Lehrstuhl für Angewandte Mathematik

Masterarbeit

Genetische Algorithmen zur Lösung eines Human Ressource Allocation Problems

Ole Zechmann, BSc

Juni 2020

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Ich erkläre, dass ich die Richtlinien des Senats der Montanuniversität Leoben zu "Gute wissenschaftliche Praxis" gelesen, verstanden und befolgt habe.

Weiters erkläre ich, dass die elektronische und gedruckte Version der eingereichten wissenschaftlichen Abschlussarbeit formal und inhaltlich identisch sind.

Datum 11.05.2020

Unterschrift Verfasser/in
Ole, Zechmann

Danksagung

Ich möchte mich herzlichst bei allen Personen bedanken, die mir beim Verfassen meiner Diplomarbeit geholfen haben.

Ein ganz besonderer Dank gebührt meinem Betreuer Norbert Seifter (Titel siehe oben) für zahlreiche interessante Gespräche, eine stets offene Tür und seine Geduld und Gelassenheit. Seine Bereitschaft mich zu betreuen sowie sein nahender Pensionsantritt haben einen wesentlichen Beitrag dazu geleistet, dass diese Arbeit überhaupt in endlicher Zeit fertig gestellt wurde.

Des weiteren möchte ich mich bei meinen Eltern bedanken, welche mir mein Studium überhaupt erst möglich gemacht haben und mich auf meinem gesamten Studienweg stets unterstützt haben.

Ich möchte mich außerdem auch bei meiner Frau bedanken, welche sich, in den vielen Stunden während derer ich mich mit meiner Diplomarbeit befasst habe, um unser Kind und den Haushalt gekümmert hat.

Inhaltsverzeichnis

1 Einleitung **1**

2 Ähnliche Probleme und bekannte Lösungsansätze **2**
2.1 Einführung . . . 2
2.2 Das Zuweisungsproblem . . . 2
2.2.1 Problemschilderung . . . 2
2.2.2 Differenzen zur Problemstellung . . . 4
2.2.3 Die Ungarische Methode . . . 4
2.3 Das verallgemeinerte Zuordnungsproblem . . . 5
2.4 Assembly Line Worker Assignment and Balancing Problem . . . 6
2.5 Weitere veröffentlichte Probleme . . . 9
2.6 Lösungsansätze . . . 9

3 Problembeschreibung **13**
3.1 Formulierung des Grundproblems . . . 13
3.2 Formulierung der Nebenbedingungen . . . 15
3.2.1 Abhängigkeiten . . . 15
3.2.2 Puffer . . . 17
3.2.3 Initialbelegung . . . 19
3.2.4 Unerfüllbare Bedarfe . . . 22
3.3 Unterstützende Prozessschritte . . . 24
3.4 Alternative Zielfunktionen . . . 24

4 Flussprobleme **26**
4.1 Algorithmus von Ford und Fulkerson . . . 27
4.2 Maximum Flow Algorithmen und deren Laufzeit . . . 28
4.2.1 Algorithmus von Edmonds und Karp . . . 28
4.2.2 Algorithmus von Dinic . . . 31
4.3 Problemspezifische Anpassungen . . . 32

5 Heuristiken **39**
5.1 Selektionsverfahren für Stationen . . . 39
5.2 Selektionsverfahren für Ressourcen . . . 40

6 Genetische Algorithmen **46**

6.1 Einführung 46
6.2 Repräsentation der Individuen 48
6.3 Die erste Generation 50
6.3.1 Zufällig erzeugte Individuen 50
6.3.2 Erzeugungsvarianten 50
6.4 Selektionsverfahren 51
6.4.1 Fitnessproportionale Selektion 51
6.4.2 Rangselektion 53
6.4.3 Turnierverfahren 54
6.4.4 Elitismus 56
6.5 Kreuzungsverfahren 56
6.5.1 Single Point Crossover 56
6.5.2 Two Point Crossover 57
6.5.3 Uniform Crossover 58
6.6 Mutation 59
6.7 Reparaturalgorithmus 60
6.8 Parameter 66

7 Analyse **67**
7.1 Testproblem 1 69
7.2 Testproblem 2 71
7.3 Testproblem 3 75
7.4 Testproblem 4 82

8 Resümee **89**

1 Einleitung

Das in dieser Diplomarbeit betrachtete Problem wurde von einer konkreten Problemstellung, welche im Bereich der Intralogistik aufgetreten ist, abgeleitet und anschließend verallgemeinert. Die in der Praxis aufgetretene Situation war folgende: Zur Bearbeitung von verschiedensten Aufträgen innerhalb eines Großlagers war eine Prozeskette gegeben, welche sowohl manuelle als auch automatische Prozessschritte beinhaltete. Diese waren zum Teil auch seriell verkettet und konnten daher nur dann ihre volle Leistung erbringen, wenn sie von den vorgeschalteten Prozesschritten ausreichend beliefert wurden und die von ihnen erledigten Aufträge anschließend auch an ihre Nachfolger weitergeben konnten, wobei zum Ausgleich leichter Leistungsunterschiede in der Prozesskette teilweise Pufferlager vorhanden waren. Zum Bearbeiten der manuellen Prozesschritte stand eine bestimmte Anzahl an Mitarbeitern zur Verfügung, wobei nicht jeder Mitarbeiter für jeden Prozessschritt geschult war. Welcher Mitarbeiter wo arbeitet, wurde ein bis zwei Wochen im vorhinein eingeteilt und anschließend nicht mehr verändert. Durch sehr kurzfristige Änderungen in der Auftragslage, Maschinenausfälle, unvorhersehbare Abwesenheit von Mitarbeitern (z.B. durch Erkrankung) und tägliche Leistungsschwankungen war diese Einteilung oft nicht mehr ideal und bedurfte daher einer Änderung. Für den zuständigen Lagerleiter war es fast nicht mehr möglich, bei der großen Anzahl an Mitarbeitern und Prozesschritten, kurzfristig eine sehr leistungsfähige und kosteneffiziente Umschichtung der Mitarbeiter vorzunehmen. Das Ziel dieser Diplomarbeit ist es, in der beschriebenen Situation rasch eine Belegung zu finden, welche, unter Berücksichtigung der gegebenen Prozesskette, der einzelnen Mitarbeiter und deren Fähigkeiten, den Gesamtdurchsatz des Systems maximiert und dabei alle Bedarfe vollständig erfüllt.

2 Ähnliche Probleme und bekannte Lösungsansätze

2.1 Einführung

Bei einer ersten, ganz grundlegenden Betrachtung dieses Themas ergibt sich folgende Problemstellung. Man hat eine bestimmte Anzahl von Stationen und eine bestimmte Anzahl von Ressourcen, welche man auf diese Stationen verteilen möchte. Zusätzlich ist bekannt, welche Leistung die verschiedenen Ressourcen auf den einzelnen Stationen erzielen können, wie viele Ressourcen auf jeder Station mindestens zugeteilt werden müssen bzw. wie viele Ressourcen der Station maximal zugeteilt werden dürfen und welche Prozesschritte voneinander abhängig sind da sie seriell zueinander arbeiten. Des weiteren ist für jede Station auch eine Mindestleistung bekannt, welche erfüllt werden muss. Das Ziel ist nun, die Gesamtleistung aller Ressourcen zu maximieren, unter Berücksichtigung der soeben genannten Nebenbedingungen.

Problemstellungen dieser Art werden in der Literatur als **Human Resource Allocation Problems (HRAP)** betitelt und sind als solche ein Spezialfall der allgemeineren **Resource Allocation Problems (RAP)**. Bouajaja und Dridi [2] geben in ihrer Veröffentlichung einen Überblick über das HRAP im Allgemeinen, sowie einiger spezifischer Nebenbedingungen, welche wiederum zu unterschiedlichen Modellen und Lösungsansätzen führen. Um einen groben Überblick zu bekommen, sollen in diesem Kapitel einige bekannte Varianten des HRAP vorgestellt werden. Dabei wird auch darauf eingegangen, für welche praktischen Problemstellungen diese geeignet sind und inwiefern sich das in dieser Arbeit betrachtete Problem von diesen Problemstellungen unterscheidet.

2.2 Das Zuweisungsproblem

2.2.1 Problemschilderung

Das HRAP ist eine Variation eines der klassischen Probleme der kombinatorischen Optimierung, nämlich des **Zuweisungsproblems**. In seiner allgemeinsten Form lässt sich

dieses wie folgt beschreiben:

Gegeben sei eine Menge an Ressourcen (Arbeiter, Maschinen, etc.) und eine Menge and Tätigkeiten, sowie die Ausführungskosten/-Erträge pro Tätigkeit und Ressource. Nun sollen so viele Tätigkeiten wie möglich ausgeführt werden, indem jeder Tätigkeit höchstens eine Ressource und jeder Ressource höchstens eine Tätigkeit zugewießen wird, wobei die Gesamtausführungskosten/-Erträge optimiert werden sollen.

Ist die Anzahl an Tätigkeiten gleich der Anzahl der zur Verfügung stehenden Ressourcen, so spricht man von einem **symmetrischen** oder **ausgewogenen** Zuordnungsproblem.

Sind die Gesamtkosten/der Gesamtertrag gleich der Summe der Kosten/Erträge aller Zuweisungen, so wird das Problem als **linear** bezeichnet.

Üblicherweise, wenn vom Zuordnungsproblem gesprochen wird, ohne dass irgendwelche spezifischeren Angaben getroffen werden, so ist das lineare, ausgewogene Zuordnungsproblem gemeint.

Mathematisch lässt sich dieses wie folgt definieren:

Minimiere

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \tag{2.1}$$

Unter den Nebenbedingungen

$$\sum_{i=1}^{n} x_{i,j} = 1, \quad j = 1, ..., n \tag{2.2}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \quad i = 1, ..., n \tag{2.3}$$

$$x_{i,j} \in \{0, 1\} \tag{2.4}$$

Wobei $x_{i,j} = 1$ ist, falls einer Tätigkeit j die Ressource i zugewießen ist. Andernfalls ist $x_{i,j} = 0$. Durch die erste Nebenbedingung wird sichergestellt, dass jeder Tätigkeit nur eine Ressource zugewießen wird. Die zweite Nebenbedingung stellt sicher, dass jede Ressource nur einer Tätigkeit zugewießen ist. $c_{i,j}$ gibt an, mit welchen Kosten die Ausführung von Tätigkeit j durch die Ressource i verbunden ist. Es sei hier erwähnt, dass dieses Modell problemlos von einem Minimierungs- zu einem Maximierungsproblem (und

auch umgekehrt) umgewandelt werden kann. Hat man zum Beispiel für die einzelnen Tätigkeiten, anstelle der Ausführungskosten, einen mit der Ausführung verbundenen Profit gegeben und möchte diesen maximieren, so müsste man nur die einzelnen Profite mit -1 multiplizieren und es würde anstelle eines Maximierungs- ein Minimierungsproblem vorliegen.

Beispiele für das Zuordnungsproblem sind:

- **Das Auktionsmodell:** Gegeben sind eine Anzahl an Kunstgegenständen sowie eine gleiche Anzahl an Käufern, wobei von jedem Käufer bekannt ist, wie viel ihm jeder Kunstgegenstand wert ist. Nun soll jedem Interessenten genau ein Gegenstand verkauft werden, wobei der Gesamtverkaufspreis maximiert werden soll.

- **Das Jobproblem:** Gegeben sind eine Anzahl von Arbeitsaufträgen sowie eine gleiche Anzahl von Mitarbeitern bzw. Maschinen, welche diese Aufträge bearbeiten können, wobei bekannt ist in welcher Qualität jeder Mitarbeiter/jede Maschine die einzelnen Aufträge erledigen kann. Nun soll jedem Auftrag ein Mitarbeiter/eine Maschine zugeordnet werden wobei die Gesamtqualität maximiert werden soll.

2.2.2 Differenzen zur Problemstellung

Betrachtet man nun die in dieser Arbeit vorliegende Problemstellung, so sollen auf die gegebenen Stationen nicht nur höchstens eine Ressoure, sondern eben auch mehrere Ressourcen zugeteilt werden können, wodurch das Zuordnungsproblem in dieser Form als Modell nicht geeignet ist. In Abschnitt 2.3 wird eine Version des Zuordnungsproblems vorgestellt, welche diese Möglichkeit berücksichtigt.

2.2.3 Die Ungarische Methode

Für das Zuordnungsproblem sind in der Literatur zahlreiche verschiedene Lösungsansätze zu finden. Auf einige exakte Verfahren wie zum Beispiel Branch and Bound und Lineare Programmierung, welche sich vor allem für kleine und mittelgroße Probleme eignen, sowie auf ein paar gängige Metaheuristiken wird in Abschnitt 2.6 eingegangen. Es handelt sich dabei allesamt um Verfahren, welche sich auf verschiedenste Problemstellungen anwenden lassen, weshalb sie später allgemein betrachtet werden. Ein exaktes Verfahren jedoch, welches speziell für das Zuordnugnsproblem entwickelt wurde, soll hier erwähnt werden, nämlich die **Ungarische Methode**.

Die von Kuhn 1955 veröffentlichte Ungarische Methode [15] ist einer der ersten veröffentlichten Algorithmen zur Lösung des Zuordnungsproblems. Munkres [20] beschäftigte

sich in seiner 1957 verfassten Arbeit mit dieser Methode und konnte zeigen, dass es sich bei dieser um einen streng polynomiellen Algorithmus handelte, wobei die originale Implementierung mit einer Komplexität von $\mathcal{O}(n^4)$ zu einer optimalen Lösung kam. Die Ungarische Methode ist daher oft auch unter dem Namen Kuhn-Munkres Algorithmus bekannt. n ist dabei gleich der Anzahl der vorliegenden Ressourcen bzw. Stationen. Sollte die Anzahl an Stationen und Ressourcen unterschiedlich sein, so ist n das Maximum der beiden. Das Verfahren wurde in den letzten Jahrzehnten noch oft weiter verbessert und für spezielle Problemstellungen adapiert. So konnten Edmonds und Karp [7] den Algorithmus zum Beispiel so modifizieren, dass dieser bereits in $\mathcal{O}(n^3)$ zu einer optimalen Lösung kam. Aufgrund ihrer Komplexität eignet sich die Ungarische Methode nur für Probleme moderater Größe.

2.3 Das verallgemeinerte Zuordnungsproblem

Für das verallgemeinerte Zuordnungsproblem wird, wie schon für das klassische Zuordnungsproblem, angenommen, dass jeder Tätigkeit genau eine Ressource zugeordnet wird. Allerdings ist es in diesem Fall möglich, dass einer Ressource mehr als eine Tätigkeit zugewießen wird. Zu diesem Zwecke erhält jede Ressource eine Kapazität und von jeder Tätigkeit ist zusätzlich bekannt, wie viel von der Kapazität einer Ressource verbraucht wird, um diese Tätigkeit auszuführen.

Mathematisch formuliert sieht dies wie folgt aus:

Minimiere

$$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{i,j}x_{i,j} \tag{2.5}$$

Unter den Nebenbedingungen

$$\sum_{i=1}^{m} x_{i,j} = 1, \quad j = 1, ..., n \tag{2.6}$$

$$\sum_{j=1}^{n} a_{i,j}x_{i,j} \leq b_i, \quad i = 1, ..., m \tag{2.7}$$

$$x_{i,j} \in \{0, 1\} \tag{2.8}$$

Zusätzlich zum in Abschnitt 2.2 eingeführten Modell, wird hier für eine Ressource i noch deren Kapazität b_i berücksichtigt, sowie die Menge $a_{i,j}$, die von dieser Kapazität

verbraucht wird, wenn die Ressource einer Tätigkeit j zugewießen wird. Nebenbedingung 2.7 stellt sicher, dass durch die Zuweisung von Tätigkeiten zu einer Ressource deren Kapazität nicht überschritten wird.

Betrachten wir noch einmal die Beispiele aus Abschnit 2.2, so würden sich diese wie folgt verändern:

- **Das Auktionsmodell:** Nun könnte man jedem Käufer erlauben, mehrere Kunstgegenstände zu kaufen. Die Kapazität jedes Käufers wäre das Geld, welches ihm zur Verfügung steht und der Gesamtwert aller von ihm gekauften Gegenstände dürfte diese Kapazität nicht überschreiten.

- **Das Jobproblem:** Jeder Mitarbeiter könnte in diesem Fall mehrere Arbeitsaufträge annehmen, wobei für jeden Arbeistauftrag zusäztlich bekannt ist, wie viel Zeit dieser in Anspruch nimmt. Die Summe der Dauern der Arbeitsaufträge, welche ein Mitarbeiter annimmt darf außerdem die verfügbare Arbeitszeit des Mitarbeiters nicht überschreiten.

Mit dieser verallgemeinerten Version des Zuweisungsproblems würde sich das in dieser Arbeit vorliegende Problem also schon besser modellieren lassen, da es hiermit möglich wäre, einer Station mehrere Ressourcen zuzuweisen. Des weiteren ist für die Stationen bekannt, wie viele Ressourcen maximal dort arbeiten können. Dieses Maximum könnte man als Kapazität der einzelnen Stationen einsetzen, wobei jede Zuweißung einer Ressource zu einer Station deren Kapazität um genau 1 verringern würde. Aber auch mit diesem Modell lässt sich das vorliegende Problem nicht vollständig modellieren, da die Beziehungen zwischen den einzelnen Stationen nicht berücksichtigt werden. Der Spezialfall des vorliegenden Problems, bei welchem keine Station seriell zu einer zweiten Station geschalten ist, würde sich allerdings schon als verallgemeinertes Zuweisungsproblem abbilden lassen. In Abschnitt 2.4 wird ein Modell vorgestellt, welches zusätzlich auch seriell arbeitende Stationen berücksichtigt.

2.4 Assembly Line Worker Assignment and Balancing Problem

Um das Problem der Abhängigkeiten zwischen den Stationen ebenfalls Abbilden zu können, betrachten wir nun das Assembly Line Worker Assignment and Balancing Problem (ALWABP). Bei diesem möchte man, bei einer gegebenen Anzahl von Ressourcen, die Produktivität einer Montagelinie maximieren, wobei die einzelnen Ressourcen dabei zu Stationen zugeteilt werden sollen und den Ressourcen wiederum Aufgaben zugeteilt werden. In der Montagelinie arbeiten sämtliche Stationen in Serie. Als Maß für die Produktivität wird die Taktzeit genommen. Diese ist gleich dem Maximum der Bearbeitungszeiten

der einzelnen Stationen. Das ALWABP wurde von Miralles et al. [17] eingeführt und, unter Verwendung der in Tabelle 2.1 angeführten Notation, wie folgt definiert:

Tabelle 2.1: Notation für das ALWABP

i, j	Aufgabe
h	Ressource
s	Station
N	Menge an Aufgaben
H	Menge verfügbarer Ressourcen
S	Menge der Stationen
A	Menge der a priori Aufgabe-Ressource Zuteilungen (i, h)
I	Menge der inkompatiblen Aufgabe-Ressource Zuteilungen (i, h)
Z	Menge der a priori Ressource-Station Zuteilungen (h, s)
C	Taktzeit
m	Anzahl der Stationen
p_{hi}	Bearbeitungszeit von Aufgabe i für Ressource h
$lowpi_i$	Kürzeste Bearbeitungsdauer von Aufgabe i unter allen verfügbaren Ressourcen
D_j	Menge der Aufgaben welche direkte Nachfolger von Aufgabe j sind
x_{shi}	1 falls Aufgabe i Ressource h auf Station s zugewießen wurde. 0 andernfalls
y_{sh}	1 falls Ressource h Station s zugewießen wurde

Minimiere

$$z = C \tag{2.9}$$

Mit den Nebenbedingungen

$$\sum_{h \in H} \sum_{s \in S} x_{shi} = 1 \quad \forall i \in N \tag{2.10}$$

$$\sum_{s \in S} y_{sh} \leq 1 \quad \forall h \in H \tag{2.11}$$

$$\sum_{h \in H} y_{sh} \leq 1 \quad \forall s \in S \tag{2.12}$$

$$\sum_{h \in H} \sum_{s \in S} s * x_{shi} \leq \sum_{h \in H} \sum_{s \in S} s * x_{shj} \quad \forall i, j | i \in D_j \tag{2.13}$$

$$\sum_{i \in N} p_{hi} * x_{shi} \leq C \quad \forall h \in H; \forall s \in S \tag{2.14}$$

$$\sum i \in N x_{shi} \leq M * y_{sh} \quad \forall h \in H; \forall s \in S \tag{2.15}$$

wobei

$$y_{sh} \in [0,1] \quad \forall h \in H; \forall s \in S \tag{2.16}$$

$$x_{shi} \in [0,1] \quad \forall h \in H; \forall s \in S; \forall i \in N \tag{2.17}$$

$$M > \sum_{h \in H} \sum_{i \in N} p_{hi} \tag{2.18}$$

Die angeführten Gleichungen und Ungleichungen sind wie folgt zu interpretieren

- Die Zielfunktion 2.9 minimiert die Taktzeit.
- Mit Bedingung 2.10 wird festgelegt, dass jede Aufgabe i genau einer Station s und einer Ressource h zugewießen wird.
- Durch 2.11 und 2.12 wird sichergestellt, dass jede Ressource genau einer Station zugewießen wird und dass jede Station genau eine Ressource aufnimmt.
- Nebenbedingung 2.13 stellt die Vorrangregeln zwischen den Aufgaben i und j dar, wobei i der Vorgänger von j ist.
- 2.14 und 2.15 legen fest, dass jede Ressource h, welche einer Station s zugewießen ist, mehr als eine Aufgabe zugeteilt bekommen kann, solange die Taktzeit C dabei nicht überschritten wird.

Im Gegensatz zu den beiden bisher betrachteten Modellen, bietet dieses Modell die Möglichkeit, Abhängigkeiten zwischen verschiedenen Arbeitsschritten zu berücksichtigen. Möchte man die in dieser Arbeit vorliegende Problemstellung als ALWABP formulieren, so stößt man aber auf folgendes Problem. Beim ALWABP wird davon ausgegangen, dass jedes Erzeugnis sämtliche Arbeitsschritte durchlaufen muss, um fertiggestellt zu werden, wohingegen in dem vorliegenden Problem Erzeugnisse durchaus auch Arbeitsschritte auslassen können. Dieser Fall ist sogar sehr wahrscheinlich. Man denke zum Beispiel an Güter, welche auf einer Palette an ein Großlager geliefert werden und dort anschließend depalettiert und in einem Hochregal gelagert werden sollen. In vielen Lägern ist es üblich, diese Depalettierung automatisiert vorzunehmen, allerdings kann bei zu hoher Auslastung, einer Störung der Anlage oder bei speziellen Gütern alternativ auch auf manuelle

Depalettierung umgestellt werden. In keinem Fall aber muss sowohl automatisch als auch manuell depalettiert werden. Dieses Problem spiegelt sich auch in der Zielfunktion wieder. Die Minimierung der Taktzeit, welche ihrerseits das Maximum der Bearbeitungszeiten der einzelnen Stationen darstellt, ist zwar für Fließbandarbeit plausibel, für ein System bei welchem Erzeugnisse nicht zwingend alle Stationen durchlaufen müssen, ist dies aber nicht der Fall.

2.5 Weitere veröffentlichte Probleme

In der Literatur finden sich noch zahlreiche Arbeiten, welche sich mit dem HRAP beschäftigen. Eine umfangreiche Zusammenfassung bietet die schon in Abschnitt 2.1 erwähnte Arbeit von Bouajaja und Dridi [2]. Auch die Ausführungen von Pentico [21] über Zuweisungsprobleme bieten einen ausgezeichneten Ausgangspunkt auf der Suche nach weiterer Literatur. Allerdings konnte ich, trotz ausgiebigem Studium der vorhanden Literatur, kein Modell finden, welches das in dieser Arbeit vorliegende Problem samt sämtlichen Nebenbedingungen vollständig abbildet. Aus diesem Grund wird in Kapitel 3 ein neues Modell vorgestellt, welches speziell für dieses Problem erstellt wurde.

2.6 Lösungsansätze

Im folgenden Abschnitt werden einige Verfahren vorgestellt, welche in der Literatur häufig im Zusammenhang mit dem HRAP zu finden sind. Dabei werden sowohl exakte als auch heuristische Verfahren behandelt. Grundsätzlich soll hier nur ein Überblick über die einzelnen Lösungsmethoden gegeben werden, ohne dabei zu sehr ins Detail zu gehen.

Branch and Bound

Das Branch and Bound Verfahren ist eines der gängigsten exakten Lösungsverfahren in der kombinatorischen Optimierung. Die Idee dabei ist, den Lösungsraum schrittweise in kleinere Teilmengen aufzuteilen (Branching) und für diese Teilmengen anschließend eine untere/obere Schranke (je nachdem ob ein Minimierungs- oder Maximierungsproblem vorliegt) auszurechnen, mithilfe derer man bestimmen kann ob dieser Teil des Lösungsraums eine optimale Lösung beinhalten kann (Bounding). Bei den Schranken handelt es sich üblicherweise um die Lösung einer vereinfachten Version des Problems. Kann man anhand der Schranke feststellen, dass diese Teilmenge des Lösungsraums keine optimale Lösung enthalten kann, so muss man sie nicht weiter unterteilen und kann sämtliche

in ihr enthaltenen Lösungen verwerfen. Ein Beispiel für einen Branch and Bound Algorithmus in Verbindung mit einem komplexeren Zuordnungsproblem findet man in Borba und Ritt [1], welche sich mit dem in Abschnitt 2.4 vorgestellten Assembly Line Worker Assignment and Balancing Problem auseinandergesetzt haben.

Lineare Optimierung

Die lineare Optimierung, oft auch lineare Programmierung genannt, gehört zu den wichtigsten Verfahren des Operations Research. Das Ziel dabei ist, eine lineare Zielfunktion zu optimieren, welche durch eine Anzahl linearer Gleichungen und Ungleichungen eingeschränkt ist. Eines der ersten Verfahren, welches Probleme der linearen Optimierung effizient lösen konnte, war die von Dantzig 1947 beschriebene Simplex Methode [4]. Einer der großen Vorteile der linearen Programmierung ist die Möglichkeit, verschiedenste Nebenbedingungen in das Problem miteinzubauen. Die erfolgreiche Anwendung linearer Optimierung auf ein komplexes Zuordnungsproblem sieht man zum Beispiel in der Arbeit von Daskalaki et al. [5], die sich mit dem University Timetabling Problem beschäftigten.

Armeisenalgorithmen

Im Gegensatz zu den bisher genannten Methoden handelt sich bei den Armeisenalgorithmen nicht um ein exaktes Verfahren, sondern um eine Metaheuristik. Somit kommt man mithilfe der Armeisenalgorithmen zwar nicht zwingend zu einer optimalen Lösung, allerdings lassen sich mit ihnen wesentlich größere Probleme in vertretbarer Zeit zumindest näherungsweise lösen. Wie der Name schon verrät, orientiert sich dieses Verfahren an dem Verhalten von Armeisen. Wenn diese auf Futtersuche gehen, scheiden sie Pheromone aus, um den zurückgelegten Weg zu markieren. Zu Beginn der Futtersuche wählen die Armeisen ihren Weg noch zufällig aus, allerdings stellt sich auf kürzeren Wegen bald eine höhere Pheromonkonzentration ein, da die Armeisen auf diesem Weg schneller wieder zurückkehren, wodurch die Armeisen sich in der Folge bevorzugt für diese Wege entscheiden.

Dieses Konzept lässt sich in ein künstliches Modell übertragen. Die künstlichen Armeisen in diesem Modell müssen Schritt für Schritt Entscheidungen treffen bis schlussendlich eine vollständige Lösung des Problems entstanden ist. Die Grundlage für diese Entscheidungen bilden gewisse heuristische Informationen, sowie eine künstliche Pheromonmatrix. Die so generierten Lösungen werden anschließend bewertet und abhängig von deren Qualität wird dann die Pheromonmatrix aktualisiert. So setzen sich langfristig Entscheidungen durch, welche zu guten Lösungen für das Problem führen. Ein Beispiel für einen auf ein Zuordnungsproblem angewandten Armeisenalgorithmus findet man in der Arbeit von Jin

und Weng [24].

Partikelschwarmoptimierung

Die Partikelschwarmoptimierung ist ebenfalls eine Methaheuristik welche sich an in der Natur auftretenden Phänomenen orientiert. Konkret wird versucht, das Verhalten von Schwärmen nachzustellen. 1987 gelang es Graig Reynolds [22] die Bewegung von Tierschwärmen anhand folgender drei Gesetze zu simulieren, welche er auf alle Individuen des Schwarms anwandte:

- Kollisionsvermeidung: Vermeide Kollisionen mit anderen Schwarmindividuen und Hindernissen.
- Geschwindigkeitsanpassung: Passe deine Geschwindigkeit der Geschwindigkeit der anderen Individuen an.
- Schwarmzentrierung: Versuche stets beim Schwarm zu bleiben.

