HAW HAMBURG

BACHELORTHESIS
Jonas Steinhauser

Einsatz der Genetischen Programmierung zur Lösung des Storage Location Assignment Problems anhand eines konkreten Anwendungsfalls

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Jonas Steinhauser

Einsatz der Genetischen Programmierung zur Lösung des Storage Location Assignment Problems anhand eines konkreten Anwendungsfalls

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg
Betreuender Prüfer: Prof. Dr. Michael Neitzke
Zweitgutachter: Prof. Dr. Peer Stelldinger
Eingereicht am: 24.05.2023

Jonas Steinhauser

Thema der Arbeit

Einsatz der Genetischen Programmierung zur Lösung des Storage Location Assignment Problems anhand eines konkreten Anwendungsfalls

Stichworte

Genetische Programmierung, Storage, SLAP, Optimierung

Kurzzusammenfassung

Wirtschaftliches Wachstum gilt als zentrales Ziel jedes Unternehmens. Um die Leistungsfähigkeit eines Unternehmens bei wachsendem Konkurrenzdruck zu steigern, kann eine Optimierung des Lagers vorgenommen werden. Jene wird zum zentralen Gegenstand dieser Arbeit, wobei die Optimierung eines konkreten Lagers mit Hilfe der genetischen Programmierung untersucht wird. Das Ziel besteht darin, eine möglichst gute Sortierungsmethode für das Lager zu ermitteln. Hierzu wird das Potential von genetischer Programmierung für das Storage Location Assignment Problem geprüft. Im Anschluss wird der Algorithmus in einer Implementierung getestet. Grundlegende Problem dieser werden aufgearbeitet, woran anknüpfend Lösungsansätze untersucht werden. Dabei hat sich herauskristallisiert, dass die Tournament Selection in der genutzten Konfiguration des Algorithmus die besten Ergebnisse liefert. Des Weiteren konnte herausgestellt werden, dass eine zu starke Größenkontrolle dem Algorithmus die Möglichkeit nimmt, eine gute Lösung ermitteln zu können. In finalen Tests, die die Erkenntnisse der vorangestellten Versuche berücksichtigen, konnte festgestellt werden, dass die genetische Programmierung ein erhebliches Potential zur Optimierung eines chaotischen Lagers mit einem dynamischen Sortiment birgt. Die Ergebnisse dieses Tests wurden durch einen Validierungsdatensatz geprüft. Daraus kann der Schluss gezogen werden, dass die Sortierung des Lagers erhebliche Ersparnisse in der Kommissionierung bewirken kann.

Jonas Steinhauser

Title of Thesis

Application of genetic programming to solve the storage location assignment problem using a case study.

Keywords

Genetic Programming, Storage Location Assignment Problem, SLAP

Abstract

Economic growth is considered a central goal of every company. To increase the performance of a company in the face of growing competitive pressure, the warehouse can be optimized. This becomes the central subject of this thesis, where the optimization of a concrete warehouse is investigated with the help of genetic programming. The goal is to determine the best possible sorting method for the warehouse. For this purpose, the potential of genetic programming for the storage location assignment problem is examined. Subsequently, the algorithm is tested in an implementation. Fundamental problems of this are worked out, whereupon following solution beginnings are examined. It was found that the Tournament Selection in the used configuration of the algorithm provides the best results. Furthermore, it could be shown that a too strong size control takes away the possibility of the algorithm to determine a good solution. In final tests, considering the findings of the preceding experiments, it was found that genetic programming has considerable potential for optimizing a chaotic warehouse with a dynamic assortment. The results of this test were verified by a validation data set. It can be concluded that the sorting of the warehouse can bring about significant savings in the picking process.

Inhaltsverzeichnis

1 Einleitung

„Stagnation bedeutet Rückschritt“ (Schwenker/Bötzel 2006, 11).

Optimierung ist ein kontinuierlicher Prozess, der alle Lebensbereiche berührt und nach Schwenker und Bötzel als wirksame Maxime gilt. Insbesondere im wirtschaftlichen Sektor präsentiert sich das Streben nach Verbesserung als relevant, um als Unternehmen mit der stetig wachsenden Konkurrenz mithalten zu können. Neben dem Verbessern von Einkaufs- und Verkaufsprozessen ist das Optimieren von Prozessen im logistischen Bereich zum Einsparen von Ressourcen, Kosten und Zeit für viele Branchen eine wichtige Stellschraube und wird als essenziel zur Gewinnsteigerung eines Unternehmens erachtet (vgl. Koch 2014, 7). Des Weiteren ermöglicht eine effiziente Planung des logistischen Bereiches das Wachstum eines Unternehmens, da die zugrunde liegenden Prozesse jenes maßgeblich beeinflussen.

In der Lagerlogistik finden sich viele Anwendungsbereiche, in denen eine Optimierung zu zeitlichen, aber auch finanziellen Ersparnissen führen kann. Die Lagerhaltung einer Firma hat große Auswirkung auf die Fähigkeit, Aufträge von Kunden möglichst schnell und gleichzeitig auch kosteneffizient erfüllen und den Kunden somit zufriedenstellen zu können (vgl. Koch 2014, 7). Damit birgt jene das Potential, die Kapazität von Aufträgen, die ein Unternehmen gleichzeitig bearbeiten kann, zu erhöhen und das Wachstum des Unternehmens zu ermöglichen. In einem vertriebsorientierten Unternehmen geht das Wachstum des Unternehmens mit dem des Lagers einher. Mit steigender Nachfrage wird auch das Lager zunehmend belastet. Der veränderten Dynamik des Lebensverhaltens ist geschuldet, dass Kunden heutzutage eine immer schnellere Lieferung erwarten. Um diese Erwartungen zu erfüllen und die steigende Belastung des Lagers aufzufangen, wird es immer wichtiger, das Lager so effizient wie möglich zu gestalten. Dafür müssen die internen Prozesse innerhalb des Lagers so

optimiert werden, dass es zu keinen Verzögerungen bei der Lieferung kommen kann. Hierbei spielt besonders die Kommissionierung eine große Rolle, für deren Optimierung es einer Sortierung des Lagers bedarf.

Die vorliegende Arbeit untersucht den Einfluss der Sortierung des Lagers mittels genetischer Programmierung auf die Optimierung der Kommissionierung im Betrieb der APS Glass and Bar Supply GmbH. Das Ziel ist es hierbei, zu ermitteln, ob sich der Mehraufwand im Vergleich zu einer unsortierten Lagerhaltung für das im Folgenden vorgestellte Unternehmen positiv auf die Effizienz der Auftragsabwicklung auswirkt. Der Fokus der Arbeit liegt darauf, das Potential der genetischen Programmierung zur Lösung dieses Problems zu ermitteln. Konkret stellt diese Arbeit die These auf, dass die genetische Programmierung ein erhebliches Potential zur Lösung des Storage Location Assignment Problems in Bezug auf den konkreten Anwendungsfall aufweist. Außerdem wird erwartet, dass die Sortierung des Lagers mittels genetischer Programmierung erhebliche Vorteile gegenüber einer unsortierten Lagerhaltung birgt. Dies liegt darin begründet, dass die in der bisherigen Lagerhaltung unbeachtete Nachfrage der Artikel sehr ungleich verteilt ist. Durch die Berücksichtigung dieser und weiterer Faktoren können kürzere Kommissionierungswege erreicht werden, die zu einem größeren Bestellungsdurchsatz führen.

Die APS Glass and Bar Supply GmbH ist ein zum Großteil B2B orientiertes Unternehmen und vertreibt alles rund um den Gastronomiebedarf. Das Sortiment besteht hauptsächlich aus Gläsern, Geschirr und Besteck sowie Barzubehör. Allerdings gibt es auch größere Artikel wie zum Beispiel Eismaschinen. Eine Herausforderung im Lager stellt dar, die Artikel so zu sortieren, dass es einerseits möglich ist, eine optimierte Route durch das Lager beim Kommissionieren zu realisieren und andererseits das Verschwenden von Lagerplatz zu vermeiden. Das Hauptlager, von dem aus die meisten Bestellung versendet werden, hat nur begrenzten Platz. Daher gilt es zu beachten, dass die frequentierten Artikel vor Ort gelagert werden können. Mit

einer zunehmenden Anzahl an Artikeln im Sortiment gewinnt diese Aufgabe an Bedeutung. Um zukünftiges Wachstum zu ermöglichen und dabei die Kosten einer Umfuhr zwischen den Lagern zu sparen, ist es wichtig, den Lagerplatz so effizient wie möglich zu nutzen. Solch eine effiziente Lagerplatznutzung kann unter anderem durch eine chaotische Lagerhaltung realisiert werden. Der Vorteil besteht darin, dass diese Art der Artikelanordnung sehr platzsparend ist, da keine Lagerfläche für bestimmte Artikel vorgesehen ist und somit leer bleiben könnte, falls der Artikel zurzeit nicht vorhanden ist. Für dieses System ist eine sorgfältige Dokumentation der Lagerplätze aller Artikel notwendig, damit diese auch ohne klare Sortierung gefunden werden können. Daher wird in dieser Art der Lagerhaltung ein Warenwirtschaftssystem für die Organisation des Lagers verwendet.

Um das aufgestellte Optimierungsproblem sowie die sich daraus resultierende These begründet beantworten zu können, wird die vorliegende Arbeit wie folgt gegliedert:
In einer theoretischen Einführung sollen zunächst die für die Arbeit notwendigen Grundbegriffe geklärt werden, um eine Grundlage zu schaffen, auf der sich die darauffolgenden Ausführungen bewegen. Zur Lösung des Optimierungsproblems wird das Potential verschiedener Algorithmen diskutiert. Daraus wird eine mögliche Lösungsstrategie für das Problem theoretisch erarbeitet, um im Anschluss eine Implementierung vorzustellen. Zu Beginn werden hier die einzelnen Komponenten des Algorithmus vorgestellt bevor in einigen Versuchen verschiedenen Phasen der Entwicklung erfasst und untersucht werden, woraus Verbesserungen des Lösungsansatzes abgeleitet werden können. Die Arbeit schließt mit einem Fazit ab, das die gewonnen Erkenntnisse zusammenfasst und die eingangs aufgestellte These prüft. Ein Ausblick bietet weitere Forschungsansätze sowie interessante Themen für zukünftige Arbeiten an.

2 Grundlagen der Lagerlogistik

Im ersten Kapitel dieser Arbeit werden relevante Begriffe aus der Lagerlogistik erläutert, deren Kenntnis als Voraussetzung für die folgenden Ausführungen gilt.

2.1 Operational Research

Operational Research (OR) stammt ursprünglich aus dem militärischen Kontext und beschreibt „die Entwicklung spezieller Methoden zur Lösung strategischer Probleme“ (Koop/Moock 2018, 1). Nach dem Ende des 2. Weltkrieges fand das Verfahren „Einzug in den Bereich der Wirtschaft“ (Koop/Moock 2018, 1) und „beschreibt die anwendungsbezogene Optimierung von Situationen, die i.d.R. bei wirtschaftlichen Prozessen und Planungen in Industrieunternehmen auftreten“ (Homberger/Bauer/Preissler 2019, 17). Somit stellt OR den Oberbegriff für wirtschaftliche Optimierungsprobleme dar. Hierbei verstehen Unternehmen unter Optimierung weniger das konkrete mathematische Optimierungsverfahren, sondern eher die Verbesserung einer nicht zufriedenstellenden Situation. Bei den Lösungsansätzen werden zwei Verfahrenskategorien, die exakten Verfahren und Heuristiken, unterschieden (vgl. Homberger/Bauer/Preissler 2019, 17 f.). Beide werden im Folgenden näher erläutert.

2.1.1 Exakte Verfahren

Bei exakten Verfahren werden mathematisch exakte Lösung gesucht. Mit wachsenden Datenmengen und Anforderungen werden diese Verfahren aufwendiger und bei vielen praktischen Anwendungen nicht mehr in einer akzeptablen Zeit berechenbar (vgl. Homberger/Bauer/Preissler 2019, 18).

2.1.2 Heuristiken

Heuristiken stellen das Gegenstück zu den mathematisch exakten Verfahren dar. Das Ziel ist es, eine möglichst gute, aber nicht zwingend optimale Lösung in vertretbarer Zeit zu ermitteln (vgl. Homberger/Bauer/Preissler 2019, 18).

2.2 Zuordnungsproblem

Mit dem klassischen Zuordnungsproblem wird die „paarweise Zuordnung von Elementen aus zwei unterschiedlichen Mengen“ (Koop/Moock 2018, 143) beschrieben. Sie stellt ein Sonderfall des Transportproblems dar (vgl. Koop/Moock 2018, 143).

