Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Bachelorarbeit
im Studiengang *Bachelor of Computer Science*

Modellierung des Schedulingproblems der Behälterförderanlage bei Rossmann

vorgelegt von:	Tim Daniel Metzler
Fachbereich:	Informatik
Erstgutachter:	Prof. Dr. Kurt-Ulrich Witt
Zweitgutachter:	Dipl.-Ing. (FH) Jens Heinrich

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

In der Wirtschaft spielt die optimierte Verteilung von Aufträgen auf Anlagen in der Produktion und der Logistik eine zentrale Rolle. Die Reihenfolgeplanung als Untergebiet der Produktionsprozessplanung bietet bereits einige Algorithmen für einfache Anlagenmodelle. Jedoch müssen in der Realität oft noch eine Vielzahl von Nebenbedingungen beachtet werden, die nicht ohne Weiteres mit den bekannten Ansätzen kombiniert werden können. Somit ist es weiterhin schwierig, eine optimierte Auftragsreihenfolge für eine konkrete Anlage zu finden. Im Lager der Firma Rossmann in Landsberg gibt es eine Anlage, in die Aufträge der Verkaufsstellen eingebracht werden. Diese Aufträge werden auf Kisten verteilt und an einer Quelle in das System eingespeist. Von der Hauptstrecke gibt es Abzweigungen zu Schleifen, in denen sich mehrere Bahnhöfe befinden. An diesen Bahnhöfen werden die Kisten automatisch auf ein Abstellgleis gefahren, von den Mitarbeitern befüllt und von diesen wieder manuell auf die jeweilige Nebenstrecke (in die Schleife) geschoben. Nach dem Passieren einer Schleife fahren die Kisten wieder auf die Hauptstrecke und steuern entweder weitere Bahnhöfe oder den Ausgang der Anlage an. Im Lager kommt es häufig zu Staus. Diese entstehen bei einem sehr hohen Kistenstrom. Dabei kommt es einerseits dazu, dass die Kapazität des Bahnhofs ausgeschöpft ist und keine weiteren Kisten diesen Bahnhof mehr anfahren können. Andererseits kommt es dazu, dass der Kistenstrom am jeweiligen Bahnhof so hoch ist, dass der Mitarbeiter keinen Platz auf der Nebenstrecke findet, um eine befüllte Kiste wieder in den Strom einzubringen. Kisten, die nun nicht in einen vollen Bahnhof einfahren können, überspringen diesen. Dabei werden zuerst alle unbesetzten nachfolgenden Bahnhöfe angefahren. Anschließend wird die Kiste wieder an der Quelle in das System eingebracht, um die übersprungenen Bahnhöfe anzufahren. Kommt eine Kiste ein drittes Mal an einem Bahnhof vorbei und kann diesen immer noch nicht anfahren, stoppt die Kiste und verweilt so lange vor dem Bahnhof, bis dieser wieder genug Kapazitäten hat, um die jeweilige Kiste aufzunehmen. Die Feststellung, ob ein Bahnhof voll ist, wird erst getroffen, wenn sich die Kiste direkt vor dem Bahnhof befindet. Somit werden Schleifen mit vollen Bahnhöfen nicht einfach übersprungen, sondern die Kiste passiert jede Schleife, in der sich ein Bahnhof für diese Kiste befindet. Die Staus führen zu einer längeren Verweilzeit von Kisten im System, zum mehrfachen Durchlaufen des Systems und zu einer Blockierung der Strecken, so dass keine bearbeiteten Kisten auf die Strecke zurückgeschoben werden können. Der einzige Parameter, über den sich das System beeinflussen lässt, ist die Reihenfolge, in der die Kisten in die Anlage eingebracht werden. Daher stellt sich die Frage, in welcher Reihenfolge eine gegebene Menge von Aufträgen in das System eingespeist werden soll.

1.2 Zielsetzung

In dieser Bachelorarbeit soll untersucht werden, wie sich die Behälterförderanlage der Firma Rossmann in Landsberg als Maschinenbelegungsmodell darstellen lässt. Dabei wird zwischen Online und Offline Scheduling unterschieden. Beim Offline Scheduling sind alle zu bearbeitenden Aufträge vor der Ausführung des Algorithmus bekannt. Beim Online Scheduling liegen diese Informationen erst zur Laufzeit vor. Weiterhin soll untersucht werden, inwieweit sich bekannte Schedulingverfahren aus der Literatur für die Lösung des Rossmann-Problems eignen. Daneben soll ein eigener Algorithmus entworfen werden, der sich sowohl für das Online Scheduling, als auch für das Offline Scheduling eignet. Schließlich sollen die Verfahren bezüglich ihrer Laufzeit und Qualität bewertet werden.

1.3 Vorgehensweise

Die Erstellung eines Schedulingverfahrens wird am Beispiel des Lagers der Firma Rossmann in Landsberg dokumentiert und analysiert. Hierzu werden verschiedene Schedulingverfahren untersucht. Dabei werden zuerst bereits vorhandene Ansätze aus der Literatur untersucht. Weiterhin wird das Rossmann-Problem in die bekannten Maschinenbelegungsmodelle eingeordnet. Danach werden auf dieser Basis neue Schedulingverfahren entwickelt und experimentell untersucht. Daneben soll versucht werden, für eine gegebene Auftragsmenge eine obere und untere Schranke für die Ausführungszeit zu berechnen.

2 Stand der Forschung

Scheduling Problemen wird in der Literatur seit jeher eine große Aufmerksamkeit zuteil. Bereits seit den frühen 60er Jahren wird an der rechnergestützten Lösung von verschiedensten Scheduling Problemen geforscht ([1]). Für die Modellierung von Scheduling Problemen, die bei Produktionsprozessen auftreten, hat sich die Aufteilung der Probleme in verschiedene Klassen durchgesetzt ([2]). Diese Klassen werden als Maschinenbelegungsmodelle bezeichnet. Hier wird zunächst ein Überblick über die wichtigsten Maschinenbelegungsmodelle gegeben. Anschließend werden einige ausgewählte Algorithmen zur Lösung von Shop Scheduling Problemen vorgestellt.

2.1 Klassifikation von Maschinenbelegungsmodellen

Die folgenden Ausführungen orientieren sich im Wesentlichen an der Definition von Graham, Lawler, Lenstra und Rinnooy Kan ([2]). In der Literatur werden Maschinenbelegungsmodelle üblicherweise als Tripel dargestellt.

$$(\alpha|\beta|\gamma) \tag{1}$$

Dabei beschreibt α die Maschinencharakeristik. Diese besteht aus der Maschinenzahl, -art und -anordnung. Die Auftragscharakteristika werden mit dem Parameter β beschrieben. Dazu gehören beispielsweise die Auftragszahl und Bedingungen, die für die Aufträge gelten. Darunter fallen Beschränkungen, wie die Bearbeitungszeit oder die Unterbrechbarkeit.
Der letzte Parameter γ beschreibt das zu miminierende Zielkriterium. Dies kann beispielsweise die Gesamtbearbeitungszeit C_{max} oder die Summe der Einzelbearbeitungszeiten $\sum C_i$ sein.

2.1.1 Open Shop

Das allgemeinste Maschinenbelegungsmodell ist der Open Shop ([3, S. 15]). Dieser beschreibt ein flexibles Fertigungssystem. Für jeden Job J_i muss eine Menge von Operationen $\{O_{i1}, O_{i2}, ..., O_{im}\}$ auf einer Menge von Maschinen $\{M_1, M_2, ..., M_m\}$ vollzogen werden. Jeder Operation O_{ij} eines Jobs J_i wird eine Maschine M_j und eine Bearbeitungszeit p_{ij} zugeordnet. Die Bearbeitungszeiten können auch den Wert Null annehmen. Dabei kann die Reihenfolge der Operationen auf den jeweiligen Maschinen frei gewählt werden. Somit gibt es zwei Teilprobleme. Zunächst muss für jeden Auftrag J_i eine optimale Operationsreihenfolge bestimmt werden. Anschließend muss eine optimale Reihenfolge π der Aufträge ermittelt werden.
Open Shops werden mit folgendem Tripel beschrieben.

$$(O_m|\beta|\gamma) \tag{2}$$

Das Feld β kann dabei beispielweise Informationen über Präzedenzrelationen, Fertigungstermine oder Transportzeiten zwischen den Maschinen enthalten.

2.1.2 Job Shop

Der Job Shop ist das klassische Maschinenbelegungsmodell, dem in der Literatur die größte Aufmerksamkeit zuteil wird ([3, S. 15]). Dieser ist ein Spezialfall des Open Shops. Analog zum Open Shop besteht jeder Auftrag J_i aus einer Menge von Operationen, die auf einer Menge von Maschinen bearbeitet werden müssen. Im Gegensatz zum Open Shop ist die Reihenfolge der Operationen für jeden Auftrag vorgegeben. Diese kann sich für verschiedene Jobs unterscheiden. Dabei kann es beispielsweise vorkommen, dass der Job J_1 zuerst auf der Maschine M_1 und anschließend auf der Maschine M_2 bearbeitet werden muss. Ein weiterer Job J_2 kann dann die Maschinenreihenfolge M_2, M_1 haben. Die Problemstellung besteht nun darin, eine optimale Reihenfolge π für die Aufträge zu ermitteln.
Üblicherweise wird ein Job Shop mit folgendem Tripel beschrieben.

$$(J_m|\beta|\gamma) \tag{3}$$

Mit J_m wird ein Job Shop mit m Maschinen bezeichnet. Dabei gibt es zwei grundsätzliche Unterscheidungen. Zum einen gibt es Job Shops, bei denen ein Auftrag eine Maschine höchstens einmal durchlaufen kann. Zum anderen gibt es Job Shops, bei denen jeder Auftrag eine Maschine auch mehrfach durchlaufen kann. Dies wird durch den Parameter $rcrc$ (Rezirkulation) im Feld β spezifiziert. Das klassische Job Shop Problem wird mit folgendem Tripel bezeichnet.

$$(J_m||C_{max}) \tag{4}$$

Dabei gibt es keine besonderen Beschränkungen bezüglich der Auftragscharakteristika. Daher ist das Feld β leer. Die Zielsetzung besteht darin, eine Reihenfolge π für eine gegebene Menge von Aufträgen zu finden, so dass die Gesamtbearbeitungszeit C_{max} minimal wird. Dabei kann die Bearbeitung eines Jobs zwischen den einzelnen Bearbeitungschritten unterbrochen werden.
Da die Parallelisierung von Fertigungsanlagen in der Realität immer wichtiger wird, wurde als Erweiterung des Job Shops der flexible Job Shop eingeführt.

$$(FJ_c|\beta|\gamma) \tag{5}$$

Der Parameter FJ_c bezeichnet einen flexiblen Job Shop mit c verschiedenen Arbeitsstationen. An jeder Arbeitsstation steht eine Menge von gleichen, parallel angeordneten Maschinen zur Verfügung. Der Shop ist flexibel, da ein Auftrag auf jeder der parallelen Maschinen bearbeitet werden kann.

2.1.3 Flow Shop

Der Flow Shop ist ein Spezialfall des Job Shops und beschreibt die klassische Fließbandfertigung. Analog zum Job Shop ist die Reihenfolge der Operationen für jeden Auftrag fest vorgegeben. Im Gegensatz zum Job Shop darf sich

diese für zwei Aufträge J_i und J_k mit $i \neq k$ nicht unterscheiden. Weiterhin wird meist davon ausgegangen, dass ein Auftrag zwischen zwei Maschinen beliebig lang auf den Bearbeitungsbeginn warten darf. Üblicherweise muss jeder Auftrag auf jeder Maschine bearbeitet werden. Wenn die Bearbeitungszeit p_{ij} eines Jobs J_i auf einer Maschine M_j auch den Wert Null annehmen darf, wird von einem generalisierten Flow Shop gesprochen. Ein Flow Shop wird mit folgendem Tripel beschrieben.

$$(F_m|\beta|\gamma) \tag{6}$$

Beim klassischen Flow Shop ist ein Überholen von Aufträgen nicht ausgeschlossen. Beim Permutations Flow Shop wird hingegen davon ausgegangen, dass sich ein Job, der die Maschine M_i verlässt, in die Warteschlange für die Maschine M_{i+1} einreiht. Alle Warteschlangen arbeiten dabei nach dem First In First Out Prinzip (FIFO). Die Kapazität der Warteschlangen wird üblicherweise als unendlich angenommen. Damit ist ein Überholen von Aufträgen in einem Permutations Flow Shop ausgeschlossen. Ein Permutations Flow Shop wird wie folgt spezifiziert.

$$(F_m|prmu|\gamma) \tag{7}$$

Der Parameter *prmu* im Feld β gibt dabei an, dass es sich um einen Permutations Flow Shop handelt. Flow Shops, bei denen ein Job nicht zwischen zwei aufeinanderfolgenden Maschinen warten darf, werden durch die no-wait Bedingung *nwt* im β Feld spezifiziert. Ein Beispiel hierfür ist die Herstellung von Stahlträgern in einem Stahlwerk. Dabei müssen die Stahlträger auf der Fertigungsstraße eine bestimmte Temperatur halten. Ohne die no-wait Bedingung könnte es dazu kommen, dass ein Stahlträger während des Wartens zu sehr abkühlt.
In einem Flow Shop ohne no-wait Bedingung wird davon ausgegangen, dass ein Job, für den die nachfolgende Maschine belegt ist, auf der vorigen Maschine bis zu seinem Bearbeitungsbeginn wartet. Ein weiterer Parameter, der im β Feld eines Flow Shops angegeben werden kann, ist der *block* Parameter. Dabei blockiert ein Job eine Maschine so lange, bis die nächste Maschine frei ist. Hierbei handelt es sich also um einen Flow Shop, bei dem keine Pufferplätze vor oder nach den Maschinen zur Verfügung stehen.
Schließlich gibt es, analog zum flexiblen Job Shop, den flexiblen Flow Shop. In diesem gibt es statt m Maschinen c verschiedene Arbeitsstationen mit gleichen parallel angeordneten Maschinen.

$$(FF_c|\beta|\gamma) \tag{8}$$

Ein flexibler Flow Shop wird mit dem Parameter FF_c bezeichnet. Dabei gibt c die Anzahl der Arbeitsstationen an.

2.1.4 Ausgewählte Parameter

In diesem Abschnitt werden einige gebräuchliche Parameter für das β Feld und das γ Feld vorgestellt. Diese sind in Tabelle 1 aufgelistet.

