Lehrstuhl für Effiziente Algorithmen
des Instituts für Informatik
der Technischen Universität München

Produktionsplanung in der Prozessindustrie: Modelle, effiziente Algorithmen und Umsetzung

Mark Scharbrodt

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Dr. h.c. W. Brauer
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. A. Steger
2.	Univ.-Prof. Dr. H. Weisser

Die Dissertation wurde am 10.07.2000 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 11.12.2000 angenommen.

Zusammenfassung

Dieser Arbeit liegt ein Industrieprojekt zugrunde. Ziel ist der Aufbau und die Implementation eines Produktionsplanungssytems (PPS–System) für die Brau- und Getränkeindustrie. Ausgehend von dieser Aufgabenstellung liegt der Fokus der Untersuchungen auf dem Teilbereich der effizienten Algorithmen innerhalb einer PPS–Lösung. Diese Arbeit gliedert sich in zwei Teile. Der erste Teil bildet eine grundlegende theoretische Analyse effizienter Algorithmen zur Produktionsplanung. Wir erhalten bestmögliche Approximationsaussagen für zwei von uns betrachtete Schedulingprobleme und geben zusätzliche Performanceanalysen von einfachen (online) Schedulingalgorithmen. Der zweite Teil der Arbeit behandelt die praktische Umsetzung der Planungsalgorithmen in die entwickelte PPS-Lösung und betrachtet entstehenden Probleme bei der Systemintegration.
Die im Theorieteil der Arbeit vorgestellten Planungsprobleme sind aus der konkreten Anwendung der Brau- und Getränkeindustrie abgeleitet. Es handelt sich dabei um Varianten klassischer Schedulingprobleme, die aber im Gegensatz zu ihren traditionellen Verwandten aus der Schedulingtheorie näher an der industriellen Praxis angeordnet sind. Konkret modellieren wir Fragestellungen, bei denen Maschinen und Anlagen — wie im industriellen Alltag üblich — zu gewissen Zeitpunkten nicht zur Verfügung stehen. Auf diese Weise können wir Fragestellungen, wie Systemausfälle, Wartungsmaßnahmen und Störungen betrachten. Zweitens spiegelt sich in einem solchen Ansatz eine Planung "in der Produktion" wider, bei der Maschinen und Anlagen bereits belegt sind und kontinuierlich mit neuen Aufträgen versorgt werden. Für die von uns betrachteten Probleme gelingt es uns unter Anwendung neuer Algorithmen, im Wesentlichen die gleichen Güteaussagen, wie für die klassischen Scheduling–Probleme zu treffen. Der zweite Teil der theoretischen Untersuchungen liegt im Bereich der Performance von Algorithmen. Es stellt sich die Frage, wann ein Algorithmus als "gut" bewertet werden kann. Hier verlassen wir die klassische Worst–Case–Analyse, die bestimmt, wie sich ein Algorithmus im schlimmsten Fall verhalten kann. Statt dessen versuchen wir durch eine Average–Case–Analyse das typische Verhalten eines Algorithmus zu erfassen. Zugrundegelegt wird ein stochastisches Modell, in dem die Prozesszeiten der Algorithmen durch Zufallsvariablen modelliert sind.
Im zweiten Teil der Arbeit wird eine Brücke vom theorischen Modell zur Praxisanwendung gespannt. Für die konkrete Anwendung der Brau- und Getränkeindustrie stellen wir ein Betriebsmodell auf, das typische Informations- und Datenströme und ihre Wechselwirkung mit den zu automatisierenden Planungsabläufen in abstrakter Weise darstellt. Dieses Modell ist im gewissen Sinn stellvertretend für die gesamte Prozessindustrie, da sich die typischen Elemente der Prozessindustrie wie Batchproduktion, Bestandsorientierung und mehrstufige Produktion in der Brauereianwendung wiederfinden.
Anhand unseres Modells zeigen wir die Grenzen der im ersten Teil dieser Arbeit gewonnenen theoretischen Erkenntnisse. Als Konsequenz werden zum Teil völlig

neue Ansätze notwendig. Es wird deutlich, dass die Planungsprobleme zwar theoretisch bestmöglich gelöst sind, aber dass die zugehörigen Algorithmen in der gegebenen Anwendung zum Teil nicht praktikabel sind.
Das entwickelte Systemkonzept ist speziell auf die Anforderungen der Brau- und Getränkeindustrie ausgerichtet. Die Systemarchitektur ist hierarchisch aufgebaut: In der Vertikalen verbindet sie die Ebene der Unternehmensführung (Leitebene) mit der Produktion (Feldebene), horizontal folgt sie im Wesentlichen der Einteilung eines Brauereibetriebes in mehrere Produktionsstufen. In entsprechender Weise sind die Schedulingalgorithmen aufgebaut, so dass sie mehrstufig planen können und nur auf den Informationen aufsetzten, die ihnen aufgrund ihrer hierarchischen Einordnung zur Verfügung stehen. Wir beschreiben die Systemintegration des Algorithmenkerns in das hierarchische Planungssytem. Die Herausforderung liegt dabei vor allem in der Verknüpfung verschiedener Systemkomponenten und in der Integration der Planungsalgorithmik in die informatorischen Abläufe des Unternehmens.

Danksagung

Ich möchte an dieser Stelle meinen beiden Betreuern Prof. Angelika Steger und Prof. Horst Weisser sowie Prof. Ernst W. Mayr nachdrücklich danken. Sie haben mir verdeutlicht, mit wieviel Eleganz und unglaublichen Ideen Fragestellungen aus der Theorie der Informatik gelöst werden können und wie komplex ein Problem aus den "Niederungen der Praxis" geraten kann und dass es dennoch zufriendenstellend gelöst werden kann.
Dank gilt den beiden Firmen *Gesellschaft für Informationstechnik Weihenstephan mbH* und *Werum Datenverarbeitungssysteme GmbH*, die diese Arbeit ins Leben gerufen, finanziert und durch Know–How–Transfer unterstützt haben.
Die Lektorentätigkeit für diese Arbeit übernahmen freundlicherweise Alexander Hall, Ulrich Voll und Martin Raab, aber auch meinen anderen lieben Kollegen am Lehrstuhl für Effiziente Algorithmen und am Lehrstuhl für Brauereianlagen und Lebensmittel–Verpackungstechnik gilt mein Dank für das besonders schöne und motivierende Arbeitsumfeld.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

In dieser Arbeit betrachten wir Modelle und Algorithmen zur automatisierten Produktionsplanung in der Prozessindustrie. Wir nähern uns unserem Gegenstand aus zwei Richtungen. Erstens betrachten wir die Planungsmodelle aus theoretischer Sicht und geben fundamentale Analysen spezieller Scheduling–Probleme. Zweitens beschreiben wir ein von uns zusammen mit Industriepartnern entwickeltes Produktionsplanungs- und Steuerungssystems (PPS–System) für eine spezielle Anwendung in der Brau- und Getränkeindustrie.
In beiden Teilen dieser Arbeit behandeln wir eine ähnliche Fragestellung unter unterschiedlichen Blickwinkeln. Gesucht werden effiziente Algorithmen, um konkrete Planungs- und Schedulingprobleme zu lösen. Aus der theoretischen Sicht interessiert das abstrakte Problem und der Algorithmus zur Problemlösung. Wie gut kann ein effizienter Algorithmus bestmöglich sein: Welche Güteaussagen sind möglich? Kann das Laufzeitverhalten abgeschätzt werden? — Die Lösung derartiger Fragestellungen erfordert den Einsatz elaborierter Techniken und gelingt in der Regel nur für vergleichsweise "reine" und klar strukturierte Probleme. Aus dem Blickwinkel der Anwendung interessieren dagegen Algorithmen, die eine Vielzahl von Restriktionen und Nebenbedingungen produktionstechnischer, organisatorischer und datentechnischer Art zu integrieren vermögen. Auch hier wird nach einer bestmöglichen Performance gesucht, allerdings immer in Hinblick auf ein übergreifendes Gesamtsystem mit unterschiedlichen Ausgangsdaten und unterschiedlichen Zielfunktionen. In einem solchen Umfeld müssen neue Algorithmen entwickelt werden, die statt eine konkret gegebene Kostenfunktion zu minimieren, eher einen reibungslosen und noch näher zu spezifizierenden "flüssigen" Produktionsverlauf garantieren sollen.
Im theoretischen Teil dieser Arbeit orientieren wir uns an Modellen aus der klassischen Theorie des Scheduling und konfrontieren diese mit typischen Restriktionen aus der Produktionsplanung. Die von uns betrachteten Probleme sind in der Praxis von hoher Bedeutung, aber sie sind in ihrer Komplexität und Approximierbarkeit noch nicht oder noch nicht hinreichend genau untersucht worden.

Als konkrete Probleme untersuchen wir das Scheduling unter Berücksichtigung von Maschinenausfallzeiten in verschiedenen Varianten, wobei es uns gelingt, die Komplexität und Approximierbarkeit dieser Probleme vollständig zu klären. Damit ist ein weiterer Schritt in die Richtung eines umfassenden Schedulingmodells der industriellen Produktionsplanung gemacht worden. Das von uns betrachtete Schedulingmodell beschränkt sich auf eine Facette der Produktionsplanung, aber es kann nun in Kombination mit verschiedenen anderen theoretischen Schedulingmodellen analysiert werden. Auf diese Weise kann bereits ein weites Spektrum von Nebenbedingungen und Zielkriterien abgedeckt werden, wie sie im industriellen Alltag auftreten, wenngleich ein vollständiges Modell des Produktionsscheduling immer noch in weiter Ferne liegt.
Als zweites wichtiges Problem in der Modellbildung betrachten wir nicht nur die Restriktionen und Zielfunktionen der Scheduling–Probleme, sondern wir führen darüber hinaus einen praxisnahen Begriff der Güte der Planung- und Scheduling–Algorithmen ein. Konkret versuchen wir, durch den Übergang vom deterministischen zu einem stochastischen Modell, ein realistischeres Bild der Eingabeinstanzen zu geben und daraufhin das Verhalten von einfachen Schedulingalgorithmen im Average–Case zu studieren. Bisher hat man die Güte eines solchen stochastischen Algorithmus durch den Erwartungswert der Zielfunktion beschrieben. Wir verlassen diesen traditionellen Ansatz, indem wir die Güte eines stochastischen Scheduling–Algorithmus als den Erwartungswert der Kostenfunktion relativ zur *jeweils* optimalen Lösung definieren. Unser Gütebegriff lehnt sich an die *kompetitive Analyse* von Online–Algorithmen an, indem er die erwartete relative Performance des Algorithmus in Relation zu einer optimalen Offline–Lösung betrachtet, bei der alle zukünftigen Ereignisse bereits bekannt sind. Durch den von uns betrachteten Gütebegriff für einen stochastischen Scheduling–Algorithmus eröffnet sich eine neue Sicht auf die bis dato angenommene Optimalität von bestimmten Scheduling–Strategien.
Der zweite Teil dieser Arbeit betrachtet die Anwendung der Scheduling–Modelle in die Praxis. Ausgehend von einer konkreten Anwendung in der Brau- und Getränkeindustrie wird ein System zur Produktionsplanung und zum Produktionsscheduling unter industriellen Bedingungen entwickelt und umgesetzt. Die Problemanalyse zeigte, dass verschiedene Aspekte der im ersten Teil dieser Arbeit betrachteten Schedulingprobleme als Teilprobleme der Produktionsplanung zu lösen sind. Von der algorithmischen Seite haben sich vor allem einfache und schnelle Schedulingalgorithmen als praktikabel erwiesen, da sie sich einerseits als äußerst flexibel im Einbezug der praxisrelevanten Nebenbedingungen zeigten, und da sie andererseits dem Wunsch der Anwender nach ausgesprochen kurzen Rechenzeiten genügen konnten. Eine weitere Herausforderung bei der Systementwicklung ist die Integration der Planungsprobleme in ein Gesamtsystem. Es zeigte sich, dass klassische Schedulingmodelle in einem integrierten Planungsumfeld nicht mehr gültig sind, und in Teilen vollständig neue Konzepte entwickelt werden müssen. Daher besteht ein wesentlicher Teil dieser Arbeit in der Entwicklung

des Systemkonzepts, der Datenmodelle und der Schnittstellen zwischen verschiedenen Planungsalgorithmen. Hier liegt eine Herausforderung an die interdisziplinäre Arbeit, um in Kombination des Wissens um die betriebswirtschaftlichen, die produktionstechnischen und informatorischen Abläufe mit dem Verständnis der algorithmischen Seite der Probleme zu einem Systemansatz zu gelangen, der eine bestmögliche Produktionsplanung mit Hilfe des Rechners gestattet.
Die Arbeit ist wie folgt aufgebaut: Im folgenden Abschnitt 1.1 werden zentrale Definitionen aus der Theorie des Scheduling eingeführt. Darüber hinaus wird der Begriff der *Approximation* formalisiert. In den Kapiteln 2 und 3 betrachten wir zwei Scheduling–Modelle mit Maschinenausfallzeiten. Im ersten Modell hat der Algorithmus noch einen gewissen Einfluss auf die Zeitpunkte, an denen Maschinen nicht zur Verfügung stehen und im zweiten Modell sind die Maschinenausfallzeiten fest vorgegeben. In Kapitel 4 werden die deterministischen Scheduling–Modelle zugunsten einer stochastischen Variante verlassen. Wir untersuchen einen an die *kompetitive Analyse* von Online–Algorithmen angelehnten Performance–Begriff am Beispiel des Problems "Minimierung der Summe der Fertigstellungszeiten" im stochastischen Modell. Im zweiten Teil dieser Arbeit wird das Produktionsplanungssystem für eine Praxisanwendung in der Brau- und Getränkeindustrie vorgestellt. Nach einer Einführung in die Problematik der Produktionsplanung (Kapitel 5) wird in Kapitel 6 das Planungsumfeld in einem Betrieb der Brau- und Getränkeindustrie beschrieben. In Kapitel 7 wird der Lösungsansatz für das Produktionsplanungssytem vorgestellt. Dazu wird das Systemkonzept zusammen mit Ansätzen zur Lösung verschiedener in der Planung für Brauereien auftretender Teilprobleme erörtert. Kapitel 8 und 9 beschreiben die Implementation des Planungssystems als C–Modul. Speziell werden die Datenstrukturen des Planungssystems und die verwendeten Scheduling–Algorithmen vorgestellt. In Kapitel 10 erfolgt schließlich noch einmal eine zusammenfassende Betrachtung dieser Arbeit.

1.1 Scheduling und Approximation

In diesem Abschnitt werden Definitionen und Konzepte aus der Theorie des Scheduling und der Approximationsalgorithmen eingeführt. In den weiteren Kapiteln werden wir dann Approximationsalgorithmen für spezielle Scheduling–Probleme betrachten. Die Ausführungen in diesem Kapitel konzentrieren sich auf die für das Verständnis dieser Arbeit notwendigen Begriffe und Konzepte. Gerade im Bereich des Scheduling existieren aber unzählige Modelle und Problemvarianten, zum Beispiel Modelle mit Maschinenanfahrzeiten, mit Präzedenzbedingungen oder verschiedenen Zielfunktionen. Die Literatur zum Thema Scheduling scheint beinahe unbegrenzt, aber als Startpunkt für weitere Untersuchungen eignen sich die Übersichtsartikel von Lawler, Lenstra, Rinnoy Kan und Shmoys [LLK91] und die Bücher von Blazewicz et. al. [BESW94], Lawler et. al. [LLK91] und Pinedo

[Pin95].

Approximationsalgorithmen wurden nicht nur für Schedulingprobleme entwickelt, sondern sind mittlerweile ein Standardkonzept für die Behandlung schwieriger Optimierungsprobleme. Auch hier beschränken wir uns auf die zentralen Definitionen und verweisen für alles weitere auf die Standardliteratur, etwa [Hoc97]. Zudem geben wir zwei Beispiele für Approximationsalgorithmen von Scheduling–Problemen, einmal für das Problem des Minimierens des Makespan auf parallelen Maschinen und einmal für die Minimierung der gewichteten Summe der Fertigstellungszeiten der Jobs. Beide Analysen werden für die weiteren Betrachtungen in dieser Arbeit eine wichtige Rolle spielen.

Wir betrachten zunächst den Bereich *Scheduling*. Grundsätzlich kann man Scheduling als das Problem verstehen, Ressourcen über einen Zeitraum verschiedenen auszuführenden Prozessen zuzuordnen. Scheduling–Probleme werden dabei durch drei Mengen charakterisiert: Eine Menge $\mathcal{J} = \{1, \ldots, n\}$ von *Aufträgen* oder *Jobs* j, die sich möglicherweise noch in einzelne Operationen (engl. *Tasks*) $O_{1j}, \ldots O_{kj}$ untergliedern, eine Menge $\mathcal{M} = \{m_1 \ldots m_n\}$ von Maschinen oder Prozessoren und eine Menge $\mathcal{R} = \{R_1, \ldots, R_s\}$ von s Typen zusätzlicher Ressourcen. *Scheduling* bezeichnet die Aufgabe, Maschinen aus M und Ressourcen aus R den Jobs J zuzuordnen, so dass alle Jobs unter Berücksichtigung verschiedener Nebenbedingungen bearbeitet werden können. In der klassischen Schedulingtheorie gibt es zwei grundlegende Restriktionen: Jeder Job kann nur von einer Maschine gleichzeitig bearbeitet werden und jede Maschine kann nur einen Job zur gleichen Zeit bearbeiten.

Schedulingprobleme können anhand verschiedener Charakteristika klassifiziert werden. Eine erste Unterscheidung liefern die Maschinen. Sie können entweder *parallel* sein, also beliebige Jobs ausführen dürfen oder *spezialisiert*, d. h. auf die Ausführung bestimmter Jobs eingeschränkt sein. Unter den parallelen Maschinen werden in Abhängigkeit von der Geschwindigkeit drei Arten von Prozessoren unterschieden. Falls alle Maschinen bei allen Jobs die gleiche Geschwindigkeit haben, so spricht man von *identischen* Maschinen. Falls die Maschinen dagegen unterschiedliche Geschwindigkeiten haben, diese jedoch unabhängig von den einzelnen Jobs sind, so spricht man von *uniformen* Maschinen. Im letzten Fall, wenn die Geschwindigkeit der Maschinen von dem gerade ausgeführten Job abhängt, spricht man von *unabhängigen* Maschinen. Ein *Schedule* oder *Belegungsplan* σ ist eine Zuordnung der einzelnen Jobs zu Startzeiten und zu den Maschinen und Ressourcen, so dass jeder Job genau einer Maschine zugeordnet ist und keine Maschine mehrere Jobs zur gleichen Zeit bearbeitet. Gegebenenfalls sind zusätzlichen Ressourcenrestriktionen zu genügen. Falls die Bearbeitung eines Jobs unterbrochen werden darf und sie dann zu einem späteren Zeitpunkt fortgeführt wird, so spricht man von einem Modell mit *Preemption*.

Die Jobsysteme sind des Weiteren durch die folgenden Charakteristika determiniert:

- *Prozesszeit:* Jedem Job j sind Prozesszeiten p_{ij} $(i = 1, \ldots, m)$ zugeordnet. Der Wert p_{ij} gibt an, wieviele Zeiteinheiten die Durchführung von Job j auf Maschine i benötigen würde. Im Falle von identischen Prozessoren gilt $p_{ij} = p_j$, $i = 1, \ldots, m$. Sind die Prozessoren uniform, gilt $p_{ij} = p_j/s_i$, $i = 1, \ldots m$, wobei p_j eine Standardprozesszeit bezeichne und s_i der Geschwindigkeitsfaktor von Maschine i.

- *Zeit der Jobfreigabe:* Die Zeit der Jobfreigabe (engl. *Release time*) r_j von Jobs j gibt den frühsten Zeitpunkt an, zu dem Job j bearbeitet werden darf. Sie kann auch als Ankunftszeit eines Jobs in einem rollierenden Planungssystem aufgefasst werden.

- *Fälligkeit:* Die Fälligkeit (engl. *Due Date*) d_j eines Jobs setzt ein Zeitlimit, zu dem der betreffende Job fertiggestellt sein sollte. Üblicherweise wird das Überschreiten dieses Limits mit einer zeitabhängigen Kostenfunktion bestraft.

- *Deadline:* Die Deadline $\tilde{d}_j$ ist eine nicht relaxierbare Zeitgrenze, zu der der Job beendet sein muss.

- *Gewicht:* Das Gewicht w_j beschreibt die relative Dringlichkeit eines Jobs j. Das Gewicht geht in eine zu minimierende Kostenfunktion ein, so dass Jobs mit hohem Gewicht in der Regel möglichst früh beendet werden sollten.

- *Ressourcenanforderung:* Jobs können zu ihrer Bearbeitung zusätzliche Ressourcen benötigen. Es gibt verschiedene Ressourcenmodelle. Man geht aber davon aus, dass Ressourcen in diskreten Einheiten gegeben und erneuerbar sind. Im Modell nehmen wir also an, dass s Typen von zusätzlichen Ressourcen in $r_1, \ldots, r_s$ Einheiten zur Verfügung stehen. Jeder Job j benötigt für seine Bearbeitung eine Maschine und eine Menge von zusätzlichen Ressourcen, die durch einen Ressourcenvektor $\mathbf{R}(j) = (R_1(j), \ldots, R_s(j))$ spezifiziert werden, wobei $R_l(j)$ $(0 \leq R_l(j) \leq m_l)$, $l = 1, 2, \ldots, s$ die zur Bearbeitung von Job j notwendige Anzahl der Einheiten von Ressource R_l beschreibt.

Auf der Menge der Jobs können *Präzedenzrelationen* definiert sein. Die Relation $i \prec j$ bedeutet, dass die Bearbeitung von Job i beendet sein muss, bevor Job j beginnen kann.
Einem gegebenen Schedule σ können die folgenden Werte zugeordnet werden:

- *Fertigstellungszeit C_j:* Zeit der Beendigung von Job j.

- *Makespan C_{max}:* Fertigstellungszeit des zuletzt fertiggestellen Jobs. Der Makespan gibt also den frühsten Zeitpunkt an, zu dem alle Jobs abgearbeitet sind, im gewissen Sinne also das "Ende" des Schedules.

- *Durchlaufzeit* $F_j = C_j - r_j$*:* Zeit, die von der Ankunft des Jobs j bis zu seiner Fertigstellung verstreicht.
- *Abweichung vom Fälligkeitstermin (engl.* Lateness*)* $L_j = C_j - d_j$*:* Differenz zwischen der Zeit der Fertigstellung von Job j und seiner Fälligkeit.
- *Verspätung (engl.* Tardiness*)* $D_j = \max\{C_j - d_j, 0\}$*:* Zeit, die Job j noch über seine Fälligkeit d_j hinaus bearbeitet wird.
- *Verfrühung* (engl. *Earliness*) $E_j = \max\{d_j - C_j, 0\}$: Zeit, die Job j vor seiner Fälligkeit beendet wird.

Obig genannte Charakteristika eine Schedules σ werden dazu benutzt, diesen Schedule zu evaluieren. Ziel ist, einen Schedule mit möglichst geringen Kosten zu erhalten. Die drei wichtigsten Kostenfunktionen in der Schedulingtheorie sind:

- *Makespan* C_{max}
- *Mittlere Durchlaufzeit (engl.* Mean flow time*)* $\bar{F}$*:*
  Die mittlere Durchlaufzeit berechnet sich als $\bar{F} := \frac{1}{n}\sum_{j=1}^{n} F_j$.
- *(Gewichtete) Summe der Fertigstellungszeiten* $\sum_{j=1}^{n} w_j C_j$:
  Die Fertigstellungszeiten der Jobs werden mit ihrem Gewichtsfaktor (bzw. Dringlichkeitsfaktor) multipliziert und aufsummiert.

In der Praxis werden einem Schedule üblicherweise eine Reihe von (teilweise gegenläufigen) Kostenzielen zugeordnet, wie zum Beispiel das Minimieren von Rüstzeiten, das im Gegensatz zu dem Ziel steht, geringe Lagerbestände zu erzeugen. Für die folgenden Ausführungen wird aber von einer einzigen zu minimierenden Zielfunktion ausgegangen.
Nach dieser Einführung in die Grundbegriffe des Scheduling, soll nun ein kurzer Abriß über den Begriff der *Approximationsalgorithmen* folgen. Das Konzept der Approximationsalgorithmen wurde von Garey, Graham und Ullmann [GGU72] und später auch Johnson [Joh74] formalisiert. Wir folgen der Terminologie aus [Hoc97]. Ein Approximationsalgorithmus δ wird im Folgenden stets als effizient vorausgesetzt, genauer als *polynomiell*. Wir setzten ferner voraus, dass der Approximationsalgorithmus eine zulässige Lösung zu einem $\mathcal{NP}$–schweren Optimierungsproblem mit einer Menge von Instanzen $\{I\}$ liefert.

Definition 1.1
Ein polynomieller Algorithmus A wird δ–Approximationsalgorithmus genannt, falls er für jede *Instanz I eine Lösung liefert, deren Wert höchstens das δ–fache der Optimallösung beträgt. Der kleinste dieser Werte δ wird* Approximationsgüte *oder* Performancegarantie $R_{\mathcal{A}}$ *von Algorithmus $\mathcal{A}$ genannt.*

Der Wert δ wird häufig auch *Approximationsfaktor* oder, aus dem englischen, *Worst–Case–Ratio* genannt. In manchen Fällen besteht der Fehler noch aus einem zusätzlichen additiven Term. Für diese Fälle wurde der Begriff des asymptotischen Approximationsfaktors eingeführt:

Definition 1.2
Die absolute Approximationsgüte eines Approximationsalgorithmus $\mathcal{A}$ ist

$$R_{\mathcal{A}} = \inf\{r \geq 1 \mid R_{\mathcal{A}}(I) \leq r \text{ für alle Instanzen } I\}$$

und die asymptotische Approximationsgüte $R_{\mathcal{A}}^{\infty}$ von $\mathcal{A}$ ist

$$R_{\mathcal{A}}^{\infty} = \inf\{r \geq 1 \mid \exists n \in \mathbb{Z}^{+}, \mathrm{R}_{\mathcal{A}}^{\infty}(\mathrm{I}) \leq \mathrm{r} \text{ für alle Instanzen I mit } \mathrm{Opt(I)} \geq \mathrm{n}\}.$$

Zwar ist die polynomielle Laufzeit bei den Approximationsalgorithmen entscheidend, aber innerhalb dieses Rahmens kann es wünschenswert sein, dass man eine längere Laufzeit erlaubt, um eine bessere Approximationsgüte zu erzielen. Eine solcher "Trade–off" zwischen Laufzeit und Approximationsgüte lässt sich mit der Notation eines Approximationsschemas formalisieren.

Definition 1.3
Eine Familie von Approximationsalgorithmen für ein Problem P, $\{\mathcal{A}_\epsilon\}$ nennt man ein polynomielles Approximationsschema *oder PTAS (engl.* polynomial time approximation scheme*), falls jeder Algorithmus $\{\mathcal{A}_\epsilon\}$ eine $(1+\epsilon)$–Approximation liefert.*

Definition 1.4
Eine Familie von Approximationsalgorithmen für ein Problem P, $\{\mathcal{A}_\epsilon\}$ nennt man voll–polynomielles Approximationsschema *oder FPTAS (engl.* fully polynomial time approximation scheme*), falls jeder Algorithmus $\{\mathcal{A}_\epsilon\}$ eine $(1+\epsilon)$–Approximation liefert und die Laufzeit polynomiell in der Eingabegröße und in $\frac{1}{\epsilon}$ ist.*

Ein solches *FPTAS* ist ein bestmögliches Approximationsergebnis für ein $\mathcal{NP}$–schweres Problem.
Eine weiter Klasse von Problemen bilden die *Online–Probleme*. Bei Online–Algorithmen ist die Eingabeinstanz nicht von Anfang an bekannt. Statt dessen müssen Ereignisse verarbeitet werden, ohne dass ein Wissen über zukünftige Ereignisse verwendet werden kann. Scheduling–Probleme können ebenfalls in das Online–Modell fallen. Hier würden die einzelnen Jobs im Laufe der Zeit eintreffen und sie sind einzuplanen, ohne dass bekannt ist, ob und welche weiteren Jobs noch eintreffen werden. *Online–Algorithmen* sind Approximationsalgorithmen eines besonderen Typs: Sie versuchen, die Performance eines optimalen Offline–Algorithmus zu approximieren, der ein vollständiges Wissen über die zukünftigen

Ereignisse hat. Die *kompetitive Analyse* vergleicht die Performance eines Online–Algorithmus $\mathcal{A}$ mit der Performance eines optimalen Offline–Algorithmus für jede Eingabeinstanz und betrachtet darüber die *Worst–Case–Ratio*.
Ein Nachteil im bisher betrachteten Konzept der Approximationsalgorithmen ist, dass der Algorithmus bezüglich des schlechtest möglichen relativen Fehlers zur optimalen Lösung über alle möglichen Probleminstanzen ausgewertet wird. Es genügt also, eine einzige ungünstige Instanz zu haben, für die der Wert δ angenommen wird, selbst wenn allen anderen Instanzen einen deutlich niedrigeren Wert aufweisen. Eine derartige Analyse eines Algorithmus hinsichtlich des schlechtesten möglichen Falles wird *Worst–Case–Analyse* genannt. Typischerweise ist aber das Verhältnis der Lösung unter einem Algorithmus $\mathcal{A}$ zur optimalen Lösung für die meisten Instanzen deutlich besser, als es die Approximationsgüte vermuten lässt. In Hinblick auf dieses Missverhältnis kommt dem Konzept der *Average–Case–Analyse* eine wichtige Bedeutung zu. Die Average–Case–Analyse versucht zu beschreiben, wie sich ein Algorithmus typischerweise verhält. Bei einer solchen Average–Case–Analyse ist die Verteilungsfunktion der Probleminstanzen bekannt. Mit einem derartigen Grundwissen kann dann versucht werden, eine Aussage über die Performance eines Algorithmus im Erwartungswert zu erzielen, oder sogar die Verteilung der Zielfunktionswerte zu erfassen. Für Scheduling–Probleme gibt es nur wenige Arbeiten in Richtung einer Average–Case–Analyse. In Kapitel 4 geben wir einen neuen Ansatz zur Beschreibung des Average–Case–Verhaltens von stochastischen Scheduling–Algorithmen.
Im Folgenden werden zwei Beispiele für Approximationsalgorithmen geben, auf die in dieser Arbeit mehrfach verwiesen wird. Das erste betrachtete Problem ist das Minimieren des Makespan bei parallelen Maschinen. Historisch ist dieses Problem von speziellem Interesse, da es als das erste Problem überhaupt gilt, für das eine Worst–Case–Analyse durchgeführt worden ist [Gra66].

Problem Makespanminimierung:
EINGABE: Endliche Menge J von n Jobs und Anzahl m von parallelen Maschinen.
GESUCHT: Zulässiger Schedule σ der Jobs J auf den m Maschinen mit minimalem Makespan C_{max}.

Die obig genannte Worst–Case–Analyse durch Graham bezieht sich auf einen einfachen Scheduling–Algorithmus, der unter dem Namen *List*–Scheduling bekannt ist: Sortiere die Jobs J in eine geeignete Reihenfolge. Jedesmal, wenn eine Maschine verfügbar wird, wähle den nächsten Job der Liste und plane ihn auf dieser Maschine ein.

Theorem 1.5
[Gra66] Algorithmus List *ist ein* $2 - (1/m)$*–Approximationsalgorithmus für das Scheduling–Problem des Minimierens des Makespan auf* m *parallelen Maschinen.*

Beweis. Die Startzeit und Fertigstellungszeit von Job j im unter *List* erstellten

Schedule seien mit s_j und C_j $(j = 1 \ldots n)$ bezeichnet. Der Makespan des unter *List* erstellten Schedule sei C_{List} und der des optimalen Schedules C_{Opt}. Der zuletzt fertiggestellte Job sei k und damit sind alle Maschinen mindestens bis zum Zeitpunkt s_k vollständig belegt. Dann gilt

$$C_{Opt} \geq p_k$$

und

$$C_{Opt} \geq \frac{1}{m} \sum_{j=1}^{n} p_j.$$

Die zweite untere Schranke stellt die beste mögliche Situation dar, bei der die letzten Jobs auf allen Maschinen zu genau der gleichen Zeit beendet werden. Der Schedule ist dann vollständig ausbalanciert. Für eine obere Schranke des unter *List* erstellten Makespan C_{List} wird der Schedule in zwei Teile zerlegt:

$$\begin{aligned} C_{List} = s_k + p_k &\leq \frac{1}{m} \sum_{j \neq k} p_j + p_k \\ &\leq \frac{1}{m} \sum_{j=1}^{n} p_j + (1 - \frac{1}{m}) p_k \\ &\leq C_{Opt} + (1 - \frac{1}{m}) C_{Opt} = (2 - 1/m) C_{Opt}. \end{aligned}$$

□

Dieses $(2 - 1/m)$–Resultat wurde in weiteren Arbeiten mehrfach bis hin zu einer 1.2–Approximation verbessert [Gra69], [CGJ78], [Fri84].
Bei dem zweiten für diese Arbeit zentralen Problem kann sogar ein optimaler Algorithmus angegeben werden.

Problem Gewichtete Summe der Fertigstellungszeiten:
EINGABE: Endliche Menge J von n Jobs mit Gewichten $w_1, \ldots, w_n$ und eine Maschine.
GESUCHT: Zulässiger Schedule σ der Jobs J auf der Maschinen mit minimaler gewichteter Summe $\sum_{i=1}^{n} w_i \cdot C_i$ der Fertigstellungszeiten.

Für diese Problem wurde in [Smi56] die Optimalität der *WSPT*–Regel (engl. *weighted shortest processing time first*) bewiesen, bei der Jobs nach nicht absteigenden Verhältnissen p_j/w_j sortiert werden.

Theorem 1.6
[Smi56] WSPT *ist ein optimaler Algorithmus für das Problem des Minimierens der gewichteten Summe der Fertigstellungszeiten auf einer Maschine.*

Der Beweis ist ein einfaches Austauschargument: Wenn zwei Jobs i und j nicht anhand der *WSPT*–Regel einsortiert sind, erniedrigt das Vertauschen der beiden Jobs die gewichtete Summe der Fertigstellungszeiten.

Teil I

Theoretische Analyse von Scheduling–Problemen

In den folgenden beiden Kapiteln untersuchen wir Scheduling–Probleme, bei denen die Maschinen nicht beliebig zur Verfügung stehen. Die Jobs dürfen also nur zu bestimmten Zeiten auf den Maschinen eingeplant werden. Scheduling mit einer solchen limitierten Maschinenverfügbarkeit ist ein Standardproblem aus der industriellen Praxis und wird im zweiten Teil dieser Arbeit, in dem das Planungssystem für die Brauindustrie vorgestellt wird, noch an verschiedenen Stellen auftreten.
Es gibt verschiedene Modelle für Scheduling–Probleme mit limitierter Maschinenverfügbarkeit, je nachdem, welches Szenario der Produktion dargestellt werden soll. In dieser Arbeit werden zwei für die spätere Anwendung relevante Modelle betrachtet. Im ersten Modell muss die Produktion auf den Maschinen periodisch unterbrochen werden, um Servicearbeiten, wie Reinigung oder Wartungsmaßnahmen auf den Maschinen zu gestatten. Das zugehörige Scheduling–Problem wird *Maintenance–Scheduling* [1] genannt. Im zweiten Modell werden die Restriktionen in der Form verschärft, dass Maschinen zu beliebigen, aber fest gegebenen Zeiten nicht belegt werden dürfen. Da die Zeiten, zu denen keine Belegung erfolgen darf, durch fest eingeplante Jobs repräsentiert werden können, spricht man von *Scheduling mit fixierten Jobs*.
Beim *Maintenance–Scheduling* können die Maschinenausfallzeiten durch den Benutzer beziehungsweise den Schedulingalgorithmus unter bestimmten Nebenbedingungen selbst bestimmt werden. Wir verwenden dieses Modell zum Beschreiben notwendiger periodischer Wartungs- und Maschinenreinigungsarbeiten: Damit die Produktion möglichst reibungslos verlaufen kann, ist es notwendig, dass die Maschinen in regelmäßigen Abständen gereinigt und überholt werden. Der genaue Zeitpunkt für diese Produktionsausfallzeiten ist nicht festgelegt, jedoch dürfen die einzelnen Reinigungen und Wartungsarbeiten nicht zu weit auseinander liegen. In unserem Modell ist eine feste Zeitspanne S gegeben, die den maximalen zulässigen Zeitraum zwischen zwei Wartungs- oder Reinigungsaufträgen beschreibt. Unter Berücksichtigung dieser durch S gegebenen zulässigen Produktionsintervalle dürfen die Jobs beliebig eingeplant werden. Das zugehörige Problem werden wir aus der theoretischen Sichtweise analysieren. Wir erhalten stärkere Approximationsaussagen, als für das später betrachtete Scheduling–Problem mit fixierten Jobs. Wir können für den Fall einer konstanten Maschinenzahl ein polynomielles Approximationsschema angeben und für den Fall, dass die Zahl der Maschinen Teil der Eingabe ist, ein asymptotisches polynomielles Approximationsschema (cf. Abschnitte 2.2 und 2.4). Dies sind wiederum bestmögliche Approximationsaussagen, da wir außerdem nachweisen können, dass kein polynomielles Approximationsschema im Falle einer variablen Maschinenzahl existiert und dass in beiden Fällen kein voll polynomielles Approximationsschema existieren kann, wenn $\mathcal{P} \neq \mathcal{NP}$ gilt. Die Approximationsaussagen sind auf der anderen Seite aber schwächer als die für das klassische Scheduling–Problem des Minimie-

[1] Engl. *Maintenance* = Wartung

rens des Makespan mit kontinuierlich zur Verfügung stehenden Maschinen. Bei letzterem konnte die Existenz eines *PTAS* auch im Falle einer variablen Maschinenzahl nachgewiesen werden [HS87]. Bei konstanter Maschinenzahl existiert sogar ein FPTAS [Sah76] und in der Online–Version erzielt jeder *List*–Algorithmus eine Approximationsgüte von $2 - 1/m$.
Da die Approximationsschemata aufgrund der hohen Rechenzeit für Praxisanwendungen nicht geeignet sind, betrachten wir in Abschnitt 2.5 zusätzlich einen einfachen auf dem *List*–Algorithmus basierenden Algorithmus für das Problem des *Maintenance–Scheduling*. Mit einer Approximationsgüte von 2 ist das Worst–Case–Verhalten zumindest bei einer hohen Maschinenzahl ähnlich der Performance des vergleichbaren *List*–Algorithmus beim Standardproblem des Minimierens des Makespan bei kontinuierlich zur Verfügung stehenden Maschinen.
Eine einfache Beschreibung des zweiten Modells besteht darin, dass eine Zahl k der n gegebenen Jobs bereits fix in den Schedule eingeplant sind und nun die restlichen $n - k$ Jobs eingeplant werden müssen, ohne dass sie sich mit den fixierten Jobs überlappen. Die k fixierten Jobs repräsentieren damit genau die Zeiträume, zu denen die Maschinen nicht zur Verfügung stehen. In der im zweiten Teil dieser Arbeit betrachteten Anwendung in der Brauindustrie sind dies zunächst einmal alle arbeitsfreien Zeiten, wie Wochenenden, Feiertage und arbeitsfreie Schichten. Zudem können für gewisse Zeiträume größere Wartungsmaßnahmen oder gar Stilllegungen von Maschinen festgelegt sein, so dass auch zu diesen Zeitpunkten kein weiterer Job eingeplant werden kann. In allen genannten Beispielen repräsentieren fixierte Jobs also Zeiten, zu denen keine Produktion auf den zugehörigen Maschinen stattfinden kann. Sie sind also eigentlich gar keine Jobs im klassischen Sinne. Fixierte Jobs eignen sich aber auch zur Darstellung von Aufträgen, die einen Produktionsvorgang repräsentieren, der zu einem bestimmten Zeitpunkt auf einer fest gegebenen Maschinen stattfinden soll, während die übrigen Jobs beliebig eingeteilt werden dürfen. In der täglichen Produktionsplanung gibt es für ein derartiges Modell zwei typische Situationen: Üblicherweise werden die Produktionspläne mit Hilfe des Planungssystems wöchentlich erstellt und täglich angepasst. Während eines Planungsvorgangs wird die Produktion nicht unterbrochen. Vielmehr findet sie unabhängig von den aktuellen Planungsvorgängen weiterhin statt, so dass die Maschinen insbesondere bereits teilweise belegt sind. Dies kann durch fixierte Jobs, die die laufende Produktion repräsentieren, berücksichtigt werden. Ein solcher Planungsvorgang, bei dem die Pläne während der laufenden Produktion angepasst und um neu auftretende Aufträge ergänzt werden, wird auch *rollierende Planung* genannt.
Die weitere Anwendung der fixierten Jobs ist die interaktive Planung. Häufig wünschen die Anwender Planungssysteme, die ihnen die Möglichkeit bieten, automatisch erzeugte Pläne manuell anzupassen. So können sie einzelne Aufträge verschieben, manuell erzeugen oder auf andere Maschinen umplanen. Es entstehen Teilpläne, die nicht mehr verändert werden sollen, während die automatische Planung die verbleibenden Aufträge anhand bestimmter Zielkriterien neu einfügen

soll. Konkret sind den Jobs in einem automatischen Planungssystem verschiedene Stati zugeordnet. Diese sind *eingeplant, fixiert, freigegeben, gestartet* und *beendet/abgebrochen*. Zu Beginn der Planung wird ein Scheduling–Algorithmus einen Plan erzeugen, der gewissen Zielfunktionskriterien genügt. Alle eingeplanten Aufträge erhalten den Status *eingeplant*. Nun können die Benutzer die Pläne modifizieren oder Teilpläne durch Fixierung festschreiben. Der darauf folgende Planungsalgorithmus entfernt alle nicht fixierten Aufträge und plant sie unter Berücksichtigung der fixierten Aufträge neu ein. Durch iterierte manuelle und algorithmengesteuerte Planung entsteht schließlich ein Belegungsplan, der in der Produktion umgesetzt werden soll. Durch die *Freigabe*, wird ein Auftrag an das Produktionssteuerungssystem gesendet und daraufhin kann der Auftrag entsprechend seines Termins gestartet werden. In dem Scheduling–Modell mit fixierten Jobs entsprechen alle Jobs mit den Stati *fixiert, freigegeben, gestartet* und *beendet* den k fixierten Jobs. Die Jobs mit dem Status *eingeplant* werden aus dem Plan entfernt und bilden zusammen mit den neu hinzukommenden Jobs die vom Scheduling–Algorithmus neu einzuplanenden $n-k$ Jobs. Wir betrachten das Scheduling–Modell mit fixierten Jobs abstrahiert von der Anwendung zunächst als rein theoretisches Problem, für das wir verschiedene Approximationsaussagen geben. Wir beweisen in Abschnitt 3.1, dass für das Scheduling–Problem mit fixierten Jobs kein voll polynomielles Approximationsschema und für den Fall einer variablen Maschinenzahl kein (asymptotisches) polynomielles Approximationsschema existieren kann, sofern $\mathcal{P} \neq \mathcal{NP}$. Für den Fall einer konstanten Maschinenzahl geben wir im Abschnitt 3.2 ein polynomielles Approximationsschema. Zusätzlich untersuchen wir in Abschnitt 3.6 einfache auf dem *List*–Algorithmus basierende Algorithmen für das Scheduling–Problem mit fixierten Jobs. Wir geben eine untere Schranke der Approximationsgüte dieser Verfahren von $2,5$ und zeigen, dass ihre Performance damit schlechter ist, als der vergleichbare Algorithmus bei *Maintenance–Scheduling*.

Die meisten Arbeiten über Scheduling mit limitierter Maschinenverfügbarkeit behandeln den Fall, dass Maschinen in Folge von Störungen ausfallen. Die ersten Analysen sind auf Schmidt [Sch84] zurückzuführen. Er stellte für das preemptive Modell einen $\mathcal{O}(n + m \log m)$ Algorithmus für die Offline–Variante des Problems vor, bei der alle Maschinenausfallzeiten im Voraus bekannt sind. Dieses Problem wurde in [Sch88] auf die Variante mit *Freigabezeiten* und *Fälligkeitsterminen* der Jobs erweitert. Kalyanasundaram und Pruhs [KP94, KP97] betrachteten die Online–Variante des Problems, bei der die Maschinenausfallzeiten nicht bekannt sind. Sie gaben optimale Algorithmen für einige Spezialfälle. Auch diese Problem wurde um *Freigabezeiten* und *Fälligkeitstermine* der Jobs erweitert. Ausserdem haben Albers and Schmidt bewiesen [AS99], dass kein Online–Algorithmus ein konstantes Verhältnis zwischen der vom Online–Algorithmus erzielten Lösung und der optimalen Offline–Lösung (engl. *kompetitive Ratio*) erzielen kann, wenn Maschinen beliebig ausfallen und wieder verfügbar werden können. Sie haben ausserdem das Problem untersucht, bei dem um eine gewisse Anzahl von Zeit-

einheiten in die Zukunft gesehen werden kann. Für diesen Fall haben sie einen optimalen Algorithmus vorgestellt.

KAPITEL 2

Maintenance–Scheduling

Beim Problem des Maintenance–Scheduling betrachten wir das Scheduling–Modell mit limitierter Maschinenverfügbarkeit bedingt durch periodische Wartungs- und Reinigungsarbeiten auf den Maschinen. Die Maschinenwartung oder -reinigung kann ebenso wie ein Job im Sinne des Scheduling aufgefasst werden. Wir sprechen daher von einem *Wartungsauftrag* oder der englischen Notation folgend von einem *Maintenance–Job*. In diesem Problem sind also eine Anzahl von Jobs gegeben und noch eine vom Schedule abhängige Anzahl zusätzlicher Jobs, die einzuplanen sind. Eine abstrakte Modellierung des Problems lautet wie folgt.

Problem Maintenance–Scheduling:
EINGABE: Endliche Mengen J von n Jobs mit positiven rationalen Prozesszeiten $p_1, \ldots, p_n$ und Zeitintervall S, so dass

- $\min\{p_i\} \geq 1$,
- $\max\{p_i\} \leq S$,

und eine Zahl m identischer Maschinen.
GESUCHT: Ein nicht preemptiver Schedule σ mit minimalem Makespan, unter den Nebenbedingungen, dass

- jeder Job genau einer Maschine zugeordnet ist,
- für jede Maschine ein Wartungsauftrag spätestens alle S Einheiten von Beginn des Schedules an oder nach S Produktionseinheiten nach Ablauf des letzten Wartungsauftrages auf der Maschine durchgeführt wird (Die Prozesszeit eines Wartungsauftrags ist dabei auf 1 normiert.),
- kein Job sich mit einem Wartungsauftrag überlappt,
- jede Maschine (sogar eine nicht belegte) durch einen Wartungsauftrag abgeschlossen wird.

Abbildung 2.1 zeigt einen Schedule, bei dem spätestens nach $S = 4$ Produktionseinheiten ein Wartungsauftrag eingeplant werden muss.

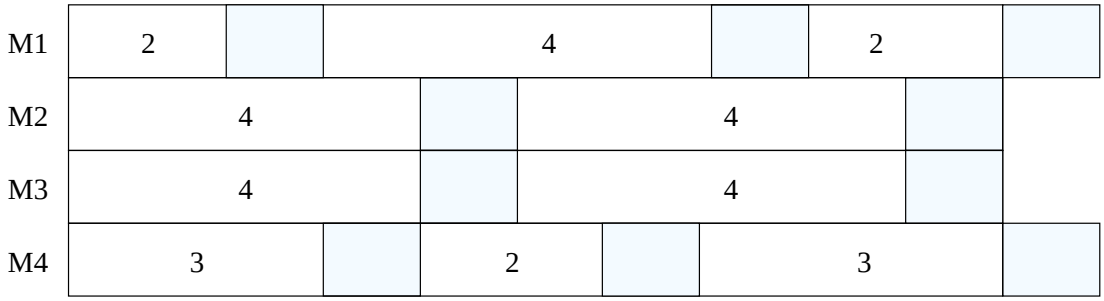


Abbildung 2.1: Schedule mit $S = 4$

Für die in diesem Abschnitt vorgestellten Approximationsaussagen ist die Nebenbedingung $\min\{p_i\} \geq 1$ eine wichtige Voraussetzung. Andernfalls, wenn also die Größe der Jobs im Verhältnis zur Dauer eines Wartungsauftrages beliebig klein werden kann, gelingen die gewünschten Approximationsaussagen nicht mehr.

Beispiel: *Sei I eine Probleminstanz, bei der die Restriktion* $\min\{p_i\} \geq 1$ *entfalle. I habe n Jobs, deren Prozesszeiten sich genau zu* 2 *addieren. Wir setzen ferner $S = 1$ und die Zahl der Maschinen auf* 2*. Dann gibt es eine Schedule mit Makespan* 2*, genau dann, wenn sich die Menge der Jobs in zwei Mengen partitionieren lässt, so dass sich die Prozesszeiten der Jobs in jeder Menge genau zu* 1 *addieren. Andernfalls liegen auf einer Maschine mindestens* 2 *Wartungsaufträge. Da die Summe der Jobprozesszeiten (ohne die Wartungsaufträge) auf dieser Maschine den Wert* 1 *überschreitet, beträgt der Makespan dann, wie in Abbildung 2.2 dargestellt, mindestens* 3*. Wenn also keine Optimallösung gefunden wird, beträgt die Approximationsgüte mindestens* $1,5$*, man kann also selbst bei konstanter Maschinenzahl nicht beliebig gut approximieren. Das Problem, eine Optimallösung zu finden, ist aber äquivalent dazu, das NP–schwere Problem* Partition *[GJ79] zu lösen.*

2.1 Negative Approximationsaussagen

Die beiden folgenden Theoreme zeigen, dass die optimale Lösung für das Problem Maintenance–Scheduling nicht beliebig approximiert werden kann, sofern $\mathcal{P} \neq \mathcal{NP}$.

Theorem 2.1

Bei einer konstanten Anzahl m von Maschinen gibt es kein voll polynomielles

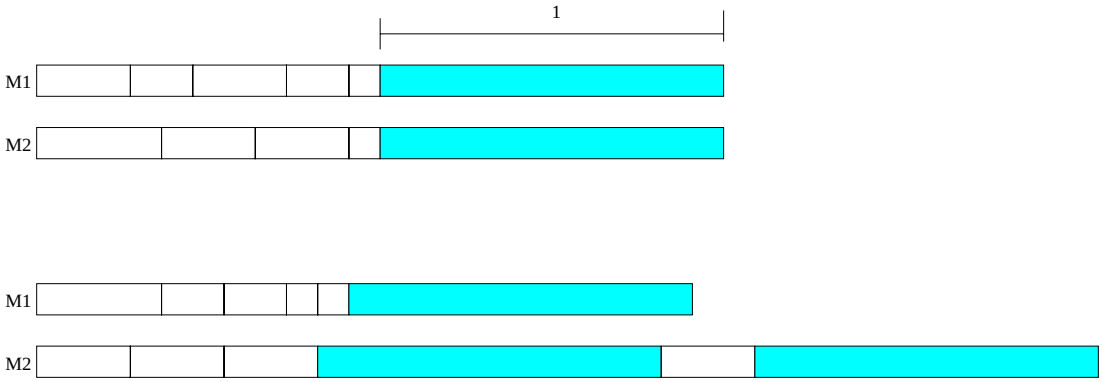


Abbildung 2.2: Approximation bei beliebigen Prozesszeiten

Approximationsschema (vorausgesetzt $P \neq NP$) für das Problem Maintenance–Scheduling.

Beweis. Es genügt, ein stark NP–vollständiges Problem auf Maintenance–Scheduling mit einer konstanten Anzahl von Maschinen zu reduzieren, da kein stark NP–vollständiges Problem durch ein voll–polynomielles Approximationsschema approximiert werden kann, sofern P $\neq$ NP (cf. [GJ79]). Wir reduzieren von *3–Partition*.

EINGABE: Endliche Menge A von $3n$ Elementen der Größen $s_1, \ldots, s_{3n}$ und eine Schranke $B \in \mathbb{Z}^+$, so dass $\frac{B}{4} < s_i < \frac{B}{2}$ für alle $1 \leq i \leq 3n$ und $\sum_{i=1}^{3n} s_i = n \cdot B$. FRAGE: Gibt es eine Partition von A in n verschiedenen Mengen $A_1, \ldots, A_n$, so dass für $1 \leq i \leq n$, $\sum_{s_j \in A_i} s_j = B$?

Das Problem *3–Partition* ist stark NP–schwer [GJ79].
Wir reduzieren auf das 1–Maschinen–Problem. Für jedes Element erzeugen wir einen Job, dessen Prozesszeit zu der Größe des Elements korrespondiert. Wir definieren die maximale Zeit zwischen zwei Wartungsaufträge durch $S := B$. Dann gibt es genau dann einen Schedule, dessen Makespan $n \cdot (B + 1)$ beträgt, wenn die gewünschte Partition in die n Mengen existiert. □

Theorem 2.2
Ist die Zahl der Maschinen Teil der Eingabe, dann existiert kein polynomieller Algorithmus, der das Problem des Maintenance–Scheduling mit einer Approximationsgüte echt kleiner als $\frac{6}{5}$ lösen kann, solange nicht $P = NP$ gilt.

Beweis. Wir reduzieren wiederum von *3–Partition*. Wie im vorherigen Beweis konstruieren wir für jedes Element einen Job, dessen Prozesszeit zur Größe des Elementes korrespondiert. Wir nehmen $n = m$ Maschinen und setzen $S := B$. Wir skalieren alle Elemente mit $\frac{1}{H}$, wobei $H = B/4$. Dann ist nach Voraussetzung

jedes Element größer oder gleich 1 und man erhält eine Instanz des Problems *Maintenance–Scheduling*. Dann ist klar, dass ein Schedule mit Makespan $\frac{B}{H}+1 = 5$ existiert, wenn die gewünschte Partition der Elemente in n Mengen existiert. Zusätzlich gilt, dass der Makespan, wenn eine derartige Partition nicht existiert, aufgrund des zusätzlich benötigten Wartungsauftrags größer als $\frac{B}{H}+2 = 6$ ist. □

Als Konsequenz des Theorems können wir im Falle einer variablen Maschinenzahl bestenfalls ein asymptotisches polynomielles Approximationsschema erhalten.

2.2 Polynomielles Approximationsschema bei konstanter Maschinenzahl

Zum Verständnis dieses und der folgenden Abschnitte hilft die in Kapitel 1 gegebene Analyse des *List*–Algorithmus für das Stadardproblem der Makespanminimierung (cf. Theorem 1.5). Die Analyse baut auf der Tatsache auf, dass ein Schedule genau dann am kürzesten ist, wenn die Jobs auf allen Maschine zur gleichen Zeit terminieren. Der Schedule ist dagegen am längsten, wenn allen Maschinen die gleiche Prozesszeit zugeordnet wird und dann bei einer Maschine ein einzelner Job übersteht. Je größer dieser überstehende Job, desto schlechter ist der Schedule im Verhältnis zum optimalen Schedule. Die gleichen Überlegungen gelten natürlich auch, wenn im Schedule zusätzlich zu den Jobs der Eingabeinstanz auch Wartungsaufträge eingeplant werden müssen. Daher ist das Ziel der Algorithmen für das Maintenance–Scheduling Problem, einen Schedule zu erzeugen, bei dem nur wenige Wartungsaufträge erzeugt werden und der gleichzeitig möglichst ausbalanciert ist.

Wir setzen voraus, dass die Zahl m der Maschinen, wie in praktischen Anwendungen üblich, konstant ist. Ausgangspunkt für das polynomielle Approximationsschema ist eine binäre Suche nach dem Optimum. Sei C ein möglicher Kandidat für den optimalen Makespan C_{Opt}. Wenn es gelingt, die Jobs mit einem Makespan von maximal $\epsilon \cdot C$ einzuordnen, ist die gewünschte $(1+\epsilon)$–Approximation des Makespan erreicht. Die binäre Suche kann mit einer unteren Schranke $\max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j\}$ initialisiert werden. Wenn die Restriktion der Wartungsaufträge vorübergehend außer Acht gelassen wird, können alle Jobs nach den Argumenten aus Theorem 1.5 bis zu dem Zeitpunkt $2 \cdot \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j\}$ anhand einer einfachen *List*–Regel eingeplant werden. Daraus folgt die obere Schranke $4 \cdot \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j\}$, indem jeweils nach Beendigung eines Jobs ein Wartungsauftrag eingeplant wird.

Wir unterscheiden zwei Fälle. Im ersten Fall gehen wir davon aus, dass der Wert S in Abhängigkeit von ϵ in einer noch genauer spezifizierten Weise "klein" ist. In diesem Fall wird das Problem als *Bin–Packing* interpretiert. Bei *Bin–Packing* sind n Objekte gegeben, wobei das i–te Objekt eine Größe s_i habe. Die Aufgabe ist, die Objekte in möglichst wenige Kisten zu packen, wobei jede Kiste eine

Größe B habe. Beim hier betrachteten Maintenance–Scheduling sind die Objekte die Jobs und die Größen der Objekte die Prozesszeiten der Jobs. Die Größe einer Kiste ist das maximal zulässige Zeitintervall S zwischen zwei Wartungsaufträgen. Im ersten Schritt werden die Jobs in eine möglichst minimale Anzahl von Kisten gepackt. Das Minimieren der Anzahl der Kisten entspricht dem Minimieren der Anzahl der Wartungsaufträge. Im zweiten Schritt werden die Kisten (genauer die Jobs der Kisten) anhand eines *List*–Algorithmus auf die Maschinen gesetzt, wobei nach jeder Kiste ein Wartungsauftrag eingeplant wird. Der Grund dafür, dass mit dem derartigen Verfahren eine hinreichend gute Approximation gelingt, liegt in dem kleinen Wert von S. Auf keiner Maschinen kann ein Jobs deutlich über die anderen Jobs hinausstehen, da alle Jobs ebenso wie S klein sind.
Im zweiten Fall mit möglichen großen Werten für S ist ein anderer Weg zu gehen, denn nun kann die Eingabeinstanz auch große Jobs enthalten. Dies birgt die Gefahr in sich, dass zum Ende des Schedules ein großer Job übersteht und der Schedule damit nicht hinreichend ausbalanciert ist. Wir gehen in zwei Schritten vor: Im ersten Schritt werden ausschließlich große Jobs eingeplant. Dies geschieht durch eine Enumeration aller möglichen Schedules σ' auf den großen Jobs. Hierbei ist entscheidend, dass die Zahl der Maschinen konstant ist und daher auch nur eine konstante Zahl großer Jobs vor dem Zeitpunkt C eingeplant werden kann. Im zweiten Schritt werden dann die verbleibenden kleinen Jobs den Maschinen zugeordnet. Aufgrund ihrer kurzen Prozesszeiten, können sie nicht mehr zu einem unausbalancierten Schedule führen, selbst wenn sie durch einen einfachen *List*–Algorithmus eingeplant werden.
Im Folgenden sei die Familie der Algorithmen, aus denen sich das polynomielle Approximationsschema zusammensetzt, mit $\{A_\epsilon\}$ bezeichnet.
Die Funktion *SMALL_JOBS(σ', Klein)* verwendet einen einfachen Einsetzungsalgorithmus um die Jobs $p \in$ *Klein* einzuplanen. Wenn Jobs nicht eingeplant werden können, wird *Failure* zurückgegeben. Der Algorithmus arbeitet in zwei Phasen. In der ersten Phase werden die Zeitintervalle vor dem ersten Wartungsauftrag jeder Maschine und zwischen zwei Wartungsaufträgen jeder Maschine im partiellen Schedule σ' als Kisten aufgefasst, die jeweils partiell mit Jobs gefüllt sind. Wir definieren S als die maximale Größe einer Kiste. Diese Kisten sind den Maschinen bereits zugeordnet und sie werden nun sukzessive. Ein Job $j \in$ *Klein* wird in eine Kiste eingesetzt, falls die Kiste momentan mit Jobs gefüllt ist, deren gesamte Prozesszeit weniger als S Einheiten beträgt. Wir erlauben also insgesamt, dass Kisten mit einem Job überfüllt werden können. Zweites Kriterium ist, dass die Summe der aktuellen Prozesszeiten für jede Maschine den Makespan von σ' nicht über eine noch zu spezifizierende Grenze hinaus überschreitet. Falls ein Job nicht in eine Kiste auf der aktuellen Maschine gepackt werden kann, aber die gesamte Prozesszeit auf der Maschine noch genügend klein ist, wird eine neue Kiste für den Job auf der Maschine eröffnet und ein Wartungsauftrag hinten angefügt.
In der zweiten Phase entfernt der Algorithmus alle Jobs, die eine Kiste überfüllen. Der verbleibende Schedule ist wieder zulässig. Als letztes setzt ein *List*–

Algorithmus 2.2.1 Algorithmus A_ϵ für das Problem *Maintenance–Scheduling* auf m Maschinen

Input: Instanz I bestehend aus n Jobs mit positiven rationalen Prozesszeiten $p_1, \ldots, p_n$, $\min(p_i) \geq 1$, und einer ganzen Zahl S, so dass $\max(p_i) \leq S$

Output: Zulässiger Schedule für I mit Makespan $\leq (1 + 8 \cdot \epsilon) C_{Opt}$

$ub := 4 \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j\}$;

$lb := \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j\}$;

while $(ub - lb \geq 1)$

$C := lb + \frac{ub-lb}{2}$;

Fall $S < \epsilon \cdot C$:

Benutze ein *FPTAS* [dlVL81] für *Bin–Packing*. Alle Kisten haben die Größe S und die Objekte seien die Jobs. Finde dadurch eine Zuordnung der Objekten zu den Kisten, die höchstens $(1 + \epsilon)$ mal die minimal mögliche Anzahl an Kisten benötigt. Wähle für jedes derartig erzeugte Kiste diejenige Maschine aus, die momentan am kürzesten belegt ist. Die in der Kiste enthaltenen Jobs werden dieser Maschine zugeordnet und ein Wartungsauftrag wird zu dem Zeitpunkt eingefügt, zu dem der letzte Job der Kiste abgearbeitet ist. Dabei erlauben wir, dass der Zeitpunkt C überschritten wird.

Fall $S \geq \epsilon \cdot C$:

Definiere zwei Arten von Jobs: *Groß* ist die Menge der Jobs j mit $p_j \geq \epsilon^2 \cdot C$. *Klein* ist die Menge der Jobs j mit $p_j < \epsilon^2 \cdot C$. Falls die Anzahl der Jobs in *Groß* den Wert $\frac{m}{\epsilon^2}$ übersteigt, kann kein zulässiger Schedule existieren (gebe *Failure* zurück). Ansonsten enumeriere alle zulässigen Schedules, wobei nur die großen Jobs verwendet werden. Falls kein solcher Schedule existiert, wähle unter allen zulässigen Schedules mit einen Makespan von höchstens C den Schedule σ', der die kleinste Anzahl an Wartungsaufträgen benötigt. Existiert kein Schedule mit Makespan höchstens C, gebe *Failure* zurück.

Rufe die Funktion *SMALL_JOBS(σ', Klein)* zum Packen der kleinen Jobs $j \in Klein$, so dass ein Schedule σ aller Jobs erzeugt wird.

Falls *Failure* = *False* setze $ub := C$, sonst setze $lb := C$.

end

Wenn kein Schedule gefunden wurde, setze $C := ub$ und finde eine Schedule für den Wert C. Gebe den endgültigen Schedule aus.

Algorithmus alle entfernten Jobs wieder in den Schedule. Die folgende Funktion beschreibt das Verfahren zum Packen der kleinen Jobs. Hierbei bezeichne k_j die Zahl der Kisten auf Maschine j und $load(s)$, für eine Kiste s, die gesamte Prozesszeit aller der Kiste zugeordneten Jobs.

Funktion 2.2.1 SMALL_JOBS(σ', Klein)

Input: Partieller Schedule σ' und Menge $Klein$ von Jobs mit durch $\epsilon^2 \cdot C$ beschränkter Prozesszeit.

Output: Schedule σ aller Jobs oder *Failure*.