Die Geschwindigkeit ist dabei als Vektor zu verstehen. Es zählt also sowohl deren Richtung als auch deren Betrag. Fügt man zum Beispiel als weitere Regel noch hinzu, dass ein Individuum, wenn es ein Futterquelle entdeckt, versuchen soll diese zu erreichen, dann wirkt sich das folgendermaßen auf den Schwarm aus: Sobald ein Individuum eine Futterquelle entdeckt, wird es aufgrund der gerade erwähnten Regel seine Bewegungen ein wenig anpassen. Dadurch werden sich aufgrund der ersten drei Regeln auch die restlichen Individuen etwas anders bewegen. Entdecken mit der Zeit mehrere Individuen die Futterquelle, so wird sich das Verhalten des Schwarms dementsprechend stärker verändern, bis sich der Schwarm schlussendlich direkt auf die Futterquelle zubeweget.

Kennedy und Eberhart [13] führten 1995 zum ersten mal den Begriff der Partikelschwarmoptimierung (PSO) ein. Die Grundidee dabei ist, für ein vorliegendes Problem eine Menge aus Lösungen (den Schwarm) zufällig zu generieren. Die generierten Lösungen, also die Elemente des Schwarms, werden dabei als Partikel bezeichnet. Die einzelnen Partikel bewegen sich dann, anhand der genannten Regeln, durch den Lösungsraum. Anstatt nach einer Futterquelle zu suchen, orientieren sich die einzelnen Partikel in diesem Fall aber an ihrem bisher gefundenen Optimum, sowie dem Optimum aller in der Nähe befindlichen Partikel bzw. am globalen Optimum. Ein Beispiel für ein HRAP welches mittels PSO gelöst wird findet man in der Arbeit von Jia und Gong [12].

Genetische Algorithmen

Eine weitere bedeutende Metaheuristik sind die Genetischen Algorithmen, deren Grundidee es ist, angelehnt an die Evolution, eine Population aus Lösungen miteinander zu kreuzen und so zu einer neuen Generation aus Lösungen zu kommen, welcher idealerweise über bessere Individuen verfügt als ihre Vorgängergeneration. Genetische Algorithmen bilden einen wesentlichen Teil dieser Arbeit und werden in Kapitel 6 detailliert beschrieben.

3 Problembeschreibung

3.1 Formulierung des Grundproblems

Die Ausgangslage des Problems gestaltet sich wie folgt. Gegeben seien m **Stationen** $s = 1, ..., m$ und n **Ressourcen** $r = 1, ..., n$, welche auf die Stationen aufgeteilt werden sollen. Für jede Station s ist eine Anzahl $R_{min,s}$ gegeben, welche angibt, wie viele Ressourcen dieser Station mindestens zugeteilt werden müssen. Gleichzeitig wird durch $R_{max,s}$ angegeben, wie viele Ressourcen die Station maximal aufnehmen kann.

Pro Station ist außerdem der **Bedarf** b_s gegeben, welcher angibt, wie viele Einheiten an dieser Station mindestens produziert werden müssen. Dies muss innerhalb der **verfügbaren Arbeitszeit** der Station geschehen, welche durch t_s gegeben ist. Dadurch ergibt sich auch die minimale Leistung, welche auf der Station erbracht werden muss, damit der Bedarf innerhalb der gegebenen Arbeitszeit erfüllt, wird mit $L_s = b_s/t_s$.

Für eine beliebige Ressource r gibt die **Leistung** $l_{s,r}$ an, wie viele Einheiten die Ressource auf der entsprechenden Station s pro Stunde herstellen kann. Ist $l_{s,r} = 0$, so ist Ressource r nicht befugt auf Station s zu arbeiten und darf dieser dementsprechend auch nicht zugeteilt werden.

Welcher Station die einzelnen Ressourcen zugeteilt sind, wird durch die **Zuteilungsmatrix** Z angebeben, mit

$$z_{s,r} = \begin{cases} 1, \text{ wenn Ressource } r \text{ Station } s \text{ zugeteilt wurde} \\ 0, \text{ andernfalls} \end{cases}$$

Das Ziel ist nun, die gegeben Ressourcen so auf die Stationen zu verteilen, dass die **Gesamtleistung** $L = \sum_{s=1}^{m} \sum_{r=1}^{n} z_{s,r} * l_{s,r}$ maximiert wird, wobei für jede Station s folgende Bedingungen erfüllt sein müssen:

- Die Anzahl der zugeteilten Ressourcen an einer Station s muss innerhalb der gegebenen Grenzen sein: $R_{min,s} <= \sum_{r=1}^{n} z_{s,r} <= R_{max,s}$
- Der Bedarf einer Station s muss mit den ihr zugeteilten Ressourcen innerhalb der

verfügbaren Arbeitszeit der Station erfüllbar sein: $(\sum_{r=1}^{n} z_{s,r} * l_{s,r}) * t_s >= b_s$

- Eine Ressource r darf nicht einer Station s zugeteilt werden, wenn sie für diese nicht befugt ist: $z_{s,r} = 0$ für alle $\{(s,r)|l_{s,r} = 0\}$

Beispiel 3.1. Gegeben sei ein System mit $m = 3$ Stationen und $n = 6$ Ressourcen. Tabelle 3.1 zeigt die Leistungen der Ressourcen auf den einzelnen Stationen und Tabelle 3.2 zeigt die Bedarfe, die zur Verfügung stehende Arbeitszeit, sowie die minimale und maximale Anzahl an Ressourcen pro Station.

Tabelle 3.1: Leistungen $l_{s,r}$ für 3.1

	r_1	r_2	r_3	r_4	r_5	r_6
s_1	25	0	25	35	0	0
s_2	24	20	0	28	16	0
s_3	10	10	10	10	10	10

Tabelle 3.2: Stationseigenschaften für 3.1

	b_s	t_s	L_s	$R_{min,s}$	$R_{max,s}$
s_1	360	8	45	0	4
s_2	160	8	20	1	2
s_3	140	10	14	1	5

Bei einem Problem dieser Größe lassen sich gültige Lösungen noch sehr schnell durch einfaches Ausprobieren finden. Die folgenden beiden Zuteilungen sind beide zulässig, wobei hier schon erkennbar wird, dass bereits bei so einem kleinen Problem, deutlich erkennbare Unterschiede in der Gesamtleistung erzielt werden können. Abbildung 3.1 zeigt die benötigte Mindestleistung pro Station und die tatsächlich erzielten Leistungen pro Station der beiden Lösungen.

Tabelle 3.3: Lösung 1 für Bsp. 3.1

	r_1	r_2	r_3	r_4	r_5	r_6
s_1	1	0	1	0	0	0
s_2	0	1	0	0	1	0
s_3	0	0	0	1	0	1

Tabelle 3.4: Lösung 2 für Bsp. 3.1

	r_1	r_2	r_3	r_4	r_5	r_6
s_1	0	0	1	1	0	0
s_2	1	1	0	0	0	0
s_3	0	0	0	0	1	1

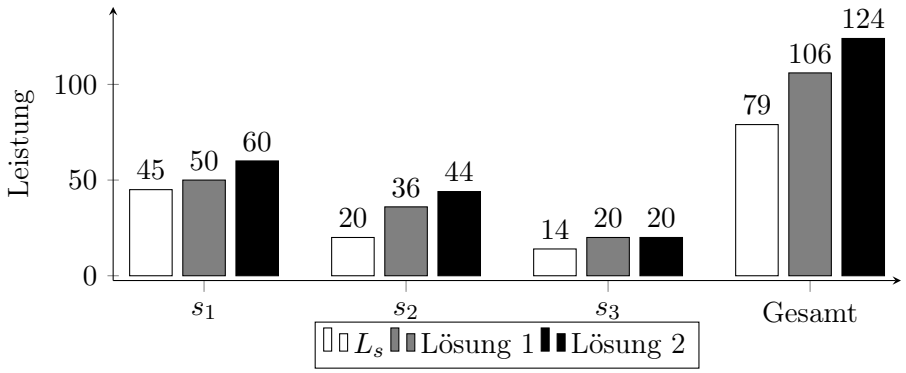


Abbildung 3.1: Leistungsvergleich der beiden Lösungen für Bsp. 3.1

3.2 Formulierung der Nebenbedingungen

3.2.1 Abhängigkeiten

Die einzelnen Stationen arbeiten nicht vollkommen unabhängig voneinander, sondern werden zum Teil voneinander beeinflusst. Stationen können sowohl **seriell** als auch **parallel** arbeiten und je nach Konstellation ergibt dies neue Beschränkungen, welche beachtet werden müssen. Um die nachfolgenden Betrachtungen etwas besser gestalten zu können, legen wir fest, dass für zwei Stationen i und j, welche in Serie geschalten sind, gilt, dass $i < j$, wenn die zu bearbeitende Einheit von Station i zu Station j weitergereicht wird. Zustätzlich sei hier noch angemerkt, dass Kreise in der Produktion in diesen Betrachtungen nicht zulässig sind.

Um die Verknüpfungen zwischen den Stationen darstellen zu können, führen wir hier die **Verteilungsmatrix** D ein, welche, aufgrund der im vorigen Absatz eingeführten Regel, als strikte Dreiecksmatrix konstruiert werden kann. Es handelt sich hier um eine $(m+2) \times (m+2)$-Matrix, denn zusätzlich zu den Verknüpfungen zwischen den einzeln Stationen können hier auch noch Verknüpfungen zu den Pseudostationen 0 und $m+1$ eingetragen werden, wobei 0 die Quelle und $n+1$ die Senke des Systems darstellt. Ein Eintrag $d_{i,j} \in \{0,1\}, i,j \in \{0,1,...,m,m+1\}$ gibt an, ob Station i mit Station j verbunden ist, wobei gilt:

$$d_{i,j} = \begin{cases} 1, \text{ wenn Station } i \text{ mit Station } j \text{ verbunden ist} \\ 0, \text{ andernfalls} \end{cases}$$

Eine Station kann also nur dann ihre volle Leistung erbringen, wenn sie zum einen von ihren Vorgängerstationen ausreichend versorgt wird und zum anderen auch nicht von ihren Nachfolgerstationen blockiert wird. Eine Verbindung von der Quelle zu einer Station bedeutet, dass die Station von einer externen Quelle beliefert wird und es wird hier in weiterer Folge davon ausgegangen, dass eine solche Station immer mit ausreichend Einheiten versorgt wird um mit voller Leistung zu arbeiten. Umgekehrt bedeutet eine Verbindung von einer Station zur Senke, dass die Einheiten dieser Station nach der Bearbeitung aus dem System ausscheiden und es wird hier in weiterer Folge davon ausgegangen, dass diese Stationen nie blockiert werden.

Durch das Einführen der Abhängigkeiten zwischen den Stationen wird das Problem maßgeblich verändert, da die Zuteilung einer Ressource r zu einer Station s nun nicht mehr, wie bisher, die Zielfunktion um einen fixen Betrag $l_{s,r}$ steigert. Stattdessen ist der Beitrag den eine solche Zuteilung liefert davon abhängig, welchen Stationen die anderen Ressourcen zugewießen werden. Man stelle sich hierfür ein simples System vor, welches nur aus den beiden Stationen "Baum fällen" und "Stamm zersägen" besteht. Würde man alle verfügbaren Ressourcen auf nur eine der beiden Stationen zuweisen, so hätte das System

eine Gesamtleistung von 0 Brettern pro Stunde, da entweder keine Stämme zur Verfügung stehen würden, oder aber niemand die Stämme zu Brettern verarbeiten würde. Um die Leistung des Systems inklusive der beschriebenen Abhängigkeiten bestimmen zu können, wird es nun also notwendig, das vorliegende Problem als Flussproblem zu betrachten.

Es gilt nun also, unter den bereits eingeführten Bedingungen, statt der maximalen Gesamtleistung aller Stationen, einen **maximalen Fluss** durch das System zu finden. Um dies zu veranschaulichen, stellen wir das System als endlichen, gerichteten Graphen G=(V,E) dar, wobei jeder Knoten im Graph eine Station repräsentiert. Bei m Stationen ergibt sich damit die Menge der Knoten als $V := \{0, 1, ..., m, m+1\}$ wobei 0 wiederum die Quelle und $(m+1)$ die Senke darstellt. Für jeden Eintrag $d_{i,j} = 1$ in der Verteilungsmatrix wird eine entsprechende Kante konstruiert. Somit ergibt sich die Menge der Kanten als $E := \{(i,j) | i,j \in V; d_{i,j} = 1\}$. Um ein vollständiges Flussproblem vorliegen zu haben, benötigt man natürlich noch für jede Kante ein Kapazität. Diese Kapazitäten werden nach folgenden Überlegungen zugewießen:

- Von jeder Station können maximal so viele Einheiten ausgehen, wie die ihr zugewießenen Ressourcen im Stande sind zu leisten. Die Kapazität aller ausgehenden Kanten jenes Knotens, der diese Station darstellt, wird daher auf einen Wert gesetzt, der der Summe der Leistungen entspricht, welche die dieser Station zugewießenen Ressourcen auf ihr erbringen können. Von dieser Regel ausgenommen sind alle Kanten, welche von der Quelle ausgehen.

$$c_{i,j} = \sum_{r=1}^{n} z_{i,r} * l_{i,r}, \; (i,j) \in E, 1 \leq i,j \leq m+1$$

- Da von einem Knoten i mehrere Kanten ausgehen können und all diese Kanten über eine Kapazität verfügen, welche der maximal von diesem Knoten erzielbaren Leistung entspricht, muss zusätzlich eine Regel eingeführt werden, welche festlegt, dass der **Fluss** durch alle von i ausgehenden Kanten in Summe nicht größer sein darf, als diese maximal erzielbare Leistung.

$$\sum_{j=1}^{m+1} f_{i,j} < \sum_{r=1}^{n} z_{i,r} * l_{i,r}, \; 1 \leq i \leq m; \; (i,j) \in E$$

- Sämtliche Kanten, welche von der Quelle ausgehen, erhalten eine Kapazität von ∞. Dies soll darstellen, dass die Quelle stets ausreichend Einheiten an die nachfolgenden Stationen liefern kann.

$$c_{0,j} = \infty, \; (0,j) \in E, 1 \leq j \leq m$$

Anhand dieser Regeln kann also aus dem gegebenen System ein Flussproblem modelliert werden. Um nun die Leistung des Systems zu bestimmen muss also der maximale Fluss berechnet werden. Diese Berechnung wird in Kapitel 4 genauer behandelt. Im folgenden Beispiel kann man diesen noch problemlos intuitiv ermitteln.

Beispiel 3.2. Betrachten wir nun noch einmal Beispiel 3.1 von vorhin mit der Lösung aus Tablle 3.4. Wir erweitern dieses Beispiel nun um die Verteilungsmatrix D, dargestellt in Tabelle 3.5, welche festlegt, dass Station 1 direkt mit der Quelle verknüpft ist. Sowohl Station 2, als auch Station 3 folgen auf Station 1 und liefern ihre Erzeugnisse direkt an die Senke.

Tabelle 3.5: Verteilungen $V_{i,j}$ für Beispiel 3.2

	s_0	s_1	s_2	s_3	s_4
s_0	0	1	0	0	0
s_1	0	0	1	1	0
s_2	0	0	0	0	1
s_3	0	0	0	0	1
s_4	0	0	0	0	0

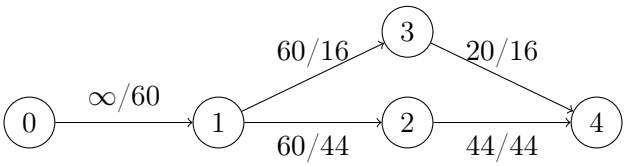


Abbildung 3.2: Graph für Beispiel 3.2

Abbildung 3.2 zeigt den Graphen für das oben eingeführte Beispiel, inklusive der neu hinzugefügten Verbindungen zwischen den einzelnen Stationen. Die Kantenbeschriftungen zeigen jeweils die Kapazität und den tatsächlichen Fluss. Aus diesem Graphen wird auch ersichtlich, dass, aufgrund der Verbindungen zwischen den Stationen, die Leistung des Systems gegenüber der Leistung aus Beispiel 3.1 um 4 Einheiten gesunken ist. Station 3 kann nicht mehr ihre volle Leistung erzielen, da sie von Station 1 nicht ausreichend versorgt wird. Die optimale Lösung ist hier allerdings nicht eindeutig und es könnten genauso gut von Station 1 20 Einheiten zu Station 3 fließen und nur 40 zu Station 2. Der Gesamtfluss durch das System von 60 Einheiten würde sich dadurch nicht ändern.

3.2.2 Puffer

In der Realität versucht man natürlich bei kleinen Leistungsschwankungen und unterschiedlichen Arbeitszeiten an den verschiedenen Stationen, die Kontinuität in der Produktion trotzdem aufrecht zu erhalten. Dies wird meist durch Pufferlager erledigt, welche dabei helfen, diese Schwankungen auszgleichen. Um dies so gut wie möglich abzubilden, soll hier noch die **Pufferkapazitätzmatrix** K eingeführt werden. Hierbei handelt es sich

um eine $m \times m$ Matrix, wobei ein Eintrag $k_{i,j}$, $i, j \in \{1, ..., m\}$ angibt, wie viel Einheiten zwischen Station i und Station j gepuffert werden können. Diese Matrix kann in jedem Fall als strikte Dreiecksmatrix modelliert werden, da man die Elemente unter der Hauptdiagonale auf 0 setzen kann und zwischen ein und der selben Station kein Pufferplatz sein kann, wodurch auch die Elemente der Hauptdiagonale allesamt 0 sind. Die **Puffermatrix** P, ebenfalls eine strikte Dreiecksmatrix, gibt an, wie viele Einheiten sich zu Beginn der Betrachtung bereits in den jeweiligen Pufferspeichern befinden. Sowohl für die Puffermatrix, als auch für die Pufferkapazitätsmatrix gilt: $p_{i,j} = k_{k,j} = 0, \forall (i, j) \notin E$. Es kann also keinen Puffer zwischen zwei Stationen geben, welche nicht miteinander verbunden sind.

Um die Puffer im Graphen darzustellen, müssen ihm neue Kanten hinzugefügt werden. Eine Kante von der Quelle zu einem Knoten repräsentiert dabei einen Puffer, aus welchem die Station Einheiten abarbeiten kann und umgekehrt stellt eine Kante von der Station zur Senke einen Puffer dar, welcher von der Station befüllt werden kann. Die Größe des Puffers wird dabei durch die Kapazität der Kante dargestellt. Angenommen, eine Station j wird von einer Station i mit Einheiten beliefert und zwischen den beiden besteht ein Puffer. In diesem Fall würde folgende neue Kanten eingeführt werden:

- Eine Kante von der Quelle zu Station j, mit $c_{0,j} = \lfloor p_{i,j}/t_j \rfloor$. Dies ist folgendermaßen zu verstehen: Der Puffer enthält $p_{i,j}$ Einheiten, welche von Station j während deren Arbeitszeit t_j entnommen werden können.

- Eine Kante von Station i zur Senke mit $c_{i,m+1} = \lfloor (k_{i,j} - p_{i,j})/t_i \rfloor$, welche so zu verstehen ist: Der Puffer enthält $p_{i,j}$ Einheiten und verfügt über eine Kapazität von $k_{i,j}$ Einheiten. Somit kann er von Station i während deren Arbeitszeit t_i mit $k_{i,j} - p_{i,j}$ Einheiten befüllt werden.

Da Kanten nur ganzzahlige Kapazitäten haben dürfen, muss hier in beiden Fällen abgerundet werden. Es kann also sein, dass bei einer Station i bis zu $t_i - 1$ Einheiten des Puffers nicht berücksichtigt werden, was als vernachlässigbar angenommen wird. Kanten mit Kapazität 0 werden dem Graphen nicht hinzugefügt. Sollten von der Quelle zu einem Knoten, oder von einem Knoten zur Senke mehrere Pufferkanten nötig sein, so können diese zu einer Kante mit summierter Kapazität zusammengefasst werden.

Beispiel 3.3. Zur Veranschaulichung dessen, wie sich die Puffer auf das bisherige Problem auswirken, erweitern wir Beispiel 3.2 nun um eine Pufferkapazitätsmatrix und eine Puffermatrix (Gemeinsam dargestellt in Tabelle 3.6). Diese geben zwischen Station 1 und Station 3 einen Puffer mit einer Kapazität von 80 Einheiten an, welcher mit 40 Einheiten befüllt ist.

Tabelle 3.6: Puffer- und Pufferkapazitätsmatrix für Bsp. 3.3

	s_1		s_2		s_3	
	K	P	K	P	K	P
s_1	0	0	0	0	80	40
s_2	0	0	0	0	0	0
s_3	0	0	0	0	0	0

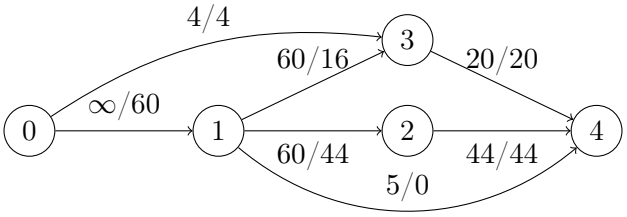


Abbildung 3.3: Graph für Beispiel 3.3

Man sieht hier, dass der Graph entsprechend dem Puffer zwischen Station 1 und Station 3 um zwei Kanten erweitert wurde. Zum einen wurde zwischen der Quelle und Station 3 eine Kante mit $c_{0,3} = P_{1,3}/t_3 = 4$ eingeführt und zum anderen zwischen Station 1 und der Senke eine Kante mit $c_{1,3} = (K_{1,3} - P_{1,3})/t_1 = 5$. Dadurch wird Station 3 wieder mit ausreichend Einheiten versorgt um mit voller Leistung arbeiten zu können. Der Gesamtfluss des Systems steigt somit auf 64 Einheiten an.

3.2.3 Initialbelegung

Erschwerend kommt zu dem bisherigen Problem nun noch hinzu, dass für die Ressourcen schon eine initiale Einteilung besteht, dargestellt durch die Initialmatrix I mit

$$I_{s,r} = \begin{cases} 1, \text{ wenn Ressource } r \text{ Station } s \text{ zugeteilt ist} \\ 0, \text{ andernfalls} \end{cases}$$

Dies bedeutet, dass eine Umschichtung einer Ressource auf eine andere Station eine kurzfristige Leistungseinbuße mit sich bringt, gegeben durch die Umschichtungsmatrix U. Hierbei handelt es sich um eine $m \times m$ Matrix, wobei ein Eintrag $U_{i,j}$, $i, j \in \{1, ..., m\}$ angibt, wie lange es für eine Ressource dauert um von Station i zu Station j zu wechseln. Somit sinkt die Leistung einer Ressource r welche auf Station j eingeteilt wird um $\lceil l_{j,r} * U_{i,j}/t_j \rceil$, wenn sie ursprünglich auf Station i eingeteilt war. Es gilt $U_{i,i} = 0$ für alle

$i \in \{1, ..., m\}$ da es keine Leistungseinbuße gibt, wenn eine Ressource auf der selben Station eingeteilt bleibt.

Beispiel 3.4. Um das Problem mit sämtlichen Zusatzbedingungen zu veranschaulichen, betrachten wir nun ein System mit $m = 4$ Stationen und $n = 8$ Ressourcen. Tabelle 3.7 zeigt die Leistungen der Ressourcen auf den einzelnen Stationen und Tabelle 3.8 zeigt die Bedarfe, die zur Verfügung stehende Arbeitszeit, sowie die minimale und maximale Anzahl an Ressourcen pro Station.

Tabelle 3.7: Leistungen für 3.4

$l_{s,r}$	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
s_1	40	0	40	50	0	0	36	40
s_2	24	20	0	28	16	0	20	0
s_3	32	0	26	24	36	0	22	34
s_4	10	10	10	10	10	10	10	10

Tabelle 3.8: Stationseigenschaften für 3.4

	b_s	t_s	L_s	$R_{min,s}$	$R_{max,s}$
s_1	400	8	50	0	4
s_2	160	8	20	1	2
s_3	180	6	30	2	8
s_4	140	10	14	1	5

Des weiteren gibt es für die Ressourcen in diesem Beispiel bereits eine initiale Belegung welche berücksichtigt werden muss. Tabelle 3.9 gibt diese Belegung an und Tabelle 3.10 gibt die Kosten für die Umschichtung eines Mitarbeiters zwischen bestimmten Stationen an.

Tabelle 3.9: Initialbelegung für Bsp. 3.4

I	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
s_1	1	0	1	1	0	0	0	0
s_2	0	1	0	0	1	0	0	0
s_3	0	0	0	0	0	0	1	0
s_4	0	0	0	0	0	1	0	1

Tabelle 3.10: Umschichtungsmatrix Bsp. 3.4

U	s_1	s_2	s_3	s_4
s_1	0	0.2	0.1	0.25
s_2	0	0	0.2	0.2
s_3	0	0	0	0.25
s_4	0	0	0	0

Zusätzlich gibt es in diesem Beispiel wieder Abhängigkeiten sowie Puffer zwischen den einzelnen Stationen. Die Pufferkapazitätsmatrix, die Puffermatrix sowie die Verteilungsmatrix sind in Tabelle 3.11 dargestellt.

Tabelle 3.11: Verteilungs-, Puffer- und Pufferkapazitätsmatrix für Bsp. 3.4

	s_1			s_2			s_3			s_4		
	V	P	K	V	P	K	V	P	K	V	P	K
s_1	0	0	0	1	30	100	1	80	80	0	0	0
s_2	0	0	0	0	0	0	0	0	0	0	0	0
s_3	0	0	0	0	0	0	0	0	0	1	20	50
s_4	0	0	0	0	0	0	0	0	0	0	0	0

In diesem Beispiel werden also die Stationen s_2 und s_3 von s_1 bedient. Die Erzeugnisse von s_3 gehen anschließend auch noch weiter an s_4. Zusätzlich gibt es Puffer zwischen den Stationen 1 und 2, 1 und 3 sowie zwischen den Stationen 3 und 4.

Wie schon in Beispiel 3.1, sollen auch hier zwei gültige Lösungen miteinander verglichen werden, um den Unterschied im Gesamtfluss aufzuzeigen, welcher bereits bei so geringer Problemgröße auftreten kann. Die Tabellen 3.12 und 3.13 zeigen die Zuteilung der Ressourcen der jeweiligen Lösung und Abbildung 3.4 sowie Abbildung 3.5 zeigen den aus der jeweiligen Zuteilung resultierenden Graphen. Zu guter Letzt werden in Grafik 3.6 für die verschiedenen Zuteilungen noch die Leistungen der einzelnen Stationen, die Gesamtleistung aller Stationen, sowie der Gesamtfluss durch das System verglichen.

Tabelle 3.12: Lösung 1 für Bsp. 3.4

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
s_1	0	0	0	1	0	0	0	0
s_2	0	1	0	0	1	0	0	0
s_3	1	0	0	0	0	0	1	0
s_4	0	0	1	0	0	1	0	1

Tabelle 3.13: Lösung 2 für Bsp. 3.4

	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
s_1	0	0	1	1	0	0	0	0
s_2	0	1	0	0	0	0	0	0
s_3	1	0	0	0	1	0	0	0
s_4	0	0	0	0	0	1	1	1

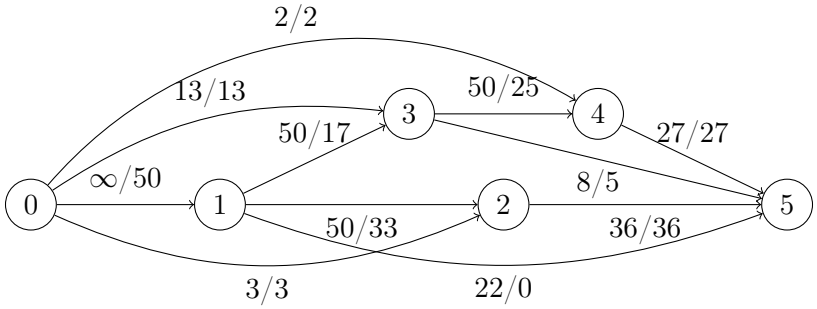


Abbildung 3.4: Graph für Beispiel 3.4, Zuteilung 1

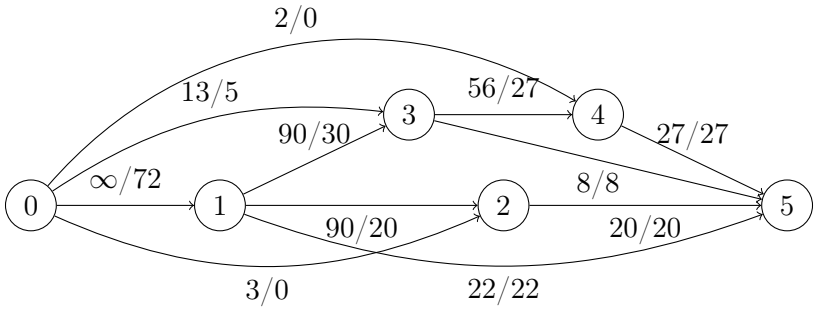


Abbildung 3.5: Graph für Beispiel 3.4, Zuteilung 2

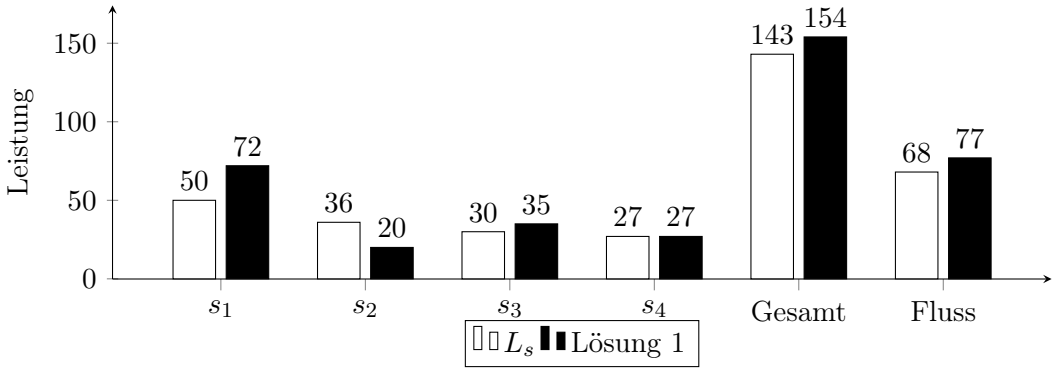


Abbildung 3.6: Leistungsvergleich der beiden Lösungen für Bsp. 3.4

Aus Abbildung 3.6 kann man ablesen, dass der Gesamtfluss durch das System in Lösung 2 um ca. 13% größer ist als jener durch Lösung 1. Die insgesamt auf allen Stationen erbrachte Leistung unterscheidet sich schon nicht mehr so deutlich und Lösung 2 erzielt hier nur in etwa 7,7% mehr Einheiten pro Stunde als Lösung 1. Im weiteren Verlauf der Arbeit wird als Zielfunktion stets der Fluss betrachtet. In Kapitel 3.4 werden allerdings noch ein paar alternative Zielfunktionen behandelt und deren Vor- und Nachteile beschrieben.

