G_{WS,b}\}$ müssen alle Wertstromgraphen $G_{WS,b}$ traversiert werden. Um eine vollständige Ordnung zwischen den Prozessstückzahlen an den ausgehenden Kanten alternativer Flusspunkte herzustellen, basiert der Algorithmus auf einer Tiefensuche. Im Unterschied zur Validierung werden nicht die Kanten, sondern die Eingänge i aller Knoten $V_{WS,b}$ traversiert, da es das Ziel ist, diesen jeweils eine Prozessstückzahl x_a und einen Koeffizienten q zuzuordnen. An alternativen und selektiven

Flusspunkten werden alle ausgehenden Kanten weiterverfolgt, wobei an alternativen Flusspunkten Kanten mit höherer Priorität zuerst beschritten werden. Die Traversierung beginnt in jedem Wertstromgraphen $G_{WS,b}$ bei der Quelle σ_b und endet, wenn kein weiterer Eingang eines Knotens zu besuchen ist. Abbildung 4.15 zeigt an einem Beispiel den Ablauf und die Zuordnung von Prozessstückzahlen und Koeffizienten in jedem Schritt.

Es ist jedoch möglich, dass ein Eingang i eines Knotens v bereits besucht wurde, nun aber erstmalig erreicht wird, indem ein Weg entlang der ersten ausgehenden Kanten alternativer Flusspunkte fortgesetzt wird. Dann muss die Information gewonnen werden, welche der nachfolgenden Eingänge i' von Knoten v' ebenfalls auf einem solchen Weg von der Quelle σ_b bis zur Senke τ_b liegen. Hintergrund ist, dass diese Eingänge Prozessschritte repräsentieren, die eine Überlastung der zugeordneten Ressourcen verursachen dürfen. Aus diesem Grund wird jeder Eingang ggf. mehr als einmal besucht. Die nachfolgenden Kanten jedes Eingangs werden mindestens einmal, höchstens aber zweimal weiterverfolgt, und die Traversierung verläuft entsprechend in zwei Modi: (1) einem *Entdeckungsmodus*, in welchem ein Eingang eines Knotens erstmalig besucht wird und welcher dazu dient, die definierten Matrizen und Folgen zu bestimmen, und (2) einem *Investitionsmodus*, in welchem ein Eingang wiederholt besucht wird, um die Information zu gewinnen, welche nachfolgenden Eingänge von Knoten in dem Wertstromgraphen $G_{WS,b}$ ausgehend von der Quelle σ_b jeweils entlang der ersten ausgehenden Kanten alternativer Flusspunkte $D_{AL,b}$ erreicht werden können. Im Weiteren werden die Datenstrukturen erläutert, die zur Verwaltung der zu besuchenden Eingänge von Knoten und zur Bestimmung der Ergebnisse dienen.

Datenstrukturen. Der Algorithmus 4.3 zur Traversierung der Knoten und der Algorithmus 4.4 zur Transformation eines grafischen Modells $\{G_{WS,b}\}$ basieren wie die Validierung auf einer Tiefensuche. Der Grund hierfür ist, dass die Reihenfolge entscheidend ist, in welcher die alternativen Flusspunkte ausgehend von der Quelle erreicht werden, um eine vollständige Ordnung der Prozessstückzahlen herzustellen. Folglich wird wie schon zuvor ein *Stapel* eingesetzt, um die als Nächstes zu besuchenden Elemente zu verwalten. Jedes dieser Elemente bezeichnet nun allerdings, wie oben erläutert, ein Paar (v, i) aus einem Eingang i eines Knotens v. Im Fall des Algorithmus 4.4 wird zudem die Information benötigt, welche Prozessstückzahl x_a und welcher Koeffizient q diesem Eingang des Knotens zuzuordnen sind. Somit wird das Paar (v, i) zu einem Tupel (v, i, a, q) erweitert.

Für den Algorithmus 4.4 zur Transformation eines grafischen Modells $\{G_{WS,b}\}$ werden neben den Skalaren, Vektoren und Matrizen weitere Datenstrukturen benötigt. Die Ordnung der Prozessstückzahlen x_a wird dadurch abgebildet, dass deren Indizes a in der Reihenfolge ihrer Erzeugung in die *MaxFolge* aufgenommen werden. Bei der *MaxFolge* handelt es sich um eine linear verkettete Liste, die nach dem FIFO-Prinzip verwaltet wird. Das heißt, zuerst hinzugefügte Elemente werden entsprechend auch zuerst entfernt und zurückgegeben. Durch sukzessives Entfernen der Indizes a aus der *MaxFolge* kann damit die Ordnung der Prozessstückzahlen x_a rekonstruiert werden.

Des Weiteren muss die Information gespeichert werden, dass ein Eingang i eines Knotens v dadurch erreicht wurde, dass stets die erste ausgehende Kante an jedem alternativen Flusspunkt weiterverfolgt wurde. Sofern der Knoten eine Ressource bezeichnet, entspricht

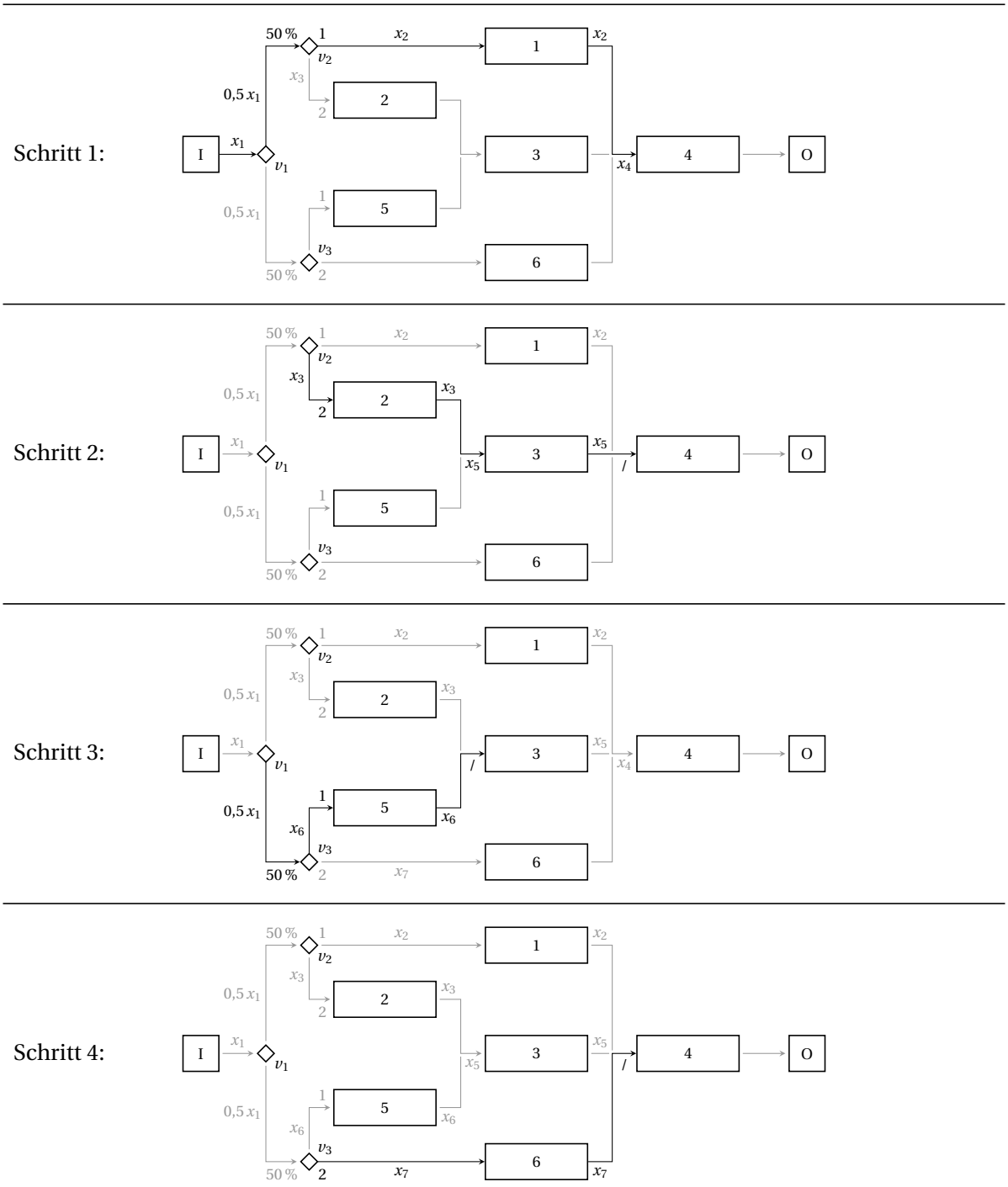


Abbildung 4.15: Schrittfolge der Transformation eines Wertstromgraphen (Beispiel). Grundsätzlich entspricht der Ablauf der Tiefensuche, da zuerst ein Weg von der Quelle bis zur Senke gesucht wird, wobei die Kante zur Senke ausgenommen ist. Eine nachfolgende Kante eines Eingangs wird nur dann beschritten, wenn der Eingang zuvor noch nicht besucht wurde oder erstmalig ein Weg fortgesetzt wird, welcher die erste ausgehende Kante jedes alternativen Flusspunkts weiterführt. Dadurch kann ermittelt werden, welche Eingänge entlang der ersten Prioritäten erreichbar sind.

ein solcher Eingang einem Prozessschritt, der zu einer Überlastung führen darf. Eingänge von Knoten, auf welche die genannte Bedingung zutrifft, werden in der *InvestMenge* verwaltet. Die Menge wird zum Abschluss des Algorithmus genutzt, um die Werte der Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ zu bestimmen. Diese Matrix unterscheidet sich von der Auslastungsmatrix $\mathbf{L}$ dahingehend, dass Eingänge aus der *InvestMenge* nicht berücksichtigt werden. Jedes Element der *InvestMenge* ist ein Tupel (b, v, i) aus einem Produkt b, einem Knoten v und einem Eingang i. Im Gegensatz zum *Stapel* ist es notwendig, den Index b des Produkts zu ergänzen, da der *Stapel* für jeden Wertstromgraphen $G_{\text{WS},b}$ initialisiert, verwaltet und vollständig abgebaut wird. Die *InvestMenge* wird dagegen im Verlauf des Algorithmus erweitert und erst nach der Traversierung aller Wertstromgraphen $G_{\text{WS},b}$ ausgewertet.

4.4.3 Beschreibung der Algorithmen

Zu Beginn soll der einführende Algorithmus 4.3 zur Traversierung der Knoten eines grafischen Modells $\{G_{\text{WS},b}\}$ beschrieben werden (Langfassung siehe Algorithmus A.1.7). Danach wird der Algorithmus 4.4 erläutert, welcher dazu dient, ein entsprechendes Modell in ein mathematisches zu transformieren (Langfassung siehe Algorithmus A.1.8).

Um den grundsätzlichen Ablauf der Transformation mit der Validierung zu vergleichen, wird im Fall des einführenden Algorithmus 4.3 noch nicht vorausgesetzt, dass das grafische Modell $\{G_{\text{WS},b}\}$ validiert ist. Aus diesem Grund werden dieselben Prüfungen auf das Vorliegen von Kanten notwendig, wie sie auch der Algorithmus 4.1 zur Traversierung der Kanten eines grafischen Modells $\{G_{\text{WS},b}\}$ enthält. Für den erweiterten Algorithmus 4.4 zur Transformation eines solchen Modells gelten danach allerdings alle genannten Vorbedingungen, sodass auf diese Prüfungen verzichtet werden kann.

Algorithmus 4.3: Traversierung der Knoten eines grafischen Modells $\{G_{\text{WS},b}\}$

Prozedur TraversiereKnotenVonGrafischemModell(...)

Zeile 01 enthält die Deklaration der Hauptprozedur zur Traversierung der Knoten eines gegebenen grafischen Modells $\{G_{\text{WS},b}\}$. Zunächst werden alle Produkte b betrachtet, um die Knoten $V_{\text{WS},b}$ in den zugehörigen Wertstromgraphen $G_{\text{WS},b}$ als nicht entdeckt zu markieren. Danach wird für jeden Wertstromgraphen $G_{\text{WS},b}$ eine Hilfsprozedur aufgerufen, welche dazu dient, die Knoten in dem jeweiligen Graphen zu traversieren.

Prozedur TraversiereKnotenVonWSGraphen(...)

Die entsprechende Hilfsprozedur ist ab Zeile 07 definiert. Zuerst wird der *Stapel* als ein leeres Tupel initialisiert, wobei jedes Element ein Paar (v, i) aus einem Knoten v und einem Eingang i bezeichnet. Daraufhin findet die Prüfung statt, ob an der Quelle σ_b eine Kante anliegt. Existiert eine solche Kante, wird diese zum nächsten Knoten v' und dessen Eingang i' weiterverfolgt, um das entsprechende Paar ans Ende des *Stapels* zu fügen. Danach wird ab Zeile 13 in einer Schleife geprüft, ob der *Stapel* Elemente enthält und entsprechend weitere Eingänge von Knoten besucht werden müssen. Falls zu Beginn eine Kante an der Quelle σ_b anlag, ist diese Bedingung erfüllt, und die Schleife wird betreten. Anschließend werden

```
    Eingabe: grafisches Modell {G_WS,b} aus Wertstromgraphen G_WS,b für alle Produkte b
    Ausgabe: markierte Eingänge i aller Knoten V_WS,b außer den Quellen σ_b
01: Prozedur TraversiereKnotenVonGrafischemModell({G_WS,b})
02:     für alle Produkte b tue
...         für alle Eingänge i der Knoten V_WS,b tue initialisiere Eingang i als nicht entdeckt
06:         TraversiereKnotenVonWSGraphen(G_WS,b, b)

07: Prozedur TraversiereKnotenVonWSGraphen(G_WS,b, b)
08:     initialisiere Stapel als leeres Tupel
09:     wenn an der Quelle σ_b eine Kante anliegt dann
...       folge der Kante, um nächsten Knoten v' und Eingang i' ans Ende des Stapels zu fügen
13:     solange Stapel nicht leer ist tue
14:         entferne aktuellen Knoten v und Eingang i vom Ende des Stapels
15:         wenn Eingang i bisher noch nicht entdeckt wurde dann
16:             markiere Eingang i als entdeckt
...             TraversiereRessource(...) bzw. TraversiereFlusspunkt(...)

21: Prozedur TraversiereRessource(G_WS,b, Stapel, v, i)
22:     betrachte nur Ausgang j des Knotens v gegenüber von Eingang i
23:     wenn an Ausgang j eine Kante anliegt dann
...       folge der Kante, um nächsten Knoten v' und Eingang i' ans Ende des Stapels zu fügen

27: Prozedur TraversiereFlusspunkt(G_WS,b, Stapel, v)
28:     für alle Kanten an Ausgängen j tue
...       folge der Kante, um nächsten Knoten v' und Eingang i' ans Ende des Stapels zu fügen
```

Algorithmus 4.3: Traversierung der Knoten eines grafischen Modells $\{G_{WS,b}\}$ (Kurzfassung).

jeweils ein Knoten v und ein Eingang i vom Ende des *Stapels* entfernt. Wurde der Eingang i noch nicht entdeckt, dann wird dieser nun als entdeckt markiert, und es wird nach dem Typ des Knotens v unterschieden. Handelt es sich bei dem Knoten v um eine Ressource, wird die Hilfsprozedur zur Traversierung solcher Knoten aufgerufen. Bezeichnet der Knoten v einen alternativen oder selektiven Flusspunkt, folgt der Aufruf der entsprechenden Hilfsprozedur. Wurde mit dem Knoten v dagegen die Senke τ_b erreicht, dann besitzt die Kante, die zu diesem Knoten führt, keinen Nachfolger. Falls danach keine Elemente mehr im *Stapel* enthalten sind, wird die Schleife an dieser Stelle verlassen.

Prozedur TraversiereRessource(...)

Ab Zeile 21 ist die Hilfsprozedur zur Traversierung einer Ressource angegeben. Die Hilfsprozedur beschreibt, welcher nachfolgende Eingang i' an einem Knoten v' betrachtet werden soll, nachdem der Eingang i des aktuellen Knotens v entdeckt wurde. Hierbei wird nur derjenige Ausgang j berücksichtigt, welcher dem Eingang i gegenüberliegt und dessen Index folglich übereinstimmt. Liegt an Ausgang j eine Kante an, wird diese Kante zum nächsten Knoten v' und Eingang i' weiterverfolgt. Danach wird das entsprechende Paar (v', i') ans Ende des *Stapels* gefügt, um den Eingang i' als Nächstes zu besuchen.

Prozedur TraversiereFlusspunkt(...)

Die Hilfsprozedur zur Traversierung eines alternativen oder selektiven Flusspunkts ist ab Zeile 27 beschrieben. In einer Schleife werden nacheinander alle Ausgänge j des jeweiligen Flusspunkts betrachtet und die Kanten zu den nächsten Knoten v' und Eingängen i' weiterverfolgt. Daraufhin wird jeweils das Paar (v', i') ans Ende des *Stapels* gefügt. Als Folge wird der Eingang i' des Knotens v', welcher nach dem letzten Ausgang folgt, zuerst besucht. Die Reihenfolge, in welcher die Ausgänge eines Flusspunkts betrachtet werden, besitzt erst im nächsten Algorithmus eine Bedeutung.

Algorithmus 4.4: Transformation eines grafischen Modells $\{G_{\mathrm{WS},b}\}$

Prozedur TransformiereGrafischesModell(...)

In Zeile 001 ist die Deklaration der Hauptprozedur angegeben, die dazu dient, die Matrizen und Folgen des mathematischen Modells zu bestimmen. Nach der Initialisierung der Daten wird nacheinander für alle Produkte b eine Hilfsfunktion aufgerufen, um den jeweiligen Wertstromgraphen $G_{\mathrm{WS},b}$ zu transformieren. Von dieser Hilfsfunktion wird die Zahl m der insgesamt erzeugten Prozessstückzahlen x_a zurückgegeben. Danach folgt der Aufruf weiterer Hilfsprozeduren, um auf Grundlage der ermittelten Zwischenergebnisse die Flussmatrix $\mathbf{F}$, die Auslastungsmatrix $\mathbf{L}$ und die Investitionsmatrix $\mathbf{L}_{\mathrm{Invest}}$ zu bestimmen.

Funktion TransformiereWSGraphen(...)

Die oben genannte Hilfsfunktion zur Transformation eines Wertstromgraphen $G_{\mathrm{WS},b}$ ist ab Zeile 015 definiert. Zuerst wird der *Stapel* als leeres Tupel initialisiert, wobei jedes Element wiederum ein Tupel (v, i, a, q) aus einem Knoten v, einem Eingang i, dem Index a einer Prozessstückzahl x_a und einem Koeffizienten q ist. Ist der Index a größer als null, wird der Wertstromgraph $G_{\mathrm{WS},b}$ im *Entdeckungsmodus* traversiert, um den Eingängen der Knoten Prozessstückzahlen und Koeffizienten zuzuordnen. Sonst findet die Traversierung im *Investitionsmodus* statt, um zu ermitteln, welche Eingänge von Knoten nach bereits besuchten Eingängen erreichbar sind, indem an alternativen Flusspunkten die erste ausgehende Kante beschritten wird. Nach der Initialisierung des *Stapels* wird eine neue Prozessstückzahl $x_{a'}$ erzeugt, die am Ausgang der Quelle σ_b anliegt, und die Prozessmatrix $\mathbf{P}$ aktualisiert. Wenn die Kante an der Quelle σ_b noch nicht zur Senke τ_b führt, werden der nächste Knoten v' und Eingang i' mit dem Index a' ans Ende des *Stapels* gefügt.

Ab Zeile 025 erfolgt in einer Schleife die Prüfung, ob der *Stapel* Elemente enthält, worauf der letzte Knoten v und Eingang i mit dem Index a und dem Koeffizienten q vom *Stapel* entfernt werden. Vereinigen sich Kanten an diesem Eingang, muss unterschieden werden, ob der Eingang bereits besucht wurde. Falls nicht, wird wieder eine neue Prozessstückzahl $x_{a'}$ erzeugt, die zugehörige Zeile der Vereinigungsmatrix $\mathbf{U}$ initialisiert und im Entdeckungsmodus fortgesetzt (siehe Ressource r_4 in Schritt 1 von Abbildung 4.15). Falls der Eingang wiederholt besucht wird, aber die Suche im Entdeckungsmodus stattfindet, wird die Vereinigungsmatrix $\mathbf{U}$ aktualisiert und im Investitionsmodus weitergesucht (siehe Ressource r_4 in Schritt 2 von Abbildung 4.15). Sofern der aktuelle Eingang i erstmalig besucht wird, schließt sich die Aktualisierung der Eingangsmatrix $\mathbf{Q}_{\mathrm{ein},bv}$ an. Die nachfolgenden Knoten werden

Eingabe: (validiertes) grafisches Modell $\{G_{WS,b}\}$ aus Wertstromgraphen $G_{WS,b}$ für alle Produkte b
Ausgabe: Prozessmatrix $\mathbf{P}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$, Investitionsmatrix $\mathbf{L}_{\text{Invest}}$, *MaxFolge*

001: **Prozedur** TransformiereGrafischesModell($\{G_{WS,b}\}, \mathbf{P}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}}$, *MaxFolge*)
… initialisiere Matrizen, Zahl m der Prozessstückzahlen x_a, *InvestMenge* und *MaxFolge*
011: **für alle** *Produkte b* **tue** $m \leftarrow$ TransformiereWSGraphen($G_{WS,b}, b, \ldots$)
… FülleFlussmatrix(…) und FülleAuslastungsmatrix(…)

015: **Funktion** TransformiereWSGraphen($G_{WS,b}, b, m, \mathbf{P}, \{\mathbf{Q}_{\text{ein},bv}\}, \{\mathbf{Q}_{\text{aus},bv}\}, \mathbf{U}$, *InvestMenge*, *MaxFolge*)
… initialisiere *Stapel*, erzeuge Prozessstückzahl $x_{a'}$ und erweitere Prozessmatrix $\mathbf{P}$
021: **wenn** *nach der Quelle* σ_b *nicht die Senke* τ_b *folgt* **dann**
… füge nächsten Eingang i' ans Ende des *Stapels*
025: **solange** *Stapel nicht leer ist* **tue**
… entferne aktuellen Eingang i vom Ende des *Stapels*
028: **wenn** *sich mehrere Kanten vor Eingang i vereinigen* **dann**
029: **wenn** *Eingang i noch nicht besucht wurde* **dann** erzeuge Prozessstückzahl $x_{a'}$ und
… erweitere entsprechend Vereinigungsmatrix $\mathbf{U}$
036: **sonst wenn** *vorhergehender Eingang erstmalig besucht wurde* **dann**
… aktualisiere Vereinigungsmatrix $\mathbf{U}$
042: **wenn** *Eingang i erstmalig besucht wird* **dann** aktualisiere Eingangsmatrix $\mathbf{Q}_{\text{ein},bv}$
… **wenn** *vorherige Bedingung gilt oder*
Eingang i in der InvestMenge enthalten ist **dann** TransformiereRessource(…),
TransformiereALFlusspunkt(…) bzw. TransformiereSLFlusspunkt(…)
051: **liefere** Zahl m der Prozessstückzahlen x_a **zurück**

052: **Prozedur** TransformiereRessource($G_{WS,b}, b$, *InvestMenge*, *Stapel*, v, i, a, q)
… **wenn** *nach Ausgang j gegenüber von Eingang i nicht die Senke* τ_b *folgt* **dann**
057: **wenn** *Eingang i erstmalig besucht wird* **dann**
058: **wenn** *dieser in der InvestMenge enthalten ist* **dann** ergänze nächsten Eingang i'
… übernimm Prozessstückzahl x_a und Koeffizient q von Eingang i
062: füge Eingang i' ans Ende des *Stapels*
063: **sonst wenn** *Eingang* i' *nicht in der InvestMenge enthalten ist* **dann**
… ergänze ihn und füge diesen ans Ende des *Stapels*

067: **Funktion** TransformiereALFlusspunkt($G_{WS,b}, b, m, \mathbf{Q}_{\text{aus},bv}$, *InvestMenge*, *MaxFolge*, *Stapel*, v, a, q)
068: **wenn** *Eingang des Knotens v erstmalig besucht wird* **dann**
069: **wenn** *dieser in der InvestMenge enthalten ist und*
nach erstem Ausgang des Knotens v nicht die Senke τ_b *folgt* **dann**
… ergänze nächsten Eingang i' in der *InvestMenge*
075: **für alle** *Ausgänge j des Knotens v in absteigender Folge* **tue**
… erzeuge Prozessstückzahl $x_{a'}$ und aktualisiere Ausgangsmatrix $\mathbf{Q}_{\text{aus},bv}$
079: **wenn** *nach Ausgang j nicht die Senke* τ_b *folgt* **dann**
… füge nächsten Eingang i' ans Ende des *Stapels*
… füge erzeugte Prozessstückzahlen $x_{a'}$ außer der letzten ans Ende der *MaxFolge*
085: **sonst wenn** *nach erstem Ausgang nicht die Senke* τ_b *folgt und*
nächster Eingang i' *nicht in der InvestMenge enthalten ist* **dann**
… ergänze ihn und füge diesen ans Ende des *Stapels*
092: **liefere** Zahl m der Prozessstückzahlen x_a **zurück**

(nächster Teil auf Seite 157)

Algorithmus 4.4(a): Transformation eines grafischen Modells $\{G_{WS,b}\}$ (Kurzfassung, Teil 1 von 2).

(vorheriger Teil auf Seite 156)

```
093: Prozedur TransformiereSLFlusspunkt(G_WS,b, b, InvestMenge, Stapel, v, a, q)
 ...   für alle Ausgänge j des Knotens v in absteigender Folge tue
097:     wenn nach Ausgang j nicht die Senke τ_b folgt dann
 ...       wenn Eingang des Knotens v erstmalig besucht wird dann
100:         wenn dieser in der InvestMenge enthalten ist dann ergänze nächsten Eingang i'
 ...         übernimm Prozessstückzahl x_a vom Eingang und
               multipliziere Koeffizient q mit der definierten Quote von Ausgang j
104:         füge Eingang i' ans Ende des Stapels
105:       sonst wenn Eingang i' nicht in der InvestMenge enthalten ist dann
 ...         ergänze ihn und füge diesen ans Ende des Stapels

109: Prozedur FülleFlussmatrix({G_WS,b}, m, F, {Q_ein,bv}, {Q_aus,bv}, U)
 ...   für alle Vereinigungen von Kanten an Eingängen von Knoten V_WS,b tue
 ...     erweitere Flussmatrix F entsprechend Vereinigungsmatrix U
115:   für alle alternativen Flusspunkte D_AL,b tue erweitere Flussmatrix F entsprechend
 ...     Eingangsmatrix Q_ein,bv und Ausgangsmatrix Q_aus,bv

119: Prozedur FülleAuslastungsmatrix({G_WS,b}, L, L_Invest, {Q_ein,bv}, InvestMenge)
 ...   für alle Eingänge i von Ressourcen R in Wertstromgraphen G_WS,b tue
127:     erweitere Auslastungsmatrix L entsprechend Betriebsmittelzeit T_BM, effektiver Taktzeit t_eff
           am jeweiligen Eingang i und Eingangsmatrix Q_ein,bv
128:     wenn Eingang i nicht in der InvestMenge enthalten ist dann
129:       erweitere ebenso Investitionsmatrix L_Invest
```

Algorithmus 4.4(b): Transformation eines grafischen Modells $\{G_{WS,b}\}$ (Kurzfassung, Teil 2 von 2).

nur dann betrachtet, wenn der Eingang i zum ersten Mal besucht wird (Entdeckungsmodus) oder in der *InvestMenge* enthalten ist (Investitionsmodus). Unter dieser Bedingung werden je nach Knotentyp Hilfsfunktionen oder Hilfsprozeduren aufgerufen.

Prozedur TransformiereRessource(...)

In Zeile 052 folgt die Deklaration der Hilfsprozedur zur Transformation einer Ressource, welche an einem Eingang i besucht wird. Hierbei wird nur der gegenüberliegende Ausgang j mit gleichem Index betrachtet. Wenn die Kante an diesem Ausgang noch nicht zur Senke τ_b führt, wird geprüft, ob der Eingang i erstmalig besucht wird. Ist das der Fall (Entdeckungsmodus) und ist der Eingang i in der *InvestMenge* enthalten, wird der nachfolgende Eingang i' ebenso in der *InvestMenge* ergänzt. Weiterhin werden in diesem Modus der Index a sowie der Koeffizient q vom Eingang i übernommen und mit dem nächsten Eingang i' ans Ende des *Stapels* gefügt. Falls der Eingang i bereits besucht wurde (Investitionsmodus) und der nächste Eingang i' nicht in der *InvestMenge* enthalten ist, wird dieser in die Menge aufgenommen. Darauf wird der Eingang i' ans Ende des *Stapels* gefügt, um im Investitionsmodus fortzusetzen (siehe Ressource r_3 in Schritt 3 von Abbildung 4.15, die nachfolgende Kante des Eingangs i wird betrachtet, da ein Weg entlang der ersten ausgehenden Kanten an jedem alternativen Flusspunkt fortgesetzt wird, allerdings wird diese Kante nicht beschritten, da der Eingang i' bereits der *InvestMenge* hinzugefügt wurde).

Funktion TransformiereALFlusspunkt(...)

Der obigen Prozedur schließt sich in Zeile 067 die Hilfsfunktion zur Transformation eines alternativen Flusspunkts an. Zunächst muss unterschieden werden, ob dessen Eingang erstmalig besucht wird. Falls dies zutrifft (Entdeckungsmodus), der Eingang in der *InvestMenge* enthalten ist und nach dem ersten Ausgang noch nicht die Senke τ_b folgt, wird der nächste Eingang i' in die *InvestMenge* aufgenommen. Danach werden in dem Modus alle Ausgänge j absteigend betrachtet. Für jeden Ausgang j wird eine neue Prozessstückzahl $x_{a'}$ erzeugt und die Ausgangsmatrix $\mathbf{Q}_{\text{aus},bv}$ aktualisiert. Sofern nicht die Senke τ_b erreicht ist, wird jeweils der nächste Eingang i' mit dem Index a' der Prozessstückzahl $x_{a'}$ ans Ende des *Stapels* gefügt. Durch die absteigende Sortierung wird der erste Ausgang zuerst weiterverfolgt. Zudem werden alle neuen Prozessstückzahlen bis auf die letzte in aufsteigender Reihenfolge in die *MaxFolge* aufgenommen. Die Prozessstückzahl am letzten Ausgang ist dabei nicht notwendig, da diese bei bekannter Stückzahl am Eingang und unter der Bedingung des Flusserhalts eindeutig bestimmt ist. Falls der Eingang wiederholt besucht wird (Investitionsmodus), nach dem ersten Ausgang nicht die Senke τ_b folgt und der nächste Eingang i' noch nicht in der *InvestMenge* enthalten ist, wird dieser ergänzt. Dann wird der nächste Eingang i' ans Ende des *Stapels* gefügt, um im Investitionsmodus fortzusetzen. Zuletzt wird die Zahl m insgesamt erzeugter Prozessstückzahlen x_a zurückgegeben.

Prozedur TransformiereSLFlusspunkt(...)

Die Hilfsprozedur zur Transformation eines selektiven Flusspunkts ist ab Zeile 093 definiert. In absteigender Reihenfolge werden alle Ausgänge j betrachtet, an denen Kanten anliegen, die nicht zur Senke τ_b führen. Dabei wird unterschieden, ob der Eingang des Flusspunkts erstmalig besucht wird. Ist dies der Fall (Entdeckungsmodus) und ist der Eingang in der *InvestMenge* enthalten, wird der nächste Eingang i' nach dem Ausgang j entsprechend in die *InvestMenge* aufgenommen. In diesem Modus werden der Index a vom Eingang übernommen, der Koeffizient q mit der Quote an Ausgang j multipliziert und beide mit dem nächsten Eingang i' ans Ende des *Stapels* gefügt. Falls der Eingang des Flusspunkts bereits besucht wurde (Investitionsmodus) und der nächste Eingang i' nicht in der *InvestMenge* enthalten ist, wird dieser ergänzt. Danach wird in dem Modus wieder der nächste Eingang i' ans Ende des *Stapels* gefügt, um im Investitionsmodus fortzusetzen.

Prozedur FülleFlussmatrix(...) und FülleAuslastungsmatrix(...)

In Zeile 109 und 119 sind abschließend die Hilfsprozeduren zur Bestimmung der Flussmatrix **F** bzw. der Auslastungsmatrix **L** sowie der Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ aufgeführt. Die Prozeduren legen die Werte der Matrizen fest, wofür die zuvor ermittelten Zwischenergebnisse genutzt werden. Hierzu zählen die Vereinigungsmatrix **U**, die Eingangsmatrizen $\mathbf{Q}_{\text{ein},bv}$ und Ausgangsmatrizen $\mathbf{Q}_{\text{aus},bv}$ sowie die *InvestMenge*.

4.4.4 Beweis der Zeitkomplexität

Um das asymptotische Wachstum der Laufzeit zu beschreiben, ist analog zu Abschnitt 4.3 die Zeitkomplexität der zwei Algorithmen gesucht. Gegeben sei ein grafisches Modell $\{G_{\mathrm{WS},b}\}$ endlich vieler Wertstromgraphen $G_{\mathrm{WS},b}$, deren Knoten zu traversieren sind bzw. welches in ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\mathrm{Invest}})$ zu transformieren ist.

Algorithmus 4.3: Traversierung der Knoten eines grafischen Modells $\{G_{\mathrm{WS},b}\}$

Der Algorithmus 4.3 entspricht einer wiederholt aufgerufenen Tiefensuche, in deren Zuge die Wertstromgraphen $G_{\mathrm{WS},b}$ vollständig traversiert werden. Gegenüber einer klassischen Tiefensuche werden nicht nur die Knoten $V_{\mathrm{WS},b}$ jedes Wertstromgraphen $G_{\mathrm{WS},b}$ in einem *Stapel* verwaltet. Die Elemente des *Stapels* bezeichnen Paare (v, i) aus einem Knoten v und einem zusätzlichen Eingang i. Im Hinblick auf die Laufzeit des Algorithmus gilt:

Theorem 4.4: Zeitkomplexität von Algorithmus 4.3. Es sei ein grafisches Modell $\{G_{\mathrm{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\mathrm{WS},b}$ gegeben, welche die Wertströme eines Systems zur Herstellung einer entsprechenden Zahl von Produkten abbilden. Der Algorithmus zur Traversierung aller Knoten des Modells besitzt in der angegebenen Implementierung eine Laufzeit von $\mathrm{O}\left(\sum_{b=1}^{n} |V_{\mathrm{WS},b}| + \sum_{b=1}^{n} |E_{\mathrm{WS},b}|\right)$.

Beweis. Im Verlauf der Traversierung wird jeder Eingang eines Knotens höchstens einmal entdeckt. Da jeder Eingang durch eine Kante erreicht werden muss, ist die Zahl entdeckter Eingänge in einem Wertstromgraphen $G_{\mathrm{WS},b}$ höchstens gleich der Zahl $|E_{\mathrm{WS},b}|$ der Kanten. Nachdem ein Eingang i eines Knotens v entdeckt wurde, werden alle nachfolgenden Eingänge i' von Knoten v' ans Ende des *Stapels* gefügt. Ein solcher Eingang i' ist ein Nachfolger des Eingangs i, wenn der Eingang i' erreichbar ist, indem eine ausgehende Kante an einem zugeordneten Ausgang j des Knotens v weiterverfolgt wird. Die Zuordnung von Ausgängen j zu Eingängen i eines Knotens v wird dabei durch dessen Typ bestimmt.

Im Fall einer Ressource wird nur die Kante berücksichtigt, die am Ausgang j gegenüber von Eingang i anliegt, sofern diese existiert. Im Fall eines alternativen oder selektiven Flusspunkts ist nur ein einziger Eingang gegeben, und es werden die anliegenden Kanten an allen Ausgängen betrachtet. Allgemein wird jedoch jede ausgehende Kante an einem Ausgang j eines Knotens v nur nach der Entdeckung des vorhergehenden Eingangs i fortgesetzt. Das heißt, jede ausgehende Kante an einem solchen Ausgang wird nur ein einziges Mal, nach der Entdeckung des vorherigen Eingangs i, weiterverfolgt. Gleich der Zahl der entdeckten Eingänge i werden daher nicht mehr als $|E_{\mathrm{WS},b}|$ nachfolgende Eingänge i' ans Ende des *Stapels* gefügt. In jedem Durchlauf wird genau ein Eingang aus dem *Stapel* entfernt, weshalb die Schleife entsprechend oft durchlaufen wird.

Des Weiteren wird wie zuvor vorausgesetzt, dass es in einer Laufzeit von $\mathrm{O}(1)$ möglich ist, den folgenden Eingang i' und Knoten v' nach einem Ausgang j eines Knotens v zu bestimmen. Die gleiche Voraussetzung gilt für die Operationen zum Hinzufügen und Entfernen von Elementen des *Stapels* sowie für die Bestimmung, ob ein Eingang entdeckt wurde, und die entsprechende Markierung. Dies führt jeweils zu den folgenden Laufzeiten für die Zeilen im Rumpf der Prozeduren (ohne aufgerufene Hilfsprozeduren):

02:	$O(n)$	08...12:	$O(n)$
03:	$O\left(\sum_{b=1}^{n} \lvert V_{WS,b}\rvert\right)$	13...20:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$
04...05:	$O\left(\sum_{b=1}^{n} \lvert V_{WS,b}\rvert + \sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$	22...26:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$
06:	$O(n)$	28...31:	$O\left(\sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$

Zur Bestimmung der Gesamtlaufzeit werden die einzelnen Laufzeiten aller Zeilen addiert, wobei kleinere Summanden und konstante Faktoren nicht berücksichtigt werden. Folglich ist die gesuchte obere Schranke gleich $O\left(\sum_{b=1}^{n} \lvert V_{WS,b}\rvert + \sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$. □

Gegenüber den Algorithmen 4.1 und 4.2 muss keine Annahme in Bezug auf den maximalen Ausgangsgrad von Flusspunkten getroffen werden, um die Laufzeit zu erreichen. Der Grund dafür ist, dass in den beiden genannten Algorithmen eine ausgehende Kante genau dann weiterverfolgt wird, wenn eine der vorigen Kanten entdeckt bzw. erstmalig besucht wird. Im vorliegenden Algorithmus 4.3 wird eine ausgehende Kante nur ein einziges Mal, nach der Entdeckung des vorhergehenden Eingangs des Knotens, beschritten. Der Unterschied ist relevant, wenn sich mehrere Kanten an dem Eingang vereinigen.

Algorithmus 4.4: Transformation eines grafischen Modells $\{G_{WS,b}\}$

Der im Folgenden betrachtete Algorithmus 4.4 beruht auf dem vorhergehenden, einführenden Algorithmus 4.3. In beiden Fällen werden die Wertstromgraphen $G_{WS,b}$ eines grafischen Modells $\{G_{WS,b}\}$ vollständig traversiert, indem alle Eingänge der Knoten besucht werden. Um die zu besuchenden Eingänge zu verwalten, wird wieder ein *Stapel* eingesetzt. Zur Bestimmung einer oberen Schranke für die Laufzeit werden grundsätzlich ähnliche Schlüsse gezogen, wobei Unterschiede zu berücksichtigen sind. Da die Wertstromgraphen $G_{WS,b}$ in zwei Modi durchlaufen werden, müssen bestimmte Eingänge wiederholt besucht werden. Zudem werden weitere Datenstrukturen genutzt, um die gesuchten Matrizen, Mengen und Folgen zu bestimmen. Als Folge dessen gilt für die Laufzeit:

Theorem 4.5: Zeitkomplexität von Algorithmus 4.4. Es sei ein grafisches Modell $\{G_{WS,b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{WS,b}$ gegeben, welche die Wertströme eines Systems zur Herstellung einer entsprechenden Zahl von Produkten abbilden. Der Algorithmus zur Transformation des Modells besitzt in der angegebenen Implementierung eine Laufzeit von $O\left(\sum_{b=1}^{n} \lvert V_{WS,b}\rvert + \sum_{b=1}^{n} \lvert E_{WS,b}\rvert\right)$.

Beweis. Jeder Eingang i eines Knotens v kann mehrmals besucht werden, genau einmal im Entdeckungsmodus und als Folge sich vereinigender Kanten ggf. wiederholt im Investitionsmodus. Da mindestens eine Kante zu einem solchen Eingang führen muss, entspricht die Zahl der besuchten Eingänge in einem Wertstromgraphen $G_{WS,b}$ höchstens der Zahl $\lvert E_{WS,b}\rvert$ der Kanten. Wenn ein Eingang i im Entdeckungsmodus besucht wird oder die jeweils nachfolgenden Eingänge i' nicht in der *InvestMenge* enthalten sind, werden die Eingänge i' ans Ende des *Stapels* gefügt. Ein Eingang i' ist ein Nachfolger des Eingangs i, wenn der Eingang i' erreicht werden kann, indem eine ausgehende Kante an einem zugeordneten Ausgang j des Knotens v fortgesetzt wird. Die Zuordnung von Ausgängen j zu Eingängen i ist dabei nicht nur vom Typ des Knotens v, sondern auch vom Modus abhängig.

Falls der Knoten v eine Ressource bezeichnet, dann wird nur die Kante berücksichtigt, die am Ausgang j gegenüber von Eingang i anliegt. Im Fall eines alternativen Flusspunkts werden im Entdeckungsmodus die anliegenden Kanten an allen Ausgängen betrachtet, im Investitionsmodus nur die Kante am ersten Ausgang. Für selektive Flusspunkte gilt, dass stets die Kanten an allen Ausgängen betrachtet werden, unabhängig vom aktuellen Modus. Jedoch wird jede Kante höchstens zweimal beschritten, da der Entdeckungsmodus nach dem ersten Besuch des Eingangs i verlassen wird und im Investitionsmodus der Eingang i' in der *InvestMenge* ergänzt wird. Als Konsequenz werden nicht mehr als $2\,|E_{\mathrm{WS},b}|$ nachfolgende Eingänge i' ans Ende des *Stapels* gefügt. Entsprechend oft wird die Schleife durchlaufen, da jeweils genau ein Eingang vom *Stapel* entfernt wird.

Wie zuvor wird angenommen, dass für alle Flusspunkte ein maximaler Ausgangsgrad $d^{+}_{\max}$ gegeben ist, welcher als Konstante aufgefasst und damit vernachlässigt werden kann. Bezüglich des folgenden Eingangs i' und Knotens v' nach einem Ausgang j eines Knotens v wird vorausgesetzt, dass deren Bestimmung in einer Laufzeit von $\mathrm{O}(1)$ möglich ist. Gleiches gilt für die Operationen des *Stapels* sowie der Matrizen, Vektoren, Mengen und Folgen. Unter diesen Voraussetzungen können für die Zeilen im Rumpf der Funktionen und Prozeduren folgende Laufzeiten bestimmt werden (ohne aufgerufene Hilfsprozeduren):

002:	$\mathrm{O}(n)$	094 … 108:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$
003 … 004:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert V_{\mathrm{WS},b}\rvert\right)$	110:	$\mathrm{O}(1)$
005:	$\mathrm{O}\left(n+\sum_{b=1}^{n} \lvert D_{\mathrm{AL},b}\rvert\right)$	111 … 114:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$
006:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert D_{\mathrm{AL},b}\rvert\right)$	115:	$\mathrm{O}(n)$
007 … 010:	$\mathrm{O}(1)$	116:	$\mathrm{O}\left(n+\sum_{b=1}^{n} \lvert D_{\mathrm{AL},b}\rvert\right)$
011 … 012:	$\mathrm{O}(n)$	117 … 118:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert D_{\mathrm{AL},b}\rvert\right)$
013 … 014:	$\mathrm{O}(1)$	120:	$\mathrm{O}(1)$
016 … 024:	$\mathrm{O}(n)$	121 … 123:	$\mathrm{O}(\lvert R\rvert)$
025 … 050:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$	124:	$\mathrm{O}(n\,\lvert R\rvert)$
051:	$\mathrm{O}(n)$	125:	$\mathrm{O}\left(n\,\lvert R\rvert+\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$
053 … 066:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$	126 … 129:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$
068 … 092:	$\mathrm{O}\left(\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$		

Indem man die Summe der Laufzeiten über alle Zeilen bildet und hierbei kleinere Summanden sowie konstante Faktoren vernachlässigt, entspricht der Ausdruck der zu zeigenden oberen Schranke $\mathrm{O}\left(\sum_{b=1}^{n} \lvert V_{\mathrm{WS},b}\rvert+\sum_{b=1}^{n} \lvert E_{\mathrm{WS},b}\rvert\right)$. □

Bemerkung: Insbesondere die Annahme, dass die Operationen von Vektoren und Matrizen in einer Laufzeit von $\mathrm{O}(1)$ möglich sein sollen, ist in der Praxis zu prüfen. Im Allgemeinen verhält sich die Laufzeit zur Multiplikation zweier Matrizen proportional zur dritten Potenz der Zeilen und Spalten. Für die Software AURELIE wurde eine eigene Softwarebibliothek entwickelt, welche Datenstrukturen und Rechenoperationen bereitstellt, um in linearer Laufzeit über alle Komponenten einer Matrix ungleich null zu iterieren und dadurch die Multiplikation deutlich zu beschleunigen. Hierbei wird ausgenutzt, dass die Matrizen eines mathematischen Modells i. d. R. dünn besetzt sind, d. h. im Verhältnis zur Zahl der Zeilen und Spalten nur wenige Komponenten ungleich null enthalten.

4.5 Umsetzung in der Software AURELIE

Nach der theoretischen Betrachtung der Kernalgorithmen zur Validierung und Transformation eines grafischen Modells richtet sich der Blick auf deren praktische Umsetzung. Die Anwendbarkeit des Modells und der zugehörigen Algorithmen wurde durch die weltweite Einführung der Software AURELIE bei der Bosch Rexroth AG belegt. Daneben existieren noch viele weitere technische, prozessuale und organisatorische Faktoren, welche die Einführung einer Software zu einem Erfolg werden lassen (siehe u. a. Gingnell et al. 2014, außerdem Hochmuth 2011 am Beispiel von AURELIE). Aus technischer Perspektive repräsentieren die vorgestellten Algorithmen nur ca. drei Prozent des Quelltextes. Zu weiteren technischen Erfolgsfaktoren zählt die grafische Benutzeroberfläche, da sie dem Anwender den Zugang zu den Funktionen der Software ermöglicht. Jedoch würde es den Rahmen sprengen, in dem Zusammenhang auf alle Entwurfsentscheidungen einzugehen. Vielmehr soll ein Kurzüberblick zu den Funktionen und der Führung des Anwenders durch den Prozess der Modellerstellung gegeben werden. Im Anschluss daran soll bewertet werden, inwieweit die Software AURELIE die Anforderungen aus Abschnitt 2.6 im Hinblick auf die Modellierung erfüllt.

4.5.1 Funktionsübersicht und Benutzerführung

In Bezug auf die Modellierung muss eine geeignete Software die Möglichkeit bieten, die Wertströme aller Produkte an einem Produktionsstandort durch ein grafisches Modell abzubilden. Im Mittelpunkt steht ein grafischer Konfigurator, mit dessen Hilfe der Anwender zeichnerisch Wertströme erstellen kann. Das Modell muss darüber hinaus alle anderen Informationen enthalten, die zur strategischen Planung der TEK gemäß den definierten Planungszielen benötigt werden. Daraus folgt eine hierarchische Modellstruktur aus grafischen und numerischen Daten. Aus diesem Grund ist die Darstellung eines Modells in der Software AURELIE in folgende vier Stufen gegliedert:

(1) *Werk:* Definition von Produkten, Schichtmodellen und Ressourcentypen;
(2) *Plan:* Vorgabe von Stückzahlszenarien und Priorisierung untergeordneter Bereiche;
(3) *Bereich:* Anordnung von Ressourcen in einem Ressourcenlayout;
(4) *Wertstrom:* Produktfluss mit Zuordnung von Ressourcen und Prozessdaten.

Bei der Entwicklung wurde die Entscheidung getroffen, die Java Platform Standard Edition 7[1] und die Eclipse[2] Rich Client Platform (RCP) in der Version 3.7.2 als Grundlage zu verwenden. Eclipse bietet zahlreiche Vorteile wie z. B. eine Bibliothek effizienter grafischer Komponenten zur Benutzerinteraktion, eine Architektur, welche die Entwicklung und Einbindung von Plugins und Fragmenten unterstützt, sowie ein Framework, um im Hintergrund Berechnungen auszuführen und durch den Anwender zu steuern. Zudem ist die Benutzeroberfläche einer Applikation auf Basis von Eclipse übersichtlich strukturiert. Typischerweise

[1] Oracle: Java SE at a Glance. Abgerufen am 11. Februar 2018. http://www.oracle.com/technetwork/java/javase/overview/.
[2] Eclipse: Eclipse Indigo SR2 Packages. Abgerufen am 9. Dezember 2018. http://www.eclipse.org/downloads/packages/release/Indigo/SR2.

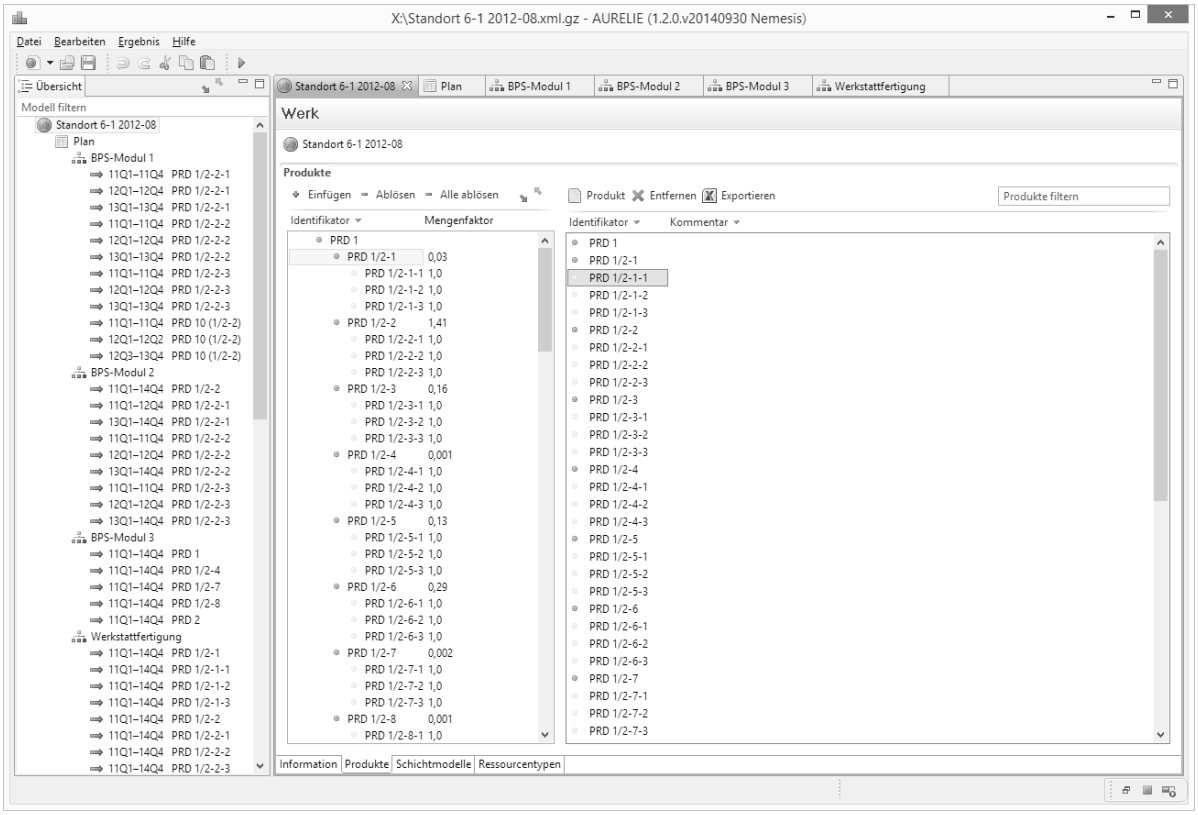


Abbildung 4.16: Bildschirmaufnahme von AURELIE (Definition von Stücklisten). Die Stücklisten bestimmen die Zuordnung von Komponenten zu Produkten, wobei jedes Produkt in einer mehrstufigen Stückliste als Komponente eines oder mehrerer anderer Produkte dienen kann. Der Anwender legt die Stücklisten der Produkte im entsprechenden Reiter fest (auf der Stufe des Werks). Der Stücklistenfaktor kann hierbei jeden reellen Wert größer als null annehmen.

wird die hierarchische Struktur des zu bearbeitenden Modells im linken Bereich der Benutzeroberfläche als Baum wiedergegeben. Im zentralen Bereich kann der Anwender Daten aus verschiedenen Teilen des Modells in übereinander gestapelten Teilfenstern anzeigen und parallel bearbeiten. Abbildung 4.16 zeigt dies am Beispiel der Software AURELIE. Im linken Bereich werden die genannten Stufen eines Modells dargestellt, auf die im nun folgenden Teil des Abschnitts eingegangen werden soll.

Modellparameter auf der Stufe des Werks

Auf der Stufe des Werks definiert der Anwender Produkte, Schichtmodelle und Ressourcentypen. Wie in Abschnitt 2.3 erläutert, ist es für die Planung zweckmäßig, Produkte und Varianten zu Produktfamilien zusammenzufassen. Die Software erlaubt es dem Anwender, die Strukturstücklisten abzubilden, indem er die Zuordnung von Komponenten zu Produkten herstellt und Stücklistenfaktoren festlegt. Jedes Produkt kann hierbei als Komponente eines anderen Produkts dienen, wie Abbildung 4.16 zeigt. Zyklen in der Produktstruktur werden automatisch erkannt, worauf eine entsprechende Warnung erscheint.

Weiterhin kann der Anwender eine beliebige Zahl von Schichtmodellen definieren, um die Zahl der Arbeitstage und der Schichten in jedem Quartal des Jahres festzulegen (siehe Beispiele in Tabelle 2.1 auf Seite 36). Unter all diesen wählt der Anwender ein sogenanntes Standardschichtmodell und ein erweitertes Schichtmodell aus. Diese beiden Schichtmodelle werden für die Kalkulation der Auslastung verwendet.

Mit Hilfe von Ressourcentypen kann der Anwender Ressourcen mit gleichen konstruktiven Eigenschaften zusammenfassen, auch wenn sie für unterschiedliche Produkte verwendet werden. Ein Ressourcentyp entspricht einer Klasse von Maschinen, Anlagen oder Einrichtungen und besitzt einen spezifischen Plan-Nutzungsgrad. Wie in Abschnitt 2.4 beschrieben, ist der Plan-Nutzungsgrad eine Zielvorgabe, weshalb Abweichungen vom Zielwert durch Kommentare begründet werden sollen. Im Rahmen der Auswertung werden die notwendigen Investitionen für jeden Ressourcentyp als Summe dargestellt.

Modellparameter auf der Stufe des Plans

In einem Modell können mehrere Pläne angelegt werden, um unterschiedliche Varianten für die Strukturierung eines Werks abzubilden. Auf der Stufe eines Plans definiert der Anwender Stückzahlszenarien und die Prioritäten der untergeordneten Bereiche. Mit Hilfe von Stückzahlszenarien ist es möglich, Annahmen bezüglich der Absatzentwicklung im Planungszeitraum zu treffen. Wie Abbildung 4.17 zeigt, legt der Anwender hierzu die geplanten Stückzahlen für alle Produkte einschließlich Primärbedarf und Sekundärbedarf fest. Dabei können Lieferungen zwischen Werken berücksichtigt werden, die je nach Vorzeichen als Bedarf oder als Kapazität in die Kalkulation einfließen.

Da Produkte ggf. in verschiedenen Bereichen hergestellt werden können, muss der Anwender für jeden Bereich eine Priorität festlegen. Zudem werden die Prozessdaten aller Wertströme in einer Tabelle zusammengefasst, um eine effiziente Eingabe der Taktzeiten und Plan-Nutzungsrade zu ermöglichen. Der Anwender kann die zugehörigen Werte bearbeiten und zur Weiterverarbeitung nach Microsoft Excel exportieren.

Modellparameter auf der Stufe des Bereichs

Bereiche dienen der räumlichen und organisationalen Strukturierung eines Werks. Beispielsweise kann der Anwender verschiedene Bereiche für die automatisierte und die manuelle Produktion anlegen oder die Abteilungen eines Werks abbilden. In jedem Bereich legt der Anwender fest, welche MAEs für die Herstellung von Produkten zur Verfügung stehen, wobei dies aktuell verfügbare und zu installierende MAEs einschließt.

Um eine Ressource zu erzeugen, muss der Anwender einen Ressourcentyp auswählen, die Position im Ressourcenlayout bestimmen und einen eindeutigen Bezeichner definieren. Abbildung 4.18 zeigt beispielhaft die Modellierung der MAEs in einem teilautomatisierten Bereich. Die Anordnung im Ressourcenlayout bildet die Grundlage für die Verknüpfung der Ressourcen in den Wertströmen. Dieser grafische Ansatz unterscheidet AURELIE von verfügbarer, insbesondere tabellarischer und formularbasierter Software für die Planung wie z. B. Microsoft Excel oder SAP APO SNP.

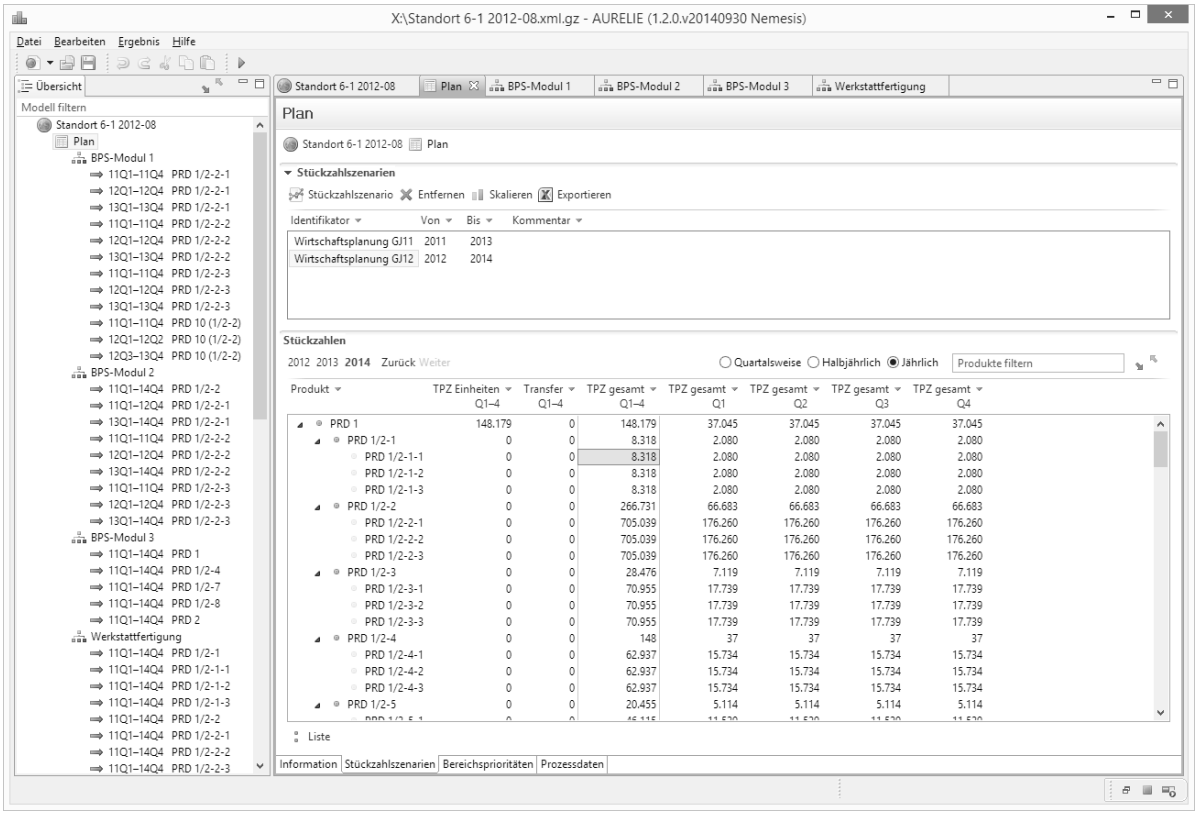


Abbildung 4.17: Bildschirmaufnahme von AURELIE (Vorgabe von Stückzahlszenarien). Die geplanten Produktstückzahlen entsprechen der Summe aus dem Primärbedarf für Kunden und Werke sowie dem Sekundärbedarf. Um eine flexible Planung zu unterstützen, können verschiedene Szenarien definiert werden, welche die Absatzentwicklung widerspiegeln. Die hierfür notwendigen Eingaben nimmt der Anwender im Reiter Stückzahlszenarien vor (auf der Stufe des Plans).

Zur Einschränkung der Verfügbarkeit einer Ressource kann der Anwender einen Gültigkeitszeitraum festlegen. Anwendungsfälle sind Maschinen, deren Beschaffung zu einem zukünftigen Datum geplant ist, oder solche, die zu einem bestimmten Datum verlagert, verkauft oder stillgelegt werden sollen. Falls der Anwender die Beschaffung einer Ressource plant, die Investition aber noch nicht genehmigt ist, kann er eine entsprechende Markierung setzen. Als Folge wird die Notwendigkeit einer Investition nicht erst bei einer Überlastung ausgewiesen, sondern bereits dann, wenn die Ressource genutzt wird. Indem der Anwender eine Ressource teilt, kann diese auch in Wertströmen anderer Bereiche eingebunden werden (nützlich für zentrale Einrichtungen eines Werks wie z. B. eine Härterei). Das erstellte grafische Ressourcenlayout kann nach Microsoft PowerPoint exportiert werden, um dieses als Grundlage für eine Präsentation zu verwenden.

Modellparameter auf der Stufe des Wertstroms

Auf der Stufe des Wertstroms erfolgt die grafische Modellierung der Prozessschritte und Prozesse zur Fertigung und Montage der Produkte. Der Wertstromgraph eines Produkts

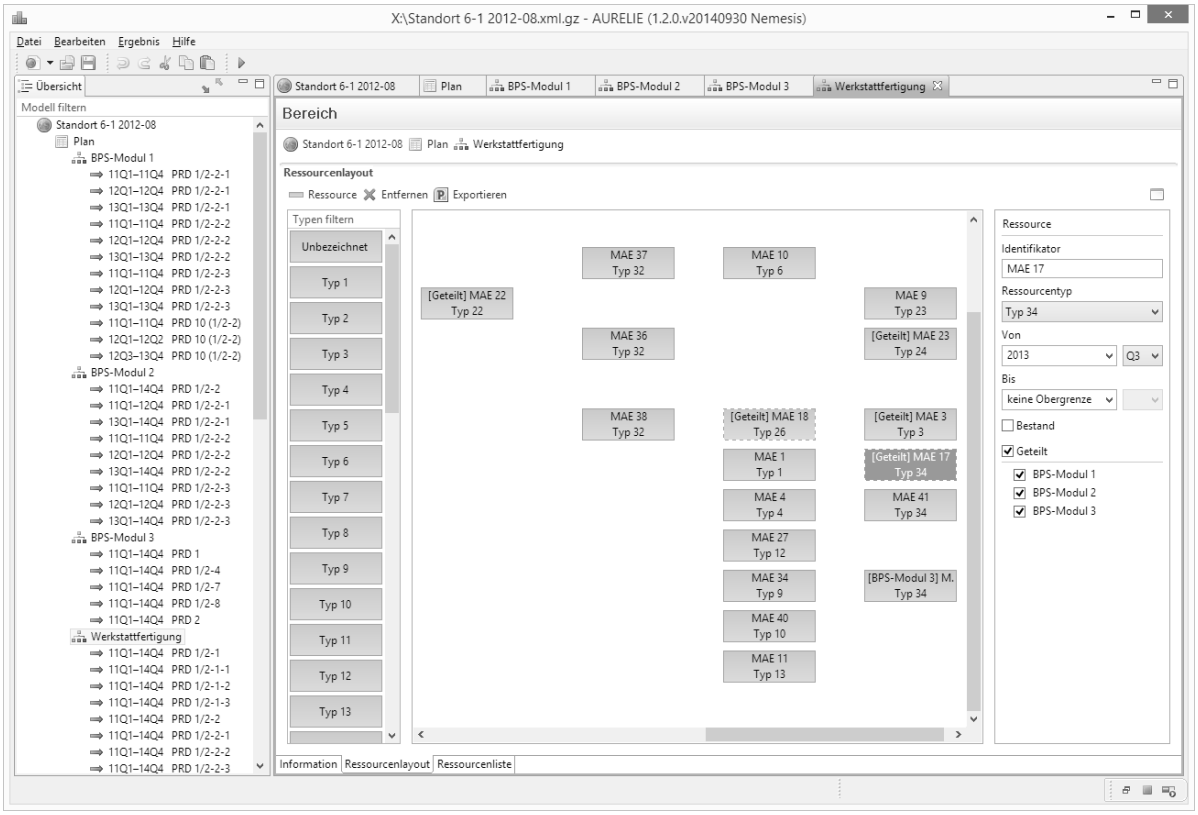


Abbildung 4.18: Bildschirmaufnahme von AURELIE (Anordnung von Ressourcen). Einer der ersten Schritte in der Modellierung ist die Abbildung der verfügbaren und zu installierenden MAEs durch Ressourcen. Der Anwender erstellt zu diesem Zweck das sogenannte Ressourcenlayout (siehe gleichnamigen Reiter auf der Stufe des Bereichs). Die Positionierung erleichtert die Verknüpfung innerhalb der Wertstromgraphen und muss nicht der räumlichen Anordnung entsprechen.

wird als Produktfluss bezeichnet. Die Software bietet dem Anwender einen zeichnerischen Zugang, um Wertströme grafisch zu modellieren, wie in Abbildung 4.19 dargestellt ist. Hierbei kommen die beschriebenen Konzepte von Knoten unterschiedlicher Typen und deren Verknüpfung an definierten Eingängen und Ausgängen zur Anwendung.

Die notwendigen Prozessdaten definiert der Anwender durch Selektion einer Ressource oder einer eingehenden Kante an einer Ressource, welche einen Prozessschritt repräsentiert. Danach kann er jeweils eine Taktzeit und einen Plan-Nutzungsgrad vorgeben, um den Wert des Ressourcentyps zu überschreiben. Alternativ zur Eingabe der Daten im Produktfluss wird auf Grundlage der Verknüpfungen automatisch eine Liste aller Prozessschritte erzeugt, die genutzt werden kann, um die Prozessdaten zu erfassen.

Innerhalb eines Bereichs kann der Anwender für jedes Produkt jeweils einen Wertstrom definieren, dessen Gültigkeit sich über den gesamten Planungszeitraum erstreckt. Indem der Anwender unterschiedliche Wertströme für jeweils verschiedene Zeitintervalle anlegt, ist es möglich, zeitliche Veränderungen in den Prozessen abzubilden. Ebenso wie das Ressourcenlayout kann der Produktfluss nach Microsoft PowerPoint exportiert werden, um die grafische Darstellung in einer Präsentation weiterzuverarbeiten.