2.3 Feste Lagerplatzzuordnung

Die feste Lagerplatzzuordnung teilt jedem Artikel feste Lagerplätze, die ausschließlich für diesen reserviert sind und freigehalten werden (s. Abbildung 1), zu (vgl. Wannenwetsch 2014, 300). Als Einflussfaktoren definiert Pfohl ergänzend die „Weglänge, Umschlagshäufigkeit, Wertigkeit, Gewicht, Zugriffshäufigkeit, Volumen und Abmessungen des zu lagernden Gutes, aber auch das Volumen der Einheit, in der das Gut an den Kunden verkauft wird“ (Pfohl 2018, 136 f.). Sie findet Nutzung in Anwendungsfällen, bei denen ein hoher Wiederholungsfaktor gegeben ist. Der große Platzbedarf stellt hierbei den Hauptnachteil dar. Als Vorteil führt Wannenwetsch die direkte Zuordnung von Artikel und Lagerplatz an (vgl. Wannenwetsch 2014, 301).

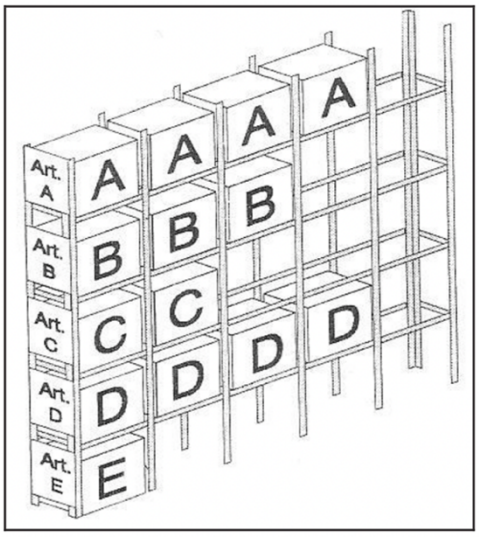

Abbildung 1: Feste Lagerplatzzuordnung
Quelle: Wannenwetsch 2014, 300

2.4 Chaotische Lagerplatzzuordnung

Als chaotische Lagerplatzzuordnung wird im Allgemeinen die freie Einlagerung von Artikeln in beliebige, nicht festgelegte Lagerplätze bezeichnet (s. Abbildung 2). Den Grundbaustein für diese Form der Lagerhaltung legt eine genaue Dokumentation der Plätze, an denen die Artikel eingelagert werden, um später auf diese zurückgreifen zu können (vgl. Wannenwetsch 2014, 301). Eine chaotische Lagerplatzzuordnung bietet sowohl Vor- als auch Nachteile gegenüber der festen Lagerhaltung, deren Konzept im vorangehenden Abschnitt erläutert worden ist. Nach Wannenwetsch (2014, 302) seien diese Vorteile immens (vgl. Wannenwetsch 2014, 302). So besteht bei der chaotischen Lagerhaltung die „Möglichkeit, Lagerbereich und Lagerplatz, angepasst an den tatsächlichen Lagervolumenbedarf, zu bestimmen“ (Wannenwetsch 2014, 302). Dies ermöglicht eine effiziente Nutzung des vorhandenen Lagerraums. Dies identifiziert Pfohl als primäres Ziel der chaotischen Lagerplatzzuordnung (vgl. Pfohl 2018, 138). Wannenwetsch bemisst die Optimierung der Nutzungsfläche mit 30% bis 50% gegenüber der festen Lagerplatzzuordnung. Die Vorteile betreffen noch weitere für das

Unternehmen relevante Bereiche wie zum Beispiel die Personalabhängigkeit und Einarbeitungszeit neuen Personals (vgl. Wannenwetsch 2014, 302).
Die hohe Abhängigkeit von einem Warenwirtschaftssystem und die damit einhergehenden hohen Anschaffungskosten stellen einen Nachteil dieser Strategie dar. Darüber hinaus besteht nur eine kurze Lagerlistenaktualität (vgl. Wannenwetsch 2014, 302).

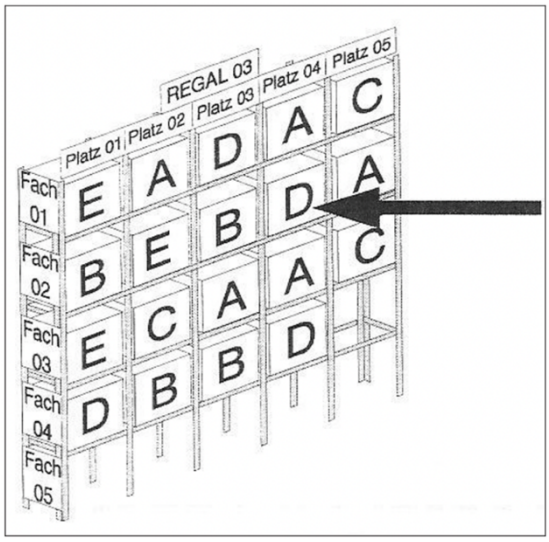

Abbildung 2: Chaotische Lagerplatzzuordnung
Quelle: Wannenwetsch 2014, 301

2.5 Zonung

Bei der sogenannten Zonung handelt es sich um eine Mischform der beiden zuvor eingeführten Lagerplatzzuordnungsstrategien. Das Lager wird nach verschiedenen Kriterien in Bereiche aufgeteilt. Innerhalb dieser findet eine chaotische Lagerplatzzuordnung statt (vgl. Wannenwetsch 2014, 302).

2.6 Aktuelle Lagerhaltung

Die APS Glass and Bar Supply GmbH verfügt über drei Lagerstandorte mit unterschiedlichen Funktionen. Das Hauptlager übernimmt den Versand der meisten Bestellungen und lagert somit die derzeit meistgebrauchten Artikel. Die zwei weiteren Lager sind für die weniger frequentierten Artikel zuständig und dienen als zusätzlicher Lagerplatz. In dieser Arbeit wird lediglich das Hauptlager betrachtet.

Beim Hauptlager handelt es sich um ein Palettenregallager, welches zu den „am weitesten verbreiteten Lagersystem[en]“ (Gleißner/Femerling 2008, 91) zählt. Ein Einzelplatzsystem wird für die Einlagerung der Paletten verwendet, sodass pro Lagerplatz genau eine Palette eingelagert wird (vgl. Gleißner/Femerling 2008, 91).

Für die Verwaltung des Lagers wird das Prinzip der chaotischen Lagerplatzzuordnung genutzt. Die Regeln, nach denen die Einlagerung von neuen Waren stattfindet, sind nicht fest definiert, sondern basieren zu einem Großteil auf der Erfahrung der schon lange im Lager arbeitenden Angestellten.

Die in einem technischen Lagersystem vorkommenden Artikel lassen sich nach Pfohl in drei Arten der Lagergüter einteilen. Er unterteilt sie in Stückgüter, Schüttgüter sowie Gase und Flüssigkeiten (vgl. Pfohl 2018, 139). Durch die APS Glass and Bar Supply GmbH werden fast ausschließlich Stückgüter vertrieben. Es handelt sich bei den Artikeln um eine Mischung aus einem Standardsortiment und einem wechselnden Sortiment. Das schließt Artikel, die über einen großen Zeitraum immer wieder vertrieben werden und wechselnde Artikel, wie zum Beispiel während der Corona Pandemie diverse Hygieneartikel für Gastronomien, mit ein. Welche Artikel im Lager vorrätig sind, hängt stark von der Nachfrage ab. Daher bietet sich eine chaotische Lagerplatzzuordnung an. Allerdings birgt diese Art von Lagerhaltung viele Risiken. Die Annahmen über die Qualität der Lagerhaltung basieren auf Erfahrungen und Einschätzungen von Mitarbeitenden, aber nicht auf mathematischen Größen. Des Weiteren macht sich das Unternehmen sehr abhängig von einzelnen Mitarbeitenden,

sodass sich die Einarbeitung neuer Lagerangestellter schwierig gestaltet. Das Expertenwissen der Lagerarbeiter*innen lässt sich nicht einfach dokumentieren und für neue Mitarbeiter*innen erlernbar machen. Ein weiteres Problem stellt die Diskrepanz der Vorstellung der einzelnen Mitarbeitenden in Hinblick auf die Sortierung des Lagers dar. Alle Mitarbeiter*innen machen ihre eigenen Erfahrungen und haben ein eigenes System entwickelt, wie, warum und wo die neue Ware einlagert wird. Da es aber keine klaren Richtlinien gibt, wird es zwangsläufig zu unterschiedlichen Sortierungsstrategien kommen, welche nicht unbedingt harmonieren. So entsteht beispielsweise ein Gang mit Porzellan. Diese Einteilung nach Material oder Kategorie ist aus menschlicher Sicht sehr nachvollziehbar, mag allerdings in Hinblick auf die Kommissionierungskosten suboptimal sein.

Das Kommissionieren findet über Packlisten statt, die ohne Vorsortierung der Artikel ausgedruckt werden. Über einen Scanner können die Kommissionierenden die möglichen Lagerplätze des Artikels einsehen und die entsprechende Menge der Ware aus dem System ausbuchen. Je nach Größe des Auftrags können die Lagermitarbeiter*innen zu zweit oder allein an einem Auftrag arbeiten. Wenn die Artikelgröße oder -menge einer Palette bedarf, geht einer als Picker los und holt die Ware, während der andere diese auf der Palette stapelt und anschließend mit Folie umwickelt.

Es handelt sich um eine „Mann-zur-Ware-Kommissionierung (WzM)“ (Pulverich/Schietinger 2009, 424) für klein- bis mittelgroße Teile. Hierbei wird sowohl per Hand als auch mit der Unterstützung von Gabelstaplern kommissioniert.

3 Hauptteil

Im Hauptteil dieser Arbeit wird das Artikelanordnungsproblem nach dem OR-gestützten Planungsprozess (s. Abbildung 3), welcher dem Projekt eine klare und nachvollziehbare Struktur geben soll, bearbeitet:
Hierzu ist es notwendig, das Problem zunächst zu erfassen und zu analysieren. Alle „restriktiv wirkende[n] Daten, Aktionsparameter, funktionale[n] Zusammenhänge und Zielgrößen [müssen] ermittelt und quantifiziert“ (Koop/Moock 2018, 2) werden. In einem zweiten Schritt wird ein konkretes Ziel für die Optimierung festgelegt. Nun kann ein mathematisches Modell, welches das Problem möglichst genau beschreibt, entworfen werden. Um eine Lösung zu finden, müssen die relevanten Daten erhoben werden (vgl. Koop/Moock 2018, 2). Nach dieser Vorarbeit kann mit Hilfe eines Algorithmus oder einer mathematischen Optimierung eine Lösung gefunden werden. Abschließend werden die gefundenen Ergebnisse bewertet (vgl. Koop/Moock 2018, 2) und in Hinblick auf das bestehende Problem diskutiert. Der soeben beschriebene Prozess wird in der nachstehenden Abbildung visualisiert.

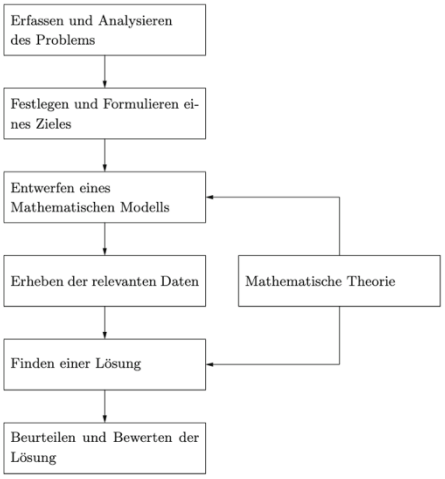

Abbildung 3: OR-gestützten Planungsprozess
Quelle: Koop/Moock 2018, 3

Für diese Arbeit muss der beschriebene Prozess leicht angepasst werden, da auf Grund der Problemdefinition, die im Folgenden stattfindet, keine exakten mathematischen Verfahren in Frage kommen. Daher wird angestrebt, anstelle eines mathematischen Modells einen Algorithmus zur Lösung des Problems zu finden.

3.1 Problemdefinition

Im ersten Schritt wird das Problem formal definiert. Die Literatur bezeichnet jenes als NP-schweres Artikelanordnungsproblem (Storage Location), was neben der Tourenplanung (Picker Routing) und der Auftragsbildung (Order Batching) zu den drei Teilproblemen der Kommissionierung gehört. Sollte die Menge von Lagereinheiten und Lagerplätzen genau gleich sein, so wird dieses Problem als „quadratic allocation problem“ (QAP) (vgl. Reyes et al. 2019, 200) definiert. Im Folgenden wird das Problem mit dem Namen SLAP (Storage Location Assignment Problem) benannt.
Auf Grund seiner Komplexität wird es in der Praxis häufig durch Heuristiken oder Meta-Heuristiken gelöst (vgl. Koch 2014, 15-27). Heuristiken und Meta-Heuristiken bilden das Gegenstück zu den exakten Verfahren. Sie können unter dem Begriff Approximation zusammengefasst werden.
Eine Heuristik beschreibt eine erfahrungsbasierte Problemlösungsstrategie. Anwendung finden diese, wenn exakte Verfahren zu aufwendig werden, um ein Problem in einem akzeptablen Zeitaufwand lösen zu können. Im Austausch für die optimale Lösung wird die Bearbeitungszeit drastisch gesenkt (vgl. Desale et al. 2015, 298). Meta-Heuristiken sind diejenigen Algorithmen, die einen Lösungskandidaten iterativ basierend auf einem Qualitätsmerkmal verbessern. Genauso wie die Heuristiken garantieren diese keine optimale Lösung (vgl. Desale et al. 2015, 298).