3.2.4 Unerfüllbare Bedarfe

Selbstverständlich kann es auch vorkommen, dass von dem System mehr verlangt wird, als es mit den zur Verfügung stehenden Ressourcen produzieren kann. In diesem Fall gibt es bei jeder möglichen Ressourcenaufteilung für mindestens eine Station i einen Bedarf

b_i, welcher nicht erfüllt werden kann. In diesem Fall soll natürlich trotzdem eine bestmögliche Aufteilung gefunden werden. Um diese Situationen bewerten zu können, muss die harte Bedingung, welche voraussetzt, dass alle Bedarfe erfüllt werden, aufgehoben werden und dafür eine **Strafe** auf das Nichterfüllen eines Bedarfes gesetzt werden. Es gibt nun verschiedene Möglichkeiten diese Strafe zu formulieren. Ein paar Beispiele wären

- Für jede nichterfüllte Einheit des Bedarfs b_i auf Station i wird ein festgesetzter Strafbetrag S von der Zielfunktion abgezogen.

$$L = L - \sum_{i=1}^{m}(\max(b_i - (\sum_{j=1}^{n} Z_{i,j} * l_{i,j}), 0)) * S$$

- Für jede Station i, auf welcher der Bedarf nicht vollständig erfüllt wird, wird ein festgesetzter Strafbetrag S von der Zielfunktion abgezogen.

$$L = L - \sum_{i=1}^{m}(\min(\max(b_i - (\sum_{j=1}^{n} Z_{i,j} * l_{i,j}), 0), 1)) * S$$

- Für jede Station i wird ein festgesetzter Strafbetrag pro Anteil an nicht erfülltem Bedarf vergeben.

$$L = L - \sum_{i=1}^{m}(\frac{\max(b_i - (\sum_{j=1}^{n} Z_{i,j} * l_{i,j}), 0)}{b_i}) * S$$

Da es für größere Systeme bereits eine aufwendige Aufgabe ist, festzustellen, ob es überhaupt eine Lösung gibt, welche alle Bedarfe abdeckt und dies den Rahmen dieser Arbeit sprengen würde, wird für die in dieser Arbeit untersuchten Probleme stets vorausgesetzt, dass es für die gegebenen Ressourcen immer mindestens eine Zuteilung gibt, mit welcher alle Bedarfe erfüllbar sind. Es gibt allerdings einige Bedingungen, welche zum Ermitteln der Lösbarkeit des Systems zwar nicht ausreichend aber zwingend notwendig sind, welche man mit sehr geringem Rechenaufwand überprüfen kann.

- Die Anzahl n verfügbarer Ressourcen muss mindestens gleich der Summe der Mindestressourcen aller Stationen sein:

$$n \geq \sum_{s=1}^{m} R_{min,s}$$

- Für jede beliebige Station s muss die Summe der erbringbaren Leistungen aller Ressourcen mindestens so groß sein, wie die erforderliche Mindestleistung dieser Station:

$$\sum_{r=1}^{n} l_{s,r} \geq L_{s,r}$$

- Die Summe der maximalen Leistungen, welche die einzelnen Ressourcen auf allen Stationen erbringen können, muss mindestens gleich der Summe der Mindesleistungen aller Stationen sein:

$$\sum_{r=1}^{n} \max_{s=1}^{m} l_{s,r} \geq \sum_{s=1}^{m} b_s$$

3.3 Unterstützende Prozessschritte

Es kann für das System auch notwendig sein, dass Stationen besetzt werden, welche zur Gesamtleistung des Systems nicht direkt beitragen. Man stelle sich zum Beispiel eine Station *Lagerleitung* vor, welche aus organisatorischen Gründen zwingend notwendig ist, aber eigentlich keine Einheiten produziert und somit keine direkt messbare Leistung erbringt. Um zu erzwingen, dass auf diesen Stationen Ressourcen eingesetzt werden, muss man zuerst einmal die Ressourcen befähigen auf diesen Stationen zu arbeiten. Dies schafft man, indem man die entsprechende Leistung $l_{s,r}$ auf 1 setzt. Der statische Ansatz zur Lösung dieses Problems, wäre es über $R_{min,s}$ und $R_{max,s}$ eine gewisse Mindestanzahl bzw. ein Maximum an Ressourcen zu erzwingen. Da R_{min}/R_{max} aber eigentlich Konstanten sind welche die Systemgrenzen beschreiben, sollte man diesen Ansatz vermeiden.

Möchte man nun etwas dynamischer erzwingen, dass x Ressourcen auf Station s eingesetzt werden, so setzt man den Bedarf $b_s = t_s * x$. Dadurch ergibt sich für die Mindestleistung $L_s = b_s/t_s = x$. Da jeder Mitarbeiter, welcher auf dieser Station arbeiten kann, nur über eine Leistung von 1 verfügt, müssen also x Mitarbeiter zugeteilt werden um den Bedarf zu decken.

3.4 Alternative Zielfunktionen

Selbstverständlich gibt es für das bisher beschriebene Problem nicht nur eine einzige Zielfunktion, sondern es lassen sich noch einige weitere formulieren. Auf ein paar dieser Zielfunktionen soll in diesem Unterkapitel eingegangen werden und dabei sollen diese auch in einen praxisnahen Kontext gesetzt werden.

- Als erste Zielfunktion betrachten wir den **maximalen Fluss**, welcher auch im weiteren Verlauf der Arbeit die größte Relevanz besitzt. Dieser ist als Zielfunktion deshalb so interessant, da das Maximieren des FLusses bedeutet, dass man die Anzahl der pro Zeiteinheit erzeugten Endprodukte maximiert. Dies bedeutet umgekehrt natürlich, dass man eine gewünschte Menge an Enderzeugnissen in kürzerer Zeit produzieren kann, was sowohl zur Steigerung der Flexibilität als auch zur Senkung der Kosten, insbesondere der Personalkosten, beiträgt. Berechnet wird der Fluss mit:

$$L = \sum_{s=1}^{m} f_{0,m}$$

- Als nächstes wenden wir uns jener Zielfunktion zu, welche ganz zu Beginn, bei der Einführung des Problems schon verwendet wurde, nämlich der **Gesamtleistung** aller Stationen. Diese unterscheidet sich von der Zielfunktion des maximalen Flusses dadurch, dass beim maximalen Fluss nur die Anzahl der Endprodukte zählt, während zur Gesamtleistung auch die Teilerzeugnisse beitragen. Es kann durchaus sein, das ein System bei steigender Gesamtleistung, eine geringere Anzahl von Endprodukten erzeugt. Dies könnte zum Beispiel auf eine Überbearbeitung zurückzuführen sein, d.h. dass ein Produkt mehr Produktionsschritte durchläuft, als für die erforderliche Qualität eigentlich notwendig wäre. Diese Zielfunktion eignet sich also besonders dann, wenn man neben dem Maximieren der Anzahl von Enderzeugnissen auch noch ein anderes Ziel verfolgt, wie zum Beispiel das Maximieren der Qualität. Würde man in Beispiel 3.4 den Stationen 2 und 4 noch eine optionale Station Qualitätskontrolle nachlagern, auf welcher die Mitarbeiter eine sehr hohe Leistung erbringen können, so würde die Gesamtleistung steigen wenn man diese Station mit einem Mitarbeiter besetzt, wohingegen der Fluss sinken würde. Die Gesamtleistung ist definiert als:

$$L = \sum_{s=1}^{m} \sum_{r=1}^{n} z_{s,r} * l_{s,r}$$

- Geht man, wie in Abschnitt 3.2.4 beschrieben, davon aus, dass nicht immer alle Bedarfe erfüllt werden können, so bietet sich als Zielfunktion auch das **Minimieren der Strafe** für deren Nichterfüllung an. Dies wäre in der Praxis sehr zielführend, wenn man beim Nichteinhalten von Fristen große Pönalen zu zahlen hat und man daher in erster Line versucht ist, Aufträge zeitgerecht fertigzustellen. Wie schon im vorigen Abschnitt beschrieben gibt es mehrere Möglichkeiten eine Strafe zu definieren. Exemplarisch könnte die zu minimierende Zielfunktion zum Beispiel so aussehen:

$$\sum_{i=1}^{m} (\min(\max(b_i - (\sum_{j=1}^{n} z_{i,j} * l_{i,j}), 0), 1)) * S$$

4 Flussprobleme

Gegeben sei ein endlicher, zusammenhängender, gerichteter **Graph** G(V,E) mit $|V| = m$. Zwei Knoten des Graphen verfügen über spezielle Eigenschaften, nämlich zum einen die Quelle s für welche gilt $\delta_G^-(s) = 0$ sowie $\delta_G^+(s) > 0$ und zum anderen die Senke t für welche gilt $\delta_G^-(t) > 0$ sowie $\delta_G^+(t) = 0$. Hierbei bezeichnet $\delta_G^+(s)$ die Anzahl aller aus s hinausführenden Kanten und $\delta_G^-(s)$ die Anzahl aller in s hineinführenden Kanten. Für alle anderen Knoten $v \in V \setminus \{s,t\}$ gilt $\delta_G^-(v) > 0$ sowie $\delta_G^+(v) > 0$. Jedes Problem, bei welchem es mehrere Quellen und/oder Senken gibt, kann problemlos in ein Problem mit nur einer Quelle und einer Senke überführt werden. Des weiteren wird jeder Kante $e \in E$ eine nichtnegative, rationale Zahl $u(e)$ als **Kapazität** zugeordnet. Der Graph, gemeinsam mit den Kapazitäten der einzelnen Kanten sowie der Quelle und der Senke, werden als **Netzwerk** (G,u,s,t) bezeichnet. Unter dem **Fluss** versteht man eine Abbildung $f : E(G) \to \mathbb{Q}_{\geq 0}$, welche folgende Bedingungen erfüllt:

- *Kapazitätsbeschränkung*: $0 \leq f(e) \leq u_{i,j}$, $\forall\ e = (v_i, v_j) \in E(G)$
- *Flusserhaltung*: $\sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)} f(e)$, $\forall\ v \neq s, v \neq t$

Das Ziel ist es nun, die maximale Anzahl von Einheiten die pro Zeiteinheit von der Quelle zur Senke transportiert werden können, den sogenannten **maximalen Fluss**, zu bestimmen. Der einfachste Weg, um dies zu erreichen ist der Algorithmus von Ford und Fulkerson [9]. Bevor wir uns diesen ansehen, bedienen wir uns zuerst noch der Definitionen des **Residualgraphens** und des **augmentierenden Weges** nach Korte und Vygen[14].

Sei G ein gerichteter Graph und $e = (i,j) \in E(G)$. Dann sei $\overleftarrow{e}$ eine neue Kante von j nach i, die sogennante **gegenläufige Kante** von e. Ebenso ist e die gegenläufige Kante von $\overleftarrow{e}$. Nun definieren wir den Graphen $\overleftrightarrow{G} := (V(G), E(G) \cup \{\overleftarrow{e} : e \in E(G)\})$.

Gegeben sei ein gerichteter Graph G mit Kapazitäten $u : E(G) \to \mathbb{R}_+$ und ein Fluss f. Dann definieren wir die **Residualkapazitäten** $u_f : E(\overleftrightarrow{G}) \to \mathbb{R}_+$ durch $u_f(e) := u(e) - f(e)$ und $u_f(\overleftarrow{e}) := f(e)$ für alle $e \in E(G)$. Der **Residualgraph** G_f ist der Graph $(V(G), \{e \in E(\overleftrightarrow{G} : u_f(e) > 0\})$.

Gegeben sei ein Fluss f und ein Weg P in G_f. Den Fluss f entlang P um γ zu **aug-**

mentieren bedeutet, man erhöhe $f(e)$ um γ für jedes $e \in E(P)$ mit $e \in E(G)$, und man verringere $f(e_0)$ um γ für jedes $e \in E(P)$ mit $e = \overleftarrow{e_0}$ für ein $e_0 \in E(G)$.

Gegeben sei ein Netzwerk (G, u, s, t) und ein s-t-Fluss f. Dann ist ein **f-augmentierender Weg** ein s-t-Weg in dem Residualgraphen G_f.

4.1 Algorithmus von Ford und Fulkerson

Unter Verwendung der oben angeführten Definitionen lässt sich nun der folgende Algorithmus leicht beschreiben.

Algorithmus 1 Algorithmus von Ford und Fulkerson

Input Ein Netzwerk (G, u, s, t) mit $u : E(G) \to \mathbb{Z}_+$
Output Ein s-t-Fluss mit maximalem Wert

1: Setze $f(e) := 0$ für alle $e \in E(G)$
2: **while** Ein s-t-Weg in G_f existiert **do**
3: Bestimme einen s-t-Weg P in G_f
4: Bestimme $\gamma := min_{e \in E(P)} u_f(e)$
5: Augmentiere f entlang P um γ
6: **end while**

Es ist zu zu beachten, dass hier nur ganzzahlige Kantenkapazitäten zugelassen werden. Erlaubt man aber zum Beispiel irrationale Kantenkapazitäten, so können Probleme konstruiert werden, welche, bei schlechter Wahl der augmentierenden Wege, nicht terminieren, was von Ford und Fulkerson durch ein geeignetes Netzwerk gezeigt werden konnte [10]. Es gilt nun noch zu beweisen, dass der Algorithmus, wenn er terminiert, einen maximalen Fluss findet. Für die Beweisführung werden erneut die Darstellungen von Korte und Vygen [14] herangezogen.

Lemma 4.1. Für jede Knotenmenge $A \leq V(G)$ mit $s \in A$ und $t \notin A$ und für jeden s-t-Fluss f gilt:

(a) $wert(f) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e)$

(b) $wert(f) \leq \sum_{e \in \delta^+(A)} u(e)$

Beweis. (a): Da die Flusserhaltungsregel für $\nu \in A \backslash \{s\}$ erfüllt ist, gilt:

$$wert(f) = \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e)$$
$$= \sum_{\nu \in A} (\sum_{e \in \delta^+(\nu)} f(e) - \sum_{e \in \delta^-(\nu)} f(e))$$

$$= \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e)$$

(b): Diese Aussage folgt aus (a), da $0 \leq f(e) \leq u(e)$ für alle $e \in E(G)$ gilt.

□

Theorem 4.2. Ein s-t-Fluss f hat genau dann maximalen Wert, wenn es keinen f-augmentierenden Weg gibt.

Beweis. Gibt es einen augmentierenden Weg P, so wird in Schritt 3 ein Fluss mit höherem Wert berechnet, also ist f kein Fluss mit maximalem Wert. Gibt es andererseits keinen augmentierenden Weg, so bedeutet dies, dass t nicht von s aus in G_f erreichbar ist. Sei R die Menge der von s aus in G_f erreichbaren Knoten. Nach der Definition von G_f folgt $f(e) = u(e)$ für alle $e \in \delta_G^+(R)$ und $f(e) = 0$ für alle $e \in \delta_G^-(R)$. Nach Lemma 4.1 (a) folgt aber

$$wert(f) = \sum_{e \in \delta_G^+(R)} u(e),$$

und mit Lemma 4.1 (b) folgt dann, dass f ein Fluss mit maximalem Wert ist. □

4.2 Maximum Flow Algorithmen und deren Laufzeit

In diesem Abschnitt sollen einige Verbesserungen des Algorithmus von Ford und Fulkerson vorgestellt werden, welche einen positiven Einfluss auf die Laufzeit mit sich bringen. Die in diesem Abschnitt verwendeten Definitionen, Sätze und Beweise lehnen wieder an die Ausführungen von Korte und Vygen [14] an.

4.2.1 Algorithmus von Edmonds und Karp

Dieser Algorithmus wurde von Edmonds und Karp 1972 [8] dargestellt und dient zur Berechnung eines maximalen s-t-Fluss in Netzwerken mit positiven reellen Kapazitäten. Die Laufzeit des Algorithmus von $\mathcal{O}(|V| * |E|^2)$ wird erreicht, indem in jedem Berechnungsschritt jeweils der kürzeste augmentierende Pfad gewählt wird, welcher üblicherweise durch eine Breitensuche ermittelt wird.

Algorithmus 2 Algorithmus von Edmonds und Karp

Input Ein Netzwerk (G, u, s, t) mit $u : E(G) \to \mathbb{Z}_+$
Output Ein s-t-Fluss mit maximalem Wert

1: Setze $f(e) := 0$ für alle $e \in E(G)$
2: **while** Ein s-t-Weg in G_f existiert **do**
3: Bestimme einen kürzesten s-t-Weg P in G_f
4: Bestimme $\gamma := min_{e \in E(P)} u_f(e)$
5: Augmentiere f entlang P um γ
6: **end while**

Beispiel 4.1. Da in dieser Arbeit eine abgewandelte Version des Algorithmus von Edmonds und Karp angewandt wird, soll dessen Funktionsweise anhand des in Abbildung 4.1 dargestellten Graphen demonstriert werden. Dieser Graph wurde speziell konstruiert um später die Notwendigkeit der in Abschnitt 4.3 eingeführten Zusatzbedingungen zu veranschaulichen.

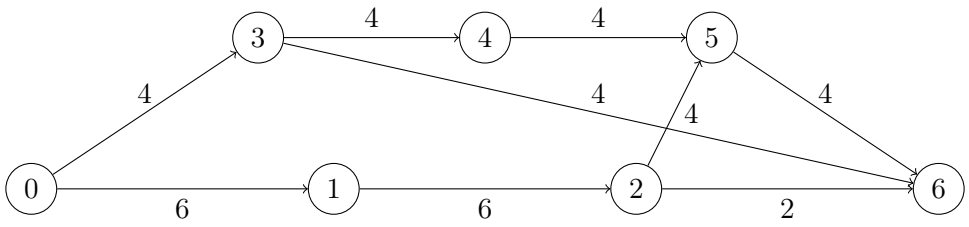


Abbildung 4.1: Graph für Beispiel 4.1

Zuerst werden, wie im Algorithmus beschrieben, alle Flüsse auf 0 gesetzt. Anschließend wird solange ein kürzester s-t-Weg gesucht, bis es keinen solchen mehr gibt. Auf diese Weise wird als erstes der Weg 0-3-6 gefunden, welcher um $\gamma = 4$ augmentiert wird. Der resultierende Graph sieht wie folgt aus (Für die folgenden Graphen wird für jede Kante deren Fuss, sowie in Klammern deren Kapazität angegeben):

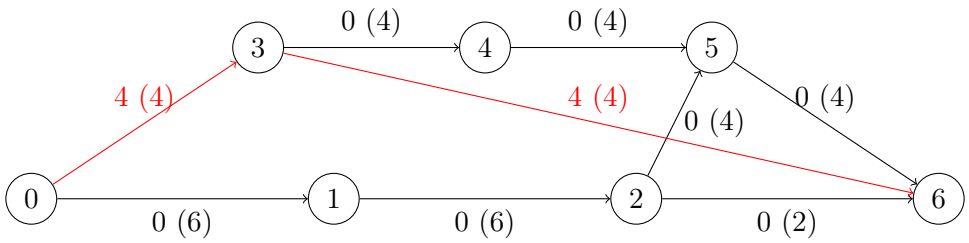


Abbildung 4.2: Graph aus Abbildung 4.1, nachdem entlang des Weges 0-3-6 augmentiert wurde.

Im nächsten Schritt wird der kürzeste Weg 0-1-2-6 gefunden und um $\gamma = 2$ augmentiert. Es wird in diesem Beispiel bewusst auf die Darstellung des Residualgraphen verzichtet, da bei der Bestimmung des zu augmentierenden Pfades nie ein Pfad gewählt wird, welcher die im Residualgraphen zusätzlich enthaltenen Kanten beinhaltet.

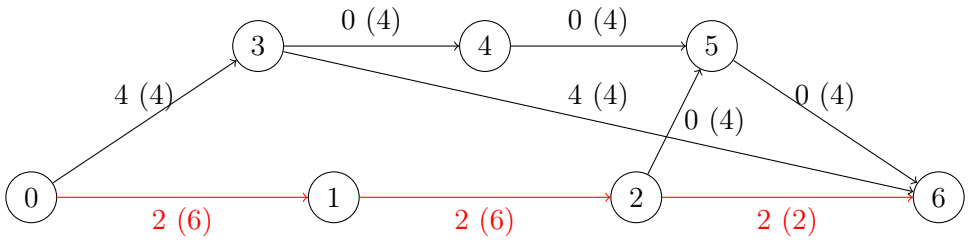


Abbildung 4.3: Graph aus Abbildung 4.2, nachdem entlang des Weges 0-1-2-6 augmentiert wurde.

Zu guter Letzt wird noch entlang des Weges 0-1-2-5-6 um $\gamma = 4$ augmentiert. Der daraus resultierende Graph enthält keine weiteren augmentierbaren Pfade mehr und zeigt daher den maximalen Fluss zwischen der Quelle (Knoten 0) und der Senke (Knoten 6).

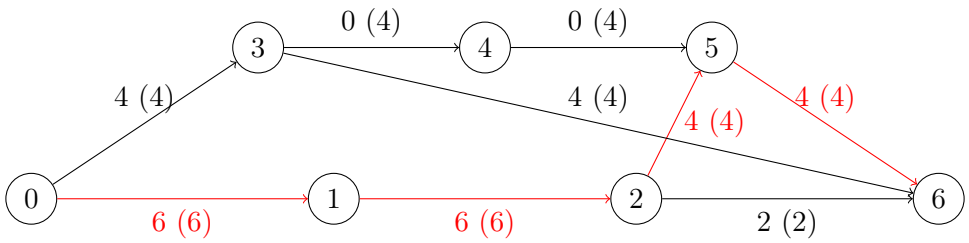


Abbildung 4.4: Graph aus Abbildung 4.3, nachdem entlang des Weges 0-1-2-5-6 augmentiert wurde.

Der maximale Fluss von der Quelle zur Senke, welcher der Summe der Flüsse über die aus der Quelle ausgehende Kanten entspricht (oder gleichermaßen der Summe der Flüsse über die in die Senke eingehenden Kanten), beträgt in diesem Fall also 10 Einheiten.

4.2.2 Algorithmus von Dinic

In etwa zur selben Zeit, in welcher Edmonds und Karp ihren Algorithmus zur Berechnung des maximalen Flusses veröffentlichten, fand E.A. Dinic [6] unabhängig davon eine Lösung, welche über eine noch bessere Laufzeit $\mathcal{O}(|V|^2 * |E|)$ verfügt.

Zur Beschreibung dieses Algorithmus benötigen wir als erstes die Definition des **Level-Graphen**.

Definition 4.3. Gegeben sei ein Netzwerk (G, u, s, t) und ein s-t-Fluss f. Der Level-Graph G_f^L von G_f ist der gerichtete Graph

$$(V(G), \{e = (x, y) \in E(G_f) : dist_{G_f}(s, x) + 1 = dist_{G_f}(s, y)\}).$$

Dieser azyklische, gerichtete Graph kann leicht mittels Breitensuche in $\mathcal{O}(|E|)$ erstellt werden. Die s-t-Wege im Level-Graphen sind genau die kürzesten s-t-Wege in G_f. Nun benötigt man folgende Definition eines **blockierenden Flusses**.

Definition 4.4. Ein s-t-Fluss f in einem gegebenen Netzwerk (G, u, s, t) heißt blockierend, wenn der Graph $(V(G), \{e \in E(G) : f(e) < u(e)\})$ keine s-t-Wege enthält.

Aus den oben gebrachten Darstellungen ergibt sich nun wie folgt der Algorithmus von Dinic:

Algorithmus 3 Algorithmus von Dinic

Input Ein Netzwerk (G, u, s, t) mit $u : E(G) \to \mathbb{Z}_+$
Output Ein s-t-Fluss mit maximalem Wert

1: Setze $f(e) := 0$ für alle $e \in E(G)$
2: Erstelle den Level-Graph G_f^L von G_f
3: Bestimme einen blockierenden s-t-Fluss in f' in (G_f^L, u_f).
4: **if** $f' = 0$ **then**
5: **stop**
6: **else**
7: Augmentiere f um f' und gehe zu 2
8: **end if**

Das Augmentieren von f um f' bedeuted, dass wir $f(e)$ um $f'(e)$ für jedes $e \in E(G_f^L) \cap E(G)$ erhöhen und $f(e)$ um $f'(e)$ für jedes $e \in E(G)$ mit $\overleftarrow{e} \in E(G_f^L)$ verringern.

4.3 Problemspezifische Anpassungen

In dem in dieser Arbeit behandelten Problem möchte man zwar ebenfalls den maximalen Fluss berechnen, allerdings unter der Vorraussetzung, dass jede Station dabei ihre Mindestleistung erbringt. Daher ist es notwendig, bei der Berechnung des Flusses ein paar problemspezifische Anpassungen zu treffen. Da in dieser Arbeit eine abgewandelte Version des Algorithmus von Edmonds und Karp implementiert wurde, beziehen sich auch die folgenden Ausführungen ausschließlich auf diesen. Folgende vier Punkte müssen bei dieser abgewandelten Version des Algorithmus berücksichtigt werden.

- **Bedarfsgesteuerte Pfadauswahl:** Gibt es bei der Ermittlung des zu augmentierenden Pfades noch Pfade, welche mindestens einen Knoten enthalten, dessen ausgehender Fluss geringer ist als die auf diesem Knoten zu erbringende Mindestleistung, so wird unter diesen ein kürzester Pfad gewählt. Erst wenn es keine solchen Pfade mehr gibt, wird unter allen möglichen Pfaden von der Quelle zur Senke ein kürzester gewählt. Durch diese Art der Pfadauswahl soll bewirkt werden, dass durch die Erhöhung des Flusses zuerst sämtliche Mindestleistungen der einzelnen Stationen erfüllt werden. Sobald dies der Fall ist, werden die Pfade wieder so ausgewählt wie in Kapitel 4.2.1 beschrieben. Dies kann, im Vergleich zum originalen Algorithmus von Edmond und Karp, zu einer etwas schlechteren Laufzeit führen, beeinträchtigt aber nicht den Wert des gefundenen maximalen Flusses.

- **Bedarfssäquivalente Augmentierung:** Wurde ein zu augmentierender Pfad P ausgewählt und sei $V_P \leq V$ die Menge aller Knoten entlang dieses Pfades, so muss

für jeden Knoten $v \in V_P$ die Differenz aus ausgehendem Fluss und Mindestleistung L_v ermittelt werden:

$$\Delta_{l,v} = (\sum_{e \in \delta_v^+} f(e)) - L_v$$

Hier bezeichnet δ_v^+ die Menge aller aus v hinausführenden Kanten. Sei nun $V_{P_{neg}}$ die Menge aller Knoten entlang des ausgewählten Pfades, deren ausgehender Fluss geringer ist als die für diesen Knoten erforderliche Mindestleistung:

$$V_{P_{neg}} := \{v \in V_P | (\sum_{e \in \delta_v^+} f(e)) - L_v < 0\}$$

Ist in $V_{P_{neg}}$ mindestens ein Knoten enthalten ($|V_{P_{neg}}| > 0$), so wird der Betrag um den entlang des Pfades augmentiert werden soll, wie folgt ermittelt:

$$\gamma := \min\ [\min_{e \in E(P)} u_f(e),\ \min_{v \in V_{P_{neg}}} L_v - (\sum_{e \in \delta_v^+} f(e))]$$

Befindet sich kein Knoten in $V_{P_{neg}}$ ($|V_{P_{neg}}| = 0$), so wird γ, wie schon im Algorithmus von Edmonds und Karp beschrieben, berechnet als:

$$\gamma := min_{e \in E(P)} u_f(e)$$

Um den Effekt dieser Regel zu demonstrieren, betrachten wir die in Beispiel 4.2 dargestellte Situation.

Beispiel 4.2. Gegeben ist ein einfacher Graph mit 3 Knoten (1,2,3), wobei auf den Knoten 1 und 2 je eine Mindestleistung von 2 Einheiten erzielt werden muss. Bei Knoten 0/Knoten 4 handelt es sich um die Quelle/Senke. Für jede Kante ist außerdem deren Kapazität angegeben. Die zu erbringende Mindestleistung an jedem Knoten ist in eckigen Klammern angegeben.