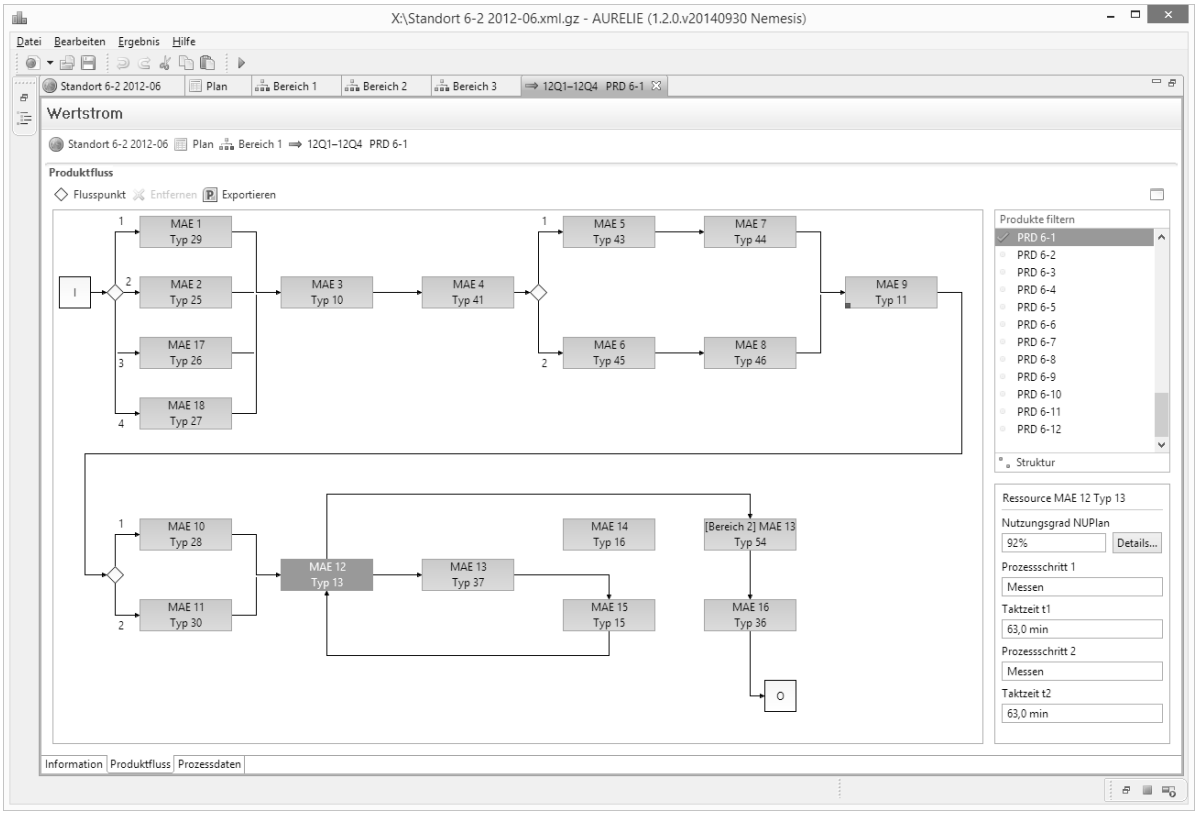


Abbildung 4.19: Bildschirmaufnahme von AURELIE (grafische Wertstrommodellierung). Nachdem der Anwender das Ressourcenlayout angelegt hat, definiert er für jedes Produkt im Planungsintervall einen Wertstrom. Hierzu verknüpft er die Ressourcen im aktuellen Bereich und definiert ggf. zusätzliche Flusspunkte (siehe Reiter Produktfluss auf der Stufe des Wertstroms). Durch automatisches Routing der Kanten wird eine schnelle Modellierung komplexer Wertströme ermöglicht.

4.5.2 Erfüllungsgrad der Anforderungen

Im Anschluss an die Bewertung verfügbarer Software im vorigen Kapitel soll nun belegt werden, dass AURELIE alle Anforderungen in Bezug auf die Modellierung erfüllt. Dies schließt insbesondere die Leistungsanforderungen A7$^+$ bzw. A8$^+$ zur grafischen Modellierung mit einer minimalen Symbolmenge und A9$^+$ zur Modelltransformation ein.

Erfüllungsgrad der Basisanforderung A1
Superposition aller Wertströme

Die Software bietet die Möglichkeit, für jedes Produkt, das in einem Werk gefertigt oder montiert wird, einen Wertstrom zu erstellen. Hierbei können Prozessschritten verschiedener Wertströme dieselben Ressourcen zugeordnet werden, wobei der Anwender für jeden Prozessschritt eine spezifische Taktzeit vorgeben kann. Durch die Strukturierung des Werks in verschiedene Bereiche kann der Anwender mehrere Wertströme für ein Produkt anlegen. Zudem kann der Anwender in einem Bereich für ein Produkt verschiedene Wertströme mit

jeweils unterschiedlicher Gültigkeit definieren, um die zeitliche Veränderung der Prozesse abzubilden. Die Basisanforderung A1 ist damit *erfüllt*.

Erfüllungsgrad der Basisanforderung A2
Systemstruktur

Die Software setzt die Konzepte zur grafischen Modellierung von Wertströmen um, die in diesem Kapitel vorgestellt wurden. Entsprechend werden Wertströme durch Wertstromgraphen, Ressourcen durch spezifische Knoten und Prozessschritte durch eingehende Kanten an diesen Knoten repräsentiert. Mit Hilfe der Unterscheidung oder Vereinigung eingehender Kanten an Ressourcen sind beide Anwendungsfälle abbildbar, die in Abschnitt 2.6 beschrieben wurden: Entweder werden die Prozessschritte an der jeweiligen MAE vereinigt, oder sie durchlaufen die MAE getrennt voneinander. Im zweiten Fall kann der Anwender jeweils unterschiedliche Taktzeiten und/oder nachfolgende Prozessschritte definieren. Auf diese Weise lässt sich die Struktur eines Systems von Wertströmen für den gegebenen Einsatzzweck vollständig abbilden, und die Basisanforderung A2 ist *erfüllt*.

Erfüllungsgrad der Basisanforderung A3
Systemschnittstelle

Nachdem der Anwender ein grafisches Modell erstellt hat, wird dieses, wie im Weiteren noch detailliert bewertet werden soll, in ein mathematisches Modell umgewandelt. Nach dieser vollständig automatisiert ablaufenden Modelltransformation liegt das Modell in Form von Matrizen und Folgen vor, die es ermöglichen, nach Vorgabe von geplanten Stückzahlen für alle Produkte und Prozessschritte die Auslastung der jeweils genutzten MAEs zu bestimmen. Die Ein- und Ausgaben des mathematischen Modells spiegeln somit die Ein- bzw. Ausgaben des Informationsflusses an der Systemschnittstelle des abzubildenden Systems wider. Entsprechend ist die Basisanforderung A3 *erfüllt*.

Erfüllungsgrad der Basisanforderungen A4–A6
Sequenzielle, alternative und selektive Verknüpfung

Der Anwender erzeugt sequenzielle Verknüpfungen in einem Wertstrom, indem er Ressourcen oder andere Knoten durch Kanten miteinander verbindet (A4). Alternative und selektive Verknüpfungen können durch alternative bzw. selektive Flusspunkte abgebildet werden. Hierbei definiert der Anwender die zugehörigen Prioritäten bzw. Quoten an den ausgehenden Kanten der Flusspunkte (A5 bzw. A6). Gegenüber den Definitionen aus Abschnitt 2.2 ist es möglich, mehr als zwei Operanden durch eine alternative oder selektive Verknüpfung zu verbinden, wodurch sich das grafische Modell einfacher und verständlicher darstellen lässt. Zudem kann der Anwender beliebig viele Flusspunkte nacheinander verbinden und dadurch Verkettungen von Verknüpfungen verschiedener Typen abbilden. Die Bedingungen, wie sie in den Definitionen aus Abschnitt 2.2 formuliert sind, werden bei der Transformation des grafischen Modells in ein mathematisches Modell berücksichtigt. Damit sind die Basisanforderungen A4–A6 jeweils vollständig *erfüllt*.

Erfüllungsgrad der Leistungsanforderung A7$^+$
Grafische Modellierung

Wie in diesem Kapitel beschrieben wurde, erlaubt es die Software AURELIE, ein gegebenes System von Wertströmen grafisch abzubilden. Zu diesem Zweck stellt die Software Symbole bereit, welche den eingeführten Knotentypen entsprechen und sich frei kombinieren lassen. Im Gegensatz zu vielen anderen Softwareanwendungen und üblichen grafischen Notationen können sowohl die erforderlichen Elemente des Systems als auch die Beziehungen zwischen diesen Elementen mit Hilfe solcher Symbole ausgedrückt werden. Der Anwender muss allein notwendige Prozessdaten wie z. B. Taktzeiten, Betriebsmittelzeiten und Quoten ergänzen. Die bereitgestellten Symbole sind eindeutig, und auch ihre Kombination lässt keinen Raum für Mehrdeutigkeit. Als Konsequenz kann jedes gültige, vom Anwender erstellte grafische Modell eindeutig in ein mathematisches Modell umgewandelt und optimiert werden. Die Leistungsanforderung A7$^+$ ist daher *erfüllt*.

Erfüllungsgrad der Leistungsanforderung A8$^+$
Minimale Symbolmenge

Im vorherigen Absatz wurden die Symbole angesprochen, welche die Software AURELIE zur grafischen Modellierung bereitstellt. Diese entsprechen den Knotentypen, die zur Modellierung von Wertstromgraphen differenziert werden müssen, und den Kanten, welche die Knoten verbinden. Das heißt, es existiert jeweils ein Symbol für Quellen und Senken, alternative und selektive Flusspunkte sowie Ressourcen (vgl. ordnungsmäßige Modellvisualisierung, Deelmann und Loos 2004, S. 290). Theoretisch könnten Quellen und Senken mit demselben Symbol dargestellt werden, jedoch wurde zugunsten der Verständlichkeit des grafischen Modells darauf verzichtet. Als Ergebnis ist die Symbolmenge als minimal zu betrachten, und die Leistungsanforderung A8$^+$ ist *erfüllt*.

Erfüllungsgrad der Leistungsanforderung A9$^+$
Modelltransformation

Nachdem der Anwender mit AURELIE ein gegebenes System grafisch modelliert und alle Eingaben zur Festlegung von Prozessdaten und geplanten Produktstückzahlen vorgenommen hat, ruft er die Funktion zur Kalkulation des Modells auf. Daraufhin validiert die Software das grafische Modell, wandelt es nach erfolgreicher Validierung automatisch in ein mathematisches Modell um und optimiert dieses gemäß den definierten Planungszielen. Im Zuge der Transformation in ein mathematisches Modell bleiben alle Modellelemente und Beziehungen zwischen den Modellelementen erhalten. Das Ergebnis der Modelltransformation liegt in Form der Matrizen und Folgen vor, die in diesem Kapitel beschrieben wurden. Auf dieser Basis können exakte Verfahren zur mathematischen Optimierung eingesetzt werden. Auch die Leistungsanforderung A9$^+$ ist damit *erfüllt*.

4.6 Fazit: Erreichen des vorgegebenen Entwicklungsziels

In der Diskussion zum Stand der Technik wurde festgestellt, dass keine Software existiert, welche die strategische Planung der TEK wie gefordert unterstützt. Keiner der evaluierten Softwaretypen erlaubt es, ein System von Wertströmen mit Hilfe einer minimalen Menge grafischer Symbole abzubilden, das resultierende Modell automatisch in ein mathematisches Modell zu transformieren und gemäß den Planungszielen zu optimieren. Um diese Lücke zu schließen, war es das Ziel, in einem ersten Lösungsschritt die Anforderungen in Bezug auf die Modellierung zu erfüllen. Mit den Erkenntnissen aus dem zurückliegenden Kapitel lässt sich zusammenfassen, dass die Software AURELIE dieses Entwicklungsziel erreicht. Zum Abschluss des Kapitels liegen nun folgende Ergebnisse vor:

(1) Beschreibung eines formal eindeutigen, *grafischen Modells*: Weiterentwicklung von Graphen zu Wertstromgraphen durch Einführung spezifischer Knotentypen und deren Verknüpfung an definierten Eingängen und Ausgängen (siehe Abschnitt 4.2);

(2) Algorithmus zur *Validierung* eines grafischen Modells: detaillierte Beschreibung von Ziel, Grundidee, Datenstrukturen und Ablauf sowie Beweis der Zeitkomplexität (siehe Abschnitt 4.3 sowie Abschnitt A.1.5 und A.1.6 im Anhang);

(3) Algorithmus zur *Transformation* in ein mathematisches Modell: Definition der mathematischen Modellstruktur, wie zuvor Beschreibung und Beweis der Zeitkomplexität (siehe Abschnitt 4.4 sowie Abschnitt A.1.7 und A.1.8 im Anhang);

(4) Beschreibung der *Umsetzung* in der Software AURELIE: Funktionsübersicht und Benutzerführung sowie Beleg der Erfüllung aller formulierten Anforderungen in Bezug auf die Modellierung (siehe Abschnitt 4.5).

Damit ist der erste Lösungsschritt abgeschlossen, und es gilt im verbleibenden, zweiten Lösungsschritt, das mathematische Modell zu optimieren. Im folgenden Kapitel wird an das Ergebnis der Modelltransformation angeknüpft, indem die eingeführten Vektoren, Matrizen und Folgen wieder aufgegriffen werden. Sie werden genutzt, um abhängig vom vorgegebenen Planungsziel Zielfunktionen und Nebenbedingungen zu formulieren.

5 Lösungsschritt II: mathematische Optimierung

In der Analyse des Stands der Technik wurde festgestellt, dass es einer neuen Software bedarf, welche den Workflow zur strategischen Planung der TEK unterstützt. Um diese Lücke zu schließen, wurde die Software AURELIE entwickelt und bei der Bosch Rexroth AG eingeführt. Wie im vorigen Kapitel dargelegt, erlaubt es die Software AURELIE, ein System von Wertströmen grafisch zu modellieren, das Modell zu validieren und in ein mathematisches Modell zu transformieren. Nach diesem ersten Lösungsschritt folgt nun durch die Optimierung des mathematischen Modells der zweite, um den softwaregestützten Workflow fortzuführen. Hierfür wird auf Standardverfahren der mathematischen Optimierung zurückgegriffen, die wiederholt mit veränderten Parametern aufgerufen werden. Der wissenschaftliche Beitrag besteht in den eigens entwickelten Algorithmen, in deren Verlauf jeweils geeignete Zielfunktionen und Nebenbedingungen formuliert und optimiert werden. Wie im vorherigen Kapitel werden die entsprechenden Kernalgorithmen erläutert und durch Kurzfassungen in natürlicher Sprache veranschaulicht. Die vollständigen Fassungen in formalem, kommentiertem Pseudocode sind mitsamt allen Datenstrukturen im Anhang zu finden.

Der Begriff der mathematischen Optimierung bezieht sich in diesem Zusammenhang auf die Maximierung oder Minimierung einer Zielfunktion unter Berücksichtigung von Nebenbedingungen. Es werden die drei Planungsziele betrachtet, die in Abschnitt 2.3 eingeführt und in Abschnitt 2.6 durch Anforderungen konkretisiert wurden: die Maximierung der Kapazitäten, die Minimierung der notwendigen Investitionen und die Optimierung der Auslastung. Die Optimierungsvariablen sind hierbei die Eingaben des Informationsflusses, welche das Modell anstelle des Systems mit seiner Umwelt austauscht. Bei diesen Eingaben handelt es sich um die Stückzahlen aller Produkte und deren Verteilung in den Wertströmen durch die jeweils zugeordneten Prozessstückzahlen. Auf Basis der Stückzahlen werden die Werte für die Auslastung der Produktionsanlagen bestimmt und als Ausgaben des Modells zurückgegeben. Die Eingaben und Ausgaben werden genutzt, um geeignete Zielfunktionen und Nebenbedingungen zu formulieren.

Das Kapitel ist analog zum vorhergehenden wie folgt aufgebaut: In Abschnitt 5.1 werden die Begriffe der linearen Optimierung und der Korrektheit von Algorithmen eingeführt, die in diesem Kapitel benötigt werden. Dem schließt sich die mathematische und algorithmische Betrachtung der drei oben genannten Planungsziele an. Entsprechend folgt in Abschnitt 5.2 die Maximierung der Kapazitäten, in Abschnitt 5.3 die Minimierung der Investitionen und in Abschnitt 5.4 die Optimierung der Auslastung. Zuerst wird jeweils das Ziel des Algorithmus von den Anforderungen und den Abhängigkeiten in den Wertströmen des Systems abgeleitet. Daraufhin wird der Ablauf einschließlich wesentlicher Datenstrukturen beschrieben, bevor die Korrektheit und die Zeitkomplexität des Algorithmus bewiesen werden. Wie im Fall des

vorherigen Kapitels richtet sich der Blick danach in Abschnitt 5.5 auf die Umsetzung der Konzepte in der Software AURELIE. Abgeschlossen wird das Kapitel mit Abschnitt 5.6 durch den Beleg, dass das vorgegebene Entwicklungsziel erreicht wurde.

5.1 Kurzeinführung: lineare Optimierung und Korrektheit

Nach der Transformation eines grafischen Modells liegt dieses in Gestalt eines mathematischen Modells vor, welches sich zur Optimierung eignet. Im weiteren Verlauf des Kapitels werden Algorithmen vorgestellt, deren Ablauf auf einer iterativen Maximierung oder Minimierung linearer Zielfunktionen basiert. Um die Aussage zu belegen, dass diese Algorithmen das jeweils formulierte Ziel erreichen, muss ihre Korrektheit nachgewiesen werden. In diesem Abschnitt werden die hierzu notwendigen Grundlagen der linearen Optimierung und des Beweises der Korrektheit von Algorithmen eingeführt.

5.1.1 Lineare Optimierung

Wie in Abschnitt 4.2 beschrieben, erstellt der Anwender zur Abbildung eines Systems von Wertströmen ein grafisches Modell. Dieses wird in ein mathematisches Modell transformiert, welches gemäß den Definitionen in Abschnitt 4.4 als ein Tupel definierter Matrizen dargestellt werden kann. Die Matrizen werden genutzt, um lineare Zielfunktionen und Nebenbedingungen zu formulieren und die Zielfunktionen zu maximieren oder zu minimieren. Hierzu wird auf Standardverfahren der linearen Optimierung zurückgegriffen.

Zusätzlich zu den Skalaren, Vektoren und Matrizen aus Abschnitt 4.2 und 4.4, die weiterhin gelten, werden folgende Symbole eingeführt.

$\mathbf{A}$	Matrix reeller Koeffizienten eines Vektors $\mathbf{x}$, $\forall i \in \{1,2,\ldots\}, a \in \{1,\ldots,m\}\ \mathbf{A}[i,a] \in \mathbb{R}$
$\mathbf{b}$	Vektor reeller oberer Schranken, $\mathbf{Ax} \leqslant \mathbf{b}$, $\forall i \in \{1,2,\ldots\}\ \mathbf{b}[i,1] \in \mathbb{R}$
$\mathbf{c}$	Vektor reeller Koeffizienten eines Vektors $\mathbf{x}$, $\forall a \in \{1,\ldots,m\}\ \mathbf{c}[a,1] \in \mathbb{R}$
$f(\mathbf{x})$	lineare Zielfunktion für einen Vektor $\mathbf{x}$, $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$, $f(\mathbf{x}) \in \mathbb{R}$
$\mathbf{y}_{\text{plan}}$	Vektor $[y_{\text{plan},1}\ \ldots\ y_{\text{plan},n}]^\top$ der geplanten Produktstückzahlen $y_{\text{plan},b}$, je nach Planungsziel ggf. vorläufig festgelegt, $\mathbf{y}_{\text{plan}} \in \mathbb{R}^n_{>0}$
$y_{\text{plan},b}$	geplante Stückzahl des Produkts $b \in \{1,\ldots,n\}$, $y_{\text{plan},b} \in \mathbb{R}_{>0}$

Klassen mathematischer Optimierung. Die lineare Optimierung bezeichnet ein Teilgebiet der konvexen Optimierung, die ihrerseits ein Teilgebiet der mathematischen Optimierung darstellt. Um diese Klassen eindeutig beschreiben zu können, müssen zunächst die Begriffe des mathematischen Optimierungsproblems, der Zielfunktion und der Nebenbedingung konkretisiert werden. Ein entsprechendes Optimierungsproblem ist dadurch definiert, dass es gilt, eine Funktion $f(\mathbf{x})$ eines Vektors $\mathbf{x}$ zu maximieren oder zu minimieren. Zur Beschränkung der Lösungen ist eine endliche Zahl von weiteren Funktionen des Vektors $\mathbf{x}$ gegeben, die jeweils eine obere oder untere Schranke nicht überschreiten bzw. unterschreiten dürfen. Der Vektor $\mathbf{x}$ bezeichnet die Optimierungsvariablen, die Funktion $f(\mathbf{x})$ die Zielfunktion, und die weiteren Funktionen mit den zugehörigen Schranken bilden die Nebenbedingungen. Eine optimale Lösung liegt vor, wenn die Zielfunktion $f(\mathbf{x})$ unter allen gültigen Vektoren $\mathbf{x}$

ihren maximalen oder minimalen Wert annimmt. Dabei ist ein Vektor **x** genau dann gültig, wenn dieser alle Nebenbedingungen erfüllt (siehe u. a. Aigner 2006, Boyd und Vandenberghe 2004, Hochstättler 2010, Korte und Vygen 2012, Unger und Dempe 2010).

Unter Verwendung dieser Begriffe ist es möglich, Optimierungsprobleme nach der Form der Zielfunktion und der Nebenbedingungen zu unterscheiden. Eine wichtige Klasse von Optimierungsproblemen bildet die konvexe Optimierung, gemäß welcher die Zielfunktion und die Menge aller gültigen Lösungen konvex sein müssen. Geometrisch bedeutet dies, dass alle Lösungen, die sich auf der Strecke zwischen zwei gültigen Lösungen befinden, ebenfalls gültig sein müssen (siehe Aigner 2006, S. 307). Probleme dieser Art sind dadurch gekennzeichnet, dass jedes lokale Optimum zugleich ein globales Optimum ist.

Innerhalb der konvexen Optimierung repräsentiert die lineare Optimierung ein besonders wichtiges Teilgebiet. Im Fall der linearen Optimierung wird zusätzlich gefordert, dass es sich sowohl bei der Zielfunktion als auch den Nebenbedingungen um lineare Funktionen handelt (siehe Hochstättler 2010, S. 199). In diesem Fall bildet die Menge der gültigen Lösungen ein Polytop, wie anhand der Beispiele AL2 und SL2 in Abschnitt 2.5 gezeigt wurde. Eine Vielzahl betriebswirtschaftlicher Fragestellungen lässt sich durch lineare Optimierungsprobleme ausdrücken (für Beispiele siehe u. a. Unger und Dempe 2010).

Formulierung linearer Optimierungsprobleme. Um lineare Optimierungsprobleme einheitlich zu beschreiben, wird in der Praxis eine sogenannte *Standardform* verwendet. Diesbezüglich gilt die Vereinbarung, dass der Vektor **x** der Optimierungsvariablen ausschließlich nichtnegative reelle Werte enthält. Zur Definition der Zielfunktion $f(\mathbf{x})$ wird ein Vektor **c** mit reellen Komponenten angegeben, dessen Zahl der Zeilen mit der Zahl der Zeilen des Vektors **x** übereinstimmt. Die Komponenten des Vektors **c** entsprechen den Koeffizienten der Zielfunktion $f(\mathbf{x})$. Daneben wird eine Matrix **A** mit reellen Komponenten definiert, deren Zahl der Spalten der Zahl der Zeilen des Vektors **x** entspricht. Weiterhin wird ein Vektor **b** angegeben, welcher im Hinblick auf die Zahl der Zeilen mit der Matrix **A** übereinstimmt. Die Matrix **A** und der Vektor **b** beschreiben die Nebenbedingungen, welche die Zielfunktion $f(\mathbf{x})$ beschränken. Unter Verwendung dieser Vektoren und Matrizen lassen sich lineare Optimierungsprobleme wie folgt definieren (siehe Aigner 2006, S. 296):

$$\max\left\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \geqslant \mathbf{0} \wedge \mathbf{A}\mathbf{x} \leqslant \mathbf{b}\right\}. \tag{5.1}$$

Die Zielfunktion $f(\mathbf{x})$ wird in Gleichung (5.1) als Matrixprodukt des transponierten Vektors **c** und des Vektors **x** wiedergegeben. Um Doppelbelegungen mit anderen Symbolen zu vermeiden, wird zur Bezeichnung der Komponenten von Vektoren und Matrizen im Folgenden die Notation verwendet, die im vorigen Kapitel eingeführt wurde. Entsprechend lässt sich die Zielfunktion $f(\mathbf{x})$ wie folgt auflösen:

$$\mathbf{c}^\top \mathbf{x} = \mathbf{c}[1,1]\, x_1 + \mathbf{c}[2,1]\, x_2 + \ldots + \mathbf{c}[m,1]\, x_m. \tag{5.2}$$

Die Zeilen der Matrix **A** beschreiben mit den jeweils entsprechenden Zeilen des Vektors **b** die Nebenbedingungen. Die Komponenten der Matrix **A** repräsentieren die Koeffizienten des Vektors **x** und die Komponenten des Vektors **b** die zugehörigen oberen Schranken. Löst

man die Bedingung in Gleichung (5.1) auf, führt dies zu folgenden Ungleichungen, welche die in Gleichung (5.2) angegebene Zielfunktion $f(\mathbf{x})$ beschränken:

$$\begin{aligned}
&\mathbf{A}[1,1]\,x_1 + \mathbf{A}[1,2]\,x_2 + \ldots + \mathbf{A}[1,m]\,x_m \leqslant \mathbf{b}[1,1]\,,\\
&\mathbf{A}[2,1]\,x_1 + \mathbf{A}[2,2]\,x_2 + \ldots + \mathbf{A}[2,m]\,x_m \leqslant \mathbf{b}[2,1]\,,\\
&\mathbf{A}[3,1]\,x_1 + \mathbf{A}[3,2]\,x_2 + \ldots + \mathbf{A}[3,m]\,x_m \leqslant \mathbf{b}[3,1]\,,\\
&\ldots \qquad\quad \ldots \qquad\quad \ldots \qquad\quad \ldots
\end{aligned} \tag{5.3}$$

Die Auslassungszeichen in Gleichung (5.3) zeigen, dass die Matrix **A** entsprechend der Zahl der Nebenbedingungen beliebig viele Zeilen besitzen kann. Jedoch wird verlangt, dass die oben genannten Bedingungen in Bezug auf die Zahlen der Zeilen der Matrix **A** und des Vektors **b** gelten. Durch Negation der Komponenten des Vektors **c** kann die Standardform auf Probleme der Maximierung und der Minimierung angewandt werden. Indem man die Zeilen der Matrix **A** und des Vektors **b** negiert, ist es möglich, Nebenbedingungen mit oberen und unteren Schranken zu beschreiben. Eine Nebenbedingung, welche die Gleichheit mit einem Wert fordert, kann durch zwei Nebenbedingungen mit einer oberen und einer entsprechenden unteren Schranke in die Standardform übertragen werden. Variablen, die auch negative Werte annehmen dürfen, können beschrieben werden, indem diese als Differenz zweier nichtnegativer Variablen ausgedrückt werden.

Lösung linearer Optimierungsprobleme. Zunächst kann festgestellt werden, dass eine optimale Lösung genau dann existiert, wenn mindestens eine Lösung gültig und die Zielfunktion beschränkt ist. Hierbei muss es sich im Fall der Maximierung der Zielfunktion um eine obere Schranke und im Fall der Minimierung um eine untere Schranke handeln (siehe Aigner 2006, S. 303). Da im Folgenden jeweils die Startlösung gültig ist und eine solche Schranke existiert, ist es in jedem Fall möglich, eine optimale Lösung zu finden.

Zur Lösung linearer Optimierungsprobleme existieren drei Typen universell einsetzbarer Verfahren: das Simplex-Verfahren, das Innere-Punkte-Verfahren und die Ellipsoidmethode. Das Simplex-Verfahren ist am weitesten verbreitet und gilt als effizient, wenn auch keine Variante existiert, welche eine polynomielle Laufzeit garantiert. Innere-Punkte-Verfahren gewinnen zunehmend an Bedeutung, da sie als effizient gelten und zudem versprechen, die optimale Lösung in polynomieller Laufzeit zu finden. Zwar besitzt die Ellipsoidmethode eine polynomielle Laufzeit, jedoch ist das Verfahren in der Praxis nicht effizient einsetzbar (siehe u. a. Hochstättler 2010, Korte und Vygen 2012).

Die drei Verfahren werden seit ihrer Einführung im Hinblick auf Effizienz, numerische Stabilität und andere Kriterien weiterentwickelt. In dieser Arbeit wird auf bestehende Implementierungen zurückgegriffen, um Algorithmen zu entwickeln, in deren Verlauf lineare Zielfunktionen maximiert oder minimiert werden. Zu diesem Zweck wird in der vorliegenden Arbeit die Bibliothek `lp_solve`[1] genutzt, die in verschiedene Programmiersprachen eingebunden werden kann. Die Bibliothek ist frei verfügbar und stellt effiziente, universell einsetzbare Standardverfahren zur linearen Optimierung bereit.

[1] `lp_solve` Reference Guide: Introduction to `lp_solve` 5.5.2.5. Abgerufen am 22. Juni 2018. http://lpsolve.sourceforge.net/5.5/.

Um ein Beispiel anzugeben, nach welchem Muster in den nachfolgenden Abschnitten die Lösung von Optimierungsproblemen beschrieben wird, veranschaulicht Algorithmus A.2.1 im Anhang die Minimierung einer allgemeinen linearen Zielfunktion $f(\mathbf{x})$. Im ersten Schritt wird eine Startlösung $\mathbf{x}$ ermittelt, welche eine definierte Menge von Bedingungen erfüllen muss. Im vorliegenden Fall muss die Lösung $\mathbf{x}$ den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entsprechen, wie in Abschnitt 4.4 definiert. Darüber hinaus gelten die Bedingungen in Bezug auf den Flusserhalt und die Auslastung, die in Abschnitt 4.4 eingeführt wurden. Um eine Startlösung $\mathbf{x}$ zu ermitteln, die diesen Bedingungen genügt, wird die Summe der Stückzahlen y_b aller Produkte b maximiert, wobei diese die geplanten Produktstückzahlen $y_{\text{plan},b}$ nicht überschreiten dürfen. Unter der Voraussetzung, dass mindestens eine gültige Lösung existiert, ist dies äquivalent zur Maximierung der Stückzahl y_b jedes Produkts b bis zum jeweils geplanten Wert $y_{\text{plan},b}$. Im zweiten Schritt wird die daraus folgende Startlösung $\mathbf{x}$ verwendet, um ausgehend von dieser die Zielfunktion $f(\mathbf{x})$ zu minimieren. Hierbei müssen die Stückzahlen y_b aller Produkte b beibehalten werden und die zuvor genannten Bedingungen in Bezug auf den Flusserhalt und die Auslastung erfüllt sein. Zuletzt wird der gefundene Wert der Zielfunktion $f(\mathbf{x})$ als Ergebnis zurückzugeben.

5.1.2 Korrektheit von Algorithmen

Im Gegensatz zu den Algorithmen im vorigen Kapitel, die auf Standardverfahren der Graphentheorie basieren, existiert für die Algorithmen in diesem Kapitel keine vergleichbare Grundlage. Ergänzend zur Beschreibung des Ablaufs und dem Beweis der Zeitkomplexität der Algorithmen muss daher auch deren Korrektheit nachgewiesen werden. Unter der Korrektheit wird die Eigenschaft eines Algorithmus verstanden, einer Spezifikation zu genügen, indem ein vorgegebenes formales Ziel erreicht wird. Damit ist nicht gesagt, dass die zugrundeliegende Spezifikation dazu geeignet ist, das vorliegende Problem zu lösen. Aus diesem Grund muss der Formulierung des Ziels eine Diskussion zum jeweils gewählten Ansatz vorangestellt werden. Im Folgenden werden die Grundzüge der Beweisführung dargelegt, nach denen die Korrektheit eines Algorithmus gezeigt werden kann. Für weiterführende Erläuterungen wird auf die Literatur verwiesen (siehe u. a. Cormen et al. 2010 zu Grundlagen von Algorithmen und z. B. Liggesmeyer 2009 zur Verifikation von Software).

Partielle und totale Korrektheit. Zunächst muss zwischen partieller und totaler Korrektheit eines Algorithmus unterschieden werden. Ein Algorithmus ist genau dann partiell korrekt, wenn nach einer Eingabe, welche eine definierte Vorbedingung erfüllt, jede Ausgabe einer definierten Nachbedingung genügt. Hierbei wird keine Aussage darüber getroffen, ob der Algorithmus für jede gültige Eingabe eine Ausgabe liefert. Ist dies aber der Fall, spricht man nicht mehr nur von partieller, sondern von totaler Korrektheit. Folglich ist ein Algorithmus genau dann total korrekt, wenn er partiell korrekt ist und für eine gültige Eingabe stets eine Ausgabe zurückgibt. Im Rahmen dieser Kurzeinführung wird vorausgesetzt, dass der zu untersuchende Algorithmus auf einer Schleife basiert. Diese Voraussetzung ist im Fall der folgenden Algorithmen erfüllt und erlaubt es jeweils, zum Beweis der Korrektheit sogenannte Schleifeninvarianten und Schleifenvarianten zu formulieren.

Zum Beweis der *partiellen Korrektheit* existieren verschiedene Verfahren wie z. B. das wp-Kalkül (engl. *Weakest Precondition*, siehe Dijkstra 1976). Basiert der betrachtete Algorithmus wie vorausgesetzt auf einer Schleife, kann eine sogenannte *Schleifeninvariante* definiert werden. Darunter wird ein logischer Ausdruck verstanden, der zu jedem Zeitpunkt im Ablauf der Schleife wahr ist, sofern die Eingabe der Vorbedingung genügt. Zudem muss das formale Ziel des Algorithmus genau dann erreicht sein, wenn sowohl die Schleifeninvariante als auch die negierte Schleifenbedingung erfüllt sind. Danach ist zu zeigen, dass die Schleifeninvariante vor Beginn der Schleife, nach jedem Durchlauf und als Folge nach deren Beendigung gilt. Hierzu wird das Prinzip der vollständigen Induktion genutzt, indem die Gültigkeit der Schleifeninvariante nach dem jeweils vorhergehenden Durchlauf vorausgesetzt wird. Weiter wird im Folgenden vorausgesetzt, dass die Prüfung der Schleifenbedingung die Schleifeninvariante nicht verändert. Da die Schleifenbedingung nach Verlassen der Schleife falsch sein muss, ist dadurch bewiesen, dass das Ziel des Algorithmus erreicht wird.

Im Anschluss daran gilt es, die *totale Korrektheit* des Algorithmus nachzuweisen, wofür z. B. das Hoare-Kalkül angewandt werden kann (siehe Hoare 1969). Wird wieder vorausgesetzt, dass der Algorithmus auf einer Schleife basiert, muss gezeigt werden, dass die Zahl der Schleifendurchläufe endlich ist. Das ist genau dann der Fall, wenn die Schleifenbedingung nach einer endlichen Zahl von Durchläufen der Schleife verletzt wird. Eine Möglichkeit, einen entsprechenden Beweis zu führen, besteht in der Definition einer sogenannten *Schleifenvariante*, einem Ausdruck, welcher eine natürliche Zahl beschreibt. Die Schleifenvariante muss vor Beginn der Schleife größer als null sein und mit jedem Durchlauf um einen Wert größer als null reduziert werden. Weiterhin muss die Schleifenbedingung immer dann verletzt sein, sobald die Schleifenvariante den Wert null erreicht. Wenn es gelingt, eine solche Schleifenvariante zu formulieren, dann ist damit zugleich bewiesen, dass die Schleife nach einer endlichen Zahl von Durchläufen verlassen wird.

Grenzen der Beweisbarkeit. Den Ausführungen ist hinzuzufügen, dass ein Nachweis der Korrektheit eines Algorithmus nicht immer möglich ist. Zwar ist es in bestimmten Fällen durchaus möglich, einen Beweis zu führen, um die partielle und die totale Korrektheit zu bestätigen. Im Allgemeinen gehört aber die Klärung der Frage, ob ein Algorithmus korrekt ist, zur Klasse der nicht berechenbaren Probleme. Die Grundlagen hierzu sind in der Literatur unter dem Begriff des Halteproblems und dem Gödelschen Unvollständigkeitssatz zu finden (siehe Davis 1958 und die darauf Bezug nehmenden Arbeiten).

5.2 Maximierung der Kapazitäten

Wie in Abschnitt 2.3 erläutert, besteht das erste Planungsziel darin, die maximal erreichbaren Kapazitäten zur Herstellung der gegebenen Produkte zu ermitteln. Gesucht sind die maximalen Produktstückzahlen, bezeichnet als technische Kapazitäten $\mathbf{y}_{\max}$, welche mit Hilfe der verfügbaren und zu installierenden MAEs erzielt werden können. Diesbezüglich gilt die Bedingung, dass die Auslastung keiner genutzten MAE einen Wert von eins überschreiten darf. Das Ergebnis wird im weiteren Verlauf des Planungsprozesses genutzt, um über Eigenproduktion und Zukauf zu entscheiden.

Zur formal eindeutigen Beschreibung des Algorithmus werden die folgenden Symbole eingeführt. Diese gelten ergänzend zu allen weiteren Symbolen, die in den vorhergehenden Abschnitten und im entsprechenden Verzeichnis definiert sind.

$B_{\max}$	Menge der Indizes aller Produkte b, deren Stückzahl y_b maximal ist, $B_{\max} \subseteq \{1,\dots,n\}$
k	natürliche Zahl, bezeichnet den aktuellen Iterationsschritt, $k \in \mathbb{N}_0$
p	Index zur Bezeichnung des Referenzprodukts im aktuellen Iterationsschritt, beliebig ausgewählt unter allen Produkten b, deren Stückzahl y_b nicht maximal ist, $p \in \{1,\dots,n\}$
$\mathbf{x}'$	Vektor $[x'_1 \dots x'_m]^\top$ der Prozessstückzahlen x'_a, vorherige oder zu vergleichende Lösung, $\mathbf{x}' \in \mathbb{R}^m_{\geqslant 0}$
x'_a	vorherige oder zu vergleichende Teillösung mit Index $a \in \{1,\dots,m\}$, $x'_a \in \mathbb{R}_{\geqslant 0}$
$\mathbf{y}'$	Vektor $[y'_1 \dots y'_n]^\top$ der Stückzahlen y'_b aller gegebenen Produkte b, zur Lösung $\mathbf{x}'$ zugehörig, $\mathbf{y}' \in \mathbb{R}^n_{\geqslant 0}$
y'_b	Stückzahl des Produkts $b \in \{1,\dots,n\}$, zur Lösung $\mathbf{x}'$ zugehörig, $y'_b \in \mathbb{R}_{\geqslant 0}$
$\mathbf{y}_{\max}$	Vektor $[y_{\max,1} \dots y_{\max,n}]^\top$ der maximalen Produktstückzahlen $y_{\max,b}$ (TEK), $\mathbf{y}_{\max} \in \mathbb{R}^n_{\geqslant 0}$
$y_{\max,b}$	maximale Stückzahl des Produkts $b \in \{1,\dots,n\}$ (TEK), $y_{\max,b} \in \mathbb{R}_{\geqslant 0}$
ε	beliebig kleine Zahl größer als null, verwendet als Schranke, $\varepsilon \in \mathbb{R}_{>0}$

Der Algorithmus zur Maximierung der technischen Kapazitäten $\mathbf{y}_{\max}$ wird nach dem gleichen Vorgehen wiedergegeben, welches bereits für die Algorithmen im vorigen Kapitel angewandt wurde. Nach der Formulierung des Ziels folgt die Beschreibung des Ablaufs, bevor die Korrektheit und die Zeitkomplexität nachgewiesen werden.

5.2.1 Ziel, Grundidee und Datenstrukturen

Das Ziel des Algorithmus folgt aus der Beschreibung in Abschnitt 2.3 und der zugehörigen, zusammenfassenden Anforderung A10 in Abschnitt 2.6. Wie an den genannten Stellen erläutert, besteht die wesentliche Herausforderung in der Festlegung einer Strategie, welche die Auswahl einer nicht dominierten Lösung beschreibt. Im Allgemeinen existiert im reellwertigen Suchraum eine unendliche Zahl von Lösungen, die einander nicht dominieren. Zwei Lösungen dominieren sich nicht gegenseitig, wenn folgende zwei Bedingungen erfüllt sind: Gemäß der ersten Lösung ist die TEK im Fall mindestens eines Produkts größer als die TEK desselben Produkts gemäß der zweiten Lösung. Umgekehrt ist gemäß der zweiten Lösung die TEK im Fall mindestens eines anderen Produkts größer als die TEK desselben Produkts gemäß der ersten Lösung. Das heißt, keine der beiden Lösungen ist der jeweils anderen eindeutig vorzuziehen, und beide Lösungen sind als optimal anzusehen. Hintergrund ist, dass zur Herstellung verschiedener Produkte in vielen Fällen dieselben MAEs genutzt werden. Als Folge kann häufig die TEK eines Produkts ohne Auswirkung auf die Auslastung der MAEs gesteigert werden, indem die TEK mindestens eines anderen Produkts reduziert wird. Ein solches Ergebnis, das nicht eindeutig bestimmt ist, kann jedoch nicht als Entscheidungsgrundlage in Unternehmen dienen.

Vorabdiskussion zum gewählten Ansatz. Es muss somit ein Ziel für den zu entwickelnden Algorithmus formuliert werden, welches die maximalen Produktstückzahlen $\mathbf{y}_{\max}$ eindeutig bestimmt. Der gewählte Ansatz besteht darin, die Stückzahlen y_b aller Produkte b beginnend mit dem Nullvektor schrittweise zu maximieren. Solange die Stückzahlen y_b zweier Produkte b gesteigert werden können, muss deren Verhältnis jenem der vorläufigen geplanten

Produktstückzahlen $y_{\text{plan},b}$ entsprechen. Das heißt, wenn die TEK für ein Produkt erhöht werden kann, indem man die TEK für ein anderes Produkt reduziert, muss das vorgegebene Verhältnis gelten. Sobald es aber nicht mehr möglich ist, die Stückzahl y_b eines Produkts b zu steigern, wird dessen Stückzahl unverändert beibehalten, um im weiteren Verlauf die Stückzahlen y_b der verbleibenden Produkte weiter zu maximieren.

Alternativ existieren andere Ansätze, die allerdings aus folgenden Gründen nicht weiterverfolgt werden. Wie in Abschnitt 3.4 zur Bewertung von Software für Supply Chain Management diskutiert, ist es grundsätzlich möglich, die optimalen Lösungen durch Schranken zu begrenzen. Jedoch müsste der Anwender durch die Analyse oder die wiederholte Optimierung des Modells geeignete Schranken suchen, weshalb der Gedanke verworfen wird. Einen Spezialfall bildet der Ansatz, zwischen den Stückzahlen von jeweils zwei Produkten ein fixes Verhältnis festzulegen. Dieser Fall ist verwandt mit dem letztlich gewählten Ansatz, jedoch mit dem Unterschied, dass nur ein einziger Iterationsschritt ausgeführt wird. Problematisch wäre hierbei, dass keine der optimalen Lösungen den festgelegten Verhältnissen zwischen den Produktstückzahlen entsprechen muss. Das heißt, würde man solche Verhältnisse voraussetzen, dann träte häufig der Fall ein, dass ein Produkt mit geringerer maximaler Stückzahl alle anderen Produktstückzahlen beschränkt. Dies wäre auch dann der Fall, wenn die verfügbaren Ressourcen noch nicht vollständig ausgelastet sind.

Ein weiterer Ansatz bestünde in der Priorisierung aller Produkte, um die verfügbaren Ressourcen in einer bestimmten Reihenfolge für die Produkte zu nutzen. In Abschnitt 3.4 wurde erläutert, dass ein solcher Ansatz mit Hilfe einer Kostenfunktion in der Praxis nur begrenzt umsetzbar ist. Algorithmisch ließe sich eine entsprechende Priorisierung zwar auf einfache Weise realisieren, indem ein iteratives Vorgehen angewandt wird. Allerdings würden in diesem Fall all diejenigen Ressourcen, die Prozessschritten verschiedener Produkte zugeordnet sind, bevorzugt nur durch bestimmte Produkte ausgelastet. Die Folge wäre ein deutliches Ungleichgewicht zugunsten höher priorisierter Produkte, sodass im Extremfall die Stückzahl nur eines einzigen Produkts maximiert würde. Währenddessen würden die Stückzahlen aller anderen Produkte nicht den Wert null übersteigen.

Auch der Ansatz, auf der Ebene einer Ressource jeweils einen fixen Zeitanteil für jedes Produkt festzulegen, in welchem die Ressource für das Produkt genutzt werden soll, muss verworfen werden. Analog zu den vorigen Gedanken müsste der Anwender durch Analysen oder wiederholte Optimierung geeignete Zeitanteile suchen. Der Grund ist, dass Ressourcen nicht vollständig ausgelastet würden, wenn die Stückzahl eines Produkts nicht mehr gesteigert werden kann, bevor der fixe Zeitanteil erreicht ist. Als die Entwicklung der Software AURELIE bei der Bosch Rexroth AG begann, wurde dieser Ansatz zunächst von den zukünftigen Anwendern vorgeschlagen. Das Verhältnis der Zeitanteile für die Produkte sollte jeweils den geplanten Produktstückzahlen entsprechen.

Der eingangs beschriebene Ansatz stellt dagegen sicher, dass alle Ressourcen bestmöglich genutzt werden. Die Nutzung der Ressourcen wird optimiert, sodass das Verhältnis der zu maximierenden Produktstückzahlen y_b dem Verhältnis der vorläufig festgelegten, geplanten Produktstückzahlen $y_{\text{plan},b}$ entspricht. Dieser Gedanke wurde in der Software AURELIE umgesetzt und wird seither bei der Bosch Rexroth AG angewandt.

Ziel und Vorbedingungen. Zur Bestimmung der maximalen Produktstückzahlen $\mathbf{y}_{\max}$ findet ein iteratives Vorgehen Anwendung. Laut der obigen Beschreibung werden in jedem Iterationsschritt die Stückzahlen y_b all derjenigen Produkte b erhöht, die noch nicht maximal sind. Die Stückzahl eines Produkts wird als maximal bezeichnet, wenn diese unter den vorgegebenen Bedingungen nicht weiter erhöht werden kann, ohne die Stückzahlen anderer Produkte zu reduzieren. Sobald eine Produktstückzahl y_b entsprechend dieser Definition ihr Maximum erreicht, wird ihr Wert unverändert beibehalten. Die Stückzahlen y_b aller anderen Produkte werden weiter gesteigert, wobei deren Verhältnis den vorläufigen geplanten Produktstückzahlen $y_{\text{plan},b}$ entsprechen muss. Durch die Definition dieser zusätzlichen Bedingung ist sichergestellt, dass die maximalen Produktstückzahlen $\mathbf{y}_{\max}$ eindeutig bestimmt sind. Der Begriff der Vorläufigkeit bezieht sich darauf, dass die Stückzahlen für Produktion und Zukauf im Planungsprozess erst nach Bestimmung der Kapazitäten festgelegt werden. Die Iterationsschritte werden wiederholt, solange die Stückzahl y_b mindestens eines Produkts b nicht ihren maximalen Wert erreicht hat.

Entsprechend basiert die Formulierung des Ziels auf einer Iterationsvorschrift, welche die Schritte definiert, nach denen die Stückzahlen y_b aller Produkte b zu maximieren sind. Gegeben sei ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$, welches aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ hervorgegangen ist. Zu jedem Zeitpunkt vor, während und nach der Iteration muss zwischen Produkten b unterschieden werden, deren Stückzahl y_b maximal bzw. nicht maximal ist. Bevor die Iteration beginnt, gelten die Stückzahlen y_b aller Produkte b als noch nicht maximal. Um maximale Stückzahlen y_b von solchen zu trennen, die nicht maximal sind, wird die Menge $B_{\max}$ definiert (der eingeklammerte Exponent k bezeichnet im Folgenden den Iterationsschritt):

$$k = 0\colon \qquad B_{\max}^{(0)} = \varnothing\,, \tag{5.4a}$$

$$k \in \mathbb{N}\colon \qquad B_{\max}^{(k)} = \left\{b\colon\, b \in \{1,\ldots,n\} \wedge y_b^{(k)} \text{ ist maximal}\right\}. \tag{5.4b}$$

Die Menge $B_{\max}$ bezeichnet die Teilmenge der Indizes aller Produkte b, deren Stückzahlen y_b unter allen gültigen Lösungen $\mathbf{x}$ maximal sind. Eine Lösung $\mathbf{x}$ ist genau dann gültig, wenn diese alle Nebenbedingungen in Bezug auf die Produktstruktur, den Flusserhalt und die Auslastung erfüllt. Dies bezieht sich auf die Bedingungen, die in Abschnitt 4.4 im Zuge der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ in ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ eingeführt wurden. Gegeben sei jeweils die Lösung $\mathbf{x}$ aus dem aktuellen Iterationsschritt, von welcher vorausgesetzt wird, dass diese gültig ist. Der nachfolgende Ausdruck beschreibt auf dieser Basis die Suche einer zu vergleichenden Lösung $\mathbf{x}'$, nach welcher die Produktstückzahl y_b ihren größten Wert annimmt. Diese folgt aus der Matrixmultiplikation $\mathbf{P}[b,*]\,\mathbf{x}'$, wie in Abschnitt 4.4 definiert. Formal ist die Stückzahl y_b eines Produkts b im aktuellen Iterationsschritt gemäß folgender Äquivalenz maximal:

$$\begin{aligned} k \in \mathbb{N}_0\colon \qquad & y_b^{(k)} \text{ ist maximal} \Leftrightarrow \\ & y_b^{(k)} = \max\left\{\mathbf{P}[b,*]\,\mathbf{x}'\colon\, \mathbf{x}' \geqslant \mathbf{0} \wedge \right. \\ & \qquad \left. \mathbf{P}\mathbf{x}' \geqslant \mathbf{P}\mathbf{x}^{(k)} \wedge (\mathbf{I}-\mathbf{S})\,\mathbf{P}\mathbf{x}' \geqslant \mathbf{0} \wedge \mathbf{F}\mathbf{x}' = \mathbf{0} \wedge \mathbf{L}\mathbf{x}' \leqslant \mathbf{1}\right\}. \end{aligned} \tag{5.5}$$

Weiter oben wurde erläutert, dass in jedem Iterationsschritt die Produktstückzahlen y_b, die noch nicht maximal sind, maximiert werden müssen. Das Verhältnis der Produktstückzahlen y_b wird durch die geplanten Produktstückzahlen $y_{\text{plan},b}$ paarweise festgelegt. Da sich die Verhältnisse nicht widersprechen, ist dies äquivalent zur Maximierung einer beliebigen Produktstückzahl y_b, wobei folgende Bedingungen gelten: Die Stückzahl y_b des Produkts b darf gemäß Gleichung (5.5) noch nicht maximal sein und muss im Verhältnis zum geplanten Wert $y_{\text{plan},b}$ paarweise allen anderen Produkten entsprechen. Diesem Gedanken folgend wird in jedem Iterationsschritt die Stückzahl y_p eines sogenannten Referenzprodukts mit einem Index p maximiert. Das Referenzprodukt p bezeichnet ein beliebig ausgewähltes Produkt b, dessen Stückzahl y_b nach dem vorigen Iterationsschritt noch nicht ihren maximalen Wert erreicht hatte. Für alle Produkte b, deren Stückzahl y_b bereits maximal war, wird diese im aktuellen und den danach folgenden Iterationsschritten unverändert beibehalten. Unter Nutzung der Ausdrücke in Gleichung (5.4a, 5.4b) und (5.5) gilt für die Lösung $\mathbf{x}$ im aktuellen Iterationsschritt der folgende Ausdruck:

$$k = 0\colon \qquad \mathbf{x}^{(0)} = \mathbf{0}\,, \tag{5.6a}$$

$$\begin{aligned} k \in \mathbb{N}\colon \qquad \mathbf{x}^{(k)} = \arg\max \big\{ & \mathbf{P}\,[p,*]\,\mathbf{x}\colon \mathbf{x} \geqslant \mathbf{0} \wedge \forall b \in \{1,\ldots,n\} \setminus \{p\} \\ & b \notin B_{\max}^{(k-1)} \rightarrow (y_{\text{plan},p}\,\mathbf{P}\,[b,*] - y_{\text{plan},b}\,\mathbf{P}\,[p,*])\,\mathbf{x} = 0 \wedge \\ & b \in B_{\max}^{(k-1)} \rightarrow \mathbf{P}\,[b,*]\,\mathbf{x} = \mathbf{P}\,[b,*]\,\mathbf{x}^{(k-1)} \wedge \\ & (\mathbf{I} - \mathbf{S})\,\mathbf{P}\mathbf{x} \geqslant \mathbf{0} \wedge \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \mathbf{L}\mathbf{x} \leqslant \mathbf{1} \big\}\,. \end{aligned} \tag{5.6b}$$

Von der Lösung $\mathbf{x}$ aus Gleichung (5.6b) werden die Stückzahlen y_b aller Produkte b abgeleitet, die in diesem Iterationsschritt maximiert wurden. Analog zu Gleichung (5.5) folgen die Produktstückzahlen y_b aus der Matrixmultiplikation $\mathbf{P}\,[b,*]\,\mathbf{x}$, entsprechend der allgemeinen Definition in Abschnitt 4.4. Im Anschluss muss erneut geprüft werden, ob die Produktstückzahlen y_b, die zuvor noch nicht maximal waren, nun ihren maximalen Wert erreicht haben. Trifft dies nicht auf alle Produkte b zu, wird die Iteration fortgesetzt.

Zusammenfassend ist festzustellen, dass Gleichung (5.6a, 5.6b) die iterative Maximierung der Stückzahl y_p eines jeweils ausgewählten Referenzprodukts p beschreibt. Dabei werden alle Produktstückzahlen y_b, die noch nicht maximal sind, durch ein fixes Verhältnis an die Stückzahl y_p gebunden und alle maximalen Produktstückzahlen y_b beibehalten. Die Lösung ist durch die Stückzahl y_b mindestens eines Produkts b beschränkt, welche zuvor noch nicht maximal war, nun aber ihren maximalen Wert erreicht. Zur Bestimmung dieser Produkte wird jeweils deren Stückzahl y_b maximiert, wobei gemäß Gleichung (5.5) gefordert wird, dass die Stückzahlen aller anderen Produkte nicht reduziert werden. Genau dann, wenn die Stückzahl y_b eines Produkts b im Zuge dieser Maximierung nicht zu steigern ist, beschränkt diese die Stückzahl y_p des Referenzprodukts p. Es ist nicht möglich, dass eine Stückzahl y_b als Folge des fixen Verhältnisses die Stückzahl y_p des Referenzprodukts p beschränkt, aber gemäß Gleichung (5.5) nicht maximal ist. Dies ist durch die nachfolgenden Vorbedingungen sichergestellt, welche die Vektoren und Matrizen in den Nebenbedingungen betreffen. Hervorzuheben ist hierbei, dass nach jedem Iterationsschritt die Stückzahl y_b mindestens eines weiteren Produkts b ihren maximalen Wert annimmt.

Zur Ausführung des Algorithmus müssen vier Vorbedingungen erfüllt sein. Erst dann ist gewährleistet, dass eine gültige Lösung existiert, die Iteration wie beschrieben verläuft und die gefundene optimale Lösung sinnvoll interpretiert werden kann. Die Vorbedingungen und deren Begründungen lauten wie folgt:

V1 *Das mathematische Modell* $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ *muss als Ergebnis aus der Transformation eines validierten grafischen Modells* $\{G_{\text{WS},b}\}$ *folgen.* Dies betrifft die Zeilen, Spalten und die Komponenten der Matrizen, die in ihrer Gesamtheit das mathematische Modell repräsentieren. Andernfalls wären die Zielfunktionen und Nebenbedingungen nicht immer mathematisch definiert, oder es wäre nicht möglich, die Lösung sinnvoll zu interpretieren. Hierbei werden die Vorbedingungen und die Korrektheit von Algorithmus 4.2 und 4.4 zur Validierung bzw. Transformation vorausgesetzt.

V2 *Die Stückzahlen* y_b *aller Produkte* b *müssen durch die Bedingungen in Gleichung* (5.5) *und* (5.6b) *nach oben beschränkt sein.* Dies ist genau dann der Fall, wenn die Prozessstückzahlen x_a, welche den Produktstückzahlen y_b jeweils zugeordnet sind, durch die Auslastung von Ressourcen begrenzt werden. Das heißt für das zugrundeliegende grafische Modell $\{G_{\text{WS},b}\}$, dass in den Wertstromgraphen $G_{\text{WS},b}$ der Produkte b jeweils mindestens eine Ressource verknüpft sein muss. Nur wenn diese Vorbedingung gilt, existiert eine optimale Lösung für Gleichung (5.6b).

V3 *Die geplanten Stückzahlen* $y_{\text{plan},b}$ *aller Produkte* b *müssen, auch wenn diese als vorläufig gelten, größer als null sein.* Durch die Bedingung in Gleichung (5.6b) wird gefordert, dass das Verhältnis der Produktstückzahlen y_b mit jenem der zugehörigen geplanten Werte $y_{\text{plan},b}$ übereinstimmt. Wäre nun die geplante Stückzahl $y_{\text{plan},b}$ mindestens eines Produkts b gleich null, entspräche die einzig gültige Lösung der Produktstückzahlen y_b dem Nullvektor. Obwohl die Stückzahl y_b keines Produkts b gemäß Gleichung (5.5) ihren maximalen Wert erreichen würde, wäre keine andere Lösung gültig.

V4 *Die geplanten Produktstückzahlen* $\mathbf{y}_{\text{plan}}$ *müssen die Bedingung in Bezug auf die Produktstruktur erfüllen, wie durch Gleichung* (4.2) *in Abschnitt 4.4 definiert.* Das heißt, die geplanten Stückzahlen $y_{\text{plan},b}$ der untergeordneten Produkte b müssen mindestens dem Sekundärbedarf entsprechen. Sonst könnten für übergeordnete Produkte b keine Stückzahlen y_b größer als null erzielt werden. Das fixe Verhältnis, welches durch Gleichung (5.6b) vorgegeben ist, würde bewirken, dass die Stückzahlen y_b untergeordneter Produkte b den Sekundärbedarf nicht erreichen.

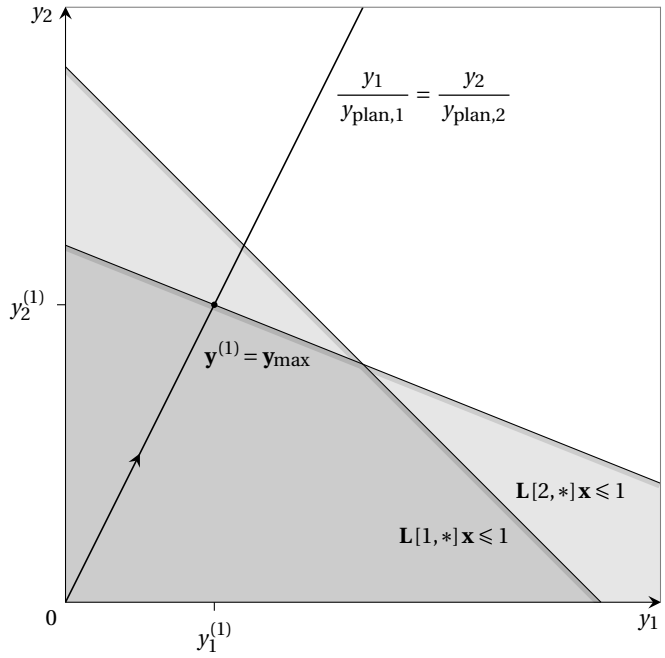
Grundidee zum Ablauf. Der Ablauf zur Maximierung der technischen Kapazitäten $\mathbf{y}_{\text{max}}$ wird durch die iterative Formulierung des Ziels vorweggenommen. Im verbleibenden Teil des Abschnitts werden die Schritte, die zur Erreichung des Ziels ausgeführt werden müssen, durch Algorithmus 5.1 veranschaulicht. Bei diesem handelt es sich um eine Kurzfassung, die zum besseren Verständnis in natürlicher Sprache erstellt wurde. Die zugehörige, detaillierte Langfassung ist im Anhang als Algorithmus A.2.2 angefügt. Analog zu den Algorithmen im vorherigen Kapitel ist die Langfassung in formalem, kommentiertem Pseudocode wiedergegeben. Zusätzlich sind im Anhang an gleicher Stelle alle verwendeten Objekte und Variablen sowie Funktionen und Prozeduren aufgeführt.

Abbildung 5.1 stellt den Ablauf von Algorithmus 5.1 an einem einfachen Beispiel grafisch dar. Die zwei Produktstückzahlen $y_{1/2}$ spannen einen Raum auf, in welchem die maximalen Produktstückzahlen $y_{\max,1/2}$ gesucht werden. Entsprechend den Ressourcen besitzt die Auslastungsmatrix $\mathbf{L}$ zwei Zeilen, wobei die Auslastung der Ressourcen komponentenweise dem Ergebnis der Matrixmultiplikation $\mathbf{Lx}$ entspricht. Die Beschränkung der Auslastung wird dadurch ausgedrückt, dass die Komponenten des Vektors $\mathbf{Lx}$ nicht größer als eins sein dürfen. Diese Bedingungen werden im allgemeinen, mehrdimensionalen Fall durch Hyperebenen und im vorliegenden, zweidimensionalen Fall durch Geraden repräsentiert. Die Hyperebenen bzw. Geraden teilen den Raum der Produktstückzahlen $y_{1/2}$ jeweils in einen abgeschlossenen, gültigen Halbraum und einen offenen, ungültigen Halbraum. Resultierend aus der Menge aller gültigen Lösungen $\mathbf{x}$ bildet die konvexe Schnittmenge der gültigen Halbräume die Menge erreichbarer Produktstückzahlen $y_{1/2}$. Im allgemeinen Fall wird diese Schnittmenge durch ein Polytop und im gegebenen, zweidimensionalen Fall durch ein Polygon repräsentiert. Die optimalen Lösungen, welche die technischen Kapazitäten $y_{\max,1/2}$ repräsentieren, liegen auf den Seitenflächen des Polytops bzw. Polygons.

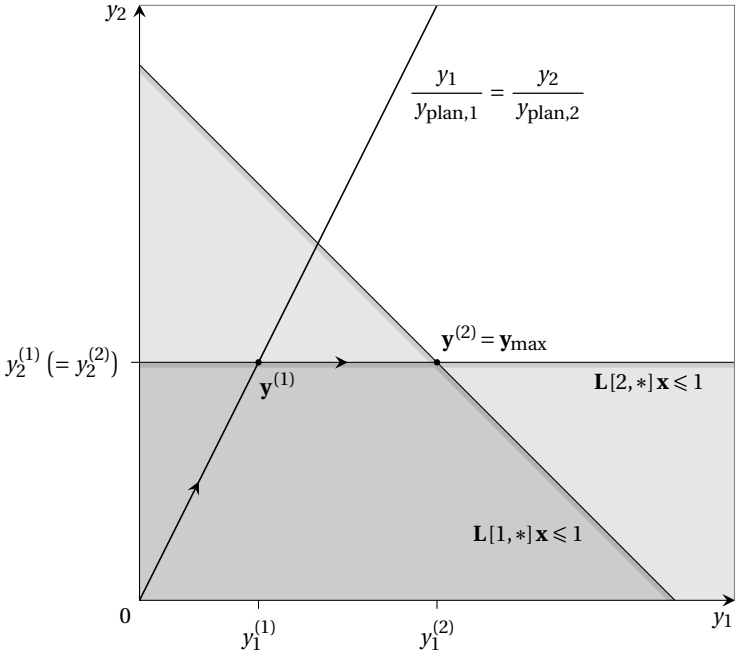
Das Verhältnis der geplanten Produktstückzahlen $y_{\text{plan},1/2}$ ist als Gerade durch den Koordinatenursprung dargestellt. Ausgehend von der Startlösung $\mathbf{x}^{(0)}$, welche dem Nullvektor entspricht und folglich den Koordinatenursprung bezeichnet, werden beide Produktstückzahlen $y_{1/2}$ maximiert. Die gefundene Lösung $\mathbf{y}^{(1)}$ beschreibt den Schnittpunkt der Geraden $y_1/y_{\text{plan},1} = y_2/y_{\text{plan},2}$ mit der Geraden $\mathbf{L}[2,*]\,\mathbf{x} = \mathbf{1}$, entsprechend der größten zulässigen Auslastung der zweiten Ressource. Der weitere Ablauf ist davon abhängig, in welchem Iterationsschritt die Produktstückzahlen $y_{1/2}$ ihr Maximum erreichen. In dem dargestellten Beispiel werden zwei Fälle unterschieden, je nachdem, ob die Produktstückzahl $y_1^{(1)}$ nach dem ersten Iterationsschritt bereits maximal ist.

Im Fall (a) begrenzt die Auslastung der zweiten Ressource, entsprechend der zweiten Zeile der Auslastungsmatrix $\mathbf{L}$, beide Produktstückzahlen $y_{1/2}$. Folglich sind die technischen Kapazitäten $y_{\max,1/2}$ gleich den Produktstückzahlen $y_{1/2}^{(1)}$ nach dem ersten Iterationsschritt. Demgegenüber begrenzt im Fall (b) die Auslastung der zweiten Ressource zwar die Produktstückzahl y_2, nicht aber die Produktstückzahl y_1. In dem Fall ist die Produktstückzahl $y_2^{(1)}$ nach dem ersten Iterationsschritt maximal, worauf diese beibehalten wird, wohingegen die Produktstückzahl $y_1^{(1)}$ weiter maximiert wird. Als Konsequenz entspricht danach das Verhältnis der Produktstückzahlen $y_{1/2}$ nicht mehr den geplanten Produktstückzahlen $y_{\text{plan},1/2}$. Im zweiten Iterationsschritt werden mit den Produktstückzahlen $y_{1/2}^{(2)}$ die gesuchten maximalen Produktstückzahlen $y_{\max,1/2}$ gefunden. Wie die grafische Darstellung zeigt, bezeichnet die Lösung $\mathbf{y}^{(2)}$ den Schnittpunkt der Geraden $\mathbf{L}[1,*]\,\mathbf{x} = \mathbf{1}$ und $\mathbf{L}[2,*]\,\mathbf{x} = \mathbf{1}$, jeweils entsprechend der größten zulässigen Auslastung der beiden Ressourcen.

Datenstrukturen. Neben den Skalaren, Vektoren und Matrizen beschränken sich die Datenstrukturen auf die Information, welche Produktstückzahlen im aktuellen Iterationsschritt maximal sind. Die Indizes all derjenigen Produkte b, deren Stückzahl y_b maximal ist, müssen der Menge $B_{\max}$ zugeordnet werden, wie in Gleichung (5.4a, 5.4b) angegeben. Hierzu wird das eindimensionale, zweiwertige Feld *istMax* eingeführt, dessen Elemente $istMax[b]$ genau dann wahr sind, wenn die Indizes b in der Menge $B_{\max}$ enthalten sind.



(a) Produktstückzahl y_1 ist nach einem Iterationsschritt maximal



(b) Produktstückzahl y_1 ist nach einem Iterationsschritt noch nicht maximal

Abbildung 5.1: Visualisierung der Suche maximaler Kapazitäten $\mathbf{y}_{\max}$ (Beispiel). Die Produktstückzahlen $y_{1/2}$ werden schrittweise maximiert, solange diese laut Definition noch nicht ihren maximalen Wert erreicht haben. Bis dieser Punkt erreicht ist, muss das Verhältnis der Produktstückzahlen $y_{1/2}$ den geplanten Werten $y_{\text{plan},1/2}$ entsprechen, welches als Gerade durch den Koordinatenursprung dargestellt ist. Die Lösungen werden durch die Auslastung der Ressourcen gemäß der Matrix $\mathbf{L}$ begrenzt, im gegebenen, zweidimensionalen Fall jeweils repräsentiert durch eine Gerade.

