Robuste Lokale Suche für das Job Shop Scheduling Problem

von
Friederike Menge

Masterarbeit in Mathematik

vorgelegt der
Fakultät für Mathematik, Informatik und Naturwissenschaften der
Rheinisch-Westfälischen Technischen Hochschule Aachen

im März 2014

angefertigt am Lehrstuhl für Operations Research and Supply Chain Management
Prof. Dr. Marco E. Lübbecke

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Immer häufiger wird durch die Medien auf die Schließung ländlicher Arztpraxen und den Mangel an Nachfolgern aufmerksam gemacht. Als Folge dieser medizinischen Unterversorgung in ländlichen Gebieten steigt die Zahl der zu behandelnden Patienten in städtischen Krankenhäusern. Dieser Trend wird weiter zunehmen und wirkt sich negativ auf den Krankenhausalltag in Form von überfüllten Notaufnahmen und langen Wartezeiten aus. Sind die Notaufnahmen überfüllt, so bedeutet dies eine Überlastung des Personals und der medizinischen Geräte. Dies kann zu einer schlechteren Behandlungsqualität und einem steigenden Fehlerrisiko führen. Dieser bedenkliche Zustand muss sich ändern. Mit unserer Arbeit möchten wir uns diesem Problem widmen.

Das Ziel dieser Arbeit ist es, einen Algorithmus zu entwickeln, der Behandlungen einer Menge gegebener Patienten so Behandlungszeitpunkte auf Ressourcen zuweist, dass Behandlungen von Notfallpatienten zu einem gewissen Maß spontan in den ermittelten Zeitplan eingefügt werden können. Dabei darf die Aufenthaltsdauer der geplanten Patienten nicht unberücksichtigt bleiben. Das Problem Patienten in optimaler Weise Behandlungszeitpunkte zuzuweisen, so dass sie eine möglichst geringe Aufenthaltsdauer im Krankenhaus besitzen, bezeichnen wir als Krankenhaus Scheduling Problem.

Beim Krankenhaus Scheduling Problem ist eine Menge von Patienten und eine Menge von Ressourcen gegeben. Jeder Patient besitzt eine Menge von Behandlungen, die er im Krankenhaus erhalten soll. Zwischen verschiedenen Behandlungen eines Patienten kann es Rangfolgebeziehungen geben, das heißt, dass eine Behandlung vor einer anderen Behandlung durchgeführt werden muss. Eine Behandlung muss während einer ununterbrochenen Zeitdauer auf einer vorgegebenen Ressource stattfinden. Zwei Behandlungen eines Patienten dürfen sich nicht überschneiden. Jeder Patient besitzt einen Ankunftszeitpunkt, an welchem er das Krankenhaus betritt und nach dem er für seine Behandlungen zur Verfügung steht. Für jede Ressource gibt es feste Öffnungszeiten. Nur innerhalb der Öffnungszeiten können Behandlung auf einer Ressource stattfinden. Auch kann auf einer Ressource immer nur eine Behandlung zeitgleich stattfinden. Ziel des Krankenhaus Scheduling Problems ist es, die Summe der Wartezeiten aller Patienten zu minimieren.

Das Krankenhaus Scheduling Problem ist mit dem viel untersuchten Job Shop Scheduling Problem eng verwandt. Das Job Shop Scheduling Problem gehört zu einem der schwersten kombinatorischen Optimierungsprobleme. Es ist nicht nur NP-hard, sondern gehört sogar zu den schwieriger zu lösenden Problemen dieser Klasse [55]. Daher ist es nicht einfach, das Job Shop Scheduling Problem optimal zu lösen. In den letzten Jahrzehnten wurden einige Lokale Such Algorithmen entwickelt, die das Problem sehr effizient lösen können [54]. Daher möchten wir in dieser Arbeit einen Algorithmus für das Krankenhaus Scheduling Problem entwickeln, der auf einer Lokalen Suche beruht.

Der von uns entwickelte Algorithmus soll nicht einfach eine möglichst gute Lösung für das Krankenhaus Scheduling Problem ermitteln, sondern in der Lösung berücksichtigen, dass Notfallpatienten auftreten können. Wir möchten einen Zeitplan für die Behandlungen, einen sogenannten Schedule, generieren, in dem zu einem gewissen Maß Behandlungen von

Notfallpatienten integriert werden können. Je besser ein Schedule gegen das Auftreten von Notfallpatienten abgesichert ist, das heißt, je besser Notfallpatienten, gemessen an einem vorgegebenem Maß, in den Schedule eingeplant werden können, desto robuster nennen wir einen Schedule.
In unserer Arbeit entwickeln wir einen Lokalen Such Algorithmus für das Krankenhaus Scheduling Problem und passen diesen so an, dass im Algorithmus das Auftreten von Notfallpatienten berücksichtigt wird und so der Algorithmus genutzt werden kann, um einen zu einem gewissen Maß robusten Schedule zu erzeugen. Daher nennen wir unser Verfahren Robuste Lokale Suche. Eine Robuste Lokale Suche kann im Allgemeinen als ein Verfahren beschrieben werden, dass eine Lösung zu einem kombinatorischen Optimierungsproblem nicht nur anhand seiner Kostenfunktion bewertet, sondern auch in Betracht zieht, wie die Lösung auf Unsicherheiten reagiert.

In dieser Richtung wurde ein Ansatz für das Job Shop Scheduling Problem von Akker, Blokland und Han Hoogeveen entwickelt. In ihrem Paper [4] nutzen sie eine Lokale Suche, um robuste Lösungen in Bezug auf unsichere Prozesszeiten zu finden. Dabei wird die Robustheit einer Lösung anhand des Erwartungswertes gemessen, den eine Lösung in Bezug auf gegebene Wahrscheinlichkeitsverteilungen für die unsicheren Prozesszeiten besitzt. Auf weitere Ansätze für Scheduling Probleme unter Unsicherheiten wird in Abschnitt 2.3 dieser Arbeit näher eingegangen.

Der Aufbau dieser Arbeit gliedert sich folgendermaßen. Im ersten Kapitel erfolgt die Definition des Krankenhaus Scheduling Problems. Es wird dessen Zusammenhang zum bekannten und viel untersuchten Job Shop Scheduling Problem erläutert. Im Anschluss wird die Definition des Krankenhaus Scheduling Problems unter Unsicherheiten gegeben. In Kapitel zwei beschreiben wir einige bestehende Ansätze zur Optimierung unter Unsicherheiten und gehen im Speziellen auf die Optimierung unter Unsicherheiten von Scheduling Problemen ein. Im dritten Kapitel erläutern wir die allgemeine Idee hinter einer Robusten Lokalen Suche und stellen zwei mögliche Ansätze vor. Anschließend beschreiben wir im vierten Kapitel, wie eine Robuste Lokale Suche für das Krankenhaus Scheduling Problem unter Unsicherheiten umgesetzt werden kann. Zur Messung des Einplanens von Notfallpatienten stellen wir zwei verschiedene Maße vor. Tests und Ergebnisse zu den vorgestellten Ansätzen in Kombination mit den beiden Maßen werden in Kapitel fünf dargestellt. Das daran anschließende Kapitel sechs fasst die ermittelten Resultate zusammen und gibt einen Ausblick, wie die entwickelten Ansätze erweitert werden können.

Kapitel 1

Definition des Problems

1.1 Das Krankenhaus Scheduling Problem

Beim Krankenhaus Scheduling Problem wird die Situation betrachtet, dass Patienten zu im Vorfeld bekannten Zeitpunkten im Krankenhaus eintreffen, um eine vorgegebene Menge von Behandlungen zu durchlaufen. Ziel ist es, die Behandlungstermine der Patienten so festzulegen, dass die Aufenthaltsdauer der einzelnen Patienten im Krankenhaus möglichst gering ist.
Wie bereits erwähnt, gibt es für jeden Patienten einen Ankunftszeitpunkt, an dem er das Krankenhaus betritt und vor dem er zu keiner Behandlung zur Verfügung steht. Allen Behandlungen eines Patienten können somit nur Termine nach seinem Ankunftsdatum zugewiesen werden. Bei der Terminvergabe für die Behandlungen muss darüber hinaus darauf geachtet werden, dass sich verschiedene Behandlungen eines Patienten nicht überschneiden, weil ein Patient nicht zeitgleich bei mehreren Behandlungen anwesend sein kann. Zwischen verschiedenen Behandlungen eines Patienten kann es Rangfolgebeziehungen geben. Eine Rangfolgebeziehung zwischen zwei Behandlungen besteht dann, wenn festgelegt ist, welche der Behandlungen zuerst stattfinden soll. Erst wenn diese Behandlung abgeschlossen wurde, darf die andere Behandlung stattfinden. So müssen beispielsweise erst alle Voruntersuchungen für eine Operation eines Patienten durchgeführt worden sein, bevor er operiert werden kann. Des Weiteren muss eine Behandlung auf einer geeigneten Ressource geplant werden, das heißt einer Ressource, die den für die Behandlung erforderlichen Geräten entspricht. Eine Operation kann nur in einem Operationssaal stattfinden, in einem Raum kann nur dann geröntgt werden, wenn sich in dem Raum ein Röntgengerät befindet und so weiter. Zur Vereinfachung nehmen wir an, dass es für jeden Behandlungstypen wie zum Beispiel Röntgen nur ein zur Verfügung stehendes Gerät, das heißt, nur eine zur Verfügung stehende Ressource gibt. Für jede Ressource gibt es feste Öffnungszeiten. Nur innerhalb der Öffnungszeiten können Behandlung auf einer Ressource stattfinden. Darüber hinaus kann auf einer Ressource zu jedem Zeitpunkt nur eine Behandlung stattfinden.
Die Aufenthaltsdauer eines Patienten zu minimieren ist äquivalent damit die Wartezeit eines Patienten zu minimieren, wobei die Wartezeit der Zeit zwischen Ankunftsdatum und Ende der letzten Behandlung abzüglich der Behandlungsdauern entspricht. Als Ziel des Krankenhaus Scheduling Problems kann also auch die Minimierung der Summe der Wartezeiten aller Patienten betrachtet werden.

1.1.1 Definition

Gegeben sei eine Menge $P = \{p_1, \ldots, p_n\}$ von Patienten, die im Krankenhaus behandelt werden sollen. Das Krankenhaus hat dazu $R = \{r_1, \ldots, r_m\}$ Ressourcen zur Verfügung, die während ihrer jeweiligen Öffnungszeiten zur Behandlung der Patienten genutzt werden

können. Für eine Ressource r_k sind die Öffnungszeiten, abhängig vom Wochentag w, durch einen Öffnungszeitpunkt $r_k^{open}(w)$ und einen Schlusszeitpunkt $r_k^{close}(w)$ festgelegt. Ein Patient p_i soll im Rahmen seines Krankenhausaufenthaltes q_i Behandlungen $B_i = \{b_{i1}, \ldots, b_{iq_i}\}$ durchlaufen. Für ein geordnetes Paar von Behandlungen (b_{ij},b_{ik}) kann es eine *Rangfolgebeziehung* geben, das heißt, dass Behandlung b_{ij} vor Behandlung b_{ik} stattfinden soll. Sei die Menge der Rangfolgebeziehungen für Behandlungen von Patient p_i gegeben durch Θ_i. Die Rangfolgebeziehungen zwischen den Behandlungen eines Patienten p_i lassen sich mittels eines Digraphen $D(p_i)$ darstellen. Dabei wird der Digraph wie folgt konstruiert. Für jede Behandlung besitzt der Graph $D(p_i) = (V, E)$ einen Knoten. Es gilt

$$v_{ij} \in V \Leftrightarrow b_{ij} \in B_i. \tag{1.1.1}$$

Und für jede Rangfolgebeziehung (b_{ij},b_{ik}) besitzt der Graph einen Bogen $v_{ij}v_{ik}$. Es gilt

$$v_{ij}v_{ik} \in E \Leftrightarrow (b_{ij}, b_{ik}) \in \Theta_i. \tag{1.1.2}$$

Ein Beispiel für einen solchen Digraphen, den wir im Folgenden als *Behandlungsgraphen* bezeichnen werden, ist in Abbildung 1.1 dargestellt.

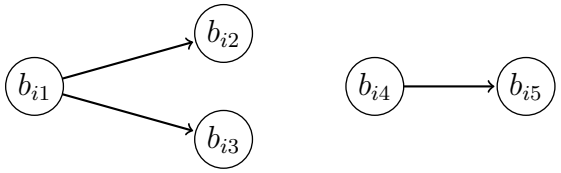


Abbildung 1.1: Beispiel für einen Behandlungsgraphen $D(p_i)$ von Patient p_i.

Gibt es eine Rangfolgebeziehung (b_{ij}, b_{ik}) zwischen zwei Behandlungen, so bezeichnen wir b_{ij} als *Vorgänger-Behandlung* von b_{ik}. Eine Behandlung kann erst dann stattfinden, wenn all seine Vorgänger-Behandlungen abgeschlossen wurden.
Jede Behandlung b_{ij} benötigt eine vorgeschriebene Zeit t_{ij} und kann nur auf einer fest vorgegebenen Ressource $R(b_{ij}) \in R$ durchgeführt werden. Der frühestmögliche Termin, an dem ein Patient p_i für Untersuchungen zur Verfügung steht, ist durch den Termin Arr_i gegeben und wird als *Ankunftszeit* bezeichnet.
Beim Zuweisen von Terminen zu den Behandlungen muss Folgendes beachtet werden:

(I) Eine Behandlung kann nicht unterbrochen werden.

(II) Eine Behandlung muss vollständig in den Öffnungszeiten der Ressource liegen.

(III) Auf einer Ressource kann zu jedem Zeitpunkt nur eine Behandlung stattfinden.

(IV) Ein Patient kann zu jedem Zeitpunkt nur eine Behandlung bekommen.

Wird jeder Untersuchung $b \in B = \bigcup_{i=1}^{n} B_i$ ein Starttermin $S_t(b)$ zugeordnet, so wird eine solche Zuordnung $S = (S_t)$ als *Schedule* bezeichnet. Ein Schedule ist *zulässig*, wenn er die Bedingungen (I)-(IV) erfüllt.
Den Zeitpunkt, an dem die letzte Behandlung von Patient p_i abgeschlossen ist, bezeichnen wir als *Entlassungszeitpunkt* C_i mit

$$C_i = C(p_i) = \max_{l \in \{1,\ldots,q_i\}} S_t(b_{il}) + t_{il} \tag{1.1.3}$$

von Patient p_i.
Das *Krankenhaus Scheduling Problem* besteht darin, einen zulässigen Schedule zu finden, der die Summe der Aufenthaltszeiten aller Patienten im Krankenhaus

$$\min \sum_{i=1}^{n} (C_i - Arr_i) \tag{1.1.4}$$

minimiert.

Um das Krankenhaus Scheduling Problem zu veranschaulichen und die eingeführten Begriffe zu verdeutlichen, betrachten wir das folgende Beispiel.

Beispiel 1.1.1. *Gegeben seien fünf Patienten $p_1, \ldots, p_5$ und drei Ressourcen r_1, r_2 und r_3. Die Öffnungszeiten für jede Ressource sind täglich von 8:00-14:00 Uhr. Die Ankunftszeiten der Patienten sowie deren Behandlungen mit zugehörigen Behandlungsdauern und Ressourcen sind in Tabelle 1.1 dargestellt.*

Patient	Ankunftszeit	Behandlungen	Ressource	Behandlungsdauer
$\mathbf{p_1}$	01-mm-yyyy 10:00	b_{11}	r_1	2
		b_{12}	r_3	2
		b_{13}	r_2	1
$\mathbf{p_2}$	01-mm-yyyy 8:00	b_{21}	r_3	3
		b_{22}	r_2	3
$\mathbf{p_3}$	01-mm-yyyy 12:00	b_{31}	r_1	1
		b_{32}	r_2	3
		b_{33}	r_3	2
$\mathbf{p_4}$	01-mm-yyyy 8:00	b_{41}	r_2	2
		b_{42}	r_3	2
		b_{43}	r_1	2
$\mathbf{p_5}$	02-mm-yyyy 10:00	b_{51}	r_1	1
		b_{52}	r_2	1
		b_{53}	r_3	2

Tabelle 1.1: Krankenhaus Scheduling Instanz.

Die Rangfolgebeziehungen zwischen Behandlungen eines Patienten sind durch die Behandlungsgraphen der Patienten in Abbildung 1.2 dargestellt.

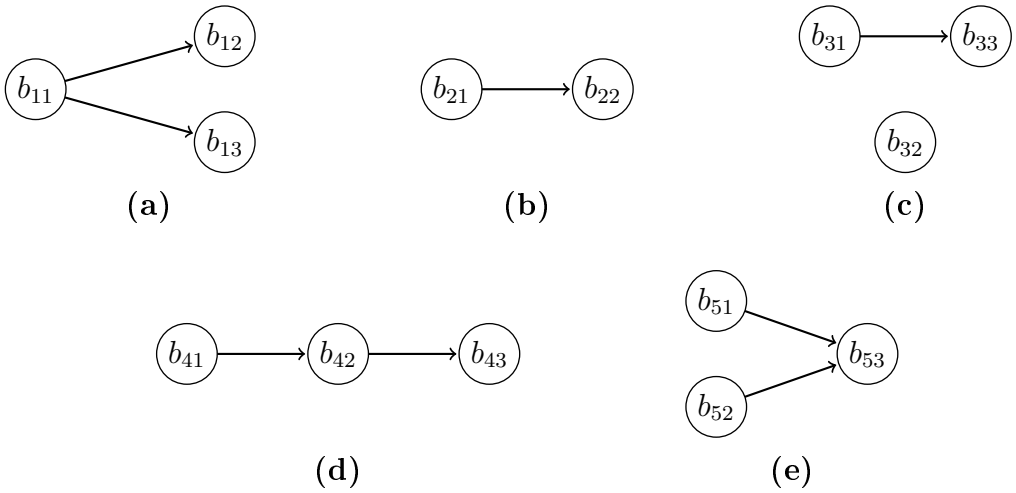


Abbildung 1.2: Behandlungsgraphen von Patienten $p_1, \ldots, p_5$.

Ein zulässiger Schedule ist durch Startzeiten für die Behandlungen gegeben, so dass die Bedingungen (I)-(IV) erfüllt sind.
Die Darstellung einer Lösung in Abbildung 1.3 zeigt anschaulich und leicht nachvollziehbar, dass die Bedingungen (I)-(IV) erfüllt sind.

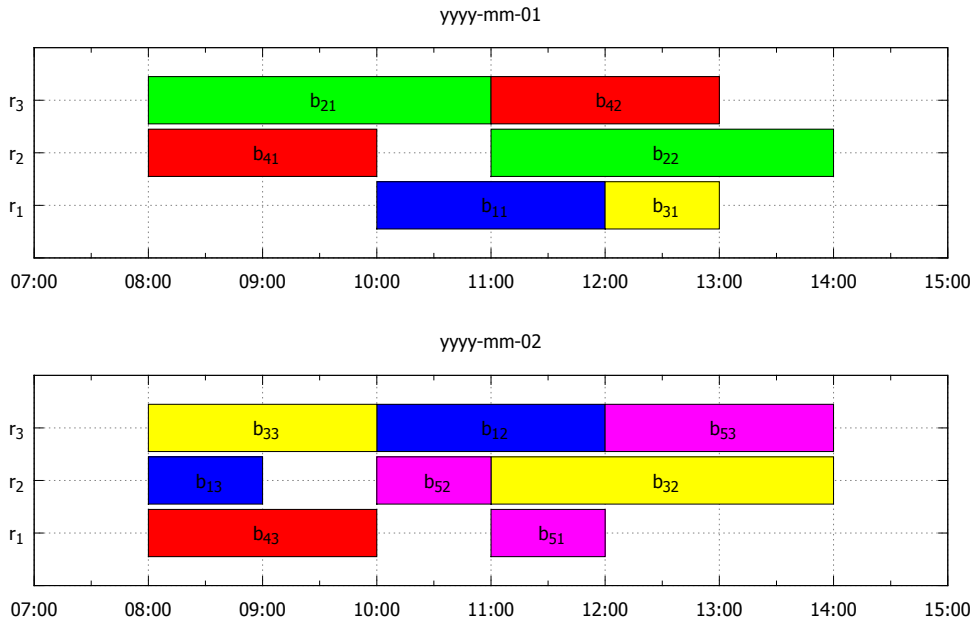


Abbildung 1.3: Lösungsrepräsentation mittels Gantt Chart.

Die Abbildung 1.3 verbildlicht eine Lösung in Form eines Gantt Charts. Das Gantt Chart wurde für Projekt Scheduling Probleme eingeführt. Mittels waagerechter Balken werden die Projekte entsprechend ihrer Dauer gegenüber der Zeit aufgetragen. Das Gantt Chart war zunächst die beliebteste Form, um Lösungen für das Job Shop Scheduling Problem darzustellen [33] und wird in Gantt (1919)[26], Clark (1922)[22] und Porter (1968) [45] beschrieben. Das Job Shop Scheduling Problem kann als Vereinfachung des Krankenhaus Scheduling Problems betrachtet werden. Während in der Literatur das Krankenhaus Scheduling Problem wenig Beachtung findet, ist das Job Shop Scheduling Problem ein viel

untersuchtes Problem. Da einige Ideen in dieser Arbeit vom Job Shop Scheduling Problem herrühren, wird das Job Shop Scheduling Problem im nachfolgenden Abschnitt vorgestellt und die Beziehung zum Krankenhaus Scheduling Problem erläutert.

1.1.2 Das Job Shop Scheduling Problem und seine Beziehung zum Krankenhaus Scheduling Problem

Das Job Shop Scheduling Problem kann als eine Vereinfachung des Krankenhaus Scheduling Problems gesehen werden. Es gibt demgemäß einen starken Zusammenhang zwischen den beiden Problemen. Im nächsten Abschnitt soll zunächst das Job Shop Scheduling Problem beschrieben werden, bevor im Anschluss die Unterschiede und Gemeinsamkeiten der beiden Probleme erörtert werden.

Das Job Shop Scheduling Problem

Das Job Shop Scheduling Problem entstammt der Produktionsplanung und ist von seinem ersten Auftreten Anfang der Fünfzigerjahre bis heute ein viel untersuchtes Problem. Es ist nicht bekannt, wer das Job Shop Scheduling Problem als erstes formulierte, aber es ist weitgehend akzeptiert, dass das Buch '*Industrial Scheduling*', geschrieben von *Muth* und *Thompson*, in dem alle wichtigen Ergebnisse der Fünfzigerjahre zusammengetragen sind, grundlegend für die darauffolgenden Untersuchungen war [33].

Das Job Shop Scheduling Problem richtet sich an die Situation, dass eine gegebene Menge von Aufträgen, auch Jobs genannt, auf einer Menge von Maschinen bearbeitet werden soll. Jeder Job besteht aus einer Menge von Operationen, die in einer festen Reihenfolge bearbeitet werden müssen. Dabei dürfen sich Operationen desselben Jobs nicht überschneiden und müssen während einer ununterbrochenen Zeitdauer stattfinden. Auf einer Maschine kann zeitgleich immer nur eine Operation bearbeitet werden. Ziel des Job Shop Scheduling Problems ist es, den Operationen so Anfangszeitpunkte zuzuweisen, dass der späteste Bearbeitungszeitpunkt aller Operation minimiert wird.

Problembeschreibung Gegeben ist eine Menge von Jobs $J = \{j_1, \ldots, j_n\}$, die auf einer Menge von Maschinen $M = \{m_1, \ldots, m_m\}$ bearbeitet werden sollen. Ein Job j_i besteht aus einer Folge von Operationen $O_i = (o_{i1}, \ldots, o_{iq_i})$. Die Folge der Operationen gibt eine eindeutige Bearbeitungsreihenfolge der Operationen vor. Eine Operation o_{il} kann nur dann bearbeitet werden, wenn die in der Folge vorangegangene Operation $o_{i,l-1}$ bereits abgeschlossen wurde. Eine Operation muss auf einer eindeutig vorgeschriebenen Maschine $\mathfrak{M}(o_{il}) \in M$ bearbeitet werden. Zur Bearbeitung einer Operation o_{il} wird die Zeit t_{il} benötigt.
Beim Zuweisen von Startzeiten zu Operationen muss beachtet werden, dass

(I) eine Operation während ihrer Bearbeitung nicht unterbrochen werden kann,

(II) die Operationen eines Jobs in der gegebenen Reihenfolge bearbeitet werden müssen

(III) und eine Maschine zu jedem Zeitpunkt nur eine Operation durchführen kann.

Wird jeder Operation $o \in O = \bigcup_{i=1}^{n} O_i$ eine Startzeit $S_t(o)$ zugeordnet, so wird diese Zuordnung $(S_t)_{o \in O}$ als *Schedule* bezeichnet. Werden die Bedingungen (I)-(III) durch den Schedule eingehalten, so bezeichnet man den Schedule als *zulässig*.

Das *Job Shop Scheduling Problem* besteht darin, einen zulässigen Schedule zu finden, der den *Makespan*, den letzten Bearbeitungszeitpunkt aller Jobs,

$$\min_{i\in\{1,\dots,n\}} (S_t(o_{iq_i}) + t_{iq_i}) \tag{1.1.5}$$

minimiert.

Das Job Shop Scheduling Problem kann in ein graphentheoretisches Problem umgewandelt werden. Die Darstellung des Job Shop Scheduling Problems als graphentheoretisches Problem wird als Disjunctive Graph Modell bezeichnet. Da wir eine Darstellung, die aus diesem Modell hervorgeht, für den Algorithmus der Robusten Lokalen Suche für das Krankenhaus Scheduling Problem verwenden, wird im Folgenden das Disjunctive Graph Modell vorgestellt.

Das Disjunctive Graph Modell Das Disjunctive Graph Modell liefert eine Darstellung des Job Shop Scheduling Problems in Form eines graphentheoretischen Problems. Es wurde von Roy und Sussman(1964) [47] entwickelt und dient vielen Algorithmen als Grundlage. Mit Hilfe des Disjunctive Graph Modells kann einerseits eine Lösung ermittelt sowie andererseits eine Lösung repräsentiert werden.

Gegeben sei ein Job Shop Scheduling Problem. Dieses soll mittels eines Graphen $G = (V, E)$ darstellt werden. Die Bezeichnungen seien analog zu den Bezeichnungen aus der Problemdefinition. Dann wird der Graph G wie folgt konstruiert.
Für jede Operation o_{ij} besitze der Graph G einen Knoten v_{ij}. Es gilt

$$o_{ij} \in O \Leftrightarrow v_{ij} \in V.$$

Darüber hinaus enthalte der Graph einen Knoten $s \in V$ und einen Knoten $t \in V$.
Jeder Knoten besitze ein Gewicht. Dem Knoten v_{ij} werde als Gewicht die Bearbeitungsdauer t_{ij} der zugehörigen Operation o_{ij} und den Knoten s und t das Gewicht 0 zugewiesen. Sei $\phi : V \to \mathbb{R}_{\geq 0}$ die Funktion, die jedem Knoten sein Gewicht zuordnet. Dann gilt

$$\phi(v) = \begin{cases} 0, & \text{falls } v \in \{s,t\}, \\ t_{ij}, & \text{falls } v = v_{ij}, \end{cases} \qquad \forall\ v \in V.$$

Die Kantenmenge des Graphen setze sich sowohl aus gerichteten als auch aus ungerichteten Kanten zusammen. Die gerichteten Kanten E_J repräsentieren die Reihenfolge der Operationen der Jobs. Es gilt

$$vw \in E_J \Leftrightarrow (v = v_{i,l} \ \wedge\ w = v_{i,l+1} \text{ für einen Job } j_i)\,.$$

Darüber hinaus enthalte der Graph jeweils eine gerichtete Kante von dem Knoten s zu der ersten Operation aller Jobs

$$sv_{i1} \in E_J \text{ für alle } j_i \in J$$

und eine gerichtete Kante von der jeweils letzten Operation eines Jobs zu dem Knoten t

$$v_{iq_i}t \in E_J \text{ für alle } j_i \in J.$$

Die Kanten aus der Kantenmenge E_J werden als *Job-Kanten* bezeichnet. Zu den gerichteten Kanten enthält der Graph die ungerichteten Kanten E_M. Sie repräsentieren die Maschinenzugehörigkeit und werden auch als *Maschinen-Kanten* bezeichnet. Die Maschinen-Kanten

verbinden alle Operationen, die auf derselben Maschine bearbeitet werden müssen, das heißt

$$vw \in E_M \Leftrightarrow \mathfrak{M}(v) = \mathfrak{M}(w).$$

Die Kantenmenge E des Graphen G ist gegeben durch $E = E_J \cup E_M$.

Das folgende Beispiel 1.1.2 soll verdeutlichen, wie zu einer Job Shop Scheduling Instanz das zugehörige Disjunctive Graph Modell konstruiert wird.

Beispiel 1.1.2. *Gegeben seien drei Maschinen m_1, m_2, m_3 und drei Jobs j_1, j_2, j_3. Die zu den Jobs gehörigen Operationen sind mitsamt ihrer Bearbeitungsdauer und ihren vorbestimmten Maschinen in Tabelle 1.2 angegeben.*

Job	Operationen	Maschine	Bearbeitungsdauer
$\mathbf{j_1}$	o_{11}	m_1	1
	o_{12}	m_2	3
	o_{13}	m_3	2
$\mathbf{j_2}$	o_{21}	m_1	3
	o_{22}	m_2	2
$\mathbf{j_3}$	o_{31}	m_3	2
	o_{32}	m_2	1
	o_{33}	m_3	2

Tabelle 1.2: Job Shop Scheduling Instanz.

Konstruiert man das Disjunctive Graph Modell zu der gegebenen Job Shop Scheduling Instanz, so ergibt sich der Graph in Abbildung 1.4.

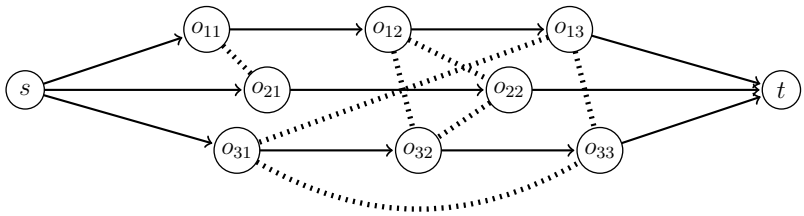


Abbildung 1.4: Disjunctive Graph Modell zur Instanz aus Tabelle 1.2.

Die Job-Kanten sind durch die gerichteten Kanten dargestellt, während die Maschinen-Kanten durch die gestrichelten Linien repräsentiert werden.

Sei $M_k = \{o \in O \ : \ \mathfrak{M}(o) = m_k\}$ die Menge der Operationen, die auf Maschine $m_k \in M$ bearbeitet werden müssen und sei weiter $y_k = |M_k|$. Dann ist eine *Maschinenreihenfolge* der Behandlungen auf Maschine $m_k \in M$ ist durch eine Permutation $\pi_k = (\pi_k(1), \ldots, \pi_k(y_k))$ der Behandlungen M_k definiert, wobei $\pi_k(l)$ das Element von M_k bezeichnet, das als l-tes auf Maschine m_k bearbeitet wird. Ein m-Tupel $\pi = (\pi_1, \ldots, \pi_m)$, dessen i-te Komponente

durch eine Maschinenreihenfolge π_i für Maschine m_i, $i = 1, \ldots, m$, gegeben ist, bezeichnen wir im Folgenden als *Bearbeitungsreihenfolge*.

Werden die Maschinen-Kanten so gerichtet, dass ein azyklischer Graph entsteht, so werden durch die Maschinen-Kanten Maschinenreihenfolgen impliziert [55]. Die durch einen azyklischen Graphen implizierte Bearbeitungsreihenfolge werde als *zulässig* bezeichnet.
Ist der entstehende Graph nicht azyklisch, so ist durch die Kanten entweder für eine Maschine keine Maschinenreihenfolge gegeben oder eine Maschinenreihenfolge widerspricht der Reihenfolge, die durch die Jobs gegeben ist. Diese Fälle sind in Beispiel 1.1.3 veranschaulicht.

Beispiel 1.1.3. ***Unzulässige Maschinenreihenfolgen.***

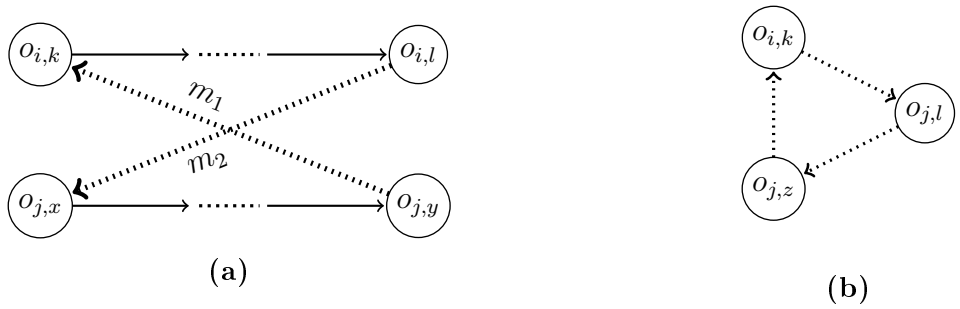


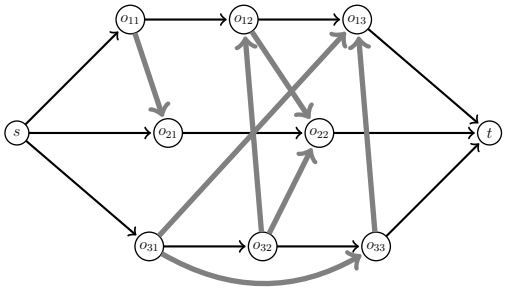
Abbildung 1.5: Gerichtete Maschinen-Kanten, die zu einer unzulässigen Maschinenreihenfolge führen.

In Abbildung 1.5a ist der Fall dargestellt, dass die Maschinenreihenfolge der Jobreihenfolge widerspricht und in Abbildung 1.5b ist dargestellt, dass durch die gerichteten Maschinen-Kanten keine Maschinenreihenfolge gegeben ist.

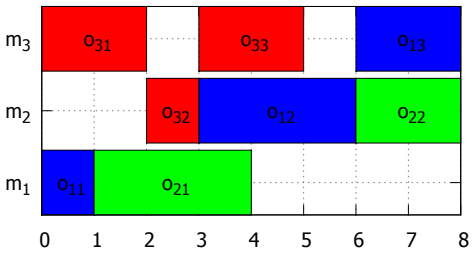
Ist eine zulässige Bearbeitungsreihenfolge gegeben, so kann für jede Operation o_{ij} die Startzeit berechnet werden, indem der längste Weg von s zum Knoten v_{ij} berechnet wird. Berechnen wir auf diese Weise die Startzeiten für alle Operationen, so erhalten wir einen zulässigen Schedule. Das Disjunctive Graph Modell, in dem alle Maschinen-Kanten gerichtet sind, bezeichnen wir im Folgenden als *gerichtetes* Disjunctive Graph Modell.

Ein gerichtetes Disjunctive Graph Modell zu der Job Shop Scheduling Instanz in Tabelle 1.2 ist in Abbildung 1.6a dargestellt. Das dem Graphen entsprechende Gantt Chart ist in Abbildung 1.6b zu sehen.

Beispiel 1.1.4. ***Fortsetzung.***



(a) Gerichtetes Disjunctive Graph Modell.



(b) Lösungsrepräsentation mittels Gantt Chart.

Abbildung 1.6: Repräsentationen eines zulässigen Schedules für die Job Shop Scheduling Instanz aus Tabelle 1.2.

Das Ziel des Job Shop Scheduling Problems ist die Minimierung des Makespans. Der Makespan entspricht im gerichteten Disjunctive Graph Modell der Länge eines längsten Weges vom Knoten s zum Knoten t.
Wir möchten zeigen, dass das Richten der Maschinen-Kanten dem Job Shop Scheduling Problem entspricht. Dazu müssen wir zeigen, dass mit Richten der Maschinen-Kanten ein Schedule erstellt werden kann, der einer optimalen Lösung entspricht. Der nachfolgende Satz liefert die nötige Aussage.

Satz 1.1.5. *Zu jeder optimalen Lösung für das Job Shop Scheduling Problem gibt es eine Lösung mit gleichem Makespan, die durch ein gerichtetes Disjunctive Graph Modell dargestellt werden kann.*

Beweis. *Gegeben sei eine optimale Lösung für das Job Shop Scheduling Problem. Dann ist durch die Lösung eine Maschinenreihenfolge für jede Maschine festgeschrieben. Erstellt man das gerichtete Disjunctive Graph Modell, das dieser Maschinenreihenfolge entspricht, so kann die dadurch repräsentierte Lösung keinen größeren Makespan besitzen, da jede Operation so früh wie möglich stattfindet. Keine Operation aus der optimalen Lösung kann vor einer Operation aus dem Disjunctive Graph Modell starten.* □

Da es zu jeder optimalen Lösung eine Lösung mit gleichem Makespan gibt, die durch das gerichtete Disjunctive Graph Modell repräsentiert werden kann, kann eine optimale Lösung mittels des Richtens der Maschinen-Kanten gefunden werden.
Das Job Shop Scheduling Problem entspricht somit dem Problem eine Orientierung des Graphen zu finden, so dass der dabei entstehende Graph azyklisch ist und die Länge des längsten Weges zwischen den Knoten s und t minimal ist.

Nachdem wir das Job Shop Scheduling Problem nun eingeführt haben, möchten wir als nächstes, dessen Verbindung zum Krankenhaus Scheduling Problem erläutern.

Die Beziehung zwischen Job Shop und Krankenhaus Scheduling Problem

Das Job Shop Scheduling Problem kann als Spezialfall des Krankenhaus Scheduling Problems gesehen werden. Nimmt man beim Krankenhaus Scheduling Problem an, dass alle Patienten zum Zeitpunkt Null ankommen, die Behandlungsgraphen zusammenhängende lineare Graphen sind und die Ressourcen rund um die Uhr geöffnet haben, so gleicht das Krankenhaus Scheduling Problem fast dem Job Shop Scheduling Problem. Lediglich die Zielfunktionen unterschieden sich. Während beim Krankenhaus Scheduling Problem die Summe der Aufenthaltszeiten und somit die Summe der Abschlusszeitpunkte der letzten Behandlungen minimiert wird, betrachtet man beim Job Shop Scheduling Problem die Minimierung des Makespans. Die Minimierung der Summe der Abschlusszeitpunkte der letzten Behandlungen bezeichnet man auch als Minimierung der Summe der Completiontimes.

Das Job Shop Scheduling Problem mit der Zielfunktion der Minimierung der Summe der Completiontimes ist in der Praxis weit weniger untersucht als das Problem mit der Minimierung des Makespans. Obwohl die Minimierung des Makespans nicht wie eine sinnvolle theoretische Zielfunktion scheint, ist diese sowohl in der Forschung als auch in der Praxis weit verbreitet. Das Makespan Kriterium ist auch von großer historischer Bedeutung. Denn es war die als erstes untersuchte Zielfunktion in den frühen Fünfzigerjahren [33]. Es ist das meist gebrauchte Kriterium in der Forschung, da es die grundlegenden Schwierigkeiten enthält, die implizit beim Berechnen eines optimalen Schedules gegeben sind, und aus mathematischer Sicht leicht zu formulieren ist.

Beim Krankenhaus Scheduling sind für die Ressourcen Öffnungszeiten gegeben. Beim klassischen Job Shop Scheduling Problem wird hingegen angenommen, dass die Maschinen durchgehend zur Verfügung stehen. Ansätze, in denen die Maschinen nicht während des gesamten Planungshorizonts durchgehend verfügbar sind, werden als Job Shop Scheduling Problem with Machine Availability Constraints bezeichnet. Da das Job Shop Scheduling Problem with Availability Constraints sehr komplex ist, gibt es nur wenig Ansätze, die sich mit diesem Problem beschäftigen. Availability Constraints wurden zunächst für das Single Machine Problem [2] [37], ein Job Shop Scheduling Problem mit $m = 1$, und das Parallel Machine Problem [49] [50], ein Job Shop Scheduling Problem, indem alle Jobs auf allen Maschinen bearbeitet werden können, eingeführt. Das Job Shop Scheduling Problem with Availability Constraints wurde unter anderem von Aggoune [3] betrachtet. In seinem Paper entwickelte Aggoune einen Branch-and-Bound Algorithmus.

Ein weiterer Unterschied zwischen dem Krankenhaus Scheduling Problem und dem Job Shop Scheduling Problem ist, dass die Behandlungsgraphen bei dem Krankenhaus Scheduling Problem keine zusammenhängenden linearen Graphen sein müssen, wohingegen dies beim Job Shop Scheduling Problem der Fall ist. In der Literatur wurde unseres Wissens nach der Fall, dass eine partielle Ordnung für die Bearbeitungsreihenfolge der Operationen gegeben ist, noch nicht betrachtet. Wären die zusammenhängenden Teilgraphen eines Behandlungsgraphen linearer Graphen, so würde das Krankenhaus Scheduling Problem unter Vernachlässigung der beschränkten Öffnungszeiten dem Job Shop Scheduling Problem with Job Families gleichkommen. Beim Job Shop Scheduling Problem with Job Families sind mehrere Jobs zu sogenannten Jobfamilien zusammengeschlossen. Zwischen den Jobs einer Familie gibt es keine Vorrangsbeziehungen. Wird die totale Familien Flow Time minimiert, so wird die Summe, der Zeitpunkt einer letzten Behandlung minimiert. Das Job Shop Scheduling Problem with Job Families wird zum Beispiel von Yu et al. [58] betrachtet.