Identifiziere für jede Maschine l die Kisten s_{il}, $1 \leq i \leq k_l$, die durch die Zeitintervalle vor dem ersten Wartungsauftrag bzw. zwischen zwei hintereinander liegenden Wartungsaufträgen auf Maschine l gegeben sind;
$l = 1$;
if $(\frac{\sum_{j \in J} p_j}{S} + \sum_{j \in J} p_j) > m \cdot (1 + \epsilon \cdot C)$ gebe *Failure* zurück, **else**
  while $l \leq m$ **und** $Klein \neq \emptyset$
    $i = 1$; $found = FALSE$; wähle nächsten Job $l \in Klein$;
    while not *found* **and** $i \leq k_l$
      if $load(s_{il}) < S$ und $\sum_{r=1}^{k_l}(load(s_{rl}) + 1) + p_j \leq (1 + \epsilon) \cdot C$, füge j in die Kiste s_{rl} ein; entferne j aus $Klein$;
      $found = TRUE$;
      else $i++$
    if not *gefunden*
      if $\sum_{r=1}^{k_l}(load(s_{rl}) + 1) + p_j + 1 \leq (1 + \epsilon) \cdot C$, öffne eine neue Kiste $s_{(k_l+1)l}$ und setze j in $s_{(k_l+1)l}$ ein; $k_l := k_l + 1$; entferne j aus $Klein$; $found := TRUE$;
    if not *found* $l++$;
if $Small \neq \emptyset$, gebe *Failure* zurück;
Entferne alle Jobs, die ein Bin überfüllen und ordne sie einer Liste J_R zu.
Betrachte die Bins s_{ij} nacheinander und verschiebe die Jobs und Wartungsaufträge, so dass im Schedule keine Lücken entstehen. Setze einen Wartungsauftrag nach dem letzten Job eines jeden Bins.
Setze die Jobs $j \in J_R$ mit einer *List*–Regel auf die Maschinen. Falls ein Job nicht auf eine Maschine gesetzt werden kann, ohne dass die Maintenance–Restriktionen verletzt werden, setze einen neuen Wartungsauftrag auf die Maschine.
Gebe den Schedule σ aus.

2.3 Analyse

Theorem 2.3
Algorithmus A_ϵ erzeugt in linearer Zeit einen Schedule σ für eine Instanz I des Problems Maintenance–Scheduling bei einer konstanten Anzahl von Maschinen. Die Approximationsgüte von Algorithmus A_ϵ beträgt $1 + 8\epsilon$.

Wir zeigen erst, dass der Algorithmus immer einen zulässigen Schedule mit der gewünschten Approximationsgüte erzeugt. Danach werden wir die lineare Laufzeit des Algorithmus verifizieren.

Lemma 2.4
Algorithmus A_ϵ erzeugt für alle $C \geq C_{Opt}$ einen zulässigen Schedule.

Beweis. Es genügt den Fall $S \geq \epsilon \cdot C$ zu betrachten. Für diesen Fall genügt es zu zeigen, dass das Einsetzen der kleinen Jobs nicht zu *Failure* führt. *Failure* bedeu-

tet aber, dass die gesamte Prozesszeit zuzüglich der minimalen Anzahl benötigter Wartungsaufträge die gegebene Maschinenkapazität übersteigt. Andererseits werden die Jobs aus *Groß* mit einer minimalen Anzahl an Wartungsaufträge auf die Maschinen gesetzt. Nach dem Einsetzen der kleinen Jobs ist die Kapazität S jeder Kiste vollständig ausgenutzt, da wir die Kisten durch ein Item sogar überfüllen. Daher wird der Algorithmus für $C \geq C_{Opt}$ nicht den Wert *Failure* zurückgeben. Somit wird der Algorithmus aber im Falle $C \geq C_{Opt}$ niemals terminieren, ohne einen zulässigen Schedule erzeugt zu haben. □

Korollar 2.5
Für $\epsilon < 1$ findet der Algorithmus einen zulässigen Schedule mit Makespan C_{A_ϵ}, so dass die Approximationsgüte des Algorithmus höchstens

$$\frac{C_{A_\epsilon}}{C_{Opt}} \leq (1 + 7\epsilon)$$

beträgt.

Beweis. Wir betrachten die beiden Fälle für Algorithmus A_ϵ getrennt.
Fall $S \leq \epsilon \cdot C$:
Sei k die Zahl der Wartungsaufträge in einem optimalen Schedule. Da der Algorithmus ein *FPTAS* zum Lösen von *Bin–Packing* mit Approximationsgüte $(1+\epsilon)$ anwendet, benötigt er maximal $k \cdot \epsilon$ zusätzliche Wartungsaufträge. Da die Wartungsaufträge jeweils die Prozesszeit 1 haben, wird die gesamte Prozesszeit im Schedule also nur um maximal $k \cdot \epsilon$, größer als im optimalen Schedule sein. Nach Voraussetzung wissen wir außerdem, dass die Prozesszeit des längsten mit dem *List*–Algorithmus eingeplanten Jobs nicht den Wert $\epsilon \cdot C$ übersteigt. Daher erfüllt der Makespan des in diesem Teil des Algorithmus erzeugten Schedules wird, die Ungleichung

$$C_{A_\epsilon} \leq \frac{1}{m}(\sum_{i=1}^{n} p_i + k + k \cdot \epsilon) + C \cdot \epsilon.$$

Andererseits wissen wir, dass $C_{Opt} \geq \frac{1}{m}(\sum_{i=1}^{n} p_i + k)$ und dass $\frac{C}{4} \leq C_{Opt}$. Insgesamt erhalten wir eine Approximationsgüte von

$$\frac{C_{A_\epsilon}}{C_{Opt}} \leq 1 + \frac{\frac{1}{m} \cdot k\epsilon}{\frac{1}{m} \cdot k} + 4\epsilon \leq 1 + 5\epsilon.$$

Fall $S > \epsilon \cdot C$:
Wir erinnern uns, dass der Algorithmus erst einen (nicht zulässigen) Schedule σ' mit Makespan $(1+\epsilon)\cdot C$ erzeugt, bei dem sich jeweils ein einzelner Job mit je einem Wartungsauftrag überlappen darf. Für jede Kiste wird dann ein Job entfernt und der Liste J_R zugeordnet. Diese Jobs J_R werden mit einem *List*–Algorithmus den Maschinen zugeordnet. Da $S \geq \epsilon \cdot C$ und da jede Kiste, mit Ausnahme

der letzten Kiste auf jeder Maschine, bis zu ihrer vollen Kapazität gefüllt ist, kann der Schedule σ' höchstens $\frac{m\cdot(1+\epsilon)}{\epsilon}$ viele Kisten enthalten. Daher beträgt die gesamte Größe der Jobs in J_R maximal $\frac{m\cdot(1+\epsilon)}{\epsilon}\cdot\epsilon^2\cdot C = m\cdot(1+\epsilon)\cdot\epsilon\cdot C$ (wegen $J_R \subset \mathit{Small}$ ist die Prozesszeit dieser Jobs durch $\epsilon^2\cdot C$ beschränkt). Die Jobs J_R können in einen anfangs leeren Schedule mit einem Makespan von höchstens $(\frac{1}{m}\sum_{j\in J_R} p_j + \max_{j\in J_R}(p_j))\cdot 2 \leq (2\epsilon + 4\epsilon^2)\cdot C$ eingeplant werden, wobei sich der Faktor 2 aus der Tatsache ergibt, dass man einfach nach jeder Einheit einen Wartungsauftrag setzten kann. Außerdem verwenden wir $S \geq \max_{j\in J_R}(p_j) \geq 1$. Wenn wir nun noch die übrige Kapazität bis zu $C + \epsilon\cdot C$ auf jeder Maschine ausnutzen, so hat der resultierende Schedule eine Makespan von höchstens

$$C\cdot(1+3\epsilon+4\epsilon^2) < C\cdot(1+7\epsilon).$$

. □

Es verbleibt, die lineare Laufzeit des Algorithmus nachzuweisen. Zuerst bestimmen wir die Laufzeit des Algorithmus in einer Iteration der binären Suche:
Fall $S < \epsilon\cdot C$*:*

- Das *FPTAS* zum Approximieren von *Bin–Packing* benötigt lineare Laufzeit.
- Der nachfolgende *List*–Algorithmus benötigt ebenfalls lineare Laufzeit.

Fall $S \geq \epsilon\cdot C$*:*

- Die Mengen *Groß* und *Klein* können in linearer Zeit ermittelt werden.
- Die vollständige Enumeration kann in konstanter Zeit durchgeführt werden, da m und ϵ konstant sind.

Man kann leicht nachrechnen, dass der resultierende Schedule nach k Iterationen der binären Suche einen Makespan von höchstens $(1+7\epsilon)(1+3\ \cdot 2^{-k})C_{Opt}$ hat. Daher werden insgesamt $\mathcal{O}(\log(\frac{1}{\epsilon}))$ Iterationen benötigt, um eine $(1+8\epsilon)$–Approximation zu erhalten. Damit ist die lineare Laufzeit des Algorithmus und somit auch das Theorem vollständig beweisen.

2.4 Asymptotisches polynomielles Approximationsschema

Ein einfaches *APTAS* für das Problem Maintenance–Scheduling bei einer variablen Anzahl von Maschinen ist das folgende: Als erstes wird ein *PTAS* für das einfachere Schedulingproblem angewendet, bei dem zunächst keine Wartungsaufträge eingeplant werden müssen. Dieses Problem stimmt mit dem klassischen

Problem des Scheduling auf parallelen Maschinen überein, bei dem der Makespan durch ein *PTAS* von Hochbaum und Shmoys [HS87] beliebig gut approximiert werden kann. Eine Instanz besteht dabei aus der Liste J der Jobs, der Zahl m der Maschinen und der gewünschten Approximationsgüte ϵ. In der folgenden Phase wird der Schedule um die Wartungsaufträge ergänzt. Dazu werden die Maschinen der Reihe nach von links nach rechts durchmustert und sobald die gesamte Prozesszeit einer Sequenz hintereinanderliegender Jobs startend vom letzten Wartungsauftrag (bzw. vom Beginn des Schedules für den ersten Wartungsauftrag) den Wert S übersteigt, wird ein Wartungsauftrag eingesetzt und die übrigen Jobs werden entsprechend verschoben. Schließlich wird jede Maschine mit einem Wartungsauftrag abgeschlossen. Im folgenden bezeichnen wir die Familie der Algorithmen, die das *APTAS* bilden, mit $\{A_\epsilon^\infty\}$.

Theorem 2.6
Jeder Algorithmus A_ϵ^∞ findet in linearer Zeit einen zulässigen Schedule für eine Instanz I des Problems Maintenance–Scheduling auf einer beliebigen Anzahl von Maschinen. Die Approximationsgüte von Algorithmus A_ϵ^∞ beträgt $1+6\epsilon$ für Instanzen I mit $C_{Opt}(I) > \frac{1}{\epsilon^2}$.

Beweis. Betrachte eine Instanz I mit einem optimalen Makespan $C_{Opt} \geq \frac{1}{\epsilon^2}$. Sei $C \geq C_{Opt}$ der Kandidat für den Makespan in einer Iteration der binären Suche. Da $S \geq \epsilon \cdot C$ und $C_{Opt} \geq \frac{1}{\epsilon^2}$, folgt $S > \frac{1}{\epsilon}$. Sei C^* der temporäre Makespan nach Anwenden des *PTAS*. Dann gilt,

$$C^* \leq (1+\epsilon)C_{Opt}. \tag{2.1}$$

Die Zahl der Wartungsaufträge, die im folgenden Schritt von Algorithmus A_ϵ^∞ eingesetzt werden, ist für jede Maschine beschränkt durch

$$\frac{2C^*}{S} + 2 \leq 2\epsilon C^* + 2 \leq 2\epsilon C^* + \epsilon C_{Opt}, \text{für } \epsilon < \frac{1}{2}. \tag{2.2}$$

Um dies zu zeigen, nehmen wir das Gegenteil an, also dass mehr als $\frac{2C^*}{S}+2$ Wartungsaufträge auf einer Maschine k benötigt werden. Da die gesamte Prozesszeit der Jobs zweier aufeinanderfolgender Intervalle zwischen den Wartungsaufträgen den Wert S übersteigen muss — zumindest für die ersten $\frac{2C^*}{S}$ Intervalle —, würde die gesamte Prozesszeit der Jobs auf der Maschine den Wert C^* übersteigen. C^* ist aber wiederum der Makespan, wenn Wartungsaufträge nicht benötigt werden. Daher gilt die Ungleichung (2.2) und zusammen mit Ungleichung (2.1) erhalten wir einen Makespan $C_{A_\epsilon^\infty}$ durch Algorithmus A_ϵ^∞:

$$C_{A_\epsilon^\infty} \leq (1+\epsilon)C_{Opt} + 2\epsilon(1+\epsilon)C_{Opt} + \epsilon C_{Opt} \leq (1+5\epsilon)C_{Opt}. \tag{2.3}$$

Wenn man nun wieder $\mathcal{O}(\log(\frac{1}{\epsilon}))$ Iterationen bei der binären Suche innerhalb des *PTAS* für das Problem ohne Maintenance–Restriktionen durchführt, erhält man schließlich eine Approximationsgüte von $1+6\epsilon$. □

2.5 Online–Algorithmen für das Problem des Maintenance–Scheduling

Approximationsschemata sind bestmögliche Approximationsresultate für ein NP–schweres Problem. Sie ermöglichen hier, dass ein beliebig guter Schedule erzielt werden kann. Für die praktische Anwendung ergeben sich jedoch zwei gravierende Probleme, so dass die in den vorigen Abschnitten betrachteten Algorithmen letztlich nicht praktikabel sind. Erstens steigt die Laufzeit der vorgestellten Algorithmen exponentiell in $1/\epsilon$. Damit wird jeder realistische Rahmen für die zur Verfügung stehende Rechenzeit selbst bei kleinen Instanzen und moderater Approximationsgüte überschritten. Zweitens sind die Algorithmen nicht für den Online–Fall verwendbar. Unsere Approximationsschemata setzten voraus, dass alle Jobs mit ihren Prozesszeiten von vorneherein bekannt sind. In einem Online–Szenario hingegen entstehen die Jobs erst im Laufe der Zeit. Bei der im zweiten Teil dieser Arbeit betrachteten Anwendung der Produktionsplanung tritt aber zum Beispiel genau dieser Fall ein. So ist bei der Produktionsplanung am Anfang der Woche noch nicht bekannt, mit welchen Aufträgen bis zum Wochenende zu rechnen ist. Die Aufträge entstehen "Online" erst im Laufe der Woche.
Motiviert aus der Anwendung werden wir daher in diesem Kapitel das Online–Modell betrachten dafür die Approximationsgüte des *List*–Algorithmus bestimmen. In unserem Online-Modell entstehen die Jobs sukzessive und sie sind bei ihrer Ankunft sofort einzuplanen. Sind sie einmal eingeplant, dürfen sie nicht wieder auf andere Zeitpunkte oder andere Maschinen gesetzt werden. Wir untersuchen eine natürliche Erweiterung *List–M* des standard *List*–Algorithmus: *List–M* setzt jeden Job j auf diejenige Maschine, die aktuell die geringste gesamte Prozesszeit bearbeiten muss. Sei J die Liste der Jobs sortiert nach den Ankunftszeiten der Jobs.

Algorithmus 2.5.1 List–M

while $J \neq \emptyset$

Nehme den ersten Job $j \in J$ und setze j auf die Maschine, die aktuell die kürzeste Belegung hat. Falls der Job nicht abgearbeitet werden kann, ohne dass das maximale Intervall zwischen zwei Wartungsaufträgen überschritten wird, setze einen Wartungsauftrag vor j;
Entferne j aus J;

Schließe jede Maschine mit einem Wartungsauftrag.

Man kann das Einplanen der Jobs wiederum als ein Packen von Jobs in Kisten, die auf den einzelnen Maschinen liegen, auffassen. Dabei werden allerdings einzelne Kisten geschlossen, obwohl möglicherweise noch zukünftige Jobs in diese Kisten hineinpassen würden.
Im Folgenden bezeichnen wir mit C_{Opt} den Makespan eines optimalen (offline) Schedules und mit C_{List-M} den Makespan des durch *List–M* erzeugten Online–Schedules.

Zur weiteren Analyse des Algorithmus benötigen wir zunächst das folgende:

Definition 2.7
Ein Schedule σ für eine Instanz I des Problems Maintenance–Scheduling heißt reduziert, *falls kein Wartungsauftrag aus σ entfernt werden kann, ohne dass die maximale Zeit zwischen zwei Wartungsaufträgen überschritten wird.*

Lemma 2.8
Sei σ_{Opt} ein reduzierter optimaler Schedule für eine gegebene Instanz des Problems Maintenance–Scheduling. Ist $k > 1$ die maximale Anzahl von Wartungsaufträgen auf einer Maschine in σ_{Opt}, so erfüllt jeder Job $j \in J$

$$p_j \leq C_{Opt} - k - i, \text{ falls } k = 2 + i.$$

Beweis. Sei l diejenige Maschine mit $k = 2+i$ Wartungsaufträgen. Angenommen, es gäbe einen Job j mit $p_j > C_{Opt} - k - i$. Die gesamte Prozesszeit [1] auf Maschine l, die vor dem zweiten Wartungsauftrag liegt, übersteigt S, da $p_j > C_{Opt} - k - i$, also $S > C_{Opt} - k - i$. Außerdem gibt es für jeden der folgenden i Wartungsaufträge mindestens einen weiteren Job $j' \in J$ und damit auch mindestens einen zusätzliche Einheit an Prozesszeit. Daraus würde folgen

$$C_{Opt} > S + k + i > C_{Opt} - k - i + k + i = C_{Opt}.$$

□

Lemma 2.9
Sei σ_{Opt} ein reduzierter optimaler Schedule für eine gegebene Instanz des Problems Maintenance–Scheduling und sei k die maximale Anzahl an Wartungsaufträgen auf einer Maschine in σ_{Opt}. Sei k_{Opt} die gesamte Anzahl von Wartungsaufträgen in σ_{Opt}. Dann gelten die folgenden unteren Schranken für den optimalen Makespan C_{Opt}:

$$C_{Opt} \geq \max\{\frac{1}{m}(\sum_{j=1}^{n} p_j) + 1, \max_{j \in J}\{p_j\} + 1, \frac{1}{m}(\sum_{j=1}^{n} p_j + k_{Opt})\}. \tag{2.4}$$

Für $k \geq 2$ gilt

$$C_{Opt} \geq \frac{1}{m}(\sum_{j=1}^{n} p_j) + 2. \tag{2.5}$$

[1] Unter *gesamter Prozesszeit* wird die Prozesszeit der Jobs ohne Wartungsaufträge verstanden.

Beweis. Gleichung (2.4) folgt unmittelbar, da sich auf jeder Maschine mindestens ein Wartungsauftrag befindet. Daher beweisen wir nur Gleichung (2.5). Für $k \geq 2$ betrachten wir die Maschine r mit der größten gesamten Prozesszeit P (da $k \geq 2$, gilt $P > S$). Mindestens 2 Wartungsaufträge liegen auf r, da die gesamte Prozesszeit auf den Maschinen mit nur einem Wartungsauftrag durch S beschränkt ist. Es folgt $C_{Opt} \geq P + 2$. Da P die größte gesamte Prozesszeit auf einer Maschine in σ_{Opt} bezeichnet, gilt $\sum_{i=j}^{n} p_j \leq m \cdot P$. Daher gilt

$$\frac{1}{m}(\sum_{j=1}^{n} p_j) + 2 \leq P + 2 \leq C_{Opt}.$$

□

Nun wenden wir Lemma 2.8 und Lemma 2.9 an, um eine obere Schranke für den Algorithmus *List–M* im Worst–Case zu erhalten.

Theorem 2.10
Algorithmus LIST–M ist 2–kompetitiv.

Beweis. Wir betrachten einen beliebigen reduzierten optimalen Schedule σ_{Opt}. Sei $J = \{1, \cdots, n\}$ und sei l der als letztes fertiggestellte Job im Schedule σ_{List-M}, der unter *List–M* entsteht. Sei ferner i die Maschine, auf der l eingeplant ist. Wir definieren zusätzlich k und k_{Opt} als die *maximale* Anzahl an Wartungsaufträgen auf einer Maschine beziehungsweise die *gesamte* Anzahl an Wartungsaufträgen in σ_{Opt}.
Wir geben nun fünf charakteristische Eigenschaften des Schedules σ_{LIST-M}:

Eigenschaft 1: *Sei $\hat{k}_r$ die Anzahl der Wartungsaufträge auf Maschine r ($r = 1 \cdots m$), die vor Job l starten. Dann gilt für alle r,*

$$\hat{k}_r \leq 2 \cdot k - 1. \tag{2.6}$$

Beweis. Angenommen, die Aussage gelte nicht. Das bedeutet $\hat{k}_{r_0} \geq 2 \cdot k$, für eine Maschine r_0. Dann hat r_0 vor dem Start von l eine gesamte Prozesszeit von mehr als $k \cdot S$, da die gesamte Belegungsdauer zweier hintereinanderliegender Produktionsintervalle den Wert S übersteigt. Die gleiche gesamte Prozesszeit, oder sogar eine größere, haben die Jobs auf den Maschinen mit mindestens $2 \cdot k$ Wartungsaufträgen *und* ebenso alle Maschinen r mit $\hat{k}_r < 2 \cdot k$ (da jede Maschine mindestens bis zum Start von l belegt ist). Daher ist also jeder Maschine mehr als $k \cdot S$ an gesamter Prozesszeit zugeordnet, was der Tatsache widerspricht, dass auf jeder Maschine höchstens k Wartungsaufträge in σ_{Opt} liegen. □

Eigenschaft 2: *Sei $\hat{k}_r$ die Anzahl der Wartungsaufträge auf Maschine r ($r = 1 \cdots m$), die vor Job l starten. Dann gilt*

$$C_{LIST-M} \leq \frac{1}{m}(\sum_{j \neq l} p_j + \sum_{r=1}^{m} \hat{k}_r) + p_l + 1. \tag{2.7}$$

Beweis. Die Prozesszeit der Jobs und Wartungsaufträge, die vor l in σ_{List-M} starten, ist durch $\sum_{j\neq l} p_j + \sum_{r=1}^{m} \hat{k}_r$ beschränkt. Da jede Maschine zumindest bis zum Start von l belegt ist, ist Maschine i wiederum bis zum Start von l mit maximal $\frac{1}{m}(\sum_{j\neq l} p_j + \sum_{r=1}^{m} \hat{k}_r)$ belegt. □

Eigenschaft 3: *Für den Makespan C_{LIST-M} des Schedules σ_{LIST-M} gilt:*

$$C_{LIST-M} \leq \frac{1}{m}(\sum_{j\neq l} p_j) + 2 \cdot k + p_l.$$

Beweis. Anwenden der Eigenschaften 1 und 2. □

Eigenschaft 4: *Sei k_{LIST-M} die Anzahl der Wartungsaufträge, die im Schedule σ_{LIST-M} enthalten sind. Dann gilt*

$$C_{LIST-M} \leq \frac{1}{m}(\sum_{j\neq l} p_j + k_{LIST-M} - 1) + p_l + 1. \tag{2.8}$$

Beweis. Summiert man über alle Belegungsdauern aller Maschinen im Schedule σ_{LIST-M}, erhält man einen Wert von $\sum_{j=1}^{n} p_j + k_{List-M}$. Der zuletzt abgearbeitete Job l wird noch von einem Wartungsauftrag gefolgt. Zusätzlich gilt, dass jede Maschine mindestens bis zu dem Start von l belegt ist. Daher wird die maximale Belegung auf Maschine i genau dann erreicht, wenn die verbleibende gesamte Belegung $\sum_{j\neq l} p_j + k_{List-M} - 1$ auf die Maschinen vor dem Start von l gleichverteilt ist. □

Eigenschaft 5: List–M *erzeugt weniger als $2 \cdot k_{Opt} + m$ Wartungsaufträge.*

Beweis. Um zu sehen, dass *List–M* weniger als $2 \cdot k_{Opt} + m$ Wartungsaufträge erzeugt, nehmen wir an, wir hätten mindestens $2 \cdot k_{Opt} + m$ Wartungsaufträge. Da wiederum die gesamte Prozesszeit in zwei aufeinanderfolgenden Intervallen zwischen Wartungsaufträgen den Wert S übersteigt, hat jede Maschine r eine gesamte Prozesszeit von mehr als $\lfloor \frac{k_r}{2} \rfloor \cdot S \geq \frac{k_r - 1}{2} \cdot S$, wobei k_r die Zahl der Wartungsaufträge auf Maschine r bezeichne. Daher gilt

$$\sum_{j=1}^{n} p_j > \frac{S}{2} \sum_{r=1}^{m} (k_r - 1) \geq \frac{S}{2} 2 \cdot k_{Opt} = S \cdot k_{Opt}.$$

Auf der anderen Seite ist die gesamte Prozesszeit durch die Anzahl der Wartungsaufträge in einem optimalen Schedule multipliziert mit S, also durch $S \cdot k_{Opt}$ beschränkt. □

Wir werden diese fünf Eigenschaften anwenden, um die Approximationsgüte von 2 von Algorithmus *List–M* nachzuweisen. Wir unterscheiden mehrere Fälle:

Fall $k = 1$:
Unter Verwenden von Eigenschaft 3 und Ungleichung (2.4) von Lemma 2.9 erhalten wir

$$\begin{aligned} C_{LIST-M} &\leq \frac{1}{m}(\sum_{j=1}^{n} p_j) + 1 + p_l + 1 - \frac{p_l}{m} \\ &\leq 2 \cdot C_{Opt} - \frac{p_l}{m} < 2 \cdot C_{Opt}. \end{aligned}$$

Fall $k = 2, 3$:
Lemma 2.8 ergibt $p_l \leq C_{Opt} - k$ (bzw. $p_l \leq C_{Opt} - k - 1$ für $k = 3$). Die Abschätzung (2.5) von Lemma 2.9 zeigt, dass $C_{Opt} \geq \frac{1}{m}(\sum_{j=1}^{n} p_j) + k$ (bzw. $C_{Opt} \geq \frac{1}{m}(\sum_{j=1}^{n} p_j) + k - 1$). Wir wenden wiederum Eigenschaft 3 an:

$$\begin{aligned} C_{LIST-M} &\leq \frac{1}{m}(\sum_{j \neq l} p_j) + 2 \cdot k + p_l \\ &\leq C_{Opt} + k + (C_{Opt} - k)(1 - \frac{1}{m}) < 2 \cdot C_{Opt}. \end{aligned}$$

Fall $k > 4$:
Da $k > 4$ gilt, können wir nicht die untere Schranke aus Lemma 2.9 (Ungleichung (2.5)) anwenden. Lemma 2.8 besagt jedoch, dass $p_l \leq C_{Opt} - k - 2$. Verwenden wir $k_{Opt} \leq m \cdot k$ und Eigenschaften 4 und 5, so erhalten wir:

$$\begin{aligned} C_{LIST-M} &\leq \frac{1}{m}(\sum_{j \neq l} p_j \ + 2k_{Opt} + m - 1) + p_l + 1 \\ &\leq \frac{1}{m}(\sum_{j=1}^{n} p_j \ + k_{Opt}) + 1 + \frac{k_{Opt}}{m} - \frac{1}{m} + 1 + p_l(1 - \frac{1}{m}) \\ &< C_{Opt} + k + 2 + (C_{Opt} - k - 2)(1 - \frac{1}{m}) < 2 \cdot C_{Opt}. \end{aligned}$$

Das schließt den Beweis des letzten Falles ab und damit ist das gesamte Theorem bewiesen. □

Bemerkung 2.11
Wenn wir nicht mehr von rationalen *Prozesszeiten ausgehen, sondern nur noch ganzzahlige Prozesszeiten zulassen, kann unsere Aussage auf einfachere Weise bewiesen werden. Wir geben dazu nur die Schranke $p_j \leq C_{Opt} - k - 1$ aus Lemma 2.8 als Beispiel. Setzt man ganzzahlige Prozesszeiten voraus, so kann sie auch für $k = 2$ nachgewiesen werden. Das vereinfacht den Beweis von Theorem 2.10 da der Fall $p_l > C_{Opt} - k - 1$ nicht mehr für $k = 2$ betrachtet werden muss. Für rationale Prozesszeiten gilt dies Schranke jedoch nicht. Betrachte zum Beispiel das Problem auf zwei Maschinen mit drei Jobs 1, 2 und 3, wobei $p_1 = S - 1 + \frac{1}{10}$, $p_2 = 1$ und $p_3 = S$. Der optimale Schedule ordnet 1 und 2 einer Maschine zu und 3 der zweiten Maschine. Dann gilt $C_{Opt} - 3 = S + \frac{1}{10} - 1 < S = p_3$.*

Schon das einfache Beispiel der Familie von Instanzen $\{I_m\}$ auf m Maschinen, $S := m$ und einer Menge von $m(m-1)$ Jobs mit Prozesszeit 1, sowie einem Job mit der Prozesszeit m zeigt, dass die Schranke aus dem Beweis von Theorem 2.10 für $k = 1$ scharf ist, denn es gilt $C_{Opt} = m + 1$ und $C_{Alg} = 2m + 1$.
In der praktischen Anwendung finden wir jedoch häufig eine konstante Anzahl von Maschinen und im Vergleich zum betrachteten Zeithorizont verhältnismäßig kleine Intervalle zwischen zwei Wartungsaufträgen. In diesem Fall, wenn also m fest gewählt ist und k und C_{Opt} verhältnismäßig groß sind, hat *List–M* eine deutlich bessere Performance. Um dies zu zeigen, verwenden wir das folgende Lemma um die Prozesszeit der Jobs zu beschränken:

Lemma 2.12
Sei σ_{Opt} ein reduzierter optimaler Schedule für eine gegebene Instanz des Problems Maintenance–Scheduling. Ist $k > 1$ die maximale Anzahl der Wartungsaufträge auf einer Maschine im Schedule C_{Opt}, dann gilt $p_j \leq \frac{2}{k-1} \cdot C_{Opt} - \frac{2k}{k-1}$ für jeden Job j.

Beweis. Wir nehmen an, es gäbe einen Job j mit

$$p_j > \frac{2}{k-1} \cdot C_{Opt} - \frac{2k}{k-1}.$$

Dann gilt auch

$$S > \frac{2}{k-1} \cdot C_{opt} - \frac{2k}{k-1}.$$

Betrachte nun die Maschine, auf der k Wartungsaufträge liegen. Da die Summe der Prozesszeiten für zwei aufeinanderfolgende Intervalle zwischen zwei Wartungsaufträgen den Wert S übersteigt, gilt:

$$C_{Opt} > \lfloor \frac{k}{2} \rfloor \cdot S + k \geq \frac{k-1}{2}(\frac{2}{k-1} \cdot C_{Opt} - \frac{2k}{k-1}) + k \geq C_{Opt} - k + k \geq C_{Opt}.$$

Dies ist aber ein Widerspruch und daher gilt die Aussage des Lemmas. □

Mit Lemma 2.12 erhalten wir:

Theorem 2.13
Für $k > 1$ gilt

$$C_{LIST-M} \leq C_{Opt}(1 + \frac{2}{k-1} - \frac{2}{m(k-1)}) + k + 2 - \frac{1}{m}\frac{2k}{k-1} + \frac{2k}{m(k-1)}.$$

Beweis. Aufgrund der Eigenschaften 4 und 5, $k_{Opt} \leq m \cdot k$, Lemma 2.9 und Lemma 2.12 gilt,

$$\begin{aligned}
C_{LIST-M} &\leq \frac{1}{m}(\sum_{j \neq l} p_j \ + m - 1 + 2 \cdot k_{Opt}) + p_l + 1 \\
&\leq \frac{1}{m}(\sum_{j=1}^{n} p_j \ + k_{Opt}) + 1 - \frac{1}{m} + \frac{k_{Opt}}{m} + 1 + p_l(1 - \frac{1}{m}) \\
&\leq C_{Opt} + k + 2 - \frac{1}{m} + (\frac{2}{k-1} \cdot C_{Opt} - \frac{2k}{k-1})(1 - \frac{1}{m}) \\
&\leq C_{Opt}(1 + \frac{2}{k-1} - \frac{2}{m(k-1)}) + k + 2 - \frac{1}{m} - \frac{2k}{k-1} + \frac{2k}{m(k-1)}.
\end{aligned}$$

□

Setzt man zum Beispiel $m := 10$ und $k := C_{Opt}/4$, so folgt für nach unendlich strebendes C_{Opt}, dass unsere obere Schranke für $\frac{C_{LIST-M}}{C_{Opt}}$ nach 1.35 strebt. Allerdings kann der Algorithmus *List–M* für großes C_{Opt} nicht beliebig gut werden. Genauer gilt, dass Algorithmus *List–M* sich nicht der Approximationsgüte 1 nähert, auch nicht asymptotisch. Um dies einzusehen, betrachten wir das Problem auf einer Maschine mit $S := 2$ und $2n$ Jobs mit Prozesszeit 1 und $2n$ Jobs mit Prozesszeit 2. Im schlimmsten Falle treten die Jobs mit Prozesszeiten 1 und 2 alternierend auf und dann gilt $C_{LIST-M} = C_{Opt} + n$.

KAPITEL 3

Scheduling mit fixierten Jobs

In diesem Kapitel betrachten wir das zweite Modell zum Scheduling bei limitierter Maschinenverfügbarkeit. In dieser Variante nehmen wir an, dass eine Zahl k der gegebenen n Jobs bereits fixiert in dem Schedule vorliegt. Die Aufgabe besteht darin, die verbleibenden $n - k$ Jobs mit einem minimalem Makespan C_{max} einzuplanen.
Durch eine einfache Reduktion von 3–Partition sieht man sofort, dass das Scheduling–Problem mit fixierten Jobs selbst bei einer einzigen Maschine stark NP–vollständig ist. Auch für das Scheduling–Problem mit fixierten Jobs, gelingt es uns, die Approximierbarkeit genau zu spezifizieren. Wir erhalten ähnliche Aussagen, wie für *Maintenance–Scheduling*, jedoch wird der Aufwand, um an derartige Aussagen zu gelangen, deutlich höher. Ebenso sind die verwendeten Algorithmen technisch anspruchsvoller, als die in den vorherigen Kapiteln vorgestellten Algorithmen.
Wir stellen ein polynomielles Approximationsschema für den Fall einer konstanten Maschinenzahl vor. Wir erweitern die Resultate auf die Problemvariante, bei der Maschinen mit verschiedenen Geschwindigkeiten arbeiten. Auch hier sind unsere Ergebnisse bestmöglich, denn auch für das Scheduling mit fixierten Jobs existiert kein voll–polynomielles Approximationsschema selbst bei konstanter Maschinenzahl. Bei variablen Maschinenzahl existiert nicht einmal ein asymptotisches polynomielles Approximationsschema, sofern P $\neq$ NP.
Auch beim Scheduling mit fixierten Jobs ist es sinnvoll, die Aufgabenstellung als eine geschickte Art des Packens von Jobs in Kisten zu interpretieren. Dazu werden die Lücken zwischen zwei aufeinanderfolgenden fixierten Jobs auf einer Maschine als "geschlossene" Kisten aufgefasst, die in keinem Fall überfüllt werden dürfen. Zusätzlich erhält man eine weitere Kiste für jede Maschine, die nach dem letzten fixierten Job der Maschine startet und "offen" ist, d. h. beliebig gefüllt werden darf. Die zu packenden Objekte sind die $n - k$ nicht fixierten Jobs. Diese Idee wird allen im Folgenden betrachteten Algorithmen zugrunde liegen.

3.1 Negative Approximationsresultate

Mit den folgenden beiden negativen Approximationsaussagen zeigen wir die Grenzen für die Entwicklung potentieller Approximationsschemata beim Scheduling mit fixierten Jobs auf.

Theorem 3.1
Für eine konstante Anzahl m von Maschinen existiert kein voll–polynomielles Approximationsschema für das Problem des Scheduling mit fixierten Jobs, solange nicht $P = NP$ gilt.

Beweis. Analog zum Beweis von Theorem 2.1 werden wir eine Reduktion zum stark NP–schweren Problem *3–Partition* durchführen. Damit kann kein voll polynomielles Approximationsschema existieren.
Sei für unsere Reduktion eine Instanz von *3–Partition* gegeben. Daraus erstellen wir eine Instanz des Schedulingproblems auf einer Maschine in der folgenden Weise. Für jedes Element erzeugen wir einen Job, dessen Prozesszeit genau der Größe des Elementes entspricht. Außerdem platzieren wir einen fixierten Job mit Prozesszeit 1 zu den Zeitpunkten $k \cdot B + (k-1)$ für alle $1 \leq k \leq n$. Es ist dann klar, dass genau dann ein Schedule mit Makespan $n \cdot (B+1)$ existiert, wenn die gewünschte Partition der Elemente in n Mengen möglich ist. □

Theorem 3.2
Ist die Zahl m der Maschinen Teil der Eingabe existiert kein polynomieller Algorithmus, der das Schedulingproblem mit fixierten Jobs mit einer garantierten Approximationsgüte von $\frac{3}{2} - \epsilon$ für alle $\epsilon > 0$ lösen kann, solange nicht $P = NP$ gilt.

Beweis. Diesmal reduzieren wir von der folgenden ebenfalls NP–vollständigen ([GJ79]) Variante von *3–Partition*:

EINGABE: Disjunkte Mengen A und B, die n bzw. $2n$ Elemente der Größen $a_1, \ldots, a_n \in \mathbb{Z}^+$ bzw. $b_1, \ldots, b_{2n} \in \mathbb{Z}^+$ enthalten und eine Schranke $L \in \mathbb{Z}^+$, so dass $\sum a_i + \sum b_j = nL$. FRAGE: Gibt es eine Permutation $\pi \in \mathcal{S}_{2n}$, so dass für alle $1 \leq i \leq n$ gilt: $a_i + b_{\pi(2i-1)} + b_{\pi(2i)} = L$?

Nun betrachten wir n Maschinen und wählen eine ganze Zahl K, so dass $K \geq (\frac{1}{2} - \epsilon)L/(2\epsilon)$. Wir fixieren für jede Maschine M_i, $1 \leq i \leq n$ einen Job mit Prozesszeit a_i der zum Zeitpunkt $2K+L$ endet. Neben den fixierten Jobs besteht die Eingabeinstanz aus $2n$ nicht fixierten Jobs mit Prozesszeiten $K + b_i$, $1 \leq i \leq 2n$. Es ist klar, dass genau dann ein Schedule mit Makespan $2K+L$ existiert, wenn die gewünschte Permutation π existiert. Außerdem gilt, dass, wenn eine derartige Permutation nicht existiert, jeder Schedule einen Makespan von mindestens $3K+$

L hat. Da aber $(3K + L)/(2K + L) \geq 3/2 - \epsilon$ nach Wahl von K, folgt damit die Aussage des Theorems. □

Der Beweis von Theorem 3.2 impliziert, dass die untere Schranke von $\frac{3}{2}$ auch für große Werte C_{Opt} gültig bleibt. Als Konsequenz kann dann aber auch — im Gegensatz zum Problem des Maintenance–Scheduling — kein *asymptotisches* polynomielles Approximationsschema (*APTAS*) für das Scheduling–Problem mit fixierten Jobs und einer variablem Maschinenzahl existieren, sofern nicht P = NP gilt.
Es verbleibt als bestmögliches Resultat ein polynomielles Approximationsschema für den Fall einer konstanten Maschinenzahl. Ein solches wollen wir im folgenden Abschnitt vorstellen.

3.2 Polynomielles Approximationsschema

Wir setzten voraus, dass die Zahl m der Maschinen konstant gegeben ist. Für unser polynomielles Approximationsschema interpretieren wir das Problem wieder als eine Variante des Problems, Kisten zu packen. Diesmal wird es als Subproblem in einer binären Suche für den Makespan auftreten. Die noch zu spezifizierende Variante dieses Problems nennen wir *VBP* (für engl. *variable size bin packing*). Die Kisten sind jeweils als geschlossene Kisten für die Zwischenräume zwischen zwei fixierten Jobs auf einer Maschine und als offene Kisten nach dem letzten fixierten Job auf einer Maschine gegeben. Genauer verwenden wir innerhalb einer binären Suche einen Kandidaten C für den Makespan und definieren m offene Kisten gegeben durch die Zeiträume nach dem letzten fixierten Job bis zu dem Zeitpunkt C auf den m Maschinen. Das Attribut "offen" ist hier in dem Sinne zu verstehen, dass die Größe der Kisten keine starre Restriktion ist und dass die offenen Kisten überfüllt werden dürfen. Es ist klar, dass C genau dann eine obere Schranke für den Makespan ist, wenn alle Jobs in die Kisten gepackt werden können. Zusammengefasst hat unser Algorithmus die folgende Struktur:

- Wähle einen Kandidaten C für den Makespan durch binäre Suche.
- Prüfe, ob das durch C definierte Problem *VBP* eine zulässige Lösung hat, die die "geschlossenen" Kisten gar nicht und die "offenen" Kisten nur leicht überlädt.
- Gebe den Schedule zu dem kleinsten Wert C aus, zu dem eine zulässige Lösung gefunden werden konnte.

Um das weitere Vorgehen verstehen zu können, betrachten wir zunächst das Scheduling–Problem, bei dem kein Job fixiert ist. In diesem Fall entspricht *VBP* dem Problem *Bin–Packing*. Alle Kisten haben die gleiche Kapazität und sind offen. Das von Hochbaum und Shmoys [HS87] vorgestellte Approximationsschema

approximiert die *Zulässigkeit* von Bin–Packing, indem jede Kiste bis zu einem Faktor $(1+\epsilon)$ über ihre Kapazität hinaus gefüllt werden darf. Dies ergibt — jetzt als Scheduling–Problem aufgefasst — eine $(1+\epsilon)$–Approximation für den Makespan. Das dabei zu lösende, in der Zulässigkeit relaxierte Packproblem, ist als lineares Programm mit einer konstanten Anzahl an Variablen und geschickt gerundeten Jobgrößen verhältnismäßig einfach zu lösen. Kleine Jobs werden dabei erst in einem zweiten Schritt mit einem *First–Fit*–Verfahren gepackt.
Bei dem Versuch, ein derartiges Verfahren auf das Scheduling–Problem mit fixierten Jobs zu übertragen, treten jedoch zwei gravierende Probleme auf: Erstens kann das Runden der Jobgrößen zu einem unzulässigen Schedule bedingt durch Überlappen von fixierten mit nichtfixierten Jobs führen. Zweiten können selbst die kleinen Jobs nicht durch einen einfachen Greedy–Algorithmus gepackt werden, wie das folgende Beispiel zeigt.

Beispiel: *Sei* $\epsilon < \frac{1}{3}$ *gegeben und sei* $t > 0$ *eine ganze Zahl. Wir betrachten das folgende* 6 *Maschinenproblem mit* $18 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Jobs. Auf jeder Maschine befinden sich* $\lceil \frac{1}{\epsilon^{t+1}} \rceil$ *geschlossenen Kisten mit Länge* $\frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil}$*, die jeweils von einem infitisimal kleinen fixierten Job getrennt werden. Die gesamte Länge* Δ *der fixierten Jobs wird ebenfalls als beliebig klein vorausgesetzt. Die* $18 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Jobs der Eingabeinstanz* I *setzen sich wie folgt zusammen:*

- $6 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Jobs der Länge* $\frac{1}{7} \frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil} + \delta$,
- $6 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Jobs der Länge* $\frac{1}{3} \frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil} + \delta$,
- $6 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Jobs der Länge* $\frac{1}{2} \frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil} + \delta$.

Dabei bezeichne δ *eine geeignete kleine Konstante. Alle Jobs können in die geschlossenen Kisten gepackt werden, indem ein Job von jedem Typ in jede geschlossenen Kiste gesetzt wird. Daher gilt* $C_{opt} = 1+\Delta$*. Der* First–Fit*–Algorithmus wird dagegen wie folgt vorgehen:*

- $\lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Kisten werden mit* 6 *Jobs der Länge* $\frac{1}{7} \cdot \frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil} + \delta$ *gefüllt,*
- $3 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Kisten werden mit* 2 *Jobs der Länge* $\frac{1}{3} \cdot \frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil} + \delta$ *gefüllt,*
- $2 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Kisten werden mit einem Job der Länge* $\frac{1}{2} \cdot \frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil} + \delta$ *gefüllt.*

Dann verbleiben aber noch $4 \cdot \lceil \frac{1}{\epsilon^{t+1}} \rceil$ *Jobs der Länge* $\frac{1}{2} \cdot \frac{1}{\lceil \frac{1}{\epsilon^{t+1}} \rceil} + \delta$*, welche noch in die offenen Kisten gepackt werden müssen. Daher gilt für den Makespan* C_{FF} *unter Anwenden von* First–Fit*:*

$$C_{FF} \geq 1 + \Delta + \frac{2}{6} \geq 1 + \Delta + \epsilon.$$

Dann kann aber die Approximationsgüte von $1+\epsilon$ nicht erreicht werden, auch wenn alle Jobs durch Verwenden von großen Werten t beliebig klein sind. Mit ähnlichen Argumenten lässt sich nachweisen, dass sich unter keinem Greedy–Verfahren zum Packen der kleinen Jobs eine beliebig gute Approximation erreichen lässt.

Aufgrund der gerade genannten Schwierigkeiten ist ein elaborierterer Algorithmus notwendig. Das hier von uns vorgestellte Approximationsschema lässt sich in drei Bestandteile zerlegen. Der erste Baustein des Algorithmus ist eine Zerlegung des Problems *VBP* in eine Familie von hinreichend unabhängigen Packproblemen. Als zweites lösen wir die einzelnen Packprobleme als ein ganzzahliges lineares Programm. Der letzte Teil besteht aus einer Sequenz von Greedy–Algorithmen die die Jobs, welche bei den Lösungen der einzelnen Subprobleme übriggeblieben sind, noch hinzufügt. In den nächsten Abschnitten werden wir diese Bausteine unseres *PTAS* im Einzelnen vorstellen und schließlich zeigen, wie sie zu einem gesamten Algorithmus zum Approximieren des VBP zusammengefasst werden können.
Wir beginnen dabei mit den folgenden Definitionen:

Definition 3.3
Für eine beliebige Menge I bestehend aus Kisten oder Jobs sei $SIZE(I)$ die gesamte Größe/Prozesszeit aller Elemente in I. Wir definieren außerdem $\max(I)$ *und* $\min(I)$ *als die maximale resp. minimale Größe eines Elementes in I. Wir bezeichnen die Größe von Jobs i mit p_i und die Größe von Kiste j mit s_j. Bei Jobs verwenden wir die Begriffe* Prozesszeit *und* Größe *synonym.*

Zur Vereinfachung der Präsentation setzten wir des Weiteren voraus, dass alle Kisten und Jobs in den Instanzen von *VBP* mit $\frac{1}{C}$ skaliert seien. Ihre Größe ist daher durch 1 beschränkt. Schließlich sei noch ϵ eine beliebige, aber fest gewählte Konstante.

3.2.1 Baustein 1: Erzeugen der Subprobleme

Wir erzeugen eine Familie von Packproblemen, indem wir die Menge der Kisten entsprechend der Kistengrößen partitionieren. Wir erhalten die Partition, indem wir bestimmte Kisten identifizieren, die für die Gesamtlösung unerheblich sind und daher vernachlässigt werden können. Die erlaubt uns, das Packproblem *VBP* in separate Packprobleme zu zerlegen, für die sich die maximale und minimale Größe der Kisten nicht zu sehr unterscheidet.

Lemma 3.4
Gegeben sei eine Instanz I des via C definierten Packproblems VBP, in dem alle Kisten und Jobs mit $\frac{1}{C}$ skaliert und in dem die Kisten nach nicht aufsteigender Größe sortiert sind. Setzen wir $t = \lceil \frac{4}{\epsilon} \rceil + 3$ und $\epsilon < 1$, dann können wir in linearer

Zeit eine Zerlegung $B = \hat{B} \cup B_1 \cup \ldots \cup B_r$ *der Menge der Kisten finden, so dass*

$$\max(\hat{B}) \leq \epsilon \cdot \max(B), \tag{3.1}$$

$$SIZE(\hat{B}) \leq \epsilon \cdot SIZE(B), \tag{3.2}$$

und für alle $i = 1, \ldots, r$*:*

$$\begin{aligned} \min(B_i) &\geq \epsilon^t \max(B_i), \quad \textit{und} \\ \max(B_{i+1}) &< \epsilon \min(B_i). \end{aligned} \tag{3.3}$$

Beweis. Wir partitionieren die Menge B der Kisten in linearer Zeit in $u \leq |B|$ Gruppen $\bar{B}_i$, $1 \leq i \leq u$. Dazu ordnen wir die Kisten sukzessive in die Gruppen, wobei wir immer dann eine neue Gruppe eröffnen, wenn die Größe der aktuell betrachteten Kiste kleiner als das ϵ–fache der Größe der größten Kiste der aktuellen Gruppe ist. Die lineare Laufzeit folgt sofort aus der Tatsache, dass die Kisten nach nicht aufsteigender Größe vorsortiert sind. Die Gruppen der Kisten $\bar{B}_i$ haben daher die folgenden Eigenschaften:

$$\min(\bar{B}_i) \geq \epsilon \max(\bar{B}_i) \qquad \text{und} \qquad \min(\bar{B}_i) > \max(\bar{B}_{i+1}) \qquad \text{für alle } i \geq 1.$$

Die folgende Funktion erzeugt die gewünschte Partition in linearer Laufzeit.

Funktion 3.2.1 Partition

$s := 1$; $j := 2$; $\hat{B} := \emptyset$; $i := 1$; $B_1 := \emptyset$;
while $j \leq u$ **do begin**
  if $\max(\bar{B}_j) \geq \epsilon \cdot \min(\bar{B}_{j-1})$ **then do begin**
    if $j - s = \lfloor \frac{t}{4} \rfloor - 1$ **then do begin**
      Wähle $s \leq i^* \leq j$, so dass $SIZE(\bar{B}_{i^*}) = \min_{s \leq i \leq j} SIZE(\bar{B}_i)$;
      if $i^* > s$ **then** $B_i := B_i \cup \bar{B}_s \cup \cdots \cup \bar{B}_{i^*-1}$;
      if $B_i \neq \emptyset$ **then** $i++$; $B_i := \emptyset$;
      if $i^* = 1$ **then** $B_i := \bar{B}_{i^*}$ **else** $\hat{B} := \hat{B} \cup \bar{B}_{i^*}$;
      if $i^* < j$ **then** $B_i := B_i \cup \bar{B}_{i^*+1} \cup \cdots \cup \bar{B}_j$;
      $s := j + 1$;
      end
    end
  else do begin
    if $s < j$ **then** $B_i := B_i \cup \bar{B}_s \cup \cdots \cup \bar{B}_{j-1}$;
    if $B_i \neq \emptyset$ **then** $i++$; $B_i := \emptyset$;
    $s := j$;
    end
  $j++$;
  end
$B_i := B_i \cup \bar{B}_s \cup \cdots \cup \bar{B}_u$;

Informal beschrieben betrachtet diese Funktion die Gruppen der Kisten von $\bar{B}_1$ bis $\bar{B}_u$. Jedesmal, wenn sie eine Folge von $\lfloor \frac{t}{4} \rfloor$ hintereinander liegenden Gruppen

von Kisten durchlaufen hat, wählt sie die Gruppe mit kleinster gesamter Kistengröße und ordnet sie der Menge $\hat{B}$ der "unwichtigen" Kisten zu. Man kann leicht einsehen, dass die Zerlegung $\hat{B} \cup B_1 \cup \cdots \cup B_r$ die gewünschten Eigenschaften hat. Die Klassen B_j setzten sich aus höchstens $2\lfloor \frac{t}{4} \rfloor \leq \frac{t}{2}$ aufeinanderfolgenden Gruppen $\bar{B}_i$ zusammen. Nach Konstruktion liegen die minimalen Kistengrößen von zwei aufeinanderfolgender Kistengruppen innerhalb eines Faktors ϵ^2. Daraus folgt (3.3). Eigenschaft (3.1) resultiert aus der besonderen Behandlung des Falls $i^* = 1$. Zum Nachweis von (3.2) sei i^* ein gewählter Index. Dann gilt das folgende:

$$SIZE(\bar{B}_{i^*}) \leq \frac{1}{j-s+1} \sum_{i=s}^{j} SIZE(\bar{B}_i) = \frac{1}{\lfloor \frac{t}{4} \rfloor} \sum_{i=s}^{j} SIZE(\bar{B}_i).$$

Summation über alle Gruppen von Kisten $\bar{B}_{i^*}$ ergibt (man beachte, dass sich die Kisten auf der rechten Seite für verschiedene Gruppen von Kisten $\bar{B}_{i^*}$ unterscheiden)

$$\sum_{j \in \hat{B}} s_j = \sum_{i=1}^{r} SIZE(\bar{B}_{i^*}) \leq \frac{1}{\lfloor \frac{t}{4} \rfloor} SIZE(B) \leq \epsilon \cdot SIZE(B).$$

□

Lemma 3.4 induziert die folgende Familie von beschränkten Bin–Packing Problemen mit variabler Kistengröße (engl. *restricted variable size bin packing*), die wir im Folgenden $RVBP_i$, $i = 1, \ldots, r$ nennen wollen:
Problem $RVBP_i$ besteht aus

allen Kisten, die in B_i enthalten sind,

der Menge der Jobs $J_i := \{j \in J \mid \epsilon \min(B_i) \leq p_j \leq \max(B_i)\}$.

Unser Ziel ist, für jedes $RVBP_i$ die Gesamtgröße der Jobs zu minimieren, die wir nicht in die Kisten des gegebenen Problems zu packen vermögen. Es ist dabei eine wichtige Beobachtung, dass sich die Größe des kleinsten Jobs resp. der kleinsten Kiste und des größten Jobs resp. der größten Kiste für jedes Packproblem $RVBP_i$ um höchstens einen Faktor ϵ^{t+1} resp. ϵ^t unterscheiden. Wie wir im nächsten Abschnitt noch sehen werden, können für derartige Packprobleme unter Verwendung der ganzzahligen linearen Programmierung gute Approximationslösungen erzielt werden. Wir werden diese Probleme *getrennt* lösen und im Folgeschritt die einzelnen Lösungen der Probleme $RVBP_i$ mit einem geeigneten Greedy–Verfahren zusammensetzten (siehe Abschnitt 3.2.3). Dieses Greedy–Verfahren packt die bei den Problemen $RVBP_i$ übriggebliebenen Jobs. Dabei werden die Kisten leicht überfüllt werden. Wir werden jedoch zeigen können, dass die Überladung einer Kiste B_i durch das ϵ–fache der Kistengröße beschränkt ist. Für die Kisten $\hat{B}$, die wir mittels Lemma 3.4 erhielten, können wir dagegen nur garantieren, dass die Überladung nicht größer als die Kistengröße ist. Ungleichung (3.2) aus Lemma 3.4 impliziert jedoch, dass man sich eine derartig große Überladung von $\hat{B}$ erlauben kann, da $SIZE(\hat{B})$ nur einen kleinen Anteil der gesamten Kistengrößen

ausmacht. Wir können daher zeigen, dass die korrespondierenden überstehenden Jobs später noch umgeplant werden können, ohne dass sich der resultierende Makespan zu sehr ändert.

3.2.2 Baustein 2: Approximation von *RVBP*

Wir betrachten zwei Versionen des beschränkten Bin–Packing Problems ($RVBP$). Die erste Version des $RVBP$ formulieren wir in der Form $RVBP[\delta, \rho, u, v]$. Die Parameter δ, ρ, u, v besagen, dass alle Kisten bzw. Jobgrößen zum Intervall $[\delta, 1]$ bzw. dem Intervall $[\rho, 1]$ gehören müssen, wobei u und v die maximale Anzahl möglicher *verschiedener* Job- und Kistengrößen beschreibt. Die zweite Version des Problems — $RVBP[\delta, \rho]$ mit Parametern δ und ρ genannt — relaxiert die Einschränkungen an die Eingabeinstanz und erlaubt für Kisten und Jobs eine beliebige Anzahl verschiedener Größen. Die minimale Größe der Kisten und Jobs ist jedoch nach unten durch δ und ρ beschränkt. Daher liegen alle Jobgrößen im Intervall $[\delta, 1]$ und alle Bingrößen im Intervall $[\rho, 1]$. Wir betrachten die Probleme $RVBP$ als *Optimierungsprobleme*. Unser Ziel ist also, eine Lösung zu finden, die die summierte Größe der nicht gepackten Jobs minimiert. In den nächsten beiden Abschnitten beweisen wir, dass das Problem $RVBP[\delta, \rho]$ in linearer Zeit durch exaktes Lösen eines zugehörigen Problems $RVBP[\delta, \rho, u, v]$ gut approximiert werden kann.
Zuvor möchten wir aber noch die Bedeutung dieser Probleme klären. Wir benötigen einen Ansatz, um das Problem $RVBP[\delta, \rho]$ zu lösen, da die am Ende des letzten Abschnitts eingeführt Probleme $RVBP_i$ nichts anderes, als skalierte Instanzen eines einzigen Problems $RVBP[\epsilon^{t+1}, \epsilon^t]$ mit Skalierungsfaktor $1/\max(B_i)$ sind.

3.2.2.1 $RVBP[\delta, \rho, u, v]$

Die Eingabeinstanz $I = I(J, B)$ des Problems $RVBP[\delta, \rho, u, v]$ kann durch zwei Multimengen $J = \{n_1 : p_1, n_2 : p_2, \ldots, n_u : p_u\}$ und $B = \{k_1 : s_1, k_2 : s_2, \ldots, k_v : s_v\}$ beschrieben werden, so dass $1 \geq s_1 > s_2 > \cdots > s_v \geq \rho$, $1 \geq s_1 \geq p_1 > p_2 > \cdots > p_u \geq \delta$, $n = \sum_{i=1}^{u} n_i$ und $k = \sum_{i=1}^{v} k_v$, wobei n die gesamte Anzahl der Jobs und k die Zahl der Kisten sei und wo n_i und k_i jeweils die Anzahl der Jobs resp. Kisten der Größe p_i und s_i seien. Wir definieren eine *Konfiguration* als eine gepackte Kiste einer bestimmten Größe mit einer bestimmten Menge der Jobs, so dass die Summe der Jobgrößen die Größe der Kisten nicht übersteigt. Eine derartige Konfiguration kann durch einen u–Vektor $(l_1, \ldots, l_u)$ von nicht negativen Zahlen dargestellt werden, so dass l_i die Zahl der in der Kiste gepackten Jobs der Größe p_i wiedergibt. Da wir nur Jobs packen, deren Größe mindestens δ beträgt, können in einer Kiste höchstens $\lfloor \frac{1}{\delta} \rfloor$ Jobs liegen. Dies begrenzt die Anzahl der möglichen Konfigurationen auf eine Konstante $q = q(\delta, u) \leq (u+1)^{\lfloor \frac{1}{\delta} \rfloor}$ für jede der v möglichen Kistengrößen.

Betrachte nun eine zulässige Lösung x für eine Instanz I von $RVBP[\delta, \rho, u, v]$. Dann kann x durch einen Vektor $x = (x_1, \ldots, x_{q \cdot v})$ beschrieben werden, wobei x_j die Zahl der Kisten angibt, die nach Konfiguration j gepackt wurden.
Wir definieren

- $\mathcal{T}_i$, als die Menge der zulässigen Konfigurationen für Kisten der Größe s_i.
- a_{lj}, als die Zahl der Jobs der Größe p_l, die in Konfiguration j auftreten.

Dann kann $RVBP[\delta, \rho, u, v]$ als das folgende ganzzahlige lineare Programm geschrieben werden.

$$\begin{aligned} \text{minimiere} \quad & \sum_{l=1}^{u} (n_l - \sum_j a_{lj} x_j) p_l \\ s.\ t. \quad & \\ \forall 1 \leq l \leq u \quad & \sum_j a_{lj} x_j \leq n_l \\ \forall 1 \leq i \leq v \quad & \sum_{j \in \mathcal{T}_i} x_j \leq k_i \\ & x_j \ \text{ ganzzahlig} \end{aligned} \tag{3.4}$$

Sei $OPT(I)$ der Wert einer optimalen Lösung für das Packproblem, d. h. die minimal mögliche Summe über die Größen der nicht gepackten Jobs. Da wir δ, ρ und u als Konstanten vorausgesetzt haben, können wir obiges lineares Programm in linearer Zeit in v lösen.
Nach Lenstra [Len81] kann das lineare Programm in einer polynomiell von seiner Größe abhängenden Zeit gelöst werden, vorausgesetzt, dass u und v Konstanten sind, also in $Poly(\log n, \log k, \frac{1}{\delta}, \frac{1}{\rho})$ bei festen δ und ρ.

3.2.2.2 $RVBP[\delta, \rho]$

Das zweite von uns betrachtete Problem ist $RVBP[\delta, \rho]$ welches ähnlich wie $RVBP[\delta, \rho, u, v]$ definiert ist, außer dass die Zahl der verschiedenen Kisten- und Jobgrößen nicht länger konstant zu sein braucht. Kisten- und Jobgrößen sind aber immer noch auf Intervalle $[\delta, 1]$ für Jobs und $[\rho, 1]$ für Kisten beschränkt. Wir nehmen außerdem wieder an, dass die Jobs nach nicht aufsteigender Größe sortiert sind. Wir werden nun zeigen, wie $RVBP[\delta, \rho, u, v]$ verwendet werden kann, um $RVBP[\delta, \rho]$ zu approximieren. Die Idee unserer Reduktion des Problems $RVBP[\delta, \rho]$ auf $RVBP[\delta, \rho, u, v]$ ist ein Gruppierungsansatz, der bereits in [dlVL81] vorgestellt wurde. Wir erweitern diese Technik, indem wir das Verfahren simultan auf Jobs *und* Kisten anwenden.
Eine Instanz des $RVBP[\delta, \rho, u, v]$ erhält man aus einer Instanz $I = (J, B)$ des $RVBP[\delta, \rho]$ durch die folgende Konstruktion: Als erstes gruppieren wir die Menge der Jobs $J = \{1, \ldots, n\}$ in Gruppen $G_j = \{(j-1) \cdot K_1 + 1, \ldots, j \cdot K_1\}$ für

$j = 1, \ldots, u$ und $G_{u+1} = \{u \cdot K_1 + 1, \ldots, n\}$, wobei jede, außer der letzten Gruppe, aus einer Zahl von K_1 Jobs besteht. K_1 ist dabei eine nicht negative Zahl, die wir später spezifizieren werden. Wir definieren die beiden Funktionen *Aufrunden* und *Abrunden*. Die erste Funktion rundet die Jobs einer jeden Gruppe auf ihr größtes Element. *Abrunden* rundet die Jobs der Gruppe G_j auf das größte Element der Gruppe G_{j+1}. Die Jobs in der letzten Gruppe werden auf die Größe des kleinsten Elementes abgerundet. Wir nennen die durch *Aufrunden* entstandene Gruppe $H = H_1 H_2 \ldots H_{u+1}$ und die durch *Abrunden* entstandene Gruppe $F = F_1 F_2 \ldots F_{u+1}$. Die zentrale Idee dieser Definition ist, dass die Listen F_j und H_j $(j = 1, \ldots u + 1)$ im wesentlichen übereinstimmen, außer der ersten und der letzten Gruppe der Jobs. Genauer gilt $F_j = H_{j+1}$ für $j < u$ und $H_{u+1} \subset F_u$, so dass eine Zuordnung der Jobs zu den Kisten für $\bigcup_{j \leq u} F_j$ eine Zuordnung für $\bigcup_{j=2}^{u+1} H_j$ induziert. Diese Überlegung motiviert den folgenden Algorithmus: Finde eine Zuordnung für die Liste F, wobei Gruppe F_{u+1} nicht betrachtet wird. Die Lösung ist mindestens so gut, wie die optimale Lösung, da die Jobs in ihrer Größe abgerundet wurden. Die Lösung wird dann aber nicht als eine Zuordnung für Gruppe F betrachtet, sondern als Zuordnung der Jobs der Gruppe H, wobei Gruppe H_1 übrigbleibt. Die summierte Größe der Jobs aus H_1 beträgt höchstens K_1. Schließlich erzeugen wir eine zulässige Zuordnung der Jobs der Originalinstanz von $RVBP[\delta, \rho]$ zu den einzelnen Kisten, indem wir einfach die Jobs auf ihre Originalgrößen abrunden.
Es gilt:

$$OPT(I_H) \leq OPT(I_F) + K_1 \leq OPT(I) + K_1,$$

wobei I_H und I_F die Eingabeinstanzen für die Joblisten H resp. F bezeichnet.
Auf eine ähnliche Weise verfahren wir simultan mit den Kisten. Wir gruppieren die nicht aufsteigend sortierte Folge der Kisten in einzelne Gruppen, wobei sich jede Gruppe (außer der letzten Gruppe) aus K_2 Kisten zusammensetzt. Nun definieren wir *Aufrunden* als die Funktion, welche die Kisten aus der j–ten Gruppe auf die kleinste Kiste der Gruppe $j - 1$ aufrundet und die der ersten Gruppe auf das größte Kiste. Die Funktion *Abrunden* rundet alle Kisten einer Gruppe auf die kleinste Kiste in dieser Gruppe. Das Lösen des Problems $RVBP$ auf dieser durch *Aufrunden* modifizierten Eingabe und das anschließende Abrunden der Kisten auf ihre Originalgröße ergibt wiederum eine zulässige Lösung, bei der die Summe der Größen über alle nicht gepackten Jobs durch $OPT(I) + K_2$ beschränkt ist. Durch simultanes Durchführen beider Gruppierungsansätze, die der Kisten und die der Jobs, in einem einzelnen Schritt, erhält man die gewünschte Reduktion von I zu einer Instanz von $RVBP[\delta, \rho, u, v]$, wobei wir $u = \lfloor \frac{|J|}{K_1} \rfloor$ und $v = \lfloor \frac{|B|}{K_2} \rfloor$ definieren. Es verbleibt, die Werte für K_1 und K_2 zu spezifizieren. Wir setzten

$$K_1 := \lceil |J|\mu \rceil \qquad \text{und} \qquad K_2 := \lceil |B|\mu \rceil$$

für beliebige aber feste Konstanten $1 \geq \mu > 0$ und erhalten dadurch für u und v jeweils Konstanten. Dann kann aber, wie wir im letzten Abschnitt festgestellt

hatten, das korrespondierende Problem $RVBP[\delta, \rho, u, v]$ in linearer Zeit gelöst werden. Wir erhalten das folgende Lemma:

Lemma 3.5
Sei $1 \geq \mu > 0$ beliebig aber fest gewählt und seien $K_1 := \lceil |J|\mu \rceil$ und $K_2 := \lceil |B|\mu \rceil$. Dann können wir für eine beliebige Instanz $I = I(J, B)$ des $RVBP[\delta, \rho]$ in linearer Zeit eine Lösung finden, so dass die gesamte Größe der nicht gepackten Jobs maximal

$$OPT(I) + K_1 + K_2 \leq OPT(I) + \mu(|J| + |B|)) + 2$$

beträgt. ($OPT(I)$ ist hierbei die Gesamtgröße der Jobs, die in einer Optimallösung für die Instanz $I = I(J, B)$ nicht gepackt werden können.)