Eingabe: Vektor (vorläufiger) geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$, Prozessmatrix $\mathbf{P}$, Produktstrukturmatrix $\mathbf{S}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$
Ausgabe: Vektor maximaler Produktstückzahlen $\mathbf{y}_{\max}$ unter den vorgegebenen Bedingungen

```
01: Funktion MaxKapazitäten(y_plan, P, S, F, L)
02:     initialisiere Lösung x als Nullvektor
03:     für alle Produkte b tue initialisiere Produktstückzahl y_b als nicht maximal
...     lege erstes Produkt als Referenzprodukt p fest
06:     wiederhole
07:         speichere vorherige Lösung x als Lösung x'
08:         bestimme eine Lösung x, welche die Produktstückzahl y_p wie folgt maximiert:
                x ← arg max{P[p,*]x : x ≥ 0 ∧ ∀b ∈ {1,...,n} \ {p}
                        Produktstückzahl y_b ist nicht maximal → (y_plan,p P[b,*] − y_plan,b P[p,*])x = 0 ∧
                        Produktstückzahl y_b ist maximal → P[b,*]x = P[b,*]x' ∧
                    (I − S)Px ≥ 0 ∧ Fx = 0 ∧ Lx ≤ 1}
09:         setze Referenzprodukt p zurück
10:         für alle Produkte b, deren Stückzahl y_b nicht maximal ist tue
...             wenn IstMaxKapazität(...) gilt dann
13:               markiere Produktstückzahl y_b als maximal
14:             sonst wenn noch kein Referenzprodukt p bestimmt wurde dann
15:               lege Produkt b als neues Referenzprodukt p fest
16:     solange Referenzprodukt p bestimmt ist
...     liefere technische Kapazitäten y_max gemäß Lösung x zurück

19: Funktion IstMaxKapazität(x, b, P, S, F, L)
20:     speichere aktuellen Wert der Produktstückzahl y_b
21:     initialisiere Lösung x' mit aktueller Lösung x
22:     maximiere Produktstückzahl y'_b gemäß folgendem Ausdruck:
            y'_b ← max{P[b,*]x' : x' ≥ 0 ∧ Px' ≥ Px ∧
                    P[b,*]x' < y_b + ε ∧ (I − S)Px' ≥ 0 ∧ Fx' = 0 ∧ Lx' ≤ 1}
23:     wenn Wert y'_b nicht größer als Wert y_b ist dann
24:       liefere wahr zurück
25:     liefere falsch zurück
```

Algorithmus 5.1: Maximierung der technischen Kapazitäten $\mathbf{y}_{\max}$ (Kurzfassung).

5.2.2 Beschreibung des Algorithmus

Im folgenden Teil des Abschnitts wird der Algorithmus 5.1 zur Maximierung der Kapazitäten $\mathbf{y}_{\max}$ beschrieben, welcher das weiter oben formulierte Ziel umsetzt (Langfassung siehe Algorithmus A.2.2). Gegeben seien die vorläufigen geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und die Matrizen des mathematischen Modells ($\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}}$).

Funktion MaxKapazitäten(...)

In Zeile 01 ist die Hauptfunktion deklariert, welche dazu dient, die maximalen Produktstückzahlen $\mathbf{y}_{\max}$ entsprechend obiger Definition zu ermitteln. Der Ablauf der Funktion basiert auf einer Schleife, deren wiederholter Durchlauf die Iterationsvorschrift aus Gleichung (5.6a, 5.6b) umsetzt. Vor Betreten der Schleife wird der Nullvektor als Startlösung $\mathbf{x}$

der Prozessstückzahlen x_a übernommen, und die Stückzahlen y_b aller Produkte b werden als nicht maximal initialisiert. Gemäß dem Ausdruck in Gleichung (5.6b) muss unter allen Produkten b, deren Stückzahl y_b noch nicht maximal ist, ein beliebiges Referenzprodukt p ausgewählt werden. Da diese Bedingung vor Betreten der Schleife auf alle Produkte zutrifft, wird das erste Produkt als Referenzprodukt p festgelegt.

Ab Zeile 06 werden in einer Schleife alle noch nicht maximalen Produktstückzahlen y_b iterativ maximiert. In jedem Iterationsschritt wird die bisherige Lösung $\mathbf{x}$ der Prozessstückzahlen x_a aus dem vorhergehenden Iterationsschritt als Lösung $\mathbf{x}'$ gespeichert. Im Anschluss daran wird eine neue Lösung $\mathbf{x}$ bestimmt, welche die Stückzahl y_p des Referenzprodukts p maximiert, wobei die zuvor gefundene Lösung als Startlösung dient. Dabei existieren die folgenden zwei Nebenbedingungen in Bezug auf die Stückzahlen y_b aller Produkte b außer dem Referenzprodukt p: Für alle nicht maximalen Produktstückzahlen y_b gilt, dass diese im Verhältnis zum jeweils geplanten Wert $y_{\text{plan},b}$ der Stückzahl y_p des Referenzprodukts p entsprechen müssen. Für alle maximalen Produktstückzahlen y_b gilt dagegen, dass deren Werte unverändert beibehalten werden und folglich mit der vorherigen Lösung $\mathbf{x}'$ übereinstimmen müssen. Daneben müssen weitere, allgemeine Nebenbedingungen erfüllt sein, die in Abschnitt 4.4 eingeführt wurden. Dies betrifft die Bedingung in Bezug auf die Produktstruktur gemäß Gleichung (4.2), nach welcher gefordert wird, dass die Produktstückzahlen y_b mindestens dem Sekundärbedarf entsprechen. Wie durch Gleichung (4.13) angegeben, muss an Vereinigungen von Kanten und an alternativen Flusspunkten in den Wertstromgraphen der Flusserhalt gelten. Des Weiteren darf die Auslastung der Ressourcen gemäß Gleichung (4.14) einen Wert von eins nicht überschreiten.

Ab Zeile 09 wird die zuletzt gefundene Lösung $\mathbf{x}$ ausgewertet. Zunächst wird das Referenzprodukt p zurückgesetzt, um im nächsten Schritt ein neues Referenzprodukt festzulegen. Dann werden alle Produktstückzahlen y_b, die zuvor noch nicht maximal waren, dahingehend geprüft, ob diese nun ihren maximalen Wert erreicht haben. Hierzu wird eine Hilfsfunktion aufgerufen, die weiter unten erläutert wird und welcher die gefundene Lösung $\mathbf{x}$ übergeben wird. Das erste Produkt b, dessen Stückzahl y_b in dem Zuge noch nicht maximal ist, wird als Referenzprodukt p übernommen. Nach jedem Iterationsschritt muss mindestens eine Produktstückzahl y_b, die noch nicht maximal war, ihren maximalen Wert erreichen, wie oben erläutert wurde. Abschließend wird die Schleifenbedingung geprüft, nach welcher im aktuellen Durchlauf der Schleife ein Referenzprodukt p bestimmt worden sein muss. Das ist genau dann der Fall, wenn die Stückzahl y_b mindestens eines Produkts b noch nicht maximal ist. Nach Verlassen der Schleife werden die maximalen Produktstückzahlen $\mathbf{y}_{\max}$ auf Grundlage der ermittelten Lösung $\mathbf{x}$ bestimmt, wie in Gleichung (4.1) angegeben. Zuletzt wird der Vektor $\mathbf{y}_{\max}$ als Ergebnis der Funktion zurückgegeben.

Funktion IstMaxKapazität(...)

Ab Zeile 19 folgt die Hilfsfunktion, welche der Feststellung dient, ob eine Produktstückzahl y_b im aktuellen Iterationsschritt ihren maximalen Wert erreicht hat. Die Hilfsfunktion setzt die Prüfung der Bedingung um, welche durch die oben stehende Gleichung (5.5) definiert ist. Zum Vergleich wird der aktuelle Wert der Produktstückzahl y_b gespeichert und die zuvor ermittelte Lösung $\mathbf{x}$ als Startlösung $\mathbf{x}'$ übernommen. Im Anschluss daran wird ausgehend von der Lösung $\mathbf{x}'$ die Produktstückzahl y'_b maximiert, wobei folgende Nebenbedingun-

gen erfüllt sein müssen: Die im vorhergehenden Iterationsschritt ermittelten Stückzahlen y_b aller Produkte b müssen mindestens beibehalten werden und dürfen entsprechend nicht reduziert werden. Weiterhin soll die Produktstückzahl y'_b gegenüber dem gespeicherten Wert y_b höchstens um einen festgelegten Betrag ε erhöht werden. Der Zweck dieser Nebenbedingung besteht darin, den Algorithmus zu beschleunigen, da allein entscheidend ist, ob die Produktstückzahl y'_b gegenüber dem aktuellen Wert y_b erhöht werden kann. Darüber hinaus gelten für die gesuchte Lösung $\mathbf{x}'$ die Bedingungen in Bezug auf die Produktstruktur, den Flusserhalt und die Auslastung. Sofern die ermittelte Produktstückzahl y'_b nicht größer als der aktuelle Wert y_b ist, muss die Produktstückzahl y_b unter den gegebenen Bedingungen maximal sein. In diesem Fall wird ein positives Ergebnis zurückgegeben, sonst wird die Hilfsfunktion mit einem negativen Ergebnis verlassen.

5.2.3 Beweis der Korrektheit und Zeitkomplexität

Analog zu den Algorithmen im vorigen Kapitel soll der Algorithmus 5.1 zur Maximierung der technischen Kapazitäten $\mathbf{y}_{\text{max}}$ formal untersucht werden. Zunächst muss dessen Korrektheit bewiesen werden, da für diesen keine Grundlage in der Literatur existiert, wie sie im Fall der vorhergehenden Algorithmen bestand. Anschließend folgt wieder die Untersuchung der Komplexität und insbesondere der Komplexität in Bezug auf die Laufzeit. Die Komplexität in Bezug auf den Speicherplatzbedarf ist abhängig von den verwendeten Datenstrukturen. Im gegebenen Fall betrifft dies neben den Skalaren, Vektoren und Matrizen das eindimensionale Feld *istMax*, dessen Länge der Zahl n der gegebenen Produkte entspricht.

Beweis der Korrektheit. Um die Korrektheit des Algorithmus nachzuweisen, ist entscheidend, welche Nachbedingung in Bezug auf dessen Ausgabe formuliert wird. Allgemein bezieht sich das Ziel des Algorithmus darauf, die verfügbaren Ressourcen bestmöglich auszulasten, um die Produktstückzahlen y_b zu maximieren. Das heißt, die Stückzahl y_b jedes Produkts b muss den maximalen Wert annehmen, der erreichbar ist, ohne die Stückzahlen der jeweils anderen Produkte zu reduzieren. Dass im Zuge der Maximierung zwischen jedem Paar von Produktstückzahlen y_b jeweils ein fixes Verhältnis erfüllt werden muss, ist nicht Teil der Definition des Ziels. Stattdessen wird dies bei der Formulierung des Ziels durch eine Nebenbedingung berücksichtigt. Vor diesem Hintergrund wird als Nachbedingung die folgende zu beweisende Aussage formuliert:

Theorem 5.1: Korrektheit von Algorithmus 5.1. Gegeben seien ein Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ mit den genannten Matrizen. In Bezug auf das Modell wird vorausgesetzt, dass dieses aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\text{WS},b}$ folgt. Des Weiteren seien alle Vorbedingungen des Algorithmus erfüllt. Unter diesen Voraussetzungen ist der Algorithmus zur Maximierung der technischen Kapazitäten $\mathbf{y}_{\text{max}}$ partiell und total korrekt. Korrekt heißt hierbei, dass nach Beendigung des Algorithmus die Stückzahlen y_b aller Produkte b maximal sind, wie in Gleichung (5.5) definiert.

Beweis. Aus der oben stehenden Formulierung des Theorems geht bereits hervor, dass nacheinander die partielle und die totale Korrektheit nachgewiesen werden müssen. Wie in der

Kurzeinführung erläutert, wird zum Beweis der partiellen Korrektheit zunächst eine geeignete Schleifeninvariante gewählt. Die Schleifeninvariante muss einem logischen Ausdruck entsprechen, der zu jedem Zeitpunkt im Ablauf des Algorithmus wahr ist, sofern die Eingabe des Algorithmus den Vorbedingungen genügt. Weiterhin muss das Ziel des Algorithmus genau dann erreicht sein, wenn die Schleifeninvariante und die negierte Schleifenbedingung erfüllt sind. Die Schleifenbedingung besagt im Fall des gegebenen Algorithmus, dass im aktuellen Durchlauf der Schleife ein Referenzprodukt p bestimmt wurde. Ein entsprechendes Referenzprodukt p wird genau dann bestimmt, wenn der Index mindestens eines Produkts b noch nicht in der Menge $B_{\max}$ enthalten ist. Mit Bezug auf Gleichung (5.5) wird als Schleifeninvariante folgender logischer Ausdruck definiert:

$$\textit{Schleifeninvariante:} \qquad \forall b \in B_{\max}^{(k)} \;\; y_b^{(k)} \text{ ist maximal.} \tag{5.7}$$

Oben stehende Gleichung (5.7) beschreibt den Teil der Definition gemäß Gleichung (5.4b), welcher sich auf die Elemente der Menge $B_{\max}$ bezieht. Für den Beweis der Korrektheit ist dabei jedoch nicht vorauszusetzen, dass die Menge $B_{\max}$ wie in Gleichung (5.4b) angegeben definiert ist. Nicht notwendigerweise mit der Definition übereinstimmend enthält die Menge $B_{\max}$ genau diejenigen Indizes b von Produkten, die dieser im Ablauf des Algorithmus zugeordnet werden. Ein Index b ist jeweils genau dann in der Menge $B_{\max}$ enthalten, wenn das Element $istMax[b]$ für das Produkt b zu diesem Zeitpunkt wahr ist.

Wenn die Schleifenbedingung nicht erfüllt ist, existiert kein Produkt b, dessen Index nicht in der Menge $B_{\max}$ enthalten ist. Umgekehrt formuliert heißt das, wenn die Schleifenbedingung verletzt ist, dann sind die Indizes aller Produkte b in der Menge $B_{\max}$ enthalten. Unter dieser Bedingung und der Gültigkeit der Schleifeninvariante sind folglich die Stückzahlen y_b aller Produkte b maximal, und das Ziel des Algorithmus ist erreicht. Somit muss als Nächstes die Gültigkeit der Schleifeninvariante vor Beginn der Schleife, nach jedem Durchlauf und daraus folgend nach deren Beendigung gezeigt werden.

Da die Menge $B_{\max}^{(0)}$ als leere Menge initialisiert wird, ist der Ausdruck in Gleichung (5.7) vor Beginn der Schleife wahr (Induktionsanfang). Im Weiteren wird vorausgesetzt, dass die definierte Schleifeninvariante für den jeweils vorhergehenden Schleifendurchlauf galt und der oben stehende Ausdruck folglich für $B_{\max}^{(k-1)}$ erfüllt ist. Für alle Produkte b, deren Indizes in der Menge $B_{\max}^{(k-1)}$ enthalten waren, gilt, dass die Indizes auch nach dem aktuellen Schleifendurchlauf in der Menge $B_{\max}^{(k)}$ enthalten sind. Unter der genannten Voraussetzung waren die Stückzahlen $y_b^{(k-1)}$ der Produkte b in der Menge $B_{\max}^{(k-1)}$ nach dem vorherigen Schleifendurchlauf maximal. Da all diese Stückzahlen y_b beibehalten und andere Produktstückzahlen nicht reduziert werden, müssen auch die Produktstückzahlen $y_b^{(k)}$ maximal sein. Damit ist die Schleifeninvariante für alle Produkte b erfüllt, deren Indizes nach dem vorhergehenden Durchlauf der Schleife in der Menge $B_{\max}^{(k-1)}$ enthalten waren.

Es verbleibt, die Schleifeninvariante für alle Produkte b zu beweisen, deren Indizes nicht in der Menge $B_{\max}^{(k-1)}$ enthalten waren. In jedem Schleifendurchlauf wird die Stückzahl y_p des Referenzprodukts p maximiert, eines beliebig ausgewählten Produkts, dessen Stückzahl nicht maximal ist. Dabei muss die Nebenbedingung erfüllt sein, dass alle anderen Produktstückzahlen y_b, die nicht maximal sind, zur Stückzahl y_p des Referenzprodukts in einem fixen Verhältnis stehen. Danach wird für all diese Produkte b eine Hilfsfunktion aufgerufen, wel-

che die Bedingung in Gleichung (5.5) umsetzt und prüft, ob die Stückzahl y_b nun maximal ist. Wenn die Prüfung bestätigt, dass die genannte Bedingung erfüllt ist, wird der Index b des jeweils betreffenden Produkts in die Menge $B_{\max}^{(k)}$ aufgenommen. Unter der oben stehenden Voraussetzung, dass die Schleifeninvariante nach dem vorhergehenden Schleifendurchlauf galt, ist diese folglich auch für alle anderen Produkte b erfüllt. Das Zwischenergebnis lautet entsprechend zusammengefasst, dass die definierte Schleifeninvariante nach jedem Durchlauf der Schleife wahr ist (Induktionsschritt).

Die zuvor getroffenen Aussagen gelten gleichermaßen für den letzten Schleifendurchlauf, nach welchem die Schleife verlassen wird. Folglich ist die partielle Korrektheit des Algorithmus bewiesen, und es gilt nun, die totale Korrektheit des Algorithmus nachzuweisen. Zu diesem Zweck wird, wie in der Kurzeinführung des Kapitels erläutert, eine Schleifenvariante festgelegt, die folgenden Anforderungen genügen muss: Die Schleifenvariante muss einen Ausdruck bezeichnen, der zu Beginn einer natürlichen Zahl größer als null entspricht und mit jedem Durchlauf der Schleife reduziert wird. Zudem muss die Schleifenbedingung immer dann verletzt sein, wenn die Schleifenvariante einen Wert von null annimmt. Zur Umsetzung dieser Anforderungen wird folgender Ausdruck definiert:

$$\textit{Schleifenvariante:} \qquad n - |B_{\max}^{(k)}|. \tag{5.8}$$

Auf dieser Grundlage muss nun gezeigt werden, dass die Schleifenvariante die genannten Anforderungen erfüllt. Vor Beginn der Schleife wird die Menge $B_{\max}^{(0)}$ als leere Menge initialisiert, sodass der Ausdruck in Gleichung (5.8) der endlichen Zahl n der Produkte entspricht. Nach jedem Schleifendurchlauf erreicht die Stückzahl y_b mindestens eines Produkts b, die zuvor nicht maximal war, ihren maximalen Wert, worauf die Menge $B_{\max}^{(k)}$ erweitert wird. Dies folgt aus den Vorbedingungen des Algorithmus und den Nebenbedingungen der Zielfunktionen, wie oben an der entsprechenden Stelle festgestellt. Insbesondere wird durch die Vorbedingungen gefordert, dass die Stückzahlen y_b aller Produkte b nach oben beschränkt sein müssen. Werden alle Produktstückzahlen y_b, die zuvor nicht maximal waren, in einem fixen Verhältnis maximiert, beschränkt mindestens eine Produktstückzahl y_b die gefundene Lösung. Als Konsequenz wird der Index mindestens eines Produkts b, welcher zuvor noch kein Element der Menge $B_{\max}^{(k-1)}$ war, der Menge $B_{\max}^{(k)}$ hinzugefügt. Mit jedem Durchlauf der Schleife wächst entsprechend die Kardinalität der Menge $B_{\max}^{(k)}$, und die Schleifenvariante wird jeweils mindestens um den Wert eins reduziert.

Nach jedem Durchlauf wird die Schleifenbedingung geprüft, welche der Feststellung dient, ob ein Referenzprodukt p bestimmt werden konnte. Ein Referenzprodukt p kann genau dann nicht bestimmt werden, wenn die Menge $B_{\max}^{(k)}$ die Indizes aller Produkte b enthält und die Schleifenvariante gleich null ist. Das heißt, erreicht die Schleifenvariante den Wert null, dann ist die Schleifenbedingung verletzt, und die Schleife wird verlassen. Damit ist bewiesen, dass der Algorithmus partiell und total korrekt ist. □

Beweis der Zeitkomplexität. Der Fokus liegt im Folgenden auf der Untersuchung der Zeitkomplexität von Algorithmus 5.1. Gesucht ist entsprechend eine obere Schranke für die asymptotisch wachsende Laufzeit, deren Definition von den jeweiligen Eingaben des Al-

gorithmus abhängig ist. Der betrachtete Algorithmus basiert auf einer Schleife, welche die iterative Formulierung des Ziels umsetzt, sodass bezüglich der Laufzeit gilt:

Theorem 5.2: Zeitkomplexität von Algorithmus 5.1. Gegeben seien ein Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ mit den genannten Matrizen. In Bezug auf das Modell wird vorausgesetzt, dass dieses aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\text{WS},b}$ folgt. Der Algorithmus zur Maximierung der technischen Kapazitäten $\mathbf{y}_{\text{max}}$ besitzt in der angegebenen Implementierung eine Laufzeit von $\mathrm{O}(n^2)$.

Beweis. Da vorausgesetzt wird, dass sämtliche Vorbedingungen des Algorithmus gelten, müssen die Stückzahlen y_b aller Produkte b nach oben beschränkt sein. Bezug nehmend auf die Definition in Gleichung (5.5) ist bekannt, dass vor Beginn der Schleife die Stückzahl y_b keines Produkts b ihren maximalen Wert annimmt. In jedem Schleifendurchlauf wird die Stückzahl y_p des jeweils gewählten Referenzprodukts p maximiert, wobei die in Gleichung (5.6b) angegebenen Bedingungen gelten. Daraufhin ist die Stückzahl y_b mindestens eines weiteren Produkts b maximal, wie im Zuge des Beweises der Korrektheit festgestellt wurde. Die Schleife wird nur dann wiederholt, wenn mindestens ein Produkt b verbleibt, dessen Stückzahl y_b noch nicht maximal ist. Als Folge ist die Zahl der Schleifendurchläufe nicht größer als die endliche Zahl n der gegebenen Produkte b.

Im Hinblick auf die Laufzeiten wird vorausgesetzt, dass Aufrufe zur Maximierung von Zielfunktionen in einer Laufzeit von $\mathrm{O}(1)$ erfolgen. Weiterhin wird angenommen, dass mathematische Operationen mit Vektoren und Matrizen in einer Laufzeit von $\mathrm{O}(1)$ ausgeführt werden. Unter diesen Voraussetzungen können für die Zeilen im Rumpf der Funktionen folgende Laufzeiten angegeben werden (ohne aufgerufene Hilfsfunktionen):

02:	$\mathrm{O}(1)$	10 . . . 15:	$\mathrm{O}(n^2)$
03 . . . 04:	$\mathrm{O}(n)$	16:	$\mathrm{O}(n)$
05:	$\mathrm{O}(1)$	17 . . . 18:	$\mathrm{O}(1)$
06 . . . 09:	$\mathrm{O}(n)$	20 . . . 25:	$\mathrm{O}(n^2)$

Die Gesamtlaufzeit resultiert aus der Addition der Laufzeiten für alle Zeilen, wobei kleinere Summanden und konstante Faktoren vernachlässigt werden. Demzufolge ist die gesuchte obere Schranke gleich dem zu zeigenden Ausdruck $\mathrm{O}(n^2)$. □

5.3 Minimierung der Investitionen

Nachdem die maximalen Kapazitäten ermittelt wurden, folgt im Planungsprozess die Entscheidung über die Stückzahlen für Eigenproduktion und Zukauf. Hierzu wird der erwartete Bedarf unter Berücksichtigung von strategischen Faktoren wie z. B. einem festgelegten Eigenproduktionsanteil mit den technischen Kapazitäten $\mathbf{y}_{\text{max}}$ verglichen. Das Ergebnis sind die im Unternehmen herzustellenden, geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$, wobei anzumerken ist, dass diese die technischen Kapazitäten $\mathbf{y}_{\text{max}}$ überschreiten können. Das heißt, durch die Herstellung der geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ ist es möglich, dass die Auslastung

einer oder mehrerer MAEs den Wert eins übersteigt. Von dieser Überlastung werden, wie in Abschnitt 2.3 beschrieben, die notwendigen Investitionen abgeleitet, indem die überlasteten MAEs vervielfacht werden. Gesucht ist die minimale Überlastung, die erforderlich ist, um die Produkte in den geplanten Stückzahlen $\mathbf{y}_{\text{plan}}$ herzustellen.

Um das Ziel und den Ablauf des Algorithmus formal eindeutig zu beschreiben, werden einige zusätzliche Symbole eingeführt. Diese werden neben den bekannten Symbolen aus den vorigen Abschnitten und dem entsprechenden Verzeichnis verwendet.

i, i'	natürliche Zahl, nicht näher bezeichnete Zeile des Vektors $\mathbf{Lx}$, $i, i' \in \mathbb{N}$
$I_{\min}$	Menge der Zeilen i, an denen der Wert des Vektors $\mathbf{Lx}$ minimal ist, $I_{\min} \subseteq \{1, \ldots, \mathrm{N}(\mathbf{L})\}$
j	natürliche Zahl, bezeichnet im aktuellen Iterationsschritt diejenige oder ggf. eine derjenigen Zeilen des Vektors $\mathbf{Lx}$ mit dem größten nicht minimalen Wert größer als eins, $j \in \mathbb{N}$
j'	s. o., nächste Zeile des Vektors $\mathbf{Lx}$ mit dem größten nicht minimalen Wert größer als eins, $j' \in \mathbb{N}$
$\mathrm{N}(\mathbf{X})$	größte Zeile einer Matrix $\mathbf{X}$ ungleich dem Nullvektor, $\mathrm{N}(\mathbf{X}) \in \mathbb{N}_0$
$\mathbf{x}'$	Vektor $[x'_1 \ \ldots \ x'_m]^\top$ der Prozessstückzahlen x'_a, zu vergleichende Lösung, $\mathbf{x}' \in \mathbb{R}^m_{\geqslant 0}$
x'_a	zu vergleichende Teillösung mit Index $a \in \{1, \ldots, m\}$, $x'_a \in \mathbb{R}_{\geqslant 0}$

Analog zu Algorithmus 5.1 im vorigen Abschnitt folgt zunächst das Ziel, welches durch den Algorithmus erreicht werden soll. Danach wird der Ablauf des Algorithmus beschrieben, um zuletzt dessen Korrektheit und Zeitkomplexität nachzuweisen.

5.3.1 Ziel, Grundidee und Datenstrukturen

Die Formulierung des Ziels knüpft an die Erläuterung in Abschnitt 2.3 und die zusammenfassende Anforderung A11 in Abschnitt 2.6 an. Es besteht analog zum vorigen Abschnitt die Herausforderung, eine geeignete Strategie festzulegen, welche die Auswahl einer nicht dominierten Lösung beschreibt. In diesem Fall dominieren zwei Lösungen im reellwertigen Suchraum einander nicht, wenn zwei Bedingungen gelten: Nach der ersten Lösung ist die Überlastung im Fall mindestens einer Ressource kleiner als die Überlastung derselben Ressource nach der zweiten Lösung. Umgekehrt ist nach der zweiten Lösung die Überlastung im Fall mindestens einer anderen Ressource kleiner als die Überlastung derselben Ressource nach der ersten Lösung. Folglich ist es nicht möglich, eine Lösung der jeweils anderen eindeutig vorzuziehen, und beide Lösungen gelten als optimal. Zwar legt der Anwender fest, welche Ressourcen im Wertstromgraphen jedes Produkts überlastet werden sollen, jedoch werden oft Gruppen von Ressourcen für verschiedene Produkte genutzt. Als Folge besteht in vielen Fällen die Möglichkeit, die Überlastung einer Ressource zu reduzieren, indem im Gegenzug die Überlastung anderer Ressourcen erhöht wird. Wie schon im Zusammenhang mit der TEK festgehalten, ist ein solches, mehrdeutiges Ergebnis nicht als Entscheidungsgrundlage für die Planung geeignet.

Vorabdiskussion zum gewählten Ansatz. Um aus der Menge der optimalen Lösungen eindeutig eine Lösung zu bestimmen, wird folgender Ansatz gewählt: Schrittweise wird die Überlastung all derjenigen Ressourcen minimiert, deren Auslastung unter allen betrachteten Ressourcen am größten ist. Hierbei werden nur diejenigen Ressourcen betrachtet, deren mindestens notwendige Überlastung noch nicht feststeht. Als Folge wird die Überlastung,

die zur Herstellung der geplanten Produktstückzahlen notwendig ist, gleichmäßig verteilt. Das heißt, wenn die Überlastung einer Ressource reduziert werden kann, indem man die Überlastung anderer erhöht, werden die Werte einander angeglichen.

Analog zur Maximierung der Kapazitäten und dem hierzu angegebenen Algorithmus sind grundsätzlich auch andere Ansätze denkbar. Zunächst erscheint es naheliegend, spezifische Kosten für die Überlastung von Ressourcen zu definieren, um die Aufwände für Beschaffung und Betrieb abzubilden. Auf dieser Basis könnten theoretisch die Gesamtkosten zur Herstellung der Produkte minimiert werden, wodurch nur bestimmte Ressourcen überlastet würden. Die praktische Umsetzung wird aus dem gleichen Grund erschwert, der bereits in Abschnitt 2.3 im Zusammenhang mit der Optimierung der Auslastung dargelegt wurde: Zum Zeitpunkt der Planung liegen die notwendigen Informationen über den laufenden Betrieb im Allgemeinen noch nicht in ausreichender Genauigkeit vor. Auszuschließen ist der Ansatz jedoch aus dem Grund, dass eine unendliche Menge optimaler Lösungen existieren kann, deren Gesamtkosten sich nicht unterscheiden.

Alternativ könnte der Anwender eine Priorisierung aller Ressourcen definieren, um festzulegen, welche Ressourcen bevorzugt überlastet werden sollen. Diese Priorisierung könnte als Grundlage dienen, um die Überlastung der Ressourcen in der umgekehrten Reihenfolge zu minimieren. Dadurch würden Ressourcen in einem größeren Maße überlastet, wenn diesen eine höhere Priorität zugeordnet ist. In Abschnitt 3.4 wurde im Zusammenhang mit Software für Supply Chain Management angemerkt, dass sich eine Kostenfunktion nur eingeschränkt eignet, um dies umzusetzen. Mit Hilfe eines iterativ ablaufenden Algorithmus wäre es zwar möglich, eine entsprechende Priorisierung zu realisieren. Jedoch ist analog zum vorigen Abschnitt zu bemerken, dass ein Ungleichgewicht zugunsten höher priorisierter Ressourcen bestünde, sodass im Extremfall nur eine einzige Ressource überlastet würde. Um eine Priorisierung aller Ressourcen abzubilden, müsste zudem das Modell erweitert werden, wodurch für den Anwender der Aufwand zur Modellierung stiege.

Demgegenüber kann der gewählte Ansatz realisiert werden, ohne dass eine Erweiterung des grafischen Modells erforderlich ist. Hierzu wird die Sortierung ausgehender Kanten von alternativen Flusspunkten genutzt, welche der Anwender festlegt, um die zugeordneten Prozessschritte zu priorisieren. Wie in Abschnitt 4.2 beschrieben, dürfen nur solche Prozessschritte, die nach den ersten Ausgängen alternativer Flusspunkte folgen, zu einer Überlastung führen. Die Information, welche Prozessschritte dies betrifft, wird im mathematischen Modell durch die Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ abgebildet.

Ziel und Vorbedingungen. Damit gilt es nun, das Ziel des Algorithmus in formal eindeutiger Notation auszudrücken. Betrachtet werden zunächst alle Lösungen $\mathbf{x}$ der Prozessstückzahlen x_a, welche den geplanten Stückzahlen $y_{\text{plan},b}$ der gegebenen Produkte b entsprechen. Wie in Abschnitt 4.4 definiert wurde, ist die Auslastung der Ressourcen jeweils gleich den Komponenten des Vektors $\mathbf{Lx}$, wobei jeder Zeile eine Ressource zugeordnet ist. Der Vektor $\mathbf{Lx}$ bezeichnet dabei das Ergebnis aus der Multiplikation der Auslastungsmatrix $\mathbf{L}$ und einer Lösung $\mathbf{x}$, welche die vorgegebenen Bedingungen erfüllt. Unter dieser Voraussetzung beschreibt die gesuchte minimale Überlastung die Menge der minimalen oberen Schranken für die Komponenten des Vektors $\mathbf{Lx}$. Genauer sind die minimalen oberen Schranken größer

als oder gleich eins gesucht, für welche unter den vorgegebenen Bedingungen eine Lösung $\mathbf{x}$ existiert. Wenn eine dieser Schranken für eine Komponente des Vektors $\mathbf{Lx}$ größer als eins ist, muss die jeweils zugeordnete Ressource in dem angegebenen Maße überlastet werden. Ist die Schranke dagegen gleich eins, ist keine Überlastung der Ressource notwendig, um die geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ herzustellen.

Um die minimalen oberen Schranken größer als oder gleich eins zu bestimmen, wird gedanklich ein iteratives Vorgehen angewandt. Zu Beginn wird für alle Werte des Vektors $\mathbf{Lx}$, von denen bekannt ist, dass diese nicht größer als eins sein können, eine minimale obere Schranke von eins festgelegt. Für alle anderen Werte ist die minimale obere Schranke vor Beginn der Iteration dagegen noch nicht definiert. Danach wird in jedem Iterationsschritt der größte nicht minimale Wert minimiert, wobei alle anderen Werte, die noch nicht minimal sind, diesen nicht überschreiten dürfen. Abweichend vom mathematischen Verständnis eines Extremwertes wird ein Wert genau dann minimal genannt, wenn seine minimale obere Schranke feststeht. Das ist genau dann der Fall, wenn die obere Schranke des Wertes gleich eins ist oder der Wert gegenüber seiner oberen Schranke nicht reduziert werden kann, ohne andere Werte über deren obere Schranken zu steigern. Nach jedem Iterationsschritt wird für alle Werte, die zuvor noch nicht minimal waren, der größte unter all diesen Werten als obere Schranke übernommen. Die Iteration wird fortgesetzt, solange der Vektor $\mathbf{Lx}$ mindestens einen Wert enthält, welcher nicht minimal und größer als eins ist.

Entsprechend wird das Ziel durch eine Iterationsvorschrift ausgedrückt, welche die Minimierung oberer Schranken für die Komponenten des Vektors $\mathbf{Lx}$ definiert. Wie im vorigen Abschnitt sei ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ gegeben, welches aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ folgt. Vor, während und nach der Iteration müssen die Zeilen i des Vektors $\mathbf{Lx}$ dahingehend unterschieden werden, ob die zugehörigen Werte laut der obigen Beschreibung minimal oder nicht minimal sind. Der Wert an einer Zeile i ist jeweils gleich dem Ergebnis der Matrixmultiplikation $\mathbf{L}[i,*]\mathbf{x}$, wie in Abschnitt 4.4 allgemein definiert. Die Startlösung sei durch eine Lösung $\mathbf{x}^{(0)}$ gegeben, von welcher vorausgesetzt werden kann, dass die Werte des Vektors $\mathbf{Lx}^{(0)}$ größer als oder gleich null sind. Wie weiter unten noch dargelegt wird, darf eine Ressource, entsprechend einer Zeile der Auslastungsmatrix $\mathbf{L}$, nur dann überlastet werden, wenn der Wert des Vektors $\mathbf{Lx}^{(0)}$ an dieser Zeile größer als null ist. Wenn dieser Wert gleich null ist, dann ist dessen minimale obere Schranke gleich eins, und der Wert gilt bereits vor dem ersten Iterationsschritt als minimal. Um die Menge der Zeilen i zu beschränken, wird die Funktion $\mathrm{N}(\mathbf{X})$ eingeführt, welche die größte Zeile einer Matrix $\mathbf{X}$ ungleich dem Nullvektor zurückgibt. Auf dieser Basis wird zur Unterscheidung minimaler und nicht minimaler Werte die Menge $I_{\min}$ definiert (der Iterationsschritt wird wie zuvor durch den Exponenten k ausgedrückt):

$$k = 0\text{:} \qquad I_{\min}^{(0)} = \left\{ i\text{: } i \in \{1,\ldots,\mathrm{N}(\mathbf{L})\} \wedge \mathbf{L}[i,*]\mathbf{x}^{(0)} = \mathbf{0} \right\}, \tag{5.9a}$$

$$k \in \mathbb{N}\text{:} \qquad I_{\min}^{(k)} = \left\{ i\text{: } i \in \{1,\ldots,\mathrm{N}(\mathbf{L})\} \wedge \mathbf{L}[i,*]\mathbf{x}^{(k)} \text{ ist minimal} \right\}. \tag{5.9b}$$

Die Menge $I_{\min}$ repräsentiert die Teilmenge aller Zeilen i des Vektors $\mathbf{Lx}$, deren Werte innerhalb der gültigen Lösungen $\mathbf{x}$ minimal sind. Eine Lösung $\mathbf{x}$ ist genau dann gültig, wenn

diese den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht und die Bedingung in Bezug auf den Flusserhalt erfüllt. Zur Bezeichnung der oberen Schranken, welche für die Komponenten des Vektors $\mathbf{Lx}$ gelten, wird das eindimensionale, reellwertige Feld *minMax* eingeführt. Wie erläutert, wird für jeden Wert des Vektors $\mathbf{Lx}$ an einer Zeile i, der vor Beginn der Iteration minimal ist, eine minimale obere Schranke $minMax[i]$ von eins festgelegt. Solange ein Wert noch nicht minimal ist, wird seiner oberen Schranke $minMax[i]$ nach jedem Iterationsschritt jeweils der größte nicht minimale Wert zugeordnet. Sobald ein Wert aber minimal ist, wird dessen obere Schranke $minMax[i]$ nicht mehr verändert. Da die minimalen oberen Schranken größer als oder gleich eins gesucht sind, gilt ein Wert bereits als minimal, wenn die obere Schranke $minMax[i]$ gleich eins ist. Wenn dagegen der Wert an einer Zeile i kleiner als dessen obere Schranke $minMax[i]$ und letztere größer als eins ist, kann dieser Wert noch nicht als minimal gelten. Entspricht der Wert seiner oberen Schranke und ist diese größer als eins, muss eine Lösung $\mathbf{x}'$ gesucht werden, welche den Wert des Vektors $\mathbf{Lx}'$ an Zeile i weiter reduziert. Formal ausgedrückt gilt (vorausgesetzt, die obere Schranke $minMax[i]$ ist definiert, sonst wird der Wert als nicht minimal aufgefasst):

$$
\begin{aligned}
k \in \mathbb{N}_0\colon \quad & \mathbf{L}[i,*]\,\mathbf{x}^{(k)} \text{ ist minimal} \Leftrightarrow \\
& \left(minMax[i]^{(k)} = 1\right) \vee \left(\mathbf{L}[i,*]\,\mathbf{x}^{(k)} = minMax[i]^{(k)} \wedge\right. \\
& \quad \mathbf{L}[i,*]\,\mathbf{x}^{(k)} = \min\left\{\mathbf{L}[i,*]\,\mathbf{x}'\colon \mathbf{x}' \geqslant \mathbf{0} \wedge\right. \\
& \quad\quad \mathbf{P}\mathbf{x}' = \mathbf{P}\mathbf{x}^{(k)} \wedge \mathbf{F}\mathbf{x}' = \mathbf{0} \wedge \forall i' \in \{1,\ldots,\mathrm{N}(\mathbf{L})\} \setminus \{i\} \\
& \quad\quad\quad \left.\left.\mathbf{L}[i',*]\,\mathbf{x}' \leqslant minMax[i']^{(k)}\right\}\right). \qquad (5.10)
\end{aligned}
$$

Die Startlösung $\mathbf{x}^{(0)}$ muss alle oben genannten Bedingungen erfüllen, wobei eine Bedingung ergänzt wird: Die Auslastung einer Ressource darf jeweils nur dann größer als null sein, wenn diese gemäß der Festlegung des Anwenders bei Bedarf überlastet werden soll. Das heißt, es werden nur solche Prozessschritte betrachtet, welche im grafischen Modell nach den ersten Ausgängen alternativer Flusspunkte folgen. Diese Bedingung ist erfüllt, wenn das Ergebnis der Matrixmultiplikation $\mathbf{L}_{\text{Invest}}\,\mathbf{x}$ dem Nullvektor entspricht, wie in Abschnitt 4.4 definiert. Um unter dieser Voraussetzung die Startlösung $\mathbf{x}^{(0)}$ zu ermitteln, wird analog zu Algorithmus A.2.1 die Summe aller Produktstückzahlen y_b maximiert. Da die Stückzahl y_b eines Produkts b jeweils durch den geplanten Wert $y_{\text{plan},b}$ beschränkt ist, entspricht dies in Bezug auf das Ergebnis der Maximierung aller Produktstückzahlen y_b. In jedem Iterationsschritt wird die Zeile j bestimmt, welche den größten nicht minimalen Wert des Vektors $\mathbf{Lx}$ größer als eins bezeichnet. Existieren mehrere Zeilen, deren Werte mit dem größten übereinstimmen, ist nicht entscheidend, welche von diesen als Zeile j festgelegt wird. Der Wert an Zeile j wird minimiert, wobei Werte an Zeilen i, die nicht minimal sind, diesen nicht überschreiten dürfen. Für alle minimalen Werte gilt jeweils die obere Schranke $minMax[i]$. Unter Verwendung der Ausdrücke in Gleichung (5.9a, 5.9b) und (5.10) folgt daraus für die Lösung $\mathbf{x}$ im aktuellen Iterationsschritt:

$$
\begin{aligned}
k = 0\colon \quad & \mathbf{x}^{(0)} = \arg\max\left\{\mathbf{1}^{\top}\mathbf{P}\mathbf{x}\colon \mathbf{x} \geqslant \mathbf{0} \wedge\right. \\
& \quad \left.\mathbf{P}\mathbf{x} \leqslant \mathbf{y}_{\text{plan}} \wedge \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \mathbf{L}_{\text{Invest}}\,\mathbf{x} = \mathbf{0}\right\}, \qquad (5.11a)
\end{aligned}
$$

$$k \in \mathbb{N}: \qquad \mathbf{x}^{(k)} = \arg\min \big\{ \mathbf{L}[j,*]\,\mathbf{x} : \mathbf{x} \geqslant \mathbf{0} \wedge \mathbf{P}\mathbf{x} = \mathbf{y}_{\text{plan}} \wedge \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \forall i \in \{1,\ldots,\mathrm{N}(\mathbf{L})\} \setminus \{j\} \; i \notin I_{\min}^{(k-1)} \rightarrow (\mathbf{L}[i,*] - \mathbf{L}[j,*])\,\mathbf{x} \leqslant 0 \wedge i \in I_{\min}^{(k-1)} \rightarrow \mathbf{L}[i,*]\,\mathbf{x} \leqslant minMax[i]^{(k-1)} \big\}. \tag{5.11b}$$

Der aus Gleichung (5.11b) resultierende Wert des Vektors $\mathbf{L}\mathbf{x}$ an Zeile j wird verwendet, um nach jedem Iterationsschritt die oberen Schranken $minMax[i]$ für all diejenigen Werte festzulegen, die zuvor nicht minimal waren. Im Anschluss muss geprüft werden, ob diese Werte nun als minimal anzusehen sind. Solange mindestens ein Wert größer als eins existiert, der noch nicht minimal ist, muss die Iteration fortgesetzt werden.

Zusammenfassend beschreibt Gleichung (5.11a, 5.11b) die iterative Minimierung des Wertes an Zeile j, welcher jeweils den größten unter allen nicht minimalen Werten repräsentiert. Dabei werden alle anderen Werte an Zeilen i nach oben beschränkt, fallweise durch den Wert an Zeile j oder ihre minimale obere Schranke $minMax[i]$. Die resultierende Lösung $\mathbf{x}$ ist durch mindestens einen Wert an einer Zeile i beschränkt, der gemäß Gleichung (5.10) zuvor noch nicht minimal war, nun aber als minimal gilt. Um die betreffenden Zeilen zu bestimmen, wird jeder nicht minimale Wert nach der Festlegung seiner oberen Schranke geprüft, wie in Gleichung (5.10) angegeben ist. Wenn der Wert an einer Zeile i seiner oberen Schranke $minMax[i]$ entspricht und nicht zu reduzieren ist, beschränkt dieser den größten nicht minimalen Wert an Zeile j. Damit ist nach jedem Iterationsschritt mindestens ein weiterer Wert des Vektors $\mathbf{L}\mathbf{x}$ minimal.

Zuletzt werden die Vorbedingungen angegeben, die erfüllt sein müssen, damit nach den Vorgaben des Anwenders eine sinnvoll zu interpretierende, optimale Lösung gefunden wird. Diese entsprechen inhaltlich den ersten drei Vorbedingungen von Algorithmus 5.1, werden aber wiederholt, da deren Begründung z. T. abweicht:

V1 *Das mathematische Modell* $(\mathbf{P},\mathbf{S},\mathbf{F},\mathbf{L},\mathbf{L}_{\text{Invest}})$ *muss wie zuvor aus der Transformation eines validierten grafischen Modells* $\{G_{\text{WS},b}\}$ *folgen.* Diese Vorbedingung ist notwendig, um sicherzustellen, dass die genannten Matrizen Werte innerhalb ihrer Definitionsbereiche annehmen. Damit kann vorausgesetzt werden, dass die Zielfunktionen und Nebenbedingungen mathematisch definiert sind und die Lösung sinnvoll zu interpretieren ist. In Bezug auf die Validierung und Transformation werden die Vorbedingungen und die Korrektheit von Algorithmus 4.2 bzw. 4.4 angenommen.

V2 *Die Stückzahlen* y_b *aller Produkte* b *müssen durch die Bedingungen in Gleichung* (5.5) *und* (5.6b) *nach oben beschränkt sein.* Mit Hilfe dieser Vorbedingung ist gewährleistet, dass in jedem Wertstromgraphen $G_{\text{WS},b}$ eines Produkts b jeweils mindestens ein Prozessschritt definiert ist. Im Fall von Algorithmus 5.2 müsste nur ein einziger Prozessschritt mit einer Ressource vorausgesetzt werden, welcher die Produktstückzahlen y_b nicht nach oben beschränken muss. Wichtig ist, dass die Auslastungsmatrix $\mathbf{L}$ als Folge mindestens eine Zeile ungleich dem Nullvektor enthält.

Bemerkung: Die Vorbedingung wird in der Fassung formuliert, die in Abschnitt 5.2 angegeben ist, da die drei Planungsziele nicht unabhängig voneinander betrachtet werden. Bevor die Funktion der Software AURELIE zur mathematischen Optimierung aufgerufen wird, muss sichergestellt sein, dass die Vorbedingungen aller Algorithmen erfüllt sind. Es ist daher nicht notwendig, die angegebene Formulierung zu konkretisieren, da eine allgemeinere, aus einem anderen Grund zu erfüllende Vorbedingung existiert.

V3 *Die geplanten Stückzahlen $y_{\text{plan},b}$ aller Produkte b, welche zu diesem Zeitpunkt nicht mehr als vorläufig gelten, müssen größer als null sein.* Die Startlösung $\mathbf{x}^{(0)}$ wird auf Basis der geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ bestimmt, wobei nur Ressourcen ausgelastet werden, deren Überlastung zulässig ist. Danach wird für alle Werte des Vektors $\mathbf{L}\mathbf{x}^{(0)}$, die gleich null sind und nach Gleichung (5.9a) minimal sein müssen, eine minimale obere Schranke von eins festgelegt. Wäre diese Vorbedingung nicht erfüllt, würden ggf. Ressourcen, deren Überlastung zulässig sein soll, nicht überlastet.

Grundidee zum Ablauf. Der Ablauf zur Minimierung der Überlastung $\mathbf{L}\mathbf{x}$ ist analog zum vorigen Algorithmus 5.1 durch die iterative Formulierung des Ziels bestimmt. Im Weiteren werden die einzelnen Schritte anhand von Algorithmus 5.2 beschrieben, welcher eine Kurzfassung darstellt und in natürlicher Sprache vorliegt. Die zugehörige Langfassung ist wie im Fall der vorangegangenen Abschnitte in Gestalt des Algorithmus A.2.3 im Anhang zu finden. Diese gibt den Ablauf detailliert wieder und ist in formalem, kommentiertem Pseudocode verfasst. An der genannten Stelle im Anhang sind außerdem alle genutzten Objekte und Variablen sowie Funktionen und Prozeduren aufgeführt.

Im Gegensatz zum vorigen Algorithmus 5.1 ist eine beispielhafte grafische Darstellung nicht auf vergleichbar einfache Weise möglich. Da der Ablauf der Algorithmen aber Parallelen aufweist, werden zur Förderung des Verständnisses stattdessen wesentliche Übereinstimmungen und Unterschiede herausgestellt. Durch diese vergleichende Betrachtung soll das Muster verdeutlicht werden, welches den beiden Algorithmen zugrundeliegt. Zunächst ist festzuhalten, dass in jedem Iterationsschritt von Algorithmus 5.1 eine Zielfunktion maximiert wird, wohingegen der Algorithmus 5.2 die Minimierung einer Zielfunktion beschreibt. Folglich unterscheiden sich die Startlösungen: Im ersten Fall ist dies der Nullvektor, im zweiten Fall der Vektor größtmöglicher nicht minimaler Werte.

Im Verlauf von Algorithmus 5.1 werden schrittweise all diejenigen Produktstückzahlen y_b gesteigert, die noch nicht maximal sind. Analog hierzu werden gemäß Algorithmus 5.2 die oberen Schranken all derjenigen Werte des Vektors $\mathbf{L}\mathbf{x}$ reduziert, die noch nicht minimal sind. Die Definition minimaler Werte bezieht sich dabei jedoch nicht auf die Werte selbst, sondern darauf, dass deren minimale obere Schranken noch nicht feststehen. Unter allen Produkten b, deren Stückzahl y_b noch nicht maximal ist, wird im Fall von Algorithmus 5.1 ein beliebiges Referenzprodukt p ausgewählt und dessen Stückzahl y_p maximiert. Das Gegenstück bildet im Fall von Algorithmus 5.2 der größte nicht minimale Wert des Vektors $\mathbf{L}\mathbf{x}$, welcher sich an Zeile j befindet und minimiert wird.

Hierbei gilt, dass bereits maximale Produktstückzahlen y_b bzw. minimale obere Schranken des Vektors $\mathbf{L}\mathbf{x}$ beibehalten bzw. berücksichtigt werden müssen. Für alle verbleibenden noch nicht maximalen bzw. minimalen Werte werden jeweils Bedingungen in Bezug auf den zu

maximierenden bzw. zu minimierenden Wert formuliert. Im Fall von Algorithmus 5.1 ist dies ein fixes Verhältnis, wohingegen im Fall von Algorithmus 5.2 der zu minimierende Wert nicht überschritten werden darf. Als Folge erreicht nach jedem Iterationsschritt von Algorithmus 5.1 die Stückzahl y_b mindestens eines weiteren Produkts b ihren maximalen Wert. Damit vergleichbar ist im Fall von Algorithmus 5.2 nach jedem Iterationsschritt mindestens ein weiterer Wert des Vektors $\mathbf{Lx}$ minimal.

Der Grundgedanke ist folglich übereinstimmend, die Menge gültiger Lösungen $\mathbf{x}$, die zu Beginn jeweils unendlich ist, schrittweise zu beschränken. Im Verlauf der Iteration werden fortlaufend zusätzliche Bedingungen formuliert, welche die Freiheitsgrade reduzieren, die durch die Prozessstückzahlen x_a repräsentiert werden. Die Iteration endet jeweils, sobald die Menge gültiger Lösungen $\mathbf{x}$ in hinreichender Weise begrenzt worden ist. Im Fall von Algorithmus 5.1 ist diese Bedingung genau dann erfüllt, wenn die Stückzahlen aller Produkte b ihre maximalen Werte erreicht haben. Dagegen ist die Iteration im Fall von Algorithmus 5.2 beendet, wenn kein Wert des Vektors $\mathbf{Lx}$ existiert, welcher noch nicht minimal und größer als eins ist. Der Grund für den Vergleich mit dem Wert eins ist, dass die Überlastung betrachtet wird und folglich nur Komponenten größer als eins relevant sind.

Datenstrukturen. Zusätzlich zu den Skalaren, Vektoren und Matrizen, die eingangs genannt wurden, muss die Information abgebildet werden, welche Werte des Vektors $\mathbf{Lx}$ im aktuellen Iterationsschritt minimal sind. Wie in Gleichung (5.9a, 5.9b) festgehalten, müssen die Zeilen i all derjenigen Werte des Vektors $\mathbf{Lx}$, deren minimale obere Schranke zum jeweiligen Zeitpunkt feststeht, der Menge $I_{\min}$ zugeordnet werden. Zur Abbildung der entsprechenden Information wird das eindimensionale, zweiwertige Feld *istMin* definiert, dessen Index i die Zeile des Vektors $\mathbf{Lx}$ bezeichnet. Ein Element $istMin[i]$ des Feldes ist genau dann wahr, wenn die Zeile i des Vektors $\mathbf{Lx}$ in der Menge $I_{\min}$ enthalten ist.

Um festzustellen, ob ein Wert des Vektors $\mathbf{Lx}$ minimal oder noch nicht minimal ist, muss gemäß Gleichung (5.10) dessen obere Schranke betrachtet werden. Zur Verwaltung dieser oberen Schranken, die im aktuellen Iterationsschritt feststehen oder noch vorläufig sind, wird das eindimensionale Feld *minMax* definiert. Die Elemente $minMax[i]$ bezeichnen die oberen Schranken größer als oder gleich eins, welche für die Komponenten des Vektors $\mathbf{Lx}$ an den Zeilen i festgelegt sind. Im Iterationsverlauf sind die oberen Schranken noch vorläufig, da die Werte des Vektors $\mathbf{Lx}$ mit Bezug auf die Elemente $minMax[i]$ nicht minimal sein müssen. Nach dem Durchlauf des Algorithmus erfüllen alle Werte des Vektors $\mathbf{Lx}$ und alle zugehörigen Elemente des Feldes *minMax* die Bedingung in Gleichung (5.10).

Daneben wird die Variable *zeilenzahl* eingeführt, die zwar nicht in der Kurzfassung, aber in der Langfassung im Anhang verwendet wird. Die Variable bezeichnet die größte Zeile der Auslastungsmatrix $\mathbf{L}$ ungleich dem Nullvektor und wird genutzt, um die Beschreibung des Ablaufs zu vereinfachen. Dies betrifft verschiedene Stellen, an denen all diejenigen Zeilen der genannten Matrix betrachtet werden, die nicht dem Nullvektor entsprechen. Da vorausgesetzt wird, dass die Stückzahlen y_b aller Produkte b nach oben beschränkt sind, muss stets mindestens eine solche Zeile existieren.

5.3.2 Beschreibung des Algorithmus

Im Anschluss folgt nun die Beschreibung von Algorithmus 5.2, welcher das definierte Ziel umsetzt, die nötige Überlastung genutzter Ressourcen zu minimieren (Langfassung siehe Algorithmus A.2.3). Die Minimierung der Überlastung bezieht sich, wie zuvor formuliert, auf diejenigen Komponenten des Vektors $\mathbf{Lx}$, die Werte größer als eins annehmen. Gegeben seien die geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$, die an dieser Stelle nicht mehr als vorläufig gelten, und die Matrizen des mathematischen Modells ($\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}}$).

Funktion MinÜberlastung(...)

Ab Zeile 01 ist die Hauptfunktion aufgeführt, welche die minimalen oberen Schranken größer als oder gleich eins für die Komponenten des Vektors $\mathbf{Lx}$ ermittelt. Die Funktion basiert auf einer Schleife, die wiederholt durchlaufen wird, um die Iterationsvorschrift nach Gleichung (5.11a, 5.11b) umzusetzen. Zuallererst muss eine Startlösung $\mathbf{x}$ bestimmt werden, welche die Bedingungen erfüllt, die in Gleichung (5.11a) angegeben sind. Zu diesem Zweck wird eine Hilfsfunktion aufgerufen, die eine Lösung $\mathbf{x}$ zurückgibt, welche den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht. Die Startlösung ist außerdem dadurch gekennzeichnet, dass die gültigen Lösungen $\mathbf{x}$, im Gegensatz zur Maximierung der Kapazitäten im vorigen Abschnitt, nicht durch die Auslastung der Ressourcen beschränkt werden. Jedoch darf nur die Auslastung derjenigen Ressourcen größer als null sein, die laut dem Anwender überlastet werden sollen, wie weiter unten erläutert wird.

Danach werden alle Zeilen der Auslastungsmatrix $\mathbf{L}$ betrachtet, und es wird der Wert des Vektors $\mathbf{Lx}$ an jeder Zeile i ermittelt. Wenn dieser größer als null ist, dann bezeichnet die Zeile i eine Ressource, die gemäß der Festlegung des Anwenders bei Bedarf überlastet werden soll. Da die minimalen oberen Schranken für die Werte an den entsprechenden Zeilen i folglich noch nicht feststehen, werden diese Werte als noch nicht minimal initialisiert. In dem Zuge wird die Zeile j bestimmt, die im Folgenden den größten nicht minimalen Wert des Vektors $\mathbf{Lx}$ bezeichnet, der größer als eins ist. Falls dieser Wert an mehreren Zeilen zu finden ist, wird die erste Zeile übernommen. Wenn der Wert des Vektors $\mathbf{Lx}$ an Zeile i gleich null ist, darf die Ressource nicht überlastet werden. Folglich werden diese Werte vor Beginn der Schleife als minimal initialisiert, und für ihre oberen Schranken $minMax[i]$ wird jeweils ein Wert von eins festgelegt. Da alle anderen Werte zu diesem Zeitpunkt noch nicht minimal sind, wird für diese noch keine obere Schranke $minMax[i]$ definiert.

Ab Zeile 13 folgt die Schleife, in deren Verlauf der jeweils größte nicht minimale Wert des Vektors $\mathbf{Lx}$ iterativ minimiert wird. Zu Beginn wird die Schleifenbedingung geprüft, die genau dann erfüllt ist, wenn die Zeile j bestimmt ist und folglich noch nicht alle minimalen oberen Schranken feststehen. Danach wird eine neue Lösung $\mathbf{x}$ gesucht, welche den Wert an Zeile j minimiert, wobei von der vorigen Lösung ausgegangen wird. In Bezug auf die Werte an allen anderen Zeilen i existieren zwei Nebenbedingungen, welche die Werte nach oben beschränken: Für alle nicht minimalen Werte gilt, dass diese den jeweils zu minimierenden Wert des Vektors $\mathbf{Lx}$, bezeichnet durch die Zeile j, nicht überschreiten dürfen. Für alle minimalen Werte gilt deren minimale obere Schranke $minMax[i]$, die im Fall der betreffenden Werte feststeht. Im Hinblick auf die gesuchte Lösung $\mathbf{x}$ wird gefordert, dass diese wie die vorige den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht. Analog zur Suche der maximalen

Eingabe: Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$, Prozessmatrix $\mathbf{P}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$, Investitionsmatrix $\mathbf{L}_{\text{Invest}}$
Ausgabe: minimale obere Schranken größer als oder gleich eins für die Komponenten des Vektors $\mathbf{Lx}$, für welche unter den vorgegebenen Bedingungen eine Lösung $\mathbf{x}$ existiert

01: **Funktion** MinÜberlastung($\mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}}$)
… initialisiere Lösung $\mathbf{x}$ entsprechend InitMinÜberlastung(…) und Zeile j als unbestimmt
05: **für alle** *Zeilen i der Auslastungsmatrix* $\mathbf{L}$ **tue**
06: **wenn** *Wert des Vektors* $\mathbf{Lx}$ *an Zeile i größer als null ist* **dann**
… initialisiere diesen als nicht minimal und bestimme Zeile j mit größtem Wert größer als eins
10: **sonst** initialisiere Wert des Vektors $\mathbf{Lx}$ an Zeile i als minimal und
… lege eins als Schranke $minMax[i]$ fest
13: **solange** *Zeile j bestimmt ist* **tue**
14: bestimme eine Lösung $\mathbf{x}$, welche den Wert des Vektors $\mathbf{Lx}$ an Zeile j wie folgt minimiert:
$\mathbf{x} \leftarrow \arg\min\{\mathbf{L}[j,*]\mathbf{x} : \mathbf{x} \geqslant \mathbf{0} \wedge \mathbf{Px} = \mathbf{y}_{\text{plan}} \wedge \mathbf{Fx} = \mathbf{0} \wedge$
$\forall i \in \{1, \ldots, \text{Zeilen der Matrix } \mathbf{L}\} \setminus \{j\}$
Wert des Vektors $\mathbf{Lx}$ an Zeile i ist nicht minimal $\rightarrow (\mathbf{L}[i,*] - \mathbf{L}[j,*])\mathbf{x} \leqslant 0 \wedge$
Wert des Vektors $\mathbf{Lx}$ an Zeile i ist minimal $\rightarrow \mathbf{L}[i,*]\mathbf{x} \leqslant minMax[i]\}$
15: initialisiere nächste Zeile j' als unbestimmt
16: **wenn** *Wert des Vektors* $\mathbf{Lx}$ *an Zeile j größer als eins ist* **dann**
17: **für alle** *Zeilen i des Vektors* $\mathbf{Lx}$, *deren Wert nicht minimal ist* **tue**
… lege Wert an Zeile j als Schranke $minMax[i]$ fest
19: **für alle** *Zeilen i des Vektors* $\mathbf{Lx}$, *deren Wert nicht minimal ist* **tue**
… **wenn** *dieser Wert gleich jenem an Zeile j ist und* IstMinÜberlastung(…) *gilt* **dann**
22: markiere Wert des Vektors $\mathbf{Lx}$ an Zeile i als minimal
… **sonst** bestimme nächste Zeile j' mit größtem Wert größer als eins
25: übernimm nächste Zeile j' als Zeile j
26: **für alle** *Zeilen i des Vektors* $\mathbf{Lx}$, *deren Wert nicht minimal ist* **tue**
… lege eins als Schranke $minMax[i]$ fest
28: **liefere** minimale obere Schranken $minMax$ **zurück**

29: **Funktion** InitMinÜberlastung($\mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L}_{\text{Invest}}$)
30: initialisiere Lösung $\mathbf{x}$ als Nullvektor
31: bestimme eine Lösung $\mathbf{x}$, welche den geplanten Produktstückzahlen $y_{\text{plan},b}$ wie folgt entspricht:
$\mathbf{x} \leftarrow \arg\max\{\mathbf{1}^\top \mathbf{Px} : \mathbf{x} \geqslant \mathbf{0} \wedge \mathbf{Px} \leqslant \mathbf{y}_{\text{plan}} \wedge \mathbf{Fx} = \mathbf{0} \wedge \mathbf{L}_{\text{Invest}}\mathbf{x} = \mathbf{0}\}$
32: **liefere** initiale Lösung $\mathbf{x}$ **zurück**

33: **Funktion** IstMinÜberlastung($\mathbf{x}, \mathbf{P}, \mathbf{F}, \mathbf{L}, i, minMax$)
… initialisiere Lösung $\mathbf{x}'$ mit aktueller Lösung $\mathbf{x}$
36: bestimme eine Lösung $\mathbf{x}'$, welche den Wert des Vektors $\mathbf{Lx}'$ an Zeile i wie folgt minimiert:
$\mathbf{x}' \leftarrow \arg\min\{\mathbf{L}[i,*]\mathbf{x}' : \mathbf{x}' \geqslant \mathbf{0} \wedge \mathbf{Px}' = \mathbf{Px} \wedge \mathbf{Fx}' = \mathbf{0} \wedge$
$\mathbf{L}[i,*]\mathbf{x}' \geqslant \mathbf{L}[i,*]\mathbf{x} - \varepsilon \wedge \forall i' \in \{1, \ldots, \text{Zeilen der Matrix } \mathbf{L}\} \setminus \{i\}$
$\mathbf{L}[i',*]\mathbf{x}' \leqslant minMax[i']\}$
37: **wenn** *Wert des Vektors* $\mathbf{Lx}'$ *an Zeile i nicht kleiner als Wert des Vektors* $\mathbf{Lx}$ *ist* **dann**
38: **liefere** `wahr` **zurück**
39: **liefere** `falsch` **zurück**

Algorithmus 5.2: Minimierung der Überlastung $\mathbf{Lx}$ (Komponenten größer als eins) (Kurzfassung).

Kapazitäten muss zudem die Bedingung in Bezug auf den Flusserhalt erfüllt sein, welche durch Gleichung (4.13) im vorigen Kapitel definiert ist.

Ist der zuvor minimierte Wert des Vektors $\mathbf{Lx}$ an Zeile j auch weiterhin größer als eins, wird dieser als obere Schranke $minMax[i]$ für alle nicht minimalen Werte festgelegt. Dann werden alle Werte an Zeilen i, die noch nicht minimal waren, jeweils dahingehend geprüft, ob diese nun gemäß der Definition in Gleichung (5.10) minimal sind. Hierzu werden die betreffenden Werte mit dem minimierten Wert an Zeile j verglichen, welcher nach dem vorherigen Schritt ihrer oberen Schranke $minMax[i]$ entspricht. Stimmen die Werte an Zeile i und j überein, dann wird zur weiteren Prüfung eine Hilfsfunktion aufgerufen, welcher die Lösung $\mathbf{x}$ übergeben wird. Wie oben erläutert, muss nach jedem Iterationsschritt mindestens ein Wert an einer Zeile i, der zuvor noch nicht minimal war, diese Bedingung erfüllen. Daraufhin wird unter allen Werten, die nicht minimal und größer als eins sind, der größte Wert ermittelt und die zugehörige Zeile j' bestimmt. Als Folge ist die Zeile j' nur dann bestimmt, wenn nach dem aktuellen Schleifendurchlauf mindestens ein Wert verbleibt, der nicht minimal und größer als eins ist. Im Anschluss wird die Zeile j' für den nächsten Schleifendurchlauf als Zeile j übernommen und die Schleifenbedingung erneut geprüft. Nach Verlassen der Schleife wird für die oberen Schranken $minMax[i]$ aller verbleibenden Werte, die noch nicht minimal sind, der Wert eins festgelegt. Zuletzt wird das Feld $minMax$ zurückgegeben, welches nun die minimalen oberen Schranken größer als oder gleich eins für alle Komponenten des Vektors $\mathbf{Lx}$ enthält.

Funktion InitMinÜberlastung(...)

In Zeile 29 folgt die zuvor angesprochene Hilfsfunktion zur Ermittlung einer Startlösung, welche den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht. Ausgehend vom Nullvektor wird analog zu Algorithmus A.2.1, der eingangs erläutert wurde und im Anhang aufgeführt ist, die Summe der Stückzahlen y_b aller Produkte b maximiert. Da der geplante Wert $y_{\text{plan},b}$ jeweils nicht überschritten werden darf, entspricht dies einer Maximierung der Stückzahl y_b jedes einzelnen Produkts b bis zu dem jeweiligen Wert. Von der Lösung $\mathbf{x}$ wird gefordert, dass diese die Bedingung in Bezug auf den Flusserhalt erfüllt. Wie in Algorithmus A.2.1 wird keine Bedingung in Bezug auf die Auslastung der Ressourcen formuliert. Jedoch dürfen nur bestimmte Prozessstückzahlen x_a einen Wert größer als null annehmen, wie durch Gleichung (4.15) im vorigen Kapitel definiert. Genauer formuliert gilt die Bedingung, dass das Ergebnis der Matrixmultiplikation $\mathbf{L}_{\text{Invest}}\mathbf{x}$ aus der Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ und der Lösung $\mathbf{x}$ gleich null sein muss. Dadurch dürfen nur diejenigen Prozessschritte zu einer Auslastung von Ressourcen führen, die im grafischen Modell nach den ersten ausgehenden Kanten alternativer Flusspunkte folgen. Das heißt, es werden nur solche Ressourcen genutzt, die gemäß der Festlegung des Anwenders bei Bedarf überlastet werden sollen. Durch diese Bedingung und den Flusserhalt werden die gültigen Lösungen $\mathbf{x}$ soweit beschränkt, dass die Startlösung eindeutig bestimmt ist. Nach Ermittlung der entsprechenden Startlösung $\mathbf{x}$ wird diese als Ergebnis zurückgegeben und die Hilfsfunktion verlassen.