In der Lagerlogistik wird zwischen den im Grundlagenteil bereits vorgestellten Strategien, der festen und chaotischen Lagerplatzzuordnungen sowie der Zonung, unterschieden.
In dieser Arbeit wird von einer chaotischen Lagerplatzzuordnung ausgegangen, die durch die KI-Unterstützung optimiert werden soll. Im vorliegenden Lager herrschen eine Person-zu-Ware-Kommissionierung und ein dynamisches Sortiment. Das bedeutet, dass sich das Sortiment nach derzeitigen Trends aber auch aus saisonalen Gründen verändern kann. Deshalb bietet sich eine feste Zuordnung von Artikel zu Lagerplätzen nicht an.

3.2 Problemanalyse

Nach der Definition des Problems wird im OR-gestützten Planungsprozesses eine Problemanalyse durchgeführt:

Dafür werden zunächst einige Lösungsansätze, die bereits in anderen Arbeiten zur Lösung des Problems angewendet wurden, benannt:

Neben den exakten Verfahren (EX) werden die „Storage policies and Rules“ (SP&R), welche in vielen Fällen als Index bezeichnet werden, als häufig benutze Methoden zur Lösung des SLAP benannt (vgl. Reyes/Solano-Charris/Montoya-Torres 2019, 208). Mit Hilfe eines solchen Index kann das Lager jederzeit durch die Erhebung der entsprechenden Daten an Veränderungen im Sortiment angepasst werden. Darüber hinaus ist es möglich, mittels entsprechender SP&R einen Wareneingang zu realisieren, welcher einem Artikel den möglichst optimalen, freien Lagerplatz zuordnet, wodurch eine Sortierung implizit umgesetzt wird.
Ein prominentes Beispiel für solch einen Index ist der Cube per Order Index (COI). Dieser gilt als verbreitete Faustregel, die den Platzbedarf eines Artikels ins Verhältnis

zu seiner Zugriffshäufigkeit stellt. Bei einer COI basierten Zuordnung werden die Artikel nach ihrem COI aufsteigend sortiert und in dieser Reihenfolge denjenigen Lagerplätzen zugeordnet, die der Kommissionierungsstelle am nächsten sind (vgl. Malmborg/Bhaskaran 1990, 87). Das bedeutet, dass ein Artikel, der mehr Platz in Anspruch nimmt und somit durch die Besetzung eines guten Lagerplatzes vielen kleineren Artikeln den Platz wegnehmen würde, eine höhere Zugriffshäufigkeit aufweisen müsste, um einen ähnlichen COI-Wert zu erreichen wie ein kleinerer Artikel.

Ein anderer Sortierungsstrategie ist Order Oriented Slotting (OOS). Beim OOS sind ein leeres Lager, eine Menge von Bestellungen und eine Kommissionierungsstrategie als Ausgangspunkt gegeben. Mit Hilfe dieser Bestellungen werden die Artikel eingelagert, sodass die Kommissionierungskosten für die gegebenen Bestellungen minimiert werden (vgl. Mantel et al. 2007, 301).

Bei diesem Verfahren werden die Bestellungen zur Optimierung einer festen Lagerplatzzuordnung genutzt (vgl. Mantel et al. 2007, 304). Somit entspricht sie nicht dem in dieser Arbeit gestellten Anspruch einer Methode, die beim Wareneingang die Sortierung implizit durchführt, in Bezug auf ein dynamisches Sortiment und eine freie Lagerplatzzuordnung.

Währen COI zu den artikelorientierten Ansätzen zählt, ist OOS den bestellorientierten Ansätzen zuzuordnen. Beide Ansätze unterscheiden sich darin, dass artikelorientierte Ansätze Artikeldaten verwenden, um eine Sortierung des Lagers vorzunehmen, bestellungsorientierte Ansätze hingegen keine Artikeldaten berücksichtigen. Stattdessen werden historische Bestellungen verwendet, um das Lager so zu sortieren, dass diese Bestellungen möglichst geringe Kommissionierungskosten aufweisen. Ein weiterer Unterschied der benannten Lösungsansätze COI und OOS liegt in der Art der Kommissionierung, von der ausgegangen wird. Hierbei sind die „Single Command“ und die „Multi Command“ Kommissionierung anzuführen. Bei der Single Command beginnt der Picker am Input / Output (I/O), kommissioniert einen Artikel vom Lagerplatz und kehrt zum I/O zurück. Für diese Art der Kommissionierung ist der COI

geeignet. Im Gegensatz dazu steht die Multi Command Kommissionierung, bei der mehrere Artikel einer Bestellung in einem Durchlauf zusammengesammelt werden, bevor der Picker zum I/O zurückkehrt (vgl. Schuur 2014, 2). Hierfür stellt der OOS einen geeigneten Ansatz dar.
Eine Single Command Situation ist in vielen Fällen nicht gegeben und aus diesem Grund keine akzeptable Lösung für das vorliegende Problem.

Aus genannten Gründen sind die beiden vorgestellten Ansätze nicht direkt auf das in dieser Arbeit untersuchte Problem anwendbar. Trotzdem ist es möglich, bestimmte Kennwerte und Konzepte aus diesen Ansätzen abzuleiten: Als besonders relevant haben sich die im COI verwendeten Parameter „Zugriffsfrequenz“ und „Platzbedarf“ in Bezug auf die Erstellung einer effizienten Sortierung erwiesen. Daher kann man diese Informationen als Teil der Eingabedaten für den Algorithmus übernehmen. Die Berücksichtigung von Bestellungen hat sich in einem Multi Command Szenario bereits als hilfreich erwiesen und könnte daher als Ergänzung zu den Artikelinformationen sinnvoll sein.

3.3 Zielsetzung

Das Ziel der Arbeit besteht in der Optimierung der chaotischen Lagerhaltung bei der APS Glass und Bar Supply GmbH in Bezug auf ein sich dynamisch entwickelndes Sortiment. Hierbei soll eine Funktion entwickelt werden, welche die Zuordnung eines Artikels zu einem möglichst optimalen freien Lagerplatz ermöglicht. Diese Funktion soll Informationen über die verfügbaren Lagerplätze sowie den einzulagernden Artikel erhalten.

3.4 Lösungsansätze

Zur Lösung des Artikelanordnungsproblems stehen einige Heuristiken und Metaheuristiken zur Wahl. Exakte Verfahren kommen nur für kleine Problemgrößen in Frage und sind in dem konkreten Fall, der in dieser Arbeit untersucht wird, auf Grund der Komplexität, welche in der Literatur als NP-schwer festgestellt wird, nicht praktikabel (vgl. Koch 2014, 16). Im Folgenden werden einige dieser Methoden und ihre zu Grunde liegenden Konzepte vorgestellt. Anschließend wird eine Diskussion darüber geführt, welcher dieser Ansätze für die Lösung des vorliegenden Problems genutzt werden soll.

Zu Beginn ist eine wichtige Entscheidung in Bezug auf die Herangehensweise zu treffen, da das SLAP auf zwei unterschiedliche Weisen gelöst werden kann: Einerseits ist es möglich, eine feste Sortierung zu entwickeln, die in dem Lager angewendet oder durch eine abgeleitete Regel Stück für Stück umgesetzt wird. Dies ist die in der Literatur zumeist vertretene Strategie (vgl. Xie/Mei/Ernst/Li/Song 2014). Andererseits kann direkt eine Regel oder Funktion entwickelt werden, die das Einsortieren in das Lager beim Wareneingang übernimmt. Die Entwicklung einer festen Sortierung bietet sich im konkreten Falle dieser Arbeit weniger an, da in einem chaotischen Lager sowie bei einem dynamischen Sortiment eine feste Zuweisung von Artikeln zu einem bestimmten Lagerplatz nicht vorgesehen ist, um die ineffiziente Nutzung von Lagerplatz zu vermeiden. Eine implizite Sortierung durch eine Funktion ist vorteilhaft, da auf ein sich veränderndes Sortiment reagiert werden kann.

Im Folgenden werden einige Methoden zur Lösung von SLAP präsentiert. Die gefundenen Ansätze werden miteinander verglichen, damit zum Schluss eine Auswahl der Methode getroffen werden kann.

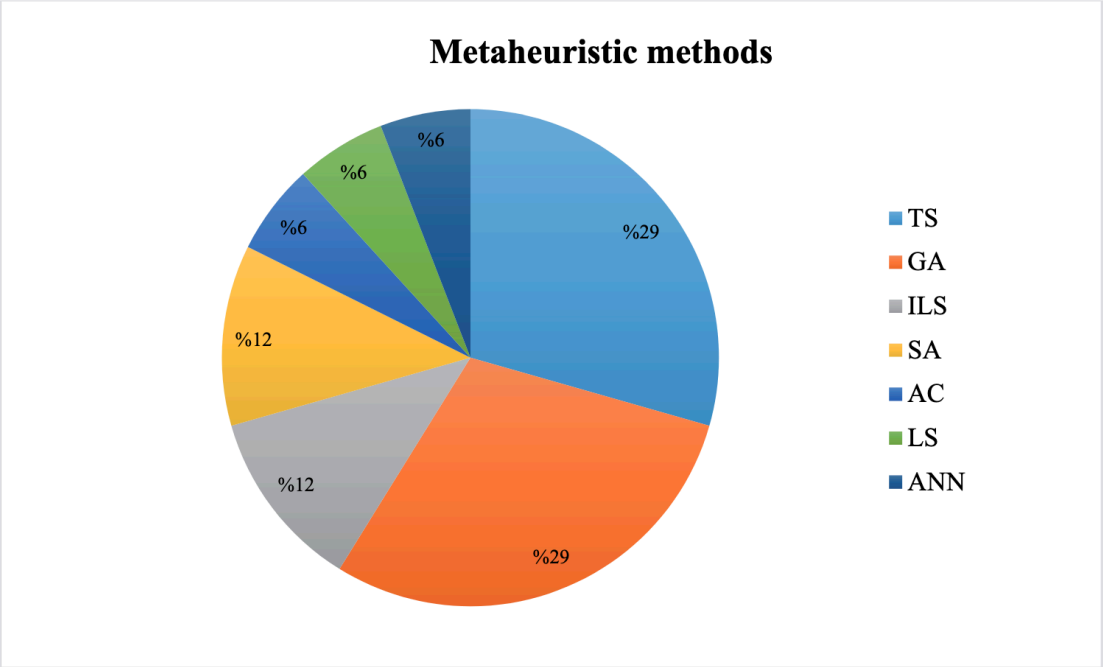

Abbildung 4: Verteilung von Metaheuristiken
Quelle: Reyes/Solano-Charris/Montoya-Torres 2019, 210

Aus dem vorangestellten Diagramm lassen sich die zwei beliebtesten Metaheuristiken nach dem „Literature Review“ von Reyes et al. herauslesen: Tabu Search (TS) und genetische Algorithmen (GA) (vgl. Abb.04). Diese beiden Algorithmen werden im Folgenden vorgestellt und miteinander verglichen. Auf die anderen Algorithmen, die im Diagramm aufgeführt werden, wird kein Bezug genommen.

3.4.1 Tabu Search

Tabu Search ist eine Metaheuristik, die eine andere Heuristik überlagert. Sie wurde 1986 von Glover erstmals vorgestellt und wird verwendet, um lokale Optima zu überwinden und ein globales Optimum zu finden. Dies wird durch das Verbieten oder das Bestrafen bestimmter Operationen umgesetzt. Der Zweck dieser Verbote besteht darin, Zyklen zu verhindern (vgl. Glover 1986, 541). Wenn eine Bewegung immer

wieder durchgeführt werden kann, wie zum Beispiel der Schritt zurück zu einem lokalen Optimum, da dieses eine bessere Lösung darstellt als die anderen Optionen, kommt es zu einem Zyklus, der das Finden eines globalen Optimums bei einem nicht konvexen Suchraum unmöglich werden lässt.

Oftmals wird das menschliche Verhalten durch ein gewisses Maß an Zufall definiert, das dazu führen kann, einen weniger optimalen Weg einzuschlagen und somit Fehler zu erlauben. Doch durch das Auftreten dieser vermeidlichen Fehler könnten lokale Optima überwunden werden, um eine bessere Lösung zu finden. Auf diese Weise operieren die stochastischen Optimierungsverfahren (vgl. Glover 1986, 541).

Tabu Search unterscheidet sich grundlegend von anderen künstlichen Intelligenzen, da aktiv kein Zufall zugelassen wird. Die Verwendung des Zufalls widerspräche dem Ziel, grundsätzlich immer die beste mögliche Lösung auszuwählen. Der Algorithmus operiert unter der Annahme, dass es keinen Mehrwert in einer schlechteren Lösung gibt. Die Ausnahme bilden die verbotenen Operationen, da diese generell nicht verwendet werden dürfen (vgl. Glover 1986, 541).

