MASTERARBEIT

Simulationsbasierte metaheuristische Optimierung eines realen Auftragsreihenfolgeproblems

durchgeführt an der

MONTANUNIVERSITÄT LEOBEN

am

Department Mathematik und Informationstechnologie

Lehrstuhl für Angewandte Mathematik

Betreuer: Ao. Univ.-Prof. Dipl.-Ing. Dr.techn. Norbert Seifter

Vorgelegt von: Florian Felix Kamhuber, BSc
Matrikelnummer: 0435138

zum

Erlangen des akademischen Grades

DIPLOMINGENIEUR

(Dipl.-Ing.)

"Species do not evolve to perfection, but quite the contrary. The weak, in fact, always prevail over the strong, not only because they are in the majority, but also because they are the more crafty."

(Friedrich Nietzsche, The Twilight of the Idols)

"It is not the strongest of the species that survives, nor the most intelligent, but the one most responsive to change.

(Charles Darwin)

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.

Leoben, am 18.2.2010

Unterschrift:

Felix Kamhuber

Danksagung

Die vorliegende Masterbeit wurde in der Zeit von Juli 2009 bis Februar 2010 am Institut für Angewandte Mathematik der Montanuniversität Leoben verfasst.

Die Begeisterung für das Thema heuristische Optimierung wurde im Laufe des Studiums in mir geweckt und entstand speziell im Rahmen der Vorlesungen “Mathematische Grundlagen des Operations Research” sowie “Optimierung für Industrielogistiker”, welche von Herrn Prof. Dr. Seifter abgehalten worden sind.

Besonderen Dank seitens der Montanuniversität gilt daher meinem Betreuer Herrn Prof. Dr. Seifter für seine ausgezeichnete Betreuung während des kompletten Zeitraums.

Außerdem gilt mein Dank meinen Betreuern seitens Profactor, Herrn Dipl.-Ing. Dr. Thomas Löscher sowie Herrn Dipl.-Ing. Matthias Gruber. Beide waren während der gesamten Zeit für fachliche Rückfragen und problemspezifische Fragestellungen für mich erreichbar.

Weiters möchte ich mich bei Herrn Dipl.-Ing. Robert Steringer, Herrn Mag. Rinner sowie Herrn Dipl.-Ing. Arnold Wollschlager und Herrn Dipl.-Ing. Dr. Markus Vorderwinkler herzlich bedanken. Sie alle haben mir während der Erstellung dieser Arbeit beigestanden.

Diese Arbeit widme ich meinen Eltern, die mich während meines Studiums fortwährend unterstützt haben.

Kurzfassung

Im Rahmen dieser Masterarbeit werden Metaheuristiken zur Optimierung eines realen Auftragsreihenfolgeproblems vorgestellt und eingesetzt. Dieses Spektrum umfasst vor allem hybride genetische und evolutionäre Algorithmen, Ameisenalgorithmen, Simulated Annealing sowie iterierte lokale Suche. Diese Algorithmen wurden inklusive kundenspezifischer Nebenbedingungen implementiert und optimiert, wobei der Endbenutzer, konkret der Produktionsplaner, jeweils die einzelnen Nebenbedingungen für jeden Optimierungsalgorithmus ein- und ausschalten kann. Im Rahmen von Testläufen wurde der Fitnessverlauf der Algorithmen beispielhaft anhand von Simulated Annealing für mehrere Datensätze festgestellt. Anschließend wurden die Algorithmen selbst unter gleichen Nebenbedingungen sowie möglichst ähnlichen Bedingungen in Bezug auf ihre Lösungsgüte verglichen. Weitere Testläufe wurden im Zusammenhang mit den Nebenbedingungen durchgeführt. Es wurden verschiedene Parametrisierungen betrachtet, welche den Suchraum jeweils unterschiedlich intensiv durchsuchen. Das dazugehörige reale Auftragsreihenfolgeproblem beinhaltet zum Teil komplexe Nebenbedingungen, welche den Lösungsraum einerseits einschränken, andererseits einfache Lösungen von Teilen innerhalb des gesamten Optimierungsproblems erlauben. Außerdem ist der Lösungsraum nicht einheitlich, sondern durch unterschiedlich große Kampagnen, die voneinander getrennt betrachtet werden müssen, charakterisiert.

Während neben einer ausführlichen Problemcharakterisierung vor allem die diversen Metaheuristiken in den ersten fünf Kapiteln dieser Arbeit detailliert beschrieben werden, um dem Leser einen Überblick über die Arbeitsweise dieser Verfahren zu geben, wird in den letzten drei Kapiteln auf die Implementierung der Algorithmen näher eingegangen, sowie das Optimierungspotenzial der getesteten Verfahren in Bezug auf dieses Auftragsreihenfolgeproblem unter realen Bedingungen untersucht. Im letzten Kapitel werden im Rahmen der Schlussfolgerungen aus den Ergebnissen zwei Algorithmen empfohlen, welche sich für dieses spezielle Auftragsreihenfolgeproblem insgesamt am besten eignen.

Abstract

Within the scope of this master thesis a real permutation flow shop problem is solved and optimized by metaheuristics. In particular we implemented hybrid genetic and evolutionary algorithms, ant colony optimization algorithms, simulated annealing as well as iterated local search. These algorithms have been implemented and optimized including specific customer constraints, whereas the final end user, who is represented by a production planner, has the opportunity to choose the constraints under which the problem has to be optimized. First the behaviour of simulated annealing was considered, depending on simulation runs to determine the convergence of the algorithms on several datasets. The algorithms themselves were compared under the same constraints and similar conditions. This permutation flow shop problem comprises to some extent complex constraints, which restrict the solution space but also enable the algorithm to solve certain parts within the whole optimization problem quite easily within short time. The solution space is not homogenous, because we have to consider campaigns which extremely differ in size and restriction of the solution space by their constraints.

Next to a detailed problem characterization the metaheuristics are described in detail within the first five chapters of this master thesis to give the reader an overview and an outline of the main ideas of these methods. The last three chapters describe the implementation and discuss the performance of the algorithms and compare their usefulness for the optimization of this real permutation flow shop problem. Within the last chapter we give a recommendation for two algorithms which have delivered the best overall performance on this concrete permutation flow shop problem.

Inhaltsverzeichnis

Algorithmenverzeichnis

Nomenclature

Abb. Abbildung

ACO Ant Colony Optimization (dt. Ameisenalgorithmus)

ACS Ant Colony System

Alg. Algorithmus

AS Ant System

bzw. beziehungsweise

ca. circa

d.h. das heißt

EA Evolutionary Algorithm (dt.: Evolutionärer Algorithmus)

FlowShop Reihenfertigung

GA Genetic Algorithm (dt.: Genetischer Algorithmus)

HGA Hybrid Genetic Algorithm (dt.: Hybrider genetischer Algorithmus)

i.A. im Allgemeinen

i.d.R. in der Regel

ILS Iterated Local Search (dt.: Iterierte lokale Suche)

JobShop Werkstattfertigung

MA Memetic Algorithm (dt.: Memetischer Algorithmus)

PFS(S)P Permutation Flow Shop (sequencing) problem (dt.: Auftragsreihenfolgeproblem)

Tab. Tabelle

u.s.w. und so weiter

v.a. vor allem

Vgl. Vergleiche

z.B. zum Beispiel

TSP Travelling Salesman Problem

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

1 Einleitung

Im Rahmen der Produktionsplanung trifft man häufig auf sogenannte Auftragsreihenfolgeprobleme, welche im Englischen auch als Permutation Flow Shop Sequencing Problem(kurz: PFSSP) bekannt sind. Auf Grund der verschärften Wettbewerbssituation in den letzten Jahren ist eine effiziente Produktionsplanung unumgänglich, sodass man mit einfachen Heuristiken, wie z.B. der Johnson Heuristik für 2- Maschinen bzw. für 3- Maschinenprobleme unter bestimmten Bedingungen, nicht mehr auskommt. Viele Nebenbedingungen wie z.B. das Einhalten der Liefertreue bzw. zumindest eine Minimierung der Pönalkosten, sowie weiters eine hohe Aggregatauslastung, eine geringe Pufferauslastung und eine geringe Gesamtdurchlaufzeit wird bei der Abarbeitung des Produktionsprogramms angestrebt, um die Gesamtkosten möglichst gering zu halten und damit wettbewerbsfähig zu bleiben. Da diese Ziele erheblich von der konkreten Auftragsreihenfolge beeinflusst werden, wird im Rahmen dieser Masterarbeit versucht, ein konkretes gegebenes Produktionsproblem vor allem mit genetischen und evolutionären Algorithmen zu lösen. Diese Algorithmen werden mit einem Ameisenalgorithmus, einem Simulated Annealing Verfahren sowie mit iterierter lokaler Suche verglichen.

Ausgangspunkt für die Erforschung von Reihenfolgeproblemen war die Veröffentlichung des Johnson Algorithmus (1954), welcher unter bestimmten Bedingungen nicht nur 2- Maschinen Probleme, sondern auch 3- Maschinen Probleme deterministisch löst.

Allgemein lässt sich bezüglich der Organisationsform die Werkstattfertigung (“job shop”) von der Reihenfertigung (“flow shop”) unterscheiden. Permutation Flow Shop legt für alle Aufträge (“jobs”) die gleiche Maschinenfolge fest[9].

Diese Masterarbeit ist wie folgt aufgebaut: Kapitel 2 gibt einen detaillierten Überblick über das konkrete Auftragsreihenfolgeproblem. Neben der allgemeinen Charakterisierung wird vor allem auf das Zielsystem bzw. das damit verbundene Bewertungsmodell für die Fitness, die konkreten soft constraints und hard constraints, sowie die Suchraumgröße- und Komplexität eingegangen.

Kapitel 3 beschreibt Metaheuristiken abseits von genetischen und evolutionären Verfahren, welche sich sowohl als eigenständige Verfahren als auch in möglicher Kombination mit genetischen Algorithmen bzw. anderen Heuristiken für Auftragsreihenfolgeprobleme eignen.

Kapitel 4 beschreibt die allgemeinen Charakteristika von genetischen Algorithmen. Es werden speziell die grundlegenden sowie fortgeschrittenen Varianten der primären Operatoren diskutiert. Weitere Ansätze, einige spezifische Strategien, welche sich auf konkrete Problemstellungen beziehen, runden das Kapitel ab.

Kapitel 5 beschreibt die wichtigsten evolutionären Strategien und diskutiert die Unterschiede zwischen evolutionären und genetischen Verfahren.

Kapitel 6 beschreibt die grundlegende Programmstruktur sowie die wesentlichen implementierten Algorithmen.

Kapitel 7 gibt Auskunft über die Lösungsqualität der einzelnen Verfahren auch in Bezug auf die verwendeten Parameter der einzelnen Verfahren. Für jeden verfügbaren Datensatz wurde eine sogenannte

Fitnessverlaufskurve in Abhängigkeit der Anzahl an durchgeführten Bewertungen erstellt, welche Auskunft darüber gibt, wie schnell ein Algorithmus auf diesem Datensatz konvergiert. Diese Kurve wird unter realen Bedingungen erstellt. Außerdem werden die Algorithmen anhand spezifischer Kampagnen miteinander verglichen. Weitere Ergebnisse werden dargestellt, wenn eine bestimmte Nebenbedingung aufgeweicht bzw. deaktiviert wird.

Diese Masterarbeit soll, ausgehend von einem realen Produktionsproblem mit diversen komplexen Nebenbedingungen, Aufschluss über die Eignung von Metaheuristiken bzw. v.a. (vor allem) genetischen und evolutionären Algorithmen für Auftragsreihenfolgeprobleme geben. Außerdem werden auch hybride Ansätze und andere Metaheuristiken vergleichend betrachtet.

Die zu beantwortende Forschungsfrage besteht darin, festzustellen, wie groß das Optimierungspotenzial für dieses konkrete Auftragsreihenfolgeproblem unter realen Bedingungen, also unter Berücksichtigung sämtlicher zu implementierender Nebenbedingungen, ausfällt. Außerdem gilt es die Frage zu beantworten, welche Laufzeit für die Erreichung eines Großteils des bestehenden Optimierungspotenzials notwendig ist. Zuletzt wird das zusätzliche Potenzial betrachtet, welches sich durch die Deaktivierung einer entscheidenden Nebenbedingung ergibt. Daneben werden weitere problemspezifische Erkenntnisse in dieser Arbeit festgehalten. Außerdem werden auch die einzelnen Verfahren miteinander unter möglichst ähnlichen Bedingungen in Bezug auf die Eignung für dieses Problem verglichen. Im Rahmen dieses Vergleichs wird eine abschließende Empfehlung für zwei Verfahren gegeben.

Der Vergleich der einzelnen Algorithmen bezieht sich dabei jeweils auf die Gesamtperformance (Lösungsqualität, benötigte durchschnittliche Laufzeit (in Bewertungen) für eine bestimmte Lösungsqualität) der einzelnen Metaheuristik für das reale Produktionsproblem.

Kapitel 2

2 Problemcharakterisierung

2.1 Beschreibung und Charakteristika von Permutation Flow Shop (Sequencing) Problemen

Ein Auftrag wird durch eine Reihe von mehreren aufeinander folgenden technologisch zusammenhängenden Arbeitsgängen beschrieben. Für ein PFSSP (Permutation Flow Shop Sequencing Problem) gilt dabei, dass jeder Auftrag dieselbe Maschinenfolge hat. Bei vorausgesetzter identer Maschinenfolge (M Maschinen) gibt es für N Aufträge (*"jobs"*) $N!$ mögliche Lösungen. Das gesamte Produktionsprogramm bestehend aus N Aufträgen wird auf M unabhängigen Maschinen gefertigt, wobei jeder Auftrag (*"job"*) auf jeder Maschine exakt einmal bearbeitet werden muss. Eine Maschine kann dabei i.d.R. (in der Regel) nicht mehr als einen Auftrag gleichzeitig bearbeiten [1].

2.1.1 Charakteristika des konkreten Produktionsproblems

2.1.1.1 Beschreibung des Produktionsproblems

Das Optimierungsziel besteht grundsätzlich darin, die von internen Experten zusammengestellte Auftragsreihenfolge, hinsichtlich mehrerer Kriterien zu optimieren bzw. (beziehungsweise) zu verbessern. Die mittels heuristischer Methoden arbeitende Software soll dem Produktionsleiter hochwertige Vorschläge liefern, welche operative und strategische Ziele beinhalten. Das operative Ziel ist in diesem Beispiel die Termintreue bei Lieferungen, weiters geht es darum, mögliche Engpässe im Rahmen der Simulation besser zu erkennen und diese durch die Optimierung zu umgehen. Das strategische Ziel stellt die gleichmäßige Auslastung des Walzwerks dar, außerdem soll die Simulation Auskunft über die notwendige Dimensionierung der Adjustage für die Zukunft geben [19, 38].

Die Produktion soll gleichmäßig und robust verlaufen, außerdem sollen die Zwischenpuffer hinsichtlich ihrer Auslastung optimiert werden. Durch eine gleichmäßige Auslastung soll auch eine bessere Aggregatauslastung erreicht werden.

Die Lösung und Optimierung des zugrundeliegenden Auftragsreihenfolgeproblems orientiert sich an erwerbswirtschaftlichen Prinzipien[9]. Da die Reihenfolgeplanung auf der Erlösseite keinen Einfluss hat, weil diese durch die Absatz- und Produktionsprogrammplanung bestimmt ist, zielt die Optimierung der Reihenfolgeplanung auf die Minimierung entscheidungsrelevanter Kosten ab. Diese Kosten umfassen v.a. Lagerkosten, Rüstkosten sowie Pönalkosten.

Lagerkosten setzen sich dabei aus Kapitalbindungskosten, Lagerraumkosten sowie Lagerhaltungskosten zusammen. Die ablaufbedingte Wartezeit setzt sich aus der Vorlagerzeit, den Zwischenlagerzeiten sowie der Nachlagerzeit zusammen.

2.1.1.2 Produktionsablauf und Prozessbeschreibung

Da die genauen Wirkzusammenhänge der Aggregate in dieser Arbeit aus Geheimhaltungsgründen nicht weiter ausgeführt werden können, findet im folgenden Abschnitt lediglich eine kurze Prozessbeschreibung statt.

Die Systemgrenzen stellen die Walzstraße dar, welche den Simulationseingang abbildet, sowie der Schienenabgang aus der Endkontrolle in die Läger, wobei dieser Abgang das Simulationsende darstellt.

Das Schichtmodell kann an dieser Stelle nicht erläutert werden, es ist jedenfalls im Rahmen der Simulation möglichst realitätsnahe abgebildet.

Im Rahmen der Prozessbeschreibung wird an dieser Stelle erläutert, dass die Simulation die Eingangsdaten über eine MS (Microsoft) Excel Datei erhält, welche mehrere Arbeitsblätter (Produktionsplan, Schienenliste,...) enthält. Diese Informationen sind die Grundlage für einen Simulationslauf.

Prozessbeschreibung: Prinzipiell versorgt das Walzwerk die Adjustage mit gewalzten und ggf. wärmebehandelten Schienen . Zuerst werden Vorblöcke von einem Mitarbeiter mit Hilfe eines Krans auf den Hubbalkenofen aufgelegt. Danach werden die erwärmten Vorblöcke automatisch in das Walzgerüst eingebracht. Nach dem Walzen werden die fertigen Schienen zum Hubbalkenkühlbett transportiert. Dieses Hubbalkenkühlbett mit integrierter HSH Anlage übernimmt die Abkühlung sowie ggf. die Härtung der Schienen. Anschließend wird die H+V Rollenrichtmaschine händisch im FIFO Prinzip beschickt. Danach erfolgt die Übergabe an den Automatikzyklus der Adjustage [38] .

Der Puffer vor dem Prüfzentrum wird im FIFO (First In - First Out) Prinzip betrieben. Im Prüfzentrum erfolgt die Prüfung der Schiene, welche dabei nicht stehenbleibt.

Anschließend gelangen die Schienen in den Railman 1, welcher den Auslaufrollgang des Prüfzentrums entsorgt. Ausgehend von Railman 1 gibt es zwei Möglichkeiten der Schienenweiterleitung. Während ein Rollgang in Richtung Richtpresse 1 und 2 führt, führt ein weiterer Richtgang in Richtung Sägepuffer und Lagerplatz.

Railman 2 übernimmt die Entsorgung der Auslaufrollgänge der Richtpressen, von wo aus eine Schiene an SBL 2 weitergereicht wird.

Beide Railman Aggregate stellen grundsätzlich wiederum FIFO-Puffer dar.

Drei Richtpressen übernehmen in Abhängigkeit von der Schienenqualität die Richtung der Schienen. Sobald der Auslaufrollgang von Railman 2 entleert wurde, wird die nächste Schiene angefordert.

Bevor die Schienen in eine der beiden Sägebohrlinien gelangen, kommen sie in den Puffer vor SBL 1, wo sie jeweils im FIFO Prinzip ein- und ausgelagert werden. Die Bearbeitungszeit in der darauf folgenden Sägebohrlinie (SBL) ist abhängig von der Qualität sowie der Anzahl der durchgeführten Schnitte, wobei jede Schiene mindestens einmal geschnitten wird. Da SBL 1 nur Schienen mit einer maximalen Länge von 62 Metern bearbeiten kann, müssen längere Schienen auf SBL 2 geschnitten werden. Nach einer bestimmten Anzahl an Schnitten ist ein Sägeblattwechsel notwendig, nach einer bestimmten Anzahl an Bohrungen ist ein Bohrerwechsel notwendig [38].

Anschließend gelangen die Schienen in das Inspektionsbett bzw. die Endkontrolle (EK). Zu beachten ist dabei, dass Schienen aus SBL 1 ausschließlich in EK 1 gelangen dürfen, während Schienen aus SBL 2 sowohl in EK 2, EK 3 oder auch EK 4 einlangen können. Zu beachten ist dabei, dass die Langschienenlager (EK 3 und EK 4) nur Schienen ab 30 Meter Länge aufnehmen können. Alle kürzeren Schienen vom SBL 2 müssen den Produktionsfluss über EK 2 verlassen [38].

2.2 Zielsystem und Bewertungsmodell

In der Schienenproduktion besteht das Produktionsprogramm grundsätzlich aus n Walzkampagnen ($n \geq 1$), welche jeweils aus m_i Losen ($m \geq 1; 1 \leq i \leq n$) bestehen. Dazwischen gibt es noch von Seiten des Kunden den Kundenauftrag, welcher auf mehrere Lose aufgeteilt wird. Ein Los stellt dabei eine Menge Schienen dar, die produziert werden sollen. Diese Menge an Schienen gehört zu einem oder mehreren bestimmten Kundenauftrag(en). Ein Walzblock ist das Rohmaterial, aus welchem die Schienen gewalzt werden.

2.2.1 Zielsystem

Alle zu optmierenden Parameter werden über die zu simulierende Kampagnenmenge zu einer Gesamtfitness akkumuliert. Die Einzelterme sind verschieden stark gewichtet, wobei jeder Term virtuelle Kosten liefert, die sich aus der Abarbeitung des vorgeschlagenen Produktionsprogramms ergeben würden.

Im Rahmen der Optimierung ist grundsätzlich zu beachten, dass jede Kampagne lediglich für sich optimiert werden kann, die Auswirkungen der optimierten Kampagne werden dann in Bezug zum Gesamtsystem betrachtet und neu bewertet. Auf der Basis des jeweils besten Kampagnenergebnisses wird in der nächsten Kampagne weitergearbeitet, bis alle zu optimierenden Kampagnen bewältigt worden sind und das insgesamt beste Ergebnis ausgegeben wird.

In dieser Fitnessfunktion sind direkt Termintreue, Lagerkosten sowie die Auslastung der entscheidenden Aggregate und Puffer enthalten. [19]

2.2.2 Bewertungsmodell zur Validierung der Lösungsgüte

Bewertungsfunktion (Fitness) : Zur Bewertung des Simulationsergebnisses werden die Termintreue, welche einer Abweichung des Fertigstellungsdatums vom Lieferdatum entspricht, die Stillstandszeiten des Walzwerks sowie des Prüfzentrums sowie die Auslastungszustände der beiden Railman Aggregate herangezogen.

Die Bewertungsfunktion, welche eine Kostenminimierungsfunktion darstellt, setzt sich gemäß diesem Sachverhalt [19] wie folgt zusammen:

$f = w_1 \cdot \sum_i (f_1(c_\mathrm{i})) + w_2 k_2 d_1 + w_3 k_3 d_2 + \int_t f_4(b_1(t)) + w_5 \int_t f_5(b_2(t))$

Legende: (Quelle: [19])

$w_1 - w_5$.....Gewichtungen

i.....Zählervariable für Lose

f_1.....Bewertungsfunktion für Lieferverzug und Lagerkosten

c_i.....Relativer Fertigstellungszeitpunkt

k_2.....Konstante für Stillstandskosten im Walzwerk

d_1.....Akkumulierte Stillstandszeit Walzwerk

k_3.....Konstante für Stillstandskosten im Prüfzentrum

d_2.....Akkumulierte Stillstandszeit Prüfzentrum

f_4.....Bewertungsfunktion für virtuelle Kosten im Railman 1

b_1.....Relativer Pufferfüllstand im Railman 1

f_5.....Bewertungsfunktion für virtuelle Kosten im Railman 2

b_2.....Relativer Pufferfüllstand im Railman 2

Bewertungsfunktion in Bezug auf die Termintreue (f_1) : Für die Bewertung der Differenzen zwischen zugesagtem Liefertermin und Auftragsfertigstellungszeitpunkt wird folgende Funktion (Abb 2.1: Quelle:[19]) verwendet:

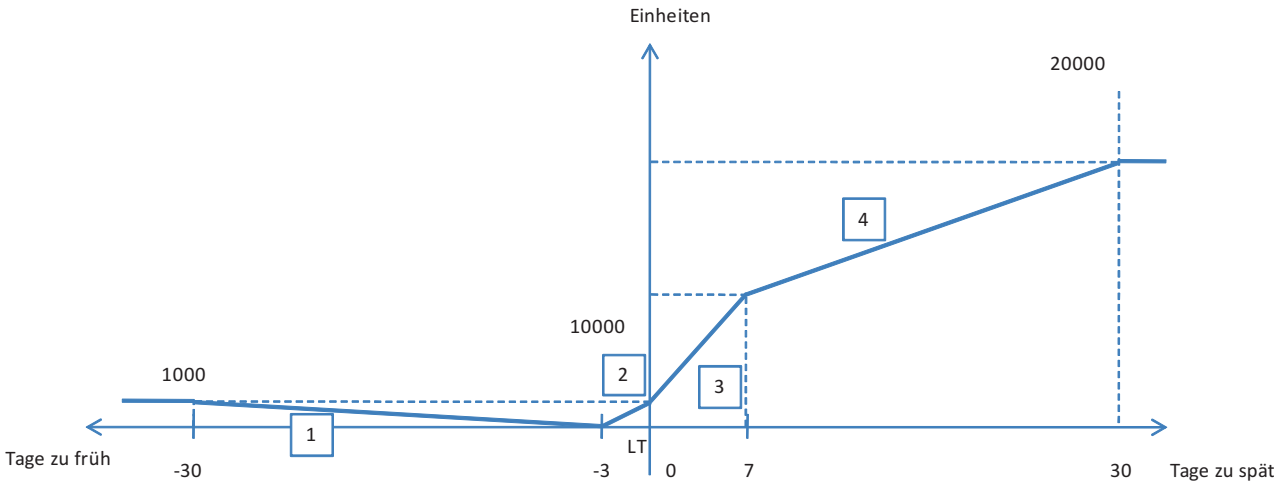


Abb. 2.1: Bewertungsfunktion Termintreue (f_1)

In Abb. 2.1. sind die 4 verschiedenen Funktionsbereiche, welche die Wertigkeiten spiegeln, ersichtlich.

Von Seiten des Kunden wird ein Fertigstellungszeitpunkt angestrebt, welcher 3 Tage vor dem Liefertermin liegt. Dieser Punkt stellt in Bezug auf das Kriterium *Termintreue* das Optimum bzw. den optimalen Fertigstellungszeitpunkt dar und wird selbst nicht pönalisiert.

Bereich 1 [-30,-3] beschreibt den Funktionsverlauf der Lagerkosten, wenn der Fertigstellungszeitpunkt mehr als 3 Tage vor dem Lieferdatum liegt. Bei über 30 Tagen werden die Strafkosten auf 1000 Einheiten (Euro) für das entsprechende Los festgesetzt. Der Pönalkostenbetrag verringert sich dabei in Richtung optimalem Fertigstellungszeitpunkt (= 3 Tage vor Lieferdatum) linear, wobei die Steigung in diesem Bereich relativ gering ist. Bei einer frühzeitigen Fertigstellung zwischen 30 und 3 Tagen wird der sich durch die Auflösung der Geradengleichung ergebende Bruchteil der Pönalkosten berechnet.

Bereich 2 [-3,0] zeigt die Zeitspanne zwischen optimalem Fertigstellungszeitpunkt und vereinbartem Liefertermin. Die Fertigstellungszeitpunkte nach dem optimalen Fertigstellungszeitpunkt werden mit linear steigenden Kosten pönalisiert, bis das Maximum von 1000 Einheiten pro Los am Tag der vereinbarten Lieferung erreicht ist. Die Steigung der Funktion in diesem Bereich ist bereits deutlich stärker als in Bereich 1.

Bereich 3 [0,7] stellt den Zeitraum zwischen vereinbartem Liefertermin und einer Woche nach vereinbartem Lieferdatum dar. Linear werden bei Erreichen von 7 Tagen Lieferverzug 10.000 Einheiten Pönalkosten für das entsprechende Produktionslos. Die Steigung der Funktion ist in diesem Bereich am Größten.

Bereich 4 [7,30] deckt die Zeitspanne zwischen einer Woche Lieferverzug und 30 Tagen nach vereinbartem Liefertermin ab, bei Erreichen dieser 30 Tage werden die Strafkosten auf 20.000 Einheiten pro Los festgesetzt. Die Steigung der Funktion ist hier deutlich geringer als in Bereich 3.

Bewertungsfunktion in Bezug auf die Stillstandszeit des Walzwerks: Auf Grund der besonders hohen Anschaffungskosten nimmt das Walzwerk eine zentralle Stellung in der Gesamtanlage ein, weshalb eine höchstmögliche Aggregat uptime erstrebenswert ist. Daher wird während eines Simulationslaufs der Stillstand des Walzwerks laufend mitprotokolliert. Stillstand bedeutet entweder, dass sich das Walzwerk im Zustand BLOCKED, SETUP oder PAUSED befindet. Im Statistikobjekt befindet sich die Methode getStatesOfAggregate(String aggregate), welches die einzelnen Statuswechsel für das jeweilige

Aggregat ausliest. Die konkret festgelegte Pönale für eine Stunde Stillstand beträgt 2000 Einheiten (Euro), dieser Wert wird mit der gesamten Stillstandszeit des Walzwerks multipliziert.

Analog zum Walzwerk werden im Modell auch die Pönalkosten für die Stillstandszeiten im Prüfzentrum berücksichtigt.

Bewertungsfunktion in Bezug auf die Auslastung von Railman 1 und Railman 2 (f_1, f_4, f_5) Die Auslastung der beiden Railman Aggregate definiert sich über Einfüge- und Auslagerungsvorgänge. Daher wird der Status bei jedem Einlagerungs- und Auslagerungsvorgang dementsprechend verändert und aktualisiert mitprotokolliert bis zur nächsten derartigen Statusänderung. Für die Zeitspanne bis zur nächsten Statusänderung gilt dieser mitprotokolliere Auslastungsstand des Railman 1 bzw. des Railman 2 Aggregats.

Die Pufferstände beider Railman Aggregate haben einen großen Einfluss auf die Gesamtperformance des Produktionssystems. Hohe Pufferfüllstände implizieren einen Rückstau, während bei unzureichend geringen Füllständen die Gefahr der Blockade der nachfolgenden Aggregate besteht. Simulationsexperimente bestätigen, dass die optimalen Füllstande beider Aggregate bei ca. 9% liegen. Daher wird dieser Wert als kostenoptimaler Betriebspunkt angesehen. Abb 2.2 (Quelle: [19]) zeigt den Funktionsverlauf von Railman 1.

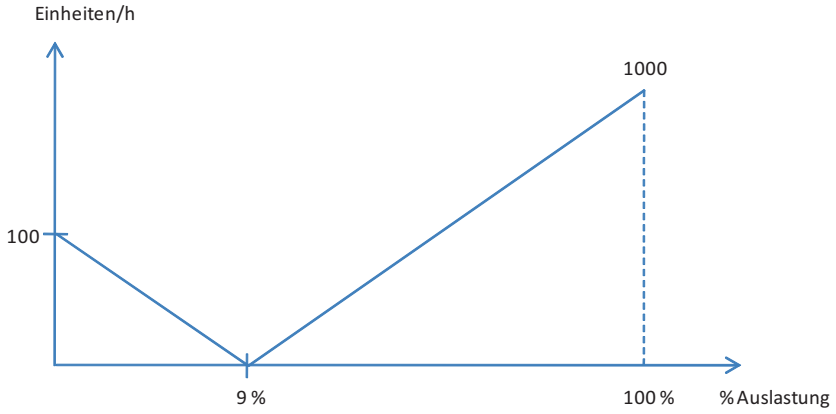


Abb. 2.2: Auslastungsfunktion für Railman 1 (f_4)

Abb 2.2 zeigt also, dass bei einer Auslastung von 0% 100 Einheiten/Stunde als Strafkosten angesetzt werden. Diese relativ hohe Pönale spiegelt die Gefahr wider, dass durch diesen Auslastungszustand die nachfolgenden Aggregate zum Stillstand kommen. Diese Strafkosten sinken linear bis zum optimalen Betriebspunkt, welcher bei 9% liegt (dies entspricht ca. 2 Schienen) und steigen von da linear an bis zu einem Maximalwert von 1000 Einheiten/ Stunde bei 100% iger Auslastung.

Die Kostenfunktion für Railman 2 gleicht der Funktion für Railman 1 mit der Ausnahme, dass auf Grund der geringeren Bedeutung von Railman 2 jeweils halbierte Strafkosten angesetzt werden, da der Einfluss auf Grund der geringeren Anzahl nachfolgender Aggregate geringer anzusetzen ist.

Neben den Gewichtungen für die Gesamtbewertungsfunktion kommen noch Modellparameter und Optimierungsparameter. Dabei wird in globale (Planungshoizont,...) und Modellelement basierte Parameter (Taktzeiten, Umrüstzeiten, Puffergrößen,...) unterschieden.

Die Optimierungsparameter hängen v.a. von der eingesetzten Optimierungsstrategie ab. Für den genetischen Algorithmus lauten diese Parameter Populationsgröße, Anzahl Iterationen, Kreuzungsrate und Mutationsrate. Für die lokale Suche mittels Insertion Search kann der Benutzer das Ausmaß der lokalen Suche mittels einstellbarem Wert (α) bestimmen. Diese spezifischen Parameter für jeden Optimierungsalgorithmus sind in Abschnitt 6.3 zu finden.

Gewichtung: Die Wichtigkeit der einzelnen Kriterien kann prinzipiell eingestellt werden. Die konkreten Werte bleiben im Rahmen sämtlicher durchgeführter Simulationsdurchläufe zwecks Vergleichbarkeit konstant und können in dieser Arbeit aus Geheimhaltungsgründen nicht näher konkretisiert werden.

2.3 Nebenbedingungen im Auftragsreihenfolgeproblem

2.3.1 Harte Nebenbedingung

Die einzige harte Nebenbedingung besteht in diesem Produktionsproblem darin, dass nicht kampagnenübergreifend getauscht werden darf.

2.3.2 Weiche Nebenbedingungen

1. Nebenbedingung betreffend die gleiche Auslastung der Sägebohrlinien: Sogenannte ZSG

(Zielsägebohrlinien- bzw. Zielsäge-) Lose sollen generell innerhalb der Simulation nicht getauscht werden, weil derartige Lose im realen Produktionssystem als ein einzelnes Los abgebildet werden und somit nur in der Simulation aus mehreren Losen bestehen. In der Simulation betrifft dies Lose mit den Kennzeichnungen xxx und xxx_ bzw. xxx___ etc.

Durch die Zuordnung eines großen Blocks zu den beiden Sägebohrlinien wird ein großes Los künstlich in mehrere Lose aufgeteilt. Diese Aufteilung, welche bereits in der Ausgangssequenz vorhanden ist, kann nur getauscht werden, sofern der Tausch auch innerhalb des ZSG Loses stattfindet. Alternativ dazu kann auch die Auslastung der beiden Sägebohrlinien verändert werden, indem anstelle eines ZSG losinternen Tausches die Zuordnung zu einer Sägebohrlinie geändert wird.

Die Überprüfung, ob es sich bei einem Los im Simulationslauf um ein ZSG Los handelt, passiert, indem man bei der Auswahl die Los-Id berücksichtigt und dort die Strings überprüft. Auf einem aus boolean Werten bestehenden array wird für die übergebene nach aufsteigenden Auftragsnummern sortierten Sequenz gespeichert, welche Lose in Bezug auf diese Nebenbedingung frei tauschen können bzw. welche Lose von dieser Nebenbedingung betroffen sind.

Vor jeder Änderung der Auftragsreihenfolge wird in den Optimierungsalgorithmen überprüft, ob diese Nebenbedingung eingehalten wird, sofern sie aktiviert ist. Der dazugehörige Codeabschnitt (Globale Tauschmethode) befindet sich im Anhang.

Aufträge | Aggregate | Pufferspeicher

Ausgangsreihenfolge

Simulationsbeginn: 07.09.2009 08:49:38
Simulationsende: 22.10.2009 06:31:10
Dauer: 44d 21:41:32
Fitness Value: 4940396,00

AscO...	Wa...	LotId	#Blocks	Profil	Qualität	Schnittmuster	ZS	ZEK	Lieferdatum	Fertigstellungsdatum
12	1	122856-3	17	VI49E1	HSH	3x36010	ZS1	EK1	31.08.2009	07.09.2009
13	1	121822-106	7	VI49E1	HSH	4x30100	ZS1	EK1	11.09.2009	07.09.2009
14	1	121822-105	2	VI49E1	HSH	4x29800	ZS1	EK1	11.09.2009	07.09.2009
15	1	121822-102	1	VI49E1	HSH	4x29600	ZS1	EK1	11.09.2009	07.09.2009
16	1	122971-1	12	VI49E1	HSH	6x18000	ZS2	EK2	21.09.2009	07.09.2009
17	1	122965-1	15	VI49E1	HSH	3x36000	ZS1	EK1	07.09.2009	08.09.2009
18	1	123011-2	21	VI49E1	HSH	8x15000	ZS2	EK2	11.09.2009	08.09.2009
19	1	122942-1	6	VI49E1	HSH	8x15000	ZS2	EK2	29.09.2009	08.09.2009
20	2	122922-3	40	VI60E1	Normal	1x120000	ZS2	EK4	17.09.2009	08.09.2009
21	2	122922-2	40	VI60E1	Normal	1x120000	ZS2	EK4	16.09.2009	08.09.2009
22	2	121591-67	42	VI60E1	Normal	3x40000	ZS1	EK1	09.09.2009	08.09.2009
23	2	122993-1	22	VI60E1	HSH	4x30000	ZS1	EK1	15.09.2009	09.09.2009
24	2	123013-2	20	VI60E1	HSH	1x60500;1x52...	ZS1	EK1	07.09.2009	09.09.2009
25	2	123013-3	5	VI60E1	HSH	2x60000	ZS1	EK1	07.09.2009	09.09.2009
26	2	123015-1	11	VI60E1	HSH	4x30000	ZS1	EK1	14.09.2009	08.09.2009
27	2	122657-2	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	08.09.2009
28	2	122657-2_	10	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	08.09.2009
29	2	122657-2__	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	08.09.2009
30	2	122657-2___	10	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	08.09.2009
31	2	122657-2____	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	09.09.2009
32	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	09.09.2009
33	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	09.09.2009
34	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	09.09.2009
35	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	09.09.2009
36	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	09.09.2009
37	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	09.09.2009
38	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	09.09.2009
39	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	09.09.2009
40	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	09.09.2009
41	2	122657-2_____	10	VI60E1	HSH	6x18000	ZS2	EK2	10.09.2009	09.09.2009
42	2	122657-2_____	12	VI60E1	HSH	6x18000	ZS1	EK1	10.09.2009	09.09.2009
43	3	122986-1	80	VI60E2	HSH	1x120000	ZS2	EK4	14.09.2009	12.09.2009
44	3	122902-2	240	VI60E2	HSH	1x120000	ZS2	EK4	07.09.2009	11.09.2009
45	4	122858-1	220	VISAR57	HSH	2x60000	ZS2	EK4	21.09.2009	12.09.2009
46	5	122968-1	19	ZU54E1A2	Normal	2x45000	ZS1	EK1	08.09.2009	12.09.2009
47	5	122968-2	19	ZU54E1A2	Normal	2x45000	ZS2	EK4	08.09.2009	15.09.2009
48	5	122968-4	19	ZU54E1A2	Normal	2x45000	ZS1	EK1	22.10.2009	15.09.2009
49	5	122968-3	19	ZU54E1A2	Normal	2x45000	ZS2	EK4	22.10.2009	15.09.2009
50	5	122860-2	17	ZU54E1A2	Normal	8x11110	ZS1	EK1	15.09.2009	15.09.2009
51	5	122860-3	3	ZU54E1A2	Normal	10x8640	ZS2	EK2	15.09.2009	15.09.2009
52	5	123043-1	6	ZU54E1A2	Normal	5x16780	ZS1	EK1	02.10.2009	15.09.2009
53	5	122969-1	10	ZU54E1A2	HSH	2x46870	ZS2	EK4	17.09.2009	15.09.2009

Abb. 2.3: ZSG Los in Kampagne 2: 122657-2

2. Nebenbedingung: Tauschen innerhalb derselben Qualitäts Güteklasse ↔ Weiche Schienen vor harten Schienen: Die 2. Nebenbedingung, besagt, dass Lose möglichst innerhalb einer Qualitäts Güteklasse getauscht werden sollen und zwar so, dass weiche Schienen vor harten Schienen gewalzt werden. Da die strenge Einhaltung dieser Bedingung den Lösungsraum einschränkt, wird in diesem Rahmen auch der Suchraum (teilweise stark) eingeschränkt.

In Abb. 2.3 ist auch ausschnittsweise ein Einblick in einige Kampagnen in Bezug auf die Güteklasse gegeben. Während z.B. Kampagne 1 von dieser Nebenbedingung nicht betroffen ist, betrifft diese Nebenbedingung die meisten metaheuristisch zu optimierenden Kampagnen, siehe auch Abb. 2.4.

Aufträge | Aggregate | Pufferspeicher

30.09.2009 20:06:42 Bewertung
Simulationsbeginn: 09.09.2009 09:33:30
Simulationsende: 22.09.2009 17:46:06
Dauer: 13d 08:12:36
Fitness Value: 1087079,00

AscOrderId	Walzkampagne	LotId	#Blocks	Profil	Qualität	Schnittmu...	ZS	ZEK	Lieferdatum	Fertigstellungsdatum
10	3	122860-3	3	ZU54E1A2	Normal	10x8640	ZS2	EK2	15.09.2009	11.09.2009
11	3	122969-1	10	ZU54E1A2	HSH	2x46870	ZS1	EK1	17.09.2009	12.09.2009
12	4	122939-2	30	VIVRC60	HSH	2x51700	ZS1	EK1	17.09.2009	12.09.2009
13	4	122939-1	18	VIVRC60	HSH	3x33500	ZS1	EK1	17.09.2009	12.09.2009
14	4	122915-5	39	VIVRC60	HSH	4x24596	ZS1	EK1	23.09.2009	12.09.2009
15	4	122915-1	21	VIVRC60	HSH	11x10251	ZS2	EK2	21.09.2009	12.09.2009
16	4	122915-2	8	VIVRC60	HSH	14x8535	ZS2	EK2	21.09.2009	12.09.2009
17	4	122915-3	4	VIVRC60	HSH	28x4294	ZS2	EK2	21.09.2009	12.09.2009
18	4	122915-4	3	VIVRC60	HSH	29x4121	ZS2	EK2	21.09.2009	12.09.2009
19	5	121915-161	52	VI54E2	Normal	1x120000	ZS2	EK4	14.09.2009	15.09.2009
20	5	121915-160	26	VI54E2	Normal	2x60000	ZS2	EK4	14.09.2009	15.09.2009
21	5	122963-9	5	VI54E2	Normal	4x30010	ZS1	EK1	15.09.2009	15.09.2009
22	5	121915-159	9	VI54E2	Normal	4x30000	ZS2	EK2	14.09.2009	15.09.2009
23	5	121915-162	44	VI54E2	HSH	2x60000	ZS2	EK4	14.09.2009	15.09.2009
24	5	122847-1	7	VI54E2	HSH	4x30316	ZS1	EK1	10.09.2009	15.09.2009
25	5	122820-2	3	VI54E2	HSH	9x12256	ZS2	EK2	10.09.2009	15.09.2009
26	5	122988-1	8	VI54E2	HSH	1x108000	ZS2	EK4	24.09.2009	15.09.2009
27	5	122603-3	6	VI54E2	Chrom	2x60000	ZS2	EK4	05.10.2009	15.09.2009
28	5	122851-1	11	VI54E2	Chrom	2x60000	ZS2	EK4	05.10.2009	15.09.2009
29	6	123003-1	14	VI60E2-40	Normal	3x32485	ZS1	EK1	14.09.2009	15.09.2009
30	6	123003-2	14	VI60E2-40	Normal	3x35065	ZS2	EK4	14.09.2009	16.09.2009
31	6	123006-1	7	VI60E2-40	Normal	5x23000	ZS1	EK1	17.09.2009	15.09.2009
32	6	123004-2	14	VI60E2-40	HSH	3x35066	ZS2	EK4	14.09.2009	16.09.2009
33	6	123004-1	14	VI60E2-40	HSH	3x30566	ZS1	EK1	14.09.2009	16.09.2009
34	6	undef	84	VI60E2-40	Normal	1x120000	ZS2	EK4	09.09.2009	18.09.2009
35	7	122802-2	40	VI60E1	Normal	1x120000	ZS2	EK4	24.09.2009	17.09.2009
36	7	123019-2	16	VI60E1	Normal	1x120000	ZS2	EK4	21.09.2009	17.09.2009
37	7	121890-29	7	VI60E1	Normal	1x120000	ZS2	EK4	21.09.2009	17.09.2009
38	7	121890-14	73	VI60E1	Normal	1x120000	ZS2	EK4	23.09.2009	17.09.2009
39	7	122821-4	10	VI60E1	Normal	2x60000	ZS2	EK4	03.09.2009	17.09.2009
40	7	122012-3	39	VI60E1	HSH	1x120000	ZS2	EK4	21.09.2009	18.09.2009
41	7	122821-2	20	VI60E1	HSH	2x60000	ZS2	EK4	03.09.2009	18.09.2009
42	7	121591-124	40	VI60E1	HSH	2x60000	ZS2	EK4	21.09.2009	18.09.2009
43	7	121591-126	40	VI60E1	HSH	2x60000	ZS2	EK4	23.09.2009	18.09.2009
44	7	121591-125	40	VI60E1	HSH	2x60000	ZS2	EK4	22.09.2009	18.09.2009
45	8	122858-2	218	VISAR57	Normal	2x60000	ZS2	EK4	21.09.2009	22.09.2009
46	8	122885-1	42	VISAR57	HSH	2x60000	ZS2	EK4	07.10.2009	22.09.2009
47	9	122976-1	7	ZU60E1A1	Normal	5x20000	ZS1	EK1	28.09.2009	22.09.2009
48	9	122962-1	29	ZU60E1A1	HSH	2x47400	ZS2	EK4	24.09.2009	22.09.2009
49	9	123051-2	18	ZU60E1A1	HSH	2x44400	ZS1	EK1	15.10.2009	22.09.2009
50	9	123051-1	39	ZU60E1A1	HSH	2x44400	ZS2	EK4	17.09.2009	22.09.2009
51	9	undefA	19	ZU60E1A1	Normal	1x120000	ZS2	EK4	09.09.2009	22.09.2009

Abb. 2.4: Nebenbedingung bezüglich der Einschränkung auf Grund der Güteklassen

In jeder Kampagne können sich Lose mit bis zu 4 unterschiedlichen Qualitätsgüteklassen (die Schienenhärte betreffend) befinden. Die Reihenfolge der Härte der Güteklassen, aufsteigend sortiert, beträgt:

Normal → HSH → Bainit → Chrom

In dieser Reihenfolge wird eine Kampagne üblicherweise abgearbeitet um so den Walzenverschleiß minimieren zu können.

HSH steht in diesem Zusammenhang als Abkürzung für *"Head Special Hardened"*.

3. Nebenbedingung: Lange Schienen vor kurzen Schienen: Eine weitere in das System zu integrierende Nebenbedingung besteht darin, lange Schienen vor kurzen Schienen zu produzieren, da diese Vorgabe das tatsächliche Planungsverhalten besser nachbildet und die Zahl möglicher Permutationen reduziert [27]. Zu jedem Los gibt es ein Schnittmuster, nach welchem die Blöcke in einem Los bearbeitet werden müssen. Aus diesem Schnittmuster erhält man Informationen über die Länge der zu schneidenden Schienen. Die Nebenbedingung lange Schienen vor kurzen Schienen wird in der Regel verwendet, um im Falle langer Ausschussschienen diese noch zu kürzeren Schienen schneiden zu können.

Somit kann man diese Schienen in der laufenden Produktion direkt weiterverwenden. Die Frage, die sich in diesem Zusammenhang stellt, ist, ob sich diese Nebenbedingung nicht aufweichen lässt, wenn man unter Vernachlässigung dieser Nebenbedinung ein wesentlich größeres Optimierungspotenzial erhält. Die Aufweichung dieser Nebenbedingung wird durch vom Benutzer einstellbare Toleranzgrenzen erreicht. 10% Toleranz bedeuten dabei, dass z.B. ein Los mit 28 Meter Schienen auch vor einem Los mit 30 Meter Schienen gewalzt werden kann. Durch diese Toleranzen bleibt die grobe Struktur (Lose mit 120 Meter Schienen vor Losen mit 60 Meter Schienen vor Losen mit 45 Meter Schienen) weiterhin erhalten, sofern die Toleranzgrenzen $\leq 25\%$ bleiben. Für die Simulation wird deshalb die 'lang vor kurz' Regel scharf (0% Toleranz) sowie aufgeweicht mit 20% Toleranz angewendet, siehe dazu auch die Simulationsergebnisse in Abschnitt 7.10. Der Grund dieser Aufweichung der Nebenbedingung liegt einerseits an der besseren Nachbildung des realen Planungsverhaltens, weil der Kunde in bestimmten Toleranzbereichen selbst seine Nebenbedingung verletzt. Die Verletzung betrifft v.a. kürzere Schienen und die Toleranz übertrifft in den seltensten Fällen 25%. Durch die Aufweichung dieser Nebenbedingung wird auch der Suchraum etwas vergrößert, wodurch potenziell bessere Ergebnisse möglich sind.

In den Abb. 2.3 und 2.4 kann man an der Ausgangssequenz erkennen, dass innerhalb derselben Güteklasse die Bedingung lange Schienen vor kurzen Schienen i.d.R. zur Anwendung kommt.

2.3.3 Auswirkungen auf die Simulation sowie auf die Ergebnisse dieser Arbeit auf Grund der gegebenen Nebenbedingungen

Die Nebenbedingung, welche ein Tauschverbot für ZSG Lose bedeutet, hat zur Folge, dass eine bzw. mehrere Kampagnen auf einem einzelnen Datensatz unterschiedlich stark davon betroffen sind. Während in Abb. 2.3 16 der 23 Lose der zweiten Kampagne davon betroffen sind, bevor die 2. Nebenbedingung bzw. 3 Nebenbedingung abgeprüft wird, sind die meisten Kampagnen von dieser Nebenbedingung nicht betroffen bzw. gibt es ein Tauschverbot für m Lose mit den verbleibenden $n - m$ Losen innerhalb einer Kampagne, welche aus n Losen besteht. (i.d.R.: $m << n$)

Durch die Zuordnung der jeweils anderen Sägebohrlinie zu einem ZSG Los ist insgesamt eine Verschiebung der Kapazitäten auf die einzelnen Sägebohrlinien möglich. Während in der Ausgangsreihenfolge ein ZSG Los gleichmäßig im Verhältnis 50 : 50 auf die beiden Sägebohrlinien aufgeteilt ist, kann z.B. im Rahmen einer optimierten Kampagne ein Verhältnis von 70 : 30 entstehen, was sich über die unterschiedlich starke Auslastung der Sägebohrlinien begründen lässt.

Die zweite Nebenbedingung betrifft fast jede Kampagne auf einem Datensatz. Sie beschränkt die Anzahl der zulässigen Lösungen und somit das Potenzial für die Optimierung relativ stark.

Die dritte Nebenbedingung schränkt den Suchraum noch weiter ein, sodass die Anzahl der zulässigen Lösungen unter Einhaltung aller drei Nebenbedingungen in vielen Kampagnen relativ klein, aber in der Regel nicht deterministisch lösbar, wird. Durch die Aufweichung dieser Nebenbedingung lässt sich der Suchraum wieder deutlich vergrößern.

Im Rahmen der Ergebnisse dieser Masterarbeit wird v.a. das Optimierungspotenzial der implementierten Algorithmen unter Einhaltung der ersten beiden weichen Nebenbedingungen sowie allen drei weichen Nebenbedingungen beleuchtet. Die harte Nebenbedingung muss immer eingehalten werden.

2.4 Größe des Suchraums und Komplexität

Den Optimierungsalgorithmen stehen unter bestimmten Bedingungen 2 Freiheitsgrade zur Verfügung: Einerseits der freie Austausch der Lose innerhalb einer Walzkampagne, andererseits die Zuordnung von Schienen zu Sägebohrlinien. Während der erste Freiheitsgrad für alle Lose prinzipiell anwendbar ist, steht der zweite Freiheitsgrad nur ZSG Losen zur Verfügung. Das heißt, es wird jeweils eine Kampagne für sich optimiert und danach werden die Auswirkungen dieses lokalen Optimums auf die anderen Kampagnen im System betrachtet und der dazugehörige globale Fitnesswert berechnet. Die optimierte Kampagne wird anschließend für die nächste zu optimierende Kampagne als Rahmen verwendet.

Generell werden die jeweils nachfolgenden nicht optimierten Kampagnen sowie die vorangegangenen optimierten Kampagnen zur aktuell optimierten Kampagne in Beziehung gesetzt.

Es kann während der Simulation die Anzahl der zu simulierenden Kampagnen eingestellt werden, was den Planungshorizont und die Größe des Suchraums für Testzwecke reduziert.

Ohne Berücksichtigung der Umsortiermöglichkeiten innerhalb der Produktionsstraße lässt sich die Problemgröße gemäß [19] ungefähr wie folgt beschreiben:

$(\sum_{i=1}^{i=Anzahl(Kampagnen)} Größe\ (\text{Kampagne}_\text{i})!) \cdot 2^{Anzahl(Blöcke)}$

Bei gleichzeitiger Optimierung der Kampagnen könnte man das Summenzeichen mit einem Produktzeichen vertauschen, wodurch das Problem weiter wachsen würde. Bei einem Planungshorizont von einem Monat mit ca. 50 Kampagnen und einer durchschnittlichen Kampagnengröße von 200 Blöcken ergibt sich als untere Schranke eine Suchraumgröße von ca. 10^{10^3} bzw. als obere Schranke eine Größe von ca. 10^{10^4} [19].

Kapitel 3

3 Vorstellung vielversprechender Metaheuristiken für Auftragsreihenfolgeprobleme

3.1 Metaheuristiken im Vergleich mit exakten Lösungsalgorithmen

Metaheuristiken liefern nicht zwingend das Optimum, aber in relativ kurzer Zeit - verglichen mit deterministischen Lösungen - gute Näherungslösungen [3]. Sie werden v.a. dort eingesetzt, wo der komplette Suchraum auf Grund der Komplexität nicht deterministisch abgesucht werden kann. Prinzipiell unterscheidet man zwischen Konstruktionsheuristiken, welche in jedem Schritt eine Lösungskomponente hinzufügen, und Verbesserungsheuristiken, welche Lösungen durch Änderungen schrittweise zu verbessern versuchen. Im Rahmen des sogenannten *"Greedy Prinzips"* wird von den schrittweise jeweils bewerteten Lösungskomponenten immer die beste ausgewählt. Für ein TSP wird dies z.B. im Rahmen der Nearest Neighbor Heuristik sichergestellt, da als nächstes immer der Knoten mit der kürzesten Entfernung zum aktuellen Knoten gesucht und die dazugehörige Kante eingefügt wird [34].

Weiters existiert die Unterscheidung zwischen Verfahren, die nur bessere Lösungen akzeptieren und solchen, die zwischendurch auch schlechtere Lösungen akzeptieren.

I.d.R. handelt es sich dabei um kombinatorische Optimierungsprobleme. Diese sind durch eine Menge von Instanzen charakterisiert, welche ein Paar(S, f) darstellt, wobei S eine endliche Menge aller zulässigen Lösungen ist und f die konkrete Zielfunktion darstellt. Typische kombinatorische Optimierungsprobleme stellen beispielsweise Scheduling, Routen- und Rundreiseprobleme, Pack- und Verschnittprobleme bzw. Auftragsreihenfolgeprobleme dar. Einige typische NP- harte Probleme werden in Abschnitt 3.2 näher beschrieben.

Die üblichen Lösungsverfahren stellen diverse Konstruktionsheuristiken, lokale Suchverfahren, Variable Neighbourhood Search Verfahren, Simulated Annealing, Ameisenalgorithmen und genetische bzw. evolutionäre Algorithmen dar. In der Praxis werden die verschiedensten Heuristiken miteinander kombiniert um bei relativ geringer zusätzlicher Laufzeit noch bessere Lösungen zu erhalten, die näher am Optimum liegen.

3.1.1 Definition Metaheuristik

1. Definition *Metaheuristik: "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high- quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or a construction method." (Quelle: [3])*

2. Definition Metaheuristik: *"Eine Metaheuristik ist in der Mathematik ein Algorithmus zur näherungsweisen Lösung eines kombinatorischen Optimierungsproblems. Im Gegensatz*

zu problemspezifischen Heuristiken, die nur auf ein bestimmtes Optimierungsproblem angewendet werden können, definieren Metaheuristiken eine abstrakte Folge von Schritten, die (theoretisch) auf beliebige Problemstellungen angewandt werden können. ... " (Quelle: [25])

Metaheuristiken sind also dadurch gekennzeichnet, dass sie universell für verschiedene Problemstellungen anwendbar sind, ohne das Problem näher kennen zu müssen. Diese Vielseitigkeit bereitet einerseits Vorteile, andererseits müssen die Verfahren, wenn sie sehr gute Ergebnisse liefern sollen, trotzdem mit den für das konkrete Problem geeigneten Operatoren ausgestattet werden. Metaheuristiken werden v.a. dort eingesetzt, wo es keinen bestimmten effizienten Lösungsalgorithmus gibt, also z.B. bei nicht näher bestimmten kombinatorischen Optimierungsproblemen. Als Metaheuristiken bezeichnet man insbesondere Verfahren, welche in akzeptabler Rechenzeit (auch: "laufzeiteffizient") sukzessive bessere Lösungen durch die iterative Anwendung bestimmter (Such-) Operatoren generieren [29].

Ein wichtiges Qualitätsmerkmal stellt dabei die Robustheit der Verfahren dar. Robustheit impliziert, dass in Abhängigkeit definierter Rechenzeiten sich die daraus resultierenden Lösungsqualitäten unwesentlich von der problembezogenen Aufgabenstellung unterscheiden. In diesem Zusammenhang ist die Erwähnung des "No Free Lunch (NFL)"- Theorems wichtig, da dieses theoretisch betrachtet die generelle Überlegenheit eines bestimmten Verfahrens gegenüber anderen ausschließt [3, 28].

Metaheuristiken sind insbesondere dadurch gekennzeichnet, dass sie unterschiedliche Suchstrategien kombinieren, z.B. globale und lokale Suche vereinen, wobei ein Meta- Algorithmus den eingebetteten Algorithmus steuert. Durch die Kapselung problemspezifischer Komponenten ist eine relativ leichte Übertragbarkeit auf neue bzw. andere Probleme prinzipiell gewährleistet [34].

3.2 Typische Anwendungsgebiete von Metaheuristiken

Wie bereits erwähnt, werden Metaheuristiken v.a. für kombinatorische Optimierungsprobleme angewendet, für die im Rahmen der NP-Vollständigkeit kein effizienter (polynomieller) Algorithmus existiert [10]. NP steht dabei für die Menge aller Aufgabenstellungen, die mittels nichtdeterministischen Algorithmen in polynomieller Zeit gelöst werden können. Nichtdeterminismus bedeutet dabei, dass sich bei der Wahl unterschiedlichster Varianten das Verfahren für die richtige Variante entscheidet.

P bezeichnet alle Problemstellungen, die mittels deterministischen Verfahren polynomiell gelöst werden können. Polynomiale Algorithmen zeichnen sich dabei durch ihre beschränkte Laufzeit $O(n^K)$ aus, wobei K eine Konstante darstellt [10].

NP- vollständige Probleme (NP-complete, NPC) sind definiert als solche Probleme, die (a) aus NP sind und (b) in die sich alle anderen Probleme aus NP in polynomieller Zeit transformieren lassen ([10]). Als "NP-schwierig" bzw. "NP-hart" definiert man Probleme, auf die sich sämtliche NP Probleme polynomiell reduzieren lassen, wobei deren Zugehörigkeit zu NP nicht bewiesen werden kann [10]. D.h., dass für diese Aufgabenstellungen keine Polynomialzeit-Algorithmen bekannt sind.

In den folgenden Unterabschnitten werden einige wichtige NP- schwierige Probleme vorgestellt und mit dem Logistik Kontext in Zusammenhang gebracht.

3.2.1 Travelling Salesman Problem (TSP)

Das TSP ist das bekannteste Beispiel für ein NP- hartes Problem. Die Aufgabe besteht darin, dass alle Städte genau jeweils einmal während einer Rundreise besucht werden, wobei die Städte allesamt durch Kanten, deren Gewichtung sich über die Distanzen der einzelnen Städte voneinander berechnen, verbunden sind und die Gesamtdistanz der Rundreise zu minimieren ist. Außerdem muss die Rundreise in der Stadt enden, in der sie begonnen hat. Dabei werden die Städte als Knotenpunkte (V) und die verbindenden Straßen als Kanten (E) betrachtet, wobei jede Straße eine positive Gewichtung ($d_{ij} > 0$) hat [28].

Formaler wird das Problem durch einen vollständigen Graphen G beschrieben. $G = (V, E)$ mit Kantengewichten $d_{ij} > 0 \; \forall (i,j) \, \varepsilon \, E, \; i \neq j$ (Quelle: [3])

Gesucht wird ein sogenannter *Hamilton´scher Kreis* mit geringstem Kanten(summen)gewicht ([3]).

Auf Grund der Tatsache, dass jeder Knoten exakt einmal besucht werden darf, werden die Lösungen eines TSP auch als *"Permutationen"* bezeichnet [14].

Hierarchische Unterklassen stellen Euklidische Distanzen sowie das symmetrische TSP und das asymmetrische TSP dar. Während für das symmetrische TSP für alle Verbindungen gilt, dass $d_{ij} = d_{ji}$ gibt es beim asymmetrischen TSP mindestens eine Verbindung, für die dieses Kriterium nicht zutrifft [3].

Die Ähnlichkeit des TSP zu dem PFSP im konkreten Produktionsproblem ist insofern gegeben, als ein PFSP auf Grund seiner Charakteristik auch eine Permutation von N Aufträgen (bzw. konkreter: Losen) darstellt mit dem Unterschied zum TSP, dass der letzte Auftrag nicht dem ersten Auftrag entspicht, wohingegen bei einem TSP der letzte Knotenpunkt dem ersten Knotenpunkt in der Rundreise entspricht.

3.2.2 Quadratic Assignment Problem (QAP)

Im Rahmen eines quadratischen Zuordnungsproblems sind zwei $n \times n$ Matrizen A und B gegeben, wobei hier die Doppelsumme $\sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} b_{\varphi(i),\varphi(j)}$ zu minimieren ist und φ eine Permutation darstellt [15]. Gesucht ist eine Permutation φ, im Rahmen dieser die Doppelsumme minimal wird. Ohne nähere Informationen ist dieses Problem schwer in akzeptabler Zeit zu lösen. Anwendungsbeispiele liegen v.a. im Bereich der Fabrikplanung bzw. Logistiksystemgestaltung. So kann Matrix A das Verkehrsaufkommen zwischen einzelnen Gebäuden bzw. Fabriken enthalten, während in Matrix B die Distanzen zwischen den Fabriken gespeichert sind [15].

Ein weiteres Anwendungsbeispiel im Bereich Fabrikplanung definiert in der Matrix A die zu transportierenden Mengen zwischen den einzelnen n Maschinen in einer Fabrikshalle, während die Matrix B die Entfernungen zwischen den n zur Aufstellung der Maschinen in Frage kommenden Standorten beinhaltet. Wenn die Maschinenfolge keine Rolle spielt, liegt ein klassisches QAP vor [15].

In einem weiteren Beispiel müssen n Fabriken zu n Standorten zugeordnet werden, wobei in einer Matrix die auszutauschenden Materialmengen gespeichert werden und in der anderen die Distanzen zwischen den Orten. Im Rahmen der optimalen Zuordnung aller Fabriken zu jeweils einem Standort ist so zu minimieren, dass die benötigte Zeit (ausgedrückt durch die Distanz) pro Material minimal ist [28]. Dieses Problem kann auch anhand eines Graphen in Abb. 3.1 (Quelle: [28]) veranschaulicht werden. Wichtig ist in diesem Zusammenhang, dass jeweils alle Fabriken einem Standort bereits zugeordnet sein müssen, bevor man die Zielfunktion vollständig berechnen kann, da man vorher nur Teilkosten berechnen kann.