Funktion IstMinÜberlastung(...)

In Zeile 33 ist zuletzt die Hilfsfunktion angegeben, welche der Prüfung dient, ob ein Wert des Vektors $\mathbf{Lx}$ an einer Zeile i minimal ist. Die Hilfsfunktion setzt den letzten Teil der Bedingung in Gleichung (5.10) um und wird folglich nur aufgerufen, wenn der Wert seiner oberen Schranke $minMax[i]$ entspricht. Zunächst wird die zuvor ermittelte Lösung $\mathbf{x}$ als Startlösung $\mathbf{x}'$ übernommen, um einen zu vergleichenden Wert zu suchen. Anschließend wird der Wert des Vektors $\mathbf{Lx}'$ an Zeile i ausgehend von der Lösung $\mathbf{x}'$ minimiert, wobei folgende Nebenbedingungen gelten: Die Stückzahlen y_b aller Produkte b müssen unverändert beibehalten werden, wodurch sichergestellt ist, dass diese auch weiterhin den jeweils geplanten Werten $y_{\text{plan},b}$ entsprechen. Die gesuchte Lösung $\mathbf{x}'$ muss zudem die Bedingung in Bezug auf den Flusserhalt erfüllen, wie schon in den vorigen Funktionen und Hilfsfunktionen. Um den Algorithmus analog zur Maximierung der Kapazitäten zu beschleunigen, wird die Reduktion des Wertes an Zeile i auf einen vorab festgelegten Betrag ε begrenzt. Für alle Zeilen i' des Vektors $\mathbf{Lx}'$ außer der Zeile i gilt, dass die zugehörigen Werte ihre oberen Schranken $minMax[i']$ nicht überschreiten dürfen. Sofern der minimierte Wert des Vektors $\mathbf{Lx}'$ nicht kleiner als der aktuelle Wert ist, muss letzterer gemäß Definition minimal sein. Wenn dies der Fall ist, wird die Hilfsfunktion nach Rückgabe eines positiven Ergebnisses beendet, sonst wird diese mit einem negativen Ergebnis verlassen.

5.3.3 Beweis der Korrektheit und Zeitkomplexität

Analog zu den Betrachtungen im vorigen Abschnitt soll der Algorithmus 5.2 zur Minimierung der Überlastung formal untersucht werden. Da für diesen wie auch für den vorangegangenen Algorithmus 5.1 keine Grundlage in der Literatur existiert, muss dessen Korrektheit nachgewiesen werden. Danach folgt die Untersuchung der Komplexität, wobei sich der Blick wieder auf die Laufzeit richtet. Die Komplexität im Hinblick auf den Speicherplatzbedarf ist abhängig von den Datenstrukturen, die weiter oben beschrieben wurden. Neben den Skalaren, Vektoren und Matrizen sind dies die eindimensionalen Felder *istMin* und *minMax*, deren Länge der Zeilenzahl $\mathrm{N}(\mathbf{L})$ der Auslastungsmatrix $\mathbf{L}$ entspricht.

Beweis der Korrektheit. Für den Beweis der Korrektheit ist wie zuvor entscheidend, welche Nachbedingung in Bezug auf die Ausgabe des Algorithmus definiert wird. Allgemein formuliert lautet das Ziel des Algorithmus, die Überlastung der Ressourcen zur Herstellung der geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ zu minimieren. Das heißt, gesucht ist die minimale Überlastung jeder Ressource, entsprechend den Komponenten des Vektors $\mathbf{Lx}$ größer als eins, die erreichbar ist, ohne die Überlastung anderer Ressourcen zu steigern. Der Ausgleich der Überlastung zwischen Ressourcen ist nicht Teil der Definition des Ziels, sondern wird durch eine Nebenbedingung abgebildet. Auf dieser Grundlage wird als Nachbedingung folgende Aussage formuliert, die es zu beweisen gilt:

Theorem 5.3: Korrektheit von Algorithmus 5.2. Gegeben seien ein Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ mit den genannten Matrizen. In Bezug auf das Modell wird vorausgesetzt, dass dieses aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\text{WS},b}$ folgt.

Des Weiteren seien alle Vorbedingungen des Algorithmus erfüllt. Unter diesen Voraussetzungen ist der Algorithmus zur Minimierung der Überlastung aller gegebenen Ressourcen, repräsentiert durch die Komponenten des Vektors $\mathbf{Lx}$ größer als eins, partiell und total korrekt. Korrekt heißt hierbei, dass nach Beendigung des Algorithmus alle Werte des Vektors $\mathbf{Lx}$ minimal sind, wie in Gleichung (5.10) definiert.

Beweis. Analog zum vorigen Algorithmus wird zunächst die partielle Korrektheit nachgewiesen, wofür eine geeignete Schleifeninvariante gewählt wird. Diese muss einem logischen Ausdruck entsprechen, der zu jedem Zeitpunkt wahr ist, sofern die Eingabe den Vorbedingungen genügt, und folgende Anforderung erfüllt: Das Ziel des Algorithmus muss genau dann erreicht sein, wenn die Schleifeninvariante und die negierte Schleifenbedingung wahr sind. Die Schleifenbedingung besagt, dass vor dem Beginn bzw. nach dem vorherigen Durchlauf der Schleife eine Zeile j bestimmt wurde. Die Bedingung ist genau dann erfüllt, wenn mindestens eine Zeile i existiert, die noch nicht in der Menge $I_{\min}$ enthalten ist und deren zugehöriger Wert größer als eins ist. Bezug nehmend auf die Definition in Gleichung (5.10) wird folgender logischer Ausdruck als Schleifeninvariante formuliert:

$$\textit{Schleifeninvariante:} \qquad \forall\, i \in I_{\min}^{(k)} \;\; \mathbf{L}[i,*]\,\mathbf{x}^{(k)} \text{ ist minimal.} \tag{5.12}$$

Der obige Ausdruck gibt den Teil der Definition in Gleichung (5.9b) wieder, der sich auf die Elemente der Menge $I_{\min}$ bezieht. Analog zum vorigen Abschnitt kann für den zu führenden Beweis allerdings nicht vorausgesetzt werden, dass die Menge $I_{\min}$ wie in Gleichung (5.9b) definiert ist. Vielmehr enthält die Menge $I_{\min}$ genau diejenigen Zeilen i, die dieser im Ablauf des Algorithmus zugeordnet werden. Eine Zeile i ist jeweils genau dann in der Menge $I_{\min}$ enthalten, wenn das Element $istMin[i]$ für die Zeile i wahr ist.

Ist die Schleifenbedingung nicht erfüllt, dann sind alle Zeilen i in der Menge $I_{\min}$ enthalten oder die zugehörigen Werte nicht größer als eins. Wenn dies der Fall ist und daneben die Schleifeninvariante wie in Gleichung (5.12) angegeben gilt, müssen alle Werte des Vektors $\mathbf{Lx}$ minimal oder nicht größer als eins sein. Nach Beendigung der Schleife wird für alle verbleibenden nicht minimalen Werte eine minimale obere Schranke $minMax[i]$ von eins festgelegt. Als Ergebnis sind alle Werte des Vektors $\mathbf{Lx}$ gemäß Gleichung (5.10) minimal, und das Ziel des Algorithmus ist erreicht, wie in oben stehendem Theorem festgehalten. Das heißt, die formulierte Schleifeninvariante genügt der genannten Anforderung und ist zur Führung des Beweises geeignet. Im nächsten Schritt muss gezeigt werden, dass die Schleifeninvariante vor Beginn und nach jedem Schleifendurchlauf gültig ist.

Vor Beginn der Schleife wird gemäß Gleichung (5.11a) eine Startlösung $\mathbf{x}^{(0)}$ ermittelt, nach welcher nur ausgewählte Prozessstückzahlen x_a Werte größer als null annehmen dürfen. Daraufhin werden all diejenigen Zeilen i in die Menge $I_{\min}^{(0)}$ aufgenommen, deren zugehörige Werte des Vektors $\mathbf{Lx}^{(0)}$ gleich null sind. Die minimalen oberen Schranken $minMax[i]$ all dieser Werte werden im Zuge der Initialisierung auf den Wert eins festgelegt. Damit sind zu Beginn alle Zeilen i in der Menge $I_{\min}^{(0)}$ gemäß Gleichung (5.10) minimal, und der Ausdruck in Gleichung (5.12) ist erfüllt (Induktionsanfang).

Im Weiteren wird vorausgesetzt, dass die Schleifeninvariante für den vorherigen Schleifendurchlauf galt und der Ausdruck für $I_{\min}^{(k-1)}$ entsprechend wahr ist. Nach dem aktuellen

Schleifendurchlauf sind alle Zeilen i, die in der Menge $I_{\min}^{(k-1)}$ enthalten waren, auch weiterhin Elemente der Menge $I_{\min}^{(k)}$. Unter der zuvor angegebenen Voraussetzung waren nach dem vorherigen Schleifendurchlauf die Werte an den Zeilen i in der Menge $I_{\min}^{(k-1)}$ minimal. Da die minimalen oberen Schranken $minMax[i]$ dieser Werte nicht verändert werden und der Wert an Zeile j nicht erhöht wird, müssen die Werte an den Zeilen i wie zuvor minimal sein. Folglich gilt die Schleifeninvariante für all diejenigen Zeilen i, die nach dem vorhergehenden Schleifendurchlauf in der Menge $I_{\min}^{(k-1)}$ enthalten waren.

Als Nächstes steht der Beweis aus, dass die Schleifeninvariante für alle Zeilen i gilt, die nicht in der Menge $I_{\min}^{(k-1)}$ enthalten waren. In jedem Schleifendurchlauf wird der größte unter allen nicht minimalen Werten größer als eins, bezeichnet durch die Zeile j, minimiert. Dabei muss die Nebenbedingung erfüllt sein, dass alle anderen Werte, die noch nicht minimal sind, den zu minimierenden Wert nicht überschreiten. Ist der daraus folgende, minimierte Wert größer als eins, wird dieser für alle nicht minimalen Werte als obere Schranke $minMax[i]$ übernommen. Jeder Wert, der zuvor noch nicht minimal war, wird danach dahingehend geprüft, ob dieser nun mit dem minimierten Wert übereinstimmt. Ist dies der Fall, wird eine Hilfsfunktion aufgerufen, welche die in Gleichung (5.10) formulierte Bedingung umsetzt. Wenn die Prüfung zu dem Ergebnis führt, dass der Wert minimal ist, wird die betreffende Zeile i des Wertes der Menge $I_{\min}^{(k)}$ hinzugefügt. Unter der zuvor genannten Voraussetzung, dass die Schleifeninvariante nach dem vorhergehenden Schleifendurchlauf galt, ist diese folglich auch für alle anderen Zeilen i erfüllt. Als Zwischenergebnis ist festzuhalten, dass die Schleifeninvariante nach jedem Durchlauf wahr ist (Induktionsschritt).

Die Schleife wird verlassen, wenn entweder kein Wert verbleibt, der nicht minimal ist, oder der zuletzt minimierte Wert an Zeile j nicht mehr größer als eins ist. In letzterem Fall müssen alle Werte, die zuvor nicht minimal waren, kleiner als oder gleich eins sein, da diese den Wert an Zeile j nicht überschreiten dürfen. Auf dieser Grundlage wird nach Beendigung der Schleife der Wert eins als minimale obere Schranke $minMax[i]$ für alle verbleibenden Werte festgelegt. Damit ist Gleichung (5.10) auch für diejenigen Werte erfüllt, die beim Verlassen der Schleife noch nicht minimal waren.

Fasst man die Erkenntnisse zusammen, ist der Algorithmus partiell korrekt, und es muss im nächsten Schritt die totale Korrektheit nachgewiesen werden. Hierzu wird eine Schleifenvariante festgelegt, die entsprechend der Kurzeinführung zu Beginn des Kapitels folgenden Anforderungen genügen muss: Die Schleifenvariante muss einen Ausdruck bezeichnen, welcher vor Beginn der Schleife einer natürlichen Zahl größer als null entspricht und mit jedem Durchlauf reduziert wird. Sobald die Schleifenvariante einen Wert von null annimmt, muss die Schleifenbedingung verletzt sein. Dabei wird derjenige Teil der Schleifenbedingung betrachtet, der sich darauf bezieht, dass die Werte an allen Zeilen i minimal sind. Zwar wird die Schleife bereits zu einem früheren Zeitpunkt verlassen, wenn der größte nicht minimale Wert, bezeichnet durch die Zeile j, nicht größer als eins ist. Da dies jedoch allein dazu führt, dass die Schleife früher beendet wird, genügt zum Nachweis der totalen Korrektheit der angegebene Teil der Bedingung. Um diese Anforderungen an eine Schleifenvariante umzusetzen, wird folgender Ausdruck definiert:

Schleifenvariante: $$\mathrm{N}(\mathbf{L}) - |I_{\min}^{(k)}|. \tag{5.13}$$

Damit gilt es nun zu zeigen, dass die Schleifenvariante die genannten Anforderungen erfüllt. Nach der Initialisierung ist die Menge $I_{\min}^{(0)}$ eine endliche Teilmenge aller Zeilen i des Vektors $\mathbf{Lx}$ und der Ausdruck in Gleichung (5.13) größer als oder gleich null. Ist der Ausdruck gleich null, dann sind die Werte an allen Zeilen i minimal, und entsprechend wird die Schleife nicht betreten. Die Schleife wird auch dann nicht betreten, wenn zwar Werte noch nicht minimal sind, aber keiner von diesen größer als eins ist. Wie oben angemerkt, genügt es jedoch zum Nachweis der totalen Korrektheit, dass die Schleife nicht betreten bzw. verlassen wird, wenn der Ausdruck gleich null ist. Nach jedem Schleifendurchlauf ist ein Wert an mindestens einer zusätzlichen Zeile i minimal, welche der Menge $I_{\min}^{(k)}$ hinzugefügt wird. Dies folgt aus den Vorbedingungen und den Nebenbedingungen der Zielfunktionen, die im Verlauf des Algorithmus minimiert werden, wie zuvor festgestellt. Wenn der größte aller nicht minimalen Werte minimiert wird, während alle anderen diesen nicht überschreiten dürfen, beschränkt mindestens einer von diesen Werten die Lösung. Als Folge wird mindestens eine Zeile i, die nicht in der Menge $I_{\min}^{(k-1)}$ enthalten war, in die Menge $I_{\min}^{(k)}$ aufgenommen. Entsprechend wächst mit jedem Durchlauf der Schleife die Kardinalität der Menge $I_{\min}^{(k)}$, und die Schleifenvariante wird jeweils mindestens um den Wert eins reduziert.

Vor jedem Durchlauf wird die Schleifenbedingung geprüft, die besagt, dass eine Zeile j bestimmt werden konnte. Eine entsprechende Zeile j kann genau dann nicht bestimmt werden, wenn alle Zeilen i in der Menge $I_{\min}^{(k)}$ enthalten sind oder die Werte an diesen Zeilen kleiner als oder gleich eins sind. Das heißt, eine solche Zeile j kann insbesondere dann nicht bestimmt werden, wenn die Menge $I_{\min}^{(k)}$ alle Zeilen i enthält und die Schleifenvariante als Konsequenz gleich null ist. Daraus folgt, dass die Schleifenbedingung in jedem Fall verletzt sein muss, sobald die Schleifenvariante den Wert null erreicht. Zusammengefasst liegt damit der Beweis vor, dass der Algorithmus partiell und total korrekt ist. □

Beweis der Zeitkomplexität. Abschließend soll die Zeitkomplexität von Algorithmus 5.2 untersucht werden. Genauer ist eine obere Schranke für die asymptotisch wachsende Laufzeit gesucht, die abhängig von den jeweiligen Eingaben des Algorithmus formuliert wird. Da der Algorithmus wie der vorhergehende auf einer Schleife basiert, welche die iterative Formulierung des Ziels umsetzt, gilt für die Laufzeit:

Theorem 5.4: Zeitkomplexität von Algorithmus 5.2. Gegeben seien ein Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ mit den genannten Matrizen. In Bezug auf das Modell wird vorausgesetzt, dass dieses aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\text{WS},b}$ folgt. Der Algorithmus zur Minimierung der Überlastung, der Komponenten des Vektors $\mathbf{Lx}$ größer als eins, besitzt wie angegeben eine Laufzeit von $\mathrm{O}(\mathrm{N}(\mathbf{L})^2)$.

Beweis. Vor Beginn der Schleife wird gemäß Gleichung (5.11a) eine Startlösung $\mathbf{x}^{(0)}$ ermittelt, nach welcher eine bestimmte Zahl der Werte des Vektors $\mathbf{Lx}$ minimal ist. Im ungünstigsten Fall ist keiner dieser Werte gemäß Gleichung (5.10) minimal und der größte unter den Werten größer als eins. Daraufhin wird in jedem Schleifendurchlauf unter allen nicht minimalen Werten der größte Wert minimiert, wobei die in Gleichung (5.11b) angegebenen Bedingungen erfüllt sein müssen. Als Ergebnis ist mindestens ein Wert minimal, der vorher nicht

minimal war, wie im Rahmen des Beweises der Korrektheit festgestellt wurde. Die Schleife wird immer nur dann wiederholt, wenn mindestens ein Wert verbleibt, der nicht minimal und größer als eins ist. Das heißt, die Schleife endet in jedem Fall, wenn alle Werte gemäß obiger Definition minimal sind, unabhängig von dem zuletzt genannten Vergleich. Folglich ist die Zahl der Schleifendurchläufe höchstens gleich der endlichen Zeilenzahl $\mathrm{N}(\mathbf{Lx})$ des Vektors $\mathbf{Lx}$, welche der Zeilenzahl $\mathrm{N}(\mathbf{L})$ der Auslastungsmatrix $\mathbf{L}$ entspricht.

Wie an der gleichen Stelle im vorigen Abschnitt wird vorausgesetzt, dass die Aufrufe zur Minimierung von Zielfunktionen in einer Laufzeit von $\mathrm{O}(1)$ erfolgen. Dieselbe Annahme wird im Hinblick auf die Ausführung mathematischer Operationen mit Vektoren und Matrizen getroffen. Sind diese Voraussetzungen erfüllt, gelten für die Zeilen im Rumpf der Funktionen folgende Laufzeiten (ohne aufgerufene Hilfsfunktionen):

02 … 04:	$\mathrm{O}(1)$	28:	$\mathrm{O}(1)$
05 … 16:	$\mathrm{O}(\mathrm{N}(\mathbf{L}))$	30 … 32:	$\mathrm{O}(1)$
17 … 24:	$\mathrm{O}(\mathrm{N}(\mathbf{L})^2)$	34 … 39:	$\mathrm{O}(\mathrm{N}(\mathbf{L})^2)$
25 … 27:	$\mathrm{O}(\mathrm{N}(\mathbf{L}))$		

Um die Gesamtlaufzeit zu bestimmen, werden wieder die Laufzeiten aller Zeilen addiert, ohne hierbei kleinere Summanden und konstante Faktoren zu berücksichtigen. Die resultierende obere Schranke entspricht dem zu zeigenden Ausdruck $\mathrm{O}(\mathrm{N}(\mathbf{L})^2)$. □

5.4 Optimierung der Auslastung

Nach der Minimierung der Investitionen ist zu entscheiden, welche Prozessschritte zur Herstellung der geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ ausgeführt werden sollen. Da jedem Prozessschritt im Wertstrom eines gegebenen Produkts eine MAE zugeordnet ist, bestimmt diese Entscheidung die Auslastung der betreffenden Ressourcen. Gemäß der Beschreibung in Abschnitt 2.3 besteht das dritte Planungsziel darin, die resultierende Auslastung aller im Planungszeitraum zur Verfügung stehenden Ressourcen zu optimieren. Zu diesem Zweck legt der Anwender für alle Prozessschritte, die in den Wertströmen der Produkte alternativ miteinander verknüpft sind, jeweils eine Priorität fest. Auf der Grundlage dieser Priorisierung werden die Prozessstückzahlen x_a, die diesen Prozessschritten zugeordnet sind, iterativ maximiert. Die Auslastung der Ressourcen wird dabei jeweils durch die minimale Überlastung begrenzt, die zur Herstellung der geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ notwendig ist. Entsprechend ist das Ergebnis des vorigen Planungsziels, repräsentiert durch die minimalen Investitionen, eine Eingangsgröße für die folgende Optimierung.

Zur eindeutigen Beschreibung des Algorithmus ist es wie zuvor notwendig, einige zusätzliche Symbole einzuführen. Diese werden neben den weiterhin geltenden Symbolen aus den vorherigen Abschnitten und dem entsprechenden Verzeichnis genutzt.

a	Index zur Bezeichnung der aktuell zu maximierenden Prozessstückzahl x_a, $a \in \{1, \ldots, m\}$
a'	s. o., Index einer vorher maximierten Prozessstückzahl $x_{a'}$, $a' \in \{1, \ldots, m\}$
$\mathbf{A}$	Matrix mit Koeffizienten gleich null oder eins, dient der Formulierung von Nebenbedingungen zur Beibehaltung der vorherigen Lösung $\mathbf{x}'$, $\forall i \in \{1, 2, \ldots\}, a \in \{1, \ldots, m\}\ \mathbf{A}[i, a] \in \{0, 1\}$

i	natürliche Zahl, nicht näher bezeichnete Zeile des Vektors $\mathbf{Lx}$, $i \in \mathbb{N}$
j	s. o., zuletzt ergänzte Zeile der Matrix $\mathbf{A}$, $j \in \mathbb{N}$
k'	natürliche Zahl, angegeben zur Bezeichnung eines vorherigen Iterationsschritts, $k' \in \mathbb{N}_0$
$\mathrm{M}(X)$	Menge der Glieder einer Folge $X = (x_1, x_2, \ldots)$, $\mathrm{M}(X) = \{x_1, x_2, \ldots\}$
$\mathbf{x}'$	Vektor $[x'_1 \ \ldots \ x'_m]^\top$ der Prozessstückzahlen x'_a, vorherige Lösung, $\mathbf{x}' \in \mathbb{R}^m_{\geqslant 0}$
x'_a	vorherige Teillösung mit Index $a \in \{1, \ldots, m\}$, $x'_a \in \mathbb{R}_{\geqslant 0}$

Der Algorithmus zur Optimierung der Auslastung wird nach dem Muster der vorangegangenen Abschnitte wiedergegeben. Zunächst werden das Ziel und der Ablauf beschrieben, bevor die Korrektheit und die Zeitkomplexität nachgewiesen werden.

5.4.1 Ziel, Grundidee und Datenstrukturen

Das Ziel des Algorithmus wird durch die informelle Beschreibung der Iterationsvorschrift in Abschnitt 2.3 vorweggenommen und durch die Anforderung A12 in Abschnitt 2.6 konkretisiert. Der Aspekt, dass der Anwender die Priorisierung bestimmen soll, wird durch die grafische Modellierung und die Transformation des grafischen Modells in ein mathematisches Modell erfüllt. Als Grundlage dienen die Prioritäten, welche der Anwender den ausgehenden Kanten alternativer Flusspunkte zuweist, wie in Abschnitt 4.2 beschrieben wurde. Im mathematischen Modell wird die Priorisierung durch die *MaxFolge* abgebildet, die in Abschnitt 4.4 eingeführt wurde und welche die Indizes a zu maximierender Prozessstückzahlen x_a enthält. Abweichend von den zuvor betrachteten Planungszielen ist die optimale Lösung eindeutig bestimmt, und es ist aus diesem Grund nicht notwendig, eine Strategie zur Auswahl einer optimalen Lösung festzulegen. Da wesentliche Teile des Abschnitts wie die vorigen aufgebaut sind, werden diese kürzer gefasst.

Vorabdiskussion zum gewählten Ansatz. In Abschnitt 2.3 wurde diskutiert, dass zur Optimierung der Auslastung der Gedanke nahe liegt, den allgemeinen Bedarf an bestimmten Betriebsmitteln zu minimieren. Da aber zum Zeitpunkt der Planung im Allgemeinen keine genauen Daten über den Verbrauch im laufenden Betrieb vorliegen, wird stattdessen der Ansatz gewählt, die Prozessschritte zu priorisieren. Als Folge werden Prozessschritte mit höherer Priorität und die jeweils zugeordneten Ressourcen gegenüber solchen mit niedrigerer Priorität bevorzugt. Durch die Vergabe von geeigneten Prioritäten ist es möglich, die Zielvorgaben umzusetzen, welche von der Unternehmensleitung festgelegt werden. Diese betreffen zumeist die Variabilität, Qualität, Geschwindigkeit und die Wirtschaftlichkeit der Produktionsprozesse, wie an genannter Stelle erläutert wurde.

Eine solche Priorisierung lässt sich mit Hilfe einer Kostenfunktion, die in einem einzigen Schritt maximiert oder minimiert wird, nur begrenzt abbilden. Die Gründe hierfür wurden im Zuge der Bewertung von Software für Supply Chain Management in Abschnitt 3.4 ausgeführt und gelten für dieses Planungsziel wie für die vorigen gleichermaßen. Um die Priorisierung der Prozessstückzahlen umzusetzen, wird im Folgenden ein Algorithmus beschrieben, welcher auf einem iterativen Vorgehen basiert. Der Algorithmus wurde in der Software AURELIE implementiert und wird seither bei der Bosch Rexroth AG im Rahmen der strategischen Planung der TEK angewandt.

Ziel und Vorbedingungen. Die Iterationsvorschrift, die in Abschnitt 2.3 und 2.6 in natürlicher Sprache beschrieben wurde, soll nun in formal eindeutiger Notation ausgedrückt werden. Wie zuvor sei ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ gegeben, wobei vorausgesetzt wird, dass dieses aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ resultiert. Zudem seien die minimalen oberen Schranken $minMax[i]$ für die Komponenten des Vektors $\mathbf{Lx}$ gegeben, welche jeweils die zulässige Überlastung der Ressourcen widerspiegeln. Weiterhin sei die *MaxFolge* gegeben, welche die Priorisierung der Prozessstückzahlen x_a abbildet und folglich die Basis für die Formulierung des Ziels darstellt. Anzumerken ist diesbezüglich, dass nicht alle Indizes a als Glieder in der *MaxFolge* enthalten sein müssen, sofern eine Teilmenge genügt, um die optimale Lösung eindeutig zu bestimmen.

Betrachtet werden alle Lösungen $\mathbf{x}$ mit Prozessstückzahlen x_a, welche den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entsprechen und welche die nachfolgend angegebenen Nebenbedingungen erfüllen. In jedem Iterationsschritt wird die Prozessstückzahl x_a mit dem jeweils nächsten Index a aus der *MaxFolge* maximiert, solange ein solcher Index existiert. Die Werte aller Prozessstückzahlen x_a, die jeweils in einem vorhergehenden Iterationsschritt maximiert wurden, werden dabei als zusätzliche Nebenbedingung übernommen. Dadurch werden die gültigen Lösungen $\mathbf{x}$ im Verlauf der Iteration immer weiter beschränkt, sodass nach Maximierung der Prozessstückzahl x_a mit dem letzten Index a der *MaxFolge* die optimale Lösung eindeutig bestimmt ist.

Im Hinblick auf die Nebenbedingungen gilt, dass die Lösungen $\mathbf{x}$ aller Iterationsschritte, wie oben beschrieben, den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entsprechen müssen. Vor Beginn der Iteration wird eine beliebige Startlösung $\mathbf{x}^{(0)}$ ermittelt, welche diese Nebenbedingung und alle nachfolgend genannten erfüllt. Danach müssen die Produktstückzahlen y_b im Zuge der Maximierung jeweils einer Prozessstückzahl x_a in jedem Iterationsschritt unverändert beibehalten werden. Für jede Lösung $\mathbf{x}$ müssen darüber hinaus die Nebenbedingungen in Bezug auf den Flusserhalt und die Auslastung erfüllt sein, die bereits von den vorigen Planungszielen bekannt sind. Allerdings wird die Auslastung der Ressourcen, jeweils repräsentiert durch die Komponenten des Vektors $\mathbf{Lx}$, dabei nicht durch Wert eins beschränkt. Hintergrund ist, dass die geplanten Produktstückzahlen $y_{\text{plan},b}$ die technischen Kapazitäten $y_{\text{max},b}$ für die Produkte b übersteigen können. Da es entsprechend notwendig sein kann, Ressourcen zu überlasten, gelten für die Komponenten des Vektors $\mathbf{Lx}$ die minimalen oberen Schranken $minMax[i]$, die jeweils größer als eins sein können.

Zunächst ist ein logischer Ausdruck gesucht, um festzustellen, ob eine Prozessstückzahl x_a im aktuellen Iterationsschritt maximal ist. Die Definition eines solchen Ausdrucks ist insbesondere nützlich, um im nachfolgenden Teil des Abschnitts die Korrektheit des Algorithmus nachzuweisen. Im Gegensatz zur Maximierung der Kapazitäten und zur Minimierung der Überlastung bestimmt die Abfolge der Iterationsschritte, welcher Wert maximiert bzw. minimiert werden muss. Ist im Iterationsschritt k eine Lösung $\mathbf{x}$ gegeben, dann ist zunächst zu prüfen, ob der Index a einem der ersten k Glieder der *MaxFolge* entspricht. Wenn dies zutrifft, müssen alle Prozessstückzahlen $x_{a'}$, deren Indizes a' als Glieder in der *MaxFolge* vor dem Index a enthalten sind, mit den Werten aus dem vorherigen Iterationsschritt übereinstimmen. Unter dieser Voraussetzung ist die betrachtete Prozessstückzahl x_a genau dann maximal, wenn sie ihren maximalen Wert annimmt, der erreichbar ist, solange alle der zuvor genannten Nebenbedingungen gelten. In formaler Notation lässt sich diese Aussage

durch die folgende Äquivalenz ausdrücken (der Iterationsschritt k wird wie in den vorigen Abschnitten als Exponent in Klammern angegeben):

$$\begin{aligned} k \in \mathbb{N}_0\colon \quad & x_a^{(k)} \text{ ist maximal} \Leftrightarrow \exists k' \in \{1,\ldots,k\} \\ & a = \mathit{MaxFolge}[k'] \wedge x_a^{(k)} = \max\big\{x_a\colon \mathbf{x} \geqslant \mathbf{0} \wedge \\ & \qquad \forall a' \in \{\mathit{MaxFolge}[1],\ldots,\mathit{MaxFolge}[k'-1]\}\ x_{a'} = x_{a'}^{(k'-1)} \wedge \\ & \qquad\quad \mathbf{P}\mathbf{x} = \mathbf{P}\mathbf{x}^{(k'-1)} \wedge \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \forall i \in \{1,\ldots,\mathrm{N}(\mathbf{L})\} \\ & \qquad\qquad \mathbf{L}[i,*]\mathbf{x} \leqslant \mathit{minMax}[i]\big\}. \end{aligned} \tag{5.14}$$

Mit Hilfe von Gleichung (5.14) ist es nun möglich, einen Ausdruck für die Lösung $\mathbf{x}$ im aktuellen Iterationsschritt anzugeben. Die Startlösung $\mathbf{x}^{(0)}$ bezeichnet eine beliebige Lösung, welche den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht und analog zu Algorithmus A.2.1 im Anhang ermittelt wird. Das heißt, die Summe aller Produktstückzahlen y_b wird maximiert, wobei die Stückzahl y_b jedes Produkts b durch den geplanten Wert $y_{\text{plan},b}$ nach oben beschränkt ist. Dabei gelten die Nebenbedingungen in Bezug auf den Flusserhalt und die Auslastung, die bereits aus der Beschreibung der vorherigen Algorithmen bekannt sind. Der einzige Unterschied besteht nun darin, dass die Komponenten des Vektors $\mathbf{L}\mathbf{x}$ nicht durch den Wert eins, sondern durch die minimalen oberen Schranken $\mathit{minMax}[i]$ beschränkt sind. In den darauffolgenden Iterationsschritten k wird jeweils diejenige Prozessstückzahl x_a maximiert, deren Index a dem nächsten Glied der *MaxFolge* entspricht. Einschränkend gelten alle Nebenbedingungen, die in Gleichung (5.14) aufgeführt sind. Zusammengefasst kann für die Lösung $\mathbf{x}$ folgender Ausdruck angegeben werden:

$$\begin{aligned} k = 0\colon \quad & \mathbf{x}^{(0)} = \arg\max\big\{\mathbf{1}^\top \mathbf{P}\mathbf{x}\colon \mathbf{x} \geqslant \mathbf{0} \wedge \\ & \qquad \mathbf{P}\mathbf{x} \leqslant \mathbf{y}_{\text{plan}} \wedge \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \forall i \in \{1,\ldots,\mathrm{N}(\mathbf{L})\} \\ & \qquad\quad \mathbf{L}[i,*]\mathbf{x} \leqslant \mathit{minMax}[i]\big\}, \end{aligned} \tag{5.15a}$$

$$\begin{aligned} k \in \mathbb{N}\colon \quad & \mathbf{x}^{(k)} = \arg\max\big\{x_a\colon \mathbf{x} \geqslant \mathbf{0} \wedge \\ & \qquad \forall a' \in \{\mathit{MaxFolge}[1],\ldots,\mathit{MaxFolge}[k-1]\}\ x_{a'} = x_{a'}^{(k-1)} \wedge \\ & \qquad\quad \mathbf{P}\mathbf{x} = \mathbf{P}\mathbf{x}^{(k-1)} \wedge \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \forall i \in \{1,\ldots,\mathrm{N}(\mathbf{L})\} \\ & \qquad\qquad \mathbf{L}[i,*]\mathbf{x} \leqslant \mathit{minMax}[i]\big\}. \end{aligned} \tag{5.15b}$$

Nachdem im aktuellen Iterationsschritt die Lösung $\mathbf{x}$ gemäß obiger Gleichung bestimmt wurde, steht fest, welche Prozessstückzahl x_a ihren maximalen Wert erreicht hat. Da die Iterationsvorschrift die in Gleichung (5.14) formulierte Bedingung umsetzt, muss dies jeweils die Prozessstückzahl x_a mit dem zuletzt betrachteten Index a aus der *MaxFolge* sein. Die Prüfung verbleibender noch nicht maximaler bzw. minimaler Werte, die jeweils im Fall der vorigen Algorithmen nötig war, entfällt entsprechend. Ebenso folgt aus den Gleichungen (5.14) und (5.15a, 5.15b), dass nach jedem Iterationsschritt mindestens ein weiterer Wert maximal ist. Die Iteration endet, sobald alle Prozessstückzahlen x_a maximiert wurden, deren Indizes a in der *MaxFolge* enthalten sind.

Bemerkung: In Abschnitt 2.3 und 2.6 wurde ausgesagt, dass die Iteration mit der vorletzten Priorität enden kann, um das Ergebnis eindeutig zu bestimmen. Die Grundannahme war, dass alle Prozessstückzahlen x_a betrachtet werden und als Folge nach der Maximierung aller vorherigen der Wert der letzten Prozessstückzahl x_a feststeht. Im Zuge der Transformation werden an alternativen Flusspunkten jeweils die Indizes a' aller erzeugten Prozessstückzahlen $x_{a'}$ außer der letzten in die *MaxFolge* aufgenommen. Das heißt, die getroffene Aussage wird umgesetzt, indem dieser Folge nicht notwendigerweise die Indizes a aller Prozessstückzahlen x_a hinzugefügt werden. Daher muss auch die Prozessstückzahl x_a, deren Index a als letztes Glied in der *MaxFolge* enthalten ist, maximiert werden.

Es existieren drei Vorbedingungen, die erfüllt sein müssen, damit die angegebene Iterationsvorschrift zu einer sinnvoll zu interpretierenden, optimalen Lösung führt. Die ersten zwei Vorbedingungen sind von den vorhergehenden Algorithmen bekannt, wobei nun zusätzlich die *MaxFolge* aufgenommen wird:

V1 *Das mathematische Modell* $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ *muss aus der Transformation eines validierten grafischen Modells* $\{G_{\text{WS},b}\}$ *folgen.* Durch diese Vorbedingung ist sichergestellt, dass die Matrizen Werte innerhalb ihrer Definitionsbereiche annehmen. Zusätzlich wird gefordert, dass die *MaxFolge* die Indizes a zu maximierender Prozessstückzahlen x_a in der notwendigen Sortierung enthält, um die Priorisierung des Anwenders umzusetzen. Wie an gleicher Stelle in den vorigen beiden Abschnitten werden die Vorbedingungen und die Korrektheit von Algorithmus 4.2 und 4.4 vorausgesetzt.

V2 *Die Stückzahlen y_b aller Produkte b müssen durch die Bedingungen in Gleichung* (5.5) *und* (5.6b) *nach oben beschränkt sein.* Die Vorbedingung wird aus den vorhergehenden Abschnitten übernommen und ist nur erfüllt, wenn für jedes Produkt b mindestens ein Prozessschritt definiert ist. Der Grund für die Vorbedingung ist wie im vorherigen Abschnitt, dass die Auslastungsmatrix $\mathbf{L}$ mindestens eine Zeile ungleich dem Nullvektor enthalten muss. Zwar könnte die Formulierung konkretisiert werden, jedoch wird wie zuvor die allgemeinere Fassung verwendet.

V3 *Die minimalen oberen Schranken $minMax[i]$, welche dem Algorithmus als Argument übergeben werden, müssen geeignet gewählt sein.* Das heißt, die Werte des Vektors $\mathbf{Lx}$ müssen gemäß der Definition im vorigen Abschnitt minimal sein, und es muss mindestens eine gültige Lösung $\mathbf{x}$ existieren. Die Vorbedingung ist erfüllt, wenn die Elemente des Feldes *minMax* aus der Minimierung der Überlastung, d. h. der Komponenten des Vektors $\mathbf{Lx}$ größer als eins, folgen. Diesbezüglich wird vorausgesetzt, dass die Vorbedingungen von Algorithmus 5.2 gelten und dieser korrekt ist.

Grundidee zum Ablauf. Der Ablauf zur Optimierung der Auslastung, d. h. der Komponenten des Vektors $\mathbf{Lx}$, folgt wie zuvor der iterativen Formulierung des Ziels. Im weiteren Verlauf des Abschnitts werden die einzelnen Schritte anhand von Algorithmus 5.3 erläutert, welcher eine Kurzfassung in natürlicher Sprache darstellt. Die Langfassung in formalem, kommentiertem Pseudocode ist analog zu den vorigen Abschnitten als Algorithmus A.2.4 im Anhang zu finden. An der gleichen Stelle sind im Anhang alle verwendeten Objekte und Variablen sowie Funktionen und Prozeduren aufgeführt.

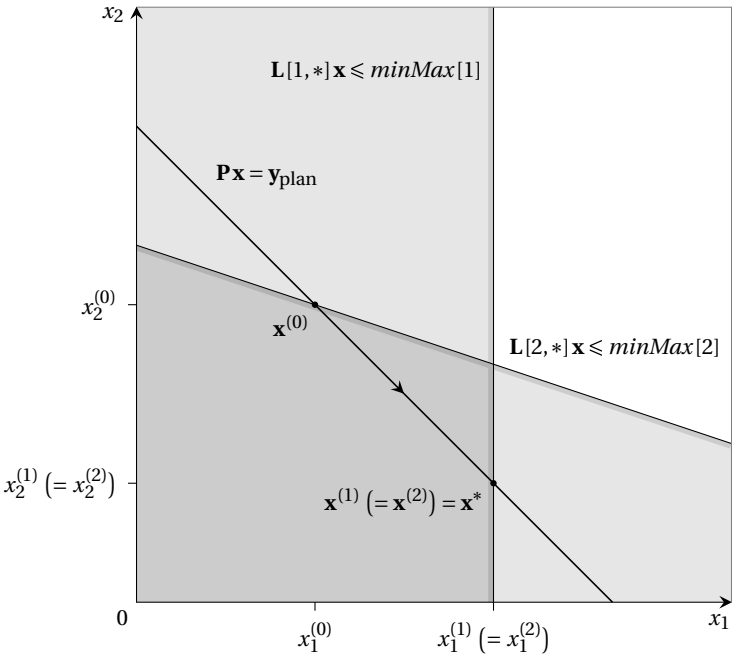


Abbildung 5.2: Visualisierung der Suche optimaler Auslastung $\mathbf{L}\mathbf{x}$ (Beispiel). Dargestellt ist die iterative Maximierung der Prozessstückzahlen $x_{1/2}$, welche die vorgegebene Priorisierung umsetzt. Die gültigen Lösungen $\mathbf{x}$ liegen auf der Geraden der geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und werden durch die zulässige Auslastung der Ressourcen begrenzt, entsprechend den Schranken $minMax[1/2]$ für die Komponenten des Vektors $\mathbf{L}\mathbf{x}$. Die gesuchte Lösung $\mathbf{x}^*$ bezeichnet den Schnittpunkt mit der Geraden $\mathbf{L}[1,*]\mathbf{x} = minMax[1]$, der größten zulässigen Auslastung der ersten Ressource.

Analog zur Maximierung der Kapazitäten soll der grundsätzliche Ablauf von Algorithmus 5.3 an einem einfachen Beispiel grafisch veranschaulicht werden. Wie in Abbildung 5.2 dargestellt, sind zwei Prozessstückzahlen $x_{1/2}$ gegeben, welche den Raum der möglichen Lösungen $\mathbf{x}$ aufspannen. Gleich dem Beispiel in Abschnitt 5.2 werden die Lösungen $\mathbf{x}$ durch die Auslastung zweier Ressourcen begrenzt, wobei die zugehörigen Werte jeweils den Komponenten des Vektors $\mathbf{L}\mathbf{x}$ entsprechen. In diesem Fall werden die Komponenten des Vektors $\mathbf{L}\mathbf{x}$ jedoch nicht durch den Wert eins, sondern durch die zuvor ermittelten, minimalen oberen Schranken $minMax[i]$ beschränkt. Die entsprechenden Bedingungen werden im allgemeinen, mehrdimensionalen Fall durch Hyperebenen und im vorliegenden, zweidimensionalen Fall durch Geraden repräsentiert. Die Geraden teilen den Raum der Prozessstückzahlen $x_{1/2}$ jeweils in einen abgeschlossenen, gültigen und einen offenen, ungültigen Halbraum. Die konvexe Schnittmenge der gültigen Halbräume, im allgemeinen Fall ein Polytop, wird im vorliegenden Fall als Polygon dargestellt. Neben dem Flusserhalt gilt die Bedingung, dass die Produktstückzahlen $\mathbf{y}$, entsprechend dem Vektor $\mathbf{P}\mathbf{x}$, mit den geplanten Werten $\mathbf{y}_{\text{plan}}$ übereinstimmen müssen. Indem man diese Bedingung wie in Abbildung 5.2 als Gerade darstellt, ist es möglich, die Menge der gültigen Lösungen $\mathbf{x}$ wie folgt zu beschreiben: Die Ortsvektoren der gültigen Lösungen $\mathbf{x}$ entsprechen den Punkten, die innerhalb der Schnittmenge der gültigen Halbräume auf der genannten Gerade liegen.

Bevor die Iteration beginnt, wird eine beliebige Startlösung $\mathbf{x}^{(0)}$ ermittelt, welche die genannten Bedingungen erfüllt. In dem gegebenen Beispiel ist dies diejenige Lösung $\mathbf{x}$, welche zugleich die Prozessstückzahl x_2 maximiert, wobei auch jede andere gültige Lösung $\mathbf{x}$ möglich ist. Somit beschreibt die Startlösung $\mathbf{x}^{(0)}$ den Schnittpunkt der zwei Geraden $\mathbf{P}\mathbf{x} = \mathbf{y}_{\text{plan}}$ und $\mathbf{L}[2,*]\mathbf{x} = \mathit{minMax}[2]$, entsprechend der größten zulässigen Auslastung der zweiten Ressource. Die *MaxFolge* sei in dem Beispiel durch die Folge $(1,2)$ gegeben, woraus folgt, dass im ersten Iterationsschritt die Prozessstückzahl x_1 maximiert werden muss. Daraus resultiert die Lösung $\mathbf{x}^{(1)}$, welche den Schnittpunkt der zwei Geraden $\mathbf{P}\mathbf{x} = \mathbf{y}_{\text{plan}}$ und $\mathbf{L}[1,*]\mathbf{x} = \mathit{minMax}[1]$ bezeichnet, entsprechend der größten zulässigen Auslastung der ersten Ressource. Im zweiten Iterationsschritt ist die Prozessstückzahl x_2 zu maximieren, wobei der ermittelte Wert der Prozessstückzahl x_1 beibehalten werden muss. Da eine Steigerung der Prozessstückzahl x_2 aber nur erreicht werden kann, indem die Prozessstückzahl x_1 reduziert wird, ist die Lösung $\mathbf{x}^{(2)}$ gleich der vorherigen Lösung $\mathbf{x}^{(1)}$. Damit endet die Iteration, und die Lösung $\mathbf{x}^{(2)}$ entspricht der letztlich gesuchten optimalen Lösung $\mathbf{x}^*$.

Datenstrukturen. Im Hinblick auf die verwendeten Datenstrukturen ist zuerst die *MaxFolge* zu nennen, welche dem Algorithmus als Argument übergeben wird. Die *MaxFolge* enthält die Indizes a zu maximierender Prozessstückzahlen x_a und bildet durch deren Sortierung, wie in Abschnitt 4.4 beschrieben, die Priorisierung des Anwenders ab. Beginnend bei dem ersten Glied der Folge werden im Verlauf der Iteration nacheinander alle Glieder betrachtet. In der Implementierung, welche in dieser Arbeit durch Algorithmus 5.3 angegeben ist, wird das erreicht, indem fortlaufend das erste Glied der Folge entfernt wird. Es ist daher wichtig zu unterscheiden, zu welchem Zeitpunkt vor, während oder nach der Iteration die *Max*↵*Folge* betrachtet wird. Zu diesem Zweck wird in der Notation, die auch zur Formulierung des Ziels verwendet wurde, der Iterationsschritt als Exponent in Klammern ergänzt. Das heißt, der Ausdruck $\mathit{MaxFolge}^{(0)}$ bezeichnet die Folge vor Beginn der Iteration, der Ausdruck $\mathit{MaxFolge}^{(k)}$ nach dem Iterationsschritt k.

Ergänzend zu den bereits eingeführten Skalaren, Vektoren und Matrizen werden zudem die minimalen oberen Schranken $\mathit{minMax}[i]$ benötigt. Diese werden durch die Ausführung des zuvor beschriebenen Algorithmus 5.2 ermittelt und dem aktuellen Algorithmus 5.3 als Argument übergeben. Im Gegensatz zur *MaxFolge* wird das eindimensionale, reellwertige Feld *minMax* vor Beginn und im Verlauf der Iteration nicht verändert.

Um die Beschreibung zu vereinfachen, wird in der Langfassung im Anhang darüber hinaus die Variable *zeilenzahl* verwendet, welche die gleiche Bedeutung wie im vorigen Abschnitt besitzt. Entsprechend gibt die Variable die größte Zeile der Auslastungsmatrix $\mathbf{L}$ an, welche nicht dem Nullvektor entspricht. Unter den genannten Vorbedingungen müssen diese Zeile und folglich die Zeilenzahl der Matrix größer als null sein.

5.4.2 Beschreibung des Algorithmus

Im nächsten Schritt wird Algorithmus 5.3 beschrieben, mit dessen Hilfe das Ziel verfolgt wird, die Auslastung der Ressourcen zu optimieren (Langfassung siehe Algorithmus A.2.4). Wie im vorigen Abschnitt seien die geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und die Matrizen des

Eingabe: Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$, Prozessmatrix $\mathbf{P}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$, minimale obere Schranken *minMax*, *MaxFolge*
Ausgabe: Lösung $\mathbf{x}$, welche unter den vorgegebenen Bedingungen die Prozessstückzahlen x_a gemäß der Sortierung der *MaxFolge* maximiert

```
01: Funktion OptAuslastung(y_plan, P, F, L, minMax, MaxFolge)
02:     initialisiere Lösung x entsprechend InitOptAuslastung(...), Matrix A als Nullmatrix und
...       Index a der Prozessstückzahl x_a als unbestimmt
07:     solange MaxFolge nicht leer ist tue
08:         wenn vorher eine Prozessstückzahl x_a maximiert wurde dann
...           ergänze fortlaufend in einer neuen Zeile der Matrix A an Spalte a einen Eintrag gleich eins
11:         entferne Index a der aktuellen Prozessstückzahl x_a vom Beginn der MaxFolge
12:         speichere vorherige Lösung x als Lösung x'
13:         bestimme eine Lösung x, welche die Prozessstückzahl x_a wie folgt maximiert:
                x ← arg max{x_a: x ≥ 0 ∧ Ax = Ax' ∧ Px = Px' ∧ Fx = 0 ∧ ∀i ∈ {1,...,Zeilen der Matrix L}
                    L[i,*]x ≤ minMax[i]}
14:     liefere optimale Lösung x zurück

15: Funktion InitOptAuslastung(y_plan, P, F, L, minMax)
...     initialisiere Lösung x als Nullvektor
18:     bestimme eine Lösung x, welche den geplanten Produktstückzahlen y_plan,b wie folgt entspricht:
            x ← arg max{1^T Px: x ≥ 0 ∧ Px ≤ y_plan ∧ Fx = 0 ∧ ∀i ∈ {1,...,Zeilen der Matrix L}
                L[i,*]x ≤ minMax[i]}
19:     liefere initiale Lösung x zurück
```

Algorithmus 5.3: Optimierung der Auslastung $\mathbf{Lx}$ (alle Komponenten) (Kurzfassung).

mathematischen Modells ($\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}}$) gegeben. Hinzu kommen die zuvor ermittelten, minimalen oberen Schranken $minMax[i]$ und die *MaxFolge*, welche die sortierten Indizes a aller iterativ zu maximierenden Prozessstückzahlen x_a enthält.

Funktion OptAuslastung(...)

In Zeile 01 ist die Hauptfunktion deklariert, die dazu dient, die Auslastung zu optimieren, repräsentiert durch die Komponenten des Vektors $\mathbf{Lx}$. Wie im Fall von Algorithmus 5.2 im vorigen Abschnitt basiert die Funktion auf einer Schleife, welche die Iterationsvorschrift nach Gleichung (5.15a, 5.15b) umsetzt. Vor Beginn der Iteration wird eine Startlösung $\mathbf{x}$ ermittelt, wofür eine entsprechende, weiter unten erläuterte Hilfsfunktion aufgerufen wird. Darüber hinaus wird die Matrix $\mathbf{A}$, die für diesen Algorithmus eine besondere Bedeutung besitzt, mit der Nullmatrix initialisiert. Im Iterationsverlauf wird nach der Maximierung einer Prozessstückzahl x_a in einer neuen Zeile dieser Matrix die Komponente an Spalte a mit dem Wert eins festgelegt. Als Folge wächst die Matrix $\mathbf{A}$ mit der Zahl der Prozessstückzahlen x_a, die in vorherigen Iterationsschritten maximiert wurden. Durch die Nebenbedingung, dass sich die Komponenten des Vektors $\mathbf{Ax}$ nicht verändern dürfen, werden die vorher maximierten Prozessstückzahlen x_a unverändert beibehalten. Um eine neue Zeile in der Matrix $\mathbf{A}$ zu ergänzen, wird die jeweils letzte Zeile mit dem Wert null initialisiert und nach jedem Iterationsschritt aktualisiert. Ebenso wird der Index a der zuletzt maximierten Prozessstückzahl x_a als unbestimmt initialisiert und aktualisiert.

Danach folgt in Zeile 07 die Schleife, in deren Verlauf die Prozessstückzahlen x_a gemäß der Sortierung der Indizes a in der *MaxFolge* iterativ maximiert werden. Die Schleife wird betreten, wenn die *MaxFolge* mindestens einen Index a enthält und entsprechend mindestens eine solche Prozessstückzahl x_a existiert. Wie beschrieben, werden die bisher maximierten Prozessstückzahlen x_a unverändert beibehalten, indem fortlaufend entsprechende Zeilen in der Matrix $\mathbf{A}$ ergänzt werden. Ausgenommen ist die letzte Prozessstückzahl x_a, da es nach deren Maximierung nicht notwendig ist, eine solche Zeile hinzuzufügen. In der angegebenen Implementierung wird dies umgesetzt, indem vor der Maximierung der aktuell zu betrachtenden Prozessstückzahl x_a eine Zeile für die jeweils vorhergehende ergänzt wird. Da im ersten Durchlauf der Schleife der Index a der zuletzt maximierten Prozessstückzahl x_a unbestimmt ist, wird dieser Schritt übersprungen.

Anschließend wird der Index a der aktuell zu maximierenden Prozessstückzahl x_a bestimmt, indem das erste Glied der *MaxFolge* entfernt und gespeichert wird. Zur Formulierung der Nebenbedingungen wird die Lösung $\mathbf{x}$, die im vorhergehenden Iterationsschritt gefunden wurde, als Lösung $\mathbf{x}'$ übernommen. Daraufhin wird eine neue Lösung $\mathbf{x}$ gesucht, welche die aktuelle Prozessstückzahl x_a maximiert, wobei folgende Nebenbedingungen gelten: Die Komponenten des Vektors $\mathbf{Ax}$ müssen im Vergleich zur Lösung $\mathbf{x}'$ die gleichen Werte annehmen, wodurch die zuvor gefundenen, bereits maximierten Prozessstückzahlen x_a beibehalten werden. Zudem müssen die Stückzahlen y_b aller Produkte b, entsprechend dem Vektor $\mathbf{Px}$, mit der vorherigen Lösung $\mathbf{x}'$ übereinstimmen. Da die Startlösung, wie weiter unten erläutert wird, den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht, ist dadurch sichergestellt, dass dies auf alle Lösungen $\mathbf{x}$ zutrifft. Analog zu den vorhergehenden Algorithmen muss jede Lösung $\mathbf{x}$ den Flusserhalt erfüllen, wie durch Gleichung (4.13) im vorigen Kapitel definiert. In Bezug auf die Komponenten des Vektors $\mathbf{Lx}$ gilt die Nebenbedingung, dass diese die minimalen oberen Schranken $minMax[i]$ nicht überschreiten dürfen. Nachdem alle Indizes a in der *MaxFolge* betrachtet wurden und letztere entsprechend keine Glieder enthält, wird die Schleife verlassen und die Lösung $\mathbf{x}$ zurückgegeben.

Funktion InitOptAuslastung(...)

Ab Zeile 15 ist die Hilfsfunktion zur Bestimmung einer Startlösung definiert, die weiter oben erwähnt wurde. Ausgehend vom Nullvektor wird die Summe der Stückzahlen y_b aller Produkte b maximiert, wobei diese jeweils den geplanten Wert $y_{\text{plan},b}$ nicht überschreiten dürfen. Analog zu Algorithmus A.2.1, der in der Kurzeinführung zu Beginn des Kapitels erläutert wurde, entspricht dies der Maximierung jeder Produktstückzahl y_b bis zum geplanten Wert $y_{\text{plan},b}$. Die gesuchte Lösung $\mathbf{x}$ muss die Bedingungen in Bezug auf den Flusserhalt und die Auslastung erfüllen, welche im vorigen Kapitel durch Gleichung (4.13) bzw. (4.14) definiert wurden. Abweichend von dieser Definition gelten für die Komponenten des Vektors $\mathbf{Lx}$ anstelle des Wertes eins jeweils die zuvor ermittelten, minimalen oberen Schranken $minMax[i]$. Anzumerken ist, dass unter diesen Bedingungen im Allgemeinen eine unendliche Menge gültiger Lösungen $\mathbf{x}$ existiert. Zum Abschluss wird eine dieser gültigen Lösungen $\mathbf{x}$ zurückgegeben und damit die Hilfsfunktion verlassen.

5.4.3 Beweis der Korrektheit und Zeitkomplexität

Zum Abschluss folgt die Untersuchung von Algorithmus 5.3, wobei analog zu den vorigen Abschnitten zuerst dessen Korrektheit nachgewiesen werden muss. Danach wird die Komplexität des Algorithmus in Bezug auf die Laufzeit betrachtet. Der Speicherplatzbedarf ist abhängig von den Datenstrukturen, welche neben den Skalaren, Vektoren und Matrizen die minimalen oberen Schranken $minMax[i]$ und die *MaxFolge* einschließen. Die Größe des Feldes *minMax* ist durch die Zeilenzahl $\mathrm{N}(\mathbf{L})$ der Auslastungsmatrix $\mathbf{L}$ beschränkt, die *MaxFolge* durch die Zahl m der Prozessstückzahlen x_a.

Beweis der Korrektheit. Im Fall des vorliegenden Algorithmus 5.3 ist die Nachbedingung, die zum Beweis der Korrektheit erfüllt werden muss, durch Gleichung (5.14) gegeben. Das Ziel des Algorithmus lautet dementsprechend, dass alle Prozessstückzahlen x_a iterativ maximiert werden müssen, deren Indizes a in der *MaxFolge* enthalten sind. Genauer formuliert muss jede dieser Prozessstückzahlen x_a gemäß der Definition, die in der genannten Gleichung angegeben ist, ihren maximalen Wert annehmen. Im Hinblick auf die Nachbedingung des Algorithmus gilt es somit, die folgende Aussage zu beweisen:

Theorem 5.5: Korrektheit von Algorithmus 5.3. Gegeben seien ein Vektor geplanter Produktstückzahlen $\mathbf{y}_{\mathrm{plan}}$ und ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\mathrm{Invest}})$ mit den genannten Matrizen. In Bezug auf das Modell wird vorausgesetzt, dass dieses aus der Transformation eines grafischen Modells $\{G_{\mathrm{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\mathrm{WS},b}$ folgt. Zudem seien die minimalen oberen Schranken $minMax[i]$ und die *MaxFolge* mit den Indizes a iterativ zu maximierender Prozessstückzahlen x_a gegeben. Des Weiteren seien alle Vorbedingungen des Algorithmus erfüllt. Unter diesen Voraussetzungen ist der Algorithmus zur Optimierung der Auslastung, der Komponenten des Vektors $\mathbf{Lx}$, partiell und total korrekt. Korrekt heißt hierbei, dass nach Beendigung des Algorithmus die Prozessstückzahlen x_a maximal sind, wie in Gleichung (5.14) definiert.

Beweis. Um zunächst die partielle Korrektheit des Algorithmus nachzuweisen, wird eine geeignete Schleifeninvariante gewählt. Dieser logische Ausdruck muss stets wahr sein, sofern die Eingabe des Algorithmus den Vorbedingungen genügt, und dem obigen Ziel entsprechen, wenn die Schleifenbedingung verletzt wird. Im Fall des betrachteten Algorithmus ist die Schleifenbedingung dadurch gegeben, dass die $MaxFolge^{(k)}$ mindestens einen Index a einer Prozessstückzahl x_a enthält. Mit Verweis auf Gleichung (5.14) wird die folgende Schleifeninvariante definiert, welche sich auf die *MaxFolge* vor dem Beginn und nach jedem Durchlauf der Schleife bezieht. Zur Formulierung des Ausdrucks wird die Funktion $\mathrm{M}(X)$ eingeführt, welche die Menge aller Glieder einer Folge X zurückgibt:

Schleifeninvariante:

$$\forall\, a \in \mathrm{M}\left(MaxFolge^{(0)}\right) \left[a \notin \mathrm{M}\left(MaxFolge^{(k)}\right) \rightarrow x_a^{(k)} \text{ ist maximal} \right]. \tag{5.16}$$

Demnach sind alle zu maximierenden Prozessstückzahlen x_a, deren Indizes a nach dem aktuellen Schleifendurchlauf nicht in der $MaxFolge^{(k)}$ enthalten sind, maximal. Ist die Schleifenbedingung nicht erfüllt, dann ist kein Index a, der zu Beginn in der *MaxFolge* enthalten

war, ein Glied dieser Folge. Das heißt, ist die Schleifenbedingung entsprechend verletzt, dann ist der linke Teil der Implikation in Gleichung (5.16) für die Indizes a aller zu maximierenden Prozessstückzahlen x_a wahr. Wenn dies der Fall ist und die Schleifeninvariante wie in Gleichung (5.16) angegeben gilt, sind alle zu maximierenden Prozessstückzahlen x_a maximal. Folglich ist unter diesen Bedingungen das Ziel des Algorithmus erreicht, wie in dem oben stehenden Theorem definiert. Im nächsten Schritt ist zu zeigen, dass die Schleifeninvariante vor dem Beginn der Schleife und nach jedem Durchlauf gültig ist.

Bevor die Schleife beginnt, sind alle Indizes a iterativ zu maximierender Prozessstückzahlen x_a in der $MaxFolge^{(0)}$ enthalten. Entsprechend ist der linke Teil von Gleichung (5.16) für keinen dieser Indizes a erfüllt und die Implikation wahr (Induktionsanfang). Im Folgenden wird vorausgesetzt, dass die Schleifeninvariante für den vorhergehenden Schleifendurchlauf galt und der Ausdruck für die $MaxFolge^{(k-1)}$ wahr ist. Alle Indizes a iterativ zu maximierender Prozessstückzahlen x_a, die vor dem aktuellen Schleifendurchlauf nicht in der $MaxFolge^{(k-1)}$ enthalten waren, sind danach auch weiterhin nicht in der $MaxFolge^{(k)}$ enthalten. Die betreffenden Prozessstückzahlen $x_a^{(k-1)}$ wurden jeweils in einem vorherigen Durchlauf der Schleife maximiert und sind unter der genannten Voraussetzung der Definition entsprechend maximal. Da zuvor maximierte Werte durch eine Nebenbedingung beibehalten werden, müssen auch die entsprechenden Prozessstückzahlen $x_a^{(k)}$ maximal sein. Die Schleifeninvariante gilt folglich für alle iterativ zu maximierenden Prozessstückzahlen x_a, deren Indizes a bereits zuvor nicht mehr in der $MaxFolge^{(k-1)}$ enthalten waren.

Danach verbleibt nur der Beweis für die Prozessstückzahl x_a, deren Index a ein Glied der $MaxFolge^{(k-1)}$ war und welche im aktuellen Schleifendurchlauf maximiert wird. Der jeweilige Index a wird als erstes Glied aus der *MaxFolge* entfernt und ist entsprechend nicht mehr in der $MaxFolge^{(k)}$ enthalten. Da die Vorschrift zur Maximierung in Gleichung (5.15b) die Bedingung gemäß Gleichung (5.14) umsetzt, ist die Prozessstückzahl x_a nach dem aktuellen Durchlauf der Schleife maximal. Unter der obigen Voraussetzung, dass die Schleifeninvariante nach dem vorhergehenden Schleifendurchlauf galt, ist diese folglich auch für den zuletzt entfernten Index a erfüllt. Als Zwischenergebnis lässt sich festhalten, dass die Schleifeninvariante nach jedem Durchlauf der Schleife wahr ist (Induktionsschritt).

Nachdem folglich die partielle Korrektheit des Algorithmus bewiesen ist, muss im nächsten Schritt dessen totale Korrektheit nachgewiesen werden. Hierzu wird eine Schleifenvariante festgelegt, ein Ausdruck, welcher eine natürliche Zahl größer als null bezeichnen und mit jedem Durchlauf der Schleife reduziert werden muss. Erreicht die definierte Schleifenvariante einen Wert von null, muss die Schleifenbedingung verletzt sein. Ein geeigneter Ausdruck, welcher diese Anforderungen erfüllt, lautet wie folgt:

$$\textit{Schleifenvariante:} \qquad \left|\mathrm{M}\left(MaxFolge^{(k)}\right)\right|. \tag{5.17}$$

Der Ausdruck in der obigen Gleichung (5.17) bezeichnet die Zahl der Indizes a, die nach dem aktuellen Schleifendurchlauf in der $MaxFolge^{(k)}$ verbleiben. Vor Beginn der Schleife ist der Ausdruck nicht größer als die endliche Zahl m der erzeugten Prozessstückzahlen x_a, in jedem Fall aber größer als oder gleich null. Falls der Ausdruck gleich null ist und die $MaxFolge^{(0)}$ entsprechend keine Glieder enthält, wird die Schleife nicht betreten. Da mit

jedem Durchlauf der Schleife genau ein Index a, genauer das erste Glied, aus der *MaxFolge* entfernt wird, sinkt die Zahl der enthaltenen Indizes a jeweils um den Wert eins. Ist diese Zahl gleich null, dann ist kein Index a in der $MaxFolge^{(k)}$ enthalten, die Schleifenbedingung ist verletzt, und die Schleife wird verlassen. Fasst man dies zusammen, ist damit der Beweis erbracht, dass der Algorithmus sowohl partiell als auch total korrekt ist. □

Beweis der Zeitkomplexität. Im letzten, nun folgenden Teil dieses Abschnitts soll die Zeitkomplexität von Algorithmus 5.3 untersucht werden. Gesucht ist wie zuvor ein Ausdruck, welcher von den jeweiligen Eingaben des Algorithmus abhängig ist und eine obere Schranke für die asymptotisch wachsende Laufzeit beschreibt. Kennzeichnend für den Algorithmus ist, dass dieser auf einer Schleife basiert, deren wiederholter Durchlauf die iterative Formulierung des Ziels umsetzt. Für die Laufzeit gilt:

Theorem 5.6: Zeitkomplexität von Algorithmus 5.3. Gegeben seien ein Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$ und ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}})$ mit den genannten Matrizen. In Bezug auf das Modell wird vorausgesetzt, dass dieses aus der Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$ mit einer endlichen Zahl von Wertstromgraphen $G_{\text{WS},b}$ folgt. Zudem seien die minimalen oberen Schranken $minMax[i]$ und die *MaxFolge* mit den Indizes a iterativ zu maximierender Prozessstückzahlen x_a gegeben. Der Algorithmus zur Optimierung der Auslastung, repräsentiert durch die Komponenten des Vektors $\mathbf{Lx}$, besitzt in der angegebenen Implementierung eine Laufzeit von $\mathrm{O}(m)$.

Beweis. In jedem Durchlauf wird genau ein Index a aus der *MaxFolge* entfernt, um die jeweilige Prozessstückzahl x_a zu maximieren. Nach der Maximierung wird die Schleife immer nur dann wiederholt, wenn die verbleibende $MaxFolge^{(k)}$ mindestens einen Index a einer noch zu maximierenden Prozessstückzahl x_a enthält. Als Folge entspricht die Zahl der Schleifendurchläufe gerade der Zahl der Indizes a, die vor Beginn der Schleife in der $MaxFolge^{(0)}$ enthalten waren. Da jeder Index a höchstens einmal in der $MaxFolge^{(0)}$ enthalten sein kann, ist diese Zahl nicht größer als die endliche Zahl m der Prozessstückzahlen x_a.

Bezüglich der Laufzeiten ist anzumerken, dass für die Aufrufe zur Maximierung von Zielfunktionen eine Laufzeit von $\mathrm{O}(1)$ vorausgesetzt wird. Wie in den vorherigen Abschnitten gilt außerdem die Annahme, dass gleichermaßen alle mathematischen Operationen mit Vektoren und Matrizen in einer Laufzeit von $\mathrm{O}(1)$ ausgeführt werden. Unter diesen Voraussetzungen können für die Zeilen im Rumpf der Funktionen folgende Laufzeiten angegeben werden (ohne aufgerufene Hilfsfunktionen):

02 . . . 06:	$\mathrm{O}(1)$
07 . . . 13:	$\mathrm{O}(m)$
14:	$\mathrm{O}(1)$
16 . . . 19:	$\mathrm{O}(1)$

Die Gesamtlaufzeit lässt sich bestimmen, indem die Laufzeiten aller Zeilen addiert werden, wobei kleinere Summanden und konstante Faktoren zu vernachlässigen sind. Die daraus folgende obere Schranke entspricht dem zu zeigenden Ausdruck $\mathrm{O}(m)$. □

5.5 Umsetzung in der Software AURELIE

Nachdem in den vorigen Abschnitten die Algorithmen zur Optimierung des mathematischen Modells betrachtet wurden, folgen nun einige Anmerkungen zur praktischen Umsetzung. Wie in Abschnitt 4.5 angesprochen, existiert neben den Algorithmen eine Vielzahl weiterer technischer, prozessualer und organisatorischer Erfolgsfaktoren zur Einführung einer Software. Aus technischer Sicht repräsentieren die Algorithmen, die in dieser Arbeit vorgestellt werden, zudem nur ca. drei Prozent des Quelltextes von AURELIE. Darüber hinaus ist insbesondere die grafische Benutzeroberfläche von Bedeutung, da sie dem Anwender die Funktionen zur Optimierung des Modells zugänglich macht. Nach einer kurz gefassten Beschreibung der Benutzerführung soll bewertet werden, inwieweit die Software AURELIE die Anforderungen aus Abschnitt 2.6 in Bezug auf die Optimierung erfüllt. Abschließend sollen Erweiterungen genannt werden, die in der Software implementiert wurden, und das Vorgehen zur Validierung der Ergebnisse erläutert werden.