Die wichtigste Komponente des Tabu Search bildet die so genannte „Tabu List“. Innerhalb dieser werden Operationen typisiert und nach ihrer Richtung klassifiziert. Für jede dieser Klassen wird eine eigene Tabu List erstellt. Die meisten Probleme lassen sich mit einem Operatoren-Typen lösen, sodass die Tabu List nur durch die Richtung der Operation differenziert wird (vgl. Glover 1986, 541). Die Tabu List enthält die letzten m Operationen, wobei m als Parameter übergeben werden kann. Somit steuert m, wie lange eine Operation verboten ist. Um die Funktionsweise der Tabu List zu veranschaulichen, wird ein von Glover illustriertes Beispiel angeführt:

Bei einem ganzzahligen Optimierungsproblem existiert nur ein Operatoren-Typ, bei welchem eine Variable einen Wert adjazent zu einem derzeitigen Versuchswert zugewiesen bekommt. Da man die Variable entweder inkrementiert oder dekrementiert, können zwei Richtungen, hoch und runter, festgestellt werden. Somit werden analog dazu eine „hoch“ und eine „runter“ Tabu List erstellt. Eine Variable x wird beispielsweise von 4 auf 5 inkrementiert. So landet diese auf der „hoch“ Liste.

Damit wird gleichzeitig die entgegengesetzte Operation verhindert, womit x von 5 nicht wieder auf 4 dekrementiert werden kann. Die Variable x kann sowohl auf dieser Tabu List als auch auf anderen mehrfach gleichzeitig existieren (vgl. Glover 1986, 541-542).

3.4.2 Genetische Algorithmen

Genetische Algorithmen zählen zu den populationsbasierten Metaheuristiken und bilden eine Subkategorie der evolutionären Algorithmen. Sie repräsentieren unter anderem einen wichtigen Teil der „Computational Intelligence“ (vgl. Koch 2014, 59). Des Weiteren stellen sie innerhalb der evolutionären Algorithmen „den in Forschung und Anwendung dominierenden Ansatz dar“ (Koch 2014, 59).
Konzeptionell sind die genetischen Algorithmen wesentlich von der Natur inspiriert (vgl. Koch 2014, 60). Um die Lösungskandidaten namens Individuen, welche in der Summe die Population ergeben, weiterzuentwickeln, werden an die biologische Evolution angelehnte Methoden verwendet (vgl. Harbich 2007, 4). Ein Chromosom enthält in der Natur einen Bauplan für ein Lebewesen und stellt in genetischen Algorithmen meist ein Individuum dar (vgl. Koch 2014, 60).
Die Selektion wählt „zufällig diejenigen Chromosomen aus, die die nächste Generation der Population bilden“ (Harbich 2007, 7) werden. Jedes Individuum hat in Abhängigkeit seiner Fitness eine Chance, ausgewählt zu werden. Je angepasster das Individuum, desto höher die Fitness und somit auch die Chance, ausgewählt zu werden und an den genetischen Operationen teilzunehmen (vgl. Harbich 2007, 5 ff.). Die genetischen Operationen beinhalten Crossover und Mutation.
Beim Crossover werden aus je zwei Chromosom durch den zufälligen Austausch einiger ihrer Gene neue Chromosom gebildet. Jedes Chromosom wird hierbei genau einmal berücksichtigt. Zudem gilt, dass ein Chromosom nicht mit sich selbst gepaart werden kann (vgl. Harbich 2007, 7 f.).

Bei der Mutation hingegen werden bei einigen Chromosom der Population zufällig ein oder mehrere Gene verändert (vgl. Harbich 2007, 7 f.).

Der Ablauf einer Generation kann mit einer Evaluation der Fitness der Population mit anschließender Selektion und abschließenden genetischen Operationen beschrieben werden. Dabei kann der Algorithmus beliebig viele Generationen durchlaufen. Aus diesem Grund ist es notwendig, ein Abbruchkriterium, wie zum Beispiel eine maximale Anzahl an Generationen oder einen bestimmten Fitnessdurchschnittsschwellenwert, festzulegen (vgl. Harbich 2007, 8).

3.4.3 Genetische Programmierung

Die genetische Programmierung bezeichnet eine Methode aus der Familie der genetischen Algorithmen, welche Probleme automatisch, ausgehend von einem „High-Level Statement“, löst. Dies funktioniert, indem eine Population von Computerprogrammen evolutionär gezüchtet wird (vgl. Koza/Poli 2005, 1 ff.). Um den Unterschied zwischen den herkömmlichen genetischen Algorithmen und der genetischen Programmierung herauszustellen, kann hier das sogenannte Knappsackproblem als Beispiel herangezogen werden. Während ein Individuum bei einem genetischen Algorithmus eine mögliche Konfiguration für das Problem darstellt, handelt es sich im Falle der genetischen Programmierung um ein Programm, das solch eine Konfiguration erzeugen soll.
Die genetischen Operationen können, ähnlich wie bei genetischen Algorithmen, Crossover und Mutation sowie „Gene Duplication“ und „Gene Deletion“ beinhalten.
Die Programme werden als Syntaxbäume anstelle von einzelnen Codezeilen repräsentiert. Dabei setzt sich solch ein Syntaxbaum aus „Nodes“ (Knoten) und „Links“ (Verbindungen) zusammen. Die Nodes repräsentieren den Operanden, die Links verbinden die Parameter mit den Nodes. Interne Nodes werden als „Functions“

und die Blätter als „Terminals“ bezeichnet. Dies liegt daran, dass jedes Blatt einen Wert und jeder interne Node einen Operanden enthalten muss (vgl. Koza/Poli 2005, 1 ff.). Die folgende Abbildung stellt ein Beispiel von Koza/Poli (2005) eines Syntaxbaums zu der Funktion $max((x * x)(x + (3y)))$ dar.

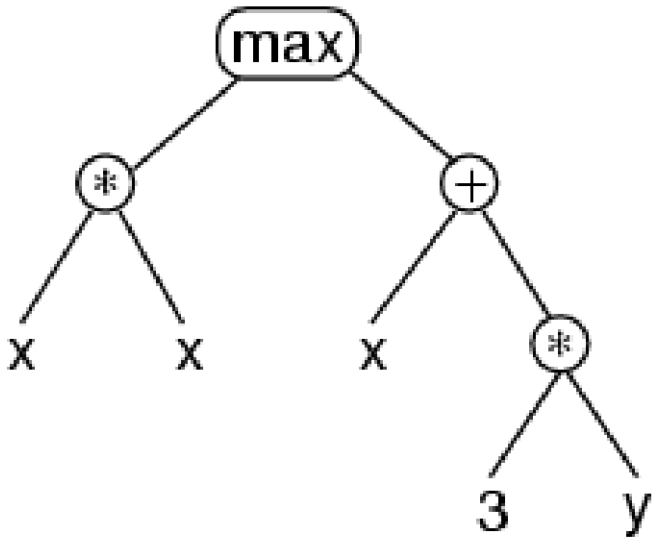

Abbildung 5:
Quelle: Koza/Poli 2005

Jene haben fünf großen Vorbereitungsschritte, die vom Menschen vor dem Start des Algorithmus durchgeführt werden müssen, definiert:
Der erste Schritt befasst sich mit der Definition der Terminals. Die Terminals beschreiben die Blätter des Baumes und stellen Variablen oder Konstanten dar. Gemeinsam mit dem zweiten Schritt, der die Functions definiert, werden hier alle Bausteine, welche dem Algorithmus zum Erstellen eines Programms zur Verfügung stehen, bestimmt. Durch diese soll eine diverse Population erzeugt werden können (vgl. Koza/Poli 2005, 3 f.).
Der nächste Schritt beinhaltet das Erstellen einer Fitness-Funktion. Diese gibt das High-Level Statement, das von der genetischen Programmierung gelöst werden soll,

vor. Sie bemisst, wie gut ein Lösungsprogramm ist und dient zum Vergleich der verschiedenen Individuen (vgl. Koza/Poli 2005, 3).
Die letzten beiden Schritte beziehen sich auf die Ablaufkontrolle. Dies umfasst alle anwendungsspezifischen Parameter, die den Verlauf beeinflussen, sowie das Abbruchkriterium, welches, wie schon bei den genetischen Algorithmen, durch eine maximale Generationsanzahl oder Durchschnittsfitness festgelegt werden kann. Zu den wichtigsten Ablaufparametern zählen die Populationsgröße, die maximale Programmgröße und die Wahrscheinlichkeiten des Auftretens der genetischen Operationen (vgl. Koza/Poli 2005, 5).

Genetische Programmierung wurde bereits auf SLAP angewendet. Durch die Nutzung von genetischer Programmierung konnte eine sogenannte „Matching Function“, die ein Satz von Artikeln einem Satz von Lagerplätzen zuordnet und somit eine Lagersortierung vornimmt, entwickelt werden (vgl. Xie/Mei/Ernst/Li/Song 2014). Wie Xie et al. (2014) bereits feststellten, besteht ein größerer Wert für die Zukunft darin, eine Funktion zu erzeugen, welche Artikel beim Wareneingang freien Lagerplätzen zuordnet, als eine feste zu dem Zeitpunkt optimale Sortierung zu ermitteln. Diese Sortierung kann bereits in naher Zukunft nicht mehr annährend optimal sein.

3.4.4 Diskussion

Im Fokus dieser Arbeit steht die Flexibilität der Lösung und die damit einhergehende Nachhaltigkeit. Sowohl genetische Algorithmen als auch die Tabu Search stellen beliebte Lösungsansätze für SLAP dar. Jene suchen allerdings statische Lösungen. Es ist nicht zu verkennen, dass diese Lösungen bereits gute Ergebnisse geliefert haben. Daher wäre es denkbar, solch eine Lösung zu verwenden, um das Lager schrittweise an diese anzupassen. Dabei stellt das dynamische Sortiment die größte Herausforderung dar. Durch wechselnde Artikel aber auch durch eine schwankende

Größe des Sortiments wäre eine statische Lösung wahrscheinlich bereits zu dem Zeitpunkt, zu dem diese umgesetzt werden könnte, irrelevant. Das liegt in der langsamen Umsetzung dieser Lösungen begründet. Um solch eine statische Sortierung in einem Lager zu realisieren, gibt es zwei Ansätze. Der erste beinhaltet das Stoppen des Arbeitsbetriebes und eine vollständige Umsortierung, sodass alle Artikel an dem vorgesehenen Lagerplatz eingelagert sind. Das kommt für die meisten Unternehmen genauso wie für die APS Glass und Bar Supply GmbH nicht in Frage. Daher muss eine andere Strategie genutzt werden. Diese würde die Sortierung als Vorbild nehmen und die Artikel vom Wareneingang an den bestmöglichen freien Lagerplatz nach der vorgegebenen Sortierung einlagern. Auf diese Weise dauert der Prozess so lange, dass sich bei einem dynamischen Sortiment bereits so viel verändert haben könnte, dass die Sortierung irrelevant geworden wäre. Insgesamt ist festzuhalten, dass eine Veränderung des Sortiments oder der Lagerstruktur die Lösung nicht beeinflussen sollte. Vor diesem Hintergrund eignen sich die beiden Ansätze, genetische Algorithmen und Tabu Search, für den Anwendungsfall dieser Arbeit nicht.

Anders verhält es sich bei der genetischen Programmierung. Diese hat den Vorteil, vielseitige Lösungen finden zu können. Auf Grund derer könnte ein Programm generiert werden, welches die Lösung für diesen sehr speziellen Anwendungsbereich liefert. Diese Lösung könnte zum Beispiel eine wie bereits erwähnte Matching Function sein oder eine neue Heuristik, die dynamisch auf das Sortiment angewendet werden kann. Da mit dieser Methode ein Lösungsweg und nicht eine konkrete Lösung erzeugt wird, erweist sich genetische Programmierung als die flexibelste Methode und wird im Folgenden umgesetzt.

4 Implementierung

Die Implementierung des „Tree-Based Genetic Programing“ Ansatzes wird mit Hilfe der Programmiersprache Lisp umgesetzt. Der Name Tree-Based ergibt sich durch die Darstellung der Programme in dem Algorithmus. Dabei werden Funktionsbäume erzeugt, welche wie Programme zu verstehen sind. Jeder Knoten ist eine Funktion oder ein Terminal. In dieser Anwendung werden nur mathematische Operationen als Funktionen verwendet. Daher sind die Bäume als Terme zu verstehen, wobei die Terminals Variablen sind, welche zur Ermittlung eines Matching Scores durch die Daten des Artikels und des Lagerplatzes ersetzt werden müssen. Lisp bietet einige Vorteil für die Entwicklung dieses Algorithmus, da die Methodendarstellung nativ eine Baumstruktur nutzt und somit ohne zusätzlichen Aufwand übernommen werden kann. Des Weiteren finden sich viele Referenzprojekte in Lisp, da eine große Community zu Lisp vor allem im Bereich der künstlichen Intelligenz existiert.

Im Folgenden werden die Grundbausteine des Algorithmus vorgestellt und deren konkrete Implementierung erläutert.