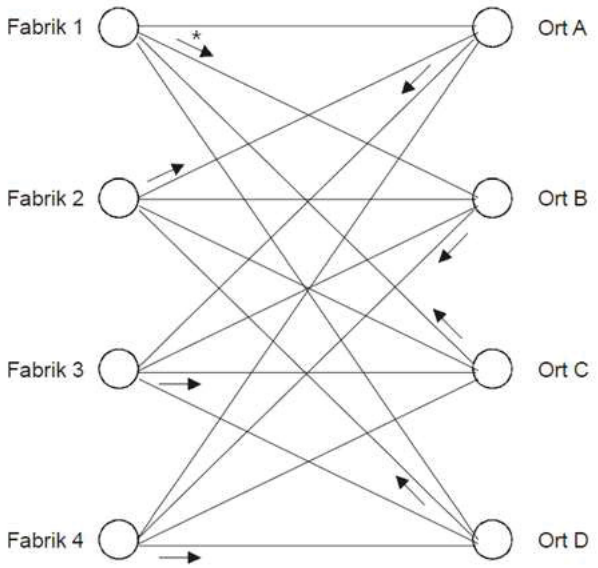


Abb. 3.1: Darstellung eines QAP als Graph

Das QAP stellt eines der härtesten praktischen kombinatorischen Optimierungsprobleme dar, neben Tabu Suche hat sich hier v.a. das Max-Min Ant System (MMAS) erfolgreich bewährt [3].

3.2.3 Vehicle Routing Problem (VRP)

Das VRP stellt eine Vereinigung der Problemtypen eines TSP und eines Bin Packing dar [30]. Bei Bin Packing sind n gewichtete Items gegeben, sowie beliebig viele Bins mit Kapazität W, gesucht ist dabei eine Packung in möglichst wenige Bins [3].

Der bekannteste Vertreter eines VRP ist das sogenannte *"Capacitated Vehicle Routing Problem"* (CVRP) Dieses ist durch eine Fahrzeugflotte mit einheitlicher Kapazität sowie ein gemeinsames Warenlager und eine bestimmte Anzahl an Kundenaufträgen mit Bedarfen an Waren charakterisiert. Gesucht ist die Routenzusammenstellung mit den geringsten Kosten, welche die gegebenen Nebenbedingungen einhält. Die Funktion muss entsprechend der Anzahl an Fahrzeugen sowie Länge und Dauer der Routen gewichtet werden [31]. Dieses NP- schwierige Problem ist nicht nur theoretisch, sondern v.a. auch auf Grund der Vielzahl an Anwendungen im Bereich der operativen Transportplanung, besonders interessant. Neben der Planung der Routen müssen den einzelnen Routen Fahrzeuge zugewiesen werden und ggf. weitere Nebenbedingungen eingehalten werden wie z.B. die Kapazität, Beladung, Zeitfenster und Arbeitszeiten.

Formal wird das (Asymetric) Capacitated Vehicle Routing durch einen vollständigen Graphen $G = (V, A)$ mit Knoten $V = \{0, ..., n\}$ und Kanten A beschrieben. Die Knoten $\{1, ...n\}$ stellen die Kunden dar, während Knoten 0 das Depot ist. Die Kosten c_{ij} bezeichnen die Reisekosten, welche sich aus der Distanz oder/und der Leichtgängigkeit des Weges ergeben. Jeder Knoten $i \geqslant 1$ hat einen bestimmten Bedarf d_i ($d_i > 0$) an Waren. Ausgangspunkt stellen K identische Fahrzeuge mit Kapacität C im Depot dar [31].

Im Rahmen der K gesuchten Routen mit minimalen Kosten muss sichergestellt werden, dass jede Route das Depot besucht, jeder Knoten der Route exakt einmal besucht wird, sowie die Summe des Bedarfs der besuchten Knoten nicht die Fahrzeugkapazität C übersteigt [31].

Falls zum CVRP zusätzlich auch Zeitfenster hart oder weich berücksichtigt werden müssen, wird jeder Knoten mit einer bestimmten Servicezeit betrachtet. Im Rahmen dessen müssen zusätzlich auf jeder Kante zu den bestehenden Reisekosten c_{ij} die Fahrtzeiten t_{ij} berücksichtigt werden. Außerdem müssen bei den Knotenpunkten zusätzlich die Servicezeiten s_i gespeichert werden. Jedem Knoten wird

ein Zeitfenster bestehend aus frühestem Abfahrtszeitpunkt und spätester Ankunftszeit zugeordnet, innerhalb dessen operiert werden kann.

Neben den bekannten evolutionären oder ameisenbasierten Lösungsstrategien kommen beim CVRP auch spezielle Konstruktionsheuristiken zum Einsatz. Der Savings- Algorithmus beruht auf dem Savings Kriterium, anhand dessen überprüft wird, ob zwei Routen unter Einhaltung der Kapazitätsnebenbedingung zusammengefügt werden können, um eine Einsparung zu erzielen.

Zwei-Phasen Heuristiken stellen z.B. der Sweep Algorithmus sowie der Fisher und Jaikumar Algorithmus dar, welche die Klasse der Cluster-first, Route-second Heuristiken vertreten. Der Algorithmus von Beasley entspricht der Klasse Route-first, Cluster-second.

Neben den für das TSP klassischen Verbesserungsheuristiken kommen auch spezielle routenübergreifende Verbesserungsverfahren zum Einsatz, wobei man drei Kategorien unterscheidet: Während bei String Cross zwei Routen durchgetrennt und anschließend gekreuzt werden, tauscht man bei String Exchange zwei Knotenketten zwischen zwei Routen aus, während String Relocation darauf beruht, dass eine Knotenkette von einer Route in eine andere Route verschoben wird [31].

3.3 Lösungsraumanalyse

Bei der Verwendung von Metaheuristiken empfiehlt es sich generell, die Lösung zuerst in einen oder mehrere vielversprechende(n) Bereich(e) zu lenken und diesen bzw. diese anschließend gezielter zu untersuchen [9]. Während die Identifikation dieses vielversprechenden Bereichs durch eine Metaheuristik wie z.B. einem genetischen Algorithmus (GA) oder mittels Ant Colony Optimization (ACO) bzw. mittels Simulated Annealing (SA) erreicht wird, werden diese Lösungen danach mittels eines lokalen Suchverfahrens optimiert. Derartige Ansätze werden in der Literatur auch als *"hybride Ansätze"* bezeichnet. Im Rahmen der eingesetzten Metaheuristik sollte dabei eine frühzeitige Konvergenz des Verfahrens auf ein lokales Optimum hin vermieden werden, da man aus dieser gefundenen Lösung oft nur mehr schwer herauskommt, v.a. wenn man SA bzw. einen GA mit hohem Selektionsdruck verwendet. Es stellt sich dabei die Frage, ob ein globales Optimum in unmittelbarer Nähe eines lokalen Optimums liegt oder nicht. Jedenfalls wurde in Experimenten im Rahmen des TSP die häufige *"Existenz des großen Tals"* festgestellt. Dieses beschreibt die Möglichkeit, innerhalb weniger Züge mit einem lokalen Suchverfahren von einem lokalen Optimum in ein anderes zu gelangen. Es liegt also vor, wenn die lokalen Optima in einem (meist vielversprechenden) Suchbereich nahe zusammenliegen. Zu beachten ist dabei, dass dieses große Tal nicht grundsätzlich vorliegen muss [9].

3.3.1 Analyse der Fitnesslandschaft eines Optimierungsproblems

Grundsätzlich spannt der Definitions- und Wertebereich einer Zielfunktion einen $n+1-$dimensionalen Suchraum auf (n entspricht der Anzahl an Variablen, von denen die Fitnessfunktion abhängt), den man sich als Fitnesslandschaft bzw. auch als Gebirge vorstellen kann. Dabei entspricht der Fitnesswert der jeweiligen Höhe einer Lösung, siehe auch Abb. 3.2 dazu. Während im Rahmen der Minimierung das tiefste Tal zu finden ist, sucht man bei Maximierungsproblemen nach dem höchsten Berg. Im Gegensatz zu Abb. 3.2 ist das Fitnessgebirge im konkreten Produktionsproblem einerseits diskret, außerdem können die Nebenbedingungen den Lösungsraum so weit einschränken, dass der nächste diskrete Punkt in der Fitnesslandschaft sehr weit vom letzten Punkt entfernt sein kann. In Abb 3.2 (Quelle: [40]) ist außerdem ersichtlich, dass der Weg des steilsten Gradienten vom Ausgangspunkt A in Richtung B nicht das globale Optimum findet, sondern lediglich ein lokales Optimum. Erst der deutlich längere Weg von A nach C findet das globale Optimum. Dieser Punkt C ist jedoch nur durch das Überwinden eines Tals (bzw. je nach Fitnesslandschaft auch mehrerer Täler) erreichbar. Daher empfiehlt es sich in diesem Zusammenhang eine Optimierungsstrategie zu verwenden, welche in der Lage ist, diverse Täler überwinden zu können. Eine Fitnessfunktion mit mehreren lokalen Optima wird als *"multimodal"* bezeichnet [40].

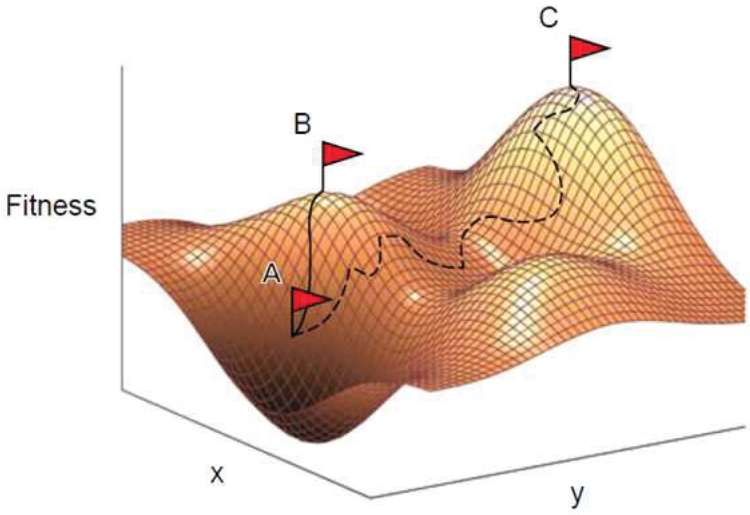


Abb. 3.2: Optimierung in einer Fitnesslandschaft

3.4 Metaheuristiken abseits von genetischen und evolutionären Algorithmen

3.4.1 Eine Heuristik zur Gesamtdurchlaufzeitminimierung eines PFSP

Die folgende Heuristik von Laha und Sarin ([12]) basiert auf der konstruktiven Heuristik von Framinan und Leisten ([11]), wobei in Schritt 4 eine Modifikation getätigt wurde. Beide Heuristiken haben eine Gesamtlaufzeit von $O(N^4 \cdot M)$. Auf Grund dieser vergleichsweise langen Laufzeit ist diese Heuristik für das in dieser Arbeit betrachtete Auftragsreihenfolgeproblem ungeeignet. Dennoch wird dieses Verfahren auf Grund der Lösungsqualität erwähnt, eine genaue Beschreibung dazu findet sich in Alg. 3.1.

In Schritt 4 wird eine Art *"iteratives Insertion Verfahren"* verwendet, was eine effizientere Nachbarschaft generiert als der paarweise Tausch in Step 4 bei der Heuristik von Framinan und Leisten. Die von Laha und Sarin vorgestellte Heuristik leistet statistisch bessere Ergebnisse als jene von Framinan und Leisten.

Algorithmus 3.1 Heuristik von Laha und Sarin

Step 1: for each job i, find the total processing time T_i which is given by
T_i=$\sum_{j=1}^{m} T_{ij}$ for all $i = 1, 2, ..., n$.
Step 2: sort the jobs in increasing order with respect to the sum of their processing times on all machines T_i .
Step 3: set $k = 2$. Select the first two jobs from the sorted list and select the better between the two possible sequences. After this selection set $k = 3$.
Step 4: for $k = 3$ to n do the following procedures:
Insert the k^{th} job on the sorted list into k possible positions of the $(k-1)$-job current sequences, and select from these a k-job partial sequence with the best total flow time value. Designate this as a $k-$job current sequence. Place each job (except for the k^{th} job of the sorted list) of this sequence into its $(k-1)$ positions and select the best $k-$job sequence having the least total flow time value from among those generated. This becomes the next $k-$job current sequence.
Step 5: if $k = n$, then STOP

3.4.2 Lokale Suche (LS)

Definition: Ein lokales Minimum (bzw. Maximum) x in Bezug auf eine Nachbarschaftsstruktur $N(x)$ ist eine Lösung x für die gilt: $f(x) \leqq f(x')\forall x'\epsilon N(x)$ [3]

Unabhängig von genetischen Algorithmen kann man auch lokale Suchalgorithmen zur Lösung eines PFSSP verwenden. Es besteht auch die Möglichkeit, die Lösung von genetischen Algorithmen als Ausgangslösung für die lokale Suche zu verwenden. Der mögliche Qualitätsvorteil bedingt eine höhere Gesamtlaufzeit des Programms. In der Praxis bedient man sich dabei zumeist der lokalen Nachbarschaftssuche. Diese besteht im Wesentlichen aus einer Zielfunktion $f(x)$, einer Ausgangslösung, einer Nachbarschaftsstruktur, einer Schrittfunktion und einer Abbruchbedingung [3]. Im konkreten Problem stellt die Ausgangslösung die initiale Reihenfolge des Kunden dar. Die Zielfunktion bleibt dabei die unveränderte Fitnessfunktion. Das Basisschema einer lokalen Suche ist in Alg. 3.2. abgebildet (Quelle: [3])

Algorithmus 3.2 Lokale Suche

1: choose an initial tour f (initial solution);
2: **while** there exists $g \epsilon$ N_k(f) with $\omega(g) < \omega(f)$ ***do***
3: choose $g \epsilon$ N_k(f) with $\omega(g) < \omega(f)$;
4: $f \leftarrow g$;

3.4.2.1 K-opt lokale Suche

Ausgehend von einer Lösung eines TSP kann man k Kanten (e_i, e_j,...) kurzfristig aus der Rundreise entfernen und diese danach wieder an einer anderen Position einfügen. Die dazugehörige Nachbarschaft $N_k(f)$ wird "$k-$change neighbourhood" genannt. Eine Rundreise mit minimal bewerteten Kanten ist "$k-$optimal" [3].

Generell unterscheidet man 2 Strategien in Bezug auf die Schrittfunktion, entweder die nächste bessere Lösung zu wählen ("first improvement" bzw. "next improvement") oder eine Rundreise mit minimaler Gewichtung ("steepest descent" bzw. "best improvement"). Die Wahl kann dabei einen starken Einfluss auf die Performance (trade-off zwischen Lösungsqualität und Laufzeit) haben [3]. Ein größerer Wert von k ergibt i.A. eine bessere Lösung, die Laufzeitkomplexität steigt mit zunehmendem k und beträgt allgemein 2 oder 3 [3]. Lin schlägt für k den Wert 3 vor, angewandt auf das konkrete Produktionsproblem eignet sich ein Wert von 2 eher zu Gunsten der geringeren Laufzeit. Das Maß zur Verbesserung wird als $\delta(g)$ bezeichnet. $\delta(g)= \omega(f)- \omega(g)= \omega(e_i) + \omega(e_j) - \omega(e_{i'}) - \omega(e_{j'})$

Man setze $\delta := max\{\delta(g) : g\epsilon N_2(f)\}$. Für jedes $\delta > 0$ wird die jeweils aktuelle Rundreise f durch die verbesserte Tour g ersetzt. Der Pseudo Code des 2− Opt Algorithmus von Croes (1958) ist in Alg. 3.3. (Quelle: [14]) ersichtlich.

Algorithmus 3.3 2-Opt Algorithmus

1: **repeat**
2: $\delta \leftarrow 0, g \leftarrow f$;
3: **for** $h \in N_2(f)$ **do**
4: **if** $\delta(h) > \delta$ **then** g$\leftarrow h$; $\delta \leftarrow \delta(h)$ **fi**
5: **od.**
6: $f \leftarrow g$;
7: **until** $\delta = 0$

Der 2 Opt Algorithmus findet eine 2- optimale Rundreise g als finale Lösung des TSP. Die erhaltene Lösung muss nicht optimal sein, es ist v.a. möglich, dass der Algorithmus in einer schlechten Nachbarschaft stecken bleibt. Die erhaltene k-optimale Lösung garantiert, dass durch das Ersetzen von k Kanten keine andere kürzere Tour mehr gefunden werden kann. Derselbe Algorithmus kann auch auf das konkrete PFSP angewendet werden.

Wird der 2-Opt Algorithmus für das konkrete Produktionsproblem für die besten m Individuen des GA angewandt, entspricht dies einer "multi-start" lokalen Suche, was einer Laufzeit von $m \cdot n^k$ entspricht, wobei m vom Endbenutzer einstellbar ist und $k = 2$ gesetzt wird.

Beim 2.5-opt-Austauschschritt werden 3 Kanten entfernt, wobei auch die Stadt, die von den 2 zu entfernenden Kanten umgeben ist, entfernt und an einer anderen Stelle wieder eingefügt wird. Die neue Rundreise wird durch die Einfügung zweier neuer Kanten zur verschobenen Stadt sowie einer Kante zwischen den beiden Teilen der Rundreise gebildet [33]. Der 2.5-opt Tausch ist in Abb. 3.3 (Quelle: [33]) dargestellt.

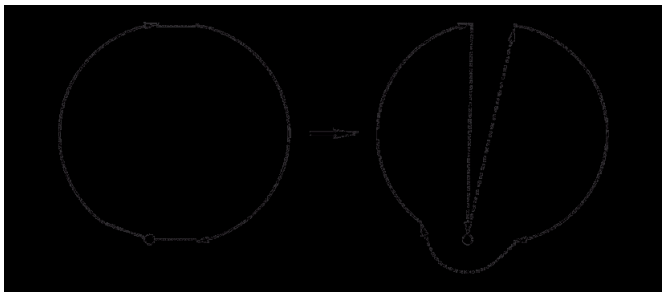


Abb. 3.3: 2.5-opt Austauschschritt

Im Zuge der verschiedensten $k-$ opt Algorithmen können auch bestimmte Beschleunigungstechniken verwendet werden. *Don´t Look Bits* (DLB) beschränkt die Suche auf vielversprechende Austauschoperationen. Dazu wird in jeder Stadt ein *Don´t Look Bit* mitgeführt, der kennzeichnet, ob der Knoten für eine Austauschoperation geeignet ist oder nicht. Für jede Stadt, für die nach einem lokalen Suchschritt keine Verbesserung erreicht werden konnte, wird ein *Don´t Look Bit* gesetzt, wodurch sie bis zur Rücksetzung des DLB nicht mehr in Betracht genommen wird. Für den Fall einer verbesserten Lösung werden die DLB aller beteiligten Städte zurückgesetzt [33, 34].

Die *fixed-radius* Methode schränkt die Suche dadurch ein, dass nur Städte innerhalb eines bestimmten Radius rund um die auszutauschende Stadt betrachtet werden. Der Radius richtet sich nach der Distanz zwischen der auszutauschenden Stadt und ihre aktuellen Nachbarstadt in der Rundreise. Es werden nur die Nachbarschaftsstädte berücksichtigt, welche innerhalb des betrachteten Radius liegen[33]. Diese Methode ist mit dem $\alpha-$Wert bei Insertion Search vergleichbar, der die Austauschreichweite steuert.

Das Hauptproblem der lokalen Suche besteht darin, dass in der Regel lokale Optima gefunden werden. Es besteht die Möglichkeit, die Nachbarschaft zu vergrößern bzw. eine multi-start lokale Suche durchzuführen . Jede Erweiterung des Suchraums bedingt prinzipiell eine Laufzeiterhöhung [3].

3.4.2.2 Insertion Search (IS)

Im Rahmen der Vorstellung eines genetischen hybriden lokalen Suchalgorithmus wird das Insertion Search Verfahren näher erläutert.

Nach einigen Versuchen sind die Autoren von [17] der Meinung, dass sich Insertion Search für GA besser eignet als pairwise exchange. Als Analogie dazu wurde bereits in 3.2.1 bei der Heuristik von Laha und Sarin festgestellt, dass Insertion Search für ein PFSP in Bezug auf eine Nachbarschaft bessere Ergebnisse bringt als pairwise exchange. Im Folgenden wir dieses lokale Suchverfahren, dessen Pseudocode in Alg. 3.4. ([17]) ersichtlich ist, näher beschrieben.

Für ein Chromosom, welches aus n jobs besteht, existieren prinzipiell $n \times n - 1$ Möglichkeiten im Rahmen der Anwendung des Insertion Operators. Der Parameter α definiert die Reichweite, in welche ein aus einer Auftragsreihenfolge entnommener Job wieder eingefügt werden kann. Während Π das Chromosom (die Basislösung) repräsentiert, stellt $\Pi(i)$ den Job an der $i-$ten Position im Chromosom dar. Zuerst wird eine Suchliste mit der Chromosomenlänge n erzeugt, wobei in dieser Liste die Indizes $\{1, 2, ...n\}$ eingetragen sind (Zeile 1). Aus dieser Suchliste wird zufällig der Index p entnommen. (Zeile

2) Ausgehend von dieser Position p wird der Job an $p \pm \alpha$ Stellen wieder im Chromosom eingefügt (Zeile 5). Jede dieser $p \pm \alpha$ Stellen stellt eine neue Lösung dar. Nachdem alle $2 \cdot \alpha$ Lösungen für diese Position p generiert worden sind, wird erneut ein Index p aus der Suchliste entnommen, solange diese noch nicht leer ist. (Zeile 2). Insgesamt können somit $2 \cdot n \cdot \alpha$ Lösungen erzeugt werden, da die Suchliste aus insgesamt n zu entnehmenden Positionen entsteht. C steht für die zu optimierende Eigenschaft bzw. Funktion, im Artikel also die Gesamtdurchlaufzeit bzw. im Rahmen eines GA generell die Fitness. Für jede bessere gefundene Lösung wird eine neue Suchliste initialisiert und das Verfahren wiederholt sich (Zeile 8). Dies ist allerdings eine Möglichkeit, die nicht notwendigerweise umgesetzt werden muss. Man könnte stattdessen auch die bestehende Suchliste abarbeiten und danach abbrechen bzw. im Falle einer gefundenen besseren Lösung für diese nachfolgend nocheinmal das Verfahren anwenden. Die konkrete Umsetzung hängt damit auch vom Ausmaß der gewünschten Intensität der lokalen Suche vom Benutzer ab. Im Folgenden wird die Insertion Search Prozedur am Beispiel des Kriteriums der Minimierung der Gesamtdurchlaufzeit (total flowtime minimization) näher beschrieben ([17]).

Algorithmus 3.4 Insertion Search (Π, C, α)

1: set the search list SL $\leftarrow \{1, 2, ..., n\}$;
2: **if** (SL ! = empty)
3: randomly choose p from SL and remove p from SL;
3: **else**
4: **stop** and return Π and C;
5: execute insertion operators Ins(p, k) on Π for $k = p - 1$, $p - 2$,...,max$(p - \alpha, 1)$
and $p + 1$, $p + 2, ..., min(p + \alpha, n)$;
6: calculate the total flowtime after each insertion operator has been applied and choose the best one;
7: **if** ((total flowtime of the best solution) $< C$)
8: let Π be the best solution and C be the total flowtime of the best solution;
9: reset the search list SL in row 1 and continue from here;
10: **else** (go to step 2;)

Es ist leicht zu erkennen, dass dieses Verfahren generell eine relativ kleine Nachbarschaft durchsucht und in einem lokalen Optimum hängen bleiben kann [17]. Dafür muss - in Bezug auf einen GA - $2 \cdot n \cdot \alpha$ Mal die Fitness neu berechnet werden, was bei einem im Vergleich mit n relativ kleinem α einer Laufzeit von $C \cdot n$ entspricht. Sofern α eine Größe von $n/2$ erreicht, vergrößert sich die Laufzeit zu $C \cdot n^2$. Der Aufwand dieses Verfahren ist damit, sofern α klein genug gewählt wird, relativ gering verglichen mit z.B. dem 2 Opt Algorithmus.

Im Rahmen eines genetischen TSP hat sich Insertion Search als laufzeiteffizientes lokales Suchverfahren herausgestellt, v.a. in Kombination mit geringeren Populationen und geringeren Generationenanzahlen. Bei Insertion Search bleibt auch die Struktur des Chromosoms besser erhalten als z.B. beim 2 Opt Algorithmus.

3.4.2.3 Iterated Local Search (ILS)

Iterated Local Search (dt.: Iterierte lokale Suche) stellt die einfachste Strategie dar lokal optimalen Lösungen zu entkommen. ILS verwendet Mutationen ähnlich dem Diversifikationsmechanismus in Tabu Search [34]. ILS stellt einen Algorithmus mit Populationsgröße 1 dar und arbeitet ausschließlich mit Mutationen. Der Pseudo Code für das Verfahren ist in Alg. 3.5 ([33]) beschrieben.

Algorithmus 3.5 Iterated Local Search

```
1: s = GenerateInitialSolution() ;
2: s* = LocalSearch(s);    // s* is the best solution after the local search has been applied for s
3: while (! terminated) {
4:     s' = Perturbation (s*);     // s' is the perturbated solution received from s*
5:     s'' =LocalSearch (s');     // s'' is the best solution after the local search has been applied for s'
6:     if (accept(s'', s*, history) );     // if s'' is better than s* → s'' is accepted (otherwise not)
7:         s* = s'';     // update
}
```

Auf die initialisierte (z.B.) Rundreise s (Zeile 1) wird lokale Suche angewendet und dadurch die lokal optimale Lösung $s*$ erzeugt (Zeile 2). Innerhalb der while Schleife wird eine perturbierte Lösung s' (Zeile 4) ausgehend von der lokal optimalen Lösung erzeugt (Zeile 3). Diese Lösung s' , welche sich nicht im Attraktionsbasin von $s*$ befinden sollte, wird mittels lokaler Suche wieder in eine lokal optimale Lösung s'' überführt (Zeile 5). Das Akzeptanzkriterium, welches auf verschiedenste Art und Weise implementiert werden kann, entscheidet über Annahme oder Ablehnung von s'' [33] (Zeile 6). Das Akzeptanzkriterium kann beispielsweise als better Akzeptanzkriterium bzw. wie bei SA als Metropolis Kriterium ausgeführt sein oder auch mit einem Gedächtnis arbeiten. Das better Akzeptanzkriterium führt zur Bezeichnung *"lokaler Meta Suche"*, während ILS basierend auf dem *Metropolis Kriterium* als *"Simulated Annealing Meta Suche"* bekannt ist. *"Random Walks"* akzeptieren immer die neue Lösung s'' und sind unter *"zufälliger Meta Suche"* bekannt [34].

Die Abbruchbedingung der while Schleife kann auf verschiedenste Art und Weise realisiert werden. Während es üblich ist, die Suche abzubrechen, wenn sich nach vielen Perturbationszügen und deren lokal optimierten Nachbarschaften nur mehr wenig an der Fitness ändert, wird ILS in Bezug auf das konkrete Produktionsproblem ähnlich dem Simulated Annealing im Rahmen der Erstarrungstemperatur abgebrochen. Dabei muss außerdem erwähnt werden, dass (in den meisten Fällen) nicht die komplette Nachbarschaft untersucht werden kann, sondern lediglich die vom Benutzer eingestellte Anzahl an Bewertungen je Nachbarschaft durchgeführt wird. Nähere Informationen zum Ablauf des Algorithmus befinden sich in 6.3.6.

Die Randomisierung der Perturbation führt zu einer zufälligen Suche. Eine Verbesserung stellt die *geführte adaptive iterierte lokale Suche* dar (GAILS), im Rahmen derer ein *"Lokaler Suche Agent"* (LSA) die Suche ausführt, welcher Informationen über die Umwelt erhält und Aktionen ausführen kann um die Umwelt zu verändern. Dem LSA können beispielsweise Informationen über die Lösungsgüte bzw. statistische Informationen zur Verfügung stehen [33]. Konkretere Erläuterungen bezüglich der GAILS Methode würden den Umfang dieser Arbeit bzw. dieses Unterabschnitts sprengen und sind daher in [33] zu finden.

In vielen ILS Implementationen wird als Perturbation der sogenannte *"Double-Bridge-Zug"* angewendet, der einen Austauschschritt in einer 4-opt-Nachbarschaft darstellt [33]. Dabei werden 4 Kanten aus der Rundreise entfernt. Die 4 daraus entstehenden Teilstücke werden anschließend wieder zu einer vollständigen Rundreise zusammengefügt, welche dann in der Orientierung ABDC durchlaufen wird, siehe dazu auch Abb. 3.4 (Quelle: [33]).

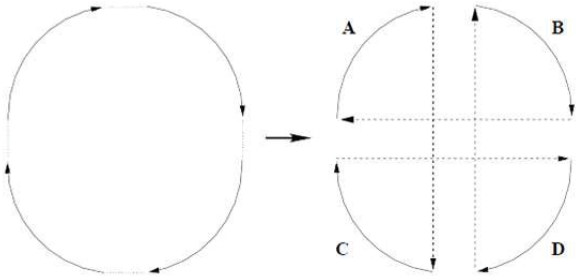


Abb. 3.4: Double-Bridge Zug

3.4.2.4 Weitere Nachbarschaftsoperatoren für lokale Suche

Als Nachbarschaft werden alle Auftragsreihenfolgen bezeichnet, welche durch die Anwendung eines lokalen Nachbarschaftsoperators generiert werden können [9].

Exchange (EX): Der exchange Nachbarschaftsoperator vertauscht einen Auftrag bzw. allgemeiner ein Element in z.B. einer Permutation mit einem anderen Element. Dadurch ist die Zulässigkeit der Lösung in jedem Fall gegeben, sofern keine speziellen harten Nebenbedingungen verletzt werden. Konkret wird dabei jeder Auftrag mit jedem anderen Auftrag ausgetauscht. Die Gesamtgröße der Nachbarschaft ergibt sich damit zu $|EX(\pi^0)| = \frac{N\cdot(N-1)}{2}$ ([9]). Der EX Operator kommt bei $k-$optimalen Nachbarschaften zur Anwendung.

Adjacent Pairwise Interchange (API): API vertauscht jeden Auftrag mit seinem direkt benachbarten Auftrag. Die Nachbarschaft stellt eine Teilmenge der EX- Nachbarschaft dar und besitzt folgende Größe: $|API(\pi^0)| = N - 1$ ([9])

Forward Shift (FSH) / Backward Shift (BSH): Während FSH einen Auftrag von seiner aktuellen Positon auf jede größere mögliche Position verschiebt, gilt dasselbe analog für BSH. Die Größe dieser Nachbarschaften ergibt sich jeweils zu $N \cdot (N-1)/2$ [9].

Double Shift (DSH): DSH verschiebt ein Element an eine beliebig wählbare andere Position. Die DSH Nachbarschaft ergibt sich durch das aufeinanderfolgende Generieren der FSH und BSH Nachbarschaft und verfügt über eine Größe von $N \cdot (N-1)$ Auftragsreihenfolgen [9].

Inversion (INV): Innerhalb der Ausgangsreihenfolge wird eine Teilreihenfolge bestimmter Länge invertiert. Dabei wird eine Teilauftragsreihenfolge selektiert, invertiert und anschließend wieder in die Gesamtreihenfolge eingefügt. Die INV Nachbarschaft entsteht durch die Inversion aller möglichen Teilreihenfolgen in der Permutation. Die Nachbarschaft verfügt dabei über eine Größe von $N \cdot (N-1)/2$ Auftragsreihenfolgen [9].

Der Insertion Operator wurde bereits in 3.3.2.2 angesprochen.

3.4.3 Variable Neighborhood Search (VNS)

Das von Hansen und Madlenovic vorgeschlagene Verfahren beruht auf mehreren Nachbarschaftsstrukturen, wobei ein lokales Optimum einer Struktur nicht das globale Optimum darstellen muss. Das globale Optimum ist das lokale Optimum bezüglich aller anderen Nachbarschaftsstrukturen, wobei für viele Probleme lokale Optima relativ nahe beieinander liegen [3, 20]. Die Nachbarschaftstrukturen werden systematisch und deterministisch ausgewechselt, siehe auch Alg. 3.6 (Quelle: [20]).

Algorithmus 3.6 Variable Neighbourhood Descent (VND)

```
1: create initial solution T;
2: i = 1;
3: while i ≤ k do
4:     find best neighbouring solution T '∈ N_i(T);
5:     if T ' better than best solution found so far then
6:         save T ' as new best solution T;
7:         i = 1;
8:     else i = i + 1;
```

Dabei müssen zuerst die Nachbarschaftstrukturen N_i defininiert und initialisiert werden. Als Schrittfunktion wird best oder next improvement gewählt. Die gefundene Lösung ist für alle Nachbarschaftstrukturen lokal optimal.

Reduced Variable Neighbourhood Search: Mittels Reduced Variable Neighbourhood Search (RVNS) kann man lokalen Optima besser entkommen. Die Idee basiert auf einer zufälligen Auswahl von Nachbarlösungen unter Berücksichtigung, dass die Nachbarschaften immer größer werden. Diese zufällige Auswahl ist auch als *"shaking"* bekannt [3, 20]. Dieses "shaking" macht es unmöglich zu garantieren, dass die bisher gefundene Lösung das (lokale) Minimum in Bezug auf alle anderen Nachbarschaftsstrukturen darstellt [20]. Sehr große Nachbarschaftsstrukturen können i.d.R. nicht vollständig durchsucht werden, die Strukturen sind generell nach Größe bzw. Entfernung ihrer Lösungen von der Basislösung sortiert [3]. Das Verfahren ist in Alg. 3.7 abgebildet (Quelle: [20]).

Algorithmus 3.7 Reduced Variable Neighbourhood Search (RVNS)

```
1: create initial solution T;
2: while termination condition not met do
3:     i = 1;
4:     while i ≤ k do
5:         randomly select a solution T ' ∈ N_i(T);
6:         if T ' better than best solution found so far then
7:             save T ' as new best solution T;
8:             i = 1;
9:         else i = i + 1;
```

RVNS wird v.a. dann angewendet, wenn die Durchsuchung der kompletten Nachbarschaft zu (laufzeit-) aufwendig ist, also im Falle einer großen Nachbarschaft mit z.B. einer exponentiellen Anzahl an Nachbarn [20].

Basic Variable Neighbourhood Search (BVNS): Das BVNS Konzept verbindet eine stochastische mit einer deterministischen Komponente, indem RVNS um eine einfache lokale Suche erweitert wird. Wenn dabei innerhalb der lokalen Suche eine bessere Lösung gefunden wird, iteriert man darauffolgend wieder von $k = 1$ weg und initiiert ein *"shaking"*. Der Nachteil auf Grund von shaking, dass eine lokal optimale Lösung nicht garantiert werden kann, bleibt auch bei BVNS weiter erhalten. Die Prozedur ist in Alg. 3.8 abbgebildet (Quelle: [20]).

Algorithmus 3.8 Basic Variable Neighbourhood Search (BVNS)

```
1: create initial solution T;
2: while termination condition not met do
3:     i = 1;
4:     while i ≤ k do
5:         randomly select a solution T ' ∈ N_i(T);
6:         try to improve T ' by using local search methods;
7:         if T ' better than best solution found so far then
8:             save T ' as new best solution T;
9:             i = 1;
10:        else i = i + 1;
```

3.4.4 Very Large Neighbourhood Search (VLNS)

Die Idee dieses Verfahrens beruht darauf eine sehr große Nachbarschaft $N(x)$ zu erzeugen, in welcher die beste Lösung effizient bestimmbar ist [3]. Effiziente Methoden zur Nachbarschaftsdurchsuchung sind z.B. das Finden von Zyklen mit minimalen Kosten in einem improvement Graphen, das Lösen von Minimum Cost Assignment oder Matching Problemen bzw. effiziente Verfahren zur Lösung von Spezialfällen des Problems. Im folgenden Abschnitt wird die Nachbarschaftsdurchsuchung mittels Assignment beschrieben.

Das Prinzip von Assignment besteht darin, k Städte aus der aktuellen Rundreise zu entfernen und ihre inzidenten Kanten durch direkte Verbindungen der jeweiligen Nachbarknoten zu ersetzen. Dadurch erhält man eine um k Städte reduzierte Rundreise T '. Die darauf generierte Nachbarschaft $N(x)$ enthält alle aus der reduzierten Tour entstandenen Rundreisen, indem die entfernten Knoten wieder eingefügt werden. Dabei dürfen keine zwei zuvor entfernten Kanten in der vollständigen Tour unmittelbar nebeneinander liegen [3], siehe auch Abb. 3.5 (Quelle: [3]).

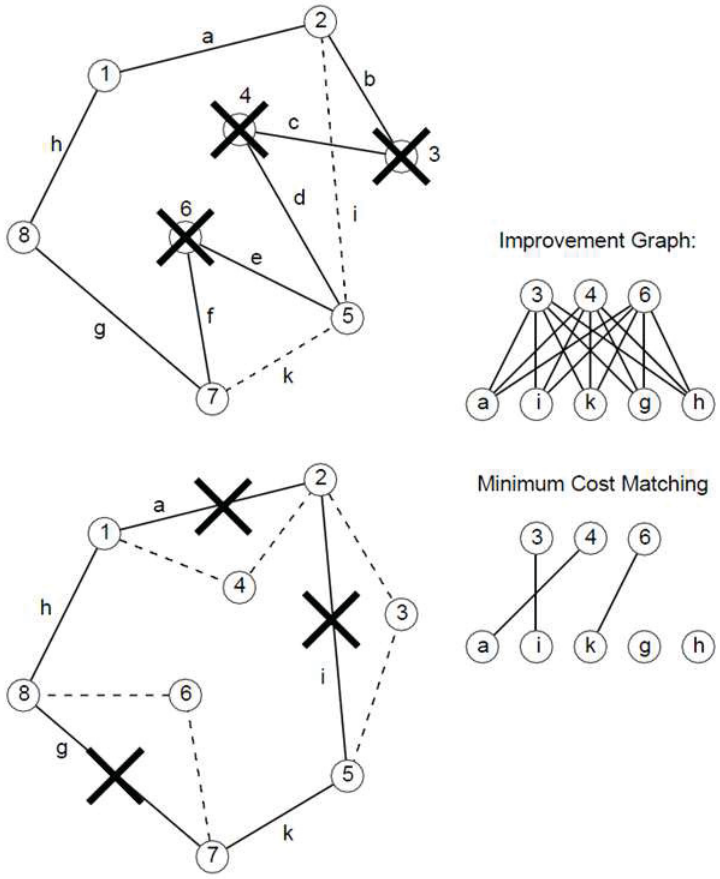


Abb. 3.5: Assignment Nachbarschaftsdurchsuchung

Das Finden der besten Nachbarschaftslösung $N(x)$ erfolgt im Rahmen der Konstruktion eines vollständigen bipartiten Improvement Graphen:

Zuerst teilt man die Knoten in zwei Knotenmengen ein, V_R repräsentiert die Menge aller entfernten Knoten, während $V_{T'}$ die Menge aller Knoten in der reduzierten Rundreise darstellt. Entsprechend erhält man auch die Kanten $(i, (u, v))$ zwischen allen Stadt-Knoten $i \in V_R$ und Kanten-Knoten (u, v) $\in V_{T'}$. Die Kantenkosten entsprechen einer Änderung der Tourlänge, wenn i zwischen u und v eingefügt wird:

$c(i, (u, v)) = d(u, i) + d(i, v) - d(u, v)$ ([3])

Übergeordnetes Ziel ist es, ein Matching mit geringsten Kosten zu finden, der Zeitaufwand dafür beträgt $O(n^3)$, wobei n die Anzahl der Knoten repräsentiert.

Für Partitionierungsprobleme, wo die Elemente einer Menge S in disjunkte Teilmengen S_i aufgeteilt werden, wie z.B. das Vehicle Routing Problem (VRP) oder das Capacitated Minimum Spanning Tree Problem, in welchem die Anzahl der in jedem Teilbaum enthaltenen Knoten beschränkt ist, wird *cyclic exchange* angewendet [3].

Während einfache Nachbarschaften ein Element in eine andere Menge verschieben oder zwei Elemente unterschiedlicher Mengen austauschen, entsteht eine Cyclic Exchange Nachbarschaft durch einen zyklischen Austausch jeweils eines Elements über beliebig viele Teilmengen hinweg, siehe auch Abb. 3.6 (Quelle: [3]) dazu. Die Nachbarschaftsgröße $N(x)$ beträgt damit $O(n^m)$, wobei m die Anzahl der disjunkten Teilmengen bezeichnet.

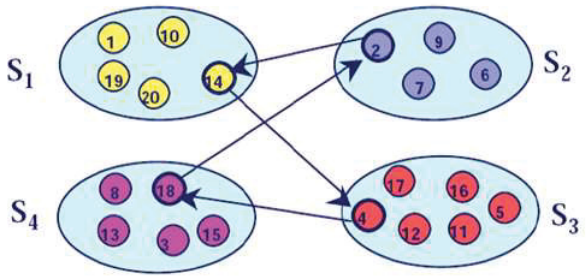


Abb. 3.6: Cyclic exchange

Der Lösungsansatz besteht in der Erstellung eines gerichteten Improvement Graphen.

Der Graph besteht aus Knoten für alle Elemente in S sowie Kanten für alle Elementpaare $(a, b) \in S^2 | a \in S_i, b \in S_j, i \neq j$ ([3])

Kante (a, b) bedeutet dabei, dass Element a von S_i nach S_j verschoben und Element b von $S_j$entfernt wird. Die Kantenkosten entsprechen einer Kostenänderung in S_j:

$c(a, b) = f(S_j \cup \{a\} \setminus \{b\}) - f(S_j)$ ([3])

Das Ziel besteht darin einen Kreis mit negativen Kosten zu finden, bei dem alle Knoten in unterschiedlichen Teilmengen liegen. Dieser Kreis stellt einen zyklischen Austausch dar, welcher die aktuelle Lösung verbessert. Das Finden eines derartigen Kreises ist wiederum ein NP- schwieriges Problem, welches in der Praxis v.a. mit guten Heuristiken gelöst wird.

Zusammenfassend stellt VLNS eine sehr problemspezifische Metaheuristik dar, welche algorithmisch anspruchsvoller als andere Metaheuristiken ist. Im Rahmen dieser Arbeit wird VLNS jedoch nicht angewandt. Obwohl die Erzeugung zweier Knotenmengen V_R und $V_{T'}$ prinzipiell möglich ist, ist die Ähnlichkeit im Vergleich zu einem genetischen Algorithmus sehr groß, wenn nur wenige (z.B. maximal 3) Kanten entfernt werden. Das Entfernen bzw. nachherige Einfügen von mehr als 3 Kanten macht in einem Schritt meiner Meinung nach wenig Sinn, weil die Ausgangslösung schon relativ gut ist und derart viele remove und insert Operatoren zu sehr einer zufälligen Suche ähneln, v.a. bei Kampagnen mit vergleichsweise wenig Losen.

3.4.5 Ant Colony Optimization (ACO)

Ameisenalgorithmen stellen eine weitere Möglichkeit dar, kombinatorische Probleme wie z.B. das TSP, effizient zu lösen. ACO verwendet den zugrundeliegenden Ameisenalgorithmus (AS) als Basis, erweitert und verbessert diesen, sodass ACO mit anderen Verfahren wie SA bzw. GA konkurrenzfähig wird. Das Funktionsprinzip dieser Art von Algorithmen ist angelehnt an die Prinzipien einer Ameisenkolonie, welches wiederum auf dem Phänomen der sogenannten “Schwarmintelligenz” beruht. ACO ist v.a. durch die Selbstorganisation von Ameisen bei der Futtersuche inspiriert [9]. Wie Termiten sowie einige Arten der Bienen und Wespen gehören Ameisen zur Gruppe der *“sozialen Insekten”*. Dass Ameisen in der Lage sind, den schnellsten Weg zu ihrer Futterquelle zu finden, wurde u.a. im Doppelbrückenexperiment untersucht und bestätigt.

Diese Ergebnisse waren der Anstoß für die Entwicklung von Ameisenalgorithmen 1990 durch Dorigo [9]. Die künstlichen Ameisen erzeugen und verändern eine Lösung auf Grund globaler Informationen in der Pheromonmatrix sowie lokaler Informationen insofern, als die Ameise unmittelbare Nachbarschaften erkennt. Der Schlüssel sind dabei die Pheromone, welche sie auf ihrem Pfad hinterlassen. Das Ameisensystem besteht zusammenfassend aus künstlichen Ameisen, diskreten Zeitschritten sowie einer globalen Pheromonmatrix und einer lokalen Informationsmatrix [3].

Der Ameisenalgorithmus läuft dabei grundlegend wie folgt ab:

1. Jede Ameise generiert eine Lösung in Abhängigkeit von lokalen Informationen und den global in der Pheromonmatrix gespeicherten Pheromonwerten.

2. Im Rahmen des *Pheromon Updates* werden die neu aufgetragenen Pheromone der Ameise in der Pheromonmatrix berücksichtigt, während vorhandene Pheromone verdunsten.

3. Darauf basierend gibt es diverse Verbesserungen bzw. problemspezifische Charakeristika, die nicht von einzelnen Ameisen ausgeführt werden können, aber die Lösungsmenge in eine bestimmte Richtung forcieren.

Die Basisstruktur eines ACO ist in Alg. 3.9 angegeben (Quelle:[9]). In Zeile 4 werden die Menge der zulässigen Elemente der Lösung (konkret für ein PFSP: N Aufträge) initialisiert. Nach der Initialisierung einer leeren Teillösung in Zeile 5 wird in den darauffolgenden 4 Zeilen iterativ eine vollständige Lösung (π') konstruiert. Im Vergleich mit anderen Metaheuristiken wie z.B. SA oder GA stellt ACO eine konstruktive Metaheuristik dar, d.h. konkret, dass sich jede Ameise ihre Lösung (Rundreise, Auftragsreihenfolge) basierend auf dem vorhandenen Wissen von lokalen und globalen Informationen komplett neu zusammenstellt. Das Einfügen von Aufträgen bzw. Städten bzw. allgemeiner das Einfügen von einzelnen Teilen der Lösung zu einer Gesamtlösung ist ein Kernelement des ACO, im Rahmen der Beschreibung der *Übergangsregel* wird näher auf dieses Kernelement eingegangen.

Algorithmus 3.9 Basisstruktur: Ant Colony Optimization

```
1: Initialisiere Pheromonmatrix T
2: do {
3:     for (ant=1; ant≤Ants; ant=ant+1) {
4:       Intialisiere Menge der nicht ausgewählten Elemente: U=1, 2, ..., N;
5:       Initialisiere aktuelle Teillösung: π' = 0;
6:       while (U ≠ 0) {
7:           Wähle ein Element n aus U aus;
8:           Füge n in π' ein;
9:          U = U \ n;
10:       } // end while
11:       Führe lokale Suche aus // optional
12:    } // end for
13:    Aktualisiere Pheromonmatrix T;
14: } while (!Abbruchbedingung )
```

Doppelbrückenexperiment ([9, 26]): Das Prinzip hinter diesem Experiment beruht auf einem autokatalytischen Prozess, im Rahmen dessen der kürzere Weg (mit zunehmender Zeit) systematisch verstärkt (von den Ameisen) ausgewählt wird, was seine Ursache darin hat, dass immer mehr Pheromone auf diesem Weg liegen, welches die Ameisen auf diesem Weg abgelegt haben. Während Ameisen ihre Entscheidung in Abwesenheit von Pheromonen zufällig treffen, beeinflusst sie die Anwesenheit ebendieser Substanz bei ihrer Entscheidung in Bezug auf den ausgewählten Weg.

Zu beachten ist dabei, dass auf dem Hinweg zur Futterquelle keiner der beiden Wege bevorzugt werden kann. Dadurch, dass die Ameisen, die auf dem Hinweg den kürzeren Weg gewählt haben, denselben kürzeren Weg wieder zurückwählen, beeinflussen sie durch ihren früher angetretenen Rückweg auch die anderen Ameisen, die den längeren Hinweg genommen haben, auf Grund ihres Pheromons. Mit hoher Wahrscheinlichkeit werden nun sehr viele Ameisen im Rahmen ihres Rückweges auch den kürzeren Weg wählen, weil hier schon eine deutlich ausgeprägte Pheromonspur der Ameisen vorliegt, die allesamt den kürzeren Weg gewählt haben. Dieser Effekt verstärkt sich mit zunehmender Zeit.

In Bezug auf Abb. 3.7 (Quelle: [9]) starten zum Zeitpunkt $t = 0$ hundert Ameisen, wobei sich auf Grund fehlender Pheromoninformationen jeweils 50% für den Weg über Punkt B und 50% für den Weg über Punkt C entscheiden. Der Zeitpunkt $t = 1$ bildet ab, dass die 50 Ameisen, die Weg C genommen haben, bereits Punkt D erreicht haben und dementsprechend Pheromone entlang dieser Wege abgelegt haben. Die Ameisen, welche den längeren Weg genommen haben, haben in der Zwischenzeit lediglich Punkt B

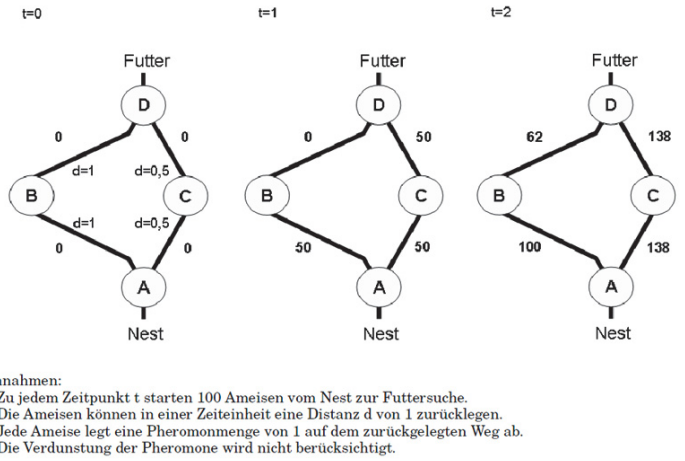


Abb. 3.7: Indirekter Kommunikationsmechanismus der Ameisen

erreicht. Da nun bereits eine Pheromonspur zurück nach A über den Weg mit dem Punkt C existiert, aber noch keine über den Punkt B, wird sich der größere Teil der 50 Ameisen für den Weg über Punkt C entscheiden. Gleichzeitig starten 100 neue Ameisen vom Nest aus in Richtung Futterplatz, wobei sich wiederum 50 für jeden der beiden Wege entscheiden, da auf beiden Wegen dieselbe Anzahl an Pheromonen abgelegt worden ist. Zum Zeitpunkt $t = 3$ erreichen die ersten 38 Ameisen das Nest, dies ergibt mit den 50 Ameisen, die zum Zeitpunkt $t = 1$ vom Nest aus in Richtung Futterplatz gegangen sind, einen Gesamtpheromonwert von 138. Die 50 Ameisen, die über Punkt B gegangen sind, erreichen nun erst den Futterplatz. Bedingt durch eine deutlich größere Anzahl abgelegter Pheromone über Punkt C wird der Großteil von ihnen auch über den kürzeren Weg die Heimreise zum Nest antreten. Mit zunehmender Zeit werden immer mehr Ameisen jeweils den kürzeren Weg benutzen und dort ihre Pheromone abgeben. Dieses Verhalten der Ameisen beeinflusst auch ihre Entscheidung bei Punkt A.

Im Rahmen eines weiteren Brückenexperiments wurde auch festgestellt, dass, wenn nach einiger Zeit ein neuer (z.B. künstlich geschaffener) kürzerer Weg geschaffen wird, dieser Weg von den Ameisen kaum ausgewählt wird, weil dort kein Pheromon liegt [26].

Im Rahmen eines Experiments von Doneuburg wurde festgestellt, dass bei 2 gleich langen Wege nicht beide in gleichem Maße genutzt werden, sondern dass sich einer der beiden Wege pheromonbedingt durchsetzt [28].

Die 4 Prinzipien der Selbstorganisation und weitere Charakteristika des ACO

> Definition: “Self-organization is a process in which a pattern at the global level of a system emerges solely from numerous interactions among lower-level components of the system. Moreover, the rules specifying interactions among the system’s components are executed using only local information, without reference to the global pattern.” Camazine et al., S.8 (Quelle: [3])

Die 4 von Bonabeau angeführten Prinzipien der Selbstorganisation lassen sich im ACO Ansatz wiederfinden ([9]):

1. *Positives Feedback*: Durch das häufige Benutzen kurzer Wege liegt auf diesen Wegen eine stärkere Pheromonspur, welche (mit hoher Wahrscheinlichkeit) mehr Ameisen dazu verleitet, diesen Weg zu nehmen. Dieser selbstverstärkende Prozess wird auch als Autokatalyse bezeichnet.

2. *Negatives Feedback*: Dem Ablegen von Pheromonen wirkt das Verdampfen der Pheromone entgegen.

3. Der *stochastische Charakter jeder Ameise* bedeutet, dass jede Ameise für sich zufällig den Weg auswählt, da die Pheromonwerte die Entscheidung nicht determinieren, sondern lediglich statistisch beeinflussen.

4. Als viertes Merkmal wird *multiple Interaktion* angeführt. Diese Art der Kommunikation kann direkt bzw. auch indirekt über die Veränderung der Umwelt erfolgen. Im Falle der Ameisenkolonie erfolgt die **Kommunikation durch das Ablegen der Pheromone, wobei die Pheromonkonzentration von den anderen Ameisen wahrgenommen wird und ihr Verhalten beeinflusst**.

Dabei spielen die Mechanismen *Rückkopplung* und *Stigmergie*, der Informationsaustausch in einem dezentralen System durch die Veränderung der (näheren) Umgebung, eine besondere Rolle [3].

Zusätzlich zu diesen Merkmalen besitzt jede Ameise eine Art Gedächtnis, in der die bisher konstruierten Lösungen bewertet sowie auf ihre Zulässigkeit überprüft werden können.Zu beachten ist, dass künstliche Ameisen im Gegensatz zu realen Ameisen über 2 Varianten des *Pheromon Updates* verfügen [9].

1. *Online Step-by-Step Pheromon Update*: Hier findet das Ablegen der Pheromone während der Erstellung einer neuen Lösung statt.

2. *Online Delayed Pheromon Update*: Das Pheromon Update findet statt, nachdem die vollständige Lösung von der Ameise generiert worden ist.

Das Verdunsten der Pheromone wird dabei als eine von den Ameisen unabhängige Funktion behandelt. Neben den diversen Gemeinsamkeiten zwischen realen und künstlichen Ameisen gibt es auch Unterschiede. Künstliche Ameisen verfügen über ein Lösungsgedächtnis und wie schon erwähnt, die Auswahl zweier Möglichkeiten des Pheromon Updates, während reale Ameisen ihre Pheromone nur während der Konstruktion der Lösung ablegen können. Bei künstlichen Ameisen kann man auch die Pheromonkonzentration anhand der Qualität der gefundenen Lösung steueren, während realen Ameisen diese Option nicht zur Verfügung steht. Außerdem können im Algorithmus heuristische zusätzliche Informationen verwendet werden, welche im realen System einer einzelnen Ameise nicht zur Verfügung stehen.

Ausgestaltungsmöglichkeiten der Pheromonmatrix τ für ein PFSP Das sogenannte "Langzeitgedächtnis" speichert die von den Ameisen generierte Pheromonspur und stellt die Basis jeder neuen Lösungskonstruktion dar [9]. Prinzipiell bestehen für die Repräsentation im Rahmen eines PFSP mehrere Möglichkeiten, wobei hier 2 Ansätze beschrieben werden, siehe Tabelle 3.1 (Quelle: [9]).

i/j	1	...	j	...	N
1					
...					
i			τ_{ij}		
...					
N					

n/h	1	...	h	...	N
1					
...					
n			τ_{nh}		
...					
h					

Tab. 3.1: Gestaltungsmöglichkeiten der Pheromonmatrix τ

Die linke Gestaltungsmöglichkeit entspricht der Realisierung in einem TSP. Während die Spalten und Zeilen die N Aufträge beinhalten, gibt der Pheromonwert τ_{ij} im Schnittpunkt an, wie sinnvoll es ist, Auftrag j nach Auftrag i zu fertigen. Für das PFSP verwendet man jedoch i.d.R. die rechte Gestaltungsmöglichkeit. Dabei ist in der Pheromonmatrix die Vorteilhaftigkeit τ_{nh} kodiert, wenn man Auftrag n an die Position h der Auftragsreihenfolge platziert [9].

3.4.5.1 ACO: Unterschiede zwischen AS und ACS

Das Ant Colony System (ACS) verwendet im Gegensatz zum Ant System (AS) eine andere Übergangsregel, außerdem wird die Pheromon Update Regel pro Iteration nur von der besten Ameise durchgeführt. Die Pheromonspur wird dagegen jedesmal verringert. Abschließend werden die beiden Verfahren algorithmisch gegenübergestellt. Die Übergangsregel wird in den folgenden beiden Paragraphen für das AS und ACS am Beispiel des TSP sowie für das ACS am Beispiel des PFSP beschrieben.

Übergangsregel: Der Übergang in den nächsten Knoten hängt nach [28] zunächst von 3 Faktoren ab:

1. Der Knoten j darf in derselben Tour noch nicht besucht worden sein. Daher existiert für jede Ameise k und jeden Knoten i eine Liste J_i^k. Diese Liste enthält dabei für jede Ameise die Menge der noch nicht besuchten Knoten in der aktuellen Iteration. Die Liste wird daher nach jedem Übergang aktualisiert.

2. In jedem Knoten i, in dem sich die Ameise befindet, hat sie auch lokale Informationen zur Verfügung, wobei diese Informationen als jeweils reziproke Distanz $1/d_{ij}$ zu allen Nachbarknoten in einer Matrix gespeichert sind.

3. Auf die Ameise wirken v.a. auch globale Informationen in Form von Pheromonen ein. Die Pheromonkonzentration ändert sich dabei von Iteration zu Iteration und repräsentiert die bisher erworbenen Erfahrungen der Ameisen.

Übergangsregel für AS: Die Übergangsregel beschreibt die Wahrscheinlichkeit p_{ij}^k, mit der die $k-$te Ameise in ihrer $t-$ten Tour von Knoten i zu Knoten j wechselt.

$p_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \varepsilon N_i^k} ([\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta)}$ *(Quellen: [3, 28, 9])*

Die Parameter α und β sind für das Verhältnis zwischen Pheromonen (in τ gespeichert) und lokalen Informationen (in η gespeichert) zuständig. Für $\alpha = 0$ realisiert man den klassischen greedy Ansatz, wobei lokale Informationen nicht verwendet werden, während $\beta = 0$ eine rasche Konvergenz zu einem Pfad liefert, da hier sämtliche bisher gesammelten Informationen auf Grund der Pheromonspur wegfallen. Der Ansatz $\alpha = 0$ endet mit der Auswahl des am häufigsten benutzen Weges als Lösung. Die Nachbarschaftsliste N_i^k steht im Falle des TSP für die Liste unbesuchter Städte [3].

Weiters wird durch die Wahrscheinlichkeitsverteilung nicht automatisch der insgesamt, also lokal und global, beste Knoten ausgewählt, sondern es kann auch passieren, dass ein relativ schlechter Knoten ausgewählt wird, wodurch neue Lösungen erzeugt werden.

Übergangsregel für ACS: Während beim AS immer die Wahrscheinlichkeitsverteilung als Basis für die Erstellung einer Rundreise herangezogen wird, wird stattdessen beim ACS zwischen 2 Möglichkeiten gewählt.

Die erste Möglichkeit (Fall 1) besteht darin, den Knoten zu wählen, der insgesamt, also lokal und global, am meisten Nutzen bringt, wodurch jedoch die Entdeckung neuer Lösungen verhindert wird, sondern die Ausnutzung des vorhandenen Wissens forciert wird. Die zweite Möglichkeit (Fall 2) ist der Auswahlwahrscheinlichkeit beim AS sehr ähnlich, da auch hier die Auswahl des nächsten Knotens j proportional zum Nutzen der lokalen und globalen Information ist. Der Parameter q_0 steuert dadurch die Diversifizierung bzw. die Intensivierung der Suche [3]. In welchem Verhältnis die beiden Methoden verwendet werden, kann mittels q_0 im Intervall $[0, 1]$ parametrisiert werden. Die Übergangsregel in den nächsten Knoten j besteht somit aus 2 Fällen:

$$j = \begin{cases} arg\, max_{u \in J_i^k} \left\{ [\tau_{iu}(t)] \cdot [\eta_{iu}]^\beta \right\} & wenn\ q \leq q_0 \\ \Psi & wenn\ q > q_0 \end{cases}$$ (Quelle: [28])

Legende: q ist eine Zufallszahl, q_0 ist ein fixer Parameter im Intervall $[0, 1]$. Während q nach jedem Übergang neu bestimmt wird, wird q_0 am Anfang des Algorithmus festgelegt.

Hinter dem Ausdruck $[\tau_{iu}(t)]$ ist kein $\alpha-$ Wert im Exponent explizit angegeben, weil dieser überlicherweise auf 1 gesetzt wird .

Ψ bezeichnet einen Knoten, der mit der Wahrscheinlichkeit $p_{i\Psi}^k(t) = \frac{[\tau_{i\Psi}(t)] \cdot [\eta_{i\Psi}]^\beta}{\sum_{l \in J_i^k} ([\tau_{il}(t)] \cdot [\eta_{il}]^\beta}$ (Quelle: [28]) selektiert wird.

Diese Vorgangsweise, mit Wahrscheinlichkeit q_0 den maximalen Pheromonwert auszuwählen, während im anderen Fall auf Basis einer Wahrscheinlichkeitsverteilung selektiert wird, ist im Ant Colony System auch als *pseudorandom proportional rule* bzw. *pseudozufällige Proportionalitätsregel* bekannt [9, 41].

Pheromon Update Regel für AS: Jede Ameise k, die eine Tour nach einer bestimmten Zeit bzw. im Rahmen einer Iteration t vollendet hat, verstärkt die von ihr benutzten Kanten (i, j) um eine bestimmte Pheromonmenge $\Delta\tau_{\mathrm{ij}}^{k}(\mathrm{t})$, wobei diese Menge von der Lösungsqualität abhängt [28].

$\Delta\tau_{ij}^{k}(t) = Q/L^{k}(t)$ (Quelle: [28])

$L^{k(t)}$ entspricht der Länge der Tour und Q liegt möglichst nahe an der optimalen Tourlänge und wird daher vorab heuristisch errechnet [28]. Um die Erforschung neuer Wege zu erleichtern und eine Stagnation in der aktuellen Lösung zu verhindern, wird im Rahmen der Pheromon Update Regel auch die Verdunstungsrate ρ verwendet. Diese Rate steht für den Anteil der Pheromone, die nach einer Iteration in der Pheromonmatrix verdampft.

Die Verstärkung der benutzten Wege ergibt nach [3] $\tau_{\mathrm{ij}}(t) = \tau_{\mathrm{ij}}(t) + \Delta\tau_{\mathrm{ij}}(t)$.

Die gesamte Pheromondifferenz ergibt sich nach [3, 28] zu $\Delta\tau_{\mathrm{ij}}(t) = \sum_{k=1}^{m} \Delta\tau_{\mathrm{ij}}^{k}(t)$,

wobei die Pheromondifferenz jeder Ameise k vom Knoten i zum Knoten j nach [3]

$\Delta\tau_{\mathrm{ij}}^{k}(t) = \begin{cases} 1/length & (T^{k}(t)),\ wenn\ (i,j)\ \in\ T^{k}(t) \\ 0 & sonst \end{cases}$ beträgt.

T^{k} repräsentiert dabei die von der $k-$ten Ameise konstruierte Tour.

Die Berücksichtigung der Pheromon Verdampfung ergibt nach [28, 3] $\tau_{\mathrm{ij}}(t+1) = (1-\rho)\cdot\tau_{\mathrm{ij}}(t)$, wobei der Pheromonwert mit $\tau_0 \geq 0$ initialisiert wird und $0 \leq \rho < 1$ gesetzt wird.

Pheromon Update Regel für ACS: Im Gegensatz zu AS legt im ACS nur jene Ameise eine Pheromonspur, welche die bisher beste Lösung gefunden hat. Dadurch legt in jeder Iteration genau eine Ameise eine Pheromonspur, wodurch die Ameisen v.a. in der Umgebung der besten gefundenen Lösung weitersuchen [28]. Die Pheromon Update Regel lautet: $\tau_{\mathrm{ij}}(t) \leftarrow (1-\rho)\cdot\tau_{\mathrm{ij}}(t) + \rho\cdot\Delta\tau_{\mathrm{ij}}(t)$, wobei (i, j) zur besten Tour T^{+} gehört, ρ wiederum die Pheromonverdunstungsrate darstellt und $\Delta\tau_{\mathrm{ij}}(t) = 1/(L^{+})$,wobei L^{+} die Länge der Tour T^{+} repräsentiert [28].