5.5.1 Funktionsübersicht und Benutzerführung

Im Hinblick auf die Optimierung werden drei Planungsziele betrachtet: die Maximierung der Kapazitäten, die Minimierung der Investitionen und die Optimierung der Auslastung. Die Grundlage bildet das vom Anwender erstellte, grafische Modell, welches validiert und in ein mathematisches Modell transformiert wird. Zur Erfüllung der Anforderungen muss eine geeignete Software folglich neben dem Modell einen Optimierer einschließen, welcher mit dem Modell Eingaben und Ausgaben austauscht. Bei diesen Eingaben und Ausgaben handelt es sich um die Informationen, die zwischen dem abgebildeten System und seiner Umwelt ausgetauscht werden. Genauer sind dies die Stückzahlen für alle Produkte und deren Verteilung in den Wertströmen bzw. die daraus resultierende Auslastung der genutzten Ressourcen. Auf Basis dieser Informationen müssen Zielfunktionen und Nebenbedingungen formuliert werden, um die definierten Planungsziele umzusetzen.

Im Anschluss müssen die Zielfunktionen durch Nutzung eines exakten, mathematischen Verfahrens der linearen Optimierung maximiert bzw. minimiert werden. In der Software AURELIE kommt die Bibliothek `lp_solve` zum Einsatz, die entsprechende Standardverfahren bereitstellt und in Programmiersprachen wie z. B. Java[2] eingebunden werden kann. Für reellwertige lineare Probleme nutzt die Bibliothek ein verbessertes Simplex-Verfahren, wie in der Kurzeinführung beschrieben. Zu jedem Zeitpunkt während der Modellerstellung kann der Anwender die Funktion zur Optimierung des Modells aufrufen. Daraufhin wird das grafische Modell validiert, und wenn die Validierung erfolgreich ist, wird dieses automatisch in ein entsprechendes mathematisches Modell transformiert. Auf der Grundlage des mathematischen Modells ruft AURELIE wiederholt Funktionen von `lp_solve` auf, um Zielfunktionen mit Nebenbedingungen zu maximieren bzw. zu minimieren. Das Ergebnis wird an AURELIE zurückgegeben und in der grafischen Benutzeroberfläche angezeigt, sodass es dem Anwender möglich ist, die Lösung auszuwerten.

[2] Die Implementierung der Software AURELIE erfolgte, wie in Kapitel 4 angemerkt, auf Basis der Java Platform Standard Edition 7. Weiterführende Informationen (abgerufen am 11. Februar 2018): `http://www.oracle.com/technetwork/java/javase/overview/`.

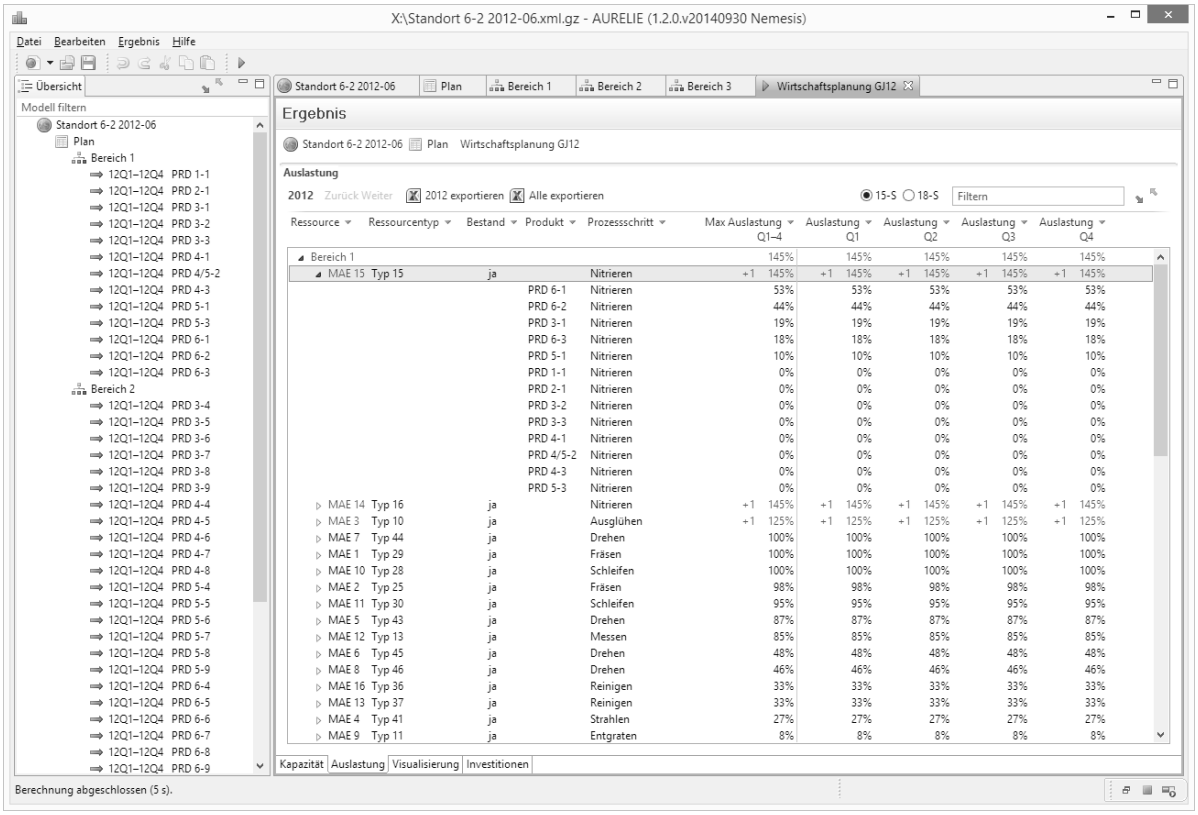


Abbildung 5.3: Bildschirmaufnahme von AURELIE (tabellarische Ergebnisdarstellung). Die Ergebnisse für die Kapazität, die Auslastung und die Investitionen werden jeweils in einer Tabelle dargestellt. Zur weiteren Analyse ist es im Fall der Auslastung durch Auswahl einer Ressource möglich, die Produkte zu ermitteln, welche zur Auslastung der Ressource beitragen. Mit Hilfe einer Filterfunktion kann der Anwender in den Wertströmen nach überlasteten Ressourcen suchen.

Tabellarische Darstellung der Ergebnisse. Um im Sinne einer umfassenden Planung ein möglichst detailliertes Bild zu zeichnen, werden alle numerischen Ergebnisse der Optimierung tabellarisch dargestellt. Für die Kapazität, die Auslastung und die Investitionen wird in eigenen Reitern jeweils eine Tabelle mit den zugehörigen Ergebnissen erzeugt, wie Abbildung 5.3 zeigt. Im ersten Reiter werden die technischen Kapazitäten, d. h. die maximalen Stückzahlen aller Produkte, mit dem Primärbedarf und dem gesamten Bedarf einschließlich des Sekundärbedarfs verglichen. Die optimale Auslastung der verfügbaren Ressourcen ist im zweiten Reiter nach Bereichen gruppiert und mit absteigender Auslastung sortiert angegeben. Zuletzt sind im dritten Reiter die minimalen Investitionen aufgeführt, wobei Ressourcen gleicher Ressourcentypen zusammengefasst werden.

Um die gefundene Lösung untersuchen zu können, sind die drei Tabellen in Abbildung 5.3 durch verschiedene Ebenen strukturiert: Die Kapazitäten werden nach den Stücklisten der Produkte gegliedert, um aufzuzeigen, ob die Auslastung einer Ressource oder die Verfügbarkeit einer Komponente die Kapazität begrenzt. Die Werte der Auslastung werden zusammenfassend für alle Bereiche und Ressourcen sowie für all diejenigen Produkte angegeben, die jeweils zur Auslastung einer Ressource beitragen. In ähnlicher Weise werden die

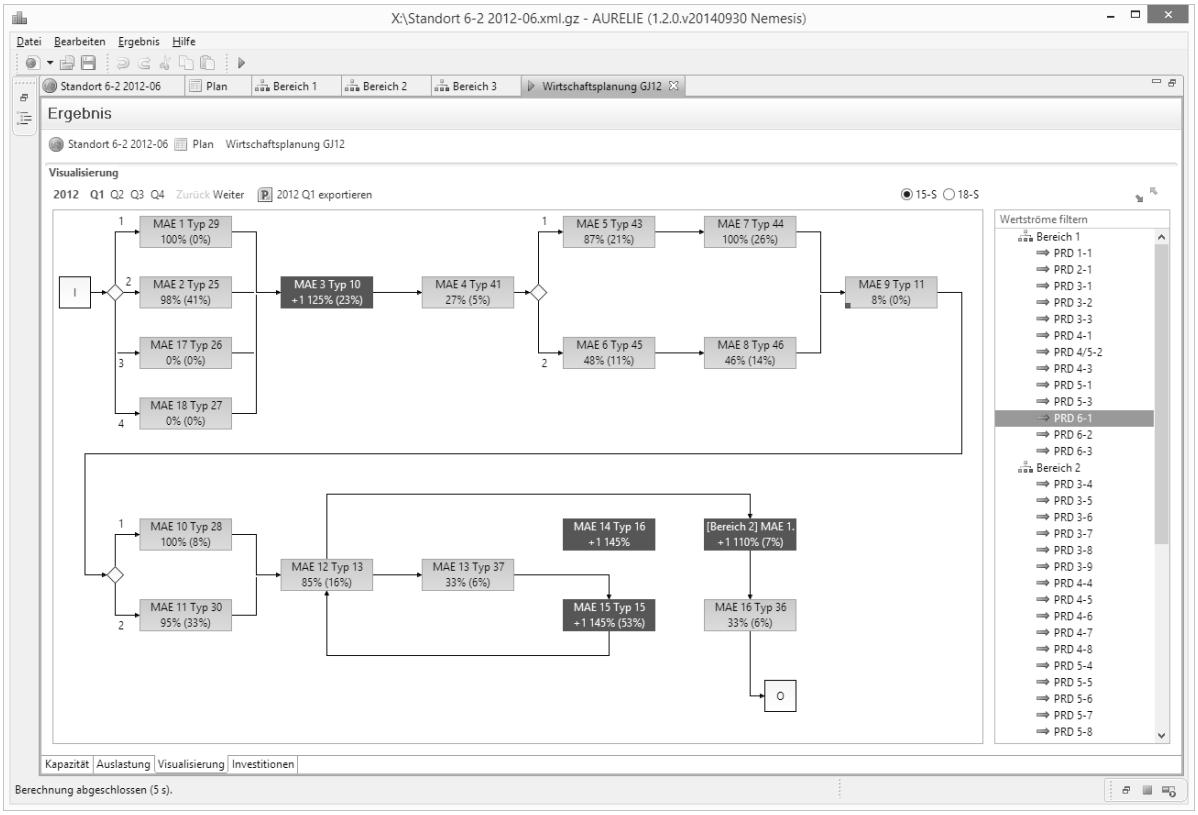


Abbildung 5.4: Bildschirmaufnahme von AURELIE (Visualisierung von Engpässen). Für Bereiche und Wertströme ist eine Übersicht verfügbar, welche die Auslastung der Ressourcen und ggf. deren Auslastung durch ein Produkt (in Klammern) wiedergibt. Hierbei werden überlastete Ressourcen hervorgehoben, um die Identifikation bestehender Engpässe zu beschleunigen. Die grafische Übersicht ergänzt die tabellarische Darstellung und unterstützt die Auswertung.

Investitionen auf übergeordneter Ebene nach Ressourcentypen zusammengefasst und für jede einzelne Ressource angegeben. Um einen Wert auf einer Ebene zu analysieren, kann der Anwender die Werte auf untergeordneten Ebenen anzeigen, wie z. B. die Auslastung einer Ressource durch ein Produkt. Zudem ist eine Filterfunktion verfügbar, um die Werte in den Tabellen nach einem Begriff durchsuchen und z. B. Engpässe für ein Produkt ermitteln zu können. Ein Engpass bezeichnet eine Ressource, die überlastet ist und die Stückzahlen eines oder ggf. mehrerer Produkte beschränkt.

Grafische Darstellung der Auslastung und der Engpässe. Der Kern des softwaregestützten Workflows ist die grafische Modellierung eines gegebenen Systems von Wertströmen. Wie in Abschnitt 2.6 dargelegt wurde, führt der grafische Ansatz insbesondere im Vergleich zu tabellenbasierten Ansätzen zu einer Steigerung der Effizienz, Transparenz und Flexibilität. Daher liegt es nahe, die Ergebnisse der Optimierung wie auch das Modell nicht nur in einer tabellarischen Übersicht, sondern auch grafisch wiederzugeben. Der Anwender kann hierzu einen Bereich oder Wertstrom auswählen, um die Auslastung der Ressourcen im grafischen Modell anzuzeigen, wie in Abbildung 5.4 dargestellt ist. Dies betrifft die Auslastung der Res-

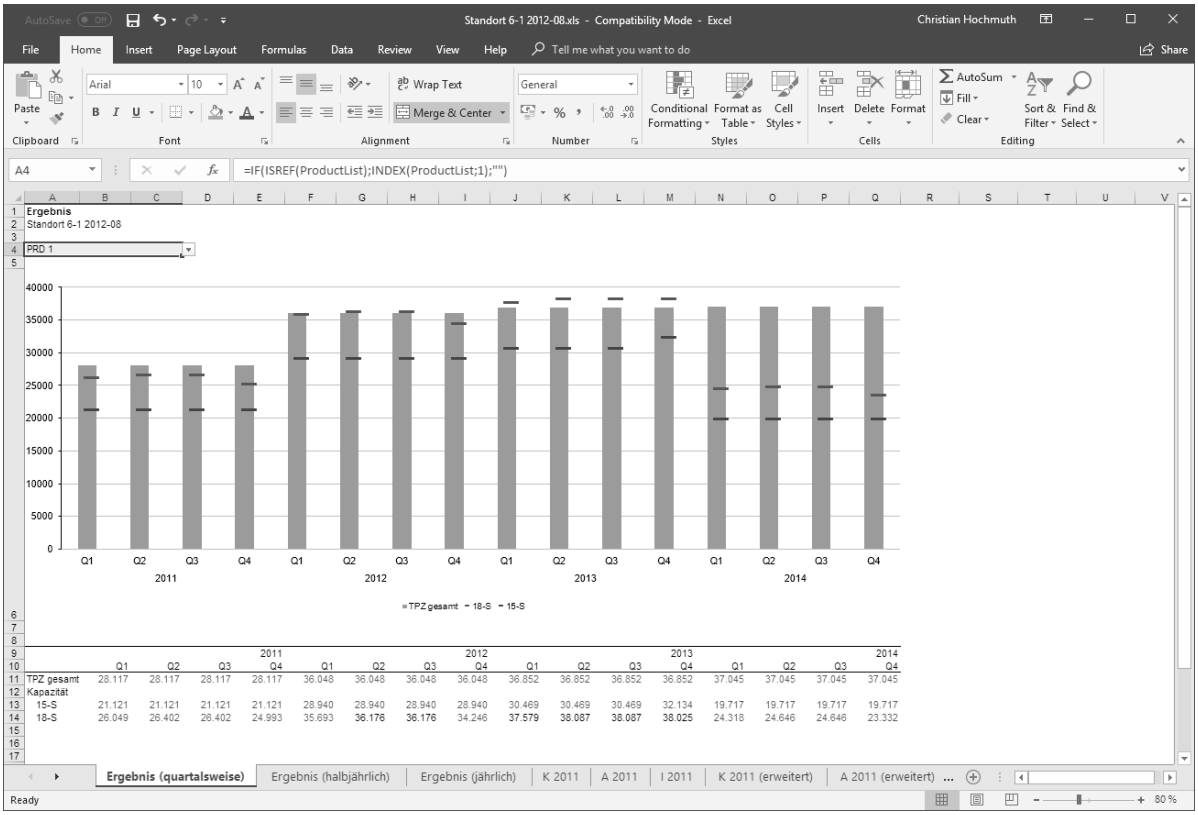


Abbildung 5.5: Bildschirmaufnahme von AURELIE (Export nach Microsoft Excel). Um die Ergebnisse der Optimierung weiterzuverarbeiten und zusätzliche Auswertungen durchzuführen, kann der Anwender alle numerischen Daten exportieren. Die Funktion gewährleistet eine nahtlose Verknüpfung mit den nachfolgenden Prozessen im Unternehmen. Zum Export der Daten wird eine Tabellenvorlage genutzt, welche eine grafische Auswertung der Kapazitätsentwicklung bietet.

sourcen und ggf. deren Auslastung durch das jeweilige Produkt, welches dem ausgewählten Wertstrom zugeordnet ist. Ist eine Ressource überlastet, wird diese visuell von allen anderen unterschieden, um Engpässe in den Wertströmen der Produkte hervorzuheben.

Weiterverarbeitung der Ergebnisse. Für einen effektiven und effizienten Einsatz im Unternehmen muss die Möglichkeit bestehen, die Ergebnisse der Optimierung in nachfolgenden Prozessen weiterzuverarbeiten. Da zur Verarbeitung numerischer Daten zumeist Tabellenkalkulationen verwendet werden, ist eine Integration mit Microsoft Excel notwendig, dem Standard zur Erstellung von Tabellenkalkulationen. Zu diesem Zweck kann der Anwender die Ergebnisse für ein ausgewähltes Jahr oder den gesamten Planungszeitraum nach Microsoft Excel exportieren. Durch die Funktion werden alle numerischen Daten, die in den genannten Tabellen dargestellt werden, in der gleichen Struktur in eine eigens definierte Tabellenvorlage übertragen. Die Tabellenvorlage stellt zusätzlich eine grafische Auswertung zur Entwicklung der Kapazität im Planungszeitraum bereit, wie Abbildung 5.5 beispielhaft zeigt. Unter Zuhilfenahme der Standardfunktionen von Microsoft Excel besteht zudem für den Anwender die Möglichkeit, eigene Auswertungen zu erstellen.

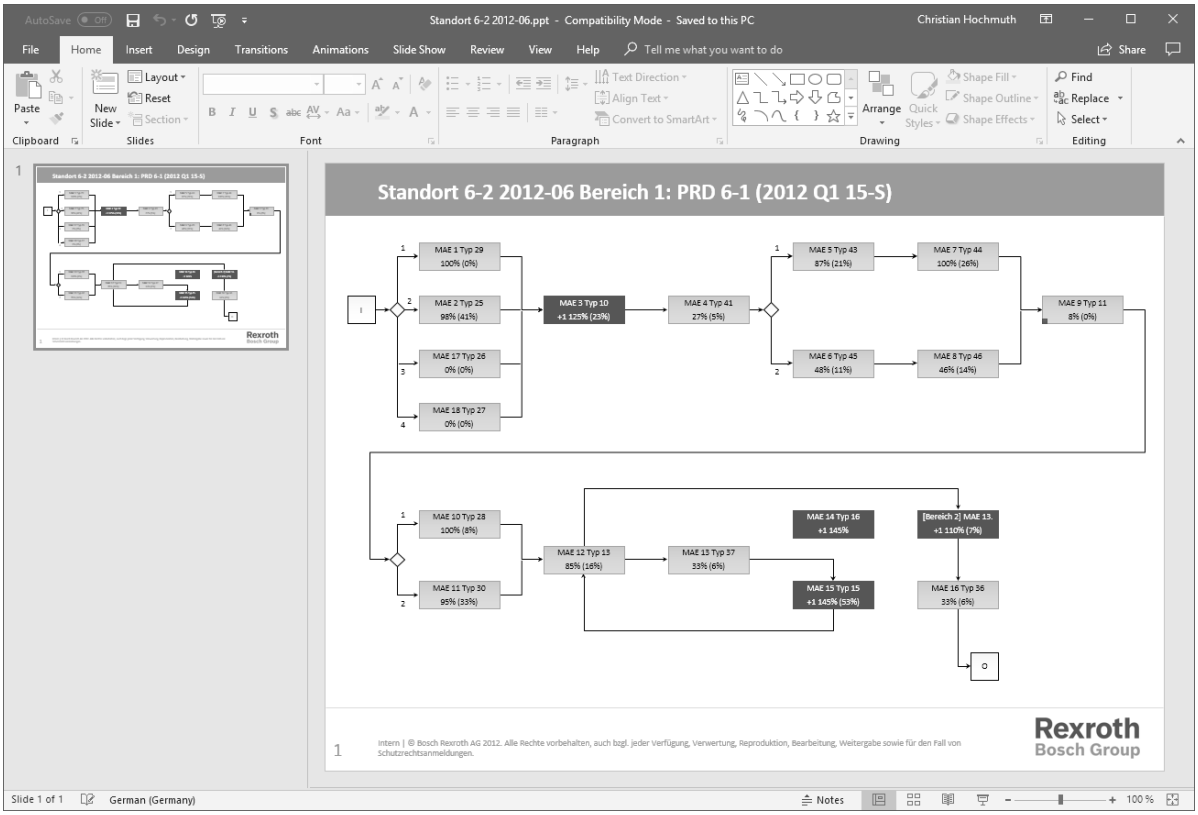


Abbildung 5.6: Bildschirmaufnahme von AURELIE (Export nach Microsoft PowerPoint). Wie die tabellarische Ergebnisdarstellung kann auch die grafische Übersicht der Auslastung in den Bereichen und Wertströmen zur Weiterverarbeitung exportiert werden. Der Export enthält die Auslastung der Ressourcen und ggf. deren Auslastung durch das ausgewählte Produkt. Die Funktion ermöglicht es, die Erstellung von Entscheidungsvorlagen für Investitionen zu automatisieren.

Im Prozess der Investitionsplanung muss zuerst der Bedarf an Investitionen ermittelt und auf dieser Basis eine Entscheidung durch die Unternehmensleitung herbeigeführt werden. Eine häufig wiederkehrende Aufgabe besteht darin, entsprechende Entscheidungsvorlagen zu erstellen, welche die Daten zur Auslastung der Ressourcen zusammenfassen. Hierfür bietet sich die grafische Darstellung eines Bereichs oder Wertstroms an, um zu verdeutlichen, an welcher Stelle und ggf. für welches Produkt zusätzliche Ressourcen benötigt werden. Um die Erstellung von Entscheidungsvorlagen zu automatisieren, ist es möglich, die grafische Darstellung nach Microsoft PowerPoint zu exportieren. Wie Abbildung 5.6 zeigt, schließt dies die Auslastung aller Ressourcen ein, wobei Engpässe hervorgehoben werden und ggf. die Auslastung durch das jeweilige Produkt angegeben wird.

5.5.2 Erfüllungsgrad der Anforderungen

In Kapitel 3 wurde festgehalten, dass keine Software existiert, welche die definierten Anforderungen in Bezug auf die Optimierung erfüllt. An diese Feststellung anknüpfend soll im Folgenden belegt werden, dass die im Rahmen der Arbeit entwickelte Software AURELIE alle

dieser Anforderungen umsetzt. Dies gilt für die Basisanforderungen A10–A13 im Hinblick auf die gegebenen Planungsziele und die Güte der Lösungen wie auch für die Leistungsanforderung A14$^+$ bezüglich der Laufzeit der Optimierung.

Erfüllungsgrad der Basisanforderung A10
Maximierung der Kapazitäten

Die Herausforderung im Hinblick auf die Maximierung der Kapazitäten bestand darin, vor Beginn der Optimierung eine Strategie festzulegen, welche die optimale Lösung eindeutig bestimmt. Betrachtet man die TEK verschiedener Produkte, ist es oftmals möglich, die TEK eines Produkts durch die Reduktion der TEK eines oder mehrerer anderer Produkte zu steigern. Der vorgestellte Algorithmus löst dieses Problem, indem die Stückzahlen der Produkte entsprechend dem Verhältnis der vorläufig festgelegten, geplanten Werte maximiert werden. Um die Produktstückzahlen zu maximieren, werden die verfügbaren Ressourcen bestmöglich genutzt. Sobald die Stückzahl eines Produkts ihren maximalen Wert erreicht, wird dieser Wert beibehalten, und die Stückzahlen der verbleibenden Produkte werden weiter erhöht. Durch die Vorgabe der geplanten Werte kann der Anwender vor Beginn der Optimierung festlegen, welche Lösung in dem Fall, dass mehrere nicht dominierte Lösungen vorliegen, ausgewählt werden soll. Die Basisanforderung A10 ist damit *erfüllt*.

Erfüllungsgrad der Basisanforderung A11
Minimierung der Investitionen

Analog zur vorigen Anforderung führt die Suche nach der minimalen Überlastung, die zur Herstellung der geplanten Produktstückzahlen notwendig ist, oft zu mehreren Lösungen. In vielen Fällen ist es möglich, die Überlastung einer Ressource zu reduzieren, indem im Gegenzug die Überlastung einer oder mehrerer anderer Ressourcen erhöht wird. Der Algorithmus, der in diesem Kapitel vorgestellt und in der Software implementiert wurde, führt zu einem Ausgleich der Überlastung zwischen den Ressourcen. Hierbei wird berücksichtigt, dass nur ausgewählte Ressourcen überlastet werden dürfen, wie durch den Anwender im grafischen Modell festgelegt. Genauer dürfen nur solche Prozessschritte zur Überlastung von Ressourcen führen, die nach den ersten ausgehenden Kanten alternativer Flusspunkte folgen. Durch die Anwendung dieser Strategie wird in dem Fall, dass mehrere nicht dominierte Lösungen existieren, stets eine eindeutig bestimmte, optimale Lösung ausgewählt. Als Ergebnis ist die Basisanforderung A11 *erfüllt*.

Erfüllungsgrad der Basisanforderung A12
Optimierung der Auslastung

Um die Auslastung der Ressourcen zu optimieren, müssen die Prozessstückzahlen von alternativ verknüpften Prozessschritten gemäß der vorgegebenen Iterationsvorschrift maximiert werden. Grundlage ist wie zuvor die Priorisierung ausgehender Kanten von alternativen Flusspunkten, von welcher eine vollständige Ordnung der zugeordneten Prozessstückzahlen abgeleitet werden kann. Wird der Weg zu zwei Kanten mit je einer Prozessstückzahl betrachtet, ist der erste alternative Flusspunkt entscheidend, an welchem verschiedene ausgehende

Standort*	Region	PS	Matrizen, Zeilen × Spalten				Laufzeit**, in s				
			P	**S**	**F**	**L**	PZ 1	PZ 2	PZ 3	Σ	Σ/PS
Standort 1	EMEA	391	36×446	36×36	137×445	**266×446**	8,76	**1,55**	1,88	12,18	0,031
Standort 2	EMEA	520	**57×617**	57×57	138×618	**394×617**	**23,18**	0,46	**4,72**	**28,36**	**0,055**
Standort 3/1	AMER	253	31×347	31×31	142×356	151×355	4,28	0,59	1,45	6,32	0,025
Standort 3/2	AMER	182	27×212	27×27	76×211	35×212	0,86	0,06	0,32	1,24	0,007
Standort 4	EMEA	490	62×480	**62×62**	199×486	137×486	15,32	0,02	1,31	16,66	0,034
Standort 5	EMEA	212	42×224	43×43	83×222	162×224	7,20	0,16	0,51	7,88	**0,037**
Standort 6/1	EMEA	299	55×227	55×55	98×226	72×227	2,83	0,01	0,15	2,99	0,010
Standort 6/2	EMEA	**730**	47×578	53×53	**239×597**	54×597	2,91	0,25	1,69	4,85	0,007
Standort 7	APAC	505	38×521	38×38	156×529	148×529	4,38	**2,28**	3,59	10,24	0,020
Standort 8	AMER	84	33×108	33×33	31×111	33×111	0,59	0,03	0,11	0,73	0,009
Standort 9/1	EMEA	**1404**	**152×1586**	**152×152**	**584×1603**	60×1603	**91,80**	**6,26**	**55,91**	**153,98**	**0,110**
Standort 9/2	EMEA	**1795**	**110×1390**	**119×119**	**509×1372**	**87×1390**	**23,51**	0,83	**17,67**	**42,01**	0,023

*anonymisiert **Testsystem: Microsoft Windows 8.1, 8 GB RAM, CPU Intel Core i5-4300M, vier Kerne mit 2,60 GHz

Tabelle 5.1: Erfüllung der vorgegebenen linearen oberen Laufzeitschranke. Aufgeführt sind die Laufzeiten zur Optimierung der drei Planungsziele (PZ 1–3) mit der Software AURELIE. Die Grundlage bilden die repräsentativen Planungen aus Tabelle 2.2 (Mittelwert von je zehn Messungen). Die Zahl der Prozessschritte (PS) und die Größen der Matrizen erlauben einen Rückschluss auf die Komplexität der Modelle. In keinem Fall wird die lineare obere Laufzeitschranke von 0,15 Sekunden je Prozessschritt überschritten (siehe Leistungsanforderung A14$^+$).

Kanten verfolgt werden. Die Prioritäten, welche diesen ausgehenden Kanten zugeordnet sind, bestimmen die Ordnung zwischen den zwei Prozessstückzahlen. Dadurch ist es dem Anwender möglich, mit den Mitteln des grafischen Modells eine eindeutige Priorisierung zwischen allen Prozessstückzahlen zu definieren. Der Algorithmus setzt die Iterationsvorschrift um und berücksichtigt dabei die minimale Überlastung, welche gemäß der vorigen Anforderung ermittelt wird. Folglich ist auch die Basisanforderung A12 *erfüllt*.

Erfüllungsgrad der Basisanforderung A13
Globales Optimum

Für den Einsatz im Unternehmen ist es unerlässlich, dass im Fall aller drei Planungsziele nicht nur ein lokales Optimum, sondern stets das globale Optimum gefunden wird. Die vorgestellten Algorithmen basieren auf der wiederholten Maximierung oder Minimierung linearer Zielfunktionen, jeweils beschränkt durch lineare Nebenbedingungen. Hierzu wird ein exaktes, mathematisches Verfahren der linearen Optimierung eingesetzt, welches durch die Bibliothek `lp_solve` bereitgestellt wird. Somit ist sichergestellt, dass stets das globale Optimum ermittelt wird, und die Basisanforderung A13 ist *erfüllt*.

Erfüllungsgrad der Leistungsanforderung A14$^+$
Lineare obere Laufzeitschranke

Um die Software effizient einsetzen zu können, darf zudem die Laufzeit der Optimierung im Verhältnis zur Zahl der Prozessschritte einen festgelegten Wert nicht überschreiten. Mit Blick auf die Komplexität typischer Modelle wird ein Schwellwert von 0,15 Sekunden je

Prozessschritt angenommen, wie in Abschnitt 2.6 erläutert wurde. Um die Anforderung zu prüfen, wurden die Laufzeiten für die Modelle der repräsentativ ausgewählten Planungen gemessen, die in Tabelle 2.2 auf Seite 38 aufgeführt sind. Die Ergebnisse der Messungen, welche in Tabelle 5.1 dargestellt sind, bestätigen, dass die definierte lineare obere Schranke für die Laufzeit in keinem der geprüften Fälle verletzt wird. Dies gilt auch dann, wenn man berücksichtigt, dass zur Messung ein leistungsfähigeres Rechnersystem genutzt wurde als in der Anforderung angegeben. Damit ist die Effizienz der Algorithmen aus praktischer Sicht belegt, und die Leistungsanforderung A14$^+$ ist entsprechend *erfüllt*.

5.5.3 Wesentliche Erweiterungen

Bei der Entwicklung der Software AURELIE wurden zahlreiche zusätzliche Anforderungen der Anwender umgesetzt, die im Pseudocode der Algorithmen nicht wiedergegeben sind. Hintergrund ist, dass es den Rahmen der Arbeit sprengen würde, auf diese Erweiterungen detailliert einzugehen. Dennoch sollen in aller Kürze einige wichtige Erweiterungen genannt werden, da diese zum Erfolg der Software beitragen.

(1) *Einbindung in ein Produktionsnetzwerk:* Im Fall weltweit tätiger Unternehmen ist ein Produktionsstandort zumeist in ein Netzwerk mit anderen Standorten desselben und anderer Unternehmen eingebunden. Mit der Software AURELIE können diese Standorte in der Planung der TEK als Kapazitäten und Bedarfe berücksichtigt werden. Hierzu ist es notwendig, die Optimierungsvariablen und die Nebenbedingungen gegenüber der Beschreibung in den vorigen Abschnitten zu erweitern.

(2) *Verkettung von Wertströmen:* Mit der Einführung einer verbrauchsorientierten Steuerung in der Produktion wie z. B. dem Bosch Production System werden die Wertströme von Komponenten und Produkten miteinander verkettet. In einem solchen Fall gilt die Regel, dass die Stückzahlen der Teilefertigung gerade dem Bedarf an Komponenten in der Montage entsprechen müssen. Um Regeln dieser Art abzubilden, müssen zusätzliche Nebenbedingungen eingeführt werden.

(3) *Entscheidungsstand der Investitionsplanung:* Ressourcen, die zum Zeitpunkt der Planung vorhanden oder zukünftig eingeplant sind, müssen von solchen unterschieden werden, deren Beschaffung noch zu genehmigen ist. Im Fall der letztgenannten Ressourcen werden Investitionen nicht erst durch deren Überlastung, sondern bereits durch deren Nutzung notwendig. Entsprechend muss die Bestimmung der Investitionen als eine Funktion der Auslastung angepasst werden.

(4) *Sicherstellung numerischer Stabilität:* Die endliche Repräsentation unendlicher Gleitkommazahlen führt zu Ungenauigkeiten, die sich im Verlauf der Optimierung verstärken. Besonders in Fällen komplexer Modelle ist nicht sichergestellt, dass eine Lösung gefunden wird, selbst wenn diese aus mathematischer Sicht existieren müsste. Verschiedene Maßnahmen wie z. B. die Umformulierung der Optimierungsprobleme bewirken, dass entsprechende Fälle praktisch ausgeschlossen werden.

(5) *Optimierung der Algorithmen:* Durch die Ausnutzung von Eigenschaften des grafischen Modells ist es möglich, die Algorithmen zu beschleunigen, wenn auch die angegebenen Komplexitätsklassen in Bezug auf die Laufzeit weiterhin gelten. Beispielsweise müssen Prozessstückzahlen nicht iterativ maximiert werden, wenn bereits feststeht, dass diese den jeweils größtmöglichen Wert annehmen. Der Einsatz solcher Techniken erlaubt es, die Zahl der Schleifendurchläufe um ca. 50 Prozent zu reduzieren.

5.5.4 Validierung der Optimierungsergebnisse

Anknüpfend an die Validierung des erstellten grafischen Modells in Abschnitt 4.3 besteht die Aufgabe, die nachfolgenden Schritte im Ablauf des softwaregestützten Workflows zu validieren. Dies betrifft die Transformation des grafischen Modells in ein entsprechendes mathematisches Modell und die Optimierung des mathematischen Modells gemäß den Planungszielen. Ergänzend zum theoretischen Nachweis der Korrektheit, der im vorliegenden Kapitel erbracht wurde, muss die Implementierung in der Software AURELIE getestet werden. Eine vollständige Prüfung der Ergebnisse wird jedoch dadurch erschwert, dass keine Software mit entsprechenden Möglichkeiten existiert, wie in Kapitel 3 belegt wurde. Da es folglich nicht möglich ist, die Ergebnisse mit verfügbarer Software zu vergleichen, müssen diese manuell nachvollzogen werden. Dabei muss ein Weg gefunden werden, um einzelne Softwarekomponenten sowie deren Integration zu testen.

Um dieses Problem zu lösen, wurde bei der Bosch Rexroth AG ein Vorgehen in drei Schritten angewandt, das sich über mehrere Monate erstreckte. Die Absolvierung eines Testschritts führte zum jeweils nächsten Schritt oder, im Fall des letzten Schritts, zur Erteilung der Freigabe für die Einführung der Software. Die Schritte waren dadurch gekennzeichnet, dass gegenüber den jeweils vorherigen der Umfang der Funktionen oder der Kreis der Teilnehmer erweitert wurde. Grundsätzlich entspricht dies den allgemeinen Vorgehensmodellen, die für die Einführung von Informationssystemen etabliert sind (siehe u. a. Liggesmeyer 2009, Ammann und Offutt 2016, Witte 2016). Diese Vorgehensmodelle basieren auf wiederholten Tests, die von einzelnen Komponenten einer Software zum Test übergreifender, integrierter Funktionen führen. Im Folgenden werden die einzelnen Testschritte, die im vorliegenden Fall ausgeführt wurden, näher erläutert:

(1) *Interner Test einzelner Softwarekomponenten:* Im ersten Schritt wurden die Matrizen und Folgen des mathematischen Modells und die Ergebnisse jedes einzelnen Iterationsschritts der Optimierung geprüft. Diese Prüfung wurde parallel zur Entwicklung der Software vom *Entwickler*, dem Autor dieser Arbeit, auf der Grundlage eines noch *nicht vollständig funktionsfähigen Prototyps* durchgeführt. Hierfür wurden die Möglichkeiten der Entwicklungsumgebung, in diesem Fall der Eclipse[3] Rich Client Platform (RCP), im Hinblick auf die Analyse des Programmcodes und das Debugging genutzt. Es wurden *einfache Modelle* erstellt, die analog zu den Beispielen in Kapitel 2 von ihrem Umfang abgesehen wesentliche Merkmale realer Modelle besitzen. Im Ergebnis konnte die *Korrektheit der Kernalgorithmen auf der Ebene einzelner Iterationsschritte* bestätigt werden.

[3] Eclipse: Eclipse Indigo SR2 Packages. Abgerufen am 9. Dezember 2018. http://www.eclipse.org/downloads/packages/release/Indigo/SR2.

(2) *Interner Integrationstest ausgewählter Softwarekomponenten:* Im zweiten Schritt folgte die Prüfung aller Schritte des softwaregestützten Workflows von der grafischen Modellierung bis zur Darstellung der Optimierungsergebnisse. Wie der vorherige Testschritt wurde auch dieser vom *Entwickler* der Software durchgeführt, jedoch auf Basis eines *funktionsfähigen Prototyps*. Zu diesem Zweck wurden die Bestandteile der Benutzeroberfläche genutzt, die zu diesem Zeitpunkt fertiggestellt waren, einschließlich des grafischen Konfigurators und der Ergebnisdarstellung. Wie zuvor wurden *einfache Modelle* erstellt, wobei allerdings nicht Zwischenergebnisse, sondern die Endergebnisse nach Durchlauf der Optimierung geprüft wurden. Damit war es möglich, die *Korrektheit der Kernalgorithmen auf übergeordneter Ebene* und der Benutzeroberfläche zu bestätigen.

(3) *Erweiterter Integrationstest aller Softwarekomponenten:* Im dritten und letzten Schritt vor der Einführung wurde die Software unter realistischen Bedingungen geprüft, welche dem zukünftigen Einsatz entsprechen. Hierzu wurden Pilotstandorte im Produktionsnetzwerk der Bosch Rexroth AG definiert, die stellvertretend das Produktspektrum und die Organisationsformen der Produktion repräsentieren. An den Pilotstandorten wurden Workshops organisiert, in denen die *zukünftigen Anwender* die Software selbst testen und ihr Feedback an den Entwickler weitergeben konnten. Die Grundlage bildete eine vorläufige Version der Software, die zu diesem Zeitpunkt bereits den *vollständigen Funktionsumfang* abdeckte. Um die Ergebnisse bezüglich Plausibilität zu prüfen und mit bestehenden Kalkulationen zu vergleichen, wurden *komplexe, realistische Modelle* erstellt. Als Voraussetzung für die Einführung konnte die *Korrektheit aller Algorithmen* bestätigt werden.

Nach der erfolgreichen Absolvierung des letzten Testschritts wurde die Software AURELIE an allen Standorten der Bosch Rexroth AG eingeführt. Die Anwender wurden im Rahmen von Workshops geschult, und ihnen wurde die Möglichkeit für Feedback gegeben, welches durch Anpassung und Erweiterung der Software umgesetzt wurde. In der Praxis wurde die Software fortdauernd getestet, teilweise mit Modellen, die bezüglich Komplexität und Umfang über den ursprünglichen Einsatzzweck hinausgingen. Beispielsweise bildeten manche Anwender neben Maschinen alle Werkzeuge ab, die in Verbindung mit Maschinen genutzt werden können. Zwar erhöhte sich die Laufzeit durch den deutlich gestiegenen Modellumfang, jedoch waren die Stabilität der Software und die Korrektheit der Ergebnisse weiter gewährleistet. Abschließend kann festgestellt werden, dass die Kernalgorithmen zur Modelltransformation und Optimierung erfolgreich in der Praxis validiert wurden.

5.6 Fazit: Erreichen des vorgegebenen Entwicklungsziels

Die Bewertung des Stands der Technik führte zum Schluss, dass keine Software existiert, mit welcher die Planungsziele im gegebenen Kontext vollständig erreicht werden. Unter den evaluierten Softwaretypen bietet keiner die Möglichkeit, wie vorgegeben die Kapazitäten zu maximieren, die Investitionen zu minimieren und die Auslastung zu optimieren. Diesbezüglich wurde gefordert, dass die Lösung eindeutig bestimmt ist und stets die bestmögliche Lösung innerhalb einer definierten Laufzeit gefunden wird. Nachdem im vorigen Kapitel

die Modellierung betrachtet wurde, sollten nun in einem zweiten Lösungsschritt die Anforderungen in Bezug auf die Optimierung erfüllt werden. Dabei galt es, die zuvor erreichten Ergebnisse und insbesondere die mathematische Repräsentation des Modells aufzugreifen und weiterzuverarbeiten. Mit Blick auf die gewonnenen Erkenntnisse ist festzustellen, dass das Entwicklungsziel durch die neue Software AURELIE erreicht wird. Zum Abschluss des Kapitels liegen folgende Ergebnisse vor:

(1) Algorithmus zur *Maximierung der Kapazitäten* (technische Kapazitäten $\mathbf{y}_{\max}$): Beschreibung von Ziel, Grundidee, Datenstrukturen und Ablauf sowie Beweis der Korrektheit und der Zeitkomplexität (siehe Abschnitt 5.2 sowie Abschnitt A.2.2 im Anhang);

(2) Algorithmus zur *Minimierung der Investitionen* (Komponenten des Vektors $\mathbf{Lx}$ größer als eins): detaillierte Beschreibung und formale Beweise analog zum ersten Algorithmus (siehe Abschnitt 5.3 sowie Abschnitt A.2.3 im Anhang);

(3) Algorithmus zur *Optimierung der Auslastung* (alle Komponenten des Vektors $\mathbf{Lx}$): detaillierte Beschreibung und formale Beweise analog zum ersten Algorithmus (siehe Abschnitt 5.4 sowie Abschnitt A.2.4 im Anhang);

(4) Beschreibung der *Umsetzung* in der Software AURELIE: Funktionsübersicht und Benutzerführung, Beleg der Erfüllung aller Anforderungen, wesentliche Erweiterungen und Validierung der Optimierungsergebnisse (siehe Abschnitt 5.5).

Das zurückliegende Kapitel vervollständigt den Hauptteil dieser Arbeit. Mit dem zweiten Lösungsschritt wurde die noch fehlende Komponente einer geeigneten Software zur strategischen Planung der TEK vorgestellt: der Optimierer, welcher an das Modell gekoppelt wird. Nach der Optimierung des Modells folgen dessen Auswertung und Anpassung durch den Anwender, womit sich der softwaregestützte Workflow zur Planung schließt. Die Konzepte wurden in der Software AURELIE umgesetzt und finden seitdem an den Standorten der Bosch Rexroth AG weltweit im Unternehmensalltag Anwendung.

6 Schluss

Der Verlauf der Arbeit führt von einem praktischen Problem über eine grundlegende Analyse und die Bewertung existierender Lösungen zur Entwicklung einer neuen Lösung. In Abschnitt 6.1 sollen wesentliche Ergebnisse der einzelnen Kapitel zusammengefasst und wichtige Punkte hervorgehoben werden. Darauf folgen in Abschnitt 6.2 einige Implikationen für Forschung und Praxis, entsprechend dem Charakter der Dissertation, welche diese beiden Perspektiven verbindet. Abschnitt 6.3 bildet den Schlusspunkt in Gestalt möglicher Weiterentwicklungen in Bezug auf das vorgestellte Grundkonzept.

6.1 Zusammenfassung der Ergebnisse

Der Lösungsansatz, der zur strategischen Planung der technischen Kapazität in der Serienfertigung vorgeschlagen wurde, basiert auf einem zyklischen, *softwaregestützten Workflow*, der entsprechend der Abbildung 1.2 auf Seite 5 wie folgt abläuft:

(1) grafische Modellierung eines Systems von Wertströmen, welches alle Prozessschritte zur Herstellung der Produkte an einem Standort zusammenfasst;
(2) automatische Transformation des grafischen Modells in ein entsprechendes mathematisches Modell, welches sich zur Optimierung eignet;
(3) automatische Optimierung des zuvor erzeugten mathematischen Models gemäß den definierten Planungszielen mit Hilfe von exakten Verfahren;
(4) Auswertung der Ergebnisse durch den Planer, insbesondere der ggf. verbleibenden Engpässe, und Anpassung der Prämissen.

In dieser Arbeit wurden die kalkulatorischen Grundlagen für ein System von Wertströmen dargelegt, aufbauend auf der Struktur und der Schnittstelle eines solchen Systems. Bezug nehmend darauf wurden die notwendigen Kernalgorithmen zur grafischen Modellierung, Modelltransformation und mathematischen Optimierung beschrieben. Die zu Beginn identifizierten Potenziale werden durch die weitgehende Automatisierung der oben genannten Schritte geschöpft, die in der Software AURELIE umgesetzt ist.

Mit einer Systemanalyse wurde in Kapitel 2 die zu entwickelnde Lösung vorbereitet, beginnend mit einer Abgrenzung der Planung im gegebenen *Kontext*. Insbesondere wurde die Planung, wie sie in dieser Arbeit betrachtet wird, von der Planung der Personalkapazität, der operativen Planung und der Einzelfertigung unterschieden. Dabei werden drei Planungsziele verfolgt: die Maximierung der Kapazitäten, die Minimierung der Investitionen und die Optimierung der Auslastung. Darauf folgte die Definition der *Systemstruktur*, beruhend auf der rekursiven Verknüpfung von Prozessschritten eines Produkts zu Prozessen und Wertströmen. Zur eindeutigen Beschreibung des Produktionsablaufs wurden die

sequenziellen, alternativen und selektiven Verknüpfungstypen eingeführt. Auf dieser Basis wurde die *Systemschnittstelle* charakterisiert, welche dem Austausch von Eingaben und Ausgaben zwischen dem jeweiligen System und dessen Umwelt dient. Wie erläutert wurde, lassen sich die Planungsziele als Zielfunktionen und Nebenbedingungen der Eingaben und Ausgaben des Informationsflusses beschreiben. Im Anschluss daran wurden die *Grundlagen der Kalkulation* von Kapazität, Auslastung und Investitionen eingeführt, zunächst begrenzt auf den Fall eines einzigen Prozessschritts. Dem schloss sich folgerichtig die *Erweiterung der Kalkulation* für den allgemeinen Fall rekursiv verknüpfter Prozessschritte eines Produkts an. Die Systemanalyse mündete in der Formulierung essenzieller *Anforderungen* an eine Software, unterschieden nach Basisanforderungen und Leistungsanforderungen. Damit wurden die Voraussetzungen zur Bewertung des Stands der Technik und zur Entwicklung der Kernalgorithmen geschaffen.

Gegenstand von Kapitel 3 war die *Bewertung* kommerzieller und frei verfügbarer Software in Bezug auf die Erfüllung der formulierten Anforderungen. Hierzu wurden Softwaretypen ausgewählt, welche die Abbildung eines Systems ermöglichen und in Unternehmen mit Blick auf die nötige Qualifikation der Mitarbeiter einsetzbar sind: Software zur Erstellung von Tabellenkalkulationen, zur Materialflusssimulation, für Supply Chain Management und zur Prozessmodellierung. Für die Bewertung wurde jeweils ein repräsentativer Vertreter dieser Softwaretypen herangezogen: Microsoft Excel, Siemens Plant Simulation, SAP APO SNP bzw. BPMN mit einem idealen Interpreter und Optimierer. Als Ergebnis ist festzustellen, dass zum Zeitpunkt der Betrachtung kein Softwaretyp existierte, welcher allen zuvor formulierten Anforderungen genügt. Zudem werden die Anforderungen zur grafischen Modellierung mit Hilfe einer minimalen Symbolmenge und zur Optimierung von keinem Softwaretyp vollständig erfüllt. Hiervon wurde der Bedarf abgeleitet, eine Software zur Modellierung und Optimierung eines Systems von Wertströmen zu entwickeln.

Mit Kapitel 4 wurde der erste Lösungsschritt unternommen, die grafische Modellierung eines Systems von Wertströmen durch sogenannte *Wertstromgraphen*. Zu diesem Zweck wurde der bestehende Begriff eines gerichteten Graphen mit Mehrfachkanten erweitert und ein formal eindeutiges, grafisches Modell beschrieben. Begründet wurde die Weiterentwicklung durch die Ableitung der erforderlichen Eigenschaften von Knoten und Kanten eines geeigneten Graphen von den formulierten Anforderungen. Als Voraussetzung für die nachfolgende Verarbeitung wurde ein Algorithmus zur *Validierung* eines grafischen Modells entwickelt. Daran anknüpfend wurde ein mathematisches Modell eingeführt, welches auf Matrizen und Folgen basiert und sich zur Optimierung gemäß den Planungszielen eignet. Weiterhin wurde ein Algorithmus zur *Transformation* eines grafischen Modells in ein mathematisches Modell entwickelt. Die beiden Algorithmen wurden detailliert im Hinblick auf Ziel, Grundidee, Datenstrukturen und Ablauf beschrieben, und es wurde die Zeitkomplexität nachgewiesen. Zuletzt wurde die *Umsetzung* der entwickelten Algorithmen in der Praxis betrachtet und die Benutzerführung am Beispiel der Software AURELIE beschrieben. Als Ergebnis wurde festgehalten, dass erstmalig alle Anforderungen in Bezug auf die Modellierung eines Systems von Wertströmen erfüllt sind.

In Kapitel 5 folgte der zweite Lösungsschritt, die Optimierung des zuvor erzeugten mathematischen Modells gemäß den definierten Planungszielen. Den Algorithmen wurde jeweils eine Diskussion und die Auswahl eines Ansatzes zur Erfüllung der Anforderungen

vorangestellt, insbesondere zur eindeutigen Bestimmung des Ergebnisses. Dieser Ansatz besteht im Fall der *Maximierung der Kapazitäten* darin, die Stückzahlen aller Produkte im Verhältnis der vorläufigen geplanten Stückzahlen zu steigern. Im Fall der *Minimierung der Investitionen* ist diese Eindeutigkeit durch den Ausgleich der Überlastung zwischen den jeweiligen Ressourcen erreichbar. Mit Blick auf die *Optimierung der Auslastung* ist festzustellen, dass das Ergebnis der Iterationsvorschrift gemäß der Anforderung bereits eindeutig bestimmt ist. Für jedes Planungsziel wurden Algorithmen zur iterativen Maximierung oder Minimierung geeignet definierter Zielfunktionen entwickelt. In diesem Zuge wurden Ziel, Grundidee, Datenstrukturen und Ablauf der Algorithmen beschrieben sowie die Korrektheit und Zeitkomplexität nachgewiesen. Danach richtete sich der Blick auf die *Umsetzung* am Beispiel der Software AURELIE, insbesondere in Bezug auf die Benutzerführung und einige hervorzuhebende Erweiterungen. Nach diesem Kapitel sind erstmals alle Anforderungen im Hinblick auf die Optimierung eines gegebenen Modells erfüllt.

Zum Abschluss liegen die Kernalgorithmen für die zwei Komponenten einer Software vor, die dazu geeignet ist, den vorgeschlagenen Workflow zu unterstützen: ein Modellierer zur grafischen Erstellung und automatischen Umwandlung eines Modells und ein Optimierer zur automatischen Suche der Planungsziele.

6.2 Implikationen für Forschung und planerische Praxis

Aus Sicht der *Forschung* leistet diese Arbeit Beiträge zur Weiterentwicklung von theoretischen Methoden, welche zur Lösung des gegebenen praktischen Problems angewandt werden. Zu Beginn wurde ein Formalismus entwickelt, um die Struktur und die Schnittstelle eines Systems von Wertströmen eindeutig zu beschreiben. Hierzu wurden zueinander schlüssige Definitionen der Begriffe des Prozessschritts, des Prozesses und des Wertstroms geprägt und in ein System eingeordnet. Mit Bezug auf die Planungsziele wurden die normativen Grundlagen zur Kalkulation zusammengefasst und zur Berücksichtigung der existierenden Komplexität erweitert. Anschließend wurden die Anforderungen an eine Software zur effizienten, transparenten und flexiblen Planung der technischen Kapazität formuliert. Mit Hilfe dieser Anforderungen konnte durch den Beleg, dass zum Zeitpunkt der Betrachtung keine geeignete Software existierte, die Forschungslücke aufgezeigt werden.

Um diese Forschungslücke zu schließen, wurde der theoretische Begriff des Graphen zu einem Wertstromgraphen mit spezifischen Knoten und Kanten weiterentwickelt. Der Begriff bildete die Basis für die Formalisierung eines grafischen Modells und die Entwicklung von Algorithmen zur Validierung und Transformation in ein mathematisches Modell. Im Hinblick auf die Optimierung fand eine Bewertung möglicher Ansätze statt, mit denen sichergestellt werden kann, dass die gefundenen Lösungen eindeutig bestimmt sind. Darauf folgte eine entsprechende Formalisierung der Planungsziele und die Entwicklung von Algorithmen zur Optimierung eines solchen mathematischen Modells. Als Ergebnis konnte die identifizierte Forschungslücke geschlossen werden, wobei die Umsetzbarkeit der beschriebenen Lösung am Beispiel der Software AURELIE belegt wurde.

Aus Sicht der *Praxis* wurde das Grundkonzept der Software AURELIE vorgestellt, deren Einsatz die Effizienz, Transparenz und Flexibilität im Planungsprozess steigert. Als Bestätigung

dient die Tatsache, dass nach ihrer Einführung bei der Bosch Rexroth AG die notwendige Zeit zur Planung bei erster Modellierung um 25 Prozent reduziert werden konnte. Bei wiederholter Modellierung zur Anpassung der Planung konnte eine noch größere Zeitreduktion um 50 bis 75 Prozent erzielt werden. Wie in der Arbeit erläutert, ist die strategische Planung technischer Kapazität typischerweise die Aufgabe eines oder mehrerer Planer an jedem Standort. Im Fall größerer Standorte beansprucht die Erstellung der Kalkulationen insgesamt die Arbeitszeit einer Person je Standort. Durch die geringere Bindung von wichtigen Ressourcen steigt die Effizienz, und durch die Visualisierung von Engpässen wird gleichzeitig die Transparenz erhöht. Die Flexibilität wächst durch die Nutzung eines allgemeinen, grafischen Modells, welches die Abbildung spezifischer Abläufe ermöglicht.

Die Implikation daraus lautet jedoch nicht, dass durch den geringeren Ressourcenbedarf als Folge des Einsatzes einer neuen Software Personal reduziert werden soll. Vielmehr kann die Arbeitszeit der Planer effektiver genutzt werden, da ein größerer Zeitanteil zur Verfügung steht, um die Wertströme für die zukünftige Produktion zu gestalten. Dies eröffnet einem Unternehmen die Möglichkeit, abgeleitet von der strategischen Auswahl der Zielmärkte und dem erwartetem Absatz in jedem Markt die Produktion langfristig auszurichten. Anders formuliert ist dies die Chance, die Wettbewerbsfähigkeit des Unternehmens in einem zunehmend kompetitiven Umfeld nachhaltig zu steigern.

Es sei an dieser Stelle angemerkt, dass die Entwicklung der Software AURELIE wesentlich mehr umfasste, als in dieser Arbeit beschrieben werden konnte. Beispielsweise wird der Anwender in der grafischen Modellierung unterstützt, indem Kanten zur Verbindung von Knoten automatisch erzeugt werden. Hierbei wird sichergestellt, dass die erzeugten Kanten keine anderen Knoten kreuzen, in der grafischen Darstellung die kürzestmögliche Länge besitzen und möglichst geradlinig verlaufen. Zahlreiche weitere Funktionen führten dazu, dass die Anwender nach einer ersten, eintägigen Schulung innerhalb weniger Tage alle existierenden Wertströme an ihrem Standort abbilden konnten.

6.3 Ausblick: mögliche Weiterentwicklungen

Zusätzliche Potenziale existieren im Hinblick auf die *Integration* der strategischen Planung technischer Kapazität mit angrenzenden Planungen. Diesbezüglich sind die Planung der Personalkapazität und die operative Planung beider Produktionsfaktoren, der technischen und der Personalkapazität, hervorzuheben. In einer teilautomatisierten Produktion kann die Planung technischer Kapazität als Grundlage dienen, um von der Auslastung der Produktionsanlagen den Personalbedarf abzuleiten. Die strategische Planung kann durch Verkürzung des Zeithorizonts und Ergänzung detaillierter Informationen zur operativen Planung verdichtet werden. Theoretisch ist eine parallele Planung beider Produktionsfaktoren mit strategischem und operativem Zeithorizont durch die Integration in einer Software denkbar. Dies würde jedoch Modellerweiterungen erfordern und das Risiko bergen, die Komplexität zu erhöhen und die Effizienz zu reduzieren. Aus praktischer Sicht erscheint ein sequenzieller Ablauf der Planungen vorteilhafter, wofür Schnittstellen der Softwareanwendungen entwickelt werden müssten. Als Ergebnis könnte eine effektivere und effizientere Abstimmung der jeweiligen Planungsfunktionen erreicht werden.

Die angesprochene Integration schließt die *Verarbeitung* der erzielten Ergebnisse in nachfolgenden Prozessen ein. Beispielsweise ist aufgrund der Komplexität in vielen Fällen nicht bekannt, welche Ressourcen einen Engpass darstellen und die Stückzahlen der Produkte beschränken. Da Produktionsanlagen häufig für mehrere Produkte genutzt werden, führen Verschiebungen im Verhältnis der Stückzahlen dazu, dass die Engpässe zwischen verschiedenen Ressourcen wechseln. Im Zuge der Minimierung der Überlastung durch die Software AURELIE werden Engpässe in den Wertströmen identifiziert und visualisiert. An diesen Punkt anknüpfend könnte die Software dahingehend weiterentwickelt werden, ein vom Planer erstelltes Modell in Bezug auf Potenziale zur Prozessoptimierung zu prüfen. Als Ergebnis könnten automatisiert Maßnahmen zur Reduktion der Rüstzeiten und der Stillstandszeiten vorgeschlagen werden. Die Erkenntnisse aus der strategischen Planung könnten somit systematisch genutzt werden, um die operativen Prozesse in der Produktion zu optimieren. Voraussetzung wäre die Abbildung von Expertenwissen aus den operativen Bereichen wie z. B. der Arbeitsplanung und der Instandhaltung.

Mit Blick auf den softwaregestützten Workflow ist es darüber hinaus denkbar, den Umfang der *automatisierten Entscheidungen* zu erweitern. Laut dem vorgestellten Konzept der Software AURELIE werden alle quantitativen Informationen durch den Planer in einem grafischen Modell abgebildet. Hierbei berücksichtigt der Planer qualitative Randbedingungen, wie z. B. den Ablauf von Fertigung und Montage, durch die Definition der Prozesse und Wertströme. Auf dieser Basis folgen die automatische Transformation und Optimierung des erstellten Modells zur Unterstützung quantitativer Entscheidungen. Diese Entscheidungen betreffen ausschließlich numerische Freiheitsgrade wie z. B. die Verteilung der Stückzahlen und die Zahl zusätzlicher Ressourcen an definierten Stellen in den Wertströmen. Qualitative Entscheidungen wie z. B. die Festlegung, an welcher Stelle Ressourcen beschafft werden sollen, liegen weiterhin in Verantwortung des Planers. Mit den Fortschritten der künstlichen Intelligenz ist es grundsätzlich vorstellbar, auch diese Entscheidungen zu automatisieren. Zur Erreichung dieses Ziels wäre zuallererst die Abbildung aller notwendigen quantitativen und qualitativen Informationen in einem Modell erforderlich.

Eine solche Automatisierung hätte weitreichende Konsequenzen, wie an der Frage deutlich wird, in zusätzliche Produktionsanlagen zu investieren oder Schichtmodelle zu erweitern. Die Abbildung aller notwendigen qualitativen Informationen wäre jedoch schwierig, insbesondere im Fall personalrelevanter Fragen. Im genannten Beispiel ist u. a. der Faktor einzubeziehen, unter welchen Bedingungen eine Einigung mit der Personalvertretung zur Erweiterung der Arbeitszeit erzielbar ist. Gelänge dies aber, dann wäre der softwaregestützte Workflow zu hinterfragen, da der vollständige Ersatz des Planers durch Algorithmen möglich erschiene. Eine Voraussetzung für diese Automatisierung bestünde allerdings darin, dass Entscheidungen, die durch Algorithmen getroffen werden, im Unternehmen akzeptiert werden. Das gilt insbesondere in dem Fall, dass die getroffenen Entscheidungen maßgeblich die zukünftige Ausrichtung des Unternehmens bestimmen. Aus diesen Gründen erscheint eine Teilautomatisierung der Planung mittelfristig aussichtsreicher, weshalb der Workflow, wie in dieser Arbeit vorgeschlagen, weiterhin Relevanz besitzt. Fest steht jedoch, dass die Fortschritte künstlicher Intelligenz beobachtet und, wann immer sinnvoll, zur Verbesserung der Planung in Unternehmen genutzt werden sollten.

Anhang A

Technische Dokumentation

Die Kernalgorithmen zur Modellierung und Optimierung stellen einen wesentlichen wissenschaftlichen Beitrag der vorliegenden Arbeit dar. Implementiert wurden sie in der Software AURELIE, welche mehr als 60 000 Zeilen Programmcode umfasst. Selbst wenn man die Betrachtung auf die Kernalgorithmen beschränkt, entspricht dies noch mehr als 2000 Zeilen. Daher scheidet eine ungekürzte Wiedergabe der Algorithmen im Rahmen dieser Arbeit aus. Um eine verständliche und umfassende Beschreibung zu erreichen, werden sie stattdessen auf zwei Abstraktionsebenen wiedergegeben. Im Hauptteil wurden die Algorithmen zum Zweck der Einführung in natürlicher Sprache zusammengefasst. Darauf folgt nun jeweils eine ausführlichere Fassung in formalem, kommentiertem Pseudocode.

Die hierzu verwendete Form des Pseudocodes ist unabhängig von der Implementierung in einer Programmiersprache. Im Unterschied zu den Kurzfassungen im Hauptteil wird eine mathematische Beschreibung mit Kommentaren gewählt, um den Ablauf eindeutig und verständlich wiederzugeben. Dabei wird auf Datenstrukturen verzichtet, die nur in bestimmten Programmiersprachen zur Verfügung stehen. Die jeweils genutzten Objekte und Variablen sowie Funktionen und Prozeduren werden jedem Algorithmus vorangestellt. Mit diesen Informationen ist es möglich, die Algorithmen in jeder geeigneten Programmiersprache wie z. B. C++, Python oder Java[1] nachzuimplementieren.

Bemerkung: Werden in den folgenden Funktionen und Prozeduren Objekte oder Matrizen als Argumente definiert, dann wird vorausgesetzt, dass nicht Werte, sondern Referenzen auf die Objekte bzw. Matrizen übergeben werden. Als Folge wirken sich Änderungen der Werte auch außerhalb der aufgerufenen Funktion bzw. Prozedur aus.

A.1 Algorithmen, Teil I: grafische Modellierung und Modelltransformation

Der erste Teil umfasst die Algorithmen zur Traversierung, Validierung und Transformation von Graphen und grafischen Modellen. Zur Einführung werden die Standardverfahren der Breitensuche und Tiefensuche wiedergegeben. Danach wird die nichtrekursive Implementierung der Tiefensuche für den vorliegenden Anwendungsfall weiterentwickelt.

[1] Die Implementierung der Algorithmen in der Software AURELIE erfolgte auf Basis der Java Platform Standard Edition 7. Für Informationen siehe folgende Adresse (abgerufen am 11. Februar 2018): `http://www.oracle.com/technetwork/java/javase/overview/`.

A.1.1 Nichtrekursive Breitensuche von Knoten in einem Graphen *G*

Die Breitensuche ist ein Standardverfahren zur Traversierung der Knoten V eines allgemeinen, in diesem Fall endlichen und gerichteten Graphen G. Ausgehend von einem gegebenen Knoten σ werden alle Knoten des Graphen besucht, die erreicht werden können, indem die jeweils anliegenden Kanten zu benachbarten Knoten weiterverfolgt werden. Charakteristisch für die Breitensuche ist, dass immer erst alle Knoten besucht werden, die vom jeweils betrachteten Knoten unmittelbar erreichbar sind. Wurden all diese Knoten besucht, setzt die Suche mit deren Nachfolgern fort. In der Implementierung wird hierzu eine nach FIFO verwaltete *Warteschlange* verwendet (siehe Abschnitt 4.1).

Objekte und Variablen — Zeilennummer

Objekt *Warteschlange*: $\{(v_1, v_2, \ldots) : v \in V\}$ 04

Linear verkettete Liste variabler Länge, welche nach FIFO verwaltet wird. Das heißt, zuerst hinzugefügte Elemente werden zuerst entfernt und zurückgegeben (siehe Abschnitt 4.1). Jedes Element der *Warteschlange* ist ein Knoten v aus der Menge V des zu traversierenden Graphen G.

Funktionen und Prozeduren — Zeilennummer

Funktion ErmittleFarbe(v): {`weiß`, `schwarz`} 10

Liefert die Farbe des Knotens v aus der Menge V des zu traversierenden Graphen G zurück. Genau dann, wenn der Knoten v bereits entdeckt wurde, ist dessen Farbe `schwarz`, sonst `weiß`.

Funktion ErmittleNächsteKnoten(G, v): $\{\{v_1, v_2, \ldots\} : v \in V\}$ 09

Liefert die Menge aller Knoten v' zurück, welche unmittelbar erreicht werden, indem die ausgehenden Kanten des Knotens v im Graphen G weiterverfolgt werden.

Prozedur LegeFarbeFest(v, `weiß` oder `schwarz`) 03

Legt die Farbe des Knotens v aus der Menge V des zu traversierenden Graphen G fest, wodurch dieser als noch nicht entdeckt bzw. entdeckt markiert wird. Siehe Funktion ErmittleFarbe(v).

Prozedur NichtrekursiveBreitensuche(G, σ) 01

Traversiert all jene Knoten V eines Graphen G, die ausgehend vom Knoten σ erreichbar sind, indem alle aufeinanderfolgenden Kanten beschritten werden. Die Suche wird fortgesetzt, solange Kanten zu Knoten v führen, die noch nicht entdeckt wurden. Unmittelbar erreichbare Knoten v werden dabei zuerst besucht. Als Ergebnis werden die besuchten Knoten v `schwarz` gefärbt.

Funktion *Warteschlange*.EntferneErstes(): V 08

Entfernt das zuerst hinzugefügte Element der *Warteschlange* und liefert dieses zurück, wie definiert ein Knoten v des zu traversierenden Graphen G. Siehe Erläuterung zum Objekt *Warteschlange*.