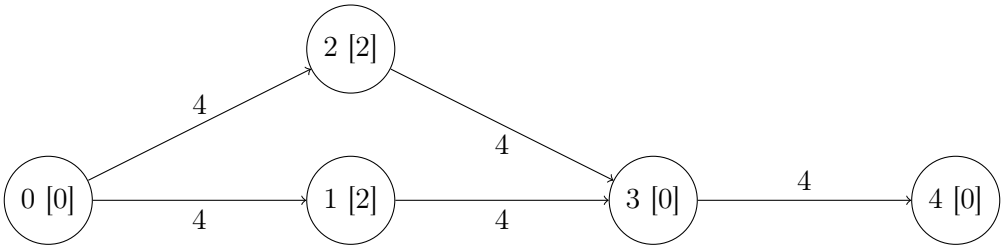


Abbildung 4.5: Graph für Beispiel 4.2

Es gibt in diesem Beispiel 2 Wege von der Quelle zur Senke, nämlich zum einen den Weg 0-1-3-4 und zum anderen den Weg 0-2-3-4. Da beide gleich lang sind

und beide einen Knoten beinhalten, dessen Mindestbedarf noch nicht erfüllt ist, wird also zufällig einer der beiden ausgewählt. Angenommen der Weg 0-1-3-4 wird ausgewählt, so kann man leicht erkennen, dass dieser maximal um einen Betrag $\gamma = 4$ augmentiert werden kann. Allerdings hat dies zur Folge, dass es anschließend nicht mehr möglich ist, den Fluss entlang des Weges 0-2-3-4 zu erhöhen, da die Kapazität der Kante zwischen Knoten 3 und Knoten 4 bereits erschöpft ist. Somit wird die auf Knoten 2 zu erbringende Mindestleistung nicht erreicht. Durch die bedarfsäquivalente Augmentierung wird aber entlang des Weges 0-1-3-4 nur um den Betrag $\gamma = 2$ augmentiert, wodurch in der nächsten Iteration der Weg 0-2-3-4 ebenfalls noch um $\gamma = 2$ augmentiert werden kann. In beiden Fällen ergibt sich ein maximaler Fluss von 4 Einheiten, allerdings werden im ersten Fall nicht alle Mindestleistungen erfüllt, wohingegen sie im zweiten Fall schon erfüllt werden. Hier soll noch angemerkt werden, dass in zweiterem Fall eine Iteration mehr nötig war, da im ersten Schritt nicht um den maximal möglichen Betrag augmentiert wurde.

Wie auch die bedarfsgesteuerte Pfadauswahl hat also auch die bedarfsäquivalente Augmentierung keine Auswirkung auf den Wert des maximalen Flusses, kann sich aber auf die Laufzeit des Algorithmus geringfügig negativ auswirken.

- **Bedarfsblockierte Augmentierung:** Diese Regel könnte man grundsätzlich mit der bedarfsäquivalenten Augmentierung zusammenfassen, allerdings wird sie aus Gründen der Nachvollziehbarkeit als selbstständige Regel angeführt. Bei der Ermittlung des Betrages γ, um welchen ein ausgewählter Pfad augmentiert werden soll, müssen folgende Punkte beachtet werden:

  - Für alle Knoten v mit $\Delta_{l,v} >= 0$, darf $\sum_{e \in \delta_v^+} f(e)$ maximal um $\Delta_{l,v}$ abnehmen. Dadurch soll sichergestellt werden, dass der ausgehende Fluss eines Knotens, dessen Mindestleistung bereits erfüllt wird, durch das Augmentieren nicht unter eben diese Mindestleistung sinken kann.

  - Für alle Knoten v mit $\Delta_{l,v} < 0$, darf $\sum_{e \in \delta_v^+} f(e)$ gar nicht abnehmen. Hierdurch wird sichergestellt, dass der ausgehende Fluss eines Knotens, dessen Mindestleistung noch nicht erfüllt wird, durch das Augmentieren nicht sinken kann.

Die bedarfsgesteuerte Pfadauswahl sowie die bedarfsäquivalente Augmentierung bewirken, durch gezielte Pfadauswahl und einer speziellen Berechnung des zu augmentierenden Betrags γ, dass der Fluss durch den Graphen zunächst so erhöht wird, dass die benötigte Mindestleistung auf den einzelnen Stationen erfüllt wird. Durch diese zusätzliche Regel wird nun sichergestellt, dass der ausgehende Fluss eines Knotens nicht durch das Augmentieren eines Pfades unter die erforderliche Mindestleistung zurückfällt bzw. dass der ausgehende Fluss eines Knotens, welcher noch nicht seine Mindestleistung erfüllt, nicht noch weiter sinkt.

Nun kann allerdings noch das Problem auftreten, dass ein Pfad ausgewählt wird, welcher aufgrund der bedarfsblockierten Augmentierung nur um den Betrag $\gamma = 0$ augmentiert werden kann. Dies würde am Graphen klarerweise keine Änderung hervorrufen. Dadurch würde aber auch im nächsten Schritt wieder der gleiche Pfad ausgewählt werden und dementsprechend würde der Algorithmus nie terminieren. Um dies zu vermeiden, müssen bei der Pfadauswahl solche Pfade vermieden werden. Dies führt zu der letzten Modifikation des Algorithmus.

Durch diese Regel kann es vorkommen, dass der Algorithmus einen geringeren maximalen Fluss errechnet, als der unmodifizierte Algorithmus von Edmonds und Karp. Dieser Effekt wird aber bewusst in Kauf genommen, da das Erfüllen der Mindestleistungen Priorität hat.

- **Bedarfsblockierte Pfadauswahl:** Pfade, welche aufgrund der bedarfsblockierten Augmentierung nur um den Betrag $\gamma = 0$ augmentiert werden dürften, gelten bei der Pfadauswahl als ungültig und dürfen daher nicht gewählt werden. Gibt es aufgrund dieser Regel keine gültigen Pfade mehr, so ist der Algorithmus zu Ende.

Beispiel 4.3. Um das Verhalten des modifizierten Algorithmus von Edmonds und Karp zu veranschaulichen, verwenden wir hier noch einmal den Graphen aus Beispiel 4.1. Zusätzlich ist für jeden Knoten in eckigen Klammern eine Mindestleistung angegeben, welche von diesem erfüllt werden muss. In diesem Fall ist diese für alle Knoten 0, außer für Knoten 4, welcher eine Mindestleistung von 2 Einheiten erbringen muss.

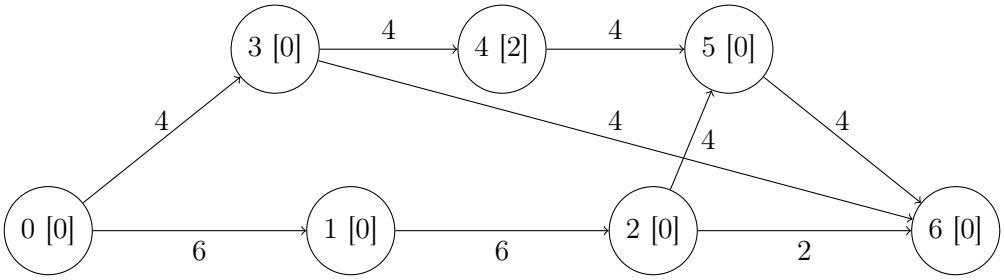


Abbildung 4.6: Graph aus Beispiel 4.1

Zu Beginn werden wieder alle Flüsse auf 0 gesetzt. Aufgrund der zu erbringenden Mindestleistung von 2 Einheiten auf Knoten 4, wird nun aber nicht der kürzeste s-t-Weg (0-3-6) gewählt, sondern, aufgrund der bedarfsgesteuerten Pfadauswahl, der Weg 0-3-4-5-6. Nun wird anhand der bedarfsorientierten Augmentierung der Betrag $\gamma = 2$ gewählt und der Pfad entsprechend augmentiert. Die zusätzlichen Kanten, welche durch das Augmentieren des Pfades entstehen, werden im Graphen blau dargestellt.

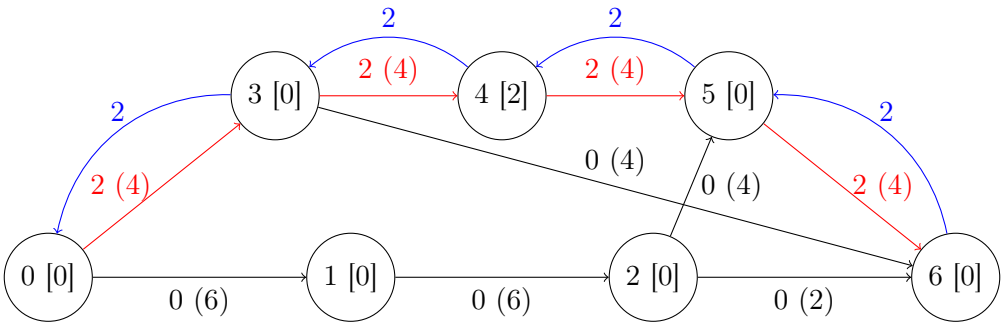


Abbildung 4.7: Graph aus Abbildung 4.6, nachdem entlang des Weges 0-3-4-5-6 augmentiert wurde.

Nun exisitieren im Graphen keine Knoten mehr, deren Mindestbedarf nicht erfüllt ist, daher wird im nächsten Schritt der kürzeste Weg 0-3-6 gewählt und um $\gamma = 2$ augmentiert.

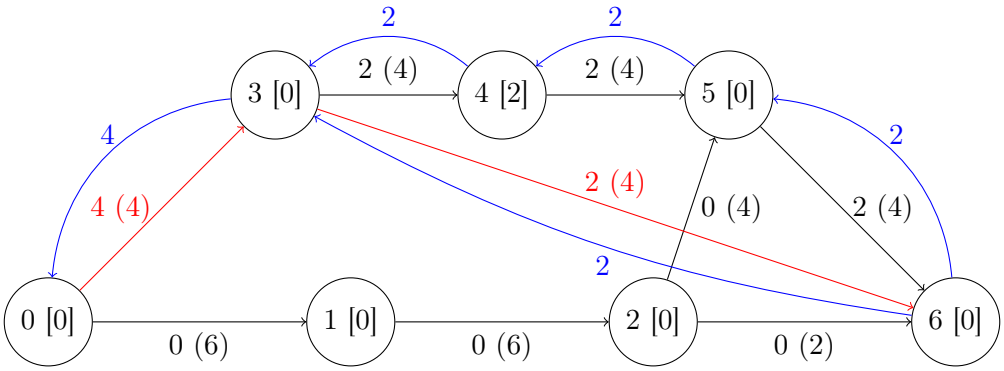


Abbildung 4.8: Graph aus Abbildung 4.7, nachdem entlang des Weges 0-3-6 augmentiert wurde.

Als nächster wird der Weg 0-1-2-6 gewählt und um $\gamma = 2$ augmentiert.

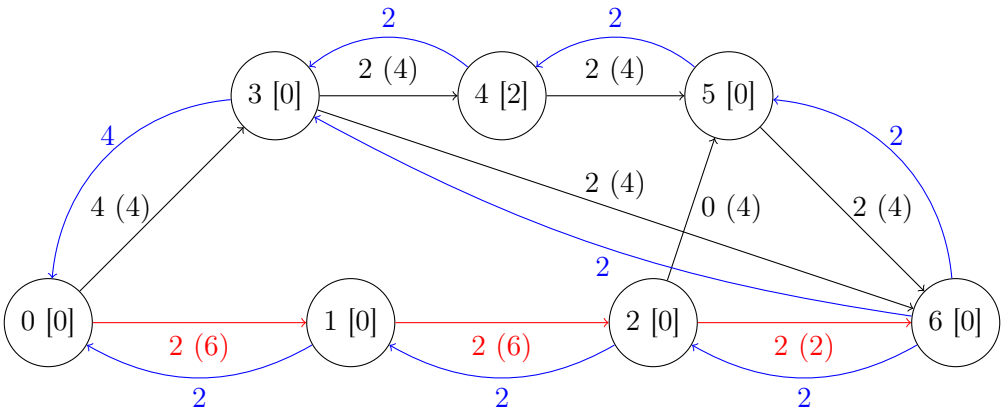


Abbildung 4.9: Graph aus Abbildung 4.8, nachdem entlang des Weges 0-1-2-6 augmentiert wurde.

Zu guter Letzt wird noch der Weg 0-1-2-5-6 gewählt und um $\gamma = 2$ augmentiert.

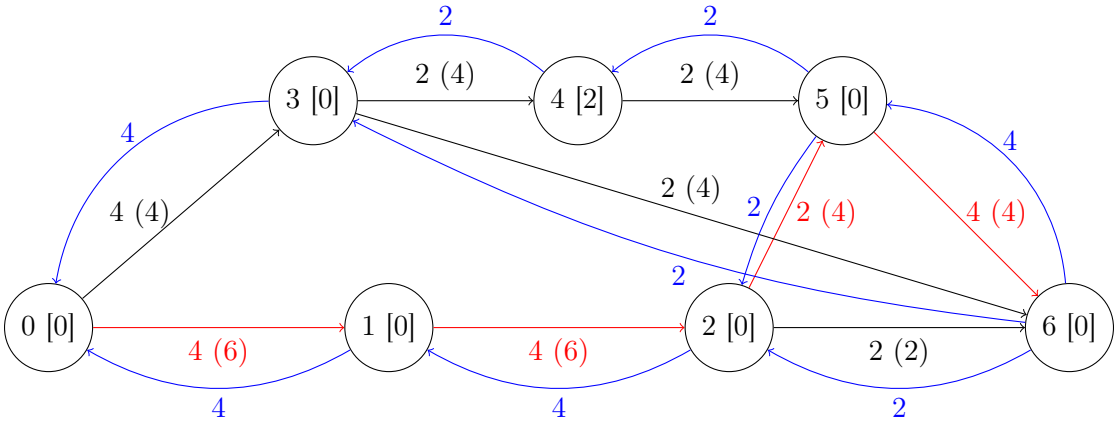


Abbildung 4.10: Graph aus Abbildung 4.9, nachdem entlang des Weges 0-1-2-5-6 augmentiert wurde.

Betrachtet man den Graphen aus Abbildung 4.10, so findet man noch einen Pfad von der Quelle zur Senke, nämlich 0-1-2-5-4-3-6. Würde man diesen um irgendeinen Betrag $\gamma > 0$ augmentieren, so könnte man den Fluss durch den Graphen noch steigern. Allerdings verbietet die bedarfsblockierte Pfadauswahl die Auswahl dieses Pfades, da dadurch die auf Station 4 erbrachte Leistung sinken würde und somit die erforderliche Mindestleistung von 2 Einheiten nicht erfüllt werden würde. Der durch die modifizierte Version des Algorithmus von Edmonds und Karp errechnete maximale Fluss beträgt also nur 8

Einheiten, während die unmodifizierte Version (siehe Beispiel 4.1) einen maximalen Fluss von 10 Einheiten lieferte.

5 Heuristiken

Im folgenden Abschnitt wird eine Auswahl an Heuristiken vorgestellt, durch welche man mit geringem Rechenaufwand zu guten Lösungen für das betrachtete Problem kommen soll. In den in Kapitel 6 beschriebenen genetischen Algorithmen werden diese Lösungen dann eingesetzt, um sicherzustellen, dass in der ersten Generation an Individuen bereits ein gewisses Maß an Qualität vorhanden ist.

Die hier betrachteten Heuristiken folgen grundsätzlich alle dem selben Schema. Man startet mit einer initialen Zuteilungsmatrix, deren Elemente allesamt 0 sind, womit also keiner Station auch nur eine einzige Ressource zugeteilt ist. Nun wird, unter Verwendung bestimmter Regeln, eine Station ausgewählt. Anschließend wird unter denjenigen Ressourcen, welche auf dieser Station eingesetzt werden können und nicht bereits einer anderen Station zugeteilt wurden, wiederum anhand verschiedener Regeln, eine passende Ressource ausgewählt und eben dieser Station zugeteilt, wobei die Zuteilungsmatrix entsprechend angepasst wird. Die einzelnen Verfahren unterscheiden sich also durch die Art und Weise in welcher die nächste Station ausgewählt wird, sowie die Auswahl der zuzuweisenden Ressource. Für manche der Verfahren (z.B. MKA) ist es zudem notwendig, auf Basis der Zuteilungsmatrix und des gegebenen Systems, in jeder Iteration zunächst einen Graphen zu konstruieren, wie in Kapitel 3 beschrieben. Diese werden weiterfolgend **graphenbasiert** genannt.

In den folgenden Abschnitten werden verschiedene Regeln zur Auswahl der Stationen sowie zur Auswahl der Ressourcen vorgestellt. Sollte es unter Einsatz einer dieser Regeln zu einer Situation kommen, in welcher keine eindeutige Entscheidung getroffen werden kann, so bietet es sich an, diese Situation durch Anwendung einer weiteren Regel aufzulösen. Dies kann natürlich beliebig lang so fortgeführt werden und konnte nach Anwendung sämtlicher Regeln immer noch keine eindeutige Entscheidung getroffen werden, so kann man guten Gewissens den Zufall entscheiden lassen.

5.1 Selektionsverfahren für Stationen

Bevor hier die einzelnen Verfahren betrachtet werden, sollen noch folgende Begriffe eingeführt werden. Sei S die Menge aller Stationen. Für jede Station $s \in S$ sei x_s die Anzahl

der Ressourcen, die dieser Station zugewiesen sind. Dann bezeichne $S_{unterbesetzt}$ die Teilmenge von S für die gilt, dass für jede Station $s \in S_{unterbesetzt}$, x_s kleiner ist als die vorgeschriebene Mindestanzahl $R_{min,s}$. Des weiteren bezeichne $S_{zulaessig}$ die Menge aller Stationen für die gilt, dass $R_{min,s} <= x_s < R_{max,s}$. Also all jene Stationen, welche schon ausreichend besetzt sind, aber immer noch weitere Ressourcen aufnehmen könnten. Zu guter letzt sei $S_{unzulaessig}$ die Menge Teilmenge von S, für die für jede in ihr enthaltene Station s gilt, dass $x_s = R_{max,s}$.

Für alle folgenden Verfahren gilt nun folgendes: Ist $|S_{unterbesetzt}| > 0$, dann müssen beim Auswählen der nächsten Station immer die Stationen aus $S_{unterbesetzt}$ bevorzugt werden. Nur wenn keine Station mehr in $S_{unterbesetzt}$ enthalten ist, können auch Stationen aus $S_{zulaessig}$ gewählt werden. Stationen aus $S_{unzulaessig}$ können offensichtlicherweise nie ausgewählt werden. Sollte es für eine Station keine Ressource mehr geben, welche auf dieser eingesetzt werden kann, so gehört diese Station automatisch zu $S_{unzulaessig}$.

- **Geringste Ressourcenauswahl (GRW):** Wähle jene Station, bei der die Anzahl zuteilbarer Ressourcen am geringsten ist.

- **Maximale erzielbare Leistung (MEZ):** Wähle diejenige Station aus, für die die höchste auf ihr von einer noch verfügbaren Ressource erzielbare Leistung maximal ist. Es eignet sich in diesem Fall, die zuzuweisende Ressource nach dem MAS-Prinzip (siehe Abschnitt 5.2) zu wählen.

- **Maximale Kapazitätsausnutzung (MKA):** Wähle diejenige Station, bei der die Differenz der Summe der Leistungen der ihr zugewießenen Ressourcen und des ausgehenden Flusses am geringsten ist. Dies zählt zu den graphenbasierten Verfahren.

- **Erfüllter Mindestbedarf (EMB):** Wähle diejenige Station, bei der das Verhältnis aus zu erbringender Mindestleistung und der Summe der Leistungen der ihr zugewießenen Ressourcen minimal ist (Werte > 1 sind als 1 zu werten). Dies eignet sich weniger als primäres Kriterium zur Stationsauswahl, sondern dient hauptsächlich als guter Tie-Breaker.

5.2 Selektionsverfahren für Ressourcen

- **Geringste Kompetenzdiversität (GKD):** Wähle unter allen zulässigen Ressourcen jene aus, bei welcher die Anzahl der Stationen der sie zugewiesen werden kann, minimal ist.

- **Maximale absolute Stationsleistung (MAS):** Wähle unter allen zulässigen

Ressourcen jene aus, welche auf der ausgewählten Station die höchste Leistung erzielt.

- **Maximale relative Stationsleistung (MRS):** Wähle unter allen zulässigen Ressourcen jene aus, bei welcher das Verhältnis der Leistung, welche sie auf der ausgewählten Station erbringen kann und der maximalen Leistung dieser Ressource auf allen in $S_{unterbesetzt}$ oder $S_{zulaessig}$ enthaltenen Stationen am höchsten ist.

Um die vorangegangenen Ausführungen anhand eines Beispiels zu erleutern, sei das folgende System gegeben. In diesem wird keine Initialbelegung angegeben, da diese in den oben beschriebenen Heuristiken nicht verwendet wird. Aus dem selben Grund wird auch auf die Angabe einer Umschichtungsmatrix verzichtet.

Beispiel 5.1. Ein Testsystem mit $m = 3$ Stationen und $n = 4$ Ressourcen zur veranschaulichung der folgenden Heuristiken.

Tabelle 5.1: Leistungen $l_{s,r}$ für 5.1

	r_1	r_2	r_3	r_4
s_1	10	0	8	12
s_2	15	20	15	25
s_3	12	16	20	20

Tabelle 5.2: Stationseigenschaften für 5.1

	b_s	t_s	L_s	$R_{min,s}$	$R_{max,s}$
s_1	80	8	10	0	2
s_2	80	8	10	0	1
s_3	120	10	12	1	4

Tabelle 5.3: Verteilungs-, Puffer- und Pufferkapazitätsmatrix für Bsp. 5.1

	s_1			s_2			s_3		
	V	P	K	V	P	K	V	P	K
s_1	0	0	0	1	32	40	1	40	80
s_2	0	0	0	0	0	0	0	0	0
s_3	0	0	0	0	0	0	0	0	0

Zu Beginn müssen die Mengen $S_{unterbesetzt}$, $S_{zulaessig}$ und $S_{unzulaessig}$ bestimmt werden. Da noch keiner Station eine Ressource zugeteilt wurde, gehören sämtliche Stationen $s \in S$ mit $R_{min,s} > 0$ automatisch zu $S_{unterbesetzt}$. Alle verbleibenden Stationen $s \in S \setminus S_{unterbesetzt}$ gehören zu $S_{zulaessig}$. Somit ergeben sich folgende Mengen:

- $S_{unterbesetzt} = \{s_3\}$
- $S_{zulaessig} = \{s_1, s_2\}$
- $S_{unzulaessig} = \{\}$

Stationen sollen in diesem Beispiel nach dem MKA-Prinzip ausgewählt werden wobei als Tie-Breaker das GRW-Prinzip gewählt wird. Für den Fall, dass sich auch dadurch noch immer keine eindeutige Entscheidung treffen lässt, soll eine solche durch das EMP-Prinzip gefällt werden. Ressourcen werden nach dem MRS-Prinzip bzw. im Falle eines Unentschiedens nach dem MAS-Prinzip ausgewählt.

Da zu Beginn des Verfahrens nur eine einzige Station, nämlich s_3, in $S_{unterbesetzt}$ enthalten ist, wird diese automatisch ausgewählt. Nun muss für jede Ressource r, deren relative Stationsleistung $l_{s,r,rel}$ für Station s_3 wie folgt berechnet werden:

- $l_{3,1,rel} = 12/15 = 0.8$
- $l_{3,2,rel} = 16/20 = 0.8$
- $l_{3,3,rel} = 20/20 = 1$
- $l_{3,4,rel} = 20/25 = 0.8$

Somit wird im ersten Schritt Ressource r_3 Station s_3 zugewießen. Daraus ergibt sich die Zuteilungsmatrix in Tabelle 5.4 , aus welcher sich der in Abbildung 5.1 ersichtliche Graph konstruieren lässt.

Tabelle 5.4: Zuteilungsmatrix für Beispiel 5.1 nach der ersten Iteration

	r_1	r_2	r_3	r_4
s_1	0	0	0	0
s_2	0	0	0	0
s_3	0	0	1	0

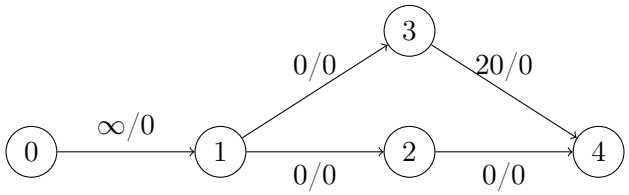


Abbildung 5.1: Graph für Beispiel 5.1 nach der ersten Iteration

Da auf Station s_3 nun eine Ressource zugteilt wurde, ist das Minimum $R_{min,3} = 1$ erfüllt, womit sich alle Stationen in $S_{zulaessig}$ befinden. Somit müseen in der nächsten Iteration, bei der Auswahl der Station, alle Stationen berücksichtigt werden. Da als Hauptkriterium zur Stationsauswahl MKA gewählt wurde, muss als erstes für jede Station die Differenz δ_s

aus erzielbarer Leistung und ausgehendem Fluss ermittelt werden. Aus Abbildung 5.1 ist leicht ersichtlich, dass $\delta_1 = \delta_2 = 0$, sowie $\delta_3 = 20 - 0 = 20$. Somit ergibt sich zwischen den Stationen s_1 und s_2 ein Unentschieden, welches über die Regel GRW aufgelöst werden muss. Da es noch 3 Ressourcen gibt, welche Station s_2 zugeteilt werden können, aber nur 2 welche Station s_1 zugeteilt werden können, wird in diesem Fall letzterer Station der Vorzug gegeben. Nun sind wieder die relativen Stationsleistungen zu ermitteln, was zu folgendem Ergebnis führt:

- $l_{1,1,rel} = 10/15 = 0.\dot{6}$
- $l_{1,4,rel} = 12/25 = 0.48$

Ressource r_1 erzielt also die höhere relative Stationsleistung und wird somit Station s_1 zugeteilt. Daraus ergibt sich wiederum eine neue Zuteilungsmatrix mit einem enstprechend neuen Graphen.

Tabelle 5.5: Zuteilungsmatrix für Beispiel 5.1 nach der zweiten Iteration

	r_1	r_2	r_3	r_4
s_1	1	0	0	0
s_2	0	0	0	0
s_3	0	0	1	0

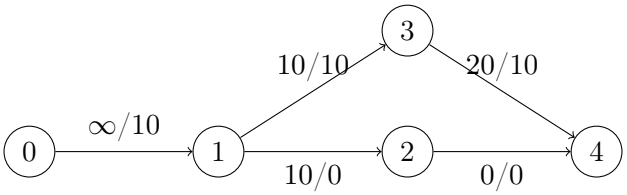


Abbildung 5.2: Graph für Beispiel 5.1 nach der zweiten Iteration

Wie schon in der vorherigen Iteration ergibt sich $\delta_1 = \delta_2 = 0$. δ_3 ist aufgrund des nun auftretenden Flusses um 10 Einheiten gesunken. Es ergibt sich erneut eine Patt-Situation zwischen s_1 und s_2, welcher wieder zu Gunsten von Station s_1 aufgelöst wird, da es für diese nur noch 1 zuteilbare Ressource gibt, während s_2 noch 2 Ressourcen zugeteilt werden könnten. Da für s_1 nur noch Ressource r_4 in Frage kommt, wird diese automatisch augewählt und es ergibt sich folgende Situation:

Tabelle 5.6: Zuteilungsmatrix für Beispiel 5.1 nach der dritten Iteration

	r_1	r_2	r_3	r_4
s_1	1	0	0	1
s_2	0	0	0	0
s_3	0	0	1	0

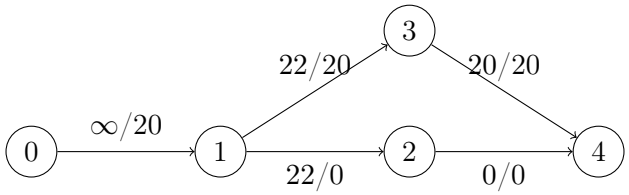


Abbildung 5.3: Graph für Beispiel 5.1 nach der dritten Iteration

$R_{max,1} = 2$ ist somit erreicht, womit Station s_1 in die Menge $S_{unzulaessig}$ verschoben wird. Da $\delta_2 = \delta_3 = 0$ und für beide noch genau 1 Ressource zur Verfügung steht, muss nun nach dem EMP-Prinzip eine Entscheidung getroffen werden. Es muss also für beide Stationen das Verhältnis η_s aus Mindestleistung und erbringbarer Leistung ermittelt werden. Hierbei ergit sich:

- $\eta_2 = min[1, 0/10] = 0$
- $\eta_3 = min[1, 20/12] = 1$

Somit wird die letzte verbleibende Ressource auf Station s_2 zugeteilt und das Verfahren kommt zu einer Lösung mit einer Gesamtleistung von 22 Einheiten, wie in Abbildung 5.4 ersichtlich.