Feld	Parameter	Beschreibung
β	r_i	Freigabezeit des Jobs J_i
β	d_i	Deadline des Jobs J_i
β	t_{jk}	Transportzeit zwischen den Maschinen j und k
β	w_i	Priorität des Jobs J_i
β	ltd(b)	Eingangspuffer der Größe b
γ	C_{max}	Gesamtbearbeitungszeit
γ	$\sum w_i C_i$	Gewichtete Gesamtbearbeitungszeit
γ	$\sum w_i T_i$	Totale gewichtete Verspätung, $T_i = max(0, C_i - d_i)$
γ	$\sum T_i$	Totale Verspätung

Tabelle 1: Parameter für Maschinenbelegungsmodelle

Unter die Auftragscharakteristika, die im β Feld angegeben werden können, fallen beispielsweise die Freigabezeit r_i des Jobs J_i, die Frist d_i, bis zu der der Job J_i bearbeitet werden muss, sowie die Prioriät w_i des Jobs J_i. Zwei weitere Parameter, die für das Rossmann-Problem eine Rolle spielen, sind die Größe des Eingangspuffers b, sowie die Transportzeiten zwischen den Maschinen t_{jk}.
Im γ Feld wird die zu minimierende Zielfunktion angegeben. Dies ist bei klassichen Shop Problemen meist die Gesamtbearbeitungszeit C_{max}. Für priorisierte Aufträge kann auch die gewichtete Bearbeitungszeit $\sum w_i C_i$ gewählt werden. Kommen weiterhin noch Fristen für die Bearbeitung hinzu, kann die totale gewichtete Verspätung $\sum w_i T_i$, oder die totale ungewichtete Verspätung $\sum T_i$ betrachtet werden. Dabei gibt der Parameter T_i an, um wieviele Zeiteinheiten der Job J_i seine Deadline d_i verpasst hat.

2.2 Algorithmen zur Lösung von Scheduling Problemen

In diesem Kapitel werden einige gebräuchliche Lösungsverfahren für das Flow Shop Scheduling vorgestellt. Dabei werden sowohl Heuristiken, als auch Verfahren, die eine optimale Lösung garantieren, behandelt. Der Fokus wird hier auf die Verfahren für Flow Shops gelegt. In der Literatur existieren bereits zahlreiche Ansätze für das Offline Scheduling ([1], [4], [5], [6], [7]). Diesen ist jedoch gemein, dass oft nur Spezialfälle behandelt werden. Dabei liegt der Fokus meist auf Permutations Flow Shops ([6], [1]) oder Flow Shops mit einer sehr geringen Anzahl an Maschinen ([1], [7]). Permutations Flow Shops haben dabei den Vorteil, dass die Anzahl der möglichen Lösungen deutlich geringer ist, als die Anzahl der Lösungen im klassischen Flow Shop. Darüberhinaus wird in den meisten Fällen davon ausgegangen,

dass zwischen den Maschinen unendlich große Puffer existieren.
Für das Online Scheduling existieren hingegen nur sehr wenige Ansätze. Dabei liegt der Fokus hauptsächlich auf flexiblen Flow Shops ([8], [9]) und Flow Shops mit einer sehr geringen Anzahl an Maschinen ([10]). Daher lassen sich diese Ansätze nicht ohne weiteres auf das Rossmann-Problem übertragen.

2.2.1 Heuristiken

Heuristiken sind Lösungsverfahren, die in der Regel keine optimalen Ergebnisse erzielen. Für einige Heuristiken lässt sich beweisen, dass die Qualität eine untere Schranke niemals unterschreiten kann. Für andere Heuristiken können auf Basis von empirischen Untersuchungen Aussagen über eine mittlere Qualität getroffen werden.
Ein Beispiel für sehr einfache Heuristiken sind Prioritätsregeln ([11]). Dabei werden die einzelnen Jobs anhand bestimmter Charakteristika geordnet. Eine Prioritätsregel legt also eine Präzedenzrelation über einer Menge von Jobs fest. Tabelle 2 zeigt eine Auswahl üblicher Prioritätsregeln.

Prioritätsregeln
First Come First Serve
Kürzeste Operationszeit zuerst
Längste Operationszeit zuerst
Frühester Liefertermin zuerst
Kürzeste Durchlaufzeit zuerst
Längste Durchlaufzeit zuerst
Höchster Umsatz zuerst

Tabelle 2: Prioritätsregeln

Diese Regeln lassen sich additiv oder multiplikativ miteinander verknüpfen. Dabei können die einzelnen Regeln noch mit Faktoren gewichtet werden. Prioritätsregeln eignen sich nicht, um eine optimale Lösung von Scheduling Problemen zu garantieren. Jedoch kann mit diesen im Optimalfall in sehr kurzer Zeit eine gute Belegung gefunden werden. Daher sind Prioritätsregeln in der Industrie weit verbreitet.
Die Schwierigkeit besteht darin, auf Basis des intrinsischen Wissens über die Fertigungsanlage und die Jobs, eine oder mehrere passende Prioritätsregel(n) auszuwählen und gegebenenfalls geschickt zu verknüpfen.

2.2.2 Branch and Bound Algorithmen

Branch and Bound (Verzweigung und Schranke) Algorithmen zählen zu den optimalen Lösungsverfahren für Scheduling Probleme. Diesen wird sowohl für Flow Shops als auch für Job Shops eine hohe Aufmerksamkeit in der Literatur zuteil ([1], [12, S. 214ff], [13], [6]). Dabei kann beispielsweise für

Flow Shops ein Baum mit allen möglichen Permutationen betrachtet werden ([1]). Ein Beispiel für einen Baum, der alle Permutationen von drei Jobs angibt, ist in Abbildung 1 dargestellt.

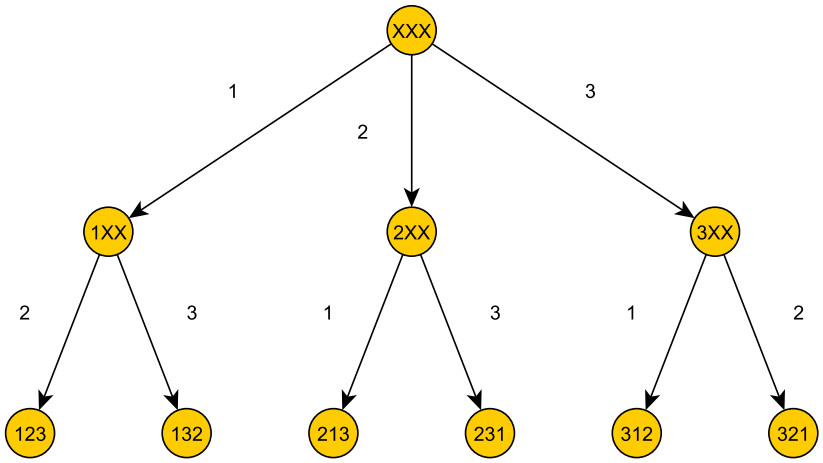

Abbildung 1: Permutationsbaum

An der Wurzel wurde noch kein Job ausgewählt. Dies wird mit der Beschriftung „XXX“ ausgedrückt. Die Kantenbeschriftung gibt an, welcher Job als nächstes ausgewählt wird. Über die Kante mit der Beschriftung „1“ erhalten wir die Permutation „1XX“, bei der für den ersten Platz der erste Job ausgewählt wurde. Die beiden andere Plätze sind noch nicht belegt. Wenn nun für den zweiten Platz der Job „2“ ausgewählt wird, muss sich der dritte Job an der dritten Position befinden. Somit erhält man die Permutation „123“. Die Blätter des Baums geben also alle möglichen Permutationen an.
Es ist leicht ersichtlich, dass die Menge der zu betrachteten Knoten des Baumes sehr stark mit der Anzahl der Jobs wächst. Die Anzahl der Knoten eines Permutationsbaums mit n Jobs lässt sich folgendermaßen berechnen.

$$K_n = \sum_{i=0}^{n-1} n^{\underline{i}} = \sum_{i=0}^{n-1} \frac{n!}{(n-i)!} \tag{9}$$

Daher lässt sich ein Permutationsbaum nicht in polynomieller Zeit traversieren. Das Ziel ist nun, jedem Knoten einen Wert zuzuordnen, der die Güte des Knotens angibt. Dieser Wert wird in der Literatur als „lower bound“ (LB) bezeichnet ([12, S. 56f]). Dabei wird die Funktion, die einem Knoten seine Güte zuordnet, so gewählt, dass ein Kindknoten niemals einen besseren Wert als sein Elternknoten erreichen kann. Im Folgenden wird diese Funktion als LB-Funktion bezeichnet. In Abbildung 2 ist ein Beispiel hierfür gegeben.

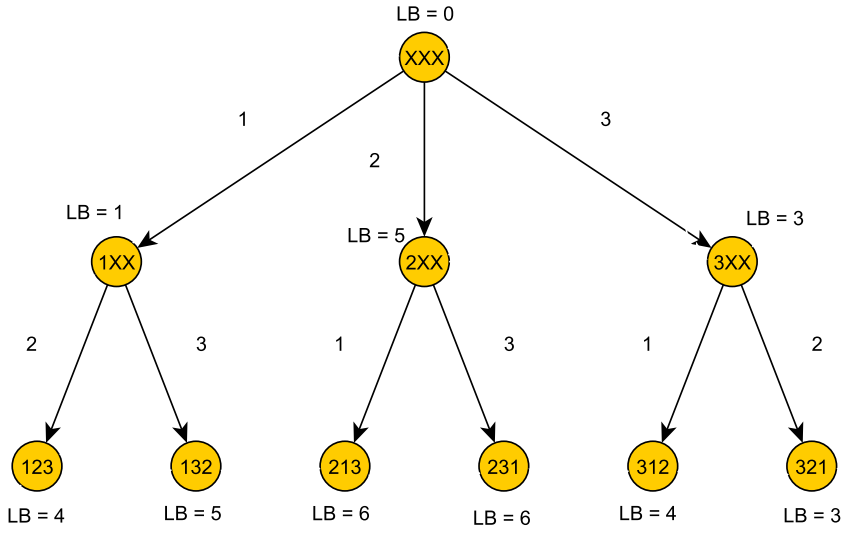

Abbildung 2: Permutationsbaum mit LB

Wenn nun über eine Tiefensuche bereits der Knoten „123“ exploriert wurde, müssen alle Knoten, die einen schlechteren LB-Wert als 4 haben, nicht mehr betrachtet werden. Wenn nun also als nächstes der Knoten „2XX“ betrachtet wird, erhalten wir einen LB-Wert von 5. Damit können die Kinder dieses Knotens keinen kleineren LB-Wert als 5 erreichen und müssen somit nicht weiter betrachtet werden. Dies wird durch den roten Schnitt in Abbildung 3 gekennzeichnet. Damit verkleinert sich der Suchraum im Beispiel um 1/3. Das Ziel des Branch and Bound Algorithmus besteht nun darin, frühzeitig zu erkennen, dass ein Ast des Baumes keine optimale Lösung enthalten kann. Damit muss im Idealfall nur ein kleiner Teil des Baums traversiert werden, um eine optimale Lösung zu erhalten. Im Worst-Case kann es jedoch dazu kommen, dass ein Großteil oder sogar der ganze Baum traversiert werden muss. Dies sollte unbedingt vermieden werden, da die Anzahl der Knoten im Permutationsbaum die Anzahl der möglichen Permutationen übersteigt. Daher besteht die Schwierigkeit von Branch and Bound Algorithmen darin, eine LB-Funktion zu finden, die einerseits leicht zu berechnen ist, und es andererseits erlaubt, frühzeitig Äste des Baumes abzuschneiden. Um zu verhindern, dass im Worst-Case der gesamte Baum traversiert wird, kann ein Zeitlimit eingeführt werden. Dabei ist das Ergebnis des Algorithmus entweder eine optimale Lösung oder die beste Lösung, die innerhalb des Zeitlimits

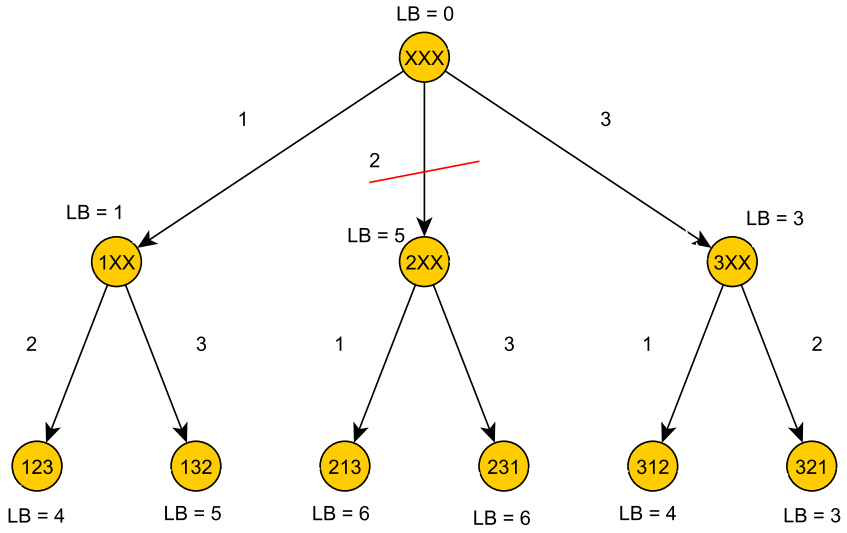

Abbildung 3: Permutationsbaum mit LB und Schnitt

gefunden wird. Wenn das Zeitlimit überschritten wird, kann die Optimalität des Ergebnisses nicht mehr garantiert werden.

2.2.3 Genetische Algorithmen

Genetische Algorithmen (GA) werden verwendet, um verschiedenste kombinatorische Optimierungsprobleme zu lösen ([14, S. 4]). Genetische Algorithmen simulieren eine Evolution nach natürlichem Vorbild. Ein GA läuft in mehreren Schritten ab ([14, S. 8f]). Zuerst wird eine Grundpopulation von potentiellen Lösungen erzeugt. Diese werden mittels einer Fitness-Funktion bewertet. Wenn eine Lösung mit hinreichender Qualität gefunden wird, terminiert der Algorithmus. Ansonsten werden gute Elemente der Grundpopulation selektiert und nach verschiedenen Regeln kombiniert, um eine neue Lösung zu erhalten. Die Kombination mehrerer Lösungen zu einer neuen Lösung erfolgt mittels sogenannter Crossover Operatoren. Anschließend kann die neue Lösung durch Mutationsoperatoren weiter verändert werden, um eine gewisse Varianz der Lösungen zu erreichen. Dabei kann ein genetischer Algorithmus entweder nur Mutationen, nur Crossover Operationen oder beide Operationen erlauben. Abbildung 4 zeigt den Prozess als Flussdiagramm.

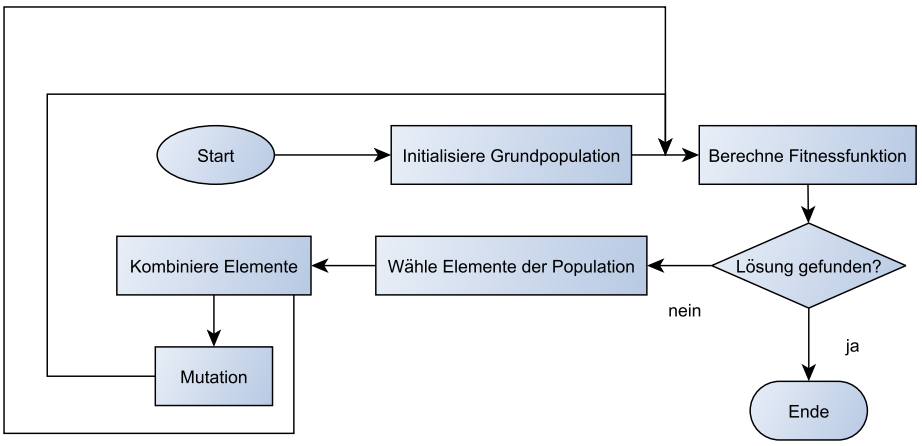

Abbildung 4: Vorgehensweise genetischer Algorithmus

Die Schwierigkeit bei vielen kombinatorischen Optimierungsproblemen besteht darin, dass für eine Lösung nur dann Optimalität garantiert werden kann, wenn der gesamte Suchraum betrachtet wird. Daher muss die Entscheidung, ob eine Lösung gut genug ist, getroffen werden, ohne das optimale Ergebnis zu kennen. Falls eine untere Schranke für das Problem bekannt ist, kann ein maximaler Abstand zur unteren Schranke vorgegeben werden. Wenn dieser Abstand vom Algorithmus nicht erreicht werden kann, terminiert dieser jedoch nie. Weiterhin kann auch eine maximale Anzahl an Iterationen oder eine maximale Rechenzeit vorgegeben werden, die dem Algorithmus zur Verfügung steht.