3.2.2.3 Lösen der Teilprobleme $RVBP_i$

Wie bereits in den vorherigen Abschnitten skizziert, können die am Ende des Kapitels 3.2.1 definierten Teilprobleme $RVBP_i$ als Instanzen des speziellen Problems $RVBP[\epsilon^{t+1}, \epsilon^t]$ interpretiert werden, indem alle Kisten und Jobs mit $1/\max(B_i)$ skaliert werden. Für diese skalierten Instanzen bleibt Lemma 3.5 weiterhin gültig. Daraus folgt, dass die Summe der Größen von nicht gepackten Jobs für ein Problem $RVBP_i$ durch $OPT(RVBP_i) + (\mu(|J_i| + |B_i|) + 2)\max(B_i)$ beschränkt werden kann. Wir werden später noch sehen, dass eine geeignete Wahl von μ garantieren wird, dass die summierte Größe der nicht gepackten Jobs für eine $(1 + \epsilon)$–Approximation des Makespan klein genug ist.
Zu beachten ist als Spezialfall noch das Problem $RVBP_1$, für das wir zwei Fälle unterscheiden müssen. Falls alle Kisten kleiner als ϵ sind, erzeugen wir wie bei allen anderen Problemen $RVBP_i$ eine Approximationslösung. Im anderen Fall müssen wir anders vorgehen, denn sonst könnten nicht gepackte Jobs eine Größe von bis zu C_{Opt} haben und dann würde ihr Effekt auf den Makespan im Gegensatz zu allen anderen Problemen zu groß sein. Daher müssen sie vorsichtiger gepackt werden. Auf der anderen Seite können wir die Existenz einer Lösung direkt ausschließen, falls die Zahl der Jobs mit einer Größe von mindestens ϵ^{t+2} den Wert $\frac{m}{\epsilon^{t+2}}$ übersteigt, da die Gesamtgröße aller Kisten in VBP durch m beschränkt ist (alle Kisten und Jobs sind mit $1/C$ skaliert). Für $RVBP_1$ betrachten wir die Menge der Kisten $\tilde{B}_1 := B_1 \cup \{j \in \hat{B} \mid s_j \geq \min(J_1)\}$. Insbesondere betrachten wir $RVBP_1$ als ein Problem von Typ $RVBP[\epsilon^{t+2}, \epsilon^{t+2}, \frac{m}{\epsilon^{t+2}}, |\tilde{B}_1|]$, das in konstanter Zeit optimal gelöst werden kann. Dazu ist natürlich entscheidend, dass $|\tilde{B}_1|$ durch $|\tilde{B}_1| \leq \frac{m}{\epsilon^{t+2}}$ beschränkt ist. Falls irgendwelche Jobs nicht gepackt werden können, ist die Existenz einer zulässigen Lösung für VBP ausgeschlossen.

3.2.3 Baustein 3: Das Greedy–Verfahren

Wir betrachten nun als letzten Baustein unseres Algorithmus ein Greedy–Verfahren, das zum Packen der verbleibenden Jobs verwendet wird. Es gibt zwei

Arten von Jobs, die noch zu packen sind. Diese sind

$\hat{J}_i \subset J_i$, und

$J^* = J - \bigcup_{i=1}^{r} J_i$,

wobei $\hat{J}_i$ die Menge der in der Lösung des Problems $RVBP_i$ nicht gepackten Jobs bezeichne. Die zweite Menge J^* ist die Vereinigung aller Jobs, die noch in keinem der Teilprobleme $RVBP_i$ betrachtet wurden, da sie mit ihrer Größe "zwischen" zwei aufeinanderfolgenden Teilproblemen $RVBP_i$ und $RVBP_{i+1}$ angeordnet sind. Wir partitionieren diese Menge in Teilmengen $J^* := \bigcup_{i=1}^{r} J_i^*$, wobei $J_i^* := \{j \in J \mid \epsilon \min(B_i) > p_j > \max(B_{i+1})\}$ (mit der Konvention $\max(B_{r+1}) = 0$).
Um eine Intuition für unser Greedy–Verfahren vermitteln zu können, betrachten wir erneut Lemma 3.4. Es ist klar, dass wir die Kistenkapazitäten bei Verwendung eines Greedy–Verfahrens nicht voll ausnützen können, es sei denn, wir erlauben, dass alle Kisten (einschließlich der geschlossenen) durch je einen Job überfüllt werden dürfen. Lemma 3.4 impliziert, dass die gesamte Größe der Überfüllung, bei der Kisten B_i^* bis zu ihrer eigenen Größe (siehe Abschnitt 3.4) und Kisten B_i bis zu einem ϵ–fachen ihrer Größte überfüllt werden, im Vergleich zur gesamten einzuplanenden Prozesszeit klein ist. Wir müssen jedoch garantieren, dass die Kisten nicht noch weiter überfüllt werden. Nachdem wir ein Problem $RVBP_i$ $(i = 1, \ldots, r)$ gelöst haben, packen wir die in der Lösung von $RVBP_i$ übriggebliebenen Jobs $p \in \hat{J}_i$ in diejenigen Kisten, deren Größe $\max(B_i)$ übersteigt. Dies schließt auch die Kisten aus $\hat{B}$ ein. Die derart betrachteten Kisten bezeichnen wir als $B_i^<$, wobei $B_1^<$ leer ist. Als zweites packen wir mit einem Greedy–Verfahren für jedes i die Jobs aus J_i^*. Wir erlauben wiederum, dass die Kisten durch einen Job j überladen werden dürfen, vorausgesetzt die Größe der Jobs j übersteigt nicht die Größe der Kisten. Dann fahren wir (für $i < r$) mit dem Problem $RVBP_{i+1}$ fort. Lemma 3.4 besagt, dass die Größen der Kisten der Probleme $RVBP_i$ und $RVBP_{i+1}$ durch mindestens das ϵ–fache von $\min(B_i)$ voneinander getrennt sind. Daher gilt, dass jeder unter dem Greedy–Verfahren zu packende Job $i \in J$ nur Kisten $j \in B - \hat{B}$ mit $p_i \leq \epsilon \cdot s_j$ betrachtet. Zusätzlich gilt, dass die von i betrachteten Kisten $\hat{B}$ mindestens so groß sind, wie p_i. Wenn i also eine Kiste überladen sollte, so geschieht dies stets im Rahmen des zulässigen Betrages einer Überfüllung. Am Ende des Algorithmus entfernen wir alle Jobs, die eine geschlossene Kiste überladen und packen sie mit einem weiteren Greedy–Algorithmus (diesmal *List*) in die offenen Kisten. Dieser letzte Schritt garantiert die Zulässigkeit des Schedules, da alle potentiellen Überschneidungen mit fixierten Jobs aufgelöst werden.

3.3 Der vollständige Algorithmus

Wir haben nun alle Bausteine des Algorithmus zum Approximieren des Scheduling–Problems mit fixierten Jobs kennengelernt. Sei I eine Instanz des Scheduling–Problems mit fixierten Jobs, wobei I gegeben sei durch

- die Menge J der n Jobs $1, \ldots, n$ mit ganzzahligen Prozesszeiten $p_1, \ldots, p_n$ und
- eine Teilmenge $\{i_1, \ldots, i_k\}$ von k fixierten Jobs. Die Fixierung ist durch eine Startzeit und die zugeordnete Maschine spezifiziert.

Die Zahl m der Maschinen und die zu erreichende Approximationsgüte sind dagegen nicht Teil der Eingabe. Die Zeit, zu der der letzte fixierte Job endet, sei C_F. Wie bereits am Anfang unserer Überlegungen dargestellt, ist das Grundverfahren des Algorithmus eine binäre Suche nach dem optimalen Makespan C_{Opt}. Eine geeignete untere Schranke für C_{opt} ist $\max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j, C_F\}$. Man kann sich außerdem leicht überlegen, dass jeder *List*–Algorithmus einen Makespan von höchstens $3\max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j, C_F\}$ erzeugt, indem einfach jeder Job nach Beendigung des letzten fixierten Jobs eingeplant wird. Es gibt einfache Kriterien, die nachweisen, dass ein Kandidat C für den Makespan kleiner als C_{Opt} ist, resp. dass das durch C definierte Packproblem $VBP[C]$ keine Lösung hat. Für eine genaue Analyse dieser Argumente verweisen wir auf den später folgenden Beweis von Lemma 3.8.

Algorithmus 3.3.1 Algorithmus A_ϵ für das Scheduling–Problem mit fixierten Jobs

Input: Instanz I bestehend aus fixierten und nicht fixierten Jobs.

Output: Ein zulässiger Schedule für I.

Sortiere Jobs nach nicht aufsteigenden Größen.
$ub := 3\max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j, C_F\}$; $lb := \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j, C_F\}$;
while $(ub - lb \geq 1)$ **do begin**
  $C := lb + \frac{ub-lb}{2}$;
  Erzeuge eine Instanz des Problem VBP mit Kisten B und Jobs J, indem die Maschinen linear durchlaufen, dabei die Kisten identifiziert und schließlich Kisten und Jobs mit $\frac{1}{C}$ skaliert werden.
  Rufe die Funktion zum Lösen des Problems VBP auf. Falls eine Lösung gefunden wurde, setze $ub := C$. Andernfalls (*Failure*) setze $lb := C$.
end
Falls bisher keine zulässige Lösung gefunden werden konnte, setze $C := ub$ und löse wiederum das Problem VBP für die mit $\frac{1}{C}$ skalierten Kisten und Jobs.
Interpretiere die erzeugte Lösung als Schedule und gebe diesen aus.

Die Funktion zum Packen der Jobs in die Kisten bildet den Hauptteil des Algorithmus.

Funktion 3.3.1 Funktion zum Lösen des Problems VBP

Input: Menge J von Jobs und Menge B von Kisten.

Output: Eine Zuordnung von Jobs zu Kisten (bei der offene Kisten leicht überfüllt werden können, geschlossenen Kisten aber nicht) oder *Failure*

Bestimme mit dem in Lemma 3.4 gegebenen Algorithmus die Kisten $B_1, \ldots, B_r$ und erzeuge die Probleme $RVBP_i$ auf der am Ende des Kapitels 3.2.1 dargestellten Weise.
For $i = 1, \ldots, r$ sei (mit der Konvention, dass $\max(B_{r+1}) = 0$)

$$\begin{aligned} B_i^* &:= \{j \in B \mid \min(B_i) > s_j > \max(B_{i+1})\} \text{ und} \\ J_i^* &:= \{j \in J \mid \epsilon \min(B_i) > p_j > \max(B_{i+1})\}; \end{aligned}$$

if $\max(B_1) \geq \epsilon$ **do begin**

if Jobs $j \in J$ existieren mit $p_j > \max(B_1)$, stoppe mit *Failure*.

if $|J_1| \geq \frac{m}{\epsilon^{t+2}}$, stoppe mit *Failure* (keine zulässige Lösung existiert) **else** löse $RVBP_1$ optimal (siehe Kapitel 3.2.2.1). Dabei betrachten wir alle Kisten $B_1 \cup \{j \in \hat{B} \mid s_j \geq \min(J_1)\}$ und alle Jobs. Stoppe mit *Failure* falls Jobs nicht gepackt werden können.

else

Berechne eine Approximationslösung für $RVBP_1$ (siehe Kapitel 3.2.2.1 resp. Kapitel 3.2.2.3), wobei wir $\mu := \epsilon^{t+2}$ setzen.

Sei $\bar{J}_1$ die Menge der Jobs, die noch nicht gepackt wurden.

if $SIZE(\bar{J}_1) > (\mu \cdot (|J_1| + |B_1|) + 2) \max(B_1) + SIZE(B_1^*)$ **then** stoppe mit *Failure* (keine zulässige Lösung);

end

Packe alle Jobs $j \in J_1^*$ mit einem Greedy–Algorithmus in die Kisten $B_1 \cup B_1^*$. Der Greedy–Algorithmus darf einen Job in eine Kiste setzten, falls die Kiste noch nicht vollständig gefüllt ist (sogar, wenn die Kiste nach dem Einsetzen des Jobs überfüllt sein sollte) *und* der Job nicht größer als die Kiste ist. Sowohl Kisten als auch Jobs werden dabei nach nicht aufsteigender Größe betrachtet. Stoppe mit *Failure*, wenn Jobs nicht gepackt werden können.

for $i = 2$ **to** r **do begin**

Berechne eine Approximationslösung für $RVBP_i$ (siehe Kapitel 3.2.2.3), wobei wir $\mu := \epsilon^{t+2}$ setzen.

Sei $\hat{J}_i$ die Menge der nicht gepackten Jobs und sei

$$B_i^< := B_1 \cup B_1^* \cup \cdots \cup B_{i-1} \cup B_{i-1}^*.$$

Packe die Jobs $j \in \hat{J}_i$ mit einem Greedy–Algorithmus in die Kisten $B_i^<$. Wiederum erlauben wir, dass Jobs in nicht vollständig gefüllte Kisten gesetzt werden können, auch wenn danach die Kapazität des Kisten überschritten wird. Sei $\bar{J}_i$ die Menge aller Jobs, die nicht gepackt wurden.

if $SIZE(\bar{J}_i) > (\mu(|J_i| + |B_i|) + 2) \max(B_i) + SIZE(B_i^*)$, stoppe mit *Failure*;

Packe alle Jobs $p \in J_i^*$ mit einem Greedy–Algorithmus in die Kisten $B_i^< \cup B_i \cup B_i^*$. Der Greedy–Algorithmus darf einen Job in eine Kistesetzen, wenn es noch nicht vollständig gefüllt ist, auch wenn die Kiste danach überfüllt wäre, vorausgesetzt, der Job ist nicht größer als die Kiste. Sowohl Kisten als auch Jobs werden dabei nach nicht aufsteigender Größe betrachtet. Stoppe mit *Failure*, wenn Jobs nicht gepackt werden können.

end

Für jede überfüllte Kiste entferne den *letzten* Job aus der Kiste. Sei J_R die Menge aller entfernten Jobs.

Sei $\bar{J} := \bar{J}_1 \cup \cdots \cup \bar{J}_r \cup J_R$.

Packe alle Jobs $\bar{J}$ mittels eines Greedy–Algorithmus in die offenen Kisten, wobei jeder Job derjenigen Kiste zugeordnet wird, die momentan am wenigsten gefüllt ist.

end

3.4 Analyse

Wir können die Ergebnisse der letzten Kapitel zu dem folgenden Theorem zusammenfassen:

Theorem 3.6
Es gibt eine Variante von Algorithmus A_ϵ mit Laufzeit $\mathcal{O}(n \log n)$, die einen zulässigen Schedule mit Approximationsgüte $1 + 10\epsilon$ für eine beliebige Instanz I des Scheduling–Problems mit fixierten Jobs auf einer konstanten Maschinenzahl findet.

Wir beweisen das Theorem in zwei Schritten. Als erstes zeigen wir, dass der Algorithmus stets einen zulässigen Schedule mit der gewünschten Approximationsgüte erzeugt. Danach modifizieren wir Algorithmus A_ϵ und verifizieren die Laufzeit $\mathcal{O}(n \log n)$ dieser Algorithmusvariante.

Lemma 3.7
Für alle $C \geq C_{opt}$ berechnet die Funktion zum Lösen des VBP eine Zuordnung der Jobs zu den Kisten, bei der die geschlossenen Kisten nicht über ihre Kapazität hinaus gefüllt werden.

Beweis. Wir zeigen, dass der Algorithmus für $C \geq C_{opt}$ nicht stoppen wird, ohne eine Lösung gefunden zu haben. Es genügt, die i–te Iteration der **for**-Schleife in der Funktion zum Lösen des Problems VBP für beliebiges i zu betrachten. Die Funktion zum Lösen des Problems VBP gibt nur dann *Failure* zurück, wenn eine der beiden Mengen von Jobs $\hat{J}_i$ oder J_i^* nicht vollständig gepackt werden kann. Wir betrachten als erstes $\hat{J}_i$, also genau die Jobs, welche in der Lösung von $RVBP_i$ nicht gepackt wurden. Wir betrachten einen beliebigen optimalen Schedule σ_{opt} mit Makespan C_{opt}. Sei $\hat{J}_i^\sigma$ die Menge der Jobs $j \in J_i$, die im optimalen Schedule σ_{opt} nicht in die Kisten B_i gepackt wurden. Innerhalb der Lösung des $RVBP_i$ werden die Jobs in J_i in die Kisten B_i gepackt, wobei wir $RVBP_i$ (nahezu) optimal (bzw. optimal für $i = 1$ und $\max(B_1) \geq \epsilon$) lösen. Der Fehler ist auf jeden Fall durch $(\mu(|J_i| + |B_i|) + 2) \cdot \max(B_i)$ (Lemma 3.5) beschränkt. Da der Wert der optimalen Lösung für $RVBP_i$ maximal $SIZE(\hat{J}_i^\sigma)$ beträgt, erhalten wir:

$$SIZE(J_i - \hat{J}_i) \geq SIZE(J_i - \hat{J}_i^\sigma) - (\mu(|J_i| + |B_i|) + 2) \cdot \max(B_i). \tag{3.5}$$

Nun passen Jobs J_i nur in die Kisten $B_i^< \cup B_i \cup B_i^*$ aber nicht in kleinere Kisten. Daher müssen die Kisten $\bigcup_{j=i+1}^{r}(B_j \cup B_j^*)$ für Jobs J_i selbst in einem optimalen Schedule nicht betrachtet werden. Wegen $C \geq C_{opt}$ folgt

$$SIZE(\hat{J}_i^\sigma) \leq fcap_i + SIZE(B_i^*), \tag{3.6}$$

wobei $fcap_i$ die verbleibende Gesamtkapazität aller Kisten $B_i^<$ vor dem Start des Greedy–Algorithmus von A_ϵ sei ($B_i^<$ enthält zu dieser Zeit nur Jobs, die auch im

optimalen Schedule σ_{Opt} in diesen Kisten liegen müssen). Daher gilt

$$SIZE(J_i) \leq fcap_i + SIZE(B_i^*) + SIZE(J_i - \hat{J}_i^\sigma). \tag{3.7}$$

Die Ungleichungen (3.5) und (3.7) ergeben zusammen

$$\begin{aligned} SIZE(\hat{J}_i) &= SIZE(J_i) - SIZE(J_i - \hat{J}_i) \\ &\leq fcap_i + SIZE(B_i^*) + (\mu(|J_i| + |B_i|) + 2) \cdot \max(B_i). \end{aligned} \tag{3.8}$$

Da der Greedy–Algorithmus Kisten überfüllen darf, können nur dann Jobs übrigbleiben, wenn alle Kisten $B_i^<$ vollständig gefüllt sind. Das bedeutet aber, dass die Gesamtgröße aller nicht gepackten Jobs maximal $SIZE(B_i^*)+(\mu(|J_i|+|B_i|)+2)\cdot \max(B_i)$ beträgt. Dies garantiert, dass der Algorithmus bei der entsprechenden *if*–Bedingung für $C \geq C_{opt}$ nicht stoppen wird. Betrachte nun die Jobs aus J_i^*: Auch diese können ausschließlich in die Kisten $B_i^< \cup B_i \cup B_i^*$ gesetzt werden. Es gilt wiederum, dass alle Jobs, die in $B_i^< \cup B_i \cup B_i^*$ gepackt wurden, bevor der Greedy–Algorithmus auf J_i^* angewendet wurde, auch in einer optimale Lösung in diese Kisten gepackt werden. Da wir zusätzlich die Jobs in sortierter Reihenfolge packen und die Kisten anhand der nicht aufsteigenden Größe packen, gilt eine ähnliche Eigenschaft während des gesamten Greedy–Algorithmus für alle Kisten B_i^*: Jedesmal, wenn wir einen Job i betrachten und j die größte nicht vollständig gefüllte Kiste sei, dann enthalten alle mindestens s_j großen Kisten nur Jobs, deren Größe mindestens p_j beträgt. Daher kann entweder i in Kiste j gepackt werden (wenn $p_i \leq s_j$), oder wir haben festgestellt, dass keine zulässige Lösung existieren kann. Letzteres kann aber für $C \geq C_{opt}$ nicht eintreten. Daher wird der Greedy–Algorithmus alle Jobs J_i^* stets erfolgreich packen.
Damit ist gezeigt, dass der Algorithmus für $C \geq C_{opt}$ nicht stoppt, ohne eine zulässige Zuordnung der Jobs zu den Kisten gefunden zu haben. Da der Algorithmus aber auch alle Jobs packt und da nach Konstruktion keine der Kisten überfüllt ist, ist damit die Aussage des Lemmas gezeigt. □

Lemma 3.8
Für $C \geq C_{opt}$ ist die Gesamtgröße der Jobs aus J_R beschränkt durch

$$SIZE(J_R) \leq 2\epsilon \cdot SIZE(B). \tag{3.9}$$

Beweis. Die Menge J_R besteht aus allen Jobs, die aus einer überfüllten Kiste entfernt wurden. Nach Konstruktion beträgt die maximale Größe eines Jobs j, der eine Kiste B_i überfüllt hat, gerade $\epsilon \min(B_i)$ und wenn er eine Kiste k aus B_i^* überfüllt hat, so beträgt seine maximale Größe s_k. Nach Definition von B_i^* gilt $\sum_{k \in \bigcup_{i=1}^r B_i^*} s_k = SIZE(\hat{B})$, wobei $\hat{B}$ wie in Lemma 3.4 definiert sei. Anwenden

von Lemma 3.4 ergibt:

$$\begin{aligned}\sum_{j\in J_R} p_j &\leq \sum_{i=1}^{r} \epsilon \cdot min(B_i)|B_i| + \sum_{i=1}^{r}\sum_{k\in B_i^*} s_k \\ &\leq \sum_{i=1}^{r}(\epsilon \cdot SIZE(B_i)) + SIZE(\hat{B}) \\ &\leq 2\epsilon \cdot SIZE(B).\end{aligned}$$

□

Zusammen mit den Schranken

$$\begin{aligned}SIZE(J) &\leq m, \text{ und} \\ SIZE(B) &\leq m\end{aligned}$$

(man beachte, dass Kisten und Jobs mit $1/C$ skaliert sind) ergibt sich das folgende Korollar:

Korollar 3.9
Für $C_{opt} \leq C$ und $\epsilon \leq \frac{1}{2}$ ist die Gesamtgröße aller Elemente aus $\bar{J}$ beschränkt durch

$$SIZE(\bar{J}) \leq \epsilon \cdot (5m + 4). \tag{3.10}$$

Beweis. Die Minimalgröße eines Jobs resp. einer Kiste in einer Instanz des Problems $RVBP_i$ ist nicht kleiner als $\max(B_i)\epsilon^{t+1}$. Daraus folgt $|B_i| \leq \frac{SIZE(B_i)}{\epsilon^{t+1}\max(B_i)}$ und $|J_i| \leq \frac{SIZE(J_i)}{\epsilon^{t+1}\max(B_i)}$. Daher gilt

$$\sum_{i=1}^{r} |B_i| \max(B_i) \leq \sum_{i=1}^{r} \frac{SIZE(B_i)}{\epsilon^{t+1}} \leq \frac{m}{\epsilon^{t+1}} \tag{3.11}$$

und

$$\sum_{i=1}^{r} |J_i| \max(B_i) \leq \sum_{i=1}^{r} \frac{SIZE(J_i)}{\epsilon^{t+1}} \leq \frac{m}{\epsilon^{t+1}}. \tag{3.12}$$

Nun ist $\bar{J}_1$ jedesmal die leere Menge, wenn wir $RVBP_1$ optimal gelöst haben, also wenn $\max(B_1) \geq \epsilon$. Mit der Notation $i_0 := 1$, falls $\max(B_1) < \epsilon$ und $i_0 := 2$

andernfalls, erhalten wir (wegen $\mu := \epsilon^{t+2}$)

$$\begin{aligned}
\sum_{j\in\bar{J}} p_j &= \sum_{i=i_0}^{r}\sum_{j\in\bar{J}_i} p_j + \sum_{j\in J_R} p_j \\
&\leq \sum_{i=i_0}^{r} [(\mu(|B_i| + |J_i|) + 2) \cdot \max(B_i) + SIZE(B_i^*)] + 2\epsilon \cdot SIZE(B) \\
&\leq 2\epsilon m + 2\epsilon \sum_{i=0}^{\infty} \epsilon^i + SIZE(\hat{B}) + 2\epsilon \cdot SIZE(B) \\
&\leq \epsilon \cdot (2m + 4 + 3m),
\end{aligned}$$

wobei wiederum $SIZE(\hat{B}) \leq \epsilon \cdot SIZE(B)$ und $SIZE(B) \leq m$ ausgenutzt wird. □

Mit Hilfe von Korollar 3.9 gelingt es uns, den durch A_ϵ erzeugten Makespan zu berechnen:

Korollar 3.10
Für $C \geq C_{opt}$ und $\epsilon \leq \frac{1}{2}$ findet der Algorithmus A_ϵ einen zulässigen Schedule mit einem Makespan

$$C_{A_\epsilon} \leq (1 + 9\epsilon) \cdot C.$$

Beweis. Zunächst werden alle Jobs $J \setminus \bar{J}$ mit einem Makespan von höchstens C eingeplant. Daher müssen wir nur noch die Jobs aus $\bar{J}$ betrachten. Würden wir diese Jobs in m ursprünglich leere Kisten mit einem Greedy–Algorithmus packen, der für jeden Job die am wenigsten gefüllte Kiste wählt, dann wäre jede Kiste für $m > 1$ höchstens bis

$$\frac{1}{m}\sum_{j\in\bar{J}} p_j + \max_{j\in\bar{J}} p_j \leq 5\epsilon + 4\frac{1}{m} \cdot \epsilon + \epsilon \leq 8\epsilon$$

gefüllt. Die erste Ungleichung folgt aus Korollar 3.9 und der Tatsache, dass nach Konstruktion alle Jobs in $\bar{J}$ eine Größe von höchstens ϵ haben.
Für den Fall $m = 1$ können wir direkt Korollar 3.9 anwenden und erhalten eine obere Schranke von

$$\sum_{j\in\bar{J}} p_j \leq 9\epsilon.$$

Wenn wir im Schedule noch die verbleibende freie Kapazität der offenen Kisten ausnutzen, so kann der resultierende Makespan nur kleiner werden. Daher erfüllt der Makespan des resultierenden Schedules

$$C_{A_\epsilon} \leq C + 9\epsilon \cdot C.$$

(Der zusätzliche Faktor C ergibt sich aus der Tatsache, dass wir alle Kisten und Jobs mit $1/C$ skaliert hatten.) □

Als letztes werden wir noch die Laufzeit von $\mathcal{O}(n \log n)$ des Algorithmus A_ϵ beweisen. Zunächst können die Jobs nach nicht aufsteigender Größe in $\mathcal{O}(n \log n)$ sortiert werden. Betrachte nun die Laufzeit zum Lösen des Problems VBP in einer Iteration der binären Suche.

- Wir benötigen Zeit $\mathcal{O}(n \log n)$, um die Kisten nach nicht aufsteigender Größe zu sortieren. Die Klassen B_i können, wie im Beweis von Lemma 3.4 beschrieben, in linearer Zeit ermittelt werden.
- Wir benötigen für jedes Problem $RVBP_i$, $1 \leq i \leq r$ lineare Laufzeit, um das ganzzahlige lineare Programm zu erzeugen und polylogarithmische Laufzeit zu seiner Lösung. Daher ist die Laufzeit für jedes $RVBP_i$ linear. Für den Fall, dass $RVBP_1$ durch vollständige Enumeration gelöst wird, benötigen wir wiederum lineare Laufzeit (siehe Kapitel 3.2.2.1 und3.2.2.3). Das Packen der Jobs mit dem Greedy–Algorithmus ist ebenfalls in linearer Laufzeit möglich, da der Algorithmus die Kisten einfach nacheinander auffüllt.
- Die Menge J_R wird in linearer Laufzeit erzeugt und das Packen aller Jobs aus $\bar{J}$ verbraucht größenordnungsmäßig die gleiche Zeit.

Dies zeigt, dass wir VBP in der Zeit $\mathcal{O}(n \log n)$ lösen können. Nun muss noch die Zahl der Iterationen der binären Suche gezählt werden. Im ursprünglichen Scheduling–Problem haben wir ganzzahlige Prozesszeiten vorausgesetzt. Daher kann die binäre Suche abbrechen, sobald $ub - lb < 1$ gilt. Damit haben wir bereits eine polynomielle Laufzeit des Algorithmus. Das folgende Lemma ermöglicht, eine verbesserte Laufzeit zu erzielen.

Lemma 3.11
Wird Algorithmus A_ϵ mit k Iterationen in der binären Suche ausgeführt, dann hat der resultierende Schedule einen Makespan von höchstens $(1+9\epsilon)(1+2^{-(k-1)})C_{opt}$.

Daher werden nur $\mathcal{O}(\log(\frac{1}{\epsilon}))$ Iterationen benötigt, um eine $(1 + 10\epsilon)$–Approximation zu erhalten. Das Lemma ist gültig, da in Iteration k die Gleichung $ub - lb = 2^{-(k-2)} \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j, C_F\}$ gilt und da $C_{opt} \geq lb \geq \max\{\sum_{j=1}^{n} \frac{p_j}{m}, \max_j p_j, C_F\}$ gilt.

3.5 Uniforme parallele Maschinen

Im Falle uniformer paralleler Maschinen setzen wir voraus, dass Maschinen mit unterschiedlichen Geschwindigkeiten $r_1, \cdots, r_m$ arbeiten. Genauer gilt, dass das Ausführen von Job j auf Maschine i genau p_j/r_i Zeiteinheiten in Anspruch nimmt. Auch in diesem Szenario bleibt unser Ziel, die Jobs derart den Maschinen zuzuordnen, dass der letzte Job so früh wie möglich abgearbeitet ist. In [HS86] wird ein *PTAS* für die Standardversion des Scheduling–Problems vorgestellt, bei dem

keine fixierten Jobs vorliegen, aber unterschiedliche Maschinengeschwindigkeiten auftreten. Die Laufzeit ihres Algorithmus ist durch ein Polynom hohen Grades in n gegeben, jedoch ist die Maschinenzahl ein Teil der Eingabe.
Das in dieser Arbeit vorgestellte *PTAS* für das Scheduling–Problem mit fixierten Jobs kann ebenfalls auf den Fall unterschiedlicher Maschinengeschwindigkeiten erweitert werden, falls die Maschinenzahl eine Konstante ist. Zur Vereinfachung der Präsentation normieren wir die Geschwindigkeit der schnellsten Maschine auf 1. Dann gilt $r_i \leq 1,\ (i = 1, \cdots, n)$. Wir gehen davon aus, dass innerhalb der binären Suche ein Kandidat C für den Makespan vorgeschlagen wird. Da jede Maschine i genau r_i Einheiten eines Jobs in einer Zeiteinheit abarbeitet, kann sie insgesamt $C \cdot r_i$ Einheiten bis zur Zeit C abarbeiten.
Nun definiert C ein Packproblem mit offenen und geschlossenen Kisten. Wir skalieren alle Kisten und alle Jobs mit $\frac{1}{C}$ und multiplizieren die Größen der Kisten der Maschine i mit r_i. Das bedeutet, dass ein Job j, der irgendeiner Kiste auf Maschine i zugeordnet wurde, im Schedule dann $\frac{p_j}{r_i} \cdot C$ Zeiteinheiten auf Maschine i beansprucht. Die Aufgabe ist, wiederum zu entscheiden, ob eine gültige Zuordnung der skalierten Jobs J zu den Kisten B existiert.
Für unser *PTAS* verfeinern wir Algorithmus A_ϵ indem wir alle $\bar{J}$ zugeordneten Jobs der schnellsten Maschine zuordnen. Wegen $SIZE(J) \leq m$ and $SIZE(B) \leq m$, bleibt Korollar 3.9 gültig und daher erhalten wir für $C \geq C_{opt}$ und $\epsilon \leq \frac{1}{2}$ einen Makespan von höchstens $C + \frac{(5m+4)\epsilon}{\max_i(s_i)} \cdot C = C + (5m + 4)\epsilon \cdot C$.
Wir nennen den derart verfeinerten Algorithmus A'_ϵ und erhalten das folgende Theorem:

Theorem 3.12
Es gibt eine $\mathcal{O}(n \log n)$ Variante von Algorithmus A'_ϵ, die einen zulässigen Schedule mit einer Approximationsgüte von $1 + (5m + 5) \cdot \epsilon$ für eine Instanz i des Scheduling–Problems mit fixierten Jobs auf m Maschinen unterschiedlicher Geschwindigkeit findet.

3.6 Online–Algorithmen zum Scheduling mit fixierten Jobs

Auch für das Scheduling–Problem mit fixierten Jobs ist es natürlich, die Online–Variante durch einen *List*–Algorithmus anzugehen. Das Verfahren ergibt sich wie folgt:

- Packe so viele Jobs wie möglich in die geschlossenen Kisten.
- Setze die verbleibenden Jobs mit einem *List*–Algorithmus in die offenen Kisten.

Mögliche Strategien zum Packen der geschlossenen Kisten sind die klassischen Bin–Packing Algorithmen wie *Next–Fit, First–Fit, Best–Fit* [1] und eine Regel, die wir *Earliest–Fit* nennen wollen. *Earliest–Fit* folgt der Idee der *List*–Algorithmen und setzt einen Job in diejenige Kiste, die dem Job den frühsten Start ermöglicht. Wir bezeichnen den Makespan unter einem dieser *List*–Algorithmen mit C_L und den optimalen Makespan mit C_{Opt}. Mit einer Argumentation analog zum Beweis der $2-1/m$ Schranke beim Standardproblem des Minimierens des Makespan ohne fixierte Jobs (cf. Theorem 1.5) kann auch hier eine Approximationsgüte von 3 für obige *List*–Algorithmen nachgewiesen werden. Im Gegensatz zu Bin–Packing und Maintenance–Scheduling können aber beim Scheduling mit fixierten Jobs Beispiele konstruiert werden, bei denen die Approximationsgüte echt schlechter als 2 ist.

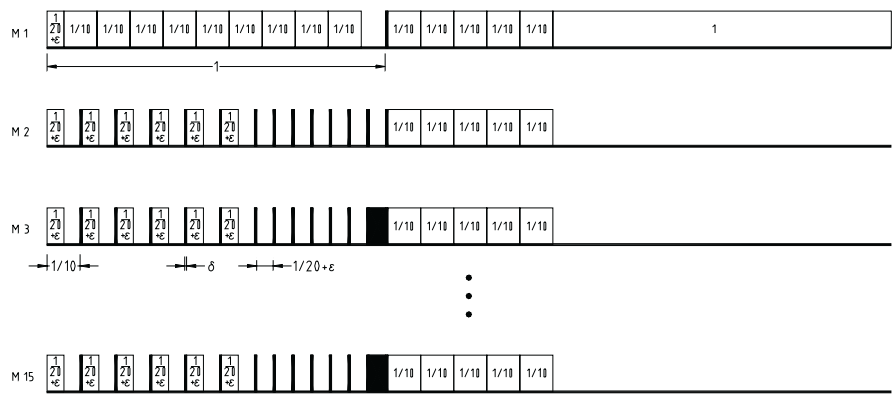


Abbildung 3.1: Schedule unter *Next–Fit, First–Fit* oder *Earliest–Fit*

Beispiel: *Gegeben sei das* 15*–Maschinenproblem mit*

- *1 Job mit Prozesszeit* 1,

[1] Die interessierten Leser seien zur näheren Information über die zahlreichen Ansätze, Bin–Packing effizient zu approximieren, an die Literatur, speziell [CGJ96] verwiesen.

- *84 Jobs mit Prozesszeiten* $\frac{1}{10}$,
- *85 Jobs mit Prozesszeiten* $\frac{1}{20} + \epsilon$, *wobei* $\epsilon > 0$ *ein hinreichend kleiner Wert sei.*

Die Instanz besteht des Weiteren aus fixierten Jobs mit unendlich kleiner Prozesszeit δ, *so dass sich auf Maschine* 1 *eine einzelne geschlossene Kiste der Länge* 1 *befindet und Maschine* 2 *sechs geschlossene Kisten der Länge* $\frac{1}{10}$ *hat, gefolgt von sieben geschlossenen Kisten der Länge* $\frac{1}{20} + \epsilon$ *sowie einem fixierten Job, der zum Zeitpunkt* $1 + \delta$ *endet. Auf den übrigen Maschinen liegen sechs geschlossene Kisten der Länge* $\frac{1}{10}$, *gefolgt von sechs geschlossenen Kisten der Länge* $\frac{1}{20} + \epsilon$ *und einem fixierten Job, der zum Zeitpunkt* $1 + \delta$ *endet. Man kann sich leicht davon überzeugen, dass alle nicht fixierten Jobs in die geschlossenen Kisten platziert werden können. Der optimale Makespan beträgt daher* $1 + \delta$. *In Abbildung 3.1 ist ein Schedule gezeigt, der nach geeigneter Permutation der Jobliste unter den Regeln* Next–Fit, First–Fit *und* Earliest–Fit *entstehen kann. Dieser Schedule hat einen Makespan von* $2 + \frac{5}{10} + \delta$. *Daher gilt für kleines* δ: $\frac{C_L}{C_{opt}} \approx 2.5$.

Erweiterte Instanzen führen sogar zu einer schlechteren unteren Schranke der Approximationsgüte von $2,6$. Für das gegeben Problem ist die *Best–Fit*–Regel den anderen Regeln deutlich überlegen, da sie sogar eine optimale Lösung erzeugt. Es ist jedoch nicht schwierig, Instanzen anzugeben, bei denen das Verhältnis der Lösung unter *Best–Fit* zur optimalen Lösung beliebig nahe an 2 heranreicht. Zudem ist die Laufzeit der *Best–Fit*–Regel von $O(n^2)$ deutlich schlechter, als die der anderen betrachteten Regeln.

KAPITEL 4

Stochastisches Scheduling

In den vorherigen Kapiteln haben wir deterministische Scheduling–Probleme untersucht, deren Praxisnähe aus dem Einbeziehen von Restriktionen aus der industriellen Produktionsplanung resultierte. In diesem Kapitel soll durch den Übergang zum stochastischen Modell nun auch der Begriff der *Performance* eines Algorithmus näher an der Praxis orientiert werden. In dem stochastischen Modell sind die Prozesszeiten nicht mehr willkürlich gegeben, sondern durch Verteilungsfunktionen bestimmt. Durch die Vorgabe dieser Verteilungsfunktionen für die Prozesszeiten der Jobs können Anwendungsprobleme modelliert werden, in denen die Verteilungen die üblichen Eingabegrößen des betrachteten Problems widerspiegeln. Bei der bisherigen Worst–Case–Analyse hingegen, wurde nicht auf die typische Struktur der Eingabe eingegangen, sondern es wurde immer vom schlimmsten, aber möglicherweise in der Praxis selten (oder gar nicht) eintretenden Fall ausgegangen und die Performance des Algorithmus für diesen Fall studiert. Der Fokus des stochastischen Scheduling liegt dagegen auf dem typischen Verhalten der Algorithmen in der gegebenen Problemstellung.
Stochastisches Scheduling ist ein Forschungsgebiet von hoher praktischer Relevanz, aber bisher konnten nur wenige zufriedenstellende Resultate erzielt werden. In dieser Arbeit soll aber dennoch ein Schritt in Richtung einer Analyse eines stochastischen Scheduling–Problems gemacht werden. Wir stellen einen neuen Performancebegriff für die Analyse stochastischer Scheduling–Algorithmen vor und führen eine Analyse eines speziellen Scheduling–Problems anhand dieses Performancebegriffes durch. Das von uns betrachtete Performancemaß verlässt die klassischen Modelle des stochastischen Scheduling und führt zu neuen, bisher noch nicht betrachteten Fragestellungen.
Wir beginnen mit einem kurzen Überblick über die bisher analysierten Fragestellungen im Bereich des stochastischen Scheduling. Für eine weitergehende Einführung in die stochastischen Varianten deterministischer Scheduling–Probleme verweisen wir auf [LLK91]. Im klassischen stochastischen Scheduling werden die Algorithmen daraufhin untersucht, ob sie den Wert der Zielfunkti-

on des betrachteten Problems im *Erwartungswert* minimieren. In verschiedenen Veröffentlichungen wurden Optimalitätskriterien für Scheduling–Strategien bestimmt, wenngleich auch Optimalitätsaussagen nur für wenige Probleme gelangen.
Die einfachste, aber häufig auch erfolgreichste Scheduling–Strategie ist die aus dem deterministischen Scheduling bekannte *List*–Strategie: Anhand vorgegebener Prioritätsregeln werden die Jobs der Jobliste in eine Reihenfolge gebracht. Der nächste Job aus der Jobliste wird immer dann ausgewählt, und zwar unabhängig vom augenblicklichen Zustand des Systems, wenn eine Maschine wieder verfügbar wird.
Die Reihenfolgen der Jobs innerhalb der Joblisten sind dabei üblicherweise über die Erwartungswerte der Prozesszeiten oder durch sonstige Eigenschaften der Verteilungen der Prozesszeiten bestimmt. In einer ganze Reihe von Arbeiten wurde das stochastische Verhalten derartiger, auf Prioritätsregeln basierender Strategien untersucht [Gla79] [MRW84][MRW85][MR85] [PT87][LS97].
Zwei typische Vertreter dieser Strategien sind die *LEPT*–Regel (von engl. *longest expected processing time first*), bei der Jobs nach nicht aufsteigenden erwarteten Prozesszeiten sortiert werden, und die gegenteilige *SEPT*–Regel (von engl. *shortest expected processing time first*), bei der die Jobs nach nicht absteigenden erwarteten Prozesszeiten sortiert werden. Zwei schon vergleichsweise früh bekannte Resultate besagen, dass die *LEPT*–Regel den erwarteten Makespan für das Scheduling–Problem auf parallelen Maschinen minimiert [WP80] [Käm87] und dass für das Problem, die erwartete Summe der Fertigstellungszeiten auf einer Maschine zu minimieren, die *SEPT*–Strategie optimal ist. Für das allgemeinere Problem der *gewichteten* Fertigstellungszeiten ist die *WSEPT*–Regel (engl. *weighted expected shortest processing time first*) im Falle einer einzigen Maschine ebenfalls optimal [Rot66]. Die *WSEPT*–Strategie sortiert die Jobs nach nicht absteigendem Verhältnis $E[\mathbf{p_j}]/w_j$, wobei $E[\mathbf{p_j}]$ die erwartete Prozesszeit von Job j bezeichne. Intuitiv sind die genannten Resultate sofort einsichtig, denn aus den bekannten Offline–Resultaten (vgl. Theoreme 1.5 und 1.6) der beiden Probleme folgt, dass es für das Problem des Minimierens des Makespan sinnvoll ist, wenn am Ende des Schedules keine langen Jobs mehr kommen, während bei dem Problem der Summe der Fertigstellungszeiten kurze Jobs vorne liegen sollten. Für das Problem des Minimierens der (gewichteten) erwarteten Summe der Fertigstellungszeiten auf parallelen Maschinen ist keine optimale Strategie bekannt. Die *SEPT*–Strategie ist jedoch optimal, wenn die Prozesszeiten der Jobs durch exponentialverteilte Zufallsvariablen gegeben sind [WP80] [WVW86]. Die *WSEPT*–Strategie ist optimal, wenn die Prozesszeiten exponentialverteilt sind und die Jobgewichte zusätzlich mit dem Verhältnis von Gewicht zur erwarteten Prozesszeit verträglich sind [Käm87]. Für generelle Strategien wurde versucht, die Distanz der *SEPT*–Strategie oder der *WSEPT*–Strategie zu einer im Sinne des Minimierens der Erwartungswerte optimalen Strategie zu bestimmen. Eine erste Abschätzung gab Weiss [Wei90] und später konnten für Strategien, welche

auf Prioritätsregeln basieren, konstante Approximationsgüten mittels Methoden der Polyedertheorie nachgewiesen werden [MSU99]. Die Güte wurde dabei als das Verhältnis zwischen dem erwarteten Wert der Zielfunktion unter der *SEPT*–Regel oder der allgemeineren *WSEPT*–Regel und der erwarteten Lösung in der — bei diesem Problem nicht bekannten — optimalen Strategie definiert. Die Polyedertheorie ermöglichte eine deutlich elegantere Beweisführung, als bei Weiss, und sie führte zu stärkeren Approximationsaussagen.
Auch wenn die Verteilungen für den Wert der Zielfunktionen bisher noch nicht abgeschätzt oder gar berechnet werden konnten, scheinen die Probleme der Makespanminimierung und der Minimierung der Summe der Fertigstellungszeiten zunächst einmal gelöst zu sein, vorausgesetzt die Prozesszeiten sind exponentialverteilt. Für generelle Verteilungen ist (abgesehen von obig erwähnten Resultat der Approximationsgüte von *WSEPT* [MSU99]) so gut wie nichts bekannt.
Wir wollen nun aber zeigen, dass die obig beschriebene traditionelle Sichtweise auf das stochastische Scheduling zu irreführenden Aussagen führen kann. Wir werden dazu einen unserer Ansicht nach exakteren Begriff der Güte eines stochastischen Schedulingalgorithmus einführen, der zeigt, dass die bisher als optimal eingeschätzten Strategien durchaus nur suboptimal sein können. Die Motivation für unseren neuen Gütebegriff bei stochastischen Scheduling–Problemen liegt in dem in jüngster Zeit leicht veränderten Verständnis der Performance von generellen Online–Problemen. Bei der Analyse von Online–Algorithmen wird die Qualität eines Algorithmus relativ zum besten möglichen Ergebnis eines Algorithmus gemessen, der eine vollständige Übersicht über alle zukünftigen Ereignisse hat. Insbesondere kennt er alle Prozesszeiten im Voraus. Diese Methode zur Untersuchung von Strategien mit Online–Entscheidungen nennt man *kompetitive Analyse* und sie ist mittlerweile zu einem Standardwerkzeug in der Theorie der Online–Algorithmen geworden. Obwohl stochastische Probleme ebenfalls "online" sind, wurde die kompetitive Analyse im Kontext des stochastischen Scheduling bisher weitgehend ignoriert.
Daher werden wir im Folgenden den traditionellen Ansatz zur Analyse stochastischer Algorithmen verlassen und die Analyse deutlicher auf die Theorien der Online–Algorithmen ausrichten. Unserer Ansicht nach ist der Ansatz der kompetitiven Analyse präziser als der Standardansatz, der ausschließlich den erwarteten Wert der Zielfunktion betrachtet, denn bei der kompetitiven Analyse wird genauer unterschieden, wie sich der Algorithmus bei schweren und bei einfachen Eingaben verhält. Speziell können Instanzen existieren, die für eine gegebene Strategie schwierig sind und daher zu einem hohen Wert der Zielfunktion führen. Wenn diese Instanzen aber auch in der optimalen Offline–Lösung zu einem hohen Zielfunktionswert führen, kann die Strategie immer noch als "gut" bewertet werden. Ein solcher Effekt wird hingegen in traditionellen stochastischen Modell nicht beachtet. Zudem besteht die Gefahr, dass die einzelnen Prozesszeiten nicht um ihren Erwartungswert konzentriert sind, so dass auch hier die Idee der erwarteten Kosten irreführend ist.

Der Grundidee der relativen Performance wurde bereits einmal in einem anderen Zusammenhang in einen Artikel von Coffmann und Gilbert [CG85] bei der Untersuchung des Problems des Minimierens des Makespan aufgegriffen. In ihrer Arbeit erstellten sie Schranken für die erwartete relative Performance von *List*–Strategien.
Das in dieser Arbeit von uns untersuchte Problem ist das in Kapitel 1.1 eingeführte Problem der Minimierung der Summe der Fertigstellungszeiten von Jobs bei einer einzigen Maschine, diesmal allerdings in der stochastischen Variante. Für dieses Problem werden wir wiederum die Frage der Optimalität stellen und zwar hier nun im Sinne der kompetitiven Analyse. Wir tun dies, obwohl die *SEPT*–Regel gemeinhin als optimal gilt, da sie den Erwartungswert der Zielfunktion minimiert. Wir werden zeigen, dass die standard SEPT–Regel entgegen der Erwartungen nicht grundsätzlich optimal im Sinne der kompetitiven Analyse ist. Als zweites zeigen wir, dass die Optimalität der SEPT–Regel aber beibehalten wird, wenn für die Prozesszeiten die Exponentialverteilung zugrundegelegt wird. In diesem Falle geben wir eine konstante obere Schranke für die erwartete relative Performance an, die unabhängig von der Zahl der Jobs ist, vorausgesetzt, alle Verteilungen haben den gleichen Parameter λ. Wenn die Exponentialverteilungen beliebige Parameter haben dürfen, können wir immer noch eine $\mathcal{O}(\log(n))$ Schranke für das n Job Problem beweisen, wobei wir allerdings vermuten, dass auch hier eine konstante relative Performance zu erwarten ist.

4.1 Modell und Notation

Wir folgenden der Terminologie aus [MRW84] und [MRW85] bezüglich der Modelle des stochastischen Scheduling. Für eine Einführung in die Begriffe und Konzepte der Wahrscheinlichkeitstheorie verweisen wir auf [Fel57].
Das betrachtete Scheduling–Problem besteht aus einer endlichen Menge von Jobs $J = \{1, \cdots, n\}$, die ohne Preemption auf einer einzigen Maschine eingeplant werden müssen, so dass die Summe der Fertigstellungszeiten $\sum_{j \in J} C_j$ minimiert wird. Hierbei bezeichnet C_j die Fertigstellungszeit von Job j. Die Prozesszeiten der einzelnen Jobs sind nicht im Voraus bekannt, aber als einzige zur Verfügung stehende Information ist für jeden Job j eine Verteilung seiner Prozesszeiten gegeben. Die Prozesszeiten können also als Zufallsvariablen $\mathbf{p} = (\mathbf{p_1}, \cdots, \mathbf{p_n})$ angesehen werden. In dem hier betrachteten Beispiel sind diese Zufallsvariablen standard exponentialverteilt mit einem Parameter λ_j für jeden Job j. Nach dem Klassifikationsschema für Schedulingprobleme von Graham et. al. [GLLK79] wird das Problem dargestellt als:

$$1 \mid \mathbf{p_j} \sim stoch \mid \sum C_j.$$

Die Jobs werden anhand einer Strategie Π eingeplant. Eine Strategie spezifiziert, welche Jobs zu bestimmten Entscheidungszeitpunkten t eingeplant werden sollen.

Entscheidungszeitpunkte sind dabei die Zeiten, zu denen ein Job fertiggestellt wird, aber auch — wenn Maschinenleerzeiten erlaubt sind — zusätzlich definierte Zeitpunkte t im Plan, zu denen eine Maschine gerade frei ist. Es wird allerdings gefordert, dass die Entscheidung in einer Strategie eindeutig aus den bedingten Verteilungen der Prozesszeiten bis zum Zeitpunkt t abgeleitet werden, nicht aber aus möglichen zukünftigen Ereignissen.
Eine gegebene Strategie Π erzeugt schließlich einen gültigen Schedule σ mit den a posteriori realisierten Prozesszeiten. Daher können Zufallsvariablen $C_1^{\Pi}, \ldots, C_n^{\Pi}$ definiert werden, welche die (zufälligen) Fertigstellungszeiten der Jobs unter Strategie Π beschreiben. Dadurch wird eine Zufallsvariable

$$\sum_{j \in J} \mathbf{C_j^{\Pi}}$$

definiert, die den Wert der Zielfunktion beschreibt.
Auf ähnliche Weise kann der Zielfunktionswert des optimalen Offline–Schedule bestimmt werden, bei dem alle Prozesszeiten von vorneherein bekannt sind. Wir beschreiben diesen Wert durch die Zufallsvariable

$$\sum_{j \in J} \mathbf{C_j^{offl.}},$$

wobei $C_j^{offl.}$ die Fertigstellungszeit von Job j im optimalen Offline–Algorithmus bezeichnet.
Mit diesen Definitionen ist das Verhältnis

$$\frac{\sum_{j \in J} w_j \cdot \mathbf{C_j^{\Pi}}}{\sum_{j \in J} w_j \cdot \mathbf{C_j^{offl.}}}$$

eine Zufallsvariable, die die relative Performance einer Strategie Π zur bestmöglichen Lösung für die a posteriori gegebenen Prozesszeiten angibt.
Wir nennen eine Strategie *relativ optimal im Erwartungswert*, wenn sie die minimale erwartete relative Performance ergibt, also wenn sie das Infimum annimmt über

$$\{E[\frac{\sum_{j \in J} \mathbf{C_j^{\Pi}}}{\sum_{j \in J} \mathbf{C_j^{offl.}}}] \mid \Pi \; zul\ddot{a}ssige Strategie.]\}$$

4.2 Nichtoptimalität der *SEPT*–Regel

In diesem Kapitel geben wir ein Beispiel dafür, dass die *SEPT*–Scheduling Regel zwar die erwartete Summe der Fertigstellungszeiten von Jobs für das Problem $1 \,|\mathbf{p_j} \sim stoch \mid \sum \mathbf{C_j}$ minimiert, jedoch nicht die erwartete relative Performance. In der deterministischen Variante des Problems kann ein optimaler Schedule in

einfacher Weise durch Sortieren der Jobs nach nicht absteigenden Prozesszeiten erzeugt werden (vergl. Theorem 1.6). In der stochastischen Variante des Problems minimiert die *SEPT*–Strategie die erwartete Summe der Fertigstellungszeiten [WP80]. Sie gilt daher für dieses Problem als optimal. Das folgende Beispiel zeigt aber, dass die gleiche *SEPT*–Regel die erwartete relative Performance nicht immer minimieren kann, also im Sinne der kompetetiven Analyse nicht optimal ist. Die *SEPT*–Regel führt in diesem Beispiel im Erwartungswert zu schlechteren Ergebnissen, als eine andere Strategie.

Beispiel:
Gegeben sei die folgende Instanz mit zwei Jobs: Job 1 *habe eine Prozesszeit* $p_1 = 1/2$ *und Job* 2 *habe eine Prozesszeit* $p_2 = 2$ *mit Wahrscheinlichkeit* $P[p_2 = 2] = 1/2$ *und eine Prozesszeit* $p_2 = \epsilon$ *bei geeignet kleinem* ϵ *mit Wahrscheinlichkeit* $P[p_2 = \epsilon] = 1/2$. *Dann gilt* $E[p_1] = 1/2$ *und* $E[p_2] > 1$. *Daher wird die* SEPT–*Strategie Job* 1 *als erstes einplanen und danach Job* 2. *Die* LEPT–*Regel dagegen plant erst Job* 2 *ein und danach Job* 1. *Je nachdem, welche Prozesszeit Job* 2 *im Zufallsexperiment erhält, wird in der optimalen Offline–Lösung als erstes Job* 1 *eingeplant, falls* $p_2 = 2$, *oder Job zwei, falls* $p_2 = \epsilon$.
Für $\epsilon \to 0$ *folgt:*

$$\begin{aligned} E[\frac{\sum_{j\in J} C_j^{SEPT}}{\sum_{j\in J} C_j^{offl.}}] &= \frac{p_1 + (p_1 + p2)}{p_1 + (p_1 + p2)} \cdot P[p_2 = 2] + \frac{p_1 + (p_1 + p2)}{p_2 + (p_1 + p2)} \cdot P[p_2 = \epsilon] \\ &= \frac{2 \cdot 1/2 + 2}{2 \cdot 1/2 + 2} \cdot 1/2 + \frac{2 \cdot 1/2 + \epsilon}{2 \cdot \epsilon + 1/2} \cdot 1/2 \longrightarrow 3/2. \end{aligned}$$

Andererseits gilt

$$\begin{aligned} E[\frac{\sum_{j\in J} C_j^{LEPT}}{\sum_{j\in J} C_j^{offl.}}] &= \frac{p_2 + (p_1 + p2)}{p_1 + (p_1 + p2)} \cdot P[p_2 = 2] + \frac{p_2 + (p_1 + p2)}{p_2 + (p_1 + p2)} \cdot P[p_2 = \epsilon] \\ &= \frac{2 \cdot 2 + 1/2}{2 \cdot 1/2 + 2} \cdot 1/2 + \frac{2 \cdot \epsilon + 1/2}{2 \cdot \epsilon + 1/2} \cdot 1/2 \longrightarrow 1/2 \cdot \frac{7.5}{3} < 3/2. \end{aligned}$$

Damit ist die SEPT–Regel in diesem Beispiel also nicht optimal. Der Grund für dieses Resultat ist, dass die Prozesszeit von Job 1 zwar um ihren Erwartungswert konzentriert ist, die Prozesszeit von Job 2 jedoch nicht. Die hohe Varianz von Job 2 führt dazu, dass der Erwartungswert alleine nicht genügend Hinweise auf die geeignetere Jobreihenfolge geben kann.
Im nächsten Abschnitt wird mit der Exponentialverteilung eine etwas "natürlichere" Verteilung betrachtet werden, bei der die *SEPT*–Strategie sowohl unter dem klassischen Performancebegriff als auch unter der neuen Sichtweise optimal ist.

4.3 Erwartete relative Performance der *SEPT*–Regel für exponentialverteilte Prozesszeiten

In diesem Abschnitt setzen wir voraus, dass die Prozesszeiten der Jobs durch exponentialverteilte Zufallsvariablen gegeben sind.

Theorem 4.1
Sind die Prozesszeiten der Jobs p_i exponentialverteilt mit Parameter λ_i, dann ist eine Strategie Π im Erwartungswert genau dann für das Problem $1\ |\mathbf{p_j} \sim exp(\lambda_i)\ |\ \sum C_j$ relativ optimal, wenn Π die SEPT*–Strategie ist.*

Beweis. Sei $J = \{1, \cdots n\}$ die Menge der Jobs und sei p_i die Prozesszeit von Job i. Die Prozesszeit eines Jobs i sei durch eine Zufallsvariable mit Dichtefunktion $\mu_i(x) = \lambda_i e^{-\lambda_i x}$ gegeben.
Wir betrachten zwei Jobs i und j wobei $E[p_i] < E[p_j]$, also $\lambda_i > \lambda_j$. Seien S_1 und S_2 zwei Strategien, wobei S_1 Job i vor Job j einplant, während Strategie S_2 die Positionen von i und j vertauscht, aber alle anderen Positionen beibehält. Die Position eines Job k im Schedule unter S_1 bzw. S_2 sei mit $s_1(k)$ und $s_2(k)$ bezeichnet. Es gilt also $(s_1(j) - s_2(j)) = -(s_1(i) - s_2(i))$.
Dann folgt

$$\begin{aligned}
E[\frac{\sum_{j\in J} \mathbf{C_j^{S_2}}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}] \quad - \quad & E[\frac{\sum_{j\in J} \mathbf{C_j^{S_1}}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}] \\
= \ & E[\frac{\sum_{i=1}^{n}(n+1-s_2(i))\mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}] - E[\frac{\sum_{i=1}^{n}(n+1-s_1(i))\mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}] \\
= \ & E[\frac{(s_1(j)-s_2(j))\cdot \mathbf{p_j}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}] + E[\frac{(s_1(i)-s_2(i))\cdot \mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}] \\
= \ & (s_1(j)-s_2(j))E[\frac{\mathbf{p_j}-\mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}].
\end{aligned}$$

Zum Beweis von Theorem genügt es daher, das folgende zu zeigen:

$$E[\frac{\mathbf{p_j}-\mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}] \geq 0, \tag{4.1}$$

Dann wird der Austausch zweier beliebiger Jobs, die nicht anhand der *SEPT*–Strategie sortiert sind, die erwartete relative Performance der Strategie verringern. Wir beweisen (4.1) wie folgt: Wir summieren über alle möglichen zufälligen Prozesszeiten für jeden Job, wobei die Prozesszeiten mit dem korrespondierenden Wert ihrer Dichtefunktion gewichtet werden. Wir setzen

$$\begin{aligned}
x: &\quad \text{Prozesszeit des kleineren der Jobs } \{p_i, p_j\}, \\
y: &\quad \text{Prozesszeit des größeren der Jobs } \{p_i, p_j\}, \\
t = (t_1, ..t_{n-2}): &\quad \text{Prozesszeit der übrigen Jobs .}
\end{aligned}$$

Der Vektor t ist durch die Produktdichtefunktion $\mu_{n-2}(t)$ gegeben, die sich aus den einzelnen Dichtefunktionen der zugehörigen Jobs zusammensetzt. Für feste Werte $(x, y, t) \in R_+^n$ entsteht ein optimaler Offline–Schedule, dessen Wert im Folgenden mit $Opt^{offl}(x, y, t)$ bezeichnet sei. Dann gilt,

$$
\begin{aligned}
& E[\frac{\mathbf{p_j} - \mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl}}}] \\
= \; & E[\frac{\mathbf{p_j} - \mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl}}} | \mathbf{p_i} < \mathbf{p_j}] \cdot P[\mathbf{p_i} < \mathbf{p_j}] + E[\frac{\mathbf{p_j} - \mathbf{p_i}}{\sum_{j\in J} \mathbf{C_j^{offl}}} | \mathbf{p_i} \geq \mathbf{p_j}] \cdot P[\mathbf{p_i} \geq \mathbf{p_j}] \\
= \; & \int_{t=0}^{\infty}\int_{x=0}^{\infty}\int_{y=x}^{\infty} \frac{y-x}{Opt^{offl}(x,y,t)} \mu_j(y)\mu_i(x)\mu_{n-2}(t) dy dx dt \\
& + \int_{t=0}^{\infty}\int_{x=0}^{\infty}\int_{y=x}^{\infty} \frac{x-y}{Opt^{offl}(x,y,t)} \mu_j(x)\mu_i(y)\mu_{n-2}(t) dy dx dt \\
= \; & \int_{t=0}^{\infty} \mu_{n-2}(t) \int_{x=0}^{\infty}\int_{y=x}^{\infty} \frac{y-x}{Opt^{offl}(x,y,t)} \cdot (\mu_j(y)\mu_i(x) - \mu_j(x)\mu_i(y)) dy dx dt \\
= \; & \int_{t=0}^{\infty} \mu_{n-2}(t) \int_{x=0}^{\infty}\int_{y=x}^{\infty} \frac{y-x}{f(p(J_{ij}), x, y)} \cdot \lambda_i\lambda_j(e^{-(\lambda_j y + \lambda_i x)} - e^{-(\lambda_j x + \lambda_i y)}) dy dx dt \\
& \geq 0,
\end{aligned}
$$

da $y \geq x$ und da $e^{-(\lambda_j y+\lambda_i x)} - e^{-(\lambda_j x+\lambda_i y)}) \geq 0$ für $\lambda_i \geq \lambda_j$. □

4.4 Schranken für die erwartete relative Performance von SEPT

Wir setzten für dieses Kapitel voraus, dass die Prozesszeiten aller Jobs durch die Exponentialverteilung mit dem gleichen Parameter λ gegeben sei. Das folgende Theorem zeigt, dass die erwartete relative Performance unter der *SEPT*–Strategie für eine beliebige Anzahl von Jobs klein ist.