1.2 Das Krankenhaus Scheduling Problem unter Unsicherheiten

Im Alltag eines Krankenhauses wird der Belegungsplan von Ressourcen meistens mit Störungen und unvorhergesehenen Ereignissen konfrontiert. So kann es vorkommen, dass eine Ressource ausfällt, eine Behandlung länger dauert als geplant oder zusätzliche Behandlungen eingeplant werden müssen. Auch wenn in der Realität verschiedene Unsicherheitsfaktoren auftreten können, die die Ausführung eines Belegungsplans beeinflussen, beschränken wir uns in dieser Arbeit auf den Fall, dass während eines Krankenhaustages zu zufälligen Zeitpunkten weitere Patienten auftreten können, deren Behandlungen in den bestehenden Plan integriert werden müssen. Dieses Problem soll durch das Krankenhaus Scheduling Problem unter Unsicherheiten beschrieben werden.
Das Krankenhaus Scheduling Problem unter Unsicherheiten richtet sich an eine Situation, bei der es zum Zeitpunkt der Planung eine Menge von Patienten gibt, für die der Ankunftszeitpunkt und die zu durchlaufenden Behandlungen bekannt sind. Darüber hinaus können im Planungszeitraum noch weitere Patienten auftreten. Es ist nicht bekannt, wie viele Patienten zusätzlich auftreten werden. Ebenso sind die Ankunftszeitpunkte und die zu durchlaufenden Behandlungen ungewiss. Ziel des Krankenhaus Scheduling Problems unter Unsicherheiten ist es, für die zum Zeitpunkt der Planung sicheren Patienten die Startzeitpunkte der Behandlungen so festzulegen, dass diese einerseits für eine möglichst kurze Aufenthaltsdauer im Krankenhaus verweilen und andererseits der Schedule freie Zeiträume besitzt, in denen die hinzukommenden zufällig auftretenden Patienten behandelt werden können. Formal lässt sich das Krankenhaus Scheduling Problem wie folgt beschreiben.

Krankenhaus Scheduling Problem unter Unsicherheiten Gegeben sei eine Menge $P = \{p_1, \ldots, p_n\}$ von Patienten, die im Krankenhaus behandelt werden möchten. Für jeden Patienten $p_i \in P$ gibt es ein Ankunftszeitpunkt Arr_i. Zum Ankunftszeitpunkt betritt der Patient das Krankenhaus und steht ab dann für Untersuchungen zur Verfügung. Ein Patient p_i soll im Krankenhaus q_i Behandlungen $B_i = \{b_{i1}, \ldots, b_{iq_i}\}$ bekommen. Zwischen den Behandlungen kann es Rangfolgebeziehungen Θ_i geben, wobei eine Rangfolgebeziehung durch ein geordnetes Paar von Behandlungen gegeben ist. Eine Rangfolgebeziehung (b_{ij}, b_{ik}) bedeutet, dass Behandlung b_{ij} vor Behandlung b_{ik} stattfinden soll. Eine Behandlung kann erst dann stattfinden, wenn alle ranghöheren Behandlungen abgeschlossen wurden. Die Rangfolgebeziehungen können durch einen Behandlungsgraphen $D(p_i)$ dargestellt werden, der gemäß der Bedingungen 1.1.1 und 1.1.2 konstruiert wird. Das Krankenhaus hat zur Behandlung der Patienten die Ressourcen $R = \{r_1, \ldots, r_m\}$ zur Verfügung, die während ihrer jeweiligen Öffnungszeiten zur Behandlung der Patienten genutzt werden können. Für eine Ressource $r_k \in R$ sind die Öffnungszeiten, abhängig vom Wochentag w, durch einen Öffnungszeitpunkt $r_k^{open}(w)$ und einen Schlusszeitpunkt $r_k^{close}(w)$ festgelegt. Eine Behandlung $b_{ij} \in \bigcup_{i=1}^{n} B_i$ benötigt die Zeit t_{ij} und kann nur auf der vorgegebenen Ressource $R(b_{ij}) \in R$ durchgeführt werden.
Zu der Menge der bekannten Patienten P gibt es eine weitere Menge von Patienten P^*, die im Krankenhaus behandelt werden möchten. Die Menge der Patienten P^* sei die Menge der *Notfallpatienten*. Die Anzahl n^* der Notfallpatienten ist genauso wie der Ankunftszeitpunkt $Arr(p_i^*)$ eines Notfallpatienten $p_i^* \in P^* = \{p_1^*, \ldots, p_{n^*}^*\}$ ungewiss. Erst bei Ankunft eines Notfallpatienten $p_i^* \in P^*$ im Krankenhaus wird die Menge seiner Behandlungen B_i^* bekannt.

Den Behandlungen der Patienten sollen Starttermine zugewiesen werden. Beim Zuweisen von Terminen zu den Behandlungen muss Folgendes beachtet werden:

(I) Eine Behandlung kann nicht unterbrochen werden.

(II) Eine Behandlung muss vollständig in den Öffnungszeiten der Ressource liegen.

(III) Auf einer Ressource kann zu jedem Zeitpunkt nur eine Behandlung stattfinden.

(IV) Ein Patient kann zu jedem Zeitpunkt nur eine Behandlung bekommen.

Wird jeder Untersuchung $b \in B = \bigcup_{i=1}^{n} B_i$ ein Starttermin $S_t(b)$ zugeordnet, so wird eine solche Zuordnung $S = (S_t)$ als *Schedule* bezeichnet. Ein Schedule ist *zulässig*, wenn er die Bedingungen (I)-(IV) erfüllt.
Das *Krankenhaus Scheduling Problem unter Unsicherheiten* besteht darin, einen zulässigen Schedule zu finden, der zum einen die Summe der Aufenthaltszeiten der Patienten P im Krankenhaus und somit die Entlassungszeitpunkte $C_i = max_{l \in \{1,\dots,q_i\}} S_t(b_{il}) + t_{il}$ minimiert. Zum anderen soll der Schedule Freiräume beinhalten, so dass die Behandlungen von Notfallpatienten, die während der Ausführung des Plans auftreten, in diesen Freiräumen stattfinden können.
Das Einplanen von Freiräumen und die Minimierung von Aufenthaltszeiten widersprechen sich. Die Aufgabe des Krankenhaus Scheduling Problems unter Unsicherheiten besteht daher darin, einen geeigneten Kompromiss zwischen den beiden Kriterien zu finden.

Zur Vereinfachung nehmen wir in dieser Arbeit an, dass alle Ressourcen die gleichen Öffnungszeiten besitzen.

Im nächsten Kapitel möchten wir vorstellen, welche Ansätze es gibt, um Probleme unter Unsicherheiten zu lösen.

Kapitel 2

Optimierung unter Unsicherheiten

Im Falle des Krankenhaus Scheduling Problems unter Unsicherheiten möchten wir den Patienten Behandlungszeitpunkte mit dem Wissen zuweisen, dass noch weitere Behandlungen auf den Ressourcen stattfinden müssen. Diese Behandlungen gehören zu Notfallpatienten. Die Anzahl der Notfallpatienten sowie deren Ankunftszeitpunkt und deren Behandlungen sind uns nicht bekannt. Wir müssen also eine Entscheidung treffen, den bekannten Behandlungen Startzeiten zuordnen, ohne dass uns sämtliche Informationen vorliegen. Daher ist es schwer zu beurteilen, wie sich unsere Lösung bei dem Eintreten der Unsicherheiten, also dem Eintreffen von Notfallpatienten, verhält.

An Probleme wie dieses, die dadurch charakterisiert sind, dass Entscheidungen getroffen werden müssen, ohne deren komplette Auswirkung zu kennen, richtet sich die Optimierung unter Unsicherheiten.

Mit der zunehmenden Anwendung der Optimierung auf Probleme aus der Praxis in der zweiten Hälfte des zwanzigsten Jahrhunderts wurde schnell festgestellt, dass viele dieser Probleme mit Unsicherheiten behaftet sind. Gerade in Bereichen wie der Produktions- oder der Transportplanung, deren plangemäße Ausführung auf dem Funktionieren von technologischen Systemen beruht, wurde die Notwendigkeit erkannt, die Unsicherheiten in das Lösen der Probleme miteinzubeziehen [48].
Angefangen mit der Arbeit von Beale (1955) [9], Bellman (1957) [10], Bellman und Zadeh (1970)[11], Charnes und Cooper (1959) [20] und Dantzig (1955) [23] haben die Theorie und die Algorithmen im Bereich der Optimierung unter Unsicherheiten eine rasante Entwicklung durchlaufen [48]. Nach Dantzig ist die Optimierung unter Unsicherheiten bis heute eines der bedeutendsten ungelösten Probleme der Optimierung [48].
Es gibt eine Vielzahl von Ansätzen, die in Bezug auf die Optimierung unter Unsicherheiten entwickelt wurden, darunter zum einen die Stochastische Optimierung mit Problemen mit Kompensation oder dem Anwenden von stochastischen Nebenbedingungen und zum anderen die Robuste Optimierung. Während die Stochastische Optimierung wahrscheinlich der bekannteste Ansatz zur Lösung von Problemen unter Unsicherheiten ist und seit der Einführung der Linearen Optimierung untersucht wurde, ist das Feld der Robusten Optimierung ein Gebiet, dem aktuell viel Aufmerksamkeit gewidmet wird.
In den nächsten beiden Abschnitten möchten wir die Ansätze Stochastische Optimierung und Robuste Optimierung vorstellen und im letzten Abschnitt den aktuellen Stand der Optimierung unter Unsicherheiten in Bezug auf Scheduling Probleme schildern.

2.1 Stochastische Optimierung

Deterministische Optimierungsprobleme bilden die Realität nur unzureichend ab, da in der Realität häufig nicht alle Daten mit Sicherheit gegeben sind. In der Praxis sind oft

Entscheidungen gefordert, die unmittelbar und ohne Wissen der Zukunft getroffen werden müssen. Bei der Stochastischen Optimierung wird davon ausgegangen, dass ein Teil der Daten mit Unsicherheiten behaftet ist und diese Daten als Zufallsvariablen gegeben sind.

Die Stochastische Optimierung hat ihren Ursprung in Dantzig. Dantzig wurde durch Diskussionen mit A. Ferguson dazu angeregt, das Auftreten von unsicheren Bedarfen für das Problem der optimalen Zuweisung von einer Menge von Gepäckwagen zu Fluglinien zu betrachten [23]. Daraufhin führte Dantzig ein Framework für Lineare Optimierung mit unsicheren Parametern ein. Seitdem wurde auf dieses Feld der Optimierung viel Aufmerksamkeit gerichtet.

Wir werden zwei Ansätze der Stochastischen Optimierung betrachten, in denen mit den Unsicherheiten auf unterschiedliche Weisen umgegangen wird. Zum einen werden wir Probleme mit Kompensation und zum anderen Probleme mit stochastischen Nebenbedingungen betrachten.

2.1.1 Probleme mit Kompensation

Bei Problemen mit Kompensation müssen Entscheidungen getroffen werden, ohne dass zum Zeitpunkt der Entscheidung die Realisation aller Zufallsvariablen bekannt ist. Zu jedem Zeitpunkt, an dem die Realisierung einer Zufallsvariablen bekannt wird, gibt es die Möglichkeit, die bisherige Entscheidung anzupassen. Das Bekanntwerden von Realisationen von Zufallsvaraiablen kann nur zu diskreten Zeitpunkten auftreten. Die Zufallsvariablen werden denjenigen diskreten Zeitpunkten innerhalb des Planungszeitraumes zugeordnet, an denen die Entscheidungsfindung stattfindet.

Werden alle Realisierungen zum gleichen Zeitpunkt bekannt, so spricht man von einem zweistufigen stochastischen Problem. In der ersten Stufe wird eine Entscheidung ohne Kenntnis der Realisationen der zufälligen Variablen getroffen und in der zweiten Stufe kann, nach Bekanntwerden der Realisationen der Zufallsvariablen, die vorher getroffene Entscheidung angepasst werden, um unerwartete Realisationen auszugleichen oder um die Zulässigkeit zu sichern. Werden die Realisierungen zu mehreren Zeitpunkten bekannt, so wird von mehrstufigen stochastischen Problemen mit Kompensation gesprochen. Müssen zu Beginn des Planungszeitraumes Entscheidungen getroffen werden, die für alle weiteren Zeitpunkte fixiert sind, so spricht man von einstufigen Problemen. Bei einstufigen Problemen findet keine Kompensation statt.

Die Grundidee stochastischer Probleme mit Kompensation geht auf Dantzig [23] zurück. Er führte ein Modell ein, um zweistufige Optimierungsprobleme zu lösen. Das Konzept der Kompensation wurde auf die lineare, ganzzahlige und nicht-lineare Optimierung angewendet. Der Ursprung liegt jedoch in der linearen Optimierung. Viele Algorithmen, die für stochastische lineare Programme entwickelt wurden, können auf den nicht-linearen Fall übertragen werden[48]. Wir möchten das Konzept daher im Folgenden durch ein zweistufiges Problem für den linearen Fall veranschaulichen.

Gegeben sei ein stochastisches lineares Programm. Eine standardisierte Formulierung [34] für das zweistufige lineare Problem ist gegeben durch

$$\begin{aligned} \min\, & c^t x + E_{w\in\Omega}[Q(x,w)], \\ \text{s.t. } & x \in X, \end{aligned} \tag{2.1.1}$$

mit

$$\begin{aligned} Q(x,\omega) = \min\, & f(\omega)^T y, \\ \text{s.t. } & D(\omega)y \geq h(\omega) + T(\omega)x, \\ & y \in Y, \end{aligned} \tag{2.1.2}$$

wobei $X \subset \mathbb{R}^{p_1}$ und $Y \subset \mathbb{R}^{p_2}$ polyedrische Mengen sind und $c \in \mathbb{R}^{p_1}$. Weiter ist ω eine Zufallsvariable aus dem Wahrscheinlichkeitsraum $(\Omega, \mathcal{F}, \mathcal{P})$ mit $\Omega \subseteq \mathbb{R}^q$, $f : \Omega \to \mathbb{R}^{p_2}$, $h : \Omega \to \mathbb{R}^{v_2}$, $D : \Omega \to \mathbb{R}^{v_2 \times p_2}$ und $T : \Omega \to \mathbb{R}^{v_2 \times p_1}$.
Das Problem in 2.1.1 beschreibt die erste Stufe. Die Variablen $x \in X$ stellen die Entscheidungen dar, die im Voraus getroffen werden müssen, bevor die Realisationen der unsicheren Parameter $\omega \in \Omega$ bekannt werden. Das Problem in 2.1.2 repräsentiert die zweite Stufe. Die Variablen $y \in Y$ beschreiben die Entscheidungen, die getroffen werden können, um unerwartete Realisierungen von ω auszugleichen.
Ein bekanntes Beispiel für ein zweistufiges Problem mit Kompensation entspricht der Bestellpolitik eines Unternehmens. Zum Zeitpunkt Null muss entschieden werden, wie viel Bedarf an Rohstoffen für einen vorgegeben Zeitraum besteht. Es ist zu diesem Zeitpunkt nicht bekannt, wie groß der Absatz der eigenen Produkte in der Zeitspanne sein wird. Zu einem späteren Zeitpunkt in der Zeitspanne kann nachbestellt werden. Wird zu diesem Zeitpunkt nachbestellt, so müssen ein zweites Mal Lieferkosten bezahlt werden. Wird zum ersten Zeitpunkt eine große Menge Rohstoffe bestellt, so müssen hohe Lagerkosten bezahlt werden. Ziel ist es, stets lieferfähig zu sein und dabei die Kosten möglichst gering zu halten.

2.1.2 Probleme mit stochastischen Bedingungen

Statt die Bedingungen mit absoluter Sicherheit zu erfüllen, kann es vorteilhaft sein, Bedingungen nur mit großer Wahrscheinlichkeit zu erfüllen. Durch den Verzicht auf absolute Sicherheit, kann es möglich sein, einen besseren Zielfunktionswert zu erreichen.
Der Ansatz der Einführung stochastischer Nebenbedingungen für stochastische Probleme geht auf Charnes und Cooper [20] zurück. Es wird die Forderung fallengelassen, dass eine Lösung für alle Nebenbedingungen und möglichen Szenarien zulässig sein muss. Stattdessen werden sogenannte stochastische Nebenbedingungen definiert, die dadurch charakterisiert sind, dass sie in einem Teil der Szenarien verletzt sein dürfen.

Wir betrachten das klassische Modell der Linearen Optimierung

$$\begin{aligned} \max\, & c^T x, \\ \text{s.t. } & Ax \geq b, \\ & x \geq 0, \end{aligned} \tag{2.1.3}$$

wobei c, $x \in \mathbb{R}^g$, $b \in \mathbb{R}^h$ und A eine $g \times h$-Matrix ist. Unter der Annahme, dass es Unsicherheiten in der Matrix A und im Vektor b gibt und dass das System die Ungleichung höchstens mit der Wahrscheinlichkeit $\epsilon \in [0,1]$ verletzten darf, kann das stochastische lineare Optimierungsproblem mit stochastischen Nebenbedingungen wie folgt formuliert werden:

$$\begin{aligned} \max\, & c^T x, \\ \text{s.t. } & P(Ax \geq b) \geq 1 - \epsilon, \\ & x \geq 0. \end{aligned} \tag{2.1.4}$$

Die Nebenbedingungen müssen nicht mit absoluter Sicherheit erfüllt werden, sondern dürfen in ϵ Prozent der Fälle nicht eingehalten werden. In 2.1.4 muss die Wahrscheinlichkeit, dass die gesamten Nebenbedingungen erfüllt sind, mindestens $1 - \epsilon$ betragen.
In Modell 2.1.4 wird für alle Nebenbedingungen eine gemeinsame stochastische Ungleichung betrachtet. Diese gemeinsame stochastische Nebenbedingung wurde von Miller und Wagner [40] eingeführt. Bei der Einführung von Problemen mit stochastischen Nebenbedingungen durch Charnes und Cooper wurden Programme betrachtet, in denen jede Nebenbedingung separat mit einer bestimmten Wahrscheinlichkeit erfüllt werden musste.

Der Ansatz der Stochastischen Optimierung, Probleme mit stochastischen Nebenbedingungen zu formulieren, führt uns in die Richtung der Robusten Optimierung. Auch bei der Robusten Optimierung muss eine Lösung nicht für alle möglichen Szenarien zulässig sein. Anders als bei Problemen mit Stochastischen Nebenbedingungen gibt es in der Robusten Optimierung jedoch keinen vorgegebenen Prozentsatz, für den die Zulässigkeit bei Betrachtung aller Szenarien eingehalten werden muss, sondern die Zulässigkeit muss für eine ausgewählte Menge von Szenarien zu hundert Prozent gewährleistet sein.

2.2 Robuste Optimierung

Die Robuste Optimierung ist ein in neuerer Zeit entstandener Zweig der Optimierung unter Unsicherheiten. Sie unterscheidet sich von der Stochastischen Optimierung dadurch, dass die Unsicherheiten nicht in Form von Zufallsvariablen und damit nicht in stochastischer Form gegeben sind, sondern die Unsicherheiten in einer eher deterministischen Weise, nämlich in Form einer Menge von potentiell eintretenden Szenarien, gegeben sind. Die Menge der potentiell eintretenden Szenarien wird als *Unsicherheitsmenge* bezeichnet. Der Entscheider sucht eine Lösung, die für alle Szenarien der gegebenen Unsicherheitsmenge optimal ist. Die Größe der Unsicherheitsmenge kann als Maß der Robustheit gesehen werden, das gegenüber der Menge aller möglichen Unsicherheiten gefordert wird.
Der Ansatz der Robusten Optimierung kann dadurch motiviert werden, dass die Idee, die Unsicherheiten in Form von Szenarien zu betrachten, ein interessantes Konzept ist und oft der Vorstellung von Unsicherheiten in vielen Anwendungen entspricht [17].
In den frühen 1970er Jahren war Soyster [51] einer der ersten, der sich mit dem Gedanken der Robusten Optimierung beschäftigte und explizite Ansätze für die Robuste Optimierung entwickelte. Soyster betrachtete ein lineares Optimierungsproblem, für das festgelegt wurde, dass die Spaltenvektoren der Unsicherheits-Matrix in einer konvexen Menge liegen müssen. Falk[25] folgte ein paar Jahre später. Aber erst in den späten 1990ern wurde die Robuste Optimierung populär, nachdem Ben-Tal and Nemirovski [13] [14] [15] und El Ghaoui et al. [27] [28] Abhandlungen veröffentlichten, die die Vorteile in der Computertechnologie und die Entwicklung einer schnellen Interior-Point-Methode für konvexe Optimierung kombinierten [17].

Da bei der Robusten Optimierung nicht allein die Optimierung der Zielfunktion im Fokus steht, sondern zudem die Robustheit einer Lösung betrachtet wird, müssen Begriffe wie zulässige Lösung und optimaler Zielfunktionswert neu definiert werden. Da die Herkunft der Robusten Optimierung in der Linearen Optimierung liegt, wollen wir die Begriffe und Konzepte für Lineare Optimierungsprobleme einführen. Dabei orientieren wir uns vor allem an Ben-Tal, Ghaoui und Nemirovski [12].

Ein Lineares Programm, in dem die Unsicherheiten durch eine Unsicherheistmenge gegeben

sind, bezeichnen wir als *Unsicheres Lineares Optimierungsproblem.* Ein Unsicheres Lineares Optimierungsproblem ist eine Kollektion Linearer Optimierungsprobleme

$$\left\{\min_x\{c^T x \ : \ Ax \leq b\} \ : \ (c, A, b) \in \mathfrak{U}\right\},$$

wobei die Daten durch eine Unsicherheitsmenge $\mathfrak{U} \subset \mathbb{R}^{(g+1)\times(h+1)}$ gegeben sind.

Wir betrachten Unsichere Lineare Opimierungsprobleme mit folgenden Eigenschaften:

(i) Die Einträge im Entscheidungsvektor x sind First-Stage Entscheidungen, das heißt, alle Entscheidungen müssen getroffen werden, bevor die Realisierung der Daten bekannt wird.

(ii) Liegen die Realisationen der Daten in der Unsicherheitsmenge $\mathfrak{U}$, so ist der Entscheider für die Auswirkungen seiner Entscheidung verantwortlich.

(iii) Die Nebenbedingungen müssen in allen Realisierungen aus der Unsicherheitsmenge eingehalten werden.

Durch die Eigenschaften ist gegeben, dass das Lösen eines Unsicheren Linearen Optimierungsproblems eine Absicherung gegen die Unsicherheiten impliziert. Daher wird eine robust zulässige Lösung wie folgt definiert.

Definition 2.2.1. *Ein Vektor $x \in \mathbb{R}^h$ ist eine robuste zulässige Lösung für ein Unsicheres Lineares Optimierungsproblem, wenn*

$$Ax \leq b \quad \forall \ (c, A, b) \in \mathfrak{U}$$

gilt, das heißt, dass wenn für alle Realisierungen in der Unsicherheitsmenge die Nebenbedingungen eingehalten werden.

Es bleibt die Frage offen, wann eine Lösung optimal ist. Dazu betrachten wir das Konzept des Robust Counterparts.

2.2.1 Robust Counterpart

Der Robust Counterpart ist ein Optimierungsproblem mit dem Ziel, eine robuste zulässige Lösung zu finden, die den schlecht möglichsten Zielfunktionswert bei dem Eintreten eines Szenarios aus der Unsicherheitsmenge optimiert.

Der schlechteste Wert der Zielfunktion, der beim Eintreten eines Szenarios aus der Unsicherheitsmenge für eine Lösung erzielt werden kann, wird als robuster Wert bezeichnet.

Definition 2.2.2. *Gegeben sei eine mögliche Lösung x. Dann ist der robuste Wert $\hat{c}(x)$ von x gegeben durch*

$$\hat{c}(x) = \sup_{(c,A,b)\in\mathfrak{U}} c^T x.$$

Der robuste Wert entspricht dem schlechtesten Wert der Zielfunktion über allen Realisierungen in der Unsicherheitsmenge $\mathfrak{U}$. Man kann den robusten Wert als den Worst-Case-Wert der Zielfunktion interpretieren. Möchte man eine Lösung mit dem besten robusten Wert ermitteln, so gelangt man zum Robust Counterpart.

Definition 2.2.3. *Der Robust Counterpart eines Unsicheren Linearen Optimierungsproblems ist das Optimierungsproblem*

$$\min_x \left\{ \hat{c}(x) = \sup_{(c,A,b)\in\mathfrak{U}} c^T x \ : \ Ax \leq b \ \ \forall\ (c,A,b) \in \mathfrak{U} \right\}. \tag{2.2.1}$$

Eine optimale Lösung für den Robust Counterpart heißt robuste optimale Lösung und der optimale Zielfunktionswert $\hat{c}(x)$ heißt robust optimaler Wert.

Beim Robust Counterpart wird eine Lösung gesucht, die gegen alle Szenarien der Unsicherheitsmenge abgesichert ist. Mit dem Konzept der Gamma-Robustheit betrachten wir ein Konzept, in dem es eine Schranke für die Menge der Unsicherheiten gibt, gegen die eine Lösung abgesichert sein soll. Bei der Vorstellung des Konzepts der Γ-Robustheit beziehen wir uns vorwiegend auf Bertsimas und Thiele [18].

2.2.2 Γ-Robustheit

Um das Konzept der Gamma-Robustheit vorzustellen, können wir ohne Einschränkung annehmen, dass das Unsichere Lineare Optimierungsproblem eine sichere Zielfunktion besitzt. Zur Vereinfachung nehmen wir weiter an, dass auch der Vektor b sicher ist. Das Konzept der Gamma-Robustheit beruht auf der Annahme, dass die Koeffizienten der Matrix A innerhalb eines symmetrischen Intervalls prognostiziert werden können. Es gilt

$$a_{ij} \in [\overline{a}_{ij} - \hat{a}_{ij}, \overline{a}_{ij} + \hat{a}_{ij}],$$

wobei $\overline{a}_{ij}$ den nominalen Wert und $\hat{a}_{ij}$ die Abweichung bezeichnet. Wenn $\hat{a}_{ij} = 0$, dann ist $a_{ij} = \overline{a}_{ij}$ sicher.
Sind die Koeffizienten auf die entsprechenden Intervalle begrenzt, so lässt sich die skalierte Abweichung

$$z_{ij} = \frac{a_{ij} - \overline{a}_{ij}}{\hat{a}_{ij}}$$

definieren. Es gilt $z_{ij} \in [-1, 1]$.

Wir möchten eine Lösung nicht gegen alle möglichen Szenarien absichern, sondern möchten die Unsicherheitsmenge auf Szenarien reduzieren, deren Abweichung von den nominalen Daten beschränkt ist. Diese Reduzierung erreichen wir, indem wir ein maximales Ausmaß der Abweichungen festlegen.

Bezeichnung 2.2.4. *Sei die maximal erlaubte Abweichung der Daten innerhalb der Zeile i der Unsicherheitsmatrix A beschränkt durch Γ_i durch*

$$\sum_{j=1}^{h} |z_{ij}| \leq \Gamma_i \quad \forall\ i.$$

Dann bezeichnen wir $\Gamma_i \in [0, h]$ als Unsicherheitsbudget.

Ist $\Gamma_i \in (0, h)$ so muss der Entscheider einen Kompromiss zwischen Maß des Konservatismus und Absicherung finden. Für $\Gamma_i = 0$ wird das Problem für die nominalen Daten gelöst und für $\Gamma_i = h$ muss die Lösung gegen alle möglichen Szenarien abgesichert werden.

Definiert man die Mengen

$$\mathfrak{U} = \{(a_{ij}) \ : \ a_{ij} = \overline{a}_{ij} + \hat{a}_{ij} z_{ij} \ \forall \ i, j, z \in \mathfrak{Z}\}$$

mit

$$\mathfrak{Z} = \left\{ z \ : \ |z_{ij}| \leq 1 \ \forall \ i, j, \ \sum_{j=1}^{h} |z_{ij}| \leq \Gamma_i, \ \forall \ i \right\},$$

dann kann der „Robust Counterpart“ des Unsicheren Linearen Optimierungsproblems formuliert werden als

$$\begin{aligned} &\min c^T x, \\ &\text{s.t. } \overline{a}_i^T x + \max_{z_i \in \mathfrak{Z}_i} \sum_{j=1}^{h} \hat{a}_{ij} z_{ij} x_j \leq b_i \quad \forall i, \end{aligned} \tag{2.2.2}$$

wobei z_i der Vektor ist, dessen j-tes Element z_{ij} ist, und $\mathfrak{Z}_i$ durch

$$\mathfrak{Z}_i = \left\{ z_i \ : \ |z_{ij}| \leq 1, \ \forall \ j, \ \sum_{j=1}^{h} |z_{ij}| \leq \Gamma_i \right\}$$

definiert ist.

Es ist klar, dass je größer Γ_i gewählt wird, desto größer ist das Level der Absicherung und umso schlechter ist der Zielfunktionswert der robust optimalen Lösung. Um den Verlust an Zielfunktion durch steigende Robustheit messen zu können, wird der prozentuale Anstieg des optimalen Zielfunktionswertes betrachtet.

Definition 2.2.5. *Mit dem Price of Robustness wird der prozentuale Anstieg des optimalen Zielfunktionswertes für $\Gamma_i = k$ in Bezug zu dem optimalen Zielfunktionswert bei $\Gamma_i = 0$ bezeichnet.*

Wir werden sehen, dass auch im Fall des Krankenhaus Scheduling Problems der Zielfunktionswert mit steigendem Level der Absicherung gegen Unsicherheiten schlechter wird. Der von uns vorgestellte Ansatz wird sowohl das Element von Szenarien als auch die nur zu einem bestimmten Prozentsatz erfüllbaren Bedingungen enthalten. Bevor wir im Kapitel 3 näher auf unsere Methode zur Lösung des Krankenhaus Scheduling Problems eingehen, soll im Abschnitt 2.3 vorgestellt werden, welche Ansätze es für Optimierung unter Unsicherheiten im Bezug auf Scheduling Probleme gibt.

2.3 Optimierung unter Unsicherheiten für Scheduling Probleme

Mit Scheduling Probleme wird die Gesamtheit aller Probleme bezeichnet, in denen Operationen Startzeiten auf Maschinen zugeordnet werden müssen, so dass sich die Bearbeitung von Operationen auf einer Maschine nicht überschneiden. Scheduling Probleme werden im Allgemeinen in drei Kategorien unterteilt.

Eine Kategorie sind die Job Shop Scheduling Probleme. Für jeden Job gibt es eine feste Reihenfolge, in der die Operationen bearbeitet werden müssen. Das allgemeine Job Shop Scheduling Problem wurde bereits in Kapitel 1 vorgestellt. Viel Aufmerksamkeit bekam der Spezialfall mit nur einer Maschine, das Single Machine Scheduling Problem. Dieses kann in polynomieller Zeit gelöst werden. Zwei weitere viel untersuchte Variationen des Job Shop Scheduling Problems sind das Flexible Job Shop und das Parallel Machine Scheduling Problem. Kann eine Operation auf einer Teilmenge von Maschinen stattfinden, so wird das Problem als Flexible Job Shop bezeichnet. Dabei kann die Bearbeitungsdauer je nach Maschine variieren. Kann jede Operation auf jeder Maschine stattfinden, so wird vom Parallel Machine Scheduling Problem gesprochen. Als weitere Kategorie sind die Flow Shop Scheduling Probleme zu nennen. Bei Flow Shop Scheduling Problemen müssen alle Jobs die gleichen Maschinen in der gleichen Reihenfolge durchlaufen. Typische Flow-Shops findet man in der Metallherstellungsindustrie. Die Flow Shop Scheduling Probleme können als ein spezielles Job Shop Scheduling Problem angesehen werden. Als dritte Kategorie gibt es die Open Shop Probleme. Die Open Shop Probleme zeichnen sich dadurch aus, dass es keinerlei Rangfolgebeziehungen zwischen den Operationen eines Jobs gibt.

Scheduling Probleme sind in der Praxis weit verbreitet. Daher wird ihnen ein hohes Interesse entgegengebracht.

Scheduling Problem

Als Scheduling Probleme verstehen wir diejenigen Probleme, deren Ziel es ist, für Operationen von Jobs Startzeiten auf Maschinen festzulegen, so dass sich die Bearbeitung von zwei Operationen auf einer Maschine nicht überschneiden.

Gegeben ist eine Menge von Jobs $J = \{J_1, \ldots, J_n\}$ und eine Menge von Maschinen $M = \{M_1, \ldots, M_m\}$. Jeder Job besteht aus q_j Operationen $O = \{O_{j1}, \ldots, O_{jq_j}\}$. Eine Lösung ist ein Schedule, in dem jede Operation auf einer passenden Maschine bearbeitet wird, wobei gegebenenfalls eine vorgeschriebene Reihenfolge der Operationen eines Jobs beachtet wird und zu jeder Zeit auf einer Maschine nur eine Behandlung stattfindet.

In der Regel dürfen die Operationen nicht unterbrochen werden und können nur auf einer vorgegebenen Teilmenge der Maschinen bearbeitet werden. Es wurden bereits mehrere Variationen von Scheduling Problemen vorgestellt. Die verschiedenen Probleme können kombiniert oder um zusätzliche Variablen oder Bedingungen ergänzt werden. So kann es für einen Job, wie im Fall des Krankenhaus Scheduling Problems, Ankunftszeiten geben, aber auch Fälligkeitsdaten sind geläufig. Die Qualität einer Lösung wird anhand einer Zielfunktion gemessen. Diese kann zum Beispiel, wie beim Job Shop Scheduling Problem, der Makespan sein oder, wenn die Jobs Fälligkeitsdaten besitzen, auf der Verspätung von Jobs basieren.

Optimierung unter Unsicherheiten für Scheduling Probleme

Ziel der Optimierung unter Unsicherheiten für Scheduling Probleme ist es, einen Schedule zu ermitteln, der möglichst wenig von Unsicherheiten beeinflusst wird. Man spricht in diesem Zusammenhang davon, dass ein Schedule möglichst robust gegenüber den Unsicherheiten sein soll, wobei es keine eindeutige Definition für Robustheit im Kontext von Scheduling Problemen gibt. Obwohl es umfangreiche Methoden in der Literatur gibt, ist das Konzept und die Definition von Robustheit für einen Schedule eine offene Frage. In der bestehenden Literatur ist das Maß der Robustheit hauptsächlich in zwei Kategorien eingeteilt: Quality Robustness und Solution Robustness [32].

Quality Robustness bezieht sich auf die Veränderung des Zielfunktionswertes bei Unsicherheit, während sich Solution Robustness in der Regel auf die Verzögerungen der Start- oder Endzeit von Operationen bezieht. Die Solution Robustness wird im Allgemeinen auch als Stabilität eines Schedules bezeichnet. Mit Quality Robustness wird die Fähigkeit eines Schedules bezeichnet, ein bestimmtes Level an Qulatität der Lösung, gemessen an dem Zielfunktionswert, zu erhalten. Wird von einem Robusten Schedule gesprochen, so ist in der Regel gemeint, dass sich die Zielfunktion für den Schedule nicht verschlechtert [56]. Im Sinne der Solution Robustness ist es schwierig, einen Schedule gegen alle Unsicherheiten zu wappnen. Daher wird das Augenmerk meist nicht auf die Solution Robustness gerichtet sondern die Ermittlung eines Schedules mit Quality Robustness in den Fokus gestellt.

Da die Realisierungen der Unsicherheiten ungewiss sind, ist es schwierig die Effekte der Unsicherheiten auf den zugrundeliegenden Schedule einzuschätzen. Eine Möglichkeit ist es, die Unsicherheiten mittels Szenarien zu simulieren. Dieses kann jedoch sehr rechenaufwendig sein. Um dieses Problem zu überwinden, werden Maße entwickelt, die die Robustheit eines Schedules approximieren sollen. Es gibt viele Kriterien eines Schedules, die als Grundlage zur Entwicklung eines Maßes genutzt werden. Zum einen gibt es Maße, die die freie Zeit der Maschinen in ihr Maß einbeziehen und zum anderen Maße, die darauf beruhen, inwiefern Operationen verschoben werden können, ohne die Startzeiten anderer Operationen zu beeinflussen. Es gibt eine Vielzahl von Literatur zu Robustheitsmaßen für Scheduling Probleme. An dieser Stelle möchten wir auf die Arbeit von Chtourou und Haouari [21] verweisen. In der Arbeit werden zwölf Robustheitsmaße entwickeltet, die die Zeitspannen messen, um die die Operationen aufgeschoben werden können, ohne den Start einer nachfolgenden Operation zu beeinflussen.

Ein anderer Ansatz zur Generierung eines robusten Schedules ist, die möglichen Realisationen der Unsicherheiten zu begrenzen. Legt man nahe, dass Informationen über die Unsicherheiten bekannt sind, so können diese genutzt werden, um einen robusten Schedule zu generieren. So nutzt Xiong [56] zum Beispiel die Wahrscheinlichkeitsverteilung für das Eintreten von Maschinenausfällen, um einen Schedule zu generieren, der gegen Maschinenausfälle möglichst gut abgesichert ist.

Je nachdem, inwiefern eine Anpassung des Schedules nach Bekanntwerden der Realisationen möglich ist, können wir die Verfahren zu Scheduling unter Unsicherheiten nach Aytug et al. [8] in drei Modelle unterteilen: Predicitve, Predictive-Reactive und Completely Reactive Scheduling.

Beim Predictive Scheduling wird angenommen, dass alle Informationen, sowohl stochastische als auch deterministische, im Voraus bekannt sind. Ziel ist es, einen robusten oder stabilen Schedule zu generieren.

Im Falle von unerwarteten Realisationen wird eine Right-Shift-Strategie angewendet, um die Zulässigkeit des Schedules zu gewährleisten. Beim Right-Shifting bleibt die Reihen-

folge der Operationen erhalten, falls notwendig werden die Startzeiten von Operationen angepasst, indem die Operationen verspätet werden.
Das Predictive-Reactive Modell stellt eine mehr ausgeklügelte, aufwendigere Methoden dar. Es ist nicht nur ein Right-Shift von Operationen möglich, sondern die zu einem Zeitpunkt noch nicht bearbeiteten Operationen können zu dem Zeitpunkt neu gescheduled werden. Werden Operationen neu gescheduled, so bezeichnen wir es als Rescheduling. Ausgehend von einem Basis-Schedule findet Rescheduling statt, wenn unerwartete Realisationen den Schedule in einer nicht tolerierbaren Weise betreffen. Das Rescheduling der verbleibenden Operationen ist ein eigenständiges Optimierungsproblem. Meistens ist es eine schwierige Entscheidung, wann die Rescheduling-Prozedur stattfinden soll. Es muss der zusätzlich benötigte Rechenaufwand gegen die Verbesserungen in der Zielfunktion abgewogen werden.
Das dritte Modell wird als Completely Reactive bezeichnet. Es werden Entscheidungen in Echtzeit getroffen. Meistens werden zu diesem Zweck Dispatching-Rules genutzt. Vollständiges Rescheduling ist in der Regel im Vergleich zum Predictive Scheduling wenig rechenaufwendig und wird typischerweise benutzt, wenn hohe Unsicherheit vorliegt oder wenn Informationen nur für einen beschränkten Zeithorizont zur Verfügung stehen.
Um einen Schedule im Voraus robust zu machen, können zwei Vorgehensweisen unterschieden werden: Buffering and Non-Buffering [56].
Beim Buffering Ansatz werden Freiräume im Schedule erzwungen. Mehrere Studien haben gezeigt, dass das Einfügen von geeigneten Freizeiten im Voraus effektiv ist, um mit Störungen umgehen zu können. Aber es gibt zwei Schwierigkeiten [5]. Es muss entschieden werden, an welcher Stelle die Freiräume eingefügt werden sollen und es muss die Anzahl von Freiräumen beziehungsweise die Dauer bestimmt werden. Eine Anpassung der Startzeiten der Operationen kann im Buffering Ansatz möglich sein. Bei Non-Buffering Ansätzen wird davon ausgegangen, dass der Schedule beim Eintritt von Unsicherheiten angepasst werden kann, das heißt, mindestens ein Right-Shift stattfinden darf. Es wird ein Schedule ermittelt, für den ein möglichst guter Zielfunktionswert erreicht wird. Die Robustheit des Schedules wird anhand eines Robustheitsmaßes gemessen. Das Robustheitsmaß findet Berücksichtigung in der Zielfunktion.

Für das Krankenhaus Scheduling Problem werden wir einen Buffering Ansatz wählen. Eine Anpassung des Schedules bei gegebenen Realisationen der Unsicherheiten betrachten wir nicht. Wir fügen Freiraum in den Schedule ein, um das zusätzliche Einplanen von Behandlungen zu ermöglichen. Behandlungen von Notfallpatienten können immer dann stattfinden, wenn im ermittelten Schedule keine Behandlung stattfindet. Die Robustheit eines Schedules ermitteln wir mittels eines Robustheitsmaßes.

Im folgenden Kapitel 3 möchten wir das allgemeine Konzept unserer Vorgehensweise vorstellen, bevor wir in Kapitel 4 auf die für das Krankenhaus Scheduling Problem spezifischen Elemente eingehen.