Zusätzlich zur globalen Pheromon Update Regel kommt auch eine lokale Update Regel zur Anwendung, welche die Pheromonmenge beim Übergang der Ameise vom Knoten i zum Knoten j um den folgenden Betrag verringert:

$\tau_{\mathrm{ij}}(t) \leftarrow (1-\rho)\cdot\tau_{\mathrm{ij}}(t) + \rho\cdot\tau_0$ (Quelle: [28])

Für den Parameter ρ $(0 < \rho \leq 1)$ hat sich experimentell ein Wert von 0,1 als günstig herausgestellt. Für τ_0 wird prinzipiell derselbe Wert genommen, den man für die Initialisierung der Kanten mit Pheromonen herangezogen hat [41].

Dadurch, dass die eben besuchte Kante einer Ameise für die nachfolgenden Ameisen pheromonbedingt an Attraktivität verliert, steigt die Wahrscheinlichkeit, dass die nachfolgenden Ameisen einen anderen Weg nehmen. Ohne diese Verfeinerung würden die nachfolgenden Ameisen oft nur lokal weitersuchen und die Lösung könnte (zu) leicht in einem lokalen Optimum stagnieren [28].

AS und ACS Algorithmus Die Auflistung des AS und ACS Algorithmus verdeutlicht die Gemeinsamheiten und Unterschiede der beiden Vertreter der Klasse von Ameisenalgorithmen. Zunächst wird der AS Algorithmus beschrieben (Quelle: [28]).

Algorithmus 3.10 Ant System Algorithmus

1: /* Initialisierung */
2: **for each** Kante (i, j)
3: $\tau_{ij} = \tau_0$;
4: **end for**
5: $L^+ = -1$ // es gibt noch keine minimale Tourlänge
6: /* Hauptschleife */
7: **for** $t = 1$ **to** t_{max}
8: Fülle die Liste J_i^k für alle Ameisen k und alle Knoten i
9: mit dem jeweils nächsten Knoten j gemäß der Übergangsregel
10: /* Hauptschleife: Für jede Ameise eine Tour finden */
11: **for** $k = 1$ **to** m
12: Knoten i =Anfangsknoten
13: T^k besteht aus dem Anfangsknoten
14: **while** Knoten i **not** = Endknoten
15: Wähle den nächsten Knoten j mit der Wahrscheinlichkeit
16: $p_{ij}^k = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \varepsilon N_i^k} ([\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta)}$ // $j \in J_i^k$
17: Aktualisiere die Liste J_i^k
18: Füge den Knoten j zu T^k hinzu
19: Knoten i = Knoten j
20: **end while**
21: **end for**
22:
23: /* Tourenlänge berechnen und kürzeste Tour bzw. deren Länge auf T^+ und L^+ speichern */
24: **for** $k = 1$ **to** m
25: Berechne die Tourlänge L^k der Tour T^k der Ameise k
26: **if** $L^k < L^+$ oder $L^+ = -1$ **then**
27: Aktualisiere T^+ und L^+
28: **end If**
29: **end for**
30:
31: /* Pheromonspur aktualisieren */
32: **for each** Kante (i, j)
33: Berechne die neue Menge an Pheromonen laut der Pheromon Update Regel:
34: $\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \Delta\tau_{ij}$, wobei
35: $\Delta\tau_{ij} = \sum_{k=1}^m \Delta\tau_{ij}^k$
36: und $\Delta\tau_{ij}^k = Q/L^k$,
37: wenn die Kante (i, j) in der Tour T^k enthalten ist, andernfalls ist $\Delta\tau_{ij}^k = 0$.
38: **end for**
39: **end for**
40: **print** die kürzeste Tour T^+mit ihrer Länge L^+

Das Kernelement des AS Algorithmus stellt die Hauptschleife (Zeile 11 - 21) dar, in der jede Ameise mittels der Übergangsregel ihre Rundreise zusammenstellt und in T^k speichert. Anschließend werden die Längen aller Touren berechnet (bzw. allgemeiner der Fitnesswert der erhaltenen Lösung) und im Falle einer besseren Lösung als der bisher besten die kürzeste Route aktualisiert. Danach wird die Pheromon Update Regel auf jede Kante angewandt und der Algorithmus wird neu gestartet [28].

Algorithmus 3.11 Ant Colony System Algorithmus

/* Initialisierung */
for each Kante (i, j)
  $\tau_{ij} = \tau_0$
end For
$L^+ = -1$ // es gibt noch keine minimale Tourlänge
Fülle die Kandidatenlisten mit cl Knoten für alle Knoten

/* Hauptschleife */
for $t = 1$ **to** t_{max}
  Fülle die Liste J_i^k für alle Ameisen k und alle Knoten i (mit dem jeweils nächsten Knoten j gemäß der Übergangsregel)

  /* Für jede Ameise eine Tour T^k finden */
  for $k = 1$ **to** m
    Knoten i= Anfangsknoten
    T^k besteht nur aus dem Anfangsknoten

    while (Knoten i ! = Endknoten
      if wenigstens ein Knoten j in der Kandidatenliste von i enthalten ist,
      der noch nicht besucht wurde **then**

        Errechne eine Zufallszahl q
        Wähle den nächsten Knoten j aus den Knoten der Kandidatenliste mittels der Formel:
        $$j = \begin{cases} arg\,max_{u \in J_i^k} \left\{ [\tau_{iu}] \cdot [\eta_{iu}]^\beta \right\} & wenn\ q \leq q_0 \\ \Psi & wenn\ q > q_0 \end{cases}$$
        wobei $\Psi \in J_i^k$ und mittels der Wahrscheinlichkeit
        $p_{i\Psi}^k = \frac{[\tau_{i\Psi}] \cdot [\eta_{i\Psi}]^\beta}{\sum_{l \in J_i^k} ([\tau_{il}] \cdot [\eta_{il}]^\beta}$ gewählt wird.
      else
        Wähle den nächstgelegenen Knoten aus als Knoten j
      end if
      Berechne die neue Menge an Pheromonen laut der lokalen Pheromon-Update-Regel:
        $\tau_{\mathrm{ij}} \leftarrow (1 - \rho) \cdot \tau_{\mathrm{ij}} + \rho \cdot \tau_0$
      Aktualisiere die Liste J_i^k
      Aktualisiere die Kandidatenliste von Knoten i
      Füge den Knoten j in T^k hinzu
      T^k hinzu Knoten i = Knoten j
    end while
  end for

  /* Tourlänge berechnen und kürzeste Tour bzw. deren Länge auf T^+ und L^+ speichern */
  For $k = 1$ **to** m
    Berechne die Tourlänge L^k der Tour T^k der Ameise k
    If $L^k < L^+$ oder $L^+ = -1$ **then**
      Aktualisiere T^+ und L^+
    End If
  End For

  /* Pheromonspur updaten */
  For each Kante $(i, j) \in T^+$
    Berechne die neue Menge an Pheromonen laut der Pheromon- Update-Regel:
    $\tau_{\mathrm{ij}} \leftarrow (1 - \rho) \cdot \tau_{\mathrm{ij}} + \rho \cdot \Delta\tau_{\mathrm{ij}}$,
    wobei $\Delta\tau_{\mathrm{ij}} = 1/L^+$
  End For
End For
Print die kürzeste Tour T^+ mit ihrer Länge L^+

Vergleichend zum AS Algorithmus wird der ACS Algorithmus in Alg. 3.11 beschrieben (Quelle: [28]). Bei ACS können zusätzlich noch sogenannte *Kandidatenlisten* initialisiert werden, in der die jeweils *cl* nächstgelegensten Knoten (nach einem heuristischen Kriterium) abgespeichert sind. Die Benützung derartiger Listen eignet sich v.a. für komplexe Probleme und führt dort zu besseren Rundreisen bzw. rascherer Konvergenz zu guten Lösungen [41, 28]. Dabei muss der Parameter *cl* anfänglich festgelegt werden und ist für jeden Knoten gleich groß. Bei der Verwendung von Kandidatenlisten werden also für eine Ameise k für den Übergang zum Knoten j nur die in der Kandidatenliste enthaltenen Knoten berücksichtigt. Sofern in dieser Liste nur mehr bereits besuchte Knoten existieren, wird der nächste Knoten gemäß der geringsten Distanz zum Knoten i ausgewählt.

Es wird in der Hauptschleife für jede Ameise k eine Rundreise konstruiert, wobei bei jedem Übergang die lokale Pheromon Update Regel zur Anwendung kommt. Nach der Erstellung aller Touren wird für jede Tour ihre Länge berechnet und, sofern notwendig, die beste Tour aktualisiert. Die Kanten der besten Tour werden abschließend mit Pheromonen (gemäß der Pheromon Update Regel) verstärkt, bevor die Hauptschleife neu gestartet wird [28].

3.4.5.2 ACO Varianten

Zu ACO gibt es zahlreiche Varianten. Der elitäre Ameisenalgorithmus (EAS) basiert z.B. auf der Idee, dass die Ameise mit dem bisher kürzesten gefundenen Weg mehr Pheromone aufträgt. Die Skalierung ist hierbei entscheidend um frühzeitige Stagnation zu vermeiden. I.d.R. werden dabei - wie bei einem elitären GA, der das beste Individuum immer in die nächste Generation mitnimmt - schnell kürzere Wege gefunden bzw. weniger Iterationen benötigt [3]. Der Grundgedanke hinter dieser Idee besteht darin, dass mehr Ameisen in der Nähe der bisher besten Lösung(en) weitersuchen sollen.

Ant System mit elitist strategy: Im Rahmen der *"elitist strategy"* werden in jeder Iteration e elitist ants zu den Ameisen hinzugefügt, wodurch die beste Tour insgesamt zusätzlich um den Faktor $e \cdot Q/L^+$ verstärkt wird [28]. Die Pheromon Update Regel wird dabei um den Anteil der *"elitist ants"* erweitert: $\tau_{ij}(t) \leftarrow (1-\rho) \cdot \tau_{ij}(t) + \triangle\tau_{ij}(t) + e \cdot \triangle\tau^e_{ij}(t)$ ([28]), wobei

$e \cdot \triangle\tau^e_{ij}(t) = Q/L^+$, wenn Kante $(i,j) \in T^+$, sonst 0. Generell sollte (in Abhängigkeit von der Gesamtanzahl an Ameisen) eine verhältnismäßig geringe Anzahl an "elitist ants" gewählt werden, da sonst die Lösung ein (nicht optimales) frühzeitig lokales Optimum darstellt.

Elitist Ant System (EAS) und rank-based AS (AS_{rank}): Im Rahmen von EAS trägt die beste Ameise mehr Pheromone auf, was prinzipiell kürzere Touren begünstigt und weniger Iterationen notwendig macht.

Ähnlich wie EAS tragen beim rankingbasierten Ameisenalgorithmus Ameisen mit höherem Rang mehr Pheromone auf. Bei dieser Variante tragen die besten Ameisen (inklusive der Ameise mit der besten Rundreise) zum Pheromon Update bei, durch die Skalierung kann der rankingbasierte Ameisenalgorithmus insgesamt bessere Ergebnisse erzielen als das einseitig in der Nähe der besten Ameise weitersuchende EAS Verfahren.

Max-Min Ant System (MMAS): Das Max-Min Ameisensystem (*MMAS*) beruht darauf, dass das Pheromon Update nur von der besten Ameise durchgeführt wird, außerdem existieren minimale und maximale Schranken für die Pheromon Werte um vorzeitige Stagnation zu vermeiden [3]. Die untere Schranke sorgt dafür, dass auch Kanten mit geringen Pheromonspuren nicht zur Gänze vernachlässigt werden (in Abhängigkeit von der konkreten Wahl des Wertes für die Schranke), während die obere Schranke sicherstellt, dass nicht überproportional viel Pheromon auf diese Kanten gelegt werden kann.

In diesem Zusammenhang wird auf eine Gemeinsamkeit zwischen MMAS und ACS hingewiesen. Während beim MMAS explizite Schranken existieren, fällt bei ACS die untere Grenze niemals unter τ_0 , darüber hinaus kann gezeigt werden, dass es eine obere Grenze für den Pheromongehalt gibt. Somit

sind also eine obere bzw. eine untere Schranke im Rahmen der verfahrensbedingten Veränderungsregeln gegeben [41].

Für den Fall, dass Stagnation eintritt, kann "trail-smoothing" zur Anwendung kommen. Das Verfahren beruht darauf, dass alle Kanten proportional um die (Pheromon-) Differenz zwischen τ_{max} und der aktuellen Pheromonmenge erhöht werden, was zu einer geringeren Verstärkung von stärker mit Pheromon belegten Wegen führt. Als weiteres Unterscheidungsmerkmal eines Max-Min AS werden anfänglich alle Kanten mit τ_{max} initialisiert, wobei dieser Wert durch das Verdampfen der Pheromone allmählich verringert wird. Dadurch erhält der Algorithmus anfänglich einen sehr explorativen Charakter. Zusammenfassend ist MMAS eines der am erfolgreichsten angewandten ACO Verfahren [28, 3].

Hybrid Ant System (HAS): Der Grundgedanke bei HAS besteht darin, einen ACO mit einem lokalen Suchalgorithmus zu kombinieren. Je nach Problem eignen sich verschiedenste lokale Suchalgorithmen, außerdem gibt es auch verschiedenste Möglichkeiten, diese in den ACO einzubauen. Die am häufigsten verwendete Möglichkeit besteht darin, den lokalen Suchalgorithmus auf jede abgeschlossene Tour anzuwenden [28]. Da dies im konkreten PFSP zu viel Performance kostet, wird eine lokale Suche, wie beim HGA, für die besten m Ameisen am Ende einer durch den Ameisenalgorithmus optimierten Kampagne angewendet.

In Bezug auf lokale Verbesserungsheuristiken werden v.a. 2- exchange Nachbarschaften bzw. k- exchange Nacharschaften (2.5- Opt bzw. 3- Opt) verwendet [3].

Die ACO Metaheuristik wird im Gebiet kombinatorischer Optimierungsprobleme für TSP, Tourenplanungsprobleme und auch für Reihenfolgeprobleme verwendet, z.B. für ein PFSP mit mehrfacher Zielsetzung. Siehe dazu auch die Dissertation von Christian Petri ([9]). Ameisenalgorithmen können für verschiedenste Routing, Assignment und Scheduling Probleme zum Einsatz kommen, sie eignen sich v.a. für das QAP bzw. Bin Packing [3].

Im Rahmen dieser Masterarbeit sowie in Bezug auf das konkrete PFSP kommt ein hybrider ACS zur Anwendung, wobei der Endbenutzer die diversen Parameter (Anzahl Ameisen, Anzahl Iterationen, Anzahl elitist ants, Anzahl Ameisen für das Pheromon Update) einstellen kann. Der Endbenutzer kann auch die Gewichtung der lokalen Informationen bzw. der Pheromone über α bzw. β einstellen, außerdem kann er die Anzahl an Ameisen einstellen, für die ein globales Pheromon Update am Ende jeder Iteration durchgeführt werden soll. Im Rahmen diverser Testläufe wird ein α Wert in der Höhe von ca. 0.9, ein $\beta-$ Wert von ca. 0.1 empfohlen. Die lokale Informationsmatrix wird für jede Ameise vor jedem Übergang aktualisiert. Es empfiehlt sich allgemein, die besten ca. 20 % der Ameisen das Pheromon Update durchführen zu lassen.

3.4.6 Particle Swarm Optimization (PSO)

Die Partikelschwarmoptimierung ist eine von Kennedy und Eberhart im Jahre 1995 entwickelte populationsbasierte stochastische Optimierung, welche durch das (Bewegungs-) Verhalten von in Schwärmen lebenden Tieren wie Vögeln bzw. Fischen inspiriert ist [1, 26]. Die Individuen orientieren sich neben ihrem eigenen kognitiven Anteil, dargestellt durch ihr Gedächtnis, auch an der Position bzw. dem Verhalten des besten Individuums im Schwarm und adaptieren ihre eigene Position sowie ihre Geschwindigkeit anhand dieser beiden Gesichtspunkte. Voraussetzung für die Anwendung von PSO ist ein reelwertiger Suchraum, d.h.: S$\subseteq \mathbb{R}^n$, die zu optimierende Funktion sei $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ([26])

Im Rahmen dieser Vorgehensweise wird jedes Individuum im Schwarm als *"Teilchen"* definiert, welches über einen Positionsvektor $\vec{x}_i$ sowie über einen Geschwindigkeitsvektor $\vec{v}_i$ verfügt. Die Teilchenschwarmoptimierung vereinigt damit Ansätze der bahnorientierten Suche und populationsbasierter Suche. Jedes Teilchen verfügt über sein eigenes Wissen *pbest*, z.B. die eigene beste Position, sowie über ein soziales Wissen der Nachbarteilchen *gbest* , z.B. *pbest* des besten Nachbarn. Man kann *pbest* auch als "*lokales Gedächtnis"* v.a. in Bezug auf den besten durch das Teilchen besuchten Ort im Suchraum bezeichnen, während *gbest* das *"globale Gedächtnis"* in Bezug auf den besten besuchten Ort, den ein

Individuum des Schwarms besucht hat, darstellt [26]. Das Geschwindigkeitsupdate ergibt sich wie folgt (Quelle: [1]):

$\vec{v_i}(t+1) = \alpha \cdot \vec{v_i} + c_1 \times rand \times (pbest)(t) - \vec{x_i}(t)) + c_2 \times rand \times (gbest(t) - \vec{x_i}(t)]$ (1)

Der sogenannte Trägheitsparameter α steuert das Verhältnis zwischen lokaler und globaler Suche im PSO, während c_1 den kognitiven und c_2 den sozialen Anteil als Beschleunigungskonstanten darstellen, welche die Geschwindigkeit des Partikels in Bezug auf *pbest* bzw. *gbest* modifizieren. Bessere Ergebnisse erzielt man, wenn α linear abnimmt. Der Trägheitsterm besitzt eine gewisse Ähnlichkeit mit dem Momentum-Term im Rahmen des Backpropagation Verfahrens, sodass für Werte mit $\alpha < 1$ konvergentes Verhalten bzw. für Werte mit $\alpha > 1$ divergentes bzw. exploaratives Verhalten erreicht wird [32]. *rand* bezeichnet eine Zufallszahl im Intervall [0,1], c_1 bzw. c_2 wird i.d.R. auf 2 gesetzt.

Eine weitere Notation für das Geschwindigkeitsupdate lautet (Quellen: [26, 1]):

$\vec{v_i}(t+1) = \alpha \cdot \vec{v_i}(t) + \beta_1 \cdot (\vec{x_i}^{(lokal)}(t) - \vec{x_i}(t)) + \beta_2 \cdot (\vec{x}^{(global)}(t) - x_i(t))$ (2)

β_1 und β_2 werden in jedem Schritt zufällig gewählt, während α mit der Zeit abnimmt.

Das lokale Gedächtnis: $\vec{x_i}^{lokal}(t) = \vec{x_i}(argmax_{u=1}^{t} f(\vec{x_i}(u))$ (3)

Das globale Gedächtnis: $\vec{x}^{global}(t) = \vec{x_j}^{lokal}(t)$, wobei $j = argmax_{i=1}^{m} f(\vec{x_i}^{lokal}(t))$ (4)

Das Positions Update lautet: $\vec{x_i}(t+1) = \vec{x_i}(t) + \vec{v_i}(t+1)$ (5)

Der Pseudo Code eines PSO wird in Alg. 3.12 beschrieben (Quelle: [1]).

Algorithmus 3.12 Pseudo Code eines PSO

```
1: for each particle
2:     initialize particle;
3: end
4: do
5:     for each particle
6:         calculate fitness value;
7:         if the fitness value is better than the best fitness value
8:         (pbest) in history set current value as the new pbest;
9:     end
10:    choose the particle with the best fitness value of all the articles as the gbest;
11:    for each particle
12:        calculate particle velocity according to the velocity update (2);
13:        update particle position according to the position update (5);
14:    end
15: while maximum iterations or minimum error criteria is not attained
```

Nach der Parameterinitialisierung (Zeile 2) wird innerhalb einer for Schleife für jedes Teilchen seine Fitness berechnet (Zeile 6) und, sofern notwendig, *pbest* aktualisiert (Zeile 8). Nachdem alle Partikel bewertet worden sind, wird *gbest* aktualisiert (Zeile 10). Schließlich wird für jeden Partikel sein Geschwindigkeitsvektor (Zeile 12) bzw. sein Positionsvektor (Zeile 13) im Rahmen einer weiteren for Schleife aktualisiert. Beide for Schleifen werden solange für jeweils alle Partikel durchlaufen, bis die im Rahmen der do-while Schleife konkret definierte Abbruchbedingung erreicht ist (Zeile 15).

In Bezug auf den Geschwindigkeitsvektor wird in jeder Dimension eine Limitierung mittels V_{max} vorgenommen [1]. Spezifische Erweiterungen v.a. in Bezug auf die Nachbarschaftsgröße beeinflussen die Qualität des PSO. Dazu werden auch spezielle sternen- oder pyramidenförmige Topologien eingesetzt [32].

Vergleich PSO - GA - ACO: Ähnlichkeiten zwischen PSO, GA und ACO liegen im Bereich der populationsbasierten Lösungssuche. Während beim GA Lösungen mehr oder weniger zufallsgesteuert im Rahmen der Rekombination und Mutation verändert werden, passiert dies beim PSO mittels

der Adaption des Positions- und Geschwindigkeitsvektors. ACO verändert hingegen nicht bestehende Lösungen, sondern jede Ameise konstruiert auf der Basis lokaler und globaler Informationen eigenständig eine Lösung. Während bei GA der Informationsaustausch auf der Basis der chromosomalen Struktur der Individuen vorgenommen wird, gibt beim PSO das beste Individuum seine Informationen an die anderen weiter. Künstliche Ameisen kommunizieren widerum indirekt auf Basis der abgelegten Pheromonmenge jeder Ameise.

Während künstliche Ameisen wie Partikelteilchen über ein eigenes "lokales Gedächtnis" verfügen und eine neue Lösung mit einer alten vergleichen können, besitzen die Individuen im GA kein derartiges lokales Gedächtnis.

Die Gemeinsamkeiten all dieser Verfahren bestehen außerdem darin, dass mehr oder weniger gute Ergebnisse mittels einer zufallsgesteuerten Initialisierung, einer Fitnessbewertung für jedes Individuum sowie einem abschließenden Update der Population erreicht werden [1].

3.4.7 Simulated Annealing (SA)

Simulated Annealing ist auch als Monte Carlo Annealing, statistische Abkühlung, probabilistisches hill-climbing bzw. stochastische Relaxation bekannt. Die Analogien zwischen dem physikalischen System und dem Optimierungsproblem sind in Tabelle 3.1 ersichtlich (Quelle: [23]).

Das Prinzip dieses Verfahrens beruht auf dem Annealing von Kristallen bzw. dem Abkühlungsprozess beim Glühen in der Werkstoffkunde. Diese Idee stammt aus der Metallurgie [3, 24]. Dabei wird ein Gleichgewichtszustand einer Menge von Atomen bei einer gegebenen Temperatur t simuliert [15]. Die Grundidee basiert darauf, dass auch gefundene schlechtere Lösungen mit einer bestimmten Wahrscheinlichkeit akzeptiert werden. Ausgangspunkt ist ein Bewegungszustand, dem bei konstanter Temperatur t eine bestimmte Energie E entspricht. Es wird der Zustand minimaler Energie gesucht. Eine zufällig gewählte Lösung wird einer zufälligen Auslenkung du unterworfen, wobei diese Auslenkung der Energieänderung dE des Systems entspricht. Im Falle einer negativen Energieänderung wird der Zustand sofort akzeptiert, im anderen Fall wird zunächst die Wahrscheinlichkeit des Auftretens dieser Energieänderung berechnet. Diese Berechnung beruht auf dem Boltzmann Gesetz, die Boltzmann Konstante k spielt in der Optimierung keine Rolle und wird daher i.A. weggelassen, sie kann allerdings auch durch eine problemspezifisch angepasste Konstante ersetzt werden [3, 15].

$P(dE) = e^{-dE/(k \cdot t)}$

Diese Wahrscheinlichkeit wird nun mit einer Zufallszahl x verglichen und, sofern $x < P(dE)$, wird die gefundene Lösung als besser akzeptiert. Bei mehrmaliger Wiederholung entwickelt sich das System in den nach dem Boltzmanngesetz wahrscheinlichsten Zustand.

Nach einer definierten Anzahl an Wiederholungen wird die Temperatur t im Rahmen eines Abkühlungsplans vermindert, wobei $t \leftarrow a \cdot t$ und $0 < a < 1$. Falls im Zuge mehrerer Wiederholungen keine Zustandsänderung auftritt, ist das System eingeforen und es wurde zumindest ein lokales Minimum gefunden. Das SA Verfahren ist in Alg. 3.13 beschrieben: (Quelle: [3])

Algorithmus 3.13 Simulated Annealing Prozedur

```
1: begin
2:     t ← 0;
3:     T ← T_init;
4:     x ← Ausgangslösung;
5:     repeat:
6:         repeat:
7:             wähle ein x' ∈ N(x) zufällig;
8:             if x' ist besser als x then
9:                 x ← x'
10:            else
11:                if Z < e^{-|f(x')-f(x)|/T} then
12:                    x ← x'
13:            t ← t + 1;
14:        until Temperatur Upgrade- Kriterium erfüllt;
15:        T ← α · T ;
16:    until Abbruchkriterium erfüllt;    // z.B. T ≤ T_min
17: end
```

Legende:
$Z < e^{-|f(x')-f(x)|/T}$ ..... *Metropolis Kriterium*
Z ..... Zufallszahl$\in [0,1]$
α ..... Konstante, wobei $0 < \alpha < 1$

Der Abkühlungsplan kann auf problemspezifisch geeigneten Schranken basieren und wird so gewählt, dass ca. 3% der Züge anfangs abgelehnt werden. Die Anzahl der Iterationen auf jedem Temperaturniveau kann z.B. ein Vielfaches der Nachbarschaftsgröße sein. Das Abbruchkriterium kann das Erreichen einer bestimmten Temperatur darstellen bzw. wenn über mehrere Temperaturniveaus hinweg keine Verbesserung erreicht wurde oder wenn eine bestimmte Anzahl an Iterationen erreicht wurde [3].

Die Funktionsweise des simulierten Ausglühens kann man anhand einer Suchlandschaft verdeutlichen. Im Gegensatz zum Verhalten einer Kugel, welche zum nächsten Minimum rollt und dort bleibt, wird dieser Kugel bei SA immer wieder ein Stoß versetzt, welcher stark genug ist, um die Kugel aus einem lokalen Minimum zu stoßen, aber nicht ausreicht, um die Kugel aus einem globalen Minimum zu manövrieren, siehe auch Abb. 3.8 (Quelle: [21]) dazu.

Physical System	Optimization Problem
state	feasible solution
energy	evaluation function
ground state	optimal solution
rapid quenching	local search
temperature	control parameter T
careful annealing	simulated annealing

Tab. 3.2: Analogien zwischen dem physikalischen System und dem Optimierungsproblem

Unter bestimmten Bedingungen wird das globale Minimum mit Wahrscheinlichkeit 1 erreicht ([21]):

1. Das globale Minimum muss sich deutlich von den lokalen Minima unterscheiden.

2. Die zugeführte Energie muss geringer sein als zur Flucht aus dem globalen Minimum nötig aber höher als nötig um aus den lokalen Minima zu fliehen.

3. Die Suche darf erst abgebrochen werden, wenn ein Minimum gefunden wurde, das nicht mehr verlassen werden kann.

Die folgende Abbildung verdeutlicht Bedingung 2 anhand einer typischen Suchlandschaft ([21, 42]):

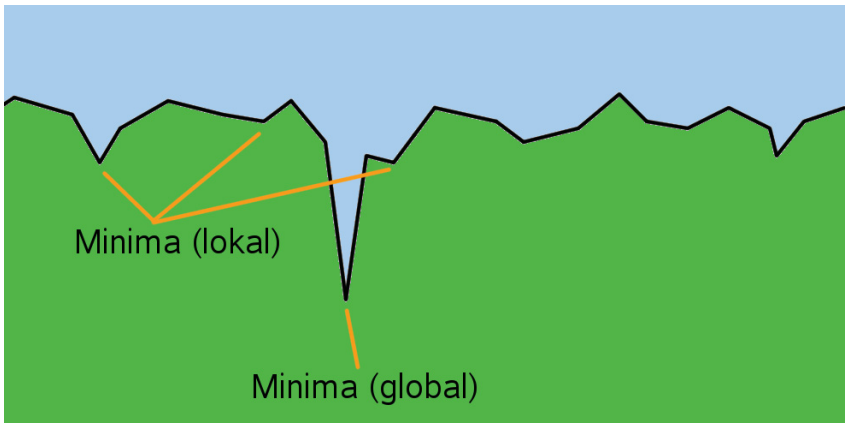


Abb. 3.8: Suchlandschaft mit lokalen Minima und globalem Minimum

Für ein TSP kann eine Implementierung z.B. eine zufällige Ausgangsrundreise mit einer 2- exchange Nachbarschaft beinhalten. Bei einer geometrischen Abkühlungsrate ($\alpha = 0.95$)sowie $n \cdot n - 1$ Iterationen pro Temperatur und z.B. als Abbruchskriterium 5 Temperaturen ohne Verbesserung wählt, erreicht man nur relativ schlechte Ergebnisse - verglichen mit führenden Verfahren für das TSP. Verbesserungen erreicht man durch eine gute Initialisierung bzw. eine auf vielversprechende Züge beschränkte Nachbarschaft.

Ingo Wegener zeigt, dass der SA Ansatz dem Metropolis Algorithmus für ein minimum spanning tree Problem überlegen ist [22].

Andere Verbesserungen stellen nichtmonotone Abkühlungspläne (*"reheating"*) dar, sowie die Überlegung zwischen dynamischen und statischen Abkühlungsplänen sowie die Kombination mit anderen Suchverfahren, z.B. lokaler Suche.

Zusammenfassend lässt sich sagen, dass SA ein sehr einfach zu implementierendes Verfahren darstellt und für viele Probleme zu guten Ergebnissen führt, aber i.d.R. keine Spitzenergebnisse erzielt. Durch einen langsamen Abkühlungsprozess gepaart mit vielen Iterationen auf derselben Temperatur entsteht dabei oft eine relativ lange Laufzeit [3].

Sintflutalgorithmus

Ein SA Derivat ist der Sintflutalgorithmus, welcher auch als *Great Deluge* Algorithmus bekannt ist [24]. Die Idee basiert darauf, eine zufällige Suche im Suchraum durch einen steigenden Wasserspiegel kontinuierlich einzuschränken. Eine schlechtere Lösung wird hierbei nur dann akzeptiert, wenn die neue Lösung einen vorgegebenen Zielfunktionswert nicht unterschreitet (bzw. überschreitet bei Minimierungsproblemen) Diese Zielfunktionswertschwelle entspricht einer Sintflut, die allmählich ansteigt. Es werden also nur Lösungen *"oberhalb des Wasserspiegels"* angenommen. Dadurch können v.a. anfangs lokale Optima überwunden werden.

Im Vergleich zu SA wird die Akzeptanz Regel, welche im SA durch das Metropolis Kriterium dargestellt ist, im Sintflutalgorithmus modifiziert: Bessere Lösungen werden sofort akzeptiert, schlechtere Lösungen nur dann, wenn sie unter dem Sintflutwert liegen. Dieser Wert muss im Laufe der Iterationen angepasst werden.

Threshold Accepting (TA)

TA wurde basierend auf SA 1990 eingeführt [24]. TA funktioniert wie SA mit dem Unterschied, dass schlechtere Lösungen bei TA immer dann akzeptiert werden, wenn die funktionsbezogen negative

Differenz einen gewissen Schwellenwert T_i (threshold) nicht übersteigt. Entsprechend der Temperaturanpassung im SA ist bei TA eine iterative Anpassung des Schwellwertes notwendig.

Während TA und Sintflutalgorithmus bei Optimierungsproblemen häufig in einem sogenannten *"Golfloch- Optimum"* (=ein lokales Optimum, dessen Funktionswert stark von den benachbarten Lösungen abweicht (Quelle: [24])) steckenbleiben, besteht bei SA auf Grund der Wahrscheinlichkeitsentscheidung die Möglichkeit, diese Lösung wieder zu verlassen.

3.4.8 Tabu Search (TS)

Tabu Search basiert auf der Ausnutzung eines Gedächtnisses des bisherigen Suchprozesses. Die Ziele liegen in der effizienten Vermeidung lokaler Optima sowie in der Ausnutzung des vorhandenen Gedächtnisses, welches den Lösungsverlauf speichert [34]. Dieses Gedächtnis übernimmt die strategische Leitung im Suchprozess. Während das Kurzzeitgedächtnis Schleifen vermeiden soll, wird im Rahmen des mittelfristigen Gedächtnisses die Suche intensiviert, das Langzeitgedächtnis soll die Suche diversifizieren.

In der jeweils aktuellen Nachbarschaft wird in jedem Schritt die beste benachbarte (auch schlechtere) Lösung akzeptiert. Dadurch entstehende Zyklen sollen durch das Verbot des wiederholten Besuchens von Lösungen vermieden werden, dieser Prozess soll durch das Gedächtnis sichergestellt werden.

Ein einfacher TS Algorithmus verwendet z.B. nur ein Kurzzeitgedächtnis, durch welches die zulässige Nachbarschaft eingeschränkt wird. Besuchte Lösungen werden dabei in einer Tabu Liste gespeichert. Dieses Vorgehen ist speicherintensiv und mit einer zeitaufwendigen Überprüfung verbunden. Die Wahl der richtigen Tabu-Listenlänge ist im TS eine sehr wesentliche und kritische Einstellung, während zu kurze Tabu Listen zu Zyklen führen, schränken zu lange Tabu-Listen die Suche dementsprechend stark ein [34, 3].

Verschiedene *"Aspirationskriterien"* machen die Ausführung eines tabuisierten Zuges dennoch möglich indem sie den Tabustatus interessanter Lösungen überschreiben, dadurch kann im Falle eines betroffenen Attributs der Tabu Status dieses Attributs inaktiv gesetzt werden [3, 34]. Mögliche Aspirationskriterien bestehen im Überschreiben des Tabu Status interessanter (lokal sehr guter) Lösungen bzw. darin, dass die verbotene Lösung besser als die beste bisher gefundene Lösung ist [35].

Der Pseudo Code eines TS ist in Alg. 3.14 (Quelle: [39]) beschrieben.

Algorithmus 3.14 Tabu Search

1: $x_{best} = x$ =Ausgangslösung;
2: TL = $\{x\}$; // Tabu Liste TL
3: **wiederhole**
4: X' =Teilmenge von $N(x)$ unter Berücksichtigung von TL;
5: x' =beste Lösung von X';
6: füge x' zu TL hinzu;
7: lösche Elemente aus TL welche älter als t_L Iterationen;
8: $x = x'$;
9: **falls** x besser als x_{best} **dann** {
10: $x_{best} = x$;
11: }
12: **bis** Abbruchkriterium erfüllt

In der Tabu Liste TL werden Informationen zu den bereits generierten Lösungen gespeichert. Die Nachbarschaft $N(x)$ wird derart eingeschränkt (Zeile 4), sodass besuchte Lösungen nicht unmittelbar wieder ausgewählt werden. Die Auswahl der konkreten Nachfolgelösung erfolgt bei TS generell deterministisch, d.h. im Sinne der Strategie *best improvement*. Diese wird anschließend immer als nächste aktuelle Lösung x akzeptiert, auch im Falle einer schlechteren Nachfolgelösung. Die insgesamt beste ermittelte Lösung wird mittels x_{best} gespeichert und zurückgegeben [39].

Robust Tabu Search (RoTs)

Diese speziell für das QAP (siehe 3.2.2) entwickelte TS Strategie bestimmt die Tabu- Listenlänge t_l zufällig aus dem Intervall $[t_{l,min}, t_{l,max}]$. In definierten Abständen (z.B. alle $2 \cdot t_{l,max}$ Iterationen) wird t_l neu bestimmt, wodurch das Problem der optimalen Tabu-Listenlänge umgangen wird [34]. Zusätzlich kann eine Lösung angenommen werden, wenn das Lösungsattribut seit mindestens m (hoher Wert für m) Iterationen nicht mehr geändert worden ist, wodurch Diversifikation möglich ist.

Reactive Tabu Search (ReTs)

Die ReTs Strategie wurde für das QAP und Knapsack Problem entwickelt. Basierend auf einer durchschnittlichen Zyklenlänge m wird diese Variable bei der Erkennung eines Zyklus um einen konstanten Faktor erhöht, während es um denselben konstanten Faktor m erniedrigt wird, wenn die letzte Erhöhung mehr als m Iterationen zurückliegt [34]. ReTs verfügt über einen Diversifikationsmechanismus, welcher ausgeführt wird, wenn die Anzahl der Lösungen, die öfter als vordefiniert wiederholt besucht wurden, einen Grenzwert überschreitet [3, 34].

Statistisch betrachtet wird mit diesen beiden TS Strategien eine höhere Lösungsgüte als mit SA erreicht. Zusammenfassend lassen sich viele unterschiedliche Varianten und Strategien in Bezug auf Gedächtnis, Diversifizierung und Intensivierung implementieren. Weiters ist TS ein Verfahren, das in relativ kurzer Zeit sehr gute Ergebnisse erreicht.

3.5 Vergleich von Metaheuristiken

I.d.R. wird die Laufzeitkomplexität anhand der erforderlichen Laufzeit des Verfahrens, die Speicherkomplexität anhand des nötigen Speicherbedarfs sowie die Lösungsgüte verglichen. Weiters ist die Robustheit des Verfahrens interessant [3]. Die Lösungsgüte wird dabei durch die Abweichung der durch die Metaheuristik erreichte Lösung vom Optimum dargestellt [34]. (*"percentage excess"*) Konkret wird die Lösungsgüte $q(x)$ folgendermaßen ermittelt:

$q(x) = (\frac{L(x_{heu})}{L(x_{opt})} - 1) \cdot 100\%$ (Quelle: [34])

Ein $q(x)$ Wert von 100% entspricht damit der doppelten Tourlänge im Vergleich zur optimalen Rundreise, während $q(x) = 0\%$ die optimale Tourlänge darstellt. Dieser Wert kann nur ermittelt werden, wenn die optimale Lösung auch bekannt ist. Im konkreten Produktionsproblem ist die optimale Lösung einer Kampagne i.d.R. nicht bekannt, sodass die Lösungsgüte der Algorithmen nur über den direkten Vergleich der Lösungen der Algorithmen selbst ermittelt werden kann.

3.6 Vorgehensweise im Rahmen simulationsgestützter Optimierung

3.6.1 Definitionen

> Simulation: "Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigem Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Im weiteren Sinne versteht man unter Simulation das Vorbereiten, Durchführen und Auswerten gezielter Experimente mit einem Simulationsmodell." (Quelle: [36])
>
> Modell: "Ein Modell ist eine vereinfachte Nachbildung eines geplanten oder real existierenden Systems mit seinen Prozessen in einem anderen System. Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Ziel abhängigen Toleranzrahmens vom Original." (Quelle: [36])

3.6.2 Ablauf und Durchführung einer Simulationsstudie

Nach VDI 3633 ([36]) empfiehlt sich i.A. folgende Vorgehensweise:

1. Problemformulierung
2. Prüfung der Simulationswürdigkeit
3. Zielformulierung
4. Datenbeschaffung und -analyse
5. Modellierung
6. Durchführung von n Simulationsläufen
7. Ergebnisanalyse und -interpretation
8. Dokumentation

Die Problemformulierung erfolgt i.d.R. mit dem Auftraggeber, mit welchem gemeinsam die Anforderungen an die Simulation definiert werden. Das Ergebnis stellt das Pflichtenheft dar [36].

Zur Einschätzung der Simulationswürdigkeit wird neben der (wiederholten) Verwendung des Simulationsmodells die Komplexität, die zu berücksichtigenden Einflüsse, das Fehlen analytischer mathematischer Modelle sowie die Datenunsicherheit (auch bezüglich der Informationen seitens des Auftraggebers) betrachtet.

Häufig zu optimierende Zielgrößen stellen im Rahmen der Simulation von Fertigungssystemen Durchlaufzeitminimierung, Auslastungsmaximierung, Bestandsminimierung sowie die Erhöhung der Termintreue dar.

Im Rahmen der Datenbeschaffung muss auf die Besonderheiten des spezifischen Problems Rücksicht genommen werden. Die Daten lassen sich allgemein in Systemlastdaten, Organisationsdaten sowie technische Daten gliedern.

Die Modellierungsphase beschäftigt sich mit der Erstellung des Simulationsmodells. Dabei wird i.d.R. zuerst ein gedankliches Modell in ein symbolisches Modell umgewandelt, welches anschließend in ein Softwaremodell umgesetzt wird. Die erste Modellierungsstufe besteht v.a. aus der Analyse und Abstraktion. Im Rahmen der Systemanalyse wird die Komplexität des Gesamtsystems in seine Einzelteile aufgegliedert. Abstraktion reduziert die systemspezifischen Systemkennzeichen, sodass ein auf die wesentlichen Systemelemente beschränktes Abbild des Originalsystems entsteht. Im Rahmen der Systemreduktion wird auf nichtrelevante Einzelheiten verzichtet, während die Idealisierung auf die Vereinfachung unverzichtbarer Einzelheiten abzielt [36]. Die zweite Modellierungsstufe enthält den Aufbau und Test sowie die Dokumentation des Simulationsmodells. Für die Dokumentation empfiehlt es sich generell, die Dokumentation während der Programmierung des Quellcodes zu erstellen.

Im Rahmen eines Versuchsplans sind anschließend Experimente durchzuführen und die Ergebnisse festzuhalten, welche sich auf spezifische Ausgangsdaten(-sätze) sowie Optimierungsparameter beziehen.

Die Ergebnisanalyse soll nach der richtigen Interpretation der Simulationsergebnisse Maßnahmen ableiten, die das modellierte System positiv verändern werden.

In der letzten Phase sollte ein Projektbericht erstellt werden, der Aufschluss über den zeitlichen Überblick sowie Verlauf der Simulationsstudie gibt.

3.6.3 Simulationsgestütze Optimierung

Im Rahmen simulationsgestützter Optimierung kann das Simulationsmodell als System verstanden werden, auf das eingangsseitig bestimmte Einflussgrößen wirken und welches ausgangsseitig mit den zu optimierenden Zielwerten reagiert. Falls die manuelle (try and error basierende) Parametrisierung der Eingangsgrößen durch einen geeigneten Suchalgorithmus ersetzt wird, erhält man als Ergebnis einen geschlossenen, eigenständig handelnden Optimierungszyklus, siehe auch Abb. 3.5 (Quelle: [37]).

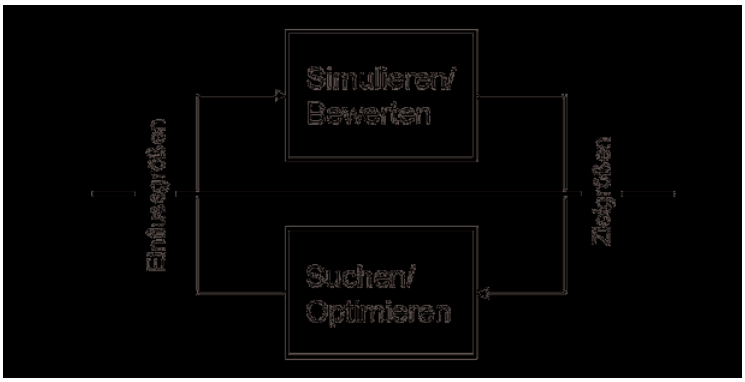


Abb. 3.9: Prinzip simulationsgestützter Optimierung

Kapitel 4

4 Genetische Algorithmen

4.1 Charakteristika von genetischen Algorithmen

Die Idee der genetischen Algorithmen (GA) wurde 1975 von Holland in seinem Buch vorgestellt und seither stetig weiterentwickelt, u.a. von Goldberg [5] und Michalewicz [4]. Die heuristische Idee hinter GA basiert auf dem natürlichen aus der Biologie stammenden *"survival of the fittest"* (Zitat: Charles Darwin, Darwin´sche Evolutionstheorie) Prinzip.

Nach den Prinzipien der Natur werden dabei die unterschiedlichsten Lösungen anhand ihrer sogenannten "Fitness" bewertet. Diese Bewertungsfunktion repräsentiert dabei die Charakteristika des speziellen Problems. Im Rahmen der Selektion werden die Individuen, die sich paaren sollen, ausgewählt und anschließend gekreuzt. Wie bei der natürlichen Vererbung werden dabei die Eigenschaften der aktuellen Lösungen (Eltern) an ihre Nachkommen (Kinder) weitergegeben.

Genetische Algorithmen unterscheiden sich grundlegend von anderen Heuristiken in Bezug auf ihre Konzeption, weil sie nicht - wie lokale Suchverfahren - einzelne Punkte im Suchraum durchforsten, sondern auf Grund ihrer Population, bestehend aus einzelnen Lösungen, eine multidirektionale Suche durchführen[18].

Genetische Algorithmen sind für die unterschiedlichsten kombinatorischen Optimierungsprobleme geeignet (TSP, Scheduling, Reihenfolgeplanung,...) und unterscheiden sich von anderen Optimierungsalgorithmen durch folgende Kriterien[1],[10]:

Vorteile genetischer Algorithmen:

1. Gute Eignung für Probleme mit großen Suchräumen, über die wenig (besondere) Informationen bekannt sind.
2. Kaum Einschränkungen in Bezug auf die zu optimierende Funktion (Stetigkeit, Ableitbarkeit,...).
3. Leichte Anwendbarkeit für verschiedenste Probleme nach nur geringen Modifikationen.
4. Komplexe Fitnesslandschaft, die einen großen Lösungsraum abdecken kann.
5. Störfunktionen können relativ gut berücksichtigt werden.
6. Durch ihre globale Sichtweise kann relativ oft das globale Optimum gefunden werden und nicht (nur) das nächste lokale Optimum.
7. Der populationsbasierende Charakter des GA verleiht der Metaheuristik eine bessere Resistenz gegenüber lokalen Minima.

Nachteile genetischer Algorithmen:

1. Manchmal ist es schwierig, eine möglichst problem-getreue Fitness Funktion zu identifizieren und realisieren.
2. Hohe Anzahl an Fitness Bewertungen notwendig.

3. Frühzeitige Konvergenz zu einem lokalen Minimum möglich.
4. Die Wahl der richtigen Parametrisierung.
5. Keine natürliche Abbruchbedingung vorhanden.

4.1.1 Kodierung und Aufbau

Die *2 Schlüsselelemente* in einem genetischen Algorithmus repräsentieren seine *Population* $P(t)$, welche eine definierte Anzahl an *Individuen* enthält. Jedes Individuum stellt dabei eine gültige Lösung dar. Dabei unterscheidete man die kodierte Form eines Individuums, das sogenannte *"Chromosom"* oder den *"Genotyp"*, und die dekodierte Form, den *"Phänotyp"* [3],[1]. Ein Chromosom besteht wiederum aus *"Genen"*, welche konkrete Werte, sogenannte *"Allele"*, bezeichnen [15]. Die Chromosomenlänge ist i.A. für alle Individuen gleich. Bei einer Auftragsreihenfolge eines PFSP stellen beispielsweise die Auftragsnummern die Gene dar.

Algorithmus 4.1 zeigt den grundsätzlichen Aufbau eines GA [3, 10]. Nach der Generierung einer Ausgangspopulation wird diese bewertet. Solange die *Abbruchbedingung* nicht eintritt, wird iterativ eine neue Population mittels Selektion gezüchtet und den Crossover (Kreuzung) und Mutations Operatoren unterzogen. Die neue Population wird danach bewertet und der ganze Zyklus wiederholt sich von vorne. Die Abbruchbedingung tritt i.d.R. nach einer bestimmten Anzahl an Generationen, nach einer bestimmten definierten Laufzeit oder dann ein, wenn sich innerhalb einer definierten Anzahl an Generationen nichts mehr an der besten Fitness ändert. Weitere mögliche Abbruchbedingungen beziehen sich auf die Konvergenz in Bezug das beste/schlechteste Individuum einer Population bzw. die aufsummierte Fitness oder die Median Fitness [1].

Algorithmus 4.1 Standard Genetischer Algorithmus (SGA)

1: $t \leftarrow 0$;
2: initialize $(P(t))$;
3: evaluate $(P(t))$;
4: **while** (not termination-condition) **do**
5: $t \leftarrow t+1$; //increase iteration
6: $Q_s(t) \leftarrow$ select$(P(t-1))$; // select individuums from the population of the last iteration
7: $Q_r(t) \leftarrow$ recombine$(Q_s(t))$; // the selected individuums get recombined with a specific probability
8: $P(t) \leftarrow$ mutate$(Q_r(t))$; // the (recombined) individuums get mutated with a specific probability
9: evaluate$(P(t))$; //evaluate the altered population
10: **end while**

4.1.2 Holland´s Schema Theorem und die Building Block Hypothese

Ein Schema ist eine Art Muster, welches eine Teilmenge aller Individuen mit gleichen Werten an bestimmten Positionen definiert [3]. Die Ordnung o(H) eines Schemas H wird durch die Anzahl der fixen Gene im Chromosom bestimmt. Die definierende Länge $\delta(H)$ beschreibt den Abstand zwischen dem ersten und letzten festgelegten Zeichen eines Schemas H. Nach einigen weiteren Überlegungen bezüglich der durchschnittlichen Fitness eines Schemas und deren Vermehrungsraten bei Kreuzung und Mutation gelangt man zum fundamentalen Schema Theorem der genetischen Algorithmen. Für die genaue Herleitung vergleiche man [15] bzw. [3] bzw. [4]. ***Das Schema Theorem besagt, dass sich Schemata mit hoher Fitness, kleiner definierender Länge und niedriger Ordnung in der Population mit exponentieller Geschwindigkeit vermehren. Diese Schemata werden auch building blocks genannt. Ein genetischer Algorithmus führt durch das Zusammenfügen derartiger Schemata eine effiziente Suche nach dem Optimum durch [3, 15, 4].***

Neben der Kodierung des Problems spielt daher die Anordnung der Gene im Chromosom eine besondere Rolle.

4.2 Initialisierung

Abgesehen von einer rein zufälligen Initialisierung besteht generell die Möglichkeit eine Ausgangspopulation durch eine Heuristik zu erzeugen oder man erhält bereits eine Ausgangspopulation aus einem Testdatensatz. Da dies im konkreten Produktionsproblem der Fall ist, besteht die Ausgangspopulation aus mutierten Individuen aus der Ausgangssequenz. Prinzipiell muss auf die Gültigkeit der Ausgangslösung geachtet werden. Außerdem sollte auch bei der Initialisierung für eine möglichst große Vielfalt unterschiedlichster Lösungen gesorgt sein [3].

Generell empfiehlt es sich, eines bzw. mehrere relativ fitte Individuen anhand einer geeigneten Heuristik zu erzeugen [13].

4.3 Primäre Operatoren

4.3.1 Selektion

Im Rahmen der Selektion gelangen neue und mit großer Wahrscheinlichkeit bessere Individuen, welche als Elternteile für die anschließende Kreuzung fungieren, entsprechend der *"natürlichen Auslese"* in den sogenannten *"Paarungspool"*. Die im Paarungspool befindlichen Individuen werden für die anschließende Rekombination bzw. Mutation zur Erstellung der neuen Population verwendet. Die Selektion treibt damit im GA die Gesamtpopulation in die Richtung besserer Lösungen bzw. der optimalen Lösung [3].

4.3.1.1 Selektionsdruck

Im Rahmen der Selektion ist es wichtig, den Selektionsdruck richtig einzustellen. Der Selektionsdruck definiert in welchem Ausmaß bessere (= fittere) Lösungen in den Paarungspool kommen im Gegensatz zu schlechteren Individuen. Während man in den Anfangsstadien eine möglichst vielfältige Population haben möchte, die nicht frühzeitig konvergiert, möchte man am Ende des GA doch eine möglichst optimale Lösung finden [3].

Den Selektionsdruck kann man z.B. durch das Verhältnis der Fitness des besten Individuums zur durchschnittlichen Fitness der Individuen in der Population bestimmen, d.h.: $S = f_{\max} / f_{\text{av}}$

Wenn der Selektionsdruck zu niedrig eingestellt ist, wird das Streben nach einer guten Lösung gering sein und der GA schleppt lange Zeit schlechte Individuen von einer Population in die nächste mit, während sich gute Individuen kaum vermehren, und das Verfahren degeneriert zur Zufallssuche [10, 1, 3].

Bei einem zu hoch eingestellten Selektionsdruck vermehren sich gute Individuen sehr rasch, während die Individuenvielfalt im selben Ausmaß abnimmt. Der GA läuft Gefahr, frühzeitig gegen ein lokales Minimum zu konvergieren. Im Rahmen der Selektion werden folgende Mechanismen unterschieden, welche sich in der Selektionswahrscheinlichkeit der einzelnen Individuen unterscheiden.

4.3.1.2 Fitnessproportionale Selektion (roulette wheel selection)

Im Rahmen dieser traditionellen Methode werden die Individuen für den Paarungspool proportional gemäß ihrere Fitness f (S_i) zufällig ausgewählt, siehe auch Abb. 4.1 (Quelle: [3]) dazu.

$p_s(S_i) = f(S_i) \frac{f(S_i)}{\sum_{j=1}^{n} f(S_j)}$

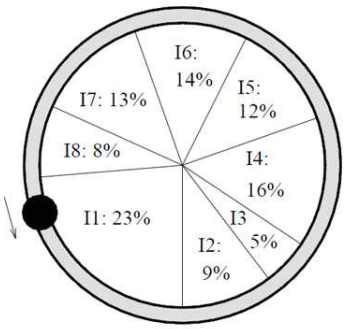


Abb. 4.1: Fitness proportionale Selektion

$p_s(S_i)$ ist dabei die Wahrscheinlichkeit mit der ein Individuum S_i bei der Selektion ausgewählt wird, n ist die Populationsgröße. Dieses Verfahren ist auch unter dem Namen *"Roulette- Wheel Selektion"* bekannt. Es gilt: $\sum_{i=1}^{n} p_{\mathrm{s}}(S_{\mathrm{i}}) = 1$. Die Kugel steht für eine zufällige Zahl im Intervall [0,1] Es ist außerdem zu beachten, dass die Population nicht nach ihrer Fitness geordnet ist. Individuen mit einem hohen Anteil an der Gesamtfitness werden dementsprechend oft ausgewählt, während weniger fitte Individuen mit sehr geringer Wahrscheinlichkeit in den Paarungspool kommen. Wenn sich die Gesamtfitness aus wenigen Individuen mit hoher Fitness zusammensetzt und viele andere Individuen im Vergleich zu diesen *"Superindividuen"* eine relativ geringe Fitness aufweisen, ist dieses Verfahren nicht für die Selektion geeignet.

Es besteht außerdem auch die Möglichkeit sogenannte Skalierungsfunktionen einzusetzen. Man unterscheidet dabei zwischen direkter, linearer und geometrischer Skalierung [10].

Bei direkter Skalierung ($f(S_{\mathrm{i}}) = g(S_{\mathrm{i}})$) wird keine Trennung zwischen der Bewertungsfunktion $g(S_i)$ sowie der Fitnessfunktion $f(S_i)$ durchgeführt. Lineare Skalierung ($f(S_{\mathrm{i}} = a \cdot g(S_{\mathrm{i}}) + b$) [3, 4]) steuert mit den beiden Konstanten a,b das Verhältnis zwischen maximaler und durchschnittlicher Fitness. Für ein $a > 1$wird die Fitness höherbewerteter Individuen weiter angehoben, während sie für ein $a < 1$ herabgesetzt wird.

Bei geometrischer Skalierung ($f(S_{\mathrm{i}} = g(S_{\mathrm{i}})^{\mathrm{p}}$) werden die Fitnesswerte in Abhängigkeit von p skaliert.

Während Holland vorschlug, ein Individuum fitnessproportional auszuwählen und ein zweites Individuum, welches für den crossover benötigt wird, zufällig auszuwählen, schlug Schaffer vor, beide Individuen fitnessproportional auszuwählen. Trotz des relativ geringen Selektionsdrucks der ersten Variante sind die Ergebnisse qualitativ relativ ähnlich [8].

4.3.1.3 Rangbasierte Selektion (rank selection)

Bei diesem Ansatz werden die Individuen gemäß ihrer Fitness so angeordnet, dass das beste Individuum Fitness N hat, während das schlechteste Individuum Fitness 1 hat [1]. Damit wird das Problem der einseitigen Fitnessverteilung, dass z.B. auf ein Individuum 90% der Gesamtfitness entfallen und die restlichen Individuen sich die verbleibenden 10% teilen, eingeschränkt, sodass diese Individuen eine höhere Chance haben, auch in den Paarungspool zu kommen.

Man unterscheidet generell zwischen der linearen rankingbasierten Selektion und einer nichtlinearen rankingbasierten Selektion.

Lineares Ranking

Bei linearem Ranking erhält das Individuum mit der besten Fitness einen s Skalierungsfaktor zwischen 1 und 2, z.B. 2. Das Individuum mit der schlechtesten Fitness erhält einen Wert von $2 - s$ [8]. Die dazwischenliegenden Fitnesswerte werden mittels Interpolation errechnet, wobei der Skalierungsfaktor nach [8] für jedes Individuum

$f(i) = s - \frac{(2i(s-1))}{(N-1)}$ beträgt.

Für $s = 2$ wird die Fitness des besten Individuums mit 2 multipliziert, während das schlechteste Individuum ($s = 0$) nicht ausgewählt wird. Prinzipiell kann man den Wert s auch höher als 2 einstellen, um einen höheren Selektionsdruck zu gewährleisten, allerdings erhält man in diesem Fall für die schlechtesten Individuen negative s Werte. Diese kann man auf 0 setzen, dabei müsste man jedoch auch die übrigen Fitnesswerte neu skalieren. Daher ist es einfacher, sofern ein höherer Selektionsdruck gewünscht ist, diesen über nichlineares Ranking zu erreichen [8].

Exponentielles Ranking

Das Individuum mit der besten Fitness erhält den Wert 1 (entspricht s^0) , das Individuum mit der zweitbesten Fitness den Wert s (entspricht s^1) , welcher z.B. 0.99 beträgt. Das Individuum mit der drittbesten Fitness erhält den Wert s^2 usw. Das Individuum mit der schlechtesten Fitness erhält den Wert $s^{(N-1)}$[8].

Der Selektionsdruck ist proportional zu $1-s$. Das bedeutet, dass $s = 0.994$ rein mathematisch betrachtet eine doppelt so schnelle Konvergenz zu einer guten Lösung ergibt wie $s = 0.998$.

4.3.1.4 Tournament Selektion

Basierend auf der rangbasierten Selektion stellt die Tournament Selektion eine effiziente und relativ schnell zu implementierende Selektionsmethode dar. Hierbei werden k *Individuen zufällig aus der Population ausgewählt*, anhand deren bereits bewerteten Fitness verglichen, und anschließend wird das bestbewertete Individuum unter den k Individuen ausgewählt. Dieses Individuum gelangt somit in den Paarungspool. Der Parameter k dient zum Einstellen des Selektionsdrucks. Bei steigendem k steigt auch der Selektionsdruck.

Mittels Tournament Selektion ($k = 2$) kann auch lineares Ranking mit $s < 2$ emuliert werden. Eine Wahrscheinlichkeit von 0.8, dass das bessere Individuum im Rahmen der Tournament selektiert wird, entspricht dabei einem Wert für $s = 2 \cdot 0.8 = 1.6$. Dieser Wert entspricht dann dem Skalierungsfaktor im linearen Ranking [8]. Goldberg und Deb zeigen, dass das erwartete Resultat einer Tournament mit $n = 2$ exakt dasselbe Resultat darstellt wie lineares Ranking mit einem Wert für $s = 2$.

Der Vergleich einer Tournament Selektion ($k = 4$) mit exponentiellem Ranking ($s = 0.96$) ergibt, dass das beste Individuum unter 100 Individuen jeweils ca. 4 Mal selektiert wird. Der Unterschied besteht v.a. darin, dass beim exponentiellem Ranking etwas weniger durchschnittliche Individuen selektiert werden zu Gunsten der schlechter bewerteten Individuen, welche etwas häufiger selektiert werden als bei Tournament Selektion [8]. Als weitere Konsequenz lässt sich dadurch feststellen, dass die Tournament Selektion in dieser Konfiguration ($k = 4$) etwas mehr Selektionsdruck verursacht als die verglichene Konfiguration des exponentiellen Rankings.

Die Vorteile der Tournament Selektion liegen dort, wo ein einzelnes Individuum unabhängig von anderen nicht bzw. kaum bewertet werden kann. Tournament Selektion bietet sich auch dort an, wo die Fitness relativ, also in Bezug auf andere Individuen, relativ gering abweicht. Daher wird Tournament Selektion im Rahmen der Implementierung eines hybriden genetischen Algorithmus in dieser Arbeit verwendet.

Für die Selektionswahrscheinlichkeit eines einzelnen Individuums i gilt bei einem beliebigen Wert für den Parameter k nach [3]: $p_s(i) = (1 - (1 - \frac{1}{n})^k) \cdot (1 - \frac{i}{n})^{k-1}$

Der (Laufzeit-) Aufwand für die Tournament Selektion beträgt $O(k)$, für $k = 2$ z.B. $O(2)$ und ist damit konstant.

4.3.1.5 Weighted Tournament Selektion

Um den Selektionsdruck feiner einzustellen, nimmt man von den k ausgewählten Individuen nicht automatisch das beste, anstelle dessen wird das beste mit einer Wahrscheinlichkeit p_1 selektiert, das zweitbeste mit einer Wahrscheinlichkeit p_2 usw., wobei $p_2 < p_1$[3].

Dabei gilt: $\sum_{i=1}^{k} p_\mathrm{i} = 1$

4.3.1.6 Boltzmann Selektion

Die Boltzmann Selektion greift auf das Prinzip von Simulated Annealing (3.2.5) zurück. Dabei steuert die sich kontinuierlich verändernde Temperatur den Selektionsdruck im GA [1]. Die Temperatur hat anfangs einen hohen Wert, wodurch ein geringer Selektionsdruck und eine große Variation sichergestellt ist. Die Temperatur wird anschließend schrittweise verringert, wobei sich der Selektionsdruck erhöht. Ausgehend vom aktuell fittesten Individuum wird jedes bessere in jedem Fall selektiert, während schlechter bewertete Individuen mit der Boltzmann Wahrscheinlichkeit selektiert werden.

Diese beträgt $p = exp[(f_{min} - f(X_i))/T]$

Dabei sind $T = T_0 \cdot (1 - \alpha)^k$ und $k = (1 + 100 \cdot g/G)$, wobei g die aktuelle Generation darstellt und G die Gesamtanzahl an Generationen repräsentiert. Der konkrete Wert im Exponent ist immer im negativen Wertebereich. Die Abkühlungsrate α ist prinzipiell zwischen [0,1] wählbar. T_0 ist generell zwischen [5,100] wählbar. Typische Werte sind z.B. $\alpha = 0.95$ und T_0 =10 bzw. 15.

4.3.1.7 Elitäre Selektionsmethode

Während die Elitismus Option auch in anderen Selektionsvarianten vorkommen kann in der Form, dass sichergestellt wird, dass das beste Individuum in jedem Fall mit in die nächste Generation genommen wird, versteht man unter elitärer Selektion, dass ausschließlich die besten Individuen selektiert werden. Diese Variante bedeutet Einschränkung an Individuenvielfalt und empfiehlt sich daher v.a. im Endstadium des GA.

Im Rahmen der von Amous, Loukil, Elaoud und Dhaenens vorgestellten elitären Selektionsmethode werden die n besten (verschiedenen) Individuen aus der Population ausgewählt, welche jeweils mehrfach in den Paarungspool gelangen um sich dort im Rahmen der Rekombination bzw. Mutation zu vermehren [18]. Aus der Gesamtpopulation bestehend aus N Individuen vermehren sich n Individuen, $n = E(\sqrt{N})$, wobei E als ganzzahliger Vermehrungsfaktor den durch den Wurzelausdruck erhaltenen Wert abrundet.