Theorem 4.2
Seien $n > 2$ Jobs gegeben, deren Prozesszeiten jeweils durch eine exponentialverteilte Zufallsvariable mit dem gleichen Parameter λ bestimmt seien. Dann ist die erwartete relative Performance $E[\frac{\sum_{j\in J} \mathbf{C_j^{SEPT}}}{\sum_{j\in J} \mathbf{C_j^{offl.}}}]$ der SEPT*–Schedulingstrategie durch eine Konstante $k < 3 + \frac{2}{n-2}$ beschränkt.*

Beweis. Da alle Jobprozesszeiten eine Verteilung mit gleichem Parameter λ und Erwartungswert $\frac{1}{\lambda}$ haben, ist unter der *SEPT*–Strategie jede Reihenfolge der Jobs gleich wahrscheinlich. Zur Vereinfachung der Notation nehmen wir für die folgenden Beweise $\lambda = 1$ an, aber die Beweise können für andere Werte λ in ähnlicher Weise durchgeführt werden. Daher können wir *SEPT* in diese Falle mit einer Strategie *RANDO* identifizieren, die die Jobs in irgendeiner Reihenfolge π einsortiert

und anhand dieser Reihenfolge einplant. Wir bezeichnen die Fertigstellungszeit von Job j unter Strategie *RANDO* mit C_j^R.
Wir betrachten $E[\frac{\sum_{j\in J}\mathbf{C_j^R}-\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}{\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}] = E[\frac{\sum_{j\in J}\mathbf{C_j^R}}{\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}] - 1$.
Die Zufallsvariable $\mathbf{p} = (\mathbf{p_1}, \ldots, \mathbf{p_n})$ der Jobprozesszeiten induziert eine Familie von *abhängigen* Zufallsvariablen $S_1, \cdots, S_n$, wobei S_i die Prozesszeit des i–ten kleinsten Jobs bezeichne. Unter Ausnutzung der bekannten Eigenschaften der Exponentialverteilung folgt $E[S_i] = \sum_{k=1}^{i} \frac{1}{(n+1-k)}$. Wir können die Zufallsvariablen umschreiben als $S_1 = Y_1'$, $S_2 = Y_1' + Y_2'$ und $S_i = \sum_{i=1}^{j} Y_j' = \sum_{i=1}^{n} \frac{1}{n+1-j} Y_j$. Dabei bezeichnet Y_j' eine *unabhängige* exponentialverteilte Zufallsvariable mit Parameter $(n+1-j)$. Wir bezeichnen ferner mit Y_j eine Zufallsvariable mit standard Exponentialverteilung, also mit Parameter $\lambda = 1$. Betrachte nun die Summe der Fertigstellungszeiten für einen gegebenen Schedule. Wie man sofort sieht, wird der erste Job n mal in der Summe der Fertigstellungszeiten gezählt, der zweite Job $n-1$ mal und allgemein, der i–te Job $(n+1-i)$ mal. Es ist außerdem bekannt, dass die Summe der Fertigstellungszeiten genau dann minimiert wird, wenn die Jobs anhand ihrer nicht absteigenden Prozesszeiten sortiert werden, wenn also kurze Jobs vor langen Jobs eingeplant werden. Wir bezeichnen die Permutation der Jobs unter der Strategie *RANDO* mit π. Es gilt,

$$\frac{\sum_{j\in J}\mathbf{C_j^R} - \sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}{\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}} = \frac{\sum_{i=1}^{n}(i-\pi_i)S_i}{\sum_{i=1}^{n}(n+1-i)S_i}. \tag{4.2}$$

Außerdem gilt,

$$\begin{aligned}
\frac{1}{n!}\Big(\sum_{\pi}\sum_{i=1}^{n}\pi_i S_i\Big) &= \sum_{i=1}^{n} S_i \cdot \frac{1}{n!}\Big(\sum_{\pi}\pi_i\Big) \\
&= \sum_{i=1}^{n} S_i \cdot \frac{1}{n!}(n-1)!\Big(\sum_{j=1}^{n} j\Big) \\
&= \sum_{i=1}^{n} S_i \cdot \frac{1}{n!}(n-1)!\frac{n(n+1)}{2} \\
&= \sum_{i=1}^{n} S_i \cdot \frac{n+1}{2}.
\end{aligned} \tag{4.3}$$

Bezeichne $s = (s_1, \ldots, s_n) \subset R_+^n$ zufällige Werte der Zufallsvariablen $\mathbf{S} = (\mathbf{S_1}, \ldots, \mathbf{S_n})$ mit einer Dichtefunktion $\mu_s(s)$ über den Produktraum $s \in R_+^n$ und bezeichne P eine Wahrscheinlichkeit. Da alle Permutationen π gleich wahr-

scheinlich sind, folgt mit (4.3),

$$\begin{aligned}
E[\frac{\sum_{j\in J}\mathbf{C_j^R}-\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}{\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}] &= \sum_{\pi}P[\pi]E[\frac{\sum_{i=1}^{n}(n+1-\pi_i)S_i-\sum_{i=1}^{n}(n+1-i)S_i}{\sum_{i=1}^{n}(n+1-i)S_i}] \\
&= \int_s \mu_s(s)\sum_{\pi}P[\pi]\frac{\sum_{i=1}^{n}(i-\pi_i)s_i}{\sum_{i=1}^{n}(n+1-i)s_i}ds \\
&= \int_s \mu_s(s)\sum_{\pi}\frac{1}{n!}\frac{\sum_{i=1}^{n}(i-\pi_i)s_i}{\sum_{i=1}^{n}(n+1-i)s_i}ds \\
&= \int_s \frac{\mu_s(s)}{\sum_{i=1}^{n}(n+1-i)s_i}(\sum_{i=1}^{n}i\cdot s_i-\frac{1}{n!}\sum_{\pi}\sum_{i=1}^{n}\pi_i s_i)ds \\
&= \int_s \mu_s(s)\frac{\sum_{i=1}^{n}(i-\frac{n+1}{2})s_i}{\sum_{i=1}^{n}(n+1-i)s_i}ds \\
&= E[\frac{\sum_{i=1}^{n}(i-\frac{n+1}{2})S_i}{\sum_{i=1}^{n}(n+1-i)S_i}]. \qquad (4.4)
\end{aligned}$$

Wir setzten $S_i := S_{i-1}+Y_i = \sum_{j=1}^{i}Y_j'$ wobei die einzelnen Y_j' aufgrund der bekannten Gedächtnislosigkeit der Exponentialfunktion *unabhängige* Zufallsvariablen mit einer Exponentialverteilung mit Parameter $(n+1-j)\lambda$ sind.
Dann lässt sich (4.4) schreiben als

$$\begin{aligned}
E[\frac{\sum_{j\in J}\mathbf{C_j^R}-\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}{\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}] &= E[\frac{\sum_{i=1}^{n}(i-\frac{n+1}{2})S_i}{\sum_{i=1}^{n}(n+1-i)S_i}] \\
&= E[\frac{\sum_{i=1}^{n}\sum_{j=1}^{i}(i-\frac{n+1}{2})Y_j'}{\sum_{i=1}^{n}\sum_{j=1}^{i}(n+1-i)Y_j'}] \\
&= E[\frac{\sum_{j=1}^{n}\sum_{i=j}^{n}(i-\frac{n+1}{2})Y_j'}{\sum_{j=1}^{n}\sum_{i=j}^{n}(n+1-i)Y_j'}] \\
&= E[\frac{\sum_{j=1}^{n}\sum_{i=j}^{n}(i-\frac{n+1}{2})\frac{1}{(n+1-j)}Y_j}{\sum_{j=1}^{n}\sum_{i=j}^{n}(n+1-i)\frac{1}{(n+1-j)}Y_j}] \\
&= E[\frac{\sum_{j=1}^{n}c_{n,j}Y_j}{\sum_{j=1}^{n}d_{n,j}Y_j}], \qquad (4.5)
\end{aligned}$$

wobei

$$\begin{aligned}
c_{n,j} &= \sum_{i=j}^{n}(i-\frac{n+1}{2})\frac{1}{(n+1-j)} \\
d_{n,j} &= \sum_{i=j}^{n}(n+1-i)\frac{1}{(n+1-j)}
\end{aligned}$$

und $Y_1,\ldots,Y_n$ voneinander unabhängige standard exponentialverteilte Zufallsvariablen sind, ihre Dichtefunktion ist also jeweils durch $f(x) := \exp^{-x}$ gegeben.

Zur Vereinfachung der Notation schreiben wir $c_j := c_{n,j}$ und $d_j := d_{n,j}$. Wir erhalten

$$\begin{aligned} c_j &= \sum_{i=j}^{n}(i - \frac{n+1}{2})\frac{1}{(n+1-j)} \\ &= (\frac{n(n+1)}{2} - \frac{j(j-1)}{2} - (n+1-j)\frac{n+1}{2})\frac{1}{(n+1-j)} \\ &= (\frac{(n+1)}{2}(n-(n+1-j)) - \frac{j(j-1)}{2})\frac{1}{(n+1-j)} \\ &= (\frac{(n+1)}{2}(j-1) - \frac{j(j-1)}{2})\frac{1}{(n+1-j)} \\ &= \frac{(j-1)}{2}, \end{aligned} \tag{4.6}$$

und

$$\begin{aligned} d_j &= \frac{1}{(n+1-j)}\sum_{i=j}^{n}(n+1-i) \\ &= \frac{1}{(n+1-j)}\sum_{i=1}^{n+1-j} i \\ &= \frac{(n+2-j)}{2}. \end{aligned}$$

Hierbei gilt $d_j > d_{j+1}$ für alle $j = 1, \ldots, n-1$.
Es gilt das folgende Lemma:

Lemma 4.3
Für unabhängige standardexponentialverteilte Zufallsvariablen $X_1, \ldots, X_n$, $n \geq 2$ gilt

$$E[\frac{X_i}{X_1 + \cdots + X_n}] = \frac{1}{n}, \tag{4.7}$$

für $i = 1, \ldots, n$, und

$$E[\frac{1}{X_1 + \cdots + X_n}] = \frac{1}{n-1}. \tag{4.8}$$

Beweis. Es gilt

$$1 = E[\frac{X_1 + \cdots + X_n}{X_1 + \cdots + X_n}] = \sum_{i=1}^{n} E[\frac{X_i}{X_1 + \cdots + X_n}].$$

Aufgrund der Symmetrie in der rechten Seite der Ungleichung folgt Gleichung (4.7). Zum Beweis von (4.8) überlege man sich erst, dass die Summe $X_1+\cdots+X_n$ eine Dichtefunktion g_n hat mit

$$g_n(x) = \frac{x^{(n-1)}}{(n-1)!}e^{-x}.$$

Daher folgt

$$E[\frac{1}{X_1+\cdots+X_n}] = \int_0^\infty \frac{1}{x}\cdot\frac{x^{(n-1)}}{(n-1)!}e^{-x}dx = \frac{1}{(n-1)}\int_0^\infty g_{n-1}(x)dx = \frac{1}{n-1}.$$

□

Man erhält unter Verwenden des gerade bewiesenen Lemma,

$$\begin{aligned}
E[\frac{\sum_{j\in J}\mathbf{C_j^R} - \sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}{\sum_{\mathbf{j}\in\mathbf{J}}\mathbf{C_j^{offl.}}}] &= \sum_{j=1}^{n} c_j E[\frac{Y_j}{\sum_{i=1}^{n} d_i Y_i}] \\
&< \sum_{j=1}^{n} c_j E[\frac{Y_j}{\sum_{i=1}^{\frac{n}{2}} d_{\frac{n}{2}} Y_i}] \\
&= \sum_{j=1}^{\frac{n}{2}} \frac{c_j}{d_{\frac{n}{2}}} E[\frac{Y_j}{\sum_{i=1}^{\frac{n}{2}} Y_i}] + \sum_{j=\frac{n}{2}}^{n} \frac{c_j}{d_{\frac{n}{2}}} E[Y_j] E[\frac{1}{\sum_{i=1}^{\frac{n}{2}} Y_i}] \\
&= \sum_{j=1}^{\frac{n}{2}} \frac{c_j}{d_{\frac{n}{2}}}\frac{1}{\frac{n}{2}} + \sum_{j=\frac{n}{2}}^{n} \frac{c_j}{d_{\frac{n}{2}}}\cdot 1\cdot\frac{1}{\frac{n}{2}-1} \\
&< \sum_{j=1}^{n} \frac{c_j}{d_{\frac{n}{2}}}\frac{1}{\frac{n}{2}-1} \\
&= \frac{1}{\frac{n}{2}-1}\sum_{j=1}^{n}\frac{(j-1)}{\frac{n}{2}+2} \\
&< \frac{1}{\frac{n}{2}-1}\sum_{j=1}^{n}\frac{(j-1)}{\frac{n}{2}} \\
&= \frac{2}{n(\frac{n}{2}-1)}\sum_{j=1}^{n}(j-1) \\
&= \frac{2}{n(\frac{n}{2}-1)}\frac{n(n-1)}{2} \\
&= 2+\frac{2}{n-2}.
\end{aligned}$$

Damit ist aber Theorem 4.2 bewiesen. □

4.5 Exakter Wert

In diesem Abschnitt geben wir einen geschlossene Formel für die erwartete relative Performance der *SEPT*–Regel. Diese Formel führt zwar nicht zu einem konkreten Wert der erwarteten Performance, sie kann aber für kleine Jobzahlen numerisch ausgewertet werden. Wir verwenden zunächst das folgende bekannte Theorem über Linearkombinationen standardexponentialverteilter Zufallsvariablen:

Theorem 4.4
[JKB94] Für standardexponentialverteilte Zufallsvariablen $X_1, \ldots, X_n$ hat die Linearform

$$Y = \sum_{i=1}^{n} \lambda_i X_i, \ \lambda_i \neq \lambda_j, \lambda_i > 0 \tag{4.9}$$

die Dichtefunktion

$$p_Y(x) = \sum_{j=1}^{n} (\prod_{k \neq j} (\lambda_j - \lambda_k)^{-1}) \lambda_j^{n-2} e^{\frac{-x}{\lambda_j}}, x > 0. \tag{4.10}$$

Wir setzten

$$E[\frac{\sum_{j \in J} \mathbf{C_j^R} - \sum_{\mathbf{j} \in \mathbf{J}} \mathbf{C_j^{offl.}}}{\sum_{\mathbf{j} \in \mathbf{J}} \mathbf{C_j^{offl.}}}] = \sum_{j=1}^{n} c_j E[\underbrace{\frac{Y_j}{\sum_{i=1}^{n} d_i Y_i}}_{:=E_j}] = \sum_{j=1}^{n} c_j E_j$$

und erhalten mit Theorem 4.4:

$$E_j = \int_z P[\frac{Y_j}{\sum_{i=1}^{n} d_i Y_i} > z] dz = \int_z P[(\frac{1}{z} - d_j) Y_j > \sum_{i=1, i \neq j}^{n} d_i Y_i] dz \tag{4.11}$$

Sei f_1 die Dichtefunktion von Y_j, d. h. $f_1(s) = \lambda e^{-\lambda s}$ und sei $f_\sum$ die Dichtefunktion von $\sum_{i=1,i\neq j}^{n} d_i Y_i$. Dann folgt für $n > 2$,

$$\begin{aligned}
E_j &= \int_{z=0}^{\frac{1}{d_j}} \int_{s=0}^{\infty} \int_{t=0}^{(\frac{1}{z}-d_j)s} \underbrace{P[(\frac{1}{z}-d_j)s \geq t]}_{1} f_1(s) f_\sum(t) dt ds dz \\
&= \int_{z=0}^{\frac{1}{d_j}} \int_{s=0}^{\infty} \int_{t=0}^{(\frac{1}{z}-d_j)s} f_1(s) f_\sum(t) dt ds dz \\
&= \int_{z=0}^{\frac{1}{d_j}} \int_{s=0}^{\infty} \int_{t=0}^{(\frac{1}{z}-d_j)s} \lambda e^{-\lambda s} \sum_{i=1,i\neq j}^{n} (\Pi_{k\neq i,j})(d_i - d_k)^{-1} d_i^{n-3} e^{-\frac{t}{d_i}} dt ds dz \\
&= \sum_{i=1,i\neq j}^{n} (\Pi_{k\neq i,j})(d_i - d_k)^{-1} d_i^{(n-3)} \int_{z=0}^{\frac{1}{d_j}} \int_{s=0}^{\infty} \int_{t=0}^{(\frac{1}{z}-d_j)s} \lambda e^{-\lambda s} e^{-\frac{t}{d_i}} dt ds dz \\
&= \lambda \sum_{i=1,i\neq j}^{n} (\Pi_{k\neq i,j})(d_i - d_k)^{-1} d_i^{(n-3)} \cdot \\
&\quad \int_{z=0}^{\frac{1}{d_j}} \int_{s=0}^{\infty} -d_i [exp(-(\lambda + \frac{1}{d_i}(\frac{1}{z}-d_j))s) - exp(-\lambda s)] ds dz \\
&= -\lambda \sum_{i=1,i\neq j}^{n} (\Pi_{k\neq i,j})(d_i - d_k)^{-1} d_i^{(n-2)} \cdot \\
&\quad \int_{z=0}^{\frac{1}{d_j}} \underbrace{\int_{s=0}^{\infty} exp(-(\lambda + \frac{1}{d_i}(\frac{1}{z}-d_j))s) - exp(-\lambda s) ds}_{=:G(z)} dz.
\end{aligned}$$

Wir erhalten

$$\begin{aligned}
G(z) &= \int_{s=0}^{\infty} exp(-(\lambda + \frac{1}{d_i}(\frac{1}{z}-d_j))s) - exp(-\lambda s) ds \\
&= [-\frac{exp(-(\lambda + \frac{1}{d_i}(\frac{1}{z}-d_j))s)}{(\lambda + \frac{1}{d_i}(\frac{1}{z}-d_j))}]_0^\infty - [\frac{-1}{\lambda} exp(-\lambda s)]_0^\infty \\
&= \frac{1}{\lambda + \frac{1}{d_i}(\frac{1}{z}-d_j)} - \frac{1}{\lambda}.
\end{aligned}$$

Daher gilt

$$E_j = -\lambda \cdot \sum_{i=1,i\neq j}^{n} (\Pi_{k\neq i,j})(d_i - d_k)^{-1} d_i^{(n-2)} (\int_{z=0}^{\frac{1}{d_j}} \frac{1}{\lambda + \frac{1}{d_i}(\frac{1}{z}-d_j)} dz - \frac{1}{d_j \lambda}).$$

Nun berechnen wir den letzten Bestandteil aus (4.12). Dann folgt,

$$\begin{aligned}\int_{z=0}^{\frac{1}{d_j}} \frac{1}{1+\frac{1}{d_i}(\frac{1}{z}-d_j)}dz &= [\frac{d_i \cdot z}{d_i-d_j} - \frac{d_i \log(1+d_i z - d_j z)}{(d_i-d_j)^2}]_0^{\frac{1}{d_j}} \\ &= \frac{d_i(d_i-d_j-d_j\log(\frac{d_i}{d_j}))}{(d_i-d_j)^2 d_j}.\end{aligned}$$

Werden alle diese Ergebnisse zusammengesetzt, erhalten wir:

$$\begin{aligned}\sum_{j=1}^{n} c_j E_j &= \sum_{j=1}^{n} c_j [\sum_{i=1, i\neq j}^{n} (\Pi_{k\neq i,j} \frac{1}{d_i-d_k}) d_i^{(n-2)} \\ &\quad \cdot (\frac{-d_i(d_i-d_j-d_j\log(\frac{d_i}{d_j}))}{(d_i-d_j)^2 d_j} + \frac{1}{d_j})] \qquad (4.12)\end{aligned}$$

Wir definieren $D_i := \frac{i+1}{2}$. Da wir in (4.12) über alle i summieren, erhalten wir

$$\begin{aligned}\sum_{j=1}^{n} c_j E_j &= \sum_{j=1}^{n} c_j [\sum_{i=1, i\neq(n+1-j)}^{n} (\Pi_{k\neq i,(n+1-j)} \frac{1}{D_i-D_k}) D_i^{(n-2)} \\ &\quad \cdot (\frac{-D_i(D_i-D_{(n+1-j)}-D_{(n+1-j)}\log(\frac{D_i}{D_{(n+1-j)}}))}{(D_i-D_{(n+1-j)})^2 D_{(n+1-j)}} + \frac{1}{D_{(n+1-j)}})] \\ &= \sum_{j=1}^{n} \frac{j-1}{2} [\sum_{i=1, i\neq(n+1-j)}^{n} (\Pi_{k\neq i,(n+1-j)} \frac{i+1}{i-k}) \\ &\quad \cdot (\frac{1}{2} \cdot \frac{-(i+1)((i-(n+1-j))-(n+2-j)\log(\frac{i+1}{n+2-j}))}{(i-(n+1-j))^2(n+2-j)} + \frac{2}{n+2-j})] \\ &= \frac{1}{2}\sum_{j=1}^{n} \frac{j-1}{n+2-j} [\sum_{i=1, i\neq(n+1-j)}^{n} (\Pi_{k\neq i,(n+1-j)} \frac{i+1}{i-k}) \\ &\quad \cdot (2 - \frac{(i+1)(i-(n+1-j)-(n+2-j)\log(\frac{i+1}{n+2-j}))}{2(i-(n+1-j))^2}]\end{aligned}$$

Damit haben wir eine geschlossene Formel für die erwartete relative Performance der *SEPT*–Regel unter der Annahme standardexponentialverteilter Zufallsvariablen erhalten. Eine solche Formel kann mittels Computersimulationen für kleines n berechnet werden.

4.6 Exponentialverteilungen mit unterschiedlichen Parametern

Für die Analyse der letzten Kapitel war es entscheidend, dass die Zufallsvariablen der Prozesszeiten der Jobs alle den gleichen Parameter λ haben, damit jede

Reihenfolge der Jobs gleich wahrscheinlich ist. Bei unterschiedlichen Parametern, kann diese Annahme nicht mehr getroffen werden. Zwar ist unter der *SEPT–*Regel die "richtige" Permutation zweier beliebiger Jobs wahrscheinlicher als die "falsche", aber wenn sie einmal falsch ist, kann der betrachtete Quotient der Summe der Fertigstellungszeiten unter SEPT und der optimalen Offline–Lösung mit höherer Wahrscheinlichkeit groß werden, als im anderen Fall. Daher kann man nicht ohne weiteres annehmen, dass die erwartete relative Performance bei standardexponentialverteilten Jobs stets eine obere Schranke für den Fall von Exponentialverteilungen mit unterschiedlichen Parametern ist. Dies zu zeigen, ist weiterhin ein offenes Problem. Hier werden wir die erwartete relative Performance für den Fall unterschiedlicher Prozesszeiten auf direktem Wege abschätzen. In diesem Abschnitt geben wir eine obere Schranke von $\mathcal{O}(\log(n))$ für das n–Job Problem.

Theorem 4.5
Es gilt für die relative Performance der SEPT–Regel gegenüber einer optimalen offline Lösung:

1. $E[\frac{\sum_{j\in J} C_j^{SEPT}}{\sum_{j\in J} C_j^{Opt} p_j}] = \mathcal{O}(\log(n))$.
2. $P[\frac{\sum_{j\in J} C_j^{SEPT}}{\sum_{j\in J} C_j^{Opt} p_j}] = \mathcal{O}(\log(n))] \geq 1 - o(1)$.

Beweis. Die Prozesszeit eines Jobs $j \in J$ sei mit p_j bezeichnet. Wir sortieren die Jobs nach der SEPT–Regel und nummerieren entsprechend um. Dann kann die Menge J der Jobs geschrieben werden als $J = \{1, \ldots, n\}$, wobei jeder Job $j \in J$ eine erwartete Prozesszeit $E[p_j] = \frac{1}{\lambda_j}$ habe. Dabei gilt $E[p_i] \leq E[p_j]$ für zwei Jobs i und j mit $i < j$. Seien ferner w_j^{SEPT} und w_j^{Opt} die Vielfachheit, mit der Job j in der Lösung unter der SEPT–Regel bzw. in der optimalen Offline–Lösung gezählt wird. Dann gilt

$$\sum_{j\in J} C_j^{SEPT} = \sum_{j=1}^{n} (n+1-j)p_j = \sum_{j=1}^{n} w_j^{SEPT} p_j,$$

und

$$\sum_{j\in J} C_j^{Opt} = \sum_{j=1}^{n} (n+1-j)\hat{p}_j = \sum_{j=1}^{n} w_j^{Opt} p_j,$$

wobei $\hat{p}_j$ die Prozesszeit des j–ten kürzesten Jobs bezeichne. Insbesondere gilt also $w_j^{SEPT} = (n+1-j)$, während w_j^{Opt} nicht a priori bestimmt werden kann. Für den Beweis des Theorems betrachten wir zwei Fälle. Im ersten Fall, den wir als *Ereignis* $\mathcal{A}$ bezeichnen, haben viele Jobs eine hinreichend große Prozesszeit. Für diesen Fall zeigen wir (cf. Lemma 4.8)

$$\frac{\sum_{j\in J} C_j^{SEPT}}{\sum_{j\in J} C_j^{Opt}} = \mathcal{O}(\log(n)).$$

Im anderen Fall, dem *Ereignis* $\bar{\mathcal{A}}$, ist obiger Quotient beim n–Job Problem dagegen nur durch n beschränkt, allerdings wird dieses Ereignis auch nur mit einer Wahrscheinlichkeit von $\frac{1}{n}$ auftreten.
Um die Ereignisse $\mathcal{A}$ und $\bar{\mathcal{A}}$ spezifizieren zu können, sind die folgenden Vorüberlegungen notwendig: Wir betrachten einen Job j und bestimmen einen Wert m_j, den die Prozesszeit p_j von j mit hoher Wahrscheinlichkeit nicht überschreiten wird. Ferner nennen wir einen Job j *groß*, wenn für seine Prozesszeit p_j gilt $p_j \geq E[\mathbf{p_j}] \cdot (-\ln(p))$ mit $p = \frac{1}{2}$. Ansonsten ist der Job *klein*. Ferner wollen wir erreichen, dass bis auf die letzten $\mathcal{O}(\log(n))$ Jobs für jeden Job j mindestens ein Viertel der ihm nachfolgenden Jobs *groß* sind. Wir setzten $k = 2 \cdot \log(n)$ und $m_j = \frac{-\ln(p)\cdot k}{\lambda_j} = E[p_j] \cdot (-\ln(p)) \cdot k$.
Wir definieren das Ereignis $\mathcal{A}$ wie folgt:

Definition 4.6
Ereignis $\mathcal{A}$ tritt ein, wenn nach Auslosen der Prozesszeiten gilt:

(1) Alle Jobs i haben eine Prozesszeit $p_i < m_i$.

(2) Für jeden Job $j \in \{1, \ldots n - 16 \cdot \ln(n)\}$ sind mindestens 1/4 der nachfolgenden Jobs groß.

Lemma 4.7
Die Wahrscheinlichkeit $P[\mathcal{A}]$ für Ereignis $\mathcal{A}$ beträgt mindestens $1 - \mathcal{O}(\frac{1}{n})$.

Beweis. Da jeder Jobs j exponentialverteilt ist mit Parameter λ_j, gilt für die Prozesszeit p_j von j:

$$\begin{aligned} P[p_j < m_j] &= 1 - e^{\ln(p)\cdot k} \\ &= 1 - \frac{1}{2^k} \\ &= 1 - \frac{1}{n^2}. \end{aligned}$$

Die Wahrscheinlichkeit, dass mindestens ein Job j der n Jobs eine Prozesszeit $p_j > m_j$ hat, beträgt damit höchstens $1/n$.
Zum Abschätzen der Wahrscheinlichkeit von Eigenschaft (2) beschränken wir die Prozesszeiten nach unten:

$$\begin{aligned} P[p_j > m_j/k] &= e^{\ln(p)} \\ &= \frac{1}{2}. \end{aligned}$$

Die Wahrscheinlichkeit für einen Job *groß* zu sein, beträgt damit $\frac{1}{2}$. Nun betrachten wir alle Jobs i bis auf die letzten $16 \cdot \ln(n)$ Jobs. Für ein solches i existiert also eine Zahl $l(i) \geq 16 \cdot \ln(n)$ von Jobs mit höherem Index. Sei also i ein Job

mit $i \in \{1, \ldots, n - 16\ln(n)\}$ und bezeichne $l(i) \geq 16\ln(n)$ die Anzahl der Jobs mit höherem Index. Dann ist die Wahrscheinlichkeit p, dass mindestens $l(i)/4$ dieser Jobs *groß* sind, größer als $1 - \frac{1}{n^2}$ und die Wahrscheinlichkeit, dass mindestens einer der Jobs $i \in \{1, \ldots, n - 16\ln(n)\}$ weniger als $l(i)/4$ *große* Nachfolger hat, höchstens $\frac{1}{n}$. Zum Beweis dazu sei X_i die Zufallsvariable, die die Anzahl der *großen* Nachfolger von Job i zählt. Im Erwartungswert sind $l(i)/4$ Nachfolgerjobs von j *groß*. Die Ungleichung von Chernoff ergibt:

$$\begin{aligned} P[X_i < (1-1/2)\frac{16 \cdot \ln(n)}{2}] &\leq P[X_i < (1-1/2)\frac{l(i)}{2}] \\ &\leq e^{-\frac{l(i)}{8}} \\ &\leq e^{-\frac{16 \cdot \ln(n)}{8}} = \mathcal{O}(\frac{1}{n^2}). \end{aligned}$$

Wenn wir alle Jobs $j \in \{1, \ldots, n - 16 \cdot \ln(n)\}$ simultan betrachten, so ist die Wahrscheinlichkeit, dass mindestens ein Job i weniger als $l(i)/4$ *große* Nachfolger hat, beschränkt durch $(n - 16 \cdot \ln(n)) \cdot \mathcal{O}(\frac{1}{n^2}) < n \cdot \mathcal{O}(\frac{1}{n^2}) = \mathcal{O}(\frac{1}{n})$.
Die Wahrscheinlichkeit, dass die Eigenschaft (1) nicht eintritt, beträgt höchstens $1/n$ und die Wahrscheinlichkeit, dass Eigenschaft (2) nicht eintritt, beträgt ebenfalls $\mathcal{O}(1/n)$. Die Wahrscheinlichkeit, dass Ereignis $\mathcal{A}$ nicht eintritt, beträgt also $\mathcal{O}(1/n)$. □

Lemma 4.8
Unter der Voraussetzung, dass Ereignis $\mathcal{A}$ eingetreten ist, gilt

$$\frac{\sum_j C_j^{SEPT}}{\sum_j w_j C_j^{Opt}} = \mathcal{O}(\log(n)).$$

Beweis. Wir setzten voraus, dass die Prozesszeiten der Jobs gegeben sind und Ereignis $\mathcal{A}$ eingetreten ist. Wir zerlegen die Menge der Jobs J in drei Teilmengen *Klein*, *Groß* und *Rest*. Die Menge *Rest* bezeichnet die hinteren $16 \cdot \ln(n)$ Jobs, d. h. $Rest = \{n - 16 \cdot \ln(n), \ldots, n\}$. Die Menge *Klein* setzt sich aus den verbleibenden *kleinen* Jobs zusammen und die Menge *Groß* bezeichnet alle übrigen Jobs, also diejenigen Jobs $j \in J - Rest$ mit $p_j \geq m_j/k$.
Es gelten die folgenden drei Lemmata:

Lemma 4.9
Für jeden Job $j \in Rest$ gilt

$$w_j^{Opt} \geq \frac{1}{16 \cdot \ln(n)} w_j^{SEPT}.$$

Beweis. Jeder Job $j \in Rest$ wird unter der SEPT–Regel höchstens $16 \cdot \ln(n)$ mal gezählt. In einer optimalen Offline–Lösung wird der gleiche Job mindestens einmal gezählt. □

Lemma 4.10
Für jeden Job $j \in Klein$ gilt

$$w_j^{Opt} \geq \frac{w_j^{Sept}}{4}.$$

Beweis. Da *Klein* nicht die Jobs aus *Rest* enthält, sind nach Bedingung (2) mindestens ein Viertel aller den Job j nachfolgenden Jobs i *groß*. Da ein solcher Job i unter der SEPT–Regel nach Job j angeordnet ist, gilt $\lambda_i > \lambda_j$. Folglich haben wir

$$p_j < m_j/k = \frac{-\ln(p)}{\lambda_j} \leq \frac{-\ln(p)}{\lambda_i} = m_i/k \leq p_i.$$

Damit ist aber p_j in einer optimalen Lösung vor allen nachfolgenden großen Jobs angeordnet, also vor mindestens einem Viertel der nachfolgenden Jobs. Daraus folgt die Behauptung. □

Lemma 4.11
Es gilt:

$$\sum_{j \in Groß} w_j^{Opt} p_j \geq \sum_{j \in Groß} \frac{w_j^{SEPT}}{4} m_j/k.$$

Beweis. Bezeichne w_j^{Opt} die Anzahl, mit der Job j in der optimalen Lösung gezählt wird. Wir betrachten die Menge $\{w_j^{Opt} \mid j \in Groß\}$. Diese lässt sich bijektiv auf die Menge $\{\bar{w}_j \mid j \in Groß\}$ abbilden, wobei die Werte $\{\bar{w}_j \mid j \in Groß\}$ aus den Werten $\{w_j^{Opt} \mid j \in Groß\}$ durch Umsortieren nach nicht aufsteigenden Werten entstehen. Dann folgt

$$\sum_{j \in Groß} w_j^{Opt} \cdot m_j \geq \sum_{j \in Groß} \bar{w}_j \cdot m_j. \tag{4.13}$$

Dies sieht man durch ein einfaches Austauschargument. Nach Konstruktion gilt $m_i \leq m_j$ für $i < j$. Außerdem gilt dann $\bar{w}_i \geq \bar{w}_j$. Wir betrachten die Folge $w_1^{Opt}, \ldots, w_n^{Opt}$ und nehmen an, dass es zwei Indizes i und j gibt, mit $i < j$, so dass $w_i^{Opt} \leq w_j^{Opt}$. Nun betrachten wir die Differenz

$$\Delta_{ij} = \sum_{k=1, k \neq i,j}^{n} w_k^{Opt} m_k + w_i^{Opt} m_i + w_j^{Opt} m_j - \Big(\sum_{k=1, k \neq i,j}^{n} w_k^{Opt} m_k + w_i^{Opt} m_j + w_j^{Opt} m_i \Big).$$

Es gilt

$$\begin{aligned} \Delta_{ij} &= w_i^{Opt} m_i + w_j^{Opt} m_j - (w_i^{Opt} m_j + w_j^{Opt} m_i) \\ &= w_i^{Opt}(m_i - m_j) + w_j^{Opt}(m_j - m_i) \\ &= (m_j - m_i)(w_j^{Opt} - w_i^{Opt}) \geq 0. \end{aligned}$$

Austauschen der Koeffizienten w_i^{Opt} und w_j^{Opt} führt also zu einer kleineren Summe und durch induktives Anwenden dieser Argumentation folgt Ungleichung (4.13). Ferner gilt nach analogen Argumenten, wie in Lemma 4.10

$$\bar{w}_j \geq \frac{w_j^{SEPT}}{4},$$

für $j \in Groß$: Dazu überlege man sich als erstes, dass $\bar{w}_j = w_j^{SEPT}$ gelten würde, wenn alle in der optimalen Lösung nachfolgenden Jobs größer als der $\bar{w}_j$ zugeordnete Job sein würden. Da andererseits $j \in Groß$, ist nach Konstruktion aber zumindest ein Viertel aller nachfolgenden Jobs größer, also kann $\bar{w}_j$ höchstens um einen Faktor $\frac{1}{4}$ kleiner als w_j^{SEPT} werden.
Wir verwenden diese Eigenschaften wie folgt:

$$\begin{aligned}
\sum_{j \in Groß} w_j^{Opt} p_j &\geq \sum_{j \in Groß} w_j^{Opt} \cdot \frac{m_j}{k} \\
&= \frac{1}{k} \sum_{j \in Groß} w_j^{Opt} \cdot m_j \\
&\geq \frac{1}{k} \sum_{j \in Groß} \bar{w}_j \cdot m_j \\
&\geq \frac{1}{k} \sum_{j \in Groß} \bar{w}_j \cdot p_j \\
&\geq \frac{1}{4 \cdot k} \sum_{j \in Groß} w_j^{SEPT} \cdot p_j
\end{aligned}$$

nach Definition von $\mathcal{A}$. □

Wir werden diese drei Lemmata anwenden, um Lemma 4.8 zu beweisen. Wir haben

$$\begin{aligned}
\sum_{j=1}^{n} C_j^{Opt} &= \sum_{j=1}^{n} w_j^{Opt} p_j \\
&\geq \min\{\frac{1}{16 \cdot \ln(n)}, \frac{1}{4}, \frac{1}{4 \cdot k}\} \sum_{j=1}^{n} w_j^{SEPT} p_j \\
&= \min\{\frac{1}{16 \cdot \ln(n)}, \frac{1}{4}, \frac{1}{4 \cdot k}\} \sum_{j=1}^{n} w_j^{SEPT} p_j.
\end{aligned}$$

Daher gilt also

$$\mathcal{O}(\log(n)) \sum_{j=1}^{n} C_j^{Opt} = \sum_{j=1}^{n} w_j^{SEPT} p_j$$

und Lemma 4.8 ist bewiesen. □

Nach Lemma 4.7 beträgt die Wahrscheinlichkeit für das Eintreten von Ereignis $\mathcal{A}$ mindestens $1 - \mathcal{O}(\frac{1}{n})$. Zusammen mit Lemma 4.8 ist damit die zweite Aussage von Theorem 4.5 bewiesen.
Im folgenden betrachten wir den Fall, wenn Ereignis $\mathcal{A}$ nicht vorausgesetzt wird.

Lemma 4.12
Sei n die Anzahl der Jobs. Dann gilt

$$\frac{\sum C_j^{SEPT}}{\sum C_j^{Opt}} \leq n.$$

Beweis. Jeder Job wird in der unter SEPT erstellten Lösung und in der optimalen Lösung mindestens 1 mal und maximal n mal gezählt. Daher gilt für jeden Job j, dass $w_j^{SEPT} \geq \frac{1}{n} w_j^{Opt}$. Somit folgt

$$\frac{\sum C_j^{SEPT}}{\sum C_j^{Opt}} = \frac{\sum w_j^{SEPT} \cdot p_j}{\sum w_j^{Opt} \cdot p_j} \leq \frac{\sum w_j^{SEPT} \cdot p_j}{\sum \frac{1}{n} w_j^{SEPT} \cdot p_j} \leq n.$$

□

Nun verbinden wir Lemma 4.12 und Lemma 4.7, um Theorem 4.5 zu erhalten. Es gilt

$$\begin{aligned} E[\frac{\sum_j w_j C_j^{SEPT}}{\sum_j w_j C_j^{Opt}}] &= E[\frac{\sum_j w_j C_j^{SEPT}}{\sum_j w_j C_j^{Opt}} \mid \mathcal{A}] \cdot \mathcal{P}[\mathcal{A}] \\ &\quad + E[\frac{\sum_j w_j C_j^{SEPT}}{\sum_j w_j C_j^{Opt}} \mid \bar{\mathcal{A}}] \cdot P[\bar{\mathcal{A}}] \\ &\leq \mathcal{O}(\log(n)) \cdot (1 - \mathcal{O}(\frac{1}{n})) + n \cdot \mathcal{O}(\frac{1}{n}) = \mathcal{O}(\log(n)). \end{aligned}$$

Dies ist aber genau die gewünschte Schranke. □

Teil II

Scheduling in der Industrieanwendung

KAPITEL 5

Einführung

Im zweiten Teil dieser Arbeit betrachten wir den Bereich Scheduling und Produktionsplanung anhand einer konkreten Anwendung aus der Praxis. Das zugrundeliegende Beispiel ist ein fiktives mittelständisches Unternehmen aus der Brauindustrie, für das die Produktionsplanung des Unternehmens automatisiert werden soll. In einem von der BMBF geförderten Kooperationsprojekt mit den Firmen *Gesellschaft für Informationstechnik Weihenstephan mbH* in Freising–Weihenstephan und der *Werum Datenverarbeitungssysteme GmbH* in Lüneburg wurde ein PPS–System zum integrierten Einsatz in einem derartigen Brauereiunternehmen entwickelt und implementiert. Das Beispiel Brauindustrie ist insofern zur Darstellung der Problematik "Produktionsplanung" geeignet, als es auf der einen Seite im Produktionsbereich der Bierherstellung typische Vorgänge der *Prozessindustrie*, wie etwa der Chemie-, Nahrungs- und Grundstoffindustrie, enthält und auf der anderen Seite die Bierabfüllung typische Charakteristika der *Fertigungsindustrie* aufweist. Damit repräsentiert die Planung in der Brau- und Getränkeindustrie eine Vielzahl typischer und interessanter Fragestellungen, die auch in verschiedenen anderen Anwendungsgebieten der industriellen Produktionsplanung zu lösen sind. So hat es sich auch konkret ergeben, dass Teile der in diesem Projekt entwickelten Planungsmodule als integrierter Bestandteil des Planungssystems *Pas–Plan* der Firma Werum GmbH bei Unternehmen der Chemie und der Genussmittelindustrie bereits im Einsatz sind.
Im Folgenden sei aber als Beispiel das Unternehmen der Brauindustrie gegeben. Ein Übersichtsartikel über die Produktionsplanung in der Brau- und Getränkeindustrie ist [BS96]. Bei der Bierherstellung eines solchen Unternehmens ist die Produktion, wie allgemein in der Prozessindustrie üblich, chargenorientiert. Sie ist durch die folgenden Merkmale gekennzeichnet:

- Es wird diskontinuierlich gefertigt und die Chargen werden häufig verschnitten und mehrstufig ineinander überführt.
- Die Produktionsanlagen können verschiedene Chargen zur gleichen Zeit auf-

nehmen und sind nicht für bestimmte Produkte fixiert.

- In den verschiedenen Produktionsschritten fallen Produkte, Nebenprodukte und Reststoffe an.
- Die Prozessteuerung erfolgt mit variablen Parametern, die von Charge zu Charge unterschiedlich sind. Die Parameter sind durch externen Bedingungen determiniert, wie die Beschaffenheit der Einsatzstoffe oder die Ergebnisse biochemischer Umwandlungsprozesse.
- Planvorgaben werden häufig kurzfristig geändert und können keinen hohen Detaillierungsgrad haben, da die Prozesse verschiedenen Indeterminismen unterliegen. So können beispielsweise benötigte Belegungszeiten in den Gär- und Lagertanks nicht von vorneherein festgelegt werden.
- Im Zuge der Qualitätssicherung müssen zu jeder Charge alle Produktionsistdaten, gleichgültig ob sie von einem Prozessleitsystem stammen oder vom Anlagenfahrer erfasst werden, in einer einheitlichen Weise verarbeitet werden und für Auswertungen aller Art zur Verfügung stehen.

Elemente der Fertigungsindustrie finden sich vor allem in der Bierabfüllung. Die Planung in der Bierabfüllung ist absatzorientiert. Ziel ist, alle eingegangenen Kundenaufträge und Absatzvorgaben erfüllen zu können und gleichzeitig einen niedrigen Lagerbestand zu halten. Bei einem zu niedrigen Lagerbestand besteht dabei die Gefahr, dass Absätze möglicherweise nicht mehr befriedigt werden können. Wichtige Aspekte in der Bierabfüllung sind:

- Die Abfüllung ist ein einstufiger Vorgang, bei dem kontinuierlich gefertigt wird.
- Die Produktfolgen sind sorgfältig zu planen um Reinigungsvorgänge und Umrüstungen möglichst zu vermeiden.
- Die Aggregate der Abfülllinien unterliegen häufig Störungen, so dass ein ständiges Neuplanen und Aktualisieren der Abfüllfolgen erforderlich wird.
- Eilaufträge müssen möglicherweise in die laufende Abfüllung eingefügt werden.
- Aufgrund von Schichtkalendern und Wartungs- und Reinigungsarbeiten stehen Abfüllanlagen zu bestimmten Zeiten nicht zur Verfügung.

Die genannten Charakteristika der Produktionsbereiche der Brauindustrie vermitteln einen ersten Eindruck von der Vielschichtigkeit der zu bewältigenden Planungsaufgaben und der ihnen zugrundeliegenden Schedulingprobleme. Bereits hier wird der Unterschied zu den Modellen aus der Theorie des Scheduling deutlich, die von einem vereinfachten Szenario mit einer Folge von fest determinierten

oder stochastischen Jobs ausgehen, die auf einer gegebenen Anzahl von Maschinen bearbeitet werden. Für ein genaueres Verständnis des betrachteten Aufgabenfeldes ist aber zunächst ein Verständnis der betrieblichen Prozesse des betrachteten Unternehmens notwendig, denn nur dann kann seine Planung in adäquater Weise automatisiert werden. Wir geben daher im folgenden Kapitel ein abstraktes Betriebsmodell der betrachteten Beispielbrauerei. Dieses Betriebsmodell spiegelt die zentralen produktionstechnischen und organisatorischen Rahmenbedingungen eines Brauereiunternehmens wider. Die Darstellung abstrahiert aber den Produktionsprozess auf die für die Produktionsplanung relevanten Faktoren. Die zweite im Zuge der Realisierung zu leistende Vorarbeit ist eine detaillierte Spezifikation der zu lösenden Teilaufgaben der Produktionsplanung und ihrer Interaktion. Die verschiedenen Teilaufgaben der Produktionsplanung, wie etwa die mittel- bis langfristige Kapazitätsplanung und die kurzfristig orientierte Belegungsplanung, stehen in enger Verbindung zueinander und können nur innerhalb eines mehrstufigen hierarchischen Gesamtsystem bearbeitet werden, das einen regen Informationsaustausch zwischen den Planungsstufen gestattet. Ein derartiges Modell der Produktionsplanung stellen wir in dieser Arbeit vor.
Die Umsetzung der zuvor erarbeiteten Modelle in ein System zur Produktionsplanung setzt sich aus einer logischen und einer technischen Komponente zusammen. Resultat der logischen Umsetzung ist ein DV–Konzept, das die Planungsaufgaben in einzelne Teilprobleme zerlegt, den Informationsstrom zwischen den Teilproblemen beschreibt und geeignete Datenstrukturen und Algorithmen zur Lösung der Teilprobleme gibt. Hierbei ist zu beachten, dass sich ein PPS–System für die Brauindustrie in seiner Architektur aber auch in der einsetzbaren Algorithmik zum Teil deutlich von Standardsystemen für die Prozessindustrie unterscheidet. So erfordert zum Beispiel die strikte Trennung von Bierherstellung und Bierabfüllung eine mehrstufige Planung, bei der die Planergebnisse des einen Produktionsbereichs die Ausgangsdaten des zweiten Bereiches bilden, bei der aber beide Bereiche voneinander unabhängig Planen können. Ebenso sind zur Planung in Brauereien Datenstrukturen zu verwenden, die in Standardsystemen zur Zeit noch nicht realisiert sind. Zum Beispiel muss es aufgrund verschiedener produktionstechnischer Rahmenbedingungen möglich werden, dass mehrere Chargen auf einer einzigen Produktionseinheit zur gleichen Zeit abgearbeitet werden können. Als zweites Beispiel seien die Chargengrößen genannt, die teilweise fest gegeben sind (Sudhaus), teilweise von der aktuellen Belegung abhängen (Filterkeller) und/oder produktabhängig gewählt werden. Bei der technischen Umsetzung sind darüber hinaus Fragestellungen der Integration der notwendigen Betriebsdaten, also aller Stamm- und Bewegungsdaten, in das Planungssytem und der Schnittstellen zu Fremdsoftwaresystemen zu lösen.

Tabelle 5.1: Produktanteil in %

Pils	(65 %)
Hell	(30 %)
Alkoholfrei	(5 %)

Tabelle 5.2: Saisonale Absatzverteilung in %

Jan.	Feb.	März	April	Mai	Juni
6,5	5,5	6,3	7,4	8,8	9,5
Juli	Aug.	Sept.	Okt.	Nov.	Dez.
10,8	10,2	9,8	8,0	7,4	9,8

5.1 Beispielplanung in einer fiktiven Brauerei

Bevor wie eine systematische Einführung in die Problematik der Produktionsplanung geben, sollen durch das nachfolgende Beispiel die Grundfunktionen der Planung anhand der Abfüllung eines Modellbetriebes einer Brauerei illustriert werden. Das Beispiel basiert auf einer im Rahmen dieser Arbeit durchgeführten Erhebung bei verschiedenen mitellständischen Unternehmen und einigen Großkonzernen aus der Brauindustrie. Ebenso sind bei diesen Erhebungen vor Ort die in Kapitel 6 dargestellten organisatorischen und produktionstechnischen Abläufe eines Brauereibetriebes erfasst worden.
Der hier betrachtete Modellbetrieb ist eine mittelständische Brauerei mit einem Jahresausstoß von 1000000 hl. Es wird für das laufende Geschäftsjahr eine Steigerung von $1,5$ % erwartet. Das betrachtete Unternehmen verkauft drei Biersorten mit der in Tabelle 5.1 dargestellten prozentualen Verteilung:
Abgefüllt wird zu 36 % in $0,33$-l-Vichy Flaschen und zu 64 % in $0,5$-l-NRW Flaschen. Die saisonale Absatzverteilung ist in Tabelle 5.2 dargestellt. Ziel der Produktion ist, dass für die Sorten Pils und Hell zu jeder Zeit ein Lagerbestand vorliegt, der dem Absatz von zwei Folgetagen genügt. Für alkoholfreies Bier soll der Bestand dem Absatz von 4 Tagen genügen. Die Brauerei besitzt zwei Abfüllanlagen mit einer Nennausbringung von 50000 Fl/h bzw. 45000 Fl/h. Die Ausnutzungsgrade betragen 70 % bzw. 80 %. Gearbeitet wird im Zweischichtbetrieb. Eine vereinfachte Datenbasis für ein Planungssystem der Brauerei könnte

wie folgt aussehen:

Stammdaten	• Produkte (Bier im Drucktank, abgefülltes Bier) Bei Automatisierung: Rezeptur, Stückliste, Produkt-/Anlagenzuordnung • Materialien (Produkte, Leergut, Leim) • Produktionseinheiten (Abfüllanlagen) • Kalender (Arbeitszeiten, Feiertage, Schichten)
Bewegungsdaten	• Absätze und Kundenaufträge • Bestände • Produktion (Sollproduktion und Istproduktion)

Im Folgenden geben wir als Beispiel eine mögliche Produkttabelle (Tabelle 5.3) und die zugehörige Produktstückliste (Tabelle 5.4) des Produktes Pils 0, 5-l-NRW dieser Tabelle 5.3. Diese Tabellen würden in gleicher Form in einer relationalen Datenbank des Planungssystems gespeichert werden.

Tabelle 5.3: Produkttabelle

Produkte	Bezeichnung	Einheit	Reichweite
Pils	0,33-l-Vichy	Kästen	2 Tage
Pils	0,5-l-NRW	Kästen	2 Tage
Hell	0,33-l-Vichy	Kästen	2 Tage
Hell	0,5-l-NRW	Kästen	2 Tage
Alkoholfrei	0,33-l-Vichy	Kästen	4 Tage
Alkoholfrei	0,5-l-NRW	Kästen	4 Tage

Tabelle 5.4: Stückliste

Position	Bezeichnung	Material	Menge bzgl. Normcharge	Einheit
1	Biersorte	Pils	100	hl
2	Gebinde	0,5-l-NRW	200	Fl
3	Etikett	Etikett 0,5 l NRW	200	Stück

Die Herstellvorschrift oder Rezeptur besteht aus dem einzigen Arbeitsgang *Füllen*. Die Chargengröße ist variabel. Die Durchlaufzeit, also die auf eine Normcharge bezogene Fülldauer, berechnet sich in Abhängigkeit von der Nennausbringung der gewählten Abfüllanlage. In der folgenden Tabelle ist eine vereinfachte Parametrisierung der Daten der Abfüllanlagen gegeben:

Tabelle 5.5: Maschinentabelle

Bezeichnung	Nennausbringung	Schichten/Tag	Ausnutzungsgrad	Zul. Gebinde
Abfüllanlage 1	50000 Fl/h	2	70 %	0,5-l-NRW, 0,33-l-Vichy
Abfüllanlage 2	45000 Fl/h	2	80 %	0,5-l-NRW

Für die Produktionsplanung der hier vorgestellten Beispielbrauerei kann der Grundablauf in der folgenden Weise durchgeführt werden.

Grundablauf der Produktionsplanung:

1. Ausgangspunkt:

Die geplanten Verkaufsmengen für den Planungszeitraum und die saisonale Absatzverteilung sind bekannt. Ebenso sind die Feiertage im Planungszeitraum bekannt. Folglich kann die erwartete Verkaufsmenge der Produkte für die Folgemonate bestimmt werden. Dies ist in Tabelle 5.6 dargestellt.

2. Bestimmen des Produktionsbedarfs:

Die Planung erfolgt bestandsorientiert. Ziel ist, stets einen durch die Absätze und die Reichweite bestimmten Lagerbestand zu halten. Der Produktionsbedarf, also die Menge der Produkte, die in den einzelnen Planungsperioden herzustellen ist, ermittelt sich aus den aktuellen Beständen im Lager, der gerade laufenden Produktion und der erwarteten Verkaufsmenge. Die aktuellen Lagerbestände und die aktuelle Produktion verringern den Produktionsbedarf, während der laufende Verkauf den Produktionsbedarf erhöht.

3. Abgleich des Produktionsbedarfs mit der Linienkapazität:

Nachdem der Produktionsbedarf für die einzelnen Produkte ermittelt worden ist, muss festgestellt werden, wann und auf welchen Anlagen die Produktion erfolgen kann. Dazu muss die erforderliche Anlagenkapazität mit der tatsächlichen freien Anlagenkapazität verglichen werden. Die Grobplanung vermittelt dazu anhand von Durchschnittszahlen einen groben Überblick, die Feinplanung erzeugt tatsächliche Belegungen und kann damit exakt bestimmen, ob die verfügbare Anlagenkapazität zur Deckung des Produktionsbedarf ausreicht.

Grobplanung:

- Der Produktionsbedarf wird den vorhandenen Linienkapazitäten gegenübergestellt. Bei Kapazitätenmangel wird die Produktion vorverlagert oder es werden zusätzliche Schichten und Überstunden eingeführt.

- Rüstzeiten und Wirkungsgrade werden pauschal berücksichtigt.

Feinplanung:

- Die Produktionsanforderungen werden zu Produktionslosen oder Chargen zusammengefasst und diese Lose werden eingeplant.
- Die Feinplanung erfolgt unter Berücksichtigung verschiedener Restriktionen. Eine Beispiel wäre, dass 0,33-l-Gebinde ausschließlich auf Abfüllanlage 1 gefahren werden kann oder dass Abfüllanlagen unterschiedliche Nennausbringungen aufweisen.
- Die Feinplanung versucht, günstige Zeiten und Produktfolgen zu bilden.
- Verschiedenden Zielfunktionen wird genügt, beispielsweise dem Minimieren von Umstellzeiten und das Vermeiden von Unterbeständen im Vollgutlager.

4. Sekundärbedarfsauflösung:

- Die zum Umsetzten des Plans benötigten Einsatzmaterialien, wie etwa Biersorte, Gebinde und Etiketten werden in Menge und Zeit bestimmt.

Tabelle 5.6 zeigt als Ergebnis von Schritt 1 die nach Produkten aufgeschlüsselte Absatztabelle.

Tabelle 5.6: Absatztabelle nach Produkten in *hl*

Produkt	Jan.	Feb.	März	April	Mai	Juni
Pils 0,33-l-Vichy	15210	12870	14742	17316	20592	22230
Pils 0,5-l-NRW	27040	22880	26208	30784	36608	39520
Hell 0,33-l-Vichy	7020	6804	6804	7992	9504	10260
Hell 0,5-l-NRW	12480	12096	12096	14208	16896	18240
Alkfr. 0,33-l-Vichy	1170	1134	1134	1332	1584	1710
Alkfr. 0,5-l-NRW	2080	2016	2016	2368	2816	3040
Produkt	Juli	Aug.	Sept.	Okt.	Nov.	Dez.
Pils 0,33-l-Vichy	25272	23868	22932	18720	17316	22932
Pils 0,5-l-NRW	44928	42432	40768	33280	30784	40768
Hell 0,33-l-Vichy	11664	11016	10584	8640	7992	10584
Hell 0,5-l-NRW	20736	19584	18816	15360	14208	18816
Alkfr. 0,33-l-Vichy	1944	1836	1764	1440	1332	1764
Alkfr. 0,5-l-NRW	3456	3264	3136	2560	2368	3136

Die nachfolgenden Abbildungen 5.1 bis 5.4 illustrieren die Schritte 2 – 4 der Planung. Aufgrund von Unterbeständen der Produkte Hell und Alkoholfrei ergibt sich ein Produktionsbedarf. Die zugehörige Menge soll eingeplant werden. Dazu

wird sie den verfügbaren Anlagenkapazitäten gegenübergestellt und eine günstige Abfüllanlage (hier Anlage 2) ausgewählt. Die freie Kapazität der gewählten Anlage verringert sich und der Produktbestand und die Produktionsmenge des Produktes wird erhöht.
Diese Abbildungen geben zugleich ein Beispiel für den Aufgabenbereich *Visualisierung* in einem Planungssystem. Die Grafiken ermöglichen einen schnellen Überblick über die Produktion und die Bestandssituation im Lager. Kritische Momente, wie der Unterbestand am zweiten Tag der Kalenderwoche 33 können auf einen Blick identifiziert werden.

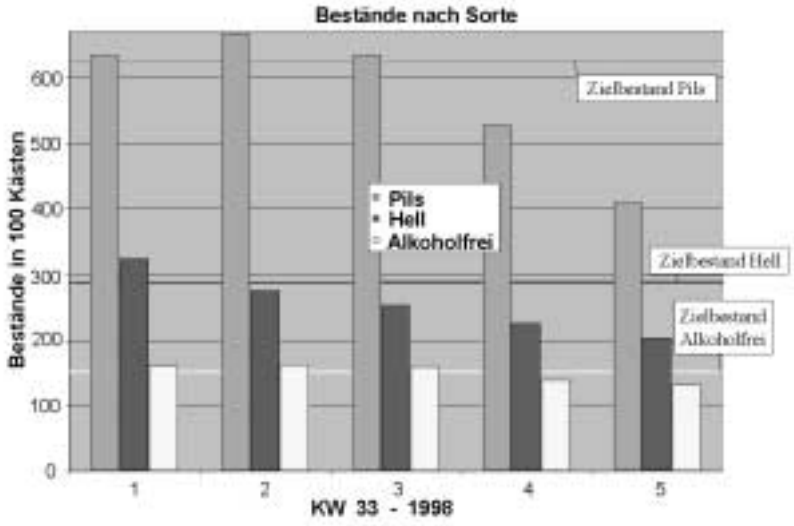


Abbildung 5.1: Grafische Darstellung der Planung (1)

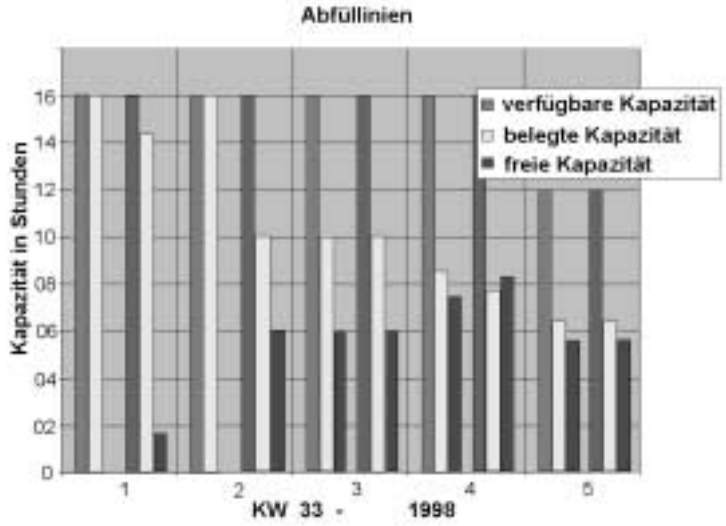


Abbildung 5.2: Grafische Darstellung der Planung (2)

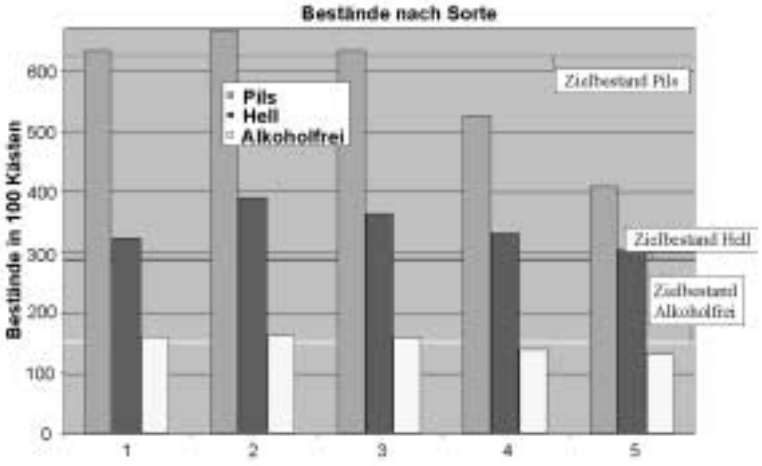


Abbildung 5.3: Grafische Darstellung der Planung (3)

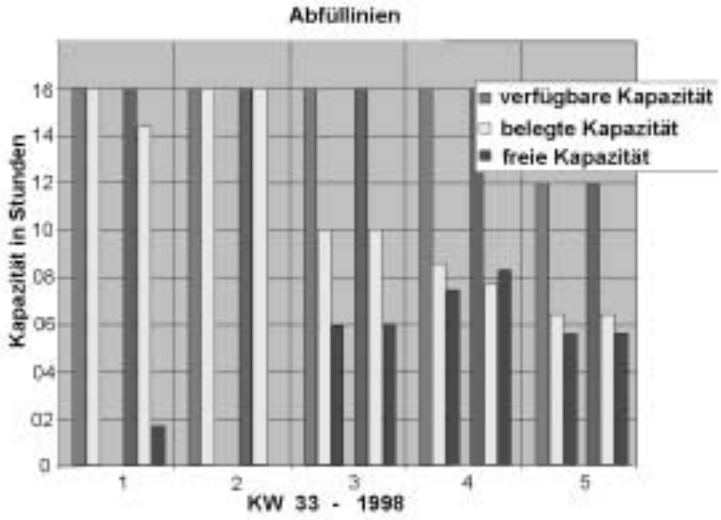


Abbildung 5.4: Grafische Darstellung der Planung (4)

Nachdem die Produktionsplanung und -steuerung anhand des obigen Beispiels informal vorgestellt wurde, soll im nächsten Abschnitt eine genaue Definition und Analyse der Planung erfolgen.

5.2 Produktionsplanung und -steuerung: Problemdefinition

Die Aufgabe der Produktionsplanung und -steuerung (PPS) ist, alle mit der Produktion verknüpften betrieblichen Vorgänge zu planen, steuern und rückzumelden. Organisation und Aufgaben der PPS greifen dabei tief in betriebliche Strukturen ein und wirken direkt auf die heutigen Erfolgsfaktoren wie Lagerbestände, Durchlaufzeiten, Kapazitätsauslastung, Transparenz und Flexibilität. Der Gebrauch des Begriffes Produktionsplanung und -steuerung ist nicht eindeutig.
Der *VDI* (Verband deutscher Ingenieure) definiert die Produktionsplanung als das "Systematische Suchen und Festlegen von Zielen für die Produktion, Vorbereiten von Produktionsaufgaben und Festlegung des Ablaufes zum Erreichen dieser Ziele" [Hr83].
Das *IPA* (Frauhenhofer–Institut für Produktionstechnik und Automatisierung) versteht unter der Produktionsplanung und -steuerung die "Abstimmung der technisch–organisatorischen Funktionsbereiche entsprechend der betrieblichen Ziele, bezogen auf die technischen Funktionen Fertigen, Lagern, Montieren und Transportieren [AUK88].

Scheer definiert die PPS als die "primär betriebswirtschaftlich–planerische Komponente des CIM–Modells, welches die durch den Auftragsfluss gesteuerten Funktionen der Kundenauftragsabwicklung über Bedarfsplanung, Zeitwirtschaft, Fertigungssteuerung, Betriebsdatenerfassung bis zum Versand umfasst" [Sch94].
Hinsichtlich Aufbau und Abwicklung der PPS existiert ein undurchschaubares Dickicht an Definitionen, Ansätzen und Meinungen. Insbesondere in der Betriebswirtschaftslehre ist eine Vielzahl von Modellen entwickelt worden, die die informatorischen und organisatorischen Abläufe der PPS zu beschreiben suchen. Da eine Beschreibung des State–of–the–Art der Produktionsplanung über den Rahmen dieser hinausgehen würde, verweisen wir auf die Standardliteratur [FFG94] [Hac89] [Ker94] [Kur93] [Zäp93]. Zudem nehmen wir eine rein (DV)–technisch orientierte Perspektive der Produktionsplanung an, für die die gültigen Modelle der Betriebswirtschaft nur bedingt greifen. Eine Beschreibung der Planung und Logistik aus der Sicht der Betriebswirtschaftslehre findet sich in [Gut76] und [Wöh90].
Für die folgenden Betrachtungen und die in dieser Arbeit geleisteten Untersuchungen genügt es jedoch, die Produktionplanung und -steuerung auf die folgenden Funktionen zu beschränken:

- Absatz- und Produktionsprogrammplanung
- Sekundärbedarfsplanung
- Kapazitätsplanung
- Losgrößen- und Belegungsplanung
- Überwachen des Produktionsfortschritts.