3 Modellbildung

In diesem Kapitel wird das Lager der Firma Rossmann zunächst in die in Kapitel 2 vorgestellten Maschinenbelegungsmodelle eingeordnet. Danach werden einige Begriffe und Definitonen eingeführt, die im weiteren Verlauf dieser Arbeit verwendet werden. Abschließend wird eine Abschätzung für eine untere und obere Schranke der Bearbeitungsdauer vorgenommen.

3.1 Einordung in die Maschinenbelegungsmodelle

Nachdem im vorigen Kapitel die wichtigsten Maschinenbelegungsmodelle vorgestellt wurden, soll nun das Lager der Firma Rossmann in die Modelle eingeordnet werden. Es handelt sich hierbei um einen speziellen Flow Shop. Eine Präzedenzrelation legt fest, in welcher Reihenfolge die Maschinen angefahren werden müssen. Diese Reihenfolge ist für jeden Job gleich. Im Gegensatz zum klassischen Flow Shop, muss ein Job im Rossmann-Lager nicht jede Maschine anfahren. Daher handelt es sich um einen Flow Shop, bei dem die Bearbeitungszeit eines Jobs auf einer Maschine auch den Wert Null annehmen kann. Dieser wird als generalisierter Flow Shop bezeichnet. Weiterhin kann ein einmal gestarteter Job nicht mehr unterbrochen werden. An jeder Maschine steht ein Eingangspuffer der Größe b zur Verfügung. Zudem sind die Transportzeiten zwischen den Maschinen bekannt. Somit kann das Rossmann-Lager durch folgendes Modell beschrieben werden.

$$(F_m|ltd(b); t_{jk}|C_{max}) \tag{10}$$

Dabei handelt es sich um einen generalisierten Flow Shop mit m Maschinen, bei dem das Ziel darin besteht, die Gesamtbearbeitungsdauer C_{max} zu minimieren. Das β Feld gibt in diesem Fall die Größe des Eingangspuffers b und die Transportzeit t_{jk} zwischen den Maschinen an.
Für das Online Scheduling kommen als weitere Parameter noch Freigabezeiten r_i, Deadlines d_i und Prioritäten w_i hinzu. Weiterhin besteht das Ziel darin, die totale gewichtete Verspätung $\sum w_i T_i$ zu minimieren.

$$(F_m|ltd(b); t_{jk}; r_i; d_i; w_i|\sum w_i T_i) \tag{11}$$

3.2 Begriffe und Definitionen

In diesem Abschnitt werden einige Begriffe und Definitionen eingeführt, die im weiteren Verlauf der Arbeit verwendet werden.

n :=	Anzahl der Jobs
m :=	Anzahl der Maschinen
b :=	Größe des Eingangspuffers
J_i :=	Job i, i = 1,...,n
M_j :=	Maschine j, j=1,...,m
d_i :=	Deadline für den Job i, $d_i \in \mathbb{N}_0$
p_{ij} :=	Bearbeitungszeit des Jobs i auf Maschine j, $p_{ij} \in \mathbb{N}_0$
O_{ij} :=	Bearbeitungsschritt (Task) des Jobs J_i auf der Maschine M_j mit der Bearbeitungszeit p_{ij}
r_i :=	Freigabetermin des Jobs i, $r_i \in \mathbb{N}_0$
s_{ij} :=	Bearbeitungsbeginn des Jobs i auf Maschine j, $s_{ij} \in \mathbb{N}_0$
t_{jk} :=	Transportzeit zwischen Maschine j und k, $t_{jk} \in \mathbb{N}_0$
w_i:=	Priorität des Jobs i, $w_i \in \mathbb{N}_0$
C_i :=	Fertigstellungstermin des Jobs i, $C_i \in \mathbb{N}$
S_i :=	Startzeitpunkt des Jobs i, $S_i \in \mathbb{N}_0$
T_i :=	Verspätung des Jobs i, $T_i = max(C_i - d_i, 0)$
P_i :=	Gesamtbearbeitungszeit des Jobs i, $P_i = \sum_{j=1}^{m} p_{ij}$
C_{max} :=	Gesamtbearbeitungszeit aller Jobs auf der Anlage
π :=	Bearbeitungsreihenfolge der Jobs

Tabelle 3: Definitionen

3.3 Abschätzung von oberen und unteren Schranken

Da die optimale Bearbeitungsdauer für eine hinreichend große Anzahl an Jobs nicht bekannt ist, muss eine Abschätzung vorgenommen werden. Dafür kann einerseits die Maschine mit der höchsten Gesamtbearbeitungsdauer herangezogen werden. Hieraus ergibt sich die Formel für die Berechnung einer unteren Schranke wie folgt.

$$S_{M,min} = max_{j=1,...,m}(\sum_{i=1}^{n} p_{ij}) \tag{12}$$

Andererseits kann der Job J_i mit der größten Gesamtbearbeitungszeit P_i betrachtet werden.

$$S_{J,min} = max_{i=1,...,n}(P_i) \tag{13}$$

Somit ergibt sich die untere Schranke als Maximum der Schranken $S_{M,min}$ und $S_{J,min}$.

$$S_{min} = max(S_{M,min}, S_{J,min}) \tag{14}$$

Diese untere Schranke dient lediglich zur Abschätzung und stellt in der Regel keine erreichbare Bearbeitungsdauer dar. Dies ist in dem Umstand begründet, dass ein Job, bevor oder nachdem dieser auf der Maschine mit der maximalen Bearbeitungsdauer bearbeitet wird, noch auf anderen Maschinen bearbeitet werden muss. Weiterhin kommen noch Transportzeiten zwischen den Maschinen hinzu.

	M_1	M_2	M_3	M_4	M_5	P_i
J_1	227	121	0	144	298	790
J_2	284	165	233	275	145	**1102**
J_3	111	269	228	222	0	829
J_4	0	0	0	0	112	112
J_5	118	0	222	156	0	496
Summe	740	555	683	**797**	655	

Tabelle 4: Berechnung der unteren Schranke für 5 Jobs und 5 Maschinen

Für das obige Beispiel ergibt sich nach diesem einfachen Verfahren zur Berechnung der unteren Schranke ein Wert von 1102 für den Job J_2. Weiterhin erhält man den Wert 797 für die Maschine M_4. Allerdings können die Jobs nicht sofort auf der Maschine M_4 bearbeitet werden. Der Job J_1 muss im Beispiel zuerst auf den Maschinen M_1 und M_2 bearbeitet werden. Damit kann Job J_1 frühestens zum Zeitpunkt 348 auf der Maschine M_4 gestartet werden. Der Job mit der geringsten Vorlaufzeit ist Job J_5 mit 340 Zeitschritten. Somit verlässt der letzte Job die Maschine M_4 nicht vor dem Zeitpunkt $797 + 340 = 1137$.
Analog zum Job mit der geringsten Vorlaufzeit muss noch der Job mit der geringsten Nachbearbeitungszeit bestimmt werden. Da sowohl der Job J_3 als auch der Job J_5 auf keinen weiteren Maschinen mehr bearbeitet werden muss, ergibt sich im Beispiel eine minimale Nachbearbeitungszeit von 0 Zeitschritten. Diese Betrachtungen werden für jede Maschine durchgeführt. Damit ergeben sich die aktualisierten Schranken für die Maschinen wie folgt.

$S_{M,1}$	$S_{M,2}$	$S_{M,3}$	$S_{M,4}$	$S_{M,5}$
1118	1108	957	**1137**	655

Tabelle 5: Aktualisierte Schranken

Somit ergibt sich im Beispiel als untere Schranke ein Wert von 1137 Zeitschritten.
Weiterhin ist es interessant abzuschätzen, welches die schlechteste Bearbeitungsdauer ist. Einige Algorithmen benötigen gegebenenfalls Informationen

darüber, wie groß der Schedule maximal werden kann. Dabei wird angenommen, dass keine Jobs parallel bearbeitet werden können. Man geht somit von einer rein sequentiellen Abarbeitung der Jobs aus. Damit lässt sich eine obere Schranke wie folgt berechnen.

$$S_{max} = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{ij} \tag{15}$$

Daraus folgt, dass die Bearbeitungsdauer für einen durch einen Algorithmus bestimmten Ablaufplan zwischen diesen beiden Schranken liegen muss.

$$S_{min} \leq C_{max} \leq S_{max} \tag{16}$$

Somit kann die Qualität einer Lösung als Differenz zwischen der berechneten Bearbeitungsdauer und der unteren Schranke, bezogen auf die untere Schranke, definiert werden.

$$Q = \frac{C_{max}}{S_{min}} - 1, Q \geqslant 0 \tag{17}$$

Dabei gibt die Qualitätsmaßzahl Q an, um wieviel Prozent die berechnete Lösung die untere Schranke übersteigt. Diese Qualitätsmaßzahl gilt es nun zu minimieren.

4 Erstellung von Schedules

In diesem Abschnitt wird das Verfahren vorgestellt, mit dem für eine gegebene Jobreihenfolge ein Schedule erstellt wird. Dabei sind die Transportzeiten t_{jk} zwischen den Maschinen, die Anzahl der Maschinen m, sowie die Größe der Eingangspuffer b bekannt. Gesucht ist nun der Startzeitpunkt S_i für jeden Job J_i. Weiterhin muss die Gesamtbearbeitungszeit C_{max} berechnet werden.

4.1 Schedulingverfahren

Jede Maschine kann genau einen Task bearbeiten und b-viele Tasks im Puffer aufnehmen. Der Belegungsplan für eine Maschine besteht somit aus einem Bearbeitungsbereich und einem Wartebereich. Diese können als Felder aufgefasst werden, bei denen die Position dem Zeitschritt entspricht. Der Wert an einem bestimmten Zeitschritt im Bearbeitungsfeld gibt an, ob diese Maschine zu diesem Zeitschritt frei ist. Analog dazu gibt der Wert im Pufferfeld an, wieviele Plätze im Wartebereich frei sind.
Um nun den Startzeitpunkt für einen Job zu bestimmen, wird, ausgehend vom Startzeitpunkt S_{i-1} des letzten Jobs J_{i-1}, für jeden Task geprüft, ob für die jeweilige Bearbeitungsdauer auf der Maschine genügend Platz bereitsteht. Ist dies nicht der Fall, wird der nächstmögliche Einfügezeitpunkt für den Task bestimmt. Daraus ergibt sich die Wartezeit auf der jeweiligen Maschine. Nun wird geprüft, ob der Puffer genügend Platz hat, um den Job bis zum Bearbeitungsbeginn aufzunehmen. Ist dies der Fall, wird der nächste Task betrachtet. Dabei ergibt sich der früheste Einfügezeitpunkt für den nächsten Task als Summe der vorherigen Warte-, Fahrt- und Bearbeitungszeiten. Wenn nun alle Tasks zu ihren jeweiligen Zeitpunkten gestartet werden können, wird der Job J_i zum Zeitpunkt t_i in den Schedule eingefügt. Ansonsten kann der Job J_i nicht zum Zeitpunkt t_i gestartet werden. Dann wird der Wert t_i inkrementiert und das Verfahren wird für den nächsten Zeitschritt erneut durchgeführt.
Ein Schedule muss also nach außen zwei Operationen anbieten. Zum einen eine Operation die prüft, ob ein Job J_i mit der Bearbeitungszeit P_i zum Zeitpunkt t_i in den Schedule eingefügt werden kann. Zum anderen eine Operation, die einen Job J_i mit der Bearbeitungsdauer P_i zum Zeitpunkt t_i einfügt. In Abbildung 5 wird der Einfügevorgang für einen Job zum Zeitpunkt t als Flussdiagramm dargestellt.

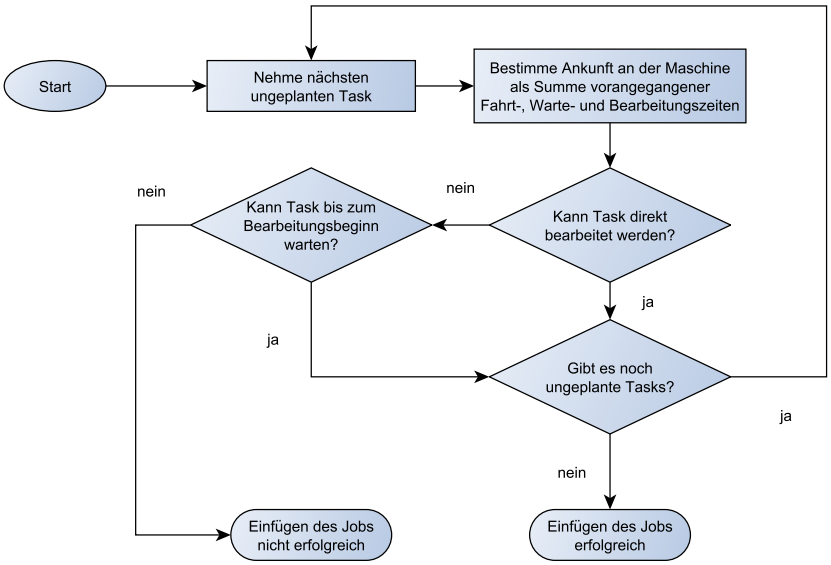

Abbildung 5: Einfügen eines Jobs zum Zeitpunkt t

4.2 Auswahl geeigneter Datenstrukturen

Wenn die Belegungspläne als Felder dargestellt werden (Abbildung 6), müssen sehr viele Positionen des Feldes betrachtet werden. Für einen Task O_{ij} eines Jobs J_i mit der Bearbeitungszeit p_{ij} müssen also mindestens p_{ij}-viele Elemente des Feldes betrachtet werden. Dies führt dazu, dass die Komplexität des Einfügevorgangs hauptsächlich von der Bearbeitungszeit p_{ij} abhängt.