Tabelle 5.7: Zuteilungsmatrix für Beispiel 5.1 nach der letzten Iteration

	r_1	r_2	r_3	r_4
s_1	1	0	0	1
s_2	0	1	0	0
s_3	0	0	1	0

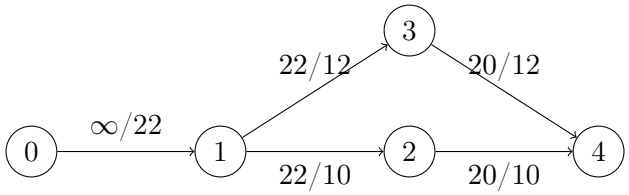


Abbildung 5.4: Graph für Beispiel 5.1 nach der letzten Iteration

6 Genetische Algorithmen

Genetische Algorithmen gehören zu jenen Methaheuristiken, welche sich in ihrer Funktionsweise an Phänomenen orientieren, welche man in der Natur beobachten kann. Die Grundidee dabei ist, eine Menge von Lösungen für ein bestimmtes Problem zu erstellen, welche sich dann, angelehnt an die Evolution, weiterentwickeln soll. Diese Entwicklung findet statt, indem die guten Lösungen miteinander gekreuzt werden, wodurch schlussendlich eine neue, bessere Menge aus Lösungen entstehenden soll. Durch das zufällige Verändern einzelner Lösungen soll außerdem sichergestellt werden, dass ein gewisses Maß an Vielfalt in der Menge der Lösungen enthalten belibt, da der Algorithmus sich ansonsten zu leicht zu einem lokalen Optimum verlaufen könnte. Die folgende grundlegende Einführung in die genetischen Algorithmen bedient sich der Ausführungen von Sivandam [23], Mitchell [18] und Michalewicz [16].

6.1 Einführung

Wie bereits eingangs erwähnt, arbeitet ein genetischer Algorithmus mit einer Menge von Lösungen, welche als **Population** bezeichnet wird. Eine Population besteht aus n **Individuen** $I_1, .., I_n$, welche Lösungen des betrachteten Problems darstellen. Individuen werden stets durch eine geeignete Datenstruktur repräsentiert, wobei deren Komplexität stark vom betrachteten Problem abhängt. Jedem Individuum wird außerdem ein **Fitnesswert** f_i zugeordnet, welcher die Qualität des Individuums repräsentiert und üblicherweiße mit der Zielfunktion des betrachteten Problems zusammenhängt. Durch ein ausgewähltes **Selektionsverfahren** werden dann verschiedene Individuen, meist jene mit einem guten Fitnesswert, bestimmt, welche gemeinsam den sogenannten **Paarungspool** bilden. Hier ist anzumerken, dass ein Individuum durchaus öfters im Paarungspool vorhanden sein kann. Durch geschickte Kombination der Individuen im Paarungspool soll schlussendlich eine neue, bessere Population entstehen. Die dabei auftretenden, verschiedenen Populationen werden auch als **Generationen** bezeichnet.

Die Kombination der Individuen im Paarungspool erfolgt durch ein geeignetes **Kreuzungsverfahren**. Dabei werden meist zwei Individuen, ein Vater- und ein Mutter-Indiviuum (I^V, I^M) gekreuzt, wodurch wiederum zwei neue Individuen, nämlich ein Sohn- und ein Tochter-Individuum (I^S, I^T) entstehen. Hier sei erwähnt, dass für das Kreuzen nicht zwingend nur zwei Individuen verwendet werden dürfen. Es gibt in der

Literatur durchaus auch Beispiele, bei denen Kind-Individuen durch das Kreuzen von mehr als zwei Eltern entstehen, wobei natürlich argumentiert werden kann, dass dieses Verhalten in der Natur eher unüblich ist. Anschließend an das Kreuzen können zufällig ausgewählte Kind-Individuen noch **mutiert** werden. Hierbei werden diese zufällig verändert, wodurch sichergestellt werden soll, dass nicht ein gewisser Teil des Lösungsraumes unerreichbar wird. Da die Individuen jeder Generation stets gültige Lösungen für das betrachtete Problem sein müssen, muss entweder durch entsprechende Kreuzungs- und Mutationsverfahren die Gültigkeit der Individuen gewährleistet werden, oder ein entsprechender Reperaturalgorithmus fehlerhafte Individuen überarbeiten.

Die Anzahl der Individuen in jeder Generation wird als **Populationsgröße** bezeichnet und kann je nach Problem variieren. Die Individuen der ersten Generation werden meist zufällig erzeugt, wobei es sich manchmal aber auch anbietet, ein paar dieser Individuen durch verschiedene Heuristiken zu erzeugen, da dadurch garantiert wird, dass die erste Generation bereits über zumindest ein paar gute Individuen verfügt. Die Anzahl der Generationen die erzeugt werden soll unterscheidet sich wieder je nach Problemstellung. In der Literatur werden Dauern von 50 bis 500 Generationen als üblich beschrieben [18]. Algorithmus 4 stellt grob den Ablauf eines genetischen Algorithmus dar.

Algorithmus 4 Genetischer Algorithmus

```
1: procedure GENETISCHER ALGORITHMUS
2:     Erzeuge eine Anfangspopulation population mit n Individuen
3:     for i = 1; i ≤ AnzahlGenerationen;i++ do
4:         Berechne einen Fitnesswert f_I für jedes Individuum I in population
5:         Selektiere n Individuen aus population für den paarungspool
6:         wiederhole
7:             Erzeuge neue Individuen durch das Kreuzen der selektierten Individuen
8:             Füge die Kindindividuen der neuen population population_neu hinzu
9:         bis population_neu über n Individuen verfügt
10:        Führe zufällige Mutationen durch
11:        population = population_neu
12:    end for
13: end procedure
```

Die folgenden Abschnitte beschäftigen sich damit, wie das in dieser Arbeit betrachtete Problem mithilfe eines genetischen Algorithmus gelöst werden kann. Es werden unterschiedliche Möglichkeiten zum Erstellen der Anfangspopulation erläutert, sowie verschiedene Verfahren zur Selektion, Kreuzung und Mutation von Individuen vorgestellt. Auch auf die Auswahl der Populationsgröße sowie der Generationenanzahl wird eingegangen. Die Berechnung der Fitnesswerte erfolgt wie in Kapitel 4 beschrieben.

6.2 Repräsentation der Individuen

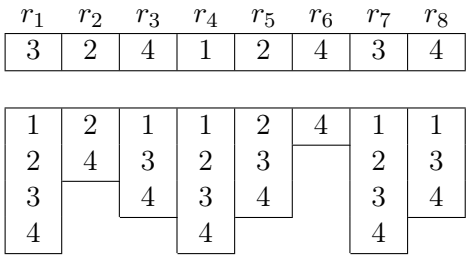
Um die folgenden Ausführungen etwas verständlicher zu machen, sollen hier zuerst einige weitere Begriffe eingeführt werden. Da Genetische Algorithmen aus der Betrachtung biologischer Systeme entstanden sind, wurden viele Begriffe aus diesem Gebiet übernommen.

Alle lebenden Organismen bestehen aus Zellen, von denen jede wiederum aus ein oder mehreren **Chromosomen** besteht. Analog werden die Zeichenketten, mit denen die Individuen eines Genetischen Algorithmus beschrieben werden, ebenfalls Chromosome genannt. Die in dieser Arbeit auftretenden Individuen werden stets durch genau eine Zeichenkette repräsentiert, weshalb der Begriff des Individuums und der des Chromosoms hier also das selbe bezeichnen. Die einzelnen Bestandteile eines Chromosoms werden wiederum als **Gene** bezeichnet. Diese entsprechen den einzelnen Zeichen des Chromosoms, oder Gruppen von Zeichen, welche gemeinsam ein bestimmtes Element des Chromosoms darstellen. Die Menge der erlaubten Zeichen, welche ein solches Element annehmen kann, wird als **Allele** bezeichnet. Im Falle einer binären Zeichenkette wären die Allelen zum Beispiel jeweils $\{0, 1\}$. Die Gesamtheit der Chromosomen eines Organismus wird als dessen **Genotyp** bezeichnet. In der technischen Umsetzung spricht man üblicherweise von der **Struktur** eines Individuums. Zwei Individuen werden also als ident behandelt, wenn sie über die selbe Struktur verfügen.

Nun gilt es also, für das in dieser Arbeit beschriebene Problem eine geeignete Datenstruktur zu finden um die Individuen darzustellen. Dies lässt sich recht einfach bewerkstelligen, indem man ein Chromosom derart konstruiert, dass jedes darin enthaltene Gen die Zuweisung einer Ressource zu einer Station darstellt. Die Allele eines jeden Gens wäre somit die Menge aller Stationen, auf denen die entsprechende Ressource eingesetzt werden kann.

Betrachtet man nun zum Beispiel das in Beispiel 3.4 angeführte Problem, so würde das Individuum für Lösung 1 wie in Tabelle 6.1 aussehen. Des weiteren ist für jedes Gen auch noch dessen Allele angeführt.

Abbildung 6.1: Beispiel eines Individuums inkl. Allelen



Zu guter Letzt soll hier noch der Begriff des Schemas eingeführt werden. Ein **Schema** ist eine Vorlage, welche bestimmten Genen innerhalb des Chromosoms einen fixen Wert vorgibt. In dem Chromosom aus Beispiel 6.1 sind zum Beispiel, unter vielen Anderen, auch folgende beiden Schemata enthalten:

$$S_1: \quad 3 \; * \; * \; * \; * \; * \; 3 \; *$$

$$S_2: \quad * \; * \; 4 \; 1 \; 2 \; 4 \; * \; *$$

Ein $*$ steht dabei für einen beliebigen Wert aus der Allele des Gens. Man kann hier leicht erkennen, dass nicht alle Schemata über die selbe Aussagekraft verfügen. So haben in Schema S_1 nur zwei Gene einen festgelegten Wert, während in Schema S_2 für vier Gene ein Wert vorgegeben ist. Die Anzahl der Gene, für welche ein Schema einen fixen Wert vorgibt, wird als **Ordnung** $o(S)$ des Schemas bezeichnet.

Neben der Ordnung gibt es noch einen weiteren Kennwert um die Aussagekraft eines Schemas zu quantifizieren, nämlich die sogenannte **definierende Länge** $\delta(S)$. Diese gibt den Abstand zwischen dem ersten fixen Gen eines Schemas und dessen letztem fixen Gen. Ein Schema mit nur einem einzigen fixen Gen hätte also eine definierende Länge von 0. Betrachtet man noch einmal die beiden Schemata von vorhin, so hat Schema S_1 mit $o(S_1) = 2$ zwar eine geringere Ordnung als Schema S_2 mit $o(S_2) = 4$, verfügt aber mit $\delta(S_1) = 6$ über die größere definierende Länge.

6.3 Die erste Generation

Die Erzeugung der ersten Generation von Individuen ist ein wichtiger Schritt zur erfolgreichen Implementierung eines genetischen Algorithmus. Burke, Gustafson und Kendall haben gezeigt, dass sich eine gut ausgewählte erste Generation im Allgemeinen positiv auf die Qualität der gefundenen Lösung auswirkt [3]. Besonders wichtig ist es dabei, ausreichend Vielfalt in der Population zu haben, da es sonst passieren kann, dass der Algorithmus sich zu leicht in einem lokalen Optimum verfängt. Hat man für das betrachtete Problem schon geeignete Vorkenntnisse, wie zum Beispiel gültige gute Lösungen oder Teillösungen, so bietet es sich an, diese Lösungen in die Anfangspopulation miteinzubauen, um in dieser ein gewisses Mindestmaß an Qualität zu garantieren.

6.3.1 Zufällig erzeugte Individuen

Die zufällige Erzeugung eines Individuums fällt bei der vorliegenden Struktur grundsätzlich recht einfach aus. Zu Beginn muss erst einmal ein leeres Chromosom erzeugt werden, dessen Anzahl an Genen durch die Anzahl der im Problem vorliegenden Ressourcen gegeben ist. Für jedes Gen wird anschließend dessen Allele bestimmt und daraus ein Element zufällig gewählt. So erhält man schon ein vollständiges Chromosom. Nun kann es allerdings sein, dass dieses die durch $R_{min,s}$ und $R_{max,s}$ gegebenen Bedingungen verletzt. Individuen, bei denen dieser Fall eintritt, müssen noch über den in Abschnitt 6.7 eingeführten Reparaturalgorithmus korrigiert werden.

6.3.2 Erzeugungsvarianten

In dieser Arbeit wurden verschiedene Varianten zur Initialisierung der Anfangspopulation getestet, wobei sich die einzelnen Varianten durch das Verhältnis aus zufällig erzeugten und heuristisch erzeugten Individuen, sowie durch die gewählten Heuristiken unterscheiden. Eine Übersicht über die verschiedenen Heuristiken ist in Kapitel 5 gegeben. Die folgenden Varianten sollen hier erwähnt werden, da sie in Kapitel 7 angewandt werden (Die Kürzel für die Stations- und Ressourcenauswahl sind in Abschnitt 5.1 bzw. Abschnitt 5.2 zu finden):

- **All Random:** Wie der Name schon vermuten lässt, werden bei dieser Variante sämtliche Individuen der ersten Generation zufällig erstellt.

- **3-Added:** Hier werden der Startpopulation 3 heuristisch erzeugte Individuen beigefügt, welche nach folgenden Regeln erstellt wurden:

  - Stationsauswahl: *GRW-EMB*, Ressourcenauswahl: *GKD*
  - Stationsauswahl: *MEZ-EMB*, Ressourcenauswahl: *MAS-GKD*
  - Stationsauswahl: *MKA-GRW-EMB*, Ressourcenauswahl: *MRS-GKD*

- **5-Added:** Hier werden der Startpopulation zusätzlich zu den 3-Added Individuen noch folgende beiden Individuen hinzugefügt:
  - Stationsauswahl: *EMB-GRW*, Ressourcenauswahl: *MAS-GKD*
  - Stationsauswahl: *MKA-GRW-EMB*, Ressourcenauswahl: *MAS-GKD*

6.4 Selektionsverfahren

Nach dem Erzeugen der Startpopulation, muss man sich für ein Verfahren entscheiden, nach welchem die einzelnen Individuen für den Paarungspool ausgewählt werden. Das Ziel dabei ist natürlich, Individuen mit einer besseren Fitness zu selektieren, in der Hoffnung, dass diese wiederum Kinder mit einer noch besseren Fitness produzieren. Die Wahl des Selektionsverfahrens ist eine Gratwanderung, bei welcher man die richtige Balance zwischen Konvergenzgeschwindigkeit und Lösungsvielfalt finden muss. Wählt man Individuen auf sehr elitäre Art und Weise, also so, dass sich die besten Individuen sehr schnell gegen schwächere durchsetzen, so wird man zwar in der Regel schneller zu guten Lösungen kommen, allerdings verliert man dadurch aber auch schneller an Vielfalt, wodurch die Gefahr steigt, dass man in einem lokalen Optimum stecken bleibt. Wählt man wiederum ein Selektionsverfahren, bei welchem sehr häufig auch schwache Individuen in den Paarungspool kommen, so kann dies zu einem schlechten Konvergenzverhalten führen. Einige der gängigsten Selektionsverfahren werden in diesem Kapitel vorgestellt, angelehnt an die Ausführungen von Mitchel [18].

6.4.1 Fitnessproportionale Selektion

Bei der fitnessproportionalen Selektion ist die Wahrscheinlichkeit, dass ein Individuum in den Paarungspool aufgenommen wird, proportional zu dessen **relativer Fitness**. Die relative Fitness eines Individuums entspricht dabei dem Quotient von absoluter Fitness des Individuums durch die Summe der Fitnesswerte aller Individuen. Es gibt verschiedene Wege, eine fitnessproportionale Selektion durchzuführen. Die zwei gängigsten Methoden sollen hier vorgestellt werden.

Roulette Wheel Selektion

Dies ist die gängigste Methode um Individuen fitnessproportional auszuwählen. Jedem Individuum wird dabei ein Teil eines Roulettekessels zugewießen, dessen Größe der relativen Fitness des Individuums entspricht. Nun wird n mal eine Kugel in den Roulettekessel geworfen und bei jedem Wurf wird ein Individuum für den Paarungspool ausgewählt. So entsteht schlussendlich ein Paarungspool mit genau n Individuen.

Beispiel 6.1. Gegeben sei eine Population mit 4 Individuen I_1, I_2, I_3, I_4 und deren Fitnesswerten $f_1 = 4, f_2 = 3, f_3 = 2, f_4 = 11$. Der Roulettekessel würde dann folgendermaßen aussehen.

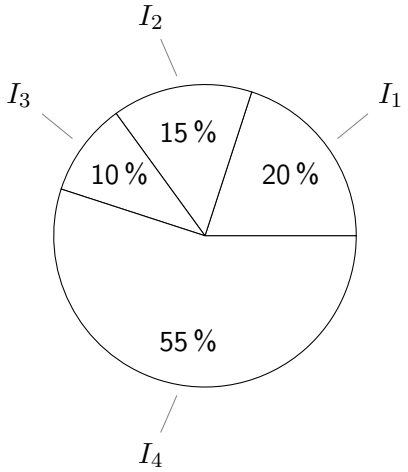


Abbildung 6.2: Roulettekessel für Beispiel 6.1

Bei der Roulette Wheel Selektion wird in jedem Durchgang zufällig ein Indivduum ausgesucht, unabhängig davon, welche Individuen in den vorherigen Durchgängen gewählt wurden. Es ist daher möglich, wenn auch extrem unwahrscheinlich, dass n mal das Individuum mit dem geringstem Fitnesswert ausgewählt wird. Gerade bei Populationen mit geringer Größe kann es aber durchaus vorkommen, dass sich, entgegen der statistischen Warscheinlichkeit, in einer Generation größtenteils schwache Individuen durchsetzen. Das im nächsten Abschnitt vorgestellte Verfahren versucht, diesem Effekt etwas entgegenzuwirken.

Stochastic universal Sampling (SUS)

Beim SUS wird wieder ein Rouletekessel erstellt, genau so wie auch bei der Roulete Wheel Selektion (siehe Abbidlung 6.2). Nun wird der Roulettekessel aber nicht n mal gedreht um je ein Individuum zu bestimmen. Stattdessen wird er nur einmal gedreht, aber mit n Zeigern, welche in gleichen Abständen angeordnet werden und je ein Individuum bestimmen. Durch dieses Verfahren werden Extremfälle, welche bei der Roulette Wheel Selektion auftreten können, vermieden und somit eine gewisse Vielfalt bei der Auswahl des Paarungspools garantiert.

6.4.2 Rangselektion

Anders als bei der fitnessproportionalen Selektion, wird hier die relative Fitness eines Individuums nicht anhand dessen absoluter Fitness bestimmt, sondern anhand dessen Ranges. Dieser wird bestimmt, indem man die Individuen nach ihrer absoluten Fitness sortiert. Hat man auf diese Weise die relative Fitness aller Individuen bestimmt, so kann man, wie schon bei der fitnessproportionalen Selektion, mithilfe von Roulette Wheel Selektion oder SUS den Paarungspool befüllen. Der Vorteil der Rangselektion ist, dass extrem starke Individuen, durch das skalieren der absoluten Fitnesswerte auf Ränge, sich nicht zu schnell durchsetzen. Hat das beste Individuum zum Beispiel einen wesentlich höheren absoluten Fitnesswert alls alle anderen, so würde dieses Individuum bei der fitnessproportionalen Selektion sehr oft in den Paarungspool gewählt werden. Bei der Rangselektion dagegen, hätte es nur eine gering höhere relative Fitness als das zweitbeste Individuum der Population und würde daher auch nur geringfügig öfter im Paarungspool landen als dieses.

Beispiel 6.2. Gegeben sei Population aus Beispiel 6.1 mit 4 Individuen I_1, I_2, I_3, I_4 und deren Fitnesswerten $f_1 = 4, f_2 = 3, f_3 = 2, f_4 = 10$. Berechnet man die relative Fitness nun nach dem Rang, so würde der Roulettekessel folgendermaßen außehen.

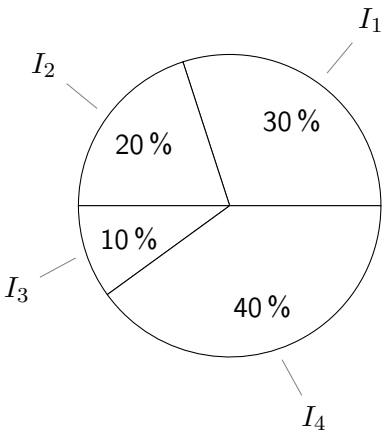


Abbildung 6.3: Roulettekessel für Beispiel 6.2

In Abbildung 6.3 ist gut zu erkennen, dass Individuum 4, welches über einen deutlich besseren absoluten Fitnesswert verfügt als die restlichen 3 Individuen, durch die Rangselektion einen nicht mehr ganz so dominanten Anteil des Roulettekessels einnimmt wie noch in Beispiel 6.1, in welchem dieser Anteil fitnessproportional berechnet wurde.

6.4.3 Turnierverfahren

Ein weiters Verfahren zur Selektion der Individuen für den Paarungspool ist das Turnierverfahren. Bei einer Populationsgröße von n werden hier in jedem Verfahrensschritt x Individuen ausgewählt, wobei gilt $2 <= x <= n$. Im einfachsten Fall gewinnt das Individuum mit dem höchsten absoluten Fitnesswert das Turnier und wird somit dem Paarungspool beigefügt. Man kann stattdessen aber auch x Werte $\delta_1, \delta_2, ..., \delta_x$ festlegen, mit $(\sum_{i=1}^{x} \delta_i) = 1$, um zu bestimmen, mit welcher Wahrscheinlichkeit eines der ausgewählten Individuen das Turnier gewinnt. Die Ermittlung des Siegers geschieht auf die selbe Art und Weise wie bei der Roulette Wheel Selektion, wie in Beispiel 6.3 dargestellt. Da die Individuen beim Turnierverfahren auf jeden Fall gereiht werden müssen, es aber vorkommen kann, dass mehrere Individuen über die selbe Fitness verfügen, muss man für diese Fälle eine zusätzliche Regel zum Auflösen eines solchen Unentschiedens anwenden. Die simpelste Möglichkeit ist, in so einem Fall jenes Individuum höher zu reihen, welches zuerst für das Turnier ausgewählt wurde.

Beispiel 6.3. Durch ein Turnier mit Größe $x = 3$ soll eine Individuum für den Paarungspool ermittelt werden. Gegeben sind die Wahrscheinlichkeiten $\delta_1 = 0.75$, $\delta_2 = 0.2$ und $\delta_3 = 0.05$, sowie die drei Individuen I_1, I_2, I_3 und deren absolute Fitnesswerte

$f_1 = 3, f_2 = 7, f_3 = 4$. Es entsteht also folgender Roulettekessel, mithilfe dessen man wie in Abschnitt 6.4.1 einen Sieger ermittelt.

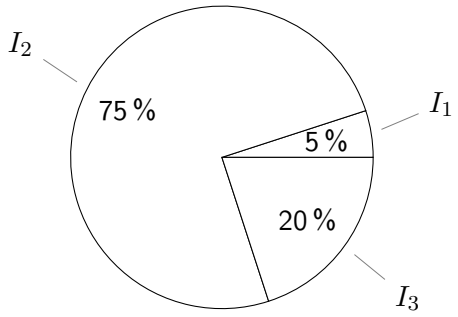


Abbildung 6.4: Roulettekessel für Beispiel 6.3

Im Gegensatz zu den vorher genannten Verfahren, ist es bei diesem Verfahren nicht notwendig, die relative Fitness eines jeden Individuums zu berechnen. Gerade bei der Rangselektion, bei welcher man die absoluten Fitnesswerte aller Indivdien sortieren muss, kann dies einen gewissen Rechenaufwand darstellen, welchen man sich beim Turnierverfahren erspart.

Dadurch, dass die Individuen, welche an einem Turnier teilnehmen, vollkommen zufällig ermittelt werden, haben auch schwächere Individuen eine reelle Chance ein Turnier zu gewinnen, da es ja sein kann, dass sie für ein Turnier gewählt werden, in welchem sie sich nur gegen andere schwache Individuen durchsetzen müssen. Der Selektionsdruck dieses Verfahrens ist daher geringer als zum Beispiel bei der fitnessproportionalen Selektion. Zusäztlich lässt sich der Selektionsdruck durch die Größe der Turniere, sowie die die Wahrscheinlichkeiten δ_i auch recht gut steuern. Je größer zum Beispiel die Turniere sind, desto geringer ist die Chance, dass sich schwache Individuen durchsetzen, wodurch der Selektionsdruck steigt. Konstruiert man ein Turnier der Größe x so, dass automatisch das stärkste Individuum gewinnt, also:

$$\delta_i = \begin{cases} 1, \text{ für } i = 1 \\ 0, \text{ für } i \neq 1 \end{cases}$$

und lässt man nicht zu, dass ein Individuum mehr als einmal für das selbe Turnier ausgewählt wird, so ist es für die schwächsten $x - 1$ Individuen nicht möglich in den Paarungspool zu gelangen.

6.4.4 Elitismus

Hierbei handelt es sich weniger um ein eigenständiges Selektionsverfahren, sondern eher um eine Ergänzung für andere Verfahren. Die Idee dabei ist, einen Teil der Population, nämlich deren beste x Individuen, unverändert in die nächste Generation zu übernehmen. Dadurch wird vermieden, dass sehr gute Individuen bei der Selektion oder durch Kreuzung und/oder Mutation verloren gehen.

6.5 Kreuzungsverfahren

Das Ziel eines Kreuzungsverfahrens ist es, durch das Vermischen der Eigenschaften zweier oder mehrerer Eltern-Individuen, ein oder mehrere Kind-Individuen zu erzeugen, welche im Idealfall über eine bessere Fitness verfügen als ihre Eltern. Sollte durch das Kreuzungsverfahren nicht sichergestellt werden können, dass die Kind-Individuen gültige Lösungen für das vorliegende Problem darstellen, so ist es notwendig, ungültige Individuen mittels eines Reparaturalgorithmus wieder in Ordnung zu bringen. Diese Problematik wird in Kapitel 6.7, mit Bezug auf das vorliegende Problem, noch genauer behandelt. In den folgenden Abschnitten werden das Vater- und das Mutter-Individuum als I^V bzw. I^M und die Kind-Individuen (Sohn, Tochter) als I^S bzw. I^T bezeichnet.

6.5.1 Single Point Crossover

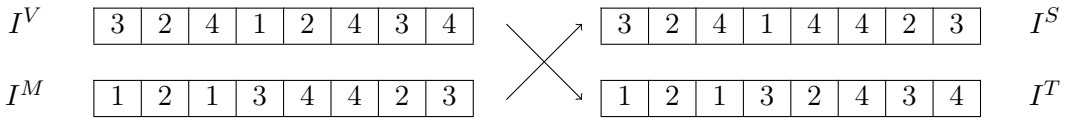
Beim Single Point Crossover wird eine einzelne Kreuzungsposition q bestimmt, mit $1 <= q < l$, wobei l der Länge des Chromosoms entspricht. Anschließend werden jene Gene der Eltern-Individuen, welcher hinter dieser Position liegen, miteinander vertauscht, wodurch zwei neue Kind-Individuen entstehen. Für jedes Gen I_i^S, des Sohnes gilt also:

$$I_i^S = \begin{cases} I_i^V, & \forall\, i \leq q \\ I_i^M, & \forall\, i > q \end{cases}$$

und für die Gene I_i^T der Tochter gilt umgekehrt:

$$I_i^T = \begin{cases} I_i^M, & \forall\, i \leq q \\ I_i^V, & \forall\, i > q \end{cases}$$

Beispiel 6.4. Gegeben seien das folgende Vater- und Mutter-Individuum und der Kreuzungspunkt $q = 4$. Die Kind-Individuen würden dann wie folgt konstruiert werden.



Man kann nicht allgemein sagen, welches Kreuzungsverfahren die besten Ergebnisse liefert. Wie gut die einzelnen Verfahren funktionieren, hängt stark vom vorliegenden Problem ab. Eines der Hauptprobleme des Single Point Crossover ist, dass Schemata mit großer definierender Länge sehr wahrscheinlich zerstört werden. Es eignet sich daher besser für Probleme, in welchem gute Individuen hauptsächlich aus Schemata mit geringer Ordnung und kurzer definierender Länge aufgebaut sind. Des weiteren werden beim Single Point Crossover gewisse Gene bevorzugt behandelt. Das letzte Gen eines jeden Chromosoms ist zum Beispiel garantiert immer in den Teilen des Vater- und Mutter-Chromosoms enthalten, welche miteinander vertauscht werden.