Absatz-, Produktionsprogrammplanung und Sekundärbedarfsplanung werden häufig unter dem Begriff der *Grobplanung* zusammengefasst. Die Aufgaben der Losgrößen- und Belegungsplanung sowie das Überwachen des Produktionsfortschritts werden in der *Produktionsfeinplanung* durchgeführt. Die Kapazitätsplanung kann sowohl der Grob- als auch der Feinplanung zugeordnet werden.
Im Einzelnen können die Aufgaben wie folgt beschrieben werden: Die *Absatz-* und *Produktionsprogrammplanung* legt den mittel- bis langfristigen Absatz und die daraus resultierende Produktion in Menge und Zeit fest (Planung des Primärbedarfs). Die Produktionsprogrammplanung bezieht sich auf das End- und die wesentlichen Zwischenprodukte. Das Endprodukt in einer Brauerei ist das abgefüllte Bier und ein Zwischenprodukt ist das Bier nach dem Lagern. Grundlage der Planung sind die Daten des Marketing und des Vertriebs insbesondere den Kundenbestellungen.
Bei der *Sekundärbedarfsplanung* werden die Einsatzmaterialien in Menge und Zeit berechnet, die zum Erfüllen des durch das Produktionsprogramm definierten

Produktionssolls benötigt werden. Da die Bierproduktion mehrere Wochen in Anspruch nimmt, müssen Eingangsstoffe wie Hopfen oder Malz rechtzeitig, also mehrere Wochen vor der Abfüllung und dem Verkauf des Biers, zur Verfügung stehen. Dieser Bedarf wird nicht durch die eigene Produktion gedeckt, sondern müssen extern bestellt werden.
Innerhalb der *Kapazitätsplanung* werden die Planmengen den vorhandenen Produktionskapazitäten gegenübergestellt. Anhand von Durchschnittswerten sollen mögliche Engpässe ermittelt werden, die gegebenenfalls ein Vorverlegen der Produktion erforderlich machen. Gerade zu saisonalen Spitzenzeiten können Kapazitätsengpässe auftreten, die durch eine Kapazitätsplanung rechtzeitig erkannt werden.
Im Unterschied zur stückorientierten Fertigung liegt in Getränkebetrieben eine chargenbezogene Produktion vor. Die *Losgrößenplanung* bestimmt die Größe der Produktionschargen. Die Losgröße kann für ausgewählte Prozesstufen, etwa im Sudhaus, fest gegeben sein. In anderen Prozesstufen, etwa in der Filtration oder Abfüllung, sind die Losgrößen dagegen variabel und könenn von den Planern bestimmt werden.
Die *Belegungsplanung* ordnet die Produktionschargen den Produktionsanlagen zu. Das Resultat ist ein Belegungsplan über alle Prozesstufen, bei dem die Chargen die Produktionsanlagen belegen. Diese Belegungen werden in Form von *Produktionsaufträgen* modelliert, mit einem geplanten Start, einem geplanten Ende und einer geplanten Anlagezuordnung. Die Reihenfolge der Chargen wird hierbei idealerweise mit dem Ziel des Minimierens der Rüstzeiten gebildet. Die Kapazitäts- und Belegungsplanung sollte simultan mit der Losgrößenplanung erfolgen, in der Praxis werden jedoch häufig erst Produktionslose gebildet und erst in einem Folgeschritt werden dann geeignete Produktionsanlagen ausgewählt. Das Bilden der Produktionslose erfolgt in diesem Falle auf Basis von langjährigen Erfahrungswerten.
Das *Überwachen des Produktionsfortschritts* erfolgt durch die Produktionssteuerung wahrgenommen. Nach der Freigabe der Produktionsaufträge für die Steuerung werden die Rückmeldungen aus der Produktion entgegengenommen. Gerade in Anbetracht des hohen Störzeitenanteils in der Abfüllung müssen Produktionsaufträge unter Umständen zeitlich verschoben oder sogar erneut eingeplant werden.

Innerhalb der genannten Aufgaben der PPS gibt es drei wesentlichen Funktionen, die der Unterstützung der Anwender bei ihren täglichen Planungsaufgaben dienen:

- Simulation,
- Auswertungsfunktionen,
- interaktives Erstellen von Belegungsplänen.

Die *Simulation* ermöglicht den Planern, verschiedene Varianten eines Planes aufgrund von Änderungen der Ressourcenkapazitäten, etwa von Anlagen, Personal und Schichten, oder sonstiger Planungsparameter durchzuspielen und derart die erzeugten Pläne zu optimieren. Durch simulierte Planungsläufe mit Plankostenrechnungen können beispielsweise Rüstkosten und Personalkosten bei zeitweise Stilllegung einzelner Linien verglichen werden Die Planer haben die Möglichkeit, umzuplanen und alternative Planungen durchzuführen, ohne eventuelle Auswirkungen für die Produktion fürchten zu müssen. Erst wenn die Planer mit dem Produktionsplan zufrieden sind und ihn gegebenenfalls mit anderen Stellen abgeglichen haben, wird der Plan veröffentlicht.
Eine Reihe von *Auswertungsfunktionen* sollen das Planungsergebnis analysieren. In einem Produktionsplanungssystem muss es möglich sein, sich schnell und aktuell einen Überblick über die geplanten Vorgänge in allen Bereichen und deren aktuellen Status zu verschaffen. Planvarianten können jederzeit in ihren Auswirkungen auf das Gesamtsystem analysiert und hinsichtlich verschiedener Kriterien dargestellt werden. So können etwa die erwarteten Lagerbestände der Produkte über die Zeit berechnet werden oder die Auslastung der einzelnen Abfülllinien, die Umstellzeiten und -kosten und ähnliche Kennzahlen des Belegungsplans.
Die direkte Anbindung an den Prozess mit Rückmeldung des Produktionsfortschritts ermöglicht darüber hinaus das Auswerten der Planproduktion im Vergleich zur tatsächlichen Istproduktion. Auswertungen ermöglichen zudem eine kostenmäßige Bewertung der Pläne. Die Auswertungsfunktionen helfen damit den Anwendern, die erstellten Pläne zu interpretieren. Die Auswertungen erfolgen in der Regel grafisch, in Listenform und mit Hilfe der grafischen Plantafel. In der grafischen Plantafel wird der Belegungsstatus der Anlagen über die Zeit dargestellt und der Fortschritt der Bearbeitung der einzelnen Chargen farbig gekennzeichnet. Gleichzeitig ist die Plantafel ein wichtiges Instrument beim interaktiven Erstellen von Belegungsplänen. Häufig wünschen die Planer, die vom System errechneten Pläne zu verbessern oder selber zu planen. Die Plantafel kann eine solche manuelle Planung unterstützen, indem sie Berechnungen übernimmt (z. B. Rüstkosten), die Belegung darstellt, Warnungen bei Inkonsistenzen erteilt und durch Drag–und Dropfunktionen ein komfortableres Arbeiten ermöglicht. Abbildung 5.5 zeigt als Beispiel die grafische Plantafel des Planungssystems Pas–Plan der Firma Werum GmbH. Ein weiteres Beispiel für grafische Auswertungen sind die am Anfang dieses Kapitels gegebenen Diagramme der Bestandsentwicklung über die Kalenderwoche und der Maschinenauslastung. Die Bestandsentwicklung ist in den Abbildungen 5.1 und 5.3 dargestellt, die Maschinenauslastung in den Abbildungen 5.2 und 5.4.

5.2.1 Grundobjekte der Planung

Nachdem die Grundaufgaben der PPS dargestellt worden sind, schließen wir mit einer kurzen Darstellung der zentralen Datenobjekte der Produktionsplanung.

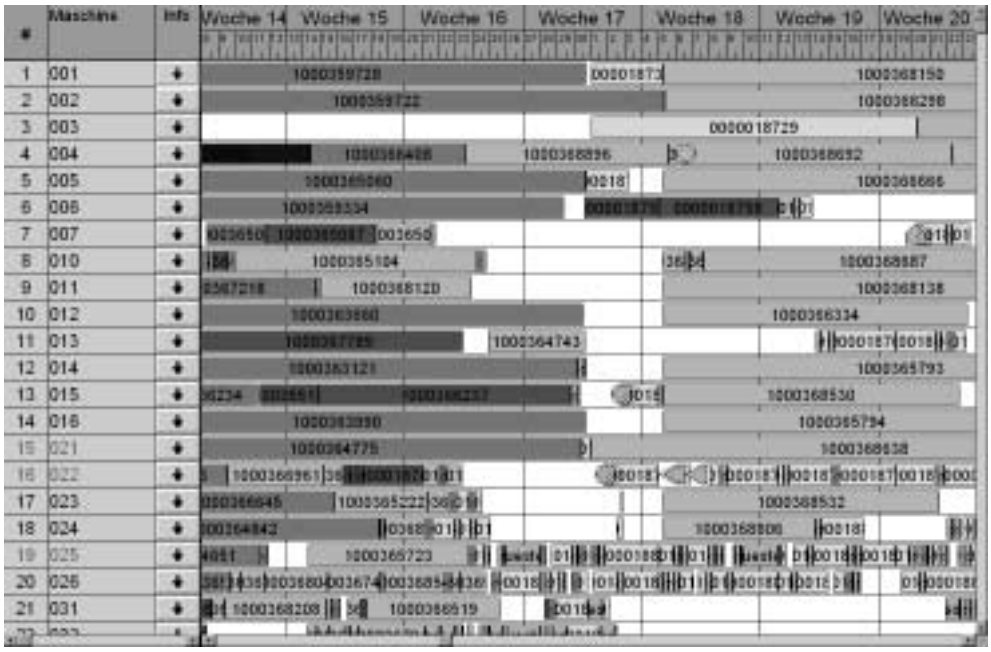


Abbildung 5.5: Grafische Plantafel

Durch die Definition dieser Grundobjekte soll auf der einen Seite eine einheitliche Terminologie für die Darstellung in den folgenden Kapiteln gefunden werden, auf der anderen Seite werden diese Datenobjekte aber auch die zentrale Grundobjekte bei der späteren DV–technischen Realisierung bilden.

Produkt
Produkte sind alle durch den Produktionsprozess herzustellenden Erzeugnisse. Produkte gliedern sich in *Halbfertigprodukte* als Ergebnis der vorgelagerten Produktionsbereiche und die verpackten *Endprodukte*. Beispiele für Halbfertigprodukte und Endprodukte sind das Jungbier und die gefüllte Getränkeflasche.

Produktgruppe
In einer *Produktgruppe* werden verschiedene Produkte zu einer Gruppe zusammengefasst. Die Produktgruppe erhält vom vom Benutzer einen eindeutigen Namen.

Stückliste
Jedem Produkt ist eine *Stückliste* zugeordnet. Die Stückliste eines Produktes beschreibt die zur Herstellung notwendigen Materialien nach Art und Menge bezogen auf eine Normcharge. Anhand der tatsächlichen Größen der eingeplanten Chargen können die benötigten Einsatzmaterialien nach Art, Menge und Termin berechnet werden.

Produktionseinheit
Die *Produktionseinheit* ist der Oberbegriff für einen Arbeitsplatz oder eine Anlage, in der ein Prozess durchgeführt werden kann. Beispiele für Produktionsein-

heiten sind Maschinen oder Tanks.

Produktionseinheitengruppe
In einer *Produktionseinheitengruppe* werden verschiedene Produktionseinheiten zu einer Gruppe zusammengefasst. Die Produktionseinheitengruppe erhält vom Benutzer einen eindeutigen Namen.

Produktionsbereich
Ein *Produktionsbereich* oder kurz *Bereich* eines Betriebes bezeichnet eine produktionstechnische Gesamtheit. Ein Bereich hat in der Regel einen Bereichsleiter, der die Durchführung der Produktion koordiniert. In das Aufgabenfeld dieser Bereichsleiter fallen auch die von uns betrachteten Teilaufgaben der Produktionsplanung. In der Brauindustrie gibt es zwei Produktionsbereiche, den Bereich *Bierherstellung* und den Bereich *Bierabfüllung*. Die Leiter dieser beiden Produktionsbereiche erzeugen ihre Belegungspläne in der Regel weitgehend unabhängig voneinander, wohingegen bei der mittel- bis langfristigen Planung meist eine gegenseitige Abstimmung erfolgt.

Fertigungsstufe/Fertigungsbereich
Die *Fertigungsstufe* bezeichnet eine planerische Einheit innerhalb derer die Planung unabhängig durchgeführt wird. Im Gegensatz zum Produktionsbereich erfolgt also nur eine organisatorische Entkoppelung produktionstechnisch naheliegender oder zusammenhängender Bereiche. Ein Produktionsbereich kann sich insbesondere in mehrere Fertigungsstufen untergliedern. Die Leiter eines Produktionsbereichs übernehmen in der Regel die Planung für alle Fertigungsstufen ihres Produktionsbereichs. Die Prozesse innerhalb einer Fertigungsstufe erzeugen als Ergebnis wiederum Produkte (End- oder Halbfabrikate). Als Beispiel betrachten wir den Produktionsbereich Bierherstellung. Dieser lässt sich zusätzlich in die Fertigungsstufen *Sudhaus* und *Gär–/Lagerkeller* untergliedern. Im Sudhaus wird als Halbfabrikat die *Ausschlagwürze* erzeugt und im anschließenden Gär–/Lagerkeller entsteht das *Jungbier*.

Fertigungsauftrag
Die Planung legt als Soll fest, welches Produkt zu welcher Zeit in welcher Menge produziert werden soll. Dazu wird bei der Planung eine Folge von entsprechenden *Fertigungsaufträgen* erzeugt. Neben den Planvorgaben sind den Aufträgen auch Informationen über den Produktionsablauf zugeordnet, beispielsweise eine Rezeptur nach der produziert werden soll. Fertigungsaufträge geben aber nur Rahmenvorgaben. Eine Auswahl der für die Herstellung vorgesehenen Anlagen und ein konkreter Starttermin muss für einen Fertigungsauftrag noch nicht vorliegen. Fertigungsaufträge können sowohl in der Grobplanung als auch in der Feinplanung erzeugt werden.

Produktionsauftrag und Maschinenauftrag
Ein Belegungsplan wird in Form von *Produktionsaufträgen* der Produktionssteuerung bekanntgegeben. Produktionsaufträge lauten auf die im Belegungsplan festgelegten Chargen. Produktionsaufträge beschreiben die Zuordnung von

Produkten zu Produktionseinheiten in Zeit und Menge. Der Belegungsplan ist damit die Gesamtheit der Produktionsaufträge. Produktionsaufträge bilden wiederum die Schnittstelle zur Produktionssteuerung in die sie eingelastet werden. Wenn sich der Herstellprozess eines Produktes in mehrere Arbeitsgänge untergliedert, umfasst der Produktionsauftrag die Maschinenbelegungen der einzelnen Arbeitsgänge. Für jeden Arbeitsgang wird dann ein eigener *Maschinenauftrag* generiert.

Herstellanweisung/Rezeptur

Die *Herstellanweisung* oder *Rezeptur* beschreibt die Produktion eines Produktes. Sie gliedert sich in Arbeitsgänge, die den einzelnen Prozessschritten entsprechen. Die *Verfeinerungen* beschreiben die für den Arbeitsgang zulässigen Anlagen und enthalten weitere für die Produktion des Produktes auf der gegebenen Anlage relevante Informationen. Ein Beispiel für eine solche Information ist die Durchlaufzeit einer Charge im Sudwerk 1 oder verschiedene Anlageneinstellung bei der Chargenabbarbeitung.

Kalender

Mit Hilfe von *Kalendern* werden zeitrelevante Informationen definiert. Beispiel sind die zeitliche Verfügbarkeit von Ressourcen und des Personal. Die Kalender speichern spezifisch für die einzelnen Produktionseinheiten Schichten, Pausen und Arbeitszeiten. Zudem sind Feiertage und Wochenenden im Kalender eingetragen.

KAPITEL 6

Beschreibung der Produktionsprozesse eines Brauereiunternehmens

Nachdem im letzten Abschnitt eine Einführung in die Problematik der Produktionsplanung und -steuerung gegeben worden ist, soll in diesem Kapitel das Anwendungsproblem betrachtet werden, also die fiktive Brauerei, für die ein PPS–System erstellt werden soll. Nur die genau Kenntnis der produktionstechnischen, organisatorischen und informatorischen Abläufe des betrachteten Unternehmens ermöglicht eine Systementwicklung, die auch in der späteren industriellen Praxis zu bestehen vermag und nicht direkt an einer fehlenden Praktikabilität oder Inkompatibilität mit bestehenden Betriebsabläufen scheitert. In diesem Kapitel werden daher die Produktionsabläufe in einem Unternehmen der Brauindustrie beschrieben. Wir beschreiben ein Ebenenmodell zur Einordnung einer Brauerei in Organisationseinheiten und gehen danach auf die Produktionsprozesse des Brauunternehmens ein. Die Einteilung eines Unternehmens in Organisationseinheiten ist für die spätere Systementwicklung relevant, da sie den modularen Aufbau des Systems bedingt und über die Funktionen und Aufgabenbereiche der entsprechenden Ebenen auch die Anforderungen an die zugeordneten Planungsfunktionen spezifiziert.
Die gewählte Darstellungsform ist abstrahierend, und damit reduziert auf diejenigen Abläufe und Einflussgrößen, die in direktem Zusammenhang mit der Produktionsplanung stehen. Sowohl die organisatorische Gliederung als auch die Beschreibung der Produktion wird in diesem Rahmen also deutlich gekürzt und vereinfacht dargestellt. So werden zum Beispiel die einzelnen Arbeitsschritte im Sudhaus beschrieben, wobei wir zum Beispiel Probleme der Anlagenzuordnung und der Durchlaufzeiten berücksichtigen, aber bei allen technischen Detailfragen, wie Anlagebeschaffenheit, Eigenschaften der Eingangsstoffe und ähnliches auf die umfangreiche Literatur verweisen. Dabei seien vor allem die Standard-

werke [hey94] [Kun94] [Nar86] genannt. Weitere Arbeiten zur Einführung in die Prozesse der Brauindustrie sind [Nar76] und [Nar85]. Einen Überblick über die Bierabfüllung sowie Aspekte der Simulation von Abfülllinien werden in [Räd99] gegeben.
In Brauereien findet sich eine deutliche Trennung zwischen *Bierherstellung* und *Bierabfüllung*. In Abschnitt 6.2 werden die wesentlichen Elemente der Bierherstellung zusammenfassend dargestellt. In Abschnitt 6.3 wird die Bierabfüllung beschrieben. Sie ist im Vergleich zur Bierherstellung ein deutlich einfacher zu planender Prozess. Allerdings ist auch hier in Anbetracht der hohen Kapitalbindung in den Abfüllanlagen eine optimierte Planung wünschenswert. Zudem beziehen sich unsere Überlegungen für allgemeine Unternehmen der Getränkeindustrie (wie etwa Mineralbrunnen) ausschließlich auf die Abfüllung, da bei allgemeinen Unternehmen der Getränkeindustrie keine vorgelagerten Prozesstufen vorliegen.

6.1 Grundaufbau einer Brauerei

Ein Unternehmen der Brauindustrie gliedert sich, wie bereits mehrfach angesprochen, üblicherweise in zwei autonome Produktionsbereiche, die *Bierherstellung* und die anschließende *Bierabfüllung*. Bei der Bierherstellung wird ausgehend von den Eingangsprodukten, also dem Hopfen und dem Malz die Würze erstellt, die nach dem Gären und Lagern zum fertigen Bier reift. In der Bierabfüllung wird das fertige Bier in Flaschen, Dosen oder Fässer abgefüllt und dann im Lagerkeller in Kisten oder einfach auf Paletten gelagert. Bindeglied zwischen der Bierherstellung und der Bierabfüllung sind der *Filter*- und der *Drucktankkeller*. Im Filterkeller wird das Bier geklärt. Der Drucktankkeller dient als Zwischenpuffer vor der Abfüllung. Die Drucktanks sind über Leitungen direkt mit den Füllern der Abfülllinien verbunden.
Neben der Gliederung in produktionstechnische Bereiche, lässt sich ein Brauereiunternehmen auch in organisatorische Bereiche einteilen. Dabei wird das Unternehmen zur Reduktion der Komplexität in eine Hierarchie von Abstraktionsebenen eingeteilt, wobei in der obersten Ebene, der Unternehmensleitebene die Informationen in verdichteter Form betrachtet werden, während in der untersten Ebene, der Feldebene sämtliche Prozessinformationen gespeichert werden.
Ein häufig betrachtetes Modell ist das Ebenenmodell des Informationsflusses in Unternehmen [AP94]. Es setzt sich aus fünf Ebenen zusammen. Auf den "oberen" Ebenen überwiegen die dispositiven Aufgaben, wie etwa die Unternehmensführung und Planung, während auf den "unteren", prozessnahen Ebenen die operativen Aufgaben (Ausführen von Eingriffen in den technischen Prozess) vorherrschen. Auf der Unternehmensleitebene findet die Unternehmensplanung statt. Diese beinhaltet die Investitions-, Personal- und Finanzplanung und diverse Kontrollfunktionen.
Die Produktionsleitebene beschäftigt sich mit der Produktionsgrobplanung, die

sich, wie in Abschnitt 5.2 dargestellt, aus der Produktionsprogrammplanung und der Disposition des Sekundärbedarfs und von Beständen zusammensetzt. Zudem können Aufgaben wie die Auftragsabwicklung und Auftragsverwaltung der Produktionsleitebene zugeordnet werden.
Die Aufgabe der Betriebsleitebene ist, aus der Produktionsgrobplanung eine Produktionsfeinplanung zu erstellen. Dazu gehört die Disposition von Personal, Einsatzstoffen und Apparaten, das Festlegen der Losgrößen und die Belegungsplanung als Vorgaben für die Prozessleitebene.
Auf der Prozessleitebene findet die prozessnahe Informationsverarbeitung statt. Von dieser Ebene aus werden die technischen Prozesse durch Funktionen wie Regeln, Steuern, Sichern und Bedienen beeinflusst. Weiterhin sind auf dieser Ebene all die Funktionen angesiedelt, die Informationen aus den technischen Prozessen zusammentragen und sie in einer dem Menschen verständlichen Form darstellen. Dazu gehört das Anzeigen, Überwachen und Melden ebenso wie das Protokollieren und die Trendaufzeichnung und -wiedergabe.
Mit ihren Sensoren beschafft die Feldebene Informationen über die technischen Prozesse und gibt diese zur Verarbeitung weiter an die Prozessleitebene. Von dort aus wird, über die Aktoren der Feldebene, Einfluss auf die technischen Prozesse genommen.
Abbildung 6.1 stellt die Einordnung eines Brauunternehmens in die organisatorischen Einheiten dar. In diesem Modell wird in der Unternehmensleitebene noch nicht in die beiden Produktionsbereiche Bierherstellung und Bierabfüllung unterschieden, da im wesentlichen betriebswirtschaftlich orientierte Funktionen durchgeführt werden, produktionstechnische Fragestellungen dagegen eine untergeordnete Rolle spielen. Bei der Produktionsleitebene beginnt die Trennung in Bierherstellung und Bierabfüllung. Eine wichtige Aufgabe der späteren Systementwicklung ist, die Planungsfunktionen den einzelnen Bereichen zuzuordnen. So ist zum Beispiel zu erwägen, ob für Bierherstellung und Bierabfüllung zwei getrennte Absatzplan- und Sekundärbedarfsrechnungen durchgeführt werden, oder ob sie noch als gemeinsame übergeordnete Funktion betrachtet werden. Die Trennung zwischen Bierherstellung und Bierabfüllung wird in der untersten Ebene, der Feldebene wieder aufgehoben. Zwar stehen die Maschinen und die ihnen angeschlossenen Sensoren in unterschiedlichen Bereichen eines Unternehmens — sie sind also real getrennt, jedoch sollte das Produktionsleitsystem einen durchgängigen Chargenfluss ermöglichen. Ziel ist, dass jede Abfüllcharge durchgängig bis zur eingegangenen Malzcharge zurückverfolgt werden kann.
Nach dieser organisatorischen Einordnung eines Unternehmens der Brauindustrie, die wiederum erheblichen Einfluss auf die Anforderungen an das Planungssystem ausübt, sollen nun die produktionstechnischen Einflussparameter der Planung beschrieben werden.

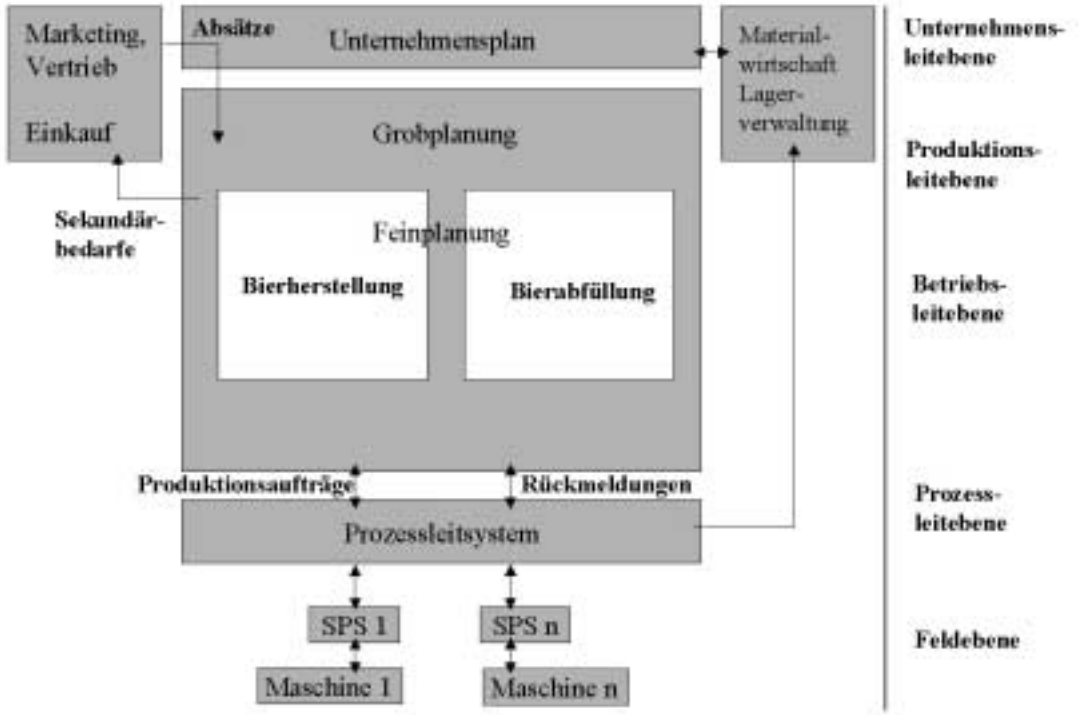


Abbildung 6.1: Einordnung eines Brauereibetriebes in das Ebenenmodell

6.2 Beschreibung der Bierherstellung

Die Bierherstellung beginnt bei der Rohstoffannahme und umfasst das *Sudhaus*, den *Gär-* und den *Lagerkeller*. Im Sudhaus wird aus Wasser, Malzschrot und Hopfen die *Bierwürze* gewonnen. Dazu wird zunächst das Braumalz geschrotet und daraufhin in einem Maischbottich das Malzschrot mit Wasser vermischt. Anschließend wird die so gewonnene *Maische* auf verschiedene Temperaturstufen erhitzt. Durch Mälzen und Maischen wird die unlösliche Stärke des Gerstenkorns freigelegten und durch Enzyme in vergärbaren Malzzucker umgewandelt. Im Läuterbottich werden lösliche und unlösliche Bestandteile getrennt. Die Treber, also die unlöslichen Bestandteile setzen sich auf dem Boden ab und bilden so einen Filter. Die Bierwürze wird dann in die Sudpfanne geleitet, in der sie unter Beigabe von Hopfen gekocht wird. Durch die Wasserverdampfung erhält die Würze die gewünschte Konzentration. Damit wird der Stammwürzegehalt festgelegt. Vor der Gärung muss die Würze jedoch noch geklärt und gekühlt werden. Im Whirlpool, einem stehenden zylindrischen Tank, werden die beim Kochen ausgefällten Eiweissteilchen und Hopfenrückstände ausgeschieden. Die heiße geläuterte Würze wird in Wärmetauschern gekühlt.

Der zweite Bereich der Bierherstellung beginnt mit dem Gärkeller. Die gekühlte Würze fließt in die Gärgefäße. Bei untergäriger Hefe dauert die Gärung etwa zehn Tage, bei obergäriger Hefe etwa vier bis sechs Tage. Die Gärung wird durch Hefebeigabe initiiert. Voraussetzung ist, dass alle Hefezellen die gleichen Eigenschaften besitzen. Die Brauhefe wird daher in Reinzuchtanlagen aus einer einzigen Zelle bis zur benötigten Menge gezüchtet (Hefereinzucht). Durch das Umwandeln in Alkohol wird der Stammwürzegehalt langsam abgebaut. Der genaue Zeitpunkt des Endes des Gärvorganges liegt von vornherein nicht fest. Daher kann eine Planung nur auf Durchschnittswerte zurückgreifen und muss bei der Umsetzung in der Produktion dynamisch angepasst werden. Das Ergebnis des Gärprozesses ist das *Jungbier*.
Das Jungbier muss nun noch mehrere Wochen nachgären und reifen. Die Nachgärung erfolgt in den Lagertanks im Lagerkeller. Dort reichert sich das Jungbier auf natürliche Weise mit Kohlensäure an und reift bis zur geschmacklichen Vollendung aus. Damit die Kohlensäure im Bier verbleibt, erfolgt die Nachgärung unter erhöhtem Druck. Bevor das Bier in Fässer und Flaschen gefüllt wird, durchläuft es noch einen Filter. Dies dient der Klärung, die im wesentlichen einer "Verschönerung" des Biers gleichkommt. Zusammengefasst werden bei der Bierherstellen die folgenden Produktionseinheiten durchlaufen:

Schrotmühle
Die Schrotmühle zerkleinert das Braumalz.

Maischbottichpfanne
Das Malzschrot wird in der Maischbottichpfanne mit Wasser vermischt. Dieser Vorgang wird *Einmaischen* genannt In der Maischbottichpfanne wird die Maische auf verschiedene Temperaturen erhitzt. Dabei verflüssigen sich die ursprünglichen schwer löslichen Bestandteile des Malzschrots.

Läuterbottich
Die Würze wird im Läuterbottich oder Maischefilter von den Malztrebern befreit. Das Resultat dieses Brauvorgangs ist die Würze, in der alle löslichen Stoffe des Malzkorns enthalten sind.

Sudpfanne
Während des Kochens wird der Würze der Hopfen zugeführt. Je mehr Hopfen zugegeben wird, desto herber schmeckt das spätere Bier. Die Würze wird in der Sudpfanne etwa $1,5$ Stunden lang gekocht.

Whirlpool
Im Whirlpool werden die Trubstoffe ausgeschieden. Die Bierwürze ist danach fertiggestellt.

Würzekühler
Bevor die Würze in den Gärkellern in Gärtanks gefüllt werden kann, muss sie im Würzekühler auf Kellertemperatur abgekühlt werden.

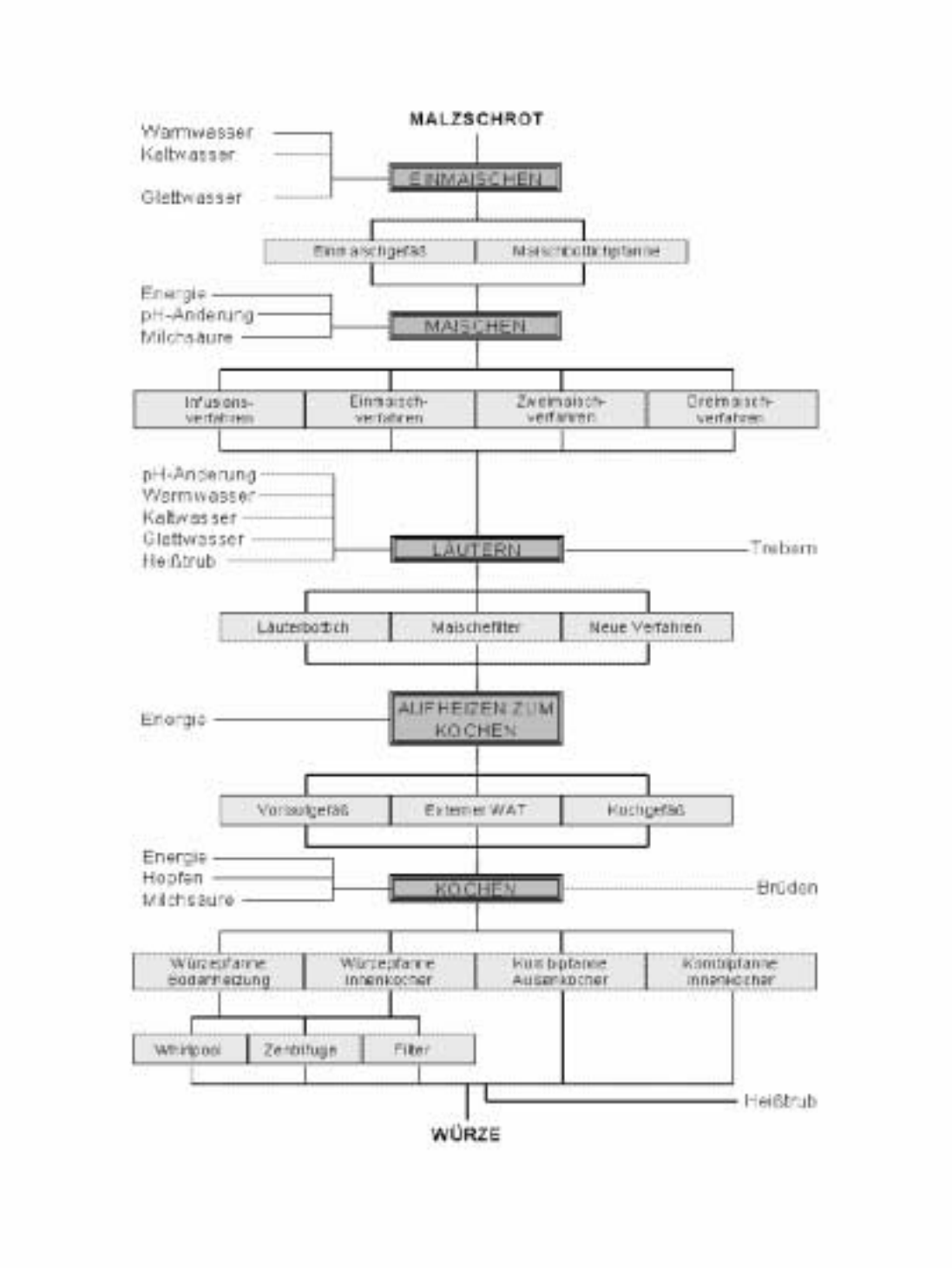


Abbildung 6.2: Schematische Darstellung der Prozesse im Sudhaus

Gärkeller
Im Gärkeller wird der Würze die Bierhefe zugegeben. Nun setzt die Gärung ein.

Die Gärung mit untergäriger Hefe erfolgt bei Temperaturen zwischen vier und zehn Grad Celsius. Sie dauert etwa zehn Tage. Obergärige Hefe vergärt die Würze in vier bis sechs Tagen bei 15 – 19 Grad Celsius.

Lagerkeller
Das Jungbier wird in Tanks im Lagerkeller umgefüllt. In den Lagertanks kommt das Jungbier zur Ruhe und die Nachgärung beginnt. Bei etwa Null Grad lagert es dort mehrere Wochen bis zur vollendeten Reife.

Abbildung 6.2 stellt die Prozessschritte der Würzeherstellung schematisch dar.

6.3 Beschreibung der Bierabfüllung

Bevor das Bier in Fässer (Kegs) und Flaschen gefüllt werden kann, muss es im Filterkeller filtriert werden. Ziel der *Filtration* ist, das Bier zu klären. Nach der Filtration wird das Bier in die *Drucktanks* gepumpt und von dort über den *Füller* der Abfüllanlage abgefüllt.
Die Abfüllanlage ist eine zusammenhängende Anlage, auf der die Flaschen und Fässer auf Förderbändern transportiert werden. Auf der Abfüllanlage befinden sich neben dem Füller noch weitere Aggregate, beispielsweise die Waschmaschine, verschiedene Kontrollaggregate oder der Palletierer. Im Einzelnen ergeben sich die folgenden Schritte:

Bierfilter
Nach Abschluss der Lagerzeit durchläuft das Bier eine Filteranlage, die der optischen Verschönerung des Biers und dem verbessern seiner kolloridalen und biologischen Stabilität dient.

Drucktank
Der Drucktank dient als Zwischenpuffer für das filtrierte Bier. Der Füller der Abfüllanlage entnimmt das fertige Bier aus den Drucktanks.

Flaschenabfüllanlage
Rund 70 % des Biers werden in Flaschen und Dosen gefüllt. Eine moderne Abfüllanlage erreicht bis zu 100000 Flaschenfüllungen pro Stunde. Anschließend presst der Verschließer die Kronkorken auf, und von der Etikettiermaschine erhält jede Flasche ihr Etikett. Flaschenfüller und Verschließer sind in einem Aggregat kombiniert. Der Flaschenfüller ist das zentrale Organ der Abfüllanlage. Die anderen Aggregate orientieren sich in ihrer Ausbringung am Füller. Abbildung 6.3 skizziert eine Abfüllanlage.

6.4 Planungsrelevante Restriktionen

In diesem Abschnitt werden verschiedene Aspekte der Bierherstellung und Bierabfüllung beschrieben, die einen direkten Einfluss auf die spätere Architektur des

Planungssystems haben, sei es im logischen Aufbau, den anwendbaren Planungsalgorithmen oder der Datenparametrisierung.

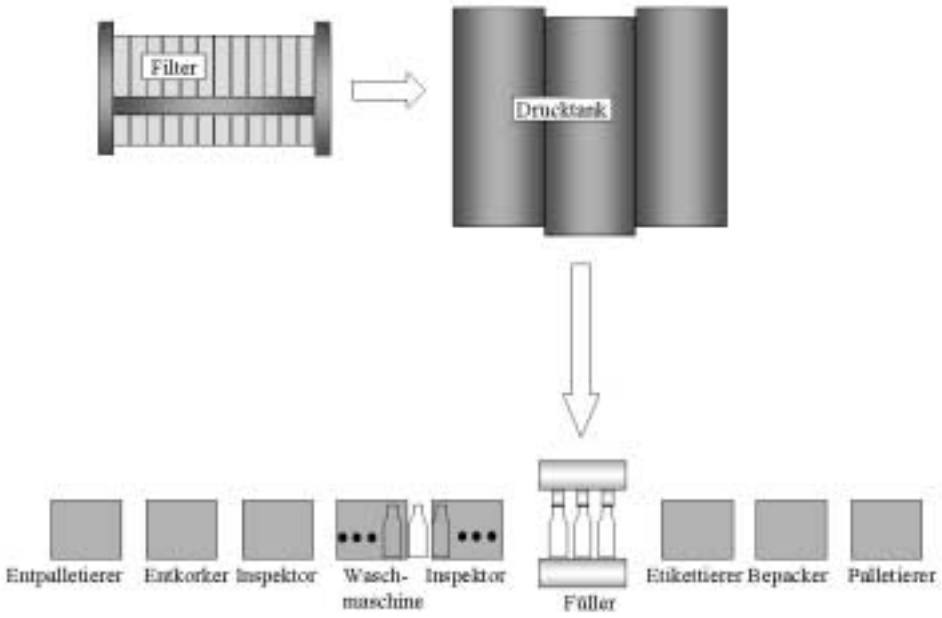


Abbildung 6.3: Schematische Darstellung einer Abfüllanlage

6.4.1 Spezielle Restriktionen in der Bierherstellung

Die Bierherstellung gliedert sich in die Prozesse des *Sudhauses*, der *Gärung* und der *Lagerung*. Brauereien haben in der Regel ein bis fünf Sudlinien. In modernen Sudhäusern können täglich ungefähr zwölf Sude pro Sudlinie gefahren werden. Die einzelnen Prozessschritte im Sudhaus folgen einander ohne Unterbrechung und ohne zeitliche Verzögerung. Durch das Festlegen des Einmaischzeitpunkts und des Sudwerkes ist somit die Belegung des gesamten Auftragsnetzes determiniert, da ein Wechseln der Sudlinien während des Sudvorgangs nicht gestattet ist. In neueren Brauereien können sich Sudlinien aber die Würzepfanne teilen. Im Sinne der im ersten Teil dieser Arbeit beschriebenen Terminologie des Scheduling können die einzelnen im Sudhaus durchgeführten Schritte zu einem einzelnen *Job* zusammengefasst werden. Zu beachten ist allerdings, dass Sude überlappend hergestellt werden. Genauer gilt, dass bereits beim Kochen des aktuellen Suds ein

neuer Sud eingemaischt wird. Die Sudfolge, zum Beispiel 1 h, beschreibt den Zeitraum zwischen zwei Einmaischvorgängen. Das Einmaischen erfolgt also in einem festen Rhythmus. Im Sudhaus wird meist im Dreischichtbetrieb gearbeitet. Die Sude können zu beliebigen Zeiten in die Gärtanks geleitet werden, auch wenn das dortige Personal im Zweischichtbetrieb arbeitet. Der gesamte Sudvorgang vom Schroten bis zum Anstellen umfasst einen Zeitraum von rund $8,5$ h bis 11 h.
Für jedes Produkt ist eine feste Chargengröße definiert. Die Durchlaufzeiten für die Arbeitsgänge sind ebenfalls fest determiniert.
Sowohl das Sudhaus, als auch der Gär-/Lagerkeller können ein Produktionengpass sein. Das Sudhaus ist in der Regel vom Wochenbeginn bis mindestens Donnerstag belegt. Ab Freitag Nachmittag bis Sonntag Abend wird häufig nicht mehr eingemaischt.
Für die einzelnen Prozessschritte gelten die folgenden weiteren Restriktionen:

- **Malzannahme:**

  Das Malz wird per LKW geliefert. Die übliche Chargengröße beträgt 15 – 30 Tonnen. Die Malzannahme erfolgt über Wiegesilos mit einer Verteilung des Malzes auf die Silos nach Malzsorten. Die Malzentnahme erfolgt aus allen Malzsilos anhand eine produktabhängigen prozentualen Verteilung. Die im Sudhaus eingesetzte Malzchargengröße wird *Schüttung* genannt. Als Beispiel kann eine Schüttung von 5800 kg eine Menge von 400 hl Ausschlagwürze ergeben.

- **Schroten:**

  Das Malz wird in Malzmühlen in einem kontinuierlichen Vorgang geschrotet. Bei einer Schroterei mit zwei Steinauslesern und Schwingsieben, Steuerwaagen, 6–Walzenmühlen mit Konditionierung kann zum Beispiel ein Durchsatz von 6 t/h angesetzt werden.

- **Einmaischen:**

  In modernen Brauereien ist der Arbeitsgang Einmaischen ein Engpass. Die Chargengröße ist beim Einmaischen fest gegeben, die Losgrößenplanung entfällt also für die Prozessschritte des Sudhauses. Die Gesamtmaischzeit ist produktabhängig, aber fest. Ein typischer Wert wäre eine Gesamtmaischzeit von 2 h.

- **Abläutern:**

  Das Abläutern ist ein kontinuierlicher Prozess. Die resultierende Läuterwürze fließt in ein Vorlaufgefäß, in dem es für das Würzekochen bereitsteht. Die Läuterzeit kann Werte von beispielsweise 170 min annehmen.

- **Würzekochen:**

  Die Durchlaufzeit beim Würzekochen beträgt mindestens einen Stunde. Anschließend wird im Whirlpool der Heisstrub entfernt. Das Entfernen des Heißtrubs dauert ungefähr 30 Minuten.

- **Würzekühlen:**

  Über mehrstufige Würzekühler wird die Würze auf die Anstelltemperatur abgekühlt. Die Leistung nimmt typischerweise Werte von 500 hl/h an, wobei eine Dauer von ungefähr 90 min angesetzt werden kann.

- **Anstellen:**

  Die gekühlte Würze wird belüftet und mit Hefezugabe angestellt. Dies leitet den Gärprozess ein. Die angestellte Würze kann je nach verwendeter Technologie direkt in die Gärtanks fließen oder in einem Anstelltank gepuffert werden. Ein häufig verwendetes Verfahren ist die Flotation, bei der ein weiterer Tank, der Flotationstank, zwischen Würzekühler und Gärtank verwendet wird. Der Flotationstank wird circa 6–8 h lang belegt.

- **Gärkeller:**

  Der Gärkeller besteht aus zylindroklokonischen Tanks unterschiedlicher Größe. Diese haben ein Fassungsvermögen von rund 300 hl bis 3000 hl. Ein Tank kann mehrere Sudchargen zur gleichen Zeit aufnehmen, sofern die Sorte übereinstimmt. Die Sude laufen in der Regel ohne Pufferung in die Gärtanks. Eine Ausnahme ist das Flotationsverfahren, bei dem der Anstelltank als kleiner Puffer verwendet wird. Die einem Tank zugeführten Sudchargen dürfen in ihren Startzeitpunkten des Einmaischens nicht zu stark differieren, da sonst die Bierqualität beeinträchtigt wird. Die Sude eines Tanks dürfen aber in unterschiedlichen Sudlinien hergestellt werden.

  Die Gärdauer wird ab dem Zeitpunkt gerechnet, ab dem alle Sude in den Tank geleitet worden sind, oder, in einer anderen Modellierung, sobald der erste Sud in den Tank überführt worden ist. Die Gärzeit ist beendet, wenn Diacetyl- und der Extraktgehalt vorgegebene Grenzwerte annehmen. Die Dauer des Gärverfahrens kann nur geschätzt werden. Planabweichungen um einen Tag sind dabei durchaus möglich.

- **Lagerkeller:**

  Das Jungbier wird im Lagerkeller für einen Zeitraum von ein bis sechs Wochen in zylindrokonischen Tanks unterschiedlicher Größen gelagert. Ähnlich dem Gärprozess kann die Dauer des Lagerprozesses nur geschätzt werden.

Kriterien für die Lagerdauer sind unter anderem der Grad der Stoffumwandlung, die Klärung des Biers und die Restextraktdifferenz zum Schlauchbierextrakt. Sowohl Gär- als auch Lagerkeller sind also in der Belegungsdauer nicht exakt determiniert. Bei größeren Brauereien, die nahe an ihren Kapazitätsgrenzen fahren, werden aber die vorgesehenen Planzeiten unter Ausnutzen gewisser verfahrenstechnische Möglichkeiten in der Regel eingehalten.

Die zylindrokonischen Tanks des Lagerkellers können ein größeres, aber auch kleineres Fassungsvermögen als die zugehörigen zylindrokonischen Tanks des Gärkellers haben. Gär- und Lagerkeller können aber auch zusammenfallen. In diesem Fall werden die Gärtanks gleichzeitig für das Lagern benutzt.

Die den Produkten zugeordneten Rezepte beziehen sich zwar auf Normchargen aus denen sich tatsächliche Mengen und Zeitvorgaben berechnen lassen, bei der Umsetzung in einen konkreten Belegungsplan, werden einzelne Chargen jedoch verschnitten. Dabei wird der Inhalt mehrerer Tanks zusammengeführt oder eine Tankbelegung umgekehrt auf mehrere Tanks verteilt. Auch hier kann die Produktionssteuerung Planvorgaben modifizieren. Als Folgerung kann nicht von einer einheitlichen Charge gesprochen werden, die als ganzes die Arbeitsschritte des Gär- und Lagerkellerbereiches durchläuft, sondern nur von einer Sequenz von Chargen. Diese Restriktion ist insofern für die Umsetzung eines PPS–Systems von Bedeutung, als die übliche Zuordnung von Chargen und ihren zugehörigen Arbeitsgängen zur Rezeptur nicht mehr möglich ist.

Nach Beendigung der Lagerung erfolgt eine rund $3,5$–stündiger Reinigung je Gefäß.

- Bei der Wahl der Gärtanks ist weiterhin zu beachten:
  - Für bestimmte Sorten sind bestimmte Tanks vorgesehen,
  - einem Tank dürfen nur Sudchargen der gleichen Sorte zugeführt werden,
  - Tanks müssen zum Belegen vorbereitet und gereinigt sein,
  - terminliche Restriktionen,
  - Volumenrestriktionen der Tanks.

Abbildung 6.4 verdeutlicht noch einmal den Chargenfluss durch das Sudhaus, den Gär- und den Lagerkeller.

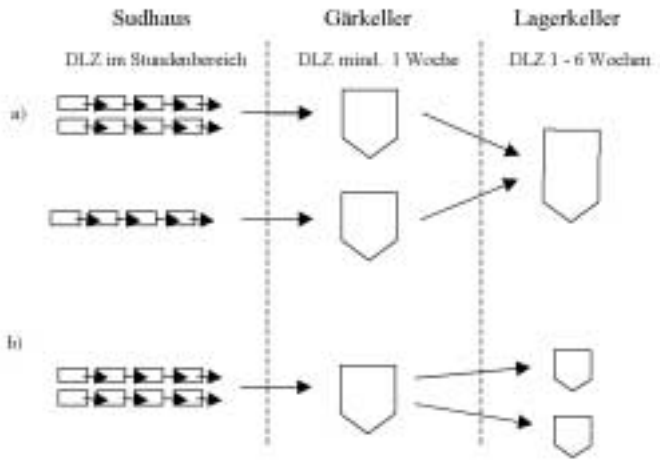


Abbildung 6.4: Chargenfluss in der Bierherstellung

6.4.2 Spezielle Restriktionen in der Bierabfüllung

Der *Filterkeller* zusammen mit dem *Drucktankkeller* bildet das Bindeglied zwischen Bierherstellung und Bierabfüllung. Ausgangspunkt der Planung im Filterkeller sind die Vorgaben der Abfüllung. Die in der Planung generierten Abfüllaufträge sind damit die Auslöser der Filtrationsaufträge. Filtrationsaufträge beziehen sich auf eine Drucktanksorte. Der Produktionsabschnitt Filtration umfasst die Bereiche *Unfiltratbeschaffung*, die *Filtration* und die *Filtratverarbeitung*.
Filtrationsaufträge setzen sich aus drei Elementen, der Vorlaufzeit, der Filtration und der Nachlaufzeit zusammen. Die Folge der Filtrationsaufträge wird durch eine Rüstperiode bei Schichtbeginn und eine Reinigung am Schichtende umgeben. Zu beachten ist, dass das für die Rüstvorgänge zur Verfügung stehende Personal häufig begrenzt ist, so dass die Filtration in den verschiedenen Filterstraßen nicht zur gleichen Zeit beginnen kann. Die Filterleistung ist im Zeitverlauf nicht konstant. Von ihrer Entwicklung hängt aber die in einem bestimmten Zeitraum filtrierbare Menge ab. Die Filterleistung lässt sich durch zwei Phasen beschreiben. Zu Beginn des Filtrationsprozesses wird der Durchfluss bei ansteigendem Druck konstant auf Nennleistung gehalten (1. Phase). Bei einem Sortenwechsel sind entsprechende Drucksprünge anzusetzen. Bei Erreichen eines sortenabhängigen Höchstdrucks wird der Volumenstrom nicht weiter konstant gehalten, aber bei abfallender Leistung und leicht ansteigendem Druck weiter filtriert, bis die Leistung einen Grenzwert unterschreitet, der von der Nennleistung der Filterstraße abhängt (2. Phase). Als Resultat für die Planung kann also nicht von einer kon-

stanten Abarbeitungszeit der Aufträge ausgegangen werden. Die Dauer ist statt dessen abhängig vom Zeitpunkt, der Phase und den eingeplanten Vorgängerbelegungen.
Dem Filterkeller schließt sich der *Drucktankkeller* an. Bei der Förderung des Biers nach der Filtration in den Drucktankbereich können Engpässe hinsichtlich der Belegung der Filtratleitungen für die Modellierung grundsätzlich ausgeschlossen werden. Die Anforderungen einer Drucktankplanung weichen von denen der anderen Produktionsstufen insofern ab, als keine Arbeitsschritte eingeplant werden, welche die Tanks für eine gegebene Dauer belegen müssen. Statt dessen bilden die Drucktanks einen Puffer zwischen Filtration und Abfüllung. Falls eine detailgetreue Planung vorgesehen ist, besteht diese aus einer geeigneten Wahl der Drucktanks und dem Verbuchen der Zu- und Abgänge über die Zeit. Die Buchungsmengen ergeben sich in direkter Folge aus den Abfüllmengen und den Filtrationsaufträgen, die geeigneten Drucktanks müssen jedoch ausgewählt werden.
Bei der Auswahl eines Drucktanks sind die folgenden Kriterien zu erfüllen:

- Der Drucktank muss für die Sorte zulässig sein.
- Falls der Drucktank nicht leer ist, darf keine neue Sorte zugegeben werden.
- Falls der Drucktank nicht leer ist, darf nur Verschnitten werden, wenn die Drucktankcharge ein maximales Alter nicht überschreitet.
- Das verbleibende Tankvolumen darf die Größe der Drucktankcharge nicht unterschreiten.

An den Drucktank schließt sich direkt die Abfüllung an. Die Abfüllung erfolgt in voneinander unabhängigen Abfüllanlagen. Produkte können auf verschiedenen Abfülllinien gefahren werden, wobei bestimmte Linien für bestimmte Produkte vorgesehen sind.
Die folgenden Eigenschaften kennzeichnen den Abfüllvorgang:

- Rüstkosten treten bei Produkt-, Etiketten- und Gebindewechsel auf.
- Die Arbeitsgänge der Abfüllung folgen einander ohne Pufferung. Chargen können sich überlagern. Daher kann eine neue Charge bereits auf der Abfüllinie gestartet werden, während die Vorgängercharge sich noch auf der Linie befindet.
- Die Abfüllung ist durch einen hohen Störzeitenanteil gekennzeichnet, so dass die Nennausbringung und die tatsächliche Ausbringung differieren.
- Gearbeitet wird im Schichtbetrieb. Zum Ende der letzten Tagesschicht wird gereinigt. Zum Ende der Woche erfolgt eine umfangreiche Wochenendreinigung.

- Kritische Ressourcen für die Planung sind das Personal, das Leergut (Flaschen) und das Bier im Drucktank.

- Produktabhängige Chargengröße:

  Die Produkte gliedern sich in Hauptsorten (Pils) und Nebensorten (Light, Alkoholfrei). Während die Hauptsorten eine Abfülllinie in der Regel durchgängig belegen, werden die Nebensorten zu Losen zusammengefasst und auf einer weiteren Linie abgefüllt.

- Rollierende Planung:

  Die Planung erfolgt in der Regel als wöchentliche Feinplanung mit täglicher Anpassung und als Monatsplanung mit wöchentlicher Anpassung. Da die Planung "im laufenden Betrieb" erfolgt, ist die aktuelle Abfüllung als Vorbelegung auf den entsprechenden Abfülllinien einzutragen. Ebenso sind die aktuellen Bestände anzupassen. Bei einer Koppelung mit der Produktionssteuerung können aktuelle Produktionsdaten entgegengenommen werden. Die Planung muss dann auf verspätete Aufträge reagieren und gegebenenfalls neu planen.

KAPITEL 7

Grundlegender Lösungsansatz

Nachdem in den vorangegangenen beiden Kapiteln erstens die Aufgaben einer Produktionsplanung festgelegt worden sind und zweitens die Bierproduktion schematisch dargestellt worden ist, wird ab diesem Kapitel die Umsetzung in ein automatisiertes Planungssystem beschrieben. Voraussetzung für die Umsetzung des Planungssystems ist ein Systemdesign in dem sich die organisatorischen und informatorischen Abläufe des Brauunternehmens widerspiegeln. Das im folgenden vorgestellte Systemkonzept basiert auf den Arbeiten des Lehrstuhls für Brauereianlagen und Lebensmittel–Verpackungstechnik zusammen mit der Gesellschaft für Informationstechnik Weihenstephan mbH und der Werum GmbH, Lüneburg. Das PPS–System ist als Baukasten verschiedener Systemkomponenten mit unterschiedlichen Eingangs- und Ausgangsdaten konzipiert. Die einzelnen Systemmodule besitzen Schnittstellen zu unterschiedlichen Fremdsystemen aus denen sie Daten importieren oder aber an die sie in der Gegenrichtung Informationen abgeben. Durch das Festlegen der Systemkomponenten zusammen mit den Daten– und Informationsströmen ist dann die Grundlage für eine algorithmische Umsetzung gegeben. Auch bei der Algorithmenentwicklung liegt der Fokus wiederum auf dem modularen Aufbau und der Bereitstellung von Schnittstellen, die eine Integration mit geringem Programmieraufwand ermöglichen.
Eine zentrale von den Anwendern gestellte Anforderung an das Planungssystem war, dass die Planungsalgorithmen inklusive allen Datentransfers nur wenige Minuten in Anspruch nehmen. Ansonsten wäre mit einer geringen Akzeptanz des Systems von Seiten der späteren Benutzer zu rechnen. Die Endbenutzer erwarten ein System, in dem sie interaktiv und mit manuellen Eingriffen Pläne erzeugen können und eine Vielzahl von Varianten mit unterschiedlichen Planparametern durchspielen können. Zudem muss die Möglichkeit bestehen, schnell und kurzfristig umzuplanen, wenn sich die Ausgangssituationen etwa in Folge von Verspätungen einzelner Aufträge, oder beim Eingang unerwarteter Kundeneilaufträge ändern. Um ein gutes Laufzeitverhalten erzielen zu können, wurden vergleichsweise einfache im wesentlichen auf Prioritätsregeln basierende Algorith-

men implementiert. Darüber hinaus wurde ein hoher Aufwand in die Entwicklung der Datenstrukturen des Planungssystems gesteckt. Als Ergebnis liegt ein Datensystem vor, in dem die wesentlichen Objekte durch komplexe Zeigerstrukturen miteinander verbunden sind. Dadurch wurde ein Direktzugriff auf alle Elemente und damit eine besonders effiziente Implementierung ermöglicht. Einige dieser Datenobjekte sind in Kapitel 8 beschrieben. Da das Planungssystem aber im Zuge einer Industriekooperation erstellt wurde, ist es nicht möglich, an dieser Stelle das gesamte Datensystem zu beschreiben.

7.1 PPS als integrierte Softwarelösung

PPS–Systeme können zwar als Stand-Alone-System operieren, ihren vollen Funktionsumfang erhalten sie aber als integrierter Bestandteil einer umfassenden DV–Architektur. Als wesentliche Voraussetzung für eine derartige DV–Architektur sind den einzelnen Systemen die aktuellen Daten sofort und in der notwendigen Detaillierung zur Verfügung zu stellen. Dazu müssen nicht alle Funktionen von einem Rechner durchgeführt werden und auch nicht von einem Programmsystem. Sie sollten vielmehr auf einer gemeinsamen Datenbasis aufsetzten und vor Ort verfügbar gemacht werden. Diese Anforderungen können mit offenen und verteilten DV–Systemen realisiert werden, die in die weiteren Systeme der Unternehmens–DV integriert sind. Der erste Schritt bei der Umsetzung eines PPS–Systems ist daher die Einordnung in eine gesamte DV–Architektur, also hier in ein unternehmensübergreifendes Management- und Kontrollsystem für das Brauereiunternehmen. Die Anforderungen und die Realisierung eines derartigen integrierten Management- und Kontrollsystem für Brauereiunternehmen ist in [Keh96] beschrieben, die Integration und die Architektur eines Planungssystems innerhalb einer solchen DV–Architektur in [SWBE99] und in [BS97] [Sch97] [Sch98]. Eine allgemeine Beschreibung von computergestützen Systemen in Brauereien findet sich in [RBKW98]
Grundlage der Integration des PPS–Systems in die unternehmensübergreifende DV–Architektur ist eine gemeinsame Datenbasis aller beteiligten Systeme. Eine solche gemeinsame Datenbasis kann heutzutage mit einer verteilten relationalen Datenbank realisiert werden. Jede Anwendung bezieht ihre Daten aus dieser Datenbank und kann modifizierte Daten wieder in die Datenbanktabellen zurück schreiben. Auf diese neuen Daten, etwa den berechneten Belegungsplan, können dann die anderen Systeme zurückgreifen. Abbildung 7.1 zeigt die von uns vorgesehene Einordnung eines PPS–Systems in die Unternehmens–DV. In der obersten Ebene finden sich die Programme zum Unterstützen der Funktionen der Unternehmensleit- und der Produktionsleitebene. Dies sind Systeme für das Rechnungswesen, den Einkauf, die Materialwirtschaft und das Vertriebssystem mit dem Verkauf. Darunter sind die DV–Systeme der Produktionsleitebene bis zur Prozessleitebene angeordnet, insbesondere auch die in dieser Arbeit betrachtete

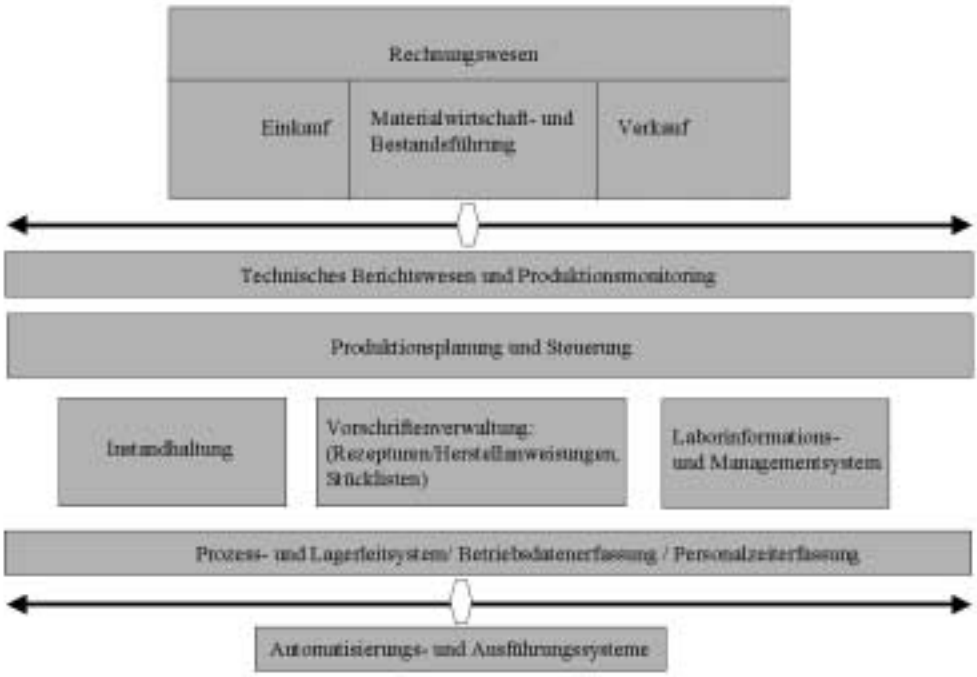


Abbildung 7.1: Globale Systemstruktur

PPS. Weitere Module sind das technische Berichtswesen und das Monitoring, das Instandhaltungssystem, die Vorschriftenverwaltung und das Laborinformations- und Managementsystem.
Das *Produktionsmonitoring* übernimmt die aus dem Prozessleitsystem anfallenden Daten, verdichtet diese und bereitet die Daten in einer für den Anwender leicht verständlichen und übersichtlichen Form, beispielsweise grafisch, auf. Eine Darstellung eines derartigen Produktionsmonitoring findest sich in [WR98].
Im *Instandhaltungssystem* werden zukünftig vorgesehene Wartungsmaßnahmen verwaltet, die zum vorübergehenden Ausfall von Anlagen führen. Die Planungsalgorithmen müssen die Produktionschargen in den verbleibenden Zeiten einplanen. Die theoretische Schwierigkeiten im Zuge derartiger Wartungsmaßnahmen wurden bereits im ersten Teil dieser Arbeit in Form des Problems *Maintenance–Scheduling* eingehen beschrieben.
Ein System zur *Vorschriften-* oder *Rezepturverwaltung* ist in der Prozessindustrie eng mit der Produktionsplanung verbunden. Sie beschreibt für jedes Produkt, welche Herstellungsschritte mit welchen Parametern durchgeführt werden dürfen. Grob gesprochen berechnet die Produktionsplanung beim Einplanen eines Produktionsloses die Belegungsdauer auf den in Frage stehen Anlagen auf

Basis der in der Rezepturverwaltung parametrisierten Werte. Auch können die Materialanforderungen mit Hilfe der Rezepturen ermittelt werden.
Das *Laborinformationssystem* speichert die Labordaten zu den Produktionschargen. Für die Planung ist die Tatsache relevant, dass verschiedene biochemische Einflussfaktoren zu Indeterminismen hinsichtlich der Produktionsdauer führen können. So ist zum Beispiel der Gärvorgang erst beendet, wenn das Diacetyl vollständig abgebaut ist und der Extraktgehalt bestimmte Werte annimmt. Die genauen Zeitpunkte können nicht im voraus bestimmt werden.
Das *Prozess-* und *Lagerleitsystem* übernimmt die in der Produktionsfeinplanung erzeugten Produktionsaufträge und lastet sie auf den vorgesehenen Anlagen ein. Alle Produktionsistdaten werden dabei vom *Betriebsdatenerfassungssystem* gespeichert und aufbereitet.
In der untersten Ebene der DV–Systeme und organisatorisch der Feldebene zugeordnet befinden sich die *Automatisierungs-* und *Ausführungssysteme*. Speicherprogrammierbare Steuerungen (SPS) werden dabei zur Steuerung und Regelung der einzelnen Maschinen verwendet.
Für den logischen Aufbau des PPS–Systems sind besonders die Schnittstellen und Datenströme zu den übergeordneten und niedriger angeordneten Systemen zu spezifizieren. Nach oben ist die PPS mit der betriebswirtschaftlichen Software verbunden. In vielen Unternehmen wird dabei heutzutage die SAP–Software [SAP94] [SAP96] eingesetzt. Das PPS–System empfängt aus dem Absatzprognose- und Vertriebssystem die für den betrachteten Planungszeitraum geplanten Absätze und tatsächlichen Kundenbestellungen. Im Gegenzug liefert sie der höheren Ebene einen groben Plan der Produktion für die einzelnen Planungsperioden. Zudem werden alle aus der Planung resultierenden Materialanforderungen an die Materialbeschaffung weitergeleitet. Das PPS–System kommuniziert des Weiteren mit dem untergelagerten Prozessleitsystem. In Form von Produktionsaufträgen wird der Belegungsplan den Anlageführern gemeldet, die für die Umsetzung des Plans auf den einzelnen Anlagen verantwortlich sind. Im Gegenzug erhält das PPS–System vom Prozessleitsystem Rückmeldungen über den Produktionsiststand und den Status der Abbarbeitung der einzelnen Produktionsaufträge. Im Falle von Störungen oder Verspätungen erhält das PPS–System dadurch die Möglichkeit, die Pläne auf die veränderte Situation hin anzupassen.