4.1 Populationserzeugung

Der erste Schritt zur Erzeugung einer Population besteht in der Erzeugung eines Individuums. Dieses setzt sich wiederum aus den Grundbausteinen, die durch Funktionen und Terminale beschrieben werden, zusammen. Dabei sind die Funktionen in dieser Implementierung einfach gehalten und bestehen aus den grundlegenden mathematischen Operationen $+, -, *, /$. Jede dieser Funktionen wird mit einer „Arity“ (deutsch: Stelligkeit) von 2 definiert, sodass je zwei Kindsknoten an eine Funktion angehängt werden. Zusätzlich gibt es noch die Negierung, die eine Arity von 1 aufweist. Die Terminale werden sowohl durch die Artikel- als auch Lagerplatzdaten

definiert. Dabei besteht ein Lagerplatz aus einer ID und drei Koordinaten, welche die Distanz zum I/O darstellen. Da die ID nur zur Identifikation genutzt wird, wird sie nicht als Terminale verwendet. Ähnlich verhält es sich bei den Artikeln: Zur Identifikation werden die Artikelnummer sowie die ArtikelID herangezogen. Die verbleibenden Artikeldaten werden als Terminale benutzt. Sie bestehen aus Informationen zu Größe, Gewicht, Kommissionierungsfrequenz, Verpackungseinheit, Menge und durchschnittlicher Liegedauer. Diese stammen aus der SQL-Datenbank, die für die späterer Nutzung direkt an die Anwendung angebunden werden muss. Allerdings werden die Daten für die Tests in JSON-Dateien abgelegt. Hierdurch werden der Umweg über die Datenbank und die Inkonsistenz der Daten in einer Live-Datenbank eingespart. Dies ist möglich, da nur wenige Artikel gebraucht werden, wodurch lediglich eine geringe Menge an Daten abgespeichert werden muss.

Lagerplatz Terminale	**Beschreibung**
ID	Einzigartiger Identifikator für jeden Lagerplatz
dist_x	Distanz vom Kommissionierungstisch auf der x-Achse
dist_y	Distanz vom Kommissionierungstisch auf der y-Achse
dist_z	Distanz vom Kommissionierungstisch auf der z-Achse

Artikel Terminale	Beschreibung
ID	Einzigartiger Identifikator für jeden Artikel
Liegedauer	Durchschnittliche Liegedauer eines Artikels von Ankunft beim Wareneingang bis zum Verlassen des Lagers nach dem Abverkauf
SUM	Summe aller Verkaufter Einheiten dieses Artikels in den letzten 2 Jahren
Gewicht	Gewicht einer Einheit des Artikels
Box	Box des Artikels für mehrere Einheiten mit Werten zur Länge, Breite, Höhe, Volumen und der Menge an Einheiten des Artikels, die zu einer vollen Box gehören.
Box_Qnty	Menge des Artikels in einer Box
Box_Length	Länge der Box
Box_Width	Breite der Box
Box_Height	Höhe der Box
Box_Volume	Volumen der Box
Quantity	Menge des Artikels die für die Platzbedarf-Berechnung in der Fitnessfunktion gebraucht wird

Im nächsten Schritt ist es notwendig, Methoden zur Generierung von zufälligen Individuen zu definieren. Um eine größere Diversität von Programmen zu

gewährleisten, werden hierzu klassischerweise zwei verschiedene Methoden verwendet: die Grow- und die Full-Methode.

Bei der Grow-Methode werden Funktionsbäume einer zufälligen Tiefe erzeugt. Limitiert wird dieser Prozess durch eine maximale Tiefe des Baumes. Es wird eine zufällige Funktion von den zu Beginn definierten mathematischen Operationen ausgewählt. Die Kindsknoten dieser werden entweder durch einen rekursiven Aufruf der Methode erzeugt oder durch eine zufällige Terminale gesetzt. Die Tiefe wird durch einen in der Rekursion übergebenen Tiefenparameter festgestellt. Spätestens beim Erreichen der maximalen Tiefe wird ein zufälliges terminales Symbol als Kindsknoten gewählt. Ein möglicher durch die Grow-Methode erzeugter Baum sieht wie folgt aus:

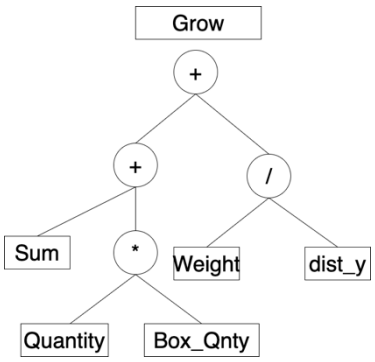

Abbildung 6: Grow Beispiel

Die Full-Methode funktioniert auf ähnlich Weise. Der Unterschied zur Grow-Methode besteht darin, dass die maximale Tiefe in jedem Fall erreicht werden muss. Dabei werden so lange Funktionen durch rekursive Aufrufe der Methode erzeugt, bis die maximale Tiefe erreicht wird. Erst dann wird eine Terminale zufällig gewählt. Ein möglicher durch die Grow-Methode erzeugter Baum sieht wie folgt aus:

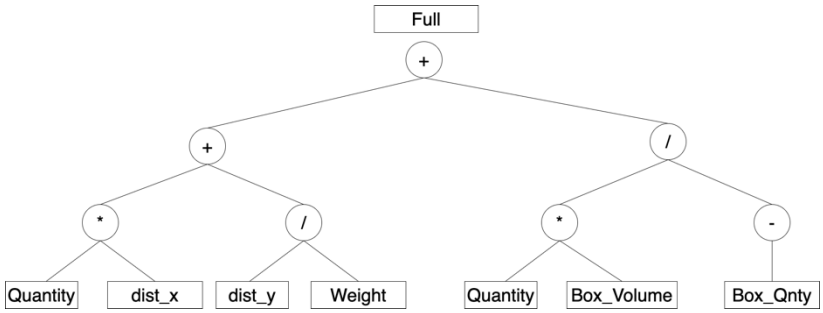

Abbildung 7: Full Beispiel

Um die initiale Population zu erzeugen, werden 50% der Individuen mit der Full-Methode und die Verbleibenden mit der Grow-Methode erstellt. Dies wird auch als „Ramped Half and Half“ (RHH) bezeichnet (vgl. Walker 2001, 4).

4.2 Fitnessfunktion

Zur Implementierung der Fitnessfunktion kommen für diesen konkreten Anwendungsfall zwei verschiedene Ansätze in Frage: Im Ersten wird ein leeres Lager von Grund auf neu einsortiert. Dies entspricht zwar nicht dem Endziel der Funktion, bietet aber eine intuitivere Bewertung der Lösung. Der zweite Ansatz bezieht sich auf ein bereits befülltes Lager, das einige freie Lagerplätze enthält. Dieses Szenario entspricht direkt dem späteren Einsatz der Funktion und ist daher naheliegend. Außerdem ließe sich in diesem Ansatz eine Artikelkorrelation besser einbeziehen, da die benachbarten Lagerplätze zumeist bereits belegt sind. Im Hinblick auf die Leistung bietet der zweite Ansatz auf Grund einer geringeren Anzahl zuzuordnender Artikel einen wesentlichen Vorteil.
Um einen Artikel zuzuordnen, müssen zunächst alle verfügbaren Lagerplätze ermittelt werden. Das erfolgt durch das Ermitteln des verbleibenden freien Platzes pro Lagerplatz. Jeder einzuordnende Artikel hat eine Menge und ein Platzbedarf pro Box.

Somit können auch mehrere Artikel dem gleichen Lagerplatz zugeordnet werden. Mit Hilfe der kultivierten Funktion wird ein Matching-Score zwischen je einem Lagerplatz und dem Artikel gebildet. Der Artikel wird nun in demjenigen Lagerplatz eingelagert, der den besten Matching-Score erhalten hat. Alle einzulagernden Artikel werden in zufälliger Reihenfolge durch diesen Prozess in das Lager einsortiert.

Eine konkrete Bewertung der Lösung findet über die Manhattan Distanz statt. Diese berechnet die Distanz zweier Punkte mittels der Summe der absoluten Differenz der einzelnen Koordinaten. Die Bewegung des Kommissionierenden finden in diesem Modell ausschließlich entlang je einer der x, y, z Achsen statt. Diagonale Bewegungen sind somit nicht möglich (vgl. Malkauthekar 2013, 2).

In der Implementierung dieser Arbeit wird ein leeres Lager durch die Individuen befüllt. Konkret wird in einer Fitnessevaluation über jeden Artikel iteriert. Da beim ersten Artikel noch alle Lagerplätze frei sind, beim letzten hingegen nur noch einige wenige, werden die Artikel vor jeder Fitnessevaluation gemischt, sodass immer andere Artikel zuerst zugeordnet werden. Die Zuordnung findet über den Matching Score, den die Funktion vergibt, statt. Die Berechnung des Matching Scores für einen Lagerplatz mit $dist_x = 5, dist_y = 3, dist_z = 2$, einen Artikel mit $sum = 300, weight = 2.5$ mit dem Funktionsbaum $((sum - dist_x) * (weight + (dist_y - dist\_z)))$ findet durch das Einsetzen der Parameter in unsere Funktion statt. Der Matching Score dieses Lagerplatz -Artikel-Paares errechnet sich nun wie folgt:

$$\Big((300 - 5) * \big(2.5 + (3 - 2)\big)\Big) = 1032{,}5$$

Für einen Artikel wird zunächst ermittelt, welche Lagerplätze noch frei sind. Für jeden freien Lagerplatz wird der Matching Score berechnet. Der Artikel wird anschließend an dem Lagerplatz mit dem besten Matching Score eingelagert. Sobald alle Artikel eingelagert wurden, kann die Fitness mittels der durchschnittlichen Manhattan Distanz feststehender Bestellungen ermittelt werden.

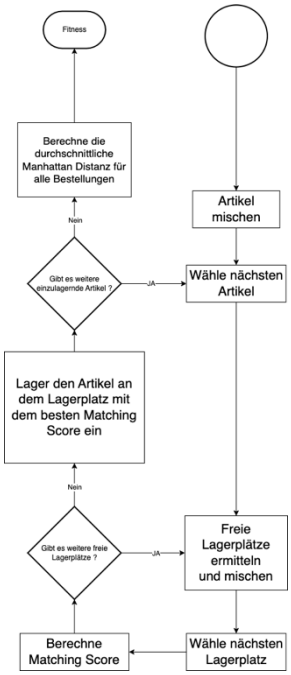

Abbildung 8: Fitnessfunktion Flowchart

Die Fitness wird für jedes Individuum einer Population ermittelt. Mit dieser kann im Folgenden die Selektion durchgeführt werden.

4.3 Selektion

Die Selektion dient der Steuerung der Entwicklung der Population. Durch die Auswahl der Eltern, die durch genetische Operationen Individuen der nächsten Generation erzeugen, wird bestimmt, welche Gene weitergegeben werden. Um einerseits die vielversprechendsten Gene in die nächste Generation einfließen zu lassen und andererseits nicht solche, die Potential haben könnten, sofern sie weiter kultiviert werden würden, aber nicht zu einer guten Fitness führen, zu verlieren, bedarf es einer Selektionsstrategie, die diese Aspekte berücksichtigt.

Im Folgenden werden einige Selektionsstrategien, die während der Entwicklung des Algorithmus getestet wurden, vorgestellt.

4.3.1 Tournament Selection

Ein weitverbreiteter Ansatz zur Realisierung der Selektion stellt die „Tournament Selection“ dar. Diese ist in zwei Schritte unterteilt, die mit „Sampling“ und „Selection“ benannt werden. Beim Sampling wird eine Gruppe von k Individuen aus einer Population der Größe N entnommen. Die gezogene Gruppe stellt ein Tournament der Größe k dar. Dieser Ziehung liegt ein Urnenmodell mit Zurücklegen zu Grunde. Im zweiten Schritt, der Selection, wird das Individuum mit der besten Fitness pro Tournament für die genetischen Operationen ausgewählt (vgl. Fang/Li 2010, 182 f.).
Auf Grund ihrer relativen Einfachheit kann diese Art der Selektion schnell und unkompliziert implementiert werden. Da keine vorherige Sortierung der Population, wie bei einigen anderen Selektionsmethoden, benötigt wird, bietet diese eine hohe Effizienz, welche bei komplexeren Anwendungsfällen, wie es bei GP oft der Fall ist, von großer Bedeutung ist. Die Komplexität kann mit $O(N)$ festgestellt werden. Aufbauend auf dem Grundgerüst der Tournament Selection wurden viele Sampling und Selection Methoden erforscht (vgl. Fang/Li 2010, 183).
Ein bekanntes Problem der Tournament Selection stellt das „Multi-Sampled Issue“ dar. Es beschreibt das mehrfache Auswählen eines Individuums während der Sampling-Phase. Da das Sampling auf einem Urnenmodell mit Zurücklegen basiert, kann ein Individuum mehrfach für ein Tournament ausgewählt werden. Der Einfluss, den dieses Problem auf den Verlauf der genetischen Programmierung nimmt, kann nicht genau bestimmt werden. Allerdings besteht für einige Forscher Grund zur Annahme, dass das mehrfache Auswählen eines Individuums die Chancen einiger anderer, ausgewählt zu werden, schmälert. Ein möglicher Lösungsansatz besteht in einem Urnenmodell ohne Zurücklegen. Hier werden die Individuen während des Samplings für ein Tournament

nicht direkt zurück in die Population gegeben, wodurch die doppelte Auswahl eines Individuums effektiv verhindert wird. Sobald die Selektion durchgeführt und ein Gewinner des Tournament ermittelt wurde, werden alle Individuen zur Population wieder hinzugefügt (vgl. Xie et al. 2008, 1323). Durch Simulation und Analyse der beiden Herangehensweisen kommen Xie et al. zu dem Schluss, dass der Unterschied zwischen dem Sampling mit oder ohne Zurücklegen nicht signifikant für das Ergebnis der gesamten Tournament Selection darstellt (vgl. Xie et al. 2008, 1323). Auf Grund der beschriebenen Insignifikanz stellt sich keine offensichtlich bessere Lösung dar. In dieser Implementierung wird ein Urnenmodell mit Zurücklegen verwendet, um jedem Individuum die Chance zu geben, für die genetischen Operationen ausgewählt zu werden. Dies wäre im Falle des Urnenmodells ohne Zurücklegen nicht möglich, da dort das schlechteste Individuum selbst bei einer kleinen Tournament Größe von $k = 2$ nicht ausgewählt werden kann. Andersherum hat jedes Individuum die Chance, ausgewählt zu werden, auch wenn diese mit sinkender Fitness abnimmt.