Zeitpunkt	0	1	2	3	4	5	6	7	8	9	10	11	12
Belegt?	false	false	false	true	true	true	true	false	false	true	true	false	false
Puffer	3	3	3	2	2	3	3	3	3	3	3	3	3

Abbildung 6: Schedule als Feld

Weiterhin muss die Prüfung, ob ein Job J_i zum Zeitpunkt t_i eingefügt werden kann, effizient implementiert werden. Dies liegt daran, dass sehr häufig geprüft werden muss, ob ein Job eingefügt werden kann. Die Anzahl der Einfügevorgänge hängt hingegen nur von der Anzahl der Jobs ab.
Zur Optimierung des Einfügevorgangs müssen also geeignetere Datenstrukturen verwendet werden. Für den Bearbeitungsbereich einer Maschine können die einzelnen Elemente des Feldes nur zwei Werte annehmen. Entweder ist die Maschine zum Zeitpunkt t frei oder sie ist belegt. Daher können die freien Positionen als Intervalle in einer Liste nachgehalten werden. Zu Beginn des

Schedulings ist die Maschine frei. Daher enthält die Liste nur das Element [0, ∞]. Dies bedeutet, dass die Maschine im Intervall [0, ∞] frei ist. Nun soll ein Task O_{ij} zum Zeitpunkt t mit der Bearbeitungsdauer $d = p_{ij}$ eingefügt werden. Zuerst muss geprüft werden, ob die Maschine ab dem Zeitpunkt t für d Zeitschritte frei ist. Dazu wird, beginnend vom ersten Intervall an, geprüft, ob die obere Grenze größer gleich $(t+d-1)$ ist. Ist dies der Fall, wird für das betreffende Intervall geprüft, ob die untere Grenze kleiner gleich t ist. Nur wenn beide Bedingungen erfüllt sind, kann der Task eingefügt werden. Damit ergibt sich folgende Liste.

$$\{[0, t-1], [t+d, \infty]\} \tag{18}$$

Nach dem Einfügen von n Tasks besteht die Liste also aus maximal $n+1$ verschiedenen Intervallen. Dabei werden Intervalle der Länge 0 aus der Liste entfernt. Somit ist die Komplexität des Einfügevorgangs unabhängig von der Bearbeitungsdauer p_{ij} eines Tasks. Weiterhin ist die Bestimmung des frühest möglichen Bearbeitungszeitpunkts eines Tasks, ausgehend vom Zeitschritt t, sehr effizient.
Für den Wartebereich eignet sich diese Datenstruktur nicht ohne weiteres, da jedes Element des Feldes $b+1$ verschiedene Werte annehmen kann. Bei eingehender Betrachtung des Feldes fällt auf, dass dieses lange Folgen gleicher Elemente enthält. Daher eignet sich hierfür eine Lauflängenkodierung. Der Wartebereich wird also als Liste modelliert, in der die einzelnen Elemente Folgen von gleichen Elementen darstellen. Zu Beginn enthält die Liste nur die Folge [b, ∞]. Dies bedeutet, dass zu allen Zeitschritten noch die volle Pufferkapaziät b zur Verfügung steht. Wenn nun ein Platz im Puffer ab dem Zeitpunkt t für die Wartezeit w reserviert werden soll, muss wiederum geprüft werden, ob der Puffer zwischen den Positionen $x = t$ und $y = (t + w - 1)$ frei ist. Dazu werden die Folgen in der Warteliste, beginnend von der ersten Folge an betrachtet. Dabei wird die Länge der Folgen summiert. Überschreitet die Summe den Wert x, muss, ausgehend von der letzten betrachteten Folge, geprüft werden, ob der Wert der Folge größer 0 ist. Dies geschieht so lange, bis die Summe den Wert y überschreitet. Der Puffer hat genau dann genügend Kapazitäten, wenn zwischen den Positionen x und y keine Nullfolge existiert. Im Beispiel kann der Puffer im Intervall [x, y] reserviert werden. Dazu werden die vorhandenen Folgen an den Punkten x und y geteilt. Damit ergibt sich folgende Liste.

$$\{[b, t-1], [b, w], [b, \infty]\} \tag{19}$$

Nun wird der Wert jeder Folge zwischen den Punkten x und y um eins dekrementiert.

$$\{[b, t-1], [b-1, w], [b, \infty]\} \tag{20}$$

In einem optionalen Kompressionsschritt können aufeinanderfolgende Folgen mit gleichen Werten wieder zusammengefasst werden.

5 Offline Scheduling

In diesem Kapitel werden Ansätze für das Offline Scheduling untersucht. Dabei ist die zu bearbeitende Auftragsmenge vor der Abarbeitung bekannt und alle Jobs stehen zu Beginn des Schedulings zur Verfügung. Weiterhin sind die Transportzeiten t_{jk} zwischen den Maschinen, die Bearbeitungszeiten auf den Maschinen p_{ij} und die Größe des Eingangspuffers b bekannt. Das Ziel besteht darin, die Gesamtbearbeitungszeit C_{max} zu minimieren. Somit muss das folgende Flow Shop Scheduling Problem gelöst werden.

$$(F_m|ltd(b); t_{jk}|C_{max}) \tag{21}$$

5.1 Heuristiken und Algorithmen

In diesem Abschnitt werden einige bekannte und neue Heuristiken verglichen.

5.1.1 Prioritätsregeln

Die einfachsten Heuristiken, die untersucht werden können, sind Prioritätsregeln. Hier werden zwei Prioritätsregeln genutzt. Die erste Regel ist die SJL (Shortest Job Last) Regel. Dabei werden die Aufträge absteigend nach ihrer Gesamtbearbeitungszeit geordnet und dann wird auf Basis dieser festen Reihenfolge ein Schedule erstellt. Die zweite Regel, die hier betrachtet wird, ist die SJF (Shortest Job First) Regel. Diese entspricht der umgekehrten Reihenfolge der SJL Regel.

5.1.2 NEH Heuristik

Die NEH Heuristik wurde 1982 von Nawaz, Enscore und Ham entwickelt ([4]). Diese Heuristik zielt darauf ab, gute Jobreihenfolgen für Permutations Flow Shops zu finden. Jedoch kann diese auch für andere Flow Shops verwendet werden.
Der Algorithmus läuft dabei in mehreren Schritten ab. Zuerst wird die Gesamtbearbeitungszeit P_i für jeden Job J_i bestimmt. Anschließend werden die Jobs absteigend nach ihrer Gesamtbearbeitungszeit geordnet. Somit erhält man eine sortierte Liste der Jobs. Nun wird der erste Job aus dieser sortierten Liste genommen und in den Schedule eingefügt. Danach wird der zweite Job gewählt und es werden zwei Teilschedules erstellt. Zum einen der Schedule mit der Reihenfolge $\pi_1 = [J_1, J_2]$ und zum anderen der Schedule mit der Reihenfolge $\pi_2 = [J_2, J_1]$. Der Teilschedule mit der geringsten Bearbeitungszeit C_{max} wird ausgewählt. Für den nächsten Job werden wieder alle Teilschedules untersucht, die sich ergeben, wenn der Job an jede Stelle in den vorher bestimmten Schedule eingefügt wird. Aus diesen wird wiederum der beste ausgewählt. Wenn alle Jobs betrachtet wurden, gibt der Algorithmus

die Reihenfolge π_{min} der Jobs zurück, die die geringste Gesamtbearbeitungszeit C_{max} erreicht hat.
Somit müssen für den Job J_i i verschiedene Teilschedules gebildet werden. Für eine Menge von n Jobs müssen also $1+2+...+n$ verschiedene Teilschedules erstellt und ausgewertet werden. Damit ergibt sich die Anzahl der zu berechnenden Schedules wie folgt.

$$a(n) = \sum_{i=1}^{n} i = \frac{n * (n+1)}{2} \tag{22}$$

Die NEH Heuristik hat also eine quadratische Komplexität. Die Erstellung eines Schedules hat im besten Fall eine lineare Komplexität. Damit ist die Laufzeitkomplexität der NEH Heuristik mindestens kubisch. Im Folgenden wird eine parallelisierte NEH Heuristik verwendet. Dabei wird die Erstellung eines Teilschedules als Task aufgefasst und parallel abgearbeitet.

5.1.3 First Fit Heuristik

Bei der First Fit Heuristik werden die Jobs analog zur NEH Heurstik absteigend nach ihrer Gesamtbearbeitungszeit P_i geordnet. Dann wird der erste Job zum Zeitpunkt 0 in den Schedule eingefügt. Für jeden neuen Job wird nun, ausgehend vom Zeitpunkt 0, der früheste Einfügezeitpunkt (First Fit) bestimmt. Das Ziel besteht also darin, jeden Auftrag möglichst früh in den Schedule einzufügen. Dadurch verändert sich in der Regel die Reihenfolge der Jobs. Somit hängt die Anzahl der zu betrachtenden Einfügepositionen für einen Job von der Gesamtbearbeitungszeit C'_{max} des bisher erstellten Schedules ab. Im besten Fall muss nur der erste Einfügezeitpunkt betrachtet werden. Im schlechtesten Fall müssen C'_{max}-viele Einfügepositionen betrachtet werden. Somit ergibt sich für die First Fit Heuristik eine quadratische Komplexität.

5.1.4 Best Fit Heuristik

Bei der Best Fit Heuristik werden die Jobs absteigend nach ihrere Gesamtbearbeitungszeit P_i sortiert. Der erste Job wird dann an der Position 0 in den Schedule eingefügt. Für die nächste zu betrachtende Einfügeposition wird zur Einfügeposition des vorigen Jobs der Mindestabstand, der zwischen zwei zu startenden Aufträgen eingehalten werden muss, addiert. Nun werden alle noch nicht geplanten Jobs betrachtet und es wird der längste Job ausgewählt, der an die neue Einfügepostion passt. Wenn sich keiner der verbleibenden Jobs zu diesem Zeitpunkt einfügen lässt, wird die zu betrachtende Einfügeposition um eins erhöht. Dies wird so lange wiederholt, bis keine Jobs mehr übrig sind. Dies entspricht einem Greedy Algorithmus, der aus einer Menge von Jobs immer den längsten Job als nächstes auswählt, der am frühesten eingefügt werden kann. Die Grundüberlegung, die hinter dem

Algorithmus steht, ist, lange Aufträge so früh wie möglich zu starten. Dabei werden die Aufträge, im Gegensatz zur SJL Prioritätsregel, nicht strikt der Länge nach gestartet, sondern es wird immer der längste passende Auftrag aus der verbleibenden Auftragsmenge als nächstes ausgewählt.

5.1.5 Branch and Bound Algorithmus

Für das gegebene Problem muss eine geeignete LB-Funktion gefunden werden. Diese muss sich dadurch auszeichnen, dass der LB-Wert eines Kindknotens den LB-Wert des Elternknotens nicht unterschreiten kann. Auf den ersten Blick erfüllt die Gesamtbearbeitungszeit C_{max} diese Bedingung. Allerdings würde dies dazu führen, dass jeweils zuerst Teilschedules, bestehend aus kurzen Jobs, vor Schedules aus längeren Jobs betrachtet werden. Es ist offensichtlich, dass das späte Hinzunehmen von längeren Jobs in der Regel nicht zu einem optimalen Ergebnis führt. Der Branch and Bound Algorithmus würde mit einer LB-Funktion, basierend auf der Gesamtbearbeitungszeit C_{max}, zwar ein optimales Ergebnis finden. Jedoch werden Äste des Baums erst sehr weit unten abgeschnitten. Im Worst-Case wird also erst auf der untersten Ebene festgestellt, dass die untersuchte Permutation kein optimales Ergebnis erzielt. Daher stellt die Gesamtbearbeitungszeit C_{max} der einzelnen Teilschedules keine gute Grundlage für eine LB-Funktion dar. Weiterhin kann auch die Qualitätsmaßzahl Q nicht als LB-Funktion verwendet werden. Das Einfügen eines Jobs in einen Teilschedule kann zu einer höheren Packungsdichte und damit zu einem geringeren Wert für Q führen. Dies widerspricht der geforderten Eigenschaft für die LB-Funktion.
Auf Grund dieser Überlegungen und dem Umstand, dass der Branch and Bound Algorithmus wegen seiner Laufzeit nur für eine sehr geringe Anzahl an Maschinen und Aufträgen verwendet werden kann, wird dieser Algorithmus im Folgenden nicht verwendet.

5.1.6 Genetische Algorithmen

In diesem Kapitel werden einige Varianten eines genetischen Algorithmus untersucht. Dabei wird der Fokus auf verschiedene Mutationsstrategien gelegt. Crossover Operationen werden nicht untersucht, da es kein sinnvolles Verfahren gibt, um zwei gute Permutationen zu einer neuen guten Permutation zu verknüpfen. Weiterhin wird ein Verfahren zur Erzeugung einer Grundpopulation vorgestellt, welches ohne Crossover- und Mutationsoperatoren auskommt.
Eine einfache Mutationsstrategie besteht im Vertauschen von zwei Jobs. Für eine Menge von n Jobs gibt es $(n-1)$ Möglichkeiten, den ersten Job zu vertauschen, $(n-2)$ Möglichkeiten für den zweiten Job und allgemein $(n-i)$

Möglichkeiten für den i-ten Job.

$$T = \sum_{i=1}^{n} (n - i) = \frac{(n-1)(n-2)}{2} \tag{23}$$

Damit gibt es T verschiedene Mutationen für eine gegebene Reihenfolge von n Jobs. Einerseits kann eine Teilmenge der Nachkommen durch zufällige Tauschoperationen erzeugt werden. Andererseits können alle möglichen Mutationen betrachtet werden. Dabei können die einzelnen Permutationen als Knoten in einem Graphen aufgefasst werden. Permutationen, die sich um nur einen Tausch unterscheiden, werden dann über Kanten verbunden. Ein Graph für eine Menge von drei Jobs ist in Abbildung 7 zu sehen. Hier gibt es

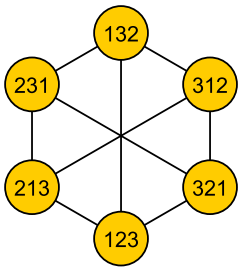

Abbildung 7: Permutationsbaum

zwei Varianten. Zum einen können alle T Vertauschungen einzeln untersucht werden. Dann wird aus allen T Tauschoperationen genau die ausgewählt, die die geringste Gesamtbearbeitungszeit C_{max} hat. Zum anderen können zuerst alle möglichen Vertauschungen für den ersten Job geprüft werden. Dann wird der Tausch ausgewählt, welcher die geringste Gesamtbearbeitungszeit C_{max} erreicht. Ausgehend von dieser neuen Permutation wird dann jede Vertauschung für den zweiten Job untersucht. Im i-ten Schritt des Algorithmus wird also jeweils die beste Vertauschung für die i-te Position bestimmt. Das Problem, welches sich bei diesen Mutationsstrategien ergibt, besteht darin, dass durch einen Tausch nur benachbarte Permutationen erreicht werden können. Wenn sich also in der direkten Nachbarschaft einer Permutation nur schlechtere Permutationen befinden, können weiter entfernte, bessere Permutationen nie erreicht werden. Daher besteht die Gefahr, dass nur lokale Minima im Permutationsgraphen gefunden werden. Eine Möglichkeit dem entgegenzuwirken besteht darin, kurzzeitig auch Verschlechterungen zuzulassen. Ein anderer Ansatz kann darin bestehen, jeweils die besten k Nachkommen zu behalten. Weiterhin können statt einer Tauschoperation auch eine Folge von n Tauschoperationen genutzt werden, um das Finden von lokalen Minima zu umgehen. Dabei werden beispielsweise zwei Tauschoperationen gewählt. Die Permutation wird durch die Tauschoperationen

verändert und anschließend mit der Fitnessfunktion bewertet. Wenn sich eine höhere Fitness ergibt, werden die Tauschoperationen beibehalten. Ansonsten werden diese zurückgenommen und es werden zwei neue zufällige Vertauschungen erzeugt. Dies kann jedoch dazu führen, dass direkt benachbarte Permutationen nicht mehr erreicht werden.
Eine weitere Möglichkeit, eine Menge von potentiellen Lösungen zu erhalten, ist das Mischen der Permutation. Dabei wird zufällig eine Anzahl von Permutationen erzeugt und diejenige ausgewählt, die die geringste Gesamtbearbeitungszeit C_{max} aufweist. Auf weitere Mutationen und Crossoveroperationen wird in diesem Fall verzichtet, da eine zufällig erzeugte Reihenfolge unabhängig von der Ausgangsreihenfolge ist.