Kapitel 3

Robuste Lokale Suche

Eine Robuste Lokale Suche ist eine Heuristik, die das Konzept der Robusten Optimierung mit der Metaheuristik Lokale Suche kombiniert. Sie vereint das Modell der Robusten Optimierung, nur gegen ein gewisses Maß von Unsicherheiten abgesichert zu sein, mit der Fähigkeit einer Lokalen Suche für nicht in polynomieller Zeit lösbare kombinatorische Optimierungsprobleme eine fast optimale Lösung in geringer Zeit zu finden. Eine Lokale Suche wird so angepasst, dass Lösungen nicht nur in Bezug auf ihren Zielfunktionswert sondern auch anhand eines Robustheitsmaßes bewertet werden. Dieses Kapitel 3 soll dazu dienen, unser Konzept einer Robusten Lokalen Suche allgemein zu erläutern.

Wie bereits erwähnt, kombiniert die Robuste Lokale Suche die Robuste Optimierung mit der Metaheuristik Lokale Suche. Daher möchten wir im nächsten Abschnitt 3.1 das Verfahren Lokale Suche vorstellen.

3.1 Lokale Suche

Lokale Suche ist eine Metaheuristik. Als Metaheuristiken werden übergeordnete Algorithmen bezeichnet, die die Lösungssuche eines oder mehrere abhängiger Algorithmen lenken. Sie basieren auf einer Menge von Strategien, die unabhängig vom zugrundeliegenden Problem und den abhängigen gesteuerten Algorithmen sind. Für die Anwendung einer Metaheuristik auf ein bestimmtes Problem müssen einzelne Elemente einer Metaheuristik problemspezifisch umgesetzt werden. Ziel ist es, den Lösungsraum in einer effizienten Weise zu erkunden. Dabei sollen Lösungen gefunden werden, die nah an der optimalen Lösung sind. Eine mittels Metaheuristik ermittelte Lösung kann jedoch beliebig schlecht im Vergleich zu einer optimalen Lösung sein, da es keine Garantien gibt, dass eine gute Lösung gefunden wird.
Metaheuristiken werden insbesondere bei schweren kombinatorischen Optimierungsproblemen eingesetzt, da es für diese in der Regel keine anderen effizienten Algorithmen gibt. Im Allgemeinen sind die Definition und die Implementierung der einzelnen Elemente maßgebend für den Erfolg und die Laufzeit metaheuristischer Verfahren.
Metaheuristiken formen einen bedeutenden Teil der aktuellen globalen Optimierungsalgorithmen. Sie arbeiten bemerkenswert schnell und haben viele Vorteile gegenüber traditionellen, deterministischen Methoden und Algorithmen. Daher wurden sie in fast allen Gebieten der Wissenschaft, Ingenieurswissenschaft und Industrie angewandt [57].
Der Begriff Metaheuristik wurde erstmals von Glover (1986) [29] benutzt und ist eine Kombination zweier griechischer Wörter [19]. Heuristik stammt vom griechischen Verb heuriskein, das so viel bedeutet wie finden. Das vorangestellte meta bedeutet über etwas hinaus. Bevor sich der Begriff Metaheuristik verbreitete, wurden diese auch oft als moderne Heuristiken bezeichnet [46].

Metaheuristiken können in von der Natur inspiriert und in nicht von der Natur inspiriert unterschieden werden [19]. Die von der Natur inspirierten Algorithmen orientieren sich an Vorgängen aus der Natur und übertragen diese auf den Lösungsvorgang des Algorithmus. Zu den von der Natur inspirierten Algorithmen zählt zum Beispiel das Simulated Annealing. Das Simulated Annealing nutzt die Vorgehensweise von Abkühlungsprozessen von Metallen, um die grundlegende Idee dieser Prozesse mit dem Algorithmus nachzubilden. Das Simulated Annealing gehört zur Familie der Lokalen Such Algorithmen.

Lokale Such Algorithmen bezeichnen die Menge der Algorithmen, in denen eine gegebene zulässige Lösung durch eine definierte Menge elementarer Operationen in eine andere zulässige Lösung so transformiert wird, dass diese neue Lösung geringere Kosten besitzt und der Algorithmus diesen Vorgang solange iterativ wiederholt, bis eine hinreichend gute Lösung gefunden oder ein Abbruchkriterium erfüllt wurde. Der Name der Lokalen Suche begründet sich darauf, dass die Transformation einer Lösung mittels elementarer Operationen auch als lokale Veränderung der Lösung angesehen werden kann. Ein Lokaler Such Algorithmus hangelt sich somit von einer Lösung zu einer nächsten Lösung, die durch lokale Veränderungen erreicht werden kann.
Eine Lokale Suche kann auf Optimierungsprobleme angewandt werden, die so formuliert werden können, dass eine Lösung in einer Menge von möglichen Lösungen gefunden werden soll, die ein bestimmtes Kriterium optimiert. Diese Probleme bezeichnet man als kombinatorische Optimierungsprobleme.

3.1.1 Lokale Suche

Gegeben sei ein kombinatorisches Optimierungsproblem durch das Tupel $(\mathfrak{L}, \mathfrak{C})$, wobei $\mathfrak{L}$ die Menge aller möglichen Lösungen und $\mathfrak{C}$ eine Kostenfunktion $\mathfrak{C} : \mathfrak{L} \to \mathbb{R}$ darstellt. Es wird eine Lösung $L \in \mathfrak{L}$ gesucht, für die $\mathfrak{C}(L)$ minimal ist. Für die Lokale Suche muss eine Funktion definiert werden, die jeder Lösung in $\mathfrak{L}$ eine Teilmenge von Lösungen zuordnet. Diese Funktion wird als *Nachbarschaftsfunktion* bezeichnet und sei gegeben durch $N : \mathfrak{L} \to Pot(\mathfrak{L})$. Für ein $L \in \mathfrak{L}$ nennen wir die Menge $N(L)$ *Nachbarschaft* von L. Den Aufbau einer Lokalen Suche stellt Algorithmus 1 dar.

Data : Kombinatorisches Optimierungsproblem $(\mathfrak{L}, \mathfrak{C})$
Berechne Startlösung $L_0 \in \mathfrak{L}$**.**
while *Abbruchkriterium nicht erfüllt.* **do**
  Wähle $L \in N(L_0)$ **mit** $\mathfrak{C}(L)$ **minimal.**
  Setze $L_0 := L$**.**
end

Algorithmus 1 : Aufbau einer Lokalen Suche.

Bei einer Lokalen Suche wird, ausgehend von einer Startlösung L_0, iterativ eine neue Lösung aus der Nachbarschaft der aktuellen Lösung mit den geringsten Kosten ausgewählt. Die neue Lösung bildet die Basis für die nächste Iteration. Der Übergang von einer Lösung zur nächsten wird als *Move* bezeichnet. Die Lokale Suche endet, wenn ein vorgegebenes Abbruchkriterium erfüllt ist.

Gängige Abbruchkriterien sind unter anderem durch eine Zeitschranke, eine maximale Anzahl von Iterationen, eine vorgegebenen Anzahl von Schritten, in denen keine Verbesserung auftritt, oder eine vorgegebene Schranke für die Kosten einer Lösung gegeben. Im letzten Fall bricht der Algorithmus ab, wenn die gefundene Lösung geringere Kosten besitzt als

durch die Schranke gefordert ist. Um einen Lokalen Such Algorithmus zu beenden, können auch verschiedene Abbruchkriterien kombiniert werden.

Werden alle Lösungen einer Nachbarschaft betrachtet und die beste Lösung ausgewählt, so bezeichnet man dieses Vorgehen als *Bestensuche*. Enthält die Nachbarschaft eines Elements viele Lösungen, so kann es sinnvoll sein, nicht die gesamte Nachbarschaft auszuwerten. Wird die Nachbarschaft durchsucht bis die erste verbessernde Lösung gefunden ist, so nennt man dieses Vorgehen *Erstensuche*. Darüber hinaus kann auch die Anzahl der Nachbarschaftselemente, die ausgewertet werden sollen, beschränkt werden und unter diesen die beste Lösung ausgewählt werden. Betrachtet man eine Nachbarschaft, die so lange Elemente auswertet bis f verbessernde Lösungen gefunden sind, und wählt unter diesen die beste Lösung aus, so bezeichnet man dieses Vorgehen als *f-Erstensuche*.

Eine Schwäche der Lokalen Suche ist, dass der Algorithmus in einem lokalen Optimum endet, egal wie gut oder schlecht dieses lokale Optimum ist. Sind zum Beispiel zwei lokale Optima gegeben, die in der Nachbarschaft des jeweils anderen liegen, die gleichen Kosten besitzen und alle anderen Lösungen aus den beiden Nachbarschaften höhere Kosten besitzen, dann tendiert ein Lokaler Such Algorithmus dazu, abwechselnd die beiden Lösungen auszuwählen. Man spricht in diesem Fall davon, dass der Algorithmus *kreiselt*. Hat der Algorithmus im Laufe der Iterationen mit Kreiseln begonnen, so kann ein globales Optimum nicht mehr erreicht werden.
Um dieses sogenannte Kreiseln zwischen Lösungen zu vermeiden, sind mehrere Ansätze entwickelt worden. Ein Ansatz ist es, Lösungen auch mit höheren Kosten auszuwählen, insofern sie einen zufälligen Akzeptanztest bestehen. Dieser Ansatz wird mit *Simulated Annealing* bezeichnet. Mit Hilfe des Zufalls wird versucht, lokale Optima zu verlassen. Ein weiterer Ansatz ist, Informationen über den bisherigen Suchprozess zu speichern. Erweitert man den oben beschriebenen Lokale Suche Algorithmus durch eine Liste, in der die Lösungen der letzten Iterationen abgespeichert sind, und sperrt alle Lösungen dieser Liste für die Auswahl einer neuen Lösung, so wird ein solcher Algorithmus als *Tabu Search* Algorithmus bezeichnet.

Wir möchten den Ansatz des Tabu Search für unsere Robuste Lokale Suche nutzen. Daher möchten wir auf diesen Ansatz der Lokalen Suche im nächsten Abschnitt 3.1.2 genauer eingehen.

3.1.2 Tabu Search

Wie bereits dargestellt, ist Tabu Search eine Variante der Lokalen Suche, bei der, um Kreiseln zu vermeiden und um zu den guten Regionen des Lösungsraums zu gelangen, ein Teil des Suchablaufs mittels einer Liste in Erinnerung behalten wird und so in der weiteren Suche verwendet werden kann. Diese Liste kann zum Beispiel die zuletzt besuchten Lösungen abspeichern oder aber Eigenschaften von den zuletzt besuchten Lösungen enthalten. Die Lösungen beziehungsweise Eigenschaften auf der Liste dürfen für eine bestimmte Anzahl nachfolgender Iterationen nicht ausgewählt werden. Daher bezeichnet man diese Liste auch als *Tabuliste*.

Tabu Search ist die meist benutzte Metaheuristik für kombinatorische Optimierungsprobleme. Die Grundidee wurde von Glover [29] entwickelt [19]. Die Theorie und Anwendung von Tabu Search wurde durch Glover und Laguna [31] und bei Hertz et al. [1] unter anderem durch die Einführung von Anspruchskriterien erheblich verbessert [44]. Durch ein

Anspruchskriterium kann der Tabu-Status von Tabu-Restriktionen bei Erfüllen des Kriteriums aufgehoben werden.

Der Aufbau eines Tabu Search Algorithmus gleicht dem eines Lokalen Such Algorithmus. Ergänzt wird der Algorithmus allerdings um eine Tabuliste. In Algorithmus 2 wird der Aufbau eines Tabu Search Algorithmus beschrieben, in dem die Tabuliste eine Eigenschaft der Lösungen speichert. Die Liste verbietet dem Algorithmus eine Lösung auszuwählen, die die gleiche Ausprägung der Eigenschaft besitzt, wie eine der letzten t_{max} besuchten Lösungen. Wird in einer Iteration eine Lösung ausgewählt, so wird eine Eigenschaft dieser Lösung der Tabuliste hinzugefügt und der älteste Eintrag aus der Liste entfernt.

Data : Kombinatorisches Optimierungsproblem $(\mathfrak{L}, \mathfrak{C})$**, Tabuliste** T**,**
Eigenschaft $\mathfrak{E}$ **einer Lösung**
Berechne Startlösung $L_0 \in \mathfrak{L}$**.**
for $t = 1, \ldots, t_{max}$ **do**
| $T[t] = \mathfrak{E}(L_0)$**;**
end
while *Stoppkriterium nicht erfüllt* **do**
| **Wähle** $L \in N(L_0)$ **mit** $\mathfrak{E}(L) \notin T$ **und** $\mathfrak{C}(L)$ **minimal.**
| **for** $t = 2, \ldots, t_{max}$ **do**
| | $T[t] = T[t-1]$;
| **end**
| $T[1] = \mathfrak{E}(L)$**;**
| **Setze** $L_0 := L$**.**
end

Algorithmus 2 : Aufbau eines Tabu Search Algorithmus.

Die Größe der Tabuliste ist beim Algorithmus von immenser Bedeutung. Wird die Größe t_{max} der Tabuliste klein gewählt, so kann es sein, dass der Algorithmus in einem lokalen Optima endet. Wird die Größe der Tabuliste groß gewählt, so kann dies den Algorithmus behindern. Es kann vorkommen, dass zu einer Lösung keine gültigen Nachbarn existieren, da alle möglichen Nachbarn durch die Tabuliste verboten werden. Oft werden auf der Tabuliste sogenannte *Undo-Moves* gespeichert. Eine Undo-Move eines betrachteten Moves ist ein Move, der die Veränderungen des Betrachteten rückgängig macht. Nachdem ein Move durchgeführt wurde, wird der zugehörige Undo-Move der Tabuliste hinzugefügt.

Es gibt keine allgemeinen Regeln, mit der die Größe einer Tabuliste festgelegt werden kann. In den meisten Fällen wird die Größe der Tabuliste daher experimentell ermittelt.

Jede Implementierung einer Lokalen Suche und damit auch eines Tabu Search ist problemspezifisch und benötigt spezifische Definitionen der grundlegenden Elemente wie Nachbarschaft, Abbruchkriterium, Startlösung und Werte für die Parameter wie die Länge der Tabuliste. Einige grundlegende Elemente werden in Kapitel 4 problemspezifisch für das Krankenhaus Scheduling Problem umgesetzt.

Nachdem in diesem Abschnitt die Metaheuristik Lokale Suche vorgestellt wurde, möchten wir im nächsten Abschnitt 3.2 erläutern, wie diese genutzt werden soll, um robuste Lösungen zu generieren.

3.2 Konzept einer Robusten Lokalen Suche

Das Verfahren der Robusten Lokalen Suche richtete sich an kombinatorische Optimierungsprobleme, bei denen ein Teil der Eingabedaten mit Unsicherheiten behaftet ist. Es soll eine Lösung ermittelt werden, die gegen die Unsicherheiten in einem gewissen Ausmaß abgesichert ist.

Gegeben sei ein kombinatorisches Optimierungsproblem durch das Tupel $(\mathfrak{L}, \mathfrak{C})$, wobei $\mathfrak{L}$ die Menge aller möglichen Lösungen und $\mathfrak{C}$ eine Kostenfunktion $\mathfrak{C} : \mathfrak{L} \to \mathbb{R}$ darstellt. Ein Teil der Eingabedaten sei mit Unsicherheiten behaftet. Ziel einer Robusten Lokalen Suche ist es, eine Lösung $L \in \mathfrak{L}$ zu ermitteln, die möglichst geringe Kosten besitzt und möglichst gut gegen potentielle Realisationen der Unsicherheiten abgesichert ist.

Für die Robuste Lokale Suche nehmen wir an, dass Informationen über die Unsicherheiten bekannt sind, so dass eine Menge $\mathfrak{S}$ möglicher Realisationen erzeugt werden kann. Die durch die Robuste Lokale Suche ermittelte Lösung soll gegen die Menge $\mathfrak{S}$ in einem gewissen Maß abgesichert sein. Die Menge $\mathfrak{S}$ möglicher Realisationen bezeichnen wir im Folgenden auch als *Szenarienmenge* und eine mögliche Realisation $\mathfrak{s} \in \mathfrak{S}$ als *Szenario*.

Wir möchten in der Robusten Lokalen Suche eine Lösung gegen alle Unsicherheiten aus einer Szenarienmenge $\mathfrak{S}' \subseteq \mathfrak{S}$ zu einem gewissen Maß absichern. Um das Ausmaß der Absicherung einer Lösung messen zu können, definieren wir ein Funktion $\mathfrak{R}$.

Definition 3.2.1. *Sei $\mathfrak{L}$ die Menge aller möglichen Lösungen. Dann bezeichnen wir eine Funktion $\mathfrak{R}$ mit*

$$\mathfrak{R} \ : \ \mathfrak{L} \times Pot(\mathfrak{S}) \to [0, 1],$$

die jeder möglichen Lösung $L \in \mathfrak{L}$ und jeder Szenarienteilmenge $\mathfrak{S}' \subseteq \mathfrak{S}$ einen Wert aus dem Intervall $[0, 1]$ zuordnet, als Robustheitsmaß.

Die genaue Definition eines Robustheitsmaßes ist abhängig vom zugrundeliegenden Problem.

Bezeichnung 3.2.2. *Den Wert des Robustheitsmaßes $\mathfrak{R}(L, \mathfrak{S}')$ für eine Lösung $L \in \mathfrak{L}$ und eine Szenarienteilmenge $\mathfrak{S}' \subseteq \mathfrak{S}$ nennen wir Robustheitswert der Lösung L gegenüber der Szenarienmenge $\mathfrak{S}'$.*

Je größer der Robustheitswert einer Lösung ist, umso besser ist die Lösung gegen die Szenarien aus der Szenarienmenge $\mathfrak{S}'$ abgesichert. Besitzt eine Lösung L_1 einen größeren Robustheitswert als eine Lösung L_2 gegenüber einer Szenarienmenge $\mathfrak{S}'$, so bezeichnen wir die Lösung L_1 als *robuster* als die Lösung L_2 gegenüber der Szenarienmenge $\mathfrak{S}'$.

Die Robuste Optimierung zeichnet sich dadurch aus, dass mit zunehmendem Maß an Robustheit die Kosten einer Lösung steigen. Beide Kriterien, sowohl Kostenwert als auch Robustheitswert, sollen simultan optimiert werden, wobei sich die beiden Kriterien widersprechen. Daher muss ein Trade-Off zwischen den Kriterien getroffen werden. Werden mehrere Kriterien bei einer Optimierung berücksichtigt, so spricht man von einer Multi-objective Optimierung. Hat man ein Problem der Robusten Optimierung gegeben, so könnte der

Zusammenhang zwischen Kostenwert und Robustheitswert ähnlich wie in Abbildung 3.1 aussehen.

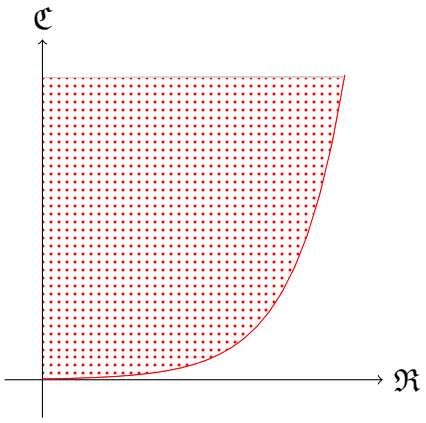


Abbildung 3.1: Zusammenhang von Robustheits- und Kostenwert.

In Abbildung 3.1 sollen beispielsweise die $(\mathfrak{R}, \mathfrak{C})$-Kombinationen aller möglichen Lösungen eines Minimierungsproblems dargestellt sein. Es ist zu sehen, dass für ein Minimierungsproblem mit zunehmendem Robustheitswert der Kostenwert einer Lösung steigt und mit abnehmendem Kostenwertwert der Robustheitwert einer Lösung abnimmt.
Ohne die Präferenzen des Entscheiders zu kennen, kann nicht bestimmt werden, welche $(\mathfrak{R}, \mathfrak{C})$-Kombinationen für einen Entscheider optimale Lösungen repräsentieren, aber die Menge der in Frage kommenden Lösungen kann anhand ihrer $(\mathfrak{R}, \mathfrak{C})$-Kombination eingeschränkt werden.

Definition 3.2.3. *Eine* $(\mathfrak{R}, \mathfrak{C})$ *Kombination heißt effizient, wenn keine Lösung* $L \in \mathfrak{L}$ *existiert, so dass* $\mathfrak{C}(L) < \mathfrak{C}$ *oder* $\mathfrak{R}(L) > \mathfrak{R}$ *gilt.*

Bei einem Minimierungsproblem würde ein Entscheider bei gleichem Robustheitswert zweier Lösungen immer die Lösung mit dem geringeren Kostenwert bevorzugen. Besitzen zwei Lösungen den gleichen Kostenwert, so zieht der Entscheider die Lösung mit dem größeren Robustheitswert vor.

Bezeichnung 3.2.4. *Die Menge aller effizienten* $(\mathfrak{R}, \mathfrak{C})$*-Kombinationen bezeichnen wir als Effizienzmenge.*

Bezeichnung 3.2.5. *Alle Lösungen, die einer* $(\mathfrak{R}, \mathfrak{C})$*-Kombination aus der Effizienzmenge entsprechen, heißen effiziente Lösungen.*

Ziel einer Robusten Lokalen Suche ist es, eine Lösung zu ermitteln, die einer den Präferenzen des Entscheiders am meisten entsprechenden effizienten Lösung möglichst nahe kommt. Um dieses Ziel zu erreichen, verfolgen wir zwei Ansätze. Im ersten Fall gibt der Entscheider eine Untergrenze für den Robustheitswert vor. Im zweiten Fall muss der Entscheider einen Trade-Off zwischen Zielfunktionswert und Robustheitswert festlegen.

Robuste Lokale Suche

Ziel der Robusten Lokalen Suche ist es, für ein gegebenes kombinatorisches Optimierungsproblem, bei dem ein Teil der Eingabedaten mit Unsicherheiten behaftet ist, eine Lösung $L \in \mathfrak{L}$ zu ermitteln, die einen im Sinne des Entscheiders möglichst guten Trade-Off zwischen den Kosten einer Lösung und dem Level der Absicherung gegen die unsicheren Daten darstellt.

Gegeben sei ein kombinatorisches Optimierungsproblem durch das Tupel $(\mathfrak{L}, \mathfrak{C})$, wobei ein Teil der Eingabedaten mit Unsicherheiten behaftet ist. Über die Unsicherheiten sind Informationen in dem Maße bekannt, dass potentielle Realisationen der Unsicherheiten entweder in Form einer Szenarienmenge gegeben sind oder eine Szenarienmenge mit potentiellen Realisationen erzeugt werden kann. Weiter haben wir ein Robustheitsmaß $\mathfrak{R}$ gegeben, mit dem ein Robustheitswert einer Lösung in Bezug auf eine Szenarienmenge berechnet werden kann. Wie bei einer Lokalen Suche definieren wir eine Nachbarschaftsfunktion N. Die Nachbarschaftsfunktion $N : \mathfrak{L} \to Pot(\mathfrak{L})$ ordnet jeder Lösung $L \in \mathfrak{L}$ eine *Nachbarschaft* $N(L) \subseteq \mathfrak{L}$ zu. Eine Lösung $L' \in N(L)$ wird *Nachbar* von L genannt.

Das Vorgehen einer Robusten Lokalen Suche ist ähnlich dem einer Lokalen Suche. Wir möchten eine Lokale Suche so anpassen, dass Lösungen nicht nur anhand ihres Kostenwertes sondern auch in Bezug auf ihren Robustheitswert ausgewählt werden. Um das Robustheitsmaß in die Lokale Suche zu integrieren, verfolgen wir zwei Ansätze. Zum einen betrachten wir eine Lokale Suche und wählen nur dann eine Lösungen aus der Nachbarschaft, wenn ihr Robustheitsmaß für eine gegebene Szenarienmenge eine bestimmte Schranke erfüllt. Diesen Ansatz nennen wir Robustheit mittels Akzeptor und stellen ihn in Absatz 3.2.1 vor. Zum anderen verändern wir den Kostenwert, der grundlegend für die Auswahl einer Lösung aus einer Nachbarschaft ist. Der zweit genannte Ansatz wird in Abschnitt 3.2.2 erörtert.

3.2.1 Robustheit mittels Akzeptoren

In diesem Ansatz wird das Robustheitsmaß in die Lokale Suche integriert, indem beim Suchprozess nur Lösungen zugelassen werden, deren Robustheitswert eine vorgegebene Schranke übersteigt. Diese Schranke muss vom Entscheider als der minimale Robustheitswert festgelegt werden, den eine Lösung nach seinen Präferenzen besitzen soll. Im Folgenden werde dieser Wert *Robustheitsschranke* genannt und mit a bezeichnet. Da die Robustheitsschranke eine Schranke für den Robustheitswert einer Lösung sein soll und der Robustheitswert auf das Intervall $[0, 1]$ normiert ist, sollte die Robustheitschranke $a \in [0, 1]$ gewählt werden.

Definition 3.2.6. *Sei $\mathfrak{R}$ ein Robustheitsmaß und $\mathfrak{S}$ eine Szenarienmenge. Ein Akzeptor ist eine Funktion $\mathfrak{A}$ mit*

$$\mathfrak{A} \ : \ [0,1] \times [0,1] \to \{0,1\},$$

wobei

$$(a, \mathfrak{R}(L, \mathfrak{S})) \longmapsto \begin{cases} 0 & \textit{falls } \alpha > \mathfrak{R}(L, \mathfrak{S}), \\ 1 & \textit{sonst} \end{cases}$$

für eine Lösung $L \in \mathfrak{L}$ und eine Robustheitsschranke a.

Für eine gegebene Lösung L, eine Szenarienmenge $\mathfrak{S}$ und ein Robustheitsmaß $\mathfrak{R}$ wird mittels Akzeptor bestimmt, ob die Lösung L aus der Nachbarschaft ausgewählt werden kann. Eine Lösung kann ausgewählt werden, wenn

$$\mathfrak{A}(a, \mathfrak{R}(L, \mathfrak{S})) = 1$$

gilt. Die Lösung L wird in diesem Fall *akzeptiert*. Eine Lösung $L \in \mathfrak{L}$ kann nicht ausgewählt werden, wenn

$$\mathfrak{A}(a, \mathfrak{R}(L, \mathfrak{S})) = 0$$

gilt. In diesem Fall spricht man davon, dass die Lösung *verworfen* wird. Der Akzeptor dient dazu, dass nur Lösungen ausgewählt werden, deren Robustheitswert oberhalb der vom Entscheider bestimmten Schranke liegt.

In der Robusten Lokalen Suche mit Akzeptor wird ausgehend von einer Startlösung iterativ eine Lösung mit dem besten Kostenwert aus der Nachbarschaft des Vorgängers gewählt, insofern die Lösung von dem Akzeptor akzeptiert wird. Gibt es keine akzeptierte Lösung, so wird die Lösung mit dem besten Robustheitswert gewählt. Ist ein Robustheitsmaß $\mathfrak{R}$, eine Robustheitsschranke a und eine Szenarienmenge $\mathfrak{S}$ gegeben, so ergibt sich die Robuste Lokale Suche mit Akzeptor wie in Algorithmus 3 dargestellt.

Data : Kombinatorisches Optimierungsproblem $(\mathfrak{L}, \mathfrak{C})$**, Tabuliste** T**,** a **,**$\mathfrak{R}$**,** $\mathfrak{S}$
Berechne Startlösung $L_0 \in \mathfrak{L}$**.**
for $t = 1, \ldots, t_{max}$ **do**
  $T[t] = L_0$**;**
end
while *Abbruchkriterium nicht erfüllt.* **do**
  Wähle $L \in N(L_0)$ **mit** $\mathfrak{A}(a, \mathfrak{R}(L, \mathfrak{S})) > 0$ **und** $\mathfrak{C}(L)$ **minimal.**
  for $t = 2, \ldots, t_{max}$ **do**
    $T[t] = T[t-1]$;
  end
  $T[1] = L$**;**
  Setze $L_0 := L$**.**
end

Algorithmus 3 : Tabu Search mit Akzeptor.

Die Iteration bricht ab, sobald ein vorgegebenes Abbruchkriterium erfüllt ist. Mögliche Abbruchkriterien wurden bereits in Abschnitt 3.1.1 vorgestellt.

Je größer die Robustheitsschranke a gewählt wird, desto robuster müssen die Lösungen sein, um ausgewählt werden zu können. Im Fall $a = 0$ akzeptiert der Akzeptor jede Lösung einer Nachbarschaft. Somit gleicht in diesem Fall die Robuste Lokale Suche einer Lokalen Suche. Für $a = 1$ kommen nur Lösungen in Frage, die zu hundert Prozent gegen die gegebene Szenarienmenge abgesichert sind. Es muss aufgepasst werden, dass falls die Robustheitsschranke a groß gewählt wird, es vorkommen kann, dass keine Lösung zugelassen wird.

Durch den Akzeptor wird der Bereich, der bei der Robusten Lokalen Suche infrage kommenden Lösungen, eingeschränkt. Dieses ist in Abbildung 3.2 beispielhaft für den Zusammenhang vom Robustheits- und Kostenwert aus Abbildung 3.1 dargestellt.

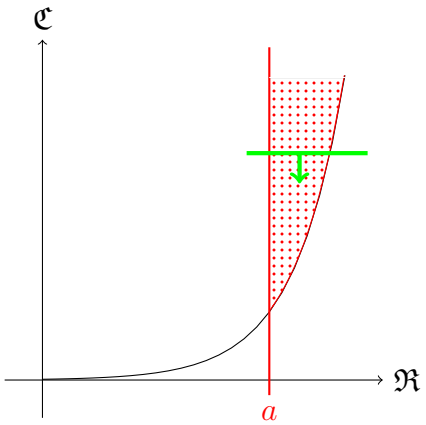


Abbildung 3.2: Zusammenhang von Robustheits- und Kostenwert mit Akzeptor.

In Abbildung 3.2 sind alle $(\mathfrak{R}, \mathfrak{C})$-Kombinationen eingezeichnet, für die $\mathfrak{R} > a$ gilt. Ziel ist es, eine Lösung mit bestem Kostenwert unter den entsprechenden Lösungen zu finden.

Mittels des Akzeptors wird für eine gegebene Robustheitsschranke versucht, eine Lösung zu ermitteln, deren Robustheitswert die Schranke übersteigt und die einen bestmöglichen Kostenwert besitzt. Auf Grund des gegenläufigen Zusammenhangs zwischen Robustheitswert und Kostenwert, wird der Robustheitswert in der Regel die Robustheitsschranke annehmen oder die Robustheitsschranke weitestgehend annähern. Führt man das Verfahren für hinreichend viele verschiedene Robustheitsschranken aus, so lässt sich eine Kurve annähern, die den Zusammenhang zwischen Robustheitswert und Kostenwertwert für das gegebene Problem zu der betrachteten Szenarienmenge widerspiegelt.

Ein Vorteil dieses Ansatzes ist, dass die Methode einfach anwendbar ist. Der Entscheider muss sich nicht die Frage stellen, wie viel Verbesserung im Kostenwert ihm ein Verlust an Robustheit bringen muss, damit eine Lösung seinen Präferenzen nach als besser angesehen wird. Die starre Robustheitsschranke kann jedoch auch als Nachteil dieser Vorgehensweise gesehen werden. Es kann sein, dass eine kleine Veränderung der Schranke bereits zu großen Änderungen in Bezug auf den Kostenwert führt.

In welchem Maß sich die Veränderung der Robustheitsschranke auf den Kostenwert auswirkt, kann durch den *Price of Robustness* berechnet werden.

Definition 3.2.7. *Der Price of Robustness ist der prozentuale Anstieg des Kostenwertes für eine Robustheitsschranke $a \in [0, 1]$ im Vergleich zum Kostenwert bei Robustheitsschranke $a = 0$ bei Betrachtung derselben Szenarienmenge $\mathfrak{S}$.*

Einen Price of Robustness haben wir bereits in Abschnitt 2.2.2 im Zusammenhang mit der Robusten Optimierung kennengelernt. Dort musste die Lösung zu hundert Prozent gegen eine Menge von Szenarien abgesichert sein. Der Price of Robustness variierte mit der Menge der Szenarien. Je größer die Abweichung von gegebenen nominalen Daten in den Szenarien erlaubt war, desto größer war der Price of Robustness. In unserem Modell eines Optimierungsproblems unter Unsicherheiten haben wir keine nominalen Daten gegeben. Wir betrachten den Price of Robustness daher für eine feste Szenarienmenge. Der Price of Robustness variiert in unserem Fall mit dem Wert für die Robustheitsschranke.

Während in dem Ansatz einer Robusten Lokalen Suche dieses Abschnittes eine möglichst gute Lösung in Bezug auf den Kostenwert bei einer gegebenen Robustheitsschranke ermittelt werden sollte und dazu nur geringe Kenntnis über die Präferenzen des Entscheiders bekannt sein musste, betrachten wir im nächsten Abschnitt 3.2.2 einen zweiten Ansatz, für den genauere Kenntnis über die Präferenzen des Entscheiders benötigt wird.

3.2.2 Robustheit durch Konvexkombination von Kostenfunktion und Robustheitswert

In diesem Ansatz wird das Robustheitsmaß in die Lokale Suche integriert, indem Lösungen anhand einer Konvexkombination aus dem Robustheitswert und dem Kostenwert ausgewählt werden. Der Entscheider muss dazu bestimmen, wie seinen Präferenzen nach, das Verhältnis zwischen Robustheitswert und Kostenwert aussehen sollte.

Gegeben sei ein kombinatorisches Optimierungsproblem $(\mathfrak{L}, \mathfrak{C})$, wobei ein Teil der Eingabedaten mit Unsicherheiten behaftet ist. Sei weiter $\mathfrak{R}$ ein Robustheitsmaß. Dann definieren wir eine neue Kostenfunktion $\mathfrak{C}'$, die aus einer Konvexkombination der beiden Funktionen $\mathfrak{C}$ und $\mathfrak{R}$ gebildet wird. Damit die beiden Werte bei gleicher Gewichtung in etwa gleich in die Kostenfunktion $\mathfrak{C}'$ zählen, ermitteln wir einen Richtwert $\mathfrak{K}$ für den Kostenwert einer Lösung. Als Richtwert $\mathfrak{K}$ kann eine obere Schranke für das gegebene Problem genutzt werden. Den Richtwert multiplizieren wir mit dem Robustheitsmaß.

Sei $\mathfrak{K}$ ein Richtwert für die Kostenfunktion $\mathfrak{C}$. Dann definieren wir eine neue Kostenfunktion $\mathfrak{C}'$ für unsere Robuste Lokale Suche durch

$$\mathfrak{C}'(L) := (1 - d)\ \mathfrak{C}(L) + d\ \mathfrak{R}(L, \mathfrak{S})\ \mathfrak{K}, \tag{3.2.1}$$

wobei $d \in [0, 1]$ durch die Präferenzen des Entscheiders gegeben ist. Den Wert d bezeichnen wir im Folgenden auch als *Robustheitsfaktor* und die Kostenfunktion $\mathfrak{C}'$ als *kombinierte* Kostenfunktion.

Der Robustheitsfaktor d dient zur Regulierung des Verhältnisses von Robustheitswert und Kostenwert. Der Robustheitsfaktor muss den Präferenzen des Entscheiders entsprechend festgelegt werden.

In der Robusten Lokalen Suche mit Konvexkombination der beiden Kriterien wird ausgehend von einer Startlösung iterativ eine Lösung mit dem besten Kostenwert $\mathfrak{C}'$ aus der Nachbarschaft des Vorgängers gewählt. Der Algorithmus für die Robuste Lokale Suche ist durch Algorithmus 4 gegeben.

Data : Kombinatorisches Optimierungsproblem $(\mathfrak{L}, \mathfrak{C})$**, Tabuliste** T**,** d **,**$\mathfrak{R}$**,** $\mathfrak{S}$**.**
Berechne Startlösung $L_0 \in \mathfrak{L}$**.**
for $t = 1, \ldots, t_{max}$ **do**
  $T[t] = L_0$**;**
end
while *Abbruchkriterium nicht erfüllt.* **do**
  Wähle $L \in N(L_0)$ **mit** $\mathfrak{C}'(L)$ **minimal.**
  for $t = 2, \ldots, t_{max}$ **do**
    $T[t] = T[t-1]$;
  end
  $T[1] = L$**;**
  Setze $L_0 := L$**.**
end

Algorithmus 4 : Tabu Search mit gemischter Kostenfunktion.

Die Iteration bricht ab, sobald ein vorgegebenes Abbruchkriterium erfüllt ist. Mögliche Abbruchkriterien wurden bereits in Abschnitt 3.1.1 vorgestellt.

Wählt man den Robustheitsfaktor $d = 0$, so entspricht die Robuste Lokale Suche einer Lokalen Suche. Wird der Robustheitsfaktor $d = 1$ gewählt, so spielt die ursprüngliche Kostenfunktion des kombinatorischen Optimierungsproblems keine Rolle mehr, sondern eine Lösung wird allein anhand ihres Robustheitswertes ausgewählt.

Der Robustheitsfaktor d muss vom Entscheider nach seinen persönlichen Präferenzen gewählt werden. Es kann vorkommen, dass mehrere Lösungen den gleichen Kostenwert unter der Kostenfunktion $\mathfrak{C}'$ besitzen. Zwischen diesen Lösungen ist der Entscheider indifferent. Da es sich bei der Funktion $\mathfrak{C}'$ um eine Konvexkombination von Robustheitswert und Kostenwert handelt, können für den Entscheider gleichwertige Lösungen mittels einer Geraden in ein $(\mathfrak{R}, \mathfrak{C})$-Diagramm eingezeichnet werden.

Definition 3.2.8. *Eine Gerade im* $(\mathfrak{R}, \mathfrak{C})$*-Diagramm, auf denen alle Kombinationen unter der Kostenfunktion* $\mathfrak{C}'$ *gleichwertig sind, wird Isopräferenzlinie genannt.*

Die Steigung der Geraden ist abhängig vom Robustheitsfaktor d. In Abbildung 3.3 sind beispielhaft zwei Isopräferenzlinien eingezeichnet.

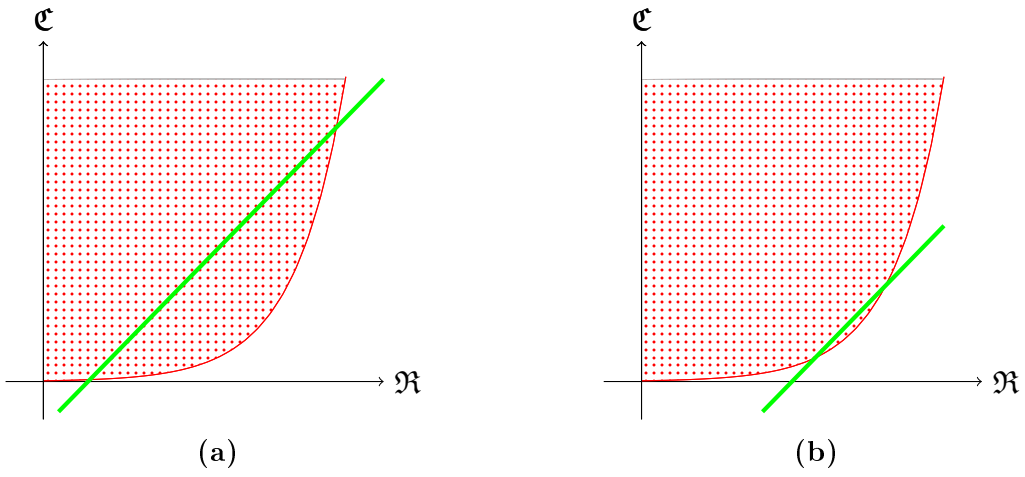


Abbildung 3.3: Zusammenhang von Robustheits- und Kostenwert mit Isopräferenzlinie.

Die Abbildung 3.3 zeigt eine Isopräferenzlinie im $(\mathfrak{R}, \mathfrak{C})$-Diagramm. Während die Steigung der Isopräferenzlinie durch den Robustheitsfaktor d festgelegt ist, kann die Isopräferenzlinie entlang der $\mathfrak{R}$-Achse verschoben werden. Eine Isopräferenzlinie repräsentiert alle Lösungen, die denselben Kostenwert $\mathfrak{C}'$ besitzen. Verschiebt man die Isopräferenzlinie entlang der $\mathfrak{R}$-Achse so, dass der Robustheitswert des $\mathfrak{R}$-Achsen-Abschnitts zunimmt, so nimmt der Kostenfunktionswert $\mathfrak{C}'$ zu. Die Lösungen, deren $(\mathfrak{R}, \mathfrak{C})$-Kombination eher der Isopräferenzlinie aus Abbildung 3.3b entsprechen, besitzen somit einen höheren Kostenwert $\mathfrak{C}'$ als die Lösungen, deren $(\mathfrak{R}, \mathfrak{C})$-Kombination näher an der Isopräferenzlinie aus Abbildung 3.3a sind.

Ziel der Robusten Lokalen Suche mit Konvexkombination von Robustheitsmaß und Kostenfunktion ist es, eine Lösung zu ermitteln, deren $(\mathfrak{R}, \mathfrak{C})$-Kombination einer Isopräferenzlinie nahe kommt, deren $\mathfrak{R}$-Achsen-Abschnitt möglichst groß ist. Eine solche Lösung entspricht den Präferenzen des Entscheiders am meisten.