Prozedur *Warteschlange*.ErgänzeAlsLetztes(v) 06

Fügt den jeweils übergebenen Knoten v aus der Menge V des zu traversierenden Graphen G an das Ende der *Warteschlange*. Siehe Erläuterung zum Objekt *Warteschlange*.

Funktion *Warteschlange*.IstLeer(): {`falsch`, `wahr`} 13

Liefert genau dann `wahr` zurück, wenn die Länge der *Warteschlange* gleich null ist, d. h., wenn diese einem leeren Tupel entspricht, sonst `falsch`. Siehe Erläuterung zum Objekt *Warteschlange*.

Für mathematische Symbole siehe Abschnitt 4.1 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A4 dargestellt.

A.1.2 Rekursive Breitensuche von Knoten in einem Graphen G

Die rekursive Implementierung der Breitensuche entspricht in Bezug auf Ablauf und Ergebnis der nichtrekursiven Variante (siehe Abschnitt A.1.1). Beide verwenden zur Verwaltung zu besuchender Knoten V eine *Warteschlange*, welche nach FIFO verwaltet wird. Der Grund ist, dass nur mit einer solchen Datenstruktur die charakteristische Eigenschaft der Breitensuche umsetzbar ist, immer erst alle unmittelbar erreichbaren Knoten zu besuchen. Folglich beschränkt sich der rekursive Ansatz darauf, die *Warteschlange* nicht in einer Schleife, sondern durch verschachtelten Aufruf einer Hilfsprozedur zu bearbeiten.

Objekte und Variablen	Zeilennummer
Objekt *Warteschlange*: $\{(v_1, v_2, \ldots): v \in V\}$	04
Siehe Abschnitt A.1.1.	

Funktionen und Prozeduren	Zeilennummer
Funktion ErmittleFarbe(v): {`weiß`, `schwarz`}	11
Siehe Abschnitt A.1.1.	
Funktion ErmittleNächsteKnoten(G, v): $\{\{v_1, v_2, \ldots\}: v \in V\}$	10
Siehe Abschnitt A.1.1.	
Prozedur InnereRekursiveBreitensuche$(G,$ *Warteschlange*$)$	07
Die Hilfsprozedur wird wiederholt aufgerufen, um den rekursiven Gedanken umzusetzen, wobei analog zur nichtrekursiven Implementierung eine *Warteschlange* verwendet wird.	
Prozedur LegeFarbeFest$(v,$ `weiß` oder `schwarz`$)$	03
Siehe Abschnitt A.1.1.	
Prozedur RekursiveBreitensuche(G, σ)	01
Der Ablauf und das Ergebnis der Hauptprozedur entsprechen der nichtrekursiven Implementierung. Siehe Prozedur NichtrekursiveBreitensuche(G, σ) in Abschnitt A.1.1.	
Funktion *Warteschlange*.EntferneErstes(): V	09
Siehe Abschnitt A.1.1.	
Prozedur *Warteschlange*.ErgänzeAlsLetztes(v)	06
Siehe Abschnitt A.1.1.	
Funktion *Warteschlange*.IstLeer(): {`falsch`, `wahr`}	14
Siehe Abschnitt A.1.1.	

Für mathematische Symbole siehe Abschnitt 4.1 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A4 dargestellt.

```
    Eingabe: Graph G mit Knoten V und Kanten E, zuerst zu besuchender Knoten σ ∈ V
    Ausgabe: gefärbte Knoten V
01: Prozedur NichtrekursiveBreitensuche(G,σ)
02:     für alle v ∈ V tue
03:         LegeFarbeFest(v,weiß)  /* initialisiere die Knoten V als nicht entdeckt */
04:     Warteschlange ← ()  /* initialisiere die Warteschlange, eine Folge von Knoten v */
05:     LegeFarbeFest(σ,schwarz)  /* markiere Knoten σ als entdeckt und ... */
06:     Warteschlange.ErgänzeAlsLetztes(σ)  /* ... besuche diesen zuallererst */
07:     wiederhole
08:         v ← Warteschlange.EntferneErstes()  /* betrachte alle nachfolgenden Knoten v' ... */
09:         für alle v' ∈ ErmittleNächsteKnoten(G,v) tue  /* ... des aktuellen Knotens v */
10:             wenn ErmittleFarbe(v') = weiß dann  /* wurde Knoten v' noch nicht entdeckt, ... */
11:                 LegeFarbeFest(v',schwarz)  /* ... markiere diesen entsprechend und ... */
12:                 Warteschlange.ErgänzeAlsLetztes(v')  /* ... besuche ihn im Anschluss */
13:     solange ¬Warteschlange.IstLeer()  /* wiederhole, solange noch Knoten zu besuchen sind */
```

Algorithmus A.1.1: Nichtrekursive Breitensuche von Knoten in einem Graphen *G*.

```
    Eingabe: Graph G mit Knoten V und Kanten E, zuerst zu besuchender Knoten σ ∈ V
    Ausgabe: gefärbte Knoten V
01: Prozedur RekursiveBreitensuche(G,σ)
02:     für alle v ∈ V tue
03:         LegeFarbeFest(v,weiß)  /* initialisiere die Knoten V als nicht entdeckt */
04:     Warteschlange ← ()  /* initialisiere die Warteschlange, eine Folge von Knoten v */
05:     LegeFarbeFest(σ,schwarz)  /* markiere Knoten σ als entdeckt und ... */
06:     Warteschlange.ErgänzeAlsLetztes(σ)  /* ... besuche diesen zuallererst */
07:     InnereRekursiveBreitensuche(G, Warteschlange)

08: Prozedur InnereRekursiveBreitensuche(G, Warteschlange)
09:     v ← Warteschlange.EntferneErstes()  /* betrachte alle nachfolgenden Knoten v' ... */
10:     für alle v' ∈ ErmittleNächsteKnoten(G,v) tue  /* ... des aktuellen Knotens v */
11:         wenn ErmittleFarbe(v') = weiß dann  /* wurde Knoten v' noch nicht entdeckt, ... */
12:             LegeFarbeFest(v',schwarz)  /* ... markiere diesen entsprechend und ... */
13:             Warteschlange.ErgänzeAlsLetztes(v')  /* ... besuche ihn im Anschluss */
14:     wenn ¬Warteschlange.IstLeer() dann  /* solange noch Knoten zu besuchen sind, ... */
15:         InnereRekursiveBreitensuche(G, Warteschlange)  /* ... wiederhole */
```

Algorithmus A.1.2: Rekursive Breitensuche von Knoten in einem Graphen *G*.

A.1.3 Nichtrekursive Tiefensuche von Knoten in einem Graphen *G*

Die Tiefensuche ist wie die Breitensuche ein Standardverfahren der Informatik, welches dazu dient, die Knoten V eines allgemeinen Graphen G zu traversieren (vgl. Abschnitt A.1.1). Deren Gemeinsamkeit ist, dass sie ausgehend von einem Knoten σ die Kanten miteinander verbundener Knoten verfolgen. Nachdem jedoch ein Knoten besucht wurde, der vom jeweils betrachteten Knoten unmittelbar erreichbar ist, werden im Gegensatz zur Breitensuche immer erst all dessen Nachfolger besucht. Danach wird mit den verbleibenden unmittelbar erreichbaren Knoten fortgesetzt. Die nichtrekursive Implementierung basiert auf einem *Stapel*, einer nach LIFO verwalteten Liste (siehe Abschnitt 4.1).

Objekte und Variablen	Zeilennummer
Objekt *Stapel*: $\{(v_1, v_2, \ldots) : v \in V\}$	04
Linear verkettete Liste variabler Länge, welche nach LIFO verwaltet wird. Das heißt, zuletzt hinzugefügte Elemente werden zuerst entfernt und zurückgegeben (siehe Abschnitt 4.1). Jedes Element des *Stapels* ist ein Knoten v aus der Menge V des zu traversierenden Graphen G.	

Funktionen und Prozeduren	Zeilennummer
Funktion ErmittleFarbe(v): {`weiß`, `schwarz`}	08
Siehe Abschnitt A.1.1.	
Funktion ErmittleNächsteKnoten(G, v): $\{\{v_1, v_2, \ldots\} : v \in V\}$	10
Siehe Abschnitt A.1.1.	
Prozedur LegeFarbeFest(v, `weiß` oder `schwarz`)	03
Siehe Abschnitt A.1.1.	
Prozedur NichtrekursiveTiefensuche(G, σ)	01
Analog zur Breitensuche in Abschnitt A.1.1 und A.1.2 traversiert die Prozedur all jene Knoten V eines Graphen G, die ausgehend vom Knoten σ erreichbar sind, indem alle aufeinanderfolgenden Kanten beschritten werden. Solange Kanten zu Knoten v führen, die noch nicht entdeckt wurden, wird die Suche fortgesetzt. Im Unterschied zur Breitensuche werden immer erst alle nachfolgenden Knoten v besucht. Wie zuvor werden die besuchten Knoten v `schwarz` gefärbt.	
Funktion *Stapel*.EntferneLetztes(): V	07
Entfernt das zuletzt hinzugefügte Element des *Stapels* und liefert dieses zurück, wie definiert ein Knoten v des zu traversierenden Graphen G. Siehe Erläuterung zum Objekt *Stapel*.	
Prozedur *Stapel*.ErgänzeAlsLetztes(v)	05
Fügt den jeweils übergebenen Knoten v aus der Menge V des zu traversierenden Graphen G an das Ende des *Stapels*. Siehe Erläuterung zum Objekt *Stapel*.	
Funktion *Stapel*.IstLeer(): {`falsch`, `wahr`}	12
Liefert genau dann `wahr` zurück, wenn die Länge des *Stapels* gleich null ist, d. h., wenn dieser einem leeren Tupel entspricht, sonst `falsch`. Siehe Erläuterung zum Objekt *Stapel*.	

Für mathematische Symbole siehe Abschnitt 4.1 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A7 dargestellt.

A.1.4 Rekursive Tiefensuche von Knoten in einem Graphen G

Die rekursive Implementierung der Tiefensuche entspricht der nichtrekursiven Variante dahingehend, dass ausgehend von einem Knoten σ eines Graphen G immer erst alle Nachfolger besucht werden. Jedoch werden die Nachfolger eines Knotens in der umgekehrten Reihenfolge besucht. Durch verschachtelten Aufruf einer Hilfsprozedur kann auf die Verwendung eines *Stapels* verzichtet werden, indem der nächste Knoten als Argument übergeben wird (vgl. Abschnitt A.1.3). Der Ablauf unterscheidet sich außerdem darin, dass in der nichtrekursiven Implementierung erst nach Entfernen eines Knotens vom *Stapel* geprüft wird, ob dieser bereits entdeckt wurde. In der rekursiven Implementierung erfolgt die Prüfung noch vor dem Aufruf der Hilfsprozedur und der Übergabe als Argument.

Funktionen und Prozeduren	Zeilennummer
Funktion ErmittleFarbe(v): {`weiß`, `schwarz`}	8
Siehe Abschnitt A.1.1.	
Funktion ErmittleNächsteKnoten(G, v): $\{\{v_1, v_2, \ldots\} : v \in V\}$	7
Siehe Abschnitt A.1.1.	
Prozedur InnereRekursiveTiefensuche(G, v)	4
Die Hilfsprozedur wird wiederholt aufgerufen, um den rekursiven Gedanken umzusetzen. Im Unterschied zur nichtrekursiven Implementierung ist die Verwendung eines *Stapels* nicht nötig. Wird nur dann aufgerufen, wenn der Knoten v noch nicht entdeckt wurde, wogegen die Prüfung in der nichtrekursiven Implementierung erst nach dem Entfernen vom *Stapel* erfolgt.	
Prozedur LegeFarbeFest(v, `weiß` oder `schwarz`)	3
Siehe Abschnitt A.1.1.	
Prozedur RekursiveTiefensuche(G, σ)	1
Der grundsätzliche Ablauf und das Ergebnis der Hauptprozedur entsprechen der nichtrekursiven Implementierung, wobei die genannten Unterschiede bezüglich der Reihenfolge und der Prüfung von Knoten gelten. Siehe Prozedur NichtrekursiveTiefensuche(G, σ) in Abschnitt A.1.3.	

Für mathematische Symbole siehe Abschnitt 4.1 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A7 dargestellt.

```
    Eingabe: Graph G mit Knoten V und Kanten E, zuerst zu besuchender Knoten σ ∈ V
    Ausgabe: gefärbte Knoten V
01: Prozedur NichtrekursiveTiefensuche(G, σ)
02:     für alle v ∈ V tue
03:         LegeFarbeFest(v, weiß)  /* initialisiere die Knoten V als nicht entdeckt */
04:     Stapel ← ()  /* initialisiere den Stapel, in diesem Fall eine Folge von Knoten v */
05:     Stapel.ErgänzeAlsLetztes(σ)  /* besuche zuallererst Knoten σ */
06:     wiederhole
07:         v ← Stapel.EntferneLetztes()  /* betrachte den aktuellen Knoten v */
08:         wenn ErmittleFarbe(v) = weiß dann  /* wurde Knoten v noch nicht entdeckt, ... */
09:             LegeFarbeFest(v, schwarz)  /* ... markiere diesen entsprechend und ... */
10:             für alle v' ∈ ErmittleNächsteKnoten(G, v) tue  /* ... besuche als Nächstes ... */
11:                 Stapel.ErgänzeAlsLetztes(v')  /* ... alle nachfolgenden Knoten v' */
12:     solange ¬Stapel.IstLeer()  /* wiederhole, solange noch Knoten zu besuchen sind */
```

Algorithmus A.1.3: Nichtrekursive Tiefensuche von Knoten in einem Graphen G.

```
    Eingabe: Graph G mit Knoten V und Kanten E, zuerst zu besuchender Knoten σ ∈ V
    Ausgabe: gefärbte Knoten V
1:  Prozedur RekursiveTiefensuche(G, σ)
2:      für alle v ∈ V tue
3:          LegeFarbeFest(v, weiß)  /* initialisiere die Knoten V als nicht entdeckt */
4:      InnereRekursiveTiefensuche(G, σ)  /* besuche zuallererst Knoten σ */
5:  Prozedur InnereRekursiveTiefensuche(G, v)
6:      LegeFarbeFest(v, schwarz)  /* markiere den aktuellen Knoten v als entdeckt */
7:      für alle v' ∈ ErmittleNächsteKnoten(G, v) tue  /* betrachte alle nachfolgenden Knoten v' */
8:          wenn ErmittleFarbe(v') = weiß dann  /* wurde Knoten v' noch nicht entdeckt, ... */
9:              InnereRekursiveTiefensuche(G, v')  /* ... besuche diesen als Nächstes */
```

Algorithmus A.1.4: Rekursive Tiefensuche von Knoten in einem Graphen G.

A.1.5 Traversierung der Kanten eines grafischen Modells $\{G_{WS,b}\}$

Der Algorithmus wendet das Konzept der nichtrekursiven Tiefensuche auf ein grafisches Modell $\{G_{WS,b}\}$ an, das aus einer endlichen Zahl von Wertstromgraphen $G_{WS,b}$ zusammengesetzt ist (vgl. Abschnitt A.1.3). Beginnend bei der Quelle σ_b jedes Wertstromgraphen $G_{WS,b}$ werden nacheinander alle Kanten $E_{WS,b}$ beschritten, die jeweils erreicht werden können, wenn ein Knoten v der Menge $V_{WS,b}$ an einem Eingang i besucht wird. Wie in der entsprechenden Implementierung des Standardverfahrens wird hierzu ein *Stapel* verwendet. Als Variation handelt es sich bei den Elementen des *Stapels* jedoch nicht um Knoten, sondern um die als Nächstes zu besuchenden Kanten e (siehe Abschnitt 4.3).

Objekte und Variablen — Zeilennummer

Objekt *Stapel*: $\{(e_1, e_2, \ldots) : e \in E_{WS,b}\}$ 07

Linear verkettete Liste variabler Länge, welche nach LIFO verwaltet wird. Das heißt, zuletzt hinzugefügte Elemente werden zuerst entfernt und zurückgegeben (siehe Abschnitt 4.1). Jedes Element des *Stapels* ist eine Kante e aus der Menge $E_{WS,b}$ des aktuell zu traversierenden Wertstromgraphen $G_{WS,b}$.

Funktionen und Prozeduren — Zeilennummer

Funktion ErmittleAusgangszahl$(G_{WS,b}, v)$: $\mathbb{N}_0$ 08

Liefert die Zahl der Ausgänge des Knotens v im Wertstromgraphen $G_{WS,b}$ zurück, insbesondere genutzt für alternative und selektive Flusspunkte der Mengen $D_{AL,b}$ bzw $D_{SL,b}$.

Funktion ErmittleFarbe(b, e): {`weiß`, `schwarz`} 15

Liefert die Farbe der Kante e aus der Menge $E_{WS,b}$ des aktuellen Wertstromgraphen $G_{WS,b}$ zurück. Genau dann, wenn die Kante e bereits entdeckt wurde, ist deren Farbe `schwarz`, sonst `weiß`.

Funktion ErmittleNächstenEingangsindex$(G_{WS,b}, v, j)$: $\mathbb{N}$ 10

Liefert den Eingang i' des nachfolgenden Knotens v' zurück, welcher unmittelbar erreicht wird, indem die Kante am Ausgang j des Knotens v im Wertstromgraphen $G_{WS,b}$ weiterverfolgt wird.

Funktion ErmittleNächstenKnoten$(G_{WS,b}, v, j)$: $V_{WS,b}$ 09

Liefert den nachfolgenden Knoten v' zurück, welcher unmittelbar erreicht wird, indem die Kante am Ausgang j des Knotens v im Wertstromgraphen $G_{WS,b}$ weiterverfolgt wird.

Prozedur LegeFarbeFest$(b, e,$ `weiß` oder `schwarz`$)$ 04

Legt die Farbe der Kante e aus der Menge $E_{WS,b}$ des aktuellen Wertstromgraphen $G_{WS,b}$ fest, wodurch diese als noch nicht entdeckt bzw. entdeckt markiert wird. Siehe Funktion ErmittleFarbe(b, e).

Funktion *Stapel*.EntferneLetztes(): $E_{WS,b}$ 14

Entfernt das zuletzt hinzugefügte Element des *Stapels* und liefert dieses zurück, wie definiert eine Kante e des aktuell zu traversierenden Wertstromgraphen $G_{WS,b}$. Siehe Erläuterung zum Objekt *Stapel*.

Prozedur *Stapel*.ErgänzeAlsLetztes(e) 12

Fügt die übergebene Kante e aus der Menge $E_{WS,b}$ des aktuell zu traversierenden Wertstromgraphen $G_{WS,b}$ an das Ende des *Stapels*. Siehe Erläuterung zum Objekt *Stapel*.

Funktion *Stapel*.IstLeer(): {`falsch`, `wahr`} 13

Siehe Abschnitt A.1.3.

Prozedur TraversiereKantenAusFlusspunkt$(G_{WS,b}, Stapel, v)$ 21

Besucht den Eingang des Knotens v, eines alternativen oder selektiven Flusspunkts aus der Menge $D_{AL,b}$ bzw. $D_{SL,b}$ im Wertstromgraphen $G_{WS,b}$ des Produkts b. Fügt dem *Stapel* alle Kanten e' hinzu, die jeweils an den Ausgängen j des Flusspunkts anliegen, um diese als Nächstes zu besuchen.

Prozedur TraversiereKantenAusRessource ($G_{WS,b}$, *Stapel*, v, i) 19

Besucht den Eingang i des Knotens v, einer Ressource aus der Menge R im Wertstromgraphen $G_{WS,b}$ des Produkts b. Liegt am gegenüberliegenden Ausgang j der Ressource eine Kante e' an, dann wird diese dem *Stapel* hinzugefügt, um sie als Nächstes zu besuchen.

Prozedur TraversiereKantenVonGrafischemModell ($\{G_{WS,b}\}$) 01

Die übergeordnete Prozedur traversiert alle Kanten $E_{WS,b}$ eines gegebenen grafischen Modells $\{G_{WS,b}\}$ und legt deren Farbe fest. Die Suche beginnt jeweils in den Wertstromgraphen $G_{WS,b}$ der Produkte b an der Quelle σ_b. Siehe Prozedur TraversiereKantenVonWSGraphen ($G_{WS,b}$, b).

Prozedur TraversiereKantenVonWSGraphen ($G_{WS,b}$, b) 05

Traversiert all jene Kanten $E_{WS,b}$ im Wertstromgraphen $G_{WS,b}$ des Produkts b, die ausgehend von der Quelle σ_b erreichbar sind, indem alle aufeinanderfolgenden Kanten bis zur Senke τ_b beschritten werden. Als Ergebnis werden die Kanten e, die in dem Zuge besucht wurden, `schwarz` gefärbt.

Für mathematische Symbole siehe Abschnitt 4.3 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A10 dargestellt.

Eingabe: grafisches Modell $\{G_{\mathrm{WS},b}\}$ aus Wertstromgraphen $G_{\mathrm{WS},b}$ mit Knoten $V_{\mathrm{WS},b}$ und Kanten $E_{\mathrm{WS},b}$ für alle Produkte $b \in \{1,\ldots,n\}$
Ausgabe: gefärbte Kanten $E_{\mathrm{WS},b}$

```
01: Prozedur TraversiereKantenVonGrafischemModell({G_WS,b})
02:     für b ← 1 bis n tue /* initialisiere die Kanten E_WS,b ... */
03:         für alle e ∈ E_WS,b tue /* ... in den Wertstromgraphen G_WS,b aller Produkte b ... */
04:             LegeFarbeFest(b, e, weiß) /* ... als nicht entdeckt */
05:         TraversiereKantenVonWSGraphen(G_WS,b, b)

06: Prozedur TraversiereKantenVonWSGraphen(G_WS,b, b)
07:     Stapel ← () /* initialisiere den Stapel, in diesem Fall eine Folge von Kanten e */
08:     wenn ErmittleAusgangszahl(G_WS,b, σ_b) > 0 dann /* wenn an der Quelle σ_b eine Kante anliegt, ... */
09:         v' ← ErmittleNächstenKnoten(G_WS,b, σ_b, 1) /* ... folge der Kante zu Knoten v' ... */
10:         i' ← ErmittleNächstenEingangsindex(G_WS,b, σ_b, 1) /* ... und Eingang i' */
11:         e' ← (σ_b, 1, v', i') /* besuche zuallererst ... */
12:         Stapel.ErgänzeAlsLetztes(e') /* ... diese Kante e' */
13:     solange ¬Stapel.IstLeer() tue /* solange noch Kanten zu besuchen sind, ... */
14:         e ← Stapel.EntferneLetztes() /* ... betrachte die aktuelle Kante e */
15:         wenn ErmittleFarbe(b, e) = weiß dann /* wurde Kante e noch nicht entdeckt, ... */
16:             LegeFarbeFest(b, e, schwarz) /* ... markiere diese entsprechend */
17:             (v, j, v', i') ← e /* um nachfolgende Kanten zu besuchen, unterscheide, ... */
18:             wenn v' ∈ R dann /* ... ob Kante e zu einer Ressource oder ... */
19:                 TraversiereKantenAusRessource(G_WS,b, Stapel, v', i')
20:             sonst wenn v' ∈ D_AL,b ∨ v' ∈ D_SL,b dann /* ... zu einem Flusspunkt führt */
21:                 TraversiereKantenAusFlusspunkt(G_WS,b, Stapel, v')

22: Prozedur TraversiereKantenAusRessource(G_WS,b, Stapel, v, i)
23:     j ← i /* betrachte nur den Ausgang j, welcher dem Eingang i gegenüberliegt */
24:     wenn j ⩽ ErmittleAusgangszahl(G_WS,b, v) dann /* wenn an Ausgang j eine Kante anliegt, ... */
25:         v' ← ErmittleNächstenKnoten(G_WS,b, v, j) /* ... folge der Kante zu Knoten v' ... */
26:         i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j) /* ... und Eingang i' */
27:         e' ← (v, j, v', i') /* besuche als Nächstes ... */
28:         Stapel.ErgänzeAlsLetztes(e') /* ... diese Kante e' */

29: Prozedur TraversiereKantenAusFlusspunkt(G_WS,b, Stapel, v)
30:     für j ← 1 bis ErmittleAusgangszahl(G_WS,b, v) tue /* folge den Kanten an allen Ausgängen j ... */
31:         v' ← ErmittleNächstenKnoten(G_WS,b, v, j) /* ... jeweils zu den Knoten v' und ... */
32:         i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j) /* ... den Eingängen i' */
33:         e' ← (v, j, v', i') /* besuche als Nächstes ... */
34:         Stapel.ErgänzeAlsLetztes(e') /* ... all diese Kanten e' */
```

Algorithmus A.1.5: Traversierung der Kanten eines grafischen Modells $\{G_{\mathrm{WS},b}\}$.

A.1.6 Validierung eines grafischen Modells $\{G_{WS,b}\}$

Nachdem der Anwender ein grafisches Modell $\{G_{WS,b}\}$ erstellt hat, muss geprüft werden, ob dieses gültig ist und weiterverarbeitet werden kann. Diesbezüglich gelten folgende vier Bedingungen (siehe Abschnitt 4.3): (1) An der Quelle σ_b eines Wertstromgraphen $G_{WS,b}$ muss eine Kante anliegen; (2) die Kanten dürfen keinen unendlichen Zyklus bilden; (3) jede Kante muss zur Senke τ_b weiterverfolgt werden können; (4) jede Kante muss von der Quelle σ_b erreichbar sein. Im Ablauf des Algorithmus werden beginnend bei den Quellen σ_b die Kanten des grafischen Modells $\{G_{WS,b}\}$ traversiert. Analog zum zuvor eingeführten Algorithmus wird zu diesem Zweck ein *Stapel* verwendet (vgl. Abschnitt A.1.5). Darüber hinaus wird der Weg von der Quelle σ_b zur aktuellen Kante in einem *Kantenzug* erfasst.

Hinweis: Der folgende Algorithmus ist gegenüber der Implementierung in der Version 1.2 von AURELIE, welche bei der Bosch Rexroth AG eingesetzt wird, optimiert.

Objekte und Variablen — Zeilennummer

Variable *ausgangszahl* : $\mathbb{N}_0$ 51

Natürliche Zahl größer als oder gleich null, welche die Zahl der Ausgänge eines alternativen oder selektiven Flusspunkts v aus der Menge $D_{AL,b}$ bzw. $D_{SL,b}$ im Wertstromgraphen $G_{WS,b}$ angibt.

Objekt *Kantenzug* : $\{(e_1, e_2, \ldots) : e \in E_{WS,b}\}$ 08

Linear verkettete Liste variabler Länge, welche nach LIFO verwaltet wird. Das heißt, zuletzt hinzugefügte Elemente werden zuerst entfernt und zurückgegeben (siehe Abschnitt 4.1). Jedes Element des *Kantenzugs* ist eine Kante e aus der Menge $E_{WS,b}$ des aktuell zu validierenden Wertstromgraphen $G_{WS,b}$. Die Folge der Kanten entspricht dem Weg von der Quelle σ_b zur Kante vom Ende des *Stapels*.

Objekt *MengeUnbesuchterKanten* : $\{\{e_1, e_2, \ldots\} : e \in E_{WS,b}\}$ 09

Menge, d. h. unsortierte Datenstruktur, mit variabler Zahl von Elementen. Jedes dieser Elemente ist eine Kante e aus der Menge $E_{WS,b}$ des aktuell zu validierenden Wertstromgraphen $G_{WS,b}$. Eine solche Kante e ist genau dann in der Menge enthalten, wenn sie zu diesem Zeitpunkt noch nicht besucht wurde (vgl. Färbung von Kanten im vorherigen Algorithmus in Abschnitt A.1.5).

Objekt *Stapel* : $\{(e_1, e_2, \ldots) : e \in E_{WS,b}\}$ 07

Siehe Abschnitt A.1.5.

Funktionen und Prozeduren — Zeilennummer

Funktion ErmittleAusgangszahl$(G_{WS,b}, v)$: $\mathbb{N}_0$ 10

Siehe Abschnitt A.1.5.

Funktion ErmittleNächstenEingangsindex$(G_{WS,b}, v, j)$: $\mathbb{N}$ 13

Siehe Abschnitt A.1.5.

Funktion ErmittleNächstenKnoten$(G_{WS,b}, v, j)$: $V_{WS,b}$ 12

Siehe Abschnitt A.1.5.

Funktion *Kantenzug*.EntferneLetztes() : $E_{WS,b}$ 23

Entfernt das zuletzt hinzugefügte Element des *Kantenzugs* und liefert dieses zurück, wie definiert eine Kante e des aktuellen Wertstromgraphen $G_{WS,b}$. Siehe Erläuterung zum Objekt *Kantenzug*.

Prozedur *Kantenzug*.ErgänzeAlsLetztes(e) 30

Fügt die übergebene Kante e aus der Menge $E_{WS,b}$ des aktuell zu validierenden Wertstromgraphen $G_{WS,b}$ an das Ende des *Kantenzugs*. Siehe Erläuterung zum Objekt *Kantenzug*.

Funktion *Kantenzug*.ErmittleLetztes(): $E_{\mathrm{WS},b}$ 20

Liefert das zuletzt hinzugefügte Element des *Kantenzugs* zurück, wie definiert eine Kante e des aktuellen Wertstromgraphen $G_{\mathrm{WS},b}$, ohne dieses zu entfernen. Siehe Erläuterung zum Objekt *Kantenzug*.

Funktion *Kantenzug*.IstLeer(): {`falsch`,`wahr`} 19

Liefert genau dann `wahr` zurück, wenn die Länge des *Kantenzugs* gleich null ist, d. h., wenn dieser einem leeren Tupel entspricht, sonst `falsch`. Siehe Erläuterung zum Objekt *Kantenzug*.

Funktion *MengeUnbesuchterKanten*.IstLeer(): {`falsch`,`wahr`} 38

Liefert genau dann `wahr` zurück, wenn die Kardinalität der obigen Menge gleich null ist, d. h., wenn diese der leeren Menge entspricht, sonst `falsch`. Siehe Erläuterung zum entsprechenden Objekt.

Funktion *Stapel*.EntferneLetztes(): $E_{\mathrm{WS},b}$ 17

Siehe Abschnitt A.1.5.

Prozedur *Stapel*.ErgänzeAlsLetztes(e) 15

Siehe Abschnitt A.1.5.

Funktion *Stapel*.IstLeer(): {`falsch`,`wahr`} 37

Siehe Abschnitt A.1.3.

Funktion ValidiereGrafischesModell({$G_{\mathrm{WS},b}$}): {`falsch`,`wahr`} 01

Die übergeordnete Funktion validiert ein gegebenes grafisches Modell {$G_{\mathrm{WS},b}$}, indem sie prüft, ob die Wertstromgraphen $G_{\mathrm{WS},b}$ aller Produkte b die definierten Bedingungen erfüllen. Liefert genau dann `wahr` zurück, wenn dies auf jeden einzelnen Wertstromgraphen $G_{\mathrm{WS},b}$ zutrifft, sonst `falsch`.

Funktion ValidiereKantenAusFlusspunkt($G_{\mathrm{WS},b}$, *Stapel*, v): {`falsch`,`wahr`} 35

Besucht den Eingang des Knotens v, eines alternativen oder selektiven Flusspunkts aus der Menge $D_{\mathrm{AL},b}$ bzw. $D_{\mathrm{SL},b}$ im Wertstromgraphen $G_{\mathrm{WS},b}$ des Produkts b. Fügt dem *Stapel* alle Kanten e' hinzu, die jeweils an den Ausgängen j des Flusspunkts anliegen, um mit diesen die Validierung fortzusetzen. Liefert genau dann `wahr` zurück, wenn mindestens eine solche Kante e' existiert, sonst `falsch`.

Funktion ValidiereKantenAusRessource($G_{\mathrm{WS},b}$, *Stapel*, v, i): {`falsch`,`wahr`} 32

Besucht den Eingang i des Knotens v, einer Ressource aus der Menge R im Wertstromgraphen $G_{\mathrm{WS},b}$ des Produkts b. Wenn am gegenüberliegenden Ausgang j der Ressource eine Kante e' anliegt, wird diese dem *Stapel* hinzugefügt, um mit ihr die Validierung fortzusetzen. Liefert genau dann `wahr` zurück, wenn eine solche Kante e' an Ausgang j existiert, sonst `falsch`.

Funktion ValidiereWSGraphen($G_{\mathrm{WS},b}$, b): {`falsch`,`wahr`} 03

Traversiert den Wertstromgraphen $G_{\mathrm{WS},b}$ des Produkts b und prüft, ob dieser die definierten Bedingungen erfüllt. Die Suche beginnt jeweils an der Quelle σ_b und endet, wenn alle aufeinanderfolgenden Kanten besucht wurden oder in nur einem einzigen Fall eine der Bedingungen nicht erfüllt ist. Liefert genau dann `wahr` zurück, wenn kein solcher Fall existiert, sonst `falsch`.

Für mathematische Symbole siehe Abschnitt 4.3 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A13–A14 dargestellt.

Eingabe: grafisches Modell $\{G_{WS,b}\}$ aus Wertstromgraphen $G_{WS,b}$ mit Knoten $V_{WS,b}$ und Kanten $E_{WS,b}$ für alle Produkte $b \in \{1,\ldots,n\}$
Ausgabe: `wahr`, wenn das grafische Modell $\{G_{WS,b}\}$ gültig ist, sonst `falsch`

```
01: Funktion ValidiereGrafischesModell({G_WS,b})
02:     für b ← 1 bis n tue  /* wenn mindestens ein Wertstromgraph G_WS,b ...                    */
03:         wenn ¬ValidiereWSGraphen(G_WS,b, b) dann  /* ... eines Produkts b ungültig ist, ...  */
04:             liefere falsch zurück  /* ... ist das grafische Modell {G_WS,b} ungültig        */
05:     liefere wahr zurück

06: Funktion ValidiereWSGraphen(G_WS,b, b)
07:     Stapel ← ()  /* initialisiere den Stapel, in diesem Fall eine Folge von Kanten e, und ... */
08:     Kantenzug ← ()  /* ... den Kantenzug, eine ebensolche Folge                              */
09:     MengeUnbesuchterKanten ← E_WS,b  /* initialisiere die Kanten E_WS,b als nicht besucht  */
10:     wenn ErmittleAusgangszahl(G_WS,b, σ_b) = 0 dann  /* liegt an der Quelle σ_b keine Kante an, ... */
11:         liefere falsch zurück  /* ... ist der Wertstromgraph G_WS,b ungültig                */
12:     v' ← ErmittleNächstenKnoten(G_WS,b, σ_b, 1)  /* folge der Kante an der Quelle σ_b zu Knoten v' ... */
13:     i' ← ErmittleNächstenEingangsindex(G_WS,b, σ_b, 1)  /* ... und Eingang i'               */
14:     e' ← (σ_b, 1, v', i')  /* besuche zuallererst ...                                       */
15:     Stapel.ErgänzeAlsLetztes(e')  /* ... diese Kante e'                                     */
16:     wiederhole
17:         e_2 ← Stapel.EntferneLetztes()  /* betrachte die aktuelle Kante e_2, ...            */
18:         (v_2, j_2, v'_2, i'_2) ← e_2  /* ... welche von einem Knoten v_2 ausgeht            */
19:         wenn ¬Kantenzug.IstLeer() dann  /* wenn der Kantenzug nicht leer ist, ...           */
20:             e_1 ← Kantenzug.ErmittleLetztes()  /* ... betrachte dessen letzte Kante e_1, ... */
21:             (v_1, j_1, v'_1, i'_1) ← e_1  /* ... welche zu einem Knoten v'_1 führt        */
22:             solange v'_1 <> v_2 tue  /* solange Kante e_2 nicht unmittelbar an Kante e_1 anknüpft, ... */
23:                 Kantenzug.EntferneLetztes()  /* ... reduziere den Kantenzug und ...        */
24:                 e_1 ← Kantenzug.ErmittleLetztes()  /* ... betrachte wieder die letzte Kante e_1 ... */
25:                 (v_1, j_1, v'_1, i'_1) ← e_1  /* ... zu einem Knoten v'_1                 */
26:         wenn e_2 ∈ Kantenzug dann  /* wenn Kante e_2 bereits im Kantenzug enthalten ist, ... */
27:             liefere falsch zurück  /* ... ist der Wertstromgraph G_WS,b ungültig           */
28:         wenn e_2 ∈ MengeUnbesuchterKanten dann  /* wurde Kante e_2 noch nicht besucht, ...  */
29:             MengeUnbesuchterKanten ← MengeUnbesuchterKanten \ {e_2}  /* ... markiere diese ... */
30:             Kantenzug.ErgänzeAlsLetztes(e_2)  /* ... und füge sie ans Ende des Kantenzugs  */
31:             wenn v'_2 ∈ R dann  /* unterscheide danach, ob Kante e_2 zu einer Ressource oder ... */
32:                 wenn ¬ValidiereKantenAusRessource(G_WS,b, Stapel, v'_2, i'_2) dann
33:                     liefere falsch zurück
34:             sonst wenn v'_2 ∈ D_AL,b ∨ v'_2 ∈ D_SL,b dann  /* ... zu einem Flusspunkt führt */
35:                 wenn ¬ValidiereKantenAusFlusspunkt(G_WS,b, Stapel, v'_2) dann
36:                     liefere falsch zurück
37:     solange ¬Stapel.IstLeer()  /* wiederhole, solange noch Kanten zu besuchen sind          */
38:     wenn ¬MengeUnbesuchterKanten.IstLeer() dann  /* wenn nicht alle Kanten besucht wurden, ... */
39:         liefere falsch zurück  /* ... ist der Wertstromgraph G_WS,b ungültig               */
40:     liefere wahr zurück
```

(nächster Teil auf Seite A14)

Algorithmus A.1.6(a): Validierung eines grafischen Modells $\{G_{WS,b}\}$ (Teil 1 von 2).

(vorheriger Teil auf Seite A13)

```
41: Funktion ValidiereKantenAusRessource(G_WS,b, Stapel, v, i)
42:     j ← i  /* betrachte nur den Ausgang j, welcher dem Eingang i gegenüberliegt */
43:     wenn j > ErmittleAusgangszahl(G_WS,b, v) dann  /* wenn an Ausgang j keine Kante anliegt, ... */
44:         liefere falsch zurück  /* ... ist der Wertstromgraph G_WS,b ungültig */
45:     v' ← ErmittleNächstenKnoten(G_WS,b, v, j)  /* folge der Kante an Ausgang j zu Knoten v' ... */
46:     i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j)  /* ... und Eingang i' */
47:     e' ← (v, j, v', i')  /* besuche als Nächstes ... */
48:     Stapel.ErgänzeAlsLetztes(e')  /* ... diese Kante e' */
49:     liefere wahr zurück

50: Funktion ValidiereKantenAusFlusspunkt(G_WS,b, Stapel, v)
51:     ausgangszahl ← ErmittleAusgangszahl(G_WS,b, v)
52:     wenn ausgangszahl = 0 dann  /* wenn an keinem Ausgang eine Kante anliegt, ... */
53:         liefere falsch zurück  /* ... ist der Wertstromgraph G_WS,b ungültig */
54:     für j ← 1 bis ausgangszahl tue  /* folge den Kanten an allen Ausgängen j ... */
55:         v' ← ErmittleNächstenKnoten(G_WS,b, v, j)  /* ... jeweils zu den Knoten v' und ... */
56:         i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j)  /* ... den Eingängen i' */
57:         e' ← (v, j, v', i')  /* besuche als Nächstes ... */
58:         Stapel.ErgänzeAlsLetztes(e')  /* ... all diese Kanten e' */
59:     liefere wahr zurück
```

Algorithmus A.1.6(b): Validierung eines grafischen Modells $\{G_{WS,b}\}$ (Teil 2 von 2).

A.1.7 Traversierung der Knoten eines grafischen Modells $\{G_{WS,b}\}$

Der nachfolgende Algorithmus entspricht bezüglich Ablauf und Ergebnis dem vorangegangenen Algorithmus zur Traversierung der Kanten eines grafischen Modells $\{G_{WS,b}\}$ (vgl. Abschnitt A.1.5). In beiden Fällen wird eine nichtrekursive Tiefensuche angewandt, um ausgehend von den Quellen σ_b alle erreichbaren Kanten $E_{WS,b}$ zu beschreiten. Im vorliegenden Fall wird der *Stapel* jedoch nicht genutzt, um die nächsten Kanten zu verwalten, sondern die als Nächstes zu besuchenden Knoten $V_{WS,b}$. Da für die Traversierung entscheidend ist, an welchem Eingang ein Knoten besucht wird, sind die Elemente des *Stapels* Paare aus je einem Knoten v und einem Eingang i (siehe Abschnitt 4.4).

Objekte und Variablen — Zeilennummer

Objekt *Stapel*: $\{((v_1,i_1),(v_2,i_2),\ldots): v \in V_{WS,b}, i \in \mathbb{N}\}$ 08

Linear verkettete Liste variabler Länge, welche nach LIFO verwaltet wird. Das heißt, zuletzt hinzugefügte Elemente werden zuerst entfernt und zurückgegeben (siehe Abschnitt 4.1). Jedes Element des *Stapels* ist ein Paar (v,i) aus einem Knoten v der Menge $V_{WS,b}$ und einem Eingang i zur Unterscheidung jeweils eingehender Kanten im aktuell zu traversierenden Wertstromgraphen $G_{WS,b}$.

Funktionen und Prozeduren — Zeilennummer

Funktion ErmittleAusgangszahl$(G_{WS,b}, v)$: $\mathbb{N}_0$ 09

Siehe Abschnitt A.1.5.

Funktion ErmittleEingangszahl$(G_{WS,b}, v)$: $\mathbb{N}_0$ 04

Liefert die Zahl der Eingänge i des Knotens v im Wertstromgraphen $G_{WS,b}$ zurück, insbesondere genutzt zur Initialisierung der Farben aller Eingänge i vor Beginn der Traversierung.

Funktion ErmittleFarbe(b, v, i): {`weiß`, `schwarz`} 15

Liefert die Farbe des Eingangs i von Knoten v im aktuellen Wertstromgraphen $G_{WS,b}$ zurück. Genau dann, wenn der Eingang i bereits entdeckt wurde, ist dessen Farbe `schwarz`, sonst `weiß`.

Funktion ErmittleNächstenEingangsindex$(G_{WS,b}, v, j)$: $\mathbb{N}$ 11

Siehe Abschnitt A.1.5.

Funktion ErmittleNächstenKnoten$(G_{WS,b}, v, j)$: $V_{WS,b}$ 10

Siehe Abschnitt A.1.5.

Prozedur LegeFarbeFest$(b, v, i,$ `weiß` oder `schwarz`$)$ 05

Legt die Farbe des Eingangs i von Knoten v im aktuellen Wertstromgraphen $G_{WS,b}$ fest, wodurch dieser als noch nicht entdeckt bzw. entdeckt markiert wird. Siehe Funktion ErmittleFarbe(b, v, i).

Funktion *Stapel*.EntferneLetztes(): $\{(v,i): v \in V_{WS,b}, i \in \mathbb{N}\}$ 14

Entfernt das zuletzt hinzugefügte Element des *Stapels* und liefert dieses zurück, wie definiert ein Paar (v,i) aus einem Knoten v und einem Eingang i. Siehe Erläuterung zum Objekt *Stapel*.

Prozedur *Stapel*.ErgänzeAlsLetztes$((v,i))$ 12

Fügt das jeweils übergebene Paar (v,i) aus dem Knoten v und dem Eingang i im aktuell zu traversierenden Wertstromgraphen $G_{WS,b}$ an das Ende des *Stapels*. Siehe Erläuterung zum Objekt *Stapel*.

Funktion *Stapel*.IstLeer(): {`falsch`, `wahr`} 13

Siehe Abschnitt A.1.3.

Prozedur TraversiereFlusspunkt ($G_{WS,b}$, *Stapel*, v) 20

Besucht den Eingang des Knotens v, eines alternativen oder selektiven Flusspunkts aus der Menge $D_{AL,b}$ bzw. $D_{SL,b}$ im Wertstromgraphen $G_{WS,b}$ des Produkts b. Fügt dem *Stapel* alle Eingänge i' von Knoten v' hinzu, die nach den Ausgängen j folgen, um diese als Nächstes zu besuchen.

Prozedur TraversiereKnotenVonGrafischemModell ($\{G_{WS,b}\}$) 01

Die übergeordnete Prozedur traversiert alle Eingänge i von Knoten $V_{WS,b}$ eines gegebenen grafischen Modells $\{G_{WS,b}\}$ und legt deren Farbe fest, jeweils beginnend in den Wertstromgraphen $G_{WS,b}$ der Produkte b an der Quelle σ_b. Siehe Prozedur TraversiereKnotenVonWSGraphen ($G_{WS,b}$, b).

Prozedur TraversiereKnotenVonWSGraphen ($G_{WS,b}$, b) 06

Traversiert all jene Eingänge i von Knoten $V_{WS,b}$ im Wertstromgraphen $G_{WS,b}$ des Produkts b, die ausgehend von der Quelle σ_b erreichbar sind, indem alle aufeinanderfolgenden Kanten bis zur Senke τ_b beschritten werden. Als Ergebnis werden die besuchten Eingänge i `schwarz` gefärbt.

Prozedur TraversiereRessource ($G_{WS,b}$, *Stapel*, v, i) 18

Besucht den Eingang i des Knotens v, einer Ressource aus der Menge R im Wertstromgraphen $G_{WS,b}$ des Produkts b. Liegt am gegenüberliegenden Ausgang j eine Kante an, dann wird dem *Stapel* der Eingang i' des folgenden Knotens v' hinzugefügt, um diesen als Nächstes zu besuchen.

Für mathematische Symbole siehe Abschnitt 4.4 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A17 dargestellt.

Eingabe: grafisches Modell $\{G_{WS,b}\}$ aus Wertstromgraphen $G_{WS,b}$ mit Knoten $V_{WS,b}$ und Kanten $E_{WS,b}$ für alle Produkte $b \in \{1,\ldots,n\}$
Ausgabe: gefärbte Eingänge i aller Knoten $V_{WS,b}$ außer den Quellen σ_b

```
01: Prozedur TraversiereKnotenVonGrafischemModell({G_WS,b})
02:     für b ← 1 bis n tue /* initialisiere die Eingänge i ... */
03:         für alle v ∈ V_WS,b tue /* ... der Knoten V_WS,b in den Wertstromgraphen G_WS,b ... */
04:             für i ← 1 bis ErmittleEingangszahl(G_WS,b, v) tue /* ... aller Produkte b ... */
05:                 LegeFarbeFest(b, v, i, weiß) /* ... als nicht entdeckt */
06:         TraversiereKnotenVonWSGraphen(G_WS,b, b)

07: Prozedur TraversiereKnotenVonWSGraphen(G_WS,b, b)
08:     Stapel ← () /* initialisiere den Stapel, in diesem Fall eine Folge von Paaren (v, i) */
09:     wenn ErmittleAusgangszahl(G_WS,b, σ_b) > 0 dann /* wenn an der Quelle σ_b eine Kante anliegt, ... */
10:         v' ← ErmittleNächstenKnoten(G_WS,b, σ_b, 1) /* ... folge der Kante zu Knoten v' ... */
11:         i' ← ErmittleNächstenEingangsindex(G_WS,b, σ_b, 1) /* ... und dem zugehörigen Eingang i' */
12:         Stapel.ErgänzeAlsLetztes((v', i')) /* besuche zuallererst diesen Eingang i' */
13:     solange ¬Stapel.IstLeer() tue /* solange Eingänge zu besuchen sind, ... */
14:         (v, i) ← Stapel.EntferneLetztes() /* ... betrachte den aktuellen Knoten v und Eingang i */
15:         wenn ErmittleFarbe(b, v, i) = weiß dann /* wurde Eingang i noch nicht entdeckt, ... */
16:             LegeFarbeFest(b, v, i, schwarz) /* ... markiere diesen entsprechend */
17:             wenn v ∈ R dann /* unterscheide, ob es sich bei Knoten v um eine Ressource oder ... */
18:                 TraversiereRessource(G_WS,b, Stapel, v, i)
19:             sonst wenn v ∈ D_AL,b ∨ v ∈ D_SL,b dann /* ... um einen Flusspunkt handelt */
20:                 TraversiereFlusspunkt(G_WS,b, Stapel, v)

21: Prozedur TraversiereRessource(G_WS,b, Stapel, v, i)
22:     j ← i /* betrachte nur den Ausgang j, welcher dem Eingang i gegenüberliegt */
23:     wenn j ⩽ ErmittleAusgangszahl(G_WS,b, v) dann /* wenn an Ausgang j eine Kante anliegt, ... */
24:         v' ← ErmittleNächstenKnoten(G_WS,b, v, j) /* ... folge der Kante zu Knoten v' ... */
25:         i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j) /* ... und Eingang i' */
26:         Stapel.ErgänzeAlsLetztes((v', i')) /* besuche als Nächstes diesen Eingang i' */

27: Prozedur TraversiereFlusspunkt(G_WS,b, Stapel, v)
28:     für j ← 1 bis ErmittleAusgangszahl(G_WS,b, v) tue /* folge den Kanten an allen Ausgängen j ... */
29:         v' ← ErmittleNächstenKnoten(G_WS,b, v, j) /* ... jeweils zu den Knoten v' und ... */
30:         i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j) /* ... den Eingängen i' */
31:         Stapel.ErgänzeAlsLetztes((v', i')) /* besuche als Nächstes all diese Eingänge i' */
```

Algorithmus A.1.7: Traversierung der Knoten eines grafischen Modells $\{G_{WS,b}\}$.

A.1.8 Transformation eines grafischen Modells $\{G_{WS,b}\}$

Der Algorithmus verbindet die Modellierung mit der Optimierung, indem er ein gegebenes grafisches Modell $\{G_{WS,b}\}$ in ein mathematisches Modell $(\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{Invest})$ umwandelt, das sich zur Optimierung eignet. Das Ergebnis sind die genannten Matrizen, wobei die Produktstrukturmatrix **S** ausgenommen ist, da diese als gegeben vorausgesetzt wird. Ein weiteres Ergebnis ist die *MaxFolge*, welche eine Teilmenge der Indizes a aller Prozessstückzahlen x_a enthält. Die Sortierung der Indizes entspricht der Reihenfolge, in welcher die Prozessstückzahlen x_a iterativ maximiert werden müssen, um die Auslastung zu optimieren. Bestimmt wird diese Reihenfolge durch die Priorisierung, welche der Anwender durch die Verknüpfung der Ausgänge alternativer Flusspunkte definiert (siehe Abschnitt 4.4).

In seinem Ablauf basiert der Algorithmus auf der zuvor beschriebenen Traversierung der Knoten eines grafischen Modells $\{G_{WS,b}\}$ (vgl. Abschnitt A.1.7). Gegenüber diesem Algorithmus wird der *Stapel* nicht nur genutzt, um die zu besuchenden Knoten v und Eingänge i zu verwalten. Zusätzlich sind jedem Knoten und Eingang der Index a und der Koeffizient q der Prozessstückzahl x_a zugeordnet, die jeweils an diesem Eingang anliegt. Die Wertstromgraphen $G_{WS,b}$ werden in zwei Modi traversiert, um die Eingänge zu erfassen, die nach den ersten Ausgängen alternativer Flusspunkte $D_{AL,b}$ folgen. Diese entsprechen Prozessschritten, die nach Vorgabe des Anwenders zu einer Überlastung von Ressourcen führen dürfen und zur Ermittlung der minimalen Investitionen dienen.

Objekte und Variablen — Zeilennummer

Variable *ausgangszahl* : $\mathbb{N}$ — 074

Natürliche Zahl, welche jeweils die Zahl der Ausgänge eines alternativen oder selektiven Flusspunkts v aus der Menge $D_{AL,b}$ bzw. $D_{SL,b}$ im Wertstromgraphen $G_{WS,b}$ angibt. Da das grafische Modell $\{G_{WS,b}\}$ als Vorbedingung für die Ausführung des Algorithmus validiert sein muss, ist die Zahl der Ausgänge für alle besuchten Knoten v außer den Senken τ_b größer als null.

Objekt *InvestMenge* : $\{\{(b_1, v_1, i_1), (b_2, v_2, i_2), \ldots\} : b \in \{1, \ldots, n\}, v \in V_{WS,b}, i \in \mathbb{N}\}$ — 009

Menge mit einer variablen Zahl an Elementen. Jedes Element ist ein Tupel (b, v, i) aus einem Produkt b sowie einem Knoten v und einem Eingang i im Wertstromgraphen $G_{WS,b}$. Ein Eingang i ist der Menge genau dann zugehörig, wenn dieser ausgehend von der Quelle σ_b erreicht werden kann, indem an jedem alternativen Flusspunkt aus der Menge $D_{AL,b}$ stets der erste Ausgang verfolgt wird.

Variable *istEingangVereinigung* : {`falsch`, `wahr`} — 027

Boolesche Variable, welche angibt, ob am Eingang i des aktuell betrachteten Knotens v im Wertstromgraphen $G_{WS,b}$ mehrere Kanten zusammengeführt werden. Das heißt, der Wert der Variable ist genau dann `wahr`, wenn verschiedene Knoten oder Ausgänge eines Knotens existieren, von denen der Eingang i des Knotens v unmittelbar erreicht werden kann, sonst `falsch`.

Objekt *MaxFolge* : $\{(a_1, a_2, \ldots) : a \in \{1, \ldots, m\}\}$ — 010

Linear verkettete Liste variabler Länge, welche nach FIFO verwaltet wird. Das heißt, zuerst hinzugefügte Elemente werden zuerst entfernt und zurückgegeben (siehe Abschnitt 4.1). Jedes Element der *MaxFolge* bezeichnet den Index a einer Prozessstückzahl x_a. Die Sortierung entspricht der Reihenfolge, in welcher die Prozessstückzahlen x_a nach Vorgabe des Anwenders zu maximieren sind.

Objekt *Stapel*: $\{((v_1, i_1, a_1, q_1), (v_2, i_2, a_2, q_2), \ldots) : v \in V_{\mathrm{WS},b}, i \in \mathbb{N}, a \in \{0, \ldots, m\}, q \in [0,1]\}$ 016

Linear verkettete Liste variabler Länge, welche nach LIFO verwaltet wird. Das heißt, zuletzt hinzugefügte Elemente werden zuerst entfernt und zurückgegeben (siehe Abschnitt 4.1). Jedes Element des *Stapels* ist ein Tupel (v, i, a, q) aus einem Knoten v, einem Eingang i sowie dem Index a und dem Koeffizienten q der Prozessstückzahl x_a, welche an Eingang i anliegt.

Variable *zeile*: $\mathbb{N}_0$ 110

Natürliche Zahl größer als oder gleich null, welche einer allgemeinen, nicht näher bezeichneten Zeile der Flussmatrix **F** oder, je nach Anwendungsfall, der Auslastungsmatrix **L** entspricht.

Funktionen und Prozeduren

Zeilennummer

Funktion ErmittleAusgangszahl$(G_{\mathrm{WS},b}, v)$: $\mathbb{N}_0$ 074

Siehe Abschnitt A.1.5.

Funktion ErmittleBetriebsmittelzeit(v): $\mathbb{R}_{>0}$ 122

Liefert die Betriebsmittelzeit T_{BM} der Ressource v zurück, in welcher die jeweilige MAE nach Vorgabe des Anwenders im Planungsintervall zur Verfügung steht.

Funktion ErmittleEffektiveTaktzeit$(G_{\mathrm{WS},b}, v, i)$: $\mathbb{R}_{>0}$ 126

Liefert die vom Anwender vorgegebene, effektive Taktzeit t_{eff} des Prozessschritts zurück, welcher durch den Eingang i der Ressource v im Wertstromgraphen $G_{\mathrm{WS},b}$ repräsentiert wird.

Funktion ErmittleEingangszahl$(G_{\mathrm{WS},b}, v)$: $\mathbb{N}_0$ 125

Liefert die Zahl der Eingänge des Knotens v im Wertstromgraphen $G_{\mathrm{WS},b}$ zurück, insbesondere genutzt zur Zusammenfassung aller Prozessschritte von Ressourcen der Menge R.

Funktion ErmittleNächstenEingangsindex$(G_{\mathrm{WS},b}, v, j)$: $\mathbb{N}$ 022

Siehe Abschnitt A.1.5.

Funktion ErmittleNächstenKnoten$(G_{\mathrm{WS},b}, v, j)$: $V_{\mathrm{WS},b}$ 020

Siehe Abschnitt A.1.5.

Funktion ErmittleQuote$(G_{\mathrm{WS},b}, v, j)$: $(0,1]$ 103

Liefert die Quote am Ausgang j des selektiven Flusspunkts v aus der Menge $D_{\mathrm{SL},b}$ im Wertstromgraphen $G_{\mathrm{WS},b}$ zurück. Bezug nehmend auf die Stückzahl, die am Eingang des Flusspunkts anliegt, bezeichnet die Quote den fixen, relativen Anteil, welcher durch den Ausgang j verläuft.

Prozedur FülleAuslastungsmatrix$(\{G_{\mathrm{WS},b}\}, \mathbf{L}, \mathbf{L}_{\mathrm{Invest}}, \{\mathbf{Q}_{\mathrm{ein},bv}\}, \mathit{InvestMenge})$ 014

Legt die Komponenten der Auslastungsmatrix **L** und Investitionsmatrix $\mathbf{L}_{\mathrm{Invest}}$ nach Traversierung des grafischen Modells $\{G_{\mathrm{WS},b}\}$ fest. Die Werte resultieren aus den Betriebsmittelzeiten T_{BM}, den effektiven Taktzeiten t_{eff} und der Struktur des Modells. Diese wird der Prozedur mit der Menge der Eingangsmatrizen $\mathbf{Q}_{\mathrm{ein},bv}$ als Argument übergeben. Siehe Funktion TransformiereWSGraphen(...).

Prozedur FülleFlussmatrix$(\{G_{\mathrm{WS},b}\}, m, \mathbf{F}, \{\mathbf{Q}_{\mathrm{ein},bv}\}, \{\mathbf{Q}_{\mathrm{aus},bv}\}, \mathbf{U})$ 013

Legt die Komponenten der Flussmatrix **F** nach Traversierung des grafischen Modells $\{G_{\mathrm{WS},b}\}$ fest. Die Werte folgen ausschließlich aus der Struktur des Modells. Die Struktur wird analog zur Prozedur FülleAuslastungsmatrix(...) mit den Eingangsmatrizen $\mathbf{Q}_{\mathrm{ein},bv}$, den Ausgangsmatrizen $\mathbf{Q}_{\mathrm{aus},bv}$ und der Vereinigungsmatrix **U** übergeben. Siehe Funktion TransformiereWSGraphen(...).

Prozedur *MaxFolge*.ErgänzeAlsLetztes(a) 083

Fügt den jeweils übergebenen Index a einer Prozessstückzahl x_a am Ausgang eines alternativen Flusspunkts aus der Menge $D_{\mathrm{AL},b}$ an das Ende der *MaxFolge*. Siehe Erläuterung zum Objekt *MaxFolge*.

Funktion *Stapel*.EntferneLetztes(): $\{(v, i, a, q) : v \in V_{\mathrm{WS},b}, i \in \mathbb{N}, a \in \{0, \ldots, m\}, q \in [0,1]\}$ 026

Entfernt das zuletzt hinzugefügte Element des *Stapels* und liefert dieses zurück, wie definiert ein Tupel (v, i, a, q) aus einem Knoten v, einem Eingang i, dem Index a der Prozessstückzahl x_a an diesem Eingang und dem Koeffizienten q. Siehe Erläuterung zum Objekt *Stapel*.

Prozedur *Stapel*.ErgänzeAlsLetztes((*v*, *i*, *a*, *q*)) 024

Fügt das übergebene Tupel (*v*, *i*, *a*, *q*) aus dem Knoten *v*, dem Eingang *i*, dem Index *a* der Prozessstückzahl x_a und dem Koeffizienten *q* an das Ende des *Stapels*. Siehe Erläuterung zum Objekt *Stapel*.

Funktion *Stapel*.IstLeer(): {`falsch`,`wahr`} 025

Siehe Abschnitt A.1.3.

Funktion TransformiereALFlusspunkt($G_{WS,b}$, *b*, *m*, $\mathbf{Q}_{aus,bv}$, *InvestMenge*, *MaxFolge*, *Stapel*, *v*, *a*, *q*): $\mathbb{N}$ 048

Besucht den Eingang des Knotens *v*, eines alternativen Flusspunkts aus der Menge $D_{AL,b}$ im Wertstromgraphen $G_{WS,b}$ des Produkts *b*, und aktualisiert die *InvestMenge* sowie die *MaxFolge*. Die Prozessstückzahl x_a und der Koeffizient *q*, welche am Eingang des Flusspunkts *v* anliegen, werden als Argumente übergeben. Fügt dem *Stapel* alle Eingänge i' von Knoten v' hinzu, die nach den Ausgängen *j* folgen. Den nächsten Eingängen i' ist jeweils eine neu erzeugte Prozessstückzahl x'_a und ein Koeffizient q' gleich eins zugeordnet. Liefert die neue Zahl *m* der Prozessstückzahlen x_a zurück.

Prozedur TransformiereGrafischesModell({$G_{WS,b}$}, **P**, **F**, **L**, $\mathbf{L}_{Invest}$, *MaxFolge*) 001

Die übergeordnete Prozedur legt auf Grundlage eines gegebenen grafischen Modells {$G_{WS,b}$} die Komponenten der Prozessmatrix **P**, Flussmatrix **F**, Auslastungsmatrix **L** und Investitionsmatrix $\mathbf{L}_{Invest}$ fest. Entsprechend der Reihenfolge, in welcher die Prozessstückzahlen x_a nach Vorgabe des Anwenders zu maximieren sind, werden zudem die zugehörigen Indizes *a* in die *MaxFolge* eingefügt.

Prozedur TransformiereRessource($G_{WS,b}$, *b*, *InvestMenge*, *Stapel*, *v*, *i*, *a*, *q*) 046

Besucht den Eingang *i* des Knotens *v*, einer Ressource aus der Menge *R* im Wertstromgraphen $G_{WS,b}$ des Produkts *b*, und aktualisiert die *InvestMenge*. Die Prozessstückzahl x_a und der Koeffizient *q*, welche an Eingang *i* anliegen, werden als Argumente übergeben. Fügt dem *Stapel* den Eingang i' des Knotens v' hinzu, der nach dem jeweils gegenüberliegenden Ausgang *j* folgt. Dem nächsten Eingang i' ist wieder die Prozessstückzahl x_a und der Koeffizient *q* zugeordnet.

Prozedur TransformiereSLFlusspunkt($G_{WS,b}$, *b*, *InvestMenge*, *Stapel*, *v*, *a*, *q*) 050

Besucht den Eingang des Knotens *v*, eines selektiven Flusspunkts aus der Menge $D_{SL,b}$ im Wertstromgraphen $G_{WS,b}$ des Produkts *b*, und aktualisiert die *InvestMenge*. Die Prozessstückzahl x_a und der Koeffizient *q*, welche am Eingang des Flusspunkts *v* anliegen, werden als Argumente übergeben. Fügt dem *Stapel* alle Eingänge i' von Knoten v' hinzu, die nach den Ausgängen *j* folgen. Den nächsten Eingängen i' sind wieder die Prozessstückzahl x_a und jeweils ein Koeffizient q' zugeordnet, welcher aus der Multiplikation des Koeffizienten *q* vom Eingang mit der Quote an Ausgang *j* resultiert.

Funktion TransformiereWSGraphen($G_{WS,b}$, *b*, *m*, **P**, {$\mathbf{Q}_{ein,bv}$}, {$\mathbf{Q}_{aus,bv}$}, **U**, *InvestMenge*, *MaxFolge*): $\mathbb{N}$ 012

Traversiert den Wertstromgraphen $G_{WS,b}$ des Produkts *b* und legt basierend auf der aktuellen Zahl *m* der Prozessstückzahlen x_a die Komponenten der Prozessmatrix **P** fest. Als Zwischenergebnis werden die Eingangsmatrizen $\mathbf{Q}_{ein,bv}$, Ausgangsmatrizen $\mathbf{Q}_{aus,bv}$ und die Vereinigungsmatrix **U** bestimmt. Indem die Funktion für alle Produkte *b* aufgerufen wird, werden die *InvestMenge* und die *MaxFolge* fortlaufend ergänzt. Um auch bereits besuchte Eingänge *i* zu erfassen, die jeweils nach dem ersten Ausgang alternativer Flusspunkte $D_{AL,b}$ folgen, wird der Wertstromgraph $G_{WS,b}$ in zwei Modi traversiert. Liefert die neue Zahl *m* der Prozessstückzahlen x_a zurück.

Für mathematische Symbole siehe Abschnitt 4.4 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A21–A24 dargestellt.

Eingabe: (validiertes) grafisches Modell $\{G_{WS,b}\}$ aus Wertstromgraphen $G_{WS,b}$ mit Knoten $V_{WS,b}$ und Kanten $E_{WS,b}$ für alle Produkte $b \in \{1,\ldots,n\}$
Ausgabe: Prozessmatrix **P**, Flussmatrix **F**, Auslastungsmatrix **L**, Investitionsmatrix $\mathbf{L}_{\text{Invest}}$, *MaxFolge* mit Indizes a iterativ zu maximierender Prozessstückzahlen x_a

001: **Prozedur** TransformiereGrafischesModell($\{G_{WS,b}\}$, **P**, **F**, **L**, $\mathbf{L}_{\text{Invest}}$, *MaxFolge*)
002: **für** $b \leftarrow 1$ **bis** n **tue** /* betrachte alle Produkte b und initialisiere ... */
003: **für alle** $v \in V_{WS,b} \setminus \{\sigma_b, \tau_b\}$ **tue** /* ... für alle Knoten $V_{WS,b}$ außer Quelle σ_b und Senke τ_b ... */
004: $\mathbf{Q}_{\text{ein},bv} \leftarrow \mathbb{0}$ /* ... die Eingangsmatrix $\mathbf{Q}_{\text{ein},bv}$ und ... */
005: **für alle** $v \in D_{AL,b}$ **tue** /* ... für alle alternativen Flusspunkte $D_{AL,b}$ ... */
006: $\mathbf{Q}_{\text{aus},bv} \leftarrow \mathbb{0}$ /* ... die Ausgangsmatrix $\mathbf{Q}_{\text{aus},bv}$ */
007: $m \leftarrow 0$ /* initialisiere die Zahl m der Prozessstückzahlen x_a, ... */
008: **U** $\leftarrow \mathbb{0}$ /* ... die Vereinigungsmatrix **U**, ... */
009: *InvestMenge* $\leftarrow \{\}$ /* ... die *InvestMenge*, eine Menge von Tupeln (b, v, i), und ... */
010: *MaxFolge* $\leftarrow ()$ /* ... die *MaxFolge*, eine Folge von Indizes a der Prozessstückzahlen x_a */
011: **für** $b \leftarrow 1$ **bis** n **tue** /* betrachte alle Produkte b und bestimme ... */
/* ... die Prozessmatrix **P** und die initialisierten Matrizen, Mengen und Folgen */
012: $m \leftarrow$ TransformiereWSGraphen($G_{WS,b}, b, m, \mathbf{P}, \{\mathbf{Q}_{\text{ein},bv}\}, \{\mathbf{Q}_{\text{aus},bv}\}, \mathbf{U}$, *InvestMenge*, *MaxFolge*)
/* bestimme die Flussmatrix **F**, die Auslastungsmatrix **L** und die Investitionsmatrix $\mathbf{L}_{\text{Invest}}$ */
013: FülleFlussmatrix($\{G_{WS,b}\}, m, \mathbf{F}, \{\mathbf{Q}_{\text{ein},bv}\}, \{\mathbf{Q}_{\text{aus},bv}\}, \mathbf{U}$)
014: FülleAuslastungsmatrix($\{G_{WS,b}\}, \mathbf{L}, \mathbf{L}_{\text{Invest}}, \{\mathbf{Q}_{\text{ein},bv}\}$, *InvestMenge*)

(nächster Teil auf Seite A22)

Algorithmus A.1.8(a): Transformation eines grafischen Modells $\{G_{WS,b}\}$ (Teil 1 von 4).