Um die Populationsgröße exakt unverändert groß bleiben zu lassen, können die restlichen für die Selektion verbleibenden Individuen n' zufällig oder anhand einer anderen Selektionsmethode gewählt werden. Der Grund dafür ist, dass E den größten ganzzahligen Faktor-Wert repräsentiert, welcher aber i.d.R. nicht ganzzahlig ist. Die verbleibende Differenz entspricht aufsummiert der Anzahl an Individuen, die zufällig oder anders selektiert werden. Dabei entspricht $n' = N - C \cdot n^2$ ([18]).

4.3.1.8 Zusammenfassung und Reflexion der Selektionsstrategien

Auf Grund der Vorstellung der Selektionsmethoden ist ersichtlich, dass es relativ wenige signifikante Unterschiede zwischen den verschiedensten Selektionsmethoden gibt. Die Entscheidungen für bzw. gegen einen bestimmten Selektionsoperator sollten daher folgende Punkte beinhalten:

1. Eine Entscheidung für rangbasierte Selektionsmethoden (Rankingbasierte Selektion, Tournament Selektion) beinhaltet eine bessere, weil gezielter mögliche, Steuerung des Selektionsdrucks, wobei dabei die Verknüpfung zum konkreten dahinterstehenden Fitnesswert sowie zum Reproduktionserfolg im Vergleich mit direkten Fitness Selektionsmaßnahmen (Fitnessproportionale Selektion,...) verloren geht.

2. Von zentraler Bedeutung im Rahmen der Selektion ist die Steuerung des Selektionsdrucks. Den Selektionsdruck kann man im Rahmen der jeweiligen Selektionsmethode parameterbedingt erhöhen bzw. senken (größere Tournament im Rahmen der Tournament Selektion, exponentielles Ranking: Selektionsdruck $\sim 1 - s$,...). Andererseits kann man auch unabhängig von der jeweiligen Methode den Ersetzungsmechanismus prinzipiell verändern bzw. abändern, siehe dazu auch Abschnitt 4.5.
3. Weiters steht man vor der Entscheidung, ob man eine absichtliche Nichtlinearität zwischen Fitness und zugehörigem Nachkommen wünscht oder dies ablehnt. Im ersten Fall bedient man sich eher einer Variante der Tournament Selektion. Exponentielles Ranking selektiert neben vielen sehr gut bewerteten Individuen auch noch relativ viele schlecht bewertete Individuen, dafür weniger durchschnittliche Individuen [8]. Es kann auch zielführend sein, die Selektionsmethode während des Algorithmus überhaupt zu ändern, abgesehen von einer iterativ angepassten verfeinerten Parametrisierung der jeweiligen Methode.
4. Es empfiehlt sich generell, eine Variante von Elitismus (siehe Unterabschnitt 4.7.3) im Rahmen der jeweiligen Selektionsmethode zu implementieren, weil dadurch oft ein verbessertes Suchverhalten gewährleistet wird [8].
5. Im Rahmen der Selektion hat man sich grundsätzlich auch für einen generationären Ansatz oder einen inkrementellen Ansatz zu entscheiden, nähere Erklärungen dazu folgen in Abschnitt 4.5.

4.3.2 Crossover (Recombination)

Crossover (dt.: Kreuzung) ist der Prozess, bei dem 2 Individuen, die Elternteile, gekreuzt werden um ein neues Individuum (→"Nachkommen", "offspring") zu erzeugen. Ein offspring soll dabei v.a. aus Attributen aufgebaut werden, die auch in der Elterngeneration vorhanden sind.(Prinzip der Genvererbung) Allgemein werden 2 Individuen zufällig als Elternteile zum Kreuzen ausgewählt. Danach wird ein bestimmter crossover Operator angewandt, der neben der Anzahl und Art der Kreuzungsstellen die Anzahl der Gene (Bereich) definiert, die getauscht werden sollen.

Die Rekombination wird i.A. für alle selektierten Individuen oder einen sehr großen Teil der im Paarungspool befindlichen Individuen der Population durchgeführt.

4.3.2.1 1-,2-,N-Point Crossover

Beim sogenannten "single point crossover" wird eine Kreuzungsstelle zufällig ausgewählt und danach das Genmaterial eines Elternteils bis zu dieser Stelle genommen, während der 2. Teil des Genmaterials nach der Kreuzungsstelle aus dem anderen Elternteil entnommen wird, siehe auch Abb. 4.2. (Quelle: [3])

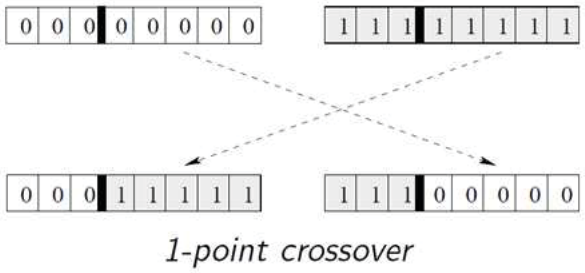


Abb. 4.2: 1 Point Crossover

Beim 2 point crossover geht man analog vor, es ist generell zu berücksichtigen, dass man 2 Kreuzungspunkte pro Individuum definiert und jedes Individuum in 3 Teile teilt. Dabei stammt der 1. und 3. Teil

des ersten offsprings vom 1. Elternteil, während der 2. Teil vom 2. Elternteil stammt. Analoges gilt für offspring 2.

Generell beinträchtigen mehrere Kreuzungsstellen beim 2 point bzw. N-point crossover die Performance des GA insofern, als bei mehreren Kreuzungsstellen die building blocks eher zerstört werden und die Schemata weniger gut erhalten bleiben als bei einem 1-point crossover. Der Suchraum kann dafür gründlicher durchsucht werden bzw. werden die Schemata besser gemischt. 2 Kreuzungsstellen vereinen widerum den Vorteil, dass der "head" und "tail" des Elternteil Chromosoms jeweils in die nächste Generation mit übernommen wird [1].

Generell lässt sich zum N Point crossover sagen, dass dessen Effizienz entscheidend von der Position der Gene innerhalb des Chromosoms abhängt.

4.3.2.2 Uniform Crossover

Beim uniform crossover wird für jedes Gen entschieden, von welchem Elternteil es entnommen wird. Dabei wird ein Vektor erstellt, der für jedes Gen des Chromosoms die Zufallszahl 0 oder 1 bereitstellt, wobei 1 für den Austausch der Gene steht, während 0 keinen Austausch zur Folge hat, siehe Abb 4.3 (Quelle: [3]) Prinzipiell vererben beide Elternteile jeweils ca. die Hälfte ihrer Gene an den Nachkommen weiter [7].

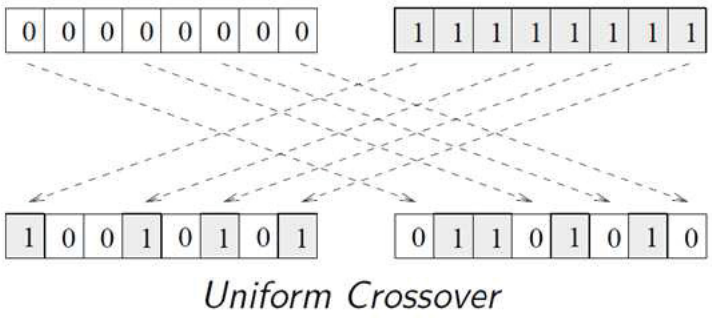


Abb. 4.3: Uniform Crossover

4.3.2.3 Crossover Operatoren für Permutationen

Für viele klassische kombinatorische Optimierungsprobleme wie das TSP oder auch das PFSP eignen sich auch spezielle crossover Operatoren, die Permutationen als Ergebnis des Crossovers hervorbringen. Dadurch spart man sich die anschließende Gültigkeitsüberprüfung, die z.B. bei der Anwendung von 1-point crossover für das TSP anfallen würde.

Generell gibt es die Möglichkeit des reciprocal change, wobei 2 Gene vertauscht werden. Bei Insertion wird ein Gen ausgewählt, an einen anderen Platz verschoben, während der Bereich dazwischen verschoben wird. Im Rahmen von Inversion wird ein zufällig gewählter Bereich gespiegelt, bei Displacement hingegen ein Teilstring ausgewählt und an einer anderen Position wieder eingefügt [3].

4.3.2.4 Partially Matched Crossover (PMX)

Zuerst wird der crossover Bereich durch Zufallsauswahl zweier Genpositionen bestimmt. Die Gene im crossover Bereich kommen vom Elternteil B zum offspring A', Analoges gilt für A und B'. Die Gene, die dadurch in A' in Bezug auf A "überschrieben" werden, werden so verschoben, dass die nun doppelt vorhandenen Allele im Chromosom überschrieben werden, siehe auch Abb. 4.4. (Quelle: [3]) dazu.

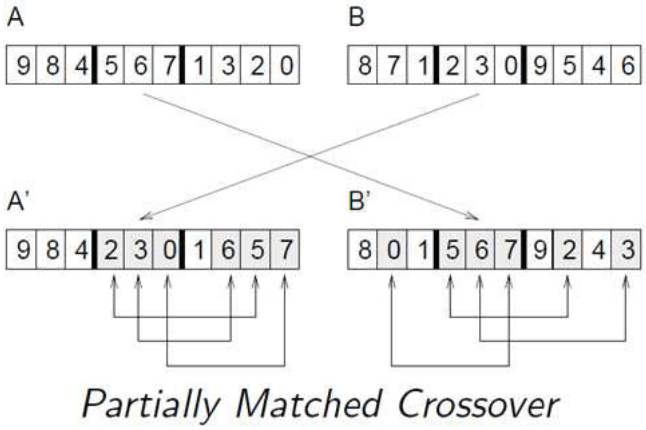


Abb. 4.4: Partially Matched Crossover (PMX)

4.3.2.5 Order Crossover (OX)

Order Crossover wird anhand eines Beispiels aus [3] erklärt:

1. Zuerst wird der crossover Bereich zufällig bestimmt.

A = 9 8 4 | 5 6 7 | 1 3 2 0

B = 8 7 1 | 2 3 0 | 9 5 4 6

2. Dann werden die Gene in diesem Bereich von A in B gelöscht und vice versa. Dadurch werden doppelte Allele in Genen vermieden.

A' = 9 8 4 | 5 6 7 | 1 _ _ _

B' = 8 _ 1 | 2 3 0 | 9 _ 4 _

3. Die Gene werden so umgeordnet, dass im ersten Chromosomenteil die Gene des eigenen crossover Bereichs (vorverschoben) stehen, während die nun freien Plätze für die Gene aus dem anderen crossover Bereich reserviert sind. Alle anderen Gene werden hinter dem crossover Bereich angesiedelt [3, 1].

A' = 5 6 7 | _ _ _ | 1 9 8 4

B' = 2 3 0 | _ _ _ | 9 4 8 1

4. Die freien Plätze werden mit Genen aus dem crossover Bereich des Partners aufgefüllt.

A' = 5 6 7 | 2 3 0 | 1 9 8 4

B' = 2 3 0 | 5 6 7 | 9 4 8 1

Während die relative Ordnung der Gene weitgehend verebt wird, bleiben absolute Genpositionen kaum erhalten. Die Schemata werden damit stark durchgemischt. Für das TSP liefert dieser crossover Operator i.A. gute Ergebnisse ([3]). Dies wird auch von Anwendern bestätigt, die diesen Operator getestet und erweitert haben[18].

4.3.2.6 Cycle Crossover (CX)

Unter Beibehaltung der aktuellen Genposition wird (ausgehend vom 1. Gen) ein ganzer Genzyklus vom Elternteil A übernommen, während alle frei bleibenden Plätze vom B Elternteil besetzt werden. Im Rahmen dieses A- Zyklus wird von Gen zu Gen darauf geachtet, dass das jeweils nächste Gen von A kommt, das in B an derselben Position in Bezug auf die vorherige Genposition ein doppeltes Allel in A' verursachen würde. Für ein TSP liefert diese crossover Methode i.d.R. eher weniger gute Ergebnisse [3].

4.3.2.7 Edge Recombination Crossover (ERX)

ERX ist v.a. für TSP sehr gut geeignet, da die Verwendung von Kantenlisten forciert wird. Diese Listen werden zuerst für alle Städte aus den Eltern erzeugt. Dabei wird zu jeder Stadt eine Liste der jeweils von ihr weggehenden Kanten zu den jeweils benachbarten Städten erstellt. Nach der Wahl einer Ausgangsstadt c wird diese aus allen Kantenlisten entfernt und die Stadt mit der kleinsten Kantenliste aus der Kantenliste von c gewählt. Die ausgewählte Stadt wird zu c und die Prozedur wird wiederholt, bis die Rundreise komplett ist [3].

Eine mögliche Verbesserung besteht darin, Kanten, die in beiden Elternteilen vorkommen, zu markieren und vorzuziehen.

Das Verfahren nach Grefenstette ([3]) liefert durch die Ausnutzung eines heuristischen Ansatzes noch bessere Ergebnisse.

4.3.2.8 Constraints in Bezug auf das konkrete Produktionsproblem

Abgesehen davon, dass wie bei einem TSP, bei dem keine Stadt i.d.R. doppelt in der Rundreise vorkommen darf, dürfen bei einem PFSP keine jobs doppelt vorkommen. Die weiteren Nebenbedingungen, siehe auch Abschnitt 2.3 dazu, schränken den gültigen Lösungsraum sehr stark ein.

Daher lässt sich allgemein festhalten, dass der crossover Operator für dieses PFSP nicht traditionell eingesetzt wird, da davon ausgegangen wird, dass ein sehr groß gewählter crossover Bereich die bisherige Lösung zu sehr zerstört. Stattdessen wird im Rahmen einer iterierten Mutation das Verhalten eines Crossovers insofern emuliert, als sich die Chromosomenstruktur des Individuums dabei relativ stark ändert, verglichen mit einer einfachen Mutation.

4.4 Sekundärer Operator: Mutation

Mutation dient v.a. dem Einbringen von neuem bzw. evtl. verlorengegangenem Genmaterial in die Population. Jedes Gen erhält mit einer relativ geringen Wahrscheinlichkeit (i.d.R. $\leqq 5\%$) einen neuen zufälligen Wert. Diese geringen Änderungen sind entfernt mit einer Variante der lokalen Suche vergleichbar [3]. Geringe Mutationsraten sind allgemein angeraten, sonst kann der GA schnell zu einer Zufallssuche degenerieren.

Swap Mutation (Exchange-Based Mutation): Hier wird der Inhalt der durch Zufall bestimmten Gene vertauscht, siehe auch Alg. 4.2 nach [10]. Es gilt:

n_{mut} entspricht der Anzahl an durchzuführenden Mutationen.

$S[j]$ stellt die Maschine i dar, die der Aufgabe j in der Reihenfolge S zugeordnet ist.

m entspricht der Anzahl Maschinen und n entspricht der Anzahl an Aufträgen.

Algorithmus 4.2 Swap Mutation

1: **for** $k = 1$ to n_{mut} **do**
2: $S[Random(1,n)] \rightleftharpoons S[Random(1,n)\}$; // swap mutation
3: **end for**

Flip Mutation: Die Auswahl des zu mutierenden Gens erfolgt zufallsgesteuert, daher wird diese Mutation auch Mutation durch Zufallsbelegung genannt. Dabei wird bei einem Bitstring das ausgewählte Gen negiert. Beim Vergleich zwischen Alg. 4.2 und Alg. 4.3 [10] fällt sofort der Unterschied in Zeile 2 auf.

Anmerkung: Auf Grund des "Tauschprinzips" bei der Swapmutation kann die erhaltene Lösung prinzipiell nicht ungültig werden, bei der Flipmutation kann die Lösung, wenn man z.B. einen TSP oder ein PFSP betrachtet, leicht ungültig werden. Daher wird für den GA im konkreten Produktionsproblem ausschließlich Swap Mutation verwendet.

Algorithmus 4.3 Flip Mutation

1: **for** $k = 1$ to n_{mut} **do**
2: $S[Random(1, n)] \leftarrow Random(1, m)$; // flip mutation
3: **end for**

4.5 Ersetzungsstrategien

4.5.1 Generationärer genetischer Algorithmus

Im Rahmen dieses klassischen Ansatzes wird innerhalb jeder Generation die gesamte Population durch eine neu selektierte, gekreuzte und mutierte Population komplett ersetzt. Dabei verläuft die Fitness des jeweils besten Individuums nicht monoton [10].

Goldberg und Deb zeigen, dass im Rahmen dieses Modells ein relativ hoher Selektionsdruck entsteht, der v.a. darauf zurückzuführen ist, dass die schlechtesten Individuen der jeweiligen Population durch bessere Individuen ersetzt werden.

4.5.2 Steady state genetischer Algorithmus

Im Rahmen des steady state GA Ansatzes wird in jeder Generation ein Individuum aus der aktuellen Population ersetzt. Häufig wird dabei das schlechteste Individuum der aktuellen Population ersetzt oder ein zufälliges Individuum, wobei sichergestellt wird, dass das neue Individuum eine bessere Fitness als das zu ersetzende Individuum bekommt. Dadurch wird ein relativ hoher Selektionsdruck erreicht, den man ggf. wieder reduzieren kann. Anstelle z.B. gezielt jeweils das schlechteste bzw. die schlechtesten Individuen zu entfernen und durch neue (bessere) Individuen zu ersetzen bietet es sich auch an, die ältesten Individuen, welche i.d.R. gut erforscht sind, zu ersetzen. Interessant bleibt in diesem Zusammenhang, dass der Unterschied zwischen der *"kill worst"* zur *"kill oldest"* Strategie relativ gering ausfällt, dennoch wird dadurch der Selektionsdruck etwas reduziert [8]. Eine weitere Alternative besteht darin, schlechte Individuen aus der Population vorzugsweise zu eliminieren, z.B. mittels *inverse ranking*.

Ein wichtiger Unterschied in der Beschaffenheit eines steady state Algorithmus besteht darin, dass ein neues Individuum (häufig) nur dann in die Population eingefügt werden darf, wenn es sich von allen anderen Individuen unterscheidet. Dadurch sollen Duplikate in der Population eliminiert werden. Dies hat Auswirkungen auf das Verhalten des Algorithmus, welche von Lance Chambers im Rahmen seiner Simulationsläufe nicht beleuchtet und damit nicht geklärt worden sind [8].

Durch steady state wird wenig Speicherplatz verbraucht. Die Laufzeit ist verglichen mit dem traditionellen Ansatz kürzer, weil in jeder Generation, in der nur ein Individuum durch ein besseres ersetzt wird, Selektion, Crossover und Mutation und anschließende Bewertung bzw. Vergleich nur auf 1 Individuum angewandt werden und die durchschnittliche Fitness der Gesamtpopulation damit in jeder Generation zunimmt. Dies lässt den Schluss zu, dass der steady state GA dazu tendiert, relativ rasch zu konvergieren.

Dadurch, dass die besseren Individuen durch die gewählte Ersetzungsstrategie geschützt werden, können höhere Crossover und v.a. Mutations Raten gewählt werden, welche dem GA einen vielfältigeren, den Suchraum erforschenden Charakter verleihen können [10]. Gleichzeitig stellt sich die Frage nach der Möglichkeit der Ausprägung des Charakters des Algorithmus auf diese Art und Weise, wenn nur einige wenige Individuen in einer Population im Rahmen einer Generation verändert werden.

4.6 Variationen genetischer Algorithmen

4.6.1 Hybrider genetischer Algorithmus (HGA)

Bei Vorhandensein von problemspezifischem Wissen empfielt sich oft die Verwendung sogenannter hybrider genetischer Algorithmen, wobei der GA auf Grund seiner globalen Sichtweise die "hills" findet, während der lokale Suchalgorithmus die "tops of the hills" findet. HGA werden auch als memetische Algorithmen bezeichnet.

Eine Möglichkeit besteht nun darin, eine Variante eines GA mit 2- Opt zu verknüpfen, wobei für die k besten Individuen eine lokale Suche durchgeführt wird. Der Parameter k kann dabei beispielsweise vom Algorithmus selbst gesteuert werden (z.B. in Abhängigkeit bestimmter Schranken), aber auch dem Benutzer überlassen werden. Im Rahmen des in dieser Arbeit implementierten hybriden genetischen Algorithmus wird dem Endbenutzer die Wahl überlassen, wie viele Individuen für die lokale Suche verwendet werden bzw. wie groß die lokale Suche für ein einzelnes Individuum sein soll (α Wert bei Insertion Search, siehe auch Unterunterabschnit 3.4.2.2.) .

Eine andere Möglichkeit besteht darin, lokale Suche in die Operatoren eines GA zu integrieren. Die lokale Suche wird dann in jeder Generation für jeden offspring durchgeführt. Diese Möglichkeit wird in dieser Arbeit nicht getestet, weil dies für große Kampagnen den Laufzeitaufwand massiv erhöhen würde. Außerdem bin ich der Meinung, dass der genetische Algorithmus eher global suchen sollte und daher die lokale Suche erst nach Beendigung des genetischen Algorithmus stattfinden sollte.

4.6.1.1 Ein hybrider genetischer Algorithmus von Zhang, Li und Wang

Der HGA von Zhang, Li und Wang beruht einerseits auf einer *weighted simple mining gene structure* (WSMGS), sowie weiters auf einer lokalen Suche, die nach der Erzeugung der offsprings diese lokal verbessert [13]. Der Algorithmus zielt auf die Minimierung der Gesamtdurchlaufzeit eines PFSSP ab. Für die Initialisierung der Population werden die Heuristiken von Reeves (1995) bzw. Liu und Reeves (2001) vorgeschlagen. Mittels einer *simple mining gene structure* wird ein künstliches Chromosom erzeugt, welches mittels eines speziellen crossover Operators (SJOX) 2 offsprings erzeugt, welche anschließend mittels lokaler Suche verbessert werden können.

Simple mining gene structure (SMGS): Die Idee beruht darauf, aus den besten Chromosomen einer Population ein neues Chromosom zu fabrizieren. Mittels des voting Prozesses werden von den besten Chromosomen die Anzahl jedes Auftrags i in einer $n \times n$ Dominanzmatrix M gezählt. M_{ij} bezeichnet die Anzahl von Aufträgen i, die an der (Chromosmen) Position j aufscheinen. Generell kann man somit mehr als ein künstliches Chromosom erzeugen.

Die SMGS Prozedur im Detail gemäß [13]:

Algorithmus 4.4 Simple mining gene structure (SMGS)

// voting
1: **for each** elite chromosom π in the population
2: **for each** position j in π
3: **if** (π_j is job i) $M_{ij}++$;
4: count $\leftarrow 1$;
5: **while** (count $\leqq n$)
6: find the element $M_{ij}*$ with the maximum value in the dominance matrix ;
// select the first one if there is more than one such element
7: add job i at position j in the artificial chromosome;
8: set all elements in row i and column j as -1;
9: count ++;
// obtain the artificial chromosome
10: reset the dominance matrix to 0;

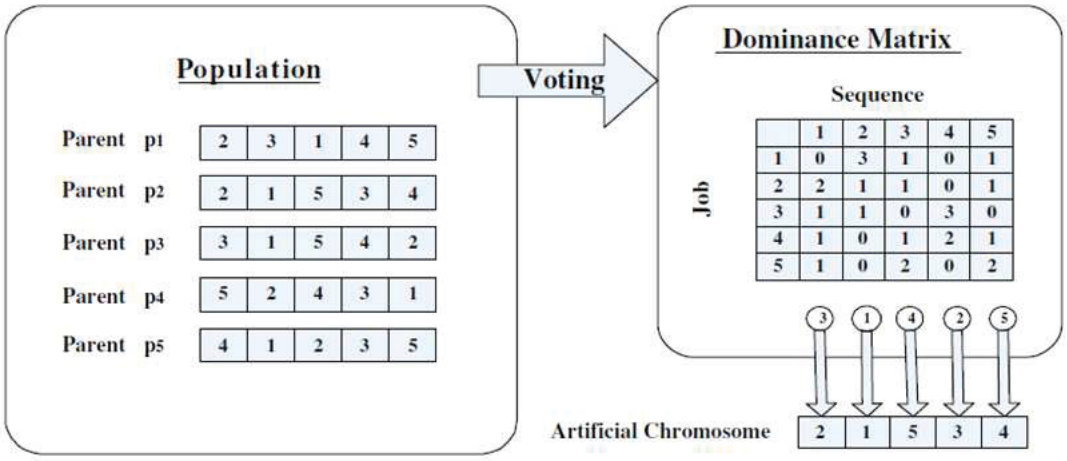


Abb. 4.5: Simple mining gene structure (SMGS)

Abb. 4.5 aus [13] zeigt die Erstellung eines künstlichen Chromosoms mittels SMGS. Auftrag 1 wird z.B. 0 Mal an Position 1, 3 Mal an Position 2, 1 Mal an Position 3, 0 Mal an Position 4 und 1 Mal an Position 5 gezählt. Die Zeile 1 in der Dominanzmatrix lautet damit {0,3,1,0,1}. Analog konstruiert man die anderen Zeilen der Matrix. Das künstliche Chromosom erhält man anschließend, indem man (Zeile 6) den jeweils größten Wert in der Dominanzmatrix sucht. In Abb. 4.5. ist der erstbeste gefundene größte Wert die 3 in Zeile 1 und Spalte 2. Das bedeutet, dass job 1 an Position 2 in das künstliche Chromosom eingefügt wird. Als nächstes wird der nächste 3er in Zeile 3 und Spalte 4 gefunden. Also wird job 3 an Position 4 eingefügt. Die Prozedur wird solange weitergeführt, bis man das vollständige künstliche Chromosom erstellt hat.

Um die Fitness der Chromosomen zu berücksichtigen, wird außerdem die sogenannte *weighted mining gene structure (WMGS)* vorgeschlagen, siehe Alg. 4.5. (Quelle: [13]) dafür. Im Rahmen von WMGS erhalten Individuen mit einer besseren Fitness eine höhere Gewichtung im Rahmen des Auswahlprozesses (*voting*).

Algorithmus 4.5 Weighted mining gene structure (WMGS)

Step 1: Generate an array of elite chromosomes in the current population by their non decreasing fitness order.
Step 2: Let Pπ_i be the position of chromosome π^{i} in the array.
Step 3: Calculate $\mathrm{M_{ij}} \leftarrow M_{\mathrm{ij}} + P_{\pi^i}$ for $i, j = 1, ..., n$.

Je besser das Chromosom ist, desto größer wird P_{π^i} und damit auch die Gewichtung. Für das Beispiel in Abb. 4.5 sei die Gesamtdurchlaufzeit der 5 Chromosomen $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5$ 15, 34, 46, 38 und 24. Nach der Durchführung von Schritt 1 beträgt die Reihenfolge $\{\pi_3, \pi_4, \pi_2, \pi_5, \pi_1\}$. Damit ergibt sich P_{π^i} zu {5,3,1,2,4} für $\pi_1, \pi_2, \pi_3, \pi_4, \pi_5$. Der 1er in Position 3 in P_{π^i} bedeutet z.B., dass das fitteste Individuum, nämlich Chromosom 3, auf Position 1 steht.

Schließlich ergibt sich die *gewichtete Dominanzmatrix zu*

$$\begin{matrix} 0 & 8 & 5 & 0 & 2 \\ 8 & 2 & 4 & 0 & 1 \\ 1 & 5 & 0 & 9 & 0 \\ 4 & 0 & 2 & 6 & 3 \\ 2 & 0 & 4 & 0 & 9 \end{matrix}$$

(Jobs werden zeilenweise, Positionen spaltenweise aufgetragen.)

Der Wert 8 für Zeile 1 in Spalte 2 (job 1 an Position 2) ergibt sich daraus, dass job 1 drei Mal an Position 2 vorkommt, nämlich in π_2, π_3, π_5. Addiert man die Werte aus P_{π^i} $(3 + 1 + 4)$, so ergibt sich

ein Wert von 8. Analog erhält man z.B. den Wert 9 für job 3 an Position 4. Zu π_2, π_4, π_5. erhält man nach der Addition von $3 + 2 + 4$ den Wert 9.

Wenn man nun die jeweils größten Werte betrachtet, erkennt man schnell, dass job 5 an Position 5 (Wert 9) im künstlichen Chromosom steht, während job 3 an Position 4 eingefügt wird (Wert 9) bzw. job 1 an Position 2 steht (Wert 8) u.s.w., bis das künstliche Chromosom erstellt worden ist.

Somit erhält man entsprechend der gewichteten Dominanzmatrix ein anderes künstliches Chromosom, nämlich $\{2, 1, 4, 3, 5\}$ [13].

SJOX crossover: Die Idee des SJOX crossover Operators beruht darauf, die Performance gewöhnlicher Operatoren in Bezug auf den Transfer guter Gene zu verbessern [13]. SJOX stellt dabei eine Erweiterung des 1 point crossover Operators dar und überträgt gute Gene direkt zu den beiden offsprings. Der von Zhang, Li und Wang vorgestellte Operator ist eine Erweiterung von SJOX. Die Idee basiert prinzipiell darauf, gleiche jobs von Vater und Mutter an Sohn und Tochter weiterzugeben. Die Prozedur im Detail wie folgt: (Quelle: [13])

Algorithmus 4.6 SJOX+' Crossover Operator

Step 1: Generate a random integer x ($1 \leqq x \leqq n$) as *crossover point.*
Step 2: Search *Common Jobs (CJ)* between father and mother (FM) as well as between the artificial chromosome and the father (AF) and between the artificial chromosome and the mother (AM): CJ(FM), CJ(AF), CJ(AM).
Step 3: Transfer CJ(FM) and CJ(AF) to the son chromosome with positions unchanged. Transfer CJ(FM) and CJ(AM) to the daughter chromosome with positions unchanged.
Step 4: The son inherits all the unassigned jobs before the "crossover point" from the father chromosome and the daughter from the mother.
Step 5: The other elements of the son after the "crossover point" are copied in the same order from his mother chromosome and the daughter´s from her father.

Der vorgestellte Operator erweitert SJOX um Informationen aus dem künstlichen Chromosom, welches durch WSMGS bzw. SMGS erzeugt werden kann. Abb. 4.6 (Quelle: [13]) verdeutlicht den Ablauf :

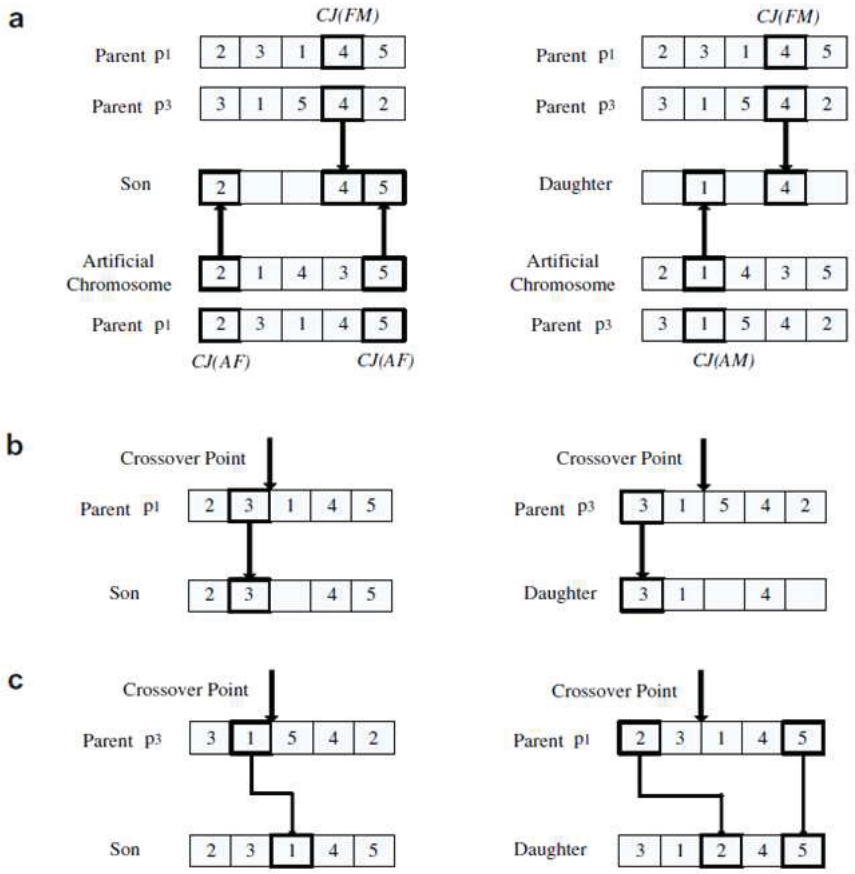


Abb. 4.6: Schritte 3 bis 5 des SJOX+' Operators

Zusätzlich können die erhaltenen offsprings mittels lokaler Suche optimiert werden. Die Anzahl der Elitechromosomen ist bei dem vorgestellten HGA von entscheidender Bedeutung. Einerseits müssen genügend Elitechromosomen ausgewählt werden, um nicht frühzeitig ähnlich lokaler Suche zu konvergieren, andererseits sollten nur die besten Individuen als Elite Chromosomen agieren [13].

4.6.2 Ein effizienter genetischer Algorithmus (ROX-MEE) von Amous, Loukil, Elaoud und Dhaenens mit innovativen Operatoren

Die Autoren stellen einen GA vor, dessen Effizienz auf einem "revised order Crossover" (ROX), einer Elite Selektionsmethode sowie einem "Mutation by Extended Elimination" beruht. Der vorgestellte GA liefert dabei gute Ergebnisse, sowohl bei einem TSP als auch bei Scheduling Problemen.

Elite Selektions Methode: Die vorgestellte Methode basiert darauf, die Population nach ihrer Fitness zu sortieren und danach die besten n Lösungen für den Paarungspool auszuwählen [18], wobei

$n = E(\sqrt{N})$

N stellt die Populationsgröße dar, E steht für den größten ganzzahligen Anteil und n steht für die Anzahl an Lösungen die sich reproduzieren dürfen.

Für eine unveränderte Populationsgröße müssen weitere n' zufällig ausgewählte Eltern in die nächste Generation hineinkopiert werden. n' ist die Differenz zwischen der fixen Populationsgröße N und der Anzahl resultierender Nachkommen, $n' = N - C \cdot n^2$. Siehe dazu auch Abschnitt 4.3.1.7 .

Die Verwendung von fitteren Eltern kann zu einseitiger, frühzeitiger Konvergenz zu einem lokalen Minimum führen. Dies soll durch den vorgestellten ROX Operator vermieden werden, der für die Diversität im GA zuständig ist.

Revised Order Crossover (ROX): Da sich die Verwendung des OX Operators für TSP als effizient herausgestellt hat - siehe auch 4.3.2.5 dazu - wird dieser im folgenden als Basis für die anhand eines Beispiels ([18]) vorgestellte Kreuzungsprozedur verwendet.

1. Der crossover Bereich wird - wie bei OX - zufällig ausgewählt.

Parent 1: 1 2 | 3 4 5 | 6 7 8

Parent 2: 6 7 | 4 2 8 | 5 3 1

2. Im Rahmen von ROX werden die Gene des ersten Elternteils in das Chromosom des 2. Nachkommen hineinkopiert und umgekehrt.

crossover Bereich von parent 2 →: Offspring 1: * * | 4 2 8 | * * *

crossover Bereich von parent 1 →: Offspring 2: * * | 3 4 5 | * * *

3. Die Vervollständigung der offsprings wird mit den verbleibenden Genen des Elternteils durchgeführt, wobei für offspring 1 mit parent 1 an Position 1 gestartet wird. Dabei werden die Gene von parent 1, die im crossover Bereich von offspring 1 bereits enthalten sind, im Rahmen der Vervollständigung ausgelassen. Damit ist sichergestellt, dass die Lösung eine Permutation darstellt.

Offspring 1: 1 3 4 2 8 5 6 7

Offspring 2: 6 7 3 4 5 2 8 1

Mutation by Extended Elimination (MEE): Da eine zufällige (Flip-) Mutation den GA in schlechte Suchregionen führen kann, kommt hier *"Mutation by Extended Elimination"* (MEE) zur Anwendung. MEE sucht die höchsten Kosten (bzw. die größte Distanz) zwischen 2 direkt benachbarten (= einander nachfolgenden) Städten im Chromosom. Sobald eine derartige Nachbarschaft gefunden wurde, wird das erste Gen mit seinem Vorgänger Gen ausgetauscht, während das zweite Gen mit seinem Nachfolger Gen ausgetauscht wird.

Offspring 1: 1 3 4 2⌣**8 5** ⌣6 7

→Offspring 2: 1 3 4 **8** 3 6 **5** 7

Das MEE Verfahren drängt den GA in lokal bessere Suchregionen, weil der Operator die größten Kosten eliminiert und damit die Konstruktion nicht wünschenswerter Individuen zu verhindern hilft [18].

Anmerkung: MEE eignet sich ausschließlich für TSP und ähnliche Probleme, nicht jedoch für PFSP, weil keine Analogie im PFSP zur Distanz zwischen 2 Städten existiert.

Zusammenfassung: Dieser GA ist durch *exploitation* und *exploration* charakterisiert. Der Selektionsoperator beruht auf exploitation auf Grund des derzeitigen Wissenstands, während die Crossover und Mutations Operatoren den Suchraum effizient durchforsten [18].

Der vorgestellte GA (ROX-MEE) erweist sich im direkten Vergleich mit ähnlichen Varianten als am effizientesten, auch in Anbetracht eines Scheduling Problems. Tab. 4.1 (Quelle: [18]) verdeutlicht die Effizienz von ROX-MEE gegenüber anderen Varianten.

Probleminstanz	Optimale Lösung	OX - OM	ROX - OM	OX - MEE	**ROX - MEE**
Bayg 29	1610	1610	1610	1610	**1610**
Bays 29	2020	2022	2025	2020	**2020**
Eil 51	426	485	502	463	**455**
Berlin 52	7542	8189	8157	7542	**7542**
Eil 76	538	584	571	550	**557**
Eil 101	629	665	672	667	**633**
Rat 195	2323	3415	3236	3064	**2812**

Tab. 4.1: Vergleich von ROX-MEE mit ähnlichen Varianten

4.6.3 Adaptiver genetischer Algorithmus (AGA)

AGA sind GA, welche ihre Parametrisierung in Bezug auf die Populationsgröße, Kreuzungsrate bzw. Mutationsrate, in Abhängigkeit der Konvergenz des GA ändern. Wenn sich die Population über Generationen hinweg nicht (mehr) verbessert, kann man einen dieser Parameter ändern, z.B. die Mutationsrate erhöhen und wieder senken, wenn eine bessere Lösung gefunden wurde [1]. Wichtig ist dabei die Ausprägung der crossover bzw. Mutations Parameter. Im Rahmen der Mutation sollte bei Anwendung dieser Strategie eine Art lokale Suche vollbracht werden, die Erhöhung des (zufälligen) swap Mutations Operators hat i.d.R. keine besonders positiven Auswirkungen auf eine Verbesserung der Lösung.

4.6.4 Genetischer Algorithmus mit variierender Populationsgröße (GAVaPS)

Michalewicz stellt einen GAVaPS vor, der das Konzept des Alters eines Chromosoms einführt, welches äquivalent zu der Anzahl Generationen ist, die es am Leben bleibt [4]. Dieses fitnessabhängige Alter ersetzt das Konzept der traditionellen Selektion und beeinflusst auf diese Art und Weise die Größe der Population in jedem Stadium des GA. Die vorgestellte Methode besitzt eine gewisse Ähnlichkeit zu Evolutionsstrategien. Der große Unterschied besteht in GAVaPS, dass hierbei die Größe der Population während den Generationen nicht konstant bleibt. Die Struktur des GAVaPS ist in Alg. 4.7 abgebildet. (Quelle: [4])

Algorithmus 4.7 GAVaPS Algorithmus

```
1: procedure GAVaPS
2: begin
3:     t = 0;
4:     initialize P(t);
5:     evaluate P(t);
6: while (not termination-condition) do
7: begin
8:     t = t + 1;
9:     increase the age of each individual by 1;
10:    recombine P(t);
11:    evaluate P(t);
12:    remove from P(t) all individuals with age greater than their lifetime;
13:    end
14: end
```

Im Rahmen der Fitnessbewertung erhält jedes Chromosom eine sogenannte *"Lebensdauer" (lifetime)* zugewiesen, welche von der konkreten Fitness des Individuums abhängt.

Die Größe der Population nach einer Generation beträgt: PopSize(t+1) = $PopSize(t)+AuxPopSize(t)-D(t)$, wobei $D(t)$ für die gestorbenen Individuen steht und $AuxPopSize(t) = [PopSize(t) \cdot \rho]$ beträgt. ρ steht für die Reproduktionsrate [4].

Um eine Art des sogenannten "Selektionsdrucks" einzuführen, wird eine aufwendigere Lebenszeitberechnung durchgeführt. Die Lebenszeitstrategien sollten Individuen mit überdurchschnittlicher Fitness unterstützen (→eine längere Lebensdauer zuweisen) und die Gesamtgröße der Population dennoch begrenzen.

Im folgenden Abschnitt werden einige Lebensdauer Berechnungsstrategien für das i-te Individuum in Bezug auf Maximierungsprobleme angeführt: ([4])

(1) proportional allocation:

$min(MinLT + \eta \cdot \frac{fitness[i]}{AvgFit}$, MaxLT)

(2) linear allocation:

$MinLT + 2{\cdot}\eta \cdot \frac{fitness[i]-AbsFitMin}{AbsFitMax-AbsFitMin}$

(3) bilinear allocation:

$$\begin{cases} MinLT + \eta \cdot \frac{fitness[i]}{AvgFit-MinFit} & if\ AvgFit \geqq fitness[i] \\ \frac{1}{2} \cdot (MinLT + MaxLT) + \eta \cdot \frac{fitness[i]-AvgFit}{MaxFit-AvgFit} & if\ AvgFit < fitness[i] \end{cases}$$

Legende: MaxLT, MinLT... maximale bzw. minimale erlaubte Lebenszeit

AvgFit, MaxFit, MinFit ... durchschnittliche bzw. maximale bzw. minimale Fitness

AbsFitMin, AbsFitMax ... bisher gefundene minmale bzw. maximale Fitness

Während die erste Idee (1) an fitnessproportionale Selektion erinnert, weil die Lebensdauer eines Individuums proportional zu seiner Fitness ist, wird die objektive Güte des Individuums außer Acht gelassen.

Die lineare Strategie (2) berechnet die Individuums Lebensdauer anhand der konkreten Fitness des $i-$ten Individuums in Bezug auf die beste bisher gefundene Fitness. Der resultierende Nachteil dieser Strategie entsteht, wenn viele Individuen eine relativ hohe Fitness besitzen, wodurch eine relativ große (weil langlebige) Population entsteht.

Die bilineare Strategie (3) ist ein Kompromiss aus den ersten beiden Strategien.

Der vorgestellte GAVaPS wurde an diversen Funktionen getestet und ermittelt in den meisten Testfällen etwas bessere Ergebnisse als der traditionelle GA von Goldberg. Die beste Performance liefert die lineare Strategie (2), welche auch mit den höchsten Kosten, repräsentiert durch die durchschnittliche Anzahl an Funktionsbewertungen, verbunden ist.

In Bezug auf die Kosten muss angemerkt werden, dass der traditionelle GA ein optimales Verhalten (beste Leistung bei minimalen Kosten) bei allen getesteten Funktionen aufweist, sofern die optimale Populationsgröße gewählt wurde, während der GAVaPS v.a. gegen Ende, wenn sich viele gute Individuen in der Population befinden, dazu neigt, die Population auf Kosten der Laufzeit zu erhöhen, wodurch viele Funktionsbewertungen notwendig sind (E), bis der jeweils beste Wert gefunden ist (V), siehe auch Abb. 4.6 ([4]) dazu. G1 - G4 stellen die 4 verschiedenen getesteten Funktionen dar.

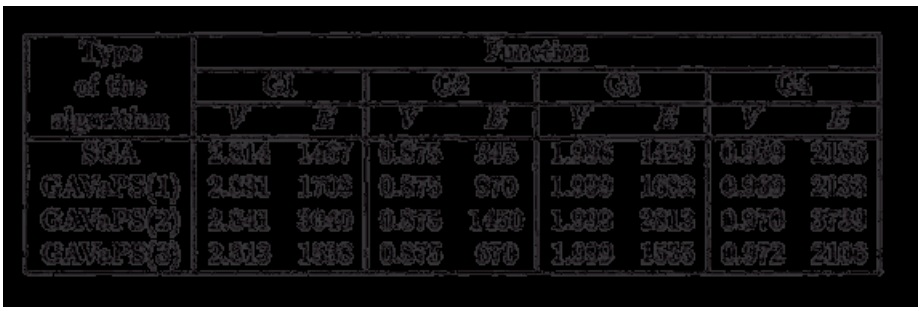


Abb. 4.7: Vergleich SGA - GAVaPS (1), (2), (3)

4.6.5 Serial Selection GA

Goldberg hat während seiner Forschungsarbeit die Eigenschaften von GA mit kleineren Populationsgrößen mit m Individuen untersucht, wobei der GA jedesmal neu initialisiert wird, sobald er konvergiert [4]. Dabei werden die n besten Individuen gespeichert ($n < m$) und mit in die neu initialisierte Population übernommen, wobei die restlichen $m - n$ Individuen zufällig initialisiert werden. Diese Strategie wird als Serial Selection bezeichnet und ist in Alg. 4.8 (Quelle: [4]) erläutert.

Algorithmus 4.8 Serial Selection GA

```
1: procedure Serial Selection
2: begin
3:     generate a (small) population with size = m;
4:     while (not termination condition) do
5:     begin
6:         apply GA; // see algorithm 4.1 for detailed information
7:         save the best solution (x);
8:         generate a new population by transferring he best n individuals of the converged population;
9:         generate the remaining m − n individuals randomly;
10:    end
11: end
```

Die ständige Reinitialisierung sorgt für die Individuenvielfalt in der Population, während durch die Mitnahme der besten n Individuen auch ein (durch n) bestimmter Selektionsdruck vorhanden ist [4].

4.7 Weitere Ansätze

4.7.1 Overlapping populations

Das Prinzip von overlapping populations stellt eine Art Mittelmaß zwischen dem traditionellen generationären und einem steady state GA dar. Dabei werden $G \cdot n$ Individuen neu erzeugt, während $G \cdot (n-1)$ zufällig gewählte Individuen von der Elternpopulation übernommen werden.

4.7.2 Crowding model

Mit Hilfe des "crowding model" Ansatzes wird versucht, eine in Bezug auf die Individuenvielfalt ausbalancierte Population zu erhalten. Dieses Problem, das ein einzelnes fittes Individuum möglicherweise eine ganze Population dominiert, wurde erstmals von DeJong beschrieben.

Zur Umsetzung wird die Population in CF (crowding factor) Gruppen eingeteilt. Für ein neu erzeugtes Individuum wird Platz geschaffen, indem eine Gruppe zufällig ausgewählt wird und das dem neuen Individuum ähnlichste Individuum gelöscht wird. Als Ähnlichkeitsmaß verwendet man i.d.R. Regel Hamming Distanzen.

Gute Werte für CF sind z.B. 2 oder 3.

Durch dieses Prinzip wird innerhalb einer Gruppe auf Artenvielfalt geachtet, es wird dem Superindividuumseffekt entgegengewirkt und ist besonders bei komplizierten Funktionen ein bewährter Ansatz [1, 10].

4.7.3 Elitismus und schwacher Elitismus

Anhand der Elitismusoption wird sichergestellt, dass das beste Individuum der aktuellen Population durch einen der Operatoren nicht verlorengeht und in die nächste Generation mitgenommen wird. Dort wird nach erfolgter Selektion, Crossover und Mutation überprüft, ob das beste Individuum der aktuellen Population fitter ist als das aus der vorangegangenen Population. In jedem Fall wird dabei immer das fittere der beiden Individuen in die nächste Generation mitübernommen und dort am Ende wieder mit dem jeweils besten Individuum aus der aktuellen Population verglichen [10].

Das Verfahren ist unter Verwendung dieser Option auch anfälliger für den Superindividuumseffekt und bleibt daher leichter bei lokalen Optima hängen. Wenn dieser Effekt eintritt, wird hauptsächlich nur mehr lokal im Bereich der besten Lösung(en) weitergesucht.

Eine weichere Variante besteht darin, das beste Individuum zu verlieren, wenn nach einer bestimmten Anzahl an Generationen (z.B.: 5) kein Fortschritt erreicht wurde [8].

Es gibt auch die Möglichkeit die μ besten Individuen in mutierter Form zu übernehmen. Diese Variante ist auch als schwacher Elitismus bekannt [16].

Daneben existieren zahlreiche weitere Varianten, z.B. die k besten Individuen zu übernehmen bzw. neben dem besten Individuum zusätzlich auch das schwächste Individuum einer Population zu übernehmen.

Generell liefert die Verwendung von Elitismus bessere Ergebnisse, u.a. weil die Fitnessverlaufsfunktion monoton ist. Wenn man mehr als ein Individuum in die nächste Population auf diese Art und Weise mitnimmt, sollte man dies in geeigneter proportionaler Form zur Gesamtpopulation machen, um frühzeitige Stagnation zu vermeiden und die Artenvielfalt beizubehalten, generell sollten maximal 20% Individuen gemessen am Anteil an der Gesamtpopulation auf diese Art und Weise mitgenommen werden.

4.7.4 Constraint handling mittels penalization

Wie auch bei dem konkreten Produktionsproblem müssen verschiedene Nebenbedingungen erfüllt sein, damit eine Lösung, welche aus einem Individuum erhalten wird, gültig ist. Im Falle des konkreten Produktionsproblems werden keine unzulässigen Lösungen im Rahmen der Operatoren generiert, dadurch muss auch das Bewertungsmodell nicht in Bezug auf Fitnessabzüge für derartige Verstöße angepasst werden. Da es jedoch nicht immer möglich ist, dass alle Individuen gültige Lösungen repräsentieren, müssen diese als Ausgleich für die Verletzung von z.B. einer weichen Nebenbedingung einen Fitness Abzug erhalten. Dabei sollte man zwischen leichten und schweren Verstößen auch in Hinsicht auf den absoluten bzw. relativen Abzug von der aktuellen Fitness unterscheiden.

Ein penalty hat i.A. folgende Form: $f'(i) = f(i) - \sum_{j=1}^{m} \gamma_\mathrm{j} \cdot \psi_\mathrm{j}(i)$ ([3])

wobei: γ_j...penalty coefficient ; $\psi_\mathrm{j}(i)$...penalty function

Die konkrete Wahl der Koeffizienten bzw. der Funktion ist dabei zu bedenken.

Oft empfielt sich die sogenannte *stepwise adaption of weights (SAW):* Dabei werden alle γ_j auf kleine Anfangswerte gesetzt. Nach einer konkret definierten Anzahl an Generationen erfolgt ein Update, im Rahmen dessen die γ_j für alle constraints erhöht werden, die in der besten Lösung nicht erfüllt waren. $\gamma_\mathrm{j} \leftarrow \gamma_\mathrm{j} + \triangle r$ ([3])

Δr entspricht einem Parameter, dessen Werte sowohl an den Penalty Koeffizienten angepasst sein müssen als auch von der Anzahl an Generationen abhängt, nach denen das Update erfolgt.

Den Nebenbedingungen werden dabei Gewichtsfaktoren zugeordnet, welche als Abschlag für verletzte Nebenbedingungen fungieren. Der Name stepwise adaption rührt von der Änderung der Gewichtsfaktoren nach einer bestimmten Anzahl Generationen. Solange die beste Lösung noch ungültig ist, werden die Gewichte weiter erhöht, bis die beste Lösung schrittweise gültig geworden ist oder durch eine bereits gültige Lösung verdrängt worden ist [10].

Das Prinzip des constraint handling beruht darauf, ungültige Individuen sukzessive aus der Population zu verdrängen.

Kapitel 5

5 Evolutionsstrategien

Die Idee der Evolutionsstrategien (ES) stammt von 2 Studenten (Hans-Paul Schwefel, Ingo Rechenberg) der Technischen Universität Berlin in den 1960ern. Bei nicht erfolgreichen Experimenten mit Koordinaten- und Gradientenstrategien an der Formbestimmung für ein gebogenes Rohr kam Rechenberg auf die Idee, zuällige Änderungen in der Parametrisierung vorzunehmen, welche dem Beispiel natürlicher Mutationen folgen [4, 3].

5.1 Unterschiede zu genetischen Algorithmen

Evolutionsstrategien unterscheiden sich von genetischen Algorithmen in zahlreichen Aspekten. Evolutionsstrategien eignen sich v.a. für numerische Optimierung, welche eine spezielle hill-climbing Prozedur mit selbstadaptierender Schrittweite σ verwenden.Während die Lösungen bei GA oftmals als binäre Vektoren repräsentiert werden, operieren ES auf Gleitkommadarstellungen [4].

Während, basierend auf der Grundidee nach Schwefel und Rechenberg, die Mutation der primäre Operator und die Rekombination der sekundäre Operator ist, verhält sich dies bei GA umgekehrt, wo Selektion und Rekombination die primären Operatoren darstellen und die Mutation den sekundären Operator repräsentiert.

Ein weiterer Unterschied besteht im Selektionsprozess. Während bei einem GA aus einer Population mit n Individuen diese allesamt für die neue Population ausgewählt werden, generieren bei einem ES-Verfahren μ Eltern λ Nachkommen, wobei dann im Rahmen der Selektion die Gesamtpopulation wieder auf μ Individuen reduziert wird, indem die am wenigsten fitten Individuen aus der Population entfernt werden. Die Selektionsprozedur ist also in diesem Sinne deterministisch, da die besten μ Individuen (ohne Wiederholung eines Individuums) aus $\mu + \lambda$ Individuen ausgewählt werden.

Bei GA ist die Selektionsprozedur zufallsgesteuert und die ausgewählten Individuen der neuen Population dürfen prinzipiell doppelt bzw. mehrfach vorkommen.

In ES ist ein Nachkomme das Resultat des crossovers von 2 Elternteilen sowie einer weiteren Mutation. Nachdem die temporär vorhandene, aus $\mu + \lambda$ Individuen bestehende Population, fertig ist, wird diese mittels Selektion auf μ Individuen reduziert. Bei GA werden zuerst mittels Selektion die Individuen bestimmt, die anschließend der Rekombination und (zu einem geringen Prozentsatz der) Mutation unterworfen werden.

Während die Reproduktionsparameter bei GA (crossover Rate, Mutationsrate,...) während der Evolution meistens konstant bleiben, können die Parameter beim ES laufend verändert und angepasst werden.

Während bei ES Individuen, die constraints verletzen, disqualifiziert werden, kommen bei GA Penalties zum Einatz, welche die Lösung nicht disqualifizieren, sondern im Rahmen eines definierten Fitnessabzugs verschlechtern.

5.2 Strategien

5.2.1 (1+1)- ES

Obwohl bei der (1+1) Evolutionsstrategie die Population aus einem Individuum besteht, welches der Mutation unterworfen wird, spricht man von einer "two-membered" Evolutionsstrategie, weil der Nachkommme mit seinem Elternteil um das Überleben kämpfen muss. Während dieser Zeit besteht die Population daher aus 2 Individuen. Dies entspricht einer probabilistischen Gradientensuche, das Verfahren ist in Alg. 5.1. aus [3] näher beschrieben .

Algorithmus 5.1 (1+1)- Evolutionsstrategie

1: **procedure** (1+1)-ES
2: **begin**
3: t←0;
4: initialize ($I(t)$);
5: evaluate ($I(t)$);
6: **while** (**not** termination condition) **do**
7: $I'(t)$ ←mutate ($I(t)$); // let the original parent individuum mutate to generate an offspring
8: evaluate ($I'(t)$); // evaluate the quality of the offspring
9: $I(t+1)$ ←select ($(I(t), I'(t))$; // select the better one of the two candidates
10: t←$t+1$;
11: **done**
12: **end**

Der Vektor $I(t)$ besteht i.d.R. aus n reellen Parametern, die Selektion folgt dem Prinzip *survival of the fittest*. Wie in der Natur sind bei dem Verfahren kleine Änderungen wahrscheinlicher als große.

$I'(t)$← $I(t)$ + $\vec{N}$ $(0, \sigma(t))$

$\vec{N}$... Vektor unabhängiger, normalverteilter Zufallsvariablen mit Mittelwert 0 und Standardabweichung $\sigma(t)$ ([3])

Bei einem zu groß gewählten Wert $\sigma(t)$ besteht die Gefahr einer zufälligen Suche, während ein kleiner Wert von $\sigma(t)$ nur ein lokales Optimum findet und zu langsam konvergiert.

Die 1/5 Erfolgsregel von Rechenberg sieht vor, dass der Anteil erfolgreicher Mutationen an allen Mutationen 1/5 betragen soll. Während diese Regel für einfache (unimodale) Funktionen oft zu sehr guten Lösungen führen kann, bleibt man bei anderen Funktionen oft an lokalen Optima oder größeren ebenen Gebieten hängen, bei unterschiedlich skalierten Parametern ist ein einziger $\sigma(t)$ Wert evtl. nicht ausreichend [4, 3].

5.2.2 (μ+λ)-, (μ,λ)- ES

(μ+λ)- ES Bei beiden Strategien handelt es sich um sogenannte *"multimembered evolution strategies"* [4]. Das generelle Ziel ist eine robustere Optimierung sowie die Möglichkeit der Selbstadaptierung der Strategieparameter. Das Prinzip hinter $(\mu + \lambda)$ besteht darin, dass μ Eltern λ Nachkommen erzeugen, wobei aus der temporären Population mit $\mu + \lambda$ Mitgliedern im Rahmen der Selektion die μ besten Individuen überleben [3, 4]. Die Strategie ist in Alg. 5.2 aus [3] näher erläutert.

Algorithmus 5.2 (μ+λ)- Evolutionsstrategie

1: procedure (μ+λ)- ES
2: **begin**
3: t← 0;
4: initialize ($P(t)$); // the population $P(t)$ consists of μ parents
5: evaluate($P(t)$);
6: **while** (not termination condition) **do**
7: $P'(t)$ ←choose λ individuals ($P(t)$); // the best λ parents are selected from the μ sized population for mutations; $\lambda < \mu$
8: $P''(t)$ ←mutate ($P'(t)$); // let $P'(t)$ mutate to generate λ offsprings
9: evaluate ($P''(t)$);
10: $P(t+1)$←select ($P(t), P''(t)$); // select the best μ candidates from $P(t)$ and $P''(t)$; perform downsize of the population
11: t←t + 1;
12: **done**
end

Die Generierung von $P'(t)$ erfolgt gleichverteilt, der Selektionsdruck kommt bei der Selektion aus $(P(t))\cup(P''(t))$ zum Ausdruck. Der monotone Bewertungsverlauf des besten Individuums kann zu frühzeitiger Konvergenz bzw. Stagnation der gefundenen Lösung führen.

(μ,λ)- ES Im Rahmen dieser Strategie generieren μ Individuen λ Nachkommen, wobei $\lambda > \mu$. Im Rahmen der Selektion wird eine neue Population bestehend aus μ Individuen ausschließlich aus den λ Nachkommen bestehend, generiert. Dabei ist die Lebensdauer jedes Individuums auf eine Generation begrenzt. Für das Verhältnis $\mu : \lambda$ empfiehlt sich generell $1 : 7$. Durch den nicht monotonen Bewertungsverlauf tritt frühzeitige Konvergenz seltener auf. Diese Strategie ist v.a. für sich zeitlich ändernde oder "noisy" Bewertungsfunktionen gut geeignet bzw. für die Selbstadaptierung von Strategieparametern.

In Bezug auf die Selbstadaptierung existiert kein spezieller Algorithmus für die Steuerung von $\sigma(t)$, stattdessen wird zu jedem Parameter x_i eine Standardabweichung σ_i mitgespeichert und mitoptimiert. Dazu muss auch die Mutation bzw. Rekombination entsprechend angepasst werden, nähere Informationen dazu sind in [3] ersichtlich.

Kapitel 6
6 Implementierung

6.1 Klassendiagramme und Beschreibung des Gesamtpakets

Die Optimierungsstrategien werden allesamt an den run thread des Optimization Handlers angekoppelt. Dieser steuert für die m Kampagnen die heuristische bzw. deterministische Optimierung. Die Klassendiagramme geben eine Übersicht über Methoden und deren Kopplung innerhalb der gesamten Programmstruktur. Die Klassendiagramme wurden automationsgestützt mit Enterprise Architecture erstellt. Um die Übersichtlichkeit beizubehalten wird auf eine ausführliche Erklärung des weiteren Zusammenhangs diverser Klassen innerhalb der gesamten Programmstruktur verzichtet

6.1.1 Klassendiagramm HGA

Die Klasse GA erbt von der abstrakten Klasse AbstractStrategy, die Klasse GA_Parameters erbt von der Klasse Parameters.

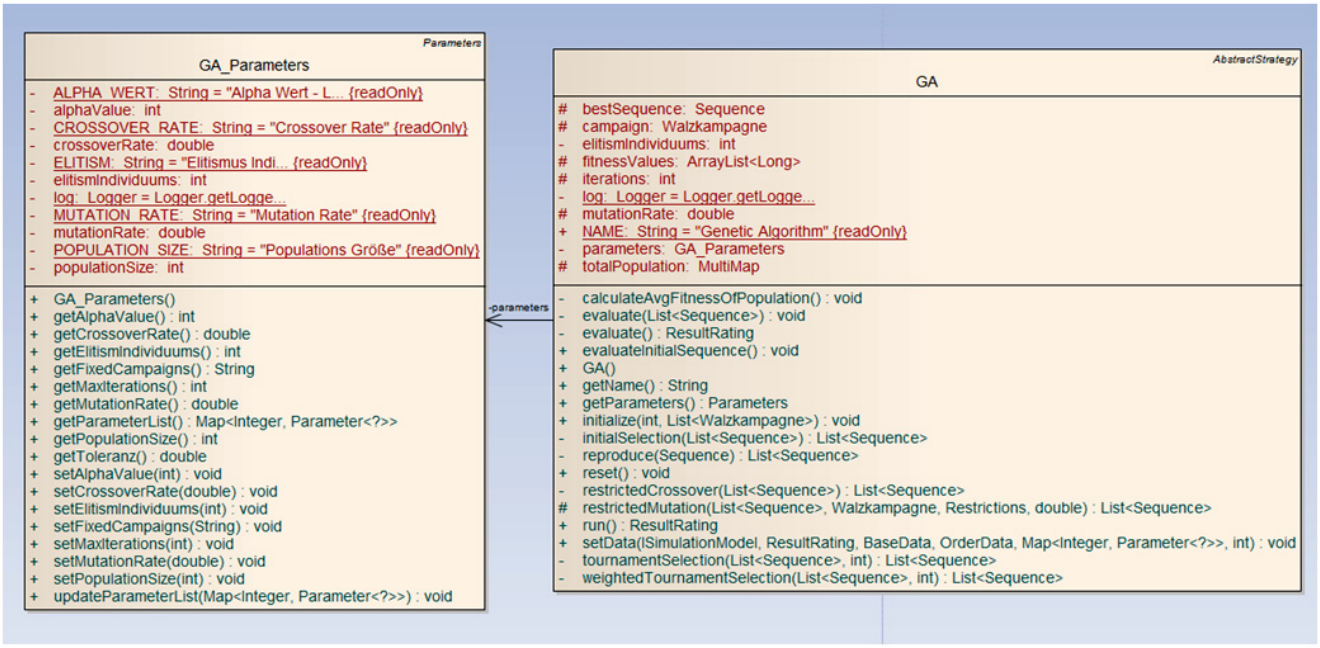


Abb. 6.1: Klassendiagramm HGA

6.1.1.1 Klassendiagramm EA

Die Klasse Evolutionary erbt wesentliche Methoden von der Klasse GA, die Klasse EvolutionaryParameters erbt von der Klasse Parameters.