In den beiden präsentierten Ansätzen für eine Robuste Lokale Suche, wird der Lösungsraum in unterschiedlicher Weise durchsucht, um zu einer präferierten Lösung zu gelangen. In Kapitel 5 werden wir untersuchen, ob eine der beiden Methoden zu besseren $(\mathfrak{R}, \mathfrak{C})$-Kombinationen führt.

Kapitel 4

Robuste Lokale Suche für das Krankenhaus Scheduling Problem

Das Krankenhaus Scheduling Problem unter Unsicherheiten haben wir bereits in Abschnitt 1.2 eingeführt. Es erweitert das Krankenhaus Scheduling Problem um den Fall, dass Notfallpatienten zu unbekannten Zeitpunkten eintreffen und im Laufe des Tages behandelt werden müssen. Um eine Robuste Lokale Suche auf das Krankenhaus Scheduling Problem unter Unsicherheiten anwenden zu können, müssen die problemspezifischen Elemente einer Robusten Lokalen Suche für das Problem definiert werden. Zu den problemspezifischen Elementen zählt zum einen die Definition einer geeigneten Nachbarschaft und zum anderen muss ein geeignetes Robustheitsmaß definiert werden. Weiter muss eine zulässige Startlösung generiert werden.

In Abschnitt 4.1 möchten wir zunächst die problemspezifischen Elemente einer Lokalen Suche für das Krankenhaus Scheduling Problem beschreiben, bevor wir in Abschnitt 4.2 die Lokale Suche anpassen und sie zu einer Robusten Lokalen Suche für das Krankenhaus Scheduling Problem erweitern.

4.1 Lokale Suche für das Krankenhaus Scheduling Problem

In diesem Abschnitt 4.1 möchten wir eine Lokale Suche für das Krankenhaus Scheduling Problem spezifizieren. Dazu müssen die grundlegenden Elemente einer Lokalen Suche problemspezifisch umgesetzt werden. Zu den grundlegenden Elementen zählt unter anderem die Generierung einer Startlösung sowie das Definieren einer geeigneten Nachbarschaft.

Als erstes stellen wir in Abschnitt 4.1.1 eine alternative Repräsentation einer Lösung zum Gantt Chart vor. Auf Basis dieser Repräsentation möchten wir in Abschnitt 4.1.2 einen Algorithmus vorstellen, mit dem eine zulässige Startlösung generiert werden kann. Im Anschluss definieren wir in Abschnitt 4.1.3 eine Nachbarschaft zu einer gegebenen Lösung.

4.1.1 Lösungsrepräsentation für das Krankenhaus Scheduling Problem

Eine Lösung für das Krankenhaus Scheduling Problem ist durch einen zulässigen Schedule gegeben. Ein zulässiger Schedule ordnet jeder Behandlung eine Startzeit zu, so dass die Bedingungen (I)-(IV) aus Abschnitt 1.1.1 erfüllt sind.
Eine ähnliche Definition für einen zulässigen Schedule existiert für das Job Shop Scheduling Problem. In Abschnitt 1.1.2 haben wir gesehen, dass für jedes Job Shop Scheduling Problem

ein optimaler Schedule existiert, der mittels eines gerichteten Disjunctive Graph Modells dargestellt werden kann. Dies beruht auf der Tatsache, dass es immer eine optimale Lösung gibt, in der keine Behandlung früher starten kann, ohne die Reihenfolge der Operationen auf einer Maschine zu ändern [54]. Einen solchen Schedule nennt man *linksgerichtet*.

Definition 4.1.1. *Ein zulässiger Schedule heißt linksgerichtet, wenn keine Operation vorverlegt werden kann, ohne die Reihenfolge der Operationen auf einer Maschine zu ändern.*

In Abschnitt 1.1.2 haben wir eine Maschinenreihenfolge als eine Permutation $\pi_k = (\pi_k(1), \ldots, \pi_k(y_k))$ definiert, wobei $\pi_k(i)$ die Operation bezeichnet, die als i-tes auf Maschine $m_k \in M$ bearbeitet wird. Eine Permutation $\pi_k(i)$ kann als eine bijektive Funktion mit

$$\pi_k : \{1, \ldots, m_k\} \longrightarrow M_k$$

interpretiert werden. Mittels der Umkehrfunktion π_k^{-1} kann zu einer Operation $o \in M_k$ ermittelt werden, als wievieltes die Operation in der betrachteten Reihenfolge auf der Maschine stattfindet.
Für gegebene zulässige Reihenfolgen $\pi_1, \ldots, \pi_m$ der Operationen auf den Maschinen, kann ein linksgerichteter Schedule erzeugt werden, in dem jeder Operation die frühestmögliche Startzeit mittels der Formel

$$S_t(o_{i,j}) = \max\{S_t(o_{i,j-1}), S_t(\pi_l\left(\pi_l^{-1}(o_{i,j}) - 1\right))\} \tag{4.1.1}$$

zugeordnet wird, wobei l so gewählt ist, dass $\mathfrak{M}(o_{i,j}) = m_l$ gilt, und die Startzeiten gemäß

$$S_t(o) = \begin{cases} 0, & \text{falls } o = \pi_j(1) \text{ für ein } j = 1, \ldots, m, \\ \inf, & \text{sonst,} \end{cases}$$

initialisiert werden.

Für ein Scheduling Problem existiert immer ein optimaler Schedule, der linksgerichtet ist, wenn durch das Optimierungskriterium impliziert wird, dass es optimal ist, einen Job so früh wie möglich zu starten beziehungsweise abzuschließen. Dies ist gegeben, wenn das Optimierungskriterium eine mit den Fertigstellungsdaten der Jobs steigende Funktion ist [39]. In diesem Fall bezeichnet man das Optimierungskriterium als *regulär*.

Definition 4.1.2. *Ein Optimierungskriterium heißt regulär, wenn es eine steigende Funktion in Bezug auf die Fertigstellungsdaten der Jobs ist.*

Da das Optimierungskriterium des Krankenhaus Scheduling Problems die Minimierung der Summe der Fertigstellungsdaten ist, haben wir beim Krankenhaus Scheduling Problem ein reguläres Optimierungskriterium gegeben.

Definition 4.1.3. *Eine lineare Reihenfolge der Behandlungen $B_i = \{b_{i1}, \ldots, b_{iq_i}\}$ eines Patienten $p_i \in P$ ist eine Permutation $\mu_i = (\mu_i(1), \ldots, \mu_i(q_i))$, wobei*

$$\mu_i(k) \in B_i \quad \forall\ k = 1, \ldots, q_i$$

und

$$\mu_i(k) \neq \mu_i(l) \quad \textit{für } k \neq l,\ k, l \in \{1, \ldots, q_i\}$$

gelten.

Weil beim Krankenhaus Scheduling Problem ein Behandlungsgraph eines Patienten kein zusammenhängender linearer Graph sein muss, kann es sein, dass durch die Rangfolgebeziehungen der Behandlungen eines Patienten keine lineare Reihenfolge der Behandlungen festgelegt ist. Besitzt ein Patient zum Beispiel zwei Behandlungen, zwischen denen es keine Rangfolgebeziehung gibt, so ist nicht vorgeschrieben, welche Behandlung ein Patient als erstes erhalten soll. Um einen eindeutigen linksgerichteten Schedule zu erzeugen, ist es notwendig, eine lineare Reihenfolge der Behandlungen festzulegen.

Definition 4.1.4. *Ist für die Behandlungen eines Patienten eine lineare Reihenfolge gegeben, so bezeichnen wir diese als total geordnete Behandlungsrangfolge des Patienten.*

Definition 4.1.5. *Eine total geordnete Behandlungsrangfolge für einen Patienten p_i heißt zulässig, wenn diese, den im Problem gegebenen Rangfolgebeziehungen Θ_i zwischen den Behandlungen des Patienten p_i, nicht widerspricht.*

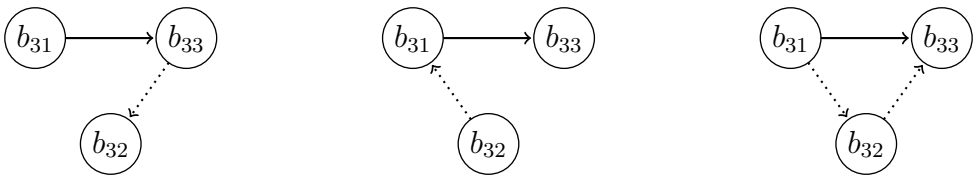
Eine zulässige total geordnete Behandlungsrangfolge eines Patienten kann mittels eines azyklischen Graphen dargestellt werden. Sei μ_i eine zulässige total geordnete Behandlungsrangfolge für einen Patienten $p_i \in P$, dann kann der Behandlungsgraph $D(p_i)$ zu einem *total geordneten Behandlungsgraphen* $D'(p_i)$ erweitert werden, indem Bogen entsprechend der Permutation μ_i eingefügt werden, so dass für den total geordneten Behandlungsgraphen

$$\mu_i(k)\mu_i(k+1) \in E(D'(p_i)) \quad \text{für } k = 1, \ldots, q_i - 1$$

gilt, wobei $E(D'(p_i))$ die Bogenmenge des Graphen $D'(p_i)$ bezeichnet. Wir bezeichnen diesen Vorgang als *Vervollständigung* eines Behandlungsgraphen.

Die Vervollständigung eines Behandlungsgraphen ist meist auf mehrere Weisen möglich. Im nachfolgenden Beispiel 4.1.6 sind drei Vervollständigungen eines Behandlungsgraphen dargestellt.

Beispiel 4.1.6. *Fortsetzung Beispiel 1.1.1.*



(a) **(b)** **(c)**

Abbildung 4.1: Mögliche Vervollständigungen des Behandlungsgraphen für Patient p_3 aus Beispiel 1.1.1.

In Abbildung 4.1 sind alle Möglichkeiten dargestellt, wie der Behandlungsgraph von Patient p_3 aus Beispiel 1.1.1 vervollständigt werden kann, um einen zulässigen total geordneten Behandlungsgraphen zu erzeugen.

Ein n-Tupel $\mu = (\mu_1, \ldots, \mu_n)$, bei dem die i-te Komponente durch eine zulässige total geordnete Behandlungsrangfolge des Patienten p_i, $i = 1, \ldots, n$, gegeben ist, bezeichnen wir im Folgenden als *Patientenrangfolge*.

Sei $R_k = \{b \in B \ : \ R(b) = r_k\}$ mit $|R_k| =: x_k$ die Menge der Behandlungen, die auf Ressource $r_k \in R$ stattfinden sollen. Dann ist eine *Behandlungsreihenfolge* für eine Ressource $r_k \in R$ durch eine Permutation $\pi_k = (\pi_k(1), \ldots, \pi_k(x_k))$ der Behandlungen R_k definiert, wobei $\pi_k(l)$ das Element von R_k bezeichnet, das als l-tes auf der Ressource r_k stattfindet. Ein m-Tupel $\pi = (\pi_1, \ldots, \pi_m)$, dessen i-te Komponente einer Behandlungsreihenfolge der Ressource r_i entspricht, werde als *Ressourcenreihenfolge* bezeichnet.

Um eine Formel ähnlich 4.1.1 für das Krankenhaus Scheduling Problem anwenden zu können, muss zu einer Patientenrangfolge eine zulässige Ressourcenreihenfolge gegeben sein. Daher wollen wir uns im Folgenden der Frage widmen, wann eine Ressourcenreihenfolge zulässig ist.

Für das Job Shop Scheduling Problem haben wir in Abschnitt 1.1.2 das Disjunctive Graph Modell kennengelernt. Eine Bearbeitungsreihenfolge für das Job Shop Scheduling Problem ist zulässig, wenn das der Bearbeitungsreihenfolge entsprechende gerichtete Disjunctive Graph Modell azyklisch ist. Auch im Fall des Krankenhaus Scheduling Problems wollen wir einen Graphen definieren, der einer Ressourcenreihenfolgen entspricht und mit dessen Kreisfreiheit die Zulässigkeit der Ressourcenreihenfolge äquivalent ist.

Das Disjunctive Graph Modell für das Krankenhaus Scheduling Problem
Sei eine Ressourcenreihenfolge $\pi = (\pi_1, \ldots, \pi_m)$ und eine Patientenrangfolge μ mit $\mu = (\mu_1, \ldots, \mu_n)$ gegeben. Dann kann ein gerichteter Disjunctive Graph $D'(\pi, \mu)$ mit Knotenmenge $V(D'(\pi, \mu))$ und Kantenmenge $E(D'(\pi, \mu))$ für das Krankenhaus Scheduling Problem wie folgt erstellt werden:

Für jede Behandlung $b_{ij} \in B$ enthalte der Graph $D'(\pi, \mu)$ einen Knoten v_{ij}. Es gilt

$$v_{ij} \in V(D'(\pi, \mu)) \Leftrightarrow b_{ij} \in B.$$

Weiter enthalte der Graph eine Quelle $s \in V(D'(\pi, \mu))$ und für jeden Patienten p_i mit $i = 1, \ldots, n$ eine Senke $t_i \in V(D'(\pi, \mu))$.
Für die Rangfolgebeziehungen der Behandlungen werden Bogen entsprechend

$$\mu_i(l)\mu_i(l+1) \in E(D'(\pi, \mu)) \quad \text{für } l = 1, \ldots, q_i - 1$$

und für die Reihenfolgebeziehungen auf den Ressourcen Bogen gemäß

$$\pi_k(l)\pi_k(l+1) \in E(D'(\pi, \mu)) \quad \text{für } l = 1, \ldots, |R_k| - 1$$

eingefügt. Weiter enthalte der Graph $D'(\pi, \mu)$ einen Bogen von der Quelle s zu der gemäß der total geordneten Rangfolge ersten Behandlung eines Patienten. Es gilt

$$s\mu_i(1) \in E(D'(\pi, \mu)) \quad \text{für } i = 1, \ldots, n.$$

Schließlich besitze der Graph einen Bogen von der gemäß der total geordneten Behandlungsrangfolge eines Patienten p_i letzten Behandlung zu der Senke t_i, so dass

$$\mu_i(q_i)t_i \in E(D'(\pi, \mu)) \quad \text{für } i = 1, \ldots, n$$

gilt.

Ein Beispiel für einen gerichteten Disjunctive Graph für das Krankenhaus Scheduling Problem ist in Beispiel 4.1.7 dargestellt.

Beispiel 4.1.7. *Gegeben seien die Patientenrangfolge und die Ressourcenreihenfolge gemäß der Lösung 1.3 zur Instanz aus Beispiel 1.1.1. Dann ergibt sich der nachfolgende gerichtete Disjunctive Graph.*

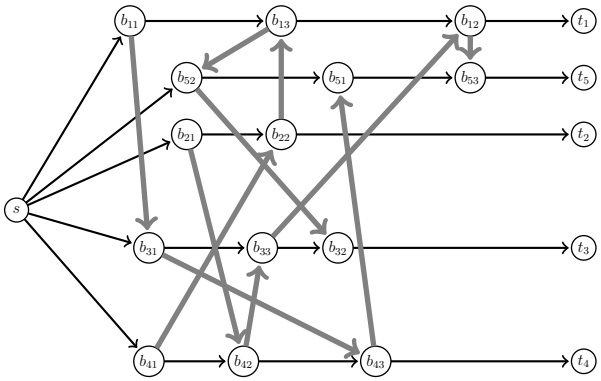


Abbildung 4.2: Gerichteter Disjunctive Graph zur Lösung 1.3.

In Abbildung 4.2 ist der gerichtete Disjunctive Graph zu der Lösung 1.3 der Krankenhaus Scheduling Instanz aus Beispiel 1.1.1 dargestellt. Die grauen Bogen stellen die Behandlungsreihenfolgen auf den Ressourcen dar, wohingegen die schwarzen Bogen die Behandlungsrangfolgen repräsentieren.

Ist der auf diese Weise erzeugte Graph $D'(\pi, \mu)$ azyklisch, so ist die Ressourcenreihenfolge π bei gegebener Patientenrangfolge μ zulässig.

Definition 4.1.8. *Gegeben sei eine Patientenrangfolge μ. Dann ist eine Ressourcenreihenfolge π zulässig, wenn der gerichtete Disjunctive Graph $D'(\pi, \mu)$ azyklisch ist.*

Ist zu einer Patientenrangfolge eine zulässige Ressourcenreihenfolge gegeben, so können die Startzeiten eines Schedules gemäß der nachfolgenden Formel 4.1.2 bestimmt werden.

Für gegebene total geordnete Behandlungsrangfolgen $\mu_1, \ldots, \mu_n$ und zugehörige zulässige Behandlungsreihenfolgen $\pi_1, \ldots, \pi_m$ auf den Ressourcen können wir die Startzeiten der Behandlungen gemäß der Formel

$$S_t(b_{i,j}) = \max\{S_t(b_{i,\mu_i(\mu_i^{-1}(b_{i,j})+1)}), S_t(\pi_l^{-1}(\pi_l(b_{i,j}) - 1))\} \tag{4.1.2}$$

unter Beachtung der Öffnungszeiten bestimmen, wobei l so gewählt ist, dass $R(b_{i,j}) = r_l$ gilt, und die Startzeiten entsprechend

$$S_t(b) = \begin{cases} Arr_i, & \text{falls } \exists\ j \in \{1, \ldots, m\} \text{ mit } b = \pi_j(1) \text{ und } b \in B_i, \\ \inf, & \text{sonst,} \end{cases}$$

initialisiert werden.

Durch die Festlegung einer Patientenrangfolge und zugehöriger zulässiger Ressourcenreihenfolgen ist mit 4.1.2 ein eindeutiger linksgerichteter Schedule gegeben.

Da es für jedes Scheduling Problem mit regulärem Optimierungskriterium immer eine linksgerichtete optimale Lösung gibt und es zu jedem linksgerichteten Schedule eine Kombination, bestehend aus einer Ressourcenreihenfolge und einer Patientenrangfolge, gibt, mittels

der der Schedule gemäß Formel 4.1.2 erzeugt werden kann, reicht es, um einen optimalen Schedule für ein Krankenhaus Scheduling Problem zu finden, alle möglichen Kombinationen von Patientenrangfolgen mit zugehörigen zulässigen Ressourcenreihenfolgen zu betrachten.

Im nächsten Abschnitt 4.1.2 möchten wir eine Startlösung mittels einer Greedy-Heuristik ermitteln. In der Greedy-Heuristik gehen wir in zwei Schritten vor. Als erstes vervollständigen wir die Patientengraphen. Zu den total geordneten Patientengraphen ermitteln wir dann in einem zweiten Schritt zulässige Behandlungsreihenfolgen.

4.1.2 Ermittlung einer Startlösung

Aufgrund der Formel 4.1.2 zur Ermittlung der Startzeiten ist ein Schedule eindeutig durch eine Patientenrangfolge und eine dafür zulässige Ressourcenreihenfolge gegeben. Daher genügt es, die Patientengraphen zu vervollständigen und für die resultierenden total geordneten Patientengraphen eine zulässige Ressourcenreihenfolge zu ermitteln. In diesem Abschnitt 4.1.2 möchten wir daher eine Startlösung generieren, indem wir zunächst die Patientengraphen vervollständigen und anschließend eine zulässige Ressourcenreihenfolge für die gegebene Patientenrangfolge erzeugen.

Vervollständigung der Behandlungsrangfolge eines Patienten

Für unsere Formel 4.1.2 zur Berechnung der Startzeiten benötigen wir eine Patientenrangfolge. Wir ergänzen daher die gegebenen Behandlungsrangfolgen so, dass wir zulässige total geordnete Behandlungsrangfolgen erhalten. Dabei gehen wir wie folgt vor:

Für einen Patienten $p_i \in P$ betrachten wir seinen Behandlungsgraphen $D(p_i)$. Wir möchten eine total geordnete Rangfolge μ_i erzeugen. Dazu besetzen wir iterativ die Positionen der Rangfolge μ_i. Wir wählen zufällig einen Knoten des Graphen mit Eingangsgrad Null aus und setzen diesen Knoten an die nächste freie Position unserer Ordnung μ_i, entfernen den Knoten aus dem Graphen und gehen für den resultierenden Graphen analog vor, bis die Knotenmenge des resultierenden Graphen leer ist. Diese Vorgehensweise ist in Algorithmus 5 dargestellt. Im folgenden Algorithmus wird für einen Graphen D die Knotenmenge mit $V(D)$ und der Eingangsgrad eines Knoten $v \in V(D)$ mit $d_D^-(v)$ bezeichnet.

Data : Patientengraph $D(p_i)$.
Result : Total geordnete Behandlungsrangfolge μ_i von Patient p_i.
$D = D(p_i)$;
$pos = 1$;
while $V(D) \neq \emptyset$ **do**
  $W = \{v \in V(D) \ : \ d_D^-(v) = 0\}$
  Wähle $v' \in W$.
  $\mu_i(pos) = v'$;
  $pos = pos + 1$;
  $D = D \setminus v'$;
end

Algorithmus 5 : Ermittlung einer total geordneten Behandlungsrangfolge.

Mittels des Algorithmus 5 vervollständigen wir die Patientengraphen eines jeden Patienten $p_i \in P$ auf zufällige Weise.

Eine Vervollständigung der Behandlungsgraphen aus Beispiel 1.1.1 ist in 4.1.9 zu sehen.

Beispiel 4.1.9. *Fortsetzung zu Beispiel 1.1.1.*

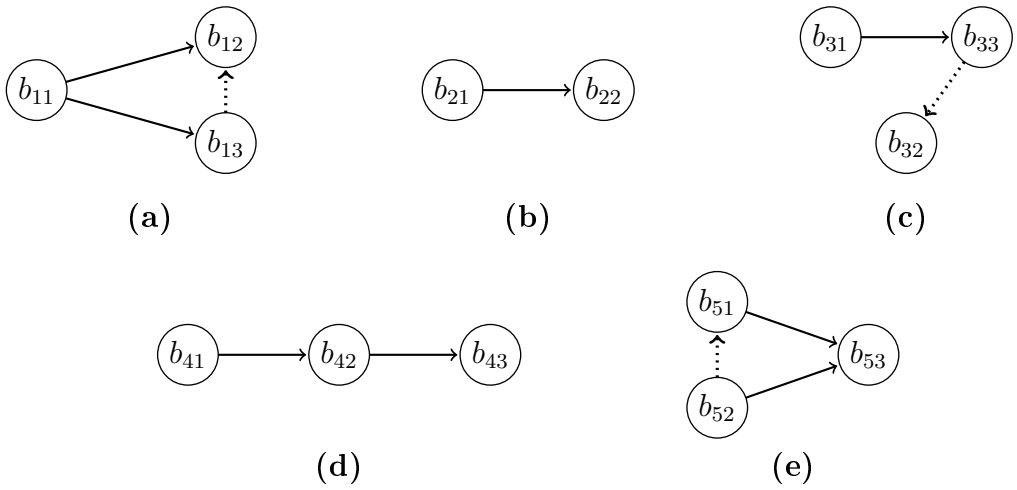


Abbildung 4.3: Total geordnete Behandlungsgraphen der Patienten $p_1, \dots p_5$ aus der Instanz 1.1.

In der Abbildung 4.3 sind total geordnete Behandlungsgraphen für die Patienten aus Instanz 1.1 dargestellt. Die Ordnungsrelationen, die durch die Vervollständigung zu den Patientengraphen hinzugefügt wurden, werden durch gestrichelte Bogen repräsentiert.

Um zu einer eindeutigen Startlösung zu gelangen, muss zu den total geordneten Behandlungsrangfolgen eine zulässige Ressourcenreihenfolge bestimmt werden. Unsere Verfahrensweise zur Generierung einer zulässigen Ressourcenreihenfolge beschreiben wir im nächsten Abschnitt.

Zulässige Behandlungsreihenfolgen auf den Ressourcen

Um einen zulässigen Schedule auf die von uns präferierte Weise erzeugen zu können, müssen wir zu den total geordneten Behandlungsrangfolgen eine zulässige Ressourcenreihenfolge festlegen. Dabei gehen wir wie folgt vor:

Zu Beginn sortieren wir die Patienten aufsteigend nach ihrem Ankunftszeitpunkt. Besitzen zwei Patienten denselben Ankunftszeitpunkt, so sortieren wir diese Patienten zufällig. Anschließend wählen wir die Patienten entsprechend der Sortierung nacheinander aus. Für jeden Patienten gehen wir seine Behandlungen entsprechend der gegebenen total geordneten Behandlungsrangfolge durch und weisen einer Behandlung die nächste freie Position auf der für die Behandlung vorgesehenen Ressource zu.

Ohne Einschränkung seien $p_1, p_2, \dots, p_n$ die Patienten sortiert nach ihrem Ankunftszeitpunkt. Weiter bezeichne π_{r_k} die Behandlungsreihenfolge von Ressource r_k für $k = 1, \dots, m$. Dann kann der oben beschriebene Algorithmus wie folgt formalisiert werden.

Data : Patienten sortiert nach Ankunftszeitpunkt $p_1, \ldots, p_n$, Patientenrangfolge $\mu = (\mu_1, \ldots, \mu_n)$.
Result : Behandlungsreihenfolgen auf den Ressourcen $\pi_{r_1}, \ldots \pi_{r_m}$.
$pos(r_k) = 0 \ \forall \ k = 1, \ldots m$;
for $i = 1, \ldots, n$ **do**
  for $k = 1, \ldots, q_i$ **do**
    $b = \mu_i(k)$
    $r = R(b)$;
    $\pi_r(pos(r)) = b$;
    $pos(r) = pos(r) + 1$;
  end
end

Algorithmus 6 : Ermittlung einer zulässigen Ressourcenreihenfolge zu der Patientenrangfolge μ.

Ist eine Patientenrangfolge und für diese eine zulässige Ressourcenreihenfolge gegeben, so kann mit dem Algorithmus 7 ein zulässiger Schedule erzeugt werden.

Data : Patientenrangfolge μ, in Bezug auf μ zulässige Ressourcenreihenfolge π.
Result : Zulässiger Schedule S.
$next_r[r_j] = 1 \ \forall \ j = 1, \ldots, m$;
$next_p[p_i] = 1 \ \forall \ i = 1, \ldots, n$;
$A_1 = \{b \in B \ : \ \exists \ j \in \{1, \ldots, m\}$ mit $\pi_j(next_r[r_j]) = b\}$;
$A_2 = \{b \in B \ : \ \exists \ i \in \{1, \ldots, n\}$ mit $\mu_i(next_p[p_i]) = b\}$;
$A = A_1 \cap A_2$;
while $A \neq \emptyset$ **do**
  for $b \in A$ **do**
    Setze $S_t(b)$ gemäß 4.1.2;
    $next_r[R(b)] = next_r[R(b)] + 1$;
    $next_p[P(b)] = next_p[P(b)] + 1$;
  end
  Aktualisiere A_1, A_2, A;
end

Algorithmus 7 : Startzeiten

Der mittels Algorithmus 7 erzeugte Schedule kann als Startlösung für die Lokale Suche genutzt werden. Für die Lokale Suche benötigen wir darüber hinaus eine geeignete Nachbarschaftsdefinition. Diese möchten wir im nächsten Abschnitt 4.1.3 vorstellen.

4.1.3 Eine Nachbarschaftsfunktion für das Krankenhaus Scheduling Problem

Eine Nachbarschaftsfunktion ordnet jeder Lösung eine Teilmenge von zulässigen Lösungen, die sogenannte Nachbarschaft, zu. In der Lokalen Suche wird in jedem Iterationsschritt eine Lösung aus der Nachbarschaft des Vorgängers mit dem Ziel ausgewählt, zu einer möglichst guten Lösung zu gelangen. Die Wahl einer geeigneten Nachbarschaft und damit einer geeigneten Nachbarschaftsfunktion ist von großer Bedeutung für eine Lokale Suche aber oft schwierig. Es muss ein Kompromiss in Bezug auf die Kardinalität der Nachbarschaft getroffen werden [54]. Ist die Kardinalität der Nachbarschaft groß, kann die Wahrscheinlichkeit

höher sein, einen guten Nachbarn zu finden. Allerdings wird auch mehr Zeit benötigt, um eine Nachbarschaft zu durchsuchen.

Eine Nachbarschaft wird mittels einer Menge von Moves generiert. Ein Move ist eine Zuordnungsvorschrift, die eine zulässige Lösung in eine andere zulässige Lösung überführt.

Definition 4.1.10. *Ein Move ist eine Funktion $M : \mathfrak{L} \to \mathfrak{L}$, die eine zulässige Lösung in eine andere zulässige Lösung überführt.*

Durch unsere Lösungsrepräsentation ist ein Schedule eindeutig durch eine Patientenrangfolge und eine zugehörige zulässige Ressourcenreihenfolgen gegeben. Eine Lösung kann somit in eine andere Lösung überführt werden, indem entweder die Rangfolge der Behandlungen eines Patienten oder die Reihenfolge der Behandlungen auf einer Ressource variiert wird. Ein Move sollte daher entweder die Rangfolge der Behandlungen eines Patienten oder die Reihenfolge der Behandlungen auf einer Ressourcen verändern.

Zunächst möchten wir die Änderung der Reihenfolge der Behandlungen auf einer Ressource betrachten.

Änderung der Behandlungsreihenfolge auf einer Ressource

Wir möchten durch die Veränderung der Reihenfolge der Behandlungen auf einer Ressource zu einer neuen zulässigen Lösung gelangen.

In Abschnitt 4.1.3 haben wir das Job Shop Scheduling Problem kennengelernt. Bei diesem kann eine linksgerichtete Lösung eindeutig mittels einer Bearbeitungsreihenfolge repräsentiert werden. Da das Job Shop Scheduling Problem ein viel untersuchtes Problem ist, wurden hierfür bereits eine Vielzahl von Nachbarschaften entwickelt. Diese beruhen meist auf der Vertauschung der Reihenfolge von aufeinanderfolgenden oder nicht-aufeinanderfolgenden Behandlungen. In „Job Shop Scheduling by Local Search“[54] werden einige Nachbarschaftsfunktionen für das Job Shop Scheduling Problem vorgestellt. Eine Nachbarschaft, die auf das Krankenhaus Scheduling Problem übertragen werden kann, möchten wir im Folgenden vorstellen.

Die grundlegende Idee der Nachbarschaft ist es, nur Vertauschungen von Operationen zu betrachten, die einen Einfluss auf den Makespan der aktuellen Lösung haben. In Abschnitt 4.1.3 haben wir das gerichtete Disjunctive Graph Modell kennengelernt. Der Makespan einer Lösung entspricht der Länge eines längsten Weges von der Quelle s zur Senke t im zugehörigen gerichteten Disjunctive Graph Modell. Den Makespan zu reduzieren kommt somit der Reduktion aller längsten Wege im gerichteten Disjunctive Graph Modell gleich. Einen längsten Weg von der Quelle s zur Senke t bezeichnen wir im Folgenden auch als *kritischen* Pfad. Ein kritischer Pfad lässt sich in Blöcke unterteilen. Ein *Block* ist eine maximale zusammenhängende Abfolge von Behandlungen auf einer Ressource auf einem kritischen Pfad. Eine viel angewendete Nachbarschaft wurde von Laarhoven et al. [55] entwickelt. Die Nachbarschaft setzt sich aus Moves zusammen, die einen Tausch zweier aufeinanderfolgender Operationen eines Blocks bewirken. Diese Nachbarschaft bringt die folgenden zwei Vorteile mit sich. Zum einen können durch die betrachteten Moves keine unzulässigen Reihenfolgen auf den Maschinen entstehen unter der Annahme, dass zwei aufeinanderfolgende Operationen eines Jobs auf unterschiedlichen Maschinen stattfinden.

Lemma 4.1.11. ***(Laarhoven [55])*** *Durch die Vertauschung zweier aufeinanderfolgender Operationen eines kritischen Blocks kann keine unzulässige Bearbeitungsreihenfolge entstehen.*

Beweis. *Sei $D'(L)$ der gerichtete Disjunctive Graph zu einer gegebenen Lösung L. Sei P der längste Weg von s nach t in $D'(L)$ mit $P = (s, v_1, \ldots, v_k, t)$. Angenommen, es existiert ein Bogen (v_i, v_{i+1}), $i \in \{1, \ldots, k\}$, so dass $\overline{D} := D'(L) \setminus (v_i, v_{i+1}) \cup (v_{i+1}, v_i)$ einen Kreis enthält. Dann muss es einen Weg von v_i nach v_{i+1} geben. Diesen Weg gab es aber auch schon in D' und der Weg wäre länger als der Bogen (v_i, v_{i+1}). Dies stünde im Widerspruch dazu, dass (v_i, v_{i+1}) auf dem längsten Weg von s nach t in $D'(L)$ liegt. Damit kann $\overline{D}$ keinen Kreis enthalten.* □

Zum anderen wird durch die Nachbarschaft die Zusammenhangseigenschaft gewährleistet. Diese besagt, dass eine optimale Lösung erreicht werden kann, egal von welcher Startlösung die Lokale Suche begonnen wird [39].

Satz 4.1.12. ***(Mati [39])*** *Für jede zulässige Lösung $L \in \mathfrak{L}$ gibt es eine endliche Sequenz von Vertauschungen, die zu einer optimalen Lösung führt.*

Beweis. *Sei $L \notin \mathfrak{L}_{opt}$ und $L^* \in \mathfrak{L}_{opt}$. Sei weiter $D'(L)$ der gerichtete Disjunctive Graph zu L und $D'(L^*)$ der gerichtete Disjunctive Graph zu L^*. Da L nicht optimal ist, existiert mindestens eine Kante, die auf einem kritischen Pfad in $D'(L)$ liegt und nicht auf einem kritischen Pfad in $D'(L^*)$. Da der Bogen auf einem kritischen Pfad liegt, kann die Kante umgekehrt werden, ohne dass ein Kreis im neuen Graphen entsteht. Wiederholt man dies für alle Bogen auf einem kritischen Pfad, die nicht in $D(L^*)$ sind, so gelangt man zu einer Lösung mit denselben kritischen Pfaden wie in $D(L^*)$.* □

Die Zusammenhangseigenschaft ist eine wichtige Eigenschaft einer Nachbarschaft, da sie gewährleistet, dass eine optimale Lösung erreicht werden kann.

Im Folgenden soll die vorgestellte Nachbarschaft auf das Krankenhaus Scheduling Problem übertragen werden.

In Abschnitt 4.1.1 haben wir bereits ein gerichtetes Disjunctive Graph Modell für das Krankenhaus Scheduling Problem eingeführt. Die Patientenrangfolge und die Ressourcenreihenfolge einer linksgerichteten Lösung können mittels des gerichteten Disjunctive Graph Modells dargestellt werden. Mit Hilfe dieses Modells kann zudem ermittelt werden, welche Vertauschungen von Behandlungen zur Verbesserung des Optimierungskriteriums führen können. Eine Verbesserung kann erzielt werden, wenn die Vertauschung eine Verschiebung des Fertigstellungsdatums eines Patienten bewirkt. Daher wird für jeden Patienten ein sogenannter kritischer Pfad ermittelt. Ein kritischer Pfad kann für einen Patienten wie folgt bestimmt werden.
Zu gegebener Patientenrangfolge μ und zugehöriger zulässiger Ressourcenreihenfolge π wird der Graph $D'(\pi, \mu)$ erstellt. Für jede Senke t_i eines Patienten $p_i \in P$ können wir den Weg zurückverfolgen, der für den Fertigstellungszeitpunkt des Patienten verantwortlich ist. Gegeben sei ein Schedule S mit zugehörigem gerichteten Disjunctive Graph $D'(\pi, \mu)$. Dann ermittelt man für Patient $p_i \in P$ den kritischen Pfad $\mathcal{P}_i$, indem man ausgehend von der

Senke t_i solange eine Vorgängerbehandlung auswählt, die am spätesten beendet wurde, bis man bei der Quelle s angelangt ist.
Sei $\mathcal{P}_i = (v_0, \ldots, v_l)$ mit $v_0 = t_i$ und $v_1 = \mu(q_i)$. Dann gilt

$$v_{x+1} = \max_{y \in N^-(v_x)} S_t(B(y)),$$

wobei $v_l = s$ ist und $N^-(v)$ die Menge aller Vorgängerknoten von v und $B(y)$ die Behandlung, die durch den Knoten y repräsentiert wird, bezeichnet.

Ein Block ist für einen gegebenen kritischen Pfad analog zu oben als eine maximale Sequenz aufeinanderfolgender Behandlungen auf einer Ressource definiert. Vertauscht man aufeinanderfolgende Behandlungen eines Blocks des Pfades $\mathcal{P}_i$, so kann wegen Lemma 4.1.11 kein Kreis entstehen, es sei denn die Behandlungen gehören zu demselben Patienten.
Betrachtet man zwei aufeinanderfolgende Behandlungen desselben Patienten $p_i \in P$, so muss als erstes geprüft werden, ob durch die gegebenen Rangfolgebeziehungen der Behandlungen Θ_i eine Änderung der Behandlungsreihenfolge zulässig wäre. Ist die Vertauschung zulässig, so muss sowohl die Behandlungsreihenfolge als auch die Behandlungsrangfolge des zugehörigen Patienten angepasst werden.

Für einen Patienten wird kein kritischer Pfad ermittelt, wenn der Fertigstellungszeitpunkt bereits frühestmöglich ist, das heißt dem Ankunftszeitpunkt plus der Summe der Behandlungsdauern entspricht.

Die gerade beschriebene Nachbarschaft besteht aus allen zulässigen Vertauschungen zweier aufeinanderfolgender Behandlungen eines Blocks eines kritischen Pfades eines Patienten. Eine Lösung kann aber zudem verändert werden, wenn die total geordnete Behandlungsrangfolge eines Patienten verändert wird. Daher möchten wir im Folgenden Moves betrachten, die die Veränderung der total geordneten Behandlungsrangfolge eines Patienten bewirken.

Änderung der Behandlungsrangfolge eines Patienten

Die Veränderung einer total geordneten Behandlungsrangfolge eines Patienten kann große Auswirkungen auf den zugehörigen Schedule haben. Wir beschränken uns darauf, nur dann Behandlungen eines Patienten zu vertauschen, wenn sie auf einem kritischen Pfad eines Patienten unmittelbar aufeinander folgen. Hierbei sind insbesondere auch Vertauschungen zweier Behandlungen möglich, die nicht auf derselben Ressource stattfinden. Es ist darauf zu achten, dass Vertauschungen nur durchgeführt werden, wenn sie den gegebenen Rangfolgebeziehungen der Behandlungen eines Patienten nicht widersprechen.

Mit der beschriebenen Nachbarschaft und dem Erzeugen einer Startlösung sind die grundlegenden Elemente einer Lokalen Suche für das Krankenhaus Scheduling Problem definiert. Auf problemspezifische Parameter wie die Größe der Tabu-Liste möchten wir an dieser Stelle nicht näher eingehen, sondern diese erst im Zusammenhang mit der Robusten Lokalen Suche erläutern. Der nächste Abschnitt 4.2 widmet sich der Anpassung der Lokalen Suche zu einer Robusten Lokalen Suche für das Krankenhaus Scheduling Problem.

4.2 Robuste Lokale Suche für das Krankenhaus Scheduling Problem unter Unsicherheiten

Mittels Robuster Lokaler Suche möchten wir in diesem Abschnitt 4.2 eine Lösung für das Krankenhaus Scheduling Problem unter Unsicherheiten generieren. In der Lösung soll in einem vorgegebenem Maß berücksichtigt werden, dass zusätzliche Patienten im Laufe eines Krankenhaustages eintreffen können. Das heißt, wir möchten einen Schedule generieren, in dem nicht einzig die Minimierung der Kostenfunktion im Vordergrund steht, sondern einen Schedule, in dem Freiräume für das Einplanen zusätzlich auftretender Patienten geschaffen werden. Unser Ziel ist es, mittels unserer Ansätze einer Robusten Lokalen Suche einen Schedule zu finden, der eine gute Kompromisslösung zwischen optimalem Kostenwert und maximalem Robustheitswert darstellt.
Dazu muss die Lokale Suche aus dem vorherigen Abschnitt 4.1 so angepasst werden, dass sie bei Bedarf Freiräume zum Einplanen der zusätzlich auftretenden Patienten in den Schedule integriert. Im folgenden Abschnitt 4.2.1 wird beschrieben, welche Anpassungen an die Lokale Suche dazu notwendig sind. Um zu messen, wie gut zusätzlich auftretende Patienten in einen Schedule mit eingeplant werden können, muss ein geeignetes Robustheitsmaß definiert werden. In Abschnitt 4.2.2 werden zwei mögliche Robustheitsmaße für das Krankenhaus Scheduling Problem definiert. Für den Ansatz einer Lokalen Suche über eine Konvexkombination des Kosten- und Robustheitswertes benötigen wir einen Richtwert, um die beiden Werte gleichwertig zu machen. Einen Vorschlag für einen Richtwert stellen wir in Abschnitt 4.2.3 vor.