(vorheriger Teil auf Seite A21)

```
015: Funktion TransformiereWSGraphen(G_WS,b, b, m, P, {Q_ein,bv}, {Q_aus,bv}, U, InvestMenge, MaxFolge)
016:     Stapel ← ()  /* initialisiere den Stapel, in diesem Fall eine Folge von Tupeln (v, i, a, q)  */
017:     m ← m + 1  /* erhöhe die Zahl m der Prozessstückzahlen x_a, ...  */
018:     a' ← m  /* ... um am Ausgang der Quelle σ_b eine neue Prozessstückzahl x_a' zu erzeugen ...  */
019:     P[b, a'] ← 1  /* ... und diese der Produktstückzahl y_b zuzuordnen  */
020:     v' ← ErmittleNächstenKnoten(G_WS,b, σ_b, 1)  /* wenn die Kante an der Quelle σ_b ...  */
021:     wenn v' <> τ_b dann  /* ... noch nicht zur Senke τ_b führt, folge dieser ...  */
022:         i' ← ErmittleNächstenEingangsindex(G_WS,b, σ_b, 1)  /* ... zu Eingang i'  */
023:         InvestMenge ← InvestMenge ∪ {(b, v', i')}  /* füge Eingang i' der InvestMenge hinzu ...  */
024:         Stapel.ErgänzeAlsLetztes((v', i', a', 1))  /* ... und besuche diesen zuallererst  */
025:     solange ¬Stapel.IstLeer() tue  /* solange Eingänge zu besuchen sind, ...  */
026:         (v, i, a, q) ← Stapel.EntferneLetztes()  /* ... betrachte den aktuellen Knoten v und Eingang i
                mit der Prozessstückzahl x_a und dem Koeffizienten q  */
027:         istEingangVereinigung ← [∃ e1 = (v1, j1, v'1, i'1), e2 = (v2, j2, v'2, i'2) ∈ E_WS,b
                    v'1 = v'2 ∧ i'1 = i'2 ∧ (v1 <> v2 ∨ j1 <> j2)]
028:         wenn istEingangVereinigung dann  /* vereinigen sich Kanten vor Eingang i und ...  */
029:             wenn Q_ein,bv[i, *] = 0^⊤ dann  /* ... wurde dieser noch nicht besucht, ...  */
030:                 m ← m + 1  /* ... erhöhe die Zahl m der Prozessstückzahlen x_a, um an Eingang i ...  */
031:                 a' ← m  /* ... eine neue Prozessstückzahl x_a' zu erzeugen  */
032:                 U[a', a] ← q  /* ergänze jeweils einen Eintrag in der Vereinigungsmatrix U
                        mit dem Koeffizienten q für Prozessstückzahl x_a und ...  */
033:                 U[a', a'] ← −1  /* ... dem Wert minus eins für Prozessstückzahl x_a'  */
034:                 a ← a'  /* setze im Entdeckungsmodus fort, d. h. mit der Prozessstückzahl x_a' ...  */
035:                 q ← 1  /* ... und einem Koeffizienten q gleich eins  */
036:             sonst  /* vereinigen sich Kanten an einem bereits besuchten Eingang i und ...  */
037:                 wenn a > 0 dann  /* ... wurde der vorhergehende Eingang erstmalig besucht, ...  */
038:                     a' ← {a': a' ∈ {1, ..., m} ∧ Q_ein,bv[i, a'] = 1}  /* ... bestimme den Index a'
                            der Prozessstückzahl x_a' nach der Vereinigung der Kanten an Eingang i  */
039:                     U[a', a] ← U[a', a] + q  /* aktualisiere die Vereinigungsmatrix U  */
040:                     a ← 0  /* setze im Investitionsmodus fort, d. h. ohne die Prozessstückzahl x_a ...  */
041:                     q ← 0  /* ... oder den Koeffizienten q zu spezifizieren  */
042:         wenn a > 0 dann  /* wird Eingang i erstmalig besucht (Entdeckungsmodus), ...  */
043:             Q_ein,bv[i, a] ← q  /* ... aktualisiere die Eingangsmatrix Q_ein,bv  */
044:         wenn a > 0 ∨ (b, v, i) ∈ InvestMenge dann  /* wenn Eingang i erstmalig besucht wird oder
                in der InvestMenge enthalten ist (Investitionsmodus), ...  */
045:             wenn v ∈ R dann  /* ... unterscheide, ob es sich bei Knoten v um eine Ressource, ...  */
046:                 TransformiereRessource(G_WS,b, b, InvestMenge, Stapel, v, i, a, q)
047:             sonst wenn v ∈ D_AL,b dann  /* ... einen alternativen Flusspunkt oder ...  */
048:                 m ← TransformiereALFlusspunkt(G_WS,b, b, m, Q_aus,bv, InvestMenge, MaxFolge, ...)
049:             sonst  /* ... einen selektiven Flusspunkt handelt  */
050:                 TransformiereSLFlusspunkt(G_WS,b, b, InvestMenge, Stapel, v, a, q)
051:     liefere m zurück
```

(nächster Teil auf Seite A23)

Algorithmus A.1.8(b): Transformation eines grafischen Modells $\{G_{WS,b}\}$ (Teil 2 von 4).

(vorheriger Teil auf Seite A22)

```
052: Prozedur TransformiereRessource(G_WS,b, b, InvestMenge, Stapel, v, i, a, q)
053:     j ← i  /* betrachte nur den Ausgang j, welcher dem Eingang i gegenüberliegt */
054:     v' ← ErmittleNächstenKnoten(G_WS,b, v, j)  /* wenn die jeweilige Kante an Ausgang j ... */
055:     wenn v' <> τ_b dann  /* ... noch nicht zur Senke τ_b führt, folge dieser ... */
056:         i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j)  /* ... zu Eingang i' */
057:         wenn a > 0 dann  /* Eingang i erstmalig besucht (Entdeckungsmodus) */
058:             wenn (b, v, i) ∈ InvestMenge dann  /* ist Eingang i in der InvestMenge enthalten, ... */
059:                 InvestMenge ← InvestMenge ∪ {(b, v', i')}  /* ... füge Eingang i' hinzu */
060:             a' ← a  /* übernimm die Prozessstückzahl x_a und ... */
061:             q' ← q  /* ... den Koeffizienten q von Eingang i */
062:             Stapel.ErgänzeAlsLetztes((v', i', a', q'))  /* besuche als Nächstes Eingang i' */
063:         sonst  /* Eingang i wiederholt besucht (Investitionsmodus) */
064:             wenn ¬(b, v', i') ∈ InvestMenge dann  /* ist Eingang i' nicht in der InvestMenge, ... */
065:                 InvestMenge ← InvestMenge ∪ {(b, v', i')}  /* ... füge ihn hinzu und ... */
066:                 Stapel.ErgänzeAlsLetztes((v', i', 0, 0))  /* ... besuche diesen als Nächstes */

067: Funktion TransformiereALFlusspunkt(G_WS,b, b, m, Q_aus,bv, InvestMenge, MaxFolge, Stapel, v, a, q)
068:     wenn a > 0 dann  /* Eingang des Knotens v erstmalig besucht (Entdeckungsmodus) */
069:         wenn (b, v, 1) ∈ InvestMenge dann  /* ist der Eingang in der InvestMenge und ... */
070:             v' ← ErmittleNächstenKnoten(G_WS,b, v, 1)  /* ... führt die Kante am ersten Ausgang ... */
071:             wenn v' <> τ_b dann  /* ... noch nicht zur Senke τ_b, folge dieser ... */
072:                 i' ← ErmittleNächstenEingangsindex(G_WS,b, v, 1)  /* ... zu Eingang i' */
073:                 InvestMenge ← InvestMenge ∪ {(b, v', i')}  /* füge Eingang i' der InvestMenge hinzu */
074:         ausgangszahl ← ErmittleAusgangszahl(G_WS,b, v)  /* folge in umgekehrter Reihenfolge ... */
075:         für j ← ausgangszahl absteigend bis 1 tue  /* ... den Kanten an allen Ausgängen j */
076:             a' ← m + j  /* erzeuge eine neue Prozessstückzahl x_a' an Ausgang j des Knotens v ... */
077:             Q_aus,bv[j, a'] ← 1  /* ... und aktualisiere die Ausgangsmatrix Q_aus,bv */
078:             v' ← ErmittleNächstenKnoten(G_WS,b, v, j)  /* wenn die Kante an Ausgang j ... */
079:             wenn v' <> τ_b dann  /* ... noch nicht zur Senke τ_b führt, folge dieser ... */
080:                 i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j)  /* ... zu Eingang i' */
081:                 Stapel.ErgänzeAlsLetztes((v', i', a', 1))  /* besuche als Nächstes diesen Eingang i' */
082:         für j ← 1 bis ausgangszahl − 1 tue  /* nimm alle neuen Prozessstückzahlen x_a' ... */
083:             MaxFolge.ErgänzeAlsLetztes(m + j)  /* ... außer der letzten in die MaxFolge auf */
084:         m ← m + ausgangszahl  /* erhöhe die Zahl m der Prozessstückzahlen x_a */
085:     sonst  /* Eingang des Knotens v wiederholt besucht (Investitionsmodus) */
086:         v' ← ErmittleNächstenKnoten(G_WS,b, v, 1)  /* wenn die Kante am ersten Ausgang ... */
087:         wenn v' <> τ_b dann  /* ... noch nicht zur Senke τ_b führt, folge dieser ... */
088:             i' ← ErmittleNächstenEingangsindex(G_WS,b, v, 1)  /* ... zu Eingang i' */
089:             wenn ¬(b, v', i') ∈ InvestMenge dann  /* ist Eingang i' nicht in der InvestMenge, ... */
090:                 InvestMenge ← InvestMenge ∪ {(b, v', i')}  /* ... füge ihn hinzu und ... */
091:                 Stapel.ErgänzeAlsLetztes((v', i', 0, 0))  /* ... besuche diesen als Nächstes */
092:     liefere m zurück
```

(nächster Teil auf Seite A24)

Algorithmus A.1.8(c): Transformation eines grafischen Modells $\{G_{WS,b}\}$ (Teil 3 von 4).

(vorheriger Teil auf Seite A23)

```
093: Prozedur TransformiereSLFlusspunkt(G_WS,b, b, InvestMenge, Stapel, v, a, q)
094:     ausgangszahl ← ErmittleAusgangszahl(G_WS,b, v)  /* folge in umgekehrter Reihenfolge ...  */
095:     für j ← ausgangszahl absteigend bis 1 tue  /* ... den Kanten an allen Ausgängen j  */
096:         v' ← ErmittleNächstenKnoten(G_WS,b, v, j)  /* wenn die Kanten an diesen Ausgängen j ...  */
097:         wenn v' <> τ_b dann  /* ... noch nicht zur Senke τ_b führen, folge ihnen jeweils ...  */
098:             i' ← ErmittleNächstenEingangsindex(G_WS,b, v, j)  /* ... zu Eingang i'  */
099:             wenn a > 0 dann  /* Eingang des Knotens v erstmalig besucht (Entdeckungsmodus)  */
100:                 wenn (b, v, 1) ∈ InvestMenge dann  /* ist der Eingang des Knotens v
                        in der InvestMenge enthalten, füge entsprechend Eingang i' ...  */
101:                     InvestMenge ← InvestMenge ∪ {(b, v', i')}  /* ... des Knotens v' hinzu  */
102:                 a' ← a  /* übernimm die Prozessstückzahl x_a vom Eingang ...  */
103:                 q' ← q ErmittleQuote(G_WS,b, v, j)  /* ... und multipliziere den Koeffizienten q
                        mit der Quote, welche für Ausgang j definiert ist  */
104:                 Stapel.ErgänzeAlsLetztes((v', i', a', q'))  /* besuche als Nächstes Eingang i'  */
105:             sonst  /* Eingang des Knotens v wiederholt besucht (Investitionsmodus)  */
106:                 wenn ¬(b, v', i') ∈ InvestMenge dann  /* ist Eingang i' nicht in der InvestMenge, ...  */
107:                     InvestMenge ← InvestMenge ∪ {(b, v', i')}  /* ... füge ihn hinzu und ...  */
108:                     Stapel.ErgänzeAlsLetztes((v', i', 0, 0))  /* ... besuche diesen als Nächstes  */

109: Prozedur FülleFlussmatrix({G_WS,b}, m, F, {Q_ein,bv}, {Q_aus,bv}, U)
110:     zeile ← 0
111:     für a' ← 1 bis m tue  /* für Vereinigungen von Kanten an Eingängen von Knoten V_WS,b gilt, ...  */
112:         wenn U[a', *] <> 0^⊤ dann  /* ... dass die Summe aller Stückzahlen q x_a ...  */
113:             zeile ← zeile + 1  /* ... vor der Vereinigung gleich der Prozessstückzahl x_a' ...  */
114:             F[zeile, *] ← U[a', *]  /* ... nach der Vereinigung sein muss  */
115:     für b ← 1 bis n tue  /* für alternative Flusspunkte D_AL,b gilt, ...  */
116:         für alle v ∈ D_AL,b tue  /* ... dass die Stückzahl q x_a am Eingang des Knotens gleich ...  */
117:             zeile ← zeile + 1  /* ... der Summe der Prozessstückzahlen x_a' ...  */
118:             F[zeile, *] ← Q_ein,bv − 1^⊤ Q_aus,bv  /* ... an allen Ausgängen sein muss  */

119: Prozedur FülleAuslastungsmatrix({G_WS,b}, L, L_Invest, {Q_ein,bv}, InvestMenge)
120:     zeile ← 0
121:     für alle v ∈ R tue  /* ermittle für alle Ressourcen R ...  */
122:         T_BM ← ErmittleBetriebsmittelzeit(v)  /* ... die Betriebsmittelzeit T_BM  */
123:         zeile ← zeile + 1  /* erzeuge eine neue Zeile für jede Ressource  */
124:         für b ← 1 bis n tue  /* ermittle in jedem Wertstromgraphen G_WS,b ...  */
125:             für i ← 1 bis ErmittleEingangszahl(G_WS,b, v) tue  /* ... für alle Eingänge i
                    von Ressourcen, d. h. für alle Prozessschritte des Produkts, ...  */
126:                 t_eff ← ErmittleEffektiveTaktzeit(G_WS,b, v, i)  /* ... die effektive Taktzeit t_eff  */
127:                 L[zeile, *] ← L[zeile, *] + t_eff / T_BM Q_ein,bv[i, *]  /* erweitere die Matrix L  */
128:                 wenn ¬(b, v, i) ∈ InvestMenge dann  /* wenn Eingang i nicht in der InvestMenge
                        enthalten ist, erweitere entsprechend die Matrix L_Invest  */
129:                     L_Invest[zeile, *] ← L_Invest[zeile, *] + t_eff / T_BM Q_ein,bv[i, *]
```

Algorithmus A.1.8(d): Transformation eines grafischen Modells $\{G_{WS,b}\}$ (Teil 4 von 4).

A.2 Algorithmen, Teil II: mathematische Optimierung

Im zweiten Teil werden die Algorithmen beschrieben, die dazu dienen, ein mathematisches Modell zu optimieren. Den Anfang bildet dabei die Minimierung einer linearen Zielfunktion, bevor darauf aufbauend die drei definierten Planungsziele folgen.

A.2.1 Minimierung einer allgemeinen linearen Zielfunktion $f(\mathbf{x})$

Der einführende Algorithmus verdeutlicht beispielhaft zwei Schritte, die nachfolgend zur Optimierung der Planungsziele aufgegriffen und fallweise abgewandelt werden. Im ersten Schritt wird ausgehend vom Nullvektor eine beliebige Startlösung $\mathbf{x}$ ermittelt, welche bestimmte Bedingungen erfüllt. In diesem Fall muss die Lösung neben dem Flusserhalt und den Schranken bezüglich der Auslastung den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entsprechen. Im zweiten Schritt wird eine gegebene Zielfunktion $f(\mathbf{x})$ minimiert, wobei die zuvor genannten Bedingungen weiterhin gelten müssen (siehe Abschnitt 5.1).

Funktionen und Prozeduren — Zeilennummer

Funktion MinLineareZielfunktion$(\mathbf{c}, \mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L}) : \mathbb{R}$ — 1

Bestimmt den minimalen Wert der linearen Zielfunktion $f(\mathbf{x})$, welche durch den Ausdruck $\mathbf{c}^\top \mathbf{x}$ definiert ist, und liefert diesen zurück. Für die gesuchte Lösung $\mathbf{x}$ gilt, dass die resultierenden Stückzahlen y_b aller Produkte b den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entsprechen müssen. Zudem gelten die Bedingungen in Bezug auf Flusserhalt und Auslastung, repräsentiert durch die Matrizen $\mathbf{F}$ bzw. $\mathbf{L}$. Voraussetzung ist, dass unter den vorgegebenen Bedingungen eine Lösung $\mathbf{x}$ existiert.

Für mathematische Symbole siehe Abschnitt 5.1 und Symbolverzeichnis.

Eingabe: Vektoren der Koeffizienten $\mathbf{c}$ und geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$, Prozessmatrix $\mathbf{P}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$
Ausgabe: minimaler Wert der Zielfunktion $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$ unter den vorgegebenen Bedingungen (vorausgesetzt, dass eine entsprechende Lösung $\mathbf{x}$ existiert)

1: **Funktion** MinLineareZielfunktion$(\mathbf{c}, \mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L})$
2: $\quad \mathbf{x} \leftarrow \mathbf{0}$ /* verwende den Nullvektor als Startlösung */
/* bestimme eine beliebige Lösung $\mathbf{x}$, welche den geplanten Produktstückzahlen $y_{\text{plan},b}$ entspricht und die vorgegebenen Bedingungen erfüllt */
3: $\quad \mathbf{x} \leftarrow \arg\max\{\mathbf{1}^\top \mathbf{P}\mathbf{x} : \mathbf{x} \geqslant \mathbf{0} \wedge$ /* $\mathbf{1}^\top \mathbf{P}\mathbf{x} = \sum_{b=1}^{n} y_b$ */
$\qquad \mathbf{P}\mathbf{x} \leqslant \mathbf{y}_{\text{plan}} \wedge$ /* Maximierung jeder Produktstückzahl y_b bis zum Wert $y_{\text{plan},b}$ */
$\qquad \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \mathbf{L}\mathbf{x} \leqslant \mathbf{1}\}$ /* Flusserhalt bzw. Auslastung */
/* bestimme eine Lösung $\mathbf{x}$, welche die Zielfunktion $f(\mathbf{x})$ minimiert, wobei die Stückzahlen y_b unverändert den geplanten Stückzahlen $y_{\text{plan},b}$ entsprechen müssen */
4: $\quad \mathbf{x} \leftarrow \arg\min\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \geqslant \mathbf{0} \wedge$ /* $\mathbf{c}^\top \mathbf{x} = f(\mathbf{x})$ */
$\qquad \mathbf{P}\mathbf{x} = \mathbf{y}_{\text{plan}} \wedge$ /* Beibehaltung der geplanten Stückzahlen $y_{\text{plan},b}$ aller Produkte b */
$\qquad \mathbf{F}\mathbf{x} = \mathbf{0} \wedge \mathbf{L}\mathbf{x} \leqslant \mathbf{1}\}$ /* Flusserhalt bzw. Auslastung */
5: $\quad$ **liefere $\mathbf{c}^\top \mathbf{x}$ zurück**

Algorithmus A.2.1: Minimierung einer allgemeinen linearen Zielfunktion $f(\mathbf{x})$.

A.2.2 Maximierung der technischen Kapazitäten $\mathbf{y}_{\max}$

Der Algorithmus bestimmt die technischen Kapazitäten $\mathbf{y}_{\max}$, welche das Ergebnis des Planungsziels der maximalen Kapazitäten repräsentieren. Beginnend mit dem Nullvektor werden in aufeinanderfolgenden Iterationsschritten die Stückzahlen y_b all derjenigen Produkte b maximiert, die noch nicht maximal sind. Eine Stückzahl y_b gilt genau dann als noch nicht maximal, wenn sie unter den gegebenen Bedingungen erhöht werden kann, ohne die Stückzahlen anderer Produkte zu reduzieren. In jedem Iterationsschritt erreicht mindestens eine Produktstückzahl y_b ihren maximalen Wert, worauf dieser beibehalten wird. Bei der Maximierung zweier Stückzahlen y_b muss deren Verhältnis jenem der geplanten Produktstückzahlen $y_{\text{plan},b}$ entsprechen. Die Iteration wird fortgeführt, solange die Stückzahl y_b mindestens eines Produkts b nicht maximal ist (siehe Abschnitt 5.2).

Objekte und Variablen	Zeilennummer
Variable *istMax*[b]: {`falsch`, `wahr`}	04
Eindimensionales Feld boolescher Werte mit Index b. Ist genau dann `wahr`, wenn die Produktstückzahl y_b maximal ist, sonst `falsch`. Siehe Funktion IstMaxKapazität(…).	

Funktionen und Prozeduren	Zeilennummer
Funktion IstMaxKapazität($\mathbf{x}, b, \mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}$): {`falsch`, `wahr`}	12
Prüft, ob die Stückzahl y_b des Produkts b erhöht werden kann, ohne die Stückzahlen anderer Produkte zu reduzieren. Es gelten die Bedingungen in Bezug auf Produktstruktur, Flusserhalt und Auslastung, repräsentiert durch die Matrizen **S**, **F** bzw. **L**. Um eine Lösung zu suchen, welche den genannten Bedingungen entspricht, wird der übergebene Vektor **x** als Startlösung verwendet. Liefert genau dann `wahr` zurück, wenn die Produktstückzahl y_b unter diesen Bedingungen maximal ist, sonst `falsch`.	
Funktion MaxKapazitäten($\mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}$): $\mathbb{R}^n_{\geqslant 0}$	01
Bestimmt die technischen Kapazitäten $\mathbf{y}_{\max}$, d. h. die maximalen Stückzahlen y_b aller Produkte b, und liefert diese zurück. Solange zwei Produktstückzahlen y_b nicht maximal sind, muss deren Verhältnis jenem der geplanten Produktstückzahlen $y_{\text{plan},b}$ entsprechen. Erreicht die Stückzahl y_b eines Produkts b ihren maximalen Wert, wird dieser Wert nicht verändert. Zusätzlich gelten die Bedingungen in Bezug auf Produktstruktur, Flusserhalt und Auslastung, repräsentiert durch die Matrizen **S**, **F** bzw. **L**. Voraussetzung ist, dass alle Produktstückzahlen y_b nach oben beschränkt sind.	

Für mathematische Symbole siehe Abschnitt 5.2 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A27 dargestellt.

Eingabe: Vektor (vorläufiger) geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$, Prozessmatrix $\mathbf{P}$, Produktstrukturmatrix $\mathbf{S}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$
Ausgabe: Vektor maximaler Produktstückzahlen $\mathbf{y}_{\max}$ unter den vorgegebenen Bedingungen (vorausgesetzt, dass die Produktstückzahlen y_b nach oben beschränkt sind)

```
01: Funktion MaxKapazitäten(y_plan, P, S, F, L)
02:     x ← 0  /* verwende den Nullvektor als Startlösung */
03:     für b ← 1 bis n tue  /* initialisiere alle Produktstückzahlen y_b ... */
04:         istMax[b] ← falsch  /* ... als nicht maximal */
05:     p ← 1  /* wähle das erste Produkt als Referenzprodukt p */
06:     wiederhole
07:         x' ← x  /* speichere die vorherige Lösung x */
            /* bestimme eine Lösung x, welche die Produktstückzahl y_p in einem fixen Verhältnis
               zu allen anderen noch nicht maximalen Produktstückzahlen y_b maximiert */
08:         x ← arg max{ P[p,*]x : x ≥ 0 ∧  /* P[p,*]x = y_p
                  ∀b ∈ {1,...,n} \ {p}  /* für alle anderen Produktstückzahlen y_b gilt ...
                      ¬istMax[b] → (y_plan,p P[b,*] − y_plan,b P[p,*])x = 0 ∧  /* ... dieses Verhältnis ...
                        istMax[b] → P[b,*]x = P[b,*]x' ∧  /* ... bzw. der jeweils vorherige Wert
                  (I − S)Px ≥ 0 ∧ Fx = 0 ∧ Lx ≤ 1 }  /* Produktstruktur, Flusserhalt bzw. Auslastung */
            /* prüfe alle zuvor nicht maximalen Produktstückzahlen y_b, ob diese nun maximal sind,
               und bestimme ein Referenzprodukt p, dessen Stückzahl y_p nicht maximal ist */
09:         p ← 0  /* setze den Index p des Referenzprodukts p zurück, um ein neues zu suchen */
10:         für b ← 1 bis n tue  /* betrachte alle Produktstückzahlen y_b, ... */
11:             wenn ¬istMax[b] dann  /* ... die zuvor nicht maximal waren, und ... */
12:                 wenn IstMaxKapazität(x, b, P, S, F, L) dann  /* ... ist das nun der Fall, ... */
13:                     istMax[b] ← wahr  /* ... markiere diese entsprechend */
14:                 sonst wenn p = 0 dann  /* wähle das erste Produkt b, dessen Stückzahl y_b ... */
15:                     p ← b  /* ... nicht maximal ist, als neues Referenzprodukt p */
16:     solange p > 0  /* wiederhole, solange mindestens eine Produktstückzahl y_b nicht maximal ist */
17:     y_max ← Px  /* bestimme die technischen Kapazitäten y_max gemäß der Lösung x */
18:     liefere y_max zurück

19: Funktion IstMaxKapazität(x, b, P, S, F, L)
20:     y_b ← P[b,*]x  /* speichere zum Vergleich den aktuellen Wert der Produktstückzahl y_b */
21:     x' ← x  /* verwende die zuvor ermittelte Lösung x als Startlösung */
        /* maximiere die Produktstückzahl y'_b unabhängig von den Stückzahlen aller anderen Produkte,
           wobei deren Werte jedoch nicht reduziert werden dürfen */
22:     y'_b ← max{ P[b,*]x' : x' ≥ 0 ∧  /* P[b,*]x' = y'_b
                  Px' ≥ Px ∧  /* mindestens Beibehaltung aller Produktstückzahlen y_b
                  P[b,*]x' < y_b + ε ∧  /* höchstens Steigerung um einen gegebenen Wert größer als null
                  (I − S)Px' ≥ 0 ∧ Fx' = 0 ∧ Lx' ≤ 1 }  /* Produktstruktur, Flusserhalt bzw. Auslastung */
23:     wenn y'_b ≤ y_b dann  /* ist der neue Wert nicht größer als die zu prüfende Produktstückzahl y_b, ... */
24:         liefere wahr zurück  /* ... muss diese maximal sein */
25:     liefere falsch zurück
```

Algorithmus A.2.2: Maximierung der technischen Kapazitäten $\mathbf{y}_{\max}$.

A.2.3 Minimierung der Überlastung Lx (Komponenten größer als eins)

Gemäß dem Planungsziel der minimalen Investitionen bestimmt der Algorithmus die Überlastung, die notwendig ist, um die geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ herzustellen. Die Auslastung der Ressourcen wird durch die Werte des Vektors **Lx** wiedergegeben, wobei jede Komponente des Vektors einer Ressource entspricht. Gesucht sind die minimalen oberen Schranken größer als oder gleich eins für die Komponenten des Vektors **Lx**, für welche unter den gegebenen Bedingungen eine Lösung **x** existiert. Der Algorithmus bewirkt einen Ausgleich der Überlastung zwischen den Ressourcen (siehe Abschnitt 5.3).

Für alle Werte des Vektors **Lx**, von denen vor Beginn der Iteration bekannt ist, dass diese höchstens gleich eins sein können, gilt eine obere Schranke von eins. In jedem Iterationsschritt wird unter allen Werten des Vektors **Lx**, die noch nicht minimal sind, der größte Wert minimiert, wobei alle anderen diesen nicht überschreiten dürfen. Ein Wert wird genau dann als minimal bezeichnet, wenn seine minimale obere Schranke feststeht. Das heißt, die obere Schranke ist gleich eins, oder der Wert kann gegenüber seiner oberen Schranke nicht reduziert werden, ohne andere Werte über deren obere Schranken zu steigern. Nach jedem Iterationsschritt wird für alle noch nicht minimalen Werte der größte unter all diesen Werten als obere Schranke festgelegt. Als Folge ist nach jedem Iterationsschritt mindestens ein weiterer Wert minimal. Die Iteration wird fortgesetzt, solange mindestens ein Wert des Vektors **Lx** noch nicht minimal und größer als eins ist.

Objekte und Variablen — Zeilennummer

Variable $istMin[i]: \{\texttt{falsch}, \texttt{wahr}\}$ — 07

Eindimensionales Feld boolescher Werte mit Index i. Ist genau dann `wahr`, wenn der Wert des Vektors **Lx** an Zeile i minimal ist, sonst `falsch`. Siehe Funktion IstMinÜberlastung(...).

Variable $minMax[i]: \mathbb{R}_{>0}$ — 12

Eindimensionales Feld positiver reeller Werte mit Index i. Bezeichnet die minimale obere Schranke größer als oder gleich eins, welche für die Komponente des Vektors **Lx** an Zeile i festgelegt ist.

Variable $zeilenzahl: \mathbb{N}$ — 03

Natürliche Zahl, unter den gegebenen Voraussetzungen größer als null. Gibt die größte Zeile der Matrix **L** ungleich dem Nullvektor an. Siehe Funktion ErmittleZeilenzahl(**L**).

Funktionen und Prozeduren — Zeilennummer

Funktion ErmittleZeilenzahl(**L**): $\mathbb{N}$ — 03

Liefert die größte Zeile der Matrix **L** ungleich dem Nullvektor zurück. Da vorausgesetzt wird, dass alle Produktstückzahlen y_b nach oben beschränkt sind, existiert mindestens eine solche Zeile.

Funktion InitMinÜberlastung($\mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L}_{\text{Invest}}$): $\mathbb{R}^m_{\geq 0}$ — 02

Bestimmt eine initiale Lösung **x**, welche den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht, und liefert diese zurück. Einschränkend gilt hierbei, dass nur bestimmte Prozessstückzahlen x_a größer als null sein dürfen, wie durch die Matrix $\mathbf{L}_{\text{Invest}}$ festgelegt. Darüber hinaus muss die Lösung **x** die Bedingung in Bezug auf den Flusserhalt erfüllen, repräsentiert durch die Matrix **F**.

Funktion IstMinÜberlastung($\mathbf{x}, \mathbf{P}, \mathbf{F}, \mathbf{L}, i, minMax$): $\{\texttt{falsch}, \texttt{wahr}\}$ — 21

Prüft, ob der Wert des Vektors **Lx** an Zeile i reduziert werden kann, ohne andere Werte über ihre oberen Schranken zu steigern. Diesbezüglich gilt einschränkend, dass die Stückzahlen y_b aller Produkte b beibehalten werden müssen. Des Weiteren muss jede Lösung die Bedingung in Bezug auf den Flusserhalt

erfüllen, repräsentiert durch die Matrix **F**. Um eine entsprechende Lösung zu suchen, wird der übergebene Vektor **x** als Startlösung verwendet. Die Funktion wird nur dann aufgerufen, wenn der Wert des Vektors **Lx** an Zeile i größer als eins ist und seiner oberen Schranke entspricht. Liefert genau dann `wahr` zurück, wenn der Wert an Zeile i unter diesen Bedingungen minimal ist, sonst `falsch`.

Funktion MinÜberlastung($\mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L}, \mathbf{L}_{\text{Invest}}$): $\mathbb{R}_{>0}[]$ 01

Bestimmt die minimale Überlastung, d. h. die minimalen oberen Schranken größer als oder gleich eins für die Komponenten des Vektors **Lx**, und liefert diese zurück. Die Werte des Vektors **Lx** geben die Auslastung der Ressourcen an, wobei nur ausgewählte zu überlasten sind, wie durch die Matrix $\mathbf{L}_{\text{Invest}}$ festgelegt. Es wird jeweils der größte nicht minimale Wert größer als eins minimiert, während andere Werte, die nicht minimal sind, diesen nicht überschreiten dürfen. Für jede Lösung **x** gilt, dass die Stückzahlen y_b aller Produkte b den geplanten Produktstückzahlen $y_{\text{plan},b}$ entsprechen müssen. Weiterhin gilt die Bedingung in Bezug auf den Flusserhalt, repräsentiert durch die Matrix **F**.

Für mathematische Symbole siehe Abschnitt 5.3 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A30–A31 dargestellt.

Eingabe: Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$, Prozessmatrix $\mathbf{P}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$, Investitionsmatrix $\mathbf{L}_{\text{Invest}}$
Ausgabe: minimale obere Schranken größer als oder gleich eins für die Komponenten des Vektors $\mathbf{Lx}$, für welche unter den vorgegebenen Bedingungen eine Lösung $\mathbf{x}$ existiert

```
01: Funktion MinÜberlastung(y_plan, P, F, L, L_Invest)
02:     x ← InitMinÜberlastung(y_plan, P, F, L_Invest)  /* bestimme die Startlösung */
03:     zeilenzahl ← ErmittleZeilenzahl(L)  /* da oft verwendet, speichere die Zeilenzahl der Matrix L */
04:     j ← 0  /* initialisiere Zeile j mit dem größten nicht minimalen Wert des Vektors Lx größer als eins */
05:     für i ← 1 bis zeilenzahl tue  /* betrachte die Werte des Vektors Lx an allen Zeilen i */
06:         wenn L[i,*]x > 0 dann  /* ist der Wert größer als null, ... */
07:             istMin[i] ← falsch  /* ... initialisiere diesen als nicht minimal */
08:             wenn (j = 0 ∧ L[i,*]x > 1) ∨  /* ist dieser größer als eins und ... */
                        (j > 0 ∧ L[j,*]x < L[i,*]x) dann  /* ... größer als jener an Zeile j, ... */
09:                 j ← i  /* ... aktualisiere entsprechend Zeile j */
10:         sonst  /* ist der Wert des Vektors Lx an Zeile i gleich null, ... */
11:             istMin[i] ← wahr  /* ... initialisiere diesen als minimal und ... */
12:             minMax[i] ← 1  /* ... lege den Wert eins als Schranke fest */
13:     solange j > 0 tue  /* solange mindestens ein Wert nicht minimal und größer als eins ist, ... */
            /* ... bestimme eine entsprechende Lösung x, welche den Wert des Vektors Lx an Zeile j
               minimiert, wobei alle anderen Werte nach oben beschränkt sind */
14:         x ← arg min{L[j,*]x : x ≥ 0 ∧  /* Wert des Vektors Lx an Zeile j */
                    Px = y_plan ∧ Fx = 0 ∧  /* Beibehaltung der Stückzahlen y_plan,b bzw. Flusserhalt */
                    ∀i ∈ {1,...,zeilenzahl} \ {j}  /* begrenze die Werte an anderen Zeilen i durch ... */
                        ¬istMin[i] → (L[i,*] − L[j,*])x ≤ 0 ∧  /* ... den Wert an Zeile j bzw. ... */
                            istMin[i] → L[i,*]x ≤ minMax[i]}  /* ... die jeweils festgelegte Schranke */
15:         j' ← 0  /* initialisiere nächste Zeile j' mit dem größten nicht minimalen Wert größer als eins */
16:         wenn L[j,*]x > 1 dann  /* ist der Wert an Zeile j weiterhin größer als eins, ... */
17:             für i ← 1 bis zeilenzahl tue  /* ... lege als Schranke für alle nicht minimalen Werte ... */
18:                 wenn ¬istMin[i] dann minMax[i] ← L[j,*]x  /* ... den Wert an Zeile j fest */
19:             für i ← 1 bis zeilenzahl tue  /* betrachte die Werte des Vektors Lx an allen Zeilen i, ... */
20:                 wenn ¬istMin[i] dann  /* ... die zuvor nicht minimal waren */
21:                     wenn L[i,*]x = L[j,*]x ∧  /* ist der Wert gleich jenem an Zeile j und ... */
                                IstMinÜberlastung(x, P, F, L, i, minMax) dann  /* ... nun minimal, ... */
22:                         istMin[i] ← wahr  /* ... markiere diesen entsprechend */
23:                     sonst wenn (j' = 0 ∧ L[i,*]x > 1) ∨  /* bestimme die nächste Zeile j', ... */
                                (j' > 0 ∧ L[j',*]x < L[i,*]x) dann  /* ... an welcher sich der größte ... */
24:                         j' ← i  /* ... nicht minimale Wert größer als eins befindet, ... */
25:         j ← j'  /* ... und übernimm diese als aktuelle Zeile j */
26:     für i ← 1 bis zeilenzahl tue  /* da alle verbleibenden Werte nicht größer als eins sein können, ... */
27:         wenn ¬istMin[i] dann minMax[i] ← 1  /* ... lege den Wert eins als Schranke fest */
28:     liefere minMax zurück
```

(nächster Teil auf Seite A31)

Algorithmus A.2.3(a): Minimierung der Überlastung $\mathbf{Lx}$ (Komponenten größer als eins) (Teil 1 von 2).

(vorheriger Teil auf Seite A30)

```
29: Funktion InitMinÜberlastung(y_plan, P, F, L_Invest)
30:     x ← 0  /* verwende den Nullvektor als Startlösung                                   */
        /* bestimme eine Lösung x, welche den geplanten Produktstückzahlen y_plan,b entspricht,
           wobei nur definierte Prozessstückzahlen x_a größer als null sein dürfen        */
31:     x ← arg max{ 1^T P x : x ≥ 0 ∧  /* 1^T P x = Σ_{b=1}^{n} y_b                      */
                P x ≤ y_plan ∧  /* Maximierung jeder Produktstückzahl y_b bis zum Wert y_plan,b */
                F x = 0 ∧ L_Invest x = 0 }  /* Flusserhalt bzw. Beschränkung aller Prozessstückzahlen x_a
                   auf den Wert null, für welche die Matrix L_Invest an Spalte a Einträge enthält */
32:     liefere x zurück

33: Funktion IstMinÜberlastung(x, P, F, L, i, minMax)
34:     zeilenzahl ← ErmittleZeilenzahl(L)  /* speichere die Zeilenzahl der Matrix L      */
35:     x' ← x  /* verwende die aktuelle Lösung x als Startlösung                         */
        /* bestimme eine Lösung x', welche den Wert des Vektors L x' an Zeile i minimiert und
           für alle anderen Zeilen die jeweils festgelegten Schranken berücksichtigt     */
36:     x' ← arg min{ L[i,*] x' : x' ≥ 0 ∧  /* Wert des Vektors L x' an Zeile i          */
                P x' = P x ∧ F x' = 0 ∧  /* Beibehaltung aller Produktstückzahlen y_b bzw. Flusserhalt */
                L[i,*] x' ≥ L[i,*] x − ε ∧  /* höchstens Reduktion um einen Wert größer als null */
                ∀ i' ∈ {1,...,zeilenzahl} \ {i}  /* für alle anderen Zeilen i' gilt ...   */
                    L[i',*] x' ≤ minMax[i']}  /* ... die jeweils festgelegte Schranke     */
37:     wenn L[i,*] x' ≥ L[i,*] x dann  /* ist der neue Wert an Zeile i nicht kleiner als der zu prüfende, ... */
38:         liefere wahr zurück  /* ... muss letzterer minimal sein                       */
39:     liefere falsch zurück
```

Algorithmus A.2.3(b): Minimierung der Überlastung $\mathbf{L}\mathbf{x}$ (Komponenten größer als eins) (Teil 2 von 2).

A.2.4 Optimierung der Auslastung Lx (alle Komponenten)

Der Algorithmus setzt das Planungsziel der optimalen Auslastung um, welches auf der iterativen Maximierung definierter Prozessstückzahlen x_a basiert. Die Reihenfolge, nach welcher die Prozessstückzahlen x_a maximiert werden, entspricht der vorgegebenen Priorisierung des Anwenders und wird durch die *MaxFolge* abgebildet. Die Folge enthält, wie in Abschnitt A.1.8 erläutert wurde, die Indizes a der entsprechend zu maximierenden Prozessstückzahlen x_a. Im Verlauf der Iteration werden die Lösungen, die in vorherigen Iterationsschritten gefunden wurden, als Nebenbedingungen übernommen. Für die Lösungen $\mathbf{x}$ aller Iterationsschritte gilt, dass diese den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entsprechen müssen. Weiterhin müssen die Lösungen $\mathbf{x}$ die zuvor ermittelten, minimalen oberen Schranken $minMax[i]$ für die Komponenten des Vektors $\mathbf{Lx}$ berücksichtigen (siehe Abschnitt 5.4).

Objekte und Variablen	Zeilennummer
Objekt *MaxFolge*: $\{(a_1, a_2, \ldots) : a \in \{1, \ldots, m\}\}$	01
Siehe Abschnitt A.1.8.	
Variable $minMax[i] : \mathbb{R}_{>0}$	01
Siehe Abschnitt A.2.3.	
Variable *zeilenzahl*: $\mathbb{N}$	03
Siehe Abschnitt A.2.3.	

Funktionen und Prozeduren	Zeilennummer
Funktion ErmittleZeilenzahl($\mathbf{L}$): $\mathbb{N}$	03
Siehe Abschnitt A.2.3.	
Funktion InitOptAuslastung($\mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L}, minMax$): $\mathbb{R}^m_{\geq 0}$	02
Bestimmt eine initiale Lösung $\mathbf{x}$, welche den geplanten Produktstückzahlen $\mathbf{y}_{\text{plan}}$ entspricht, und liefert diese zurück. Die gesuchte Lösung $\mathbf{x}$ muss die Bedingung in Bezug auf den Flusserhalt erfüllen, welche durch die Matrix $\mathbf{F}$ repräsentiert wird. Zudem dürfen die Komponenten des Vektors $\mathbf{Lx}$ die übergebenen minimalen oberen Schranken $minMax[i]$ nicht überschreiten.	
Funktion *MaxFolge*.EntferneErstes(): $\{1, \ldots, m\}$	11
Entfernt das zuerst hinzugefügte Element der *MaxFolge* und liefert dieses zurück, welches wie definiert den Index a einer Prozessstückzahl x_a bezeichnet. Siehe Erläuterung zum Objekt *MaxFolge*.	
Funktion *MaxFolge*.IstLeer(): {`falsch`, `wahr`}	07
Liefert genau dann `wahr` zurück, wenn die Länge der *MaxFolge* gleich null ist, d. h., wenn diese einem leeren Tupel entspricht, sonst `falsch`. Siehe Erläuterung zum Objekt *MaxFolge*.	
Funktion OptAuslastung($\mathbf{y}_{\text{plan}}, \mathbf{P}, \mathbf{F}, \mathbf{L}, minMax, MaxFolge$): $\mathbb{R}^m_{\geq 0}$	01
Bestimmt die optimale Auslastung, d. h. die optimalen Werte aller Prozessstückzahlen x_a gemäß den definierten Prioritäten, und liefert diese zurück. Die Prozessstückzahlen x_a werden iterativ maximiert, wie durch die *MaxFolge* vorgegeben, wobei die Lösungen vorheriger Iterationsschritte Nebenbedingungen darstellen. Für die Lösungen $\mathbf{x}$ aller Iterationsschritte gilt, dass die Stückzahlen y_b der Produkte b den geplanten Produktstückzahlen $y_{\text{plan},b}$ entsprechen müssen. Des Weiteren muss die Bedingung in Bezug auf den Flusserhalt erfüllt sein, repräsentiert durch die Matrix $\mathbf{F}$. Die Komponenten des Vektors $\mathbf{Lx}$ dürfen die minimalen oberen Schranken $minMax[i]$ nicht überschreiten.	

Für mathematische Symbole siehe Abschnitt 5.4 und Symbolverzeichnis. Der Pseudocode des Algorithmus ist auf Seite A33 dargestellt.

Eingabe: Vektor geplanter Produktstückzahlen $\mathbf{y}_{\text{plan}}$, Prozessmatrix $\mathbf{P}$, Flussmatrix $\mathbf{F}$, Auslastungsmatrix $\mathbf{L}$, minimale obere Schranken *minMax*, *MaxFolge*
Ausgabe: Lösung $\mathbf{x}$, welche unter den vorgegebenen Bedingungen die Prozessstückzahlen x_a gemäß der Sortierung der *MaxFolge* maximiert

```
01: Funktion OptAuslastung(y_plan, P, F, L, minMax, MaxFolge)
02:     x ← InitOptAuslastung(y_plan, P, F, L, minMax)  /* bestimme die Startlösung */
03:     zeilenzahl ← ErmittleZeilenzahl(L)  /* da oft verwendet, speichere die Zeilenzahl der Matrix L */
04:     A ← ∅  /* initialisiere die Matrix A zur Beibehaltung maximierter Prozessstückzahlen x_a, ... */
05:     j ← 0  /* ... die im vorherigen Iterationsschritt ergänzte Zeile j und ... */
06:     a ← 0  /* ... den Index a der jeweils zu maximierenden Prozessstückzahl x_a */
07:     solange ¬MaxFolge.IstLeer() tue  /* wiederhole, solange mindestens eine Prozessstückzahl x_a
            entsprechend der MaxFolge zu maximieren ist */
08:         wenn a > 0 dann  /* wurde vorher eine Prozessstückzahl x_a maximiert, ... */
09:             j ← j + 1  /* ... ergänze fortlaufend in einer neuen Zeile j ... */
10:             A[j, a] ← 1  /* ... der Matrix A an Spalte a einen Eintrag gleich eins */
11:         a ← MaxFolge.EntferneErstes()  /* betrachte die aktuelle Prozessstückzahl x_a */
12:         x' ← x  /* speichere die vorherige Lösung x */
            /* bestimme eine Lösung x, welche die Prozessstückzahl x_a maximiert, wobei die Werte
               aller vorherigen Iterationsschritte beibehalten werden müssen */
13:         x ← arg max{x_a: x ≥ 0 ∧  /* aktuelle Prozessstückzahl x_a */
                    Ax = Ax' ∧  /* Beibehaltung aller vorher maximierten Prozessstückzahlen x_a */
                    Px = Px' ∧ Fx = 0 ∧  /* Beibehaltung aller Produktstückzahlen y_b bzw. Flusserhalt */
                    ∀i ∈ {1, ..., zeilenzahl}  /* für alle Zeilen i des Vektors Lx gilt ... */
                        L[i, *]x ≤ minMax[i]}  /* ... jeweils die minimale obere Schranke */
14:     liefere x zurück

15: Funktion InitOptAuslastung(y_plan, P, F, L, minMax)
16:     zeilenzahl ← ErmittleZeilenzahl(L)  /* speichere die Zeilenzahl der Matrix L */
17:     x ← 0  /* verwende den Nullvektor als Startlösung */
        /* bestimme eine Lösung x, welche den geplanten Produktstückzahlen y_plan,b entspricht und
           die minimalen oberen Schranken des Vektors Lx berücksichtigt */
18:     x ← arg max{1^T Px: x ≥ 0 ∧  /* 1^T Px = Σ_{b=1}^{n} y_b */
                Px ≤ y_plan ∧  /* Maximierung jeder Produktstückzahl y_b bis zum Wert y_plan,b */
                Fx = 0 ∧  /* Bedingung in Bezug auf den Flusserhalt */
                ∀i ∈ {1, ..., zeilenzahl}  /* für alle Zeilen i des Vektors Lx gilt ... */
                    L[i, *]x ≤ minMax[i]}  /* ... jeweils die minimale obere Schranke */
19:     liefere x zurück
```

Algorithmus A.2.4: Optimierung der Auslastung $\mathbf{Lx}$ (alle Komponenten).

Abkürzungsverzeichnis

AIMMS	Advanced Interactive Multidimensional Modeling System
AMER	Nord- und Südamerika (Wirtschaftsraum)
AML	Algebraic Modeling Language
AMPL	A Mathematical Programming Language
APAC	Asien-Pazifik (Wirtschaftsraum)
APMonitor	Advanced Process Monitor
APO	Advanced Planning and Optimization
ASCEND	Advanced System for Computations in Engineering Design
AURELIE	Advanced Utilization of Resources and Locations for Industrial Engineering
BAZ	Bearbeitungszentrum
BE	bewegliches Element
BPM	Business Process Modeling
BPMN	Business Process Model and Notation
BPR	Business Process Reengineering
CPU	Central Processing Unit
CSCMP	Council of Supply Chain Management Professionals
DES	diskrete ereignisorientierte Simulation
DIN	Deutsches Institut für Normung e. V.
EMEA	Europa, Naher Osten (engl. Middle East) und Afrika (Wirtschaftsraum)
EPC	Event-Driven Process Chain
ERP	Enterprise Resource Planning
FIFO	First In, First Out
GA	genetischer Algorithmus
GAMS	General Algebraic Modeling System
GNU	GNU's Not Unix, Betriebssystem und freie Software
HANA	Entwicklungsplattform von SAP auf Basis einer In-Memory-Datenbank (ehemals High Performance Analytic Appliance)
IBP	Integrated Business Planning
IBM	International Business Machines
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
LIFO	Last In, First Out
LINDO	Linear, Interactive, and Discrete Optimizer
LINGO	AML zur Nutzung mit LINDO
LP	lineare Programmierung
LPL	Linear Programming Language

MILP	Mixed-Integer Linear Programming
MAE	Maschine, Anlage oder Einrichtung
MATLAB	Matrix Laboratory, Software zur Lösung mathematischer Probleme
MINOPT	Mixed-Integer Nonlinear Optimizer
MPL	Mathematical Programming Language
MPSX/370	Mathematical Programming System Extended/370
NOP	Eingabeformat für nichtlineare Optimierungsprobleme
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group
OMNI	AML mit Funktionen zur Verwaltung von Modellen
OPL	Optimization Programming Language
OSL	Optimization Subroutine Library
PCOMP	AML für nichtlineare Programme mit automatischer Differenzierung
PDS	Production Data Structure
PEK	Personalkapazität
PG	Produktgruppe
PP/DS	Production Planning and Detailed Scheduling
PPM	Production Process Model
PRD	Produkt
PS	Prozessschritt
PSO	Partikelschwarmoptimierung
PZ	Planungsziel
Pyomo	Python-Softwarepakete zur Erstellung von Optimierungsmodellen
RAM	Random Access Memory
RCP	Rich Client Platform
REFA	Verband für Arbeitsgestaltung, Betriebsorganisation und Unternehmensentwicklung
SAMPL	Stochastic AMPL
SAP	Anbieter betriebswirtschaftlicher Standardsoftware
SCM	Supply Chain Management
SNP	Supply Network Planning
SO	simulationsbasierte Optimierung
TEK	technische Kapazität
TOMLAB	AML zur Optimierung in MATLAB
TP/VS	Transportation Planning and Vehicle Scheduling
UML AD	Unified Modeling Language Activity Diagram
VBA	Visual Basic for Applications
VDI	Verein Deutscher Ingenieure e. V.
VSM	Value Stream Mapping
WS-BPEL	Web Services Business Process Execution Language
XML	Extensible Markup Language
YAWL	Yet Another Workflow Language

Symbolverzeichnis

Symbolklassen und Operatoren

t, T	fix vorgegebene Zeitgröße (Konstante im Kontext dieser Arbeit)
x	Skalar oder Element einer Menge
X	Menge $\{x_1, x_2, \dots\}$, Tupel $(x_1, x_2, \dots)$ oder (logische) Aussage
$X[i]$	Glied einer Folge X mit Index $i \in \mathbb{N}$ (auch Komponente genannt)
$\lvert X \rvert$	Kardinalität der Menge X (auch Mächtigkeit genannt)
$x \in X$	x ist Element der Menge X
$x \notin X$	x ist kein Element der Menge X
$\lceil x \rceil$	kleinste natürliche Zahl größer als oder gleich x
x'	Vergleichswert oder weiterer Wert neben x (aus derselben Menge)
x^*	optimaler Wert von x (gemäß gegebener Zielfunktion)
$x^{(i)}$	Wert von x im Rekursionsschritt bzw. Iterationsschritt $i \in \mathbb{N}_0$
$\mathbb{X}$	Zahlenkörper (Menge)
$\sum_{i \in I} x_i$	Summe (Addition) der Zahlen x_i mit Index $i \in I$
$\prod_{i \in I} x_i$	Produkt (Multiplikation) der Zahlen x_i mit Index $i \in I$
$f(x, \dots, z)$	Funktion f mit den Argumenten $x, \dots, z$
$f: X \to Y, x \mapsto y$	Funktion f als Abbildung von $x \in X$ auf $y \in Y$
$\mathrm{FN}(x, \dots, z)$	(insbesondere algorithmische) Funktion FN, siehe $f(x, \dots, z)$
$A(x, \dots, z)$	(logische) Aussageform A mit den Variablen $x, \dots, z$
$\{x \in X: A(x)\}$	Menge aller $x \in X$, für welche $A(x)$ wahr ist (X entfällt, falls eindeutig)
$\forall x \in X\ A(x)$	Allquantor (für alle $x \in X$ ist $A(x)$ wahr)
$\exists x \in X\ A(x)$	Existenzquantor (für mindestens ein $x \in X$ ist $A(x)$ wahr)
$x \leftarrow y$	(algorithmische) Zuweisung des Wertes von y zu x
$x <> y$	Ungleichheit der Werte von x und y (verwendet in Algorithmen)
$\neg A$	Negation (A ist nicht wahr)
$A \wedge B$	Konjunktion (sowohl A als auch B sind wahr)
$A \vee B$	Disjunktion (A oder B ist wahr)
$A \to B$	Konditional (wenn A wahr ist, dann ist B wahr)
$A \leftrightarrow B$	Bikonditional (genau dann, wenn A wahr ist, ist B wahr)
$A \Leftrightarrow B$	Äquivalenz der Aussagen A und B
$X = Y$	Gleichheit der Mengen X und Y, $\forall x\ (x \in X \leftrightarrow x \in Y)$
$X \neq Y$	Ungleichheit der Mengen X und Y, $\exists x\ (x \in X \wedge x \notin Y) \vee (x \notin X \wedge x \in Y)$
$X \subseteq Y$	X ist eine Teilmenge von Y, $\forall x \in X\ (x \in Y)$
$X \subset Y$	X ist eine echte Teilmenge von Y, $X \subseteq Y \wedge X \neq Y$
$X \cap Y$	Schnittmenge von X und Y, $\{x: x \in X \wedge x \in Y\}$
$X \cup Y$	Vereinigung der Mengen X und Y, $\{x: x \in X \vee x \in Y\}$

$X \dot\cup Y$	Vereinigung disjunkter Mengen X und Y, $X \cup Y$ mit $\neg\exists x\,(x \in X \wedge x \in Y)$
$X \setminus Y$	Differenz der Mengen X und Y, $\{x\colon x \in X \wedge x \notin Y\}$
$X \times Y$	kartesisches Produkt der Mengen X und Y, $\{(x,y)\colon x \in X \wedge y \in Y\}$
$\mathcal{P}(X)$	Potenzmenge der Menge X, $\{Y\colon Y \subseteq X\}$
$\max X$, $\min X$	größtes bzw. kleinstes Element (Maximum bzw. Minimum) der Menge X
$\max\{f(x)\colon\ldots\}$, $\min\{f(x)\colon\ldots\}$	Maximum bzw. Minimum der Funktion $f(x)$ für alle x einer gegebenen, durch Bedingungen beschränkten Menge (entfällt, falls eindeutig)
$\arg\max\{\ldots\}$, $\arg\min\{\ldots\}$	Stelle, an welcher eine Funktion gemäß jeweils vorgegebenen Bedingungen ihr Maximum bzw. Minimum annimmt, siehe max bzw. $\min\{f(x)\colon\ldots\}$
$\mathbf{x}$	Spaltenvektor $\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$, $\mathbf{x} \in \mathbb{R}^m$
$\mathbf{x} \leqslant \mathbf{y}$	Ungleichheit der Komponenten von $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$, $\forall i \in \{1,\ldots,m\}\ x_i \leqslant y_i$
$\mathbf{x} \geqslant \mathbf{y}$	s. o., $\forall i \in \{1,\ldots,m\}\ x_i \geqslant y_i$
$\mathbf{X}$	Matrix $\begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix}$, $\mathbf{X} \in \mathbb{R}^{m\times n}$
$\mathbf{X}^\top$	transponierte Matrix $\begin{bmatrix} x_{11} & \cdots & x_{m1} \\ \vdots & \ddots & \vdots \\ x_{1n} & \cdots & x_{mn} \end{bmatrix}$, $\mathbf{X}^\top \in \mathbb{R}^{n\times m}$, insbesondere
	Zeilenvektor $[x_1\ \ldots\ x_n] = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}^\top$ und Spaltenvektor $\begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} = [x_1\ \ldots\ x_m]^\top$
$\mathbf{X}[i,j]$	Komponente der Matrix $\mathbf{X}$ an Zeile $i \in \mathbb{N}$ und Spalte $j \in \mathbb{N}$, $\mathbf{X}[i,j] \in \mathbb{R}$
$\mathbf{X}[i,*]$	Vektor aller Komponenten der Matrix $\mathbf{X}$ an Zeile $i \in \mathbb{N}$, $\mathbf{X}[i,*] \in \mathbb{R}^{1\times n}$
$\mathbf{X}[*,j]$	Vektor aller Komponenten der Matrix $\mathbf{X}$ an Spalte $j \in \mathbb{N}$, $\mathbf{X}[*,j] \in \mathbb{R}^{m\times 1}$

Standardsymbole

$()$	leeres Tupel
$\varnothing$	leere Menge, äquivalent zu $\{\}$
Δ	Differenzoperator
$\square$	Schluss eines Beweises
$\mathbf{0}$	Nullvektor $[0\ \ldots\ 0]^\top$
$\mathbb{0}$	Nullmatrix $\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$
$\mathbf{1}$	Einsvektor $[1\ \ldots\ 1]^\top$

$\mathbf{I}$	Einheitsmatrix $\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{bmatrix}$
$\mathbb{N}$	Menge der natürlichen Zahlen größer als 0, $\{1, 2, \ldots\}$
$\mathbb{N}_0$	Menge der natürlichen Zahlen größer als oder gleich 0, $\{0, 1, \ldots\}$
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}_{\geqslant 0}$	Menge der nichtnegativen reellen Zahlen, $\{x \in \mathbb{R} \colon x \geqslant 0\}$
$\mathbb{R}_{>0}$	Menge der positiven reellen Zahlen, $\{x \in \mathbb{R} \colon x > 0\}$
$\mathbb{R}^m$	Menge der Vektoren im reellen Koordinatenraum mit m Dimensionen
$\mathbb{R}^m_{\geqslant 0}$	Menge der Vektoren im nichtnegativen reellen Koordinatenraum mit m Dimensionen, $\{\mathbf{x} \in \mathbb{R}^m \colon \mathbf{x} \geqslant \mathbf{0}\}$
$\mathbb{R}^{m \times n}$	Menge der Matrizen mit reellen Komponenten (auch Einträge oder Elemente genannt), angeordnet in m Zeilen und n Spalten
$[a, b]$	reelles Intervall $\{x \in \mathbb{R} \colon a \leqslant x \leqslant b\}$
$[a, b)$	s. o., $\{x \in \mathbb{R} \colon a \leqslant x < b\}$
$(a, b]$	s. o., $\{x \in \mathbb{R} \colon a < x \leqslant b\}$
(a, b)	s. o., $\{x \in \mathbb{R} \colon a < x < b\}$

Eigene Symbole (lateinisch)

Hinweis: Vektoren und Matrizen werden in den Algorithmen in dieser Arbeit als zweidimensionale Felder mit dynamischer Größe, d. h. mit einer theoretisch unbegrenzten Zahl an Zeilen und Spalten, verwendet. Das heißt, grundsätzlich ist jeder Komponente an einer Zeile und einer Spalte größer als oder gleich eins ein Wert zugeordnet. Wenn im Verlauf eines Algorithmus einer Komponente an einer gegebenen Zeile und Spalte kein Wert ungleich null zugewiesen wurde, wird ein Wert von null vorausgesetzt.

a, a'	Index zur Bezeichnung der aktuellen bzw. nächsten Prozessstückzahl x_a bzw. $x_{a'}$, je nach vorliegendem Anwendungsfall konkretisiert, $a, a' \in \{1, \ldots, m\}$, ggf. mit Index 1, 2 usw., S. 138
$\mathbf{A}$	Matrix reeller Koeffizienten eines Vektors $\mathbf{x}$, je nach vorliegendem Anwendungsfall konkretisiert, $\forall i \in \{1, 2, \ldots\}, a \in \{1, \ldots, m\}\ \mathbf{A}[i, a] \in \mathbb{R}$, S. 173
$\mathrm{AL}(c_1, c_2)$	alternative Verknüpfung zweier Operanden $c_{1/2} \in \mathrm{VMg}(P)$, $\mathrm{AL}(c_1, c_2) \in \mathrm{VMg}(P)$, siehe Gleichung (2.1), S. 16
b, b'	Index zur Bezeichnung eines gegebenen bzw. übergeordneten Produkts, $b, b' \in \{1, \ldots, n\}$, ggf. mit Index 1, 2 usw., S. 117
$\mathbf{b}$	Vektor reeller oberer Schranken, $\mathbf{Ax} \leqslant \mathbf{b}$, $\forall i \in \{1, 2, \ldots\}\ \mathbf{b}[i, 1] \in \mathbb{R}$, S. 173
$B_{\max}$	Menge der Indizes aller Produkte b, deren Stückzahl y_b im aktuellen Iterationsschritt maximal ist, $B_{\max} \subseteq \{1, \ldots, n\}$, S. 179
c	kontextabhängig Operand und/oder Ergebnis einer Verknüpfung, $c \in \mathrm{VMg}(P)$, oder konstanter, nichtnegativer reeller Faktor, $c \in \mathbb{R}_{\geqslant 0}$, ggf. mit Index 1, 2 usw., siehe Gleichung (2.1), S. 16
c^*	Element maximaler Ordnung einer Teilmenge C, $c^* \in C$, S. 18