5.2 Analyse der Heuristiken und Algorithmen

In diesem Kapitel werden die einzelnen Lösungsverfahren bezüglich ihrer Qualität und Laufzeit bewertet. Zunächst wird auf die Generierung von Probleminstanzen eingegangen. Anschließend wird jede Heuristik einzeln betrachtet und bewertet. Abschließend werden die Heuristiken untereinander verglichen.

5.2.1 Generieren von Probleminstanzen

Zur Analyse der Heuristiken wurden 200 verschiedene Probleminstanzen für einen Rossmann Flow Shop mit 20 Maschinen und einer Puffergröße von 3 generiert.

$$(F_{20}|ltd(3); t_{jk}|C_{max}) \tag{24}$$

Es werden jeweils 20 Probleminstanzen für verschiedene Auftragsanzahlen erzeugt. Für jeden Job wird die Anzahl der Operationen und die Bearbeitungszeit zufällig aus einem Intervall bestimmt. Die Parameter für die ersten 100 Instanzen sind in der Tabelle 6 aufgelistet.

	Instanzen				
Parameter	1-20	21-40	41-60	61-80	81-100
n	100	200	300	400	500
m	20				
Tasks/Job	$[1-10]$				
p_{ij}	$[100-450]$				

Tabelle 6: Probleminstanzen 1-100

Tabelle 7 zeigt die Parameter für die Probleminstanzen 101-200. Dabei wird die Anzahl der Maschinen m gleich 10 gewählt. Die übrigen Parameter entsprechen denen der vorigen Instanzen.
Jede Heuristik wird mit den ersten 100 Probleminstanzen getestet. Ausgewählte Heuristiken werden mit allen Probleminstanzen getestet. Dabei

	Instanzen				
Parameter	101-120	121-140	141-160	161-180	181-200
n	100	200	300	400	500
m	10				
Tasks/Job	$[1-10]$				
p_{ij}	$[100-450]$				

Tabelle 7: Probleminstanzen 101-200

wird die Laufzeit der Heuristik und die Gesamtbearbeitungsdauer C_{max} gemessen. Die Qualität der verschiedenen Verfahren wird mit Hilfe der Qualitätsmaßzahl Q verglichen. Dabei wird der Mittelwert über jeweils 20 Probleminstanzen gebildet.

5.2.2 Prioritätsregeln

Die beiden Prioritätsregeln SJF (kürzester Auftrag zuerst) und SJL (längster Auftrag zuerst) werden mit den Probleminstanzen 1-100 getestet. Die Laufzeit des Prioritätsschedulings bleibt über alle Probleminstanzen hinweg unter einer Sekunde. Als Referenzwert für die Qualität wird eine zufällige Reihenfolge verwendet. Dabei zeigt sich, dass keine der beiden Prioritätsregeln eine signifikant bessere Qualität als die zufällige Reihenfolge erreicht. Weiterhin schwankt die erzielte Qualität der SJL Regel sehr stark. Für die Probleminstanz 8 mit 100 Aufträgen wurde ein Wert von 49 Prozent für Q erzielt. Für die Probleminstanz 1 wurde hingegen nur ein Wert von 122 Prozent erzielt. Kleinere Werte für Q bedeuten dabei eine bessere Qualität der Lösung. Es zeigt sich also, dass sich die hier betrachteten Prioritätsregeln nicht eignen, um gute Reihenfolgen für das Rossmann-Problem zu erzeugen.

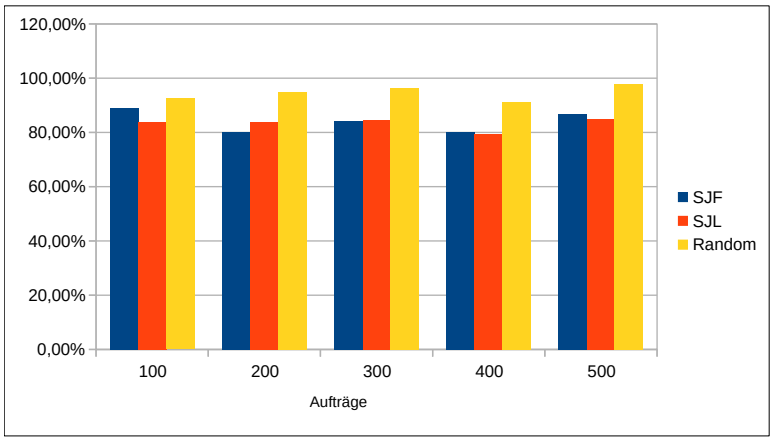

Abbildung 8: Qualität der Prioritätsregeln, Probleminstanzen 1-100

5.2.3 NEH Heuristik

Die NEH Heuristik wird zunächst mit den Probleminstanzen 1-100 getestet. Die Zahl hinter der Heuristik gibt dabei die Anzahl der Threads an, mit denen der Schedule berechnet wurde. Dabei zeigt sich, dass die Qualität über alle Probleminstanzen hinweg deutlich besser, als die der zufälligen Reihenfolge ist. Mit steigender Auftragsanzahl verbessert sich die Qualität der NEH Heuristik. Ab einer Auftragsanzahl von 400 Aufträgen wird eine Qualität von 10 Prozent erreicht.

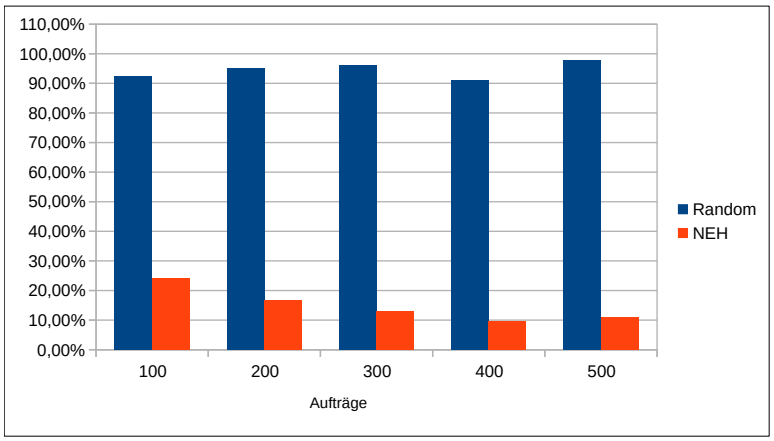

Abbildung 9: Qualität der NEH Heuristik, Probleminstanzen 1-100

Ein ähnliches Ergebnis zeigt sich für die Probleminstanzen $101 - 200$. Dabei fällt auf, dass die Qualität für diese Probleminstanzen deutlich besser ist. Daraus lässt sich folgern, dass die Qualität der NEH Heuristik ansteigt, wenn die Auslastung der Maschinen höher ist. Der beste Wert wird für die Probleminstanz 153 erreicht. Dieser beträgt 0,3 Prozent. Der schlechteste Wert ergibt sich für die Probleminstanz 116 mit 10 Prozent. Somit zeigt sich, dass die NEH Heuristik eine Qualität erzielt, die sehr nah am Optimum liegt.
Weiterhin werden die Laufzeiten für die NEH Heuristik gemessen. Zunächst werden die Laufzeiten für die parallelisierte NEH Heuristik mit 4 Threads betrachtet. Dabei zeigt sich, dass die Laufzeit der NEH Heuristik für eine große Anzahl an Aufträgen sehr stark ansteigt. Es zeigt sich, dass die Laufzeit für die Probleminstanzen 101-200 jeweils halb so groß ist, wie die Laufzeit für die Probleminstanzen 1-100.
Schließlich wird die Auswirkung der Parallelisierung auf die Laufzeit untersucht. Dafür werden nur die Probleminstanzen 1-60 betrachtet.

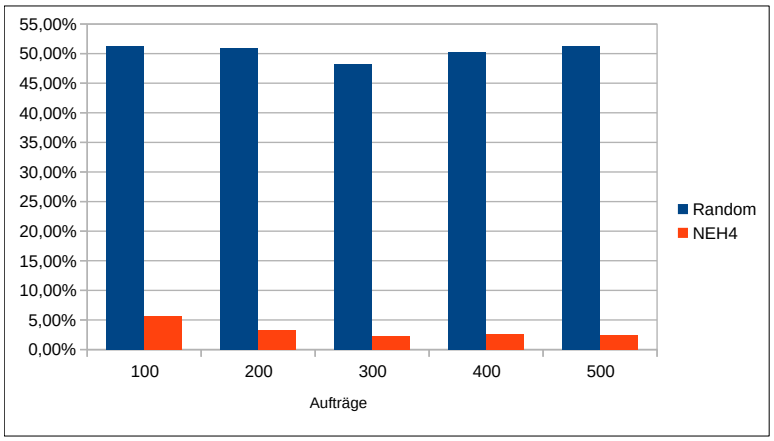

Abbildung 10: Qualität der NEH Heuristik, Probleminstanzen 101-200

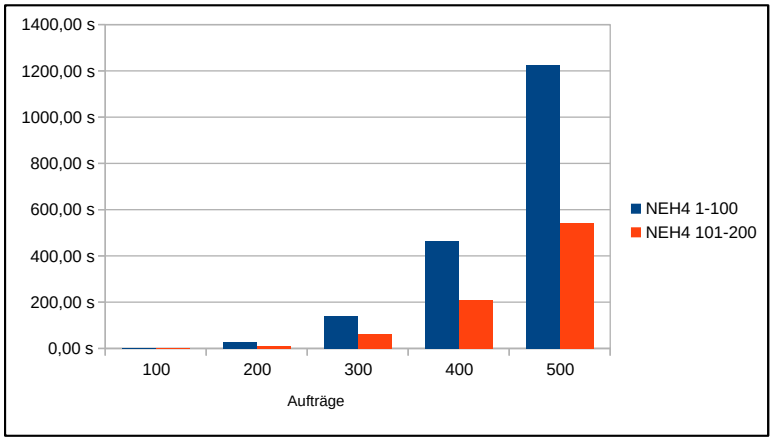

Abbildung 11: Laufzeit der NEH Heuristik mit 4 Threads

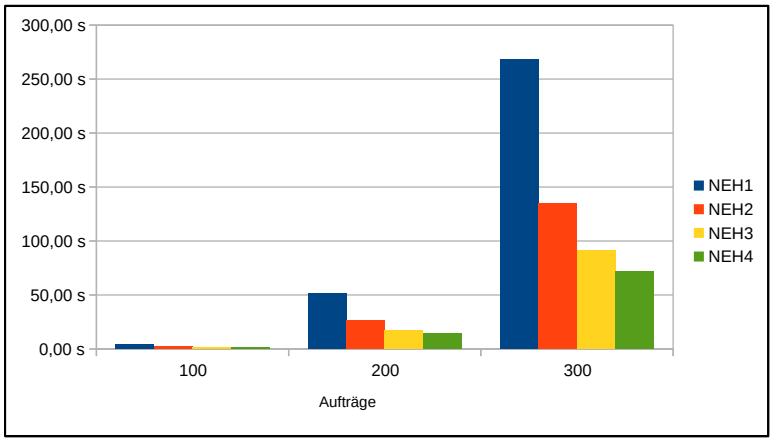

Abbildung 12: Laufzeit der NEH Heuristik in Abhängigkeit der Threads, Probleminstanzen 1-60

Wie erwartet, sinkt die Laufzeit für eine höhere Anzahl an Threads. Dabei ist die NEH2 Heuristik im Mittel 1,96 mal schneller als die serielle NEH1 Heuristik. Die NEH3 Heuristik ist 2,93 mal schneller und die NEH4 Heuristik ist 3,61 mal schneller als die NEH1 Heuristik. Somit ergibt sich durch die Parallelisierung eine große Verbesserung der Laufzeit. Der Faktor, um den sich die Laufzeit verbessert, wächst proportional mit der Anzahl der Threads.

5.2.4 First Fit Heuristik

Die First Fit Heuristik wird mit den Probleminstanzen 1-100 getestet. Als Referenzwert wird die zufällige Reihenfolge Random verwendet. Dabei zeigt sich, dass die First Fit Heuristik keinen signifikanten Qualitätsvorteil gegenüber der zufälligen Reihenfolge erzielt. Die Laufzeit der First Fit Heuristik beträgt jeweils unter zwei Sekunden. Somit eignet sich die First Fit Heuristik nicht, um gute Reihenfolgen für das Rossmann-Problem zu berechnen.

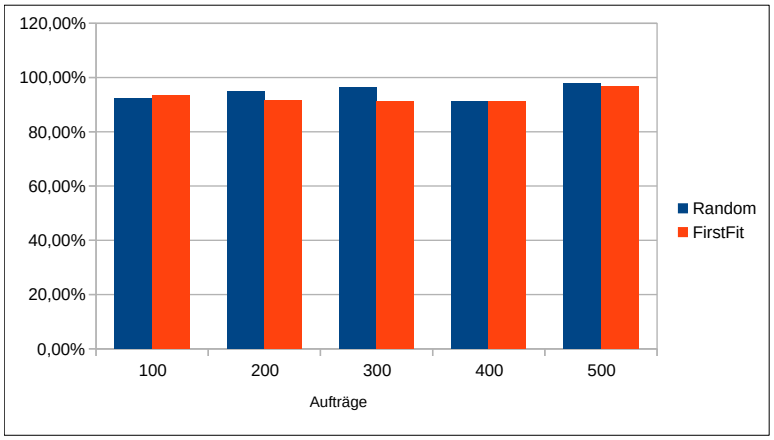

Abbildung 13: Qualität der First Fit Heuristik, Probleminstanzen 1-100

5.2.5 Best Fit Heuristik

Die Best Fit Heuristik wird zunächst mit den Probleminstanzen 1-100 getestet. Dabei wird die zufällige Reihenfolge Random als Referenzwert verwendet. Hierbei zeigt sich, dass die Best Fit Heuristik einen signifikanten Qualitätsvorteil gegenüber der zufälligen Reihenfolge erzielt. Die Qualität der Best Fit Heuristik steigt mit wachsender Auftragszahl.
Analog zur NEH Heuristik ergibt sich für die Probleminstanzen 101-200 eine noch höhere Qualität. Ab einer Auftragsanzahl von 300 beträgt der Wert für Q maximal 10 Prozent. Der beste Wert von 1,9 Prozent wird für die

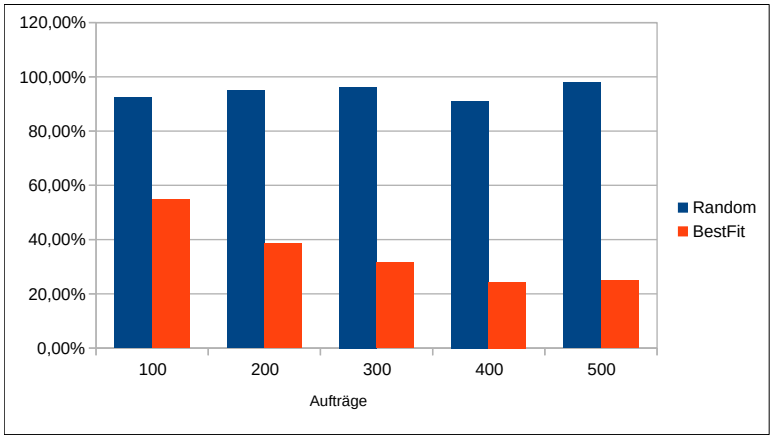

Abbildung 14: Qualität der Best Fit Heuristik, Probleminstanzen 1-100

Probleminstanz 153 erzielt. Der schlechteste Wert für Q ergibt sich für die Probleminstanz 112 mit 31,5 Prozent.