4.2.1 Anpassung der Lokalen Suche

In Abschnitt 2.3 haben wir bereits erwähnt, dass es zwei Vorgehensweisen gibt, einen Schedule robust gegenüber Unsicherheiten zu machen. Wir wählen einen Buffering-Ansatz, das heißt wir möchten in unserem Schedule Freiräume schaffen beziehungsweise erzwingen, um die Notfallpatienten mit einplanen zu können.
Eine Schwierigkeit bei Buffering-Ansätzen ist, dass entschieden werden muss, zu welchen Zeitpunkten Freiräume eingefügt werden sollen. Diese Frage soll die Lokale Suche im Zuge des Durchsuchens der Nachbarschaften lösen.
Um der Lokalen Suche zu ermöglichen, Freiräume in den Schedule einzuplanen, fügen wir sogenannte *Timeslots* ein. Ein Timeslot beschreibt ein Zeitintervall auf einer Ressource von festgelegter Dauer t_{TS} zu einem festen Zeitpunkt. Ein Timeslot besitzt zwei mögliche Konfigurationen. Ist der Timeslot *angeschaltet*, so kann zu dem Zeitraum, in dem sich der Timeslot befindet, auf der entsprechenden Ressource keine Behandlung stattfinden. Ist der Timeslot *ausgeschaltet*, so hat der Timeslot keine Konsequenzen für die Planung der Startzeiten von Behandlungen.
Alle Ressourcen sind während ihrer Öffnungszeiten vollständig mit Timeslots überdeckt. Für jede Ressource ist der Zeitraum von dem ersten Ankunftsdatum eines gegebenen Patienten bis zum Ende der Öffnungszeiten des letzten Tages, an dem das Auftreten von Notfallpatienten möglich ist, mit Timeslots überdeckt, wobei jeder Zeitpunkt von genau einem Timeslot überdeckt wird.
Der Timeslot soll als Platzhalter für die Behandlungen der Notfallpatienten dienen. Durch angeschaltete Timeslots sollen Freiräume geschaffen oder bestehende Freiräume vergrößert werden, so dass die Möglichkeit besteht, Behandlungen eines Notfallpatienten einzuplanen.

Wir haben den Ansatz von festen Timeslots gewählt, da in diesem Fall ein Freiraum auch bei Veränderung der Reihenfolge der Behandlungen auf den Ressourcen zur gleichen Zeit bestehen bleibt, da der Freiraum durch den Timeslot eindeutig festgelegt ist.
Eine Schwierigkeit bei dem Gebrauch von Timeslots ist es, eine geeignete Größe der Timeslots festzulegen. Je kleiner man die Dauer t_{TS} wählt, umso präziser können Freiräume geschaffen werden. Allerdings steigt auch die Anzahl der Timeslots.

Durch das Einführen von Timeslots müssen die grundlegenden Elemente einer Lokalen Suche angepasst werden.

Repräsentation einer Lösung

Die Repräsentation einer Lösung aus Abschnitt 4.1.1 muss um die Timeslots ergänzt werden. In der Zeit, in der angeschaltete Timeslots liegen, dürfen keine Behandlungen stattfinden. Das heißt, dass die angeschalteten Timeslots elementar für die Ermittlung einer Lösung sind. Um Startzeiten für Behandlungen zu berechnen, ist es notwendig zu wissen, welche Timeslots angeschaltet sind. Mit einer Formel analog zu 4.1.2 können unter Beachtung der angeschalteten Timeslots weiterhin die Startzeiten der Behandlungen berechnet werden. Sei $\mathfrak{T}$ die Menge der angeschalteten Timeslots. Dann kann eine linksgerichtete Lösung für das Krankenhaus Scheduling Problem eindeutig durch eine Patientenrangfolge μ, eine zugehörige zulässige Ressourcenreihenfolge π und die Menge der angeschalteten Timeslots $\mathfrak{T}$ ermittelt werden.

Ermitteln einer Startlösung

Da ein linksgerichteter Schedule eindeutig durch eine Patientenrangfolge μ, Ressourcenreihenfolgen π und die Menge der angeschalteten Timeslots $\mathfrak{T}$ gegeben ist, muss neben der Vervollständigung der Behandlungsrangfolgen und der Festlegung einer zugehörigen zulässigen Ressourcenreihenfolge entschieden werden, welche Timeslots in einer Startlösung angeschaltet seien sollen. Die Vervollständigung der Behandlungsrangfolge und die Festlegung einer Ressourcenreihenfolge kann mittels Algorithmus 5 und Algorithmus 6 vorgenommen werden. Eine Menge von angeschalteten Timeslots kann zum Beispiel zufällig ermittelt werden. Sind Informationen über die Notfallpatienten bekannt, so können diese zum Anschalten geeigneter Timeslots genutzt werden. Sind beispielsweise Szenarien gegeben, die das Eintreffen von Notfallpatienten simulieren, so können Timeslots ausgewählt werden, zu deren Zeitraum in den Szenarien besonders häufig Notfallpatienten auftreten. Es sei bemerkt, dass die Entscheidung, welche Timeslots am besten angeschaltet werden sollten, unter anderem vom Robustheitsmaß abhängig ist.

Anpassen der Nachbarschaft

Ein linksgerichteter Schedule für das Krankenhaus Scheduling Problem ist eindeutig durch das Tripel $(\mu, \pi, \mathfrak{T})$ gegeben. Variiert man eine der drei Komponenten, so verändert sich die Lösung.
In der Lokalen Suche für das Krankenhaus Scheduling Problem haben wir bereits zwei Arten von Moves kennengelernt, wobei die eine Art eine Veränderung der Ressourcenreihenfolge π und die andere eine Veränderung der Patientenrangfolge μ bewirkte. Diese beiden Arten von Moves können für das Krankenhaus Scheduling Problem unter Unsicherheiten übernommen werden. Darüber hinaus kann eine Lösung verändert werden, indem die Menge der angeschalteten Timeslots $\mathfrak{T}$ variiert wird. Dazu betrachteten wir Moves, die

die Konfiguration eines Timeslots ändern. Die in der betrachteten Lösung angeschalteten Timeslots können ausgeschaltet oder ausgeschaltete Timeslots angeschaltet werden. Während die Moves aus der Lokalen Suche für das Krankenhaus Scheduling Problem auf die Verbesserung der Kostenfunktion abzielen sollen, dienen die Moves zur Veränderung der Konfiguration eines Timeslots dazu, den Robustheitswert eines Schedules zu steigern.

In diesem Abschnitt 4.2.1 haben wir die elementaren Elemente einer Lokalen Suche so angepasst, dass robustere Lösungen mittels einer Lokalen Suche gefunden werden können. Damit diese Lösungen aber auch wirklich vom Lokalen Such Algorithmus ermittelt werden, verfolgen wir die beiden Ansätze, die in Abschnitt 3.2 allgemein beschrieben wurden. In den Ansätzen werden Lösungen mittels eines Robustheitsmaßes bewertet. Daher sollen im nächsten Abschnitt 4.2.2 mögliche Robustheitsmaße für das Krankenhaus Scheduling Problem vorgestellt werden.

4.2.2 Robustheitsmaß für das Krankenhaus Scheduling Problem

In diesem Abschnitt 4.2.2 möchten wir Robustheitsmaße für das Krankenhaus Scheduling Problem vorstellen, die für einen Schedule messen, wie groß das Ausmaß der Absicherung eines Schedules gegenüber dem Auftreten von Notfallpatienten ist.

In Abschnitt 3.2 wurde bereits eine allgemeine Definition für ein Robustheitsmaß eingeführt. Ein Robustheitsmaß ist eine Funktion $\mathfrak{R} : (\mathfrak{L}, Pot(\mathfrak{S})) \to [0, 1]$, die für eine Lösung L und eine gegebene Szenarienmenge $\mathfrak{S}$ einen Wert ermittelt, der beschreiben soll, wie robust die Lösung L in Bezug auf eine Szenarienmenge $\mathfrak{S}' \subseteq \mathfrak{S}'$ ist. Der ermittelte Wert ist dabei umso näher an eins, je besser die Lösung in Bezug auf die Szenarienmenge $\mathfrak{S}'$ gemessen am Robustheitsmaß abgesichert ist. Um Robustheitsmaße für das Krankenhaus Scheduling Problem unter Unsicherheiten entwickeln zu können, müssen wir zunächst definieren, wodurch ein Szenario charakterisiert ist.

Ein Szenario für das Krankenhaus Scheduling Problem unter Unsicherheiten ist durch eine Menge von Notfallpatienten $P^* = (p_1^*, \ldots, p_{n^*}^*)$ gegeben. Für jeden Notfallpatienten $p^* \in P^*$ gibt es einen Ankunftszeitpunkt $Arr(p^*)$, zu dem bekannt wird, welche Behandlungen $B(p^*)$ der Notfallpatient durchlaufen muss. Für die Behandlungen eines Notfallpatienten ist eine feste Abfolge gegeben. Seien $B(p_i^*) = (b_{i1}^*, \ldots, b_{iq_i^*}^*)$ die Behandlungen von Patient p_i^* mit $i \in \{1, \ldots, n^*\}$. Dann muss Behandlung $b_{i,j-1}^*$ beendet sein, bevor Behandlung b_{ij}^* stattfinden kann für alle $j = 2, \ldots, q_i^*$. Für jede Behandlung b_{ij}^* mit $j \in \{1, \ldots, q_i^*\}$ und $i \in \{1, \ldots, n^*\}$ gibt es eine eindeutige Ressource $R(b_{ij}^*) \in R$, auf der die Behandlung stattfinden kann. Des Weiteren besitzt jede Behandlung b_{ij}^* mit $j \in \{1, \ldots, q_i^*\}$ und $i \in \{1, \ldots, n^*\}$ eine feste Behandlungsdauer $t(b_{ij}^*)$.

Wir möchten in diesem Abschnitt Robustheitsmaße vorstellen, die einen Robustheitswert eines Schedules in Bezug auf ein einzelnes Szenario ermitteln. Die vorgestellten Robustheitsmaße können einfach auf Szenarienmengen angewandt werden, indem zum Beispiel der Mittelwert

$$\mathfrak{R}(L, \mathfrak{S}) = \frac{1}{|\mathfrak{S}|} \sum_{\mathfrak{s} \in \mathfrak{S}} \mathfrak{R}(L, \mathfrak{s})$$

gebildet wird.

Bei unseren Robustheitsmaßen nehmen wir an, dass der gegebene Schedule bei dem Auftreten der Unsicherheiten nicht angepasst werden kann. Das heißt, jede Behandlung findet

zu dem Zeitpunkt statt, an dem sie geplant wurde und kann nicht verschoben werden. Für die hinzukommenden Behandlungen bedeutet dies, dass sie gegebenenfalls in die Freiräume des Schedules eingeplant werden können. Eine Behandlung eines Notfallpatienten kann zu einem Zeitpunkt nur dann stattfinden, wenn im Schedule zum betrachteten Zeitpunkt ein Freiraum besteht und der Freiraum noch mindestens für die Dauer der Behandlung andauert.
Sei $F(S)$ eine Menge, die alle maximal zusammenhängenden Freiräume im Schedule S beinhaltet.

Definition 4.2.1. *Sei $\mathfrak{s}$ ein Szenario und S ein zulässiger Schedule. Dann kann einer Behandlung $b^*_{ij} \in B(\mathfrak{s})$ eine zulässige Startzeit $S_t(b^*_{ij})$ in $F(S)$ zugeordnet werden, wenn ein Freiraum $f \in F(S)$ existiert, so dass*

(i) $S_t(b^*_{ij}) \in f$,

(ii) $(S_t(b^*_{ij}) + t(b^*_{ij})) \in f$,

(iii) $R(b^*_{ij}) = R(f)$,

(iv) $S_t(b^*_{ij}) \geq (S_t(b^*_{i,j-1}) + t(b^*_{i,j-1}))$ *falls* $j > 1$ *oder* $S_t(b^*_{i1}) \geq Arr(p^*_i)$.

Bei dem Einplanen der Behandlungen in die Freiräume eines gegebenen Schedules ist darauf zu achten, dass verschiedene Behandlungen aus einem Szenario nicht zur selben Zeit stattfinden können. Nachdem für eine Behandlung eine zulässige Startzeit ausgewählt wurde, muss die Menge der Freiräume daher entsprechend angepasst werden. Es ist denkbar, dass ein Freiraum besser ausgefüllt ist, wenn zum Beispiel zwei Behandlungen kurzer Dauer in dem Freiraum anstelle einer Behandlung längerer Dauer stattfinden. Damit das Einplanen von Notfallpatienten kein eigenes Optimierungsproblem darstellt, verfolgen wir die Strategie, dass wenn ein Patient ankommt, seinen Behandlungen die nächst freie zur Verfügung stehende Zeit auf den entsprechenden Ressourcen zugewiesen wird. Diese Vorgehensweise kann auch theoretisch fundiert werden. Zum Zeitpunkt des Eintritt eines Notfallpatienten ist nicht bekannt, wann der nächste Notfallpatient eintreten wird und welche Behandlungen dieser zu durchlaufen hat. Daher werden einem Notfallpatienten die nächsten zur Verfügung stehenden Termine zugewiesen.

Wir werden im Folgenden zwei Robustheitsmaße betrachten, die sich beide auf die Wartezeiten eines Patienten beziehen. Dabei entspricht die *Wartezeit* eines Patienten, der Zeitdauer zwischen Ankunfts- und Fertigstellungszeitpunkt abzüglich der Summe der Dauer seiner Behandlungen.

Definition 4.2.2. *Die Wartezeit eines Patienten $p^*_i \in P^*$ werde mit $W(p^*_i)$ bezeichnet und sei durch*

$$W(p^*_i) = C(p^*_i) - Arr(p^*_i) - \sum_{k=1}^{q^*_i} t(b^*_{ik})$$

gegeben.

Die Wartezeit eines Patienten entspricht somit seiner Aufenthaltsdauer im Krankenhaus, abzüglich der Zeitdauer, in der er behandelt wird.

Im ersten Robustheitsmaß werden wir bestrafen, wenn die Wartezeiten für einen Patienten länger als eine vorgegebene Zeitdauer ist. Im zweiten Robustheitsmaß wird die Länge der Wartezeiten der Patienten bestraft. Je länger die Patienten zwischen ihren Behandlungen warten müssen, umso negativer wirkt sich dies auf das Robustheitsmaß aus.

Das erste Robustheitsmaß soll im nachfolgenden Abschnitt vorgstellt werden. Wir bezeichnen das Robustheitsmaß als Fit-Into-Robustheitsmaß.

Fit-Into-Robustheitsmaß

Beim Fit-Into-Robustheitsmaß ist eine Zeitdauer vorgegeben, die die maximal erlaubte Wartezeit eines Patienten darstellen soll. Nur wenn die realisierte Wartezeit eines Patienten kürzer als die maximal erlaubte Wartezeit ist, gilt der Patient als in *annehmbarer Zeit behandelt*. Die Behandlungen der Notfallpatienten können nur in den Freiräumen des Schedules und nur während der Öffnungszeiten der jeweiligen Ressource stattfinden. Kann nicht allen Behandlungen eines Patienten ein Termin im Laufe der Öffnungszeiten der entsprechenden Ressource am Ankunftstag zugewiesen werden, so gilt der Patient als in nicht annehmbarer Zeit behandelt. Es wird nicht betrachtet, dass Behandlungen eines Notfallpatienten auch noch am Folgetag nach seinem Eintreffen stattfinden können. Entweder wird einer Behandlung eine zulässige Startzeit in einem Freiraum des entsprechenden Tages zugeordnet oder es wird keine Startzeit für die Behandlung festgelegt.
Wird die maximal erlaubte Wartezeit äquivalent zu der Länge der Öffnungszeiten gewählt, so wird betrachtet, dass alle Behandlungen eines Notfallpatienten am Tag seines Eintreffens stattfinden sollen. Da beim Fit-Into-Robustheitsmaß alle Behandlungen eines Notfallpatienten am Tag seiner Ankunft stattfinden sollen, ist eine größere Wahl für die erlaubte Wartezeit nicht sinnvoll.

Als Robustheitskriterium dient beim Fit-Into-Robustheitsmaß die prozentuale Anzahl an in annehmbarer Zeit behandelten Patienten.

Um das Robustheitsmaß zu ermitteln, muss die Anzahl der Patienten bestimmt werden, deren Behandlungen in den Schedule im Rahmen der vorgegebenen maximalen Wartezeit eingeplant werden können. Dazu werden zunächst die Freiräume auf den Ressourcen ermittelt. Die Patienten aus dem Szenario werden nach ihrem Ankunftszeitpunkt sortiert und entsprechend dieser Reihenfolge durchgegangen. Für jeden Patienten werden die Behandlungen in der gegebenen Behandlungsrangfolge durchlaufen und einer Behandlung jeweils der nächstmögliche zulässige freie Zeitraum auf der Ressource zugeordnet. Kann allen Behandlungen ein Starttermin innerhalb der Öffnungszeiten zugewiesen werden, so kann im Anschluss die Wartezeit des Patienten berechnet und somit geprüft werden, ob der Patient in annehmbarer Zeit behandelt wurde.

Sei die Zeitspanne, in der Behandlungen eines Notfallpatienten nach seinem Eintreffen stattfinden sollen, gegeben durch T_{FI}. Seien weiter $P^* = (p_1^*, \ldots, p_{n^*}^*)$ die Notfallpatienten aus Szenario $\mathfrak{s}$ sortiert nach ihrem Ankunftszeitpunkt und $F(S)$ die Freiräume im Schedule S. Dann kann die Anzahl der in annehmbarer Zeit behandelten Patienten gemäß des folgenden Algorithmus berechnet werden.

Data : P^*, $F(S)$, T_{FI}.
Result : **Anzahl Patienten** $nFit$, **die in annehmbarer Zeit behandelt werden.**
$nFit = 0$;
$F = F(S)$;
for $i = 1, \ldots, n^*$ **do**
  for $j = 1, \ldots, q_i^*$ **do**
    if *Es ex. keine zulässige Startzeit in* $F(S)$ *für* b_{ij}^*. **then**
      break;
    end
    Setze Startzeit für b_{ij}^*;
    Reduziere F **um das Zeitintervall** $[S_t(b_{ij}^*), S_t(b_{ij}^*) + t(b_{ij}^*)]$.
  end
  if *Es wurde keine zulässige Startzeit für* b_{i,q_i^*} *ermittelt.* **then**
    continue;
  end
  Ermittle die Wartezeit $W(p_i^*)$;
  if $W(p_i^*) \leq T_{FI}$ **then**
    $nFit = nFit + 1$;
  end
end

Algorithmus 8 : Ermitteln der Anzahl der in annehmbarer Zeit behandelten Patienten.

Der Algorithmus 8 gibt die Anzahl der Patienten zurück, die nach unserer Definition in annehmbarer Zeit, das heißt mit einer Wartezeit kürzer gleich dem gegebenen Zeitraum T_{FI} stattfinden können. Hat man die Anzahl gegeben, so kann das Robustheitsmaß definiert werden als

$$\mathfrak{R}(L, \mathfrak{s}) = \frac{\text{nFit}}{|P(\mathfrak{s})|}, \tag{4.2.1}$$

wobei $P(\mathfrak{s})$ die Menge der Notfallpatienten des Szenarios $\mathfrak{s}$ bezeichnet.

Ist $T_{FI} = 0$, so wird geprüft, ob jede Behandlung eines Patienten aus dem Szenario zum frühestmöglichen Zeitpunkt stattfinden kann. Für den Fall dass jeder Notfallpatient nur eine Behandlung besitzt, kann dieser Fall als eine Absicherung gegen zufällige Ressourcenausfälle interpretiert werden.

Als zweites mögliches Robustheitsmaß betrachten wir das Schedule-Into-Robustheitsmaß. Beim Schedule-Into-Robustheitsmaß wird eine Funktion betrachtet, mit der die Wartezeiten der Patienten gewichtet werden. Die gewichteten Wartezeiten dienen dem Schedule-Into-Robustheitsmaß als Grundlage zur Bewertung eines Schedules.

Schedule-Into-Robustheitsmaß

Bei dem Schedule-Into-Robustheitsmaß ist die Dauer der Wartezeiten der Patienten grundlegend für das Robustheitsmaß. Je größer die Wartezeit eines Patienten ist, desto negativer wirkt sich dies auf den Robustheitswert aus. Die Dauer der Wartezeit eines Patienten wird mittels einer monoton steigenden Funktion

$$\omega : \mathbb{R}_{\geq 0} \longrightarrow \mathbb{R}_{\geq 0},\ \omega(0) = (0)$$

gewichtet. Wird $\omega = id$ gewählt, so fließen die Wartezeiten entsprechend ihrer Dauer in das Robustheitsmaß ein. Die Funktion ω ermöglicht es, Wartezeiten mit zunehmender Dauer überproportional im Robustheitsmaß zu berücksichtigen. Mittels stückweise stetiger Funktionen, können verschiedene Levels für die Berücksichtigung von Wartezeiten definiert werden. Der Bereich, der möglichen Dauer von Wartezeiten, wird in Intervalle eingeteilt, wobei jeder Dauer eines Intervalls von ω derselbe Wert zugeordnet wird. Die Wartezeit eines Patienten fließt entsprechend dem Wert zu dem Intervall, in dem die Dauer der Wartezeit liegt, in das Robustheitsmaß ein.

Für das Schedule-Into-Robustheitsmaß nehmen wir an, dass die Behandlungen eines Notfallpatienten auch außerhalb der Öffnungszeiten einer Ressource stattfinden können. Reichen die Freiräume auf den Ressourcen nicht aus, um sämtliche Behandlungen der Notfallpatienten in den Freiräumen durchzuführen, so können die Behandlungen auch noch nach dem Ende der Öffnungszeiten stattfinden. Wurde jeder Behandlung eine zulässige Startzeit innerhalb der Freiräume oder nach Ende der Öffnungszeiten zugeordnet, so kann für jeden Patienten die zugehörige Wartezeit berechnet werden. Bestimmt man die Summe der gewichteten Wartezeiten und normiert diese auf das Intervall $[0, 1]$, so gelangt man zum Schedule-Into-Robustheitsmaß.

Beim Schedule-Into-Robustheitsmaß wird die Summe der gewichteten Wartezeiten mittels der maximalen Summe der gewichteten Wartezeiten normiert.

Um die maximale Summe der gewichteten Wartezeiten zu ermitteln, müssen wir berechnen, wie groß die gewichtete Summe der Wartezeiten maximal werden kann. Dazu nehmen wir an, dass alle Behandlungen der Notfallpatienten nach Ende der Öffnungszeiten stattfinden müssen und berechnen für diesen Fall die Summe der gewichteten Öffnungszeiten.

Ist ein Szenario $\mathfrak{s}$ gegeben, so sortieren wir die Notfallpatienten aufsteigend nach ihren Ankunftszeitpunkten. Ohne Einschränkung seien $P^* = (p_1^*, \ldots, p_{n^*}^*)$ die Notfallpatienten aus Szenario $\mathfrak{s}$ aufsteigend sortiert nach ihrem Ankunftszeitpunkt. Sei F eine Menge mit Freiräumen, die Freiräume entsprechend den Zeiträumen außerhalb der Öffnungszeiten der Ressourcen bis zu dem Datum enthält, an dem das Auftreten von Notfallpatienten nicht mehr betrachtet wird. Dann kann die maximale Summe der gewichteten Wartezeiten W_{gew}^{max} mittels Algorithmus 9 berechnet werden.

Data : P^***,** F**,** ω**.**
Result : Summe der gewichteten Wartezeit W_{gew}^{max}**.**
$W_{gew}^{max} = 0$**.**
for $i = 1, \ldots, n^*$ **do**
  for $j = 1, \ldots, q_i$ **do**
    Ermittle die frühestmögliche Startzeit für b_{ij}^* **in** F**;**
    Reduziere F **um den Zeitraum, zu dem die Behandlung** b_{ij}^* **stattfindet.**
  end
  Ermittle die gewichtete Wartezeit $\omega(W(p_i^*))$**;**
  $W_{gew}^{max} += \omega(W(p_i^*))$**;**
end

Algorithmus 9 : Ermitteln der maximalen Summe der gewichteten Wartezeit.

Sei $F(S)$ die Menge der Freiräume im Schedule S. Fügen wir $F(S)$ Freiräume, die dem Zeitraum zwischen Ende der Öffnungszeiten eines Tages und Beginn der Öffnungszeiten des darauffolgenden Tages entsprechen, bis zu dem Tag hinzu, an dem das Auftreten von

Notfallpatienten nicht mehr betrachtet wird, dann kann mittels des Algorithmus 10 die Summe der gewichteten Wartezeiten aller Patienten bestimmt werden.

Data : P^***,** $F(S)$**,** ω**.**
Result : Gewichtete Wartezeit W_{gew}**.**
$W_{gew} = 0$**.**
$F = F(S)$**;**
for $i = 1, \ldots, n^*$ **do**
  for $j = 1, \ldots, q_i$ **do**
    Setze Startzeit für b^*_{ij}**;**
    Reduziere F **um das Zeitintervall** $[S_t(b^*_{ij}), S_t(b^*_{ij}) + t(b^*_{ij})]$**.**
  end
  Ermittle die gewichtete Wartezeit $\omega(W(p^*_i))$**;**
  $W_{gew} += \omega(W(p^*_i))$**;**
end

Algorithmus 10 : Ermitteln der Summe der gewichteten Wartezeit.

Ist die Summe der gewichteten Wartezeiten und die maximale Summe der gewichteten Wartezeiten berechnet, so kann der Schedule-Into-Robustheitswert mittels

$$\mathfrak{R}(L, \mathfrak{s}) = \frac{W^{max}_{gew} - W_{gew}}{W^{max}_{gew}} \tag{4.2.2}$$

bestimmt werden.

Möchten wir die Robuste Lokale Suche auf das Krankenhaus Scheduling Problem anwenden, so sind für den Fall einer Robusten Lokalen Suche mit Akzeptor alle wichtigen Elemente geklärt. Möchten wir aber eine Robuste Lokale Suche mit Konvexkombination von Kosten- und Robustheitswert durchführen, so müssen wir noch einen Richtwert für den Kostenwert einer Instanz bestimmen, um die beiden Werte gleichwertig zu machen. Einen möglichen Richtwert möchten wir im folgenden Abschnitt 4.2.3 bestimmen.

4.2.3 Richtwert bezüglich der Kostenfunktion

Für die Robuste Lokale Suche mit Konvexkombination aus Robustheits- und Kostenwert benötigen wir einen Richtwert für den Kostenwert des Problems, um den Robustheits- und den Kostenwert gleichwertig zu machen.
Der Richtwert sollte in etwa dem Kostenwert einer Lösung entsprechen, tendenziell aber eher größer gewählt sein. Daher wollen wir eine obere Schranke für das Krankenhaus Scheduling Problem wählen, die wir aufgrund möglicherweise auftretender Notfallpatienten anpassen.

Obere Schranke für das Krankenhaus Scheduling Problem

Obere Schranken für das Krankenhaus Scheduling Problem können ermittelt werden, indem ein zulässiger Schedule generiert und der zugehörige Kostenwert des Schedules berechnet wird.
Um einen zulässigen Schedule in kurzer Zeit zu generieren, wird dieser meist zunächst mittels Dispatching Rules erstellt. Dispatching Rules stellen einfache Regeln dar, mit deren Hilfe zum Beispiel im Falle des Krankenhaus Scheduling Problems entschieden werden kann, welcher Patient als nächstes bearbeitet oder welcher Behandlung als nächstes

ein Starttermin zugewiesen werden soll. Eine bekannte Dispatching Rule für Scheduling Probleme ist die First-In-First-Out-Regel. Sie beschreibt, dass immer einem Job, der am frühestmöglichen zur Verfügung stand, eine Startzeit zugewiesen werden soll.
In einem zweiten Schritt kann die so ermittelte Lösung verbessert werden.

Ist eine obere Schranke für das Krankenhaus Scheduling Problem gegeben, so kann die Schranke genutzt werden, um eine obere Schranke für das Krankenhaus Scheduling Problem unter Unsicherheiten zu erhalten.

Obere Schranke für das Krankenhaus Scheduling Problem unter Unsicherheiten

Wir nehmen an, dass eine obere Schranke für das Krankenhaus Scheduling Problem gegeben ist. Da wir beim Krankenhaus Scheduling Problem unter Unsicherheiten in dem Schedule Freiräume zum Einplanen von Notfallpatienten mittels der Timeslots erzeugen, möchten wir diese Schranke für das Krankenhaus Scheduling Problem unter Unsicherheiten anpassen.
Es ist klar, dass je größer der Einfluss des Robustheitsmaßes bei der Robusten Lokalen Suche ist, das heißt umso größer der Robustheitsfaktor d gewählt wird, desto mehr Freiräume enthält der resultierende Schedule.
Soll ein Schedule bestmöglich gegen das Auftreten von Notfallpatienten abgesichert sein, so kann dies dadurch gewährleistet werden, dass der gesamte Zeitraum, in dem Notfallpatienten auftreten beziehungsweise behandelt können, mit angeschalteten Timeslots überdeckt ist. Möchte man eine obere Schranke für das Krankenhaus Scheduling Problem unter Unsicherheiten generieren, so müssen den Behandlungen Startzeiten zugewiesen werden, die nach dem Zeitpunkt liegen, an dem Behandlungen von Notfallpatienten stattfinden können.

Die so ermittelte obere Schranke kann als Richtwert für die Robuste Lokale Suche mit Konvexkombination genutzt werden.

Richtwert für den Kostenwert des Krankenhaus Scheduling Problems unter Unsicherheiten

Als Richtwert für die Robuste Lokale Suche mit Konvexkombination kann die oben vorgestellte obere Schranke genutzt werden.

Kann die gesamte Behandlungsdauer der Notfallpatienten abgeschätzt werden, so kann ein besserer Richtwert für das Krankenhaus Scheduling Problem unter Unsicherheiten für die folgenden zwei Fälle bestimmt werden:

- für Schedule-Into-Robustheitsmaß.
- für das Fit-Into-Robustheitsmaß für den Fall, dass T_{FI} so gewählt wird, dass Behandlungen eines Patienten innerhalb der gesamten Öffnungszeiten stattfinden können und der Patient als in zulässiger Zeit behandelt gilt.

Kann die hinzukommende Behandlungsdauer pro Tag für eine Szenarienmenge $\mathfrak{S}$ durch $T_{\mathfrak{S}}$ abgeschätzt werden, das heißt, wenn die Gleichung

$$\sum_{b \in B(\mathfrak{s})} t(b) \leq T_{\mathfrak{S}} \quad \forall\, \mathfrak{s} \in \mathfrak{S}$$

erfüllt ist, so kann ein Schedule mittels Dispatching Rules generiert werden, bei dessen Erzeugung das Ende der Öffnungszeiten als um $T_{\mathfrak{S}}$ vorverlegt betrachtet wird.

Bezieht man den Robustheitsfaktor d mit ein, so kann der Richtwert weiter verbessert werden, in dem ein Schedule generiert wird, bei dessen Erzeugung das Ende der Öffnungszeiten als um $d \cdot T_{\mathfrak{S}}$ vorverlegt betrachtet wird.

Der auf diese Weise ermittelte Richtwert muss keine obere Schranke sein, da durch das Einfügen von Timeslots mehr Freiraum geschaffen werden kann als die gesamte Behandlungsdauer in einem Szenario beträgt. Betrachten wir dazu das folgende Beispiel.

Beispiel 4.2.3. *Sei die hinzukommende Behandlungsdauer pro Tag für alle Szenarien $\mathfrak{s} \in \mathfrak{S}$ aus einer gegebenen Szenarienmenge $\mathfrak{S}$ durch T beschränkt.*
Angenommen, für ein Szenario $\mathfrak{s} \in \mathfrak{S}$, sei die Behandlungsdauer aller hinzukommender Behandlungen $b \in B(\mathfrak{s})$ durch $t(b) = t_s$ und die Anzahl der hinzukommenden Behandlungen durch k gegeben. Dann gilt

$$k \leq \left\lfloor \frac{T}{t_s} \right\rfloor .$$

Sei weiter die Größe eines Timeslots auf $t_{TS} = 2t_s - \epsilon$, mit $\epsilon > 0$ sehr klein, festgelegt. Plant der Algorithmus für jede Behandlung einen angeschalteten Timeslot ein, so kann annähernd

$$k * t_{TS} \leq \left\lfloor \frac{T}{t_s} \right\rfloor * (2t_s - \epsilon) \leq 2T$$

Freiraum am betrachteten Tag entstehen. Da ein angeschalteter Timeslot den Fertigstellungszeitpunkt aller Patienten beeinflussen könnte, kann sich der Kostenwert um bis zu $2nT$ steigern.

Im nachfolgenden Kapitel 5 möchten wir die Robuste Lokale Suche für das Krankenhaus Scheduling Problem testen.

Kapitel 5

Auswertung

In diesem Kapitel 5 möchten wir die von uns entwickelten Ansätze für eine Robuste Lokale Suche für das Krankenhaus Scheduling Problem testen.

Im ersten Abschnitt 5.1 wird beschrieben, wie Instanzen zum Testen unseres Verfahrens generiert werden können. Im zweiten Abschnitt legen wir dar, wie zu einer Instanz Szenarien erzeugt werden können. Im dritten Abschnitt werden die Experimente erläutert, welche wir zum Testen unserer Verfahren durchführen sowie die zugehörigen Ergebnisse diskutiert.

5.1 Generierung von Instanzen

Um die Robuste Lokale Suche zu testen, benötigen wir geeignete Instanzen. Um eine Instanz zu generieren, müssen folgende Parameter festgelegt werden:

- Es muss die Anzahl der Patienten n festgelegt werden.
- Es muss die Anzahl der Ressourcen m festgelegt werden.
- Es muss die maximale Anzahl von Behandlungen pro Patient q_{max} festgelegt werden.
- Es muss festgelegt werden, wie häufig im Mittel ein Patient ankommen soll λ.

Sind diese Parameter gegeben, so wird eine Instanz wie folgt erzeugt.

Erzeugen von Ressourcen

Für die vorgegebene Anzahl m werden Ressourcen generiert.

Erzeugen von Behandlungstypen

Wurden die Ressourcen generiert, so folgen als nächstes die Behandlungstypen. Ein Behandlungstyp ist eine Kombination einer Ressource mit einer Zeitdauer. Ein Behandlungstyp soll eine spezielle Behandlung für eine Ressource darstellen. Für ein Röntgengerät kann das Röntgen eines speziellen Körperteils als ein Behandlungstyp gesehen werden. Es benötigt weniger Zeit einen Finger zu röntgen als einen Rücken.
Mittels Poissionverteilung wird die Gesamtzahl an Behandlungstypen bestimmt. Der Parameter der Poissionverteilung ist festgelegt durch die dreifache Anzahl der Ressourcen. Anschließend werden, mittels einer weiteren Poissionverteilung mit Parameter zehn, für die ermittelte Anzahl von Behandlungstypen Behandlungsdauern bestimmt. Es werden dabei nur Behandlungsdauern zugelassen, die größer als null sind. Gibt es mehr als eine Ressource, so wird zu einer Behandlungsdauer mittels einer Gleichverteilung eine Ressource ermittelt. Für die gegebene Anzahl von Behandlungstypen wird so zunächst mittels

Poissionverteilung eine Behandlungsdauer und anschließend mittels Gleichverteilung eine Ressource bestimmt.

Generieren von Patienten

Wurden die Behandlungstypen erzeugt, so werden als nächstes die Patienten generiert. Für die gegebene Anzahl von Patienten n werden zunächst die Ankunftszeiten mittels einer Exponentialverteilung mit dem Parameter $1/\lambda$ ermittelt. Der Parameter λ entspricht der mittleren Dauer zwischen den Ankünften zweier aufeinanderfolgender Notfallpatienten in Minuten. In der Praxis ist es üblich, Wartezeiten mittels einer Exponentialverteilung zu simulieren. Die Patienten können nur innerhalb der Öffnungszeiten auftreten.
Für jeden Patienten wird des Weiteren ein azyklischer Behandlungsgraph erzeugt. Nachdem mittels Gleichverteilung über $\{1, \ldots, q_{max}\}$ die Anzahl der Behandlungen für einen betrachteten Patienten bestimmt wird, wird ein zufälliger azyklischer gerichteter Graph erstellt, der als Knotenmenge die Menge der Behandlungstypen besitzt. Es wird ein Teilgraph mit der vorgegebenen Anzahl von Behandlungen zufällig ausgewählt. Dieser so ermittelte Teilgraph wird als Behandlungsgraph des Patienten gewählt. Die Rangfolgebeziehungen zwischen Behandlungen eines Patienten sind durch seinen Behandlungsgraphen gegeben.

Erzeugung von Instanzen

Sind die Parameter festgelegt, so kann eine Instanz wie oben beschrieben generiert werden.

In dem Namen einer Instanz spiegeln sich die ausgewählten Parameter wider. Eine Instanz ist durch den Namen

$$\text{Inst\_n}=n\text{\_q}=q_{max}\text{\_m}=m\text{\_l}=\lambda$$

gekennzeichnet.

5.2 Szenariengenerierung

Um einen Schedule gegenüber dem Auftreten von Notfallpatienten abzusichern, benötigen wir für unsere Ansätze einer Robusten Lokalen Suche Szenarien, die das zufällige Auftreten von Notfallpatienten simulieren. Daher soll in diesem Abschnitt beschrieben werden, wie solche Szenarien generiert werden können.

Zu einer Instanz soll eine passende Szenarienmenge generiert werden. Dazu müssen folgende Parameter festgelegt werden:

- die Anzahl der Szenarien $|\mathfrak{S}|$.
- eine Dauer λ_s in Minuten, die angibt, wie häufig im Mittel eine Behandlung auf einer Ressource hinzukommen soll.
- ein Endzeitpunkt T_{max}, bis wann das Auftreten von Notfallpatienten betrachtet werden soll.
- die maximale Anzahl q^*_{max} von Behandlungen pro Patient.

Entsprechend der gewünschten Anzahl von Szenarien werden Szenarios generiert. Die Generierung eines Szenarios ähnelt der Generierung einer Instanz. Bei dem Erstellen eines Szenarios wird eine Menge von Patienten generiert. Die Behandlungstypen entsprechen der in der Instanz gegebenen Behandlungstypen. Ein Unterschied zur Generierung einer Instanz liegt darin, dass keine feste Anzahl von Patienten geben ist, sondern ein Zeitpunkt, bis wann Notfallpatienten auftreten können.

Für eine gegeben Instanz ermitteln wir diesen Zeitpunkt T_{max} wie folgt:

Ermittlung eines spätesten Zeitpunktes, bis zu dem das Eintreffens von Notfallpatienten möglich ist

Wir möchten Szenarien so generieren, dass die auftretenden Notfallpatienten den Ablauf des ermittelten Schedules stören können. Dazu müssen zum Zeitpunkt des Auftretens eines Notfallpatienten Behandlungen im Schedule beziehungsweise im Anschluss an diesen Zeitpunkt stattfinden. Daher ermitteln wir zunächst eine untere Schranke in Bezug auf das Fertigstellungsdatum der am spätesten stattfindenden Behandlung. Sei im Folgenden T^*_{max} das Fertigstellungsdatum der am spätesten stattfindenden Behandlung.

Wird die Menge der Patienten betrachtet, so liegt das frühste Fertigstellungsdatum eines Patienten nicht nach dem Fertigstellungsdatum der am spätesten stattfindenden Behandlung. Es gilt

$$T^*_{max} \geq \varsigma_1 = \max_{p_i \in P} \left\{ Arr_i + \sum_{b \in B_i} t(b) \right\}.$$

Darüber hinaus kann das Fertigstellungsdatum der am spätesten stattfindenden Behandlung nicht vor dem Zeitpunkt liegen, an dem eine Ressource bei vollständiger Auslastung am frühestmöglichen alle Behandlungen abgeschlossen hat. Somit gilt

$$T^*_{max} \geq \varsigma_2 = \max_{r_j \in R} \left\{ Arr_{min} + \sum_{b \in B \,:\, R(b)=r_j} t(b) \right\},$$

wobei Arr_{min} das früheste Ankunftsdatum eines Patienten der betrachteten Instanz bezeichnet. Bestimmen wir den späteren der beiden ermittelten Zeitpunkte

$$T^*_{max} \geq \max \varsigma_1, \varsigma_2,$$

so wählen wir das Ende der Öffnungszeiten der Ressourcen des Vortages als spätest mögliches Ankunftsdatum T_{max} der Notfallpatienten.

Durch diese Wahl von T_{max} ist garantiert, dass alle Notfallpatienten für das erzeugen eines robusten Schedules relevant sind.

Erzeugen von Szenarien

Wir möchten Szenarien generieren, die das Auftreten von Notfallpatienten simulieren. Diese Szenarien sollen aus gegebenen Informationen erstellt werden.

Wir gehen davon aus, dass aus historischen Daten bekannt ist, wann und wie oft eine Behandlung auf einer Ressource im Schnitt stattfindet. Es ist aber nicht bekannt, welche

Behandlungen zu demselben Patienten gehören. Daher erzeugen wir Szenarien, in denen verschiedene Behandlungen keine Beziehungen aufweisen. Dies bewerkstelligen wir, indem wir Patient erzeugen, die nur eine Behandlung besitzen.

Für den gegebenen Parameter λ_s erzeugen wir Patienten, deren Ankunftszeiten mittels einer Exponentialverteilung mit Parameter

$$\frac{m}{\lambda_s}$$

bestimmt werden. Dabei ist die Anzahl der Behandlungen eines Patienten auf eins festgelegt. Für die Behandlung des Patienten wird ein Behandlungstyp mittels Gleichverteilung über der Menge der aus der Instanz gegebenen Behandlungstypen ausgewählt.