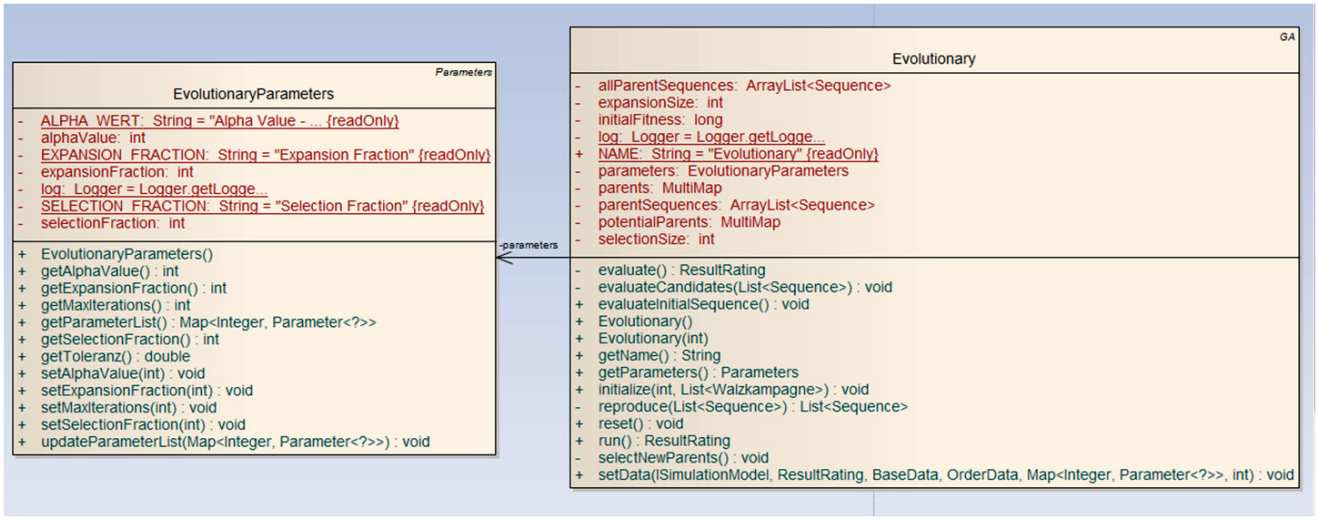


Abb. 6.2: Klassendiagramm EA

6.1.2 Klassendiagramm SA

Die Klasse SA erbt von der Klasse Abstract Strategy, die Klasse Parameters erweitert die Klasse Parameters.

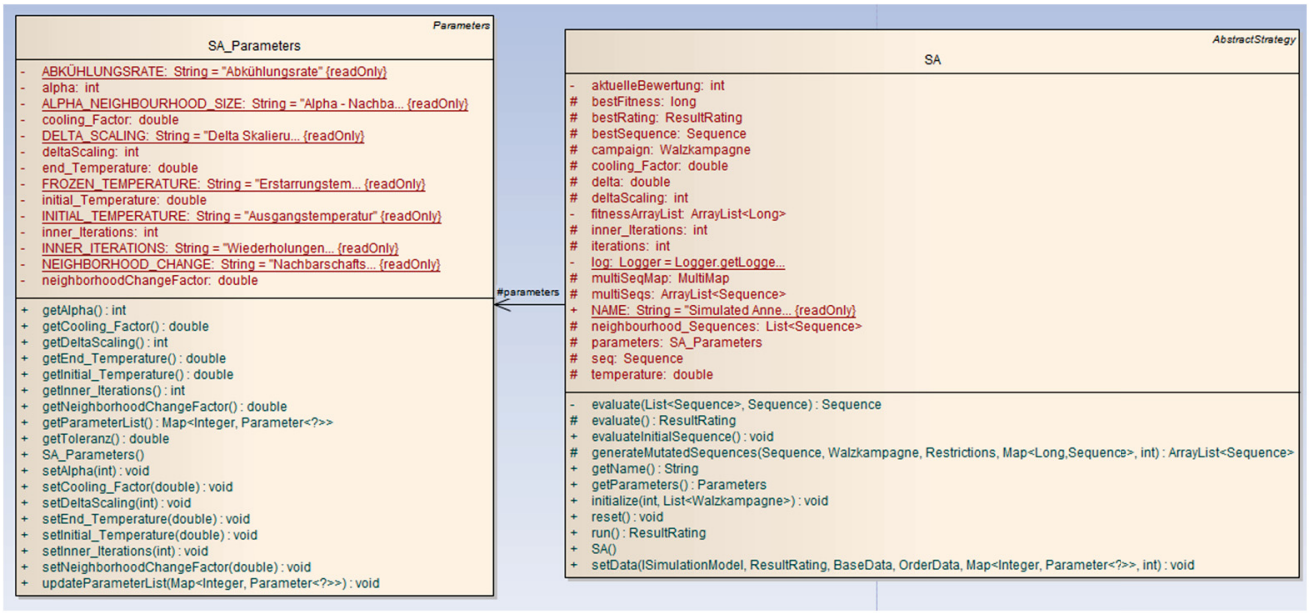


Abb. 6.3: Klassendiagramm SA

6.1.2.1 Klassendiagramm ILS

Die Klasse ILS erbt die wichtigsten Methoden mit Ausnahme der Perturbation von der Klasse SA, die ILS Parameter entsprechen den SA Parametern.

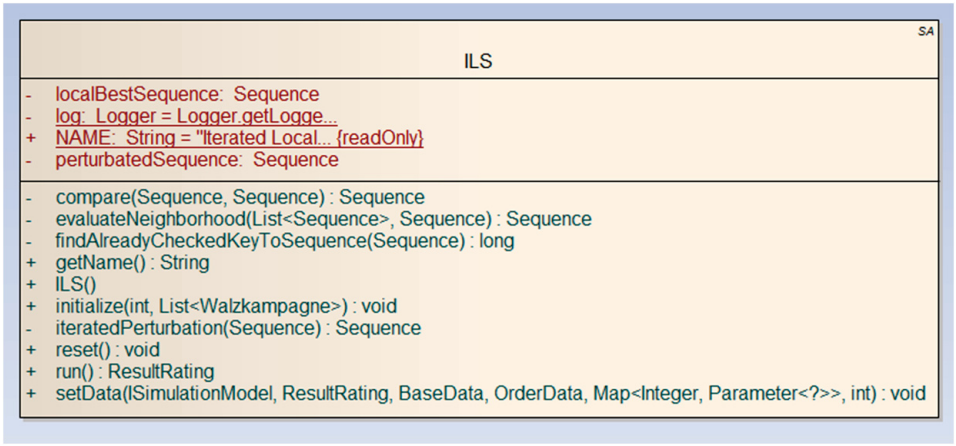


Abb. 6.4: Klassendiagramm ILS

6.1.3 Klassendiagramm ACO

Die Klasse ACOParameters erbt von der Klasse Parameters. Die Klasse ConstrainedACO erbt von der Klasse ACO.

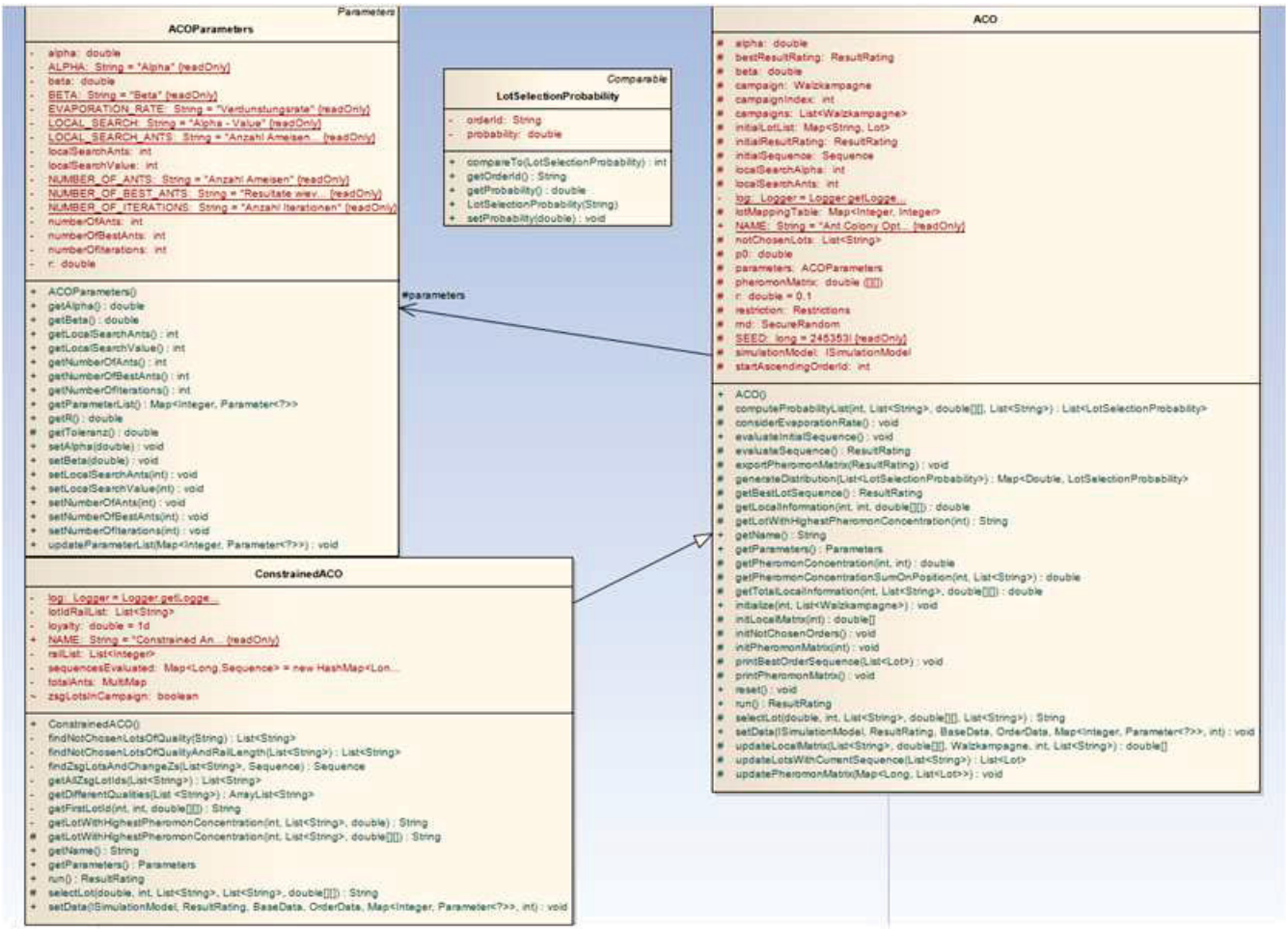


Abb. 6.5: Klassendiagramm ACO

6.2 Gemeinsamkeiten aller Optimierungsstrategien

- Alle Metaheuristiken schalten sich in den run thread ein, welcher eine metaheuristische Lösung für jede Kampagne zulässt, welche aus mindestens 4 Losen besteht. Kampagnen mit bis zu 3 Losen werden deterministisch optimiert.
- Die Nebenbedingungen gelten sowohl für metaheuristisch als auch für deterministisch zu optimierende Kampagnen.
- Jedes der Verfahren kann die in Abschnitt 2.3 vorgestellten weichen Nebenbedingungen jeweils (de-) aktivieren, während die vorgestellte harte Nebenbedingung (keine kampagnenübergreifende Optimierung) immer eingehalten werden muss. Im Rahmen der meisten Optimierungsläufe, welche im Rahmen dieser Masterarbeit durchgeführt werden, bleiben die Nebenbedingungen aktiviert, um das Planungsverhalten möglichst real simulieren zu können. Eine Ausnahme besteht in Läufen zur Erhebung des zusätzlichen Potenzials, welches sich durch das Deaktivieren der Nebenbedingung 'lang vor kurz' ergibt. Nur in diesen Testläufen ist diese Nebenbedingung deaktiviert, während die anderen aktiviert bleiben.
- Die Algorithmen verwenden eine Art *"Tabu Liste"*, wodurch im Rahmen einer HashMap < Long, ArrayList < Sequence >> redundant generierte Reihenfolgen nicht mehr bewertet werden. Diese HashMap wird aus der Checksumme als Schlüssel sowie der dazugehörigen Reihenfolge als Wert implementiert. Bei jeder generierten Lösung wird, nachdem für diese die Checksumme errechnet und gesetzt wurde, diese HashMap abgefragt, ob zu der erstellten Sequenz bereits ein Schlüssel existiert. Sofern dieser existiert, wird die generierte Lösung verworfen und nicht bewertet. Die Checksumme wird über den HashCode der Los ID mit dem zur Los ID gehörigen Reihenfolge innerhalb dieser Sequenz multipliziert. Der Fall zweier Sequenzen mit derselben Reihenfolge aber Zuweisung zu unterschiedlichen Sägebohrlinien wird über den Aggregat hash code mit der dazugehörigen Position abgefangen. Dazu wird im Falle von ZSG1 eine "1" zum String dazuaddiert, während der String "ZSG2" unverändert bleibt. Nähere Details dazu sind im Anhang ersichtlich.
- Lokale Nachbarschaften werden ausschließlich über das Insertion Search Verfahren erzeugt. Der Grund dafür liegt in der relativ einfachen Erzeugung zulässiger Lösungen sowie in der Tatsache, dass auf Grund der Nebenbedingungen weiter entfernte Nachbarschaften (3- Opt) generell kaum zulässig sind. Die Entscheidung für Insertion Search und gegen 2- Opt liegt darin, dass im Rahmen von Insertion Search kleinere (remove and insert) Austauschschritte stattfinden als die (swap Mutations-) Austauschschritte beim 2- Opt. Außerdem lässt sich die Intensität der Suche über α gut steuern.
- Globale Suche wird durch Mutationen sowie iterierte Mutationen gewährleistet. Größere Austauschschritte (z.B. für einen Crossover bzw. für einen Perturbationszug) sind i.A. nicht realisierbar in Bezug auf die Nebenbedingungen. Die Mutationen selbst können dabei als swap-exchange Austauschschritt oder auch als remove and insert Austauschschritt stattfinden.
- Im Rahmen der Optimierungs Parameter ist für alle Algorithmen prinzipiell die Anzahl der zu optimierenden Kampagnen einstellbar. Außerdem existiert ein (String-) Parameter ("Kampagnen fixieren") , über welchen spezifische Kampagnen beistrichgetrennt eingegeben werden können, die im Rahmen der Optimierung übersprungen werden sollen. Dadurch wird dem Benutzer die Möglichkeit gegeben, bestimmte im Rahmen seines Wissens selbst optimierte Kampagnen in der Simulation unverändert bleiben zu lassen.
- Für jeden Algorithmus lässt sich über den Parameter "Toleranz lang vor kurz (%)" die Toleranz einstellen. Dadurch kann der Endbenutzer selbst bestimmen, ob und in welchem Ausmaß er die Nebenbedingung 3, 'lang vor kurz', verletzen möchte. Geringere Toleranzen ($< 24\%$) erlauben nur den Tausch von etwas längeren Schienen nach vorne, während etwas kürzere nach hinten tauschen. Dies ist noch lange nicht gleichzusetzen mit einer Deaktivierung dieser Nebenbedingung, was z.B. zu Folge hat, dass ein Los mit 8 Meter Schienen vor einem Los mit 120 Meter Schienen gereiht werden kann. Dieser Fall läßt sich jedoch simulieren, indem man bei diesem Parameter z.B.

mindestens 1500% Toleranz einstellt, was einer Faktordifferenz von 15 entspricht. Der Parameter ist jedoch für die Einstellung geringerer Toleranzen gedacht.

- Da jede Kampagne für sich optimiert wird und die betrachtete Lösung mit den anderen zuvor optimierten Kampagnen sowie den nachfolgenden unveränderten Kampagnen in diesem Zusammenhang bewertet wird, ist der Optimierungsspielraum nicht so groß wie bei einem regulären TSP, wo z.B. der ganze Suchraum zusammenhängend optimiert werden kann. Dabei liefert jede Kampagne im Rahmen ihrer Optimierung ein bestes Kampagnenrating und setzt dieses fest für die zukünftig zu optimierenden Kampagnen. Das insgesamt global betrachtet beste Kampagnenrating wird schließlich als endgültige Lösung verwendet. Dabei scheint es klar zu sein, dass dieses Kampagnenrating erst in einer der letzten Kampagnen gefunden werden kann, weil es die Optima aller zuvor verwendeten Kampagnen als Basis hernimmt. Falls in einer Kampagne überhaupt kein Optimierungspotenzial besteht, wird die Ausgangsreihenfolge (dieser Kampagne) unverändert gelassen und die nächste Kampagne optimiert.
- Dem Benutzer kann neben der besten Reihenfolge außerdem die zweitbeste sowie drittbeste Reihenfolge angezeigt bekommen. Da die Fitnesswerte sehr nahe beisammen liegen können, kann in diesem Fall der Benutzer einfacher entscheiden, welche Reihenfolge er verwenden möchte.

6.3 Implementierung der Metaheuristiken

6.3.1 Hybrider genetischer Algorithmus: Tournament Selektion, Swap Mutation, Crossover, Insertion Search

Selektion: Tournament Selektion, Weighted Tournament Selektion

Crossover: Iterated Swap Mutation

Mutation: Swap or Remove and Insert Mutation

Lokale Suche für die besten Individuen: Insertion Search

Da die Initialisierung eine wesentliche Rolle im HGA spielt, wird **zur Verbesserung der Ausgangssituation für jede Kampagne einmal eine abgewandelte $(\mu + \lambda)$ Evolutionsstrategie angewandt**. Diese beruht darauf, von der erhaltenen Ausgangslösung ($\mu = 1$) eine bestimmte Anzahl (λ) an mutierten Nachkommen zu erzeugen um die vom User eingegebene Populationsgröße zu erhalten. Diese λ Nachkommen werden bewertet und alle $\lambda + \mu$ Nachkommen werden anschließend anhand ihrer Fitness sortiert. Von diesen $\lambda + \mu$ Individuen überleben anschließend so viele Individuen, wie der vom User eingestellte Parameter Populationsgröße vorgibt. Damit sichergestellt ist, dass $\lambda + \mu$ wesentlich größer als die Populationsgröße ist, werden intern für μ doppelt so viele zulässige Individuen erzeugt sofern dies möglich ist, wie der Parameter Populationsgröße angibt. Dieser interne Wert ist prinzipiell beliebig groß wählbar.

Dieser hybride genetische Algorithmus arbeitet die ersten 25% Iterationen mit Weighted Tournament Selektion, um mehr Vielfalt bereitstellen zu können. Danach werden die Individuen mittels Tournament Selektion ausgewählt, wobei $k = 2$ zwischen 25% und 50% Iterationen und $k = 3$ für die letzten 50% Iterationen genommen wird, sofern die gesamte Population aus mindestens 10 Individuen besteht. Bei Populationen mit weniger als 10 Individuen wird bis zum Ende des GA mit $k = 2$ selektiert. Der $k-$Wert beeinflusst den Selektionsdruck wesentlich. Um für mehr Artenvielfalt zu sorgen, wird für die ersten 25% Iterationen weighted tournament Selektion verwendet, wobei die Wahrscheinlichkeit, dass das bessere Individuum genommen wird, bei $p = 0.75$ liegt. Während anfangs die Individuenvielfalt durch den geringeren Selektionsdruck erhalten bleibt, konvergiert der HGA ab 25% Iterationen wesentlich stärker. Tournament Selektion erweist sich gegenüber fitnessproportionaler Selektion als wesentlich vorteilhafter, weil die Fitness Werte Zahlenwerte in der Höhe zwischen $0.5 \cdot 10^6$ (für die Simulation von wenigen Kampagnen) und $7 \cdot 10^6$ liegen und außerdem relativ knapp beieinander liegen (i.d.R. 0.001% bis maximal 0.5%). Dieser nicht signifikante Fitness Unterschied würde fitnessproportionale Selektion in zufallsgesteuerte Selektion ausarten lassen. Rangbasierte Selektion birgt widerum das Risiko einer

schlechten Skalierung. Im Rahmen der Selektion wird auch Elitismus verwendet, in folgender Form: Zu Beginn der Selektion wird die vom Benutzer einstellbare Anzahl an Elite Individuen selektiert (Anzahl Elite Individuen < Populationsgröße). Die verbleibende Differenz wird im Rahmen der Tournament Selektion ausgewählt.

Elitismus wird dabei in der Form verwendet, dass die besten Individuen in die nächste Population mitgerettet werden, auch wenn diese im Rahmen der Mutation/Crossover gar nicht mehr in der Population existent wären.

Die Rekombination wird als "Iterated Swap Mutation" realisiert, um das Problem einer ungültigen Lösung (jede Lösung, die keine Permutation darstellt bzw. die eingeschalteten Nebenbedingungen verletzt, wäre so eine Lösung) zu vermeiden. Das bedeutet, dass iteriert (z.B. 2 Mal) zwei Lose mit ihren Positionen unter Einhaltung der gegebenen Nebenbedingungen in der Reihenfolge tauschen. Der Crossover tauscht dabei prinzpiell zufallsgesteuert innerhalb der Kampagne, wobei bei jedem Tausch die Zulässigkeit sichergestellt sein muss. Mit einer Wahrscheinlichkeit von $p = 0.25$ findet auch eine 3. iterierte Mutation statt.

Im Rahmen der Swap/Remove and Insert Mutation entstehen wieder nur gültige Permutationen, womit das Problem von ungültigen Kandidatenlösungen im HGA nicht existent ist. Sofern eine Mutation stattfindet (Mutationsrate ist vom Endbenutzer einstellbar), findet mit $p_1 = 0.6$ ein remove and insert Austauschschritt statt, während mit $p_2 = 0.4$ ein Swap Mutation Austauschschritt stattfindet. Die Wahrscheinlichkeit wurde etwas zu Gunsten von Remove and Insert eingestellt, weil durch den Crossover ohnehin nur Swap Mutationen und damit größere Änderungen stattfinden können.

Sofern innerhalb einer akzeptablen Dauer keine gültigen Kandidatenlösungen (30 Versuche, prinzipiell einstellbar) gefunden werden, wird die Mutation bzw. der Crossover abgebrochen und die ursprüngliche Reihenfolge unverändert zurückgegeben.

Eine weitere mögliche Mutationsvariante besteht darin, die Lose, welche Verspätung aufweisen, gezielt auszutauschen und im Rahmen des Austauschs möglichst weit nach vorne zu versetzen. Nach einigen Versuchen wurde diese Variante eingestellt, weil innerhalb einer Kampagne die Lose maximal um 1 bis $1\frac{1}{2}$ Tage vorverschoben werden können und damit das Potenzial (der Minimierung von Verspätungen) relativ gering ist. Unter Einhaltung aller Nebenbedingungen ist ein derartiger Tausch außerdem kaum möglich.

Für das beste Individuum jeder zu optimierenden Walzkampagne wird Insertion Search angewendet, ein lokales Suchverfahren, wobei hier die chromosomale Strukture besser beibehalten wird als beim 2 Opt Algorithmus. Insertion Search eignet sich auch für TSP sehr gut als lokale Suche und liefert laut den Autoren von [17] bessere Ergebnisse als der paarweise Austausch. Der Endbenutzer kann wiederum einerseits einstellen, wie viele (beste) Individuen er für die lokale Suche verwenden möchte, andererseits mittels α, wie intensiv die Suche für jeweils ein Individuum ausfallen soll.

Der User kann als Optimierungsparameter die Anzahl zu optimierender Kampagnen einstellen, die Iterationen für jede Kampagne, die Populationsgröße, die Crossover Rate, die Mutations Rate, die Anzahl der Elite Individuen, die Anzahl an Individuen für lokale Suche sowie den $\alpha-$ Wert für die lokale Suche.

Algorithmus 6.1 Hybrider genetischer Algorithmus realisiert in Bezug auf das konkrete Produktionsproblem

1: begin
2: $t \leftarrow 0$;
3: **for** (each campaign) **do**
4: initialize $P(t)$ with 1 individuum;
5: $(\mu + \lambda)-$ES
6: **for** (each iteration) **do**
7: // receive the selected population $Q_s(t)$ from the mutated population
8: // of the last iteration $Q_m(t-1)$
9: $Q_s(t) \leftarrow$ tournament_select $Q_m(t-1)$;
10: // receive the recombined population $Q_r(t)$ from $Q_s(t)$
11: $Q_{\mathrm{r}}(t) \leftarrow$ recombine $Q_s(t)$;
12: // receive the mutated population $Q_m(t)$ from $Q_r(t)$
13: $Q_m(t) \leftarrow$ mutate $Q_r(t)$;
14: //evaluate the new population $Q_m(t)$
15: evaluate $Q_{\mathrm{m}}(t)$;
16: **end for**
17: //perform local search for the best m individuums
18: **for each** (individuum of m best individuums) **do**
19: Insertion Search (individuum);
20: **end for each**
21: **end for**
22: **end**

Es sei in diesem Zusammenhang darauf verwiesen, dass diese Variante des HGA generell sehr geringe crossover Raten (maximal 0.25) und hohe Mutationsraten ($0.85. - 0.95$) verwendet und dadurch in ihrem Verhalten einem evolutionären Algorithmus ähnelt. Der Unterschied zu einer evolutionären Strategie besteht darin, dass im Rahmen des GA eine konstante Populationsgröße aufrechterhalten wird, sowie eine für einen GA typische Selektion (weighted Tournament Selektion bzw. Tournament Selektion) verwendet, während im EA die Eltern eine bestimmte Anzahl Nachkommen erzeugen und von dieser temporären Population die besten Individuen überleben bzw. selektiert werden. Außerdem verwendet dieser GA nicht nur eine einfache Mutation als Veränderung, sondern im Zuge des (seltener stattfindenden) Crossovers mehrere Austauschschritte um ein Individuum zu verändern.

6.3.2 Simulated Annealing

Der implementierte und getestete Simulated Annealing Algorithmus basiert auf dem Standard Verfahren, wie in Alg. 3.13 beschrieben. Im Zuge jeder Temperatur Veränderung wird für das zuletzt bewertete Individuum eine neue Nachbarschaft generiert. Der Endbenutzer kann dabei in der Parameterliste einstellen, bei welcher Temperatur der Übergang von einer globalen, auf Mutation basierenden Nachbarschaft des Individuums, zu einer lokalen auf Insertion Search basierenden Nachbarschaft stattfindet.

Außerdem kann der Endbenutzer die für Simulated Annealing üblichen Parameter Ausgangstemperatur, Abkühlungsrate sowie die Anzahl der Wiederholungen pro Temperatur einstellen. Der Benutzer kann außerdem auch die Reichweite der lokalen Insertion basierenden Nachbarschaft mittels α einstellen sowie den Delta Skalierungsfaktor Wert einstellen.

Eine genaue Benutzerdokumentation befindet sich in Unterabschnit 6.3.6.

6.3.3 Iterierte lokale Suche: Simulated Annealing Meta Suche

Die implementierte Variante der iterierten lokalen Suche arbeitet auf Basis von Alg. 3.5. Dabei wird $s*$ über mutierte Reihenfolgen in Bezug auf die Ausgangssequenz erzeugt. Die perturbierte Lösung s' wird mittels einer doppelten Mutation erzeugt. Die perturbierte Lösung s' wird anschließend lokal optimiert mittels Insertion Search, wodurch s'' entsteht. Auf Grund von Laufzeitbeschränkungen kann i.d.R. nicht die gesamte Nachbarschaft untersucht werden, der Benutzer kann einstellen, wie viele lokal generierte Nachbarschaftslösungen bewertet werden. Ob $s*$ oder s'' angenommen wird, entscheidet das von SA bekannte Metropolis Akzeptanz Kriterium.

Auf Grund der Ähnlichkeit der Simulated Annealing Meta Suche zum Simulated Annealing Verfahren erbt ILS mehrere Methoden von SA, z.B. sind auch die vom Benutzer einstellbaren Parameter ident, wobei die Anzahl der inneren Wiederholungen bei ILS für die Anzahl der Bewertungen pro Nachbarschaft entspricht.

6.3.4 Evolutionärer Algorithmus

Die im Rahmen dieser Arbeit implementierte Variante eines evolutionären Algorithmus arbeitet auf der Basis einer abgewandelten (μ, λ) Strategie. Während in der Grundidee μ Eltern λ Nachkommen generieren, von denen dann die μ besten Individuen überleben, überleben in dieser Strategie alle Nachkommen, die gut genug bewertet worden sind, um als zukünftige Eltern agieren zu können. Die Idee hinter diesem Konzept beruht darauf, nicht nur streng monoton in die Richtung der besten Individuen weiterzusuchen, sondern ein breiteres Spektrum zu behandeln. Zukünftige Eltern können nur die Nachkommen werden, die zumindest eine bessere Fitness aufweisen als die Fitness der Ausgangsreihenfolge. Wie viele Eltern in jeder Iteration hinzukommen, hängt neben der konkreten Lösungsqualität auch von der Größe der Kampagne sowie der vom Benutzer eingestellten “selection fraction” ab, aus der sich die Selektionsgröße berechnen lässt.

Im Rahmen der Generierung von λ Nachkommen wird aus den Eltern zufällig ein Individuum ausgewählt und dieses mutiert. Die Mutation kann dabei sowohl in Abhängigkeit von einer 50% Wahrscheinlichkeit in Form einer swap Mutation stattfinden als auch mit 50% Wahrscheinlichkeit in Form einer Remove and Insert Mutation. Die Mutation selbst erfolgt immer, sofern sie auch möglich ist, außer es werden dadurch Nebenbedingungen verletzt.

An dieser Stelle muss festgehalten werden, dass sich diese Variante eines hybriden evolutionären Algorithmus signifikant vom hybriden genetischen Algorithmus in Bezug auf die Selektion der Individuen unterscheidet. Außerdem arbeit der evolutionäre Algorithmus ausschließlich über Mutationen, während ein Crossover nicht existiert.

Für die besten m Individuen findet am Ende jeder Kampagne ein lokaler Suchlauf statt, wobei der Benutzer die Intensität der lokalen, auf Insertion Search basierenden, Suche eingeben kann.

Algorithmus 6.2 Evolutionärer Algorithmus realisiert in Bezug auf das konkrete Produktionsproblem

```
1: begin
2: t ← 0 ;
3: for (each campaign) do
4:    μ= initial parent sequence;
5:    for (each iteration) do
6:       generate λ candidates from μ parents randomly;
7:       evaluate λ candidates;
8:       // if (sum of old parents and potential new parents)> μ_max: the best μ_max parents are selected
9:       expand parents by new parents from the best n candidates (n < λ);
10:        set new parents: μ = μ + n;      // μ ≤ μ_max ; μ_max is limited by the campaign lot size
11:   end for
12:    //perform local search for the best m individuums
13:    for each (individuum of m best individuums) do
14:       Insertion Search (individuum);
15:    end for each
16: end for
17: end
```

6.3.5 Ant Colony Optimization

Bei diesem Vertreter der ameisenbasierten Optimierung handelt es sich um ein hybrides Ant Colony System, siehe auch Alg. 3.11. Im Rahmen der Übergangsregel für diesen ACS (q, q_0) werden zwei Zufallszahlen errechnet. q_0 wird im Rahmen jeder Iteration als neue Zufallszahl berechnet und während dieser Iteration für alle Ameisen als Entscheidungsparameter herangezogen, ob sich diese für den ersten Fall $(q \leq q_0)$ bzw. für den zweiten Fall $(q > q_0)$ entscheiden. Die Auswahl in diesen beiden Fällen erfolgt wie im Paragraph "Übergangsregel für ACS" in Abschnitt 3.4.5.1 beschrieben. Nach Berücksichtigung der Nebenbedingungen wird bei jeder Ameise für jede Position eine Liste von zulässigen Losen zusammengestellt, unter denen die Ameise je nach Fall ein Los auswählen kann. Während in Fall 1 die Position mit dem höchsten Pheromonwert verwendet wird, wird im Rahmen des zweiten Falles unter Berücksichtigung der relativen Wahrscheinlichkeit eine Position mit der Wahrscheinlichkeit p_{nh} ausgewählt. Nach jeder Iteration findet ein globales Pheromon Update statt, sobald alle Ameisen ihre Pheromone gelegt haben. Die Entscheidung darüber, wie viele Ameisen im Rahmen dieses Pheromon Updates Pheromone legen dürfen, obliegt dabei dem Benutzer. Streng genommen wird bei einem ACS das Pheromon Update lediglich von einer Ameise durchgeführt, im Rahmen der Testläufe hat sich jedoch herausgestellt, dass mehr Pheromone legende Ameisen in Bezug auf die Vermeidung frühzeitiger Stagnation Sinn machen. Die lokale Update Regel wird in diesem ACS nicht verwendet. Die konkrete Anzahl an verwendeten Ameisen für das Pheromon Update ist dabei in Abhängigkeit der gesamten Anzahl an Ameisen zu treffen. Die konkrete Ausgestaltung der Pheromonmatrix orientiert sich an der für PFSP typischen zweiten Gestaltungsvariante, welche die Vorteilhaftigkeit von Auftrag n an der Position h berücksichtigt, siehe dazu auch Tab. 3.1 in Abschnitt 3.4.5.

Grundsätzlich lässt sich neben dem $\beta-$ Wert auch der $\alpha-$ Wert für dieses ACS einstellen, wobei empfohlen wird, den $\alpha-$ Wert auf 1.0 zu belassen, sowie den $\beta-$ Wert zwischen 0.1 und 0.2 zu gewichten.

Die rot markierten Matrixpositionen markieren die durch die Nebenbedingung 'lang vor kurz' fix vorgegebenen Stellen, sehr viele Positionen können auf Grund der Nebenbedingungen überhaupt nicht mit Pheromonen versehen werden und viele weitere Positionen werden auf Grund der schlechten Fitnessbewertung der dazugehörigen Ameise nicht zum Pheromon Update herangezogen. Während Abb. 6.6 eine typische initialisierte Pheromonmatrix zeigt, erkennt man nach 6 Iterationen in Abb. 6.7 eine deutliche Konvergenz, welche auf Grund der gelegten Pheromone entstanden ist.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	LosId/Pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	122673-53	3,37	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
3	122951-3	0,9	3,37	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
4	122918-1	0,9	0,9	1,81	0,9	0,9	1,81	1,81	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
5	122955-4	0,9	0,9	0,9	1,81	0,9	1,81	1,81	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
6	123000-1	0,9	0,9	0,9	0,9	1,81	0,9	0,9	0,9	2,63	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
7	122867-2	0,9	0,9	0,9	1,81	1,81	1,81	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
8	123027-1	0,9	0,9	1,81	1,81	0,9	0,9	1,81	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
9	122970-1	0,9	0,9	1,81	0,9	1,81	0,9	0,9	1,81	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
10	123028-1	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	1,81	1,81	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
11	122724-4	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	0,9	2,63	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
12	123016-1	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	3,37	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
13	123020-1	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	0,9	0,9	1,81	1,81	0,9	0,9	0,9	0,9
14	122856-3	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	2,63	0,9	0,9	0,9	0,9	0,9	0,9	0,9
15	121822-106	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	0,9	2,63	0,9	0,9	0,9	0,9	0,9	0,9
16	121822-105	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	1,81	1,81	0,9	0,9	0,9
17	121822-102	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	1,81	0,9	1,81	0,9	0,9	0,9	0,9
18	122971-1	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	1,81	0,9	1,81
19	122965-1	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	0,9	1,81	1,81	0,9	0,9
20	123011-2	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	2,63	0,9
21	122942-1	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9	1,81	2,63

Abb. 6.6: Pheromon Matrix 1. Iteration (12 Ameisen, 3 für das Pheromonupdate verwendet)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	LosId/Pos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	122673-53	7,33	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
3	122951-3	0,53	7,33	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
4	122918-1	0,53	0,53	1,07	0,53	0,53	2,46	6,69	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
5	122955-4	0,53	0,53	0,53	1,07	0,53	6,69	2,46	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
6	123000-1	0,53	0,53	0,53	0,53	1,07	0,53	0,53	0,53	6,68	2,55	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
7	122867-2	0,53	0,53	0,53	2,58	6,64	1,07	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
8	123027-1	0,53	0,53	1,07	6,64	2,14	0,53	1,07	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
9	122970-1	0,53	0,53	7,14	0,53	1,07	0,53	0,53	1,07	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
10	123028-1	0,53	0,53	0,53	0,53	0,53	0,53	0,53	2,97	2,94	5,73	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
11	122724-4	0,53	0,53	0,53	0,53	0,53	0,53	0,53	6,5	0,53	3,32	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
12	123016-1	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	7,33	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53
13	123020-1	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	1,62	0,53	0,53	1,62	6,41	1,99	0,53	0,53	0,53
14	122856-3	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	5,73	4,64	0,53	0,53	0,53	0,53	0,53	0,53	0,53
15	121822-106	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	4,06	0,53	6,11	0,53	0,53	0,53	0,53	0,53	0,53
16	121822-105	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	5,25	3,27	3,34	0,53	0,53	0,53
17	121822-102	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	5,73	4,06	0,53	1,07	1,13	0,53	0,53	0,53
18	122971-1	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	1,07	1,62	0,53	7,02
19	122965-1	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	4,65	0,53	5,49	1,07	0,53	0,53
20	123011-2	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	7,02	2,05	0,53
21	122942-1	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	0,53	7,02	2,05

Abb. 6.7: Pheromon Matrix 6. Iteration (12 Ameisen, 3 für das Pheromonupdate verwendet)

Da sich jede Ameise ihre Reihenfolge beginnend von 0 eingefügten Aufträgen neu zusammenstellen muss, wird zuerst eine Liste der Lose mit derselben Qualität erstellt. Diese Listen werden in der geforderten Reihenfolge Normal vor HSH vor Bainit vor Chrom abgearbeitet. Innerhalb dieser (maximal 4) Listen (in einer Kampagne) kommt die 'lang vor kurz' Nebenbedingung zur Anwendung. Die Listen derselben Qualität und derselben Schienenlänge werden über die Fallunterscheidung sowie die Pheromonkonzentrationen abgearbeitet. Falls sich nur ein Los in einer dieser 'lang vor kurz' Listen befindet, wird dieses Los sofort der Ameise hinzugefügt und die nächste (längensortierte) Liste erstellt. Die 'lang vor kurz' sortierten Listen berücksichtigen auch eine eventuell eingestellte Toleranz. Beim Übergang zu einer neuen Güteklasse geht diese Prozedur von vorne los.

Dieser ACS verwendet auch eine heuristische Information, welche besagt, dass die Lose abwechselnd auf Zielsäge 1 bzw. Zielsäge 2 bearbeitet werden sollen. Diese Information wird allerdings eher schwach gewichtet. Vor jedem Einfügen eines neuen Auftrags in die bestehende Reihenfolge wird daher geprüft, auf welcher Zielsäge das zuletzt eingefügte Los bearbeitet worden ist. Dementsprechend wird dann als nächstes Los eines mit Zielsäge 1 bzw. Zielsäge 2 bevorzugt bewertet. Die lokale Informationsmatrix muss im Rahmen des Einfügens jedes neuen Auftrags bei einer Ameise jedes Mal aktualisiert werden.

Am Ende der Optimierung jeder Kampagne findet eine lokale Suche (Insertion Search) für die besten m Ameisen statt, wobei m vom Endbenutzer einstellbar ist.

6.3.5.1 ACO: Unterschiede in der Implementation der Nebenbedingungen in Bezug auf die anderen Algorithmen

Während sich der ACO Vertreter streng an die Nebenbedingungen halten muss, weil er sich nicht an der Ausgangsreihenfolge orientiert, ist dies bei den Algorihmen in dieser Form nicht der Fall. Dadurch können sich die anderen Algorithmen Reihenfolgen erstellen, die genau dann nicht die Nebenbedingungen einhalten müssen, wenn sich die Ausgangsreihenfolge auch nicht an die Nebenbedingungen hält. Konkret wurde die Güteklasse Bedingung für die anderen Algorithmen in der Weise implementiert, dass Mutationen nur innerhalb derselben Güteklasse stattfinden. Wenn jedoch in der Ausgangsreihenfolge HSH Lose vor Normal Losen gereiht sind, ist dies in den optimierten Reihenfolgen dieser Algorithmen auch der Fall.

Die Idee, welche im Rahmen dieser Implementation dahintersteckt, besteht darin, das heuristische Wissen des Produktionsplaners zu verwerten. Dieser hat mit Sicherheit einen Grund, sich in bestimmten (sehr wenigen Fällen) nicht an die eigene Regel Normal vor HSH vor Bainit vor Chrom zu halten. Sehr selten passiert es auch, dass Lose derselben Güteklasse nicht durchgängig abgearbeitet werden, sondern z.B. einige Normal Lose, danach einige HSH Lose, welche wieder von einigen Normal Losen gefolgt werden.

Der ACO hält sich hingegen streng an alle 3 weichen Nebenbedingungen. Man muss in diesem Rahmen noch anmerken, dass die Verstöße der anderen Algorithmen nur sehr selten im Rahmen der Güteklassen Nebenbedingung sowie der 'lang vor kurz' Nebenbedingung auftreten. Die ZSG Nebenbedingung unterscheidet sich im Rahmen dieses ACO Vertreters nicht von den anderen Optimierungsalgorithmen.

Da jedoch die Ausgangsreihenfolge in jedem Fall (!) eine zulässige Lösung darstellt, stellen diese (sehr) seltenen Verstöße keine ungültigen Reihenfolgen dar, sondern eine Verwertung heuristisch vorhandener Informationen.

Die 'lang vor kurz' Nebenbedingung wurde für diesen ACO in der Art und Weise implementiert, dass Tauschvorgänge nur dann stattfinden, wenn das Los, welches nach vorne getauscht werden soll, eine inklusive dem durch die Toleranz aufaddierten Wert aus mindestens gleichlangen Schienen besteht. Der ACO arbeitet hingegen die Schienenlisten, welche innerhalb derselben Toleranz liegen in Bezug auf die längste Schiene der Liste, pheromongesteuert ab. Wenn in der Ausgangsreihenfolge einige Lose mit kürzerer Länge vor längeren Schienen bearbeitet werden, benötigt der ACO eine ausreichend große Toleranz, damit eine derartige Reihenfolge möglich wird. Die anderen Algorithmen benötigen in diesem Fall keine Toleranz.

Wichtig ist in diesem Zusammenhang, dass auch die anderen Algorihmen bei 0% Toleranz nicht selbst gegen die 'lang vor kurz' Nebenbedingung verletzen. Wenn jedoch die Ausgangsreihenfolge bereits an einer oder wenigen Positionen dagegen verstößt, kann diese Verletzung bis zur optimierten Reihenfolge beibehalten werden, sofern derartige Lose nicht weiter nach hinten gereiht werden. Der Grund liegt wieder darin, dass man das heuristische Wissen des Produktionsplaners verwerten will, sofern diese offensichtlich erlaubte Lösung eine gute Fitness darstellt.

Der ACO arbeitet die Schienenlisten hingegen immer streng ab, erst im Rahmen einer vom Benutzer eingegebenen Toleranzgrenze $> 0\%$ erfolgt eine Aufweichung dieser Nebenbedingung, von welcher der ACO daher am meisten profitiert.

6.3.6 Benutzerdokumentation

Der Endbenutzer kann über das User Interface einen der verfügbaren Algorithmen auswählen.

Jeder Optimierungsalgorithmus verfügt über diese folgenden beiden einstellbaren Parameter:

- Kampagnen fixieren: Dadurch können bestimmte Kampagnen beistrichgetrennt im Rahmen des gesamten Optimierungsdurchlaufs übersprungen werden
- Toleranz lang vor kurz (%): Mit diesem Parameter kann der Benutzer entscheiden, in welchem Ausmaß bzw. ob er gegen die 'lang vor kurz' Nebenbedingung verstoßen möchte. Kleinere Toleranzwerte ($< 25\%$) betreffen v.a. kürzere Schienen, während lange Schienen (120 - 60 Meter) erst bei größeren Toleranzen davon betroffen sind. Dieser Parameter spielt für die gesamte Optimierung eine außerordentlich wichtige Rolle.

Die Algorithmen 'Genetic Algorithm', 'Constrained Ant Colony Optimization' sowie 'Evolutionary' verfügen über eine an die jeweilige Optimierung anschließende lokale Nachbarschaftssuche, welche vom Benutzer über die Parameter α sowie Anzahl Individuen für lokale Suche parametrisiert werden kann. Im Falle von 0 Individuen für die lokale Suche bzw. einem $\alpha-$Wert von 0 wird die lokale Suche übersprungen. Der konkrete Wert sollte in Abhängigkeit des Umfangs der Optimierung mit Bedacht gewählt werden.

Die Algorithmen 'Simulated Annealing' und 'Iterierte lokale Suche' benötigen keine abschließende lokale Suche, da diese Verfahren in den niedrigeren Temperaturzuständen bedingt durch den Nachbarschaftsänderungsfaktor nur mehr lokale Nachbarschaften erzeugen und dadurch eine abschließende lokale Suche keinen Sinn ergeben würde.

Hybrider genetischer Algorithmus - Parameter Beschreibung: In Abb. 6.8 sind die durch den Benutzer einstellbaren Parameter für diesen hybriden genetischen Algorithmus ersichtlich.

Max. Iterations: Dieser Parameter repräsentiert die Anzahl an Iterationen, im Rahmen derer sich die Population weiterentwickeln und verändern kann.

Populationsgröße: Über diesen Parameter stellt man die Anzahl an Individuen ein, die sich über die im obigen Parameter definierte Anzahl an Iterationen weiterentwickeln kann.

Crossover Rate: Über diesen Parameter, der im Intervall $[0, 1]$ liegt, wird der Anteil an Individuen ausgewählt, welcher im Rahmen einer Rekombination verändert wird.

Mutation Rate: Dieser Parameter, ebenfalls im Intervall $[0, 1]$, steuert den Anteil an Individuen, welche durch Mutation verändert werden.

Alpha Wert - Lokale Suche: Dieser Parameter steuert die Intensität der lokalen Suche, siehe auch 3.4.2.2.

Anzahl Individuen für lokale Suche: Über diesen Parameter wird eingestellt, wie viele der (besten) Individuen für die lokale Suche ausgewählt werden.

Anzahl an Bewertungen *je Kampagne* in Abhängigkeit der Parameter: $(Max.Iterations)\cdot(Populationsgröße)+ 2 \cdot n \cdot \alpha + Populationsgröße$

Elitismus Individuen: Über diesen Parameter kann der Endbenutzer die Anzahl der besten Individuen (*Elite Individuen*) bestimmen, die unabhängig von der eigentlichen Selektion vorab auf jeden Fall in die nächste Generation mitgenommen werden. Dieser Parameter sollte in Abhängigkeit von der gewählten Populationsgröße gewählt werden. Es wird empfohlen die besten 15 - 20 % an Individuen der Population mitzunehmen.

Dieser theoretische Wert an möglichen Bewertungen soll nur einen Überblick über die maximal mögliche Anzahl an Bewertungen geben. In der Praxis wird dieser Wert auf Grund diverser Nebenbedingungen sowie der Tatsache, dass keine Sequenz doppelt bewertet wird, nicht annähernd erreicht.

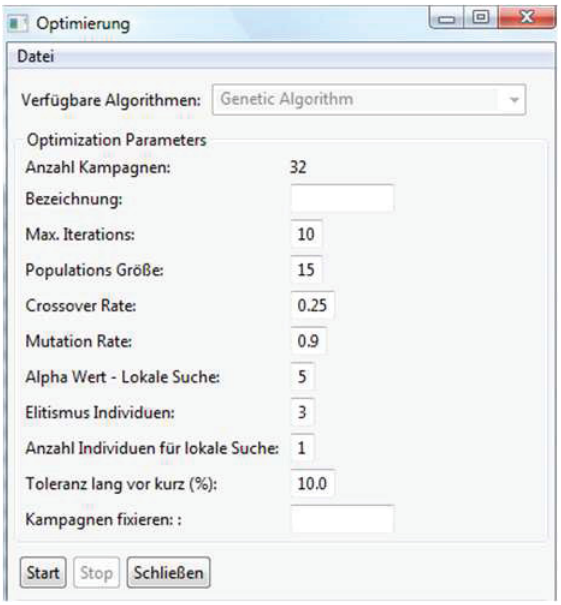


Abb. 6.8: Benutzereinstellungen: Parameter des hybriden genetischen Algorithmus

Empfohlene Parametrisierungen dieses HGA: Die Anzahl an Iterationen sowie Individuen ist prinzipiell frei wählbar. Jedoch sollten diese beiden Parameter in einem ausgewogenen Verhältnis zueinander stehen. Dabei der Benutzer generell zwischen mehr Evolution (durch mehr Iterationen) oder mehr Streuung (durch mehr Individuen) wählen oder für diese beiden Parameter denselben Wert verwenden. Für die crossover Rate wird ein Wert zwischen $0,1$ und $0,25$ empfohlen, für die Mutationsraten wird ein Wert zwischen $0,8$ und $0,99$ angeraten. Die Anzahl an Elitismus Individuen sollte im Verhältnis zur Gesamtpopulation $10-20\%$ betragen. Der Anteil an Elitismus Individuen steuert maßgeblich den Selektionsdruck mit. Der Umfang der lokalen Suche ist von der Intensität des vorangegangenen genetischen Algorithmus abhängig, empfohlen wird eine lokale Suche zumindest für das beste Individuum in einem Umfang von $\alpha = 3$.

Simulated Annealing/ Iterierte lokale Suche (Simulated Annealing Meta Suche) - Parameterbeschreibung: In Abb. 6.9 sind die vom Benutzer einstellbaren Parameter für Simulated Annealing abgebildet.

Ausgangstemperatur: Die Höhe der ausgewählten Starttemperatur entscheidet darüber, ob anfangs auch relativ schlechte Lösungen bei höherer Temperatur zu Gunsten globaler Suche akzeptiert werden oder ob mittels einer geringeren Temperatur abgesehen von besseren Lösungen ausschließlich wenig schlechtere Lösungen akzeptiert werden.

Erstarrungstemperatur: Dieser Parameter stellt einerseits das Abbruchkriterium dar, andererseits ist er für den Umfang der Optimierung mitverantwortlich.

Abkühlungsrate: Im Rahmen der Temperaturabsenkung wird die aktuelle Temperatur mit diesem Parameter $\in [0,1]$ multipliziert um eine neue Temperatur zu erhalten.

Wiederholungen pro Temperaturzustand: Dieser Wert steht für die Anzahl an Wiederholungen, die im Rahmen einer Temperatur durchgeführt werden, bevor mit einer neuen Temperatur eine neue Nachbarschaft generiert wird.

Alpha Nachbarschaftsgröße: Mit diesem Parameter kann die Größe der auf Insertion basierenden lokalen Nachbarschaft bestimmt werden, siehe auch 3.4.2.2 .

Delta Skalierungs Faktor: Auf Grund der relativ großen Fitness Werte ist ein Skalierungs Faktor notwendig. Je höher dieser Faktor gewählt wird, umso schlechter darf die bewertete Lösung sein, um noch akzeptiert zu werden. Dieser Skalierungsfaktor entspricht, gemäß der ursprünglichen Definition der Boltzmann Wahrscheinlichkeit, einer Boltzmann Konstante im Nenner.

Nachbarschafts Änderungs Faktor: Dieser Faktor wird mit der Ausgangstemperatur multipliziert und gibt die Temperatur an, ab welcher für den SA anstelle von globalen mutierten Nachbarschaften lokale auf Insertion basierende Nachbarschaften generiert werden. Die Wahl dieses Parameters hat demnach Auswirkungen auf das Verhältnis bzw. die Intensität der globalen Suche zur lokalen Suche. Dieser Parameter ist für den ILS Algorithmus bedeutungslos, da dieser ausschließlich insertion basierte Nachbarschaften generiert.

Anmerkung: Die Anzahl der Bewertungen je Kampagne ergibt sich aus der Anzahl an Temperaturzuständen, welche aus Ausgangs- und Erstarrungstemperatur sowie der dazugehörigen Abkühlungsrate berechnet werden kann, multipliziert mit der Anzahl an Wiederholungen je Temperaturzustand. Von diesem theoretischen Maximalwert sind wieder alle mehrfach erzeugten Sequenzen, welche insgesamt nur einmal bewertet werden, abzuziehen. Des weiteren sind aus diesem Grund und wegen der Nebenbedingungen die Nachbarschaften gegen Ende der Optimierung einer einzelnen Kampagne häufig leer.

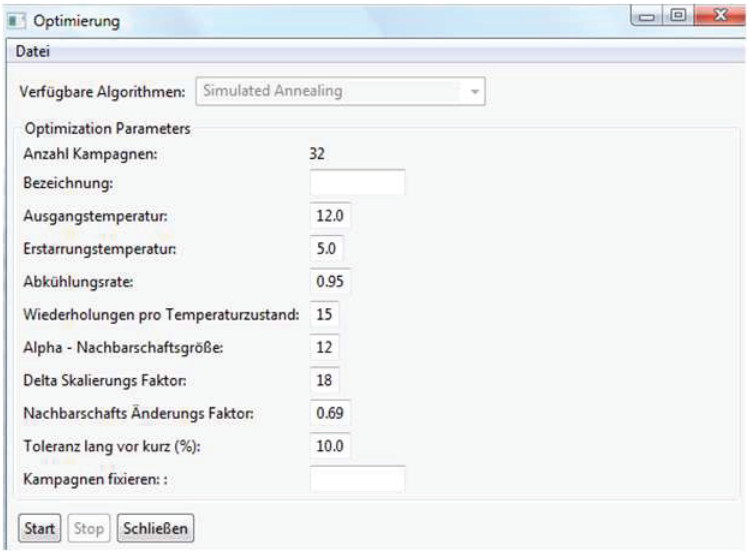


Abb. 6.9: Benutzereinstellungen: Simulated Annealing Parameter / Iterierte lokale Suche Parameter

Empfohlene Parametrisierungen dieses SA (ILS): Empfohlene Ausgangstemperaturen liegen im Bereich zwischen 10 und 15 °C, als Erstarrungstemperaturen werden Werte zwischen 3 und 8 °C empfohlen, damit der Temperaturunterschied "*spürbar*" wird. Außerdem steuert die Anzahl an Wiederholungen je Temperaturzustand die Intensität der Suche maßgeblich. Dazu werden mindestens 5 Wiederholungen und maximal 40 Wiederholungen empfohlen. Mit der Abkühlungsrate ist die Anzahl an Temperaturzuständen einstellbar. Ein wichtiger Parameter ist der Delta Skalierungsfaktor, mit dem eingestellt werden kann, wie weit in die negative Optimierungsrichtung zurückgesprungen werden kann. Während bei kurzen Testläufen Werte von 5 genommen werden sollten, empfehlen sich bei langen Simulationsläufen Werte bis maximal 40. Der Nachbarschaftsänmderungsfaktor sollte i.d.R. zwischen $0,75$ und $0,5$ liegen und hängt stark von den beiden gewählten Rahmentemperaturen ab. Es sollte dabei auf ein ausgewogenes Verhältnis zwischen swap Mutationen und remove and insert Mutationen geachtet werden.

Ant Colony Optimization - Parameterbeschreibung : Die einstellbaren Parameter dieses Ant Colony System Vertreters sind in Abb. 6.10 ersichtlich:

Anzahl Iterationen: Die Anzahl an Iterationen repräsentiert die Anzahl an Wiederholungen, die jeder Ameise im Rahmen ihrer pheromonbasierten Optimierung zur Verfügung stehen.

A

nzahl Ameisen: Die Anzahl (n) an verwendeten Ameisen für die Optimierung.

Resultate wie vieler Ameisen übernehmen: Mit diesem Parameter wird eingestellt, wie viele (m) Resultate der n zur Verfügung stehenden Ameisen für das globale Pheromon Update verwendet werden. Es werden in jedem Fall die Ergebnisse von $m \leq n$ besten Ameisen für das Pheromon Update herangezogen.

Verdunstungsrate: Dieser Parameter bestimmt die Geschwindigkeit, mit welcher die aufgetragenen Pheromone in der Pheromonmatrix verdunsten.

Alpha: Der α Parameter wird für die Übergangsregel im Rahmen eines ACS prinzipiell nicht variiert und wird daher auf 1.0 belassen, siehe auch Unterunterabschnitt 3.4.5.1 dazu. Prinzipiell ist jeder beliebige Wert $\in [0, 1]$ (v.a. zum Experimentieren) möglich. Ein geringerer α Wert bedeutet, dass die Pheromone im Rahmen des Übergangs weniger stark gewichtet werden. Der Endbenutzer kann damit selbst einstellen, wie wichtig ihm die Pheromone im Verhältnis zur heuristischen Information sind.

Beta: Dieser Parameter dient zum Einstellen der Gewichtung der heuristischen Information, eine empfohlene Gewichtung liegt zwischen $0,0$ (keine Berücksichtigung) und $1,5$ (relativ starke Berücksichtigung).

Alpha - Value: Dieser Parameter steuert die Größe der insertion basierenden Nachbarschaft. Diese Nachbarschaft wird für die beste(n) Ameise(n) erzeugt, sofern für Alpha- Value ein Wert > 0 eingegeben wird.

Anzahl Individuen für lokale Suche: Über diesen Parameter wird eingestellt, wie viele der (m besten) Individuen für die lokale Suche ausgewählt werden.

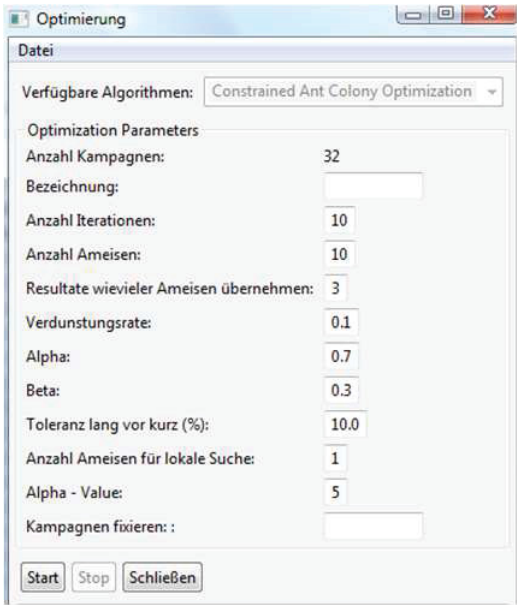


Abb. 6.10: Benutzereinstellungen: Ant Colony Optimization Parameter

Empfohlene Parametrisierungen dieses ACS: Prinzipiell sollte sich die Anzahl an Ameisen wie in [41] an der durchschnittlichen Kampagnengröße orientieren, daher werden Werte zwischen 10 und 20 empfohlen. Außerdem wird empfohlen, dass maximal 30% der Ameisen Pheromone ablegen, minimal jedoch 1 Ameise. Der alpha- Wert sollte in der Nähe von 1 gewählt werden, um den Einfluss der Pheromone geltend zu machen. Empfohlen werden Werte zwischen $0,9$ und 1. Dadurch, dass die heuristische Information nur ein Anhaltspunkt ist, der sich für einige Lose (ZSG Lose) im Rahmen der Optimierung auch negativ auswirken kann, wird ein sehr geringer Einfluss dieser Information angestrebt. Daher sollte für β maximal ein Wert in der Höhe von $0,5$ gewählt werden. Für die globale Pheromon Verdunstungsrate haben sich generell Werte zwischen $0,05$ und $0,1$ bewährt. Der Umfang der lokalen Suche ist wiederum vom Umfang des ACS abhängig.

Evolutionary - Parameterbeschreibung: In Abb. 6.11 sind die für den Benutzer einstellbaren Parameter für den evolutionären Algorithmus ersichtlich.

Max. Iterations: Dieser Parameter kennzeichnet die Anzahl an Wiederholungen innerhalb einer Kampagne. In jeder Wiederholung werden λ neue Nachkommen erzeugt.

Expansion Fraction: Die Anzahl der Kandidaten, die im Rahmen jeder Iteration generiert und bewertet werden, hängt einerseits von der Eingabe bei Expansion Fraction sowie von der Größe der Kampagne in Losen ab. Dabei wird die Losanzahl der Kampagne durch den Wert Expansion Fraction dividiert, wodurch eine flexible Anzahl von Kandidaten in Abhängigkeit der potentiellen Größe des Suchraums gewährleistet ist.

Selection Fraction: Die Anzahl der ausgewählten Individuen hängt von der Eingabe bei diesem Parameter sowie von der Kampagnengröße ab. Die Anzahl errechnet sich über die Division der Anzahl an Losen einer Kampagne durch den Wert Selection Fraction. Dadurch wird eine flexible Selektionsrate gewährleistet.

Alpha Value - Local Search: Dieser Parameter steuert die Intensität der lokalen Suche, siehe auch 3.4.2.2.

Anzahl Individuen für lokale Suche: Über diesen Parameter wird eingestellt, wie viele der (besten) Individuen für die lokale Suche ausgewählt werden.

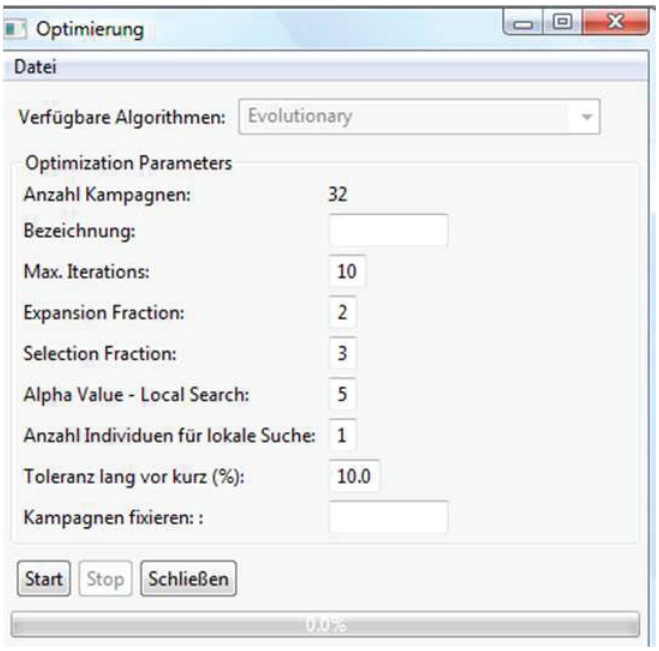


Abb. 6.11: Benutzereinstellungen: Parameter des evolutionären Algorithmus

Empfohlene Parametrisierungen dieses EA: Die Intensität des EA wird von der Anzahl an Iterationen sowie der Expansion Fraction gewährleistet. Über den Parameter Selection Fraction wird der Selektionsdruck im EA gesteuert. Der Umfang der anschließenden lokalen Suche ist wiederum von der Intensität des vorangegangenen EA abhängig.

Kapitel 7

7 Ergebnisse der metaheuristischen Optimierung

7.1 Feststellungen vor Beginn der Testläufe

7.1.1 Allgemeine Bedingungen betreffend den Ablauf der Optimierung

7.1.1.1 Simulationsablauf allgemein

Die Simulation läuft so ab, dass alle Kampagnen seriell in Bezug auf das Optimum der vorhergehenden Kampagnen optimiert werden. Anfangs wird Kampagne 1 optimiert, dieses Optimum wird für die Optimierung von Kampagne 2 als Basis herangezogen. D.h., dass die Optimierung von Kampagne 2 nicht auf der initialen Reihenfolge, sondern auf einer veränderten (= optimierten) Reihenfolge von Kampagne 1 basiert. Basierend auf der jeweils vorangegangenen optimierten Kampagne wird die jeweils nächste Kampagne optimiert. Die serielle Optimierung basiert auf einer Simulationsstudie, welche von der Universität Wien durchgeführt worden ist. Die Ergebnisse dieser Simulationsstudie besagen, dass das Optimum der jeweils vorangegangenen Kampagne keine (negativen) Einflüsse auf die Optimierung der nächsten Kampagne hat.

7.1.1.2 Simulationsdauer auf den verwendeten Testsystemen

Die benötigte Rechnerlaufzeit beträgt in Abhängigkeit des Umfangs des Datensatzes sowie der spezifischen Rechnerleistung (System 1, 2 oder 3) sowie der genauen Parametrisierung ausgehend von ca. 15 Minuten (ca. 5 Bewertungen je Kampagne) bis hin zu einigen Stunden für Testläufe mit 150 bzw. 300 Bewertungen je Kampagne. Die Simulationsdauer für die Läufe, bei denen die Nebenbedingung 3 deaktiviert wurde, beträgt im Rahmen einer intensiveren Parametrisierung von bis zu 700 Bewertungen je Kampagne bis zu 12 Stunden auf System 1. Der wichtigste Umrechnungsfaktor für die Simulationsdauer stellt die Anzahl an durchgeführten Bewertungen je Kampagne dar. Eine Bewertung dauert auf dem obengenannten aktuellen Dualcore Rechner ca. 5 - 6 Sekunden, auf einem älteren Core2Duo Prozessor benötigt eine Bewertung ca. 6 - 7 Sekunden, auf einem noch älterem Singlecore Rechner (System 3) benötigt eine Bewertung ca. 10 - 11 Sekunden. Diese Dauer ist mit der Anzahl an durchgeführten Bewertungen je Kampagne zu multiplizieren, das Resultat mit der Anzahl an Kampagnen. Dabei muss man bedenken, dass deterministische Kampagnen maximal 2 bzw. 6 Bewertungen benötigen sowie die Tatsache, dass Kampagnen mit einem Los nicht optimiert werden können. Daher ist die Simulationsdauer auch datensatzabhängig.