6.5.2 Two Point Crossover

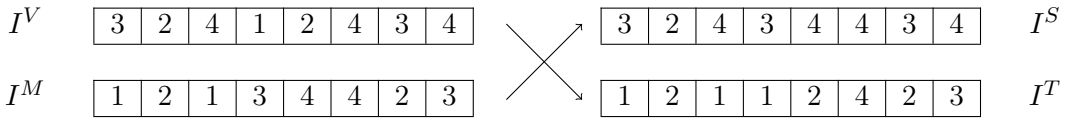
Beim Two Point Crossover werden, statt einer einzelnen Kreuzungsposition q, zwei Kreuzungspositionen q_1 und q_2 erzeugt. Anschließend werden in den Eltern-Individuen die Gene zwischen den beiden gewählten Positionen vertauscht. Für jedes Gen I_i^S des Sohnes gilt beim Two Point Crossover somit:

$$I_i^S = \begin{cases} I_i^V, & \forall\, i \leq q_1 \\ I_i^M, & \forall\, i > q_1,\ i \leq q_2 \\ I_i^V, & \forall\, i > q_2 \end{cases}$$

und für die Gene I_i^T der Tochter gilt umgekehrt:

$$I_i^S = \begin{cases} I_i^M, & \forall\, i \leq q_1 \\ I_i^V, & \forall\, i > q_1,\ i \leq q_2 \\ I_i^M, & \forall\, i > q_2 \end{cases}$$

Beispiel 6.5. Gegeben seien noch einmal das Vater- und Mutter-Individuum aus Beispiel 6.4 und die beiden Kreuzungspunkte $q_1 = 3,\ q_2 = 6$. Die Kind-Individuen sehen dann folgendermaßen aus:

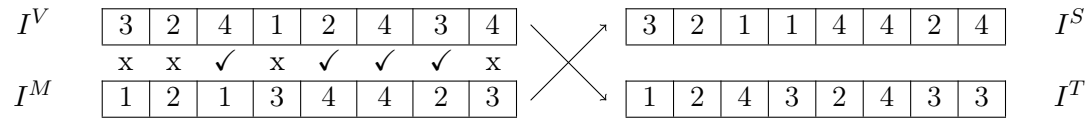


Beim Two Point Crossover werden Schemata mit großer definierender Länge nicht mit so großer Wahrscheinlichkeit zerstört wie beim One Point Crossover, wodurch es sich für Probleme, bei welchen gute Individuen eher aus solchen Schemata aufgebaut sind, besser eignet. Des weiteren befindet sich das letzte Gen eines Chromosoms nicht mehr zwinged in dem zu vertauschenden Teil der Eltern-Individuen.

6.5.3 Uniform Crossover

Eine weitere Möglichkeit zwei Individuen miteinander zu kreuzen ist das Uniform Crossover. Dabei wird jedes Gen mit einer Wahrscheinlichkeit p (üblicherweiße $0,5 \leq p \leq 0,8$) zwischen den beiden Eltern-Individuen vertauscht. Wählt man $p = 0.5$ so werden die Kind-Individuen in etwa gleich viel Genmaterial von beiden Eltern erhalten. Bei $p > 0.5$ wird eines der Kind-Individuen stärker nach dem Vater und das andere mehr nach der Mutter kommen.

Beispiel 6.6. Gegeben seien erneut das Vater- und Mutter-Individuum aus Beispiel 6.4. Des weiteren wrid zwischen den Genen der Eltern-Individuen angezeigt, ob die beiden jeweiligen Gene miteinander vertauscht werden sollen oder nicht. Die Kind-Individuen sehen dann folgendermaßen aus:



Diese Art zu kreuzen hat den Vorteil, dass sämtliche Schemata, welche in den Eltern-Individuen zu finden sind, potenziell auch in den Kind-Individuen enthalten sein können, was beim Single und Two Point Crossover nicht der Fall ist. Das Problem dabei ist allerdings, dass es beim Uniform Crossover ebenfalls sein kann, dass jegliche in den Eltern-Individuen enthaltenen Schemata zerstört werden. Wie schon in Abschnitt 6.5.1 gesagt, ist es sehr stark problemabhängig, welches Art zu kreuzen die besten Resultate hervorbringt.

6.6 Mutation

Hat man, durch das Kreuzen der Individuen des Paarungspools, eine neue Generation erschaffen, so erfolgt üblicherweise noch ein Mutationsschritt. Dabei werden zufällig Gene aus der neuen Population ausgewählt und auf einen wiederum zufällig aus deren Allele gewählten Wert gesetzt. Um dies zu tun, wird ein Wert p_m festgelegt, welcher bestimmt, mit welcher Wahrscheinlichkeit ein Gen zur Mutation ausgewählt wird. Meist wird zwar das Kreuzen als wesentlichster Bestandteil der Erzeugung neuer, besserer Individuen angesehen, aber das Mutieren leistet ebenfalls einen unverzichtbaren Beitrag zum Erfolg eines genetischen Algorithmus. Mühlbein [19] betont zum Beispiel in seiner Veröffentlichung, dass die Macht der Mutation oft unterschätzt wird. Die absolute Notwendigkeit des Mutierens wird schnell ersichtlich, wenn man die folgende Population, bestehend aus den Individuen I_1, I_2, I_3, I_4, ansieht (Es sei hier noch zusätzlich erwähnt, dass alle Gene über die gleiche Allele, nämlich {1,2} verfügen):

I_1:	1	2	1	1	2	2
I_2:	1	2	1	1	1	2
I_3:	1	1	2	2	2	2
I_4:	1	2	1	2	1	2

Betrachtet man zum Beispiel das jeweils erste Gen der oben angeführten Individuen, so sieht man, dass sich dieses durch das Kreuzen zweier beliebiger Individuen, unabhängig davon welches Kreuzungsverfahren man wählt, nicht verändern wird. Es wäre also ohne Mutation unmöglich, ein Individuum zu erzeugen, bei welchem das erste Gen den Wert 2 hat. Das selbe Phänomen lässt sich auch beim letzten Gen beobachten. Somit bliebe, ohne Mutation, ein Teil des Lösungsraumes unerreichbar. Mutation wirkt sich außerdem positiv auf die genetische Vielfalt einer Population aus.

Die Mutationsrate kann, wie vorhin schon erwähnt, über p_m gesteuert werden und liegt üblicherweise zwischen 0.1% und 1%. Wählt man einen zu geringen Wert, so steigt die Wahrscheinlichkeit, dass der Algorithmus länger in einem lokalen Optimum hängen bleibt. Des weiteren kann es bei zu geringer Mutationsrate auch zu einer sehr geringen Vielfalt innerhalb der Population kommen. Wählt man einen zu hohen Wert für p_m, so besteht die Gefahr, dass zu viele gute Individuen zerstört werden und die Erfolge der Kreuzung zunichte gemacht werden.

6.7 Reparaturalgorithmus

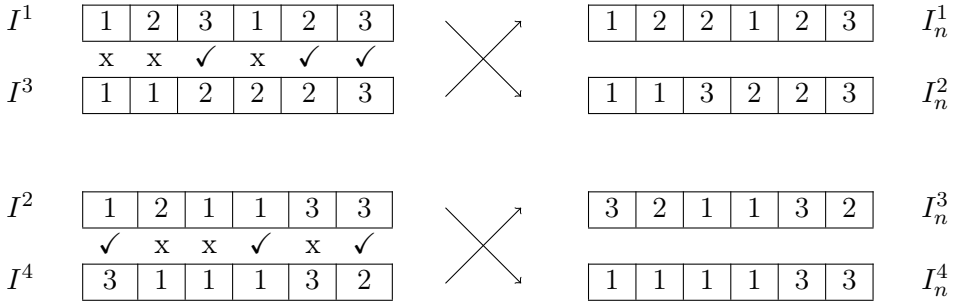
Sowohl durch das Kreuzen der Individuen als auch die zufällige Mutation der Gene kann es dazu kommen, dass in einer Population Individuen vorhanden sind, welche für das vorliegende Problem ungültige Lösungen repräsentieren. Eine Möglichkeit mit diesem Umstand umzugehen wäre, einem ungültigen Individuum eine sehr schlechte Fitness zuzuweißen und darauf zu vertrauen, dass es dadurch früher oder später aussterben wird. Möchte man allerdings ausschließlich gültige Individuen zulassen, so müssen alle ungültigen Individuen durch einen zusätzlichen Algorithmus repariert werden.

Betrachtet man das in dieser Arbeit vorliegende Problem, so können ungültige Individuen dadurch entstehen, dass für eine Station das festgelegte Minimum/Maximum an Ressourcen unterschritten/überschritten wird. Dies kann sowohl durch Kreuzung als auch durch Mutation passieren, wie in Beispiel 6.7 ersichtlich.

Beispiel 6.7. Gegeben sei ein System mit 3 Stationen und 6 Ressourcen. Jede Ressource kann jeder Station zugeteilt werden. Des weiteren muss jeder Station mindestens 1 Ressource zugeteilt sein und es dürfen keiner Station mehr als 3 Ressourcen zugeteilt werden. Gegeben sei weiters ein Paarungspool mit 4 Individuen I_1, I_2, I_3, I_4. Durch Uniform Crossover soll nun eine neue Generaton erstellt werden, in welcher anschließend noch zufällige Mutationen durchgeführt werden sollen.

I_1:	1	2	3	1	2	3
I_2:	1	2	1	1	3	3
I_3:	1	1	2	2	2	3
I_4:	3	1	1	1	3	2

Nun werden zufällig einmal die beiden Individuen I_1, I_3 und einmal die beiden Individuen I_2, I_4 als Eltern ausgewählt und ein Uniform Crossover durchgeführt.



Betrachtet man nun die durch das Kreuzen erzeugten Individuen, so sieht man, dass bei I_n^4 keine einzige Ressource auf Station 2 zugeteilt ist, wodurch das Minimum von 1 unterschritten wird. Außerdem werden 4 Ressourcen auf Station 1 zugeteilt, wodurch das zulässige Maximum für Station 1 ebenfalls überschritten wird. Würde bei den anschließenden Mutationen das letzte Gen von I_n^1 noch verändert werden, so wäre auch dieses Individuum ungültig, da keine Ressource auf Station 3 zugeteilt wäre.

Da in dieser Arbeit keine ungültigen Individuen zugelassen werden, wird der in Algorithmus 5 dargestellte Reparaturalgorithmus eingesetzt um die fehlerhaften Individuen zu korrigieren. Die Idee dabei ist, Ressourcen von überbesetzten Stationen auf unterbesetzte Stationen zu verschieben, falls dies möglich ist. Sollte dies nicht möglich sein, wird versucht, Ressourcen von überbesetzten Stationen auf eine andere, für die Ressource zulässige Station zu verschieben. Dies entspricht der Änderung eines Gens, dessen Wert der überbesetzten Station entspricht, auf einen anderes Element seiner Allele. Dabei wird zuerst versucht, ein Gen zu finden, dessen Allele eine Station beinhaltet, welche durch die Zuteilung einer weiteren Ressource nicht über ihr zulässiges Maximum geraten würde. Gibt es kein solches Gen mehr, wird einfach eines zufällig ausgewählt und dessen Wert auf ein anderes Element seiner Allele gesetzt.

Gibt es keine überbesetzten Stationen mehr, so wird versucht, eine Ressource von einer ausreichend besetzten Station auf eine unterbesetzte Station zu verschieben. Dies entspricht der Änderung eines Gens, dessen Wert nicht der unterbesetzten Station entspricht und in dessen Allele die unterbesetzte Station enthalten ist, auf eben diese Station. Dabei wird zuerst versucht, ein Gen zu finden dessen Wert einer Station entspricht, welche durch den Verlust einer Ressource nicht unter die vorgeschriebene Mindestanzahl an Ressourcen gelangen würde. Sollte es kein solches Gen geben, wird ein zufälliges Gen gewählt, dessen Allele die unterbesetzte Station beinhaltet.

Algorithmus 5 Reparaturalgorithmus

Input Ungültiges Individuum
Output Gültiges Individuum

1: Setze $unterbesetzt := \{\}$; $ueberbesetzt := \{\}$
2: Setzte $count[x]$ = Anzahl der Gene mit Wert x
3: **for** Station s in Stationen **do**
4: **if** $count[s] < min_s$ **then**
5: $unterbesetzt \ll s$
6: **end if**
7: **if** $count[s] > max_s$ **then**
8: $ueberbesetzt \ll s$
9: **end if**
10: **end for**
11: **while** $|unterbesetzt| > 0$ oder $|ueberbesetzt| > 0$ **do**
12: **if** $|ueberbesetzt| > 0$ **then**
13: Wähle eine Station s aus $ueberbesetzt$
14: Suche ein Gen mit dem Wert s, dessen Allele eine Station s_n aus $unterbesetzt$ beinhaltet
15: Wurde kein Gen gefunden, suche stattdessen ein Gen mit Wert s, dessen Allele eine Station s_n beinhaltet mit $count[s_n] < max_{s_n}$
16: Wurde kein Gen gefunden, wähle stattdessen ein beliebiges Gen mit Wert s und wähle ein beliebiges s_n aus dessen Allele
17: Setze Wert des Gens $= s_n$; $count[s]--$; $count[s_n]++$
18: **else**
19: Wähle eine Station s aus $unterbesetzt$
20: Suche ein Gen dessen Allele s beinhaltet mit $count[WertDesGens] > min_{WertDesGens}$
21: Wurde kein Gen gefunden, wähle stattdessen ein beliebiges Gen dessen Allele s beinhaltet
22: $count[s]++$; $count[WertDesGens]--$; Wert des Gens $= s$
23: **end if**
24: **for** Station s in Stationen **do**
25: **if** $count[s] < min_s$ **then**
26: $unterbesetzt \ll s$
27: **end if**
28: **if** $count[s] > max_s$ **then**
29: $ueberbesetzt \ll s$
30: **end if**
31: **end for**
32: **end while**

Beispiel 6.8. Um die Vorgehensweise des vorgestellten Reparaturalgorithmus darzustellen, sei folgend ein Individuum gegeben, welches eine Lösung für ein System darstellt, das aus 8 Ressourcen $r_1, ..., r_8$ und 4 Stationen $s_1, ..., s_4$ besteht. Welche Ressource welcher Station zugeteilt werden kann, ist in den Allelen der jeweiligen Gene ersichtlich, welche unter dem Individuum angegeben sind. Eine Lösung für dieses System ist nur dann gültig, wenn jeder Station mindestens 1 und maximal 3 Ressourcen zugeteilt sind.

Abbildung 6.5: Individuum für Beispiel 6.8 inkl. Allelen

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
1	4	4	4	4	4	1	4

1	2	1	1	1	4	1	1
2	4	2	2	2		2	2
3		4	4	4		3	4
4						4	

Zu Beginn muss also die Anzahl c_s der zugeteilten Ressourcen pro Station ermittelt werden. Dies ergibt $c_1 = 2,\ c_2 = 0,\ c_3 = 0,\ c_4 = 6$. Somit entstehen die beiden Mengen $unterbesetzt = \{s_2, s_3\}$ sowie $ueberbesetzt = \{s_4\}$.

Nachdem die Initialsierung der Mengen abgeschlossen ist, sollen nun schrittweise einzelne Gene verändert werden. Da in *ueberbesetzt* nur genau ein Element, nämlich s_4, vorhanden ist, wird als erstes ein Gen gesucht, welches eine Ressoure auf dieser Station zuteilt und in dessen Allele eine Station aus *unterbesetzt* enthalten ist. Dies ist bei dem Gen mit Position 2 der Fall. Somit wird der Wert dieses Genes von 4 auf 2 gesetzt wodurch sich folgendes neue Individuum ergibt:

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
1	2	4	4	4	4	1	4

Durch diesen Tausch steigt c_2 auf 1, wodurch s_2 aus *unterbesetzt* entfernt wird. Da sich aber noch immer Elemente in *unterbesetzt* und/oder *ueberbesetzt* befinden, müssen noch weitere Gene verändert werden. Da s_4 weiterhin das einzige Element in *ueberbesetzt* ist, gilt es, wieder ein Gen, welches einer Zuteilung auf Station 4 entspricht, zu ändern. Diesmal gibt es allerdings kein solches Gen mehr, welches in seiner Allele eine Station aus *unterbesetzt* beinhaltet, weshalb diesmal ein Gen gewählt wird, dessen Allele eine Station beinhaltet, die durch die Aufnahme einer weiteren Ressource nicht über deren

zulässiges Maximum kommen würde. Sollte es mehrere passende Gene geben, wird unter diesen ein zufälliges ausgewählt. Somit bekommt das Gen an dritter Stelle den Wert 1 zugewießen und folgendes Individuum entsteht:

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
1	2	1	4	4	4	1	4

Es folgt noch einmal ein ähnlicher Schritt wie zuvor, nur dass nun das Gen an 4ter Stelle ausgetauscht wird und es den Wert 2 bekommt, da Station s_1 durch eine weitere Zuteilung bereits über deren zulässiges Maximum geraten würde. Es entsteht ein fast schon gültiges Individuum:

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
1	2	1	2	4	4	1	4

Durch den letzten Schritt ist nun $c_4 = 3$, womit s_4 aus *ueberbesetzt* entfernt werden kann. Somit gibt es keine Station mehr, welche über ihr zulässiges Maximum hinaus besetzt ist. Allerdings befindet sich immer noch eine Station, nämlich s_3, in *unterbesetzt*. Da *ueberbesetzt* nun leer ist, wird im nächsten Schritt versucht einer Station aus *unterbesetzt* eine Ressource zuzuteilen. Da nur noch s_3 darin enthalten ist, wird nun ein Gen gesucht, welches nicht bereits einer Zuteilung auf s_3 entspricht, dessen Allele diese Station beinhaltet und dessen aktuell zugeteilte Station durch Verlust einer Ressource nicht unter das vorgeschriebene Minimum geraten würde. In diesem Fall wird das Gen an erster Stelle ausgewählt, wodurch schlussendlich folgendes gültige Individuum konstruiert wurde:

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8
3	2	1	2	4	4	1	4

Es sei zu dem Algorithmus noch angemerkt, dass es wichtig ist, in einer Situation mit mehreren passenden Genen, das zu verändernde Gen zufällig auszuwählen und dann auch aus dessen Allele zufällig einen der passenden Werte zu bestimmen. Nimmt man zum Beispiel immer das erstbeste passende Gen, so kann es dazu kommen, dass der Algorithmus nie terminiert, wie in dem folgenden Beispiel ersichtlich.

Beispiel 6.9. Gegeben sei ein System mit 4 Stationen s_1, s_2, s_3, s_4 und 4 Ressourcen

r_1, r_2, r_3, r_4. Jeder Station darf nur genau 1 Ressource zugeteilt werden. Abbildung 6.6 zeigt ein ungültiges Individuum und zusätzlich die Allelen eines jeden Gens.

Abbildung 6.6: Individuum für Beispel 6.9 inkl. Allelen

r_1	r_2	r_3	r_4
1	1	2	3

1	1	1	3
2	2	2	4
		3	

Würde man nun immer das zu vertauschende Gen nun nicht zufällig wählen, sondern immer das erstbeste wählen, würde folgendes passieren. Der Algorithmus findet im ersten Durchlauf, dass Station s_1 überbesetzt ist, da ihr 2 Ressourcen zugeteilt sind. Keines der Gene mit Wert 1 enthalten in dessen Allele eine Station die unterbesetzt ist und gleichermaßen enthalten dessen Allelen auch keine Station, welcher weniger Ressourcen als maximal erlaubt zugeteilt sind. In diesem Fall würde das erstbeste Gen, also hier das Gen an Position 1, den erstbesten Wert aus dessen Allele bekommen. Das Individuum sähe nun folgendermaßen aus:

Abbildung 6.7: Individuum für Beispel 6.9 nach dem ersten Reperaturschritt

r_1	r_2	r_3	r_4
2	1	2	3

Nun ist Station s_2 überbesetzt und wiederum enthält keine Allele der entsprechenden Gene eine Station die unterbesetzt ist, oder der zumindest weniger Ressourcen als maximal erlaubt zugeteilt sind. Vertauscht man also wieder das erstbeste Gen, so hätte man anschließend folgendes Individuum:

Abbildung 6.8: Individuum für Beispel 6.9 nach dem zweiten Reperaturschritt

r_1	r_2	r_3	r_4
1	1	2	3

Nun hat man allerdings wieder das gleiche Individuum wie am Anfang und der Algorithmus würde weiterhin immer genau die selben zwei Schritte vollführen wie eben aufgezeigt.

6.8 Parameter

Es gibt eine Vielzahl von Parametern, welche einen entscheidenden Einfluss auf die Qualität und Geschwindigkeit eines genetischen Algorithmus haben und viele Wissenschaftler haben sich bereits damit beschäftigt, optimale Werte für diese zu finden. Zu den am häufigsten betrachteten Parametern zählen dabei die **Populationsgröße**, **Kreuzungsrate** (diese gibt an, welcher Anteil der Individuen einer Generation durch Kreuzung entstehen soll), **Mutationsrate** und die **Selektionsstrategie** (elitär, nicht elitär). Während man klarerweise nicht allgemein sagen kann, welche Parameter für ein spezielles Problem die besten Resultate liefern werden, haben sich doch bestimmte Werte als gute Ausgangslage für weitere Tests hervorgetan. Ein besonders interessantes Experiment wurde dabei von Grefenstette [11] durchgeführt. Dieser entwickelte einen genetischen Algorithmus zur Optimierung der Parameter für genetische Algorithmen. Dabei wurden verschiedenen Parametersets Fitnesswerte zugeteilt und so die Qualität der Parameter gemessen. Grefenstette's Algorithmus betrachtete dabei sechs verschiedene Paremeter, wobei nur vier Parameter aufgrund ihrer Relevanz hier erwähnt werden sollen. Er ermittelte eine optimale Populationsgröße von 30, eine Kreuzungsrate von 0.95, eine Mutationsrate von 0.01 und ein elitäres Selektionsverhalten.

7 Analyse

In Kapitel 6 wurden genetische Algorithmen vorgestellt und aufgezeigt, wie diese auf das in dieser Arbeit behandelte Human Ressource Allocation Problem angewandt werden können. In den folgenden Abschnitten soll, anhand einiger konkreter Problemstellungen, die Qualität der durch einen genetischen Algorithmus erzielten Lösungen mit der Qualität jener Lösungen verglichen werden, welche durch die in Kapitel 5 vorgestellten, einfachen Heuristiken erzeugt wurden. Für kleinere Testprobleme wird, statt einer heuristisch erzeugten Lösung, die optimale Lösung zum Vergleich herangezogen. Der genetische Algorithmus wurde zu diesem Zweck mit verschiedenen Parametersets je 20 mal auf ein Testproblem angewandt. Die einzelnen Testläufe endeten, wenn sich das beste Individuum 500 Generationen lang nicht veränderte oder wenn das erlaubte Maximum von 2000 Generationen erreicht wurde. Dabei wurden verschiedene Messkriterien erfasst und sowohl die Durchschnittswerte als auch die Spitzenwerte pro Parameterset festgehalten. Die Berechnung der Fitnesswerte erfolgte, wie in Kapitel 6 beschrieben. Wurden bei einem Individuum allerdings nicht sämtliche Mindestbedarfe aller Stationen erfüllt, so wurde dessen Fitness zur Strafe halbiert. Die folgenden Kriterien wurden betrachtet, um die Leistung der verschiedenen Testläufe zu quantifizieren:

Bewertungskriterien pro Parameterset	
BFit	Die Fitness des besten gefundenen Individuums aller Testläufe
DFit	Die durchschnittliche Fitness der jeweils besten Individuen pro Teslauf
BGen	Die früheste Generation in welcher das insgesamt beste Individuum gefunden wurde
DGen	Die durchschnittliche Generation in welcher das jeweils beste Individuum pro Testlauf gefunden wurde
FGen	Die frühesete Generation in welcher das beste Individuum eines Testlaufs gefunden wurde
SGen	Die späteste Generation in welcher das beste Individuum eines Testlaufs gefunden wurde
LZ	Die Laufzeit

Tabelle 7.1: Bewertungskriterien der Algorithmen

Da die Anzahl der möglichen Parameterkombinationen einfach zu groß ist um sie alle

abzudecken, werden in den folgenden Abschnitten nur einige ausgewählte Parametersets betrachtet. Da es zum Beispiel in einer Vielzahl von Testläufen keinen spürbaen Unterschied machte, ob Individuen mittels Roulette Wheel Selektion oder SUS ausgewählt wurden, werden folgend nur jene Testergebnisse angegeben, bei welchen SUS eingesetzt wurde. Zusätzlich wurde für alle Testläufe eine konstante Kreuzungsrate von 0.95, sowie eine Populationsgröße von 30 Individuen verwendet, wobei bei der Erzeugung einer neuen Generation jeweils die 2 besten sowie das schlechteste Individuum der Eltern-Generation automatisch in die nächste Generation übernommen wurde. Somit bleiben als veränderliche Parameter noch die Generierung der Startpopulation, das angewandte Selektionsverfahren, das angewandte Kreuzungsverfahren sowie die Mutationsrate. Die verschiedenen Parametersets, welche sich aus den soeben genannten fixen Parametern, sowie aus unterschiedlichen Kombinationen der variablen Paremter zusammensetzen, werden in Tabelle 7.2, inklusive einer Kurzbezeichnung, angeführt. Für alle Parametersets, bei welchen mittels Turnierverfahren selektiert wurde, sei noch angemerkt, dass es sich dabei stets um Turniere mit je 3 Teilnehmern handelte, mit $\delta_1 = 0.75$, $\delta_2 = 0.20$ und $\delta_3 = 0.05$.

Kürzel	**Start-population**	**Selektions-verfahren**	**Kreuzungs-verfahren**	**Mutations-rate**
R1	All Random	Fitnessproportional	Two Point CO	0.01
R2	All Random	Fitnessproportional	Two PointCO	0.05
R3	All Random	Fitnessproportional	Uniform CO	0.01
R4	All Random	Fitnessproportional	Uniform CO	0.05
R5	All Random	Turnierverfahren	Two Point CO	0.01
R6	All Random	Turnierverfahren	Two Point CO	0.05
R7	All Random	Turnierverfahren	Uniform CO	0.01
R8	All Random	Turnierverfahren	Uniform CO	0.05
A1	5-Added	Fitnessproportional	Two Point CO	0.01
A2	5-Added	Fitnessproportional	Two Point CO	0.05
A3	5-Added	Fitnessproportional	Uniform CO	0.01
A4	5-Added	Fitnessproportional	Uniform CO	0.05
A5	5-Added	Turnierverfahren	Two Point CO	0.01
A6	5-Added	Turnierverfahren	Two Point CO	0.05
A7	5-Added	Turnierverfahren	Uniform CO	0.01
A8	5-Added	Turnierverfahren	Uniform CO	0.05

Tabelle 7.2: Kürzel der Parametersets

7.1 Testproblem 1

Gegeben sei ein System mit $m = 5$ Stationen und $n = 10$ Ressourcen. Tabelle 7.3 zeigt die Leistungen der Ressourcen auf den einzelnen Stationen und Tabelle 7.4 zeigt die Bedarfe, die zur Verfügung stehende Arbeitszeit, sowie die minimale und maximale Anzahl an Ressourcen pro Station. Zudem sind in Tabelle 7.5 die Verteilungen und Puffer zwischen den einzelnen Stationen gegeben. In diesem Beispiel gibt es keine Initialbelegung, wodurch auch auf die Angabe der Umschichtungsmatrix verzichtet werden kann.

Tabelle 7.3: Leistungen für 7.1

$l_{s,r}$	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
s_1	14	0	18	23	0	0	22	17	0	0
s_2	20	19	0	28	15	0	20	0	22	21
s_3	32	0	26	24	27	0	22	29	29	0
s_4	10	10	10	10	10	10	10	10	10	0
s_5	0	0	0	54	60	0	0	0	0	0

Tabelle 7.4: Stationseigenschaften für 7.1

	b_s	t_s	L_s	$R_{min,s}$	$R_{max,s}$
s_1	300	10	30	0	4
s_2	300	10	30	1	3
s_3	300	10	30	0	4
s_4	300	10	30	1	8
s_5	500	10	50	1	5

Tabelle 7.5: Verteilungs-, Puffer- und Pufferkapazitätsmatrix für Bsp. 7.1

	s_1			s_2			s_3			s_4			s_5		
	V	P	K	V	P	K	V	P	K	V	P	K	V	P	K
s_1	0	0	0	0	0	0	1	80	120	0	0	0	0	0	0
s_2	0	0	0	0	0	0	0	0	0	1	40	100	0	0	0
s_3	0	0	0	0	0	0	0	0	0	0	0	0	1	60	60
s_4	0	0	0	0	0	0	0	0	0	0	0	0	1	0	50
s_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

In dem in Abbildung 7.1 gezeigten Graphen, werden die Zusammenhänge der einzelnen

Stationen noch einmal anschaulich dargestellt. Für jeden Knoten ist in eckigen Klammern dessen erforderliche Mindestleistung angegeben.