Gegen die auf diese Weise erstellten Szenarien möchten wir einen Schedule im Laufe des Algorithmus absichern. Da Behandlungen in den Szenarien zueinander in keiner Beziehung stehen, bezeichnen wir diese Szenarien auch als *unkorreliert*.

Um zu testen, wie sich der gegen unkorrelierte Szenarien abgesicherte Schedule, bei dem Auftreten von Notfallpatienten verhält, erzeugen wir eine zweite Menge von Szenarien. In der zweiten Menge von Szenarien sind in einem Szenario Notfallpatienten gegeben, die mehrere Behandlungen besitzen. Daher bezeichnen wir diese Szenarienmenge auch als *korrelierte* Szenarien.

Die Patienten werden ähnlich zu den Patienten der gegebenen Instanz erzeugt. Die maximale Anzahl von Behandlungen ist durch die maximale Behandlungsanzahl der Patienten aus der Instanz gegeben und wird mittels einer Gleichverteilung über $\{1, \ldots, q_{max}\}$ bestimmt. Auch können die Patienten während der gesamten Öffnungszeiten auftreten. Nur in der Ermittlung der Ankunftszeitpunkte und der Patientengraphen unterscheidet sich die Erzeugung eines Szenarios.

Damit die Summe der Behandlungen aller Notfallpatienten in etwa der Anzahl der Behandlungen aus den unkorrelierten Szenarien entspricht, wird als Parameter für die Exponentialverteilung

$$\frac{m}{\lambda_s * E[\tilde{q}]}$$

gewählt, wobei $\tilde{q} \sim U(\{1, \ldots, q_{max}\})$ und U eine diskrete Gleichverteilung auf der gegebenen Menge ist.

Die Patientengraphen sind für die Notfallpatienten nicht nur azyklische Graphen, sondern sogar linear. Dies sichert eine eindeutige Reihenfolge der Behandlungen für einen Patienten.

Festlegung des Mittleren Ankunftsdatums

In unseren Experimenten werden wir uns auf die Betrachtung von Szenarienmengen beschränken, bei denen im Mittel alle 30 Minuten eine Behandlung stattfinden soll. Dementsprechend ist $\lambda_s = 30$ gewählt.

Anzahl der Szenarien Für beide Arten von Szenarien erzeugen wir hundert Stück. Während bei der Absicherung meist nicht alle Szenarien berücksichtigt werden, testen wir die Robustheit eines ermittelten Schedules anhand der hundert erzeugten Szenarien.

Bemerkung 5.2.1. *Für eine Schranke T_{max} kann die mittlere Anzahl von Notfallpatienten, die im Zeitraum $[0, T_{max}]$ auftreten, ermittelt werden als $\frac{T_{open}}{\lambda}$, wobei T_{open} die Öffnungsdauer im Zeitraum ist und λ die mittlere Zahl der Ankünfte von Notfallpatienten bezeichnet.*

Die Ankunftszeitpunkte der Patienten werden mittels einer Exponentialverteilung zum Parameter $1/\lambda$ ermittelt. Dabei sei der Zeitpunkt, zu dem der erste Patient eintritt, gegeben durch X_1, der Zeitpunkt, an dem der zweite Patient eintritt, gegeben durch $X_1 + X_2$ und so weiter, wobei $X_i \sim exp(1/\lambda)$ und die X_i stochastisch unabhängig sind. Wir definieren $S_i = \sum_{k=1}^{i} X_i$ als den Zeitpunkt, an dem der i-te Patient eintrifft. Um zu bestimmen, wie viele Patienten im Mittel bis zum Zeitpunkt T_{max} eintreten, betrachten wir die Zufallsvariable

$$N(T_{open}) := \sum_{l=1}^{\infty} \mathbb{1}(S_l \leq T_{open}), \qquad t \geq 0.$$

Die Zufallsvariable $N(T_{open})$ repräsentiert die zufällige Anzahl von Patienten, die bis einschließlich zum Zeitpunkt T_{open} aufgetreten sind.
Wir möchten den Erwartungswert von $N(T_{open})$ bestimmen. Für den Erwartungswert gilt

$$EN(T_{open}) = \sum_{k=1}^{\infty} P(N(T_{open}) \geq k) \overset{Def\ N}{=} \sum_{k=1}^{\infty} P(S_k \leq T_{open}).$$

Da $X_i \sim exp(1/\lambda) \sim \Gamma(1, 1/\lambda)$ ist, gilt mit den Faltungseigenschaften der Gammaverteilung für $n \in \mathbb{N}$, dass $S_n \sim \Gamma(n, 1/\lambda)$ ist. Die Verteilungsfunktion von S_n ist somit gegeben durch

$$F^{S_n}(x) = 1 - \exp\frac{-x}{\lambda} \sum_{j=0}^{n-1} \frac{\left(\frac{x}{\lambda}\right)^j}{j!}, \qquad x > 0$$

und für $T_{open} > 0$ gilt

$$\begin{aligned} EN(T_{open}) &= \sum_{k=1}^{\infty} \int_0^{T_{open}} \frac{1/\lambda^k}{(k-1)!} x^{k-1} \exp\frac{-x}{\lambda} dx \\ &\overset{Fubini}{=} \int_0^{T_{open}} \frac{1}{\lambda} \exp\frac{-x}{\lambda} \underbrace{\sum_{k=1}^{\infty} \frac{\left(\frac{x}{\lambda}\right)^{k-1}}{(k-1)!}}_{\exp\left(\frac{x}{\lambda}\right)} dx = \frac{T_{open}}{\lambda}. \end{aligned}$$

5.3 Implementierung

Wir haben unsere Ansätze in Java implementiert. Für die Robuste Lokale Suche benutzen wir das Framework Optaplanner [24]. Optaplanner ist ein einbettbares Programm, das Planungsprobleme optimiert. Das Programm soll Java-Programmierern helfen, Constraint Satisfaction Probleme effizient zu lösen, indem es Optimierungsheuristiken und Metaheuristiken mit einer sehr effizienten Score-Berechnung kombiniert.

5.4 Experimente

In diesem Abschnitt möchten wir die vorgestellten Ansätze einer Robusten Lokalen Suche für das Krankenhaus Scheduling Problem unter Unsicherheiten testen und vergleichen. Als erstes wird ermittelt, welche Kosten- und Robustheitswerte für eine Robuste Lokale Suche mit Akzeptor für die beiden Robustheitsmaße erreicht werden. Im Anschluss verfahren wir für die Robuste Lokale Suche mit Konvexkombination analog. Daran anschließend werden wir die Ergebnisse der beiden Ansätze vergleichen.

Um unseren Algorithmus geeignet testen zu können, müssen geeignete Werte für die Größe der Tabuliste und die Größe der Timeslots ermittelt werden. Des Weiteren muss ein Abbruchkriterium festgelegt werden.

Ermitteln einer geeigneten Länge der Tabuliste

Die Länge der Tabuliste hat große Auswirkungen auf den Lokalen Such Algorithmus. Die falsche Wahl der Länge der Tabuliste kann bewirken, dass ein Algorithmus ineffizient ist.

In „Design and Evaluation of Tabu Search Algorithms for Multiprocessor Scheduling“ [53] gibt Thesen einen Überblick über bestehende Ansätze für die geeignete Wahl der Länge einer Tabuliste für Tabu Search Algorithmen für Scheduling Probleme. Werden Undo-Moves auf der Tabuliste gespeichert, so ist nach Glover [30] sieben ein geeigneter Wert für die Länge der Tabuliste. Anderson et al. [6] beschreiben, dass beim Betrachten von Undo-Moves eine Länge der Tabuliste zwischen sieben und fünfzehn für Path Assignment Probleme gut funktioniert. Løkketangen [38] beschreibt einen Fall, in dem der effizienteste Algorithmus mehr als 200 Lösungen in der Tabuliste beinhaltet. Morton und Pentico [41] sind der Meinung, dass es für Scheduling Probleme am besten ist, alle Lösungen in die Tabuliste aufzunehmen. Neben festen Werten für die Länge einer Tabuliste, kann diese auch im Laufe eines Algorithmus variiert werden. Taillard [52] verwendet erfolgreich eine Tabuliste, deren Länge zu bestimmten Zeitpunkten zufällig geändert wird.

Es gibt keine allgemeine Regel zur Festlegung der Länge der Tabuliste, da sie anscheinend vom betrachteten Problem selbst und dem verwendeten Algorithmus abhängt.

Wir orientieren uns für die Ermittlung einer geeigneten Tabulänge an den oben vorgeschlagenen Werten. Wir möchten Undo-Moves auf der Tabuliste speichern. Daher könnte die Wahl zwischen sieben und fünfzehn geeignet sein. Da das Krankenhaus Scheduling Problem von einem Scheduling Problem vor allem aufgrund des Einfügens von Timeslots abweicht und dadurch die Kardinalität einer Nachbarschaft vergrößert wird, testen wir zudem größere Werte für die Länge der Tabuliste. Wir möchten die Größen einer Tabuliste entsprechend der Tabelle 5.1 testen.

Parameter	Werte
L_{Tabu}	$\{7, 10, 15, 25, 50, 100\}$

Tabelle 5.1: Werte für die Länge der Tabuliste.

Ermitteln einer geeigneten Größe der Timeslots

Wir nehmen an, dass sich die Qualität einer Lösung mit Variation der Größe der Timeslots verändert. Wählt man die Größe der Timeslots klein, so können Freiräume präziser geplant werden. Allerdings nimmt die Kardinalität einer Nachbarschaft mit Minimierung der Größe der Timeslots zu. Somit wird für das Durchsuchen einer Nachbarschaft mehr Zeit benötigt. Daher möchten wir testen, wie sich eine ermittelte Lösung und die benötigte Zeit bei Variation der Größe der Timeslots verändert. Dazu betrachten wir die Ausprägungen entsprechend Tabelle 5.2.

Parameter	Werte
T_{TS}	$\{5, 10, 20, 30\}$

Tabelle 5.2: Werte für die Größe der Timeslots in Minuten.

Festlegen eines Abbruchkriteriums

Bevor wir unseren Algorithmus testen können, müssen wir ein Abbruchkriterium bestimmen.

In der Literatur sind verschiedene Abbruchkriterien bei Lokaler Suche für das Job Shop Scheduling Problem vertreten. In Mati et al. [39] wird ein Zeitlimit von 18 Sekunden betrachtet. In dieser Zeit sind bei dem betrachteten Algorithmus bis zu zwei Millionen Auswertungen möglich. Bei dem Ansatz von Armentano und Scrich [7] wird für einen Tabu Search Algorithmus die maximal erlaubte Anzahl von Iterationen auf 250 beschränkt. Auch eine Kombination verschiedener Abbruchkriterien ist denkbar. Bei Pezzella und Merelli [44] stoppt der Tabu Search Algorithmus, wenn eine untere Schranke oder die maximal erlaubte Anzahl von Iterationen erreicht wurde. Die maximal erlaubte Anzahl von Iterationen ist auf $100n$ begrenzt, wobei $n \in [10, 30]$ die Anzahl der Jobs bezeichnet.

Wir verwenden zwei verschiedene Abbruchkriterien. Diese variieren je nach Testfall. Betrachtet man zum Beispiel verschiedene Timeslotgrößen, so legen wir eine maximale Anzahl Iterationen fest. In anderen Tests verwenden wir ein Zeitlimit von zehn Minuten.

Da die Untersuchung unserer Ansätze viel Zeit benötigt, führen wir unsere Untersuchungen an einer Instanz Inst_n=100_q=5_m=3_l=10 durch.

In den nachfolgenden Tests beschränken wir uns darauf, im Fit-Into-Robustheitsmaß den Wert T_{FI} entsprechend einer Dauer von acht Stunden zu setzen, wobei die Dauer der Öffnungszeiten jeder Ressource pro Wochentag acht Stunden beträgt. Werden die Behandlungen eines Notfallpatienten innerhalb der Öffnungszeiten am Tag seiner Ankunft eingeplant, so gilt der Patient somit als in annehmbarer Zeit behandelt. Für das Schedule-Into-Robustheitsmaß betrachten wir ausschließlich den Fall, dass die Wartezeiten entsprechend ihrer Dauer im Maß berücksichtigt werden.

Im nachfolgenden Abschnitt betrachten wir die Tests für eine Robuste Lokale Suche mit Akzeptor. Wir sichern die Lösungen zunächst in jeder Iteration gegen ein festes Szenario mit $\lambda_s = 30$ ab.

5.4.1 Robuste Lokale Suche mit Akzeptor

Bei der Robusten Lokalen Suche mit Akzeptor möchten wir einen Schedule erzeugen, der eine gegebene Robustheitsschranke erfüllt. Um den Algorithmus nach Möglichkeit von einer vom Akzeptor akzeptierten Startlösung zu beginnen, benutzen wir den oben vorgestellten Startalgorithmus, modifizieren die Öffnungszeiten aber so, dass der Algorithmus nur in den ersten $100 - a$ Prozent eines Tages Behandlungen einplanen darf. In den letzten a Prozent des Tages werden die Timeslots angeschaltet. Unter der Voraussetzung, dass die maximale Dauer hinzukommender Behandlungen höchstens gleich der Dauer der Öffnungszeiten eines Tages ist und die Notfallpatienten gleichmäßig über den Tag verteilt auftreten, sollte auf diese Weise in den meisten Fällen der Robustheitswert einer Startlösung größer gleich der Robustheitsschranke sein.

Ermittlung einer geeigneten Tabulänge

Um eine geeignete Tabulänge zu ermitteln, möchten wir die Instanz für den mittleren Wert einer Robustheitsschranke von $a = 50$ bei einer Größe der Timeslots von 20 Minuten testen. Für die betrachtete Instanz erzielen wir bei zehn Durchläufen die in Tabelle 5.3 und Tabelle 5.4 dargestellten Kostenwerte für eine Laufzeit von zehn Minuten pro Durchlauf.

L_{Tabu}	7	10	15	25	50	100
Durchlauf 1	243530	247203	245557	246727	247292	247307
Durchlauf 2	244875	248681	244878	237091	246156	246819
Durchlauf 3	248679	246488	246617	246511	246494	245360
Durchlauf 4	243156	244171	245461	246469	247341	242388
Durchlauf 5	243778	243196	245461	246511	246156	244799
Durchlauf 6	247119	244078	245598	244084	246156	247307
Durchlauf 7	244875	247342	246617	247567	247343	246819
Durchlauf 8	243156	243196	245557	237091	247341	247307
Durchlauf 9	247119	244078	246479	247307	246156	244799
Durchlauf 10	245452	247322	246833	246469	243902	244870
Mittelwert	245173.9	245575.5	245905.8	244582.7	246433.7	245777.5

Tabelle 5.3: Kostenwerte bei möglichen Tabulängen für das Fit-Into-Robustheitsmaß.

In Tabelle 5.3 sind die Kostenwerte für zehn Durchläufe sowie deren Mittelwerte dargestellt. Der beste Kostenwert wurde für eine Länge der Tabuliste von 25 erzielt. In diesem Fall ist auch der Mittelwert am geringsten. Allerdings lässt sich kein klarer Trend erkennen. Die Länge der Tabuliste von 15 und 50 liefern die beiden schlechtesten Mittelwerte, wohingegen im Vergleich zu diesen die Mittelwerte für eine Länge der Tabuliste von 7 oder 100 bessere Werte darstellen.

L_{Tabu}	7	10	15	25	50	100
Durchlauf 1	155716	219750	210242	194423	220938	214707
Durchlauf 2	214602	201019	212874	205895	212880	216194
Durchlauf 3	212440	206579	213771	206715	192611	220254
Durchlauf 4	183264	168457	211218	207014	192611	214707
Durchlauf 5	209308	212931	211218	207014	192611	157837
Durchlauf 6	223468	203812	211218	221263	192611	214707
Durchlauf 7	205810	210962	200301	212832	192611	191069
Durchlauf 8	214602	214593	194595	212832	220249	157837
Durchlauf 9	184671	201019	205549	207014	221805	220254
Durchlauf 10	214602	206579	214728	205895	192611	212352
Mittelwert	201848.3	204570.1	208571.4	208089.7	203153.8	201991.8

Tabelle 5.4: Kostenwerte für mögliche Tabulängen bei Betrachtung des Schedule-Into-Robustheitsmaßes.

In Tabelle 5.4 sind die Kostenwerte für die zehn Durchläufe und ihre Mittelwerte für das Schedule-Into-Robustheitsmaß gegeben. Auch hier kann kein eindeutiger Trend in Bezug auf die Mittelwerte festgelegt werden. Die Mittelwerte für eine Länge der Tabuliste von 7 und 100 sind die besten erzielten Mittelwerte, wohingegen für die Länge der Tabuliste 15 der schlechteste Wert erzielt wurde.

Sowohl bei dem Fit-Into-Robustheitsmaß als auch dem Schedule-Into-Robustheitsmaß lässt sich kein klarer Trend erkennen, welche Größe für die Länge der Tabuliste am besten geeignet ist. Während beim Fit-Into-Robustheitsmaß die mittleren Kostenwerte nur gering voneinander abweichen, werden beim Schedule-Into-Robustheitsmaß die besten mittleren Kostenwerte für eine Timeslotgröße von 7 beziehungsweise 100 erreicht.

Da sich mit den ermittelten Werten nicht feststellen lässt, welche Tabulänge für unseren Algorithmus am besten geeignet ist, beim Fit-Into-Robustheitswert der beste Mittelwert für die Tabulänge 25 und für das Schedule-Into-Robustheitsmaß die Tabulänge 100 einen guten Mittelwert geliefert hat, werden wir im Folgenden 50 als Länge der Tabuliste wählen.

Ermitteln einer geeigneten Timeslotgröße

Um eine geeignete Timeslotgröße zu ermitteln, testen wir die Werte entsprechend der Tabelle 5.2 für eine festgelegte Anzahl von 2000 Iterationen. Wir vergleichen im Anschluss, wie viel Zeit der Algorithmus für die betrachteten Timeslotgrößen bei 2000 Iterationen benötigt hat. Außerdem betrachten wir den besten ermittelten Kostenwert und die Anzahl der benötigten Iterationen, bis die beste Lösung gefunden wurde. Erneut testen wir den Algorithmus mit einer Robustheitsschranke $a = 50$ und führen zehn Durchläufe aus.

Für das Fit-Into-Robustheitsmaß ist der Zusammenhang des erzielten Kostenwertes in Abhängigkeit von der Größe der Timeslots in Abbildung 5.1 dargestellt.

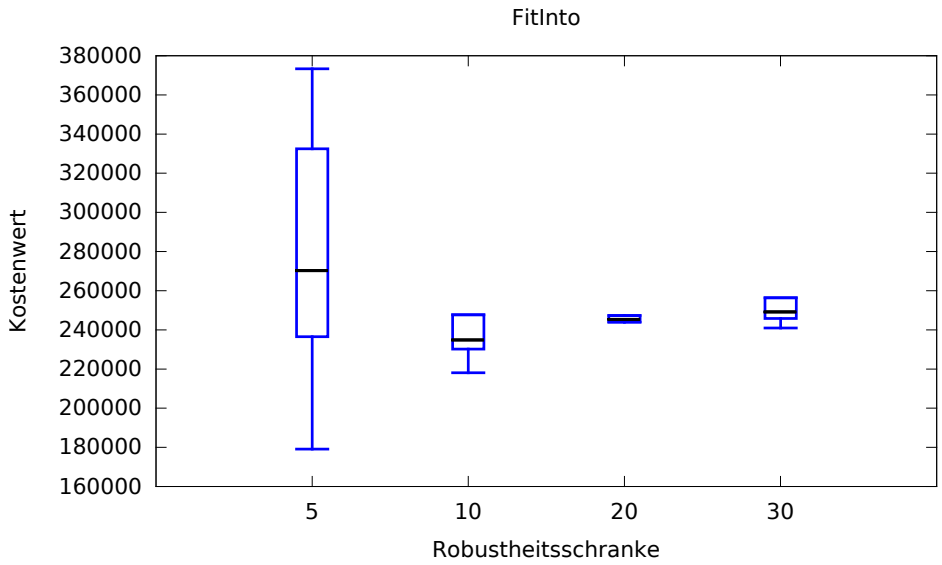


Abbildung 5.1: Bester Kostenwert bei gegebener Timeslotgröße für 2000 Iterationen.

In Abbildung 5.1 ist zu erkennen, dass im Schnitt bei einer Timeslotgröße von 10 die besten Kostenwerte erreicht werden. Bei einer Timeslotgröße von 20 unterliegt der Kostenwert den geringsten Schwankungen.

Die nachfolgende Abbildung 5.2 zeigt, wie viele Iterationen benötigt wurden, um im Falle des Fit-Into-Robustheitsmaßes bis zur besten ermittelten Lösung zu gelangen.

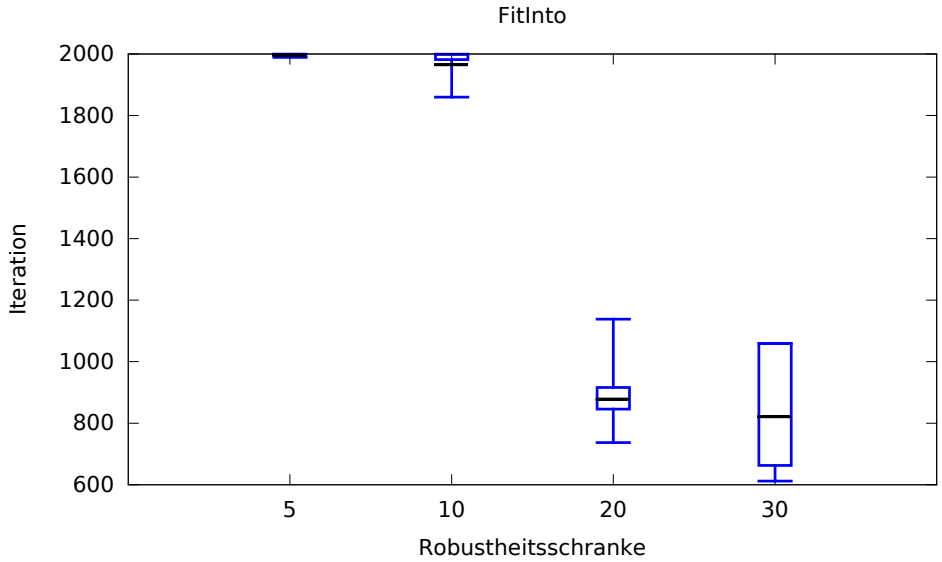


Abbildung 5.2: Anzahl benötigter Iterationen bis zur besten gefundenen Lösung bei maximal 2000 Iterationen.

In Abbildung 5.2 ist veranschaulicht, dass für eine kleinere Größe der Timeslots mehr Iterationen benötigt werden, um zu der besten ermittelten Lösung zu gelangen. Es ist zu erkennen, dass für die Timeslotgrößen 5 und 10 nahezu immer in den letzten Iterationen noch eine verbessernde Lösung gefunden wurde. Möchte man die Timeslotgröße 5 oder 10 betrachten, so sollte eine höhere Anzahl von Iterationen gewählt werden.

Für das Schedule-Into-Robustheitsmaß liefern die Tests ähnliche Ergebnisse.

In Abbildung 5.3 ist der Zusammenhang zwischen erzieltem Kostenwert und Größe der Timeslots für das Schedule-Into-Robustheitsmaß dargestellt.

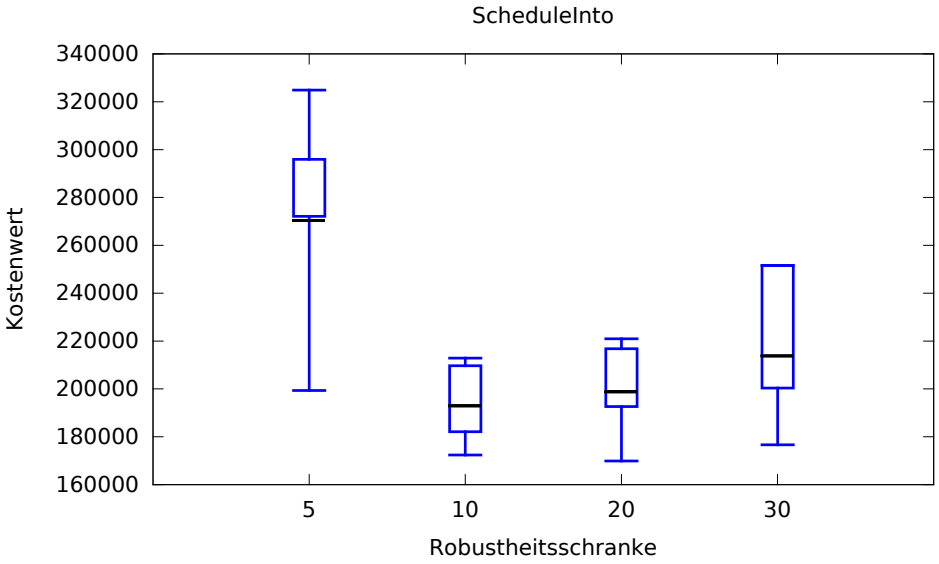


Abbildung 5.3: Bester Kostenwert bei gegebener Timeslotgröße für 2000 Iterationen.

Anhand der Abbildung 5.3 sieht man, dass beim Schedule-Into-Robustheitsmaß für eine Timeslotgröße von 10 im Schnitt die besten Kostenwerte erreicht wurden. Betrachtet man die nachfolgende Abbildung 5.2, die die Anzahl der benötigten Iteration bis zur besten ermittelten Lösung dargestellt, so hätte auch in diesem Fall die Anzahl der Iterationen im Fall einer Timeslotgröße von 5 oder 10 größer sein dürfen, da dies die Kostenwerte weiter verbessern könnte.

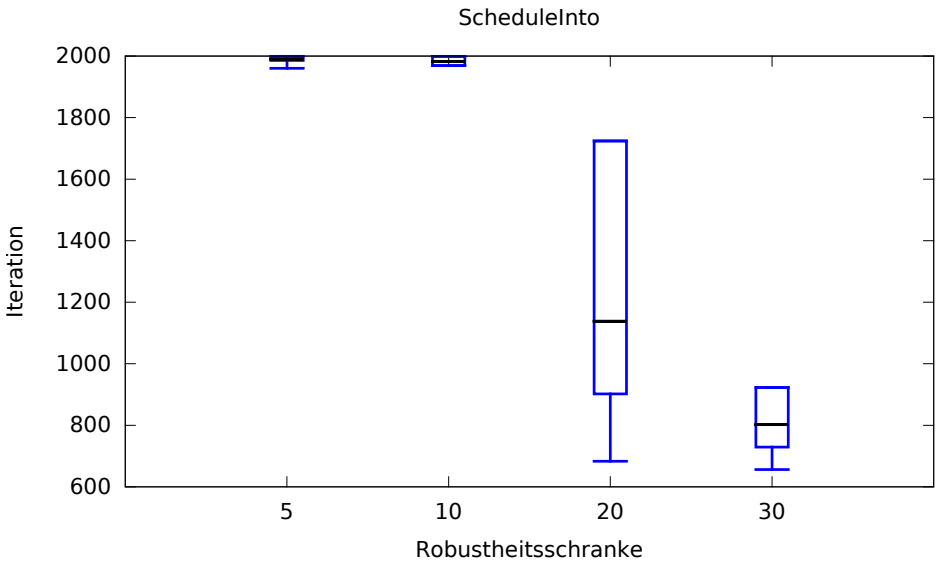


Abbildung 5.4: Anzahl benötigter Iterationen bis zur besten gefundenen Lösung bei maximal 2000 Iterationen.

Aus den obigen Abbildungen geht hervor, dass mit einer kleineren Größe der Timeslots eventuell bessere Lösungen gefunden werden können, dazu allerdings mehr Iterationen benötigt werden.

Da bei kleineren Werten für die Größe der Timeslots die Kardinalität einer Nachbarschaft einer Lösung wächst, möchten wir in unsere Entscheidung mit einbeziehen, wie viel Zeit der Algorithmus bei gegebener Timeslotgröße für 2000 Iterationen benötigt. Die ermittelten Werte sind in Tabelle 5.5 dargestellt.

T_{TS}	Anzahl Timeslots pro Tag	Benötigte Zeit für 2000 Steps in Minuten
5	96	22.0832
10	32	12.4479
20	24	8.2043
30	16	6.8467

Tabelle 5.5: Benötigte Zeit für 2000 Steps bei Variation der Größe der Timeslots.

Die Tabelle 5.5 zeigt, dass mit kleiner werdender Größe der Timeslots die Zeit für das Durchsuchen einer Nachbarschaft steigt.

Da für die Timeslotgröße 20 annähernd so gute Kostenwerte wie bei einer Timeslotgröße von 10 Minuten erzielt wurden, der Algorithmus in diesem Fall aber deutlich weniger Iterationen benötigt und innerhalb von zehn Minuten zu der besten gefundenen Lösung gelangt, wählen wir im Folgenden 20 Minuten für die Größe der Timeslots.

Nachdem wir nun eine Größe für die Tabuliste und die Timeslots festgelegt haben, möchten wir ermitteln, wie sich das Verhältnis von Robustheits- und Kostenwert bei Variation der Robustheitsschranke verändert.

Zusammenhang von Robustheitsschranke und Kostenwert

Um zu ermitteln, wie sich das Verhältnis von Robustheits- und Kostenwert bei Variation der Robustheitsschranke verändert, betrachten wir die in Tabelle 5.6 gegebenen Werte für die Robustheitsschranke.

Parameter	Werte
a	$\{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$

Tabelle 5.6: Werte für die Robustheitsschranke.

Wir testen den Algorithmus für eine Timeslotgröße von 20 Minuten bei einem Zeitlimit von 10 Minuten und führen zehn Durchläufe durch.

In Abbildung 5.5 und Abbildung 5.6 ist der erzielte Kostenwert in Abhängigkeit von der Robustheitsschranke einmal für das Fit-Into-Robustheitsmaß und einmal für das Schedule-Into-Robustheitsmaß dargestellt.

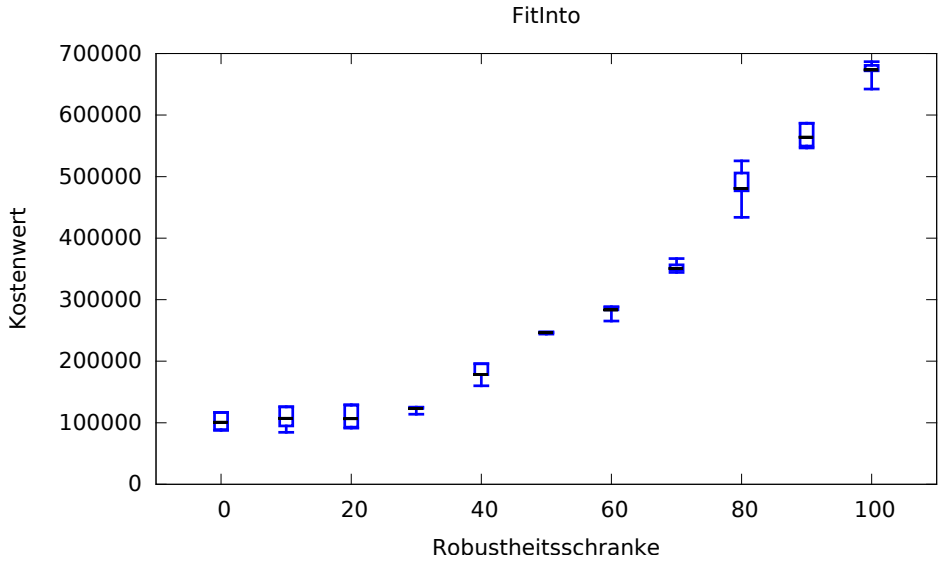


Abbildung 5.5: Zusammenhang zwischen Kostenwert und Robustheitsschranke für das Fit-Into-Robustheitsmaß.

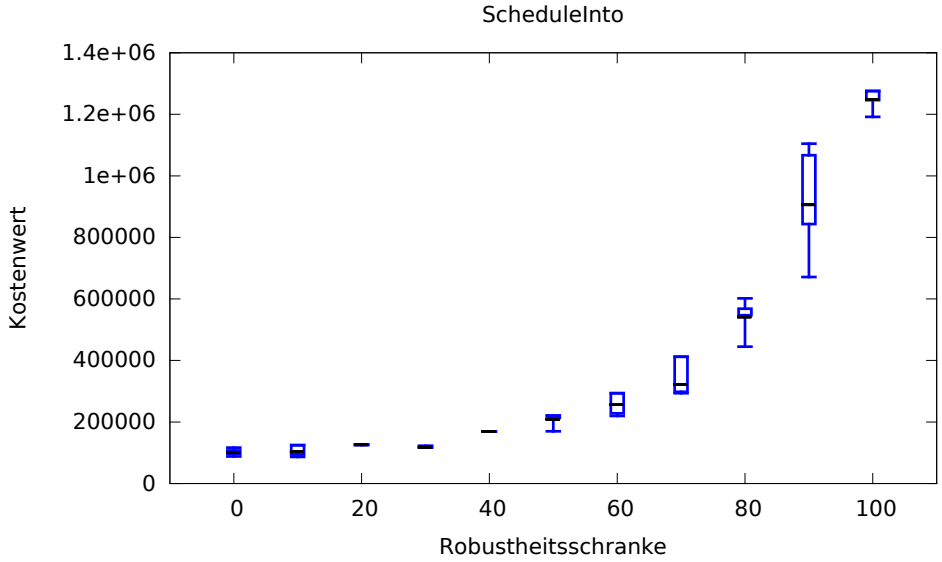


Abbildung 5.6: Zusammenhang zwischen Kostenwert und Robustheitsschranke für das Schedule-Into-Robustheitsmaß.

In den Abbildungen ist zu erkennen, dass der Kostenwert mit größer werdender Robustheitsschranke steigt. Dieses entspricht unseren Erwartungen. Je besser eine Lösung gegen das Auftreten von Notfallpatienten abgesichert ist, desto schlechter wird der Kostenwert.

Für das Schedule-Into-Robustheitsmaß sind die Kostenwerte bei hoher Robustheitsschranke deutlich schlechter als für das Fit-Into-Robustheitsmaß. Dies liegt daran, dass beim Schedule-Into-Robustheitsmaß eine Lösung zu hundert Prozent robust ist, wenn kein Patient auf den Beginn einer Behandlung warten muss. Beim Fit-Into-Robustheitsmaß gilt eine Lösung hingegen bereits als hundert Prozent robust, wenn alle Behandlungen eines Patienten im Laufe der Öffnungszeiten seines Ankunftstages stattfinden können. In diesem Fall darf der Patient zwischen zwei Behandlungen warten, insofern er bis zum Ende der Öffnungszeiten alle Behandlungen erhalten hat.

Im Laufe des Algorithmus wird ein Schedule mit Hilfe eines unkorrelierten Szenarios abgesichert. Wir möchten nun testen, wie robust ein Schedule, der zu einer gegebenen Robustheitsschranke ermittelt wurde, in Bezug auf korrelierte Szenarien ist. Für die verschiedenen Robustheitsschranken testen wir einen ermittelten Schedule gegen 100 korrelierte Szenarien. Die Ergebnisse sind in Abbildung 5.7 und Abbildung 5.8 dargestellt.

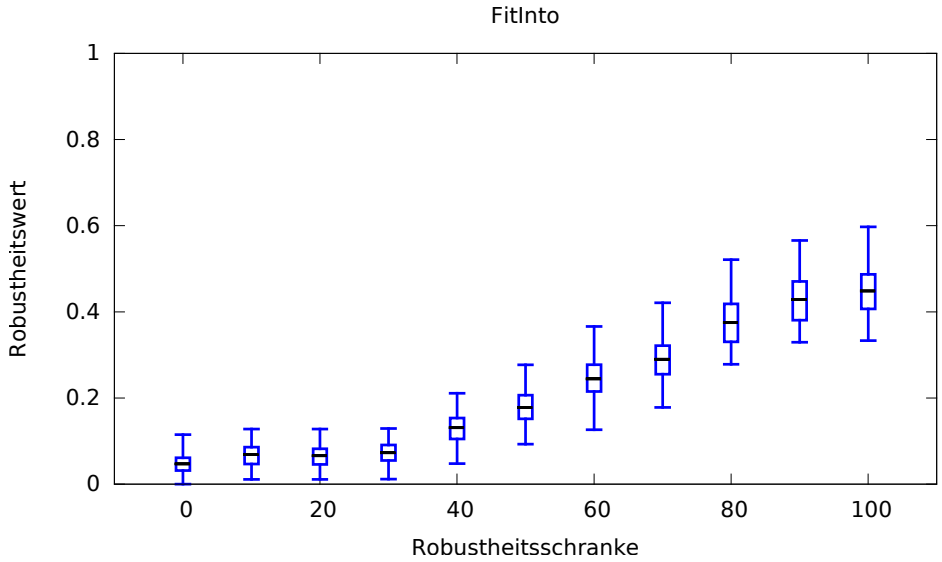


Abbildung 5.7: Zusammenhang zwischen Robustheitsschranke und -wert für das Fit-Into-Robustheitsmaß.

Die Abbildung 5.7 zeigt, welche Robustheitswerte für einen Schedule, der zu einer gegebenen Robustheitsschranke ermittelt wurde, für das Fit-Into-Robustheitsmaß für die betrachtete korrelierte Szenarienmenge erzielt wurden.

In Abbildung 5.8 ist dieselbe Beziehung für das Schedule-Into-Robustheitsmaß dargestellt.

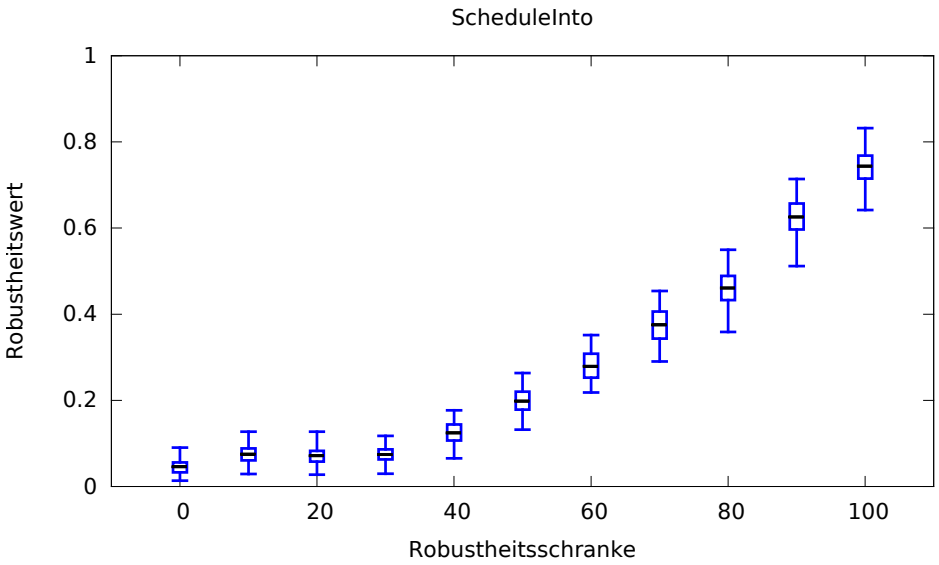


Abbildung 5.8: Zusammenhang zwischen Robustheitsschranke und -wert für das Schedule-Into-Robustheitsmaß.

In Abbildung 5.7 und Abbildung 5.8 ist zu erkennen, dass die Robustheit des Schedules für eine größere Robustheitsschranke zunimmt. Allerdings liegen die ermittelten Robustheitswerte deutlich unter der Robustheitsschranke, die beim Testen gegen unkorrelierte Szenarien erzielt wurden. Durch die Beziehungen der Behandlungen in den korrelierten Szenarien ist es schwieriger diese in die Freiräume eines Schedules einzuplanen.

Price of Robustness

Für die Robuste Lokale Suche mit Akzeptor haben wir in Abschnitt 3.2.1 den Price of Robustness definiert. Daher möchten wir diesen im Folgenden für die Mittelwerte der Kostenwerte aus dem obigen Test bestimmen.

	FitInto		**ScheduleInto**	
Robustheits-faktor	Mittelwert	Price of Robustness	Mittelwert	Price of Robustness
0	100842	0	100194	0
10	107278	6.38	102890	2.69
20	106594	5.7	126184	25.94
30	122829	21.8	118634	18.4
40	178501	77.01	169191	68.86
50	246300	144.24	208579	108.18
60	283557	181.19	256609	156.11
70	350560	247.63	321413	220.79
80	480882	376.87	541194	440.14
90	563968	459.56	905889	804.13
100	673412	567.79	1248270	1145.85

Tabelle 5.7: Price of Robustness gemessen an den Mittelwerten.

In Tabelle 5.7 ist der Price of Robustness für die beiden Robustheitsmaße in Bezug zur Robustheitsschranke angegeben. Mit zunehmender Robustheitsschranke, ab einem Wert von 40, steigt der Wert des Price of Robustness an.

5.4.2 Robuste Lokale Suche mit Konvexkombination

Um die Vergleichbarkeit der Ergebnisse von Robuster Lokaler Suche mit Akzeptor und mit Konvexkombination zu gewährleisten, beginnen wir die Robuste Lokale Suche mit derselben Startlösung wie die Robuste Lokale Suche mit Akzeptor.