Um die erforderliche Anzahl an Testläufen durchführen zu können, wurden insgesamt 3 private Testsysteme verwendet.

System 1: (Standrechner, Assembling Oktober 2007) CPU: AMD Athlon 64 X2 5000+ (Brisbane) Black Edition: 2 · 2,6 Gigaherz @ 6400+: 2 · 3,2 Gigaherz; 2 GB DDR 2 Ram Dual Channel @ 800 Megaherz (CL 4-4-4-12)

System 2: (Laptop, Dezember 2007) CPU: Intel Core 2 Duo T5450: 2 · 1,66 Gigaherz (Merom); 2 GB DDR 2 Ram Dual Channel @ 667 Megaherz

System 3: (Standrechner, Assembling Juni 2004): CPU: AMD XP 2500+ (Barton): 1.833 Gigaherz @ 3200+: 2,2 Gigaherz; 1 GGB DDR 1 Ram Dual Channel @ 400 Megaherz

Während System 1 ca. 20−25% schneller rechnet als System 2, wächst der Unterschied von System 1 gegenüber System 3 auf ca. 100% an. Diese Erfahrung deckt sich sehr gut mit den Angaben des 6400+ Niveaus gegenüber dem 3200+ Niveau. Die übertakteten Systeme 1 und 3 wurden vorab mit Prime95 jeweils einem 12 Stunden Belastungstest (Small FFTs) unterstellt, um zu sehen, ob derart lange Beanspruchungen der CPU im übertakteten Zustand möglich sind. Während bei System 1 der freie CPU Multiplikator von 13 auf 16 angehoben wurde und der Referenztakt unverändert bei 200 Megaherz bleibt, wurde bei System 3 der Referenztakt von 166 auf 200 Megaherz angehoben. Der Ram wurde dabei in beiden Fällen nicht übertaktet.

Die Zusammenstellung eines aktuellen Systems mit Intel Core i7 CPU bzw. AMD Phenom II X4 CPU kommt aus den dadurch resultierenden Mehrkostenaufwänden für ein AM3 Motherboard im Falle einer AMD CPU bzw. eines Sockel 1156 bzw. Sockel 1366 im Falle einer Intel CPU, DDR 3 Ram und der CPU selbst nicht in Frage. Mit derartigen Systemen ist auf Grund der aktuelleren CPU Architektur gewiss ein deutlicher Leistungsgewinn (in Bezug auf die Dauer der Optimierung) messbar, allerdings werden im Rahmen der Simulation mit der java virtual machine wohl kaum alle 4 Kerne zur Gänze ausgelastet werden.

7.1.1.3 Einstellungen auf der java virtual machine

Die java virtual machine erhält während der gesamten Optimierung auf allen Algorithmen eine minimale heap Größe von 128 Megabyte zugewiesen, welcher bis zu einer maximalen Heapgröße von 1028 Megabyte anwachsen kann. Dieser riesige Wert, welcher mitteles JVisualM während der Optimierung betrachtet werden kann, wird nicht annähernd erreicht. Er wird v.a. auf Grund einer besseren Performance ausreichend hoch eingestellt. Trotz einiger Speicheroptimierungen (ResultRatings werden nicht zur Gänze gespeichert, sondern nur ihr Fitnesswert, am Ende der Optimierung einer Kampagne muss dafür die beste Reihenfolge noch einmal bewertet werden, um das komplette ResultRating zu erhalten) hat sich herausgestellt, dass im Falle längerer Optimierungsläufe die heap Größe auf bis zu 700 Megabyte anwachsen kann. Während System 1 und 2 bis zu 1,5 Gigabyte Ram für die maximale heap Größe bereitstellen, kann System 3 nur 768 Megabyte bezüglich seiner heap Größe bereitstellen. Auf Grund der verknappten heap Größe dauern v.a. die intensiveren Testläufe (z.B. bei Deaktiveren von Nebenbedingung 3) auf diesem System deutlich länger (ca. 20 Stunden) als bei System 1 und 2 (ca. 9 - 12 Stunden).

7.1.2 Optimierungswürdigkeit auf Grund eines feineren Splittings von ZSG Losen

ZSG Lose werden in der Simulation nicht als ein einzelnes Los dargestellt, sondern aufgeteilt, weil man so das Los besser auf die einzelnen Sägebohrlinien zuweisen kann. Im Rahmen eines Experiments wurde die Optimierungswürdigkeit eines großen bzw. konkret aus 160 Blöcken bestehendem Zielsägebohrlinie Los untersucht. Während das gesamte Los in der Ausgangssituation in Form von 16 Teillosen zu jeweils 10 Blöcken aufgteteilt ist, welche abwechselnd und damit im Verhältnis 50:50 auf die beiden Sägebohrlinien aufgeteilt werden, wurden 2 neue Datensätze generiert, welche dieses ZSG Los einmal in der Form von 8 Teillosen zu jeweils 20 Blöcken bzw. einmal in der Form von 32 Teillosen zu jeweils 5 Blöcken in der Ausgangssituation darstellt.

Während die Ausgangsfitness aller drei Datensätze fast ident sind und sich nur minimal unterscheiden, erwarten wir durch ein feineres Splitting ein besseres optimiertes Ergebnis. Dieses entsteht dadurch, weil im Rahmen eines feineren Splittings Verhältnisse und Lösungen generiert werden können, welche durch ein grobes Splitting nicht erreicht werden. So kann z.B. im Rahmen eines 32 Mal 5er Splittings ein ZSG Los in das Verhältnis 95 Blöcke auf Sägebohrlinie 1 : 65 Blöcke auf Sägebohrlinie 2 erzeugt werden.

Eine Aufteilung in 10er Blöcke bzw. 20er Blöcke kann diese Lösung nicht generieren. Andererseits ist der Suchraum wesentlich kleiner und dadurch effizienter absuchbar, wenn nur 8 Lose optimal auf die Sägebohrlinien aufgeteilt werden müssen als 32.

Im Rahmen der Testläufe hat sich herausgestellt, dass ein Splitting in 5er Blöcke nur sehr geringe Fitnessvorteile aufweist und sich nicht wesentlich von 10er und auch nicht von 20er Blöcken absetzen kann. Der Grund dafür könnte darin liegen, dass der Optimierungsalgorithmus durch den kleineren Suchraum weniger abgelenkt wird als bei einem unüberschaubar großen Suchraum und er dadurch relativ schnell ein einigermaßen gutes Verhältnis von z.B. 65:35 findet, während das Optimum z.B. bei 67:33 liegt, aber dieses Optimum auf Grund des damit verbundenen riesigen Suchraums in akzeptabler Laufzeit gar nicht gefunden würde, siehe dazu auch Abschnitt 8.1. Daher wird das Splitting des Ausgangsdatensatzes in allen Datensätzen beibehalten und nicht verändert.

7.1.3 Durchführung der Testläufe

Es werden drei Parametrisierungen vorgeschlagen um das Verhalten der Optimierungsalgorithmen in Bezug auf die Intensität der Suche zu analysieren.

Um eine bessere Vergleichbarkeit der einzelnen algorithmenspezifischen Parametrisierungen zu gewährleisten, werden die 3 Parametrisierungen derart gewählt, dass jedem Algorithmus im Vergleich mit einem anderen Algorithmus und derselben Parametrisierung (klein/mittel/groß) ca. gleich viele Bewertungen für die Optimierung einer Kampagne zur Verfügung stehen.

Für die kleine Parametrisierung stehen den einzelnen Algorithmen maximal ca. 180 Bewertungen zur Verfügung, für die mittlere Suche werden maximal ca. 360 Bewertungen aufgewendet, während der großen Suche maximal ca. 540 Bewertungen zur Verfügung stehen.

Abweichungen von der tatsächlich durchgeführten Anzahl an Bewertungen entstehen einerseits durch den zur Verfügung stehenden Suchraum einer Kampagne, andererseits werden auch bereits bewertete Reihenfolgen nicht noch einmal bewertet. Da diese Anzahl an generierten Reihenfolgen, welche bereits bewertet worden sind, für alle Algorithmen im Laufe der Optimierung einer Kampagne gleichermaßen zunimmt (auf Grund der Anzahl an durchgeführten Bewertungen im Vergleich zur Suchraumgröße), ist dies ein vernachlässigbarer Faktor in Bezug auf den Vergleich der einzelnen Algorithmen. Derartige Abweichungen kommen auch durch die unterschiedliche Qualität der Ergebnisse der einzelnen Optimierungsalgorithmen bzw. stochastische Einflüsse zustande.

7.1.3.1 Die Bedeutung der Toleranzgrenzen

Die Toleranzgrenzen weichen die 'lang vor kurz' Nebenbedingung auf, indem sie in dem vom Benutzer bestimmten Toleranzgrad Tauschvorgänge ermöglicht, im Rahmen dessen etwas kürzere Schienen nach vorne gereiht werden können. Die Läufe wurden mit 0% (strenge Einhaltung der Nebenbedingung) sowie 20% Toleranz (Abschnitt 7.10) durchgeführt. Testläufe mit über 24% machen meiner Meinung nach keinen Sinn mehr, weil dann schon Lose mit 60 Meter Schienen hinter Lose mit 45 Meter Schienen gereiht werden können, was einen groben Verstoß darstellt. Um die Bedeutung von viel höheren Toleranzen auszutesten, wurde auch auf einem beispielhaften Datensatz die 'lang vor kurz' Nebenbedingung deaktiviert, siehe dazu auch die entsprechenden Unterabschnitte in den Abschnitten 7.2 bis 7.7.

7.1.3.2 Zusammensetzung der durchgeführten Testläufe und Wahl der Algorithmen

Die insgesamt 5 verglichenen Optimierungsalgorihmen sollten auf 6 Datensätzen mit jeweils 3 unterschiedlichen Parametrisierungen sowie 4 Toleranzgrenzen verglichen werden. Außerdem sollten sämtliche Läufe mit 2 seed Werten für die erstellten Zufallsfolgen durchgeführt werden. Dies ergibt eine Gesamtanzahl von $5 \cdot 6 \cdot 3 \cdot 4 \cdot 2 = 720$ Durchläufen.

Begründung der Wahl der Algorithmen für die Durchführung von Testläufen anstelle der obigen Simulationsstudie: Die metaheuristisch zu optimierenden Kampagnen bestehen, berechnet über den Durchschnitt von 6 Datensätzen der durchschnittlichen heuristisch zu optimierenden Kampagnengröße eines Datensatzes, aus ca. 10 Losen (exakt: 9.9972 Lose). Einerseits sind fast alle Kampagnen mehr oder weniger stark von den Nebenbedingungen betroffen und in ihrem Suchraum damit eingeschränkt, andererseits sind v.a. kleinere Kampagnen rasch optimiert, während sich auch bei größeren Kampagnen mit riesigem Suchraum herausgestellt hat, dass nur ein Bruchteil dieses Suchraumes untersucht werden muss, um bereits sehr gute Ergebnisse zu bekommen.

Daher wurden diese intensiven Testläufe nicht durchgeführt, weil sich binnen kurzer Zeit herausgestellt hat, dass der Lösungsraum bereits nach sehr wenigen Bewertungen sehr gut optimiert ist und daher mehr Bewertungen keine bzw. nur minimale Verbesserungen darstellen. Auf Grund dieser Feststellung wurden für das Problem unter den realen Nebenbedingungen Testläufe durchgeführt, welche ermitteln sollen, nach wie vielen Bewertungen die Verfahren stagnieren und nur noch minimale Verbesserungen erzielen. Intensivere Durchläufe haben auf allen Algorithmen zu keinen "wesentlichen" Verbesserungen geführt. Die Testläufe wurden mit Simulated Annealing durchgeführt, weil sich dieses Verfahren sehr gut für weniger umfangreiche Läufe eignet. Im Gegensatz zum genetischen Algorithmus verzichtet es auf eine aufwändige Vorabselektion im Rahmen einer Evolutionsstrategie, andererseits eignet es sich, wie der hybride genetische Algorithmus, sehr gut für dieses Auftragsreihenfolgeproblem.

Die iterierte lokale Suche kommt auf Grund der relativ schlechten Gesamtperformance für die Testläufe nicht in Frage.

Die Variante des ACO wurde nicht für Testläufe verwendet, weil sich dieser auf Grund der besonderen Implementierung der Nebenbedingungen Reihenfolgen zusammenstellt, die zwar gültige und zulässige Lösungen darstellen, aber v.a. anfangs eine teils relativ schlechte Fitness aufweisen.

Der evolutionäre Algorithmus liefert durchaus gute Ergebnisse, allerdings lässt sich auf Grund der kampagnenspezifischen flexiblen Anpassung der Intensität seiner Suche nicht festlegen, wie viele Bewertungen in jede Kampagne verwendet worden sind, weil dies intern im Algorithmus in Abhängigkeit von der Kampagnengröße festgelegt ist.

Parametrisierungen der Testläufe: Die Testläufe werden daher für Simulated Annealing mit 0% Toleranz durchgeführt. Um die Lösungsgüte festzustellen, werden für dieses Verfahren viele kleine Durchläufe gestartet, welche insgesamt eine Kurve ergeben. Sämtliche Parametrisierungen 5 - 160 werden mittels Simulated Annealing mit 2 seed Werten durchgeführt. Die Ergebnisse werden anschließend gemittelt und auf einer Kurve aufgetragen. Alle Testläufe wurden unter allen aktivierten Nebenbedingungen sowie 0% Toleranz durchgeführt.

Parametrisierung 0': Es werden lediglich alle deterministisch zu lösenden Kampagnen zur Gänze optimiert.

Parametrisierung 10: Anfangstemperatur: 12°C, Erstarrungstemperatur: 9°C, Abkühlungsrate: 0.80, 5 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 10, Alpha Nachbarschaftsgröße: 4, Nachbarschaftsänderungsfaktor: 0.9, Toleranz: 0%

Parametrisierung 20: Anfangstemperatur: 12°C, Erstarrungstemperatur: 8°C, Abkühlungsrate: 0.90, 5 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 15, Alpha Nachbarschaftsgröße: 7, Nachbarschaftsänderungsfaktor: 0.85, Toleranz: 0%

Parametrisierung 40: Anfangstemperatur: 12°C, Erstarrungstemperatur: 5.5°C, Abkühlungsrate: 0.90, 5 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 20, Alpha Nachbarschaftsgröße: 10, Nachbarschaftsänderungsfaktor: 0.70, Toleranz: 0%

Parametrisierung 80: Anfangstemperatur: 12°C, Erstarrungstemperatur: 5.5°C, Abkühlungsrate: 0.90, 10 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 30, Alpha Nachbarschaftsgröße: 12, Nachbarschaftsänderungsfaktor: 0.70, Toleranz: 0%

Parametrisierung 160: Anfangstemperatur: 12°C, Erstarrungstemperatur: 4.5°C, Abkühlungsrate: 0.90, 16 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 40, Alpha Nachbarschaftsgröße: 15, Nachbarschaftsänderungsfaktor: 0.60, Toleranz: 0%

Im Rahmen der Parametrisierung 160 wurde teilweise auch die Variante mit 5.5°C sowie 20 Wiederholungen je Temperaturzustand getestet, jedoch ohne nennenswerten Änderungen.

Der relativ hohe Delta Skalierungsfaktor bei den intensiveren 80 und 160 Parametrisierungen gewährleistet, dass v.a. bei hohen Temperaturen auch noch Fitnesswerte, welche um z.B. 500 schlechter sind als das derzeitige Optimierungsniveau, akzeptiert werden. Dabei muss man außerdem bedenken, dass es sich um relativ große Fitnesswerte in der Höhe von $4 \cdot 10^6$ handelt, in dieser Relation ist eine um 500 schlechtere Lösung nicht viel schlechter.

Im Zuge der Durchführung der Testläufe muss an dieser Stelle festgehalten werden, dass die Ergebnisse der Parametrisierung 160 von Simulated Annealing mit einer ähnlichen Parametrisierung eines hybriden genetischen Algorithmus verglichen worden sind (10 Iterationen zu 15 Individuen, 1 Individuum für lokale Suche, Mutations Rate: 0.95, Crossover Rate: 0.20) bzw. mit dem Ant Colony System. Da mit diesen Algorithmen keine signifikanten Abweichungen aufgetreten sind, werden diese Ergebnisse zu den Testläufen von Simulated Annealing dazugezählt. Es treten jedoch sehr wohl stochastisch bedingt Änderungen auf, wobei der Einfluss der Stochastik nur durch den Einsatz von sehr vielen (> 100) seed Werten zu minimieren ist. An dieser Stelle sei auch angemerkt, dass zur Erstellung eines Punktes in der Kurve mindestens 2, meistens jedoch mehr seed Werte verwendet worden sind, um einigermaßen verlässliche Kurvenverläufe zu erhalten. Ich möchte auch anmerken, dass die Stochastik gerade bei wenigen seed Werten doch einen erheblicheren Einfluss auf das konkrete Optimierungsergebnis hat als die Auswahl des Algorithmus, dies gilt v.a. für die weniger umfangreichen Parametrisierungen. Ab der Parametrisierung 160 sind die Unterschiede bedingt durch andere seed Werte geringer einzustufen, genauso wie die unterschiedlich erzielten Ergebnisse im Rahmen der Auswahl eines anderen Algorithmus.

Parametrisierungen der Testläufe im Rahmen der Deaktivierung von Nebenbedingung 3: Anschließend werde die Algorithmen unter einem virtuell vergrößerten Suchraum verglichen, im Rahmen dessen sie sich auch vergleichen lassen. Der Grund dafür liegt darin, dass durch das Wegfallen der lang vor kurz Nebenbedingung mehrere Lose existieren können (je nach Datensatz), die nun an mehereren Stellen als zuvor platziert werden können, was unter den realen Gegebenheiten nicht möglich ist.

Parametrisierung 160: Anfangstemperatur: 12°C, Erstarrungstemperatur: 5.5°C, Abkühlungsrate: 0.90, 20 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 40, Alpha Nachbarschaftsgröße: 15, Nachbarschaftsänderungsfaktor: 0.60, Toleranz: 0%

Simulated Annealing 320: Anfangstemperatur: 12°C, Erstarrungstemperatur: 5.5°C, Abkühlungsrate: 0.90, 40 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 40, Alpha Nachbarschaftsgröße: 15, Nachbarschaftsänderungsfaktor: 0.60, Toleranz: 0%

Simulated Annealing 600: Anfangstemperatur: 12°C, Erstarrungstemperatur: 5.5°C, Abkühlungsrate: 0.95, 40 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 40, Alpha Nachbarschaftsgröße: 15, Nachbarschaftsänderungsfaktor: 0.65, Toleranz: 0%

Hybrider genetischer Algorithmus 500: 20 Iterationen, 25 Individuen, Crossover Rate: 0.20, Mutations Rate: 0.90, Alpha Wert: 15, Elitismus Individuen: 4, Anzahl Individuen für lokale Suche: 2 (3)

Hybrider genetischer Algorithmus 600: 20 Iterationen, 30 Individuen, Crossover Rate: 0.20, Mutations Rate: 0.90, Alpha Wert: 15, Elitismus Individuen: 5, Anzahl Individuen für lokale Suche: 5

7.1.3.3 Repräsentation und Darstellung der Ergebnisse

Um eine Übersichtlichkeit der Ergebnisse gewährleisten, wird nur der optimale Fitnesswert, das sich daraus ergebende Kostenreduktionspotenzial sowie die Gesamtdurchlaufzeit der besten Reihenfolge ausgegeben.

Auf eine Ausgabe der Fitnesswerte aller optimierten Kampagnen sowie eine detaillierte Lösungsdarstellung inklusive Abbildungen der Lösung (Gegenüberstellung der Ausgangsreihenfolge mit der optimierten Reihenfolge, Vergleich der Auslastungszustände der Railman Aggregate der Ausgangsreihenfolge mit der optimierten Reihenfolge, etc,...) wird verzichtet, da diese Informationen gesammelt im optimalen *ResultRating* bzw. prinzipiell jedem *ResultRating* neben dem Fitnesswert dargestellt sind. Aus der Summe dieser Informationen setzt sich der Fitnesswert zusammen, siehe auch Abschnitt 2.2.

Sämtliche in den folgenden Abschnitten präsentierten Ergebnisse wurden in Bezug auf die Einhaltung der gestellten Nebenbedingungen kontrolliert und mittels Abbildungen protokolliert.

Beispiel betreffend die Visualisierung eines Ergebnisses: Die folgende Visualisierung ist zweckdienlich für alle anderen Durchläufe um einen Überblick und einen Vergleich einer Ausgangsreihenfolge mit einer optimierten Reihenfolge zu geben. In den folgenden beiden Abbildungen 7.1 und 7.2 sieht man deutlich die Unterschiede in der Auftragsreihenfolge der Ausgangssequenz mit der optimierten Auftragsreihenfolge innerhalb der einzelnen Kampagnen.

Aufträge | Aggregate | Pufferspeicher

19.11.2009 13:26:13 Bewertung

Simulationsbeginn: 05.10.2009 17:11:53
Simulationsende: 10.10.2009 09:09:24
Dauer: 4d 15:57:31
Fitness Value: 624375,00

Asc...	W...	LotId	#...	Profil	Qualität	Schnittmus...	ZS	ZEK	Lieferdatum	Fertigstellungs...
0	1	122724-4	0	VI49E1	Normal	6x18000	ZS2	EK2	01.10.2009	Keine Daten vo...
1	2	122885-5	0	VI60E1	HSH	2x60000	ZS1	EK1	07.10.2009	05.10.2009
2	2	121900-64	0	VI60E1	Bainit	2x27000	ZS1	EK1	31.12.2009	05.10.2009
3	2	122885-5A	0	VI60E1	HSH	2x60000	ZS2	EK4	07.10.2009	Keine Daten vo...
4	2	122885-5AA	2	VI60E1	HSH	2x60000	ZS2	EK4	07.10.2009	05.10.2009
5	2	121900-64A	1	VI60E1	Bainit	2x27000	ZS1	EK1	31.12.2009	05.10.2009
6	3	121915-180	23	RI53R1	Normal	6x18000	ZS2	EK2	05.10.2009	06.10.2009
7	3	123112-3	3	RI53R1	Normal	6x18000	ZS2	EK2	05.10.2009	06.10.2009
8	3	123130-1	8	RI53R1	Normal	6x18000	ZS2	EK2	12.10.2009	06.10.2009
9	3	123081-1	80	RI53R1	Normal	7x15000	ZS2	EK2	15.10.2009	06.10.2009
10	4	123023-1	111	VI60E1	HSH	3x36000	ZS1	EK1	09.10.2009	06.10.2009
11	4	122885-5AAA	64	VI60E1	HSH	2x60000	ZS2	EK4	07.10.2009	06.10.2009
12	4	121591-128	20	VI60E1	HSH	2x60000	ZS2	EK4	13.10.2009	07.10.2009
13	4	121591-127	40	VI60E1	HSH	2x60000	ZS2	EK4	12.10.2009	07.10.2009
14	4	123037-2	6	VI60E1	HSH	5x22350	ZS1	EK1	19.10.2009	07.10.2009
15	5	123085-4	48	RI59R2	HSH	6x18000	ZS2	EK2	07.10.2009	07.10.2009
16	5	121915-193	12	RI59R2	HSH	6x18000	ZS1	EK1	15.10.2009	07.10.2009
17	5	121915-192	18	RI59R2	Normal	6x18000	ZS2	EK2	15.10.2009	07.10.2009
18	5	121868-13	25	RI59R2	Normal	6x18000	ZS2	EK2	12.10.2009	07.10.2009
19	5	121868-15	4	RI59R2	Normal	6x18000	ZS2	EK2	12.10.2009	07.10.2009
20	5	121868-14	3	RI59R2	Normal	6x18000	ZS2	EK2	12.10.2009	07.10.2009
21	5	121868-17	4	RI59R2	Normal	6x18000	ZS2	EK2	19.10.2009	07.10.2009
22	6	123036-3	17	RI59R1	Normal	6x18000	ZS1	EK1	09.10.2009	07.10.2009
23	6	123036-2	4	RI59R1	Normal	6x18000	ZS2	EK2	09.10.2009	07.10.2009
24	7	122813-5	11	KSMRS87A	Normal	7x12000	ZS1	EK1	30.09.2009	07.10.2009
25	7	122813-6	11	KSMRS87A	Normal	7x12000	ZS2	EK2	07.10.2009	08.10.2009
26	7	122813-7	11	KSMRS87A	Normal	7x12000	ZS1	EK1	14.10.2009	07.10.2009
27	7	122813-9	10	KSMRS87A	Normal	7x12000	ZS2	EK2	28.10.2009	08.10.2009
28	7	122813-8	11	KSMRS87A	Normal	7x12000	ZS1	EK1	21.10.2009	07.10.2009
29	7	122813-10	11	KSMRS87A	Normal	7x12000	ZS2	EK2	04.11.2009	08.10.2009
30	7	122813-11	10	KSMRS87A	Normal	7x12000	ZS1	EK1	11.11.2009	08.10.2009
31	7	122813-13	6	KSMRS87A	Normal	7x12000	ZS2	EK2	24.11.2009	08.10.2009
32	7	122813-12	2	KSMRS87A	Normal	7x12000	ZS1	EK1	18.11.2009	08.10.2009
33	8	122885-4	223	VISAR48	Normal	2x60000	ZS2	EK4	07.10.2009	08.10.2009
34	8	122885-3	128	VISAR48	HSH	2x60000	ZS2	EK4	07.10.2009	09.10.2009
35	9	121900-71	5	VIR65	HSH	3x30000	ZS1	EK1	31.12.2009	09.10.2009
36	9	121900-68	9	VIR65	HSH	4x25000	ZS2	EK2	31.12.2009	09.10.2009
37	10	122406-4	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
38	10	122406-4_	10	RI54G2	Normal	6x18000	ZS2	EK2	22.10.2009	09.10.2009
39	10	122406-4__	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
40	10	122406-4___	10	RI54G2	Normal	6x18000	ZS2	EK2	22.10.2009	09.10.2009
41	10	122406-4_...	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
42	10	122406-4_...	17	RI54G2	Normal	6x18000	ZS2	EK2	22.10.2009	09.10.2009
43	10	122406-4_...	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
44	10	122406-4_...	18	RI54G2	Normal	6x18000	ZS2	EK2	22.10.2009	09.10.2009
45	10	121915-174	20	RI54G2	HSH	6x18000	ZS1	EK1	06.10.2009	09.10.2009
46	11	121591-143	44	VI50E3	Normal	2x60000	ZS2	EK4	15.10.2009	09.10.2009
47	11	121591-142	44	VI50E3	Normal	2x60000	ZS2	EK4	14.10.2009	10.10.2009
48	11	122953-2	12	VI50E3	HSH	4x24821	ZS1	EK1	26.11.2009	10.10.2009
49	11	122727-2	12	VI50E3	HSH	4x24821	ZS2	EK2	27.10.2009	10.10.2009
50	11	123075-1	12	VI50E3	HSH	4x24821	ZS1	EK1	04.11.2009	10.10.2009
51	11	123075-2	11	VI50E3	HSH	4x24821	ZS2	EK2	25.01.2010	10.10.2009
52	11	122794-3	8	VI50E3	HSH	18x6700	ZS1	EK1	01.10.2009	10.10.2009

Abb. 7.1: Auftragsreihenfolge der Ausgangssequenz

Aufträge | Aggregate | Pufferspeicher

Auftragsliste

Simulationsbeginn: 05.10.2009 17:11:53
Simulationsende: 10.10.2009 11:05:29
Dauer: 4d 17:53:36
Fitness Value: 596683,00

Asc...	W...	LotId	#...	Profil	Qualität	Schnittmus...	ZS	ZEK	Lieferdatum	Fertigstellungs...
0	1	122724-4	0	VI49E1	Normal	6x18000	ZS2	EK2	01.10.2009	Keine Daten vo...
1	2	122885-5	0	VI60E1	HSH	2x60000	ZS1	EK1	07.10.2009	05.10.2009
2	2	121900-64	0	VI60E1	Bainit	2x27000	ZS1	EK1	31.12.2009	05.10.2009
4	2	122885-5AA	2	VI60E1	HSH	2x60000	ZS1	EK1	07.10.2009	05.10.2009
4	2	121900-64A	1	VI60E1	Bainit	2x27000	ZS1	EK1	31.12.2009	05.10.2009
5	2	122885-5A	0	VI60E1	HSH	2x60000	ZS2	EK4	07.10.2009	Keine Daten vo...
6	3	121915-180	23	RI53R1	Normal	6x18000	ZS2	EK2	05.10.2009	06.10.2009
7	3	123130-1	8	RI53R1	Normal	6x18000	ZS2	EK2	12.10.2009	06.10.2009
8	3	123112-3	3	RI53R1	Normal	6x18000	ZS2	EK2	05.10.2009	06.10.2009
9	3	123081-1	80	RI53R1	Normal	7x15000	ZS2	EK2	15.10.2009	06.10.2009
10	4	121591-128	20	VI60E1	HSH	2x60000	ZS1	EK1	13.10.2009	06.10.2009
11	4	123023-1	111	VI60E1	HSH	3x36000	ZS1	EK1	09.10.2009	06.10.2009
12	4	122885-5AAA	64	VI60E1	HSH	2x60000	ZS2	EK4	07.10.2009	07.10.2009
13	4	121591-127	40	VI60E1	HSH	2x60000	ZS2	EK4	12.10.2009	07.10.2009
14	4	123037-2	6	VI60E1	HSH	5x22350	ZS1	EK1	19.10.2009	07.10.2009
15	5	123085-4	48	RI59R2	HSH	6x18000	ZS2	EK2	07.10.2009	07.10.2009
16	5	121915-193	12	RI59R2	HSH	6x18000	ZS1	EK1	15.10.2009	07.10.2009
17	5	121915-192	18	RI59R2	Normal	6x18000	ZS2	EK2	15.10.2009	07.10.2009
18	5	121868-13	25	RI59R2	Normal	6x18000	ZS2	EK2	12.10.2009	07.10.2009
19	5	121868-17	4	RI59R2	Normal	6x18000	ZS2	EK2	19.10.2009	07.10.2009
20	5	121868-14	3	RI59R2	Normal	6x18000	ZS2	EK2	12.10.2009	07.10.2009
21	5	121868-15	4	RI59R2	Normal	6x18000	ZS2	EK2	12.10.2009	07.10.2009
22	6	123036-2	4	RI59R1	Normal	6x18000	ZS2	EK2	09.10.2009	07.10.2009
23	6	123036-3	17	RI59R1	Normal	6x18000	ZS1	EK1	09.10.2009	07.10.2009
24	7	122813-5	11	KSMRS87A	Normal	7x12000	ZS1	EK1	30.09.2009	07.10.2009
25	7	122813-13	6	KSMRS87A	Normal	7x12000	ZS2	EK2	24.11.2009	08.10.2009
26	7	122813-7	11	KSMRS87A	Normal	7x12000	ZS1	EK1	14.10.2009	07.10.2009
27	7	122813-8	11	KSMRS87A	Normal	7x12000	ZS1	EK1	21.10.2009	08.10.2009
28	7	122813-12	2	KSMRS87A	Normal	7x12000	ZS1	EK1	18.11.2009	08.10.2009
29	7	122813-11	10	KSMRS87A	Normal	7x12000	ZS1	EK1	11.11.2009	08.10.2009
30	7	122813-9	10	KSMRS87A	Normal	7x12000	ZS2	EK2	28.10.2009	08.10.2009
31	7	122813-6	11	KSMRS87A	Normal	7x12000	ZS2	EK2	07.10.2009	08.10.2009
32	7	122813-10	11	KSMRS87A	Normal	7x12000	ZS2	EK2	04.11.2009	08.10.2009
33	8	122885-4	223	VISAR48	Normal	2x60000	ZS2	EK4	07.10.2009	09.10.2009
34	8	122885-3	128	VISAR48	HSH	2x60000	ZS2	EK4	07.10.2009	09.10.2009
35	9	121900-68	9	VIR65	HSH	4x25000	ZS2	EK2	31.12.2009	09.10.2009
36	9	121900-71	5	VIR65	HSH	3x30000	ZS1	EK1	31.12.2009	09.10.2009
37	10	122406-4	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
38	10	122406-4_	10	RI54G2	Normal	6x18000	ZS2	EK2	22.10.2009	09.10.2009
39	10	122406-4__	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
40	10	122406-4___	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
41	10	122406-4_...	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
42	10	122406-4_...	17	RI54G2	Normal	6x18000	ZS2	EK2	22.10.2009	09.10.2009
43	10	122406-4_...	10	RI54G2	Normal	6x18000	ZS1	EK1	22.10.2009	09.10.2009
44	10	122406-4_...	18	RI54G2	Normal	6x18000	ZS2	EK2	22.10.2009	09.10.2009
45	10	121915-174	20	RI54G2	HSH	6x18000	ZS1	EK1	06.10.2009	09.10.2009
46	11	121591-142	44	VI50E3	Normal	2x60000	ZS2	EK4	14.10.2009	10.10.2009
47	11	121591-143	44	VI50E3	Normal	2x60000	ZS2	EK4	15.10.2009	10.10.2009
48	11	123075-1	12	VI50E3	HSH	4x24821	ZS1	EK1	04.11.2009	10.10.2009
49	11	123075-2	11	VI50E3	HSH	4x24821	ZS2	EK2	25.01.2010	10.10.2009
50	11	122953-2	12	VI50E3	HSH	4x24821	ZS1	EK1	26.11.2009	10.10.2009
51	11	122727-2	12	VI50E3	HSH	4x24821	ZS2	EK2	27.10.2009	10.10.2009
52	11	122794-3	8	VI50E3	HSH	18x6700	ZS1	EK1	01.10.2009	10.10.2009

Abb. 7.2: Auftragsreihenfolge der optimierten Reihenfolge

Im folgenden Visualisierungsbeispiel werden auch die im Rahmen der Fitnesswertberechnung wichtigen Hauptpufferspeicher Railman 1 und Railman 2 in der Ausgangsreihenfolge (siehe Abb. 7.3) sowie in der optimierten Reihenfolge (siehe Abb. 7.4) dargestellt. Die Pönale bezüglich der Termintreue (siehe auch Abschnitt 2.2 bzw. Abb. 2.1) errechnet sich in Sekundengenauigkeit mittels der Differenz zwischen Fertigstellungstermin und Lieferdatum. Die beiden Zeitpunkte sind nur in der Visualisierung lediglich in Tagegenauigkeit dargestellt um einen raschen Überblick über diesen Sachverhalt geben zu können.

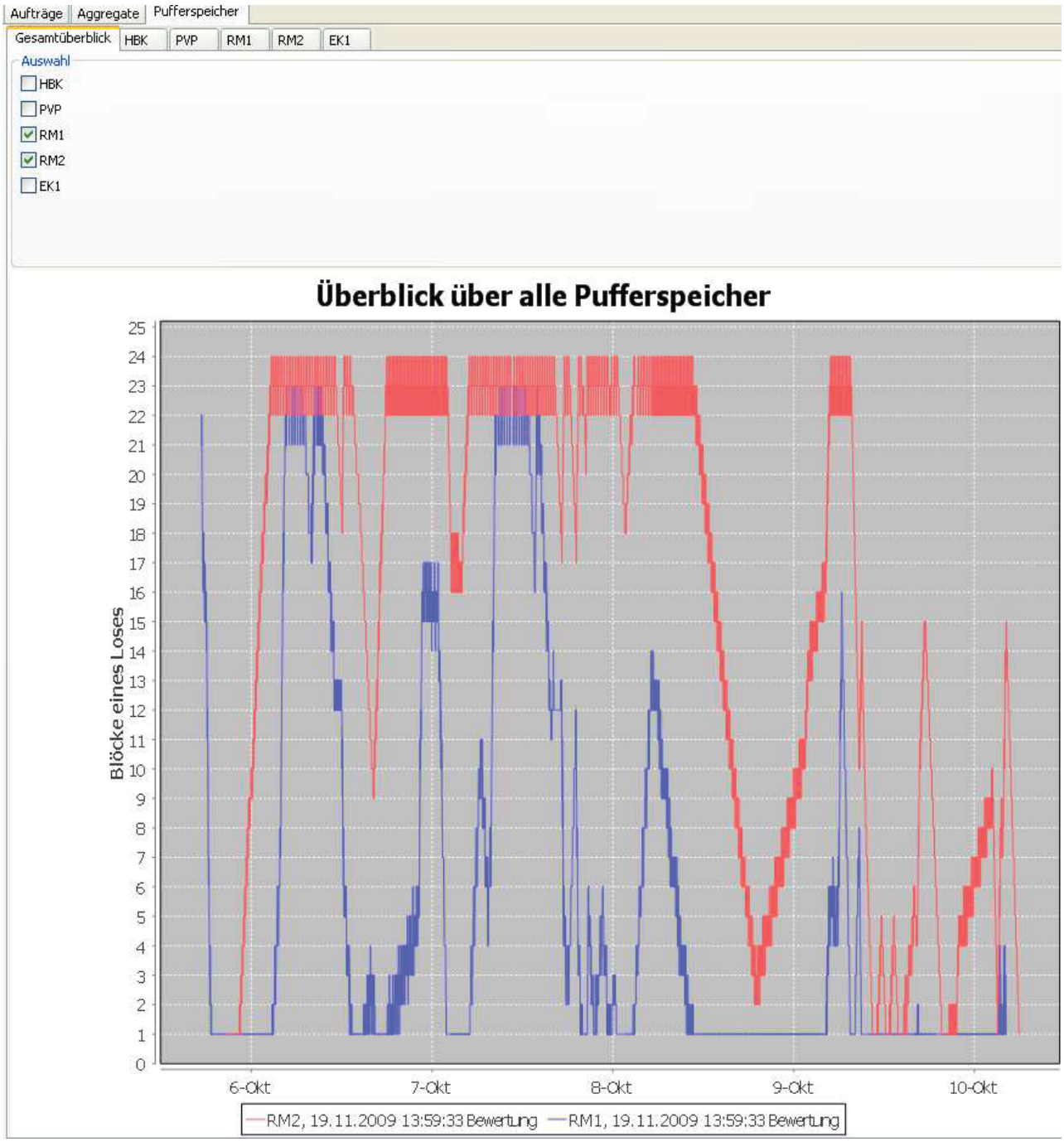


Abb. 7.3: Auslastungszustände des Railman 1 und Railman 2 Aggregates in der Ausgangsreihenfolge

Wenn man Abb. 7.3 und 7.4. miteinander vergleicht, sieht man deutlich, dass die Pufferzustände von Railman 1 und 2 in der optimierten Reihenfolge in bestimmten Zeitbereichen teils deutlich geringer und dadurch besser, weil näher am Optimum, sind. Diese Zustände haben einen erheblichen Anteil an der Fitnesswertreduktion. Die Feststellung, dass die Produktion im Rahmen der optimierten Reihenfolge ca. 2 Stunden länger dauert, ist in diesem Fall vernachlässigbar gering. Im Rahmen der gesamten Optimierung aller Kampagnen ist i.d.R. eine deutliche Gesamtdurchlaufzeitverkürzung im Umfang von einigen Prozentpunkten feststellbar, welche über jede einzelne optimierte Kampagne aufsummiert worden ist.

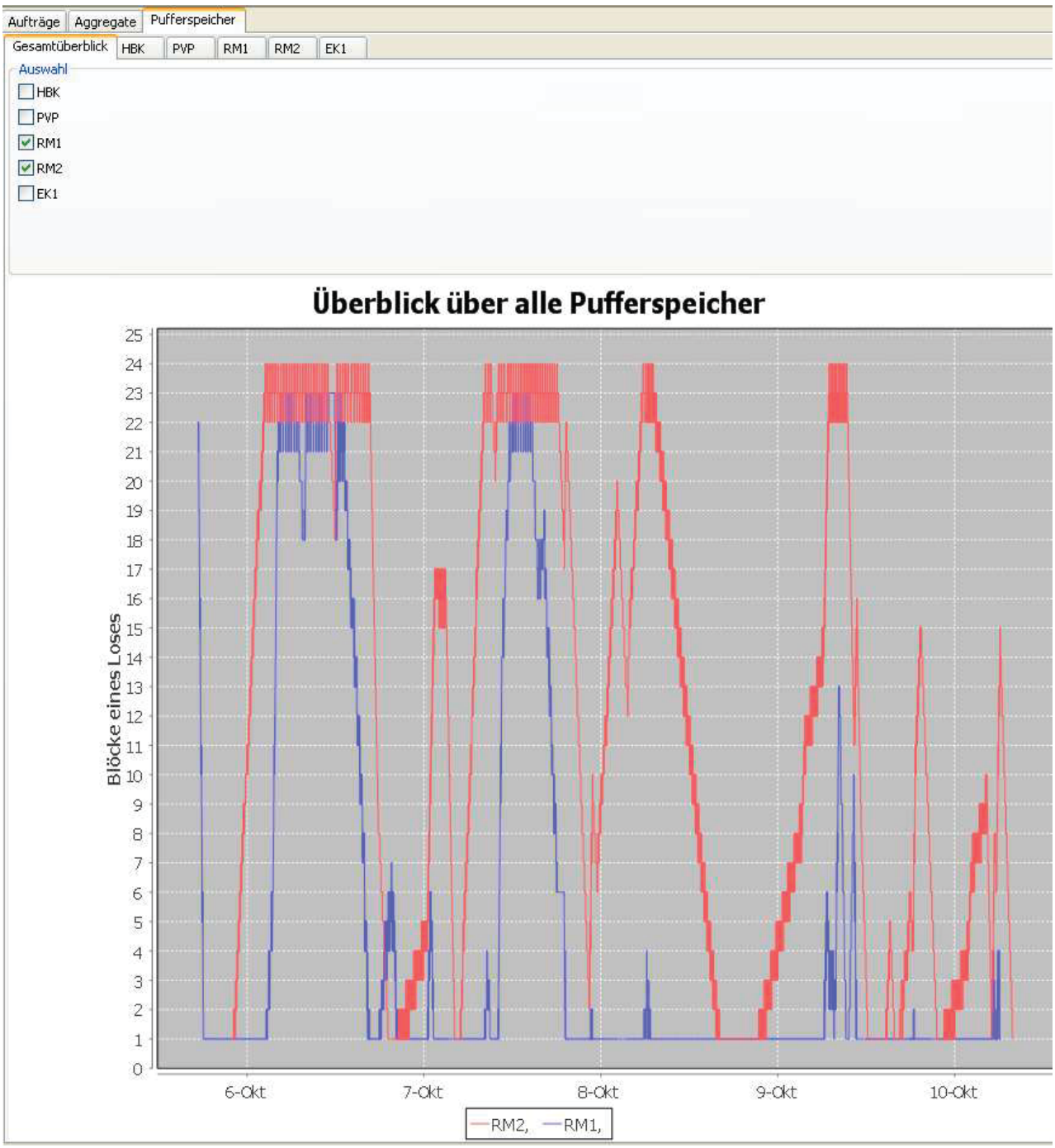


Abb. 7.4: Auslastungszustände des Railman 1 und Railman 2 Aggregates in der optimierten Reihenfolge

7.2 Simulationsergebnisse auf Datensatz 1

Datensatz 1 (Simulation200911051233_rsteri) ist zeitlich betrachtet der aktuellste Datensatz und besteht aus 28 Kampagnen. Der Betrachtungszeitraum liegt zwischen dem 5.11.2009 (12:33:15) und dem 16.12.2009 (20:44:36)

Die Simulationsdauer bzw. die benötigte Gesamtdurchlaufzeit der Ausgangsreihenfolge beträgt: 41 Tage, 8 Stunden, 11 Minuten, 21 Sekunden (=**992,18 Stunden**)

Der Fitnesswert der Ausgangssequenz aller 28 Kampagnen beträgt: 3.495.337.

Spezifische Charakteristika dieses Datensatzes: Aus Sicht der Optimierung besteht dieser Datensatz aus einer deterministisch zu optimierenden Kampagne (Kampagne 14) sowie 6 Kampagnen mit insgesamt 7 kleineren und größeren ZSG Losen. Dadurch erhält die Änderung der Zielsäge einen erheblichen Einfluss auf die Optimierung. Der Einsatz von Toleranzen vergrößert zwar den Suchraum einiger Kampagnen (3, 13), insgesamt bleibt der Einsatz von Toleranzen von z.B. 20% vernachlässigbar gering und kann - abgesehen von weiteren Störeffekten, welche sich durch die kampagnenübergreifende Optimierung ergeben, vernachlässigt werden. 12 Kampagnen bestehen aus jeweils einem Los, somit bleiben 15 Kampagnen für die metaheuristische Optimierung. Die Kampagnen 2,3,4,5 und 13 können ein wenig vom Einsatz von Toleranzen profitieren.

Die metaheuristischen Kampagnen haben eine mittlere Größe von ca. jeweils $8,93$ Losen. (148 Lose$-($ $12{\cdot}1 + 1{\cdot}2) = 134$ Lose / 15 Kampagnen $= 8,93$ Lose je heuristisch zu optimierende Kampagne) Der Datensatz zählt daher zu den einfacher zu optimierenden Datensätzen.

7.2.1 Fitnessverlauf von Simulated Annealing auf diesem Datensatz unter realen Bedingungen

Obwohl es nur eine deterministisch zu optimierende Kampagne gibt (Kampagne 14), hat diese bereits erheblichen Einfluss auf das Gesamtoptimum. Es handelt sich bei dieser aus 2 Lose gleicher Güteklasse und Länge (Lose mit 24 Meter Schienen) bestehenden Kampagne um 2 Lose mit sehr vielen Blöcken, wodurch auf Grund der relativ kurzen Schienenlänge auch relativ viele Schnitte an den beiden Sägebohrlinien notwendig werden.

In Abb. 7.5 zeigt sich, dass bereits nach relativ wenigen Bewertungen je Kampagne der Großteil optimiert ist. Diese Annahme wird dadurch bestätigt, dass es keine riesigen Kampagnen mit mehr als 20 Losen in diesem Datensatz gibt und die Nebenbedingungen die mittelgroßen Kampagnen relativ stark einschränken. Das Optimierungspotenzial, welches von der besten Reihenfolge erzielt wird, beträgt auf diesem Datensatz ca. 4,23%

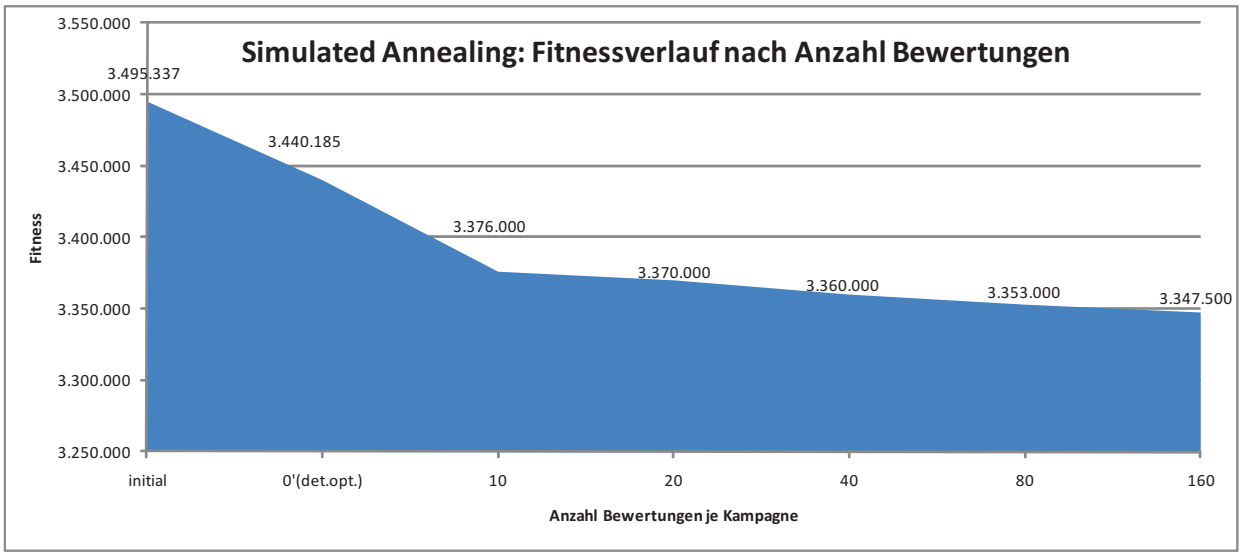


Abb. 7.5: Fitnessverlaufskurve für Datensatz 1

7.2.2 Ergebnisse bei Deaktivierung von Nebenbedingung 3

Im Rahmen der Parametrisierung hybrider genetischer Algorithmus 500 wurde einmal ein Fitnesswert von 3.340.230 sowie einmal (leicht modifiziert) ein Fitnesswert von 3.335.314 erreicht. Dies entspricht einem Gesamtkostenreduktionspotenzial von 4,52% sowie einer zusätzlichen Kostenreduktion von 0.29% bei Wegfallen von Nebenbedingung 3. Dieses zusätzliche Einsparungspotenzial ist außerordentlich gering. Dies resultiert auch daraus, dass der Datensatz über relativ kleine Kampagnengrößen, die ihrerseits

durchschnittlich weniger von dem Einsatz von Toleranzen bzw. vom Wegfall der Nebenbedingung 3 profitieren.

7.3 Simulationsergebnisse auf Datensatz 2

Datensatz 2 (Simulation20090910051711_rsteri_online.xls) ist zeitlich betrachtet ein relativ neuer Datensatz und besteht aus 32 Kampagnen. Der Betrachtungszeitraum liegt zwischen dem 15.10.2009 (17:11:53) und dem 11.11.2009 (03:49:56)

Die Simulationsdauer bzw. die benötigte Gesamtdurchlaufzeit der Ausgangsreihenfolge beträgt: 36 Tage, 11 Stunden, 38 Minuten, 3 Sekunden

Der Fitnesswert der Ausgangssequenz aller 32 Kampagnen beträgt: 3.444.169

Spezifische Charakteristika dieses Datensatzes: Der Datensatz besteht aus 4 deterministisch zu optimierenden Kampagnen, welche aus jeweils 2 Losen bestehen. Insgesamt gibt es 11 Kampagnen mit jeweils einem Los, daher verbleiben 17 Kampagnen für die metaheuristische Optimierung.

Die metaheuristischen Kampagnen haben eine mittlere Größe von ca. jeweils 7,71 Losen. (150 Lose−(11·1 + 4·2) = 131 Lose / 17 Kampagnen = 7, 71 Lose je heuristisch zu optimierende Kampagne) Dieser Datensatz ist damit der von der Größe betrachtet der am einfachsten zu optimierende Datensatz.

Außerdem ist der Datensatz von insgesamt 5 Kampagnen mit jeweils einem ZSG Los gekennzeichnet sowie relativ wenigen Losen mit verschieden langen Schienen je Kampagne. Der Einsatz von Toleranzen macht auf diesem Datensatz daher keinen Sinn. Weiter schränkt die 'lang vor kurz' Nebenbedingung den Suchaum auf Grund dieser Tatsache relativ wenig ein.

Der Suchraum auch kleinerer Kampagnen (Kampagne 7, welche über 9 Lose verfügt, welche durch die Nebenbedingungen in Bezug auf ihre Tauschmöglichkeiten nicht eingeschränkt werden, hat einen Suchraum, der eine Größe von 9! besitzt) ist auf diesem Datensatz verblüffend groß, z.B. die Kampagnen 20 und 21, welche ausschließlich bzw. im zweiteren Falle großteils aus dem ZSG Los bestehen. Abgesehen von einer Tauschmöglichkeit in Kampagne 3 bzw. 9 (deterministische 2 losige Kampagne, welche für einen Tauschvorgang eine Toleranz von 20% benötigt) ändert der Einsatz von Toleranzen im Rahmen von bis zu 24% nichts an der Suchraumgröße.

7.3.1 Fitnessverlauf von Simulated Annealing auf diesem Datensatz unter realen Bedingungen

Abb. 7.6 liefert eine Zusammenfassung der erreichten Ergebnisse. Obwohl auf den 4 deterministisch zu optimierenden Kampagnen praktisch kein Optimierungspotenzial besteht, ist dieses Potenzial auf den metaheuristisch zu optimierenden Kampagnen verblüffend groß, sowohl nach Testläufen mit sehr kleinen Parametrisierungen je Kampagne als auch im Rahmen von intensiveren Simulationsläufen. Im Rahmen dieser umfangreicheren Durchläufe (160 Bewertungen, siehe 7.1.3.2) wurde ein Fitnesswert (gemittelt) von 3.151.000 erreicht, dies entspricht einer um ca. 8,51% besseren Fitness gegenüber der Ausgangsreihenfolge. Da in diversen längeren Läufen keine besseren Ergebnisse mehr erreicht worden sind, ist damit wohl eine sehr gute Lösung in der Nähe des globalen Optimums erreicht.

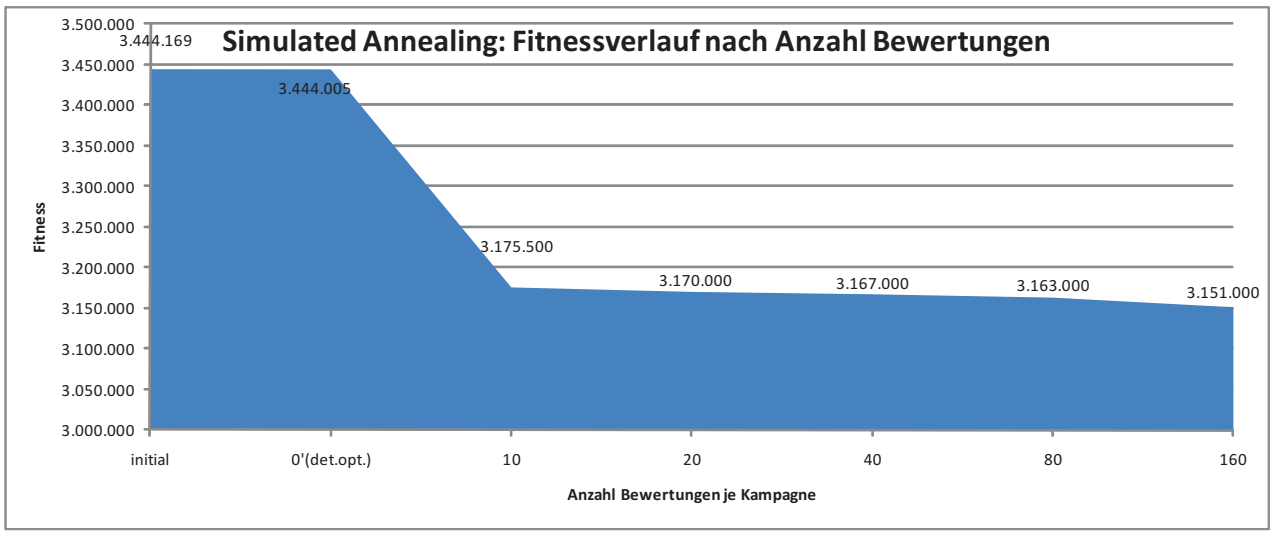


Abb. 7.6: Fitnessverlaufskurve für Datensatz 2

7.3.2 Ergebnisse bei Deaktivierung von Nebenbedingung 3

Datensatz 1 erreicht mit der Parametrisierung Simulated Annealing 600 eine Fitness von 3.115.357 sowie mit der Parametrisierunge hybrider genetischer Algorithmus 600 eine Fitness von 3.114.849. Man erkennt, dass hier beide Verfahren bereits konvergiert sind und bei neuen Durchläufen v.a. stochastisch bedingte Unterschiede bedingt durch andere Seed Werte auftreten. Stochastisch bedingt können einzelne Durchläufe an die 3.100.000 Grenze herankommen. Zusammenfassend lässt sich in Bezug auf diesen Datensatz sagen, dass das zusätzliche Kostenreduktionspotenzial bei Weglassen der Nebenbedingung gering ausfällt. Es beträgt insgesamt 9,56%, was einer 1,05% Verbesserung gegenüber realen Bedingungen entspricht.

7.4 Simulationsergebnisse auf Datensatz 3

Datensatz 3 (Simulation200909220828_rsteri.xls) ist zeitlich betrachtet ein aktueller Datensatz und besteht aus 34 Kampagnen. Der Betrachtungszeitraum liegt zwischen dem 22.9.2009 (8:28:29) und dem 5.11.2009 (9:00:20)

Die Simulationsdauer bzw. die benötigte Gesamtdurchlaufzeit der Ausgangsreihenfolge beträgt: 44 Tage, 1 Stunde, 31 Minuten, 51 Sekunden

Der Fitnesswert der Ausgangssequenz aller 34 Kampagnen beträgt: 4.439.166

Spezifische Charakteristika dieses Datensatzes: Dieser Datensatz besteht aus 12 Kampagnen mit jeweils einem Los sowie 5 deterministisch zu optimierenden Kampagnen, damit verbleiben 17 Kampagnen für die metaheuristische Optimierung. Außerdem ist der Datensatz durch 7 Kampagnen mit insgesamt 8 kleineren und größeren ZSG Losen gekennzeichnet, wodurch die Änderung der Zielsäge einen entsprechenden Anteil an der Gesamtoptimierung ausmacht. Die metaheuristischen Kampagnen haben eine mittlere Größe von ca. jeweils 11,12 Losen. (190 Lose$-($ 12·1 + 4·2 + 1 · 3) = 189 Lose / 17 Kampagnen = 11, 12 Lose je heuristisch zu optimierende Kampagne) In Bezug auf die durchschnittliche Kampagnengröße scheint dieser Datensatz schwerer zu optimieren zu sein. Die Auswirkung von Toleranzen ist auf diesem Datensatz vernachlässigbar gering, lediglich die Kampagnen 4,6,9 und 13 können ein wenig davon profitieren.

7.4.1 Fitnessverlauf von Simulated Annealing auf diesem Datensatz unter realen Bedingungen

Während in den deterministischen Kampagnen nichts zu optimieren ist, wird in den metaheuristisch zu optimierenden Kampagnen schon im Rahmen von jeweils sehr kleinen Parametrisierungen ein Optimierungspotenzial von ca. 10% erreicht. Die beste Reihenfolge im Rahmen eines intensiveren Durchlaufs erzielt eine Verbesserung im Umfang von ca. 11,36%.

Abb. 7.7 liefert eine Zusammenfassung der erreichten Ergebnisse. Im Rahmen der intensiveren Durchläufe (160 Bewertungen, siehe 7.1.3.2) wurde ein Fitnesswert von 3.935.000 erreicht. Da in diversen umfassenderen Läufen (30 Individuen zu 20 Iterationen bei einem HGA bzw. 30 Ameisen zu 20 Iterationen beim ACO) bei aktivierten Nebenbedingungen keine besseren Ergebnisse mehr erreicht worden sind, kann nach die Optimierung nach wenigen hundert Bewertungen je Kampagne abgebrochen werden. Vergleiche mit einem hybriden genetischen Algorithmus ähnlich umfangreicher Parametrisierung brachten sehr ähnliche Ergebnisse. 4 Kampagnen können von dem Einsatz von Toleranzen auf diesem Datensatz ein wenig profitieren. Verblüffend bleibt die Tatsache, dass offensichtlich je Kampagne außerordentlich wenige Positionen (1 - 3) verändert werden müssen im Vergleich zur Ausgangsreihenfolge, sonst könnte man kaum nach bereits derart wenigen Bewertungen (z.B. 10 - 20) bereits den Großteil optimiert haben.

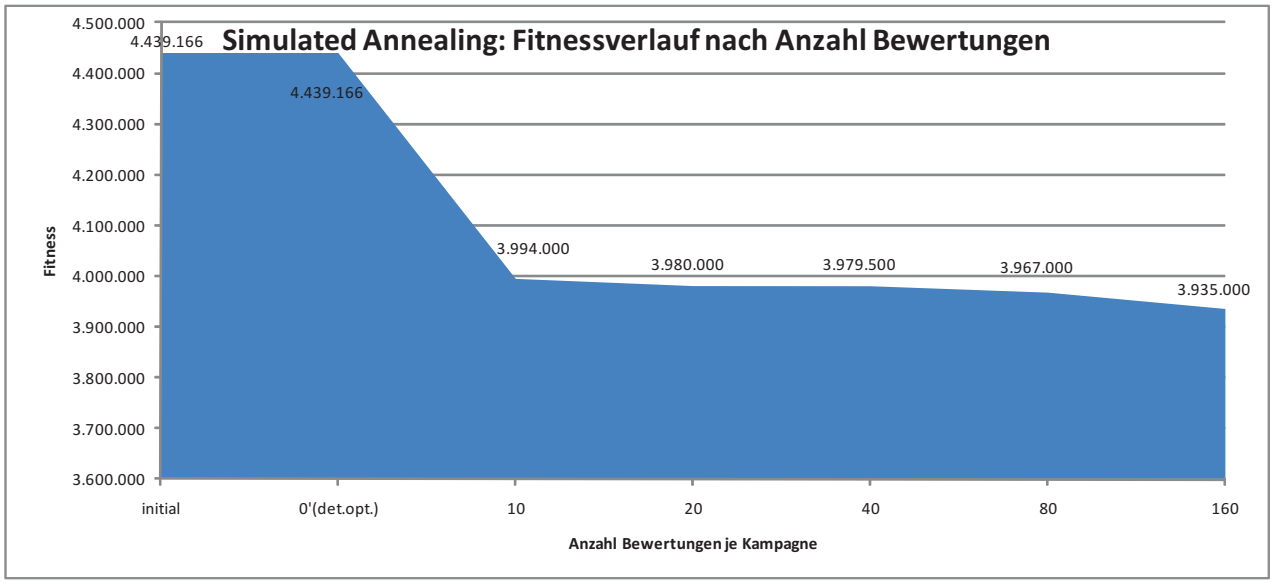


Abb. 7.7: Fitnessverlaufskurve für Datensatz 3

7.4.2 Ergebnisse bei Deaktivierung von Nebenbedingung 3

Datensatz 3 erreicht sein bestes Einzelergebnis mit der Parametrisierung hybrider gentischer Algorithmus 600, welches einen Fitnesswert von 3.771.466 erzielt. Dies entspricht einem gesamten Kostenreduktionspotenzial von 15,04%. Dahinter reihen sich diverse kleinere Parametrisierungen mit jeweils schlechteren Ergebnissen ein. Jedenfalls profitiert dieser Datensatz deutlich durch den Wegfall von Nebenbedingung 3, das zusätzliche Kostenreduktionspotenzial beträgt ca. 4%.

7.5 Simulationsergebnisse auf Datensatz 4

Datensatz 4 (Simulation.xls) ist der zeitlich betrachtet älteste Datensatz und besteht aus 32 Kampagnen. Der Betrachtungszeitraum liegt zwischen dem 20.3.2008 (08:05:07) und dem 1.5.2008 (14:28:47)

Der Fitnesswert der Ausgangssequenz aller 32 Kampagnen beträgt: 4.019.939

Die Simulationsdauer bzw. die benötigte Gesamtdurchlaufzeit der Ausgangreihenfolge beträgt: 42 Tage, 5 Stunden, 23 Minuten, 40 Sekunden

Spezifische Charakteristika dieses Datensatzes: Dieser Datensatz besteht aus 9 Kampagnen mit jeweils einem Los sowie 8 deterministisch zu optimierenden Kampagnen, damit verbleiben 15 Kampagnen für die metaheuristische Optimierung. Außerdem besteht der Datensatz aus insgesamt 12 kleineren und größeren ZSG Losen, welche sich über insgesamt 8 Kampagnen erstrecken. Demnach ist der Einfluss der Änderung der Zielsägen im Rahmen der gesamten Optimierung beachtlich. Die metaheuristischen Kampagnen haben eine mittlere Größe von ca. jeweils 12,60 Losen. (218 Lose$-(\ 9{\cdot}1 + 4{\cdot}2 + 4 \cdot 3) = 189$ Lose / 15 Kampagnen $= 12,60$ Lose je heuristisch zu optimierender Kampagne) Der Datensatz ist durch diese verhältnismäßig großen Kampagnen schwerer zu optimieren. Die Kampagnen 2,4,5,6,12,19 profitieren von dem Einsatz von Toleranzen ein wenig.