Der Selektionsdruck der Tournament Selection lässt sich einfach durch die Tournament-Größe anpassen (vgl. Fang/Li 2010, 183). Beliebte Größen sind bei allgemeinen genetischen Algorithmen $k = 2$, für genetische Programmierung $k = 7$ (vgl. Luke/Spector 1997, 2). Daher werden diese in späteren Experimenten verglichen.

4.3.2 Fitness-Proportionate Selection

Die „Fitness-Proportionate Selection“ verwendet die Fitness, um die Auswahlwahrscheinlichkeit der Individuen proportional zu ihrer Bewertung zu gestallten. Hierzu wird zunächst eine normalisierte Fitness ermittelt, welche sich aus der rohen Fitness durch einige Umwandlungen wie folgt erzeugen lässt: Zuerst muss die rohe Fitness standardisiert werden. Das dient dazu, die Fitness von Maximierungs- und Minimierungsproblemen gleichzusetzen. Bei einem Maximierungsproblem bildet

sich die standardisierte Fitness aus der maximalen rohen Fitness abzüglich der rohen Fitness des jeweiligen Individuums, sodass eine niedrigere Fitness ein besseres Individuum auszeichnet. Entgegengesetzt dazu wird bei einem Minimierungsproblem die rohe Fitness genutzt. Da nun die Unterschiede der beiden Optimierungsarten entfernt wurden, kann die angepasste Fitness folgendermaßen gebildet werden:

$$adj(i) = \frac{1}{1 + std(i)}$$

Hierbei stellt $adj(i)$ die angepasste Fitness eines Individuums i und $std(i)$ die standardisierte Fitness dessen dar. Um zum Schluss nun die normalisierte Fitness zu erhalten, muss folgende Formel angewandt werden:

$$norm(i) = \frac{adj(i)}{\sum_{k=1}^{M} adj(k)}$$

Hierbei stellt $norm(i)$ die normalisierte Fitness von einem Individuum i und M die Anzahl an Individuen in der gesamten Population dar. Im Anschluss werden die Individuen in einer Liste sortiert, sodass das Individuum mit der besten normalisierten Fitness am Anfang steht. Nun wird ein Wert r, der zwischen 0 und 1 liegt, zufällig erzeugt. Zum Schluss wird die Liste von oben nach unten durchlaufen und die normalisierte Fitness aufsummiert. Sobald diese Summe r übersteigt, wird das derzeitig betrachtete Individuum ausgewählt (vgl. Walker 2001, 6 f.).

4.3.3 Greedy Over-Selection

Bei der „Greedy Over-Selection“ werden die Individuen nach ihrer Fitness ausgewählt, wobei diejenigen mit der besten Fitness bevorzugt werden. Abermals wird die normalisierte Fitness gebildet (s. Fitness-Proportionate Selection), um zwei Gruppen zu erhalten. Die besten 20% der Individuen werden der ersten Gruppe zugeteilt. Die zweite Gruppe ergibt sich aus den restlichen 80%. Mit einer Wahrscheinlichkeit von 50% wird aus der ersten Gruppe gewählt, wodurch eine höhere

Auswahlwahrscheinlichkeit für die fittesten Individuen entsteht. Innerhalb der Gruppen findet eine Fitness-Proportionate Selection statt.

4.4 Genetische Operationen

Die ausgewählten Individuen werden mit Hilfe der genetischen Operationen zu einer neuen Generation verarbeitet. Die genetischen Operationen bestehen, wie bereits bei der Erläuterung zur genetischen Programmierung erwähnt, aus Crossover und Mutation. Gene Duplication und Gene Deletion sind in dieser Implementierung nicht berücksichtigt. Nach der Selektion werden die neuen Individuen mit einer 90-prozentigen Wahrscheinlichkeit durch Crossover gebildet. Die Verbleibenden werden durch Mutation erzeugt.

4.4.1 Crossover

Crossover stellt in dieser Arbeit die Hauptquelle zur Entstehung neuer Individuen dar. Hierzu werden je zwei Individuen der alten Generation gekreuzt. Es gibt verschiedene Arten von Crossover, die auf unterschiedliche Weise das genetische Material der Eltern vermischen. Anzuführen sind dabei das Standard-Crossover und das One-Point Crossover. Dies werden im Folgenden kurz erläutert und anschließend verglichen.

4.4.1.1 Standard Crossover

Beim „Standard Crossover“ werden zunächst zufällige Unterbäume beider Eltern ausgewählt. Diese werden vertauscht, um neue Bäume zu erzeugen (vgl. Poli/Langdon 1998, 294). In der konkreten Implementierung dieser Crossover-Methode wird nur ein neuer Baum erzeugt, bestehend aus dem gesamten Baum des ersten Elternbaumes ohne den ausgewählten Unterbaum und dem gewählten Unterbaum des zweiten Elternbaumes (s. Abb. 06).

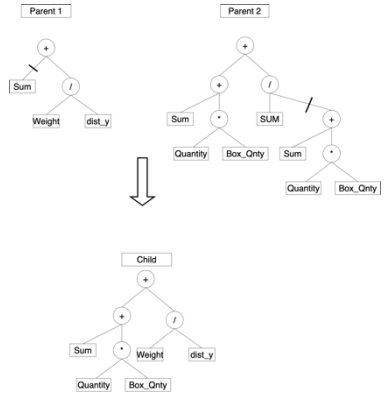

Abbildung 9: Standard Crossover Beispiel

4.4.1.2 One-Point Crossover

„One-Point Crossover“ bei genetischer Programmierung strebt an, die gleichen Schritte wie das One-Point Crossover in genetischen Algorithmen durchzuführen. Bei vielen genetischen Algorithmen sind die Individuen binär kodiert und werden in Strings abgespeichert. Das One-Point Crossover legt diese übereinander und sucht einen gemeinsamen Schnittpunkt, an dem die Individuen geteilt werden (s. Abb. 07). Die abgetrennten Teile werden vertauscht, um zwei neue Nachkommen zu erzeugen (vgl. Selzam 2003, 10).

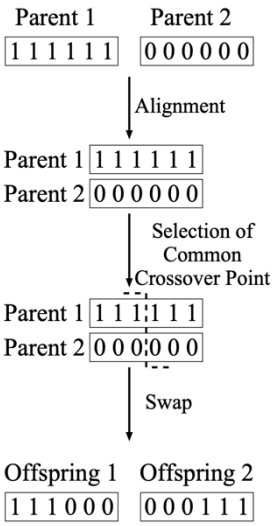

Abbildung 10: One-point Crossover in Binary GAs
Quelle: Poli/Langdon 1998, 294

Dieser Prozess funktioniert in der genetischen Programmierung, sofern die beiden Funktionsbäume die gleiche Form sowie Größe aufweisen, genauso. Um dies ebenso auf alle Funktionsbäume mit ungleicher Form und Größe anwenden zu können, muss eine „Common Region“ gefunden werden. Dazu wird geprüft, welche Links übereinstimmen. Innerhalb dieser Common Region können nun ein Schnittpunkt gewählt und die Unterbäume vertauscht werden, um somit zwei Nachkommen zu erzeugen (vgl. Poli/Langdon 1998, 2 f.).

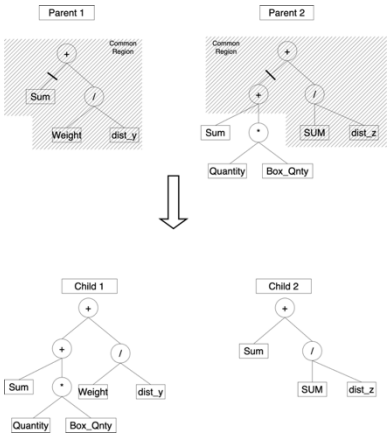

Abbildung 11: One-Point Crossover Beispiel

4.4.1.3 Vergleich von One-Point- und Standard Crossover

Die beiden Crossover Methoden haben einen vergleichbaren Austausch von genetischem Material. Über diese Metrik lassen sich keine aussagekräftigen Schlüsse über den Unterschied der beiden Methoden ziehen. Der wesentliche Unterschied besteht in der Art der Suche. Standard Crossover wird oft als lokaler Suchalgorithmus bezeichnet. Beim One-Point Crossover verändert sich die Suche im Verlauf des Algorithmus. Zunächst ist die Wahrscheinlichkeit, einen Unterbaum nahe am Wurzelknoten zu wählen, bedeutend höher, wenn eine gewisse Diversität in der Population besteht. Das ist damit zu begründen, dass sich die Individuen sehr unterscheiden und die Common Region somit kleiner ausfällt. Dies mündet darin, dass zu diesem Zeitpunkt globaler gesucht wird. Im späteren Verlauf der Suche werden die Individuen allerdings immer ähnlicher und die Common Region wächst, wodurch sich der Schnittpunkt für das Crossover tiefer im Baum befinden kann. Somit konvergiert die Suche und wird mit der Zeit immer lokaler (vgl. Poli/Langdon 1998, 5).
In der Implementierung für die Versuche wird das Standard Crossover verwendet, da sich diese Methode intuitiver programmieren lässt. Das aufwendigere One-Point Crossover ist trotz alledem ein interessanter Ansatz für zukünftige Simulationen oder Verbesserungsansätze des Algorithmus.

4.4.2 Mutation

Der Nutzen von Mutation in der genetischen Programmierung ist umstritten. Einige Forscher gehen davon aus, dass kein Nutzen in der Mutation liegt. Andere wiederum erkennen das Verfahren als einen wichtigen Teil der genetischen Operationen an (vgl. Piszcz/Soule 2006, 951).

Auf Grund dessen wird die Mutation in dieser Arbeit durch eine einfache Implementierung berücksichtigt, allerdings wird es keine ausführlichen Versuche zur Bestimmung des Nutzens der Mutation geben.
Die intuitive Implementierung besteht darin, den gleichen RHH-Ansatz (s. Kapitel 4.1) durch die Full- und Grow-Methode zu nutzen, um einen Baum zu erzeugen. Dieser Baum wird anstelle eines zufälligen Unterbaumes des zu mutierenden Individuums eingesetzt.

4.5 Bloat

Im Folgenden wird eins der grundlegenden Probleme der genetischen Programmierung erläutert: „Bloat“.
Im Zentrum dessen steht das unkontrollierte Wachstum von Individuen unabhängig von nennenswerten Fitnessverbesserungen (vgl. Luke/Panait 2006, 1). Dies birgt zwei verschiedene Teilprobleme: Einerseits führt das allgemeine Gesamtwachstum der Population zu einer aufwendigeren Evaluation und gestaltet die Ausführung des Algorithmus ressourcenintensiv. Andererseits besteht ein Problem darin, dass das finale Programm unnötig groß ist. Luke und Panait (2006) sehen eine Relevanz in der Lösung der beiden Aufgaben, benennen allerdings ersteres als das für die meisten Experimente relevantere Problem (vgl. Luke/Panait 2006, 2).
Innerhalb dieser von Bloat betroffenen, wachsenden Programme bilden sich oftmals Unterbäume, die keine Funktion erfüllen. Diese werden als Introns bezeichnet. Eine spezielle Form jener ist „Inviable Code“, der durch ein „Invalidator“, wie zum Beispiel „* 0“, verursacht wird und dafür sorgt, dass sämtliche seiner Unterbäume keine Auswirkung auf die Fitness haben können. Trotz allem muss der gesamte Unterbaum unterhalb eines Invalidators evaluiert werden und beansprucht unterdessen viel Rechenzeit, ohne die Fitness zu verändern (vgl. Luke/Panait 2006, 3).