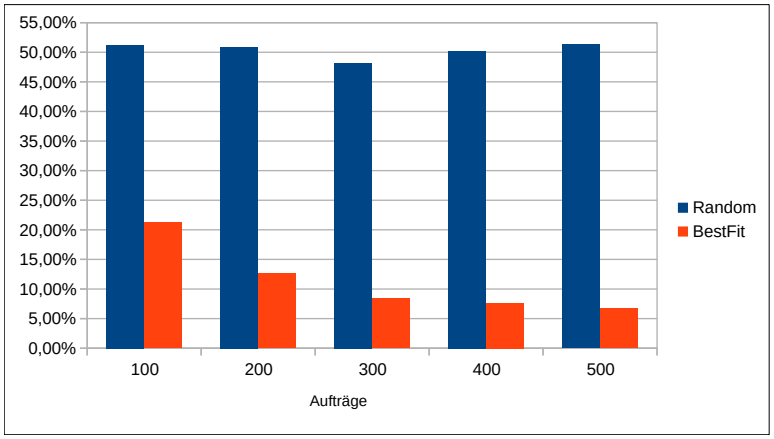

Abbildung 15: Qualität der Best Fit Heuristik, Probleminstanzen 101-200

Weiterhin werden die Laufzeiten der Best Fit Heuristik gemessen. Diese betragen über alle Probleminstanzen hinweg höchstens 7 Sekunden. Somit zeigt sich, dass sich mit der Best Fit Heuristik in einer sehr kurzen Zeit eine Verbesserung der Qualität erzielen lässt.

5.2.6 Genetische Algorithmen

Die verschiedenen genetischen Algorithmen werden mit den Probleminstanzen 1-100 getestet. Die Algorithmen, welche vom Zufall abhängen, werden für jede Probleminstanz 10 mal wiederholt. Dann wird der Mittelwert über die Ergebnisse gebildet. Es werden vier verschiedene Algorithmen getestet. Der erste Algorithmus führt zufällig 1000 Vertauschungen durch und prüft

nach jeder Vertauschung, ob eine Verbesserung erzielt wird. Ist dies der Fall, wird der Tausch beibehalten. Ansonsten wird der Tausch verworfen. Dieser Algorithmus wird mit GA2 bezeichnet. Als zweites wird ein Algorithmus getestet, der immer zwei Vertauschungen auf einmal durchführt. Dieser läuft analog zum GA2 Algorithmus ab und wird mit GA3 bezeichnet. Als drittes wird ein Algorithmus untersucht, der in jedem Schritt eine zufällige Reihenfolge erzeugt. Wenn diese besser als die vorige ist, wird die neue Reihenfolge übernommen. Dieser Algorithmus wird mit ShuffleGA bezeichnet und mit 1000 Iterationen durchgeführt. Schließlich wird ein Algorithmus getestet, der eine vollständige Nachbarschaftssuche durchführt. Dabei werden zunächst alle Vertauschungen für die erste Position untersucht. Die beste wird übernommen. Danach wird für die folgenden Positionen analog vorgegangen. Dieser Algorithmus wird als NeighbourGA bezeichnet. Die Abbildung 16 zeigt die Qualität der verschiedenen Algorithmen für die Probleminstanzen 1-100.

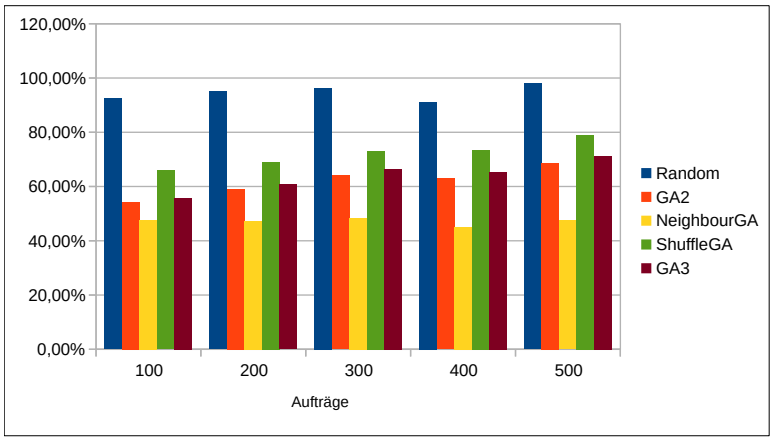

Abbildung 16: Qualität der genetischen Algorithmen, Probleminstanzen 1-100

Es zeigt sich, dass jeder der getesteten genetischen Algorithmen eine bessere Qualität als die zufällige Reihenfolge Random erreicht. Dabei erreicht der Algorithmus ShuffleGA die schlechteste Qualität. Dies ist zu erwarten, da dieser Algorithmus lediglich eine Menge von zufälligen Reihenfolgen erzeugt und die beste auswählt. Die beiden Algorithmen GA2 und GA3 erzielen eine minimal bessere Qualität. Dabei zeigt sich, dass der GA2 Algorithmus einen leichten Vorteil gegenüber dem GA3 Algorithmus aufweist. Der genetische Algorithmus mit der besten Qualität ist der NeighbourGA. Diese ist im Gegensatz zu den anderen Algorithmen über verschiedene Auftragsanzahlen hinweg konstant.
Weiterhin werden die Laufzeiten der verschiedenen genetischen Algorithmen betrachtet. Diese sind in Abbildung 17 logarithmisch aufgetragen. Die Algo-

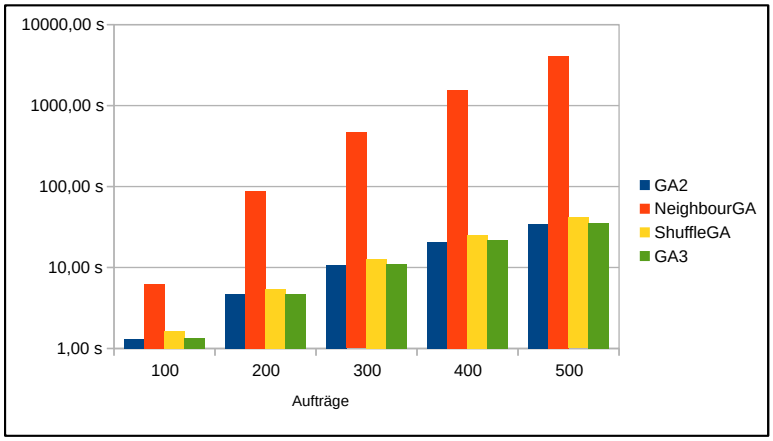

Abbildung 17: Laufzeit der genetischen Algorithmen, Probleminstanzen 1-100

rithmen GA2, GA3 und ShuffleGA haben ungefähr die gleiche Laufzeit. Dies liegt daran, dass die Anzahl der Schedules, die jeder der Algorithmen auswertet, um zu einer Lösung zu gelangen, konstant ist. Diese Anzahl beträgt jeweils 1000. Die Laufzeit des NeighbourGA ist deutlich höher. Dies liegt daran, dass die Anzahl der zu untersuchenden Vertauschungen quadratisch mit der Anzahl der Aufträge wächst. Es zeigt sich, dass die Verbesserung der Qualität durch den NeighbourGA mit einer hohen Zunahme der Laufzeit erkauft wird. Durch eine Parallelisierung kann diese noch verringert werden.

5.2.7 Einfluss der Puffergröße auf die Qualität

Weiterhin wird der Einfluss der Puffergröße auf die Qualität der Heuristiken untersucht. Dabei werden verschiedene Puffergrößen gewählt und die Qualität für die Probleminstanzen 1-20 und 21-40 gemessen. Es werden die NEH4 Heuristik, die Best Fit Heuristik, die First Fit Heuristik und die zufällige Reihenfolge verglichen. Hierbei wird einerseits deutlich, dass sich ab einer Puffergröße von 4 keine signifikante Verbesserung der Qualität erzielen lässt. Andererseits zeigt sich, dass sich die Qualität der Best Fit Heuristik und der zufälligen Reihenfolge mit zunehmender Puffergröße annähert. Dies lässt sich darauf zurückführen, dass es sich bei der Best Fit Heuristik um eine abgewandelte SJL Heuristik handelt. Wenn die Puffergröße zunimmt, kommt es daher seltener dazu, dass der jeweils längste Job nicht direkt eingefügt werden kann. Daher nähert sich die Reihenfolge, die die Best Fit Heuristik bestimmt, immer mehr der Reihenfolge an, die man erhält, wenn die Aufträge absteigend nach ihrer Gesamtbearbeitungszeit geordnet werden.
Aus den Messungen zur Puffergröße lässt sich ableiten, dass ein Puffer der Größe 4 in der Regel eine ausreichende Qualität erzielt. Daher kann auf einer

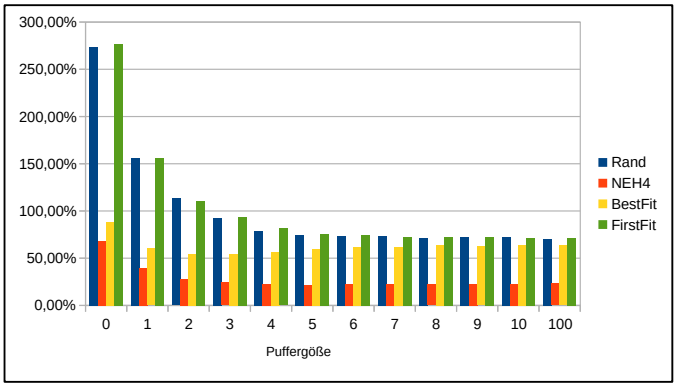

Abbildung 18: Qualität in Abhängigkeit der Puffergröße, Probleminstanzen 1-20

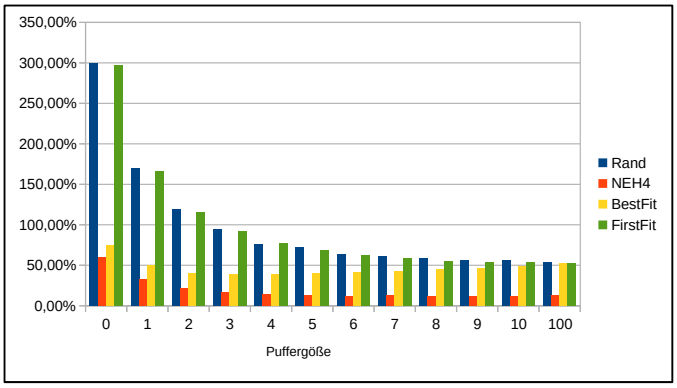

Abbildung 19: Qualität in Abhängigkeit der Puffergröße, Probleminstanzen 21-40

realen Anlage mit einem größeren Puffer mit einer geringeren Größe als der tatsächlichen geplant werden. Dann kann die verbleibende Pufferkapazität als Reserve genutzt werden. Damit ist die Anlage weniger anfällig gegenüber Abweichungen der Bearbeitungszeit auf den Maschinen.

5.2.8 Vergleich der Heuristiken

In diesem Abschnitt werden die verschiedenen Schedulingalgorithmen bezüglich ihrer Qualität und Laufzeit verglichen. Es wird eine Auswahl der Algorithmen betrachtet. Diese sind die SJL Prioritätsregel, die genetischen Algorithmen GA2 und NeighbourGA, die Best Fit Heuristik und die NEH Heuristik. Abbildung 20 zeigt die Qualität der Heuristiken in Abhängigkeit von der Anzahl der Aufträge für die Probleminstanzen 1-100.
Die NEH Heuristik erzielt jeweils die beste Qualität und bietet einen deut-

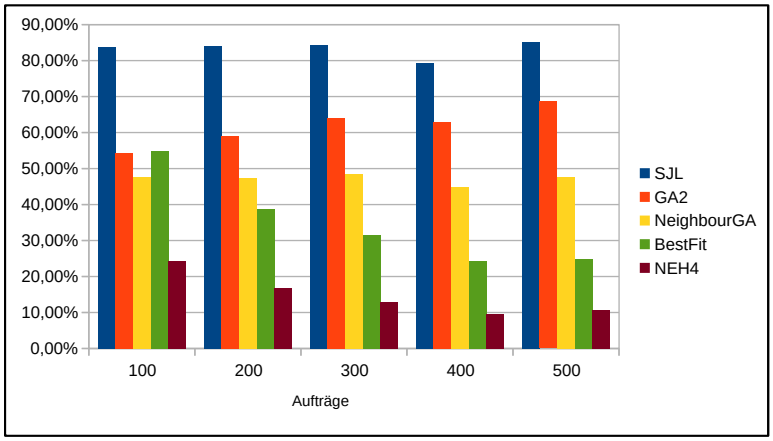

Abbildung 20: Qualität, Probleminstanzen 1-100

lichen Qualitätsgewinn gegenüber den anderen Schedulingverfahren. Diese ist um ein vielfaches besser als die der SJL Prioritätsregel. Auch gegenüber der Best Fit Heuristik kann eine signifikante Verbesserung festgestellt werden. Die Abbildung 21 zeigt die zugehörigen Laufzeiten der Heuristiken in Sekunden. Diese werden auf einer logarithmischen Skala aufgetragen. Es

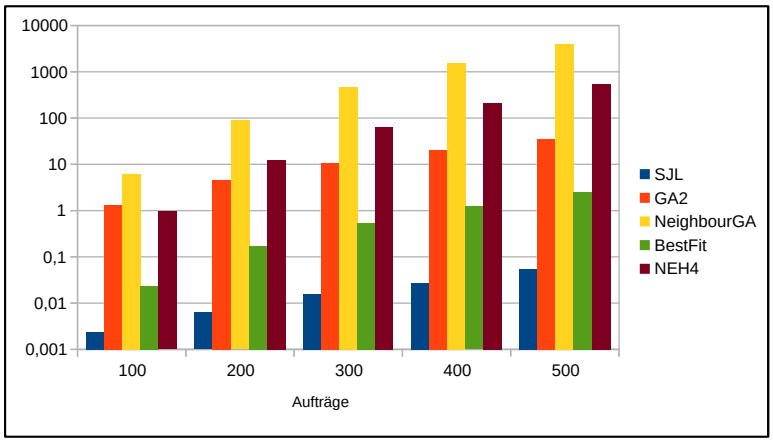

Abbildung 21: Laufzeit, Probleminstanzen 1-100

zeigt sich, dass die NEH Heuristik sowohl die beste Qualität, als auch die zweithöchste Laufzeit aufweist. Im Vergleich zum genetischen Algorithmus NeighbourGA ist die Laufzeit geringer. Dabei muss jedoch beachtet werden, dass es sich hierbei um eine parallelisierte NEH Heuristik handelt. Der Algorithmus NeighbourGA ist hingegen nicht parallelisiert. Die Laufzeit der Best Fit Heuristik ist um den Faktor 100 geringer, als die der NEH Heuristik. Damit zeigt sich, dass die Qualität der NEH Heuristik besser ist, als die der Best Fit Heuristik. Dieser Qualitätsvorteil wird jedoch durch eine sehr

viel höhere Laufzeit erkauft. Somit wäre der Laufzeitunterschied zwischen der Best Fit und der NEH Heuristik bei einer rein seriellen Ausführung des Programms noch um einiges höher. Andererseits kann die Laufzeit durch mehr Threads deutlich gesenkt werden. Die genetischen Algorithmen weisen jeweils eine schlechtere Qualität als die Best Fit Heuristik auf. Dabei erzielt die Best Fit Heuristik eine höhere Qualität bei einer geringeren Laufzeit. Da sich die Qualität der Best Fit Heuristik mit zunehmender Auftragsanzahl der der NEH Heuristik angleicht, kann für eine kleine Anzahl an Aufträgen die NEH Heuristik gewählt werden. Für eine große Anzahl an Aufträgen eignet sich die Best Fit Heuristik. Im Einzelfall muss dann entschieden werden, ob für eine Verbesserung der Qualität um höchstens 10 Prozent eine Zunahme der Laufzeit um den Faktor 100 (bei einer parallelen Berechnung mit vier Threads) in Kauf genommen wird.