7.2 Beschreibung der Systemkomponenten

Das hier vorgestellte PPS–System ist als Branchenlösung für die Brau- und Getränkeindustrie in das Produktionsplanungssystem Pas–Plan der Firma Werum Datenverarbeitungssyteme GmbH integriert. Es lehnt sich daher eng an das Systemkonzept von Pas–Plan an. Das Planungssystem ist als Access 7.0–Applikation realisiert. Das Access–System ist mit einem dezentralen Datenbanksystem (hier *Oracle*) verbunden. Die Stamm- und Bewegungsdaten werden aus Effizienz-

gründen aber auch teilweise lokal in den Access–Datenbanken gespeichert und nur die global verwendeten Daten nach Oracle hin exportiert. Die in dieser Arbeit verwendeten Planungsalgorithmen sind in verschiedenen C–Modulen realisiert, die von Access aus aufgerufen werden können. Die Access–Applikation besteht aus den folgenden Funktionen:

- Datenverwaltung
- graphische Planauswertung
- graphische Plantafel mit integrierten Planungsfunktionen
- Absatzplanung

Das Access-System bildet die Schnittstelle zu den Benutzern. Diese können menügesteuert die oben genannten Access–Applikationen starten, insbesondere die Parametrisierung der Daten, die graphische Darstellung und Aufbereitung eines Plans und Funktionen für die manuellen Planungen. Einen separaten Bereich bildet die Funktion *Planungslauf*, die als das eigentliche Herzstück der Planung bezeichnet werden kann. In einem Planungslauf werden die Stamm- und Bewegungsdaten aus den Access–Tabellen importiert. Der Planungslauf berechnet auf Basis der aktuellen Daten neue Belegungspläne und führt die zur Planevaluation notwendigen Berechnungen durch. Zudem ist es Aufgabe des Planungslaufs, manuell erzeugte, aber nicht zulässige Belegungen zu identifizieren. Damit besteht der Planungslauf aus zwei Komponenten, der eigentlichen Planung und der Überprüfung der Zulässigkeit manuell oder interaktiv erstellter Belegungspläne (Konsistenzprüfung). Aufgrund der Komplexität des Planungsfunktionen wird der Planungslauf nicht im eigentlichen Access–System durchgeführt, sondern in einer eigenständigen von Access aus gesteuerten C–Komponente. Abbildung 7.2 verdeutlicht die Struktur des Planungssystems. Der Planungslauf erhält von Access die Stamm und Bewegungsdaten in Form von ASCII–Tabellen. Darüber hinaus werden Planparameter, etwa die betrachtete Planungsperiode, übergeben. Die Planung im C–Modul verwendet spezielle Ladefunktionen um die Accessdaten einzulesen und aufzubereiten. Eine Bibliothek von Planungsalgorithmen ist für die eigentliche Planung und die Berechnungen verantwortlich. Schließlich übertragen die Ladefunktionen die Planergebnisse und Meldungen wieder an das Access–System zurück. Dort kann dann eine Auswerten der Pläne erfolgen. Das C–Modul Planung kann auch als eigene Lösung arbeiten, also abgekoppelt von Access. Dazu müssen die Eingangsdaten der Planung als ASCII–Dateien bereitstehen. Die Planergebnisse werden wieder in ASCII–Dateien geschrieben.
Das C–Modul Planung ist für die eigentliche Produktionsplanung verantwortlich. Es führt die folgenden Aufgaben der Produktionsplanung durch:

- Zulässigkeitsprüfung der Planung

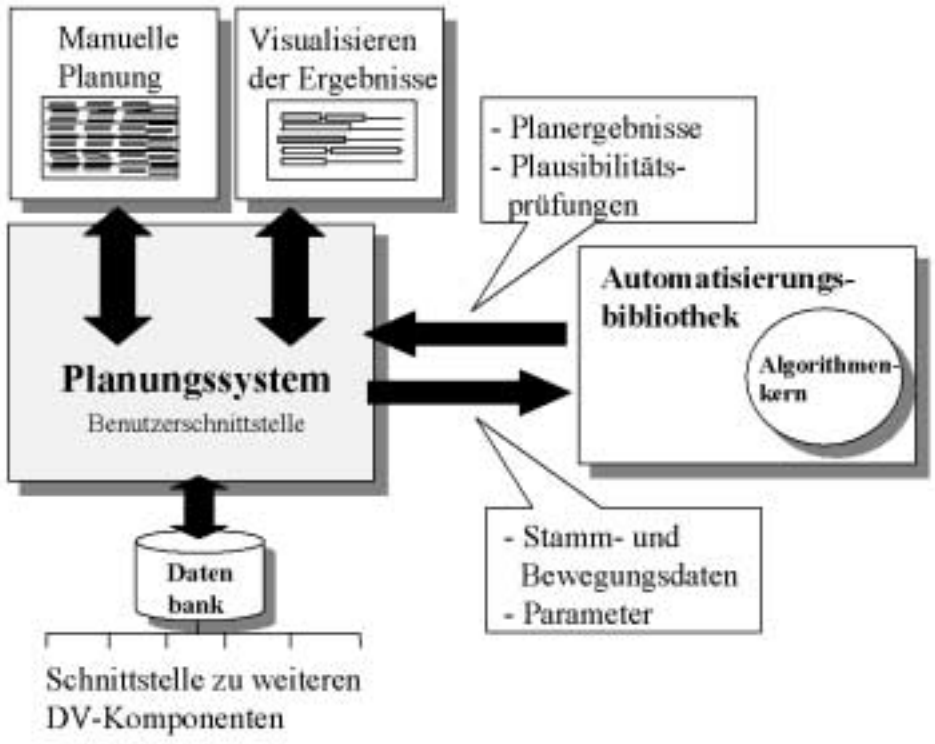


Abbildung 7.2: Aufbau des PPS–Systems

- Durchführen der Kapazitäts- und Belegungsplanung
- Datenaufbereitung zum Auswerten der erzeugten Pläne

Das C-Modul ist in mehrere Module untergliedert, die in drei Gruppen eingeteilt werden können.

1. Lade- und Exportfunktionen der Daten aus Access (IO–Module)
2. Funktionen zum Verwalten der grundlegenden Datenstrukturen (Lib–Module)
   - Listenfunktionen
   - Suchbäume
   - Speicherverwaltung
   - Sortierfunktionen
   - Funktionen zum Verwalten der Datums- und Zeitinformationen
3. Funktionen zur Planung und Auswertung (Plan–Module)

- Aufbau der Relationen (Verzeigerung der Planungselmente untereinander)
- Kapazitätsplanung
- Funktionen zum Bilden von Losen und zum Terminieren der Aufträge
- Funktionen zum Belegen der Belegungspläne mit Aufträgen

7.3 Zu lösende Teilprobleme

Bei der Realisierung des PPS–Systems für die Brau- und Getränkeindustrie treten eine Reihe von speziellen Teilproblemen auf. Einige Probleme liegen in der speziellen Struktur der Produktionsprozesse in der Brau- und Getränkeindustrie begründet. Als Beispiel sei die Belegung der Tanks genannt, bei der sich mehrere Chargen des gleichen Produktes einen Tank teilen können oder die Belegung der Abfüllanlagen sowohl durch Langläuferprodukte als auch durch Exportartikel mit vergleichsweise kleinen Chargengrößen. Weitere Schwierigkeiten beruhen auf der Tatsache, dass ein Großteil der derzeit auf dem Markt befindlichen PPS–Systeme auf Anwendungen in der Fertigungsindustrie ausgerichtet sind und von ihrer Systemstruktur eher Funktionen für die höheren Unternehmensebenen unterstützen. Insbesondere sind sie durch eine fehlende Anbindung an die Systeme zur Steuerung des Produktionsprozesses gekennzeichnet sowie durch eine mangelnde Unterstützung der Produktionsfeinplanung. Speziell in der Getränkeindustrie ist die fehlende Anbindung an die Produktion eine deutliche Einschränkung, denn in Anbetracht des hohen Störzeitenanteils einer Flaschenabfüllung ist eine Planung, die nicht auf das tatsächliche Geschehen reagieren kann, sicherlich nur von begrenztem Nutzen. Daher führen aber viele gängige Konzepte der Produktionsplanung für die hier betrachtete Anwendung nicht zum Ziel.
In den folgenden Abschnitten werden einige Fragestellungen und Lösungsansätze erörtert, die sich im Zuge der speziellen Anwendung der Brau- und Getränkeindustrie ergeben haben, die aber durch die Strukturen der derzeitigen PPS–Standardsoftware bisher nur ungenügend unterstützt werden. Es werden dabei recht unterschiedliche Teilprobleme betrachtet, allen gemein ist jedoch, dass sie die Planung von der höheren Managementebene in Richtung der Produktionsleitebene und den prozessnahen Funktionen erweitern.

7.3.1 Mehrstufige Planung

Die Hauptschwierigkeit bei der Konzeption eines PPS–Systems für die Brauindustrie ist die bereits mehrfach erwähnte Trennung in die beiden Produktionsbereiche Bierabfüllung und Bierherstellung. Die Verantwortung für die Planung in beiden Bereichen obliegt unterschiedlichen Personen, die ihre Planungsfunktionen zu unterschiedlichen Zeiten an unterschiedlichen Rechnern durchführen. Zudem

wird für unterschiedliche Zeitraster geplant. In der Bierherstellung vergehen angefangen vom Bereitstellen des Malzes bis hin zur Reifung in den Lagertanks mehrere Wochen, während der Abfüllvorgang einer Charge nur wenige Stunden dauert. Dies erschwert die Koordination der beiden Bereiche. Andererseits können Bierherstellung und Bierabfüllung nicht völlig getrennt betrachtet werden, denn die Bierabfüllung kann nur dann durchgeführt werden, wenn das Bier vorher bereitgestellt wurde. Daher ist es unerlässlich, dass die geplanten Abfüllmengen (etwa des Folgemonats) bereits der Bierherstellung gemeldet werden, so dass diese eine rechtzeitige Produktion veranlassen kann. Gefordert ist also eine mehrstufige Planung, bei der die Plandaten einer Stufe die Ausgangsdaten der vorgelagerten Prozessstufe bilden und die einzelnen Prozessstufen aber dennoch unabhängig voneinander Planen können.
Einer Planung unter derartigen Bedingungen kann durch das Konzept der Fertigungsstufen genügt werden. Fertigungsstufen erlauben, die Planung und die Sicht auf die Daten auf bestimmte Bereiche einzuschränken. Produkte und Produktionseinheiten sind dabei bestimmten Fertigungsstufen zugeordnet. Für die Dateneingabe oder auch die Planungsläufe können die Benutzer die gewünschte Planungsstufe einstellen. Eingeplant werden die Produkte der aktuellen Fertigungsstufe, für die eine gültige Herstellanweisung existiert und ein Produktionsbedarf aus Absätzen oder dem Sekundärbedarf abgeleitet werden kann.
Das Konzept der Fertigungsstufen ermöglicht auch eine mehrstufige Planung. Ein Planungslauf wird zunächst für die Planung der Fertigungsstufe des Endproduktes durchgeführt. Wenn ein gültiger Plan erstellt ist, kann durch die Sekundärbedarfsauflösung der Bedarf an Produkten der vorgelagerten Fertigungsstufe ermittelt und in die Access-Tabelle der Sekundärbedarfe eingetragen werden. Wechselt ein Planer die Fertigungsstufe, wird genau der Sekundärbedarf derjenigen Produkte sichtbar, die in der neuen Fertigungsstufe definiert sind. Das Konzept der mehrstufigen Planung ist in Abbildung 7.3 dargestellt. In dem in Abbildung 7.3 gegebenen Beispiel wurden in der Abfüllung Aufträge für die Produkte $Pils$-05-l und $Pils$-033-l erzeugt. Die Stückliste dieser Produkte nennt die benötigte Menge der Würze für das Pils normiert auf eine Normchargengröße. Anhand der tatsächlichen Größe der eingeplanten Abfüllcharge lässt sich die tatsächlich benötigte Menge der Würze berechnen. In den Produkttabellen ist neben den Produkten $Pils$-05-l und $Pils$-033-l als weiteres Produkt die Würze $Pils$ eingetragen. Die Würze Pils ist gleichzeitig ein eigenes Produkt mit einer eigenen Nummer und einer Herstellvorschrift. Dieses Produkt ist für den Produktionsbereich Bierherstellung definiert und nun liegt ein Produktionsbedarf in Form eines Sekundärbedarfes vor, der in gleicher Weise wie ein Absatz für die letzte Fertigungsstufe behandelt wird. Die Planung der Fertigungsstufe 1 versucht nun, Aufträge zum Erfüllen des Sekundärbedarfes zu erzeugen. Das obige Beispiel verdeutlicht, wie das Konzept der Fertigungsstufen eine Planung ermöglicht, bei der einzelne Produktionsbereiche autonom geplant werden können, jedoch stets auf den für ihre Planung benötigten Datensatz aufbauen können.

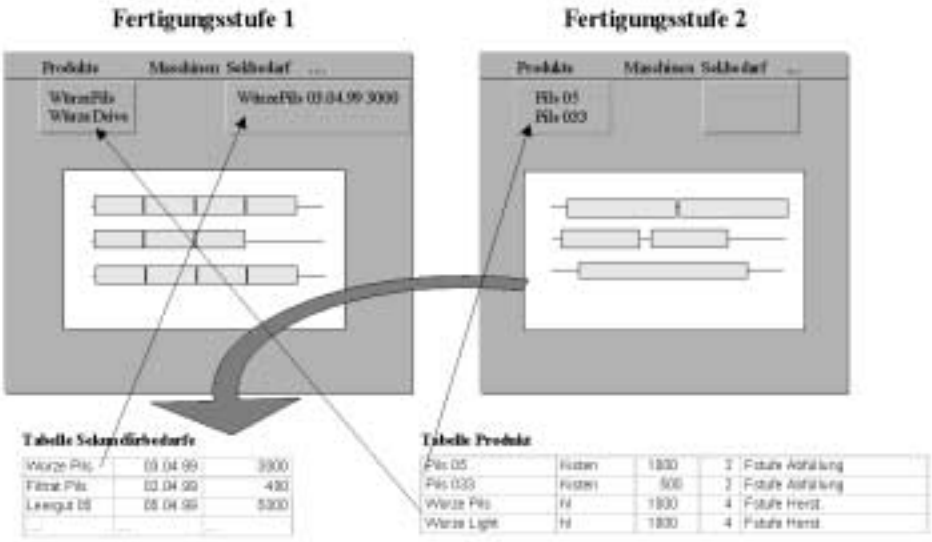


Abbildung 7.3: Mehrstufige Planung

Das Konzept der Fertigungsstufen ermöglicht zusätzlich eine feinere Abstimmung der Algorithmik für die Produktionsfeinplanung. Wie das Beispiel Bierherstellung und Bierabfüllung zeigt, sind die produktionstechnischen Gegebenheiten, die Zeithorizonte und die Planziele in den einzelnen Fertigungsstufen derart unterschiedlich, dass nicht ein einzelner Algorithmus zum Planen in allen Produktionsbereichen eingesetzt werden kann. Vielmehr muss für jede Anwendung eine Bibliothek von Planungsalgorithmen bereitgestellt werden, die genau auf die gegebenen Anforderungen abgestimmt ist. In den noch folgenden Kapiteln werden die verwendeten Algorithmen für die beiden Produktionsbereiche beschrieben. Hier sei nur erwähnt, dass in den Tabellen der einzelnen Fertigungsstufen auch eine Algorithmentabelle parametrisiert ist, in der die Folge der Algorithmen für die aktuelle Fertigungsstufe festgelegt ist. Die Planung der Fertigungsstufe führt dann sukzessive die genannten Algorithmen durch. Auf diese Weise können für verschiedene Fertigungsstufen auch unterschiedliche Planverfahren verwendet werden.
Für die Planung in Brauereien ergibt sich in natürlicher Weise eine Gliederung in die beiden genannten Fertigungsstufen

- Fertigungsstufe 1: Bierherstellung,

- Fertigungsstufe 2: Bierabfüllung.

Beide Fertigungsstufen werden unabhängig voneinander geplant. Die Bierherstellung hat Zugriff auf den aus der Bierabfüllung resultierenden Sekundärbedarf. Ziel der Planung der Fertigungsstufe 1 ist das Erstellen eines zulässigen Sudplans. Dieser soll in der Grobplanung auf Wochen und Monatsbasis und in der täglichen Feinplanung zeitgenau, also auf Stunden- und Minutenbasis planen. Eine Restriktion der Planung im Sudhaus ist die Kapazität der anschließenden Gär- und Lagertanks. Jeder Sudplan wird daher auf seine Plausibilität bezüglich dieser Kapazitäten hin untersucht. Dies geschieht in einer groben "Belegungsrechnung" für den Gär- und Lagerkeller. Für kritische Chargen im Sudhaus, zu denen kein anschließender Tank gefunden werden kann, wird eine Warnung ausgegeben. Ein Belegungsplan für Gär- und Lagerkeller wird jedoch erst zur Laufzeit innerhalb der Steuerung manuell erzeugt. Der Grund für einen Verzicht der Belegungsplanung im Gär- und Lagerkeller liegt in der hohen Unsicherheit der Eingangsdaten, den Indeterminismen in der Belegungsdauer und der Tatsache, dass sich verschiedenen Abläufe wie das Kräusen nicht oder nur ungenügend auf die Datenstrukturen einer Produktionsplanung abbilden lassen.
Das Ziel der Planung in der zweiten Fertigungsstufe ist im Wesentlichen, einen gültigen Abfüllplan zu erzeugen. Dabei wird zwischen zwei Arten von Produkten unterschieden. Langläuferprodukte sind durch die Hauptsorten gegeben. Sie belegen gewisse Produktionseinheiten fast durchgehend. Nur wenn der Produktbestand zu groß wird, wird die Belegung unterbrochen. Die zweite Produktgruppe ist durch die Nebensorten gegeben. Bei diesen sind die Bedarfe kleiner, so dass in der Regel mehrere Lose zu einer Charge zusammengefaßt werden mit dem Ziel, die Zahl der Rüstvorgänge zu verringern.
Es besteht die Möglichkeit, in der Bierabfüllung noch die Belegungen des Filterkellers und der Drucktanks zu gestalten. Die Belegungen werden jeweils durch eigene Algorithmen innerhalb der gleichen Fertigungsstufe erzeugt, jedoch erfolgt die Planung für diese beiden Bereiche nur in vereinfachter Form. Die Planungsalgorithmen für den Filterkeller und den Drucktankkeller sind in den Abschnitten 9.8 und 9.11 gegeben.
Der Sekundärbedarf ist das Bindeglied der beiden Fertigungsstufen. Da in der Bierherstellung der Sudplan erzeugt wird, mussen der Sekundärbedarf auf die Biersorte für die Würze lauten. Falls in der Bierabfüllung Drucktanks und Filterkeller geplant werden sollen, mussen zusätzlich der Sekundärbedarf bezüglich der Filtratsorte gestellt werden. Daher wird zunächst ein Abfüllplan erstellt und der zugehörige Sekundärbedarf erzeugt. Ein zweiter Algorithmenbaustein plant den Sekundärbedarf "Filtrat" ein und schließlich wird die Drucktankbelegung erzeugt.

7.3.2 Absatzplanung

Ausgangspunkt aller Planvorgänge sind die eingegangen Kundenaufträge und die Absatzvorgaben des Vertriebs. Auf Basis verschiedener Prognoserechnungen werden dabei ausgehend von den Absatzzahlen des Vorjahres für die Folgemonate neue Absatzzahlen prognostiziert. Voraussetzung dafür ist aber, dass das Planungssystem mit einem überlagerten DV–System gekoppelt ist. Da aber viele Unternehmen der Brau- und Getränkeindustrie erst im Begriff sind, umfassende DV–Systeme einzuführen, und daher die Koppelung zu einer betriebswirtschaftlichen DV möglicherweise noch nicht gelingt, sollte auch eine einfache Absatzprognoserechnung innerhalb des Planungssystems ermöglicht werden.
In der einfachsten Version führt die Funktion zur Absatzprognose und Absatzplanung zwei Rechnungen durch:

1. Bestimmung der Absatzmengen aus dem Vorjahr und Verteilung des Jahresabsatzes über die Monate nach gegebenen Schlüsseln.

2. Abgleich mit einem Trendwert, der aktuelle Feiertage und erwartete Verkaufsentwicklungen berücksichtigt.

Ausgangspunkt der Absatzprognose sind die Absatzzahlen des Vorjahres. Die Eingabetabelle *absatz_alt.tab* enthält die Absatzmengen nach Monat aufgeschlüsselt. Es wird nicht nach Produkten unterschieden. Die Angaben erfolgen in *hl*.

Tabelle 7.1: absatz_alt.tab

Monat	Menge
Jan	80000
Feb	85000
März	60000
April	70000
...	...

Aufgrund von Verschiebungen der Feiertage, veränderten Absatzprognosen und speziellen Aktionen des Vertriebs muss von einer veränderten Absatzverteilung im aktuellen Jahr ausgegangen werden. Die erwartete prozentuale Änderung ist für die einzelnen Kalendermonate in der Eingabetabelle 7.2 (*trend.tab*) dargestellt.

Tabelle 7.2: trend.tab

Monat	Menge in %
Jan	2
Feb	-1
März	3
...	...

Um ein Aufschlüsseln des Absatzes nach Produkten zu ermöglichen, wird der Prozentanteil der Produkte am gesamten Absatz in einer weiteren Datenbanktabelle namens *prodanteil.tab* gespeichert.

Tabelle 7.3: prodanteil.tab

Produkt	Prozentanteil
Pils	65
Dunkel	30
Alkfrei	5
...	...

Die Tabelle *absatz.tab* ist die Ergebnistabelle der Absatzplanung. Zur Vereinfachung in den Rechnungen bezieht sich der Absatz immer auf den letzten Kalendertag des Monats.

Tabelle 7.4: absatz.tab

Produkt	Datum Von	Datum Bis	Menge
Pils	31.1.97	31.1.97	53040
Pils	28.2.97	28.2.97	54697.5
Pils	31.3.97	31.3.97	40170
...	...	...	...
Dunkel	31.1.97	31.1.97	24480
Dunkel	28.2.97	28.2.97	25245
Dunkel	31.3.97	31.3.97	18540
...	...	...	...
Alkfrei	31.1.97	31.1.97	4080
Alkfrei	28.2.97	28.2.97	4207.5
Alkfrei	31.3.97	31.3.97	3090
...	...	...	...

Diese Absatztabelle kann als Grundlage für die weiteren Planungsfunktionen verwendet werden. Der Algorithmus zum Berechnen der erwarteten Absatzmengen führt nun auf Basis der genannten Eingabetabellen die folgenden Rechnungen durch:

Algorithmus 7.3.1 Absatzprognose

1. Abgleich der Absatzmenge aus *absatz_alt.tab* für jeden Kalendermonat um entsprechenden Wert aus *trend.tab*
2. Aufteilung des im ersten Schritt berechneten Absatzes nach Produkten anhand Tabelle *prodanteil.tab*
3. Generieren der Tabelle *absatz.tab* aus den unter Schritt 2 berechneten Werten

Bei einer späteren Anbindung an das Vertriebssystem kann das Absatzprognosesystem entfallen. In beiden Fällen kann jedoch für die folgenden Ausführungen von einer vorhandenen Tabelle *absatz.tab* ausgegangen werden die die erwarteten Absätze auf Produktbasis für die einzelnen Planungsperioden wiedergibt.

7.3.3 Sekundärbedarfsauflösung

Die Sekundärbedarfsauflösung betrachtet die Liste der fixierten Maschinenaufträge. Jedem Maschinenauftrag ist ein Produkt und eine Herstellanweisung zugeordnet, die in einer Stückliste die Materialanforderungen bezogen auf eine Normcharge enthält. Die Sekundärbedarfsauflösung berechnet für jede Position der Stückliste anhand der Chargenmenge des Maschinenauftrages die Anforderungsmenge an das Material. Der Termin der Anforderung ist der Termin des Maschinenauftrages oder der Beginn des Rasterintervalls der Grobplanung, in das der Auftrag fällt.

Beispiel:
Die Normcharge für das Produkt $Pils$–$0,5$–l betrage 1000 Flaschen. Dann kann die Stückliste die Form haben:

- 1000 Fl $0,5$–l,
- 1000 Etiketten,
- 5 hl Pils,

unabhängig davon, wie die tatsächlichen Chargen gewählt werden.
Wenn nun ein Abfüllauftrag auf den 7.10. 8:00 Uhr datiert ist und 20000 Flaschen umfasst, werden Sekundärbedarfsaufträge für das 20–fache der Normcharge gebildet. Dies sind 20000 Flaschen, 20000 Etikette und ein Auftrag für 100 hl Pils. Der Bedarf wird auf den 7.10 datiert.
Alle Aufträge des Sekundärbedarfs werden in einer Access–Tabelle gespeichert, so dass die anderen Fertigungsstufen darauf zugreifen und sie als Ausgangspunkt der Planung verwenden können. Wenn also in die Fertigungsstufe Bierherstellung umgeschaltet wird, ist dort von dem Sekundärbedarf nur das Produkt Pils definiert. Für dieses wird die erforderliche Produktionsmenge vom vorhandenen Bestand abgebucht. Der Bedarf wird also in gleicher Weise wie der Absatz für die Abfüllung bearbeitet.

7.3.4 Parametrisierbare Algorithmen

Die Funktionen der Produktionsgrobplanung sind für den mittel- bis langfristigen Planungsbereich ausgerichtet. Aufgrund ihrer Orientierung an die höheren Managementebenen brauchen produktionstechnische Gegebenheiten in der Grobplanung nicht berücksichtigt zu werden, so dass häufig ein einziges Grobplanungssystem über alle Fertigungsstufen genügt.

Bei der produktionsnahen Feinplanung dagegen müssen für jede Fertigungsstufe eigene Algorithmen verwendet werden. Wie bereits in der Beschreibung der Prozessschritte der Brauproduktion deutlich wurde, kann nicht ein Algorithmus alleine den unterschiedlichen Anforderungen, Nebenbedingungen und Zielkriterien aller Prozessstufen genügen. Vielmehr muss eine Bibliothek von parametrisierbaren Algorithmen bereitgestellt werden, von denen dann die für eine Produktionsstufe geeigneten Algorithmen ausgewählt werden können. Dabei sind bei der Entwicklung der Algorithmenbibliothek zwei Gesichtspunkte zu beachten:

- Um den Programmieraufwand gering zu halten, sollten alle Algorithmen auf den gleichen Datenstrukturen aufsetzen.
- Die Schnittstellen zu den Algorithmen sollten derart offen gehalten sein, dass sie von den Anwendern ohne zusätzlichen Programmieraufwand aufgerufen werden können.

Zur Umsetzung der genannten Aspekte wurde eine universale Datenstruktur der Belegungsplanung entwickelt, die unabhängig von den Fertigungsstufen und ihren speziellen Gegebenheiten verwendet werden kann. Die Datenstruktur wird in Abschnitt 8.1 beschrieben.
Als zweites wurde für das Planungssystem ein Baukastenprinzip von Algorithmen entworfen. Die Grundidee ist, dass die Anwender in einer Access–Tabelle eine Folge von Algorithmen, die sogenannten *Algorithmenbausteine*, eingeben können. Zusätzlich werden in die Tabelle spezielle Steuerungsparameter eingetragen. Beispiele für Steuerungsparameter sind Richtlosgrößen, einzuplanende Produktgruppen oder Prioritäten. Die derart parametrisierten Algorithmen werden sukzessive mit dem aktuellen Belegungsplan aufgerufen. Gestartet wird mit dem leeren Belegungsplan, in den im ersten Schritt alle Vorbelegungen wie fixierte Aufträge, Wartungen und aktuelle durchgeführte Produktionsaufträge eingetragen werden. Die Planungsalgorithmen führen daraufhin ihrerseits Berechnungen durch und geben einen modifizierten Plan an den nächsten Algorithmus weiter. Der vom letzten Algorithmus erzeugte Belegungsplan wird dann als Ergebnis zurückgegeben. Abbildung 7.4 verdeutlicht das Prinzip. Für die Brauereianwendung sind die folgenden Algorithmen konfiguriert:

Fertigungsstufe 1: Bierherstellung
Algorithmus Sudplan
Erzeugt einen Sudplan. Ausgangspunkt der Planung ist der Sekundärbedarf aus der Abfüllung. Die Planung muss berücksichtigen, dass die fertige Würze nicht direkt abgefüllt werden kann, sondern dass eine zeitliche Verschiebung aufgrund der anschließenden Gärung und Lagerung notwendig ist.

Algorithmus Konsistenzprüfung
Versucht für jede Charge des Sudplans eine Belegung in den Gär- und Lagertanks zu finden. Der Algorithmus gibt für Chargen, für die keine Belegung gefunden

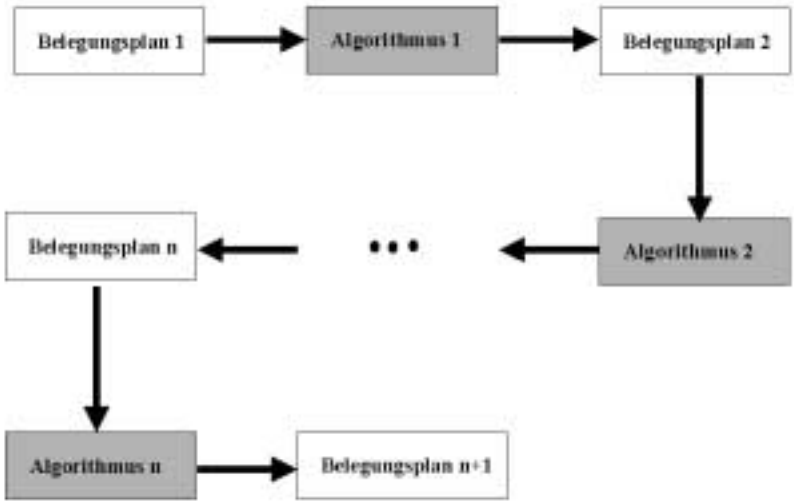


Abbildung 7.4: Baukastensystem der Algorithmen

werden kann, eine Warnung aus. Die resultierenden Tankbelegungen werden nicht an die Steuerung weitergegeben, da der Algorithmus ausschließlich der groben Abschätzung bezüglich der Realisierbarkeit des Sudplans gilt.

Fertigungsstufe 2: Bierabfüllung

Algorithmus Hauptsorten

Algorithmus zum Einplanen von Hauptsorten. Hauptsorten belegen die Schichten der ihnen zugeordneten Produktionseinheiten stets vollständig. Falls dadurch insgesamt ein zu hoher Produktbestand der betreffenden Hauptsorte entstünde, wird die Belegung unterbrochen und ein anderes Produkt an diese Stelle eingeplant.

Algorithmus Nebensorten

Algorithmus zum Einplanen der Nebensorten. Produktionsbedarfe werden zu Losen zusammengesetzt und in geeignete freie Kapazitäten eingesetzt. Ziel ist es, Rüstzeiten zu minimieren.

Algorithmus DrucktankAbbuchung

Bucht die in der Abfüllung eingeplanten Mengen von geeigneten Drucktanks ab. Für nicht einplanbare Abfüllchargen wird eine Warnung ausgegeben.

Algorithmus Filtration
Algorithmus zum Planen des Filterkellers. Dieser Algorithmus plant den Filterkeller ausgehend von den vorgesehenen Absätzen. Darüber hinaus werden für das Filtrat geeignete Drucktanks gesucht und von diesen abgebucht.

Eine genaue Darstellung der hier aufgeführten Algorithmen erfolgt in Kapitel 9.

7.3.5 C–Modul zur Planung

Wie in Abschnitt 7.2 dargestellt, ist das Planungssystem als Access–Applikation mit Schnittstellen zur Oracle–Datenbank geschrieben worden. Access bietet zwar mit *VBA* eine eigene Programmiersprache an, diese ist aber zum Durchführen komplexerer Planungsrechnungen nur bedingt geeignet. Daher werden alle Planungsläufe und Algorithmen in einem separaten C–Modul ausgeführt. Wichtig dabei ist ein effizientes Übermitteln aller Planeingangs- und Ausgangsdaten an das C–Modul. In Pas–Plan wurden dazu spezielle Ladefunktionen geschrieben, die zu einer drastisch verkürzten Laufzeit für die Datenim- und -exporte führen. Die C–Module sind vom übrigen Planungssystem derart entkoppelt worden, dass sie auch als eigenständige Lösung lauffähig bleiben. Die folgende Beschreibunge der Datenstrukturen und Algorithmen für die Planung beziehen sich auf die Realisierung als C–Modul.

7.3.6 Hierarchische Planung

In einem Planungslauf wird die Produktion für einen gegebenen Planungshorizont festgelegt. Der Planungshorizont kann kurzfristig sein und zum Beispiel die aktuelle Woche umfassen, er kann aber auch einen längerfristigen Zeitrahmen umspannen, wie etwa mehrere Wochen oder auch Monate. Je weiter die Planung jedoch in die Zukunft hineinreicht, desto unsicherer werden die Planungseingangsgrößen. Auf der anderen Seite wird die Anforderung an die Information aber auch geringer. So wird zum Beispiel eine Planung für das Folgejahr nicht auf einen Belegungsplan angewiesen sein, bei dem einzelnen Produktionschargen zu gewissen Zeiten den Anlagen zugeordnet sind. Vielmehr interessieren für die langfristige Planung nur grobe Mengenangaben der herzustellenden Produkte und eine grobe Abschätzung der Realisierbarkeit. Diesen Anforderungen kann durch eine hierarchische Planung Rechnung getragen werden. Dazu wird der Planungshorizont in drei Bereiche eingeteilt, die Tages-, Wochen- und Monatsplanung. Die Tagesplanung dient der Planung der näheren Zukunft, in der es auf genaue Angaben über Zeitpunkte und Reihenfolgen ankommt. Die Tagesplanung wird durch die parametrisierbaren Algorithmen der Belegungsplanung abgedeckt. Die Kapazitätsplanung wird für die Abschätzung der Produktion der weiteren Zukunft eingesetzt, hier also für die Planung im Wochen- und Monatsraster. Der Planungslauf arbeitet rückwärts vom Ende des Planungshorizonts zum aktuellen Datum. Ziel ist es, durch eine möglichst späte Produktion Bestandkosten zu minimieren. Zunächst

wird die Planung im Wochen- und Monatsraster durchgeführt. Wenn es nicht möglich ist, eine Produktionsmenge einer Periode zuzuordnen, wird sie in die vorherige Periode übertragen. Die Kapazitätsplanung ist dabei an die Belegungsplanung gekoppelt, d. h. die Produktionsmengen, die in der Kapazitätsplanung mangels Kapazität nicht erzeugt werden können, rutschen in den Planungszeitraum der Belegungsplanung. Eine genauere Beschreibung der Algorithmen erfolgt in Kapitel 9. Es sollte jedoch beachtet werden, dass bei einer Realisierung als C–Modul im Zuge der hierarchischen Planung verschiedene zusätzliche Probleme zu lösen sind. Insbesondere sind beim Übergang von Tages- zum Wochen- und zum Monatsraster eine Vielzahl von Berechnungen notwendig, etwa um die Anzahl der Absatztage auf die Intervallgrenzen hochzurechnen oder um Produktionsmengen den korrekten Zeiträumen zuzuordnen.
Zur Illustration ist in Abbildung 7.5 noch einmal das Grundprinzip der hierarchischen Planung dargestellt.

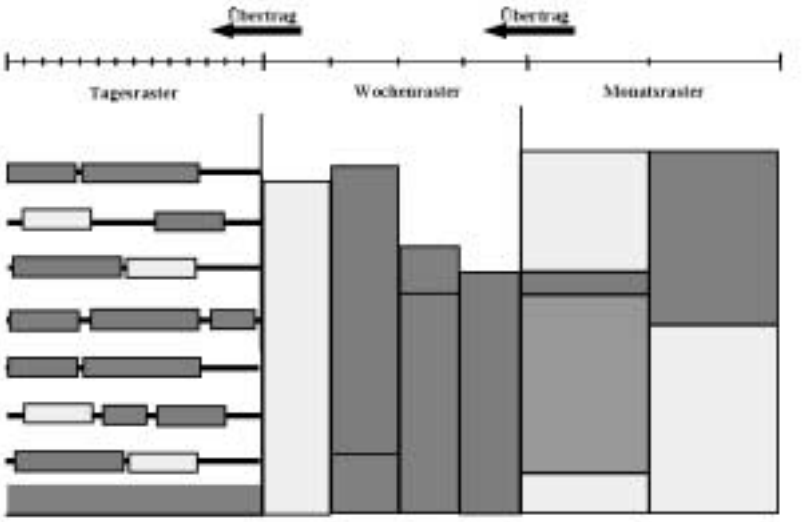


Abbildung 7.5: Kombinierte Kapazitäts- und Belegungsplanung

7.3.7 Rollierende Planung

Bei einem Planvorgang ist der Belegungsplan in der Regel nicht leer. Statt dessen liegen bereits bestimmte Aufträge fest, etwa Reinigungsarbeiten, Wartungsmaßnahmen und vor allem aber auch die Produktion, die während des Planungsvorgangs bereits durchgeführt wird. Zudem können den Planern im Zuge einer interaktiven Planung manuelle Planungsfunktionen zugebilligt werden. So sollte

die Möglichkeit bestehen, dass ein Teil der Planung von den verantwortlichen Planern manuell modifiziert und dann fixiert wird, während der verbleibende Plan mit Hilfe der Planungsalgorithmen neu berechnet wird. Um die genannten Problemstellungen modellieren zu können stellt Pas–Plan das Konzept der fixierten Aufträge bereit. Die Planer erhalten die Möglichkeit, den Aufträgen den Status *fixiert* zuzuordnen. Alle bereits in die Produktion eingelasteten Aufträge werden ebenso als fixiert gekennzeichnet. Bei einem Planungslauf werden alle fixierten Aufträge aus der Datenbank eingelesen und in den zunächst leeren Belegungsplan eingetragen. Im Folgeschritt können dann die Belegungsalgorithmen die verbleibenden Produktionsbedarfe berechnen und die zugehörigen Produktionschargen generieren und den Anlagen zuordnen. Insbesondere kann auf diese Weise eine rollierende Planung mit täglicher Anpassung gewährleistet werden.
Die theoretische Grundlage für dieses Plaungsproblem wurde in Teil I dieser Arbeit mit dem *Scheduling–Problem mit fixierten Jobs* gegeben.

7.4 Evaluation des Planes

Eine zentrale Komponente der Planung ist die grafische und tabellarische Darstellung der resultierenden Pläne. Darüber hinaus stellt PAS–Plan Bibliotheksfunktionen bereit, die eine schnelle Aufarbeitung und Interpretation der Pläne ermöglichen. Diese Funktionen arbeiten anhand des folgenden Schemas: Die Planungsperiode ist in einzelne Rasterintervalle, wie etwa Tage, Wochen oder Monate eingeteilt. Zudem liegt eine Liste von Aufträgen als Ergebnis der auszuwertenden Planung vor. Die Aufträge repräsentieren produzierte Mengen der einzelnen Produkte. Gleichzeitig belegen sie Kapazitäten. Wie im vorherigen Abschnitt bereits beschrieben, werden die entsprechenden Mengen für alle fixierten Aufträge vor jedem Planungslauf in die Rasterintervalle der Produktionseinheiten und Produkte eingetragen. In gleicher Weise werden am Ende eines Planungslaufs die Werte der neu erzeugten Fertigungsaufträge in die zugehörigen Rasterintervalle eingetragen. Nach dem Planungslauf werden dann alle Daten an das Datenbanksystem zurückgegeben und können grafisch angezeigt werden. Da die Rasterintervalle für jedes Produkt und jede Produktionseinheit gegeben sind, kann die grafische Darstellung und Evaluation auch aus Produktsicht oder Maschinensicht (Produktionseinheiten) erfolgen.

KAPITEL 8

Datenmodelle im C–Modul der Planung

In diesem Kapitel werden die zentralen Datenobjekte des C–Moduls der Planung vorgestellt. Das gesamte Datenkonzept im Planungssystem ist darauf ausgerichtet, Planungsläufe und Auswertungsrechnungen bei extrem niedrigen Rechenzeiten zu ermöglichen. Gleichzeitig wurde auf eine "saubere" Implementierung Wert gelegt, so dass das Planungssystem auch auf andere Anwendungsgebiete ohne großen Konvertierungsaufwand übertragbar wird. Daher sind die einzelnen Datenobjekte deutlich voneinander gekapselt, aber gleichzeitig durch vielfache Zeigerstrukturen verbunden, so dass in (fast) jeder Situation ein Direktzugriff auf die benötigten Daten möglich ist. Dieser Vorteil wird allerdings mit einer hohen Komplexität der Datenstrukturen erkauft. Ein weiteres wichtiges Element der Datenobjekte sind verschiedene Suchbäume, die ebenfalls einer speziellen Verzeigerung untereinander unterliegen. Hier ist der Zugriff auf die Daten, aber auch das Verwenden von Sortierung- und Suchfunktionen in diesen Daten mit vergleichsweise geringem Aufwand möglich.
In diesem Kapitel wird nicht das gesamte Datenmodell des Planungssystems vorgestellt, sondern nur einige ausgewählte Komponenten, die speziell in Hinblick auf die implementierten Schedulingalgorithmen von Bedeutung sind. Eine genaue Beschreibung von weiteren Objekten, wie die verwendeten Baum- und Listenstrukturen, die Kalenderdatemodelle oder Modelle von Grunddaten wie Herstellanweisungen, wird dagegen an dieser Stelle unterlassen, da sie wenig zum weiteren Verständnis des Planungssystems und der in dieser Arbeit betrachteten Problematik der Belegungsplanung beizutragen vermögen. Für eine weitere Beschreibung der Datenmodelle sei auf das Benutzerhandbuch des Planungssystems [Wer99] verwiesen.

8.1 Interne Darstellung des Belegungsplans

Eine wichtige zu klärende Frage ist die interne Darstellung des Belegungsplans. Es muss eine Repräsentation dessen geschaffen werden, was in der Plantafel als Gantt–Diagramm darstellt wird. Das Datenformat muss derart offen gehalten sein, dass alle Algorithmenbausteine auf dieses Format aufbauen können und eine Planung in allen Fertigungsstufen ermöglicht wird. In den folgenden Abschnitten werden die Objekte zum Verwalten des Belegungsplans dargestellt. Der Belegungsplan ist in der internen Darstellung eine Folge von Zubuchungen und Abbuchungen. Daher wird der Belegungsplan auch *Buchungsplan* genannt.

8.1.1 Buchungsplan

In dieser Arbeit ist der *Buchungsplan* speziell zur Unterstützung der Planung in Brauereien entwickelt worden, diese Datenstruktur kann aber auch für anderen Anwendungen der Prozessindustrie eingesetzt werden kann. Der *Buchungsplan* verwaltet die Belegungsinformationen von Produktionseinheiten und Ressourcen. Eine naheliegende Idee wäre, den *Buchungsplan* einfach als verkettete Liste oder als Suchbaum der einzelnen Belegungen (oder entsprechend der Theorie des Scheduling der *Jobs*) einer jeden Produktionseinheit aufzubauen. Ein solcher Ansatz wäre allerdings problematisch, denn bei verschiedenen zur gleichen Zeit oder leicht versetzt auf einer Produktionseinheit liegenden Chargen, könnte das belegte Volumen nicht zu jeder Zeit abgelesen werden. Vielmehr müssten zu einem Zeitpunkt t alle diesen Zeitpunkt umfassenden Belegungen identifiziert und dann deren Volumen berechnet werden. Als zweites wäre eine solche Datenstruktur nur geeignet, die Belegungen einzelner Jobs darzustellen. Bei einigen Produktionseinheiten, wie den Drucktanks oder anderen Puffern und Lagern, werden aber nur Materialien und Produkte in bestimmten Mengen zugeführt und irgendwann wieder in anderen Mengen abgebucht. Hier liegt also gar keine Belegung im Sinne eines *Jobs* vor. Der *Buchungsplan* hingegen unterstützt die Darstellung verschiedener Arten von Belegungen, insbesondere auch Mehrfachbelegungen von Tanks, die simultane Belegung verschiedener Ressourcen, die Inanspruchnahme von Personal und auch die Puffer- und Lagerbelegungen. Darüber hinaus lassen sich mittels des Buchungsplans überlappende Belegungen in einfacher Weise modellieren, beispielsweise der Vorgang, dass im Sudhaus eingemaischt wird, während die Vorgängercharge noch bearbeitet wird.
Jeder Produktionseinheit und jeder Ressource ist ein eigenes Objekt *Buchungsplan* zugeordnet, das alle Belegungsinformation speichert. Jede Belegung ist durch einen Start- und einen Endzeitpunkt spezifiziert, die in den Buchungsplan eingetragen werden. In manchen Fällen hat eine Belegung nicht nur eine zeitliche Dimension, sondern auch eine räumliche Dimension, das *Volumen*. Wenn zum Beispiel eine Charge von 100 *hl* für eine Woche in einem Tank gelagert wird, so

ist die zeitliche Dimension genau die Belegung über eine Woche und die räumliche Dimension das belegte Tankvolumen von 100 hl.

Die Einträge im Buchungsplan erfolgen in Form von *Buchungen*. Durch Buchungen werden Kapazitäten der Produktionseinheit belegt. Beschreibt die Produktionseinheit zum Beispiel einen Tank, so belegt eine Charge über den durch die Belegung definierten Zeitraum einen Teil des Tankvolumens. Zu Beginn des Auftrages wird vom vorhanden Volumen der entsprechende Anteil abgebucht. Zum Zeitpunkt des Endes des Auftrages wird das Volumen wieder freigegeben. Die Belegungsplanung muss garantieren, dass zu keinem Zeitpunkt das vorhandene Volumen geringer als durch die Abbuchungen angeforderte Volumen ausfällt.
Bei der Bestandsverwaltung von Ressourcen und Materialien kann der Buchungsplan ebenfalls eingesetzt wereden. Anders als bei Produktionseinheiten sind bei der Planung von Ressourcen keine Belegungen mit einem Beginn und einem Ende definiert. Statt dessen werden für Ressourcen nur Zuläufe und Abgänge vermerkt, die ebenfalls durch Buchungen repräsentiert werden können.
Wir interpretieren den Buchungsplan als ein Saldenkonto über die Zeit, in denen jeweils die Änderungen, also die Zu- und Abgänge des belegten Volumens dargestellt werden. Das Saldo zu einer Zeit t beschreibt das noch freie Volumen einer Produktionseinheit zur Zeit t. Durch Salden können verschiedenen Belegungsszenarien dargestellt werden. Repräsentiert die Produktionseinheit einen Tank oder eine andere Produktionseinheit mit einem festen Füllvolumen, so beschreibt das Saldo den verbleibenden freien Platz. Bei einer Belegung der Produktionseinheit durch eine Charge wird die Chargenmenge als Volumen der Belegung gewählt. Auf die gleiche Weise können Zu- und Abgänge bei Ressourcen modelliert werden.
Die zweite Möglichkeit ist, dass eine Produktionseinheit durch konkrete Aufträge belegt ist, die eigentlich kein Volumen sondern nur eine Dauer haben. Ein Beispiel ist die Abfüllung, bei der die Bearbeitung der Charge zwar eine zeitliche aber keine räumliche Dimension hat. In diesem Fall wird das Volumen einer Charge auf 1 gesetzt. Kann eine Produktionseinheit nur einzige Charge zur gleichen Zeit bearbeiten, so beträgt ihr Volumen ebenfalls 1. Bei einer Kapazität von n Chargen beträgt es n und falls die Produktionseinheit beliebig viele Chargen gleichzeitig bearbeiten kann, so wird das Volumen auf unendlich gesetzt.
Bei der Belegungsplanung können ebenso Mischformen vorkommen. So ist es möglich, dass Maschinen zwar grundsätzlich mehrere Aufträge gleichzeitig bearbeiten können, dies jedoch nicht für alle Aufträge gilt. Ein Beispiel bildet die Reinigung von Tanks. Die Reinigung eines Tanks wird als eigener Auftrag, also als eine Belegung, modelliert, zu der keine weiteren Chargen eingeplant werden dürfen. Derartige Belegungen werden mit einem Kennzeichen *exklusiv* versehen, das weitere Belegungen zur gleichen Zeit ausschließt.
Intern lässt sich diese Folge von Zu- und Abbuchungen als binärer Suchbaum repräsentieren. Dieser speichert sortiert nach den Zeitpunkten T_i der Abbuchung alle relevanten Buchungsinformationen, beispielsweise das Volumen oder den der

Belegung zugeordneten Maschinenauftrag. Jedem Zeitpunkt T_i entspricht also genau eine Buchung. Abbildung 8.1 verdeutlicht dies. Neben den Abbuchungsbaum

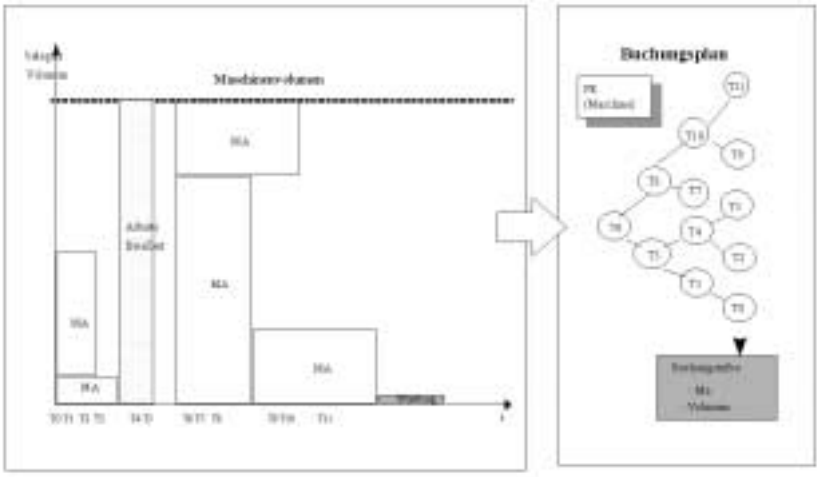


Abbildung 8.1: Belegungen einer Produktionseinheit mit Maschinenaufträgen

enthält der Buchungsplan als zweite Komponente einen Schichtplan. Dieser speichert die Folge der Schichten, zu denen auf der gegebenen Produktionseinheit gearbeitet werden kann, sortiert nach ihrer zeitlichen Reihenfolge. Jede Schicht ist dabei durch ihre Startzeit und ihre Endzeit gegeben. Einplanungen erfolgen nur zu den Schichtzeiten. Es ergibt sich das folgende Datenobjekt [1], wobei mit *BBaum* ein binären Suchbaum bezeichnet wird.

Objekt < *Buchungsplan* >		
BBaum	*s	(Suchbaum der Schichten)
BBaum	*t	(Suchbaum der Buchungszeitpunkte)

Dem Objekt *Buchungsplan* sind zwei Methoden zugeordnet:
AllocBuchungsplan
Kosntruktor: Erzeugt einen leeren Buchungsplan für eine Produktionseinheit. Zurückgegeben wird das Objekt *Buchungsplan*, also ein Zeiger auf die beiden Suchbäume. Zu Beginn und Ende des Buchungsplans wird jeweils eine Buchung mit leerem Volumen eingetragen. Damit wird der Planungszeitraum durch zwei Buchungen eingegrenzt.

[1] Datenobjekte und Variablen werden im Folgenden stets *kursiv* geschrieben.

FreeBuchungsplan
Destruktor: Freigabeprozedur des Buchungsplans. Alle Daten werden gelöscht und der reservierte Speicherplatz wieder freigegeben.

8.1.2 Buchung

Mit dem Buchungsplan eng verbunden ist das Datenobjekt *Buchung*. Eine *Buchung* füllt den zunächst leeren Buchungsplan "mit Leben". Buchungen geben den Zeitpunkt einer Veränderung der Belegungssituation der zugeordneten Maschine oder Ressource an. Wenn eine Belegung repräsentiert werden soll, wird also eine Buchung zu Beginn der Belegung erzeugt und eine zweite Gegenbuchung am Ende der Belegung, die die Kapazität wieder freigibt. Zusätzlich speichert die Buchung belegungsrelevante Informationen, zum Beispiel das Volumen einer Charge. Zur Vereinfachung des Zugriffs in diversen Suchverfahren ist der Buchung eine *buchung_id* zugeordnet, die im Buchungsbaum einen Direktzugriff auf die Buchung ermöglicht.

Objekt < *Buchung* >

time	t	(Zeit der Buchung)
produkt	**produkt*	(Produkt der Belegung)
double	*volumen*	(Volumen der Belegung)
void	**auftrag*	(eingeplanter Auftrag)
long	*kennung*	(Art der Belegung)
double	*saldo*	(belegtes Volumen kurz vor t)
buchung_id	*gegenbuchung_id*	(entgegensetzte Buchung)
BOOLEAN	*erste_buchung*	(TRUE $\rightarrow$ vollst. geleert)
BOOLEAN	*exklusiv*	(TRUE $\rightarrow$ exklusive Belegung)
void	**info*	(sonstige Informationen)

Wenn verschiedene Belegungen zum gleichen Zeitpunkt t starten oder enden, wird für jede Belegung eine eigene Buchung erzeugt. Daher können zu einem Zeitpunkt mehrere Einträge im Suchbaum der Buchungen existieren. Jeder Buchung ist ein Saldo zugeordnet. Das Saldo einer Buchung zur Zeit t gibt das gesamte belegte Volumen kurz vor dem Zeitpunkt t an. Alle Buchungen zur Zeit t haben das gleiche Saldo.
Dem Objekt *Buchung* sind die folgenden Methoden zugeordnet:
AddBuchung
Erzeugt eine neue Buchung im Buchungsplan der Produktionseinheit oder der Ressource. Das Saldo der folgenden Buchungen wird nicht tangiert. Die Gegenbuchung wird nicht gesetzt.

RemoveBuchung
Entfernt eine Buchung aus dem Buchungsplan einer Produktionseinheit oder Ressource.

FP_GetBuchung
Liefert zu einem Zeitpunkt t_0 die Buchung, die am dichtesten nach t_0 liegt.

FP_SetGegenBuchung
Erhält zwei Buchungen als Eingabe und lässt die Gegenbuchungszeiger dieser beiden Buchungen aufeinander zeigen.

FP_SaldoAnpassen
Erhöht das Saldo aller Buchungen in einem Intervall $]t_0, t_1]$ um ein vorgegebenes möglicherweise auch negatives Volumen.

GetSaldo
Bestimmt zu einem Zeitpunkt t das Saldo im Buchungsplan kurz vor t.

SummeBuchungen
Berechnet das Gesamtvolumen aller Buchungen im Buchungsplan zur Zeit t.

MoveBuchung
Verschiebt eine Buchung auf einen neuen Zeitpunkt t. Die Salden der Buchungen innerhalb des von der Verschiebung tangierten Intervalls werden angepasst.

FP_FirstBuchung
Liefert die erste Buchung im Buchungsplan.

FP_LastBuchung
Liefert die letzte Buchung im Buchungsplan.

FP_NextBuchung
Liefert die Nachfolgebuchung einer Buchung im Buchungsplan.

FP_PrevBuchung
Liefert die Vorgängerbuchung einer Buchung im Buchungsplan.

FirstBuchungZeit
Gibt von allen Buchungen eines Zeitpunktes t diejenige Buchung zurück, die im Binärbaum zuerst gespeichert ist. Gibt es keine Buchung zur Zeit t, wird die leere Buchung zurückgegeben.

LastBuchungZeit
Gibt von allen Buchungen eines Zeitpunktes t die letzte Buchung im Binärbaum zurück. Gibt es keine Buchung zur Zeit t, wird die leere Buchung zurückgegeben.

FP_Unbelegt
Prüft, ob Buchungen im *Buchungsplan* in einem Intervall $[t_0, t_1]$ liegen.

FP_VolumenAusreichend
Prüft, ob das noch freie Volumen des *Buchungsplan* im Intervall $[t_0, t_1]$ größer oder gleich dem in *Volumen* gegebenen Wert ist.

BereichBuchungExklusivNachher
Testet, ob zur Zeit ab der Buchung id_0 Buchungen exklusiv sein müssen, oder ob Mehrfachbelegungen erlaubt sind.

BereichBuchungExklusivVorher

Prüft, ob zur Zeit vor der Buchung id_0 Buchungen exklusiv sein müssen, oder ob Mehrfachbelegungen erlaubt sind.

BereichBuchungProduktNachher
Prüft, ob im Bereich nach einer gegebenen Buchung eine Belegung durch ein Produkt liegt.

8.1.3 Belegung

Wie bereits dargestellt, gibt es Buchungen zur Repräsentation einfacher Zu- und Abgänge von Materialien, es gibt aber auch Buchungen, die zur Darstellung eines Auftrages verwendet werden, der auf einer Produktionseinheit liegt. Intern wird eine solche Zuordnung eines Auftrages zu einer Produktionseinheit als *Belegung* verwaltet. Eine Belegung setzt sich dabei aus einem Beginn, dem Produktionsstart und dem Ende zusammen. Wenn Beginn und Produktionsstart nicht zusammenfallen, so liegt ein Umrüstvorgang vor, durch den die Produktionseinheit zwar belegt ist, aber an dem noch nichts produziert wird. Die Rüstdauer hängt dabei von der Vorgängerbelegung ab und sollte minimiert werden.

Objekt < *Belegung* >

time	*beginn*	(Beginn einer Belegung)
time	*prodstart*	(Beginn der Produktion (Ende Rüstzeit))
time	*ende*	(Ende einer Belegung)
double	*rüstdauer*	(Rüstzeit in h)
double	*rüstkosten*	(Kosten des Rüstvorgangs)
double	*kapazität*	(Dauer der Belegung in h)

8.1.4 BuchungAuftrag

Nach einer erfolgreichen Suche nach einer freien Kapazität für eine Produktionscharge werden die belegungsrelevanten Informationen in einem Objekt *buchung_auftrag* gespeichert. Der Buchungauftrag wird mit einer Funktion *PutBuchung* im Buchungsplan durch zwei Buchungen und Zeiger auf den zugehörigen Maschinenauftrag umgesetzt. Die Funktion *PutBuchung* setzt weiterhin in *BuchungAuftrag* zwei Zeiger auf die Buchungen im Buchungsplan. Zu beachten ist, dass nicht zwischen Beginn und Produktionsbeginn unterschieden wird, da dies im Buchungsplan nicht eingetragen wird. Derartige Informationen sind aber im zugehörigen Maschinenauftrag und damit über *BuchungAuftrag.auftrag* abrufbar.

Objekt $< BuchungAuftrag >$

time	*beginn*	(Zeitpunkt für erste Buchung)
time	*ende*	(Zeitpunkt für letzte Buchung)
produkt	*produkt*	(Produkt der Buchung)
maschine	*maschine*	(Produktionseinheit, auf der abgebucht wird)
verfeinerung	*vf*	(Verfeinerung für Buchung)
double	*menge*	(Menge des Auftrages)
double	*volumen*	(Volumen des Auftrages)
double	*kapazität*	(Kapazität in h)
void	**auftrag*	(zugeordneter Auftrag)
int	*kennung*	(Art der Belegung)
BOOLEAN	*erste_buchung*	(TRUE → vor Buchung leer)
BOOLEAN	*exklusiv*	
buchung_id	$buchung\_id_0$	(ID Buchung für start)
buchung_id	$buchung\_id_1$	(ID Buchung für ende)
void	**info*	(benutzerdefinierte Daten)

Dem Objekt *BuchungAuftrag* sind die folgenden beiden Funktionen zugeordnet:

FP_PutBuchung
Tragt die in *BuchungAuftrag* spezifizierten Buchungen in den Buchungsplan der zugeordneten Produktionseinheit ein. Zum Abbuchen wird zu den Zeitpunkten *Beginn* und *Ende* des Auftrages ein neuer Knoten im Suchbaum des Buchungsplans erzeugt. Beide Knoten verweisen aufeinander, so dass ein einfaches Identifizieren beim Entfernen des Auftrages möglich wird. Der Wert *saldo* aller Baumknoten ab dem Zeitpunkt der Buchung (exklusive) und bis zur Gegenbuchung (inklusive) erhöht sich um das im Auftrag gegebene Volumen. Die Funktion gibt einen Meldung zurück, wenn einer der Saldowerte der Knoten zwischen Beginn und Ende der Belegung das Volumen der Produktionseinheit überschreitet. Die Buchung wird aber in jedem Fall realisiert.

FP_DeleteBuchung
Gegenteil zu FP_PutBuchung

8.1.5 Erweiterung des Buchungsplans auf Ressourcen und n–fach Belegungen

Der Buchungsplan kann ebenso zum Planen von Ressourcen, wie etwa dem Personal, Materialien oder dem Leergut verwendet werden. Während das Personal genauso wie eine Produktionseinheit modelliert werden kann, das über die Zeit beansprucht und wieder freigegeben wird und nur in begrenzter Zahl zur Verfügung steht, gibt der Buchungsplan bei Ressourcen wie Materialien die Bestandsentwicklung über die Zeit wieder. Materialien können verbraucht werden, ebenso

aber durch Produktion oder Zukauf in ihrem Bestand erhöht werden. Die Bestandsveränderungen werden wie üblich durch Buchungen modelliert, nur sind die Buchungen isoliert, haben also keine Gegenbuchung. Die Buchung verweist nicht auf einen Maschinenauftrag sondern auf den Auftrag, der die Bestandsänderung verantwortet. Der Buchungsplan wird ebenfalls bei einer Kombination aus Maschinen und Ressourcenbelegung verwendet. Hier belegt ein Auftrag nicht nur eine Produktionseinheit sondern beansprucht zusätzliche Ressourcen. Diesen Ressourcen werden ebenfalls Buchungsbäume zugeordnet und die Aufgabe der Belegungsplanung ist, simultan für die Produktionseinheit und die Ressourcen eine Belegung zu finden, so dass zu keinem Zeitpunkt die Ressource über ihren Bestand hinaus verbraucht wird.

8.1.6 Lücken

Damit ein Auftrag auf einer Produktionseinheit eingeplant werden kann, muss für den Auftrag eine geeignete Einfügeposition bereitgestellt werden. Diese freien Bereiche auf einer Produktionseinheit werden in Form von *Lücken* verwaltet. Die *Lücken* stellen Kandidaten für eine Belegung dar. *Lücken* bestehen aus vier Elementen. Zwei Buchungen begrenzen die *Lücke* am Anfang und am Ende. Diese dienen dem leichteren Zugriff auf eine *Lücke*, da sie mit dem Buchungsplan gekoppelt werden können. Sie geben aber noch nicht die exakten Zeitpunkte des Beginns bzw. des Endes der *Lücke* an. Beginn und Ende der *Lücke* werden daher zusätzlich gespeichert.