Eine bekannte Theorie zu Introns ist die „Defense Against Crossover“, welche besagt, dass Introns sich durchsetzen, da es beim Crossover schwieriger ist, die Fitness des Individuums zu schädigen. Sollte Inviable Code für Crossover ausgewählt werden, so verändert dies nicht die Fitness des Individuums. Somit können Introns als Schutzmechanismus angesehen werden, welche die Wahrscheinlichkeit einer Verschlechterung der Fitness verringern (vgl. Luke/Panait 2006, 3).
Luke und Panait (2006) beschreiben die genetische Programmierung als ein Rennen gegen die Zeit. Dieses Rennen besteht in der Suche nach der besten Lösung, bevor Bloat es unmöglich macht, weiterhin effizient eine bessere Lösung zu finden.
Die Auswirkungen von Bloat waren in den ersten Tests dieses Projekts bereits deutlich zu sehen. Mit jeder Generation benötigt der Algorithmus mehr Zeit, um eine weitere durchzuführen. Außerdem lässt sich beobachten, dass die besten Individuen immer weiterwachsen.
Bei der Anwendung von Bloat-Kontrollmethoden muss nicht nur darauf geachtet werden, die durchschnittliche Größe der Bäume klein zu halten, um die Systemauslastung niedrig zu halten, sondern auch darauf, nicht zu sehr in den Algorithmus einzugreifen. Wenn durch zu starke Kontrolle von Bloat ein Bias zu kleinen Bäumen entsteht, könnten Lösungen, die größere Bäume benötigen, nicht mehr gefunden werden. Im Extremfall würde das zur Folge haben, dass die Fitness weniger relevant wäre als die Größe bei der Auswahl der Individuen (vgl Luke/Panait 2006, 5).
Ein weiterer Befund von Luke und Panait (2006) ist, dass die Kombination von Bloat-Kontrollmethoden und einer Tiefenlimitation zu besseren Ergebnissen führt als die Methode allein.

In diesem Projekt wurde bisher nur die maximale Tiefe der initialen Individuen kontrolliert. Durch Mutation und Crossover kann es allerdings immer noch zu starkem Wachstum der Programme kommen.
Um dies zu verhindern, wurden in der Vergangenheit bereits viele Methoden getestet. Einige dieser Methoden werden im Folgenden vorgestellt und diskutiert. Im Anschluss

werden diese Methoden in der Anwendung eingebunden und auf ihre Effektivität geprüft.

4.6 Lösungsansätze für Bloat

4.6.1 Tarpeian Methode

Die „Tarpeian Methode“ setzt die Fitness einiger überdurchschnittlich großer Individuen auf einen sehr schlechten Wert. Dadurch haben diese nur eine geringe Chance, für die genetischen Operationen ausgewählt zu werden. Das Ziel der Methode ist es, die Selektionswahrscheinlichkeit überdurchschnittlich großer Bäume effektiv zu mindern. Dies steigert allerdings ebenfalls die Wahrscheinlichkeit derjenigen Individuen, welche eine schlechtere Fitness haben, Einfluss auf die nächste Generation zu nehmen. Es gilt zu beachten, dass einzelne Selektionswahrscheinlichkeiten nicht verändert werden können, ohne die anderen auch zu beeinflussen (vgl. Poli 2003, 4 f.). Um diese Methode in jeden Algorithmus für genetische Programmierung einbinden zu können, stellt Poli (2003) folgenden Wrapper vor, welcher in den Evaluationsprozess eingesetzt werden kann.

```
// Tarpeian wrapper
IF size(program) > average_pop_size AND random_int MOD n = 0
THEN return( very_low_fitness ); ELSE return( fitness(program) );
```

Ein wesentlicher Vorteil hierbei liegt nicht nur in der Eindämmung von Bloat, sondern ebenfalls in der Ersparnis einiger Evaluationen derjenigen überdurchschnittlich großen Individuen, welche von der Tarpeian Methode ausgewählt wurden (vgl. Luke/Panait 2006, 7).

4.6.2 Linear Parametric Parsimony Pressure

"Linear Parametric Parsimony Pressure" gehört zu der Familie der "Parsimony Pressure" Methoden. Parsemony Pressure heißt wörtlich übersetzt Sparsamkeitsdruck und beschreibt den Einbezug der Größe von Individuen in den Selektionsprozess. Als größte Herausforderung bei der Anwendung dieser Methode präsentiert sich das Einstellen der Parameter, die den Einfluss der Größe auf die Selektion bestimmen. Hierbei erweist sich die Nicht-Linearität der meisten Fitnessfunktionen als großes Problem. Da kleine Fitnessunterschiede bei nahezu optimalen Lösungen einen höheren Wert für den Fortschritt bedeuten können als die großen Schritte zu Beginn der Suche, gestaltet sich die Implementierung dieser Methoden über ein Verhältnis zur rohen Fitness als sehr schwierig (vgl. Luke/Panait 2006, 7 f.).
Ein weitverbreiteter Ansatz, um mit dieser Problematik umzugehen, ist die Verwendung des Größenparameters als linearen Faktor in der Fitnessevaluation. Um dies umzusetzen, wird eine Formel wie folgt aufgestellt:

$$g = xf + ys$$

Dabei stellen f die Rohfitness, s die Größe des Individuums und x und y Einstellparameter dar. Eine weitere Variante dieser Formel sieht eine Mindestgröße vor, ab welcher s erst in die Fitness einbezogen wird. Wenn $s \leq z$, wobei z die Mindestgröße repräsentiert, wird die Fitness mit $g = xf$ berechnet. Sollte z allerdings überschritten werden, so würde die Fitness durch $g = xf + y(s - z)$ berechnet werden (vgl. Luke/Panait 2006, 8).

4.6.3 Tiefenbeschränkung

Die Tiefenbeschränkung kann auf verschiedene Weisen implementiert werden. Da der Fokus dieser Arbeit nicht auf der Tiefenbeschränkung liegen soll, wird eine einfache

Implementierung gewählt: Zunächst ist es notwendig, eine Helfermethode zur Erfassung der Tiefe eines Funktionsbaums zu definieren. Nun kann jedes Individuum, welches Teil einer neuen Generation werden soll, mit dieser geprüft werden. Sollte die maximale Tiefe überschritten werden, so gibt es zwei Möglichkeiten. Die erste besteht darin, dieses Individuum zu verwerfen und stattdessen einen der Elternbäume in die nächste Generation mitzunehmen. Die andere Möglichkeit ist, die genetische Operation zu wiederholen, bis ein passendes Individuum entstanden ist oder bis eine gewisse Anzahl an Versuchen gescheitert ist und das Individuum in Folge dessen wie im ersten Ansatz durch ein Elternteil ersetzt wird. In dieser Arbeit wird für die Tests die erste Methode angewendet.

4.7 Experimente

Im folgenden Kapitel werden alle Experimente sowie deren Ergebnisse vorgestellt, analysiert und diskutiert. Erkenntnisse, die aus diesen Versuchen hervorgehen, werden zur Weiterentwicklung des Algorithmus genutzt und dienen außerdem zur Bewertung der praktischen Lösung dieser Arbeit.
Für jeden Testaufbau werden zunächst die Rahmenbedingungen festgehalten. Es wird jede Komponente des Algorithmus, die nicht Gegenstand der Untersuchung dieser Tests ist, angeführt, um die Ergebnisse in einen Kontext einordnen zu können. Die folgenden Kontrollparameter werden vor jedem Test angegeben, allerdings werden die im jeweiligen Test ungenutzten Parameter ausgespart.

Parameter	**Beschreibung**
Location Number	Anzahl der Lagerplätze in diesem Testlauf
Article Number	Anzahl der Artikel in diesem Testlauf

Max-Creation-Depth	Maximale Tiefe bei der Erzeugung neuer Individuen
Bad-Fitness	Repräsentiert eine sehr schlechte Fitness
Tarp-p	Boolean für Tarpeian
Tarp-Perc	Prozent der überdurchschnittlich großen Individuen, die eine Bad-fitness erhalten sollen
Dir-Cand	Individuen, die direkt ohne genetische Operation in die nächste Generation übernommen werden
Max-Population	Maximale Populationsgröße
Depth-Limit	Maximale Tiefe der Funktionsbäume (0 = Kein Limit)
Tournament-Size	Tournament Größe für die Selektion
Lppp-p	Boolean für Linear Parametric Parsimony Pressure
Lppp-x	Gewichtungsparameter für die Fitness
Lppp-y	Gewichtungsparameter für die Größe

4.7.1 Methodik

Die folgenden Versuche werden jeweils von drei verschiedene Metriken erfasst. Die erste besteht in der besten Fitness jeder Generation. Hierzu wird für jede Generation die beste Fitness aller Individuen festgehalten und durch einen Graphen visualisiert. Die zweite Metrik ist die durchschnittliche Größe. Die Größe eines Individuums wird durch die Menge an Knoten innerhalb des Individuums dargestellt. Ein Knoten ist hierbei allerdings nicht je eine Funktion oder eine Terminale, sondern jede Teilzerlegung dieser. Somit ist die Größe der Funktion $(-A\ B) = 4$, da sowohl die

gesamte Funktion einen Knoten darstellt als auch jede Teilkomponente dieser. Pro Generation wird der Durchschnitt der gesamten Population gebildet. Die letzte Metrik stellt die durchschnittliche Fitness aller Individuen einer Population dar.
Die Entwicklung der Versuchsabläufe wird durch je ein Kurvendiagram für jede dieser Metriken visualisiert. Die x-Achse bildet die Generationen und die y-Achse die Metriken in den Diagrammen ab. Dies dient der Veranschaulichung der Ergebnisse. Durch ein Kurvendiagramm lässt sich eine generelle Tendenz der Entwicklung feststellen. Des Weiteren lassen sich die einzelnen Metriken in einen Zusammenhang bringen.

Zu beachten ist, dass die Fitness in diesem Anwendungsfall eine Kostenfunktion darstellt. Klassischerweise wird eine bessere Fitness durch einen höheren Wert dargestellt. Bei einer Kostenfunktion ist dies umgekehrt. Je niedriger der Wert, desto besser das Individuum.

Für alle Tests gilt, sofern nichts anderes angegeben wird, der gleiche Aufbau des Algorithmus. Für die Selektion wird standardmäßig die Tournament Selection genutzt Die Genetischen Operationen werden zu 90 Prozent mittels Standard Crossover sowie zu 10 Prozent durch die Mutation (s. Kapitel 4.4) durgeführt.

4.7.2 Bloat

In den folgenden Versuchen werden die Auswirkungen von Bloat und den im Kapitel 4.5 erarbeiteten Kontrollmethoden auf die Größe der Individuen und die Leistung des Algorithmus untersucht.

In den oben genannten Tests zum Thema Bloat werden die in der nachfolgenden Tabelle aufgeführten Parameter verwendet:

Location Number	64
Article Number	55
Generationen	200
Max-Creation-Depth	4
Bad-Fitness	300
Direct-Candidates	10
Max-Population	100
Tournament Size	2

Der Algorithmus wird auf ein Lager mit 64 Lagerplätzen und 55 Artikeln angewandt. Dies dient der Ausführbarkeit, da besonders in denjenigen Test mit großem Wachstum der Individuen viel Leistung verloren geht. Die Problemgröße sollte keine weiteren Auswirkungen auf das zu untersuchende Phänomen haben.

4.7.2.1 Test 1

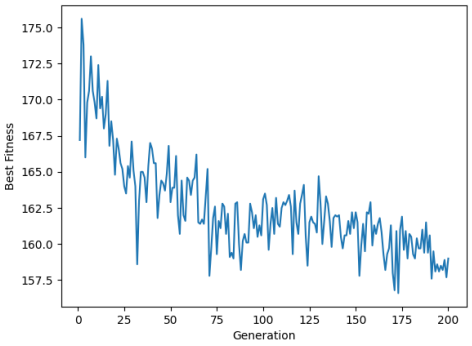

Figure 1: Best Fitness Test 1

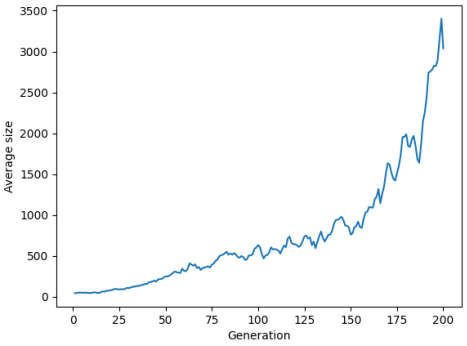

Figure 2: Average Size Test 1

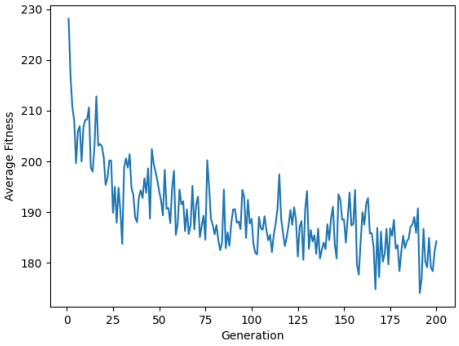

Figure 3: Average Fitness Test 1

Depth-Limit	0
Tarp-p	False
Lppp-p	False

Dieser Test dient als Kontrolle und Bestandsaufnahme zur Bewertung der Bloat-Kontrollmethoden. Es wird ein starkes Wachstum der durchschnittlichen Größe aufgrund der mangelhaften Größenkontrolle erwartet.