C	Teilmenge einer Verknüpfungsmenge, beschreibt einen geplanten Ablauf zur Fertigung und Montage, $C \subset \mathrm{VMg}(P)$, z. B. {PS 1, PS 2, PS 3, AL(PS 1, PS 2), SQ(AL(PS 1, PS 2), PS 3)}, S. 18
$\mathbf{c}$	Vektor reeller Koeffizienten eines Vektors $\mathbf{x}$ zur Definition einer linearen Zielfunktion $f(\mathbf{x})$, $\forall a \in \{1,\ldots,m\}\ \mathbf{c}[a,1] \in \mathbb{R}$, S. 173
$D_{\mathrm{AL},b}$, $D_{\mathrm{SL},b}$	Menge aller alternativen bzw. selektiven Flusspunkte von $G_{\mathrm{WS},b}$, $D_{\mathrm{AL},b}$ bzw. $D_{\mathrm{SL},b} \subset V_{\mathrm{WS},b}$, S. 117
$d_G^-(v)$, $d_G^+(v)$	Eingangsgrad bzw. Ausgangsgrad des Knotens v, $d_G^-(v), d_G^+(v) \in \mathbb{N}_0$, S. 112
$d_{\max}^+$	maximaler Ausgangsgrad aller Flusspunkte $D_{\mathrm{AL},b}$ und $D_{\mathrm{SL},b}$, S. 135
e, e'	aktuelle bzw. nächste Kante, $e, e' \in E_{\mathrm{WS},b}$, ggf. mit Index 1, 2 usw., S. 119
E	Menge der Kanten (engl. *edge*, Plural *edges*) von G, S. 110
$E_{\mathrm{WS},b}$	Kantenmenge von $G_{\mathrm{WS},b}$, $E_{\mathrm{WS},b} \subset \{(v,j,v',i') :\ v, v' \in V_{\mathrm{WS},b} \wedge j, i' \in \mathbb{N}\}$, S. 117
$\mathbf{F}$	Flussmatrix, Matrix eines mathematischen Modells $(\mathbf{P},\mathbf{S},\mathbf{F},\mathbf{L},\mathbf{L}_{\mathrm{Invest}})$, $\forall i \in \{1,2,\ldots\}, a \in \{1,\ldots,m\}\ \mathbf{F}[i,a] \in [-1,1]$, S. 138
$f(n)$	asymptotische, obere oder untere Schranke für Laufzeit oder Speicherplatzbedarf, abhängig von den Eingabedaten n, $f(n) \in \mathbb{R}_{\geqslant 0}$, S. 114
$f(\mathbf{x})$	lineare Zielfunktion für einen Vektor $\mathbf{x}$, $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$, $f(\mathbf{x}) \in \mathbb{R}$, S. 172
G	endlicher Graph mit Knoten V und Kanten E, $G = (V,E)$, S. 110
$g(n)$	Funktion zur Beschreibung von Laufzeit oder Speicherplatzbedarf, abhängig von den Eingabedaten n, $g(n) \in \mathbb{R}_{\geqslant 0}$, S. 114
$G_{\mathrm{WS},b}$	Wertstromgraph des Produkts b, $G_{\mathrm{WS},b} = (V_{\mathrm{WS},b}, E_{\mathrm{WS},b})$, S. 117
$\{G_{\mathrm{WS},b}\}$	grafisches Modell eines Systems von Wertströmen, bezeichnet die Menge aller Wertstromgraphen $G_{\mathrm{WS},b}$, Abkürzung für $\{G_{\mathrm{WS},b} :\ b \in \{1,\ldots,n\}\}$, S. 117
i, i', j, j', k	natürliche Zahl, jeweils verwendet mit kontextabhängiger Bedeutung, insbesondere als Index, $i, i', j, j', k \in \mathbb{N}$ oder $\mathbb{N}_0$, ggf. mit Index 1, 2 usw., S. 17
$I_{\min}$	Menge der Zeilen i, an denen der Wert des Vektors $\mathbf{Lx}$ im aktuellen Iterationsschritt minimal ist, $I_{\min} \subseteq \{1,\ldots,\mathrm{N}(\mathbf{L})\}$, S. 192
$\mathbf{L}$, $\mathbf{L}_{\mathrm{Invest}}$	Auslastungsmatrix bzw. Investitionsmatrix, jeweils Matrizen eines mathematischen Modells $(\mathbf{P},\mathbf{S},\mathbf{F},\mathbf{L},\mathbf{L}_{\mathrm{Invest}})$, $\forall i \in \{1,2,\ldots\}, a \in \{1,\ldots,m\}$ $\mathbf{L}[i,a], \mathbf{L}_{\mathrm{Invest}}[i,a] \in \mathbb{R}_{\geqslant 0}$, S. 138
m, n	obere Schranke eines Intervalls natürlicher Zahlen, je nach vorliegendem Anwendungsfall mit kontextabhängiger Bedeutung, $m, n \in \mathbb{N}$, S. 20
$\mathrm{M}(X)$	Menge der Glieder einer Folge $X = (x_1, x_2, \ldots)$, $\mathrm{M}(X) = \{x_1, x_2, \ldots\}$, S. 213
N	Stufe in der Strukturstückliste eines Produkts b, $N \in \{1,\ldots,n\}$, S. 140
$\mathrm{N}(\mathbf{X})$	größte Zeile einer Matrix $\mathbf{X}$ ungleich dem Nullvektor, $\mathrm{N}(\mathbf{X}) \in \mathbb{N}_0$, S. 192
$\mathrm{o}(f(n))$	Komplexitätsklasse, $\{g(n) : \forall c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geqslant n_0\ \ g(n) \leqslant c\, f(n)\}$, S. 115
$\mathrm{O}(f(n))$	s. o., $\{g(n) : \exists c > 0\ \exists n_0 \in \mathbb{N}_0\ \forall n \geqslant n_0\ \ g(n) \leqslant c\, f(n)\}$, S. 114
p	Index zur Bezeichnung des Referenzprodukts im aktuellen Iterationsschritt, beliebig ausgewählt unter allen Produkten b, deren Stückzahl y_b nicht maximal ist, $p \in \{1,\ldots,n\}$, S. 180
P	Menge der Prozessschritte eines Produkts, z. B. {PS 1, PS 2, …}, S. 16
$\mathbf{P}$	Prozessmatrix, Matrix eines mathematischen Modells $(\mathbf{P},\mathbf{S},\mathbf{F},\mathbf{L},\mathbf{L}_{\mathrm{Invest}})$, $\forall b \in \{1,\ldots,n\}, a \in \{1,\ldots,m\}\ \mathbf{P}[b,a] \in \{0,1\}$, S. 138

q	kontextabhängig fixe Quote einer selektiven Verknüpfung, $q \in (0,1)$, oder Koeffizient einer Prozessstückzahl x_a, $q \in [0,1]$, ggf. mit Index 1, 2 usw., siehe Gleichung (2.2), S. 16
q'	kontextabhängig resultierende Quote verketteter selektiver Verknüpfungen, $q' \in (0,1)$, oder Koeffizient einer Prozessstückzahl $x_{a'}$, $q' \in [0,1]$, ggf. mit Index 1, 2 usw., siehe Gleichung (2.8), S. 22
$\mathbf{Q}_{\text{aus},bv}$	Ausgangsmatrix des Knotens v im Wertstromgraphen $G_{\text{WS},b}$ des Produkts b, Hilfsmatrix zur Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$, $\forall j \in \{1,2,\ldots\}, a \in \{1,\ldots,m\}\ \mathbf{Q}_{\text{aus},bv}[j,a] \in [0,1]$, S. 142
$\{\mathbf{Q}_{\text{aus},bv}\}$	Menge der Ausgangsmatrizen $\mathbf{Q}_{\text{aus},bv}$ für alle Produkte b und alle alternativen Flusspunkte $D_{\text{AL},b}$ in Wertstromgraphen $G_{\text{WS},b}$, Abkürzung für $\{\mathbf{Q}_{\text{aus},bv} : b \in \{1,\ldots,n\} \wedge v \in D_{\text{AL},b}\}$, S. 143
$\mathbf{Q}_{\text{ein},bv}$	Eingangsmatrix des Knotens v im Wertstromgraphen $G_{\text{WS},b}$ des Produkts b, Hilfsmatrix zur Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$, $\forall i \in \{1,2,\ldots\}, a \in \{1,\ldots,m\}\ \mathbf{Q}_{\text{ein},bv}[i,a] \in [0,1]$, S. 142
$\{\mathbf{Q}_{\text{ein},bv}\}$	Menge der Eingangsmatrizen $\mathbf{Q}_{\text{ein},bv}$ für alle Produkte b und alle Knoten $V_{\text{WS},b}$ in Wertstromgraphen $G_{\text{WS},b}$ außer Quellen σ_b und Senken τ_b, Abkürzung für $\{\mathbf{Q}_{\text{ein},bv} : b \in \{1,\ldots,n\} \wedge v \in V_{\text{WS},b} \setminus \{\sigma_b,\tau_b\}\}$, S. 143
r	Maschine, Anlage oder Einrichtung, z. B. Bearbeitungszentrum, $r \in R$, ggf. mit Index 1, 2 usw., S. 35
R	Menge aller verfügbaren Maschinen, Anlagen und Einrichtungen, z. B. {'Bearbeitungszentrum', 'Drehmaschine', …}, S. 35
$\mathbf{S}$	Produktstrukturmatrix, Matrix eines mathematischen Modells ($\mathbf{P}, \mathbf{S}, \mathbf{F}, \mathbf{L}$, $\mathbf{L}_{\text{Invest}}$), $\forall b, b' \in \{1,\ldots,n\}\ \mathbf{S}[b,b'] \in \mathbb{R}_{\geqslant 0}$, S. 138
$\text{SL}(c_1, c_2, q)$	selektive Verknüpfung zweier Operanden $c_{1/2} \in \text{VMg}(P)$ mit einer fixen Quote $q \in (0,1)$, $\text{SL}(c_1, c_2, q) \in \text{VMg}(P)$, siehe Gleichung (2.2), S. 16
$\text{SQ}(c_1, c_2)$	sequenzielle Verknüpfung zweier Operanden $c_{1/2} \in \text{VMg}(P)$, $\text{SQ}(c_1, c_2) \in \text{VMg}(P)$, siehe Gleichung (2.1), S. 16
T_{BM}	Betriebsmittelzeit, $\text{T}_{\text{BM}} \in \mathbb{R}_{>0}$, ggf. mit Index 1, 2 usw., S. 35
t_{eff}	effektive, d. h. im langfristigen Mittel erreichbare Taktzeit, $\text{t}_{\text{eff}} \in \mathbb{R}_{>0}$, siehe Gleichung (2.17), S. 35
$\text{t}_{\text{eff},i}$	effektive Taktzeit, kontextabhängig mit Index $i \in \mathbb{N}$ eines sequenziell verknüpften oder fortlaufend indizierten Prozessschritts, $\text{t}_{\text{eff},i} \in \mathbb{R}_{>0}$, S. 42
$\text{t}_{\text{eff},i\alpha}$	effektive Taktzeit mit zweiteiligem Index i/α, $i \in \mathbb{N}, \alpha \in \{\text{'A'}, \text{'B'}, \ldots\}$, bezeichnet einen sequenziell, alternativ und/oder selektiv verknüpften Prozessschritt, $\text{t}_{\text{eff},i\alpha} \in \mathbb{R}_{>0}$, S. 45
t_{opt}	optimale, d. h. im besten Fall erreichbare Taktzeit, $\text{t}_{\text{opt}} \in \mathbb{R}_{>0}$, S. 33
$\mathbf{U}$	Vereinigungsmatrix, Hilfsmatrix zur Transformation eines grafischen Modells $\{G_{\text{WS},b}\}$, $\forall a', a \in \{1,\ldots,m\}\ \mathbf{U}[a',a] \in [-1,1]$, S. 141
$u_r(\mathbf{x})$	Auslastung der Maschine, Anlage oder Einrichtung r für einen gegebenen Vektor $\mathbf{x}$ von Prozessstückzahlen, $u_r(\mathbf{x}) \in \mathbb{R}_{\geqslant 0}$, S. 40
$u_r(y)$	Auslastung der Maschine, Anlage oder Einrichtung r für eine gegebene Produktstückzahl y, $u_r(y) \in \mathbb{R}_{\geqslant 0}$, siehe Gleichung (2.20), S. 37

v, v'	aktueller bzw. nächster Knoten, kontextabhängig $v, v' \in V$ oder $V_{\mathrm{WS},b}$, ggf. mit Index 1, 2 usw., S. 111
V	Menge der Knoten (engl. *vertex*, Plural *vertices*) von G, S. 110
$\mathrm{VMg}(P)$	Verknüpfungsmenge von P, z. B. $\{\mathrm{PS}\,1, \mathrm{PS}\,2, \mathrm{PS}\,3, \ldots, \varnothing, \mathrm{SQ}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{AL}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{SL}(\mathrm{PS}\,1, \mathrm{PS}\,2, q), \ldots, \mathrm{SQ}(\mathrm{SQ}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{PS}\,3), \ldots : q \in (0,1)\}$, siehe Gleichung (2.4), S. 18
$\mathrm{VMg}(P, i)$	Verknüpfungsmenge von P einer Ordnung $i \in \mathbb{N}_0$, z. B. $\mathrm{VMg}(P,0) = \{\mathrm{PS}\,1, \mathrm{PS}\,2, \ldots, \varnothing\}$, $\mathrm{VMg}(P,1) = \{\mathrm{SQ}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{AL}(\mathrm{PS}\,1, \mathrm{PS}\,2), \mathrm{SL}(\mathrm{PS}\,1, \mathrm{PS}\,2, q), \ldots : q \in (0,1)\}$ usw., siehe Gleichung (2.3a), S. 17
$V_{\mathrm{WS},b}$	Knotenmenge von $G_{\mathrm{WS},b}$, $V_{\mathrm{WS},b} = R \cup D_{\mathrm{AL},b} \cup D_{\mathrm{SL},b} \cup \{\sigma_b, \tau_b\}$, S. 117
$\mathbf{x}$, $\mathbf{x}'$	Vektor $[x_{\alpha_1} \ \ldots \ x_{\alpha_m}]^\top$ oder $[x_1 \ \ldots \ x_m]^\top$ der Prozessstückzahlen x_α bzw. x_a mit kontextabhängigem Index, ggf. mit Akzent, $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^m_{\geqslant 0}$, S. 40
$\mathbf{x}^*$	Vektor $[x^*_{\alpha_1} \ \ldots \ x^*_{\alpha_m}]^\top$ oder $[x^*_1 \ \ldots \ x^*_m]^\top$ mit kontextabhängigem Index, optimale Lösung gemäß gegebenem Planungsziel, $\mathbf{x}^* \in \mathbb{R}^m_{\geqslant 0}$, S. 49
x_α, x'_α, x_a, x'_a	Prozessstückzahl mit Index $\alpha \in \{\text{'A'}, \text{'B'}, \ldots\}$ einer alternativen Verknüpfung bzw. fortlaufendem Index $a \in \{1, \ldots, m\}$, $x_\alpha, x'_\alpha, x_a, x'_a \in \mathbb{R}_{\geqslant 0}$, S. 44
x^*_α	optimale Teillösung gemäß gegebenem Planungsziel, $x^*_\alpha \in \mathbb{R}_{\geqslant 0}$, S. 46
y, y', y_b, y'_b	Produktstückzahl, $y, y', y_b, y'_b \in \mathbb{R}_{\geqslant 0}$, ggf. näher bezeichnet durch Index eines Produkts $b \in \{1, \ldots, n\}$, S. 37
$\mathbf{y}$, $\mathbf{y}'$	Vektor $[y_1 \ \ldots \ y_n]^\top$ der Stückzahlen y_b aller gegebenen Produkte b, ggf. mit Akzent, $\mathbf{y}, \mathbf{y}' \in \mathbb{R}^n_{\geqslant 0}$, S. 139
$y_{\max}$, $y_{\max,b}$	maximale Produktstückzahl (technische Kapazität), $y_{\max} \in \mathbb{R}_{\geqslant 0}$, ggf. mit Index eines Produkts $b \in \{1, \ldots, n\}$, siehe Gleichung (2.18), S. 36
$\mathbf{y}_{\max}$	Vektor $[y_{\max,1} \ \ldots \ y_{\max,n}]^\top$ der maximalen Produktstückzahlen $y_{\max,b}$ (technische Kapazitäten), $\mathbf{y}_{\max} \in \mathbb{R}^n_{\geqslant 0}$, S. 176
$\mathbf{y}_{\mathrm{plan}}$	Vektor $[y_{\mathrm{plan},1} \ \ldots \ y_{\mathrm{plan},n}]^\top$ der geplanten Produktstückzahlen $y_{\mathrm{plan},b}$, je nach Planungsziel ggf. vorläufig festgelegt, $\mathbf{y}_{\mathrm{plan}} \in \mathbb{R}^n_{>0}$, S. 175
$y_{\mathrm{plan},b}$	geplante Stückzahl des Produkts $b \in \{1, \ldots, n\}$, $y_{\mathrm{plan},b} \in \mathbb{R}_{>0}$, S. 175

Eigene Symbole (griechisch)

α	Index zur Bezeichnung der Operanden von ggf. verketteten alternativen und/oder selektiven Verknüpfungen, $\alpha \in \{\text{'A'}, \text{'B'}, \ldots\}$, S. 44
ε	beliebig kleine Zahl größer als null, verwendet als Schranke, $\varepsilon \in \mathbb{R}_{>0}$, S. 186
η_{plan}	Plan-Nutzungsgrad, festgelegt als Zielvorgabe, $\eta_{\mathrm{plan}} \in (0,1]$, S. 34
$\Theta(f(n))$	Komplexitätsklasse, $\mathrm{O}(f(n)) \cap \Omega(f(n))$, S. 115
σ	allgemeiner, zuerst zu besuchender Knoten, $\sigma \in V$, S. 113
σ_b, τ_b	Quelle bzw. Senke, kennzeichnen jeweils als spezifische Knoten Beginn bzw. Ende eines Wertstroms, $\sigma_b, \tau_b \in V_{\mathrm{WS},b}$, S. 117
$\omega(f(n))$	Komplexitätsklasse, $\{g(n) : \forall c > 0 \ \exists n_0 \in \mathbb{N}_0 \ \forall n \geqslant n_0 \ \ g(n) \geqslant c\, f(n)\}$, S. 115
$\Omega(f(n))$	s. o., $\{g(n) : \exists c > 0 \ \exists n_0 \in \mathbb{N}_0 \ \forall n \geqslant n_0 \ \ g(n) \geqslant c\, f(n)\}$, S. 115

Index

Literaturverzeichnis

216 Quellen

Hinweis: Neben den unten stehenden, öffentlich einsehbaren Quellen sei auf alle internen Unterlagen verwiesen, die im Zuge der Entwicklung und Einführung der Software AURELIE bei der Bosch Rexroth AG in der Abteilung Manufacturing Coordination and Technology (DC/MFC) zu diesem Thema erstellt und archiviert wurden.

van der Aalst, Wil M. P. 2004a. »Business Process Management: A Personal View.« *Business Process Management Journal* 10 (2): 248–253. doi:10.1108/bpmj.2004.15710baa.001.

van der Aalst, Wil M. P. 2004b. »Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management.« In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, hrsg. von Jörg Desel, Wolfgang Reisig und Grzegorz Rozenberg. Bd. 3098 von *Lecture Notes in Computer Science*, 1–65. Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-27755-2_1.

van der Aalst, Wil M. P. 2013. »Business Process Management: A Comprehensive Survey.« *ISRN Software Engineering* 2013. 37 S. doi:10.1155/2013/507984.

van der Aalst, Wil M. P., Lachlan Aldred, Marlon Dumas und Arthur H. M. ter Hofstede. 2004. »Design and Implementation of the YAWL System.« In *Advanced Information Systems Engineering: 16th International Conference, CAiSE 2004, Riga, Latvia, June 7–11, 2004, Proceedings*, hrsg. von Anne Persson und Janis Stirna. Bd. 3084 von *Lecture Notes in Computer Science*, 142–159. Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-25975-6_12.

van der Aalst, Wil M. P., Arthur H. M. ter Hofstede und Mathias Weske. 2003. »Business Process Management: A Survey.« In *Business Process Management: International Conference, BPM 2003, Eindhoven, the Netherlands, June 26–27, 2003, Proceedings*, hrsg. von Wil M. P. van der Aalst, Arthur H. M. ter Hofstede und Mathias Weske. Bd. 2678 von *Lecture Notes in Computer Science*, 1–12. Berlin, Heidelberg: Springer. doi:10.1007/3-540-44895-0_1.

Aigner, Martin. 2006. *Diskrete Mathematik* (6., korr. Aufl.). Wiesbaden: Vieweg. doi:10.1007/978-3-8348-9039-9.

Allweyer, Thomas. 2010. *BPMN 2.0: Introduction to the Standard for Business Process Modeling* (2. Aufl.). Norderstedt: Books on Demand. ISBN 978-3-8391-4985-0.

Ammann, Paul und Jeff Offutt. 2016. *Introduction to Software Testing* (2., überarb. Aufl.). Cambridge, UK: Cambridge University Press. doi:10.1017/9781316771273.

Ammenwerth, Elske und Reinhold Haux, Hrsg. 2005. *IT-Projektmanagement in Krankenhaus und Gesundheitswesen: Einführendes Lehrbuch und Projektleitfaden für das taktische Management von Informationssystemen.* Stuttgart: Schattauer. ISBN 978-3-7945-2416-7.

Andradóttir, Sigrún. 1998. »A Review of Simulation Optimization Techniques.« In *Proceedings of the 1998 Winter Simulation Conference*, hrsg. von Deborah J. Medeiros, Edward F. Watson, John S. Carson und Mani S. Manivannan, 151–158. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/WSC.1998.744910.

Backlund, Alexander. 2000. »The Definition of System.« *Kybernetes* 29 (4): 444–451. doi:10.1108/03684920010322055.

Balla, Jochen und Frank Layer. 2007. *Production Planning with SAP APO-PP/DS.* SAP PRESS. Bonn, Boston, MA: Galileo Press. ISBN 978-1-59229-113-7.

Bangsow, Steffen. 2010. *Manufacturing Simulation with Plant Simulation and SimTalk: Usage and Programming with Examples and Solutions.* Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-05074-9.

Banks, Jerry, John S. Carson II, Barry L. Nelson und David M. Nicol. 2010. *Discrete-Event System Simulation* (5., überarb. Aufl.). Upper Saddle River, NJ, Singapur: Prentice Hall. ISBN 978-0-13-606212-7.

Bartsch, Helmut und Peter Bickenbach. 2002. *Supply Chain Management mit SAP APO: Supply-Chain-Modelle mit dem Advanced Planner & Optimizer 3.1* (2., aktual. und erw. Aufl.). SAP PRESS. Bonn: Galileo Press. ISBN 978-3-89842-111-9.

Bazaraa, Mokhtar S., John J. Jarvis und Hanif D. Sherali. 2010. *Linear Programming and Network Flows* (4. Aufl.). Hoboken, NJ: John Wiley & Sons. ISBN 978-0-470-46272-0.

Bhargava, Hemant K., Ramayya Krishnan und Peter Piela. 1994. »Formalizing the Semantics of ASCEND.« In *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, January 4–7, 1994*, 505–516. Wailea, HI: Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/HICSS.1994.323312.

Bisschop, Johannes und Marcel Roelofs. 2004. »The Modeling Language AIMMS.« Siehe Kallrath 2004, 71–104. doi:10.1007/978-1-4613-0215-5_6.

Blanchard, David. 2010. *Supply Chain Management: Best Practices* (2., illustr. Aufl.). Wiley Best Practices. Hoboken, NJ: John Wiley & Sons. ISBN 978-1-4784-2993-7.

Bourg, David M. 2009. *Excel Scientific and Engineering Cookbook: Adding Excel to Your Analysis Arsenal.* Sebastopol, CA: O'Reilly Media. ISBN 978-0-596-00879-6.

Boyd, Stephen und Lieven Vandenberghe. 2004. *Convex Optimization*. New York: Cambridge University Press. ISBN 978-0-521-83378-3.

Brocke, Henrik, Falk Uebernickel und Walter Brenner. 2009. »Success Factors in IT-Projects to Provide Customer Value Propositions.« In *Proceedings of the 20th Australasian Conference on Information Systems*. Monash University. Beitrag 24, 10 S. Abgerufen am 4. Januar 2017. http://aisel.aisnet.org/acis2009/24/.

vom Brocke, Jan und Michael Rosemann, Hrsg. 2010. *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*. International Handbooks on Information Systems. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-00416-2.

Brooke, Anthony, David A. Kendrick, Alexander Meeraus und Richard E. Rosenthal. 1988. *GAMS: A User's Guide*. Redwood City, CA: Scientific Press. ISBN 978-0-89426-118-3.

Bungartz, Hans-Joachim, Stefan Zimmer, Martin Buchholz und Dirk Pflüger. 2009. *Modellbildung und Simulation: Eine anwendungsorientierte Einführung*. eXamen.press. Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-79810-1.

Bussieck, Michael R. und Alex Meeraus. 2004. »General Algebraic Modeling System (GAMS).« Siehe Kallrath 2004, 137–157. doi:10.1007/978-1-4613-0215-5_8.

Carson, Yolanda und Anu Maria. 1997. »Simulation Optimization: Methods and Applications.« In *Proceedings of the 1997 Winter Simulation Conference*, hrsg. von Sigrún Andradóttir, Kevin J. Healy, David H. Withers und Barry L. Nelson, 118–126. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/WSC.1997.↵640387.

Chen, Injazz J. und Antony Paulraj. 2004. »Towards a Theory of Supply Chain Management: The Constructs and Measurements.« *Journal of Operations Management* 22 (2): 119–150. doi:10.1016/j.jom.2003.12.007.

Chopra, Sunil und Peter Meindl. 2013. *Supply Chain Management: Strategy, Planning, and Operation* (5., überarb. Aufl.). Boston, MA: Pearson. ISBN 978-0-13-274395-2.

Cohn, Mike. 2004. *User Stories Applied: For Agile Software Development*. Addison-Wesley Signature Series. Boston, MA: Addison-Wesley. ISBN 978-0-321-20568-1.

Colombani, Yves, Bob Daniel und Susanne Heipcke. 2004. »Mosel: A Modular Environment for Modeling and Solving Optimization Problems.« Siehe Kallrath 2004, 211–238. doi:10.1007/978-1-4613-0215-5_12.

Colombani, Yves und Susanne Heipcke. 2002. »Mosel: An Extensible Environment for Modeling and Programming Solutions.« In *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, hrsg. von Narendra Jussien und François Laburthe, 277–290. École des Mines de Nantes. Abgerufen am 4. Januar 2017. http://web.emn.↵fr/x-info/cpaior/Proceedings/CPAIOR-20.pdf.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest und Clifford Stein. 2001. *Introduction to Algorithms* (2., überarb. und illustr. Aufl.). Cambridge, MA: MIT Press. ISBN 978-0-262-03293-3.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, Clifford Stein und Paul Molitor. 2010. *Algorithmen – Eine Einführung* (3., überarb. und erw. Aufl.). München: Oldenbourg Wissenschaftsverlag. ISBN 978-3-486-59002-9.

Council of Supply Chain Management Professionals (CSCMP). 2015. »Bylaws of CSCMP.« Überarbeitet am 15. Juli, abgerufen am 4. Januar 2017. http://cscmp.org/.

Cunningham, Kevin und Linus Schrage. 2004. »The LINGO Algebraic Modeling Language.« Siehe Kallrath 2004, 159–171. doi:10.1007/978-1-4613-0215-5_9.

Davis, Martin. 1958. *Computability and Unsolvability*. New York: McGraw-Hill. Neuauflage 1982. New York: Dover Publications. ISBN 978-0-486-61471-7.

DeCarolis, Joseph, Kevin Hunter und Sarat Sreepathi. 2010. »The TEMOA Project: Tools for Energy Model Optimization and Analysis.« In *International Energy Workshop 2010, Stockholm, Sweden, June 21–23, 2010*. Abgerufen am 28. November 2019. http://www.↵temoaproject.org/publications/.

Deelmann, Thomas und Peter Loos. 2004. »Grundsätze ordnungsmäßiger Modellvisualisierung.« In *Modellierung 2004: Proceedings zur Tagung, 23–26. März 2004, Marburg*, hrsg. von Bernhard Rumpe und Wolfgang Hesse. Bd. P-45 von *GI-Edition: Lecture Notes in Informatics (LNI)*, 289–290. Bonn: Köllen. ISBN 978-3-88579-374-8.

Deming, W. Edwards. 1953. »Management's Responsibility for the Use of Statistical Techniques in Industry.« *Advanced Management* 18 (11): 8–12. ISSN 0749-7075.

Deo, Narsingh. 2004. *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall Series in Automatic Computation. New Delhi: Prentice-Hall of India, Englewood Cliffs, NJ: Prentice-Hall International. ISBN 978-81-203-0145-0.

Dickersbach, Jörg T. 2009. *Supply Chain Management with SAP APO™: Structures, Modelling Approaches and Implementation of SAP SCM™ 2008* (3. Aufl.). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-92942-0.

Diestel, Reinhard. 2005. *Graph Theory* (3., überarb. Aufl.), Bd. 173 von *Graduate Texts in Mathematics*. Berlin, Heidelberg: Springer. ISBN 978-3-540-26182-7. Elektronische Fassung abgerufen am 4. Januar 2017. http://emis.de/monographs/Diestel/en/.

Diestel, Reinhard. 2006. *Graphentheorie* (3., neu bearb. und erw. Aufl.). Springer-Lehrbuch Masterclass. Berlin, Heidelberg: Springer. ISBN 978-3-540-21391-8.

Dijkman, Remco M., Marlon Dumas und Chun Ouyang. 2008. »Semantics and Analysis of Business Process Models in BPMN.« *Information and Software Technology* 50 (12): 1281–1294. doi:10.1016/j.infsof.2008.02.006.

Dijkstra, Edsger W. 1976. *A Discipline of Programming*. Prentice-Hall Series in Automatic Computation. Englewood Cliffs, NJ: Prentice-Hall. ISBN 978-0-13-215871-8.

DIN 199-1:2002-03. *Technische Produktdokumentation – CAD-Modelle, Zeichnungen und Stücklisten – Teil 1: Begriffe*. Technische Norm. Deutsches Institut für Normung e. V. (DIN). Berlin: Beuth, 2002. Abgerufen am 4. Januar 2017. http://www.beuth.de/de/norm/din-199-1/46493205.

DIN 199-5:1981-10. *Begriffe im Zeichnungs- und Stücklistenwesen; Stücklisten-Verarbeitung, Stücklistenauflösung*. Technische Norm. Deutsches Institut für Normung e. V. (DIN). Berlin: Beuth, 1981. Abgerufen am 4. Januar 2017. http://www.beuth.de/de/norm/din-199-5/921205.

DIN 8580:2003-09. *Fertigungsverfahren – Begriffe, Einteilung*. Technische Norm. Deutsches Institut für Normung e. V. (DIN). Berlin: Beuth, 2003. Abgerufen am 4. Januar 2017. http://www.beuth.de/de/norm/din-8580/65031153.

Dinic, Jefim A. 1970. »Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation.« *Doklady Akademii Nauk SSSR* 11 (5): 1277–1280. Abgerufen am 28. November 2019. http://www.cs.bgu.ac.il/~dinitz/.

Dobmann, M., Michael Liepelt und Klaus Schittkowski. 1995. »Algorithm 746: PCOMP: A Fortran Code for Automatic Differentiation.« *ACM Transactions on Mathematical Software* 21 (3): 233–266. doi:10.1145/210089.210091.

Edmonds, Jack und Richard M. Karp. 1972. »Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems.« *Journal of the ACM* 19 (2): 248–264. doi:10.1145/321694.321699.

Ehrlenspiel, Klaus und Harald Meerkamm. 2013. *Integrierte Produktentwicklung: Denkabläufe, Methodeneinsatz, Zusammenarbeit* (5., überarb. und erw. Aufl.). München, Wien: Hanser. ISBN 978-3-446-43627-5.

Eley, Michael. 2012. *Simulation in der Logistik: Einführung in die Erstellung ereignisdiskreter Modelle unter Verwendung des Werkzeuges »Plant Simulation«*. Springer-Lehrbuch. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-27373-5.

Eleyat, Mujahed, Lasse Natvig und Jørn Amundsen. 2011. »Cache-Aware Matrix Multiplication on Multicore Systems for IPM-Based LP Solvers.« In *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS), Szczecin, Poland, September 18–21, 2011*, 431–438. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). ISBN 978-83-60810-35-4.

Elias, Peter, Amiel Feinstein und Claude E. Shannon. 1956. »A Note on the Maximum Flow through a Network.« *IRE Transactions on Information Theory* 2 (4): 117–119. doi:10.1109/TIT.1956.1056816.

Erlach, Klaus. 2010. *Wertstromdesign: Der Weg zur schlanken Fabrik* (2., bearb. und erw. Aufl.). VDI-Buch. Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-89867-2.

eVSM Group. 2015. »eVSM Application Examples: Manufacturing Mapping.« Abgerufen am 5. Januar 2018, zuletzt geändert am 15. Dezember 2015. http://evsm.com/evsm-application-examples.

Fishman, George S. 2001. *Discrete-Event Simulation: Modeling, Programming, and Analysis*. Springer Series in Operations Research and Financial Engineering. New York: Springer. doi:10.1007/978-1-4757-3552-9.

Ford, Jr., Lester R. und Delbert R. Fulkerson. 1956. »Maximal Flow through a Network.« *Canadian Journal of Mathematics* 8:399–404. doi:10.4153/CJM-1956-045-5.

Ford, Jr., Lester R. und Delbert R. Fulkerson. 1957. »A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem.« *Canadian Journal of Mathematics* 9:210–218. doi:10.4153/CJM-1957-024-0.

Fourer, Robert und David M. Gay. 2003. »Numerical Issues and Influences in the Design of Algebraic Modeling Languages for Optimization.« In *Proceedings of the 20th Biennial Conference on Numerical Analysis, Dundee, Scotland*, hrsg. von David Francis Griffiths und George Alistair Watson, 39–51. Veröffentlicht als Numerical Analysis Report NA/217, University of Dundee. Abgerufen am 3. Januar 2017. http://ampl.com/REFS/dundee03.pdf.

Fourer, Robert, David M. Gay und Brian W. Kernighan. 1990. »A Modeling Language for Mathematical Programming.« *Management Science* 36 (5): 519–554. doi:10.1287/mnsc.36.5.519.

Fourer, Robert, David M. Gay und Brian W. Kernighan. 2003. *AMPL: A Modeling Language for Mathematical Programming* (2., überarb. Aufl.). Belmont, CA: Duxbury Thomson. ISBN 978-0-534-38809-6. Abgerufen am 4. Januar 2017. http://ampl.com/resources/the-ampl-book/.

Fourer, Robert, David M. Gay und Brian W. Kernighan. 2004. »Design Principles and New Developments in the AMPL Modeling Language.« Siehe Kallrath 2004, 105–135. doi:10.1007/978-1-4613-0215-5_7.

Freund, Jakob und Bernd Rücker. 2012. *Praxishandbuch BPMN 2.0* (3., aktual. Aufl.). München, Wien: Hanser. ISBN 978-3-446-42987-1.

Fu, Michael C. 2002. »Optimization for Simulation: Theory vs. Practice.« *INFORMS Journal on Computing* 14 (3): 192–215. doi:10.1287/ijoc.14.3.192.113.

Fu, Michael C., Fred W. Glover und Jay April. 2005. »Simulation Optimization: A Review, New Developments, and Applications.« In *Proceedings of the 2005 Winter Simulation Conference*, hrsg. von Michael E. Kuhl, Natalie M. Steiger, F. Brad Armstrong und Jeffrey A.

Joines, 83–95. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/WSC.2005.1574242.

GAMS Development Corporation. 2016. *GAMS – A User's Guide*. Washington, DC. Zuletzt geändert am 21. Dezember 2016, abgerufen am 2. Januar 2017. http://gams.com/latest/docs/userguides/GAMSUsersGuide.pdf.

Gibson, Brian J., John T. Mentzer und Robert L. Cook. 2005. »Supply Chain Management: The Pursuit of a Consensus Definition.« *Journal of Business Logistics* 26 (2): 17–25. doi:10.1002/j.2158-1592.2005.tb00203.x.

Gingnell, Liv, Ulrik Franke, Robert Lagerström, Evelina Ericsson und Joakim Lilliesköld. 2014. »Quantifying Success Factors for IT Projects – An Expert-Based Bayesian Model.« *Information Systems Management* 31 (1): 21–36. doi:10.1080/10580530.2014.854033.

Goldberg, Andrew V. und Robert E. Tarjan. 1988. »A New Approach to the Maximum-Flow Problem.« *Journal of the ACM* 35 (4): 921–940. doi:10.1145/48014.61051.

Grotepaß, Lukas, Michael Sauter und Eberhard Abele. 2014. »Produktivitätsmanagement nachhaltig implementieren.« *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* 109 (9): 595–598. doi:10.3139/104.111201.

Guéret, Christelle, Christian Prins und Marc Sevaux. 2002. *Applications of Optimization with Xpress-MP*. Blisworth, Northants: Dash Optimization Ltd. ISBN 978-0-9543503-0-7.

Guerrero, Hector. 2010. *Excel Data Analysis: Modeling and Simulation*. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-10835-8.

Hahmann, Lukas. 2012. »Efficient Planning of Technical Capacities: Analysis and Definition of Design Patters for Graphical Value Streams.« Bachelorarbeit, Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt, Fakultät für Informatik und Wirtschaftsinformatik.

Halldorsson, Arni, Herbert Kotzab, Juliana H. Mikkola und Tage Skjøtt-Larsen. 2007. »Complementary Theories to Supply Chain Management.« *Supply Chain Management* 12 (4): 284–296. doi:10.1108/13598540710759808.

Hamacher, Horst W. und Kathrin Klamroth. 2006. *Lineare Optimierung und Netzwerkoptimierung: Zweisprachige Ausgabe Deutsch Englisch* (2., verb. Aufl.). Wiesbaden: Vieweg. doi:10.1007/978-3-8348-9031-3.

Hammer, Michael. 1990. »Reengineering Work: Don't Automate, Obliterate.« *Harvard Business Review* 68 (4): 104–112. ISSN 0017-8012.

Hammer, Michael. 2010. »What Is Business Process Management?« Siehe vom Brocke und Rosemann 2010, 3–16. doi:10.1007/978-3-642-00416-2_1.

Hammer, Michael und James Champy. 1993. *Reengineering the Corporation: A Manifesto for Business Revolution*. New York: HarperBusiness. ISBN 978-0-88730-640-2.

Hammer, Wilfried. 1997. *Wörterbuch der Arbeitswissenschaft: Begriffe und Definitionen*. REFA-Fachbuchreihe Betriebsorganisation. München: Hanser. ISBN 978-3-446-18995-9.

Hart, William E., Carl Laird, Jean-Paul Watson und David L. Woodruff. 2012. *Pyomo – Optimization Modeling in Python*, Bd. 67 von *Springer Optimization and Its Applications*. New York: Springer. doi:10.1007/978-1-4614-3226-5.

Hart, William E., Jean-Paul Watson und David L. Woodruff. 2011. »Pyomo: Modeling and Solving Mathematical Programs in Python.« *Mathematical Programming Computation* 3: 219–260. doi:10.1007/s12532-011-0026-8.

Hartmann, Edward H. 2013. *TPM: Effiziente Instandhaltung und Maschinenmanagement* (4. Aufl.). Management Competence. München: Vahlen. Übersetzung aus dem Englischen. doi:10.15358/9783800646340.

Haverly, Larry. 2001. »OMNI Model Management System.« *Annals of Operations Research* 104 (1–4): 127–140. doi:10.1023/A:1013143003516.

Haverly, Larry. 2004. »The OMNI Modeling System.« Siehe Kallrath 2004, 293–306. doi:10.↵1007/978-1-4613-0215-5_16.

Haverly Systems, Inc. 1976. *OMNI Linear Programming System: User and Operating Manual*. Denville, NJ.

Havey, Michael. 2005. *Essential Business Process Modeling*. Theory in Practice. Sebastopol, CA: O'Reilly Media. ISBN 978-0-596-00843-7.

Healy, Kevin und Lee W. Schruben. 1991. »Retrospective Simulation Response Optimization.« In *Proceedings of the 1991 Winter Simulation Conference*, hrsg. von Barry L. Nelson, W. David Kelton und Gordon M. Clark, 901–906. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/WSC.1991.185703.

Hedengren, John D. 2008. »A Nonlinear Model Library for Dynamics and Control.« Computer Aids for Chemical Engineering (CACHE) News, Sommer 2008. Abgerufen am 2. Januar 2017. http://cache.org/news-stand/summer-2008.

Hedengren, John D. und Ammon N. Eaton. 2015. »Overview of Estimation Methods for Industrial Dynamic Systems.« *Optimization and Engineering* 16:1–24. Special Issue on Optimization in the Oil and Gas Industry. doi:10.1007/s11081-015-9295-9.

Hedengren, John D., Reza Asgharzadeh Shishavan, Kody M. Powell und Thomas F. Edgar. 2014. »Nonlinear Modeling, Estimation and Predictive Control in APMonitor.« *Computers & Chemical Engineering* 70:133–148. doi:10.1016/j.compchemeng.2014.04.013.

Heipcke, Susanne. 2012. »Xpress-Mosel.« Siehe Kallrath 2012, 77–110. doi:10.1007/978-3-642-23592-4_5.

Van Hentenryck, Pascal, Irvin Lustig, Laurent Michel und Jean-François Puget. 1999. *The OPL Optimization Programming Language*. Cambridge, MA: MIT Press. ISBN 978-0-262-72030-4.

Van Hentenryck, Pascal, Laurent Michel, Frederic Paulin und Jean-François Puget. 2004. »The OPL Studio Modeling System.« Siehe Kallrath 2004, 307–350. doi:10.1007/978-1-4613-0215-5_17.

Hoare, Charles Antony Richard. 1969. »An Axiomatic Basis for Computer Programming.« *Communications of the ACM* 12 (10): 576–580. doi:10.1145/363235.363259.

Hochmuth, Christian A. 2011. »Innovative Software in Unternehmen: Strategie und Erfolgsfaktoren für Einführungsprojekte.« Masterthesis, Julius-Maximilians-Universität Würzburg, Fakultät für Wirtschaftswissenschaften.

Hochstättler, Winfried. 2010. *Algorithmische Mathematik*. Springer-Lehrbuch. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-05422-8.

ter Hofstede, Arthur H. M., Wil M. P. van der Aalst, Michael Adams und Nick Russell, Hrsg. 2010. *Modern Business Process Automation: YAWL and Its Support Environment*. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-03121-2.

Holmström, Kenneth. 1999. »The TOMLAB Optimization Environment in Matlab.« *Advanced Modeling and Optimization* 1 (1): 47–69. ISSN 1841-4311.

Holmström, Kenneth. 2008. »An Adaptive Radial Basis Algorithm (ARBF) for Expensive Black-Box Global Optimization.« *Journal of Global Optimization* 41 (3): 447–464. doi:10.1007/s10898-007-9256-8.

Holmström, Kenneth und Marcus M. Edvall. 2004. »The TOMLAB Optimization Environment.« Siehe Kallrath 2004, 369–376. doi:10.1007/978-1-4613-0215-5_19.

Holmström, Kenneth, Anders O. Göran und Marcus M. Edvall. 2010. *User's Guide for TOMLAB 7*. Västerås, Schweden: Tomlab Optimization AB. Zuletzt geändert am 5. Mai 2010, abgerufen am 2. Januar 2017. http://tomopt.com/docs/TOMLAB.pdf.

Hürlimann, Tony. 1999. *Mathematical Modeling and Optimization: An Essay for the Design of Computer-Based Modeling Tools*, Bd. 31 von *Applied Optimization*. Boston, MA, Dordrecht, London: Kluwer Academic Publishers. doi:10.1007/978-1-4757-5793-4.

Hürlimann, Tony. 2004. »The LPL Modeling Language.« Siehe Kallrath 2004, 173–183. doi:10.1007/978-1-4613-0215-5_10.

Hürlimann, Tony. 2016. *Reference Manual of LPL*. Universität Freiburg (Schweiz), Departement für Informatik. LPL 6.48, zuletzt geändert am 24. August 2016, abgerufen am 4. Januar 2017. http://virtual-optima.com/download/docs/manual.pdf.

Hürlimann, Tony und Jürg Kohlas. 1988. »LPL: A Structured Language for Linear Programming Modeling.« *OR Spektrum* 10 (1): 55–63. doi:10.1007/BF01720038.

IBM Corp. 2016a. *IBM ILOG CPLEX Optimization Studio: OPL Language Reference Manual, Version 12 Release 6*. Armonk, New York. Zuletzt geändert am 5. Juni 2016, abgerufen am 3. Januar 2017. http://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/pdf/opl_langref.pdf.

IBM Corp. 2016b. *IBM ILOG CPLEX Optimization Studio: OPL Language User's Manual, Version 12 Release 6*. Armonk, New York. Zuletzt geändert am 5. Juni 2016, abgerufen am 3. Januar 2017. http://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.3/ilog.odms.studio.help/pdf/opl_languser.pdf.

IEEE Std 830-1998. *IEEE Recommended Practice for Software Requirements Specifications*. Technische Norm. New York: Institute of Electrical and Electronics Engineers (IEEE), 1998. ISBN 978-0-7381-0332-7.

Illik, J. Anton. 2009. *Formale Methoden der Informatik: Von der Automatentheorie zu Algorithmen und Datenstrukturen*. Reihe Technik. Renningen: Expert-Verlag. ISBN 978-3-8169-2729-7.

ISO/IEC/IEEE 29148:2011(E). *Systems and Software Engineering – Life Cycle Processes – Requirements Engineering*. Technische Norm. Genf: International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), New York: Institute of Electrical and Electronics Engineers (IEEE), 2011. ISBN 978-0-7381-6591-2.

Jacoby, David. 2009. *Guide to Supply Chain Management: How Getting It Right Boosts Corporate Performance*, Bd. 26 von *The Economist Books*. New York: Bloomberg Press. ISBN 978-1-57660-345-1.

Kallrath, Josef, Hrsg. 2004. *Modeling Languages in Mathematical Optimization*, Bd. 88 von *Applied Optimization*. Boston, MA, Dordrecht, London: Kluwer Academic Publishers. doi:10.1007/978-1-4613-0215-5.

Kallrath, Josef, Hrsg. 2012. *Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems*, Bd. 104 von *Applied Optimization*. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-23592-4.

Kallrath, Josef und Thomas I. Maindl. 2006. *Real Optimization with SAP® APO*. Berlin, Heidelberg: Springer. doi:10.1007/3-540-34624-4.

Karlsen, Jan Terje, Jeanette Andersen, Live S. Birkely und Elise Ødegård. 2005. »What Characterizes Successful IT Projects.« *International Journal of Information Technology & Decision Making* 4 (4): 525–540. doi:10.1142/S0219622005001738.

Keller, Gerhard, Markus Nüttgens und August-Wilhelm Scheer. 1992. »Semantische Prozeßmodellierung auf der Grundlage ›Ereignisgesteuerter Prozeßketten (EPK)‹.« Heft 89,

Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Universität des Saarlandes, hrsg. von August-Wilhelm Scheer, Saarbrücken. ISSN 1438-5678.

Ko, Ryan K. L. 2009. »A Computer Scientist's Introductory Guide to Business Process Management (BPM).« *Crossroads* 15 (4): 11–18. doi:10.1145/1558897.1558901.

Ko, Ryan K. L., Stephen S. G. Lee und Eng Wah Lee. 2009. »Business Process Management (BPM) Standards: A Survey.« *Business Process Management Journal* 15 (5): 744–791. doi:10.1108/14637150910987937.

Koch, Arno. 2016. *OEE für das Produktionsteam: Das vollständige OEE-Benutzerhandbuch – oder wie Sie die verborgene Maschine entdecken* (3., korr. Aufl.). Nr. 5 in Operational Excellence. Ansbach: CETPM Publishing. ISBN 978-3-940775-04-7.

Köchel, Peter. 2009. »Simulation Optimisation: Approaches, Examples, and Experiences.« Forschungsbericht CSR-09-03, Technische Universität Chemnitz, Fakultät für Informatik. Veröffentlicht am 22. April 2009, abgerufen am 4. Januar 2017. http://nbn-resolving.de/urn:nbn:de:bsz:ch1-200900682.

Korte, Bernhard und Jens Vygen. 2012. *Kombinatorische Optimierung: Theorie und Algorithmen* (2. Aufl.). Springer-Lehrbuch Masterclass. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-25401-7.

Koskela, Mika und Jyrki Haajanen. 2007. »Business Process Modeling and Execution: Tools and Technologies Report for the SOAMeS Project.« VTT Research Notes 2407, VTT Technical Research Centre of Finland, Espoo. ISSN 1455-0865.

Kouvelis, Panos, Chester Chambers und Haiyan Wang. 2006. »Supply Chain Management Research and *Production and Operations Management*: Review, Trends, and Opportunities.« *Production and Operations Management* 15 (3): 449–469. doi:10.1111/j.1937-5956.2006.tb00257.x.

Krishnan, Ramayya, Peter Piela und Arthur Westerberg. 1993. »Reusing Mathematical Models in ASCEND.« In *Recent Developments in Decision Support Systems*, hrsg. von Clyde W. Holsapple und Andrew B. Whinston. Bd. 101 von *NATO ASI Series, F: Computer and Systems Sciences*, 275–293. Berlin, Heidelberg: Springer. doi:10.1007/978-3-662-02952-7_11.

Kristjansson, Bjarni und Denise Lee. 2004. »The MPL Modeling System.« Siehe Kallrath 2004, 239–266. doi:10.1007/978-1-4613-0215-5_13.

Krumke, Sven Oliver und Hartmut Noltemeier. 2012. *Graphentheoretische Konzepte und Algorithmen* (3. Aufl.). Leitfäden der Informatik. Wiesbaden: Springer Vieweg. doi:10.1007/978-3-8348-2264-2.

Kühn, Wolfgang. 2006. *Digitale Fabrik: Fabriksimulation für Produktionsplaner*. München, Wien: Hanser. ISBN 978-3-446-40619-3.

Kuip, C. A. C. 1993. »Algebraic Languages for Mathematical Programming.« *European Journal of Operational Research* 67 (1): 25–51. doi:10.1016/0377-2217(93)90320-M.

Larson, Paul D. und Arni Halldorsson. 2004. »Logistics versus Supply Chain Management: An International Survey.« *International Journal of Logistics Research and Applications* 7 (1): 17–31. doi:10.1080/13675560310001619240.

Law, Averill M. 2014. *Simulation Modeling and Analysis* (5., überarb. Aufl.). McGraw-Hill Series in Industrial Engineering and Management Science. Dubuque, IA: McGraw-Hill Education. ISBN 978-0-07-340132-4.

Legris, Paul und Pierre Collerette. 2006. »A Roadmap for IT Project Implementation: Integrating Stakeholders and Change Management Issues.« *Project Management Journal* 37 (5): 64–75. ISSN 1938-9507.

Liepelt, Michael und Klaus Schittkowski. 2000. »Remark on Algorithm 746: New Features of PCOMP, a Fortran Code for Automatic Differentiation.« *ACM Transactions on Mathematical Software* 26 (3): 352–362. doi:10.1145/358407.358412.

Liggesmeyer, Peter. 2009. *Software-Qualität: Testen, Analysieren und Verifizieren von Software* (2. Aufl.). Heidelberg: Springer Spektrum. doi:10.1007/978-3-8274-2203-3.

LINDO Systems Inc. 2016. *LINGO: The Modeling Language and Optimizer*. Chicago, IL. LINGO 16.0, zuletzt geändert am 23. Mai 2016, abgerufen am 4. Januar 2017. http://↵www.lindo.com/downloads/PDF/LINGO.pdf.

Makhorin, Andrew. 2016a. *GNU Linear Programming Kit: Reference Manual for GLPK Version 4.58*. Boston, MA: Free Software Foundation. Zuletzt geändert am 18. Februar 2016, abgerufen am 2. Januar 2017. http://ftp.gnu.org/gnu/glpk/.

Makhorin, Andrew. 2016b. *Modeling Language GNU MathProg: Language Reference for GLPK Version 4.58*. Boston, MA: Free Software Foundation. Zuletzt geändert am 16. Februar 2016, abgerufen am 2. Januar 2017. http://ftp.gnu.org/gnu/glpk/.

Mason, Andrew J. 2013. »SolverStudio: A New Tool for Better Optimisation and Simulation Modelling in Excel.« *INFORMS Transactions on Education* 14 (1): 45–52. doi:10.1287/↵ited.2013.0112.

Maximal Software, Inc. 2002. »Developing Large-Scale Optimization Models with the MPL Modeling System: A White Paper.« Zuletzt geändert am 23. März 2002, abgerufen am 3. Januar 2017. http://maximalsoftware.com/resources/mpl/mplwp.pdf.

Maximal Software, Inc. 2017. *MPL Modeling System: Release 5.0*. Arlington, VA. Elektronische Fassung abgerufen am 4. Januar 2017. http://maximalsoftware.com/mplman/.

May, Constantin und Peter Schimek. 2015. *Total Productive Management: Grundlagen und Einführung von TPM – oder wie Sie Operational Excellence erreichen* (3., korr. Aufl.). Nr. 1 in Operational Excellence. Ansbach: CETPM Publishing. ISBN 978-3-940775-05-4.

Meeraus, Alex. 1976. »Toward a General Algebraic Modelling System.« In *IX. International Symposium on Mathematical Programming, Budapest, Hungary, August 23–27, 1976*, 185. Budapest: Bolyai János Mathematical Society. Abgerufen am 28. November 2019. http://www.math.uwaterloo.ca/~bico/ismp/.

Mehlhorn, Kurt und Peter Sanders. 2008. *Algorithms and Data Structures: The Basic Toolbox*. Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-77978-0.

Melo, M. Teresa, Stefan Nickel und Francisco Saldanha-da-Gama. 2009. »Facility Location and Supply Chain Management – A Review.« *European Journal of Operational Research* 196 (2): 401–412. doi:10.1016/j.ejor.2008.05.007.

Mertens, Peter, Freimut Bodendorf, Wolfgang König, Arnold Picot, Matthias Schumann und Thomas Hess. 2012. *Grundzüge der Wirtschaftsinformatik* (11. Aufl.). Springer-Lehrbuch. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-30515-3.

Meyr, Herbert, Heidrun Rosič, Christian Seipl, Michael Wagner und Ulrich Wetterauer. 2008. »Architecture of Selected APS.« Siehe Stadtler und Kilger 2008, 349–366. doi:10.1007/↵ 978-3-540-74512-9_19.

Nakajima, Seiichi. 1988. *Introduction to TPM: Total Productive Maintenance*. Cambridge, MA: Productivity Press. Übersetzung aus dem Japanischen, ursprünglich herausgegeben vom Japan Institute for Plant Maintenance (Originaltitel: TPM Nyumon). ISBN 978-0-915299-23-2.

Nebel, Markus. 2012. *Entwurf und Analyse von Algorithmen*. Studienbücher Informatik. Wiesbaden: Springer Vieweg. doi:10.1007/978-3-8348-2339-7.

Neumaier, Arnold. 1997. »NOP – A Compact Input Format for Nonlinear Optimization Problems.« In *Developments in Global Optimization*, hrsg. von Immanuel M. Bomze, Tibor Csendes, Reiner Horst und Panos M. Pardalos. Bd. 18 von *Nonconvex Optimization and Its Applications*, 1–18. Boston, MA: Springer. doi:10.1007/978-1-4757-2600-8_1.

van Nuffel, Dieter, Hans Mulder und Steven van Kervel. 2009. »Enhancing the Formal Foundations of BPMN by Enterprise Ontology.« In *Advances in Enterprise Engineering III: CAiSE 2009, Amsterdam, the Netherlands, June 8–9, 2009, Proceedings*, hrsg. von Antonia Albani, Joseph Barjis und Jan L. G. Dietz. Bd. 34 von *Lecture Notes in Business Information Processing*, 115–129. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-01915-9_9.

Nüttgens, Markus und Frank J. Rump. 2002. »Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK).« In *Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen, Workshop der Gesellschaft für Informatik e. V. (GI), 9.–11. Oktober 2002 in Potsdam*, hrsg. von Jörg Desel und Mathias Weske. Bd. P-21 von *GI-Edition: Lecture Notes in Informatics (LNI)*, 64–77. Bonn: Köllen. ISBN 978-3-88579-350-2.

Object Management Group (OMG). 2010. »BPMN 2.0 by Example: Version 1.0 (Non-Normative).« Begleitdokument zur technischen Spezifikation. OMG Document Number:

dtc/10-06-02. Zuletzt geändert am 11. Juni 2010, abgerufen am 3. Januar 2018. http://www.omg.org/cgi-bin/doc?dtc/10-06-02.pdf.

Object Management Group (OMG). 2013. »Business Process Model and Notation (BPMN): Version 2.0.2.« Technische Spezifikation. OMG Document Number: formal/13-12-09. Zuletzt geändert am 9. Dezember 2013, abgerufen am 3. Januar 2018. http://www.omg.org/spec/BPMN/2.0.2/.

Object Management Group (OMG). 2015. »OMG Unified Modeling Language (OMG UML): Version 2.5.« Technische Spezifikation. OMG Document Number: formal/15-03-01. Zuletzt geändert am 15. Juli 2015, abgerufen am 3. Januar 2018. http://www.omg.org/spec/UML/2.5/.

Ólafsson, Sigurdur und Jumi Kim. 2002. »Simulation Optimization.« In *Proceedings of the 2002 Winter Simulation Conference*, hrsg. von Enver Yücesan, Chun-Hung Chen, Jane L. Snowdon und John M. Charnes, 79–84. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/WSC.2002.1172871.

Oliver, R. Keith und Michael D. Webber. 1992. »Supply-Chain Management: Logistics Catches up with Strategy.« In *Logistics: The Strategic Issues*, hrsg. von Martin Christopher, 63–75. London, New York: Chapman & Hall. Nachdruck, ursprünglich veröffentlicht in »Outlook« (Booz, Allen & Hamilton, 1982). ISBN 978-0-412-41550-0.

Organization for the Advancement of Structured Information Standards (OASIS). 2007. »Web Services Business Process Execution Language Version 2.0.« Technische Spezifikation. Zuletzt geändert am 11. April 2007, abgerufen am 4. Januar 2017. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

Orlin, James B. 1997. »A Polynomial Time Primal Network Simplex Algorithm for Minimum Cost Flows.« *Mathematical Programming* 78 (2): 109–129. doi:10.1007/BF02614365.

Ottmann, Thomas und Peter Widmayer. 2012. *Algorithmen und Datenstrukturen* (5. Aufl.). Heidelberg: Spektrum Akademischer Verlag. doi:10.1007/978-3-8274-2804-2.

Pflug, Georg C. 1996. *Optimization of Stochastic Models: The Interface between Simulation and Optimization*, Bd. 373 von *Kluwer International Series in Engineering and Computer Science*. Boston, MA, Dordrecht, London: Kluwer Academic Publishers. ISBN 978-0-7923-9780-9.

Piela, Peter C., Thomas G. Epperly, Karl M. Westerberg und Arthur W. Westerberg. 1991. »ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis: The Modeling Language.« *Computers & Chemical Engineering* 15 (1): 53–72. doi:10.1016/0098-1354(91)87006-U.

Piela, Peter C., Roy McKelvey und Arthur W. Westerberg. 1992. »An Introduction to the ASCEND Modeling System: Its Language and Interactive Environment.« *Journal of Management Information Systems* 9 (3): 91–121. doi:10.1080/07421222.1992.11517969.

Pohl, Klaus und Chris Rupp. 2011. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam – Foundation Level – IREB Compliant.* Sebastopol, CA: Rocky Nook. ISBN 978-1-4571-1192-1.

Powell, Stephen G., Kenneth R. Baker und Barry Lawson. 2008. »A Critical Review of the Literature on Spreadsheet Errors.« *Decision Support Systems* 46 (1): 128–138. doi:10.1016/j.dss.2008.06.001.

Powell, Stephen G., Kenneth R. Baker und Barry Lawson. 2009. »Impact of Errors in Operational Spreadsheets.« *Decision Support Systems* 47 (2): 126–132. doi:10.1016/j.dss.2009.02.002.

Pryor, Jennifer und John W. Chinneck. 2011. »Faster Integer-Feasibility in Mixed-Integer Linear Programs by Branching to Force Change.« *Computers & Operations Research* 38 (8): 1143–1152. doi:10.1016/j.cor.2010.10.025.

Ray, Santanu Saha. 2013. *Graph Theory with Algorithms and Its Applications: In Applied Science and Technology.* New Delhi: Springer India. doi:10.1007/978-81-322-0750-4.

Recker, Jan. 2010. »Opportunities and Constraints: The Current Struggle with BPMN.« *Business Process Management Journal* 16 (1): 181–201. doi:10.1108/14637151011018001.

Recker, Jan und Jan Mendling. 2006. »On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages.« In *18th International Conference on Advanced Information Systems Engineering, Proceedings of Workshops and Doctoral Consortium*, hrsg. von Thibaud Latour und Michael Petit, 521–532. Luxemburg: Namur University Press. Abgerufen am 4. Januar 2017. http://eprints.qut.edu.au/4637/.

Recker, Jan und Jan Mendling. 2007. »Lost in Business Process Model Translations: How a Structured Approach Helps to Identify Conceptual Mismatch.« In *Research Issues in Systems Analysis and Design, Databases and Software Development*, hrsg. von Keng Siau, 227–259. Hershey, PA: IGI Publishing. ISBN 978-1-59904-927-4.

REFA Bundesverband e. V. 2012. *REFA-Lexikon: Industrial Engineering und Arbeitsorganisation* (4. Aufl.). München: Hanser. ISBN 978-3-446-43408-0.

Roelofs, Marcel und Johannes Bisschop. 2016a. *AIMMS: The Language Reference. AIMMS 4.* Haarlem: AIMMS B. V. Zuletzt geändert am 7. Dezember 2016, abgerufen am 4. Januar 2017. http://aimms.com/english/developers/resources/manuals/.

Roelofs, Marcel und Johannes Bisschop. 2016b. *AIMMS: The User's Guide. AIMMS 4.* Haarlem: AIMMS B. V. Zuletzt geändert am 7. Dezember 2016, abgerufen am 4. Januar 2017. http://aimms.com/english/developers/resources/manuals/.

Rojnuckarin, Atipat und Christodoulos A. Floudas. 1994. »MINOPT: A Mixed Integer Nonlinear Optimizer.« Technischer Bericht, Princeton University, Department of Chemical Engineering, Computer Aided Systems Laboratory.

Rother, Mike und John Shook. 1999. *Learning to See: Value Stream Mapping to Create Value and Eliminate Muda*. Cambridge, MA: Lean Enterprise Institute. ISBN 978-0-9667843-0-5.

Sauter, Michael und Guido von Killisch-Horn. 2010. »Produktivitätsmanagement in einer variantenreichen Fertigung.« In *Methodisches Produktivitätsmanagement: Umsetzung und Perspektiven*, hrsg. von Sascha Stowasser. Nr. 204 in Angewandte Arbeitswissenschaft, Zeitschrift für die Unternehmenspraxis, 35–85. Köln: Wirtschaftsverlag Bachem. ISSN 0341-0900.

Schichl, Hermann, Stefan Dallwig und Arnold Neumaier. 2001. »The NOP-2 Modeling Language for Nonlinear Programming.« *Annals of Operations Research* 104 (1–4): 281–312. doi:10.1023/A:1013115708967.

Schichl, Hermann und Arnold Neumaier. 2004. »The NOP-2 Modeling Language.« Siehe Kallrath 2004, 279–291. doi:10.1007/978-1-4613-0215-5_15.

Schittkowski, Klaus. 2004. »PCOMP: A Modeling Language for Nonlinear Programs with Automatic Differentiation.« Siehe Kallrath 2004, 351–368. doi:10.1007/978-1-4613-0215-5_18.

Schlegel, Andreas. 2002. »Konzeption und Einsatzvorbereitung eines Werkzeuges für die Bestimmung der Prozessqualität mittels Kennzahlenüberwachung und wissensbasierter Simulation.« Dissertation, Technische Universität Chemnitz, Fakultät für Maschinenbau und Verfahrenstechnik. Abgerufen am 4. Januar 2017. http://nbn-resolving.de/urn:nbn:de:bsz:ch1-200201115.

Schlick, Gerhard H. 2001. *Projektmanagement – Gruppenprozesse – Teamarbeit: Wege, Hilfen und Mittel zu schnittstellen-minimierter Problemlösungskompetenz* (4. Aufl.). Renningen-Malmsheim: Expert-Verlag. ISBN 978-3-8169-1774-8.

Schrage, Linus E. 1986. *Linear, Integer, and Quadratic Programming with LINDO* (3. Aufl.). Palo Alto, CA: Scientific Press. ISBN 978-0-89426-090-2.

Schrage, Linus E. 2006. *Optimization Modeling with LINGO* (6. Aufl.). Chicago, IL: LINDO Systems. ISBN 978-1-893355-00-2.

Schweiger, Carl A. und Christodoulos A. Floudas. 1998a. *MINOPT: A Modeling Language and Algorithmic Framework for Linear, Mixed-Integer, Nonlinear, Dynamic, and Mixed-Integer Nonlinear Optimization*. Princeton University, Department of Chemical Engineering. Zuletzt geändert am 15. Oktober 1998, abgerufen am 4. Januar 2017. http://titan.princeton.edu/MINOPT/manual.pdf.

Schweiger, Carl A. und Christodoulos A. Floudas. 1998b. »Process Synthesis, Design, and Control: A Mixed-Integer Optimal Control Framework.« In *5th IFAC Symposium on Dynamics and Control of Process Systems (DYCOPS 5), Corfu, Greece, 8–10 June, 1998*. Bd. 31, Nr. 11 von *IFAC Proceedings Volumes*, 191–196. New York: Elsevier Science. doi:10.1016/S1474-6670(17)44927-5.

Schweiger, Carl A. und Christodoulos A. Floudas. 2004. »The MINOPT Modeling Language.« Siehe Kallrath 2004, 185–209. doi:10.1007/978-1-4613-0215-5_11.

Semini, Marco, Hakon Fauske und Jan O. Strandhagen. 2006. »Applications of Discrete-Event Simulation to Support Manufacturing Logistics Decision-Making: A Survey.« In *Proceedings of the 2006 Winter Simulation Conference*, hrsg. von Luiz Felipe Perrone, Frederick P. Wieland, Jason Liu, Barry G. Lawson, David M. Nicol und Richard M. Fujimoto, 1946–1953. Piscataway, NJ: Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/WSC.2006.322979.

Shewhart, Walter A. 1986. *Statistical Method from the Viewpoint of Quality Control*. Dover Books on Mathematics. New York: Dover Publications. ISBN 978-0-486-65232-0. Nachdruck, ursprünglich veröffentlicht in Washington, DC: Graduate School of the Department of Agriculture, 1939.

Silver, Bruce. 2011. *BPMN Method and Style, 2nd Edition, with BPMN Implementer's Guide: A Structured Approach for Business Process Modeling and Implementation Using BPMN 2*. Aptos, CA: Cody-Cassidy Press. ISBN 978-0-9823681-1-4.

Simchi-Levi, David, Philip Kaminsky und Edith Simchi-Levi. 2008. *Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies* (3., überarb. Aufl.). McGraw-Hill/Irwin Series in Operations and Decision Sciences. Boston, MA: McGraw-Hill Higher Education. ISBN 978-0-07-298239-8.

Singh, Bhim, Suresh K. Garg und Surrender K. Sharma. 2011. »Value Stream Mapping: Literature Review and Implications for Indian Industry.« *International Journal of Advanced Manufacturing Technology* 53 (5–8): 799–809. doi:10.1007/s00170-010-2860-7.

Slate, Laurel und Kurt Spielberg. 1978. »The Extended Control Language of MPSX/370 and Possible Applications.« *IBM Systems Journal* 17 (1): 64–81. doi:10.1147/sj.171.0064.

Spielberg, Kurt. 2004. »The Optimization Systems MPSX and OSL.« Siehe Kallrath 2004, 267–278. doi:10.1007/978-1-4613-0215-5_14.

Stadtler, Hartmut, Bernhard Fleischmann, Martin Grunow, Herbert Meyr und Christopher Sürie. 2012. *Advanced Planning in Supply Chains: Illustrating the Concepts Using an SAP® APO Case Study*. Management for Professionals. Berlin, Heidelberg: Springer. doi:10.↵1007/978-3-642-24215-1.

Stadtler, Hartmut und Christoph Kilger, Hrsg. 2008. *Supply Chain Management and Advanced Planning: Concepts, Models, Software, and Case Studies* (4. Aufl.). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-74512-9.

Standish Group International, Inc. 1995. »Chaos Report.« Boston, MA. Abgerufen am 4. Januar 2017. http://standishgroup.com/.

Standish Group International, Inc. 2011. »Chaos Manifesto: The Laws of Chaos and the Chaos 100 Best PM Practices.« Boston, MA.

Storey, John, Caroline Emberson, Janet Godsell und Alan Harrison. 2006. »Supply Chain Management: Theory, Practice and Future Challenges.« *International Journal of Operations & Production Management* 26 (7): 754–774. doi:10.1108/01443570610672220.

Suhl, Leena und Taïeb Mellouli. 2009. *Optimierungssysteme: Modelle, Verfahren, Software, Anwendungen* (2., überarb. Aufl.). Springer-Lehrbuch. Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-01580-9.

Turau, Volker. 2009. *Algorithmische Graphentheorie* (3., überarb. Aufl.). München, Wien: Oldenbourg Wissenschaftsverlag. ISBN 978-3-486-59057-9.

Tutte, William T. 2001. *Graph Theory* (2., illustr. Aufl.), Bd. 21 von *Cambridge Mathematical Library.* Cambridge, UK: Cambridge University Press. ISBN 978-0-521-79489-3.

Unger, Thomas und Stephan Dempe. 2010. *Lineare Optimierung: Modell, Lösung, Anwendung.* Studium. Wiesbaden: Springer Vieweg. doi:10.1007/978-3-8348-9659-9.

Vajna, Sandor, Christian Weber, Helmut Bley, Klaus Zeman und Peter Hehenberger. 2009. *CAx für Ingenieure: Eine praxisbezogene Einführung* (2., völlig neu bearb. Aufl.). Berlin, Heidelberg: Springer. doi:10.1007/978-3-540-36039-1.

Valente, Christian. 2011. »Design and Architecture of a Stochastic Programming Modelling System.« Dissertation, Brunel University London, School of Information Systems, Computing and Mathematics. Abgerufen am 2. Januar 2017. http://bura.brunel.ac.uk/handle/2438/6249.

Valente, Christian, Gautam Mitra, Mustapha Sadki und Robert Fourer. 2008. »Extending Algebraic Modelling Languages for Stochastic Programming.« *INFORMS Journal on Computing* 21 (1): 107–122. doi:10.1287/ijoc.1080.0282.

Valente, Christian, Gautam Mitra, Victor Zviarovich, Patrick Valente, Chandra Poojari, Frank Ellison und Nico Di Domenica. 2009. *SAMPL/SPInE User Manual.* Uxbridge: OptiRisk Systems. Zuletzt geändert am 22. Mai 2009, abgerufen am 2. Januar 2017. http://optirisk-systems.com/manuals/spineamplmanual.pdf.

Valente, Patrick, Gautam Mitra, Chandra A. Poojari und Triphonas Kyriakis. 2001. »Software Tools for Stochastic Programming: A Stochastic Programming Integrated Environment (SPInE).« Technical Report TR/10/2001, Centre for the Analysis of Risk and Optimisation Modelling Applications (CARISMA), Brunel University London. Abgerufen am 2. Januar 2017. http://bura.brunel.ac.uk/handle/2438/745.

VDI 3633:1996-11. *Simulation von Logistik-, Materialfluss- und Produktionssystemen – Begriffsdefinitionen.* VDI-Richtlinie (Entwurf). Verein Deutscher Ingenieure e. V. (VDI). Berlin: Beuth, 1996. Abgerufen am 4. Januar 2017. http://www.vdi.de/3633/.

Werners, Brigitte. 2013. *Grundlagen des Operations Research: Mit Aufgaben und Lösungen* (3., überarb. Aufl.). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-40102-2.

Wilson, D. George und Bernard D. Rudin. 1992. »Introduction to the IBM Optimization Subroutine Library.« *IBM Systems Journal* 31 (1): 4–10. doi:10.1147/sj.311.0004.

Winston, Wayne L. 2011. *Microsoft® Excel® 2010: Data Analysis and Business Modeling* (3. Aufl.). Redmond, WA: Microsoft Press. ISBN 978-0-7356-6029-8.

Witte, Frank. 2016. *Testmanagement und Softwaretest: Theoretische Grundlagen und praktische Umsetzung*. Wiesbaden: Springer Vieweg. doi:10.1007/978-3-658-09964-0.

Wohed, Petia, Wil M. P. van der Aalst, Marlon Dumas, Arthur H. M. ter Hofstede und Nick Russell. 2006. »On the Suitability of BPMN for Business Process Modelling.« In *Business Process Management: 4th International Conference, BPM 2006, Vienna, Austria, September 5–7, 2006, Proceedings*, hrsg. von Schahram Dustdar, Josẽ Luiz Fiadeiro und Amit P. Sheth. Bd. 4102 von *Lecture Notes in Computer Science*, 161–176. Berlin, Heidelberg: Springer. doi:10.1007/11841760_12.

Yeo, Kai Ting. 2002. »Critical Failure Factors in Information System Projects.« *International Journal of Project Management* 20 (3): 241–246. doi:10.1016/S0263-7863(01)↵00075-8.