Um die Robuste Lokale Suche mit Konvexkombination anwenden zu können, benötigen wir einen Richtwert. Für unsere Tests ermitteln wird den Richtwert ähnlich zur Startlösung. Wir bestimmen eine Lösung, in der wir die letzten d Prozent der Öffnungszeiten eines Arbeitstages frei lassen, und wählen deren Kostenwert als Richtwert. Allerdings weisen wir bei der Ermittlung der Startzeiten immer einer Behandlung eine Startzeit zu, die zum aktuellen Stand der Zuweisung am frühestmöglichen stattfinden kann.

Um die Robuste Lokale Suche geeignet testen zu können, testen wir den Algorithmus für unterschiedliche Tabulängen.

Ermittlung einer geeigneten Tabulänge

Wir testen die betrachtete Instanz für eine Robustheitsschranke von $d = 50$ und ein Zeitlimit von 10 Minuten für eine Timeslotgröße von 20 Minuten. Die Kostenwerte der kombinierten Zielfunktion sind für zehn Durchläufe sowie deren Mittelwerte in der nachfolgenden Tabellen 5.8 und 5.9 angegeben.

L_{Tabu}	7	10	15	25	50	100
Durchlauf 1	313021	308569	305032	310868	317561	319323
Durchlauf 2	313029	285338	292383	291613	303772	294704
Durchlauf 3	313543	289507	289342	307474	308349	319793
Durchlauf 4	312297	306539	308695	307805	312051	307125
Durchlauf 5	318928	337008	308965	282254	304998	281011
Durchlauf 6	315328	304135	308947	337050	284949	278301
Durchlauf 7	318139	304727	313027	305163	278552	278658
Durchlauf 8	289259	305694	303817	325783	315728	303471
Durchlauf 9	312115	336580	319086	336057	330362	325540
Durchlauf 10	342139	319782	334218	306151	330085	307051
Mittelwert	3147798	3097879	3083512	3110218	3086407	3014977

Tabelle 5.8: Kostenwerte bei mögliche Tabulängen für das Fit-Into-Robustheitsmaß.

Für das Fit-Into-Robustheitsmaß wird der beste Mittelwert im Fall einer Tabulänge von 100 erzielt. Der beste erreichte Kostenwert unter allen ermittelten Kostenwerten wurde bei einer Tabuliste der Länge 25 erreicht.

L_{Tabu}	7	10	15	25	50	100
Durchlauf 1	323160	311804	319000	454556	309826	331228
Durchlauf 2	290702	348355	346187	298344	322108	329520
Durchlauf 3	323575	317348	345649	348298	296423	319952
Durchlauf 4	322475	319881	337762	292491	330530	329756
Durchlauf 5	340668	303238	300998	337543	330183	345856
Durchlauf 6	322921	279182	297786	279133	330930	309894
Durchlauf 7	341022	333027	329565	321941	272624	344329
Durchlauf 8	331795	297377	373328	361995	306879	378067
Durchlauf 9	322882	335425	343562	362095	328458	300364
Durchlauf 10	332204	340688	284820	346798	318720	295804
Mittelwert	3251404	3186325	3278657	3403194	3146681	3284770

Tabelle 5.9: Kostenwerte bei mögliche Tabulängen bei Betrachtung des Schedule-Into-Robustheitsmaßes.

Bei Betrachtung des Schedule-Into-Robustheitsmaßes wurden sowohl der beste Mittelwert als auch der beste Kostenwert bei einer Tabulänge von 50 erreicht.

Sowohl bei dem Fit-Into-Robustheitsmaß als auch bei dem Schedule-Into-Robustheitsmaß lässt sich kein klarer Trend erkennen. Die Mittelwerte schwanken mit größer werdender Länge der Tabuliste. Daher lässt sich mit den ermittelten Werten nicht feststellen, welche Tabulänge für unseren Algorithmus am besten geeignet ist. Anhand der ermittelten Werte entscheiden wir uns im Folgenden für eine Tabuliste, die 50 Undo-Moves beinhaltet.

Ermitteln einer geeigneten Timeslotgröße

Da wir die Ergebnisse der Robusten Lokalen Suche mit Konvexkombination im Anschluss an diesen Abschnitt mit der Robusten Lokalen Suche mit Akzeptor vergleichen möchten, wählen wir die Größe der Timeslots äquivalent zu unseren Untersuchungen im Fall der Robusten Lokalen Suche mit Akzeptor. Daher betrachten wir im Folgenden Timeslots mit einer Dauer von 20 Minuten.

Nachdem wir eine Größe für die Tabuliste und die Timeslots festgelegt haben, möchten wir ermitteln, wie sich das Verhältnis von Robustheits- und Kostenwert bei Variation des Robustheitsfaktors verändert.

Zusammenhang von Robustheitsfaktor und Kostenwert

Um zu ermitteln, wie sich das Verhältnis von Robustheits- und Kostenwert bei Variation des Robustheitsfaktors verändert, betrachten wir die in Tabelle 5.10 gegebenen Werte für die Robustheitsschranke.

Parameter	**Werte**
d	$\{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$

Tabelle 5.10: Werte für den Robustheitsfaktor.

Wir testen die Robustheitsfaktoren entsprechend der Tabelle 5.10 für die betrachtete Instanz bei einem Zeitlimit von zehn Minuten und führen zehn Durchläufe aus.

Die kombinierte Kostenfunktion, anhand der gemessen wird, wie gut eine Lösung ist, setzt sich aus einer Kombination von Robustheitswert und Kostenwert der Lösung zusammen. Daher betrachten wir, wie sich zum einen der Kostenwert und zum anderen der Robustheitswert bei Variation des Robustheitsfaktors verändert.

In Abbildung 5.9 und Abbildung 5.10 ist der Zusammenhang zwischen Robustheitsfaktor und Kostenwert graphisch dargestellt.

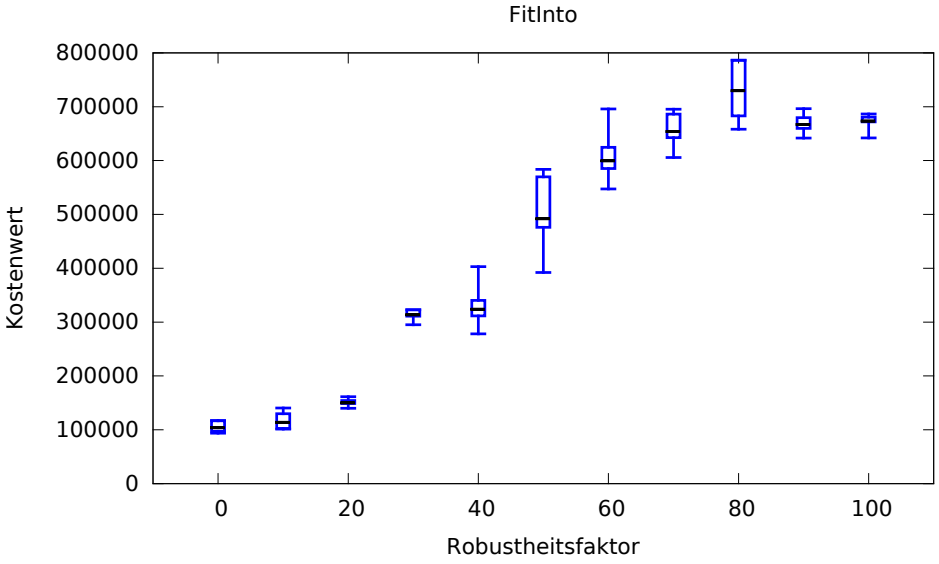


Abbildung 5.9: Zusammnehang von Robustheitsfaktor und Kostenwert einer Lösung für das Fit-Into-Robustheitsmaß.

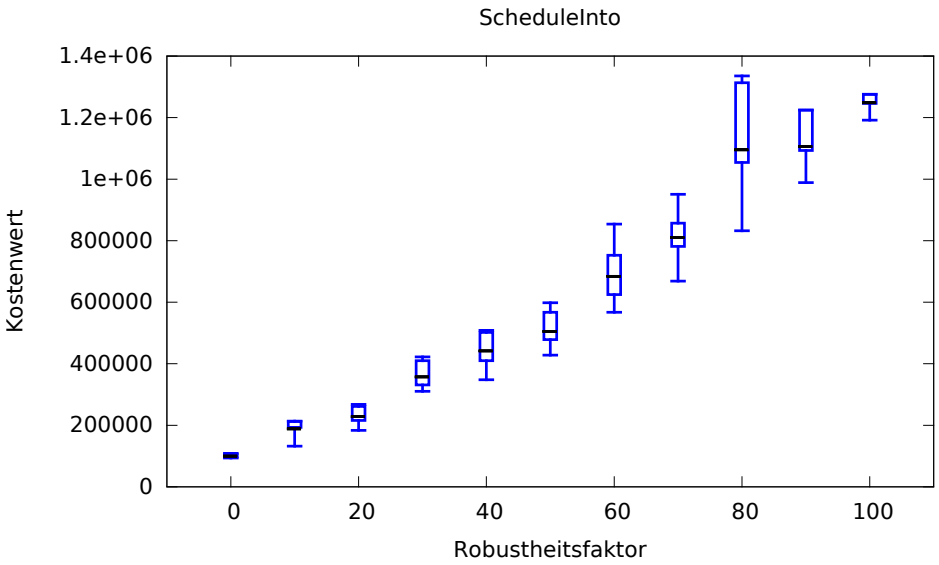


Abbildung 5.10: Zusammenhang von Robustheitsfaktor und Kostenwert für das Schedule-Into-Robustheitsmaß.

Die Abbildungen 5.9 und 5.10 veranschaulichen, dass der Kostenwert, entsprechend unserer Erwartungen, bei steigendem Robustheitsfaktor zunimmt. Ähnlich verhält sich der Robustheitswert.

In Abbildung 5.11 und Abbildung 5.12 ist der Zusammenhang zwischen Robustheitsfaktor und Robustheitswert einer Lösung veranschaulicht.

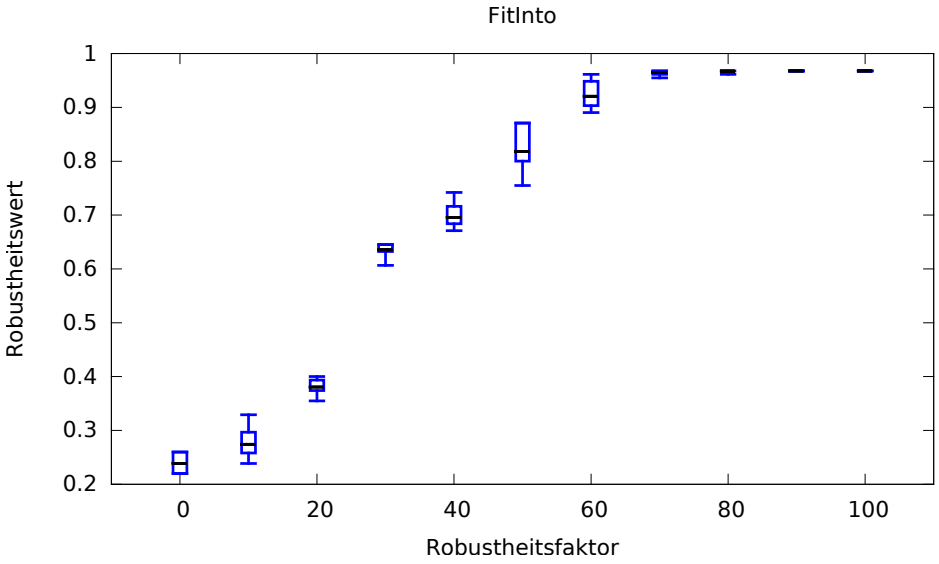


Abbildung 5.11: Zusammenhang von Robustheitsfaktor und -wert für das Fit-Into-Robustheitsmaß.

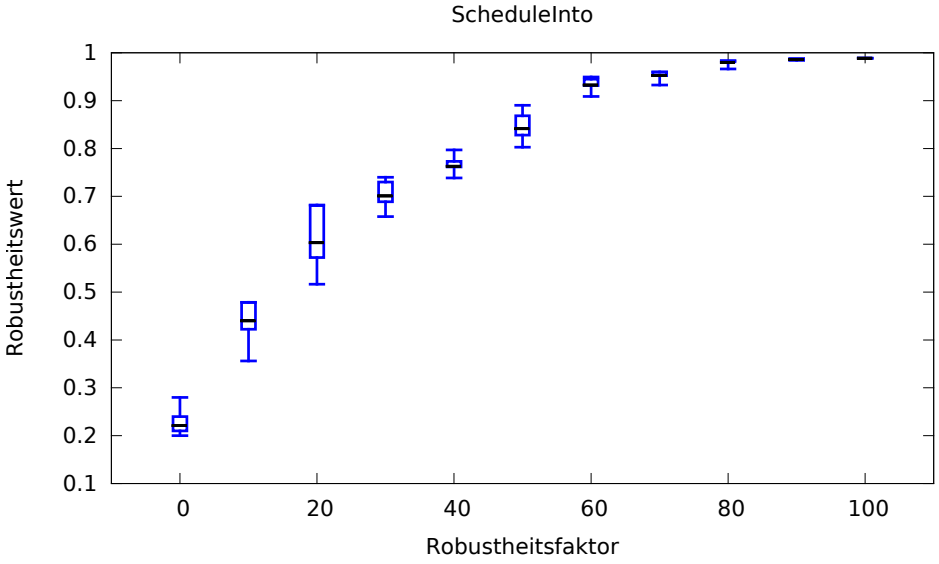


Abbildung 5.12: Zusammenhang von Robustheitsfaktor und -wert für das Schedule-Into-Robustheitsmaß.

Die Abbildungen 5.11 und 5.12 zeigen, dass mit zunehmendem Robustheitsfaktor der Robustheitswert einer Lösung zunimmt. Darüber hinaus ist zu erkennen, dass ein hoher Robustheitswert bereits bei einem Robustheitsfaktor von 60 erreicht wird. Dies deutet darauf

hin, dass der Robustheitswert überproportional in die kombinierte Kostenfunktion einbezogen wird und der Richtwert für diese Robustheitsfaktoren kleiner gewählt werden könnte.

Die gerade betrachteten Robustheitswerte sind die Werten bei der Absicherung gegen ein unkorreliertes Szenario im Algorithmus. Wie sich eine auf solche Weise ermittelte Lösung bei einem Test gegen korrelierte Szenarien verhält, ist beispielsweise für eine zu einem Robustheitsfaktor ermittelten Lösung in den nachfolgenden Abbildungen 5.13 und 5.14 dargestellt.

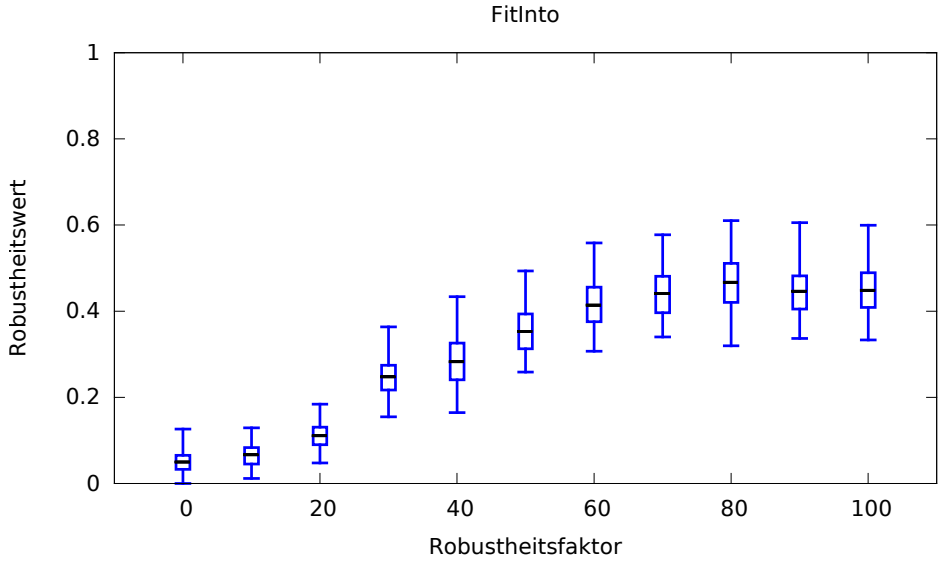


Abbildung 5.13: Zusammenhang von Robustheitsfaktor und -wert für das Fit-Into-Robustheitsmaß.

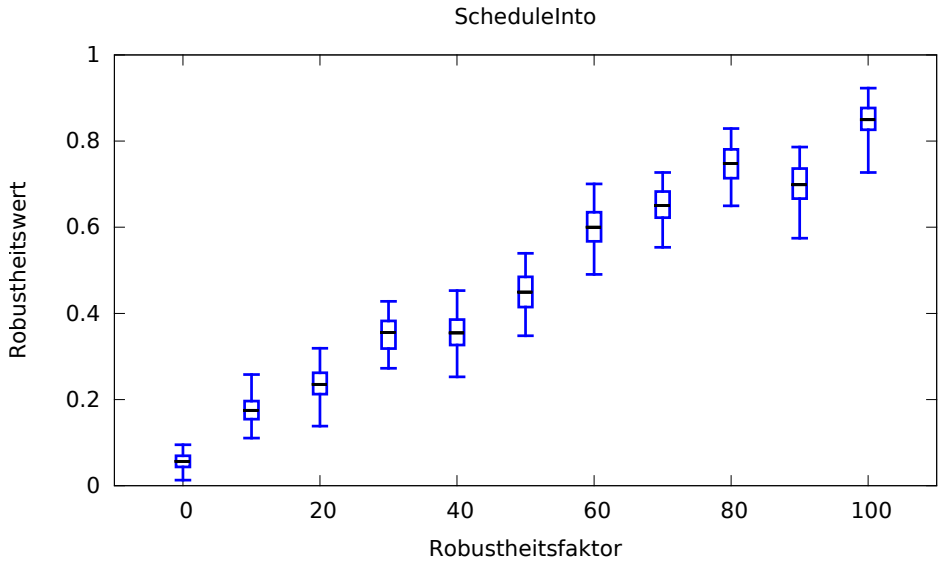


Abbildung 5.14: Zusammenhang von Robustheitsfaktor und -wert für das Schedule-Into-Robustheitsmaß.

In den Abbildungen ist zu erkennen, dass für korrelierte Szenarien schlechtere Robustheitswerte erzielt wurden als beim betrachteten unkorrelierten Szenario.

Kommen wir nun zum Vergleich der beiden Verfahren.

5.4.3 Vergleich der beiden Ansätze einer Robusten Lokalen Suche für das Krankenhaus Scheduling Problem unter Unsicherheiten

Um die beiden Verfahren zu vergleichen, betrachten wir alle Kombinationen des Robustheits- und Kostenwertes, die für unterschiedliche Robustheitsschranken beziehungsweise -faktoren für die gegebene Instanz bei einem Zeitlimit von 10 Minuten und einer Timeslotgröße von 20 Minuten aufgetreten sind.

In Abbildung 5.15 sind die Robustheits-Kostenwert-Kombinationen für das Fit-Into-Robustheitsmaß dargestellt.

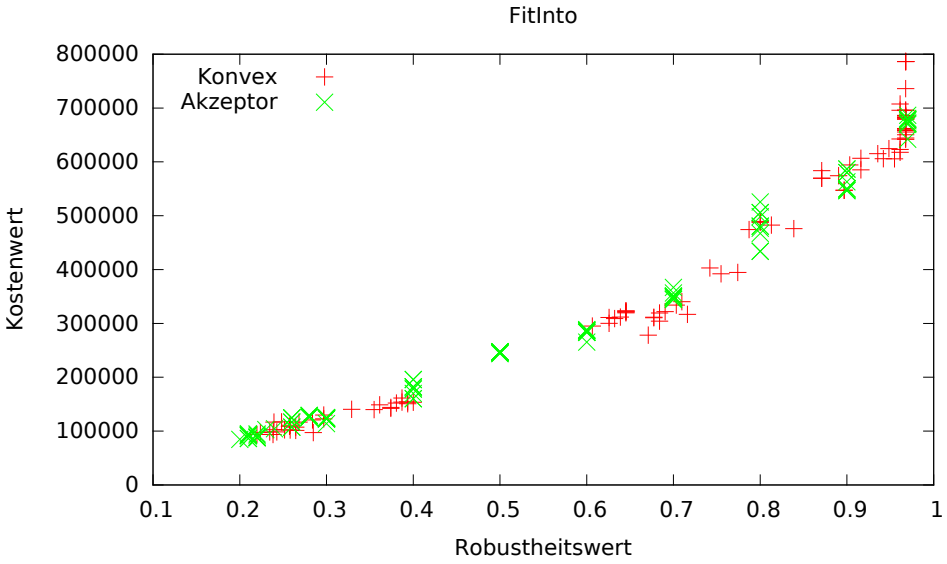


Abbildung 5.15: Wert für das Fit-Into-Robustheitsmaß im Zusammenhang mit dem Kostenwert einer besten ermittelten Lösung.

Anhand Abbildung 5.15 ist zu erkennen, dass egal welcher Ansatz gewählt wurde, ähnliche Kombinationen erreicht werden. Es kann daher nicht gesagt werden, dass ein Ansatz im Vergleich zum anderen zu besseren Kombinationen gelangt.

In Abbildung 5.16 sind die Robustheits-Kostenwert-Kombinationen für das Schedule-Into-Robustheitsmaß dargestellt.

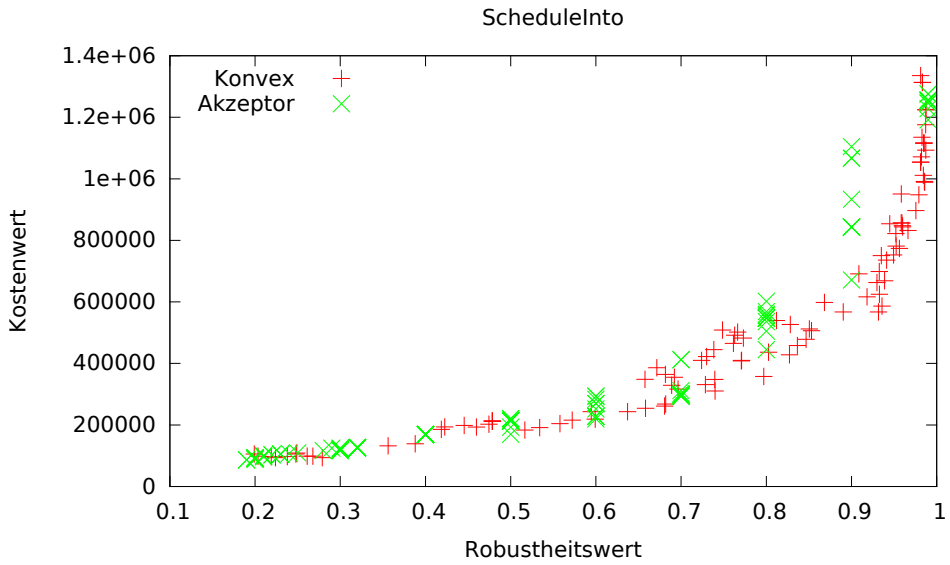


Abbildung 5.16: Wert für das Schedule-Into-Robustheitsmaß im Zusammenhang mit dem Kostenwert einer besten ermittelten Lösung.

In der Abbildung 5.16 ist zu erkennen, dass die Kostenwerte bei einer gegebenen Robustheitsschranke für die Robuste Lokale Suche mit Akzeptor stark variieren können. Die Robuste Lokale Suche mit Konvexkombination scheint für das Schedule-Into-Robustheitsmaß der Robusten Lokalen Suche mit Akzeptor überlegen zu sein.

5.4.4 Weiterführende Experimente

In den oben beschriebenen Experimenten wurde die Robustheit einer Lösung in jeder Iteration nur anhand eines unkorrelierten Szenarios bewertet. Daher widmen wir uns der Frage, ob eine Lösung robuster wird, wenn sie in jeder Iteration gegen eine Menge von unkorrelierten Szenarien absichert wird.

Level der Absicherung

Wir möchten den Robustheitswert einer Lösung in jeder Iteration gegenüber einer gegebenen Menge von unkorrelierten Szenarien bestimmen. Dazu ermitteln wir für jedes Szenario den Robustheitswert entsprechend dem gegebenen Robustheitsmaß und bilden den Mittelwert über diese Werte. In unserem Test betrachten wir verschieden Kardinalitäten für die Menge der Szenarien, gegen die die Lösung abgesichert werden soll. Die betrachteten Anzahlen der Szenarien in einer Menge, im Folgenden auch Absicherungslevel genannt, sind in Tabelle 5.11 aufgelistet.

Parameter	Werte
Absicherungslevel	$\{1, 25, 50, 100\}$

Tabelle 5.11: Anzahl von Szenarien in verschiedenen Absicherungsleveln.

Wir führen für jedes Absicherungslevel aus der Tabelle 5.11 einen Durchlauf mit einer auf 1200 festgelegten Anzahl von Iterationen bei einer Timeslotgröße von 20 Minuten aus. Wir beschränken uns bei dieser Untersuchung auf die Betrachtung der Robusten Lokalen Suche mit Akzeptor.

Bei der Absicherung einer Lösung gegen mehrere Szenarien nimmt die Zeit zu, die zu dem Durchsuchen einer Nachbarschaft benötigt wird. Die benötigten Zeiten für 1200 Iterationen für die betrachteten Absicherungslevel sind in Tabelle 5.12 gegeben.

Absicherungslevel	**Benötigte Zeit für** 1200 **Iterationen in Minuten**
1	4.9394
25	52.7961
50	104.5638
100	205.9866

Tabelle 5.12: Benötigte Zeit für 1200 Iterationen bei Variation des Absicherungslevels.

Die Werte der Tabelle 5.11 machen deutlich, dass die Dauer für die Durchführung von 1200 Iterationen mit größer werdendem Absicherungslevel stark ansteigt.

In den nachfolgenden Abbildungen 5.17 und 5.18 ist dargestellt, wie sich die Lösung, die zu einer gegebenen Robustheitsschranke für eine Szenarienmenge entsprechend des betrachteten Absicherungslevels abgesichert wurde, beim Testen gegenüber 100 korrelierte Szenarien schlägt.

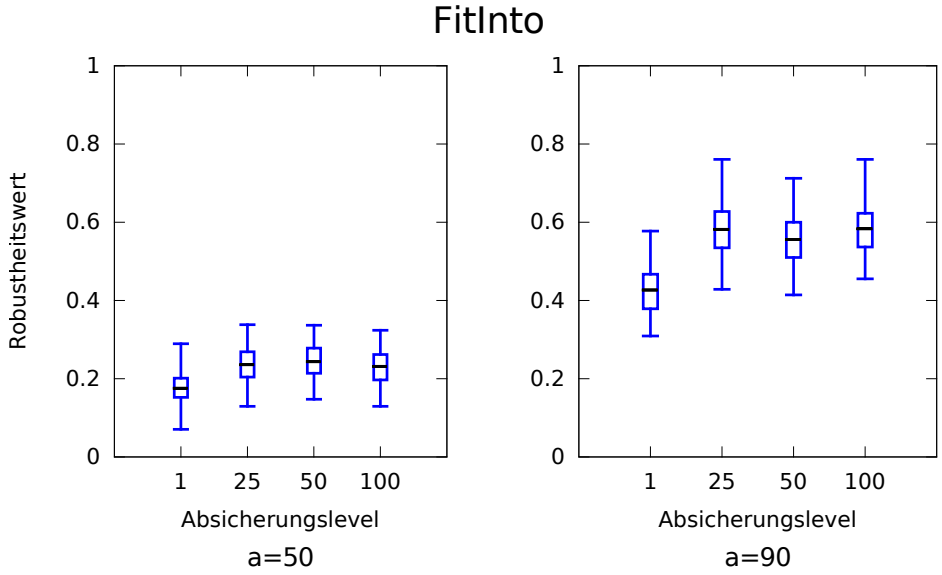


Abbildung 5.17: Robustheit bei verschiedenen Absicherungsleveln für das Fit-Into-Robustheitsmaß.

In Abbildung 5.17 ist zu erkennen, dass für das Fit-Into-Robustheitsmaß der durchschnittliche Robustheitswert bei der Absicherung gegen 25 Szenarien im Vergleich zu der Absicherung gegen ein Szenario steigt, wohingegen eine Absicherung gegen eine Szenarienmenge

mit mehr als 25 Elementen im Vergleich zu dem erreichten Absicherungslevel bei Betrachtung von 25 Szenarien keine Verbesserung des durchschnittlichen Robustheitswertes mit sich bringt.

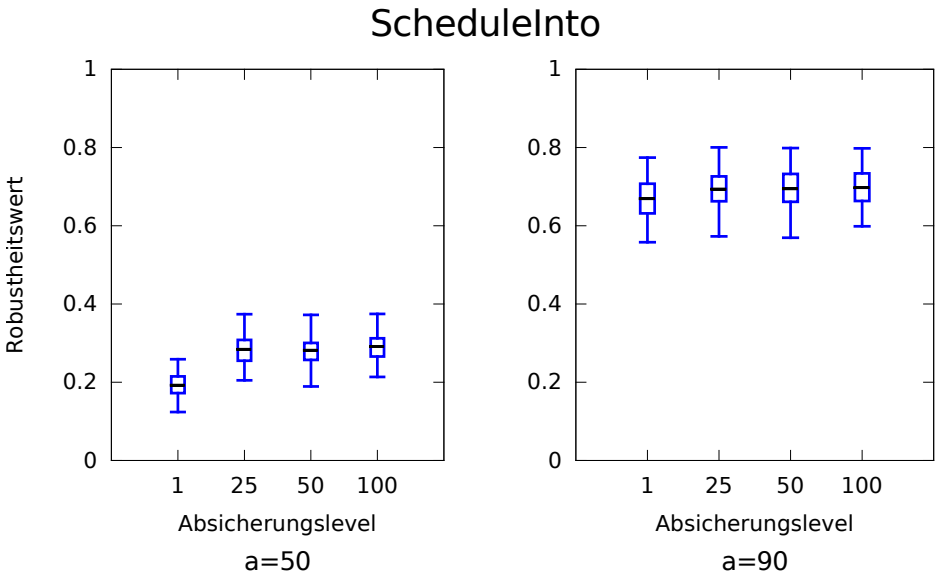


Abbildung 5.18: Robustheit bei verschiedenen Absicherungsleveln für das Schedule-Into-Robustheitsmaß.

Beim Schedule-Into-Robustheitsmaß hat das Level der Absicherung, wie in Abbildung 5.18 zu erkennen ist, bei einer Robustheitsschranke von 50 große Auswirkungen auf den durchschnittlichen Robustheitswert. Ähnlich wie beim Fit-Into-Robustheitsmaß gibt es einen großen Sprung der Robustheit zwischen den Absicherungsleveln 1 und 25. Für größere Absicherungslevel verbessert sich der durchschnittliche Robustheitswert nicht mehr. Bei einer Robustheitsschranke von 90 sind die Auswirkungen der verschiedenen Level der Absicherung hingegen gering. Dies kann daran liegen, dass sich bei einer Robustheitsschranke von 90 bereits so große Lücken in dem Schedule befinden, dass sich die genauen Ankunftszeitpunkte von Patienten nicht auf den Robustheitswert auswirken.

Wir haben gesehen, dass bis zu einem gewissen Maß mit einem größeren Level der Absicherung die Robustheit einer Lösung gesteigert werden kann. Im vorherigen Abschnitt haben wir festgestellt, dass ein steigender Robustheitswert mit einem steigenden Kostenwert einhergeht. Daher möchten wir betrachten wie sich der Kostenwert der Lösungen bei unterschiedlichen Absicherungsleveln verändert hat.

	FitInto		**ScheduleInto**	
Absicherunslevel	$a = 50$	$a = 90$	$a = 50$	$a = 90$
1	247341	588520	214704	1073507
25	229791	993750	231610	1009252
50	258240	875946	205434	990890
100	220353	993855	198560	1098615

Tabelle 5.13: Veränderung des Kostenwertes bei Variation des Absicherungslevels.

In Tabelle 5.13 sind die Kostenwerte der ermittelten Schedules für eine Robustheitsschranke von 50 und 90 für die betrachteten Absicherungslevel für beide Robustheitsmaße angegeben. Nur im Fall des Fit-Into-Robustheitsmaßes bei einer Robustheitsschranke von 90 ist eine deutliche Steigerung des Kostenwertes bei zunehmender Robustheit zu erkennen. In den anderen Fällen schwanken die Kostenwerte im üblichen Rahmen. Es kann also gesagt werden, dass die Absicherung gegen mehrere Szenarien zu robusteren Schedules führt, ohne dass sich der Kostenwert der Schedules steigert. Auch gibt es anscheinend eine Grenze für das Absicherungslevel, ab der eine Steigerung des Absicherungslevel nicht mehr zu einer Verbesserung der Robustheit eines Schedules führt.

Als weiteres Experiment möchten wir testen, wie sich bei Variation der maximalen Anzahl von Behandlungen eines Notfallpatienten der Robustheitswert eines Schedules gegenüber dem Auftreten solcher Notfallpatienten verändert.

Variation der maximalen Behandlungszahl der Notfallpatienten

Wir beschränken unsere Betrachtungen auf den Fall einer Robusten Lokalen Suche mit Akzeptor. Zudem betrachten wir ausschließlich das Fit-Into-Robustheitsmaß, da wir für dieses schlechte Robustheitswerte beim Testen gegen 100 Szenarien mit Notfallpatienten mit bis zu fünf Behandlungen erhalten haben.

Für Robustheitsschranken gemäß Tabelle 5.6 ermitteln wir einen Schedule, der im Anschluss gegen jeweils 100 Szenarien mit unterschiedlicher maximaler Behandlungsanzahl pro Patient getestet wird. Das Zeitlimit ist auf 10 Minuten beschränkt und die Größe der Timeslots auf 20 Minuten festgelegt. Wir beschränken uns auf die Absicherung gegenüber einem Szenario.

Wir testen einen Schedule gegen fünf verschiedene Szenarienmengen. In der ersten Menge dürfen Notfallpatienten nur eine Behandlungen besitzen, in der zweiten Szenarienmenge dürfen Notfallpatienten maximal zwei Behandlungen besitzen in der dritten drei, in der vierten vier und die fünfte Szenarienmenge entspricht der in den vorherigen Experimenten betrachteten Szenarienmenge, in der ein Notfallpatient bis zu fünf Behandlungen besitzen darf. Die Szenarien werden, wie in Abschnitt 5.2 beschrieben, erzeugt.

Testen wir die ermittelten Schedules gegen die Szenarienmengen, so sind die erzielten Robustheitswerte in den nachfolgenden Abbildungen dargestellt.

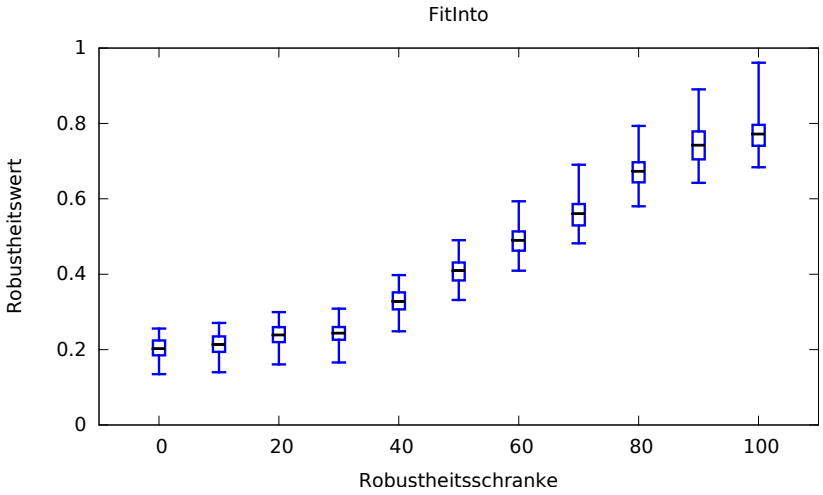


Abbildung 5.19: Notfallpatienten besitzen eine Behandlung.

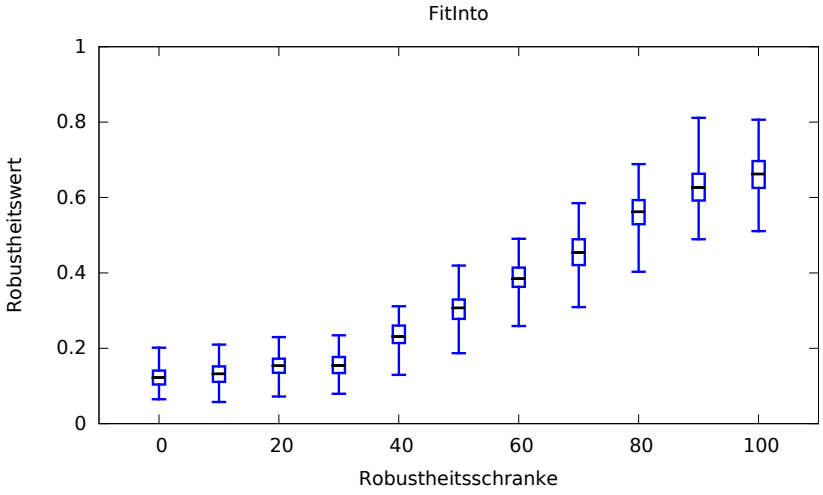


Abbildung 5.20: Notfallpatienten besitzen maximal zwei Behandlungen.

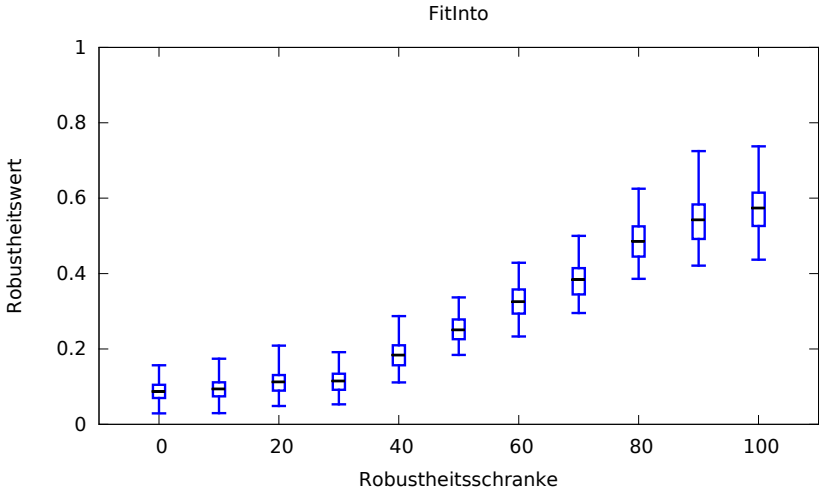


Abbildung 5.21: Notfallpatienten besitzen maximal drei Behandlungen.

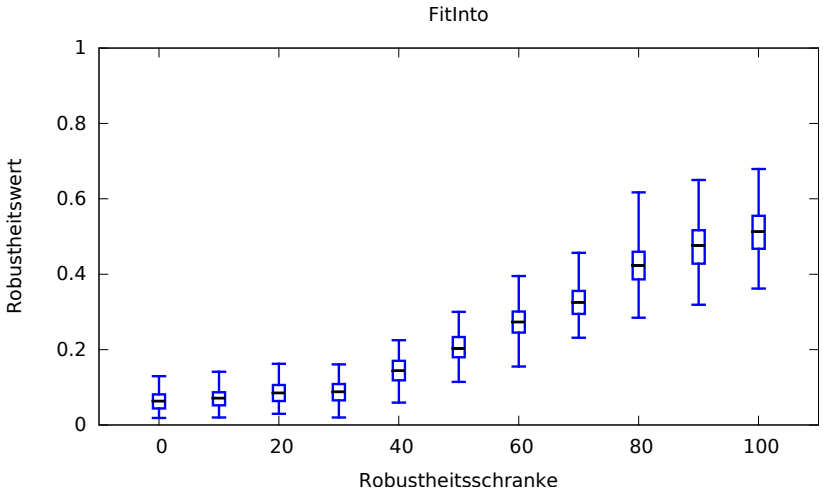


Abbildung 5.22: Notfallpatienten besitzen maximal vier Behandlungen.

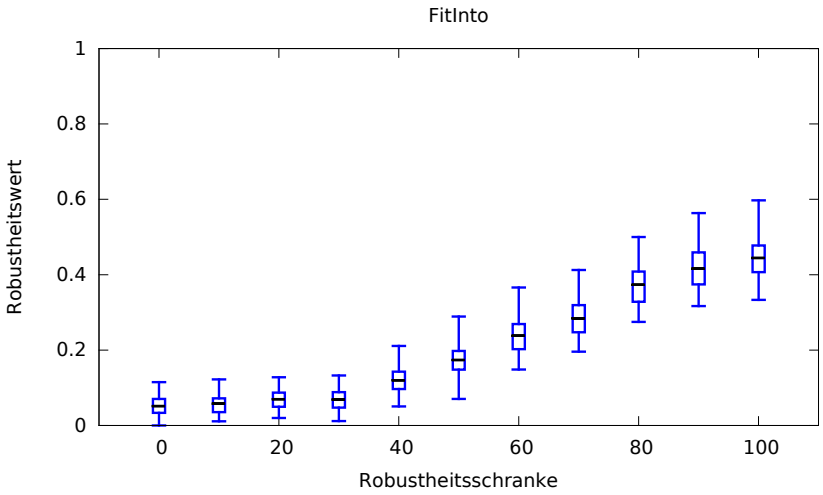


Abbildung 5.23: Notfallpatienten besitzen maximal fünf Behandlungen.