7.5.1 Fitnessverlauf von Simulated Annealing auf diesem Datensatz unter realen Bedingungen

Im Rahmen der Ergebnisse ist auffällig, dass trotz der Möglichkeit von 8 deterministisch zu optimierenden Kampagnen das Optimierungspotenzial relativ gering ausfällt. Die beste Reihenfolge mit einem Fitnesswert von 3.864.000 bringt ein Verbesserungspotenzial von $3,88\%$ gegenüber der Ausgangsreihenfolge, siehe Abb. 7.8. dazu.

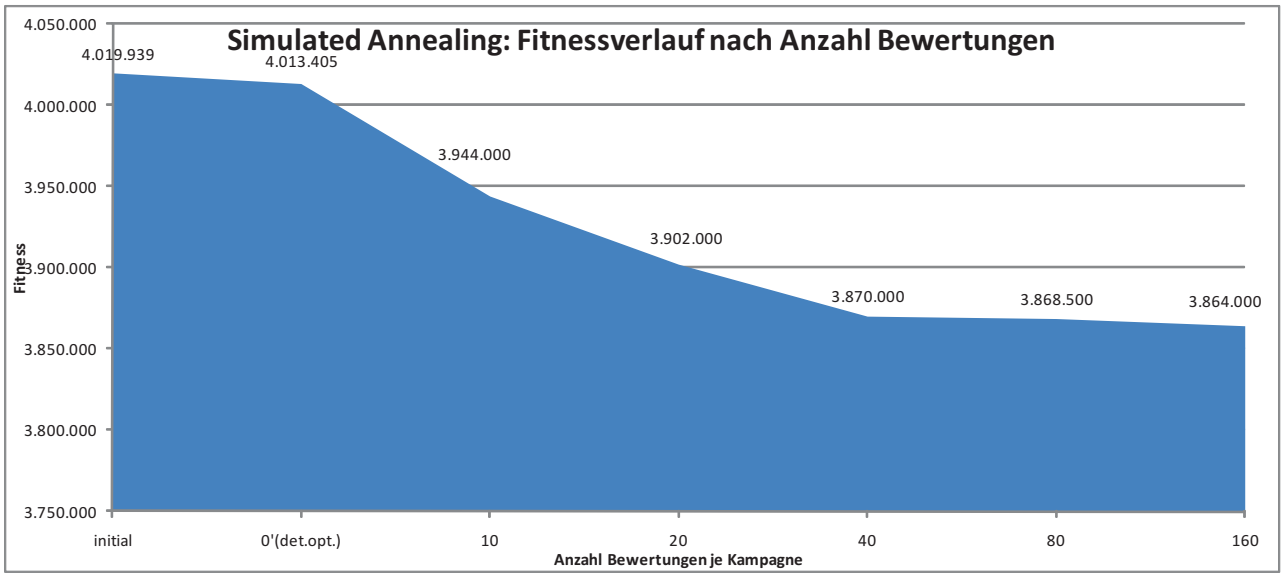


Abb. 7.8: Fitnessverlaufskurve für Datensatz 4

7.5.2 Ergebnisse bei Deaktivierung von Nebenbedingung 3

Mittels Simulated Annealing Parametrisierung 320 wurden hier 3.851.697 erreicht, eine umfangreichere hybride genetische Parametrisierung 600 erzielt 3.849.084. Damit fällt das zusätzliche Kostenreduktionspotenzials dieses Datensatzes außerordentlich gering aus. Dieses beträgt insgesamt 4,25% und stellt somit ein zusätzliches Potenzial in der Höhe von ca. 0,37% dar.

7.6 Simulationsergebnisse auf Datensatz 5

Datensatz 5 (Simulation20090904917_geändert_rsteri.xls) ist der zeitlich betrachtet ein aktueller Datensatz und besteht aus 27 Kampagnen. Der Betrachtungszeitraum liegt zwischen dem 4.9.2009 (10:56:57) und dem 19.10.2009 (22:25:40)

Der Fitnesswert der Ausgangssequenz aller 27 Kampagnen beträgt: 4.144.847

Die Simulationsdauer bzw. die benötigte Gesamtdurchlaufzeit der Ausgangsreihenfolge beträgt: 45 Tage, 11 Stunden, 28 Minuten, 43 Sekunden

Spezifische Charakteristika dieses Datensatzes: Dieser Datensatz besteht aus 8 Kampagnen mit jeweils einem Los sowie 4 deterministisch zu optimierenden Kampagnen, damit verbleiben 15 Kampagnen für die metaheuristische Optimierung. Außerdem besteht der Datensatz aus insgesamt 2 kleineren ZSG Losen sowie einem großen ZSG Los, welche sich über insgesamt 2 Kampagnen erstrecken. Demnach ist der Einfluss der Änderung der Zielsägen im Rahmen der gesamten Optimierung gering. Die metaheuristischen Kampagnen haben eine mittlere Größe von ca. jeweils 9, 80 Losen. (164 Lose−(8·1 + 3·2 + 1 · 3) = 147 Lose / 15 Kampagnen = 9, 80 Lose je heuristisch zu optimierende Kampagne) Damit ist dieser Datensatz, verglichen mit anderen Datensätzen, relativ leicht zu optimieren. 8 Kampagnen (3,4,7,8,9,10,18,22) profitieren teils mehr, teils weniger, von dem Einsatz von Toleranzen auf diesem Datensatz.

7.6.1 Fitnessverlauf von Simulated Annealing auf diesem Datensatz unter realen Bedingungen

Im Rahmen der Ergebnisse ist auffällig, dass trotz der Möglichkeit von 8 deterministisch zu optimierenden Kampagnen das Optimierungspotenzial relativ gering ausfällt. Die beste Reihenfolge bringt bei einem Fitnesswert von ca. 4.002.000 ein Verbesserungspotenzial von 3,45% gegenüber der Ausgangsreihenfolge, siehe auch Abb. 7.9.

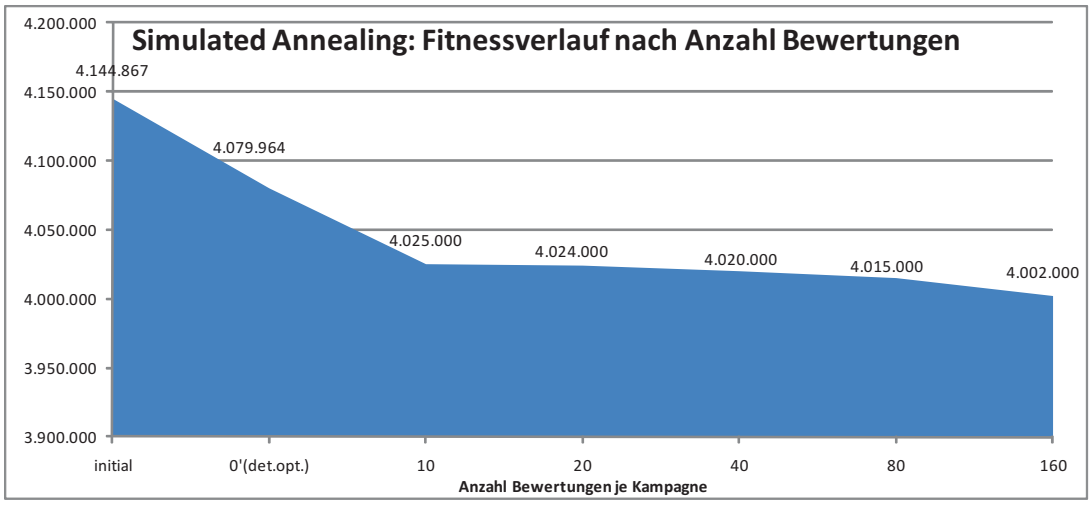


Abb. 7.9: Fitnessverlaufskurve für Datensatz 5

7.6.2 Ergebnisse bei Deaktivierung von Nebenbedingung 3

Das beste Ergebnis wurde mit der Parametrisierung hybrider genetischer Algorithmus 700 erzielt, dabei wurde ein Fitnesswert von 3.874.628 erreicht. Dies entspricht einem gesamten Kostenreduktionspotenzial von 6,62% bzw. einem durch den Wegfall der Nebenbedingung zusätzlichem Potenzial von 3,08%. Dieses zusätzliche Potenzial ist, verglichen mit den 3,45%, die unter realen Bedingungen möglich sind, beachtlich.

Hinter diesem Resultat reihen sich diverse schlechtere Ergebnisse im Rahmen von weniger umfangreichen Parametrisierungen ein.

7.7 Simulationsergebnisse auf Datensatz 6

Datensatz 6 (Simulation20090909090933.xls) ist zeitlich betrachtet ein aktueller Datensatz und besteht aus 27 Kampagnen. Der Betrachtungszeitraum liegt zwischen dem 19.9.2009 (09:33:30) und dem 22.10.2009 (19:16:12)

Der Fitnesswert der Ausgangssequenz aller 27 Kampagnen beträgt: 4.031.680

Die Simulationsdauer bzw. die benötigte Gesamtdurchlaufzeit der Ausgangsreihenfolge beträgt: 43 Tage, 9 Stunden, 42 Minuten, 42 Sekunden

Spezifische Charakteristika dieses Datensatzes: Dieser Datensatz besteht aus 6 Kampagnen mit jeweils einem Los sowie 5 deterministisch zu optimierenden Kampagnen, damit verbleiben 16 Kampagnen für die metaheuristische Optimierung. Außerdem besteht der Datensatz aus insgesamt 4 kleineren und größeren ZSG Losen, welche sich über insgesamt 3 Kampagnen erstrecken. Der Einfluss der Optimierung der Zielsägen ist daher auf diesem Datensatz eher gering. Die metaheuristischen Kampagnen haben eine mittlere Größe von ca. jeweils 8,88 Losen. (160 Lose−(6·1 + 3·2 + 2 · 3) = 142 Lose / 16 Kampagnen = 8,875 Lose je heuristisch zu optimierende Kampagne) Daher ist dieser Datensatz von der durchschnittlichen Kampagnangröße aus betrachtet eher leicht zu optimieren. 5 Kampagnen (3,4,6,14,18) profitieren geringfügig von dem Einsatz von Toleranzen auf diesem Datensatz.

7.7.1 Fitnessverlauf von Simulated Annealing auf diesem Datensatz unter realen Bedingungen

Das Optimierungspotenzial im Rahmen der 5 deterministisch zu optimierenden Kampagnen fällt außerordentlich gering aus bzw. ist vernachlässigbar.

Im Rahmen der Kurve ist auffällig, dass der Großteil des Potenzials bereits nach sehr wenigen Bewertungen gefunden werden kann, der beste erreichte Mittelwert beträgt ca. 3.923.000. Dies entspricht einem Kosteneinsparungspotenzial von ca. 2,7%. Einige Einzelergebnisse erreichen auch Werte um die 3.920.000. Dieser Datensatz lässt sich scheinbar sehr schnell optimieren, da bereits nach sehr wenigen Bewertungen je Kampagne der größte Teil optimiert worden ist. Dies lässt auch auf diesem Datensatz darauf schließen, dass nur sehr wenige Positionen in Bezug auf die Ausgangsreihenfolge (positiv) verändert werden müssen, wodurch längere Optimierungsläufe auch hier wenig Sinn ergeben würden.

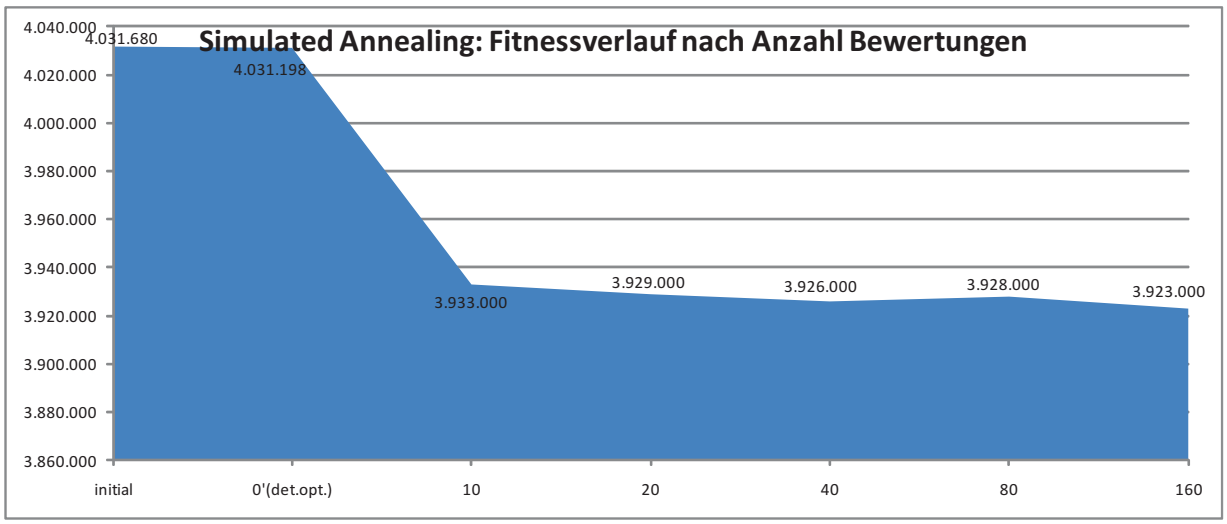


Abb. 7.10: Fitnessverlaufskurve für Datensatz 6

7.7.2 Ergebnisse bei Deaktivierung von Nebenbedingung 3

Im Rahmen der Parametrisierung hybrider genetischer Algorithmus 700 wurde das beste Einzelergebnis erreicht, welches einen Fitnesswert von 3.763.396 erreicht. Dies entspricht einem gesamten Kostenreduktionspotenzial von 6,65% bzw. einem durch den Wegfall von Nebenbedingung 3 zusätzlichem Potenzial von beachtlichen 3,95%. Dieser Datensatz zeigt damit, dass, obwohl der Einsatz von Toleranzen nicht vielversprechend ist, der komplette Wegfall von Nebenbedingung 3 hingegen einen deutlichen Unterschied aufweist.

7.8 Schlussfolgerungen aus den Datensätzen

Obwohl der Suchraum in einigen Kampagnen, welche z.B. aus einer Güteklasse mit gleich langen Schienen bestehen, riesig ist, genügen bei allen Datensätzen wenige hundert Bewertungen um bereits das Optimierungspotenzial im Rahmen des Einsatzes von Metaheuristiken ausschöpfen zu könnnen. Viele Kampagnen, v.a. auch solche mit relativ wenigen Losen, haben durch die Nebenbedingungen einen Suchraum, der kleiner oder nur wenig größer ist, von daher ist es verständlich, dass diese Kampagnen bereits nach derart wenigen Bewertungen optimiert sind. Die Wahl des Algorithmus scheint keine besondere Bedeutung zu haben, es hat sich jedoch herausgestellt, dass der hybride genetische Algorithmus und Simulated Annealing sehr gute Ergebnisse bringen. Der ACO Vertreter sowie der EA bringen auch annähernd gute Ergebnisse wie der HGA, lediglich die iterierte lokale Suche halte ich auf Grund des zu lokalen Suchverhaltens für nicht vielversprechend für die Optimierung.

Unter realen Bedingungen sind durchschnittlich, also über 6 Datensätze gemittelt, $5,69\%$ zu optimieren, wobei der konkrete Wert stark datensatzabhängig ist. Während auf einem Datensatz lediglich $2,7\%$ optimiert werden konnten, wurden auf einem anderen Datensatz $11,4\%$ erreicht. Weiters schränken die Nebenbedingungen auch jeden Datensatz bzw. jede Kampagne innerhalb eines Datensatzes nicht gleichmäßig sondern unterschiedlich stark ein.

In diesem Zusammenhang muss erläutert werden, dass der niedrigere Fitnesswert v.a. aus einer (i.d.R.) proportional dazu kürzeren Durchlaufzeit resultiert sowie aus besseren Pufferständen, v.a. von Railman 1 und Railman 2. Eine kürzere Durchlaufzeit ermöglicht eine bessere Termintreue. Die Stillstandszeiten des Walzwerks werden dabei durch eine optimierte Einlastung weiter minimiert.

Im Rahmen der Deaktivierung von Nebenbedingung 3, 'lang vor kurz' , ist der Suchraum bei einigen Kampagnen deutlich vergrößert worden, bei anderen Kampagnen fallen die Unterschiede wiederum relativ gering aus. Als Durchschnittswert über alle 6 Datensätze erhält man ein zusätzliches Potenzial von ca. $2,12\%$. Dabei ist je nach Datensatz ein zusätzliches Potenzial zwischen $0,1\%$ und 5% möglich. In diesem Zusammenhang muss angemerkt werden, dass abgesehen von den 600 und 700 Parametrisierungen keine umfangreicheren Testläufe durchgeführt worden sind. Einzelergebnisse können weit höher ausfallen, wenn man genügend viele Läufe macht. Prinzipiell ist bei intensiven Durchläufen ein durchschnittliches Potenzial von bis zu 3% möglich. Das zusätzliche Optimierungspotenzial fällt im Vergleich mit dem dadurch teils stark vergrößerten Suchraum relativ gering aus. Abgesehen davon spiegeln diese Bedingungen zwar eine mögliche Variante des Produktionsablaufs im Rahmen von produktionsbedingten Änderungen dar, allerdings empfehle ich diese Variante auf Grund des relativ geringen zusätzlichen Potenzials nicht.

7.9 Vergleich der Algorithmen anhand der Optimierung einer einzelnen Kampagne

Auf Grund der einzelnen nicht voneinander unabhängigen Kampagnenoptima lassen sich die einzelnen Algorithmen sehr schwer vergleichen, v.a. bei einem Suchraum, der relativ flach ist und in Bezug auf die Ausgangslösung oft nur wenige Positionen in einer Kampagne verändert werden müssen, um bereits den Großteil der jeweiligen Kampagne zu optimieren. Außerdem ist der Suchraum generell

stark eingeschränkt auf Grund der Nebenbedingungen. Daher werden die Algorithmen anhand der Optimierung einer einzelnen Kampagne verglichen, wobei die anderen Kampagnen fixiert werden. Konkret werden auf Datensatz 2 die ersten 7 Kampagnen bewertet, wobei die ersten 6 Kampagnen fixiert und somit im Rahmen der Optimierung übersprungen werden. Da die 7. Kampagne im Rahmen des Teildatensatzes die letzte Kampagne darstellt und die vorhergehenden Kampagnen übersprungen werden, wird der Effekt, den eine abgeänderte Reihenfolge der vorangegangenen Kampagne auf die nächste Kampagne hat, ausgeschaltet. Die Abbildungen 7.11 bis 7.16 zeigen die Ergebnisse der einzelnen Algorithmen.

Vergleich 1: Kampagne 7 in Datensatz 2: Toleranzen spielen in dieser Kampagne, welche aus 9 Losen gleicher Schienenlänge (120 Meter Schienen besteht) keine Rolle, auf Grund derselben Qualitätsgüteklasse ("Normal") besitzt der Suchraum eine Größe von 9!. Der Suchraum scheint auf Grund der Beschaffenheit dieser Kampagne sehr homogen zu sein, die Fitnesslandschaft dürfte relativ flach sein.

Das Optimierungspotenzial dieser Kampagne ist in Anbetracht des relativ großen Suchraums sehr gering, die beste Reihenfolge mit einem Fitnesswert von 388.379 bringt eine Verbesserung gegenüber der Ausgangsreihenfolge um $0,34\%$.

Es wurden mehrere Läufe mit den Parametrisierungen 400 und 600 vorgenommen. Diese lauten:

Simulated Annealing 400: Anfangstemperatur: 12°C, Erstarrungstemperatur: 4.5°C, Abkühlungsrate: 0.95, 20 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 15, Alpha Nachbarschaftsgröße: 10, Nachbarschaftsänderungsfaktor: 0.65, Toleranz: 0%

Simulated Annealing 660: Anfangstemperatur: 12°C, Erstarrungstemperatur: 4.5°C, Abkühlungsrate: 0.97, 20 Wiederholungen je Temperaturzustand, Delta Skalierungsfaktor: 15, Alpha Nachbarschaftsgröße: 10, Nachbarschaftsänderungsfaktor: 0.65, Toleranz: 0%

Im Rahmen dieser Parametrisierung werden auch die anderen (kürzeren) Parametrisierungen abgedeckt, weiters wird neben der besten Lösung auch das Optimierungsniveau von Simulated Annealing angegeben, siehe dazu auch Tab. 7.1. Das Niveau kann sehr leicht über den Delta Skalierungsfaktor eingestellt werden, siehe dazu auch 6.3.6.

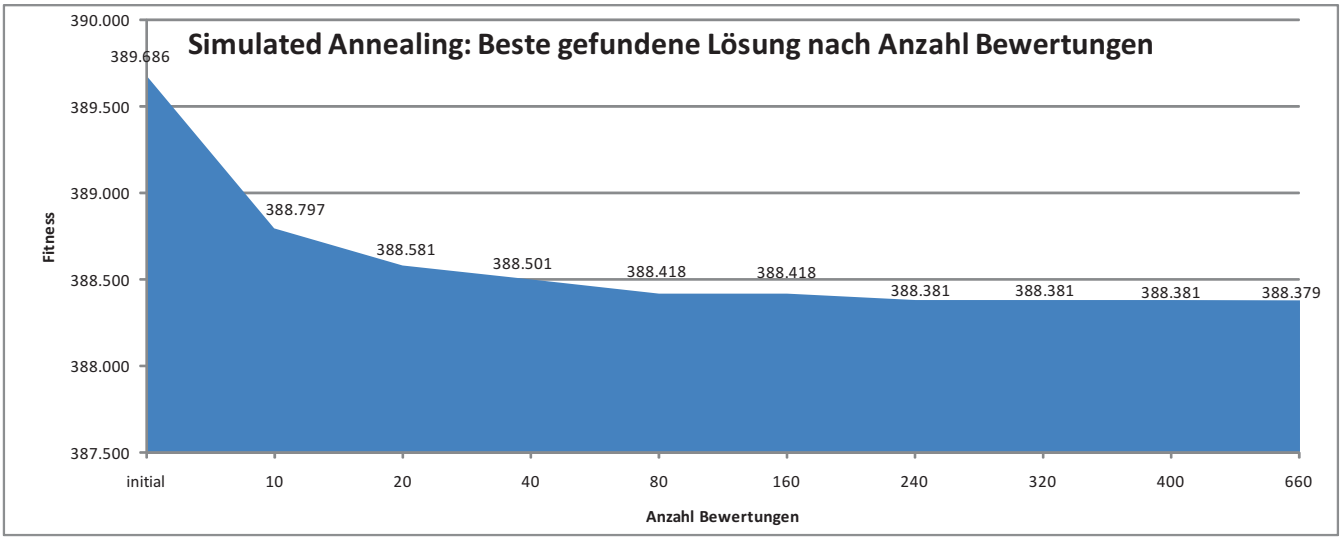


Abb. 7.11: Vergleich 1: Ergebnisse mittels Simulated Annealing

Parametrisierung SA	Optimierungsniveau SA	Beste gefundene Lösung mittels SA
initial	389.686	389.686
10	388.797	388.797
20	388.581	388.581
40	388.501	388.501
80	388.418	388.418
160	388.418	388.418
240	388.418	388.381
320	388.466	388.381
400	388.418	388.381
660	388.450	388.379

Tab. 7.1: Vergleich 1: Gegenüberstellung von Optimierungsniveau und bester gefundenen Lösung (SA)

Die hier verwendeten Parametrisierungen für den HGA lauten:

Parametrisierung 80: 10 Individuen, 10 Iterationen, Crossover Rate: 0.1, Mutations Rate: 0.95, Alpha Wert- lokale Suche: 20, Elitismus Individuen: 2, Anzahl Individuen für lokale Suche: 1

Parametrisierung 160: 13 Individuen, 13 Iterationen, Crossover Rate: 0.1, Mutations Rate: 0.95, Alpha Wert- lokale Suche: 20, Elitismus Individuen: 2, Anzahl Individuen für lokale Suche: 1

Parametrisierung 240: 16 Individuen, 16 Iterationen, Crossover Rate: 0.1, Mutations Rate: 0.95, Alpha Wert- lokale Suche: 20, Elitismus Individuen: 3, Anzahl Individuen für lokale Suche: 2

Die umfangreicheren Parametrisierungen unterscheiden sich ab Parametrisierung 240 lediglich in der Anzahl an verwendeten Individuen sowie benötigten Iterationen.

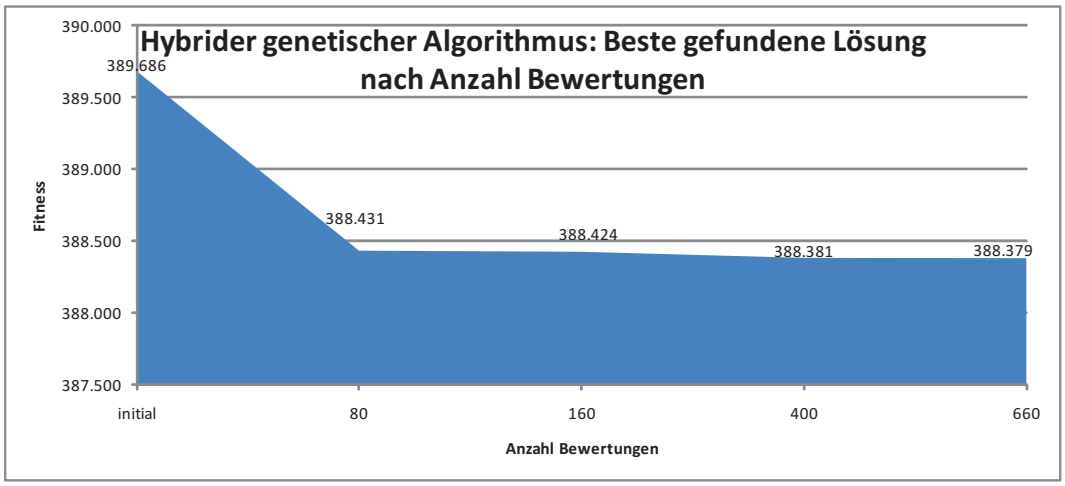


Abb. 7.12: Vergleich 1: Ergebnisse des hybriden genetischen Algorithmus

Wenn man die Abbildungen 7.11 und 7.12 vergleicht, sieht man, dass der genetische Algorithmus anfangs durch seine Individuenvielfalt an Performance und Konvergenz verliert, während er bei der größten (sinnvollen) Parametrisierung Simulated Annealing übertrifft. Simulated Annealing liefert v.a. in den kleineren Parametrisierungen sehr gute Ergebnisse. Deshalb wurde dieses Verfahren auch vorrangig für die weniger umfangreichen Testläufe verwendet.

Die hier verwendeten Parametrisierungen für den ACO (ACS) lauten:

Parametrisierung 80: 10 Ameisen, 10 Iterationen, Verdunstungsrate: 0.1, Resultate wie vieler Ameisen übernehmen: 2, Anzahl Ameisen für lokale Suche: 1, Alpha Wert- lokale Suche: 10

Parametrisierung 160: 13 Ameisen, 13 Iterationen, Verdunstungsrate: 0.1, Resultate wie vieler Ameisen übernehmen: 2, Anzahl Ameisen für lokale Suche: 1, Alpha Wert- lokale Suche: 10

Parametrisierung 240: 16 Ameisen, 16 Iterationen, Verdunstungsrate: 0.1, Resultate wie vieler Ameisen übernehmen: 3, Anzahl Ameisen für lokale Suche: 2, Alpha Wert- lokale Suche: 10

Die umfangreicheren Parametrisierungen unterscheiden sich ab Parametrsierung 240 lediglich in der Anzahl an verwendeten Ameisen und Iterationen.

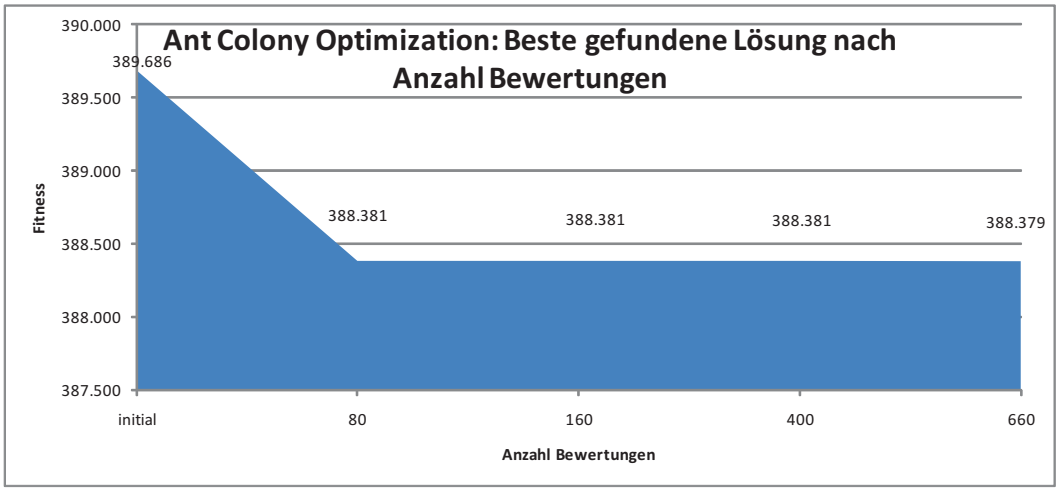


Abb. 7.13: Vergleich 1: Ergebnisse des Ant Colony System Vertreters

Der ACO (Abb. 7.13) ist deshalb in der Lage, sich relativ frühzeitig eine sehr gute Lösung zu erzeugen, weil sich jede Ameise ihre Auftragsreihenfolge neu zusammenstellt. Dadurch kann bereits eine einzelne Ameise eine Lösung generieren, welche komplett von der Ausgangsreihenfolge abweicht. Der genetische Algorithmus benötigt hingegen mehrere Iterationen, um in der Lage zu sein, mehrere zielführende Mutationen in Serie durchführen zu können, damit schließlich nach mehreren Mutationen die Individuen eine ähnlich hohe Qualität an Lösungen zu erreichen.

Generell bleibt in diesem Zusammenhang anzumerken, dass sich der ACO speziell für Kampagnen, wo viele Änderungen in Bezug auf die Ausgangsreihenfolge notwendig sind, sehr gut bzw. besser eignet als z.B. ein genetischer Algorithmus.

Die hier verwendeten Parametrisierungen für den EA, dessen Ergebnisse in Abb. 7.14 ersichtlich sind, im Detail:

EA Parametrisierung 80: 10 Iterationen, Expansion Fraction: 1, Selection Fraction: 1, Alpha Value-Local Search: 10, Anzahl Individuen für lokale Suche: 1, 0% Toleranz

Die weiteren verwendeten Parametrisierungen benötigen entsprechend mehr Iterationen.

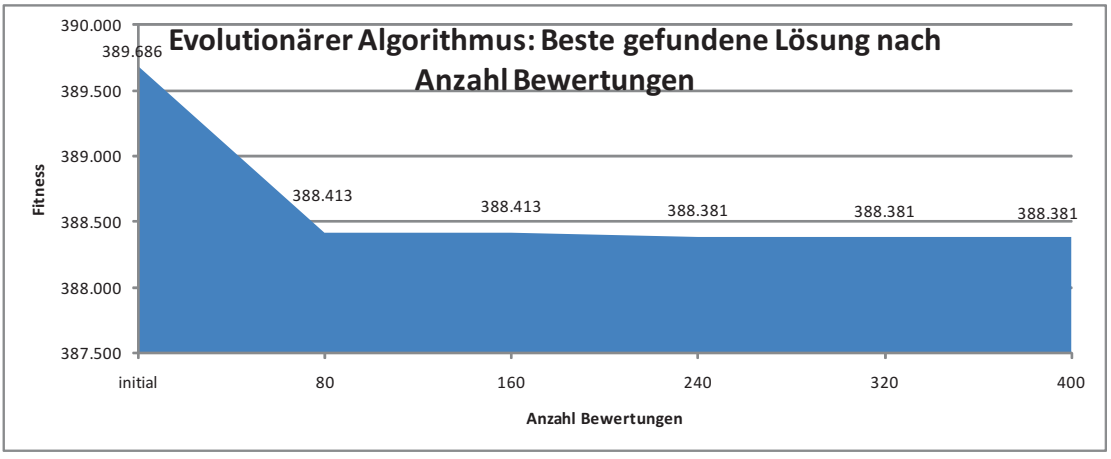


Abb. 7.14: Vergleich 1: Ergebnisse des evolutionären Algorithmus

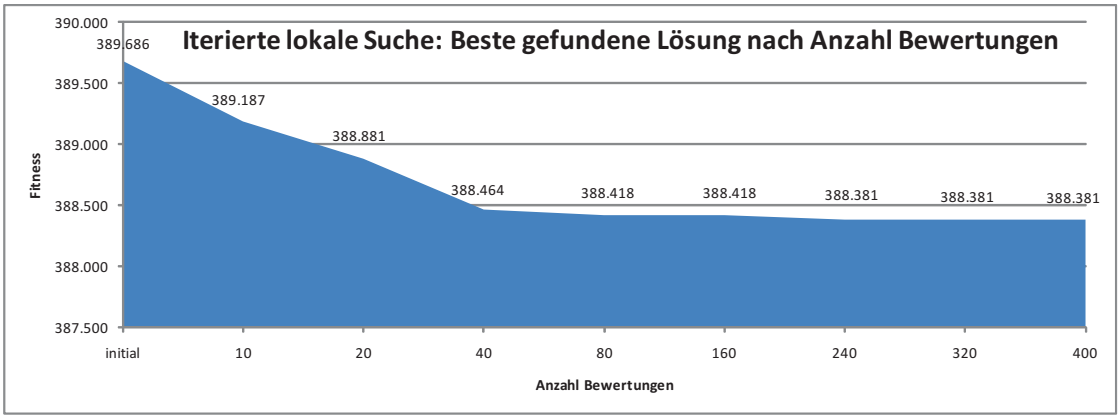


Abb. 7.15: Vergleich 1: Ergebnisse mittels iterierter lokaler Suche

ILS verwendert die Parametrisierung SA 400, um zu den Ergebnissen, wie sie in Abb. 7.15. dargestellt sind, zu gelangen.

Wie man an den Abbildungen erkennt, liegen die Ergebnisse der Algorithmen allesamt sehr eng beisammen. Dies liegt jedoch auch am Datensatz selbst, der aus sehr ähnlichen Losen besteht, welche großteils eine gleiche bzw. ähmliche Anzahl an Blöcken haben. Dies verursacht bei einem Tausch z.B. gleich viele Schnitte, ggf. dazu noch an der selben Sägebohrlinie. Auf Grund dessen ist dieser Datensatz relativ einfach und schnell zu optimieren, wie auch die Ergebnisse der einzelnen Algorithmen zeigen.

Vergleich 2: Kampagne 11 in Datensatz 4: Daher dient eine weitere Kampagne (Nummer 11) eines anderen Datensatzes (Simulation.xls) mit einem riesigen Suchraum als Basis für einen besseren, weil ausführlicheren, Vergleich der einzelnen Algorithmen, siehe auch Abb. 7.16 für die Kampagne. Die besten Ergebnisse wurden im Rahmen eines hybriden genetischen Algorithmus erreicht, welcher die Ausgangsreihenfolge um $2,76\%$ verbessert und dabei einen Fitnesswert von 1.861.205 erreicht.

Aufträge | Aggregate | Pufferspeicher

12.01.2010 16:07:09 Bewertung
Simulationsbeginn: [redacted]
Simulationsende: 10.04.2008 09:11:39
Dauer: 21d 00:06:32
Fitness Value: 1913982,00

AscOrderId	Walzkampagne	LotId	#Blocks	Profil	Qualität	Schnittmuster	ZS	ZEK	Lieferdatum	Fertigstellungsdatum
92	10	120072-34	60	VI60E1	HSH	2x60000	ZS2	EK4	14.04.2008	08.04.2008
93	11	120277-8	10	VI49E1	Normal	6x18000	ZS1	EK1	15.04.2008	08.04.2008
94	11	120277-8_	10	VI49E1	Normal	6x18000	ZS2	EK2	15.04.2008	08.04.2008
95	11	120277-8__	10	VI49E1	Normal	6x18000	ZS1	EK1	15.04.2008	08.04.2008
96	11	120277-8___	10	VI49E1	Normal	6x18000	ZS2	EK2	15.04.2008	08.04.2008
97	11	120277-8_...	10	VI49E1	Normal	6x18000	ZS1	EK1	15.04.2008	08.04.2008
98	11	120277-8_...	10	VI49E1	Normal	6x18000	ZS2	EK2	15.04.2008	08.04.2008
99	11	120277-8_...	10	VI49E1	Normal	6x18000	ZS1	EK1	15.04.2008	08.04.2008
100	11	120632-1	4	VI49E1	Normal	9x12394	ZS2	EK2	07.04.2008	08.04.2008
101	11	120393-1	10	VI49E1	Normal	6x20000	ZS1	EK1	01.04.2008	08.04.2008
102	11	120393-1_	10	VI49E1	Normal	6x20000	ZS2	EK2	01.04.2008	08.04.2008
103	11	120393-1__	10	VI49E1	Normal	6x20000	ZS1	EK1	01.04.2008	08.04.2008
104	11	120393-1___	10	VI49E1	Normal	6x20000	ZS2	EK2	01.04.2008	08.04.2008
105	11	120393-1_...	10	VI49E1	Normal	6x20000	ZS1	EK1	01.04.2008	08.04.2008
106	11	120393-1_...	17	VI49E1	Normal	6x20000	ZS2	EK2	01.04.2008	08.04.2008
107	11	120618-1	9	VI49E1	Normal	3x36000	ZS1	EK1	02.04.2008	08.04.2008
108	11	120407-5	10	VI49E1	Normal	8x15000	ZS2	EK2	14.04.2008	08.04.2008
109	11	120407-5_	10	VI49E1	Normal	8x15000	ZS1	EK1	14.04.2008	08.04.2008
110	11	120407-5__	10	VI49E1	Normal	8x15000	ZS2	EK2	14.04.2008	08.04.2008
111	11	120407-5___	10	VI49E1	Normal	8x15000	ZS1	EK1	14.04.2008	08.04.2008
112	11	120407-5_...	11	VI49E1	Normal	8x15000	ZS2	EK2	14.04.2008	09.04.2008
113	11	120622-2	13	VI49E1	HSH	2x60000	ZS2	EK4	05.05.2008	09.04.2008
114	11	120622-1	8	VI49E1	HSH	3x40000	ZS1	EK1	21.04.2008	08.04.2008
115	11	120536-12	6	VI49E1	HSH	4x30000	ZS2	EK2	01.04.2008	09.04.2008
116	11	120634-1	6	VI49E1	HSH	4x30000	ZS1	EK1	02.04.2008	08.04.2008
117	11	120536-14	7	VI49E1	HSH	4x30010	ZS2	EK4	01.04.2008	09.04.2008
118	11	120665-2	9	VI49E1	HSH	4x30000	ZS1	EK1	07.04.2008	08.04.2008
119	11	119772-4	28	VI49E1	HSH	4x30000	ZS2	EK2	01.04.2008	09.04.2008
120	11	119772-3	26	VI49E1	HSH	4x30000	ZS1	EK1	01.04.2008	09.04.2008
121	11	120598-2	53	VI49E1	HSH	4x25000	ZS2	EK2	01.04.2008	10.04.2008
122	11	120536-4	4	VI49E1	HSH	8x15000	ZS1	EK1	01.04.2008	09.04.2008
123	11	120277-7	10	VI49E1	HSH	6x18000	ZS2	EK2	15.04.2008	09.04.2008
124	11	120277-7_	10	VI49E1	HSH	6x18000	ZS1	EK1	15.04.2008	09.04.2008
125	11	120277-7__	10	VI49E1	HSH	6x18000	ZS2	EK2	15.04.2008	10.04.2008
126	11	120277-7___	10	VI49E1	HSH	6x18000	ZS1	EK1	15.04.2008	09.04.2008
127	11	120277-7_...	10	VI49E1	HSH	6x18000	ZS2	EK2	15.04.2008	10.04.2008
128	11	120277-7_...	10	VI49E1	HSH	6x18000	ZS1	EK1	15.04.2008	09.04.2008
129	11	120452-1	6	VI49E1	HSH	4x25000	ZS2	EK2	09.04.2008	10.04.2008
130	11	120536-1	15	VI49E1	HSH	8x15000	ZS1	EK1	01.04.2008	09.04.2008
131	11	120539-1	9	VI49E1	HSH	8x15000	ZS2	EK2	09.04.2008	10.04.2008
132	11	120537-1	18	VI49E1	HSH	8x15000	ZS1	EK1	22.04.2008	10.04.2008
133	11	undefAA	61	VI49E1	Normal	1x120000	ZS2	EK4	20.03.2008	10.04.2008

Abb. 7.16: Kampagne 11 (Simulation.xls) wird für Vergleich 2 der Algorithmen verwendet

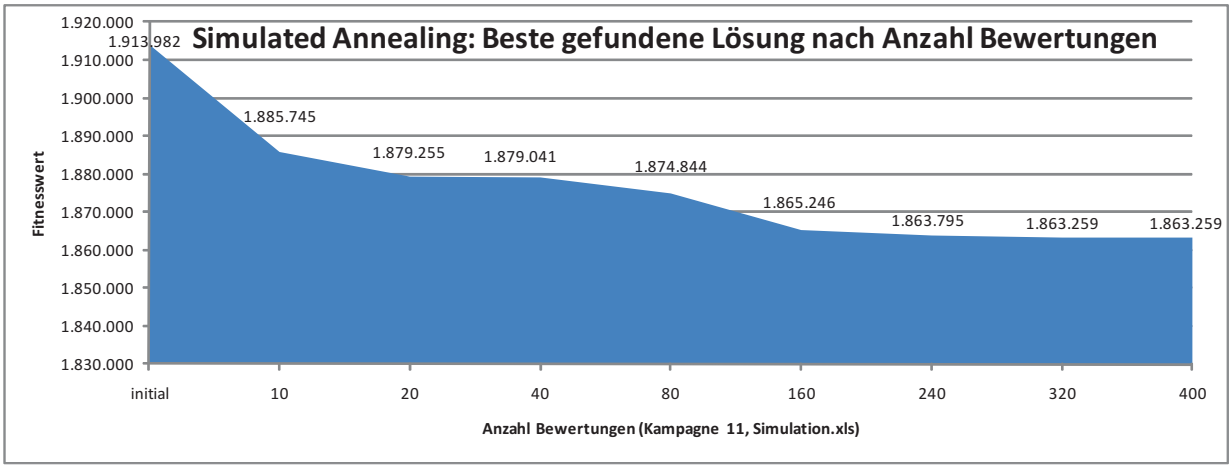


Abb. 7.17: Vergleich 2: Ergebnisse mittels Simulated Annealing

Auf Basis der Parametrisierung 400 wurden die kürzeren Parametrisierungen abgelesen und in Abb. 7.17 aufgetragen, siehe dazu auch Tab. 7.2.

Parametrisierung SA	Optimierungsniveau SA	Beste gefundene Lösung mittels SA
initial	1.913.982	1.913.982
10	1.885.745	1.885.745
20	1.879.255	1.879.255
40	1.879.041	1.879.041
80	1.874.844	1.874.844
160	1.865.317	1.865.246
240	1.863.904	1.863.795
320	1.863.259	1.863.259
400	1.863.355	1.863.259

Tab. 7.2: Vergleich 2: Gegenüberstellung von Optimierungsniveau und bester gefundenen Lösung (SA)

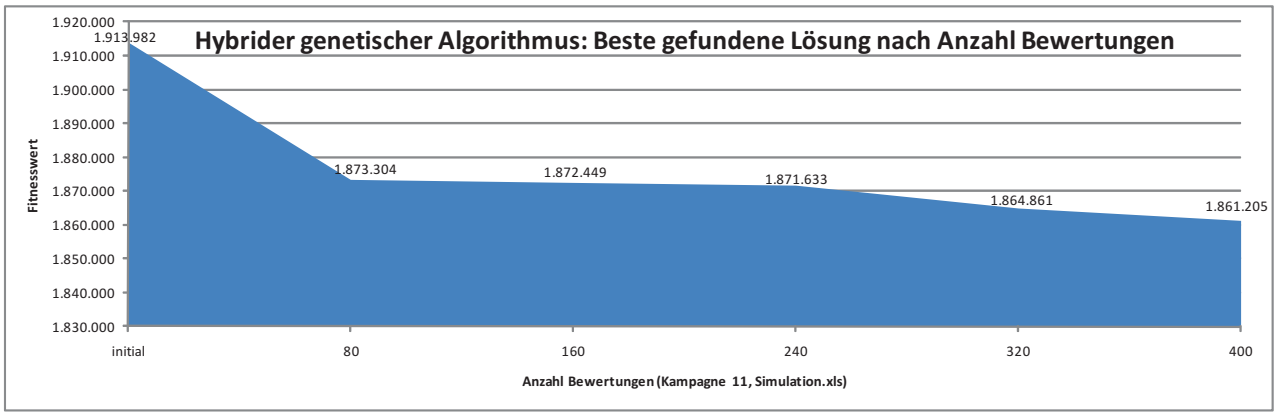


Abb. 7.18: Vergleich 2: Ergebnisse des hybriden genetischen Algorithmus

Die Ergebnisse des hybriden genetischen Alorithmus sind im gesamten Optimierungsverlauf als sehr gut im Vergleich mit allen anderen Algorithmen einzustufen.

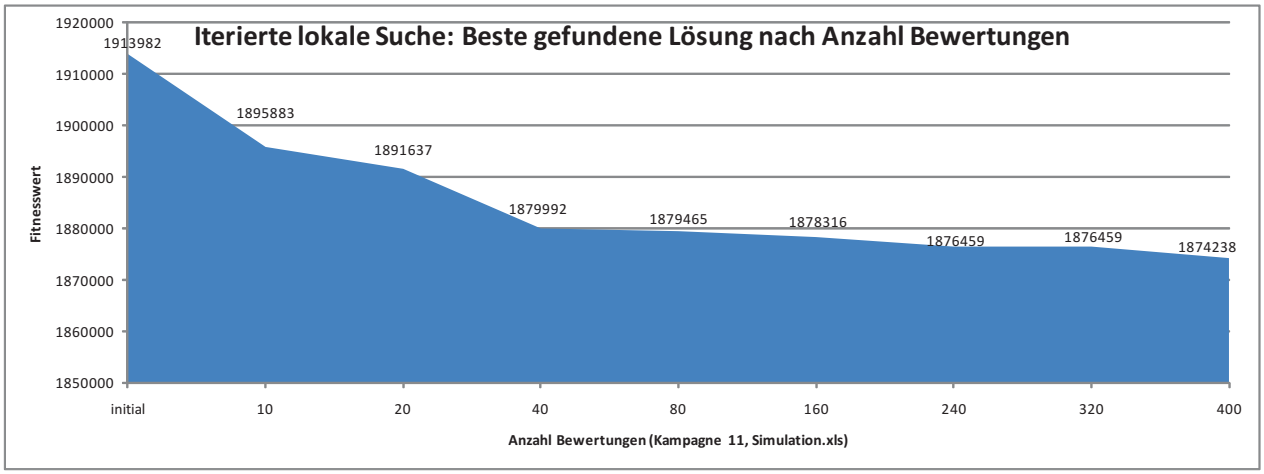


Abb. 7.19: Vergleich 2: Ergebnisse mittels iterierter lokaler Suche

Die Ergebnisse von iterierter lokaler Suche sind mit Abstand die schlechtesten Ergebnisse auf diesem Datensatz. Wie erwartet, hat sich der Algorithmus häufig in relativ schlechte Nachbarschaften hineinperturbiert, die im Laufe der Optimierung keine wesentlich besseren Ergebnisse mehr bringen. Die Ergebnisse wurden im Rahmen von 2 seed Werten der Parametrisierungen 320 und 400 gemittelt, lediglich die Ergebnisse der Punkte 320 und 400 stellen Einzelergebnisse dar. Der Vollständigkeit halber sind die beiden verwendeten Parametrisierungen gelistet:

ILS Parametrisierung 320: Ausgangstemperatur 12.0°C, Erstarrungstemperatur: 4.5°C, Abkühlungsrate: 0.95, 20 Wiederholungen pro Temperaturzustand, Alpha- Nachbarschaftsgröße: 30, Delta Skalierungsfaktor: 15, Nachbarschafts Änderungs-Faktor: 0.7, 0% Toleranz

ILS Parametrisierung 400: Wie Parametrisierung 320, jedoch: Erstarrungstemperatur: 3.5°C, Nachbarschafts Änderungs-Faktor: 0.65

Der ACO liefert auf diesem Datensatz sehr schnell gute Ergebnisse (Abb. 7.20), allerdings kommt er nicht an die Spitzenergebnisse heran. Dies kann neben der Größe dieser Kamagne auch daran liegen, dass sich der ACO bestimmte Lösungen nicht erzeugen kann, siehe auch 6.3.5.1.

Außerdem kann bei einer derart riesigen Kampagne der Nachteil zum Tragen kommen, dass die Ameisen nicht explizit die guten Auftragspositionen vererben, sondern dies nur implizit über die Pheromone passiert und sich jede Ameise ihre Auftragsreihenfolge neu zusammenstellt, wodurch zwar sehr viele gute Lösungen entstehen, aber die beste(n) Lösung(en) nicht gefunden werden.

ACO Parametrisierung 80: 10 Iterationen, 8 Ameisen, Resultate wie vieler Ameisen übernehmen: 2, Verdunstungsrate: 0.1, Alpha: 0.9, Beta: 0.1, Alpha- Value: 0, Anzahl Ameisen für lokale Suche: 0, Toleranz: 0%

ACO Parametrisierung 160: 15 Iterationen, 10 Ameisen, Resultate wie vieler Ameisen übernehmen: 3, Verdunstungsrate: 0.1, Alpha: 0.9, Beta: 0.1, Alpha- Value: 1 , Anzahl Ameisen für lokale Suche: 1, Toleranz: 0%

ACO Parametrisierung 240: 24 Iterationen, 10 Ameisen, Resultate wie vieler Ameisen übernehmen: 3, Verdunstungsrate: 0.1, Alpha: 0.9, Beta: 0.1, Alpha- Value: 1 , Anzahl Ameisen für lokale Suche: 1, Toleranz: 0%

ACO Parametrisierung 320: 16 Iterationen, 20 Ameisen, Resultate wie vieler Ameisen übernehmen: 4, Verdunstungsrate: 0.06, Alpha: 0.9, Beta: 0.1, Alpha- Value: 2 , Anzahl Ameisen für lokale Suche: 2, Toleranz: 0%

ACO Parametrisierung 400: 16 Iterationen, 25 Ameisen, Resultate wie vieler Ameisen übernehmen: 4, Verdunstungsrate: 0.05, Alpha: 0.9, Beta: 0.1, Alpha- Value: 3 , Anzahl Ameisen für lokale Suche: 2, Toleranz: 0%

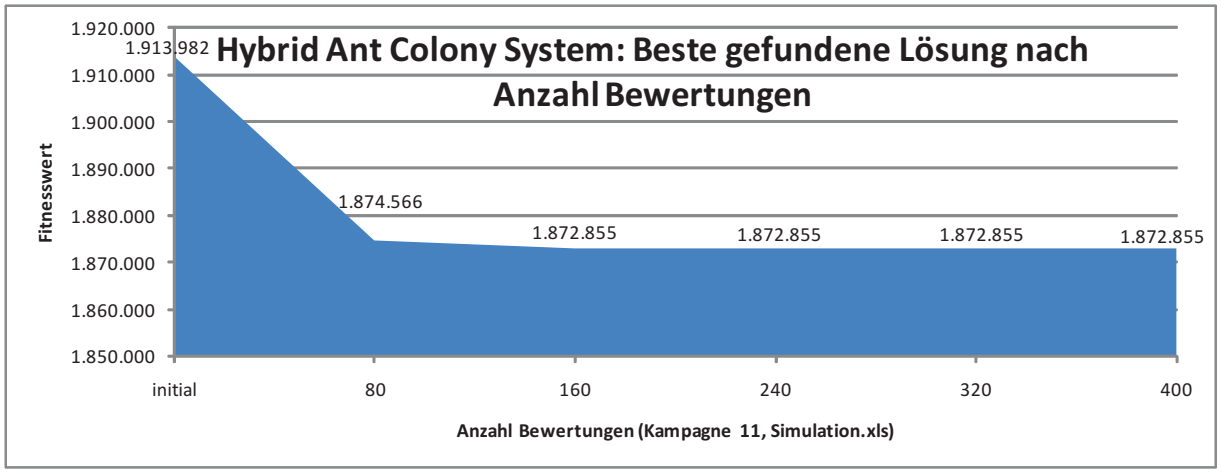


Abb. 7.20: Vergleich 2: Ergebnisse des Ant Colony System Vertreters

Der evolutionäre Algorithmus liefert auf diesem Datensatz schnell relativ gute Ergebnisse, wobei er allerdings keine Spitzenergebnisse wie der hybride genetische Algorithmus oder Simulated Annealing auf diesem Datensatz liefert. Die Ergebnisse sind in Abb. 7.21 ersichtlich. Die hier für den EA verwendeten Parametrisierungen lauten im Detail:

EA Parametrisierung 80: 4 Iterationen, Expansion Fraction: 4, Selection Fraction: 3, Alpha Value-Local Search: 1, Anzahl Individuen für lokale Suche: 1, 0% Toleranz

EA Parametrisierung 160: 6 Iterationen, Expansion Fraction: 3, Selection Fraction: 3, Alpha Value-Local Search: 2, Anzahl Individuen für lokale Suche: 1, 0% Toleranz

EA Parametrisierung 240: 6 Iterationen, Expansion Fraction: 2, Selection Fraction: 3, Alpha Value-Local Search: 3, Anzahl Individuen für lokale Suche: 1, 0% Toleranz

EA Parametrisierung 320: 8 Iterationen, Expansion Fraction: 2, Selection Fraction: 3, Alpha Value-Local Search: 4, Anzahl Individuen für lokale Suche: 1, 0% Toleranz

EA Parametrisierung 400: 10 Iterationen, Expansion Fraction: 2, Selection Fraction: 3, Alpha Value-Local Search: 5, Anzahl Individuen für lokale Suche: 1, 0% Toleranz

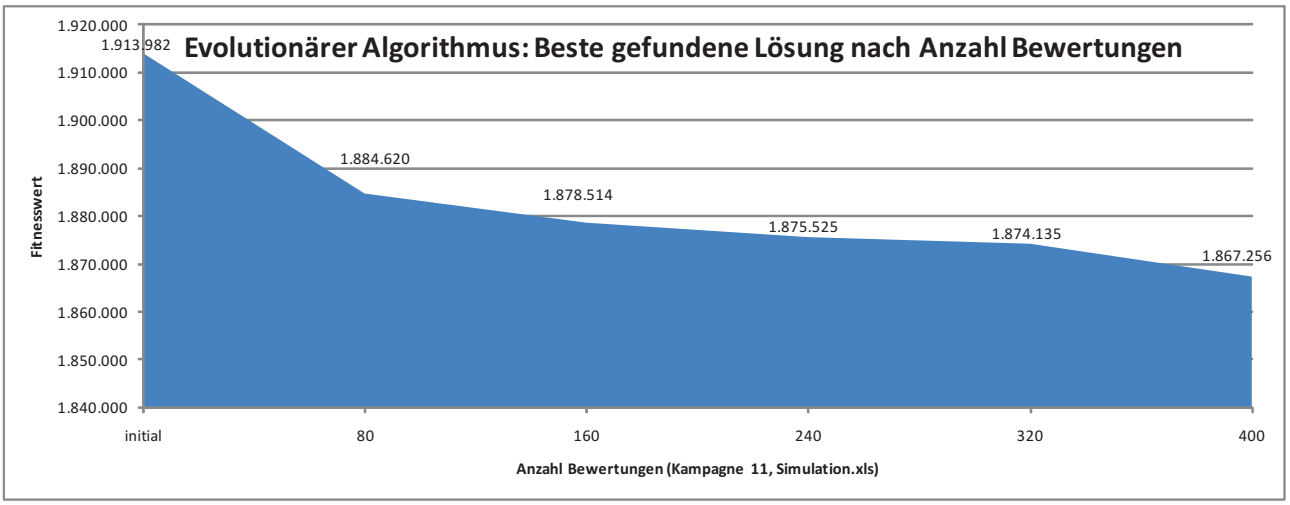


Abb. 7.21: Vergleich 2: Ergebnisse des evolutionären Algorithmus

Zusammenfassung der Ergebnisse im Rahmen des Vergleichs der Algorithmen anhand dieser beiden Kampagnen bzw. Datensätze: An diesen beiden komplett unterschiedlichen Kampagnen bzw. Datensätzen und damit verbundenen Suchräumen erkennt man zusammenfassend, dass die Algorithmen alle relativ schnell eine in Bezug auf den Fitnesswert sehr gute Lösung erreichen. Weiters ist die Gesamtperformance des hybriden genetischen Algorithmus als sehr gut einzustufen. Simulated Annealing liefert ebenfalls immer sehr gute Ergebnisse. Während der ACO auf dem 1. Datensatz (Kampagne mit wenigen Losen) sehr schnell sehr gute Ergebnisse liefert, eignet sich der 2. Datensatz weniger gut für die ameisenbasierte Optimierung.

Während eine um fast 2,8% bessere Auftragsreihenfolge (im Rahmen der Optimierung lediglich der Kampagne 11 auf dem Datensatz Simulation.xls) gefunden werden konnte, erreichte ILS ein Kostenreduktionspotenzial von bloß ca. 2,1%. Die Performance dieser Algorithmen bei einer ähnlich umfangreichen Parametrisierung unterscheidet sich damit um knapp 25%. Die übrigen Verfahren reihen sich zwischen dem HGA und ILS ein.

Allgemein hat sich in Bezug auf den 2. Datensatz herausgestellt, dass offensichtlich sehr wenige Lösungen unter 1.870.000 existieren und es auch vorkommen kann, dass derartige Lösungen vom genetischen Algorithmus bzw. von Simulated Annealing nicht erreicht werden müssen. Allerdings haben diese beiden Algorithmen im Durchschnitt trotzdem die besten Ergebnisse gezeigt.

Beim ersten Datensatz fallen die Unterschiede zwischen den Algorithmen verschwindend gering aus, lediglich der unterschiedlich rasche Fitnessverlauf ist teilweise signifikant unterscheidbar.

Wenn man davon ausgeht, dass eine Standard Kampagne über einen relativ kleinen Suchraum, in Vergleich mit Kampagne 11 aus Simulation.xls, verfügt, fallen die Unterschiede der Algorithmen im Rahmen einer Optimierung des gesamten Datensatzes deutlich geringer aus, als die Unterschiede, welche im Rahmen von Vergleich 2 festgestellt werden konnten.

7.10 Einfluss auf die Optimierung im Rahmen des Einsatzes von Toleranzen bei der Nebenbedingung 'lang vor kurz'

Der Einfluss der Toleranzen wurde in Datensatz 5 anhand einer Optimierung von 3 Kampagnen getestet, wobei die Kampagnen 1 und 2 fixiert worden sind. Damit wird nur die 3. Kampagne mittels verschiedenen Algorithmen optimiert. Alleine durch die Betrachtung von Kampagne 3 in Abb. 7.22 ist ersichtlich, dass bei Toleranzen von 20% in der Güteklasse Normal zusätzlich die Lose mit den Losnummern 12 mit 13 bzw. 13 mit 14 bzw. 14 und 15 sowie 17 mit 18 bzw. 18 mit 19 tauschen können. In der Güteklasse HSH können zusätzlich die Lose mit den Losnummern 25−27 beliebig durchtauschen, außerdem kann das Los mit der Losnummer 28 mit den Losen 31 bzw. 32 tauschen. Obwohl man davon ausgehen kann, dass eine Toleranz von 20% den Suchraum deutlich vergrößert, handelt es sich augenscheinlich um Tauschvorgänge von kürzeren Schienen (Spalte 7: Schnittmuster) mit relativ wenig Blöcken (Spalte 4: #Blöcke). Diese Tatsache ist allgemein zutreffend, da Lose mit langen 120 Meter Schienen erst bei 100% Toleranz mit Losen mit 60 Meter Schienen tauschen können. Da dies jedoch Ergebnisse liefert, welche der Endbenutzer nicht verwenden wird, wurden derartige Toleranzen nicht getestet. Anstelle dessen wurde für jeden Datensatz das zusätzliche Potenzial erhoben, welches sich durch die Deaktivierung von Nebenbedingung 3 ergibt, siehe dazu die jeweiligen Unterabschnitte 7.2 bis 7.7.

Aufträge | Aggregate | Pufferspeicher

07.01.2010 15:52:49 Bewertung

Simulationsbeginn: 04.09.2009 10:56:57
Simulationsende: 08.09.2009 18:23:55
Dauer: 4d 07:26:58
Fitness Value: 383078,00

AscO...	Wa...	LotId	#Bl...	Profil	Qualität	Schnittmu...	ZS	ZEK	Lieferdatum	Fertigstellungs...
0	1	121591-137	0	VI50E3	HSH	2x60000	ZS2	EK4	24.09.2009	Keine Daten v...
1	2	122858-2	0	VISAR57	Normal	2x60000	ZS1	EK1	21.09.2009	04.09.2009
2	2	122858-2A	0	VISAR57	Normal	2x60000	ZS2	EK4	21.09.2009	Keine Daten v...
3	2	122858-2AA	142	VISAR57	Normal	2x60000	ZS2	EK4	21.09.2009	05.09.2009
4	2	122858-1	53	VISAR57	HSH	2x60000	ZS2	EK4	21.09.2009	05.09.2009
5	3	121915-155	29	VI49E1	Normal	1x120000	ZS2	EK4	07.09.2009	05.09.2009
6	3	122673-53	42	VI49E1	Normal	1x120000	ZS2	EK4	14.09.2009	06.09.2009
7	3	123019-1	29	VI49E1	Normal	1x120000	ZS2	EK4	21.09.2009	07.09.2009
8	3	121915-154	38	VI49E1	Normal	2x60000	ZS2	EK4	07.09.2009	07.09.2009
9	3	122951-4	22	VI49E1	Normal	2x60000	ZS2	EK4	22.09.2009	07.09.2009
10	3	122951-3	22	VI49E1	Normal	2x60000	ZS2	EK4	15.09.2009	07.09.2009
11	3	122951-5	24	VI49E1	Normal	2x60000	ZS2	EK4	29.09.2009	07.09.2009
12	3	122963-8	18	VI49E1	Normal	4x30010	ZS1	EK1	15.09.2009	08.09.2009
13	3	122961-7	9	VI49E1	Normal	4x30000	ZS1	EK1	14.09.2009	08.09.2009
14	3	122918-1	11	VI49E1	Normal	4x25000	ZS1	EK1	07.09.2009	08.09.2009
15	3	122955-4	15	VI49E1	Normal	5x24000	ZS1	EK1	07.09.2009	08.09.2009
16	3	123000-1	6	VI49E1	Normal	6x18000	ZS2	EK2	07.09.2009	08.09.2009
17	3	122867-2	9	VI49E1	Normal	3x36000	ZS1	EK1	31.08.2009	08.09.2009
18	3	123027-1	4	VI49E1	Normal	4x30000	ZS1	EK1	15.09.2009	08.09.2009
19	3	122970-1	5	VI49E1	Normal	4x27521	ZS1	EK1	07.09.2009	08.09.2009
20	3	123028-1	3	VI49E1	Normal	6x18000	ZS2	EK2	08.09.2009	08.09.2009
21	3	122724-4	12	VI49E1	Normal	6x18000	ZS2	EK2	01.09.2009	08.09.2009
22	3	123016-1	32	VI49E1	HSH	1x120000	ZS2	EK4	15.09.2009	08.09.2009
23	3	123020-1	13	VI49E1	HSH	4x30000	ZS1	EK1	14.09.2009	08.09.2009
24	3	122856-3	17	VI49E1	HSH	3x36010	ZS1	EK1	31.08.2009	08.09.2009
25	3	121822-106	7	VI49E1	HSH	4x30100	ZS1	EK1	11.09.2009	08.09.2009
26	3	121822-105	2	VI49E1	HSH	4x29800	ZS1	EK1	11.09.2009	08.09.2009
27	3	121822-102	1	VI49E1	HSH	4x29600	ZS1	EK1	11.09.2009	08.09.2009
28	3	122971-1	12	VI49E1	HSH	6x18000	ZS2	EK2	21.09.2009	08.09.2009
29	3	122965-1	15	VI49E1	HSH	3x36000	ZS1	EK1	07.09.2009	08.09.2009
30	3	122942-1	6	VI49E1	HSH	8x15000	ZS2	EK2	29.09.2009	08.09.2009
31	3	123011-1	21	VI49E1	HSH	8x15000	ZS2	EK2	03.09.2009	08.09.2009

Abb. 7.22: Kampagne 3 von Datensatz 5 wurde für Testläufe mit Toleranzen eingesetzt

Abb. 7.23., welche wiederum eine Fitnessverlaufskurve darstellt, zeigt die Ergebnisse für weniger intensive Simulationsläufe. Die Ergebnisse liegen sehr nahe beisammen und können auch stochastisch

bedingt sein, sodass ich keine Aussage darüber treffen möchte, dass der Einsatz von Toleranzen nach derart wenigen Bewertungen einen Vorteil bringt. Rein mathematisch muss durch den vergrößerten Suchraum, sofern dieser komplett abgesucht werden würde, eine mindestens gleich gute Lösung entstehen, wie ohne Verwendung von Toleranzen.