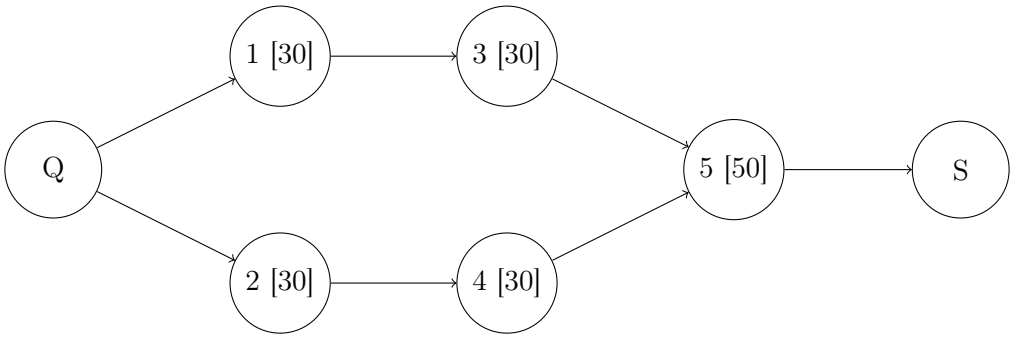


Abbildung 7.1: Graph für Beispiel 7.1

Dieses etwas kleinere und noch überschaubare Beispiel hat den Vorteil, dass sich dessen optimale Lösung in vertretbarer Zeit bestimmen lässt. Dies ermöglicht einen Vergleich der erzielten Lösungen mit eben jener optimalen Lösung, wodurch diese Lösungen wesentlich mehr Aussagekraft bekommen. Es soll hierdurch aufgezeigt werden, dass die angewandten genetischen Algorithmen grundsätzlich funktionieren und tatsächlich gute Lösungen finden. Die Ergebnisse der Testläufe für die verschiedenen Parameterkombinationen sind in Tabelle 7.6 zusammengefasst. Darin ist ersichtlich, dass bei diesem sehr kleinen Problem, unabhängig von den Parametern, fast immer nach wenigen Generationen die optimale Lösung gefunden wurde. Aufgrund der geringen Problemgröße, lassen sich für die verschiedenen Parameter noch kaum Schlussfolgerungen ziehen. Einzig die Mutationsrate hat einen klar erkennbaren Einfluss. So haben Parametersets mit der höheren Mutationsrate von 0,05 durchgehend eine bessere Durchschnittsfitness als die selben Parameter mit der geringeren Mutationsrate von 0,01. Des weiteren lieferten jene Testläufe, bei welchen die Startpopulation ein paar heuristisch erzeugte Individuen enthielt, tendenziell etwas bessere Ergebnisse als jene, bei welchen die Individuen der Startpopulation rein zufällig generiert wurden.

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
BFit	93	93	93	93	93	93	93	93
DFit	89,8	93	88,4	93	89,9	92,55	91,05	92,7
BGen	9	3	0	32	17	10	8	22
DGen	29	39,95	16,05	24,2	71	66,35	47,6	34,16
FGen	9	0	0	12	64	10	0	0
SGen	121	136	165	104	229	240	194	121
LZ	0,49	0,51	0,49	0,44	0,6	0,54	0,64	0,47

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**
BFit	93	93	93	93	93	93	93	93
DFit	91,8	93	91,1	93	91,2	93	90,9	92,55
BGen	22	0	0	1	0	0	1	0
DGen	66,05	41,3	70,35	66,3	73,5	50,75	54,75	35,2
FGen	0	0	0	1	0	0	0	0
SGen	233	140	270	200	287	181	157	150
LZ	0,54	0,46	0,61	0,5	0,59	0,51	0,54	0,48

Tabelle 7.6: Testergebnisse für Beispiel 7.1

7.2 Testproblem 2

Gegeben sei ein System mit $m = 7$ Stationen und $n = 16$ Ressourcen. Tabelle 7.7 zeigt die Leistungen der Ressourcen auf den einzelnen Stationen und Tabelle 7.8 zeigt die Bedarfe, die zur Verfügung stehende Arbeitszeit, sowie die minimale und maximale Anzahl an Ressourcen pro Station. Zudem sind in den Tabellen 7.9 und 7.10 die Verteilungen sowie die Umschichtungszeiten zwischen den einzelnen Stationen gegeben. In diesem Beispiel gibt es außerdem eine Initialbelegung, gegeben in Tabelle 7.11. Es sei angemerkt, dass es zwischen den Stationen keine Pufferspeicher gibt. Die grundlegende Struktur des Systems ist in Abbildung 7.2 dargestellt.

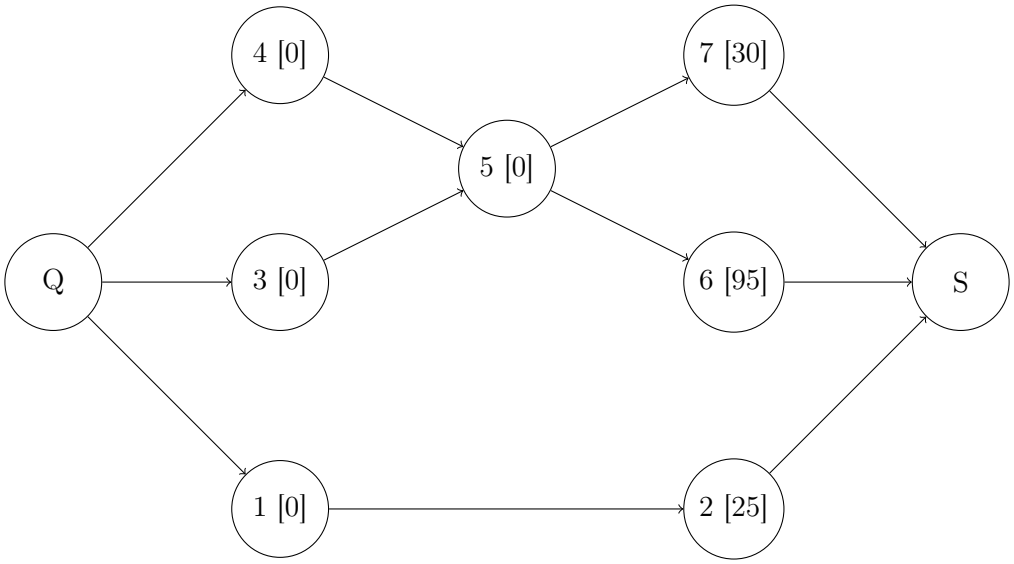


Abbildung 7.2: Graph für Beispiel 7.2

Dieses Beispiel soll eine in der Lagerlogistik häufig auftretende Situation darstellen. Dabei können die von der Quelle eintreffenden Güter entweder manuell (hier dargestellt durch den Weg Q-1-2-S) oder (teil-)automatisiert (Q-3/4-5-6/7-S) behandelt werden. Die automatisierten Prozesschritte erzielen klarerweise eine höhere Leistung und sollten daher bevorzugt werden. Allerdings müssen gewisse Güter, aufgrund unvorteilhafter Eigenschaften oder besonderer Vorsichtsmaßnahmen, manuell behandelt werden. Dies wird hier durch den Bedarf an Station s_2 dargestellt. s_5 stellt eine vollautomatisierte Arbeitstation dar, bei welcher Mitarbeiter nur Kontrolltätigkeiten zu vollführen haben, weshalb auf dieser Station alle Mitarbeiter die selbe Leistung erzielen. Idealerweise werden hier also jene Mitarbeiter eingesetzt, welche auf den anderen Stationen eher schwache Leistungen erzielen.

Tabelle 7.7: Leistungen für 7.2

$l_{s,r}$	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}
s_1	11	9	9	12	14	0	0	0	0	10	0	11	0	0	0	7
s_2	13	9	0	14	19	11	11	8	12	0	0	12	10	10	0	9
s_3	48	40	62	53	37	56	61	44	45	55	0	60	60	54	42	0
s_4	28	27	27	29	28	30	25	25	27	28	31	28	27	26	27	30
s_5	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70
s_6	44	54	51	55	49	0	0	0	0	55	53	51	55	44	41	51
s_7	24	22	24	24	29	19	18	25	22	27	31	28	26	26	21	32

Tabelle 7.8: Stationseigenschaften für 7.2

	b_s	t_s	L_s	$R_{min,s}$	$R_{max,s}$
s_1	0	10	0	1	10
s_2	250	10	25	1	10
s_3	0	10	0	0	3
s_4	0	10	0	1	6
s_5	0	10	0	2	5
s_6	950	10	95	2	10
s_7	300	10	30	0	5

Tabelle 7.9: Verteilungsmatrix für Bsp. 7.2

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_1	0	1	0	0	0	0	0
s_2	0	0	0	0	0	0	0
s_3	0	0	0	0	1	0	0
s_4	0	0	0	0	1	0	0
s_5	0	0	0	0	0	1	1
s_6	0	0	0	0	0	0	0
s_7	0	0	0	0	0	0	0

Tabelle 7.10: Umschichtungsmatrix für Bsp. 7.2

	s_1	s_2	s_3	s_4	s_5	s_6	s_7
s_1	0	0,1	0,25	0,25	0,25	0,25	0,25
s_2	0	0	0,25	0,25	0,25	0,25	0,25
s_3	0	0	0	0,1	0,1	0,15	0,15
s_4	0	0	0	0	0,1	0,15	0,15
s_5	0	0	0	0	0	0,1	0,1
s_6	0	0	0	0	0	0	0,05
s_7	0	0	0	0	0	0	0

Tabelle 7.11: Initialbelegung für 7.2

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}
s_1	s_1	s_1	s_2	s_2	s_3	s_3	s_4	s_4	s_5	s_5	s_5	s_6	s_6	s_7	s_7

Wie schon in Beispiel 7.1, wurden auch hier wieder je 20 Testläufe für jedes der in Tabelle 7.2 angegebenen Parametersets durchgeführt. Die Ergebnisse sind in Tabelle 7.12 zusammengefasst. Die beste heuristisch erzeugte Lösung kam auf einen Fitnesswert von 165.

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
BFit	211	209	211	209	209	210	208	209
DFit	163,9	188,45	164,65	194,6	155,35	192,9	165,5	193,0
BGen	487	360	261	554	301	452	185	797
DGen	659,55	844,65	745,35	791,25	550,15	966,85	708,5	920,5
FGen	33	26	10	121	74	233	25	330
SGen	1535	1882	1933	1812	1396	1824	1890	1873
LZ	1,67	1,76	1,56	1,70	1,41	2,05	1,65	1,98

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**
BFit	209	210	206	212	211	211	206	211
DFit	182,5	188,2	179,65	185,05	180,45	190,05	175,7	189,2
BGen	39	327	3	16	94	205	214	34
DGen	365,85	350,25	185,0	440,85	400,55	451	308,25	351,55
FGen	3	5	1	7	5	2	12	5
SGen	1363	822	775	1593	1555	1982	1088	1514
LZ	1,20	1,61	1,19	1,47	1,76	1,62	1,37	1,51

Tabelle 7.12: Testergebnisse für Beispiel 7.2

Aus den Ergebnissen dieses Beispiels lassen schon eindeutigere Schlüsse ziehen. Zunächst zeigt sich hier, dass mit allen Parametersets sehr gute Spitzenwerte (BFit) erzielt wurden, welche auch eindeutig besser sind als die beste heuristisch erzeugte Lösung von 165. Es zeigt sich außerdem, dass jene Parametersets mit einer höheren Mutationsrate durchschnittlich eindeutig bessere Lösungen erzielten. Dies ist darauf zurückzuführen,

dass durch den Reparaturalgorithmus einige Mutationen, welche zu ungültigen Individuen führten, sofort wieder aufgehoben werden. Eine zu geringe Mutationsrate führt dann dazu, dass der Algorithmus, aufgrund zu geringer genetischer Vielfalt, öfters in einem lokalen Optimum hängen bleibt. Auch ist hier schön erkennbar, dass bei geringerer Mutationsrate, jene Testläufe, bei welchen die Startpopulation heuristisch erzeugte Individuen enthielt (A1, A3, A5, A7), durchschnittlich eindeutig bessere Fitnesswerte erzielten.

Betrachtet man die Ergebnisse für die Parametersets A1-A8, so fällt auch auf, dass unter den 20 Testläufen pro Set immer mindestens einer enthalten war, welcher ab einer sehr frühen Generation (hier zwischen 1-12) lange keine Verbesserung mehr erzielen konnte und daher nach etwas mehr als 200 Generationen bereits terminierte. Hier könnte es sinnvoll sein, die Kriterien für das terminieren etwas lockerer zu gestalten. Es könnte allerdings auch bedeuten, dass eine noch höhere Mutationsrate hilfreich sein könnte. Insgesamt erzielten die Parametersets A1-A8 gleich gute Spitzenwerte wie die Sets R1-R8, brauchten dafür aber merklich weniger Generationen. Dies unterschtreicht die positive Auswirkung einer guten Startpopulation auf das restliche Verfahren.

7.3 Testproblem 3

Gegeben sei ein System mit 15 Stationen und 50 Ressourcen. Die erbringbaren Leistungen der Ressourcen sind in Tabelle 7.13 gegegeben. Tabelle 7.14 zeigt die Eigenschaften der Stationen und die Tabellen 7.15, 7.16 sowie 7.17 zeigen die Verteilungen, die Initialbelegung sowie die Umschichtungszeiten zwischen den Stationen. Die grundlegende Struktur des Systems ist in Abbildung 7.3 gegeben.

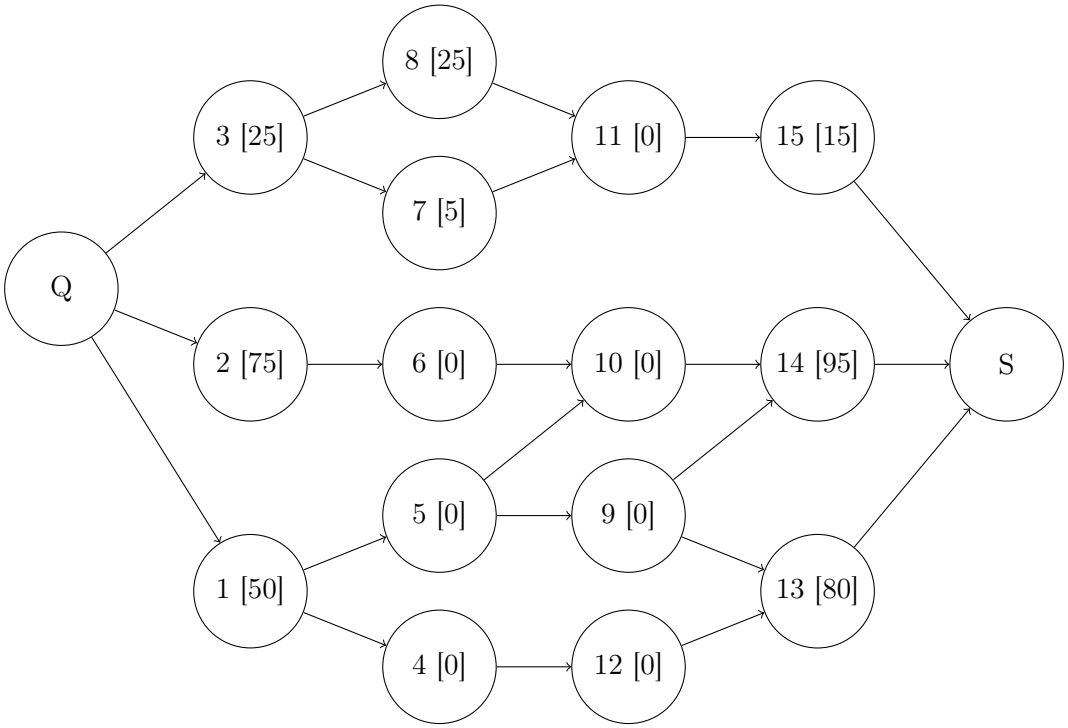


Abbildung 7.3: Graph für Beispiel 7.3

Tabelle 7.13: Leistungen für 7.3

$l_{s,r}$	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
r_1	0	0	0	0	0	35	0	0	0	80	0	0	38	0	8
r_2	25	0	9	0	0	35	10	10	56	80	0	0	0	64	8
r_3	0	0	10	0	23	34	10	0	52	80	14	0	41	55	7
r_4	27	40	10	24	0	35	10	12	0	0	14	0	40	56	9
r_5	0	0	0	26	26	0	0	15	50	80	0	24	0	59	7
r_6	25	40	0	26	0	0	0	12	0	0	16	0	0	62	8
r_7	0	0	0	0	0	0	0	0	42	80	18	25	0	0	9
r_8	27	37	10	28	0	0	10	0	0	80	15	27	0	53	8
r_9	21	0	0	0	27	0	0	13	50	0	0	0	34	59	8
r_{10}	0	0	0	0	0	35	10	0	60	80	0	0	0	0	8
r_{11}	23	0	10	0	27	37	0	11	43	0	0	24	47	0	8
r_{12}	21	40	8	0	0	36	0	0	0	0	0	0	38	63	7
r_{13}	27	0	10	0	26	39	0	0	51	0	0	0	39	0	8

r_{14}	25	46	0	23	0	0	0	14	0	0	0	0	44	0	8
r_{15}	0	0	9	26	32	35	0	0	54	0	0	0	43	60	7
r_{16}	0	0	0	0	0	0	0	0	0	0	15	23	36	0	7
r_{17}	31	34	10	28	0	0	0	0	0	0	0	24	40	0	7
r_{18}	20	0	0	0	23	0	0	0	54	0	0	23	0	0	7
r_{19}	0	43	8	24	28	0	10	11	0	0	0	0	39	56	8
r_{20}	0	44	10	23	0	0	0	0	47	0	0	0	0	63	8
r_{21}	25	0	0	0	0	0	0	0	0	80	0	0	0	49	8
r_{22}	0	0	11	0	0	0	0	14	0	0	0	0	0	60	8
r_{23}	0	0	10	0	0	0	10	0	0	80	0	24	48	0	8
r_{24}	0	38	0	0	0	0	0	0	0	0	0	29	0	0	8
r_{25}	25	30	0	26	0	36	10	0	0	0	0	0	0	0	8
r_{26}	26	42	0	23	0	0	10	0	58	80	16	0	42	0	9
r_{27}	26	42	0	0	0	0	10	0	0	0	0	0	38	0	8
r_{28}	27	47	9	26	0	35	0	0	0	0	0	0	0	0	7
r_{29}	25	49	10	26	0	0	0	0	0	0	0	23	0	0	8
r_{30}	0	43	12	0	27	34	10	0	0	0	0	0	0	67	8
r_{31}	0	0	0	0	27	0	0	0	0	80	0	0	43	58	8
r_{32}	0	0	10	0	23	0	10	0	0	0	0	0	0	0	8
r_{33}	0	0	0	22	29	33	0	0	0	80	0	0	0	55	8
r_{34}	0	0	0	0	0	33	0	0	52	0	0	0	45	0	8
r_{35}	0	34	14	27	0	0	0	0	0	0	0	23	0	0	9
r_{36}	25	39	8	0	0	0	10	0	0	0	11	0	0	67	10
r_{37}	0	0	0	29	0	0	0	10	40	0	17	0	0	61	8
r_{38}	0	0	13	29	0	0	10	0	0	80	0	0	40	60	8
r_{39}	30	42	0	29	23	34	0	13	0	80	0	0	0	0	8
r_{40}	0	0	12	0	0	33	0	13	49	80	0	0	0	0	7
r_{41}	0	42	0	27	31	35	0	14	0	0	15	25	0	0	8
r_{42}	0	0	10	0	0	37	10	0	0	0	15	24	0	54	8
r_{43}	25	0	0	0	0	0	10	12	0	80	0	23	34	56	8
r_{44}	25	0	9	0	26	0	0	0	64	80	0	0	0	68	7
r_{45}	0	0	0	0	0	0	10	0	0	0	14	28	42	0	8
r_{46}	29	0	0	26	0	0	0	12	0	0	0	0	42	0	7
r_{47}	17	0	9	19	0	0	10	12	0	0	0	20	0	0	8
r_{48}	0	0	7	0	0	33	10	0	0	0	15	0	40	57	8
r_{49}	26	0	12	26	0	37	10	0	0	80	0	0	0	64	8

r_{50}	25	0	8	0	0	34	10	0	0	80	14	0	40	0	9

Tabelle 7.14: Stationseigenschaften für 7.3

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
b_s	500	750	250	0	0	0	50	250	0	0	0	0	800	950	150
t_s	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
L_s	50	75	25	0	0	0	5	25	0	0	0	0	80	95	15
$R_{min,s}$	2	0	1	0	0	2	1	3	1	0	0	3	4	2	2
$R_{max,s}$	8	10	6	5	5	10	7	4	4	10	10	15	9	5	5

Tabelle 7.15: Verteilungsmatrix für Bsp. 7.3

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
s_1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
s_2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
s_3	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
s_4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
s_5	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
s_6	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
s_7	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
s_8	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
s_9	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
s_{10}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
s_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
s_{12}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
s_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
s_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabelle 7.16: Initialbelegung für 7.3

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}
s_6	s_1	s_3	s_1	s_4	s_{11}	s_9	s_{11}	s_8	s_6	s_5	s_2	s_3	s_2	s_3	s_{12}

r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}	r_{30}	r_{31}	r_{32}
s_2	s_5	s_2	s_4	s_{10}	s_8	s_7	s_{12}	s_6	s_7	s_7	s_{15}	s_{12}	s_5	s_{13}	s_3

r_{33}	r_{34}	r_{35}	r_{36}	r_{37}	r_{38}	r_{39}	r_{40}	r_{41}	r_{42}	r_{43}	r_{44}	r_{45}	r_{46}	r_{47}	r_{48}
s_{14}	s_9	s_{12}	s_{10}	s_8	s_{13}	s_{15}	s_{10}	s_{12}	s_{12}	s_{13}	s_{15}	s_{13}	s_{15}	s_{12}	s_{14}

r_{49}	r_{50}
s_{14}	s_{13}

Tabelle 7.17: Umschichtungsmatrix für Bsp. 7.3

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
s_1	0	0.2	0.2	0.2	0.2	0.3	0.4	0.3	0.3	0.4	0.3	0.3	0.4	0.3	0.3
s_2	0	0	0.3	0.3	0.2	0.3	0.1	0.3	0.2	0.2	0.2	0.4	0.3	0.3	0.4
s_3	0	0	0	0.2	0.4	0.1	0.3	0.3	0.2	0.2	0.3	0.3	0.3	0.1	0.2
s_4	0	0	0	0	0.3	0.2	0.2	0.4	0.4	0.3	0.1	0.3	0.3	0.2	0.2
s_5	0	0	0	0	0	0.3	0.2	0.2	0.1	0.2	0.3	0.4	0.2	0.2	0.2
s_6	0	0	0	0	0	0	0.4	0.3	0.3	0.2	0.3	0.1	0.1	0.3	0.4
s_7	0	0	0	0	0	0	0	0.2	0.4	0.3	0.2	0.2	0.2	0.2	0.4
s_8	0	0	0	0	0	0	0	0	0.2	0.1	0.2	0.4	0.2	0.1	0.3
s_9	0	0	0	0	0	0	0	0	0	0.4	0.3	0.1	0.1	0.3	0.1
s_{10}	0	0	0	0	0	0	0	0	0	0	0.3	0.1	0.2	0.2	0.2
s_{11}	0	0	0	0	0	0	0	0	0	0	0	0.4	0.2	0.2	0.3
s_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0.3	0.2	0.3
s_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0.2
s_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2
s_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Beispiel 7.1 und Beispiel 7.2 enthielten nur eine kleine bis mittelgroße Anzahl an Stationen

und Ressourcen. Da man für Probleme dieser Größe noch mit überschaubarem Aufwand eine optimale Lösung bestimmen kann, konnte anhand dieser Beispiele gezeigt werden, dass durch den genetischen Algorithmus grundsätzlich gute Lösungen gefunden werden. Für dieses Beispiel, welches wesentlich größer ist als die beiden vorherigen, lässt sich eine solche optimale Lösung nicht mehr so einfach bestimmen. Da es für jede Ressource im Durchschnitt ca. 8 Stationen gibt, welcher diese zugeteilt werden kann (siehe Tabelle 7.13) gäbe es, ohne die Beschränkungen für minimal/maximal zulässige Ressourcen pro Station, für dieses System ungefähr $8^{50} = 1,4 * 10^{45}$ verschiedene Zuteilungen. Je stärker die Beschränkungen auf den einzelnen Stationen sind, desto stärker sinkt auch die Anzahl der möglichen Zuteilungen, doch trotz der gegebenen Beschränkungen ist die Anzahl der Möglichkeiten in diesem Beispiel immer noch so groß, dass das Ermitteln der optimalen Lösung in akzeptabler Zeit nicht durchfürbar ist.

In Tabelle 7.18 sind wieder die Ergebnisse sämtlicher Testläufe zusammengefasst. Zum Vergleich wird wieder die beste heuristisch erzeugte Lösung verwendet, welche eine Leistung von 294 Einheiten pro Stunde erreichte.

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
BFit	327	333	333	314	331	321	331	330
DFit	219,6	319,6	273,4	288,2	271,4	293,8	264,7	320,0
BGen	1107	1537	1296	899	1927	941	766	1510
DGen	1517,2	1618,0	1447,2	1351	1275,2	1389,6	1288,5	1700,2
FGen	1107	1411	493	867	571	941	621	1510
SGen	1962	1807	1938	1950	1927	1822	1857	1948
LZ	18,27	19,77	19,73	20,18	21,39	20,99	18,77	20,26

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**
BFit	330	320	329	311	340	322	331	326
DFit	321,8	308,6	322,2	308,8	322	315,4	321,2	311,2
BGen	865	1099	1167	821	1757	1601	1347	1066
DGen	1340	1094,4	1720	1190,2	1642	1498,8	1620,6	1546,4
FGen	865	296	1167	821	1125	148	1327	1066
SGen	1946	1762	1934	1610	1975	1928	1895	1915
LZ	21,25	20,94	21,33	21,01	21,87	20,83	22,05	21,22

Tabelle 7.18: Testergebnisse für Beispiel 7.3

Grundsätzlich lässt sich aus den erzielten Ergebnissen ablesen, dass die genetischen Algorithmen, auch bei diesem wesentlich größeren Problem, zumindest sehr gute Lösungen liefern. Da die optimale Lösung zum Vergleich nicht zur Verfügung steht, lässt sich auch nicht sagen, wie nahe die beste erreichte Lösung von 340 diesem Optimum gekommen ist, jedoch wurde die beste heuristisch erzeugte Lösung, je nach Parameterset, in jedem Testlauf um ca. 10-15% überschritten.

Abbildung 7.4 zeigt die Entwicklung der durchschnittlichen Fitness, sowie der besten Fitness pro Generation. Dabei ist zu Beginn der für genetische Algorithmen typische starke Anstieg zu beobachten, welcher nach den ersten 100 Generationen abzuflachen beginnt. Nachdem in den nächsten 200-300 Generationen noch eine relativ konstante Steigung im Spitzenwert zu erkennen ist, wird irgendwann der Punkt erreicht, ab welchem die Kurve einen nahezu konstanten Wert einnimmt, welcher nur noch sporadisch kleine Verbesserungen erlebt. Dass sich der Spitzenwert so stark von der durchschnittlichen Fitness absetzt ist darauf zurückzuführen, dass unter den 30 Individuen jeder Population, immer solche enthalten sind, die zwar eigentlich über einen sehr guten Fitnesswert verfügen, aber, aufgrund des Nichterfüllens der vorgeschriebenen Mindestleistungen einer oder mehrerer Stationen, zur Strafe 50% ihrer Fitness einbüßen.

Betrachtet man dagegen den in Abbildung 7.5 dargestellten Verlauf für das Parameterset A8, welches im Gegensatz zu R8 von Beginn an das heuristisch erzeugte Individuum mit der Leistung von 294 enthällt, so sieht man schön die Auswirkung der Startpopulation auf den Algorithmus. Während der Spitzenwert von Anfang an schon sehr hoch ist und nur hin und wieder schwach ansteigt, steigt die durchschnittliche Fitness innerhalb weniger Generationen extrem stark an, bevor sie sich auf einem halbwegs konstaten Niveau einpendelt. Dies zeigt, wie sich das in der Startpopulation enthaltene, äußerst fitte Individuum, sehr schnell gegenüber den anderen Individuen durchsetzt. Der Algorithmus kommt also auf diese Weise schon schneller zu guten Lösungen. Je mehr Genertionen der Algorithmus läuft, desto stärker gleichen sich die Kurven allerdings wieder an.