Objekt < *Lücke* >

buchung_id	$buchung\_id_0$	(äußere Begrenzungsbuchung der Lücke)
buchung_id	$buchung\_id_1$	(äußere Begrenzungsbuchung der Lücke)
time	*beginn*	(ab hier darf belegt werden)
time	*ende*	(bis hier darf belegt werden)
double	*kapazität*	(Kapazität der Lücke in h)

Zentrales Element für die Belegungsplanung ist die Funktion zur Suche geeigneter Lücken für Produktionslose. Diese sucht der Reihe nach möglichst große Lücken im Buchungsplan, die die folgenden Rahmenbedingungen erfüllen:

- Die Lücken liegen in einem vorgegebenen Zeitintervall, gegeben durch einen frühsten Starttermin *fst* und einen spätesten Endtermin *set*. Die Zeitpunkte der beiden Begrenzungsbuchungen dürfen außerhalb dieses Zeitintervalls liegen, die beiden Zeitpunkte *Beginn* und *Ende* der Lücke müssen hingegen innerhalb des Intervalls liegen.

- Zwischen den Zeitpunkten der beiden begrenzenden Buchungen dürfen keine Buchungen mit dem Flag *exklusiv* liegen. Ansonsten sind Mehrfachbuchungen innerhalb der Lücke aber erlaubt.

- Wenn die Produktionseinheit nicht den Flag *produktmix* gesetzt hat, dürfen zwischen den beiden begrenzenden Buchungen keine Buchungen eines anderen Produktes liegen.

- Innerhalb der Lücke darf zu keinem Zeitpunkt das aktuelle belegte Volumen zuzüglich dem neu eingeplanten Volumen das Volumen der Produktionseinheit übersteigen.

Wie in den folgenden Kapiteln noch ausführlich beschrieben, arbeitet die Belegungsplanung im wesentlichen in zwei Schritten. Im ersten Schritt wird eine Charge gebildet und im zweiten Schritt werden alle Lücken als mögliche Zeiten, zu denen produziert werden kann, betrachtet und eine geeignete Lücke ausgewählt. Daher gibt es eine Funktion, welche die Lücken aufzählt und auswertet. Dazu wird der Buchungsplan jeder Produktionseinheit linear durchmustert und die aktuelle Lücke zurückgegeben, falls das Produkt auf dieser Lücke eingeplant werden kann. Am Anfang der Planung bildet der gesamte Buchungsplan eine Lücke, und nach dem sukzessiven Belegen der Maschinen ergeben sich die Lücken durch die Zwischenräume zwischen zwei Belegungen. Eine Lücke ist genau dann für ein Produkt geeignet, wenn die Produktionseinheit für das Produkt zulässig ist, die Termingrenzen eingehalten werden und die Produktionseinheit entweder nicht belegt ist, oder bereits belegt ist und eine Mehrfachbelegung zulässig ist. Die Lücke kann auch arbeitsfreie Zeiten einschließen. In diesem Falle muss bei der Belegung der Lücke dann berücksichtigt werden, dass keine arbeitsfreien Zeiten abgebucht werden.
Wenn für eine Charge eine Lücke gefunden wurde, kann ein Auftrag für die gegebene Charge erzeugt und eine Belegung innerhalb der Lücke erzeugt werden. Da die Charge in der Regel kleiner als die Lücke ist, muss innerhalb der Lücke nun eine geeignete Einfügeposition gewählt werden. Zur Vereinfachung wird momentan stets der Anfang oder das Ende eine Lücke ausgewählt.

8.1.7 Tabelle der temporären Maschinenaufträge

Da Buchungen nur die für den Buchungsplan relevanten Informationen speichern, sind alle weiteren Informationen wie zum Beispiel der Produktionsstart gesondert zu halten. Daher werden schon frühzeitig, bei der Suche nach geeigneten Lücken, Maschinenaufträge angelegt und die Belegungsinformationen innerhalb dieser Aufträge verwaltet. Da manche Belegungen nicht realisiert werden, ist zwischen diesen neu angelegten Maschinenaufträgen und den tatsächlich umgesetzten Aufträgen zu unterscheiden. Daher werden die neu angelegten Maschinenaufträge in einer Tabelle von temporären Maschinenaufträgen gespeichert. Bei der Realisierung werden dann neue Maschinenaufträge erzeugt und die Inhalte der temporären Maschinenaufträge in diese Maschinenaufträge kopiert. Zur Verwaltung der temporären Maschinenaufträge sind keine speziellen Funktionen definiert.

8.2 Auftragsnetze

In verschiedenen Bereichen der Brauerei setzt sich der Prozess aus einer Folge von einzelnen Prozessschritten zusammen, die nacheinander durchgeführt werden müssen. Als Beispiel sei die Folge der Arbeitsgänge Gären und Lagern genannt. In der Theorie des Scheduling spricht man in diesem Falle von *Präzedenzrelationen* der Jobs untereinander. In die Sprache der Produktionsplanung umgesetzt bedeutet dies, dass sich die einem Fertigungsauftrag zugeordnete Herstellanweisung aus mehreren Arbeitsgängen zusammensetzt. Diese bilden untereinander eine durch die Präzedenzrelationen gegebene Netzstruktur. Eine solche durch Präzedenzrelationen verbundene Menge von Arbeitsgängen wird *Auftragsnetz* genannt. Für jeden Arbeitsgang der Herstellanweisung muss eine Belegung gefunden und ein zugehöriger Maschinenauftrag erzeugt werden, so dass die Reihenfolgen der Arbeitsgänge berücksichtigt werden. Die Maschinenaufträge werden dabei sukzessive in der im Auftragsnetz gegebenen Reihenfolge eingeplant. Zur internen Realisierung wird ein Objekt *Auftragsnetz* definiert. Das Objekt *Auftragsnetz* unterstützt das Einplanen der Maschinenaufträge, indem es für jeden Arbeitsgang der Herstellanweisung die aktuellen Planungsinformationen speichert. In den Planungsalgorithmen werden die Arbeitsgänge unter Berücksichtigung der Präzedenzrelationen topologisch sortiert und anhand dieser Sortierung sukzessive eingeplant. Alle Zwischen- und Endergebnisse bezüglich der Belegungssituation eines Arbeitsgangs werden in einem Objekt *Netzeintrag* zwischengespeichert:

Objekt < *Netzeintag* >

ma	**ma*	(ggf. zugeordneter Maschinenauftrag)
verfeinerung	**verfeinerung*	(gewählte Verfeinerung)
BOOLEAN	*belegen*	(TRUE → Belegung wird erzeugt)
BOOLEAN	*beginn_fest*	(TRUE → Beginn Belegung fixiert)
BOOLEAN	*ende_fest*	(TRUE → Ende Belegung fixiert)
time	*fst*	(frühster Start)
time	*set*	(spätestes Ende)
time	*max_ende_prev*	(spätestes Ende aller Vorgänger)
time	*min_beginn_succ*	(frühster Start aller Nachfolger)
maschine	**maschine*	(vorgesehene Maschine)
buchung_auftrag	*buchung_auftrag*	(zu realisierender Buchungsauftrag)
belegung	*belegung*	(mögliche Belegung)
ruestinfo	*ruestinfo*	(Rüstdauer und Rüstkosten)

Beim Einplanen der Auftragsnetze ergibt sich die in Abbildung 8.2 dargestellte Datenstruktur: Grundlage zum Einplanen der Aufträge ist der *Buchungsplan*. Der Buchungsplan speichert, wie in Abschnitt 8.1 bereits dargestellt, die Beleginformationen der Produktionseinheiten als Folge von Buchungen. Die eigentlichen Belegungen werden nicht gespeichert. Statt dessen wird auf den der Belegung

zugrundeliegenden Maschinenauftrag verwiesen. Da die einzelnen Arbeitsgänge zunächst nur temporär gespeichert werden, bis ihre endgültige Einfügeposition feststeht, werden die Aufträge zunächst in einer Tabelle von temporären Maschinenaufträgen gespeichert. Zum Schluss werden die temporären Maschinenaufträge realisiert und durch tatsächliche Maschinenaufträge ersetzt, die am Ende des Planungslaufs an das Access–System zurückgegeben werden. Zur Netzeinpla-

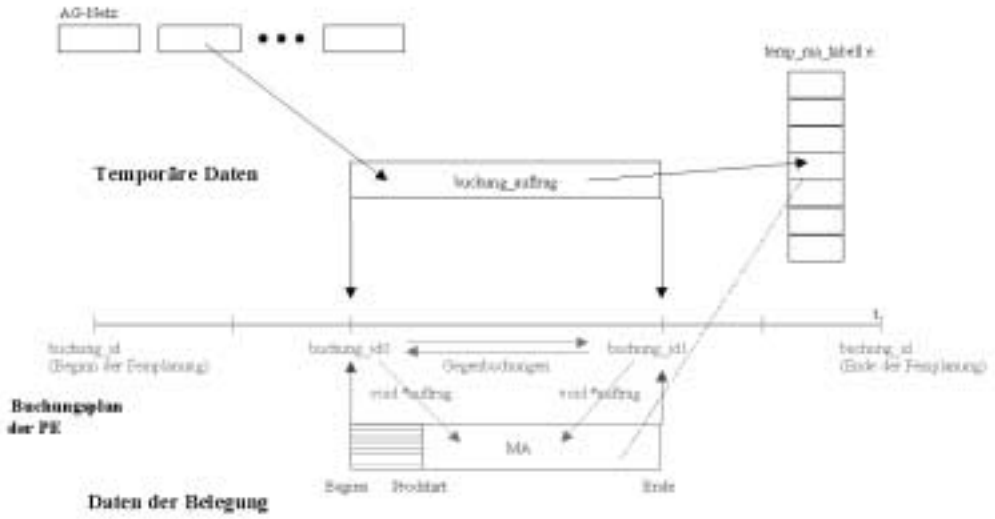


Abbildung 8.2: Beziehung zwischen Buchungsplan und Auftragnetz

nung wird ein rekursives Verfahren angewandt. Zuerst werden die Arbeitsgänge topologisch sortiert. Danach werden die Arbeitsgänge rekursiv vom ersten Arbeitsgang in der Sortierung bis zum letzten Arbeitsgang (Vorwärtsterminierung) in der Sortierung, oder umgekehrt vom letzten Arbeitsgang zum ersten Arbeitsgang (Rückwärtsterminierung), eingeplant. Dazu werden für jeden Arbeitsgang alle Einfügepositionen getestet. Wenn eine Position betrachtet worden ist, wird zunächst der folgende Arbeitsgang eingeplant und auf diese Weise auch mit den anderen Arbeitsgängen fortgefahren, bevor die nächste mögliche Einfügeposition betrachtet wird. Wenn schließlich der letzte Arbeitsgang erreicht wurde, kann das gesamte Auftragsnetz durch Belegungen realisiert werden, oder es wird zum vorherigen Arbeitsgang gesprungen und die nächste Lücke ausgewählt. Zu beachten ist, dass gewisse Arbeitsgänge nicht eingeplant werden müssen, da sie zu fixierten

Aufträgen gehören können. Diese Situation liegt zum Beispiel vor, wenn die am Anfang des Auftragsnetzes liegenden Aufträge bereits in Produktion sind und daher nicht mehr verändert werden können. Die Netzeinplanung verwendet drei zentrale Funktionen:

InitialisiereNetz
Erzeugt ein leeres Auftragsnetz und initialisiert den Speicherplatz. Dabei wird für jeden Arbeitsgang ein Netzeintrag erzeugt. Als zweites wird eine Tabelle von temporären Maschinenaufträgen erzeugt.

NetzEinplanung
Funktion zum Einplanen eines Auftragsnetzes. Gegeben ist ein Fertigungsauftrag mit einer zugehörigen Herstellanweisung und dem Auftragsnetz. Die Funktion sucht für jeden Arbeitsgang eine geeignete Maschine und eine Einfügeposition auf der Maschine, wobei die durch das Auftragsnetz gegebenen Reihenfolgebeziehungen der Arbeitsgänge untereinander beachtet werden. Die für den Abbuchungsbaum relevanten Ergebnisse, wie Beginn, Ende, Volumen, eines jeden Arbeitsgang werden in einem *BuchungsAuftrag* verwaltet, auf den der entsprechenden Netzeintrag verweist. Die auftragsrelevanten Daten, wie etwa der Produktionsbeginn, werden im zugeordneten Maschinenauftrag in der Tabelle der temporären Maschinenaufträge gespeichert.

RealisiereAGNetz
Realisiert ein Auftragsnetz. Ein Auftragsnetz wird realisiert, indem Maschinenaufträge erzeugt und Belegungen in die Belegungspläne eingetragen werden.

Im folgenden werden die drei zentralen Funktionen zur Netzeinplanung dargelegt.

Funktion 8.2.1 InitialisiereNetz

1. Speicherplatz für ein Auftragsnetz reservieren. Für die maximal mögliche Zahl von Arbeitsgängen wird Objekt *Auftragsnetz* alloziert.
2. Initialisierung jedes Netzeintrags: Frühster Start (*fst*), spätester Start (*sst*) der Belegung wird auf Beginn bzw. Ende des Buchungsplans gesetzt, alle sonstigen Werte auf Null.

Funktion 8.2.2 NetzEinplanung

Input:
- *plan* (der Belegungsplan),
- k (Engpassarbeitsgang)

Output: Fehlercode

/* Vom Engpassarbeitsgang k an wird vorwärts terminiert, nach dem letzten Arbeitsgang folgt die Rückwärtsterminierung der Arbeitsgänge vor k.*/

1. Setze also $i := k$.
2. Einplanung des Arbeitsgangs i:
   Fall 1 (Vorwärtsterminierung):
   FrühsterStart(i) := MAX(EndeArbeitsgang(i-1), FrühsterStart(i))
   Fall 2 (Rückwärtsterminierung):
   SpätestesEnde(i) := MIN(BeginnArbeitsgang(i+1), SpätestesEnde(i))
3. Schleife über für i erlaubten Produktionseinheiten m.
   3.1 *kapazität* aus Sollmenge und Losgröße ermitteln.
   Fall 1: m muss nicht beplant werden.
   Generiere *BuchungAuftrag* mit gegebenen Sollzeiten und trage *BuchungAuftrag* ein in *Netz_eintrag*. Ebenso temporären Maschinenauftrag generieren.
   Aufruf Funktion *Vorwärtsterminierung (i + 1)* bzw. für $i = n$, *Rückwärtsterminierung(i - 1)*
   Fall 2: m muss beplant werden:
   Schleife über die Lücken l auf m
   Finde *Belegung* in Lücke l.
   Falls *gefunden = TRUE* , dann
   BuchungAuftrag generieren und in in *Netz_eintrag* eintragen. Ebenso temporären Maschinenauftrag generieren.
   Aufruf Funktion *Vorwärtsterminierung(i + 1)* bzw. für $i = n$, *Rückwärtsterminierung (k - 1)*

Funktion 8.2.3 RealisiereAGNetz

Input:

Output:

1. Schleife über die Arbeitsgänge i
   1.1 Bestimme *Netzeintrag* zu i und nehme Informationen aus zugeordnetem *BuchungAuftrag*.
   1.2 Erzeuge Maschinenauftrag *ma* und kopiere Informationen aus *BuchungAuftrag* in *ma*
   1.3 Setze *BuchungAuftrag* mit Funktion *PutBuchung* in Buchungsplan der zugeordneten Produktionseinheit.
   1.4 Falls Belegung zu bestandserzeugenden Arbeitsgang gehört, trage die Produktionsmenge der Belegung in das entsprechende Rasterintervall für das Produkt der Belegung ein. Wenn Belegung mehrere Rasterintervalle umfasst, wird Produktionsmenge anteilig nach Produktionszeiten auf das Raster verteilt.

8.3 Plan

Alle Datenobjekte werden beim Aufruf des Planungslaufs von der Access–Oberfläche aus an die C–Module als Tabellen übergeben. Daraufhin werden intern

die Relationen zwischen den Planobjekten aufgebaut. Die Gesamtheit all dieser verketteten Daten wird durch das Datenobjekt *Plan* wiedergegeben. Der Plan gibt also alle für den Planungslauf benötigten Daten wieder. Damit umfasst er die Stammdaten wie Produkte, Produktionseinheiten, aber auch die Ergebnisse der Planung, also die Aufträge mit den resultierenden Belegungen auf den Produktionseinheiten. Zur internen Verwaltung zeigt der Plan auf alle oben genannten Objekte, die jeweils in Listen oder Suchbäumen eingeordnet sind. Abbildung 8.3 stellt einen Ausschnitt aus einem Plan dar.

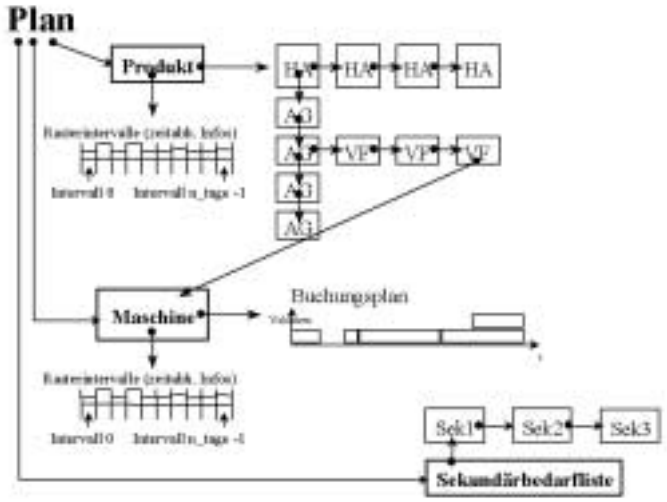


Abbildung 8.3: Der Plan und seine Objekte

Hierbei stehen Produkt und Maschine (Produktionseinheit) stellvertretend für die gesamte Liste der Produkte und Produktionseinheiten. Die Sekundärbedarfsliste sammelt den Sekundärbedarf, der aus der Einplanung der Produkte resultieren. Jedem Produkt ist eine Liste gültiger Herstellanweisungen zugeordnet. Die Herstellanweisung beschreibt die einzelnen Verfahrensschritte der Herstellung des Produktes als Folge von Arbeitsgängen. Jedem dieser Arbeitsgänge ist eine Liste von Verfeinerungen zugeordnet, die das Durchführen des Arbeitsgangs auf einer Produktionseinheit beschreiben. Folglich ist also eine Verfeinerung mittels eines Zeigers mit der Produktionseinheit verknüpft. Das Produkt verweist ferner auf eine Folge von Intervallen, die die zeitabhängigen Produktinformationen für jedes dieser Intervalle (Tag, Woche und Monat) speichern.
In ähnlicher Weise ist jeder Produktionseinheit eine Folge von Intervallen zugeordnet, die die zeitabhängigen maschinespezifischen Informationen speichern,

etwa die Belegungen in der kapazitativen Planung. Eine weitere zentrale Komponente für die Produktionseinheit ist der Buchungsplan, der über die aktuelle Belegungssituation in der Feinplanung Auskunft gibt.
Abbildung 8.4 verdeutlicht die Auswirkungen im Plan, wenn ein neuer Fertigungsauftrag zusammen mit seinen Maschinenaufträgen eingeplant wird. Der Ferti-

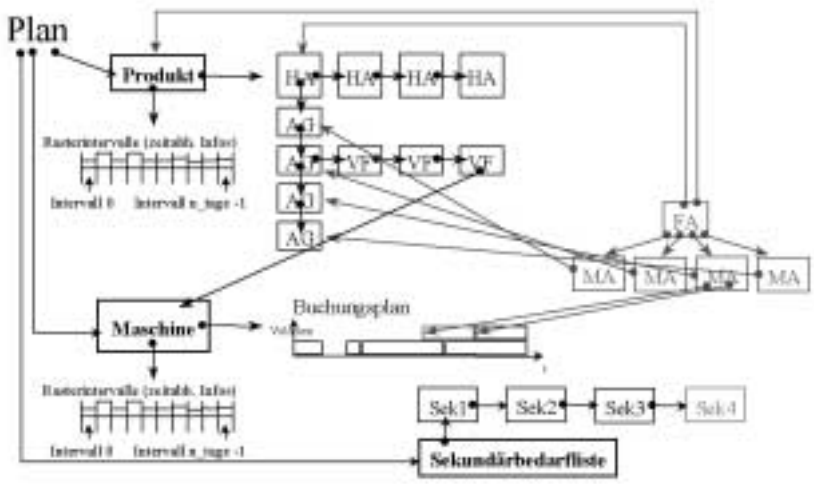


Abbildung 8.4: Einplanen eines Fertigungsauftrags

gungsauftrag ist für die Produktion einer Charge eines Produktes verantwortlich. Für dieses Produkt, wird dem Fertigungsauftrag eine geeignete Herstellanweisung zugeordnet. Für jeden Arbeitsgang der Herstellanweisung existiert ein Maschinenauftrag. Auf der anderen Seite entspricht einem Maschinenauftrag eine Belegung der Produktionseinheit, die sich durch Abbuchungen im Buchungsplan und Zeiger (Beginn und Ende) auf diese Buchungen widerspiegeln. Schließlich führt eine Belegung zu einer Sekundärbedarfsanforderung an die vorgelagerten Prozessstufen. Die Liste der Sekundärbedarfsaufträge des Plans wird also durch weitere Elemente verlängert. Beim nächsten Planungslauf, etwa beim Wechseln in eine andere Fertigungsstufe, wird der Sekundärbedarf den entsprechenden Produkten wiederum zugeordnet und im Rasterintervall als Bedarf eingetragen, bevor die neuerliche Kapazitäts- und Belegungsplanung beginnt. Zu beachten ist, dass bei einem Plan immer nur diejenigen Produkte, Produktionseinheiten und derjenige Sekundärbedarf betrachtet werden, der in der aktuellen Fertigungsstufe parametriert sind. Beim Aufruf der Funktion *Planungslauf* von Access aus, werden also auch nur diese Produkte und Produktionseinheiten dem Plan zugeordnet, alle anderen Produkte und Produktionseinheiten werden im C–Modul dagegen gar

nicht erst sichtbar. Auf diese Weise können unterschiedliche Produktionsbereiche autonom, aber durch die Sekundärbedarfsanforderungen verbunden, beplant werden.

8.4 Bibliotheksfunktionen aus Pas–Plan

Die im Planungslauf benötigten Daten werden als ASCII–Tabellen an das C–Modul zur Planung übergeben. Für den eigentlichen Planungslauf müssen die Daten daraufhin geeignet vorbereitet werden. Dazu werden spezielle Bibliotheksfunktionen des Planungssystems Pas–Plan der Firma Werum GmbH verwendet. Zu den erstellten Datenobjekten gehören:

- *Rasterintervalle:* Jedem Produkt und jeder Produktionseinheit wird eine Liste von Rasterintervallen zugeordnet. Der Planungszeitraum gliedert sich in Tage, Wochen und Monate. Für jeden Tag, jede Woche und jeden Monat wird ein Rasterintervall erzeugt, in dem die Informationen zu dem zugehörigen Zeitabschnitt gespeichert werden. Zu den Informationen gehören beispielsweise die belegte Kapazität der Produktionseinheit im Zeitintervall oder die hergestellte Menge für ein Produkt im Zeitintervall.

- *Herstellanweisung zu Produkten:* Jedem Produkt ist eine Liste von gültigen Herstellanweisungen zugeordnet. Ausgehend von der in den ASCII–Tabellen gegebenen Zuordnung von Herstellanweisungen zu Produkten wird für jedes Produkt eine verkettete Liste der zugehörigen Herstellanweisungen erzeugt.

- *Zugeordneter Fertigungsauftrag:* Vor jedem Planungslauf liegt bereits eine Liste von einzuplanenden Fertigungsaufträgen vor. Die Fertigungsaufträge resultieren aus fixierten Belegungen oder können von einem übergeordneten System (Grobplanung) übernommen werden. Die Fertigungsaufträge werden in einer verkettete Liste gespeichert und zeigen auf die ihnen zugeordneten Produkte und Maschinenaufträge.

- *Leerer Buchungsplan:* Jeder Produktionseinheit wird ein leerer Buchungsplan zugeordnet. Der Buchungsplan enthält eine leere Buchung am Beginn und am Ende des Planes sowie den Schichtkalender.

- *Leerer kapazitativer Belegungsplan:* Die in das Wochen- und Monatsraster fallenden Aufträge werden nicht im Buchungsplan, der das Tagesraster abdeckt, gespeichert, sondern in einem weiteren Belegungsplan für das Wochen- und Monatsraster. Dieser besteht aus einer verketteten Liste von Aufträgen und ist am Anfang leer.

- *Kalenderberechnung/Produkt–Zeit/Arbeitszeiten:* Für jedes Rasterintervall wird der zugehörige Kalendertermin (beispielsweise vom 3.4.99 0:00 bis

10.4.99 0:00) berechnet sowie die Summe der Arbeitszeiten und Absatzzeiten. Dabei werden Feiertage und Schichtmodelle berücksichtigt. Die Kalenderinformationen werden bei der späteren kapazitativen Planung verwendet.

- *Maschinenkapazitäten — Schichten im Schichtbaum der Produktionseinheiten eingetragen:* Ausgehend vom Werkskalender werden die Schichten, an denen gearbeitet wird, in den Schichtbaum des Buchungsplans eingetragen. Wenn bei der Belegungsplanung ein Belegung gesucht wird, können die in Frage kommenden Zeitabschnitte daraufhin untersucht werden, ob es sich um Arbeitszeiten handelt, oder um arbeitsfreie Zeiten, zu denen nicht produziert werden kann.

- *Abbuchen fixierter Maschinenaufträge und Fertigungsaufträge:* Ein wesentlicher Bestandteil der interaktiven Planung ist, Aufträge zu fixieren. Die fixierten Aufträge werden bei weiteren Planungsläufen nicht mehr verschoben. Fixierte Aufträge ermöglichen ebenfalls eine rollierende Planung. Die aktuell laufende Produktion spiegelt sich in für die Produktion freigegebene und gestartete Aufträge wieder, die allesamt fixiert sind. Vor einem Planungslauf liegt die Liste der fixierten Aufträge vor und wird in den Plan eingetragen, da sie Kapazitäten vorbelegen. Maschineaufträge werden im Buchungsplan abgebucht und Fertigungsaufträge werden im Belegungsplan der Produktionseinheiten in Form einer Verringerung der freien Kapazität abgebucht.

- *Produktion aus fixierten Aufträgen ermitteln:* Da fixierte Aufträge eine Belegung der Produktionseinheiten ergeben, ist ihnen eine Produktionsmenge zugeordnet. Diese reduziert den weiteren Produktionsbedarf und muss daher korrekt ermittelt, und dem zugehörigen Rasterintervall zugeordnet werden. Umfasst ein Auftrag mehrere Rasterintervalle, so wird die Produktionsmenge des Auftrages den Intervallen anteilig entsprechend der Arbeitszeiten der Intervalle zugeordnet.

- *Bestandsrechungen:* Vor jedem Planungslauf wird der aktuelle Bestand aus dem Lagerverwaltungssystem übernommen und in das Rasterintervall des Planbeginns eingetragen. Dieser Anfangsbestand verringert den Produktionsbedarf für die Periode. Die korrekte Abbuchung der Anfangsbestände ist für die rollierende Planung erforderlich, damit es nicht zu Überproduktionen am Planbeginn kommt.

Die zum Vorbereiten der Daten verwendeten Bibliotheksfunktionen setzten sich zusammen aus:
Ladefunktionen
Funktionen zum Laden der Inhalte der Datenbanktabellen in das C–Modul und zum Aufbau der verketteten Listen und Bäume.

Elementare Datenstrukturen
Funktionen zum Verwalten elementarer Datenstrukturen wie Listen und Bäume.

Relationen
Funktionen zum Aufbau der internen Verzeigerungen logisch zusammenhängender Objekte wie etwa Produkte und Herstellanweisungen.

Zeit- und Kalenderverwaltung

- Funktionen zum Erzeugen unterschiedlicher Kalender und Operationen auf den Kalendern (z. B. Test, ob ein Tag ein Absatztag ist).
- Funktionen zur Zeitbestimmung (Externe Zeit, Interne Rechenzeiten usw.) und zum Verwalten von Kalenderdatum und -wochen. Beispielsweise wird zu einem Datum der Index im Tageskalender geliefert, oder die Kalenderwoche im Jahr.

Berechnung der Maschinenkapazitäten
Funktionen zum Berechnen der belegten und noch verfügbaren Kapazität einer Maschine innerhalb einer gegebenen Schicht. Die Kapazität wird in Stunden berechnet.

Aufbau der Schichtenbäume der Produktionseinheiten
Verwaltungsfunktionen, welche die Schichten einer Produktionseinheit in den zugehörigen Suchbaum eintragen. Es werden nur die Schichten eingetragen, an denen auch produziert wird. Arbeitsfreie Tage werden nicht eingetragen.

Kostenauswertung
Funktionen zum Berechnen unterschiedlicher Kosten wie Herstellkosten oder Personalkosten eines Plans.

Rüsten
Funktionen zum Berechnen von Rüstzeit und Rüstkosten beim Produktwechsel. Die Rüstzeit wird anhand der Sachmerkmaltabellen der beiden betroffenen Produkte bestimmt.

KAPITEL 9

Planungsalgorithmen im PPS–System

In diesem Kapitel werden die im Planungslauf verwendeten Algorithmen beschrieben. Die Planung besteht dabei aus zwei Komponenten, der *kapazitativen Planung* und der *Belegungsplanung*. Im Gegensatz zum klassischen Scheduling des theoretischen Teils dieser Arbeit kann bei der Belgungsplanung nicht von einer Folge von *Jobs* ausgegangen werden, die mit einem Schedulingalgorithmus eingeplant werden können. Vielmehr werden im ersten Schritt Produktionslose gebildet, die daraufhin im zweiten Schritt als Job eingeplant werden müssen, so dass sich ein optimaler Bestandsverlauf über die Zeit ergibt und nur eine kleine Anzahl von Umstellzeiten durch Chargenwechsel resultieren. Damit ist die Belegungsplanung eine Hybridform aus den Problemen *Lot–Sizing* und *Scheduling*.
Die beiden Planungskomponenten kapazitative Planung und Belegungsplanung sind in einem zentralen Algorithmus eingebunden der die Planung koordiniert, alle Daten für die Planungsläufe bereitstellt, alle Vorbelegungen in den Plan einträgt und alle sonstigen Berechnungen durchführt. Diese Berechnungen umfassen zum Beispiel die Bestandsevaluationen, das Berechnen des Sekundärbedarfs oder das Ermitteln der prozentualen Maschinenauslastung. Die Planungsläufe erfolgen für jede Fertigungsstufe getrennt. Innerhalb der Planung werden die Informationen über die mittelfristig geplanten Absätze und Kundenaufträge verwendet. Die Planung leitet daraufhin unter Berücksichtigung der Bestände den Produktionsbedarf ab. Dies erfolgt in aller Regel als Kapazitätsplanung in der keine Reihenfolgen zwischen den Aufträgen gebildet und Rüstvorgänge nur pauschal berücksichtigt werden. Die Planung erfolgt zunächst für die letzte Fertigungsstufe, die Bierabfüllung. Als Ergebnis dieser Planung entstehen Fertigungsaufträge für die Abfüllung, Materialanforderungen an Leergut, die an das Bestellwesen gereicht werden und Sekundärbedarfsanforderungen bezüglich des Biers im Lagerkeller. In weiteren Planungsläufen wird dann in einem analogen Vorgehen die Bierherstellung geplant. Am Ende des Verfahrens sind die Fertigungsaufträge für

alle Fertigungsstufen sowie die Materialanforderungen an das Bestellwesen berechnet.
Wie schon in der Darstellung einiger in der Planung verwendeten Datenstrukturen ersichtlich, liegt den Algorithmen ein komplexes System von Datenobjekten und verzeigerten Strukturen zugrunde. Ziel ist, eine möglichst geringe Laufzeit für die Algorithmen zu ermöglichen. Auf der anderen Seite wird die Implementation der Algorithmen in einem derartig verzeigerten System deutlich komplexer, als in Standardimplementationen der Algorithmen. Um dennoch eine verständlicher Präsentation der Verfahren ermöglichen zu können, werden die Algorithmen in den folgenden Abschnitten nur deutlich vereinfacht dargestellt.

9.1 Grundüberlegungen zu den Schedulingalgorithmen

Im Theorieteil dieser Arbeit wurde die Approximierbarkeit wichtiger Standardschedulingprobleme untersucht. Dabei ergab sich, dass die klassischen Schedulingprobleme NP–schwer waren, selbst wenn die erschwerenden Nebenbedingungen aus der Praxis außer Acht gelassen werden und selbst bei einer in Brauereien üblichen konstanten Maschinenzahl und einfachen Zielfunktionen. Gleiches gilt unter der Annahme komplexerer Zielfunktionen und realistischerer Praxisbedingungen, wie sie in Kapitel 6 beschrieben wurden. Daher ist nicht zu erwarten, dass die in den einzelnen Fertigungsstufen auftretenden Schedulingprobleme in einer realistischen Laufzeit gelöst werden können. Im Theorieteil der Arbeit wurde außerdem gezeigt, dass selbst unter der Annahme von Maschinenausfallzeiten, wie etwa Wartungsarbeiten und arbeitsfreie Zeiten für das Minimieren des Makespans ein polynomielles Approximationsschema existiert, das Optimum also theoretisch in polynomieller Zeit beliebig gut approximiert werden kann. Allerdings ist die Konstante in der Laufzeit derart hoch, dass die zugehörigen Algorithmen für die Praxis dennoch nicht praktikabel sein dürften. Zudem ist bei den meisten Problemen nicht das Ziel, den Makespan zu minimieren, sondern es liegt in der Abfüllung eine Kombination der Teilziele *Bestandsminimierung*, *Due–Date Scheduling* (bei Kundenaufträgen) und dem *Minimieren der Umstellzeiten* vor. Bei der Bierherstellung dagegen liegt die Aufgabe darin, trotz einer Vielzahl von technischen Nebenbedingungen einen zulässigen Plan zu erzeugen, der den Vorgaben aus der Abfüllung genügen kann. Die einzelnen Planungsprobleme, wie sie sich aus den in Kapitel 6 dargestellten Anforderungen ergeben, haben alle ihre besonderen Charakteristika und daher sind alle hier betrachteten Algorithmen speziell für die gegebene Problemstellung entwickelt worden. Jedoch haben die vielen theoretischen Analysen einerseits zu einem deutlich verbesserten Verständnis der Probleme geführt und zum zweiten die Entwicklung bestimmter Techniken ermöglicht, die auch für die gegebene Anwendung erfolgversprechend scheinen. In

der kapazitativen Planung können beispielweise Techniken der linearen Programmierung eingesetzt werden. Bei der Belegungsplanung werden, motiviert durch die vergleichsweise guten Approximationsgüten bei den Standardschedulingproblemen, verschiedene *List*–Algorithmen verwendet. Diese Algorithmen sind für die Anwendung in der Brauindustrie geeignet, da sie nur eine geringe (lineare) Rechenzeit beanspruchen und daher schnell auf veränderte Planungssituationen reagieren können. Ein Algorithmus hingegen, bei dem das Ergebnis erst nach vielen Minuten oder sogar erst nach Stunden zu erwarten ist, ist in Anbetracht des hohen Störzeitenanteils in der Abfüllung und der vielen Unwegbarkeiten, die ein häufiges Neuplanen erfordern, sicherlich nur bedingt geeignet. Der zweite Vorteil der *List*–Algorithmen ist ihre einfache Struktur, so dass sie ohne großen Implementierungsaufwand an veränderte Rahmenbedingungen angepasst werden können. Beispielsweise ist ein derartiger Algorithmus einsetzbar, sowohl wenn Maschinen durchgängig verfügbar sind, aber auch — nach nur geringen Modifikationen — wenn Maschinen zu bestimmten Zeiten nicht genutzt werden können. Einher mit der Einfachheit der List–Algorithmen geht aber eine Suboptimalität dieser Verfahren. Bestehende Pläne sollten daher durch Nachoptimierungen verbessert werden. So können zum Beispiele verschiedene Aufträge getauscht werden, solange sich der Plan dadurch noch verbessert. Des Weiteren können Aufträge umgeplant werden, wenn dadurch vorteilhaftere Bestandskurven resultieren. Einige Ansätze für die Nachoptimierungen sind in der gegenwärtigen Version von Pas–Plan integriert, allerdings ist eine Nachoptimierung aufwendig, da schon Schwierigkeiten darin bestehen, überhaupt zulässige Pläne zu erzeugen. In diesem Falle können Algorithmen, deren Stärke in dem Vergleich einer Vielzahl von Lösungen besteht (was bei vielen Suchalgorithmen aber gerade der Fall ist) kaum zu Verbesserungen führen.
Die dritte aus der Theorie des Scheduling abgeleitete Technik ist das *Randomisieren*. In jüngster Zeit hat sich gezeigt, dass Algorithmen mit in Teilen zufallsgesteuerten Entscheidungen im Erwartungswert häufig eine hinreichend gute Performance aufweisen. Zudem sind auch die randomisierten Algorithmen von ihrer Struktur her sehr einfach und daher auch von der Seite der Softwareentwicklung und -wartung geeignet. Als Beispiel in der Brauindustrie ist der Algorithmus zur Planung im Sudhaus randomisiert: Für jedes Produkt wird der Anteil des Produktionsbedarfs am gesamten Produktionsbedarf bestimmt. Dieser Wert wird als Wahrscheinlichkeit dafür genommen, dass das Produkt als nächstes im Sudhaus eingeplant wird. Produkte mit hohem Produktionsbedarf, also die Hauptsorten, werden also tendenziell eher eingeplant, jedoch werden auch die Nebensorten entsprechend ihres prozentualen Anteils beachtet.
Im Folgenden werden die Planungsalgorithmen beschrieben. Dazu wird zunächst der zentrale Planungs- und Koordinationsalgorithmus vorgestellt. In Abschnitt 9.3 wird die kapazitative Planung im Rahmen der Grobplanung beschrieben. Hier beschränkt sich die Darstellung auch auf einen einfachen Greedy–Algorithmus. In Abschnitt 9.4 wird dann die Belegungsplanung beschrieben, die wir logisch der

Feinplanung zuordnen. Hier wird zwischen den Fertigungsstufen unterschieden, da wir in den beiden Fertigungsstufen unterschiedliche Algorithmen verwenden. Diese Algorithmen sind als Algorithmenbausteine in Pas–Plan integriert. Sie erhalten also jeweils einen— möglicherweise leeren — Belegungsplan als Eingabe und geben einen modifizierten Plan als Ausgabe an den nächsten Algorithmus weiter.

9.2 Zentraler Planungsalgorithmus

Algorithmus 9.2.1 ALGO_PLANUNG

Input: ASCII–Dateien der Planeingangsdaten

Output: ASCII–Dateien der Planausgangsdaten insbesondere neue Fertigungsaufträge, Maschinenaufträge, Sekundärbedarf

1. Laden des leeren Planes
2. Bereitstellen der Daten für Grob- und Feinplanung
3. Ermitteln des Produktionsbedarfs
4. Prüfen der Bestände
5. Differenzbildung
6. Festlegen der Produktion:
   - 6.1 Grobplanung: Kapazitative Planung für das Wochen und Monatsraster
   - 6.2 Feinplanung: Ausführen der Algorithmusbausteine für das Tagesraster
   - 6.3 Wiederholte Kapazitätsplanung
7. Auflösen des Sekundärbedarfs
8. Evaluieren des Plans

Die Schritte 1 und 2 laden zum Planungszeitpunkt alle aktuellen Daten in den Belegungsplan und bereiten die für die Planung benötigten Daten auf. Da alle Daten zur Laufzeit in den anfänglich leeren Plan geladen werden, ist eine rollierende Planung möglich, die den Planern die aktuelle Situation wiedergibt.
In Schritt 3 werden die einzuplanenden Produktionsmengen berechnet. Dazu werden die Sollbestände für alle Produkte und Planungsintervalle aus Aufträgen, Absätzen und Sollreichweiten berechnet. Der gesamte Produktionsbedarf eines Planungsintervalls ergibt sich aus

- Sollendbestand,
- Absätze,
- Aufträge,
- Sekundärbedarf.

Die erwarteten Bestände in Schritt 4 ergeben sich aus dem aktuellen Bestand und der Summe der eingeplanten Aufträge.

Bei der Differenzbildung wird der verbleibende Produktionsbedarf (Restproduktionsbedarf) als Differenz aus Sollbestand und tatsächlichen Bestand berechnet. Das Festlegen der Produktion erfolgt in einem Planungslauf, bei dem der Restproduktionsbedarf auf die noch nicht belegten Kapazitäten der Produktionseinheiten verteilt wird.
Der Planungslauf gliedert sich in drei Schritte:

- 1. Kapazitätsplanung mit langem Planungshorizont im Wochen- und Monatsraster.
- 2. Belegungsplanung mit kurzem Planungshorizont im Tagesraster. Hierbei werden Überträge aus der Kapazitätsplanung berücksichtigt.
- 3. Wiederholte Kapazitätsplanung unter Einbeziehung der in der Belegungsplanung erzeugten Produktionsaufträge.

Im Planungszeitraum, der durch die Belegungsplanung abgedeckt wird, werden Produktionsaufträge gebildet und exakt auf die jeweilige Produktionseinheit gelegt. Dabei werden genaue Start- und Endtermine für die Aufträge festgelegt und somit Reihenfolgen auf den jeweiligen Aufträgen der Produktionseinheiten festgelegt. Rüstzeiten werden berechnet und bei der Einplanung berücksichtigt.
Die Kapazitätsplanung hingegen bildet keine Reihenfolge der Aufträge untereinander. Statt dessen wird für jeden Auftrag nur das Planungsintervall bestimmt, in dem gefertigt werden soll. Dann wird in diesem Planungsintervall von der betroffenen Produktionseinheit soviel Kapazität abgebucht, wie für seine Produktion benötigt wird, Rüstzeiten werden dabei pauschal berücksichtigt. Bei diesem Verfahren existiert keine Reihenfolge von Aufträgen innerhalb eines Planungsintervalls, die auf der selben Produktionseinheit gefertigt werden.
Während man die Belegungsplanung für die nähere Zukunft einsetzen wird, in der es auf genaue Angaben über Zeitpunkte und Reihenfolgen ankommt, setzt man die Kapazitätsplanung für die Abschätzung der geplanten Produktion der weiteren Zukunft ein.
Die Kapazitätsplanung ist an die Belegungsplanung gekoppelt. Dies bedeutet, dass die Produktionsmengen, die in der Kapazitätsplanung mangels Kapazität nicht eingeplant werden konnten in den Planungszeitraum der Belegungsplanung vorverlegt werden.
Als Ergebnis der Planung entstehen Fertigungs- und Produktionsaufträge. Diese Aufträge sind temporär und werden bei jedem Planungslauf neu erzeugt. Es besteht die Möglichkeit, Aufträge zu fixieren oder sie an die Produktion weiterzugeben. In beiden Fällen sind sie gegen das Modifizieren bei Neuberechnungen geschützt.

9.2.1 Berechnung der Sollbestände

Für jedes Planungsintervall ist ein Lagerbestand definiert, der am Ende des Intervalls erreicht werden soll und Sollendbestand genannt wird. Er ist entweder explizit angegeben oder berechnet sich über die Sollreichweite aus den zukünftigen Absätzen. Wird der Sollendbestand nicht fest vorgegeben, berechnet er sich im Zeitpunkt t als Funktion der Absätze der nächsten, durch den Parameter Sollreichweite bestimmten Absatztage.

9.2.2 Berechnung der Sollproduktion

Die Sollproduktion wird vom Ende des Planungszeitraums zurück in die Gegenwart berechnet. Für ein Planungsintervall t berechnet sie sich nach folgender Formel:

$$
\begin{aligned}
\textit{Sollproduktion} \quad = \quad & \textit{Sollendbestand(t)} \\
- \quad & \textit{Sollendbestand(t-1)} \\
+ \quad & \textit{Absatz(t)} \\
+ \quad & \textit{Aufträge(t)} \\
+ \quad & \textit{Sekundärbedarf(t)}
\end{aligned}
$$

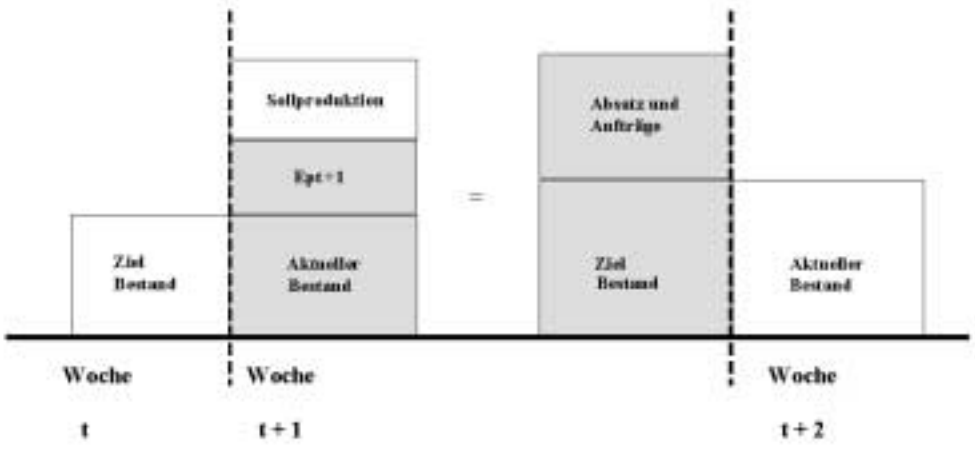
Graphisch lässt sich dies wie in Abbildung 9.1 darstellen. Die Sollproduktion definiert den Produktionsverlauf, der zu minimalen Beständen unter Berücksichtigung der Aufträge und Absätze führt. Tatsächliche Kapazitäten und Produktionskosten werden dabei außer Acht gelassen. Diese Faktoren werden in späteren Planungsphasen berücksichtigt.

9.2.3 Berechnung der erwarteten Bestandsentwicklung

Fixierte Produktionsaufträge haben für den Planungslauf im wesentlichen zwei Effekte:

- Sie belegen Kapazitäten der Produktionseinheiten, welche nicht weiter verwendet werden können
- Sie erzeugen Produkte und erhöhen dadurch den Lagerbestand

Zum Bestimmen der erwarteten Bestandsentwicklung werden, angefangen beim Planbeginn, die Produktion, Absätze, Aufträge und der Sekundärbedarf aufsummiert. Auf diese Weise kann für jedes Produkt und jedes Planintervall der erwartete Bestand am Anfang des jeweiligen Planintervalls berechnet werden. Ist dieser negativ, so liegt ein Übertrag an Bedarf (Absatz, Aufträge, Sekundärbedarf) vor.



$$\text{Zielbestand} = \sum_{\text{Zielreichweite}} \text{Absätze}$$

Abbildung 9.1: Ermitteln der Sollproduktion

9.3 Grobplanung: Kapazitative Planung

Die Grobplanung wird in allen Fertigungsstufen durch den gleichen Algorithmus abgedeckt. Das hier beschriebene Verfahren ist ein einfaches Greedy–Verfahren, bei dem die Produkte nach Prioritäten sortiert und die Planungsintervalle, beispielsweise die betrachteten Kalenderwochen rückwärts vom letzten zum ersten Intervall beplant werden. Dazu wird der Produktionsbedarf des Rasterintervalls von der Maschinenkapaizität abgebucht und alle nicht einplanbaren Mengen werden in das vorige Planungsintervall übertragen. Das spiegelt die Idee wieder, dass bei Kapazitätsengpässen früher mit der Produktion begonnen werden muss. Zu beachten ist, dass die kapazitative Planung nur einen groben Überblick vermittelt. So werden zwischen den Aufträgen keine Reihenfolgen gebildet und die Rüstvorgänge werden durch Pauschalen berücksichtigt. Ein genaueres Planen würde allerdings angesichts der Unsicherheit über die zu erwartenden Eingangsdaten möglicherweise ebenfalls ein falsches Bild bezüglich der Planungssituation liefern. Der folgende Algorithmus zeigt den Algorithmus zur kapazitativen Planung. Unter dem Begriff *Kapazität* wird für den Algorithmus, wie in der Terminologie der Produktionsplanung üblich, die noch auf einer Produktionseinheit

verbleibende freie Zeit in Stunden verstanden, welche noch verplant werden kann.

Algorithmus 9.3.1 Kapazitätsplanung

Input: Partiell belegter Belegungsplan

Output: Belegungsplan mit zusätzlichen Belegungen im Wochen- und Monatsraster

1. Schleife über alle Produkte p nach Planungspriorität sortiert
   1.1 Schleife rückwärts über alle Intervalle t der Kapazitätsplanung
   m := Bedarf(p) + Übertrag(p, t + 1).
   /* Produktionsbedarf m ist Summe des noch einzuplanenden Produktionsbedarfs von p und dem nicht eingeplanten Übertrag aus dem vorigen Intervall $t+1$ */.
   Schleife über alle für p zulässigen Produktionseinheiten i nach Priorität sortiert, solange $m > 0.0$.
   k := m/ Leistung(i)
   /* Die benötigte Kapazität k wird in Stunden gerechnet. Sie berechnet sich als Quotient des Produktionsbedarf m mit der Leistung der Produktionseinheit i. */
   Falls $k <$ *FreieKapazität(i)*, dann
   $m := 0.0$
   FreieKapazität(i) = FreieKapazität(i) −k
   Erzeuge Fertigungsauftrag fa über Belegung von Produkt p auf Produktionseinheit i mit Menge m. Beginn- und Endzeitpunkt von fa sind die Intervallgrenzen von t.
   sonst,
   m := m− FreieKapazität(i) ∗ Leistung(i).
   /* Die mögliche einzuplanende Menge berechnet sich als Produkt der freien Kapazität von i und der Leistung von i. */
   FreieKapazität(i) = 0.0.
   Erzeuge Fertigungsauftrag fa über die Belegung von Produktes p auf Maschine i mit Menge m_1. Beginn- und Endzeitpunkt des von fa sind die Intervallgrenzen von t.
   Übertrag(t) = m
   /* Übertrag im Intervall t ist nicht einplanbare Menge m */
   1.2 Übergang zu Planungsintervall $t-1$.

Es ist leicht ersichtlich, dass dieses einfache Planverfahren zu suboptimalen Plänen führen kann. Angenommen Produkt A kann auf den Abfüllinien 1 und 2 eingeplant werden, Produkt B aber nur auf Abfülllinie 1. Wenn Produkt A eine höhere Priorität als Produkt B hat, so kann es möglicherweise Maschine 1 in einem Intervall vollständig belegen, so dass Produkt B in ein vorheriges Intervall verlegt werden muss, auch wenn dies nicht der Fall wäre, wenn Produkt A auf der Abfülllinie 2 eingeplant worden wäre.
Das Problem der kapazitativen Planung lässt sich als nicht ganzzahliges lineares Programm modellieren. Damit kann in polynomieller Zeit eine *optimale* Lösung für die kapazitative Planung berechnet werden. Die obig beschriebenen Probleme beim Erstellen eines kapazitativen Planes entfallen also. Die Version der Planung mittels linearer Programmierung ist derzeit noch nicht im Planungssystem

integriert, jedoch wurde in dieser Arbeit eine eigenständige DV–Lösung für die Planung mittels linearer Programmierung entwickelt. Da die Systembeschreibung den Rahmen dieser Arbeit sprengen würde, verweisen wir an dieser Stelle auf die Dokumentation [Got99] des Systems.

9.4 Feinplanung: Belegungsplanung

Die Belegungsplanung arbeitet in dem Planungszeitraum, der durch das Tagesraster abgedeckt wird. Im Gegensatz zur Kapazitätsplanung spielen hier spezifische Rüstzeiten und damit Auftragsreihenfolgen und Losgrößen eine Rolle. Wir bereits vorher dargestellt, sind zwei Probleme innerhalb der Feinplanung zu lösen:

- Festlegen der Losgrößen
- Scheduling der aus der Losgrößenplanung resultierenden Jobs

Ein derartiges Planungsproblem kann auch als *preemptives* Schedulingproblem interpretiert werden. Dazu wird für jedes Produkt ein Job gebildet, der seinem gesamten Produktionsbedarf entspricht. Der Job kann dann eingeplant werden, wobei beliebige Unterbrechungen zulässig sind. Auf diese Weise bildet sich in natürlicher Weise ein Folge von Chargen. Allerdings ist eine Planung mit impliziter Losgrößenbestimmung aus verschiedenen Gründen problematisch:

- Beim preemptiven Scheduling müssen in den hier betrachteten Anwendungen Umstellzeiten durch Chargenwechsel berücksichtigt werden.
- Losgrößen orientieren sich an Richtwerten und sind durch Maximal- und Minimalwerte eingeschränkt.
- Bei einem mehrstufigen Prozess mit Präzedenzrelationen fällt das Abbilden eines preemptiven Jobs durch Fertigungsaufträge und zugehörigen mehrstufigen Rezepturen schwer.

Zum Vereinfachen der Planung werden Losgrößenplanung und Scheduling nur teilweise simultan durchgeführt. Konkret werden zunächst zeitlich verstreut liegende Restproduktionsbedarfe anhand bestimmter Steuerungsparameter zu größeren Produktionsmengen also einem Los zusammengefasst. Dem Los kann nun eine mehrstufige Herstellanweisung zugeordnet werden und daraufhin können geeignete Einfügepositionen unter Berücksichtigung der Umstellzeiten für die Arbeitsgänge der Herstellanweisung im Belegungsplan gesucht werden. Ist es nicht möglich, die Arbeitsgänge einzuplanen, wird das Los verkleinert und ein erneutes Einplanen versucht. Im positivem Fall wird das nächste Los zum nächsten Produkt gebildet und eingeplant, bis der gesamte Produktionsbedarf gedeckt ist. Durch die hier gewählte Form der Losgrößenberechnung nimmt man in Kauf,

dass man sich von der bestandsoptimalen Sollproduktion entfernt und temporär höhere Bestände bildet, als es für jeweiligen Planungszeitpunkt erforderlich ist. Es ist also generell zwischen großen Losen, und damit weniger Rüstvorgängen, und kleineren Beständen und häufigeren Rüstvorgängen abzuwägen. Der folgende Algorithmus gibt noch einmal das Grundprinzip der im PPS–System verwendeten Feinplanungsalgorithmen wieder:

Algorithmus 9.4.1 Feinplanung

1. Bilden eines Produktionsloses
2. Teste alle möglichen Positionen im Buchungsplan, zu denen das Los eingeplant werden kann.
3. Auswahl der günstigsten Position (z. B. mit minimalen Rüstkosten)
4. Belegen der optimalen Position und generieren eines zugehörigen MA

Zentrale Elemente für die Suche nach Einfügepositionen für die Jobs sind die in Abschnitt 8.1 vorgestellten Lücken in den Buchungsplänen der Maschinen. Diese Lücken im Buchungsplan einer Produktionseinheit bilden potentielle Einfügepositionen für ein Produktionslos. Die Suche nach freien Kapazitäten für die einem Arbeitsgang zugeordneten Charge erfolgt dabei bei den meisten im PPS–System verwendeten Einplanungsfunktionen nach einem ähnlichen Prinzip. Sequentiell werden alle Lücken einer für den Arbeitsgang in Frage kommenden Produktionseinheit durchmustert. Für jede dieser Lücken wird in einem zweiten Schritt nach einer günstigen Position für die Belegung innerhalb der Lücke gesucht. Die gefundene Position wird mit der Belegung für die bisher beste Lücke verglichen und die bessere der beiden gespeichert. Nachdem alle Lücken und alle Arbeitsgänge bearbeitet sind, werden die jeweils besten Belegungen der Arbeitsgänge realisiert.

Unter Berücksichtigung des gerade dargestellten Grundprinzips werden in der Belegungsplanung verschiedene Algorithmenbausteine verwendet. Wie in Abschnitt 7.3 bereits erörtert, handelt es sich dabei um eigenständige speziell auf das gegebene Teilproblem konfigurierte Algorithmen, die jeweils einen Belegungsplan erhalten und einen modifizierten Belegungsplan an den folgenden Algorithmusbaustein weitergeben. Die in dem PPS–System für Brauereien verwendeten Algorithmenbausteine werden im folgenden Abschnitt vorgestellt.

9.5 Algorithmenbausteine für die Fertigungsstufen

Für die Planung innerhalb der Bierherstellung ist die Hauptaufgabe, einen zulässigen Sudplan zu erstellen. Der Sudplan wird in der Regel für eine ganze Woche erzeugt. Für einen Sudplan muss zudem eine Belegung der Gär- und Lagertanks gefunden werden. Die Belegung der Gär und Lagertanks unterliegt allerding erheblichen Indeterminismen und kann auch aufgrund der in Abschnitt 6.2 angesprochenen Verschnittproblematik erst “zur Laufzeit” bestimmt werden. Daher

wird die Belegung der Gär- und Lagertanks nicht im PPS–System, sondern in der Feldebene geregelt. Auf der anderen Seite kann ein Sudplan nicht erstellt werden, ohne dass die Kapazitätsrestriktionen der Gär- und Lagertanks berücksichtigt wird. Daher wird eine Kompromisslösung gewählt: Im ersten Schritt wird ein Sudplan erstellt. Dazu wird ein eigener Algorithmusbaustein *ALGO_Sudplan* verwendet. In einem zweiten Algorithmusbaustein *ALGO_Konsistenzprüfung* wird im groben Abgeschätzt, ob dieser Sudplan in eine zulässige Belegung der Gär- und Lagertanks überführt werden kann. Ist dies der Fall, kann der Sudplan an die Prozesssteurung übermittelt werden. Die berechneten Belegungen der Gär- und Lagertanks dagegen werden nicht übermittelt, da sie zur Produktion aufgrund der tatsächlichen Gegebenheiten neu bestimmt werden müssen, was aber in der Praxis auch üblicherweise ohne größere Probleme gelingt.
Die Planung der Bierherstellung ist also ein typisches Beispiel dafür, dass es für die Planungsalgorithmen entscheidend ist, in welcher Ebene und von welchen Personen die Entscheidungen getroffen werden.
Die zweite Fertigungsstufe, die Bierabfüllung setzt sich aus zwei Bereichen zusammen, die in der gleichen Fertigungsstufe definiert sind. Zum Einen ist es die Planung des Filterkellers mit anschließender Überführung des Filtrats in die Drucktanks. Zum Anderen ist es die Belegung der Abfülllinien. Hier werden zwei Bausteine verwendet, einer zum Belegen von Langläuferprodukten, die die Abfülllinien durchgehend belegen und ein weiterer für die übrigen Produkte. Die Abfüllmengen werden schließlich in einem weiteren Baustein von den Drucktanks abgebucht. Die folgende Tabelle stellt die Algorithmusbausteine der beiden Fertigungsstufen in der Reihenfolge ihres Aufrufes dar.

Tabelle 9.1: Verwendete Algorithmenbausteine

Bierherstellung	**Bierabfüllung**
ALGO_Sudplan	ALGO_Langläufer
ALGO_Konsistenzprüfung	ALGO_Standardprodukte
	ALGO_Drucktankabbuchung
	ALGO_Filtration

9.6 Fertigungsstufe Bierherstellung: Sudhaus

Eingangsdaten für die Planung der Bierherstellung ist der Sekundärbedarf der Bierabfüllung. Zu beachten ist jedoch, dass zwischen der Würzeherstellung im Sudhaus und der späteren Abfüllung einige Wochen vergehen und dass der Sekundärbedarf daher nicht auf den Abfülltermin lauten, sondern auf einen um die Dauer der Bierherstellung versetzten Termin.
Die Planung im Sudhaus unterscheidet sich des Weiteren von der Planung der

übrigen Produktionsbereiche in den folgenden Aspekten:

- Die Problematik des Planens von Losgrößen entfällt, da die Chargengröße sortenspezifisch fest gegeben ist.
- Rüstvorgänge müssen nicht minimiert werden. Ziel ist statt dessen, die Sude unter Berücksichtigung der vorgegebenen Sekundärbedarfsanforderungen einzuplanen.
- Die Belegungen erfolgen nach einem Schema, das durch den Einmaischrhythmus (*Sudfolge*) bestimmt ist. Beträgt die Sudfolge eine Stunde, so wird im Sudwerk stündlich eine neue Charge bearbeitet.
- Chargen überlappen einander. Eine Charge kann also starten, während die Vorgängercharge noch bearbeitet wird.

Abbildung 9.2 verdeutlicht die Struktur eines Sudplanes bei einer angenommenen Zahl von drei Sudwerken und einer Sudfolge von einer Stunde. Die Chargen sind in der Abbildung versetzt dargestellt, um ihre Überlappung zu verdeutlichen.

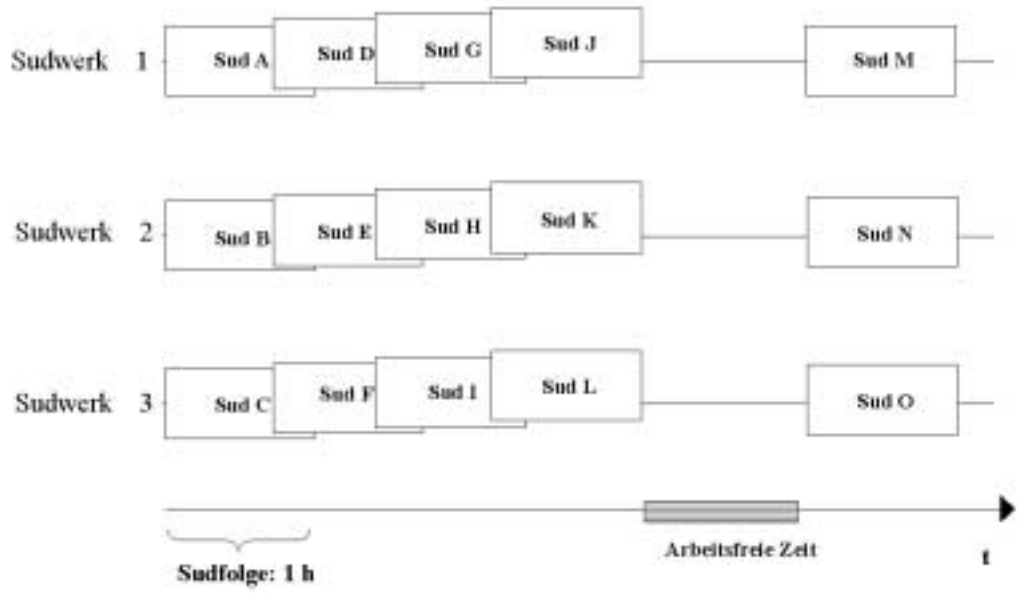


Abbildung 9.2: Belegung des Sudhauses

Die einzige vom Algorithmus zu fällende Entscheidung ist die Auswahl des Produktes für die jeweils nächste zu bearbeitende Charge für die einzelnen Sudhäuser. Der Bearbeitungszeitpunkt dagegen ist durch die Sudfolge bereits bestimmt. Die Chargengröße ist ebenso durch die Wahl des Produktes determiniert, da die Chargengrößen im Sudhaus (produktabhängig) fest sind.
Das im Algorithmusbaustein verwendete Verfahren ist randomisiert. Der Algorithmus durchmustert die Sudwerke der Reihe nach von Beginn bis Ende des Planungszeitraums und berechnet jeweils den nächsten Zeitpunkt, zu dem wieder eingemaischt werden kann. Dies ist der Beginn des Vorgängersuds zuzüglich der Sudfolge, falls die aktuelle Charge noch vor Schichtende eingeplant werden kann. Nach dem Berechnen des nächsten Startzeitpunktes wird ein Produkt ausgewählt. Die Produktauswahl erfolgt anhand zweier Kriterien: Für ein gegebenes Sudwerk wird das Produkt gewählt, das zum frühsten Zeitpunkt starten kann. In der Regel werden aber alle Produkte zur gleichen Zeit starten können, es sei denn, der betrachtete Zeitpunkt liegt nahe einem Schichtende, so dass nur noch Produkte mit kleinen Chargengröße bis zum Schichtende verarbeitet werden können. Ist nun eine Auswahl zwischen mehreren Produkten zu treffen, so kommt die Randomisierung ins Spiel. Würden nämlich immer nur die Produkte mit dem höchsten Produktionsbedarf eingeplant, so würden bei Kapazitätsengpässen bestimmte Produkte schließlich über längere Zeiträume gar nicht eingeplant werden, denn in jedem neuen Intervall kommen auch neue höhere Produktionsbedarfe für die Hauptsorten hinzu. Die Randomisierung ermöglicht dagegen, dass auch Nebensorten mit einer gewissen, aber vergleichsweise geringen Wahrscheinlichkeit eingeplant werden. Die Entscheidung für ein Produkt erfolgt durch Auslosen, wobei Produkte mit einem hohen Bedarf mit höherer Wahrscheinlichkeit ausgewählt werden. Genauer werden für alle in Frage kommenden Produkte die Bedarfe der folgenden k Planungsintervalle addiert, wobei k eine geeignete Konstante bezeichne. Sei P der Gesamtbedarf aller Produkte innerhalb der k Intervalle. Sei weiterhin p der Produktionsbedarf eines Produktes. Dann wird das Produkt mit Wahrscheinlichkeit p/P ausgewählt.
Zu beachten ist ferner, dass in der Fertigungsstufe Bierherstellung zwei Arten von Produkten auftreten, nämlich die Würze im Sudhaus und das Bier nach dem Lagern. Ebenso existieren verschiedenen Gruppen von Produktionseinheiten, nämlich die *Sudwerke* und die *Tanks*. Für die Planung im Sudhaus werden also nicht alle in der Fertigungsstufe *Bierherstellung* definierten Produkte eingeplant und auch nicht alle Produktionseinheiten betrachtet. Einplanbare Produkte und zugehörige Produktionseinheiten können allerdings über Produktgruppen und Produktionseinheitengruppen bestimmt werden.
Die folgenden Algorithmusparameter werden verwendet:

Tabelle 9.2: Parameter für Algorithmus *ALGO_Sudplan*

Name	Typ	Bedeutung
Sudfolge	Double	Sudfolge
ProdgruppeSud	String	Name der Produktgruppe Sud
Sudwerke	String	Name der PE-Gruppe Sudwerke

Algorithmus 9.6.1 ALGO_Sudplan

Input: Partiell belegter Plan, korrekt terminierter Sekundärbedarf der Abfüllung
Output: Sudplan mit zugehörigen Produktionsaufträgen

1. Schleife über die fixierten Maschinenaufträge *ma*.
   1.1 Falls Produktionseinheit i des *ma* Mitglied der Gruppe *Sudwerke*, dann
   fst(i) := max(*fst(i), ma.start + Sudfolge*)
   /* Frühster Start auf Produktionseinheit i bestimmt */
2. *weiter* := *TRUE*
3. Solange *weiter* = *TRUE*
   3.1 Schleife über Maschinen i der Gruppe *Sudwerke*
   3.1.1 *weiter* := *FALSE*
   3.1.2 Schleife über alle Produkte p der Produktgruppe *ProdgruppeSud*
   Aufruf Funktion *NächsteStartzeit* zum Berechnen der frühsten möglichen Startzeit von p.
   Lose unter allen Produkten mit gleichen frühsten Starttermin *fst* in der Funktion *ProduktAuswahl* ein Produkt $\bar{p}$ aus.
   Falls $\bar{p}$ gefunden, dann
   weiter = *TRUE*
   Einplanen von $\bar{p}$ durch generieren von Maschinenauftrag *ma* mit Werten:

art	:=	PRODUKTION
prodzeit	:=	Zeit in Sekunden zwischen *fst* und dem berechneten erwarteten Ende
start	:=	*fst*
prodstart	:=	*fst* /* keine Rüstzeit */
ende	:=	Berechnetes erwartetes Ende
maschine	:=	i
produkt	:=	$\bar{p}$
ha	:=	in *NächsteStartzeit* gewählte Herstellanweisung
ag	:=	einziger Arbeitsgang von *ha*
vf	:=	in *NächsteStartzeit* gewählte Verfeinerung

   Aufruf Funktion *PutBuchung* zum Belegen im Buchungsplan von i.
   fst(i) := *ma.start + Sudfolge*

Die Funktion *NächsteStartzeit* berechnet den frühsten möglichen Starttermin *next_start* für ein Produkt p auf einer Produktionseinheit i. Zusätzlich sucht die Funktion eine Herstellanweisung vom aktuellen Produkt p, die die Produktion von p auf der Produktionseinheit i beschreibt. Insbesondere besitzt diese Herstellanweisung zu ihrem einzigen Arbeitsgang eine Verfeinerung *vf*, die sich auf i bezieht. Ferner bestimmt die Herstellanweisung die feste Chargenmenge für das

Produkt p und auch das erwartete Ende der Belegung. Die Funktion *ProduktAuslosung* bestimmt nach dem bereits angesprochenen Zufallsverfahren ein Produkt zum Einplanen. Produkte mit höherem Produktionsbedarf werden mit höherer Wahrscheinlichkeit eingeplant.