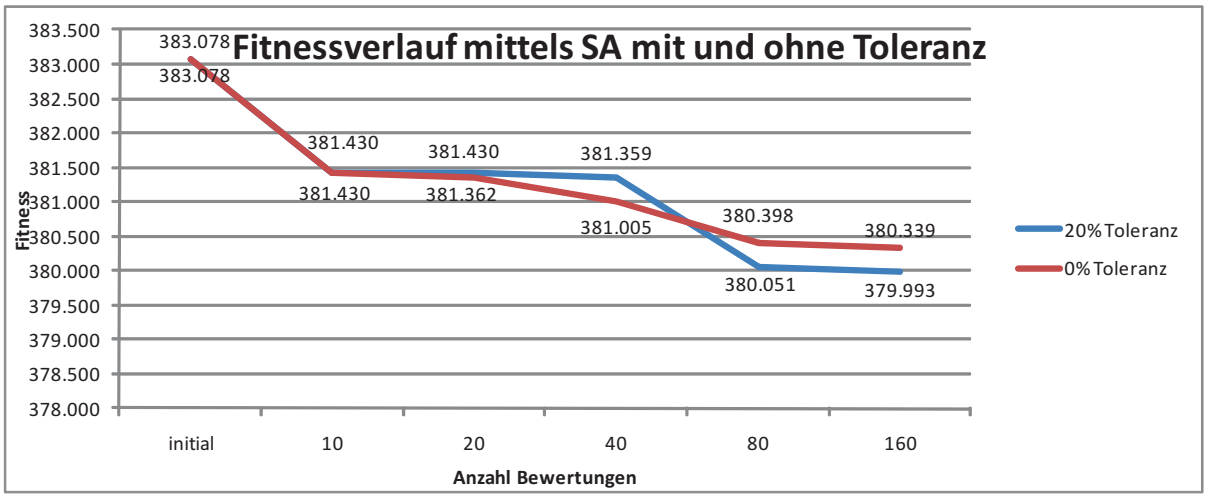


Abb. 7.23: Vergleich der Ergebnisse beim Einsatz von Toleranzen

Abb. 7.24 zeigt den Kurvenverlauf bei intensiveren Simulationsläufen. Es lässt sich vermuten, dass der Algorithmus durch den vergrößerten Suchraum *abgelenkt* werden kann und daher Bewertungen im Rahmen von Mutationen vornimmt, die ihn stagnieren lassen, anstelle essentiellere Bewertungen durchzuführen.

Zusammenfassend lässt sich in Bezug auf beide Abbildungen feststellen, dass der Einsatz von Toleranzen kein signifikantes zusätzliches Potenzial bereitstellt bzw. dieses Potenzial vernachlässigbar gering ist. Lediglich für den Fall, dass man eine einzelne Kampagne optimieren möchte und in diesem Rahmen den Suchraum zur Gänze absucht, kann eine mindestens gleich gute bzw. bessere Lösung garantiert werden.

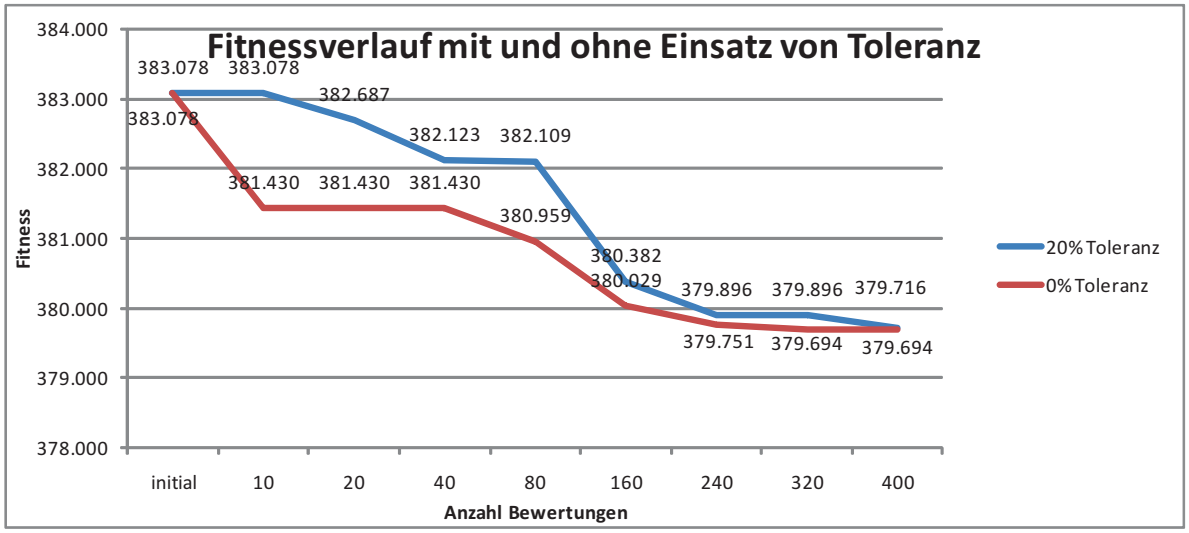


Abb. 7.24: Kurvenverlauf für intensivere Testläufe mit und ohne Einsatz von Toleranzen

Kapitel 8

8 Reflexion, Diskussion sowie Schlussfolgerungen anhand der Ergebnisse der Testläufe

Grundsätzlich möchte ich im Rahmen aller durchgeführten Simulationsexperimente an dieser Stelle darauf verweisen, dass der stochastische Einfluss nicht vernachlässigbar ist im Rahmen der Optimierung mit den einzelnen Algorithmen bzw. jeweiligen Parametrisierungen. Dieser stochastische Effekt wurde so gut wie möglich durch den Einsatz von jeweils mehreren seed Werten minimiert.

8.1 Ursachen sowie Schlussfolgerungen in Bezug auf den positiven Fitnessverlauf in dem durch Nebenbedingungen stark eingeschränkten Suchraum

Der positive Fitnessverlauf lässt sich wie folgt erklären:

1. Zuerst schränken die gegebenen Nebenbedingungen den komplexen Suchraum derart ein, dass dieser relativ überschaubar wird und folglich gute Lösungen leichter zu finden sind.
2. Außerdem wird das Auffinden einer guten bzw. sehr guten Lösung von einer bereits relativ guten Ausgangslösung begünstigt. D.h., dass die Ausgangslösung nicht ein Tal in der Fitnesslandschaft darstellt, sondern schon eine Hochebene (*Plateau*) oder einen Punkt darstellt, welcher bereits den halben Weg zum Gipfel zurückgelegt hat. Von diesem Punkt aus ist die Optimierung relativ einfach und schnell durchzuführen.
3. Andererseits beruht die rasche Konvergenz zu einer relativ guten Lösung darauf, dass es in den metaheuristisch zu optimierenden Kampagnen nur wenige Lose mit vielen Blöcken gibt, deren Positionen man bereits mit relativ wenigen Bewertungen optimieren kann. Außerdem kann die Position von Losen mit kurzen Schienen bedeutend sein, da derartige Lose viele Schnitte an den Sägebohrlinien bedeuten und auch deren Auslastung im Rahmen der Optimierung entscheidend sein kann in Bezug auf optimale Pufferstände bzw. die Durchlaufzeit. Innerhalb der ZSG Lose hat sich herausgestellt, dass sich das Verhältnis allgemein von 50:50 im Rahmen der Optimierung ein wenig zu Gunsten der einen oder anderen (65:35 bzw. 60:40) Sägebohrlinie verschiebt. Die Positionen der vielen anderen Lose mit vergleichsweise wenigen Blöcken ist im Vergleich dazu in Bezug auf die Fitness und die Optimierung nicht allzu bedeutend. Diese relativ wenigen entscheidenden absoluten (und relativ zueinander stehenden) Positionen sind also für die Qualität der Lösung entscheidend.
4. *Bei den typischen metaheuristischen Kampagnen (8-12 Lose) liegen Schemata mit relativ geringer Ordnung sowie großer definierender Länge vor, siehe dazu auch Tab. 8.1. Die 1er in Tab. 8.1 stellen Aufträge dar, deren Positionen unbedingt optimiert werden müssen, weil sie erheblichen*

Einfluss auf die Lösungsqualität haben. Die 0er bezeichnen hingegen Auftragspositionen, die im Rahmen der Optimierung nicht allzu bedeutend sind. Während die Art der Reproduktion keinen (speziellen positiven bzw. negativen) Einfluss in Bezug auf die (relativ große) definierende Länge hat, lässt sich der positive Fitnessverlauf durch die geringe Ordnung sehr gut erklären. Schemata geringer Ordnung können sich im Rahmen der implementierten Reproduktionsmechanismen rasch vermehren. Neben der absoluten Position kann auch die relative Position der 1er untereinander von entscheidender Bedeutung sein. Daher hat sich auch der relativ hohe Anteil an Mutationen ($> 90\%$) gepaart mit einem relativ geringen Anteil an iterierten zusätzlichen Mutationen ("Crossover", $< 20\%$) sehr gut bei der gesamten Optimierung bewährt, da dadurch i.d.R. nicht zu viel Genmaterial auf einmal zerstört wird. Damit erklärt das Schema Theorem in einer für die implementierten Verfahren angepassten Form, das rasche Auffinden einer guten Lösung. Dieser Aspekt ist v.a. im Zusammenhang mit einer bereits relativ guten Ausgangslösung sehr wichtig.

1	0	1	0	0	1	0	1	0	0	0	1

Tab. 8.1: Schema einer typischen metaheuristischen Kampagne mit 12 Losen: Güteklasse 1 (Normal) besteht hier aus 5 Losen und ist blau gekennzeichnet, Güteklasse 2 (HSH) besteht aus 7 Losen und ist rot markiert.

5. In Bezug auf die Analyse der Fitnesslandschaft stellt sich heraus, dass diese relativ flach ist bzw. basierend auf Punkt 2 sehr viele Lösungen schlechtere Lösungen als die Ausgangsreihenfolge darstellen, v.a. weiter entfernte Lösungen, welche große Änderungen bedeuten. Im Rahmen der Optimierung stellt sich generell heraus, dass sehr große Änderungen unter Berücksichtigung der Nebenbedingungen einerseits nur schwer möglich sind und andererseits Lösungen, welche einigermaßen in der näheren Umgebung der Ausgangsreihenfolge liegen, relativ schnell gute Ergebnisse erzielen.
6. Da zu einem Fitnesswert mehrere Reihenfolgen existieren können, besteht der Verdacht, dass eventuell der Fitnesswert zu grob definiert ist und daher viele Reihenfolgen denselben Fitnesswert haben, obwohl sie nicht exakt dasselbe Verhalten in Bezug auf die Produktion aufweisen. Dadurch verkleinert sich der Suchraum insofern, dass es z.B. nicht eine optimale Reihenfolge gibt, sondern in unserem Fall mehrere gleich gute Reihenfolgen, die möglicherweise nicht gleichwertig sind.

Die wichtigste Schlussfolgerung für diesen positiven Fitnessverlauf besteht jedenfalls darin, dass für die Testdatensätze i.d.R. Parametrisierungen mit einigen hundert Bewertungen je Kampagne völlig ausreichend sind, um bereits sehr gute Lösungen zu finden. Diese Schlussfolgerung begünstigt die Gesamtlaufzeit der Optimierung beträchtlich. Weiters erscheint mir die Erkenntnis wichtig zu sein, dass ein riesiger Suchraum nach der Implementation der problemspezifischen Nebenbedingungen in diesem konkreten Fall überschaubar groß und dadurch effizient absuchbar wird. Außerdem möchte ich für andere Optimierungsprobleme an dieser Stelle festhalten, dass man sich in Bezug auf einen größeren Suchraum auch, auf heuristischen Informationen basierend, auf vielversprechende Tauschvorgänge beschränken kann. Wie dieses Problem ausdrücklich zeigt, sind die exakten absoluten Positionen vieler Aufträge nicht von essentieller Bedeutung. Auf Grund mangelnder vollständiger heuristischer Informationen sowie eines auf Grund von Nebenbedingungen überschaubaren Suchraums und des positiven Fitnessverlaufs wurde für dieses Auftragsreihenfolgeproblem auf eine gezielte Suche mittels Beschränkung auf vielversprechende Züge verzichtet.

8.2 Beobachtungen in Bezug auf intensivere Testläufe

Intensivere Testläufe brachten auf allen Datensätzen keine signifikanten Verbesserungen. In diesem Zusammenhang möchte ich auch darauf verweisen, dass die Fitnessverlaufskurven aus Mittelwerten heraus entstanden sind, wobei das einzelne Ergebnis im Rahmen eines Durchlaufs teilweise deutlich von diesem Mittelwert abweichen kann. Auch bei der Parametrisierung 160 ist es daher gut möglich,

noch bessere Ergebnisse zu finden, die auf Grund der Stochastik der Verfahren sowie des Zufallszahlengenerators nicht erfasst worden sind. Dieser Einfluss wurde so gut wie möglich durch die Verwendung mehererer seed Werte, reduziert. Es ist nicht auszuschließen, dass einzelne intensivere Testläufe deutlich bessere Ergebnisse bringen, im Rahmen meiner intensiveren Läufe habe ich diese Erfahrung allerdings auf keinem Datensatz erlebt.

8.3 Schlussfolgerungen auf Grund der Testläufe ohne Berücksichtigung der 'lang vor kurz' Nebenbedingung

Einerseits vergrößert sich der Suchraum durch den Wegfall der Nebenbedingungen in vielen und v.a. größeren Kampagnen enorm, andererseits fällt das zusätzliche Potenzial in der Höhe von $2,12\%$ relativ gering aus. Ich möchte an dieser Stelle darauf verweisen, dass bei umfangreicheren Testläufen ein größeres Potenzial durchaus möglich ist. Interessant ist in diesem Zusammenhang jedenfalls, dass der vergrößerte Suchraum in den betreffenden Kampagnen bei weitem nicht proportional zum zusätzlichen Kostenreduktionspotenzial ist. Auf Grund dieser Ergebnisse empfiehlt es sich nicht, den Produktionsablauf (im Rahmen der Deaktivierung dieser Nebenbedingung) zu ändern.

8.4 Diskussion der Performance der einzelnen Algorithmen

Der Vergleich der Metaheuristiken liefert: $GA \geq SA \geq ACO \geq EA > ILS > Random\,Walks$ Die Unterschiede zum ILS sind signifikant, bei den anderen Algorithmen ist die Unterscheidbarkeit der Performance schwierig feststellbar. Einerseits liegen die Ergebnisse der einzelnen Verfahren oft eng beisammen, andererseits sind die Gesamtergebnisse, welche sich aus Mittelwerten errechnen, bei einer relativ geringen Anzahl an Testläufen zur Berechnung eines Mittelwerts nicht eindeutig. Diese angegebene Reihung der Algorithmen wird jedenfalls durch die Ergebnisse in 7.9 unterstützt.

Grundsätzlich wird der Einsatz von allen Metaheuristiken mit Ausnahme der iterierten lokalen Suche empfohlen. Der Grund dafür liegt darin, dass letztgenanntes Verfahren zu häufig in schlechte Nachbarschaften hineinperturbiert und es so auch passieren kann, dass in einer Kampagne gute Lösungen sehr spät oder gar nicht gefunden werden.

Prinzipiell empfehle ich den GA vor dem SA Algorithmus auf Grund seiner globaleren Sichtweise, v.a. bei intensiveren Testläufen. Für kürzere Läufe empfiehlt sich SA genauso wie der GA. Der ACO liefert ähnlich gute Ergebnisse wie die beiden zuvor genannten Verfahren, leidet jedoch an dem Nachteil, dass er sich seine Reihenfolge jedesmal neu zusammenstellt und die (bereits gute) Ausgangsreihenfolge nicht explizit verwendet, sondern lediglich indirekt im Sinne des Ablegens von Pheromonen darauf zugreift.

Dabei erkennt man, dass er bei kleineren Kampagnen, welche aus weniger Losen bestehen, keine Probleme hat, schnell eine sehr gute Lösung zu finden. Bei großen Kampagnen ist dies jedoch nicht immer der Fall, siehe dazu auch Abschnitt 7.9. Für sehr große Kampagnen bringt der hybride genetische Algorithmus v.a. dann bessere Ergebnisse, wenn die Ausgangsreihenfolge bereits relativ gut ist.

Wenn die Ausgangsreihenfolge hingegen als relativ schlecht einzustufen ist, erscheint die Verwendung des ACO besser bzw. zumindest gleich gut als die eines anderen Algorithmus. Dies wird vom ersten Datensatz in 7.9 bestätigt, siehe dazu auch Abb. 7.13.

Generell lässt sich in Bezug auf die Qualität der Algorithmen festhalten, dass diese beim GA auch stark vom gewünschten Selektionsdruck im Rahmen der Selektion abhängt. Analog dazu gilt, dass sich beim ACO relativ schnell eine deutlich ausgeprägte Pheromonspur um die beste(n) Lösung(en) herauskristallisiert. In diesem Zusammenhang spielt die Verdunstungsrate sowie die Anzahl der Ameisen, welche das Pheromonupdate durchführen, eine wichtige Rolle. Der evolutionäre Algorithmus sucht ausschließlich von besseren Lösungen aus weiter und unterliegt damit bereits einem relativ hohen Selektionsdruck. Dabei werden die besseren Lösungen allmählich durch die besten Kandidatenlösungen ersetzt. Auch simulated annealing liefert, richtig eingestellt, sehr gute Ergebnisse, weil in einem

bestimmten Fitnessrahmen auch zurückgesprungen werden darf und dadurch die Suche nicht zu monoton wird und dadurch sehr gute Ergebnisse liefert. In Bezug auf die Parametrisierung muss man dabei neben den Temperaturen v.a. auch auf die richtige Parametrisierung des Delta Skalierungsfaktors achten, welcher neben den sinkenden Temperaturen als konstanter Faktor erhalten bleibt. Im Rahmen des Skalierungsfaktors kann allgemein (d.h. während des gesamten Optimierungsverlaufs) sehr gut eingestellt werden, in welchem Ausmaß der Algorithmus schlechtere Lösungen annehmen kann.

Der EA sucht prinzipiell breit gestreut Lösungen, welche besser sind als die Ausgangsreihenfolge sind und generiert darauf basierend Eltern. Den einzigen Nachteil, den man im Zuge dieses Verfahrens in Kauf nimmt, ist, dass man nicht bereit ist, den Umweg über schlechtere Lösungen zu gehen um zu besseren Lösungen zu kommen. Dieses Verhalten wurde jedoch absichtlich nicht in den Algorithmus implementiert um nicht zu vielfältig weiterzusuchen und damit Gefahr zu laufen, einer zufallsbasierten Suche nahe zu kommen. Gerade in Bezug auf dieses Problem, wo meistens die Veränderung von wenigen Positionen (absolut bzw. relativ zueinander) ausreichend ist, halte ich das Verhalten des Algorithmus für problemspezifisch besser als wenn man derartige Umwege in Kauf nehmen würde.

Die lokale Suche erweist sich in Abhängigkeit von der Kampagne v.a. bei sehr großen Suchräumen im Zusammenspiel mit vergleichsweise kleinen Parametrisierungen als sinnvoll, sie verbessert das Ergebnis jedoch i.d.R. nicht mehr signifikant.

Metaheuristiken sind im Rahmen der gesamten Optimierung v.a. notwendig um bei Kampagnen mit größeren Suchräumen in schneller Zeit gute bis sehr gute Ergebnisse erzielen zu können. In einzelnen Kampagnen besitzt der Suchraum trotz aller Nebenbedingungen eine unveränderte Größe von $n!$, wobei n die Anzahl von Aufträgen in dieser Kampagne darstellt.

Zusammenfassend kann man also aus diesem speziellen Problem schließen, dass sich genetische Algorithmen sehr gut für Auftragsreihenfolgeprobleme eignen, v.a. bei Vorhandensein einer relativ guten Ausgangslösung. Die Performance des jeweiligen einzelnen Algorithmus hängt dabei auch immer in einem entscheidendem Ausmaß von einem problemgerechten Tuning, dem Einsatz der am sinnvollsten erscheinenden Operatoren bezüglich Selektion, Mutation und Crossover sowie auch von einer entsprechend angepassten Hybridisierung ab.

8.5 Reflexion der Ergebnisse

Das im Rahmen der Optimierung bedeutenste Ergebnis stellt das Kostenreduktionspotenzial in der Höhe von 5,69 % unter realen Bedingungen dar. In diesem Rahmen muss man berücksichtigen, dass dieses Potenzial die bereits gute Ausgangsreihenfolge um den genannten Wert durchschnittlich übertrifft. Wie bereits in Abschnitt 7.8 angesprochen, resultiert dieses Kostenreduktionspotenzial v.a. aus einer kürzeren i.d.R. proportional zum Fitnesswert verkürzten Durchlaufzeit, welche eine höhere Termintreue ermöglicht, sowie produktionsintern aus besseren Pufferständen.

Da in Abschnitt 2.2 angesprochen worden ist, dass die einzelnen Gewichtungen prinzipiell einstellbar sind, aber nicht in dieser Arbeit erläutert werden können, muss auch im Rahmen der Reflexion auf eine ausführliche Diskussion über die Aufteilung des Gesamtkostenreduktionspotenzials in seine Bestandteile verzichtet werden. Die Kostenreduktion ist jedenfalls auf eine bessere Termintreue, eine bessere Walzwerkauslastung sowie eine bessere Verfügbarkeit des Prüfzentrums und bessere Pufferstände zurückzuführen.

Die wichtigste Erkenntnis im Rahmen der Optimierung der einzelnen Kampagnen besteht darin, dass diese sehr rasch gegen eine gute Reihenfolge streben. Die Gründe dafür wurden in 8.1 genannt. In Bezug auf die Laufzeit lässt sich feststellen, dass diese in diesem Zusammenhang sehr kurz ausfällt, bei einer Parametrisierung 160 Simulated Annealing bzw. einer ähnlichen Parametrisierung eines hybriden genetischen Algorithmus hat das Programm nach maximal 2 Stunden (auf einem modernen Rechner) die optimierte Auftragsreihenfolge ermittelt. Generell kann man in diesem Zusammenhang empfehlen, dass wenige hundert Bewertungen genügen um bereits eine sehr gute Gesamtlösung ermitteln zu können.

Von entscheidender Bedeutung erscheint es, in diesem Zusammenhang erkannt zu haben, dass für die gegebenen aktuellen Datensätze das *Vertauschen von wenigen absoluten Positionen, die eventuell auch in einer bestimmten Reihenfolge, also relativ zueinander, angeordnet sein müssen,* genügt, um insgesamt sehr gute Ergebnisse zu bekommen.

In diesem Zusammenhang muss erläutert werden, dass die Optimierung jeder Kampagne v.a. von der richtigen Positionierung dieser wenigen wichtigen Lose abhängt. Für die Optimierung entscheidende Lose können aus vielen Blöcken bestehen und sind erfahrungsgemäß eher Lose mit langen Schienen, in der Regel beinhalten diese Lose Schienen mit 60 Meter bzw. 120 Meter Länge. Es können auch Lose mit kürzeren Schienen, welche mehrere Schnitte erfordern, und, relativ betrachtet, weniger Blöcke beinhalten, wichtigere Lose darstellen. Die meisten Lose einer Kampagne sind durch die Nebenbedingungen einerseits stark beschränkt in ihrer Positionierung bzw. bewirkt eine Änderung der Position nur eine geringe Fitnessdifferenz.

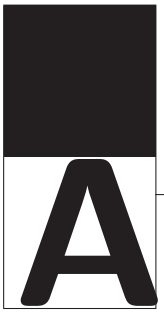
Bei dem gegebenen seriell zu optimierenden Auftragsreihenfolgeproblem kann man die Performance der einzelnen Algorithmen nur schwer vergleichen. Dies liegt v.a. auch in der Stochastik der einzelnen Verfahren begründet. Ohne Nebenbedingungen könnte man die Leistungsfähigkeit der einzelnen Algorithmen eventuell besser vergleichen, allerdings würde dies nicht die realen Bedingungen spiegeln. Daher wurden derartige Läufe nicht durchgeführt.

Der Einsatz von Toleranzen ist v.a. eine Ergänzung, um bestimmte Kampagnen gezielt optimieren zu können. In diesem Fall macht der Einsatz in Bezug auf zusätzliches Kostenreduktionspotenzial einen gewissen Sinn, allgemein gewährleistet der Einsatz von Toleranzen keine besseren Ergebnisse, vergrößert aber den Suchraum. Der gewonnene Suchraum besteht jedoch darin, dass nun mehr Lose mit kurzen Schienen tauschen dürfen, dadurch ist der Fitnessgewinn relativ gering, weil es sich offenbar um Lose und damit auch Positionen handelt, die im Rahmen der Optimierung keine allzu große Rolle spielen.

In Bezug auf die letzte angeführte Ursache in Abschnitt 8.1 kann man sich nun überlegen, ob ein feiner dargestellter Suchraum auf Grund einer detaillierteren Fitnessfunktion effizient absuchbar ist bzw. ob eine Übergenauigkeit (*overfitting*) Sinn macht, wenn die Unterschiede in den betreffenden Reihenfolgen minimal sind. Diese Reihenfolgen können z.B. auf Grund eines Tauschvorgangs ähnlicher (in Bezug auf Schienenlänge bzw. Blockanzahl) Lose zu Stande kommen. Ein derartiger Tauschvorgang hat abgesehen von eventuellen Änderung auftragsbezogener Lager- oder Pönalkosten keine weiteren Konsequenzen. Im Falle des Tausches innerhalb eines ZSG Loses handelt es sich darüber hinaus um dieselbe Auftragsnummer. Diese Tatsache liefert einen weiteren Grund, wieso ein feineres Splitting wenig bzw. keinen Sinn ergibt, siehe dazu auch Abschnitt 7.1.2

In Bezug auf die harte Nebenbedingung, welche eine kampagnenübergreifende Optimierung verhindert, muss an dieser Stelle erwähnt werden, dass damit auch Verspätungen im Rahmen der Optimierung nicht explizit (indirekt, d.h. im Rahmen von Mutationen kann es natürlich schon passieren, dass Verspätungen minimiert werden bzw. Lagerzeit minimiert wird) berücksichtigt werden, da diese Lose innerhalb einer Kampagne um maximal $1 - 2$ Tage zeitlich verschoben werden können. Dieser theoretische Wert wird durch die Nebenbedingungen noch weiter eingeschränkt. Trotzdem muss an dieser Stelle festgehalten werden, dass sich diese Kostenreduktionen für vorverschobene Lose, welche dadurch eine bessere Termintreue erreichen, im Rahmen von vielen Kampagnen stark aufsummieren und letztlich eine deutliche Gesamtdurchlaufzeitverkürzung verursachen.

Weiters würde ich es als interessant erachten, den gesamten Suchraum kampagnenübergreifend auf einmal metaheuristisch optimieren zu können. Eventuell lassen sich dadurch einerseits bessere Ergebnisse erzielen, indem das Potenzial von Metaheuristiken besser ausgenutzt wird. Andererseits vergrößert sich der Suchraum im Rahmen einer gleichzeitigen Optimierung aller Kampagnen enorm, siehe auch Abschnitt 2.4. Dadurch stellt sich die Frage, ob innerhalb einer angemessenen Zeit ähnlich gute Ergebnisse erzielt werden können.



Anhang A

Anhang - Ausschnitte aus dem Quellcode (JAVA)

A.1 Allgemeine Ausschnitte

A.1.1 Globale Tauschmethode

```
/**
* This method checks all activated constraints for the transfered
* sequence and returns the mutated sequence if possible. In the other case the
* transfered sequence is returned itself, if no barter partner
* has been found within 30 tries.
* @param Sequence sequence
* @param Walzkampagne campaign
* @param boolean smallMutation: true → removeAndInsert mutation; false →swap mutation
* @param double toleranz
* @param Random rnd
* @param smallMutation: true-> removeAnd Insert(); false–> swap()
* @return Sequence sequence
* @throws Exception
*/
public Sequence checkConstraintsAndMutate(Sequence seq, Walzkampagne campaign, boolean small-
Mutation, double toleranz, Random rnd) throws Exception
{
   //local declarations
   int campaignSize = campaign.getNumberOfLots();
   int ascendingNumber = Sequence.getAscendingNumber();
   boolean guete = false;
   boolean railLengthHeuristic = false;
   zsgChanged = false;
```

```
int size = seq.getLotIds().size();
int ascOrderId1,ascOrderId2;
double railLength1, railLength2;
int insertSlot, removeSlot;
//pick 2 not identical lots
String lotId1 = seq.getLotIds().get(rnd.nextInt(size));
String lotId2 = seq.getLotIds().get(rnd.nextInt(size));
while(lotId1.equals(lotId2)) {
  lotId2 = seq.getLotIds().get(rnd.nextInt(size));
}
//inits
firstTime = true;
z = 0;
while(lotId1.equals(lotId2) || firstTime || !guete || !railLengthHeuristic)
{
  //inits before each run
  railLengthHeuristic = false;
  guete = false;
  log.debug(z+". constrained loop!");
  //new lot id after 2nd run
  if(z >= 1)
  {
    lotId1 = seq.getLotIds().get(rnd.nextInt(size));
    lotId2 = seq.getLotIds().get(rnd.nextInt(size));
  }
  //get lot dynamically over the sequence: reason: in between actualised ascOrderIds
  Lot lot1 = seq.getLot(lotId1);
  Lot lot2 = seq.getLot(lotId2);
  //nullpointer Check (should not happen)
  if((!(lot1 == null)) || !(lot2 == null))
  {
    // constraint 2: consider qualities
    if(this.isGüteklasseRestriction() == true)
    {
    if(lot1.getSetupValue("GUETE").equals(lot2.getSetupValue("GUETE"))){guete=true;}
    else{guete = false;}
  }
  else{guete = true;}
```

```
// constraint 3: consider rail length
if(this.isLangVorKurzRestriction()==true)
{
  //only continue in the case there are not 2 zsg lots –>
  // these are changed by changeZS()
  if(!lot1.isZsgLot() && !lot2.isZsgLot())
  {
    ascOrderId1 = seq.getAscendingOrderNrForLot(lotId1);
    ascOrderId2 = seq.getAscendingOrderNrForLot(lotId2);
    railLength1 = campaign.getLot(lotId1).getMinRailLength();
    railLength2 = campaign.getLot(lotId2).getMinRailLength();
    //constraint: long before short –> smaller ascOrderId must have
    //shorter or equal long rail length for valid change
    if(ascOrderId1 < ascOrderId2)
    {
      //consider tolerance
      if(railLength1 * (1 - (toleranz/100)) <= railLength2)
      {
        railLengthHeuristic = true;
      }
      else{
        railLengthHeuristic = false;
      }
    }
    //ascOrderId2 < ascOrderId 1
    else
    {
      //consider tolerance
        if(railLength2 * (1 - (toleranz/100)) <= railLength1)
        {
          railLengthHeuristic = true;
        }
        else
        {
          railLengthHeuristic = false;
        }
      }
    }
```

```
        else{railLengthHeuristic = true;}
      }
      else{railLengthHeuristic = true;}
      //constraint 1: consider destination saw
      if(this.isZsgRestriction() == true && (lot1.isZsgLot() || lot2.isZsgLot()))
      {
        //quality and length must be O.K.
        if(guete && railLengthHeuristic)
        {
          //change the ZSG sequence: –> 50:50 proportions remain!
          //seq = restriction.changeZsgSequence(lotId1,lotId2,seq, campaign);
          //change ZS directly: → change of the proportions
          seq = this.zsgChange(lotId1, lotId2, lot1, lot2, seq);
        }
      }
      firstTime = false;
      z++;
    }
    //nullpointer (should not happen)
    else
    {
      log.info(lot1);
      log.info(lot2);
      log.info("**********");
      log.info("NULLPOINTER");
      log.info("**********");
      z++;
    }
    //abort: after 30 tries
    if(z == 30){break;}
    //or after having changed the ZS on one lot
    if(zsgChanged){break;}
  }
  //abort
  if(z == 30)
  {
    log.info("Unable to mutate individuum!");
    return seq;
```

```
    }
    //unchanged (ZSG) lots get changed now...
    else if(!zsgChanged)
    {
      // if they are not fixed and not ZSG lots
      Lot lot1 = seq.getLot(lotId1);
      Lot lot2 = seq.getLot(lotId2);
      if(!lot1.isFixed() && !lot2.isFixed() && !lot1.isZsgLot() && !lot2.isZsgLot())
      {
        //small or bigMutation desired ?
        if(!smallMutation){seq = seq.swap(lotId1,lotId2,seq,campaign);}
        else
        {
          //get indices
          removeSlot = seq.getAscendingOrderNrForLot(lotId1) - ascendingNumber;
          insertSlot = seq.getAscendingOrderNrForLot(lotId2) - ascendingNumber;
          //check valid slots
         if(removeSlot >= 0 && removeSlot < campaignSize && insertSlot >= 0 && insertSlot < campaignSize)
          {
            seq.removeAndInsert(removeSlot, insertSlot, lot1);
          }
        }
        return seq;
      }
      else{return seq;}
    }
    else if(zsgChanged)
    {
      log.info("ZSG lot change performed!");
      log.debug(seq.getLotIdSequenceNrMapping());
      return seq;
    }
    //else: should not happen
    else
    {
      log.info("This case should not occur!");
      return seq;
    }
}
```

A.1.2 Zielsäge Tauschvorgang

```
/**
* This method alters the destination saw.
* This can be considered as a mutation.
* Within the scope of this mutation the proportions on
* the destination saws can change.
* @param String lotId1, lotId2
* @param Lot lot1, lot2
* @param Sequence seq
* @throws Exception
* @return Sequence seq
*/
public Sequence zsgChange(String lotId1,String lotId2,Lot lot1, Lot lot2, Sequence seq) throws Exception
{
   //if lot 1 is a zsg lot → change the destination saw for lot 1
   if(lot1.isZsgLot())
   {
      changeZS(lot1,seq);
      zsgChanged = true;
   }
   //else if the 2nd lot is a zsg lot and the first lot is no zsg lot
   //→ change the destination saw for lot 2
   else if(lot2.isZsgLot()&& !lot1.isZsgLot())
   {
      changeZS(lot2,seq);
      zsgChanged=true;
   }
   //if none of them is a zsg lot → no change
   else{}
   return seq;
}
```

```
/**
* This method alters for the transfered lot object
* the destination saw from "ZS1" to "ZS2" and vice versa.
* Within the scope of this update the to be altered lot gets removed
```

```
* from the actual sequence and is replaced by the inserted lot.
* @param Sequence seq
* @param Lot lot
* @throws Exception
*/
public void changeZS(Lot lot, Sequence seq) throws Exception
{
   String aggregate = lot.getWorkstep("ZS").getAggregate();
   //lot.getAscendingNumber != seq.getAscendingOrderNrForLot(lot.getLotId())
   int ascendingNumber = seq.getAscendingOrderNrForLot(lot.getLotId());
   Lot zsgChangedLot = lot.clone();
   zsgChangedLot.setAscendingOrderId(ascendingNumber);
   //altes Los aus der Sequenz entfernen
   seq.removeLotFromSequence(lot);
   // Los Workstep Update! either ZS1 or ZS2; ZS1 → ZS2
   if(aggregate.equals("ZS1"))
   {
      zsgChangedLot.updateWorkstep("ZS", "ZS2");
      zsgChangedLot.updateWorkstep("EK", "EK2");
      log.debug("Changed aggregate: From ZS1 –> ZS2");
      log.debug("Changed endkontrolle: From EK1 –> EK2");
   }
   //else: ZS2 → ZS1
   else
   {
      zsgChangedLot.updateWorkstep("ZS", "ZS1");
      zsgChangedLot.updateWorkstep("EK", "EK1");
      log.debug("Changed aggregate: From ZS2 –> ZS1");
      log.debug("Changed endkontrolle: From EK2 –> EK1");
   }
   seq.addLotToSequence(zsgChangedLot);
}
```

A.1.3 Checksummen Überprüfung

```
/**
* This method computes a unique checksum for each sequence.
```

```
* It considers the lotId hash code, the position of each order
* and ZSG changes. The worksteps for a lot have to be defined to perform
* this computation.
* The aim of this method is to avoid redundant evaluations
* @return long checksum
* @throws Exception
*/
public long computeChecksum() throws Exception {
   checksum = 0;
   String aggregate;
   for (String lotId : lotIdSequenceNrMapping.keySet()) {
      checksum += lotId.hashCode() * lotIdSequenceNrMapping.get(lotId);
   }
   //consider possible ZSG changes for the computation of the checksum of this sequence
   for(Lot lot : lots)
   {
      aggregate = lot.getWorkstep("ZS").getAggregate();
      //get an unique string for each lot of this sequence
      //for instance.: "ZS11+lotId" and "ZS2+lotId" differ now in their string length
      //annotation: zsgLots differ in their lotId string length
      if(aggregate.equals("ZS1"))
      {aggregate = aggregate + "1" + lot.getLotId().toString();}
      else{aggregate = aggregate + "22"+ lot.getLotId().toString();}
      //The aggregate hash code ,multiplied by the ascendingOrderNr,
      //that is expanded by the lotId is unique
      checksum = checksum + aggregate.hashCode() * lotIdSequenceNrMapping.get(lot.getLotId());
   }
   return checksum;
}
```

A.1.4 Lokale Suche (Insertion Search)

```
/**
* This method creates a local neighborhood based on Insertion Search
* The method considers all constraints (if activated) when generating the neighborhood.
* @param Sequence best
* @param Walzkampagne campaign
```

```
* @param int alpha
* @param Restrictions restriction
* @param List<Sequence> sequencesEvaluated
* @return List<Sequenc> insertionSearchSolutions
* @throws Exception
*/
public ArrayList<Sequence> insertionSearch(Sequence best, int alpha, Restrictions restriction,
Map<Long,Sequence> sequencesEvaluated, double toleranz) throws Exception
{
   int ascendingNumber = Sequence.getAscendingNumber();
   int campaignSize = best.getLotIds().size();
   int railLength1, railLength2;
   boolean railLengthHeuristic = false;
   //get sortedLotIds
   ArrayList<String>sortedLotIds = best.getLotIdsInAscendingOrder();
   //search list as array
   ArrayList<Integer> searchList = new ArrayList<Integer>();
   //fill search list
   for(int i = 0;i < best.getLotIds().size(); i++)
   {
      searchList.add(i);
   }
   int p = 0,k = 0,K = 0, ascOrderId1 = 0, ascOrderId2 = 0;
   String lotId1 = "", lotId2 = "";
   ArrayList<Sequence> insertionSearchSolutions=new ArrayList<Sequence>();
   //solutions consists of max. 2·alpha·n solutions
   int size = campaignSize;
   boolean running;
   // –> while (size != 0) → getP() →....→ n while runs
   while(size != 0)
   {
      p = getP(searchList);
      //k...position, at which the element from position p becomes inserted!
      k = p - alpha -1;
      if(k < 0){k = -1;}
      K = Math.abs(k);
      //generate 2*alpha solutions win thin this while loop
      while(k < K+2*alpha)
```

```
{
  k++;
  //continue if k slot is valid
  if((p != k) && (k < campaignSize))
  {
    running = true;
    // consider array size and set running true
    while(running && (k + ascendingNumber < sortedLotIds.size()) &&
    (p + ascendingNumber < sortedLotIds.size()))
    {
      // get a new ID by cloning
      Sequence a = best.clone();
      // read lot ID
      lotId1 = sortedLotIds.get(p+ascendingNumber);
      lotId2 = sortedLotIds.get(k+ascendingNumber);
      //read lots
      Lot lot1 = a.getLot(lotId1);
      Lot lot2 = a.getLot(lotId2);
      //nullpointer check → abort (should not happen)
      if(lot1 == null || lot2 == null){
        log.info(lot1);
        log.info(lot2);
        log.info("************");
        log.info("NULLPOINTER");
        log.info("************");
        running = false;break;
      }
      // abort in case of one or both fixed lots
      if(lot1.isFixed() || lot2.isFixed()){running = false;break;}
      //consider qualities if restriction 2 enabled
      if(restriction.isGüteklasseRestriction() == true)
      {
        //continue only if the qualities are equal
        if(lot1.getSetupValue("GUETE").equals(lot2.getSetupValue("GUETE"))){}
        //else: abort
        else{running = false;break;}
      }
      //consider railLength if restriction 3 enabled
```

```
if(restriction.isLangVorKurzRestriction() == true)
  {
  //continue only if there are none ZSG lots –>
  //if there is at least 1 ZSG lot –> ZS change()
  if(!lot1.isZsgLot() && !lot2.isZsgLot())
  {
    railLengthHeuristic = false;
    ascOrderId1 = a.getAscendingOrderNrForLot(lotId1)-ascendingNumber;
    ascOrderId2 = a.getAscendingOrderNrForLot(lotId2)-ascendingNumber;
    railLength1 = lot1.getMinRailLength();
    railLength2 = lot2.getMinRailLength();
    //constraint: long before short:
    //smaller ascOrderId must have longer (equal long) railLength
    if(ascOrderId1 < ascOrderId2)
    {
      if(railLength1 * (1 - (toleranz/100)) <= railLength2)
      {
        railLengthHeuristic = true;
      }
       else{
         railLengthHeuristic = false;
      }
    }
    //ascOrderId2 < ascOrderId 1
    else
    {
      if(railLength2 * (1 - (toleranz/100)) <= railLength1)
      {
        railLengthHeuristic = true;

      }
      else
      {
        railLengthHeuristic = false;
      }
    }
    if(railLengthHeuristic == false){running = false;break;}
  }
```

```
        }
        // → consider ZSG and perform ZS change
        if(restriction.isZsgRestriction() == true)
        {
          // ZSG lots available ? → if yes, change ZS!
          if(lot1.isZsgLot() || lot2.isZsgLot())
          {
            a = restriction.zsgChange(lotId1, lotId2, lot1, lot2, a);
            // new or already checked solution ?
            if (!sequencesEvaluated.containsKey(a.computeChecksum()))
            {
              //insert solution...
              insertionSearchSolutions.add(a);
            }
            //...and abort
            running = false;
            break;
          }
        }
        // perform "regular" insertion for non ZSG lots
        a.removeAndInsert(p, k, lot1);
        log.debug("Insertion performed: " + lotId1 + " → " + lotId2);
        // new or already checked solution ?
        if (!sequencesEvaluated.containsKey(a.computeChecksum()))
        {
          insertionSearchSolutions.add(a);
        }
        else
        {
          log.debug("Generated IS solution already checked earlier → not generated!");
        }
        //abort
        running = false;
        break;
      }
    }
  }
  size–;
```

```
    }
    log.info("Insertion based neighbourhood size: "+insertionSearchSolutions.size());
    return insertionSearchSolutions;
}
```

```
/**
* This method gets the next randomly selected index p.
* @param ArrayList<Integer> list
* @return int index
*/
private int getP(ArrayList<Integer>list)
{
//p... starting point for each while loop
int p= rnd.nextInt(list.size());
list.remove(p);
return p;
}
```

A.2 Ausschnitt aus dem hybriden genetischen Algorithmus

A.2.1 Tournament Selektion: $k = 2$ bzw. $k = 3$

```
/**
* Within a tournament of k individuum sequences
* the one with the lesser (= better) fitness is selected.
* k = 2 the first 1/4 iterations: weighted tournament selection: p1 = 0.75, p2 = 0.25
* k = 2 between 1/4 and 1/2 iterations (higher selection pressure than before)
* k = 3 for the last 1/2 iterations to guarantee a high selection pressure in the
* final stage of the genetic algorithm (population size must be equal or over 10 individuums)
*
* elitism activated
* @param List<Sequence> individuums
* @param int actualGeneration
* @return List<Sequence> individuums
*/
private List<Sequence> tournamentSelection(List<Sequence> individuums, int actualGeneration) {
    MultiMap tempPopulation = new MultiMap();
```

```
Sequence sequence = null;
int size = individuums.size();
Collections.sort(resultRatings);
//elitism: select the best individuums in either case
for(int i = 0; i < elitismIndividuums; i++)
{
  //if there are enough different individuums
  if(resultRatings.size() > elitismIndividuums)
  {
    seqList = totalPopulation.get(resultRatings.get(i).getFitnessValue());
    if(seqList.size() > 0 && seqList != null)
    {
      sequence = seqList.get(rnd.nextInt(seqList.size()));
      individuums.remove(i);
      individuums.add(sequence);
    }
  }

  //if there are not enough different individuums: always select the best instead
  else
  {
    seqList = totalPopulation.get(resultRatings.get(0).getFitnessValue());
    sequence = seqList.get(rnd.nextInt(seqList.size()));
    individuums.remove(i);
    individuums.add(sequence);
  }
}
long betterFitness = 0;
//the regular selection
for(int i=elitismIndividuums;i<size;i++)
{
  //choose 2 fitness values (keys)
  int a = rnd.nextInt(size);
  int b = rnd.nextInt(size);
  while(a==b){b = rnd.nextInt(size);}
  //from 1/2 iterations to the end: k = 3 if populationSize >= 10
  if(actualGeneration>parameters.getMaxIterations()/2 && parameters.getPopulationSize() > 10)
  {
```

```
    //get a 3rd fitness value (key)
    int c = rnd.nextInt(size);
    while(c == a || c == b ){c = rnd.nextInt(size);}
    if((fitnessValues.get(a)) <= (fitnessValues.get(b)) &&
    (fitnessValues.get(a)) <= (fitnessValues.get(c)))
    {
       betterFitness = fitnessValues.get(a);
    }
    else if (fitnessValues.get(b) <= fitnessValues.get(a) &&
    fitnessValues.get(b) <= fitnessValues.get(c))
    {
       betterFitness = fitnessValues.get(b);
    }
    else
    {
       betterFitness = fitnessValues.get(c);
    }
}
//for the first 1/2 iterations: tournament with k = 2
//if populationSize < 10: perform this selection too for the last 1/2 iterations
else
{
    if ((fitnessValues.get(a)) < (fitnessValues.get(b)))
    {
       betterFitness = fitnessValues.get(a);
    }
    else
    {
       betterFitness = fitnessValues.get(b);
    }
}
individuums.remove(i);
//replace with the selected individuum
seqList = totalPopulation.get(betterFitness);
if(seqList.size() > 0 && seqList != null)
{
    sequence = seqList.get(rnd.nextInt(seqList.size()));
    individuums.add(i, sequence);
```

```
            tempPopulation.put(betterFitness, sequence);
        }
    }
    log.info("Population size after selection: "+individuums.size());
    //erase dispensable objects
    tempPopulation.clear();
    fitnessValues.clear();
    return individuums;
}
```

A.3 Ausschnitt aus dem Simulated Annealing Verfahren

A.3.1 Bewertung

```
/**
* Within the scope of this method one individuum gets randomly selected and evaluated.
* @param List<Sequence> neighbourhoodSequences
* @return Sequence bestSequence
* @throws Exception
*/
private Sequence evaluate(List<Sequence> neighbourhoodSequences, Sequence bestSequence) throws
Exception {
    int size = neighbourhoodSequences.size();
    int sequenceIndex = rnd.nextInt(size);
    //get sequence from neighborhood and remove it
    Sequence newSequence = neighbourhoodSequences.get(sequenceIndex);
    neighbourhoodSequences.remove(sequenceIndex);
    //get a new sequence until one has been found which has not already been checked
    while(sequencesEvaluated.containsKey(newSequence.getChecksum()))
    {
        size = neighbourhoodSequences.size();
        if(size == 0)
        {
            log.info("Mutated neighborhood empty –> Abort!");
            break;
        }
        sequenceIndex = rnd.nextInt(size);
```

```
    newSequence = neighbourhoodSequences.get(sequenceIndex);
    neighbourhoodSequences.remove(sequenceIndex);
  }
  //evaluate if possible
  if(size > 0)
  {
    campaigns.get(campaignIndex).updateLotSequence(newSequence);
    //evaluate, set ResultRating and compute checksum and put it into sequencesEvaluated
    rating = evaluate();
    fitness = rating.getFitnessValue();
    multiSeqMap.put(fitness, newSequence);
    fitnessArrayList.add(fitness);
    aktuelleBewertung++;
    newSequence.setResultRating(rating);
    sequencesEvaluated.put(newSequence.computeChecksum(), newSequence);
    //if new solution is better or equal, accept it immediately
    if(fitness <= bestSequence.getResultRating().getFitnessValue())
    {
      bestSequence = newSequence;
      log.info("FORWARD UPDATE!");
    }
    //accept a worse solution with a defined probability. (delta scaling factor)
    else
    {
      x = rnd.nextDouble();
    delta = (bestSequence.getResultRating().getFitnessValue() - newSequence.getResultRating().getFitnessValue())/deltaScaling;
      //important: in the else branch delta becomes a negative value –> x < e^(-delta/temperature)
      if(x < Math.exp(delta/temperature))
      {
        bestSequence = newSequence;
        log.info("BACKWARD UPDATE!");
      }
    }
    bestFitness = bestSequence.getResultRating().getFitnessValue();
    log.info(aktuelleBewertung+". evaluation: "+fitness);
    log.info("SA - optimization level: "+bestSequence.getResultRating().getFitnessValue());
    //reset rating!
    rating.reset();
```

```
    }
    //return a better new sequence or the old one
    return bestSequence;
}
```

A.4 Ant Colony Optimization

A.4.1 Berechnung der Wahrscheinlichkeiten für die Auswahl des nächsten Auftrags

```
/**
* This method computes the probabilities of the order to be selected based on
* the pheromon concentration and local information on a certain position.
*
* @param int pos
* the position of concern
* @param List<String> notChosenLots
* @param double[][] localMatrix
* @param List<String> partialSolutionPerAnt
* @return
*/
 protected List<LotSelectionProbability> computeProbabilityList
(int pos, List<String> notChosenLots, double[][]localMatrix, List<String> partialSolutionPerAnt) {
    //inits
    double probability = 0d;
    double probabilitySum = 0d;
    double pheromonConcentration = 0d;
    double localInformation = 0d;
    double totalPheromonConcentrationOnPos = 0d;
    double totalLocalInformationOnPos = 0d;
    LotSelectionProbability selectionProbability = null;
    List<LotSelectionProbability> probabilityList = new ArrayList<LotSelectionProbability>();
    //re-initialize local matrix before updating it
    localMatrix = initLocalMatrix(localMatrix.length);
    //update localMatrix based on the current position & last selectedLotId for this ant!
    if(partialSolutionPerAnt.size() > 0){localMatrix =
    updateLocalMatrix(partialSolutionPerAnt,localMatrix, this.campaign, pos, notChosenLots);}
    //compute the total sum of pheromonConcention of the position
```

```
totalPheromonConcentrationOnPos = getPheromonConcentrationSumOnPosition(pos, notChosen-
Lots);
//compute the total sum of localInformation of the position
totalLocalInformationOnPos = getTotalLocalInformation(pos, notChosenLots, localMatrix);
for (String lotId : notChosenLots) {
  //load pheromon concentration of order on position pos
  pheromonConcentration = getPheromonConcentration(
pos, lotMappingTable.get(initialLotList.get(lotId+"").getAscendingOrderId()));
  //load local information of order on position pos
  localInformation = getLocalInformation(
  pos, lotMappingTable.get(initialLotList.get(lotId+"").getAscendingOrderId()), localMatrix);
  selectionProbability = new LotSelectionProbability(lotId);
  if(pheromonConcentration > 0 && totalPheromonConcentrationOnPos > 0) {
    log.debug(lotId + ": " + pheromonConcentration + " on position " + pos);
  //Compute selectionProbability based on pheromons+localInformations:(pheromons^alpha*localInformation^beta)/(total
  probability=(Math.pow(pheromonConcentration,parameters.getAlpha())·Math.pow(localInformation,parameters.getBeta
  (Math.pow(totalPheromonConcentrationOnPos,parameters.getAlpha())·Math.pow(totalLocalInformationOnPos,
parameters.getBeta()));
    selectionProbability.setProbability(probability);
    //Compute p(order,pos)
    probabilityList.add(selectionProbability);
    //compute probabilitySum
    probabilitySum += probability;
  }
  else
  probabilityList.add(selectionProbability);
}
//sort in decreasing order
Collections.sort(probabilityList);
Collections.reverse(probabilityList);
return probabilityList;
}
```

A.4.2 Globales Update der Pheromonmatrix

```
/**
* This method performs the update of the pheromon matrix after each iteration for the best ant(s).
* @param Map<Long, List<Lot>> resultsPerAnt
```

```
*/
protected void updatePheromonMatrix(Map<Long, List<Lot>> resultsPerAnt) {
   //retrieve the fitnessValues
   List<Long> fitnessValues = new ArrayList<Long>(resultsPerAnt.keySet());
   //sort in ascending order
   Collections.sort(fitnessValues);
   log.info("Ants: " + parameters.getNumberOfAnts());
   log.info("best Ants: " + parameters.getNumberOfBestAnts());
   log.info("Number of Fitness Values: " + fitnessValues.size());
   //consider Evaporation Rate
   considerEvaporationRate();
   int bestFitnessValues = 0;
   if(parameters.getNumberOfBestAnts() >= fitnessValues.size())
      {bestFitnessValues = fitnessValues.size();}
   else{bestFitnessValues = parameters.getNumberOfBestAnts();}
   //only the results of the best ant(s) shall be used
   for(int j = 0; j<bestFitnessValues; j++) {
      //Retrieve Ant with the best result
      List<Lot> resultPerAnt = resultsPerAnt.get(fitnessValues.get(j));
      Collections.sort(resultPerAnt);
      log.info(j+". ant used for pheromon update!");
      for (Lot lot : resultPerAnt) {
         log.info(lot.getAscendingOrderId() + ": " + lot.getOrderId());
      }
      log.info("**************************************************");
      int initialAscendingOrderId = 0;
      //Iterate over orders
      for (int i = 0; i < resultPerAnt.size(); i++) {
      log.debug("CurrentOrder: " + resultPerAnt.get(i).getOrderId() + ", " + resultPerAnt.get(i).getAscendingOrderId());
         log.debug(initialLotList);
         initialAscendingOrderId = lotMappingTable.get(
         initialLotList.get(resultPerAnt.get(i).getLotId()).getAscendingOrderId());
       log.debug("initial OrderId: " + resultPerAnt.get(i).getOrderId() + ": " + initialAscendingOrderId);
         //update the pheromon matrix →
        pheromonMatrix[initialAscendingOrderId][i] = (pheromonMatrix[initialAscendingOrderId][i] · (1-r)) + 1d;
      }
   }
   //printPheromonMatrix();
}
```

A.5 Evolutionary Algorithm

A.5.1 Selektion der Eltern beim EA

```
private void selectNewParents() {
  //add all parents from the last iteration to potential parents!
  fitnessValues = new ArrayList<Long>(parents.keySet());
  for(int i=0;i<parents.size();i++)
  {
    //get ArrayList<Sequence>
    seqList = parents.get(fitnessValues.get(i));
    //add each sequence if possible: the check is done in MultiMap.put()
    for(int j=0;j<seqList.size();j++)
    {
      potentialParents.put(fitnessValues.get(i), seqList.get(j));
    }
  }
  //clear old parents
  parents.clear();
  parentSequences.clear();
  if(potentialParents != null && potentialParents.size() > 0)
  {
    fitnessValues = new ArrayList<Long>(potentialParents.keySet());
    //sort fitnessValues
    Collections.sort(fitnessValues);
    //take the best X candidates to enable them as potential future parents
    int border = Math.min(selectionSize, fitnessValues.size());
    for(int i = 0; i<border; i++)
    {
      long fitnessValue = fitnessValues.get(i);
      //set new parents
      seqList = potentialParents.get(fitnessValue);
      for(Sequence sequence : seqList)
      {
        parents.put(fitnessValue, sequence);
        totalPopulation.put(fitnessValue, sequence);
        log.info("new Parent: " + fitnessValue);
        parentSequences.add(sequence);
```

```
            i++;
            if(i==border){break;}
          }
        }
      }
      potentialParents.clear();
}
```

A.6 Integration der Nebenbedingungen

A.6.1 NB 1: ZSG Lose

Die Überprüfung wird in der jeweiligen Tauschmethode intern durchgeführt. Vgl. A.1.1.

In diesem Zusammenhang wird das boolean Attribut zsgLot im Objekt Lot abgefragt.

A.6.2 NB 2: Güteklasse (Härte)

Diese Überprüfung wird in der jeweiligen Tauschmethode intern durchgeführt. Vgl. A.1.1.

Konkret werden die verglichenen Qualitätsgüteklassen mittels equals() auf Gleichheit geprüft.

A.6.3 NB 3: Lange vor kurzen Schienen

Diese Nebenbedingung wird in der jeweiligen Tauschmethode überprüft, indem bei den beiden zu tauschenden Losen jeweils die Länge verglichen wird.

Im Rahmen dieses Vergleichs ist zu beachten, dass längere Schienen an eine Position weiter vorne getauscht werden können bzw. kürzere Schienen auf eine Position weiter nach hinten getauscht werden können. Lose mit gleichlangen Schienen können ebenfalls getauscht werden. Ggf. werden im Rahmen des Vergleichs auch Toleranzen berücksichtigt. Vgl. A.1.1.

Literaturverzeichnis

[1] SIVANANDAM, S.N.; DEEPA, S.N.: Introduction to Genetic Algorithms, Springer Verlag Berlin, Oktober 2007

[2] SCHOENEBURG, E.; HEINZMANN, F.; FEDDERSON, S.: Genetische Algorithmen und Evolutionsstrategien - Eine Einführung in Theorie und Praxis der simulierten Evolution, Addison Wesley Verlag 1994

[3] RAIDL, G.:, CHWATAL, A.: Heuristische Optimierungsverfahren, Skriptum zur Vorlesung Heuristische Optimierungsverfahren an der TU Wien, 2008

[4] MICHALEWICZ, Z.: Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, New York, 1992

[5] GOLDBERG, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, MA, 1989

[6] BEYER, H.G.: Theory of Evolution Strategies, Springer, 2001

[7] CHAMBERS, L.: Practical Handbook of Genetic Algorithms - Applications Volume I, 1995

[8] CHAMBERS, L.: Practical Handbook of Genetic Algorithms - New Frontiers Volume II, 1995

[9] PETRI, C.: Ablaufplanung bei Reihenfertigung mit mehrfacher Basis auf der Basis von Ameisenalgorithmen, 2006, Dissertation, Universität Passau

[10] FELTL, H.: Ein genetischer Algorithmus für das Generalized Assignment Problem, 2003, Diplomarbeit, TU Wien

[11] FRAMINAN, J.M.; LEISTEN, R.: An efficient constructive heuristic for flowtime minimisation in permutation flow shops, Omega, Volume 31, Issue 4, August 2003, Pages 311-317

[12] LAHA, D.; SARIN, S.C.: A heuristic to minimize total flowtime in permutation flow shop, Omega 37: Pages 734-739 (2009)

[13] ZHANG, Y.; LI, X.; WANG, Q.: Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization, European Journal of Operational Research, Volume 196 (2009), Issue 3, Pages 869-876

[14] JUNGNICKEL,D.: Graphs, Networks and Algorithms, Springer Verlag 1998.

[15] SEIFTER, N.: Mathematische Grundlagen des Operations Research, Skriptum zur Vorlesung Mathematische Grundlagen des Operations Research an der MU Leoben, 2005

[16] JETZKE, S.: Grundlagen der modernen Logistik, Hanser Verlag

[17] TSENG, L.-Y.; LIN, Y.-T.: A hybrid genetic local search algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research, Volume 198 (2009), Pages 84 - 92

[18] AMOUS, S.K.; LOUKIL, T.; ELAOUD, S.; DHAENENS, C.: A new genetic algorithm for the travelling salesman problem, International Journal of Pure and Applied Mathematics, Volumen 48 No. 2 2008, Pages 151 - 166

[19] GRUBER, M.; RINNER, M.; LÖSCHER, T.: Vorausschauende Produktionsregelung - Simulationsbasierte heuristische Optimierung, PROFACTOR, 2009

[20] KOPINITSCH, B.: An Ant Colony Optimisation Algorithm for the Bounded Diameter Minimum Spanning Tree Problem, Magisterarbeit, TU Wien, 2006

[21] Online im Internet: http://de.wikipedia.org/wiki/Simulierte_Abkühlung; Stand: Juli 2009

[22] WEGENER, I.: Simulated Annealing Beats Metropolis in Combinatorial Optimization; Electronic Colloquium on Computational Complexity, Report No. 89 (2004); Springer Verlag Berlin/Heidelberg 205, Pages 589 - 601

[23] MICHALEWICZ, Z.; FOGEL, D.B.: How to Solve It: Modern Heuristics, Springer Verlag Berlin 2000

[24] SCHONER, P.: Operative Produktionsplanung in der verfahrenstechnischen Industrie, Dissertation, Universität Kassel

[25] Online im Internet: http://de.wikipedia.org/wiki/Metaheuristik, Stand: September 2008

[26] KRUSE, R.: Genetische Algorithmen, Skriptum zur Vorlesung, Universität Magdeburg, Sommersemester 2006

[27] VORDERWINKLER, M.: Bridge-Projekt 814323: VPR - Vorausschauende Produktionsregelung in Walzwerken, PROFACTOR, 2007

[28] GRAF, S.: Auswahl und Implementierung eines Ameisenalgorithmus' zur Steuerung von Patienten im Planspiel "INVENT", Diplomarbeit, Universität Wien, 2003

[29] FINK, A.; ROTHLAUF, F.: Heuristische Optimierungsverfahren in der Wirtschaftsinformatik, Working Paper 10/2006, Universität Mannheim

[30] MACHADO, P.; TAVARES,J.; PEREIRA,F.B; COSTA,E.: Vehicle Routing Problem: Doing it the Evolutionary Way

[31] PUCHINGER,J.: Optimierungsverfahren in der Transportlogistik, Skriptum zur Vorlesung Optimierungsverfahren in der Transportlogistik an der TU Wien, 2009

[32] KRAMER, O.: Computational Intelligence - Eine Einführung, Springer Verlag Berlin Heidelberg, 2009

[33] KORB, O.: Das Traveling Salesman Problem im GAILS-Framework: Integration und Analyse, Studienarbeit, Technische Universität Darmstadt, 2004

[34] MERZ, P.: Moderne heuristische Optimierungsverfahren: Meta-Heuristiken, Skriptum zur Vorlesung: Moderne Heuristische Optimierungsverfahren: Meta-Heuristiken, Universität Tübingen, 2003

[35] KUHN, H.C.: Seminararbeit zum Thema: Praktische Anwendungen der Suchstrategie Tabu Search - ein Überblick, Fern Universität Gesamthochschule Hagen

[36] BANGSOW, S.: Fertigungssimulationen mit Plant Simulation und SimTalk, Carl Hanser Verlag München Wien, 2008

[37] WEIGERT, P.: Praktikum Feinwerktechnik, Versuch C5 Simulationsgestützte Optimierung von Fertigungsprozessen, Skriptum zum Praktikum, Technische Universität Dresden

[38] STERINGER, R.; KABELKA, B.: Simulationsgestützte Prozessanalyse der Schienenproduktion am Standort Donawitz, Simulationsstudie, Profactor GmbH/ ABF, Version 2.0, Juli 2008

[39] RAIDL, G.; PATOCKA, A.: Algorithmen und Datenstrukturen I, Skriptum zur Vorlesung im SS 2004

[40] SCHÜLLER, A.: Adaptive Optimierung fokussierter Substanzbibliotheken, Dissertation, 2008, Universität Frankfurt

[41] SEIFTER, N.: Optimierung für Industrielogistiker; Skriptum zur Vorlesung Optimierung für Industrielogistiker im WS 2009, Version: 14. Dezember 2009

[42] GELL-MANN, M.: Google TechTalk Vortrag: On Getting Creative Ideas, March 2007