Vergleicht man die Abbildungen 5.19 bis 5.23 so ist zu erkennen, dass der Robustheitswert einer Lösung gegenüber einer Szenarienmenge von der maximalen Anzahl der Behandlungen eines Patienten abhängig ist. Dies erweckt den Eindruck, dass die Absicherung gegen unabhängige Behandlungen nicht ausreicht, um sich gegen das Auftreten von Notfallpatienten mit mehreren Behandlungen abzusichern.

Kapitel 6

Fazit und Ausblick

Ziel dieser Arbeit war das Entwickeln eines Algorithmus, der Behandlungen einer Menge gegebener Patienten so Behandlungszeitpunkte auf Ressourcen zuweist, dass Behandlungen von Notfallpatienten zu einem gewissen Maß spontan in den ermittelten Zeitplan eingefügt werden können.

Im Rahmen dieser Arbeit haben wir einen solchen Algorithmus konstruiert. Dazu wurde zunächst das betrachtete Problem in Form des Krankenhaus Scheduling Problems unter Unsicherheiten definiert. Darauffolgend wurden bestehende Ansätze zur Lösung von Optimierungsproblemen unter Unsicherheiten resümiert. Um das Krankenhaus Scheduling Problem unter Unsicherheiten effizient lösen zu können, schien ein Lokaler Such Algorithmus eine gute Wahl darzustellen. Daher haben wir als Grundlage für unseren Algorithmus eine Lokale Suche gewählt. Damit in einer Lokalen Suche robuste Lösungen ausgewählt werden, haben wir zunächst allgemein zwei Ansätze vorgestellt. Für diese Ansätze wird eine Szenarienmenge benötigt, die das Auftreten der Unsicherheiten repräsentiert. Anschließend haben wir diese Ansätze auf das Krankenhaus Scheduling Problem unter Unsicherheiten angewendet. Dazu haben wir als erstes die problemspezifischen Elemente einer Lokalen Suche für das Krankenhaus Scheduling Problem umgesetzt. Im Anschluss haben wir die Lokale Suche so erweitert, dass das Erzeugen von Freiräumen in einem Schedule möglich ist. Um das Ausmaß der Absicherung eines Schedules gegenüber dem Auftreten von Notfallpatienten messen zu können, haben wir zwei Robustheitsmaße definiert. Mit den Robustheitsmaßen wird für einen Schedule für eine Szenarienmenge, die das Auftreten von Notfallpatienten darstellen soll, ein Robustheitswert ermittelt. Bei der Auswahl des nächsten Schedules im Rahmen einer Iteration der Lokalen Suche, wird mittels einer festen Szenarienmenge der Robustheitswert aller Nachbarn bestimmt. Zum Schluss haben wir die entwickelten Ansätze für die beiden Robustheitsmaße getestet.

Für beide Ansätze hat sich unsere Vermutung bestätigt, dass sich mit steigendem Robustheitswert eines Schedules der Kostenwert verschlechtert. Auch haben wir gezeigt, dass die Kardinalität der betrachteten Szenarienmenge Auswirkungen auf die Robustheit eines ermittelten Schedules hat. Eine Gegenüberstellung der beiden entwickelten Ansätze in Form eines Vergleichs der erzielten Kosten-Robustheitswert-Kombinationen lieferte das folgende Ergebnis. Während sich für das Fit-Into-Robustheitsmaß die beiden Ansätze in der Ermittlung möglichst guter Robustheits-Kostenwert-Kombinationen nur wenig unterschieden, scheint für das Schedule-Into-Robustheitsmaß der Ansatz einer Robusten Lokalen Suche mit Konvexkombination dem einer Robusten Lokalen Suche mit Akzeptor überlegen zu sein.

Im Vergleich zu Lokalen Such Algorithmen für das eng verwandte Job Shop Scheduling Problem benötigt der von uns entwickelte Algorithmus viel Zeit, um zu einer guten Lösung zu gelangen. Daher könnte der Algorithmus verbessert werden, indem man ihn beschleunigt. Für eine Reduktion der Laufzeit einer Lokalen Suche werden in der Literatur vor

allem zwei Ansätze betrachtet. Zum einen wird betrachtet, dass mögliche Lösungen einer Nachbarschaft schneller ausgewertet werden können. Dies kann zum Beispiel mittels approximativer Werte geschehen. Mit diesen Werten kann entschieden werden, ob ein Nachbar als verbessernde Lösung in Frage kommt. Der zweite Ansatz ist, die Nachbarschaft einer Lösung zu reduzieren. In den von uns gewählten Ansätzen wird der Zeitraum, in dem Behandlungen von Notfallpatienten stattfinden können, mit Timeslots überdeckt. Wird eine Nachbarschaft durchsucht, so wird für jeden Timeslot ein Konfigurationswechsel betrachtet. Daher wird durch das Einführen von Timeslots die Kardinalität einer Nachbarschaft erheblich vergrößert. Gerade wenn eine kleine Dauer für die Größe der Timeslots gewählt wird, benötigt das Durchsuchen einer Nachbarschaft sehr viel Zeit. Daher könnte überlegt werden, ob mittels eines geeigneten Kriteriums entschieden werden kann, welche Nachbarn auszuwerten sinnvoll ist.

In unseren Experimenten haben wir gezeigt, dass mit steigender Kardinalität der betrachteten Szenarienmenge die Robustheit eines betrachteten Schedules verbessert werden kann. Allerdings fand vor allem eine Steigerung der Robustheit zwischen der Absicherung gegen ein Szenario und gegen 25 Szenarien statt. Da sich bei der Absicherung gegen eine große Anzahl von Szenarien die Laufzeit des Algorithmus verlängert, könnte noch genauer untersucht werden, wie sich der Robustheitswert bei Variation der Kardinalität der Szenarienmenge verändert. Es könnte auch getestet werden, ob eine Steigerung der Robustheit stattfindet, wenn ein Schedule in verschiedenen Iterationen gegen ein zufällig ermitteltes Szenario getestet würde.

In dieser Arbeit haben wir die zwei Robustheitsmaße, das Fit-Into-Robustheitsmaß und das Schedule-Into-Robustheitsmaß, vorgestellt. Im Fit-Into-Robustheitsmaß kann eine Zeitschranke angegeben werden, in der ein Patient als in annehmbarer Zeit behandelt gilt. In den betrachteten Experimenten haben wir uns auf den Fall beschränkt, dass diese Schranke so groß gewählt ist, dass Patienten im Laufe der gesamten Öffnungsdauern ihres Ankunftstages eingeplant und als in annehmbarer Zeit behandelt gelten. Daher könnte untersucht werden, wie sich ein Schedule verändert, wenn man diese Schranke reduziert. Die Schranke könnte auch je nach Patienten variiert werden. Es gibt Patienten, bei denen bekannt ist, dass eine gewisse Behandlung zeitnah durchgeführt werden soll, eine unmittelbare Durchführung aber nicht notwendig ist. Es können aber auch Notfallpatienten hinzukommen, deren Behandlungen unmittelbar durchgeführt werden müssen.
Genauso kann beim Schedule-Into-Robustheitsmaß eine Funktion zur Gewichtung der Dauer der Wartezeiten angegeben werden. Wir haben uns auf den Fall beschränkt, dass die Wartezeiten entsprechend ihrer Dauer im Robustheitsmaß berücksichtigt werden. Zusätzlich kann untersucht werden, wie sich eine Lösung bei Variation dieser Funktion verändert.

Das von uns betrachtete Krankenhaus Scheduling Problem konzentriert sich auf die Berücksichtigung des Auftretens von Notfallpatienten als Unsicherheiten. In der Realität sind aber eine Vielzahl von Unsicherheitsfaktoren denkbar. So können zum Beispiel die tatsächlichen Behandlungsdauern von den geplanten abweichen. Um das Problem realitätsnaher zu gestalten, könnte der Algorithmus erweitert werden, so dass eine Berücksichtigung unterschiedlicher Unsicherheitsmengen möglich ist.

Da in der Zukunft aufgrund der gegenwärtigen Entwicklung die Anzahl der Patienten in Krankenhäusern weiter zunehmen wird, wird das Problem, alle Patienten in angemessener Zeit zu behandeln, größere Bedeutung erlangen. Mit dieser Arbeit haben wir einen Ansatz zur Lösung dieses Problems entwickelt. Für die Anwendung unseres Ansatzes in der Praxis müsste dieser allerdings noch erweitert werden.

Literaturverzeichnis

[1] D. D. W. A. Hertz, E. Taillard. Local search in combinatorial optimizatios. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, volume 5, pages 121–136. Princeton University Press, 2003.

[2] I. Adiri, J. Bruno, E. Frostig, and A. Rinnooy Kan. Single machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26(7):679–696, 1989.

[3] R. Aggoune. *Ordonnancement d'ateliers sous contraintes de disponibilité des machines*. 2002.

[4] M. Akker, K. Blokland, and H. Hoogeveen. Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search. In V. Bonifaci, C. Demetrescu, and A. Marchetti-Spaccamela, editors, *Experimental Algorithms*, volume 7933 of *Lecture Notes in Computer Science*, pages 402–413. Springer Berlin Heidelberg, 2013.

[5] N. Al-Hinai and T. ElMekkawy. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics*, 132(2):279 – 291, 2011.

[6] C. Anderson, K. Fraughnaugh, M. Parker, and J. Ryan. Path assignment for call routing: An application of tabu search. *Annals of Operations Research*, 41(4):299–312, 1993.

[7] V. A. Armentano and C. R. Scrich. Tabu search for minimizing total tardiness in a job shop. *International Journal of Production Economics*, 63(2):131 – 140, 2000.

[8] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86 – 110, 2005. IEPM: Focus on Scheduling.

[9] E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *Journal of the Royal Statistical Society. Series B (Methodological)*, 17(2):pp. 173–184, 1955.

[10] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[11] R. Bellman, L. Zadeh, and C. U. E. R. Laboratory. *Decision-making in a fuzzy environment*. Number Bd. 1594 in NASA contractor report. National Aeronautics and Space Administration; for sale by the Clearinghouse for Federal Scientific and Technical Information, Springfield, Va., 1970.

[12] A. Ben-Tal, L. Ghaoui, and A. Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, 2009.

[13] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998.

[14] A. Ben-Tal and A. Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1 – 13, 1999.

[15] A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424, 2000.

[16] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Computational Management Science*, 2(1):3–19, 2005.

[17] D. Bertsimas, D. B. Brown, and C. Caramanis. Theory and applications of robust optimization. *SIAM Rev.*, 53(3):464–501, Aug. 2011.

[18] D. Bertsimas and A. Thiele. Robust and data-driven optimization: Modern decision-making under uncertainty, 2006.

[19] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, Sept. 2003.

[20] A. Charnes and W. W. Cooper. Chance-constrained programming. *Management Science*, 6(1):73–79, 1959.

[21] H. Chtourou and M. Haouari. A two-stage-priority-rule-based algorithm for robust resource-constrained project scheduling. *Computers & Industrial Engineering*, 55(1):183 – 194, 2008.

[22] W. Clark, W. Polakov, and F. Trabold. *The Gantt chart: a working tool of management*. Ronald Press, 1923.

[23] G. B. Dantzig. Linear programming under uncertainty. *Manage. Sci.*, 50(12 Supplement):1764–1769, Dec. 2004.

[24] G. e. a. De Smet. Optaplanner planning engine, 2012.

[25] J. E. Falk. Exact solutions of inexact linear programs. *Operations Research*, 24(4):783–787, 1976.

[26] H. L. Gantt. Efficiency and democracy. *ASME Transactions*, 40:799 – 808, 1919.

[27] L. E. Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data, 1997.

[28] L. E. Ghaoui, F. Oustry, and H. Lebret. Robust solutions to uncertain semidefinite programs. *SIAM J. on Optimization*, 9(1):33–52, May 1998.

[29] F. Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5):533–549, May 1986.

[30] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986. Applications of Integer Programming.

[31] F. Glover and M. Laguna. *Tabu Search*. Number Bd. 1 in Tabu Search. Springer US, 1998.

[32] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289 – 306, 2005. <ce:title>Project Management and Scheduling</ce:title>.

[33] A. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390 – 434, 1999.

[34] P. Kall and S. Wallace. *Stochastic programming*. Wiley-Interscience series in systems and optimization. Wiley, 1994.

[35] U. Kamps. Erneuerungstheorie; vorlesungsskript.

[36] A. M. C. A. Koster. Optimierung unter unsicherheiten; vorlesungsskript.

[37] V. J. Leon and S. D. Wu. On scheduling with ready-times, due-dates and vacations. *Naval Research Logistics (NRL)*, 39(1):53–65, 1992.

[38] A. Løkketangen and U. i Bergen. Department of Informatics. *Tabu Search as a Metaheuristic Guide for Combinatorial Optimization Problems*. Reports in informatics. Department of Informatics, University of Bergen, 1995.

[39] Y. Mati, S. Dauzère-Pérès, and C. Lahlou. A general approach for optimizing regular criteria in the job-shop scheduling problem. *European Journal of Operational Research*, 212(1):33 – 42, 2011.

[40] B. L. Miller and H. M. Wagner. Chance constrained programming with joint constraints. *Operations Research*, 13(6):930–945, 1965.

[41] T. Morton and D. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. Number Bd. 3 in A Wiley-Interscience publication. Wiley, 1993.

[42] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Manage. Sci.*, 42(6):797–813, June 1996.

[43] A. S. of Mechanical Engineers. *ASME Transactions*. Transactions of the American Society of Mechanical Engineers. The Society, 1919.

[44] F. Pezzella and E. Merelli. A tabu search method guided by shifting bottleneck for the job shop scheduling problem, 2000.

[45] D. B. Porter. The gantt chart as applied to production scheduling and control. *Naval Research Logistics Quarterly*, 15(2):311–317, 1968.

[46] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[47] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives, 1964. D.S. vol. 9, SEMA, Paris, France.

[48] N. V. Sahinidis. Optimization under uncertainty: state-of-the-art and opportunities. *Computers & Chemical Engineering*, 28(6–7):971 – 983, 2004. {FOCAPO} 2003 Special issue.

[49] G. Schmidt. Scheduling on semi-identical processors. *Zeitschrift für Operations Research*, 28(5):153–162, 1984.

[50] G. Schmidt. Scheduling independent tasks with deadlines on semi-identical processors. *Journal of the Operational Research Society*, pages 271–277, 1988.

[51] A. L. Soyster. Technical note—convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973.

[52] r. D. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

[53] A. Thesen. Design and evaluation of tabu search algorithms for multiprocessor scheduling. *Journal of Heuristics*, 4(2):141–160, 1998.

[54] R. J. M. Vaessens, E. Aarts, and J. Lenstra. Job shop scheduling by local search. *INFORMS JOURNAL ON COMPUTING*, 8:302–317, 1994.

[55] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Oper. Res.*, 40(1):113–125, Jan. 1992.

[56] J. Xiong, L. ning Xing, and Y. wu Chen. Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics*, 141(1):112 – 126, 2013. <ce:title>Meta-heuristics for manufacturing scheduling and logistics problems</ce:title>.

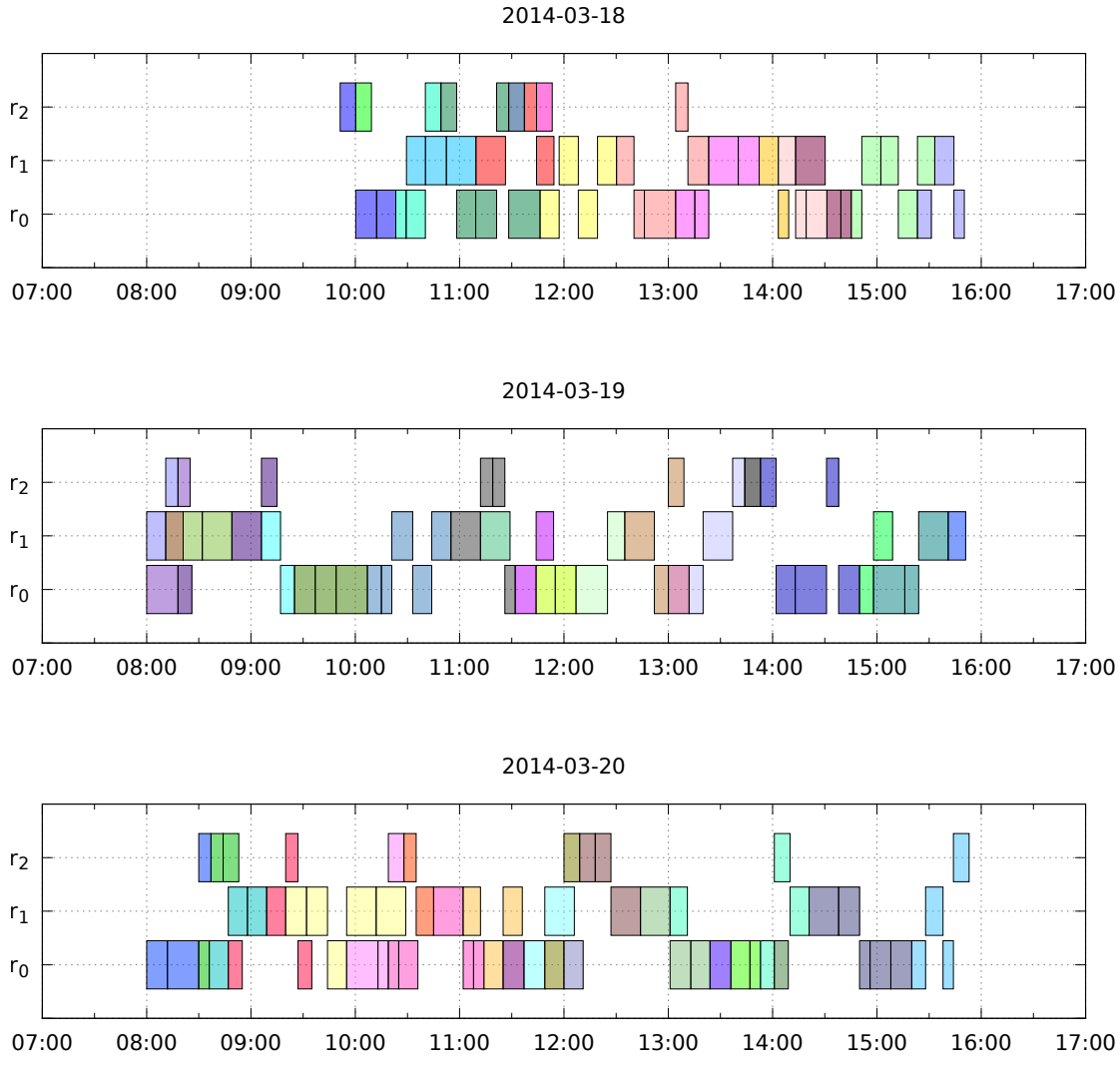
[57] X.-S. Yang. Metaheuristic optimization: Nature-inspired algorithms and applications. In X.-S. Yang, editor, *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, volume 427 of *Studies in Computational Intelligence*, pages 405–420. Springer Berlin Heidelberg, 2013.

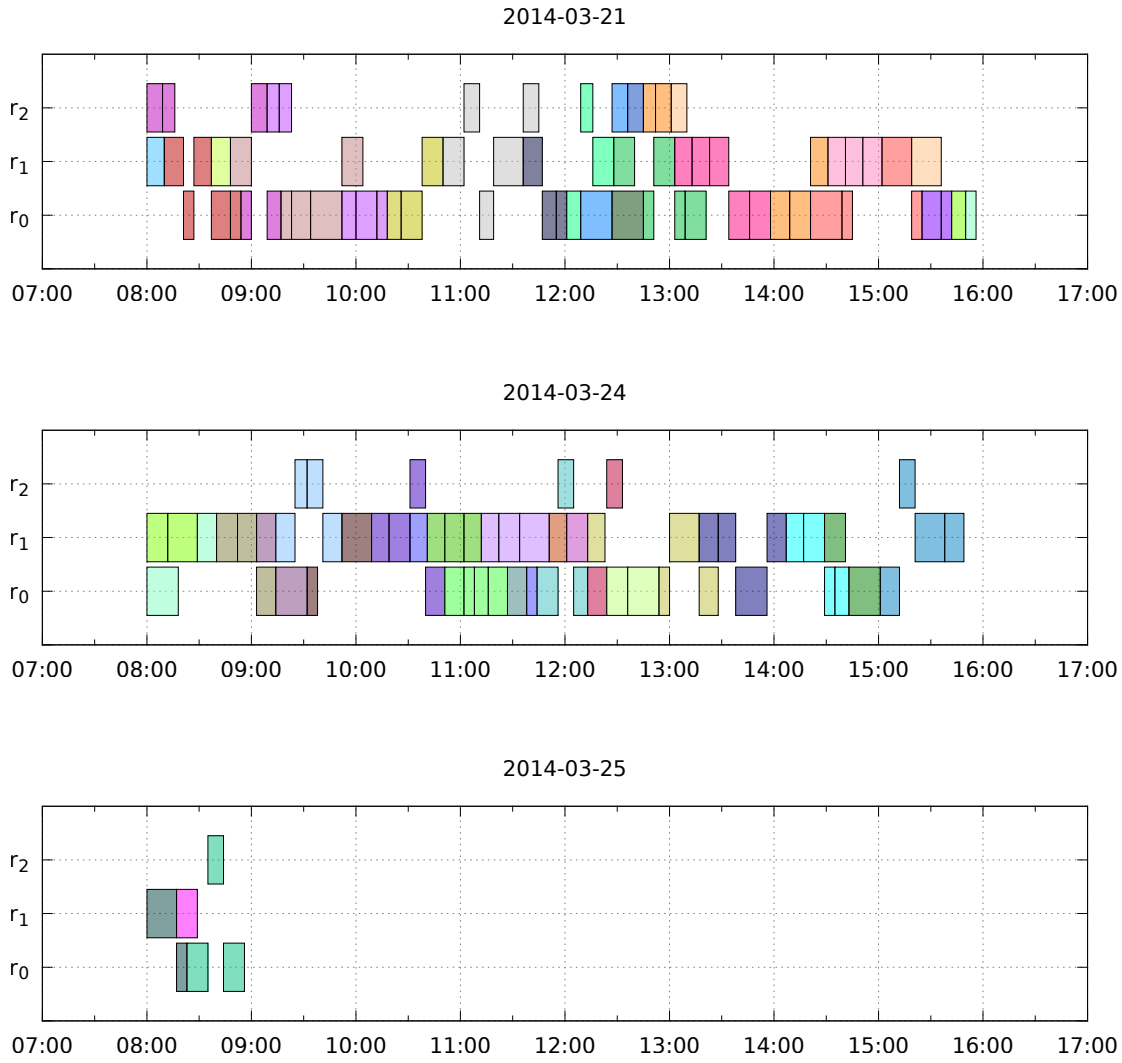
[58] J.-M. Yu, J.-S. Kim, and D.-H. Lee. Scheduling algorithms to minimise the total family flow time for job shops with job families. *International Journal of Production Research*, 49(22):6885–6903, 2011.

Anhang

In diesem Abschnitt möchten wir ein Bild von der betrachteten Instanz schaffen und aufzeigen, wie sich ein Schedule zu dieser Instanz je nach gewünschtem Robustheitswert unterscheidet. Dazu betrachten wir die beiden Extrema, dass entweder nur der Kostenwert oder nur der Robustheitswert im Fokus der Robusten Lokalen Suche steht. In den betrachteten Fällen stimmen unsere Ansätze einer Robusten Lokalen Suche überein.

Zunächst sei eine Startlösung gemäß Algorithmus 7 dargestellt. In der Startlösung lässt sich gut erkennen, wie viele Behandlungen ein Patient besitzt.

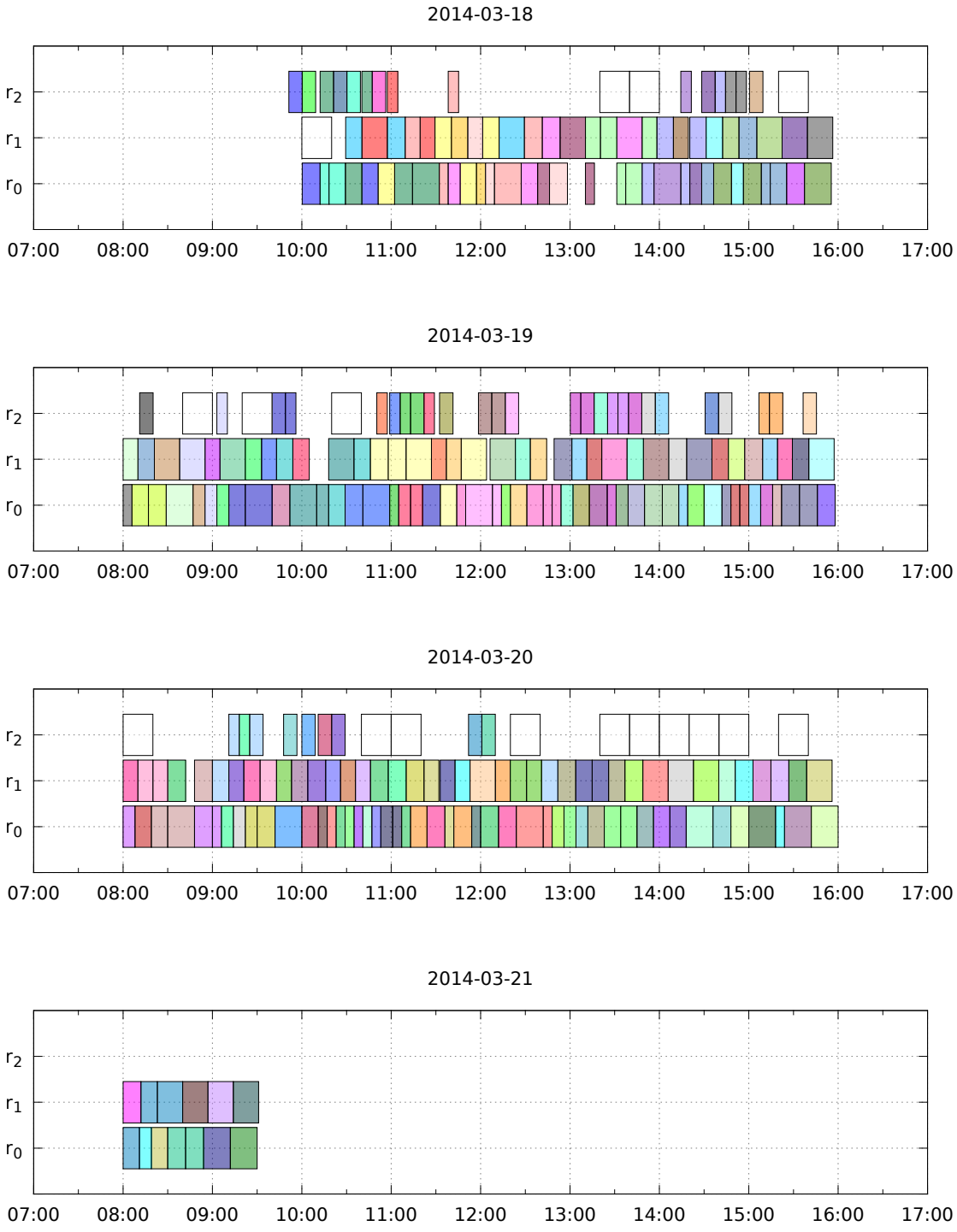




Schedule einer Startlösung gemäß Algorithmus 7.

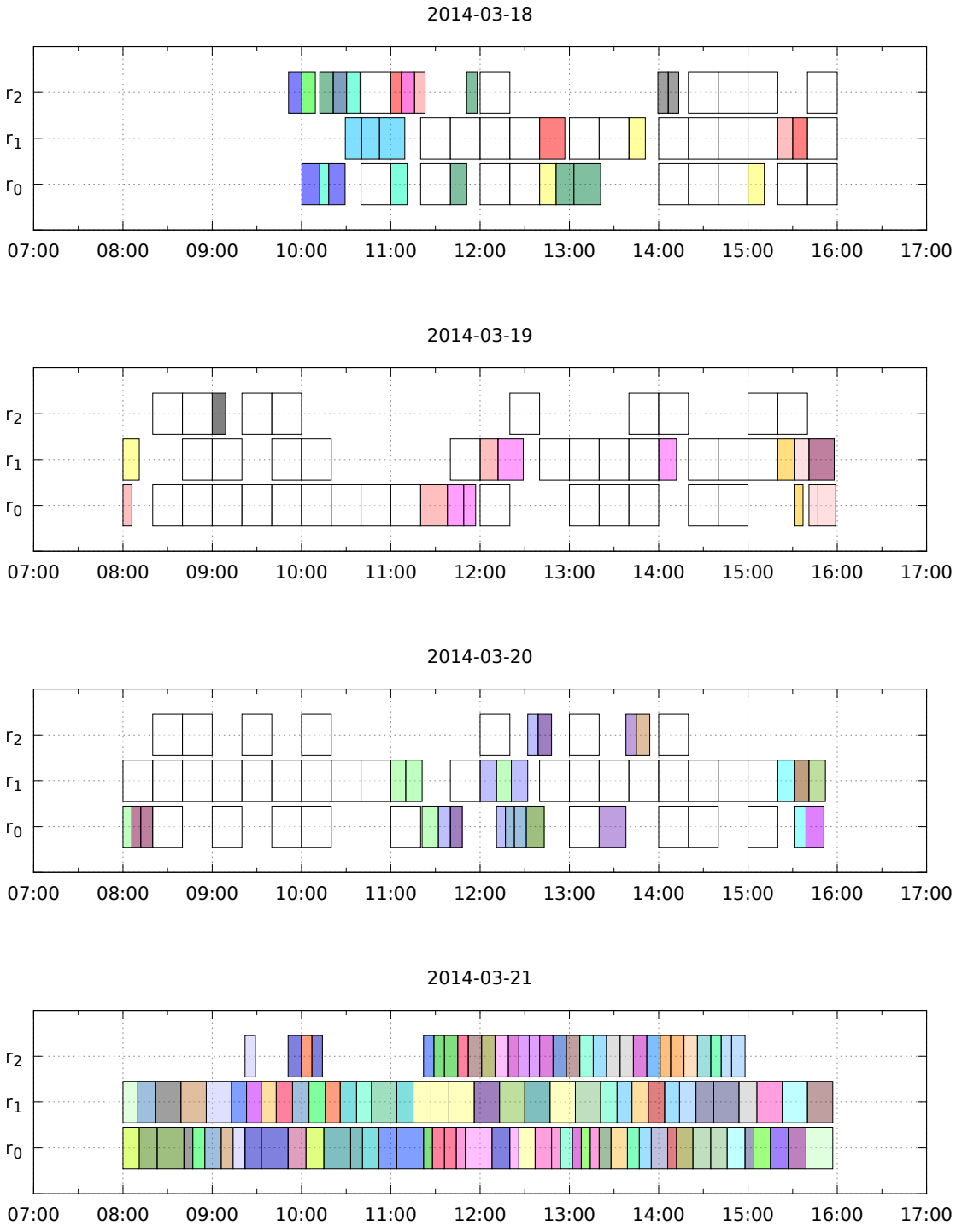
Die Behandlungen eines Patienten sind in derselben Farbe dargestellt. Sind in den folgenden Abbildung „weiße Behandlungen“ dargestellt, so symbolisieren diese Timeslots.

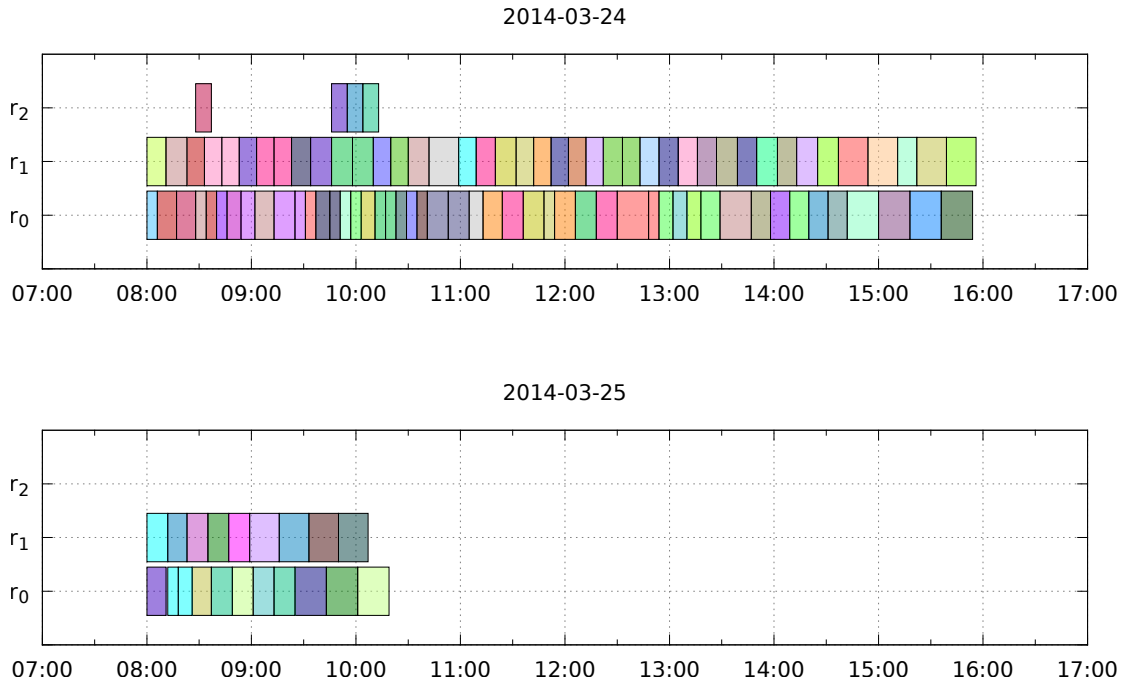
Wählen wir die Robustheitsschranke gleich null oder den Robustheitsfaktor gleich null, so stimmen die beiden Ansätze einer Robusten Lokalen Suche überein und entsprechen einer Lokalen Suche. Auch spielt in diesem Fall das Robustheitsmaß keine Rolle. Für diesen Fall ergibt sich ein Schedule wie in der folgenden Abbildung.



Schedule bei $a = d = 0$.

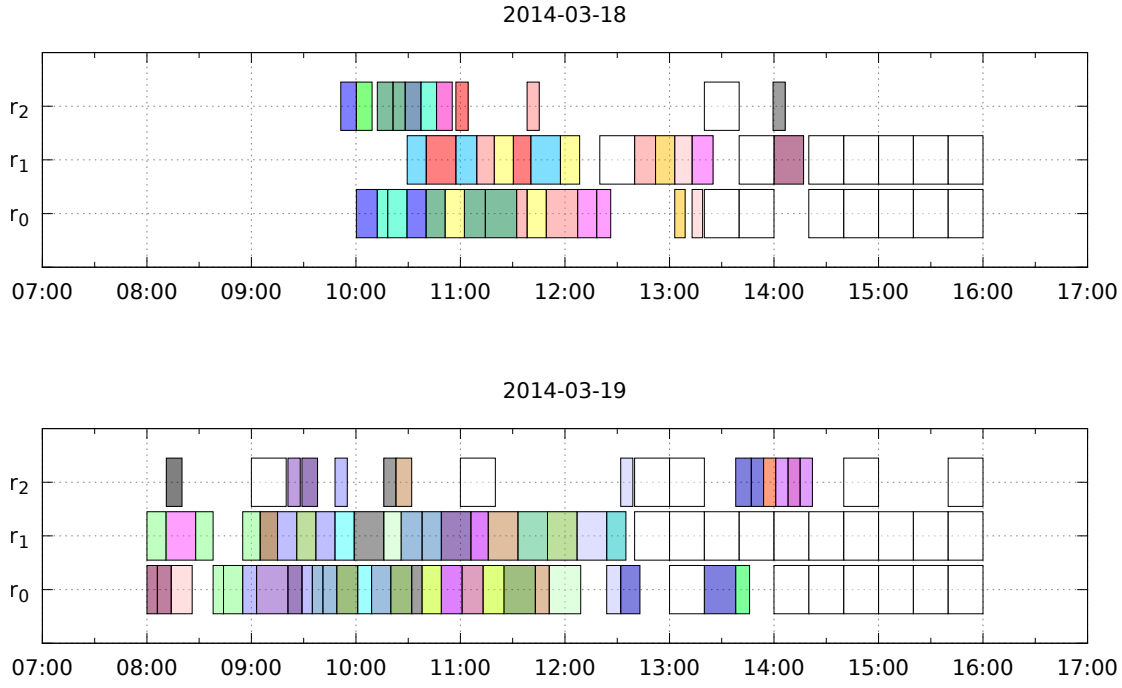
Ist die Robustheitsschranke gleich hundert und der Robustheitsfaktor gleich hundert, so wird ein Schedule nur in Bezug auf seine Robustheit bewertet. Betrachtet man das Schedule-Into-Robustheitsmaß, so könnte ein Schedule wie folgt aussehen.

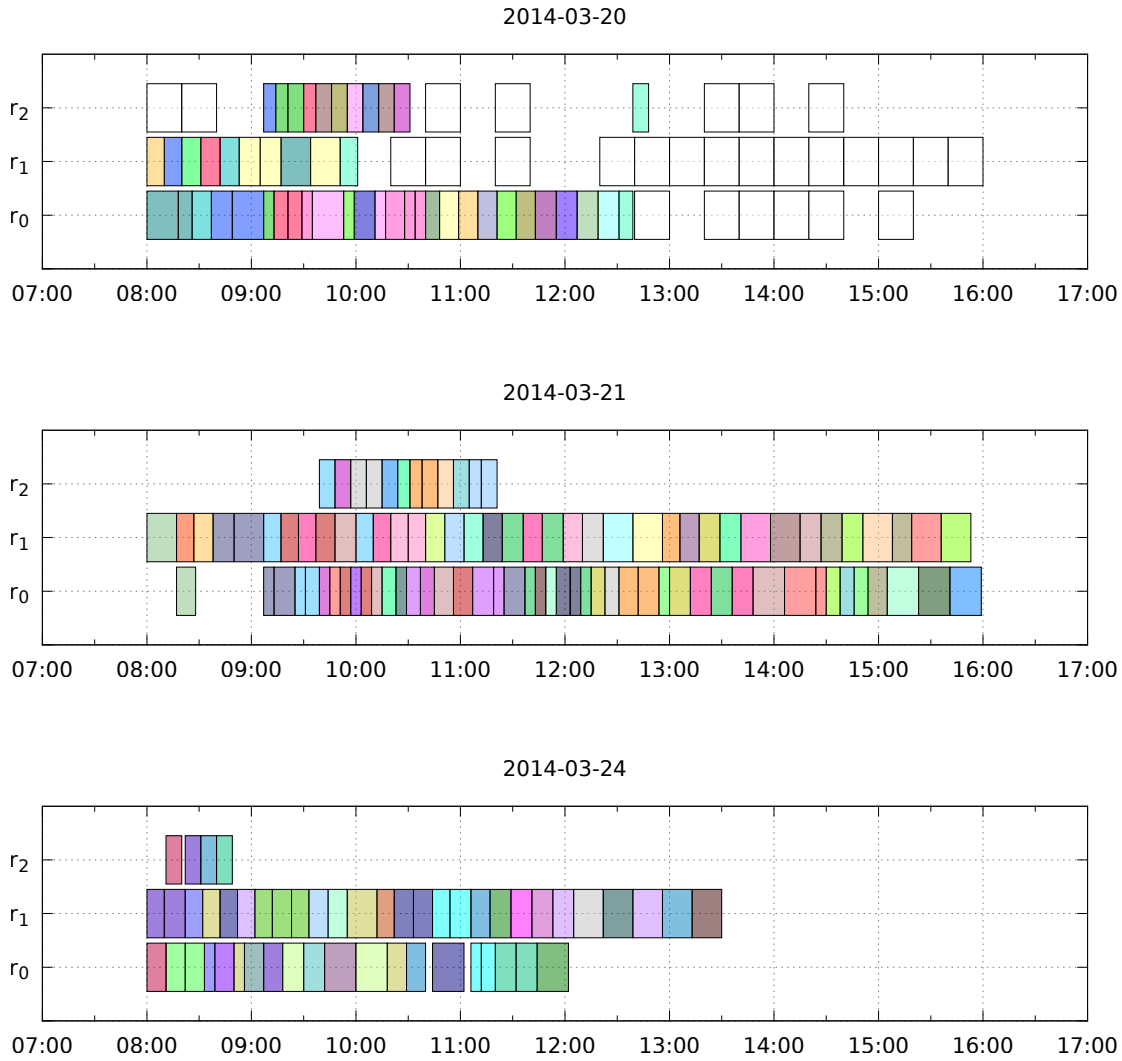




Schedule für das Schedule-Into-Robustheitsmaß für $a = d = 100$.

Betrachtet man dagegen das Fit-Into-Robustheitsmaß, so ergibt sich ein Schedule wie in der folgenden Abbildung.





Schedule für das Fit-Into-Robustheitsmaß für $a = d = 100$.

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Aachen, den 31. März 2014 Friederike Menge