6 Online Scheduling

Beim Online Scheduling handelt es sich um ein Echtzeit Scheduling, bei dem die Aufträge zu Beginn noch nicht feststehen. Jeder Job J_i hat dabei einen Freigabetermin r_i, der angibt, zu welchem Zeitpunkt der Job im System ankommt. Weiterhin wird jedem Job J_i eine Deadline d_i und gegebenenfalls eine Priorität w_i zugeordnet. Das Ziel besteht darin, die Verspätungen und damit die Summe der totalen gewichteten Verspätung $\sum w_i T_i$ zu minimieren.

$$(F_m|ltd(b); t_{jk}; r_i; d_i; w_i|\sum w_i T_i) \tag{25}$$

6.1 Heuristiken

In diesem Abschnitt werden einige Ansätze für das Online Scheduling vorgestellt. Zunächst wird darauf eingegangen, wie ein Belegungsplan für das Online Scheduling erstellt wird. Danach werden zwei auf Prioritätsregeln basierende Verfahren vorgestellt.

6.1.1 Schedulingverfahren

Für die Erstellung eines Schedules werden die Aufträge in drei Mengen unterteilt. In der Menge S befinden sich alle bereits gestarteten Jobs. In der Menge A befinden sich die aktiven Jobs. Dies sind die Aufträge, die bereits im System angekommen sind und die noch nicht gestartet wurden. Die dritte Menge U enthält die Jobs, die noch nicht im System angekommen sind.
In jedem Berechnungsschritt werden zuerst die Mengen A, S und U aktualisiert. Dabei werden Jobs, deren Freigabezeit kleiner gleich der Systemzeit ist, aus der Menge U in die Menge A verschoben. Danach werden für die aktiven Jobs anhand einer Heuristik Starttermine bestimmt. Sobald die Systemzeit größer gleich der Startzeit S_i eines aktiven Jobs J_i ist, wird dieser gestartet, aus der Menge A herausgenommen und in die Menge S eingefügt. Danach wird die Systemzeit inkrementiert. Dies wird so lange wiederholt, bis sich in den Mengen U und A keine Jobs mehr befinden. Der Wert, um den die Systemzeit inkrementiert wird, entspricht der Auflösung des Schedulingverfahrens.

6.1.2 Prioritätsscheduling

Das Online Scheduling kann auf der Basis von Prioritätsregeln durchgeführt werden. Tabelle 8 zeigt die hier verwendeten Prioritätsregeln. Dabei wird der aktive Job zuerst eingefügt, für den mit Hilfe der ausgewählten Prioritätsregel die höchste Priorität bestimmt wurde. Danach wird für die weiteren Jobs analog verfahren. Wenn nun im nächsten Berechnungsschritt des Algorithmus ein neuer aktiver Auftrag hinzukommt, können sich die

Regel	Beschreibung
FCFS	First Come First Serve, Auftrag mit geringster Freigabezeit zuerst
EDF	Earliest Deadline First, Auftrag mit geringster Deadline zuerst
EWDF	Earliest Weighted Deadline First, Auftrag mit geringster gewichteter Deadline $\frac{d_i}{w_i}$ zuerst
HPF	Highest Priority First, Auftrag mit höchster Priorität zuerst
SJL	Shortest Job Last, Auftrag mit höchster Bearbeitungszeit zuerst
EDF1	Auftrag mit geringster Differenz aus Deadline und Bearbeitungszeit $d_i - P_i$
EWDF1	Auftrag mit geringster gewichteter Differenz aus Deadline und Bearbeitungszeit $\frac{d_i - P_i}{w_i}$

Tabelle 8: Prioritätsregeln im Online Scheduling

bereits bestimmten Starttermine der Jobs verändern. Ein statisches Prioritätsscheduling kann dazu führen, dass ein Auftrag mit einer niedrigen Priorität niemals gestartet wird. Dies kann dann passieren, wenn in jedem Schritt neue Aufträge mit einer höheren Priorität hinzukommen. Um dies zu verhindern, kann ein dynamisches Prioritätsscheduling verwendet werden. Dabei kann die Priorität eines Jobs aus einer Kombination der Prioritätsregel und der Wartezeit bestimmt werden. Beispielsweise kann in jedem Berechnungsschritt die bereits vergangene Wartezeit des Auftrags zur Priorität addiert werden.

6.1.3 Best Fit Prioritätsscheduling

Bei dem Best Fit Prioritätsscheduling werden die Starttermine für die aktiven Jobs analog zur Best Fit Heuristik im Offline Scheduling bestimmt. Dabei kommen verschiedene Sortierungen der Jobs zum Einsatz. Einerseits können die aktiven Jobs aufsteigend anhand ihrer Deadlines d_i geordnet werden. Andererseits kann auch die Priorität w_i oder die Länge des Jobs P_i als Sortierkriterium herangezogen werden.
In jedem Berechnungsschritt wird ein Einfügezeitpunkt untersucht. Dabei wird jeweils der erste passende Job (Best Fit) aus der Liste der aktiven Jobs eingefügt. Die Sortierung der Liste ergibt sich durch Anwendung der gewählten Prioritätsregel (Tabelle 8). Wenn nun neue aktive Aufträge hinzukommen, kann sich der vorher bestimmte Startzeitpunkt S_i eines Jobs J_i verändern. Analog zum Prioritätsscheduling aus dem vorigen Abschnitt, kann es auch beim Best Fit Scheduling dazu kommen, dass ein niedrig prio-

risierter Auftrag niemals gestartet wird. Wenn beispielsweise immer neue passendere Aufträge ankommen, werden diese vor bereits vorhandene Aufträge eingefügt. Dies kann verhindert werden, indem einem Job eine maximale Wartezeit, die bis zum Bearbeitungsbeginn verstreichen darf, zugeordnet wird. Wenn diese Wartezeit verstrichen ist, wird der Job zum nächstmöglichen Zeitpunkt gestartet. Damit wird ausgeschlossen, dass ein Auftrag niemals gestartet wird.

6.2 Analyse der Heuristiken

In diesem Kapitel werden die vorgestellten Heuristiken für das Online Scheduling bezüglich ihrer Laufzeit und Qualität bewertet. Zunächst werden die Probleminstanzen, die zum Testen der Heuristiken verwendet werden, vorgestellt. Anschließend wird das einfache Prioritätsscheduling und das Best Fit Prioritätsscheduling bewertet. Abschließend wird das einfache Prioritätsscheduling mit dem Best Fit Prioritätsscheduling verglichen.

6.2.1 Generieren von Probleminstanzen

Zur Analyse der einzelnen Verfahren für das Onlinescheduling werden 100 verschiedene Probleminstanzen für einen Rossmann Online Flow Shop mit 20 Maschinen und einer Puffergröße von 3 erzeugt.

$$(F_{20}|ltd(3); t_{jk}; r_i; d_i; w_i|\sum w_i T_i) \tag{26}$$

Für jede Anzahl an Aufträgen werden jeweils 20 Probleminstanzen erzeugt. Für jeden Job werden die jeweiligen Parameter (Freigabetermin, Bearbeitungszeit, Deadline und Priorität) zufällig aus einem Intervall gewählt.

	Instanzen				
Parameter	1-20	21-40	41-60	61-80	81-100
n	100	200	300	400	500
m	20				
Tasks/Job	$[1-10]$				
p_{ij}	$[1000-4500]$				
r_i	$[0-2000]$				
w_i	$[1-10]$				
Deadlinefaktor α	$[5-10]$				
d_i	$P_i * \alpha + r_i + 1000$				

Tabelle 9: Probleminstanzen Online 1-100

Die Deadlines werden dabei so erzeugt, dass diese auf einer leeren Anlage eingehalten werden können. Dazu wird zunächst ein Deadlinefaktor bestimmt. Danach wird die Deadline als die Summe der Freigabezeit r_i und

der Bearbeitungszeit P_i, multipliziert mit dem Deadlinefaktor α, gebildet. Schließlich wird ein fester Wert von 1000 hinzuaddiert. Dieser entspricht ungefähr der Transportzeit eines Jobs durch die gesamte Anlage. Die totale ungewichtete Verspätung $\sum T_i$ und die totale gewichtete Verspätung $\sum w_i T_i$ der Schedulingverfahren wird jeweils über 20 Probleminstanzen mit einer festen Auftragsanzahl gemittelt. Diese Werte werden summiert und auf das beste erzielte Ergebnis des Verfahrens bezogen. Beide Verfahren werden mit einer Auflösung von 10 Zeitschritten getestet.

6.2.2 Prioritätsscheduling

Für das Prioritätsscheduling werden 7 Prioritätsregeln verglichen (Tabelle 8). Abbildung 22 zeigt die totale gewichtete Verspätung $\sum w_i T_i$ für die jeweiligen Regeln. Es zeigt sich, dass die Regeln HPF (höchste Priorität zuerst) und FCFS (Auftrag mit geringster Freigabezeit zuerst) jeweils das beste Ergebnis erzielen. Daher werden alle anderen Werte auf dieses Ergebnis bezogen. Die übrigen Prioritätsregeln erzielen im Vergleich zu diesen Regeln keine gute Qualität. Somit eignen sich diese Regeln (EDF, EWDF, SJL, EDF1 und EWDF1) nicht für das hier vorgestellte Online Prioritätsscheduling.

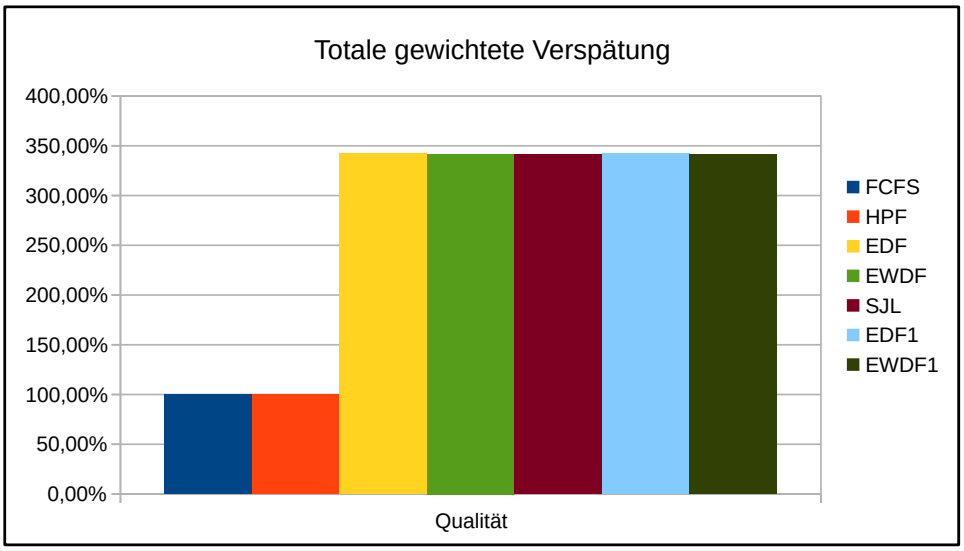

Abbildung 22: Prioritätsscheduling, gewichtete Verspätung

Für die totale ungewichtete Verspätung $\sum T_i$ ergibt sich das gleiche Bild. Daher wird hier auf ein weiteres Diagramm verzichtet. Somit eignen sich von den hier betrachteten Regeln einzig die FCFS und die HPF Regel für ein Prioritätsscheduling.

6.2.3 Best Fit Prioritätsscheduling

Das Best Fit Prioritätsscheduling wird mit 7 Prioritätsregeln durchgeführt (Tabelle 8). Die totale gewichtete Verspätung $\sum w_i T_i$ und die totale ungewichtete Verspätung $\sum T_i$ wird jeweils über 20 Probleminstanzen gemittelt und aufaddiert. Abbildung 23 zeigt die totale gewichtete Verspätung $sumTi$ für die einzelnen Prioritätsregeln.