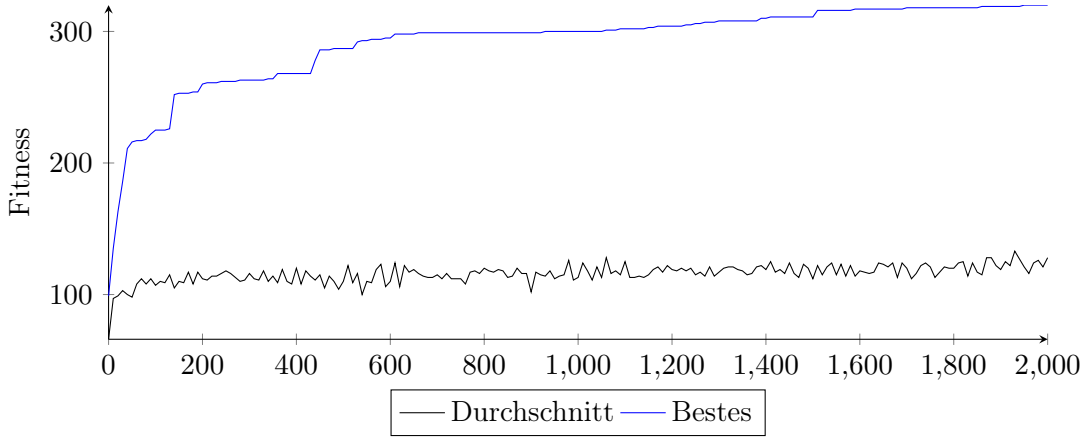


Abbildung 7.4: Generationenverlauf für Parameterset R8 (Testbeispiel 7.3)

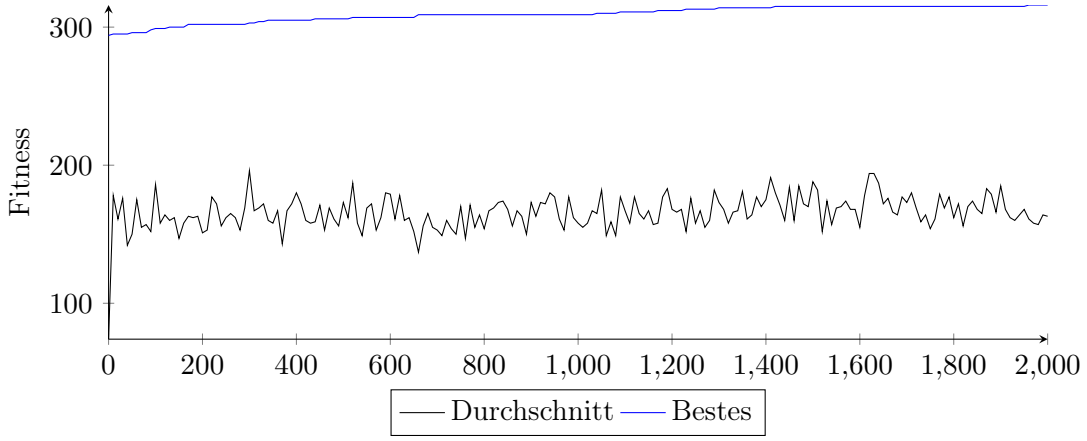


Abbildung 7.5: Generationenverlauf für Parameterset A8 (Testbeispiel 7.3)

7.4 Testproblem 4

Gegeben sei ein System mit 17 Stationen und 80 Ressourcen. Sämtliche relevanten Informationen sind den Tabellen 7.19 und 7.20 zu entnehmen. Bei diesem System gibt es keine Restriktionen bezüglich der minimalen und maximalen Anzahl an Ressourcen pro Station. Es gibt also keine Station, welcher eine Ressource zugewießen werden muss und gleichzeitig könnten jeder Station alle 80 Ressourcen zugewießen werden. Tabelle 7.19

kann man entnehmen, dass jede Ressource auf durchschnittlich 10 Stationen eingesetzt werden kann, wodurch es insgesamt ca. 10^{80} Möglichkeiten gibt, die Ressourcen auf den Stationen zu verteilen. Aufgrund der Größe dieses Systems, wurden für dieses Problem bis zu 5000 Generationen zugelassen. Außerdem wurden die Testläufe, aufgrund der Größe der Chromosome, statt mit den oben angeführten Mutationsraten von 0.01 und 0.05, mit verringerten Mutationsraten von 0,005 und 0,01 durchgeführt, da bei einer Mutationsrate von 0,05 an nahezu jedem Individuum multiple Mutationen durchgeführt worden wären, wodurch der Verlauf des Algorithmus zu stark vom Zufall abhängig gewesen wäre. Die grundlegende Struktur des Systems ist in Abbildung 7.6 dargestellt.

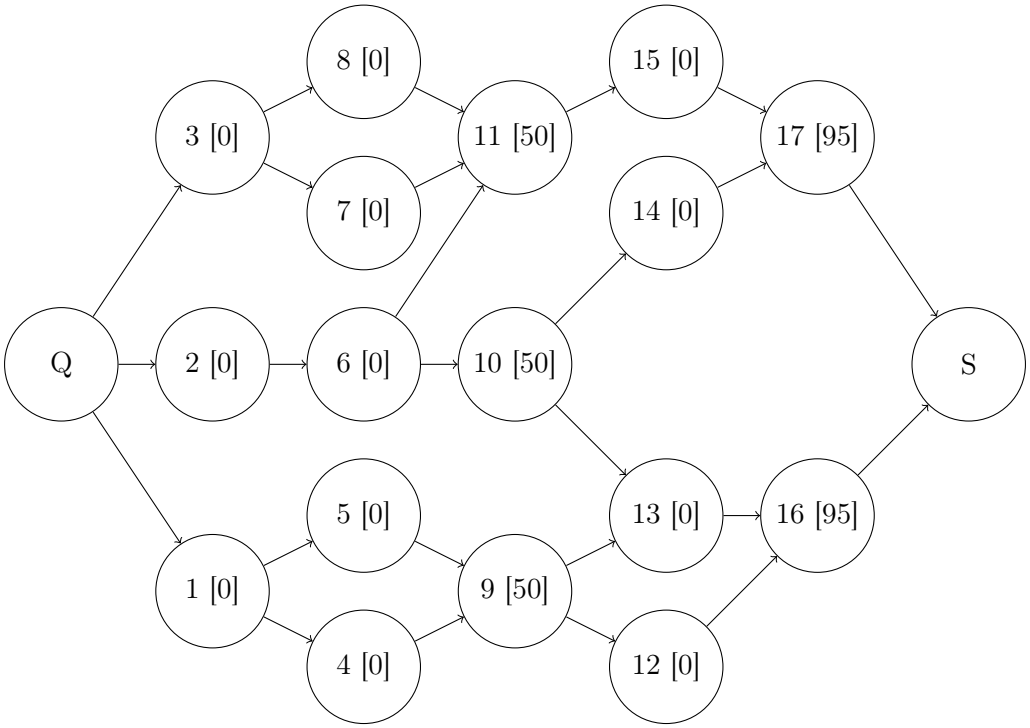


Abbildung 7.6: Graph für Beispiel 7.4

Tabelle 7.19: Leistungen für 7.4

$l_{s,r}$	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}
r_1	10	45	28	8	0	0	0	38	13	80	80	24	15	0	0	36	50
r_2	0	0	0	0	7	38	34	38	14	0	80	26	0	0	49	24	50
r_3	0	42	30	8	12	0	0	46	15	0	80	24	13	43	47	0	65
r_4	11	0	0	8	6	40	0	38	14	0	80	0	15	0	48	17	52

r_5	10	41	0	0	10	46	0	39	9	80	0	26	0	42	48	27	48
r_6	10	44	26	7	7	0	0	33	14	80	80	30	16	40	0	12	44
r_7	10	0	28	8	0	41	32	46	12	80	80	0	0	38	51	27	37
r_8	9	0	30	0	11	0	37	34	14	80	0	24	15	38	48	27	55
r_9	10	45	29	0	0	0	0	0	0	0	0	22	0	39	0	0	0
r_{10}	10	38	30	0	10	43	33	31	0	80	80	25	12	49	49	20	53
r_{11}	10	38	34	0	0	41	37	39	0	80	0	24	13	0	0	26	56
r_{12}	11	0	28	8	9	0	32	0	16	80	80	0	15	39	0	21	46
r_{13}	11	46	0	8	0	41	32	44	13	0	0	0	16	0	0	35	58
r_{14}	0	0	28	7	9	38	35	33	14	0	80	24	0	38	49	13	0
r_{15}	11	0	29	9	9	0	31	40	14	80	80	24	0	0	51	18	35
r_{16}	10	0	27	8	8	39	31	0	16	80	80	18	0	49	50	25	50
r_{17}	0	41	0	0	9	39	32	40	15	80	80	24	14	41	0	20	0
r_{18}	11	42	33	7	0	42	0	0	0	80	80	26	0	28	46	24	0
r_{19}	9	44	30	8	0	37	29	43	12	80	80	22	14	46	43	24	52
r_{20}	10	42	30	8	0	40	31	0	0	80	0	20	20	45	48	31	0
r_{21}	0	0	30	0	9	0	34	36	0	80	0	0	12	33	48	24	51
r_{22}	10	38	34	0	9	40	0	34	15	0	80	25	0	34	46	27	48
r_{23}	10	0	31	0	10	38	34	47	15	80	0	21	14	0	0	0	48
r_{24}	0	42	28	8	0	41	39	40	0	80	80	23	0	0	48	17	53
r_{25}	10	0	28	9	10	0	31	39	19	80	80	28	15	44	48	31	0
r_{26}	11	38	0	8	9	42	31	47	14	80	80	28	0	38	0	20	0
r_{27}	11	35	31	0	0	0	0	50	14	80	80	27	11	43	48	0	42
r_{28}	11	34	0	8	11	45	35	0	13	80	0	0	0	38	49	29	53
r_{29}	10	40	0	0	12	0	31	0	14	0	80	26	17	38	47	17	0
r_{30}	9	33	28	8	11	0	32	0	14	80	80	27	20	0	0	7	52
r_{31}	0	41	29	0	0	0	30	38	14	80	80	25	14	0	0	29	56
r_{32}	10	41	32	8	11	0	34	40	15	80	80	23	0	41	48	18	39
r_{33}	9	40	30	7	9	39	0	40	0	0	0	23	15	0	48	27	44
r_{34}	0	40	0	9	8	49	0	0	0	80	80	20	13	34	47	24	49
r_{35}	9	45	27	8	10	0	0	27	0	0	80	28	16	40	51	13	0
r_{36}	10	0	29	8	10	40	28	26	14	0	0	28	16	43	48	0	0
r_{37}	10	0	24	0	0	41	32	31	0	80	80	24	0	46	50	0	49
r_{38}	9	44	32	0	0	0	31	33	15	0	80	0	11	51	0	0	41
r_{39}	10	0	29	8	11	0	31	38	14	80	80	0	0	0	0	34	48
r_{40}	10	0	33	8	10	40	32	38	15	80	0	28	14	41	48	0	54

r_{41}	10	0	30	9	11	39	32	45	0	80	80	22	14	43	48	24	48
r_{42}	12	38	0	9	9	0	32	24	15	0	0	23	16	0	47	20	41
r_{43}	10	0	0	7	10	37	0	0	0	80	80	0	17	36	0	20	0
r_{44}	0	38	30	8	6	43	31	32	13	80	0	19	16	40	48	29	46
r_{45}	0	41	28	8	9	37	0	0	15	0	80	0	15	33	47	19	53
r_{46}	10	39	0	9	0	35	0	0	16	80	80	27	0	46	0	24	55
r_{47}	12	0	31	8	0	43	29	43	0	0	0	24	0	42	48	23	53
r_{48}	10	40	26	8	7	40	33	0	0	80	80	0	14	42	48	22	48
r_{49}	10	0	26	8	9	0	34	38	14	80	80	0	13	45	0	0	0
r_{50}	0	0	36	8	10	0	0	43	0	80	0	0	0	36	47	0	47
r_{51}	10	34	31	0	12	34	30	41	16	80	80	0	0	37	46	21	50
r_{52}	0	0	31	7	0	39	32	0	13	80	80	0	16	0	0	30	49
r_{53}	0	0	33	9	10	39	32	33	14	0	80	20	14	0	48	0	0
r_{54}	10	41	0	8	0	44	31	34	13	0	80	26	14	42	48	27	48
r_{55}	10	35	0	8	11	41	31	38	14	0	80	0	11	46	0	0	0
r_{56}	10	50	30	0	8	35	33	0	19	80	80	0	13	45	48	31	47
r_{57}	10	42	24	10	9	0	0	39	14	80	80	21	0	0	47	0	0
r_{58}	10	38	35	9	10	44	31	0	0	80	80	27	15	48	48	21	44
r_{59}	8	40	34	0	9	41	30	41	14	80	80	0	16	36	0	25	0
r_{60}	0	38	0	8	0	30	30	33	14	0	80	24	0	41	46	27	50
r_{61}	10	42	28	8	9	43	32	0	0	80	0	0	0	47	48	25	48
r_{62}	10	0	30	0	9	40	25	23	14	80	80	0	17	0	0	0	54
r_{63}	0	41	30	0	9	35	0	38	14	80	0	0	15	0	48	25	56
r_{64}	0	0	32	8	0	31	32	40	14	0	80	25	17	47	48	21	48
r_{65}	10	35	0	6	12	39	0	47	16	80	0	0	12	0	51	27	52
r_{66}	0	40	28	8	9	39	31	33	16	0	0	24	0	41	47	35	36
r_{67}	0	35	29	8	6	0	32	38	0	80	80	23	0	41	48	17	0
r_{68}	0	45	0	0	9	44	0	0	15	80	0	0	16	40	0	24	0
r_{69}	0	0	30	9	9	41	30	49	15	0	80	24	13	0	49	17	0
r_{70}	9	40	34	0	11	36	0	44	15	80	80	24	20	40	48	0	45
r_{71}	0	41	29	8	13	0	31	38	13	80	80	24	0	41	48	32	41
r_{72}	11	46	30	8	10	0	37	39	14	80	0	29	0	36	48	26	50
r_{73}	9	39	22	8	0	42	26	0	12	0	80	26	13	51	48	20	45
r_{74}	10	0	28	8	0	37	0	0	17	80	0	20	14	0	0	21	0
r_{75}	10	43	0	0	12	39	29	43	16	80	80	20	13	0	48	19	0
r_{76}	10	39	0	9	9	38	32	33	15	80	80	26	11	44	0	18	59

r_{77}	10	0	0	8	8	48	35	0	15	80	0	0	14	0	0	0	44
r_{78}	10	43	26	8	9	40	30	35	16	80	80	20	0	42	46	0	55
r_{79}	10	42	31	0	11	40	0	0	0	80	80	0	0	40	49	15	55
r_{80}	10	38	28	8	9	36	36	37	0	80	80	25	14	40	47	27	0

Tabelle 7.20: Stationseigenschaften für 7.4

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}	s_{17}
b_s	0	0	0	0	0	0	0	0	500	500	500	0	0	0	0	950	950
t_s	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
L_s	0	0	0	0	0	0	0	0	50	50	50	0	0	0	0	95	95
$R_{min,s}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$R_{max,s}$	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80	80

Die Ergebnisse sämtlicher Testläufe sind in Tabelle 7.21 zusammengefasst.

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
BFit	581	575	570	583	567	563	575	568
DFit	560,6	565,2	564,0	561,2	555,0	553,0	562,6	548,6
BGen	3641	3845	3974	2456	2606	1990	3144	2784
DGen	4145,7	3922,9	3513,6	3675,8	2979,4	3951,8	3367,4	2792,2
FGen	2599	1670	2511	2456	2305	1990	2002	995
SGen	4974	4898	4771	4997	3869	4901	4348	4472
LZ	66,99	64,48	72,36	71,56	68,87	68,97	67,86	69,78

	A1	**A2**	**A3**	**A4**	**A5**	**A6**	**A7**	**A8**
BFit	564	582	576	582	583	571	574	590
DFit	543,9	561,7	513,5	551,7	429,3	465,5	491,5	409,2
BGen	4953	4038	1350	4421	2046	1209	1616	187
DGen	4255,5	3684,6	3260	3877,5	3415,8	3058,2	4285,8	2900,4
FGen	2580	1064	618	922	784	1209	1616	187
SGen	4953	4963	4935	4910	4904	4783	4974	4975
LZ	68,27	65,71	73,17	71,21	58,24	60,60	66,13	60,91

Tabelle 7.21: Testergebnisse für Beispiel 7.4

Da die optimale Lösung als Vergleichswert auch hier nicht zur Verfügung steht, wird wiederum die Veränderung der Fitnesswerte über die Generationen betrachtet. Abbildung 7.7 zeigt sowohl die durchschnittliche als auch die beste Fitness pro Generation für die Testläufe mit Parameterset R1. Es sei erwähnt, dass fast alle Parametersets recht ähnliche Ergebnisse erzielten und der Verlauf von R1 daher represäntativ für all diese Parametersets ist. Über diese sehr große Anzahl von Generationen sieht man wieder sehr schön die für genetische Algorithmen typische Kurve, welche ‚nach einem zunächst starken Anstieg, langsam abflacht und anschließend nahezu konstant weiter geht und nur sporadisch kleine Verbesserungen erfährt.

Die Haupterkenntnis, welche dieses Beispiel zusätzlich zu den vorherigen noch bietet, ist, dass selbst bei einem Problem dieser Größe und einer Weiterentwicklung der Individuen über 5000 Generationen, eine Laufzeit von etwa einer Minute erzielt werden konnte. Dies lässt darauf schließen, dass mit dem vorhanden Modell auch noch weit größere Probleme in annehmbarer Zeit behandelt werden können.

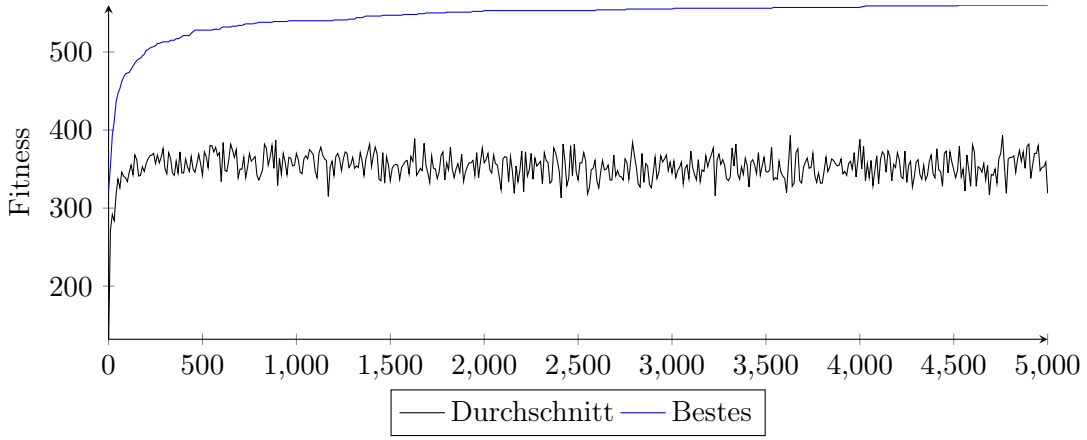


Abbildung 7.7: Generationenverlauf für Parameterset R1 (Testbeispiel 7.4)

8 Resümee

In dieser Arbeit wurde eine spezielle Form eines Human Ressource Allocation Problems vorgestellt und eine Methode vorgeschlagen um dieses zu Modellieren. Des weiteren wurde, mittels einer abgewandelten Version des Algorithmus von Ford und Fulkersion, eine Möglichkeit entwickelt, um die Qualität einer Lösung, also einer Zuteilung der Ressourcen auf die verschiedenen Stationen, eines solchen Problems zu bestimmen. Um unter den unzähligen möglichen Zuteilungen eine möglichst gute zu finden, wurden genetische Algorithmen eingeführt und es wurde gezeigt, wie sich die Lösungen des vorliegenden Problems als Individuen eines solchen modellieren lassen.

Bezüglich der Genetischen Algorithmen stellte sich heraus, dass, für das in dieser Arbeit betrachtete Problem, die Wahl des Kreuzungsverfahrens auf den Verlauf des Algorithmus keinen wesentlichen Einfluss hat. Auch bei den verschiedenen Selektionsverfahren haben sich keine gravierenden Unterschiede gezeigt, lediglich eine zu elitäre Auswahl der Individuen sollte vermieden werden. Den wesentlichsten Einfluss auf den Verlauf des Algorithmus und die Qualität der gefundenen Lösung hat die Auswahl der Mutationsrate. Da die genetische Vielfalt einer Population durch die Anwendung des Reperaturalgorithmus oft abnimmt, ist eine etwas höhere Mutationsrate notwendig, um eben diese Vielfalt ausreichend zu erhalten.

Insgesamt konnte in der Arbeit gezeigt werden, dass die angewandten genetischen Algorithmen definitiv geeignet sind, um für das beschriebene Problem sehr gute Lösungen zu finden. Die Laufzeit, welche selbst bei großen Problemen noch sehr gering bleibt, sorgt außerdem dafür, dass das Verfahren auch für die praktische Anwendung attraktiv ist.

Literatur

[1] Leonardo Borba und Marcus Ritt. „Exact and Heuristic Methods for the Assembly Line Worker Assignment and Balancing Problem“. In: *arXiv:1308.0299 [cs]* (01. 08. 2013). arXiv: 1308.0299

[2] Sana Bouajaja und Najoua Dridi. „A survey on human resource allocation problem and its applications“. In: *Operational Research* 17.2 (07/2017), S. 339–369

[3] E.K. Burke, S. Gustafson und G. Kendall. „Diversity in Genetic Programming: An Analysis of Measures and Correlation With Fitness“. In: *IEEE Transactions on Evolutionary Computation* 8.1 (02/2004), S. 47–62

[4] GB Dantzig. *GB,(1947).“Maximization of a linear function of variables subject to linear inequalities”, 1939 also published pp. 339–347 in TC Koopmans (ed.): Activity Analysis of Production and Allocation*. 1951

[5] S Daskalaki, T Birbas und E Housos. „An integer programming formulation for a case study in university timetabling“. In: *European Journal of Operational Research* 153.1 (02/2004), S. 117–135

[6] E.A. Dinic. „Algorithm for solution of a problem of maximum flow in a network with power estimation“. In: *Soviet Mathematic Doklady* 11 (1970), S. 1277–1280

[7] Jack Edmonds und Richard M. Karp. „Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems“. In: *Journal of the ACM* 19.2 (01. 04. 1972), S. 248–264

[8] Jack Edmonds und Richard M. Karp. „Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems“. In: *Journal of the ACM* 19.2 (01. 04. 1972), S. 248–264

[9] L. R. Ford und D. R. Fulkerson. „Maximal Flow Through a Network“. In: *Canadian Journal of Mathematics* 8 (1956), S. 399–404

[10] Lester Randolph Ford und D. R Fulkerson. *Flows in Networks*. OCLC: 973400412. 2016

[11] John Grefenstette. „Optimization of Control Parameters for Genetic Algorithms“. In: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1 (01/1986), S. 122–128

[12] Zhengyuan Jia und Lihua Gong. „Multi-criteria Human Resource Allocation for Optimization Problems Using Multi-objective Particle Swarm Optimization Algorithm“. In: *2008 International Conference on Computer Science and Software Engineering*. 2008 International Conference on Computer Science and Software Engineering. Wuhan, China: IEEE, 2008, S. 1187–1190

[13] J. Kennedy und R. Eberhart. „Particle swarm optimization“. In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. ICNN'95 - International Conference on Neural Networks. Bd. 4. Perth, WA, Australia: IEEE, 1995, S. 1942–1948

[14] Bernhard Korte und Jens Vygen. *Kombinatorische Optimierung: Theorie und Algorithmen*. Übers. von Ulrich Brenner und Rabe von Randow. 3. Auflage. Masterclass. OCLC: 1041490532. Berlin: Springer Spektrum, 2018. 732 S.

[15] H. W. Kuhn. „The Hungarian method for the assignment problem“. In: *Naval Research Logistics Quarterly* 2.1 (03/1955), S. 83–97

[16] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. OCLC: 1086436717. 1996

[17] Cristóbal Miralles et al. „Branch and bound procedures for solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work centres for Disabled“. In: *Discrete Applied Mathematics* 156.3 (02/2008), S. 352–367

[18] Melanie Mitchell. *An introduction to genetic algorithms*. 7. print. Complex adaptive systems. OCLC: 256769418. Cambridge, Mass., 2001. 209 S.

[19] Heinz Mühlenbein. „How Genetic Algorithms Really Work: Mutation and Hillclimbing.“ In: 01/1992, S. 15–26

[20] James Munkres. „Algorithms for the Assignment and Transportation Problems“. In: *Journal of the Society for Industrial and Applied Mathematics* 5.1 (03/1957), S. 32–38

[21] David W. Pentico. „Assignment problems: A golden anniversary survey“. In: *European Journal of Operational Research* 176.2 (01/2007), S. 774–793

[22] Craig W. Reynolds. „Flocks, Herds and Schools: A Distributed Behavioral Model“. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, S. 25–34

[23] S. N. Sivanandam und S. N. Deepa. *Introduction to genetic algorithms: with 13 tables*. OCLC: 255495372. Berlin: Springer, 2008. 442 S.

[24] Peng-Yeng Yin und Jing-Yu Wang. „Ant colony optimization for the nonlinear resource allocation problem“. In: *Applied Mathematics and Computation* 174.2 (03/2006), S. 1438–1453

Abbildungsverzeichnis

3.1 Leistungsvergleich der beiden Lösungen für Bsp. 3.1 14
3.2 Graph für Beispiel 3.2 17
3.3 Graph für Beispiel 3.3 19
3.4 Graph für Beispiel 3.4, Zuteilung 1 21
3.5 Graph für Beispiel 3.4, Zuteilung 2 22
3.6 Leistungsvergleich der beiden Lösungen für Bsp. 3.4 22

4.1 Graph für Beispiel 4.1 29
4.2 Graph aus Abbildung 4.1, nachdem entlang des Weges 0-3-6 augmentiert wurde. 30
4.3 Graph aus Abbildung 4.2, nachdem entlang des Weges 0-1-2-6 augmentiert wurde. 30
4.4 Graph aus Abbildung 4.3, nachdem entlang des Weges 0-1-2-5-6 augmentiert wurde. 31
4.5 Graph für Beispiel 4.2 33
4.6 Graph aus Beispiel 4.1 35
4.7 Graph aus Abbildung 4.6, nachdem entlang des Weges 0-3-4-5-6 augmentiert wurde. 36
4.8 Graph aus Abbildung 4.7, nachdem entlang des Weges 0-3-6 augmentiert wurde. 36
4.9 Graph aus Abbildung 4.8, nachdem entlang des Weges 0-1-2-6 augmentiert wurde. 37
4.10 Graph aus Abbildung 4.9, nachdem entlang des Weges 0-1-2-5-6 augmentiert wurde. 37

5.1 Graph für Beispiel 5.1 nach der ersten Iteration 42
5.2 Graph für Beispiel 5.1 nach der zweiten Iteration 43
5.3 Graph für Beispiel 5.1 nach der dritten Iteration 44
5.4 Graph für Beispiel 5.1 nach der letzten Iteration 45

6.1 Beispiel eines Individuums inkl. Allelen 49
6.2 Roulettekessel für Beispiel 6.1 52
6.3 Roulettekessel für Beispiel 6.2 54
6.4 Roulettekessel für Beispiel 6.3 55
6.5 Individuum für Beispiel 6.8 inkl. Allelen 63
6.6 Individuum für Beispel 6.9 inkl. Allelen 65

6.7 Individuum für Beispel 6.9 nach dem ersten Reperaturschritt 65
6.8 Individuum für Beispel 6.9 nach dem zweiten Reperaturschritt 65

7.1 Graph für Beispiel 7.1 70
7.2 Graph für Beispiel 7.2 72
7.3 Graph für Beispiel 7.3 76
7.4 Generationenverlauf für Parameterset R8 (Testbeispiel 7.3) 82
7.5 Generationenverlauf für Parameterset A8 (Testbeispiel 7.3) 82
7.6 Graph für Beispiel 7.4 83
7.7 Generationenverlauf für Parameterset R1 (Testbeispiel 7.4) 88

Tabellenverzeichnis

2.1 Notation für das ALWABP 7

3.1 Leistungen $l_{s,r}$ für 3.1 14
3.2 Stationseigenschaften für 3.1 14
3.3 Lösung 1 für Bsp. 3.1 14
3.4 Lösung 2 für Bsp. 3.1 14
3.5 Verteilungen $V_{i,j}$ für Beispiel 3.2 17
3.6 Puffer- und Pufferkapazitätsmatrix für Bsp. 3.3 19
3.7 Leistungen für 3.4 20
3.8 Stationseigenschaften für 3.4 20
3.9 Initialbelegung für Bsp. 3.4 20
3.10 Umschichtungsmatrix Bsp. 3.4 20
3.11 Verteilungs-, Puffer- und Pufferkapazitätsmatrix für Bsp. 3.4 21
3.12 Lösung 1 für Bsp. 3.4 21
3.13 Lösung 2 für Bsp. 3.4 21

5.1 Leistungen $l_{s,r}$ für 5.1 41
5.2 Stationseigenschaften für 5.1 41
5.3 Verteilungs-, Puffer- und Pufferkapazitätsmatrix für Bsp. 5.1 41
5.4 Zuteilungsmatrix für Beispiel 5.1 nach der ersten Iteration 42
5.5 Zuteilungsmatrix für Beispiel 5.1 nach der zweiten Iteration 43
5.6 Zuteilungsmatrix für Beispiel 5.1 nach der dritten Iteration 44
5.7 Zuteilungsmatrix für Beispiel 5.1 nach der letzten Iteration 44

7.1 Bewertungskriterien der Algorithmen 67
7.2 Kürzel der Parametersets 68
7.3 Leistungen für 7.1 69
7.4 Stationseigenschaften für 7.1 69
7.5 Verteilungs-, Puffer- und Pufferkapazitätsmatrix für Bsp. 7.1 69
7.6 Testergebnisse für Beispiel 7.1 71
7.7 Leistungen für 7.2 73
7.8 Stationseigenschaften für 7.2 73
7.9 Verteilungsmatrix für Bsp. 7.2 73
7.10 Umschichtungsmatrix für Bsp. 7.2 73
7.11 Initialbelegung für 7.2 74
7.12 Testergebnisse für Beispiel 7.2 74

7.13 Leistungen für 7.3 76
7.14 Stationseigenschaften für 7.3 78
7.15 Verteilungsmatrix für Bsp. 7.3 78
7.16 Initialbelegung für 7.3 79
7.17 Umschichtungsmatrix für Bsp. 7.3 79
7.18 Testergebnisse für Beispiel 7.3 80
7.19 Leistungen für 7.4 83
7.20 Stationseigenschaften für 7.4 86
7.21 Testergebnisse für Beispiel 7.4 87