9.7 Fertigungsstufe Bierherstellung: Konsistenzprüfung Gär- und Lagerkeller

Innerhalb der Fertigungsstufe Bierherstellung ist das Ziel der Planung, einen zulässigen Sudplan zu erstellen. Die folgende Belegung der Gär- und Lagertanks wird als Restriktion betrachtet. Der Algorithmusbaustein *Konsistenzprüfung* testet, ob die gegebenen Kapazitäten im Gär- und Lagerkeller genügen, den Sudplan umzusetzen. Dazu wird für jeden Maschinenauftrag des Sudhauses eine korrespondierende Belegung der Tanks gesucht. Kann keine Belegung gefunden werden, erfolgt eine Warnung, der Maschinenauftrag des Sudhauses wird aber nicht ausgeplant. Die endgültige Entscheidung bleibt der manuellen Planung überlassen.
Die Planung erfolgt, als sei die Gärung und Lagerung eine eigene Fertigungsstufe. Zu jeder Sorte des Sudplans gibt es ein weiteres Produkt nämlich das Bier nach dem Lagern. Dieses hat die Würzesorte in seiner Sekundärbedarfsliste. Dieser Zusammenhang wird in Abbildung 9.3 verdeutlicht.
Der Algorithmus betrachtet sukzessive die Maschinenaufträge des Sudhauses. Er sucht zu dem im Maschinenauftrag genannten Produkt, also zu der Bierwürze einer bestimmten Sorte, das korrespondierende Produkt nach dem Lagern. Dies ist hier also die fertiggestellte Biersorte. Zu diesem korrespondierenden Produkt ist eine Herstellanweisung für den Gär- und Lagerkeller definiert. Anhand dieser Herstellanweisung erzeugt der Algorithmus eine Belegung im Gär- und Lagerkeller. Der Start der Belegung entspricht dabei gerade dem Ende der durch den Maschinenauftrag gegebenen Belegung, da die Würze ohne Pufferung in die Tanks geführt wird.
Die folgenden Algorithmusparameter werden verwendet:

Tabelle 9.3: Parameter für Algorithmus *ALGO_Konsistenzprüfung*

Name	Typ	Bedeutung
Verschneidung	double	Erlaubte Zeit zum Verschneiden
ProduktgruppeSud	String	Name der Produktgruppe Sude
ProduktgruppeBier	String	Name der Produktgruppe Bier

Die Funktion *TankVorwärtsplanung* sucht eine Belegung im Gärtank und eine anschließende Belegung im Lagertank. Dabei ist zu beachten, dass ein Tank durch

Funktion 9.6.1 NächsteStartzeit

Input:
- *plan* (Belegungsplan)
- p (Produkt, für das Startzeit gesucht wird)
- i (Produktionseinheit, auf der Startzeit gesucht wird)
- *fst* (Frühst möglicher Start der einzuplanenden Charge auf der Produktionseinheit)

Output:
- *ha* (Herstellanweisung, nach der Produkt eingeplant werden kann)
- *ag* (Arbeitsgang der Herstellanweisung)
- *buchung_auftrag* (Daten für das mögliche Einplanen von p)

1. Schleife über im Planungszeitraum gültige Herstellanweisungen *ha* von p bis Verfeinerung *vf* zu i gefunden.
2. *menge* := Feste Chargengröße von *ha*.
3. m = *Durchlaufzeit(vf)* $*$ *menge*.
   /* Die b enötigte Kapazität m ist Produkt aus Abarbeitungszeit für Normcharge und Chargengröße. */
4. Schleife über Schichten *schicht_id* bis *gefunden*
   beginn := max(*fst, schicht_id.beginn*).
   Freie Kapazität *freie_kap* := *schicht_id.ende* - *beginn*.
   Falls *freie_kap* >= m, dann /* (einplanen möglich) */
   gefunden = *TRUE*.
5. Erzeuge *buchung_auftrag* mit Werten:

beginn	:=	*beginn*
ende	:=	*beginn* $+m$
produkt	:=	p
maschine	:=	i
vf	:=	vf
menge	:=	*menge*
volumen	:=	1 /* Charge hat keine räumliche Dimension */
kapazität	:=	m
kennung	:=	*PRODUKTION*

/* *buchung_auftrag* wird benutzt, falls p später zum Einplanen ausgelost wird. */

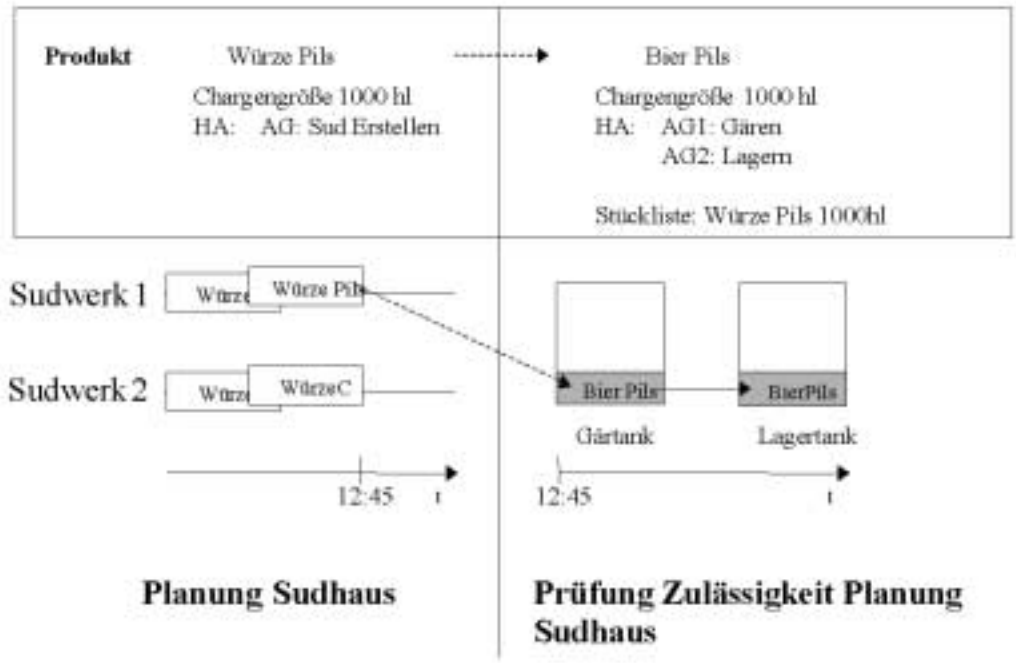


Abbildung 9.3: Konsistenzprüfung der Planung Sudhaus

Funktion 9.7.1 ProduktAuslosung

Input:
- *plan* (Belegungsplan)
- *produkt_tab* (Feld der Produkte, unter denen eines ausgelost wird)
- k (Zahl der Intervalle für die Bestandsrechnung)

Output:
- $\bar{p}$ (gewähltes Produkt)

1. *start* := gemeinsame frühste Startzeit der betrachteten Produkte.
2. *index* := Index des Tages im Intervallkalender zu *start*.
3. Schleife über Produkte p, unter denen ausgelost wird.
   3.1 Berechne für p Produktionsbedarf *bedarf* über k Rasterintervalle *index, ... , index + (k - 1)*:
   bedarf := sollendbestand
   + absatz
   + sekundärbedarf
   - bestand
   - produktion
   /* Die Werte *absatz*, *sekundaerbedarf*, *bestand* und *produktion* sind dabei bereits für jedes Rasterintervall vom zentralen Planungsalgorithmus berechnet worden. */
   prodbedarf_gesamt := Summe der Werte *bedarf* aller Produkte
4. /* Ordne jedem Produkt p den Quotienten aus Bedarf und Bedarf aller Produkte als Wahrscheinlichkeit *prob* für seine Auswahl zu. Ein Roulette entscheidet anhand der Wahrscheinlichkeiten, welches Produkt gewählt wird: */
   $t = 0$
   Schleife über Produkte p
   prob(p) := *bedarf(p)/ prodbedarf_gesamt*
   Zerlege Intervall $[0, 1]$ in durch die Wahrscheinlichkeiten $prob(p)$ gegebene Teilabschnitte und wähle Zufallszahl q aus $[0, 1]$.
5. Gebe Produkt $\bar{p}$ zurück, in dessen Teilabschnitt q fällt.

Algorithmus 9.7.1 ALGO_Konsistenzprüfung

Input: Partiell belegter Plan

Output: Plan erweitert um Belegungen in den Gär- und Lagertanks, die aus den Belegungen im Sudhaus korrespondieren oder Warnung, dass nicht belegt werden kann.

1. Schleife über die Maschinenaufträge *ma*
   1.1 Falls *ma.produkt* in *ProduktgruppeSud* enthalten, dann
   /* Für Charge des *ma* muss Belegung des korrespondierenden Produktes in Tanks gefunden werden.*/
   1.1.1 Aufruf Funktion *ZugeordnetesProdukt* um für Würze *ma.produkt* zugeordnetes Produkt (Biersorte) *p*als Endprodukt der Lagerung zu finden.
   1.1.2 Schleife über die Herstellanweisungen *ha* von *p* bis *gefunden = TRUE*.
   1.1.2.1 Initialisierung des zu *ha* gehörenden Auftragsnetztes
   1.1.2.2 Rekursiver Aufruf der Funktion *TankVorwärtsPlanung* mit 0–ten Arbeitsgang als Start.
   TankVorwärtsPlanung setzt Variable *gefunden* ggf. auf *TRUE*.
   1.1.3 Falls *gefunden = TRUE*, dann
   übertrage Belegungsdaten in den neuen Fertigungsauftrag *fa*.
   Erzeuge mit Funktion *RealisiereAGNetz* Maschinenaufträge
   sonst
   Warnung, dass für *ma* keine Belegung gefunden.

Funktion 9.7.2 ZugeordnetesProdukt

Input:
- *plan* (Belegungsplan)
- *ma.produkt* (Ausgangsprodukt)
- i_1 (Index der Produktgruppe des Ausgangsproduktes)
- i_2 (Index der Produktgruppe des Zielproduktes)

Output:
- *p* (Zielprodukt, also Produkt, welches *ma.produkt* in Stückliste enthält.)

1. Ausgabe *ERROR*, falls *ma.produkt* nicht in Produktgruppe mit Index i_1 enthalten.
2. Schleife über Produkte *p* bis *gefunden = TRUE*.
   2.1 Falls *p* in Produktgruppe mit Index i_2 enthalten, dann
   2.1.1 Schleife über Herstellanweisungen *ha* von *p*:
   Bestimme ersten Arbeitsgang *ag* von *ha*.
   Schleife über die Verfeinerungen *vf* von *ag*:
   Falls *ma.produkt* in Stückliste von *vf* enthalten, dann
   gefunden = TRUE.
   Gebe *p* zurück.

Funktion 9.7.3 TankVorwärtsPlanung

Input:
- p (Einzuplanendes Produkt)
- n (Nummer des aktuellen Arbeitsgangs)
- ma (Maschinenauftrag der Sudcharge)

Output:
- Belegungsdaten
- *gefunden* (*TRUE* oder *FALSE*)

1. Zeitliche Bindungen setzen:
   Fall $n = 0$: Frühster Start $fst := ma.ende$.
   Fall $n > 0$: Frühster Start $fst :=$ Ende vorheriger Arbeitsgang.
2. Schleife über Verfeinerungen vf des n–ten AG solange $weiter := TRUE$
   2.1 Kapazität $m :=$ *Durchlaufzeit(vf)*.
   2.2 Volumen $v :=$ *Chargenmenge(ha)*.
   2.3 Falls Tank der Verfeinerung vf ab Intervall von fst für durch m zuzüglich erlaubter Verschneidung gegebenen Zeit unbelegt, dann /* Fall 1: Tank ist leer */
   2.3.1 Für *buchung_auftrag* die folgenden Werte setzen:

beginn	=	fst
ende	=	$fst + verschneidung + m$
produkt	=	p
maschine	=	$vf.maschine$
vf	=	vf
volumen	=	*volumen*
menge	=	*Chargenmenge(ha)*
kennung	=	*PRODUKTION*
kapazität	=	Arbeitszeit im Intervall *[beginn, ende]*
erste_buchung	=	*TRUE*
exklusiv	=	*FALSE*

   2.3.2 *buchung_auftrag* mit *PutBuchung* einplanen.
   Sonst /* Fall 2: Tank belegt, aber ggf. gleiches Produkt und innerhalb Verschneidezeit */
   Gehe zu erster Buchung id_0 vor fst mit Flag *erste_buchung*.
   Falls $id_0.produkt = p$, dann
   falls $fst + m$ vor Ende der zu id_0 gehörenden Belegung liegt, dann
   /* Verschneidung zulässig */
   falls Volumen im Intervall ausreichend, dann
   Für *buchung_auftrag* die folgenden Werte setzen:

beginn	=	fst
ende	=	Ende der zu id_0 gehörenden Belegung
produkt	=	*produkt*
maschine	=	$vf.maschine$
vf	=	vf
volumen	=	*volumen*
menge	=	Chargenmenge des *ha*
kennung	=	*PRODUKTION*
kapazität	=	Arbeitszeit im Intervall *[beginn, ende]*
erste_buchung	=	*FALSE*
exklusiv	=	*FALSE*

   2.3.3 *buchung_auftrag* mit *PutBuchung* einplanen.
3. Falls betrachtete Arbeitsgang nicht letzter Arbeitsgang im Auftragsnetz, dann rekursiver Aufruf von *TankVorwärtsplanung* mit Arbeitsgang $n + 1$.

mehrere Chargen belegt werden darf, sofern sie zum gleichen Produkt gehören. Eine Einschränkung ist jedoch, dass zwischen dem Einfüllen der ersten Charge und der letzten Charge eines Tanks eine maximale Zeitspanne nicht überschritten werden darf. Diese Zeitspanne wird durch die Variable *Verschneidung* spezifiziert. Die Belegung wird erst gewertet, wenn die letzte Charge in den Tank gefüllt worden ist. Daher wird die Kapazität der ersten in den Tank gefüllten Charge um die *Verschneidung* verlängert.

9.8 Fertigungsstufe Bierabfüllung: Filtration

Der Algorithmusbaustein *FiltrationsPlanung* dient der vereinfachten Planung im Filterkeller. Geplant wird die Belegung der Filter und das anschließende Überführen des Filtrats in die Drucktanks. Während im Filterkeller Belegungen mit einem Beginn und einem Ende erzeugt werden, wird der Drucktank nur als Puffer beplant. Anstelle einer Belegung wird also einfach eine Zubuchung erzeugt, sobald das Filtrat in einen Drucktank geleitet wird und eine Abbuchung, wenn es an die Abfüllung weitergeleitet wird. Die Zeitspanne, in der es im Drucktank verbleibt, ist nicht vorgegeben. Abbildung 9.4 stellt die Zu- und Abbuchungen graphisch dar. Problematisch ist die getrennte Planung von Filtration und Abfüllung. Es kann nicht von vorneherein abgeschätzt werden, wann zugeführte Filtratmengen wieder von der Abfüllung beansprucht werden. Das freie Volumen der Tanks über die Zeit betrachtet ist also nicht bekannt.
Um überhaupt planen zu können, wird von einer festen Befüllzeit und einer festen Abgabezeit ausgegangen. Nach Abgabezeit und Reinigung können die Tanks wieder befüllt werden. Dabei wird von folgenden Vereinfachungen ausgegangen:

- Die Arbeitsschritte im Filterkeller werden zu einem Arbeitsgang zusammengefasst. Die zugehörigen Belegungen dürfen einander nicht überlappen.
- Die Durchlaufzeiten sind produktabhängig, aber fest.
- Das Filtrationsprofil wird vernachlässigt.

Die Planung des Filterkellers funktioniert nach dem gleichen Prinzip wie die Planung im Sudhaus. Die Filter werden mit festen Chargengrößen von links nach rechts belegt. Intern wird für jeden Filter (Produktionseinheit) der frühste Startetermin *fst* bestimmt, zu dem die nächste Filtrationscharge auf der Produktionseinheit eingeplant werden kann. Das Maschinenfeld enthält nur Maschinen der Produktionseinheitengruppe *PE_Filter*. Die Auswahl eines Produktes für die Filtrationscharge erfolgt analog dem im Abschnitt 9.6 beschriebenen randomisierten Verfahren zum Bestimmen von Sudchargen.
Bei der Drucktankbelegung werden die folgenden Kriterien überprüft:

- Zulässigkeit des Drucktanks für die Sorte,

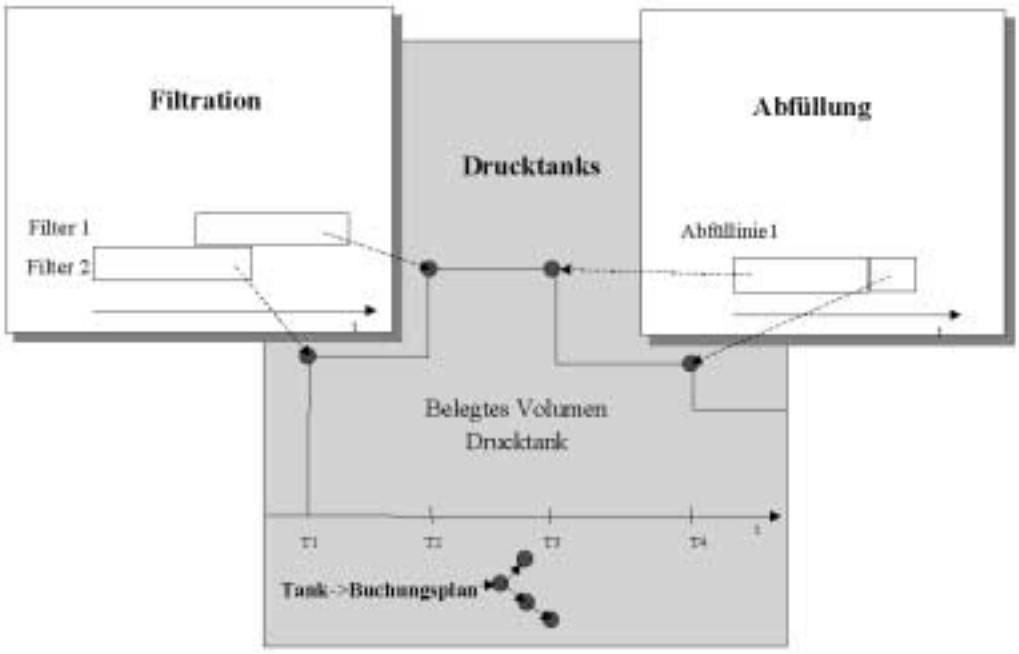


Abbildung 9.4: Drucktankbelegung zwischen Filtration und Abfüllung

Algorithmus 9.8.1 ALGO_Filtration

Input: Partiell belegter Plan

Output: Plan erweitert um Belegungen, die aus den Anforderungen der Abfüllung resultieren.

1. Schleife über Maschinen i der Gruppe *PE_Filter*
   1.1 Ordne i die folgenden Werte zu:
   start_zyklus := Beginn des Plans
   hilf_t := Beginn des Plan
2. Schleife über fixierten Maschinenaufträge ma:
   2.1 Falls Produktionseinheit i des ma Mitglied der Gruppe Gruppe *Filter*, dann
   fst(i) := max(*fst(i), ma.ende)*
   2.2 Falls $ma.start > hilf_t$, dann
   /* Start eines neuen Filtrationszyklus */
   2.2.1 Setze für i:
   start_zyklus := *ma.start*.
   hilf_t := *ma.ende* setzen.
3. Aufruf Funktion *FiltrationsPlanungMaschinen*
4. Aufruf Funktion *DrucktankBelegungen* zum Zuordnen der Filtratmengen zu Drucktanks.
5. Realisieren der in *FiltrationsPlanungMaschinen* erzeugten Maschinenaufträge.

- falls der Drucktank nicht leer ist, darf keine neue Sorte hinzugefügt werden,
- falls der Drucktank nicht leer ist, darf nur verschnitten werden, wenn die erste Befüllung des Druchtanks nicht zu weit zurückliegt,
- das Tankvolumen muss zur Aufnahme der Charge ausreichen.

Die folgenden Algorithmusparameter werden verwendet:

Tabelle 9.4: Parameter für Algorithmus *ALGO_Filtration*

Name	Typ	Bedeutung
Δ_t	Boolean	Befüllzeit + Abgabezeit in h
Filter	String	PE-Gruppe Filter
Drucktank	String	PE-Gruppe Drucktank
ProduktFiltrat	String	Produktgruppe Filtrat
Standzeit	Double	Max. Filtrationszeit bis Reinigung
Reinigungszeit	Double	Dauer der Reinigung

Der Algorithmus setzt sich aus zwei Komponenten zusammen: Im ersten Schritt werden alle Vorbelegungen durch fixierte Maschinenaufträge eingetragen. Im zweiten Schritt wird ein Belegungsplan des Filterkellers erstellt. Dabei ist zu beachten, dass nach einer maximalen Filtrationszeit, auch Standzeit genannt, wieder eine Reinigung erfolgen muss [1]. Daher wird im Algorithmus eine Variable *start_zyklus* mitgeführt, die den Startzeitpunkt bezeichnet, ab dem nach einer Reinigung wieder kontinuierlich produziert wird. Die Funktion *Filtrations-PlanungMaschinen* führt die Planung im Filterkeller durch. Die anschließende Funktion *DrucktankBelegungen* bucht die erzeugten Mengen in Drucktanks ab. Die Funktion *FindeNächsteStartzeitFiltration* berechnet den frühsten möglichen Starttermin *next_start* von p auf i. Dabei berücksichtigt sie die Schichteinteilung, die Standzeit und Reinigungszeiten. Zusätzlich findet sie eine Herstellanweisung, nach der auf i produziert werden kann. Die Herstellanweisung hat also zu ihrem einzigen Arbeitsgang eine zu i gehörende Verfeinerung *vf*. Die Herstellanweisung bestimmt die feste Chargenmenge und auch das erwartete Ende der Belegung. Falls die Startzeit nicht dem Ende der letzten Charge auf i entspricht, beginnt ein neuer Filtrationszyklus. Dann wird *start_zyklus* auf den neuen Start gesetzt.

[1] Die hier beschriebene Problematik entspricht dem im Teil I dieser Arbeit vorgestellten Problem des *Maintenance–Scheduling*.

Funktion 9.8.1 FiltrationsPlanungMaschinen

Input:
- *plan* (Belegungsplan)
- *zahl_maschinen* (Zahl der betrachteten Produktionseinheiten)
- *standzeit* (Maximale Filtrationszeit bis Reinigung)
- *reinigungszeit* (Dauer der Reinigung)
- i_1 (Index der Produktgruppe $Filtrat$)
- *ProduktAuslosung* (Funktion zur Auswahl eines Produktes zum Einplanen)

Output:
- Maschinenaufträge

1. $weiter = TRUE$
2. Solange $weiter = TRUE$
   - 2.1 Schleife über Produktionseinheiten i:
     - 2.1.1 Filtrationsgeschwindigkeit von i feststellen: Schleife über alle Produkte, Herstellanweisungen und Verfeinerungen des ersten Arbeitsgangs der Herstellanweisung bis eine auf i verweisenden Verfeinerung *vf* gefunden wird. /* Filtrationsgeschwindigkeit von i ist in Verfeinerung *vf* genannt. */
     - 2.1.2 Schleife über Produkte p der Gruppe *ProduktFiltrat*
       Aufruf Funktion *NächsteStartzeitFiltration*.
       Lose unter allen Produkten mit gleichem frühsten Starttermin *fst* in der Funktion *ProduktAuswahl* eine Produkt $\bar{p}$ aus.
       Falls $\bar{p}$ gefunden, dann
       $weiter = TRUE$
       Einplanen von $\bar{p}$ durch generieren von Maschinenauftrag *ma* mit Werten:

art	=	*PRODUKTION*
prodzeit	=	Zeit in Sekunden zwischen *fst* und berechneten erwarteten Ende
start	=	*fst*
prodstart	=	*start* /* keine Rüstzeit */
ende	=	berechnetes erwartetes Ende
maschine	=	i
produkt	=	$\bar{p}$
ha	=	in *NächsteStartzeitFiltration* gewählte Herstellanweisung
ag	=	einziger Arbeitsgang von *ha*
vf	=	in *NächsteStartzeitFiltration* gewählte Verfeinerung

       Aufruf Funktion *PutBuchung* zum Erzeugen einer Belegung im Buchungsplan von i.
     - 2.1.3 Frühster Start *fst* von i auf *ma.ende* verschieben.

Funktion 9.8.2 FindeNächsteStartzeitFiltration

Input:
- *plan* (Belegungsplan)
- *i* (Maschine, auf der Startzeit gesucht wird)
- *standzeit* (Maximale Filtrationszeit)
- *reinigungszeit* (Dauer der Reinigung)
- *start_zyklus* (Start des letzten Filtrationszyklus)
- *fst* (Maschine ist bis fst belegt)
- *filtrationsgeschw* (Durchlaufzeit der Filtration)

Output:
- *buchung_auftrag* (Daten, die für das spätere Einplanen benötigt werden)
- *next_start* (Nächste Startzeit)

1. Falls *belegt_bis* − *start_zyklus* > *standzeit*, dann
   1.1 /* Es muss wieder gereinigt werden */
   belegt_bis += *reinigungszeit*
   start_zyklus := *belegt_bis*
   1.2 Finde Schicht *schicht_id* , in der *belegt_bis* liegt.
   1.3 *weiter* = *TRUE*
   1.4 Kapazität *m* := min(*schicht_id.ende* - *belegt_bis, standzeit)*
   1.5 *menge* = *m* * *filtrationsgeschw*
2. Falls *menge* < minimal zulässige Chargengröße, dann
   schicht_id := Nächste Schicht
   belegt_bis := *schicht_id.beginn*
   start_zyklus := *belegt_bis*
   Schleife über Schichten *schicht_id*, bis arbeitsfreie Zeit kommt:
   weiter := *TRUE*
   Kapazität *m* := min(*schicht_id.ende* - *belegt_bis, standzeit)*
   menge := *m* * *filtrationsgeschw*
3. Setze berechnete Belegungsdaten in *buchung_auftrag*.
   /* *buchung_auftrag* wird verwendet, falls *p* später für das Einplanen ausgelost wird. */

beginn	=	*belegt_bis*
ende	=	*Beginn* +*m*
produkt	=	*p*
maschine	=	*i*
vf	=	*vf*
menge	=	*menge*
volumen	=	1 /* keine räumliche Dimension */
kapazität	=	*m*
kennung	=	*PRODUKTION*

Funktion 9.8.1 DrucktankBelegungen

Input:
- *plan*
- i_3 (Index der Produktionseinheitengruppe der Drucktanks)
- Δ_t (Befüllzeit zuzüglich Abgabezeit in h)

Output:

1. Schleife über temporär erzeugte Maschinenaufträge *ma*.
   1.1 *gefunden* := *FALSE*
   1.2 t := *ma.ende*
   1.3 p := *ma.produkt*
   1.4 /* Suche Drucktank, für den Belegung erzeugt werden kann*/
   Schleife über Produktionseinheiten i der Gruppe *Drucktank*.
      1.4.1 *weiter* := *TRUE*
      Fall 1: Drucktank ist leer, gereinigt und hat ausreichendes Volumen:
         gefunden := *TRUE*
         Erzeuge neue Buchung *buchung_auftrag* in *buchungsplan* von i mit den Werten:

volumen	:=	*ma.menge*
kennung	:=	*RESSOURCE*
auftrag	:=	*NULL* /* keine Belegung zu Maschinenauftrag */
erste_buchung	:=	*TRUE*
exklusiv	:=	*FALSE*
entleerung	:=	*FALSE*

      Fall 2: Tank ist nicht leer, aber das gleiche Produkt, wie *ma.produkt* liegt im Tank und Restvolumen ist für *ma.volumen* ausreichend:
         gefunden := *TRUE*
         Erzeuge neue Buchung *buchung_auftrag* in *buchungsplan* von i mit den Werten:

volumen	:=	*ma.menge*
kennung	:=	*RESSOURCE*
auftrag	:=	*NULL* /* keine Belegung zu Maschinenauftrag */
erste_buchung	:=	*FALSE*
exklusiv	:=	*FALSE*
entleerung	:=	*FALSE*

      gefunden := *TRUE*
   1.5 Falls *gefunden* = *FALSE* Warnung, dass für Maschinenauftrag kein Drucktank bereitsteht.

9.9 Fertigungsstufe Bierabfüllung: Langläufer

In der Abfüllung werden gewisse Abfülllinien durchgehend mit einem Produkt belegt. Derartige Produkte sind die im Unternehmen gefertigten Hauptsorten. So kann beispielsweise auf mehreren Abfülllinien über alle Schichten hinweg Pils gefahren werden, wenn Pils die Hauptsorte darstellt. Nur für die Nebensorten muss dann eine Belegungsplanung durchgeführt werden.

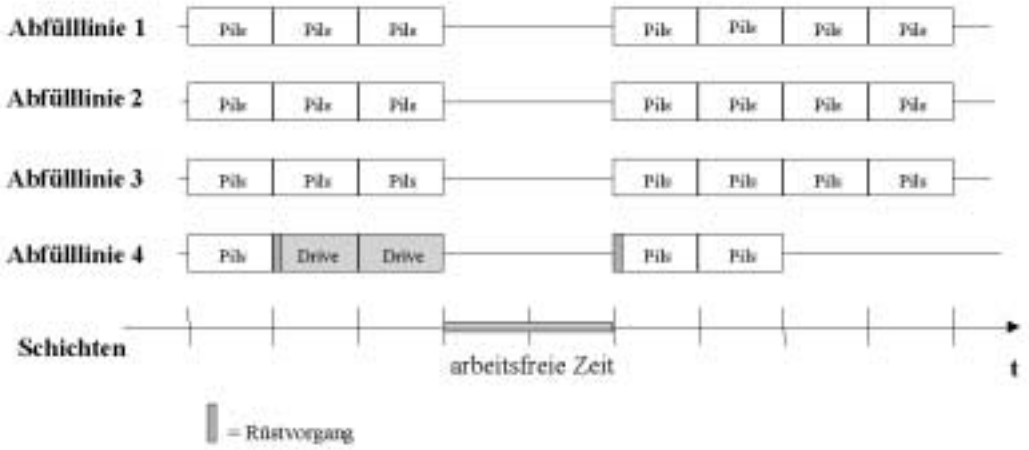


Abbildung 9.5: Einplanen der Langläuferprodukte

Der Algorithmusbaustein *ALGO_Langläufer* versucht, dieser Situation gerecht zu werden. Er durchmustert die Schichten der erlaubten Maschinen von hinten nach vorne und belegt sie immer entweder vollständig oder gar nicht. Die Entscheidung wird anhand der Bestandssituation gefällt. Eine Belegung erfolgt, wenn nach der Belegung abzüglich des Übertrags aus den vorigen Intervallen der Sollbestand nicht mehr, als um einen vorgegebenen Parameter Δ überschritten wird. In der Regel werden die Produktionseinheiten mit der höchsten Priorität über den Planungszeitraum vollständig belegt sein, während die anderen Maschinen auch Lücken im Plan haben können, in die dann Nebenprodukte eingeplant werden können. Diese Situation ist in Abbildung 9.5 dargestellt. Langläuferprodukte können über eine Produktgruppe identifiziert werden.
Die folgenden Algorithmusparameter werden verwendet:

Tabelle 9.5: Parameter für Algorithmus *ALGO_Langläufer*

Name	Typ	Bedeutung
Übertrag_flag	Boolean	Übertrag aus der Grobplanung verwenden
Gruppenname	String	Name der Produktgruppe des Langläufers
PE-GruppeLangl	String	Name der Produktionseinheitengruppe für Langläufer
Delta_bestand	Double	Maximal zulässige Bestandsüberschreitung

Für den Algorithmus gelten die folgenden Voraussetzungen:

- Für jeden Langläufer ist eine eigene Produktgruppe definiert.
- Die Liste aller möglichen Produktgruppen und Produktionseinheitengruppen ist mit Indizes versehen, so dass jedem Namen ein eindeutiger Index zugeordnet ist.

9.10 Fertigungsstufe Bierabfüllung: Standardprodukte

Standardprodukte sind die nicht durch die Langläufer abgedeckten Produkte. Für diese müssen Produktionslose gebildet werden, wobei die Losgrößen sich anhand der aktuellen Absatzsituation orientieren. Für jedes Produkt wird der Restproduktionsbedarf über das Tagesraster ermittelt. Solange dieser positiv ist, wird eine Teilmenge dieses Bedarfs eingeplant und der Bedarf gemäß dieser Menge reduziert. Zum Einplanen der Teilmenge, also der aktuellen Charge, wird die Bestandsentwicklung für das Produkt anhand der bereits vorhandenen Belegungen berechnet und ein Zeitpunkt t_0 identifiziert, zu dem der Sollendbestand unterschritten wird. Die optimale Losgröße der Charge wird daraufhin als das Maximum der minimal möglichen Losgröße für das Produkt und den kumulierten Absätzen des Intervalls von t_0 und einer parametrierbaren Zahl n von Folgeintervallen berechnet. Im nächsten Schritt wird die beste freie Belegung auf allen zugeordneten Produktionseinheiten möglichst dicht vor t_0 gesucht und die Belegung realisiert. Die in Tabelle 9.10 dargestellten Algorithmusparameter werden verwendet:

Tabelle 9.6: Parameter für Algorithmus *ALGO_MinMengeAbsatzTage*

Name	Typ	Bedeutung
Übertrag_flag	Boolean	Übertrag aus der Grobplanung verwenden
Gruppe	String	Einzuplanende Produktgruppe

Algorithmus 9.10.2 Langläufer

Input: Partiell belegter Plan

Output: Plan erweitert um Belegungen für Langläuferprodukte

1.0 Schleife über Produkte p

1.1 Erzeuge neuen Fertigungsauftrag *fa* für p

1.2 Identifiziere Produktionseinheit i, die *PE-GruppeLangl* repräsentiert.

1.3 Setze *ha*, *ag*, und *vf* als zugehörige Herstellanweisung, der Arbeitsgang und die Verfeinerung.

1.4 Falls Produktionseinheitengruppe gefunden

/* Einplanen auf den Maschinen der in Verfeinerung gegebenen Produktionseinheitengruppe. */

1.4.1 Schleife über die Produktionseinheiten i von *PE-GruppeLangl*

1.4.1.1 Schleife über Schichten *schicht_id* im Buchungsplan von i.

/* Produkt von voriger Schicht ermitteln zwecks Rüstzeitberechnung */

Schicht $schicht\_id_0$ identifizieren, die direkter Vorgänger von *schicht_id* ist.

Finde Buchung $buchung\_id_0$, die zwischen Beginn und Ende von $schicht\_id_0$ liegt und die Buchung einer Produktion ist.

Nehme Produkt p_0 von $buchung\_id_0$.

Berechne Rüstzeit bei Produktwechsel von p_0 zu p.

Berechne Kapazität der Schicht in Stunden:

Fall 1: Beginn des Planungshorizonts liegt nach dem Start von *schicht_id*. Kapazität m ist Zeitdifferenz von Beginn des Planungshorizonts bis zu Ende *schicht_id* in h.

Fall 2: Andernfalls ist die Kapazität m Zeitdifferenz von Beginn *schicht_id* bis Ende *schicht_id* in h abzüglich Rüstzeit.

/* Bestimme in *schicht_id* produzierbare Menge *menge* */

menge :=*m * Chargenmenge(ha)/Durchlaufzeit(vf)*

Bestimme das Rasterintervall *Intervall*, in das Schicht *schicht_id* fällt.

Bestimme für Produkt p und Intervall *Intervall* den aus einer Belegung resultierenden Bestand b.

Falls $b <$ *Sollendbestand + Delta_bestand* und *schicht_id* ist nicht belegt, dann

/* einplanen von p über gesamte Schicht */

Neuen *fa* erzeugen und *ha* und *menge* zuordnen.

Neuen *ma* erzeugen mit Werten

art	:=	*PRODUKTION*
prodzeit	:=	*m*
start	:=	Beginn von *schicht_id*
prodstart	:=	*ma.start + rüstdauer*
ende	:=	Ende von *schicht_id*
maschine	:=	*i*
produkt	:=	*p*
ha	:=	*ha*
ag	:=	*ag*
ma.vf	:=	*vf*
ma.fa	:=	*fa*

Erzeuge *buchung_auftrag* mit den Daten von *ma* und erzeuge die Belegung im Buchungsplan von i.

Algorithmus 9.10.1 MinMengeAbsatzTage

Input: Partiell belegter Abfüllpan

Output: Abfüllplan mit zusätzlichen Belegungen

1. Schleife über die Produkte p nach Priorität sortiert
2. Falls p Mitglied von *Gruppe*, dann
   - 2.1 Falls *übertrag_flag* $=$ *TRUE*, *prodbedarf* letztes Intervall des Tagesrasters um Übertrag erstes Intervall der kapazitativen Planung erhöhen.
   - 2.2 Produktionsbedarf *prodbedraf* := Summe Produktionsbedarfe über Tagesraster. Produktionsbedarfe werden aus Sollendbestand, Bestand, Produktion und Absätzen berechnet.
   - 2.5 Solange *prodbedarf* > 0
     - 2.5.1 Bestandsentwicklung für p über bereits vorhandene Belegungen berechnen.
     - 2.5.2 Suche Zeitpunkt t_0, an dem Bestand den Sollendbestand unterschreitet.
     - 2.5.3 Losgröße *menge* :=
       max(*Minimale Losgröße, Summe Absätze von* t_0 *bis* t_0+ *p.reichweite)*
       /* *p.reichweite* definiert Anzahl der Absatztage, denen Bestand von p genügen soll */
     - 2.5.4 Suche beste freie Belegung auf allen zugeordneten Produktionseinheiten möglichst vor t_0 und setzte Flag *gefunden*.
     - 2.5.5 Falls *gefunden* $=$ *TRUE*, dann
       Erzeuge *ma* mit Losgröße *menge*.
       prodbedarf := *prodbedarf* $-$ *menge*
     - sonst
       Warnung "Nicht produzierbare Restmenge"
       prodbedarf $= 0$
   - 3. Aufruf Funktion *Bestandskorrektur*

Die Planresultate durch die Funktion zum Einplanen durch die Funktion *MinMengeAbsatzTage* sollen verbessert werden. Ziel ist, die resultierenden Bestandskurven zu glätten und Überbestände durch zeitliches Verschieben von Aufträgen auszugleichen. Es wäre möglich und sinnvoll, die Funktion mehrfach und für verschiedene aufzurufen Produktgruppen aufzurufen.

Funktion 9.10.1 Bestandkorrektur

Input:
- Gruppe der umzuplanenden Produkte
- Faktor, der Überbestand relativ zum Sollbestand angibt, ab dem Algorithmus aktiv wird.

Output: Modifizierter Belegungsplan

1. Schleife über Produkte p der Produktgruppe nach Summe der Überbestände sortiert.
2. *fertig* := *FALSE*
   - 2.1 Schleife solange *fertig* $=$ *FALSE*
   - 2.1.1 t_0 := Zeitpunkt der nächsten Überschreitung des Sollendbestandes um angegebenen Faktor.
   - 2.1.2 Falls t_0 innerhalb des Tagesrasters liegt, dann
     - 2.1.2.1 Schicht *schicht_id* identifizieren, in der t_0 liegt.
     - 2.1.2.2 Belegung in Schicht zu Produkt p entfernen und zu nächsten Zeitpunkt einplanen, zu dem Sollendbestand wieder unterschritten wird.

9.11 Fertigungsstufe Bierabfüllung: Drucktankabbuchung

Drucktanks bilden Puffer zwischen der Filtration und der Bierabfüllung. Die Filtration führt die Filtratmengen in die Drucktanks, die Abfüllung entnimmt das Bier aus den Drucktanks. Der folgende Algorithmus steuert die Entnahme des Biers aus den Drucktanks.
Die Drucktanks werden linear durchmustert und jeweils das ganze Volumen von den zulässigen Drucktanks abgebucht bis die Restmenge kleiner als das freie Volumen ist. Dann wird nur die Restmenge abgebucht und mit dem nächsten Maschinenauftrag weitergearbeitet. Wenn der letzte Drucktank erreicht ist, wird der nächste Drucktank betrachtet. Im Falle, dass nach zwei Durchläufen kein Tank gefunden wird, kann keine Abbuchung erfolgen und eine Warnung wird ausgegeben.
Eine Abbuchung einer Menge ist in einem Drucktank möglich, wenn die erste Buchung vor dem Zeitpunkt der gewünschten Abbuchung den Tank vollständig entleert und eine Reinigungszeit und die notwendige Zeit zum neuen Befüllen des Tanks verstrichen ist, oder im Tank aktuell das abzubuchende Produkt liegt.
Wenn eine Abbuchung möglich ist, sind weiterhin zwei Fälle zu unterscheiden. Der Tank kann partiell gefüllt werden. Dann kann er bis zum vollen Volumen abgebucht werden, höchstens aber um seine Restmenge. Er kann aber auch komplett geleert sein. Dann ist die Abbuchungsmenge das Minimum aus dem Tankvolumen und der Restmenge.
Die folgenden Algorithmusparameter werden verwendet:

Tabelle 9.7: Parameter für Algorithmus *ALGO_Drucktankabbuchung*

Name	Typ	Bedeutung
PE–GruppeDrucktank	String	Name Gruppe Drucktanks
ProduktFiltrat	String	Name Gruppe Filtrat
Reinigungszeit	Double	Pauschale Reinigungszeit für die Drucktanks

Algorithmus 9.11.1 DrucktankAbbuchung

Input: Abfüllpan mit Maschinenaufträgen

Output: Plan, bei dem die Abfüllmengen von Drucktanks abgebucht sind.

1. Produktionseinheiten der Gruppe $PE - GruppeDrucktank$ linear sortieren.
2. Schleife über ma der Abfüllung.

/* Zugehörige Mengen werden in den Drucktanks eingeplant. */

2.1 Schleife über die Positionen der Stückliste von $ma.produkt$ um Filtrat p zu finden, dass aus den Drucktanks entnommen werden soll.
/* Buche p vom Drucktank ab */
2.2 $gefunden = FALSE$
2.3 $weiter = TRUE$
2.4 $i = 1$
2.5 Nehme $ma.menge$ als einzuplanende $restmenge$
2.6 Falls $restmenge <= 0$, dann
$weiter = FALSE$
2.7 Schleife über die Herstellanweisung, und Verfeinerungen des ersten Arbeitsgangs der Herstellanweisung um Verfeinerung act_vf zur i–ten Maschine der Produktionseinheitengruppe $PE\text{-}GruppeDrucktank$ zu finden.
2.8 Aufruf Funktion $DrucktankAbbuchung$ um ma auf i–ter Maschine abzubuchen. Die Funktion setz einen Flag $erfolgreich$ der kennzeichnet, ob Abbuchen möglich.
Falls $erfolgreich = TRUE$, dann
$erster\_durchlauf = TRUE$,
sonst
Falls i letzter Durcktank, dann
Falls $erster\_durchlauf$, dann $zweiter\_durchlauf$
sonst,
$weiter = FALSE$
Falls $i <$ Zahl der Drucktanks, dann
$i++$,
sonst
$i = 0$
Falls $restmenge$ kleiner als minimal zulässige Drucktankcharge, dann
$weiter = FALSE$

Funktion 9.11.1 DrucktankAbbuchung

Input:
- *plan* (Belegungsplan)
- *ma* (Maschinenauftrag, der Abbuchung im Drucktank initiiert)
- *p* (Abzubuchendes Produkt)
- *vf* (Verfeinerung für Maschine)
- *reinigungszeit* (Dauer der Reinigung Drucktanks)
- *i* (betrachteter Drucktank)

Output: *k* (Index des aktuell betrachteten Drucktanks)

1. $t = ma.ende$
2. Sei id_0 die erste Buchung vor der nach t endenden Buchung.
3. Falls Zeitpunkt von id_0 = Beginn des Planungszeitraums
   oder
   $id_0.produkt = p$
   oder
   durch die Buchung wird Tank entleert
   und
   die Entleerung liegt mindestens um Reinigungszeit zuzüglich Befüllzeit des Tanks vor *ma.start*, dann
   3.1 /* Abbuchung möglich */
   Fall 1: Tank ist bei id_0 nur partiell abgebucht: Buche bis zum Tankvolumen ab, jedoch nicht mehr als Restmenge
   Fall 2: Tank komplett geleert: Buche komplettes Tankvolumen ab, jedoch nicht mehr als Restmenge
   3.2 Falls Tank bei id_0 entleert, dann
   summe_buchungen = 0.0
   menge := min(*i.volumen, restmenge)*
   sonst
   Bestimme Rest bis zur vollen Abbuchung.
   menge := min(*i.volumen - summe_buchungen, restmenge)*
   /* *summe_buchungen* ist dabei Summe der Volumina aller Buchungen seit letzter Entleerung */
   3.3 Falls *menge = i.volumen - summe_buchungen*, dann
   entleerung = TRUE
   3.4 Erzeuge in Buchungsplan von Drucktank *i* mit *AddBuchung* eine neue Buchung mit Werten:

produkt	:=	*produkt*
zeitpunkt	:=	t
volumen	:=	- menge
kennung	:=	RESSOURCE
auftrag	:=	NULL
erste_buchung	:=	*FALSE*
exklusiv	:=	FALSE
entleerung	:=	*entleerung*

   3.5 *restmenge -= menge*
   3.6 *erfolgreich = TRUE*

KAPITEL 10

Schlussbetrachtungen

In dieser Arbeit haben wir algorithmische Fragestellungen der Produktionsplanung betrachtet. Im ersten Teil dieser Arbeit haben wir zwei Scheduling–Probleme untersucht, bei denen Maschinen nur zu begrenzten Zeiten zur Verfügung stehen und die Jobs nur in den verbleibenden Zeiträumen eingeplant werden können. Für diese Probleme erzielten wir bestmögliche Approximationsaussagen und darüber hinaus konnte der Approximierbarkeitsstatus dieser Probleme vollständig geklärt werden. Scheduling–Modelle mit limitierter Maschinenverfügbarkeit sind ein Beispiel, wie zunächst rein theoretische Probleme durch den Einbezug verschiedener Restriktionen näher an die Problemstellungen der industriellen Praxis geführt werden können. Neben den Betrachtungen weiterer praxisrelevanter Nebenbedingungen wäre es eine wichtige Aufgabe zukünftiger Arbeiten, auch verstärkt die Aspekte einer hierarchischen Planung, wie sie im zweiten Teil dieser Arbeit beschrieben wurde, zu berücksichtigen. Diese Planungshierarchien verlaufen sowohl in vertikaler Richtung von den höheren Managementebenen zur Feldebene, als auch vertikal über mehrere Produktionsstufen hinweg. Beispielprobleme, deren Untersuchung lohnenswert sein könnte, wäre eine Integration von Losgrößenplanung und dem Scheduling mit dem Ziel, ausgewogene Bestandskurven zu erzielen und dazu simulatan den klassischen Zielfunktionen aus der Theorie des Scheduling zu genügen, etwa dem Minimieren der Durchlaufzeiten der Jobs im System.
Im theoretische Teil der Arbeit haben wir zudem versucht, durch den Übergang zum stochastischen Modell eine praxisnahe Analyse der Performance der Algorithmen zu ermöglichen. Dabei haben wir einen ersten Schritt in Richtung einer Average–Case–Analyse von Scheduling–Problemen unternommen. Average–Case-Analysen von Scheduling–Problemen gestalten sich äußerst schwierig und so gibt es nur wenige Resultate zu deutlich eingeschränkten Problemklassen. In dieser Arbeit gelang es uns, ein weiteres Problem, das des Minimierens der Summe der Fertigstellungszeiten der Jobs, zu analysieren. Anhand dieses Problems konnten wir neue Aspekte der Performance bekannter Scheduling–Algorithmen

nachweisen, wobei wir die bis dato angenommene Optimalität von Scheduling–Algorithmen durch den von uns betrachteten Gütebegriff eines Algorithmus in Frage stellten. Unser Gütebegriff eines Algorithmus im stochastischen Modell ist eng an die *kompetitive Analyse* von Online–Algorithmen angelehnt. Genauer betrachten wir die erwartete relativen Performance eines stochastischen Scheduling–Algorithmus gegenüber einer optimalen Offline–Lösung. Dieses Performancemaß bietet breiten Raum für weiterführenden Fragestellungen. Als erstes sollte die Optimalität der Scheduling–Strategien für verschiedene klassische Scheduling–Probleme unter der Annahmen unterschiedlicher Verteilungsfunktionen in Hinblick auf den von uns betrachteten Gütebegriff neu bestimmt werden. Zweitens kann versucht werden, die erwartete relative Performance von einfachen *List*–Algorithmen auch für weiterführende Probleme zu bestimmen, oder zumindest abzuschätzen. Dem schließen sich Fragestellungen der Konzentration der Verteilung der relativen Performance um den Erwartungswert an. Ziel ist festzustellen, ob der Erwartungswert ein hinreichend genaues Bild von der tatsächlichen Güte eines Scheduling–Algorithmus geben kann.
Im zweiten Teil dieser Arbeit wurde ein System zur Produktionsplanung und zum Produktionsscheduling für eine Praxisanwendung konzipiert und umgesetzt. Hier lag der Fokus der Entwicklung auf einer abteilungsübergreifenden integrierten Systemarchitektur. Wir haben verschiedene Scheduling–Algorithmen in diese Gesamtsystem eingebettet und dabei hierarchische, mehrstufige Planungsalgorithmen erhalten. Derzeit beschränken sich die verwendeten Algorithmen im Wesentlichen auf Varianten einfacher List–Algorithmen. Für die weitere Systementwicklung sollte versucht werden, elaboriertere Algorithmen in das Planungssystem zu integrieren um eine weitgehend optimale Planung der verschiedenen Produktionsbereiche zu gewährleisten. Die Systemimplementation hat allerdings gezeigt, dass sich die Planungsprobleme derart vielschichtig gestalten und dass einer Vielzahl von produktionstechnischen aber auch von datentechnischen Restriktionen genügt werden muss, so dass es häufig schon schwer fällt, überhaupt zulässige Lösungen zu erzeugen, geschweige denn optimale. Es hat sich gezeigt, dass eine zentrale Herausforderung bei der Entwicklung nicht nur im Entwurf effizienter Algorithmen zum Produktionsscheduling, sondern auch im Systemdesign liegt, mit einer geschickten Kombination der Algorithmen und der Schnittstellen einzelner Planungsprobleme untereinander und mit einer Integration der Algorithmen in die Informationsflüsse des Unternehmens.

Literaturverzeichnis

[AP94] W. Ahrens and M. Polke. Informationsstrukturen in der Leittechnik. In M. Polke, editor, *Prozessleittechnik*, pp. 21–89. Oldenbourg, München, 1994. 2. Auflage.

[AS99] S. Albers and G. Schmidt. Scheduling with unexpected machine breakdowns. *Lecture Notes in Computer Science*, 1671, 1999.

[AUK88] H. Aue-Uhlhausen and H. Kühnle. *Von ABS bis OPT — PPS–Methoden im Vergleich*, pp. 177–230. In: PPS 88, AWF, Eschborn, 1988.

[BESW94] J. Błazewicz, K. H. Ecker, G. Schmidt, and J. Węglarz. *Scheduling in Computer and Manufacturing Systems*. Springer-Verlag, 1994. Second Edition.

[BS96] F. Bechmann and M. Scharbrodt. Direkte Anbindung an den Prozess – Produktionsplanung in der Getränkeabfüllung. *Getränkeindustrie*, 50(3):154–156, 1996.

[BS97] F. Bechmann and M. Scharbrodt. Ein hierarchisches Modell zur Produktionsplanung in der Brau- und Getränkeindustrie. In *Schriftenreihe des Fachbereichs Mathematik – Arbeitskreis Mathematik in Forschung und Praxis. Operations Research, 15.*, Bad Honnef, 1997.

[CG85] E. G. Coffman, Jr. and E. N. Gilbert. Expected relative performance of list scheduling rules. *Oper. Res.*, 33:548–561, 1985.

[CGJ78] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, February 1978.

[CGJ96] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation algorithms*, pp. 46 – 86. PWS Publishing Company, 1996.

[dlVL81] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within 1 + o in linear time. *Combinatorica*, 1:349–355, 1981.

[Fel57] W. Feller. *An Introduction to Probability Theory and Its Applications, Volume I.* John Wiley & Sons, New York, 1957. Third Edition.

[FFG94] G. Fandel, P. Francois, and K. M. Gubitz. *PPS–System, Grundlagen, Methoden, Software, Marktanalyse.* Springer-Verlag, Berlin, 1994.

[Fri84] Donald K. Friesen. Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal on Computing*, 13(1):170–181, 1984.

[GGU72] M. R. Garey, R. L. Graham, and J. D. Ullman. Worst-case analysis of memory allocation algorithms. In *Conference Record, Fourth Annual ACM Symposium on Theory of Computing*, pp. 143–150, Denver, Colorado, 1–3 May 1972.

[GJ79] Michael R Garey and David S Johnson. *Computers and Intractibility, a Guide to the Theory of NP-Completeness*, chapter A2. W.H. Freeman and Co., San Francisco, 1979.

[Gla79] K. D. Glazebrook. Scheduling tasks with exponential service times on parallel machines. *Journal of Applied Probability*, 16:685–689, 1979.

[GLLK79] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Mathematics*, 5:287–326, 1979.

[Got99] O. Gote. Produktionsgrobplanung für getränkeabfüllende Betriebe unter Einsatz der linearen Programmierung. Diplomarbeit, TU–München, Lehrstuhl für Brauereianlagen und Lebensmittel–Verpackungstechnik, 1999.

[Gra66] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[Gra69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.

[Gut76] E. Gutenberg. *Grundlagen der Betriebswirtschaftslehre, Band I: Die Produktion.* Springer–Verlag, 1976.

[Hac89] R. Hackstein. *Produktionsplanung- und Steuerung (PPS).* VDI–Verlag, Düsseldorf, 1989.

[hey94] In K. U. Heyse, editor, *Handbuch der Brauerei–Praxis*. Carl, Nürnberg, 1994. 3. Auflage.

[Hoc97] D. S. Hochbaum. Approximation algorithms for NP-hard problems. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 28, 1997.

[Hr83] VDI (Hrsg.). *Lexikon der Produktionsplanung- und Steuerung, VDI–Taschenbuch T77*. VDI–Verlag, Düsseldorf, 1983.

[HS86] Hochbaum and Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 6, 1986.

[HS87] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of the ACM*, 34:144–162, 1987.

[JKB94] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous Unvariate Distributions*. John Wiley & Sons, 1994. Second Edition.

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[Käm87] T. Kämpke. On the optimality of static priority policies in stochastic scheduling on parallel machines. *Journal of Applied Probability*, 24:430–448, 1987.

[Keh96] H. Kehl. *Organisation und Technik integrierter Informations- und Kommunikationssysteme in Brauereien*. Doktorarbeit, TU–München, Lehrstuhl für Brauereianlagen- und Lebensmittel–Verpackungstechnik, 1996.

[Ker94] H. Kernler. *PPS der 3. Generation*. Hüthig Buch Verlag, Heidelberg, 1994.

[KP94] B. Kalyanasundaram and K. R. Pruhs. Fault-tolerant scheduling. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pp. 115–124, Montréal, Québec, Canada, 23–25 May 1994.

[KP97] B. Kalyanasundaram and K. R. Pruhs. Fault-tolerant real-time scheduling. In *ESA: Annual European Symposium on Algorithms*, 1997.

[Kun94] W. Kunze. *Technologie der Brauerei und Mälzerei*. VLB Berlin, Berlin, 1994. 7. Auflage.

[Kur93] K. Kurbel. *Produktionsplanung- und Steuerung.* Oldenbourg–Verlag, München/Wien, 1993.

[Len81] H. W. Lenstra. Integer programming with a fixed number of variables. Technical Report 81-03, University of Amsterdam, Amsterdam, 1981.

[LLK91] E. L. Lawler, J. K. Lenstra, D. B. Shmoys and A. H. G. Rinnooy Kan. Sequencing and scheduling: Algorithms and complexity. Technical Report BS-R8909, Centre for Mathematics and Computer Science., Amsterdam, 1991.

[LS97] Zhen Liu and E. Sanlaville. Stochastic scheduling with variable profile and precedence constraints. Technical Report RR-1525, Inria, Institut National de Recherche en Informatique et en Automatique, 1997.

[MR85] R. H. Moehring and F. J. Radermacher. Introduction to stochastic scheduling problems. In *Contributions to operations research, Proc. Conf., Oberwolfach/Ger. 1984, Lect. Notes Econ. Math. Syst. 240, 72-130*, 1985.

[MRW84] R. H. Moehring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems. I: General strategies. *Z. Oper. Res., Ser. A 28, 193-260*, 1984.

[MRW85] R. H. Moehring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems. II: Set strategies. *Z. Oper. Res., Ser. A 29, 65-104*, 1985.

[MSU99] R. Moehring, A. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *JACM: Journal of the ACM*, 46, 1999.

[Nar76] L. Narziß. *Die Bierbrauerei – Die Technologie der Malzbereitung.* Enke, Stuttgart, 1976. 6. Auflage.

[Nar85] L. Narziß. *Die Bierbrauerei – Die Technologie der Würzebereitung.* Enke, Stuttgart, 1985. 6. Auflage.

[Nar86] L. Narziß. *Abriß der Bierbrauerei.* Enke, Stuttgart, 1986. 5. Auflage.

[Pin95] M. Pinedo. *Scheduling: theory, algorithms, and systems.* Prentice-Hall, Englewood Cliffs, 1995.

[PT87] Christos H. Papadimitriou and John N. Tsitsiklis. On stochastic scheduling with in-tree precedence constraints. *SIAM Journal on Computing*, 16(1):1–6, February 1987.

[Räd99] T. Rädler. *Modellierung und Simulation von Abfülllinien*. Doktorarbeit, TU–München, 1999.

[RBKW98] T. Rädler, F. Bechmann, H. Kehl, and H. Weisser. Computer aided techniques in breweries. In *Report for the European Brewery Convention Technology and Engineering Forum*, EBC, Zouterwoude, 1998.

[Rot66] M. H. Rothkopf. Scheduling with random service times. *Management Science*, 12:703–713, 1966.

[Sah76] S. Sahni. Algorithms for scheduling independent tasks. *J. Assoc. Comput. Mach.*, 23:116–127, 1976.

[SAP94] SAP. *Projekt–System, System R/3*. SAP (Hrsg.), Walldorf, 1994.

[SAP96] SAP. *System R/3, Produktionsplanung*. SAP (Hrsg.), Walldorf, 1996.

[Sch84] G. Schmidt. Scheduling on semi–identical processors. *Z. Oper. Res*, 28:153–162, 1984.

[Sch88] G. Schmidt. Scheduling independent tasks with deadlines on semi–identical processors. *J. Oper. Res. Soc*, 39:271–277, 1988.

[Sch94] A. W. Scheer. *Wirtschaftsinformatik*. Springer Verlag, 1994.

[Sch97] M. Scharbrodt. DV–gestützte Abfüllplanung. In *Unterlagen zum 4. Flaschenkellerseminar*, Freising: TU München, Lehrstuhl für Brauereianlagen und Lebensmittel–Verpackungstechnik, 1997.

[Sch98] M. Scharbrodt. Integration der Produktionsplanung in eine Unternehmens–DV für die Getränkeanfüllung. In *Unterlagen zum 5. Flaschenkellerseminar*, Freising: TU München, Lehrstuhl für Brauereianlagen und Lebensmittel–Verpackungstechnik, 1998.

[Smi56] W. Smith. Various optimizers for single–stage production. *Naval Res. Logist. Quart.*, 3:59–66, 1956.

[SWBE99] M. Scharbrodt, H. Weisser, F. Bechmann, and R. Ewert. Den Wildwuchs eindämmen – Einsatz von PPS–Systemen in der Getränkebranche. *Getränkeindustrie*, 53(2):78–82, 1999.

[Wei90] G. Weiss. Approximation results in parallel machine stochastic scheduling. *Annals of Operations Research*, 26, 1990.

[Wer99] Werum. *Pas–Plan Produktionsplanungssystem, Benutzerhandbuch.* Werum, Lüneburg, 1999.

[Wöh90] G. Wöhe. *Einführung in die allgemeine Betriebswirtschaftslehre.* Verlag Vahlen, München, 1990.

[WP80] G. Weiss and M. Pinedo. Scheduling tasks with exponential service times on non–identical processors to minimize various cost functions. *Journal of Applied Probability*, 17:187–202, 1980.

[WR98] H. Weisser and T. Rädler. Dataprocessing and monitoring. In *Proceedings 1998 Asean Brewing Conference: Modern Brewery*, 1998.

[WVW86] R. R. Weber, P. Varaiya, and J. Walrand. Scheduling jobs with stochastically ordered processing times on parallel machines to minimize expected flowtime. *Journal of Applied Probability*, 23:841–847, 1986.

[Zäp93] G. Zäpfel. Produktionsplanungs- und steuerungssysteme. In W. Wittman et. al., editor, *Handwörterbuch der Betriebswirtschaft*, pp. 3467–3478. Schäffer-Poeschel Verlag, Stuttgart, 1993.