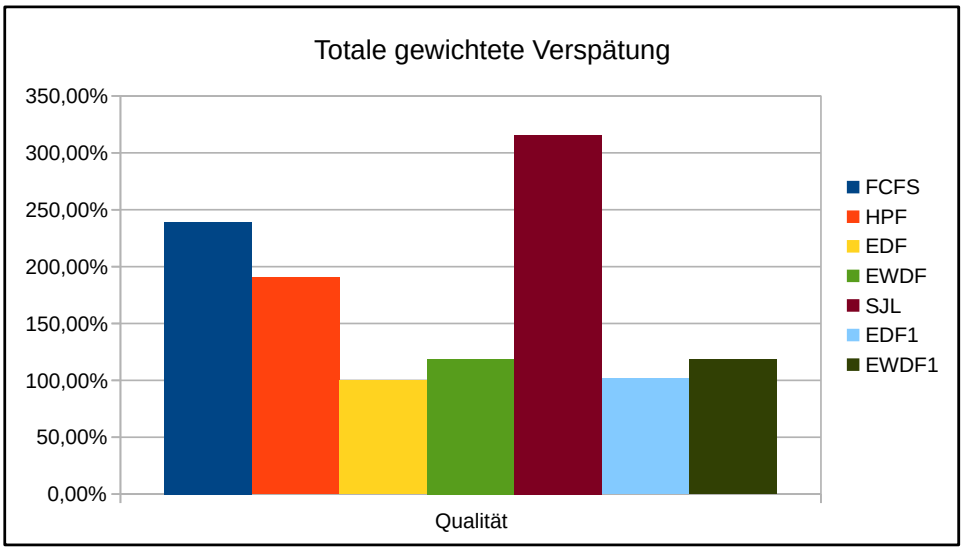

Abbildung 23: Best Fit Prioritätsscheduling, gewichtete Verspätung

Dabei zeigt sich, dass der beste Wert für die EDF Regel erzielt wird. Das Ergebnis der EDF1 Regel liegt um 1,4 Prozent über dem der EDF Regel. Die beiden Regeln EWDF und EWDF1 erzielen beide ein ähnliches Ergebnis. Dieses liegt 18 Prozent über dem der EDF Regel. Die anderen Prioritätsregeln sind deutlich schlechter. Dabei erhält man eine um 100 bis 200 Prozent höhere totale gewichtete Verspätung $\sum w_i T_i$. Abbildung 24 zeigt die totale ungewichtete Verspätung $\sum T_i$. Auch hier wird das beste Ergebnis mit der EDF Regel erzielt. Die EDF1 Regel liegt ungefähr gleich auf. Mit den übrigen Regeln fällt die totale ungewichtete Verspätung $\sum T_i$ deutlich höher aus. Diese übersteigt das Ergebnis der EDF Regel um 75 bis 216 Prozent.
Damit zeigt sich, dass das hier vorgestellte Best Fit Scheduling, in Kombination mit der EDF und der EDF1 Prioritätsregel, sowohl die beste totale gewichtete Verspätung $\sum w_i T_i$, als auch die beste totale ungewichtete Verspätung $\sum T_i$ erzielt.

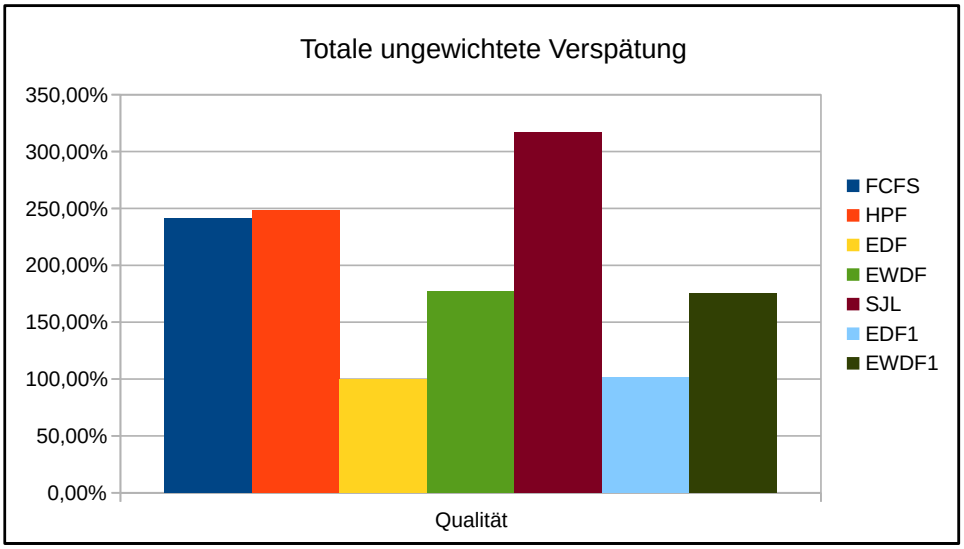

Abbildung 24: Best Fit Prioritätsscheduling, ungewichtete Verspätung

6.2.4 Vergleich der Heuristiken

In diesem Abschnitt werden die einzelnen Online Heuristiken bezüglich ihrer Qualität verglichen. Dabei wird nur eine Auswahl der Prioritätsregeln betrachtet.

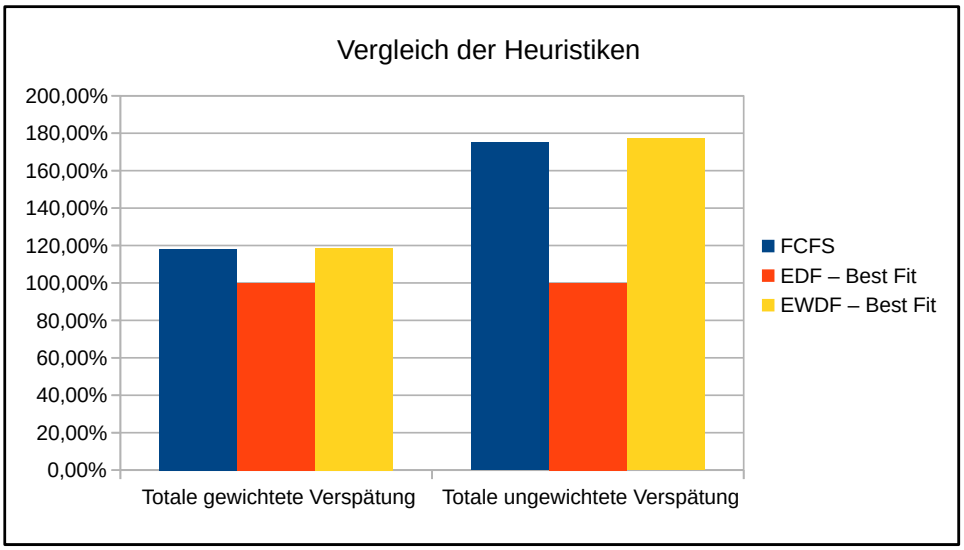

Abbildung 25: Online Scheduling, Vergleich

Die erste Prioritätsregel ist die FCFS Regel aus dem Prioritätsscheduling. Die HPF Regel wird nicht betrachtet, da diese die gleiche Qualität erzielt. Für das Best Fit Prioritätsscheduling werden die Regeln EDF und EWDF

betrachtet. Im Folgenden werden die Regeln, die mit dem Best Fit Prioritätsscheduling verwendet werden, mit dem Index Best Fit gekennzeichnet. Es zeigt sich, dass das $\text{EDF}_{BestFit}$ Scheduling die beste Qualität erreicht. Somit werden die anderen Werte auf dieses Ergebnis bezogen. Sowohl für die totale gewichtete Verspätung $\sum w_i T_i$ als auch für die totale ungewichtete Verspätung $\sum T_i$ liegen die FCFS und die $\text{EWDF}_{BestFit}$ Heuristik ungefähr gleich auf. Für die totale gewichtete Verspätung $\sum w_i T_i$ liegen beide Heuristiken 18 Prozent über dem Ergebnis der $\text{EDF}_{BestFit}$ Heuristik. Betrachtet man hingegen die totale ungewichtete Verspätung $\sum T_i$ fällt dieser Abstand mit 75 Prozent deutlich höher aus. Somit fällt die Verspätung für Jobs, die mit dem FCFS oder dem $\text{EWDF}_{BestFit}$ Scheduling geplant werden, im Mittel um 75 Prozent höher aus, als für Jobs, die mit der $\text{EDF}_{BestFit}$ Heuristik geplant werden. Damit zeigt sich, dass die Kombination des Best Fit Schedulings mit einer geeigneten Prioritätsregel dem klassischen Prioritätsscheduling qualitativ überlegen ist.

7 Zusammenfassung

In dieser Bachelorarbeit wird untersucht, wie sich das Schedulingproblem der Behälterförderanlage bei Rossmann modellieren lässt. Für das Offline Scheduling zeigt sich, dass das Problem als generalisierter Flow Shop mit fester Puffergröße und Transportzeiten zwischen den Maschinen dargestellt werden kann. Beim Online Scheduling kommen noch Bearbeitungsfristen, Freigabezeiten und Prioritäten hinzu. Ansonsten unterscheidet sich das Offline und das Online Scheduling nur durch das zu minimierende Zielkriterium. Beim Offline Scheduling ist dies die Gesamtbearbeitungszeit C_{max} und beim Online Scheduling die totale gewichtete Verspätung $\sum w_i T_i$. Weiterhin wird untersucht, welche Verfahren für das jeweilige Scheduling in der Literatur verwendet werden. Anschließend wird auf die Implementierung und die Datenstrukturen für einen Maschinenbelegungsplan eingegangen. Dabei hat sich gezeigt, dass die Laufzeit für die Erstellung eines Schedules durch eine geeignete Wahl dieser Datenstrukturen signifikant verbessert werden kann. In den folgenden Abschnitten wird untersucht, inwiefern sich bereits bekannte Heuristiken und Algorithmen auf das Offline Scheduling übertragen lassen. Dabei wird zunächst eine untere Schranke S_{min} und, darauf basierend, eine Qualitätsmaßzahl Q eingeführt. Diese wird verwendet, um die einzelnen Offline Scheduling Heuristiken zu bewerten. Für genetische Algorithmen zeigt sich, dass eine Verbesserung der Qualität gegenüber einer zufälligen Reihenfolge erzielt wird. Diese fällt jedoch nur gering aus. Die einzelnen genetischen Algorithmen konvergieren dabei nur sehr langsam und finden nur lokale Minima. Der beste genetische Algorithmus ist die Nachbarschaftssuche. Dieser Algorithmus eignet sich auf Grund der sehr hohen Laufzeit jedoch nicht für das Offline Scheduling. Die neu eingeführte First Fit Heuristik erzielt keinen Qualitätsgewinn gegenüber der zufälligen Reihenfolge und eignet sich daher auch nicht für das Offline Scheduling. Die hier vorgestellte Best Fit Heuristik bietet einen guten Kompromiss aus Laufzeit und Qualität. Die beste Qualität wird mit der NEH Heuristik erzielt. Dieser Qualitätsvorteil wird jedoch mit einer sehr hohen Laufzeit erkauft. Diese ist um den Faktor 100 größer, als die der zweitbesten Best Fit Heuristik. Durch Parallelisierung der Heuristik wurde die Laufzeit stark verbessert. Daher zeigt sich für das Offline Scheduling, dass die aus der Literatur bekannte NEH Heuristik über alle Probleminstanzen hinweg die beste Qualität aufweist. Die Qualität der Best Fit Heuristik und der NEH Heuristik steigt mit wachsender Auftragszahl. Daher kann für Probleminstanzen mit wenigen Aufträgen die NEH Heuristik gewählt werden. Für große Probleminstanzen wirkt sich der Laufzeitunterschied deutlich stärker aus. Damit kann die Best Fit Heuristik für diese Probleminstanzen gewählt werden.
Abschließend wird das Online Scheduling untersucht. Für das Online Scheduling existieren nur sehr wenige Ansätze in der Literatur, die sich meist nur auf kleine oder sehr spezielle Flow Shops beziehen. Daher werden zwei On-

line Scheduling Heuristiken auf Basis von Prioritätsregeln verglichen. Dabei zeigt sich, dass die Kombination der EDF Prioritätsregel mit der Best Fit Heuristik das beste Ergebnis erzielt. Das Best Fit Prioritätsscheduling erzielt gegenüber dem einfachen Prioritätsscheduling sowohl für die totale gewichtete Verspätung $\sum w_i T_i$, als auch für die totale ungewichtete Verspätung $\sum T_i$ ein signifikant besseres Ergebnis. Damit zeigt sich, dass das Best Fit Prioritätsscheduling dem klassischen Prioritätsscheduling überlegen ist.
Somit ergibt sich, dass das hier vorgestellte Best Fit Scheduling sowohl für das Offline als auch für das Online Scheduling geeignet ist.

8 Ausblick

Betrachtet man abschließend die Vor- und Nachteile der hier vorgestellten Heuristiken für das Offline Scheduling, wird ersichtlich, dass bisher noch kein Verfahren existiert, welches sowohl eine geringe Laufzeit als auch eine sehr hohe Qualität aufweist. Die Best Fit Heuristik erzielt bei einer sehr geringen Laufzeit eine mittlere Qualität. Die NEH Heuristik erreicht eine sehr hohe Qualität bei einer hohen Laufzeit. Damit könnte ein Hybridverfahren aus der Best Fit und der NEH Heuristik gebildet werden, um die Vorteile beider Heuristiken zu vereinen. Dieses Verfahren kann dann beispielsweise mit zwei Parametern gesteuert werden. Dies kann zum einen eine vorgegebene Qualität und zum anderen eine vorgegebene Laufzeit sein. Zuerst könnte dann eine optimierte Reihenfolge mit der Best Fit Heuristik erstellt werden. Wenn diese bereits die geforderte Qualität erreicht, terminiert das Verfahren. Ansonsten kann die NEH Heuristik verwendet werden, um eine bessere Qualität zu erreichen. Sobald die NEH Heuristik das vorgegebene Zeitlimit überschreitet, kann das Verfahren abgebrochen werden. Dann wird die beste bisher berechnete Reihenfolge ausgegeben. Weiterhin könnten auch Verfahren untersucht werden, die auf Basis der optimierten Best Fit Reihenfolge weitere Optimierungen durchführen.
Betrachtet man abschließend das Online Scheduling, wird ersichtlich, dass dieses bisher noch wenig untersucht wurde. Das vorgestelle Best Fit Prioritätsscheduling ist dem einfachen Prioritätsscheduling überlegen. Aufgrund fehlender Vergleichsmöglichkeiten kann jedoch keine allgemeingültige Aussage über dessen Qualität getroffen werden. Daher sollte das Best Fit Prioritätsscheduling zukünftig noch mit weiteren Verfahren verglichen werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] E. Ignall and L. Schrage. Application of the branch and bound technique to some flowshop scheduling problems. *Operations Research*, 13:400–412, 1965.

[2] 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. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[3] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Science & Business Media, Berlin, 4th edition, 2012.

[4] Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11:91–95, 1982.

[5] Shi Qiang Liu and Erhan Kozan. Scheduling a flow shop with combined buffer conditions. *International Journal of Production Economics*, 117:371–380, 2009.

[6] A. Gharbi and A. Mahjoubi. New lower bounds for flow shop scheduling. *International Journal of Humanities and Management Sciences IJHMS*, 1:28–32, 2013.

[7] Deepak Gupta, Payal Singla, and Shashi Bala. N x 2 flow shop scheduling model using branch and bound technique, set up times are separated from processing times, with job block criterion and interval of non availability of machines. *International Journal of Engineering and Innovative Technology IJEIT*, 3:529–532, 2013.

[8] Wei Weng and Shigeru Fujimura. Online scheduling of flexible flow shop manufacturing. In *International Joint Conference on Computational Sciences and Optimization*, pages 112–116, 2009.

[9] Kochiro TAKAKU and Kenji YURA. Online scheduling aiming to satisfy due date for flexible flow shops. *JSME International Journal, Series C*, 48:21–25, 2005.

[10] Peihai Liu and Xiwen Lu. A best possible on-line algorithm for two-machine flow shop scheduling to minimize makespan. *Computers & Operations Research*, 51:251–256, 2014.

[11] R. Haupt. A survey of priority rule-based scheduling. *OR Spektrum*, 11:3–16, 1989.

[12] Peter Brucker. *Scheduling Algorithms*. Berlin, 5th edition, 2007.

[13] Peter Brucker, Bernd Jurisch, and Bernd Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

[14] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, 1st edition, 1998.

Erklärung der Urheberschaft

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum	Unterschrift