Die beste Fitness jeder Generation im Verlauf von 200 Generationen (s. Figure 1) weist einen Abfall der Fitness auf. Dieser findet in einem schwankenden Verlauf statt. Besonders im ersten Drittel des Versuches ist ein starker Abfall der Kurve zu beobachten.

Die durchschnittliche Fitness (s. Figure 3) weist ein sehr ähnliches Verhalten wie die beste Fitness auf. Es ist im ersten Drittel der stärkste Abfall zu verzeichnen. Im weiteren Verlauf flacht dieser ab. Allerdings bleibt im gesamten Verlauf eine abfallende Tendenz bestehen.

Die durchschnittliche Größe (s. Figure 2) folgt nahezu einem exponentiellen Wachstum. Die Kurve beginnt flach und steigt nur sehr gering an. Im weiteren Verlauf

wird der Anstieg immer größer. In den letzten Generationen gibt es große Sprünge, wenn auch mit einigen kleinen Stufen, in der durchschnittlichen Größe zu sehen.

Die für die folgenden Versuche relevanteste Metrik stellt die durchschnittliche Größe dar. Es lässt sich ein sehr starkes Wachstum erkennen. Dies resultiert nicht nur in übergroßen Individuen, sondern auch in einem starken Leistungsverlust. Mit jeder weiteren Generation verliert der Algorithmus an Effizienz. Die kommenden Kontrollmethoden sollen diesem Wachstum Einhalt gebieten und ermöglichen, den Algorithmus in einer realistischen Zeit zu noch besseren Ergebnissen zu führen.

4.7.2.2 Test 2

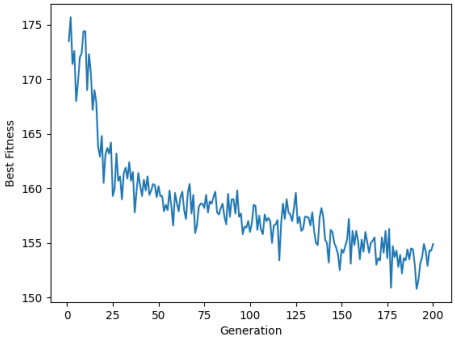

Figure 4: Best Fitness Test 2

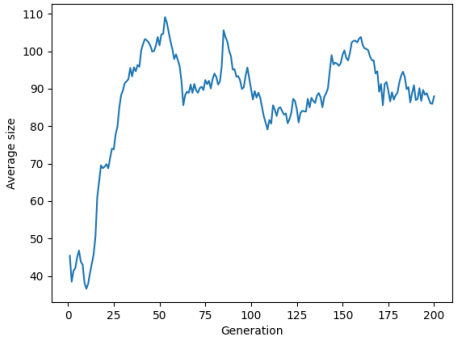

Figure 5: Average Size Test 2

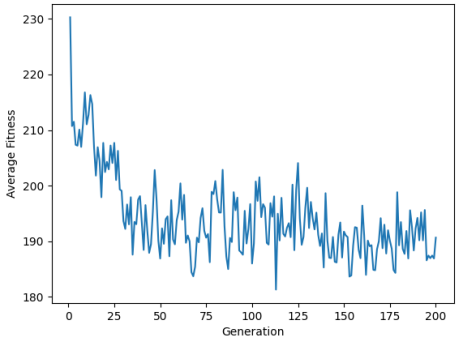

Figure 6: Average Fitness Test 2

Depth-Limit	20
Tarp-p	False
Lppp-p	False

In diesem Test wird die in Kapitel 4.6.3 vorgestellte Tiefenlimitierung, die jedes durch eine genetische Operation generiertes Individuum auf seine Größe überprüft und sicherstellt, dass kein zu großes Individuum in die neue Generation gelangt, verwendet.

Ein eingedämmtes Wachstum der durchschnittlichen Größe ohne größere Auswirkungen auf die Fitness wird erwartet.
Die beste Fitness (s. Figure 4) unterliegt abermals starken Schwankungen zwischen einzelnen Generationen. Im Gesamtbild ist eine abfallende Kurve zu erkennen. Zu Beginn fällt die Fitness noch deutlich stärker ab. Im späteren Verlauf der Generationen flacht der Funktionsgraph ab.
Die durchschnittliche Größe (s. Figure 5) der Individuen weist ein starkes Wachstum auf. Dieses Wachstum setzt sich stufenweise bis zu einer durchschnittlichen Größe von 110 Knoten fort. Ab diesem Punkt pendelt sich die durchschnittliche Größe zwischen 80 und 100 Knoten ein.
Ähnlich wie bei der durchschnittlichen Fitness lassen sich bei der besten Fitness (s. Figure 6) jeder Generation sehr starke Schwankungen erkennen. Eine Tendenz zum Fallen der Fitness im Verlauf der gesamten 200 Generationen zeigt sich analog zur durchschnittlichen Fitness.

Im Vergleich zu dem vorherigen Test ohne die Tiefenbeschränkung zeichnet sich in diesem Versuch ein wie eingangs erwartet deutlich moderateres Wachstum der Individuen ab. Es scheint, dass durch die gewählte maximale Tiefe eine Größe von 110 Knoten in einem Individuum die Obergrenze darstellt. Dies hat unabhängig von der Fitness einen großen Einfluss auf die Leistung und Ausführbarkeit des Algorithmus. Das Ziel ist, in einer realistischen Zeit eine möglichst gute Lösung zu finden. Daher ist dies, ohne die Fitness zu betrachten, bereits eine Verbesserung des Algorithmus.

4.7.2.3 Test 3

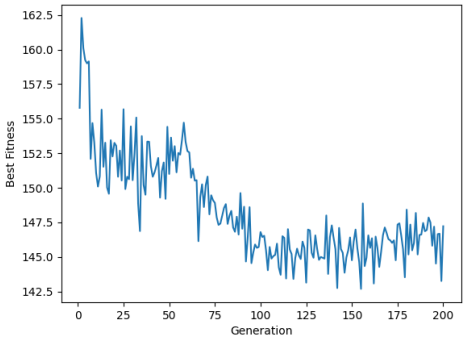

Figure 7: Best Fitness Test 3

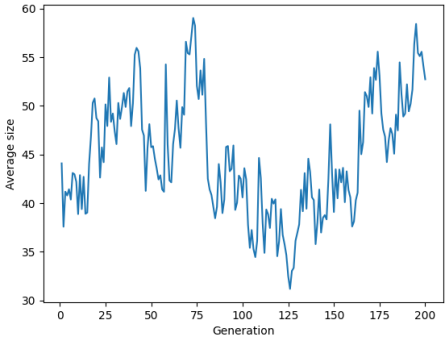

Figure 8: Average Size Test 3

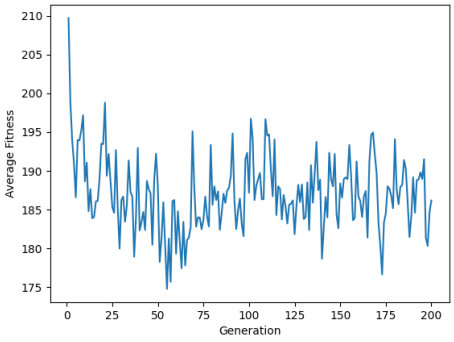

Figure 9: Average Fitness Test 3

Depth-Limit	0
Tarp-p	False
Lppp-p	True
Lppp-x	0.9
Lppp-y	0.1

In diesem Test wird die in Kapitel 4.6.2 vorgestellte Linear Parametric Parsimony Pressure Methode verwendet. Der x Wert stellt den prozentualen Einfluss der rohen Fitness und der y Wert den der Größe eines Individuums auf die Fitness, die für die Selektion verwendet wird, dar. Daher kann diese Fitness nicht direkt mit der der anderen Tests verglichen werden. Es werden ein moderateres Wachstum sowie eine veränderte Fitness durch den Einfluss des Größenparameters erwartet.

Bei der besten Fitness (s. Figure 7) zeichnet sich ein leicht flacherer Funktionsgraph ab. Die Verringerung der Fitness ist zwar noch immer zu erkennen, jedoch kann eine Stagnation bis hin zu einem Anstieg dieser nach der Hälfte der Generationen erkannt werden. Der Graph unterliegt wie bereits alle dieser Metrik starken Schwankungen.

Die durchschnittliche Größe (s. Figure 8) schwankt über den gesamten Verlauf des Tests zwischen 30 und 60 Knoten.
Die durchschnittliche Fitness (s. Figure 9) startet mit einer deutlichen Senkung der Fitness. Im restlichen Verlauf bleibt diese allerdings, abgesehen von den typischen Schwankungen zwischen den Generationen, auf einem konstanten Niveau.

In diesem Test kann man bereits erkennen, dass die Linear Parametric Parsimony Pressure die durchschnittliche Größe der Population in einen konstanten Rahmen hält. Die Schwankungen befinden sich in einem Bereich von 30 Knoten. Dieses Verhalten ähnelt dem des Tests zur Tiefenlimitierung, allerdings ist die durchschnittliche Größe bei jenem Test zunächst auf ein weitaus höheres Niveau gestiegen. Dies lässt auf eine ähnlich starke Kontrolle der Größe bei diesen beiden Methoden schließen. Beim Vergleich der Fitness ist zu beachten, dass sie in diesem Test nicht durch die rohe Manhattan Distanz definiert wurde, sondern durch die finale für die Selektion verwendete Fitness, bei der die Größe der Individuen einberechnet wurde.

4.7.2.4 Test 4

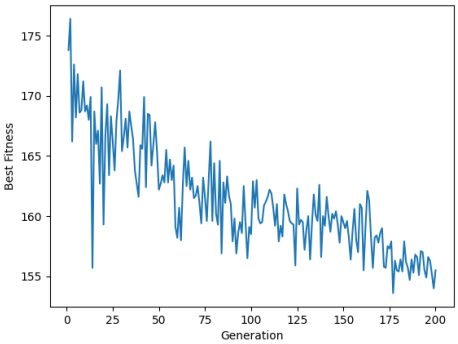

Figure 10: Best Fitness Test 4

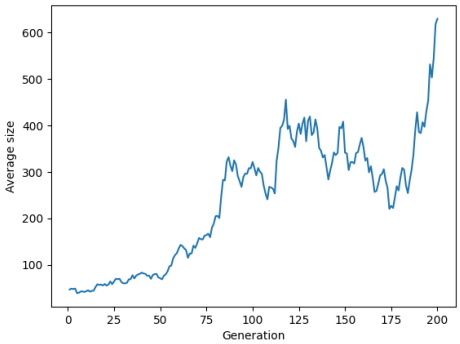

Figure 11: Average Size Test 4

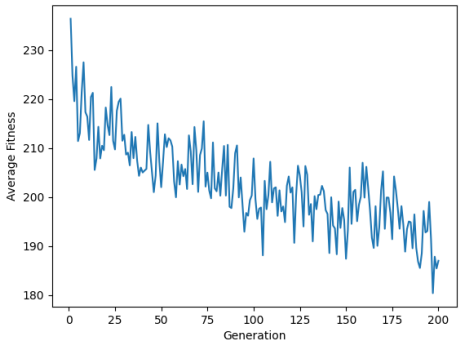

Figure 12: Average Fitness Test 4

Depth-Limit	0
Tarp-p	True
Tarp-Perc	20
Lppp-p	False

In diesem Test wird die in Kapitel 4.6.1 vorgestellte Tarpeian Methode verwendet. Die „Tarp-Perc“ stellt dabei die prozentuale Wahrscheinlichkeit, dass ein überdurchschnittlich großes Individuum mit einer „Bad-Fitness“ versehen wird, dar.
Es lässt sich bei der besten Fitness (s. Figure 10) eine deutliche Tendenz zum Abfallen der Fitness erkennen. Abgesehen von einem leichten Abflachen in der Mitte des Tests und den für diese Metrik typischen Schwankungen stellt der Graph einen konstanten Abstieg dar. Durch dieses Aussortieren überdurchschnittlich großer Individuen wird ein verzögertes und somit langsameres Wachstum als im ersten Test erwartet.
Die durchschnittliche Größe (s. Figure 11) wächst im gesamten Verlauf des Tests stufenweise an. Zwischenzeitlich findet eine Stabilisierung der Kurve statt, die zum Ende des Tests allerdings durch ein erneutes starkes Wachstum aufgehoben wird.
Die durchschnittliche Fitness (s. Figure 12) zeichnet in diesem Test genauso wie die beste Fitness eine absteigende Linie.

Insgesamt scheint eine Leistungsverbesserung des Algorithmus durch die Tarpeian Methode in Hinblick auf die Fitness zu bestehen. Eine Eindämmung des Wachstums ist im Vergleich zum Test ohne jegliche Bloat-Kontrollmethode ebenfalls zu erkennen, wenn auch nicht so stark wie bei den anderen Methoden. Das Wachstum wurde wie eingangs erwartet nur verlangsamt. Das mag allerdings bereits genug sein, um eine hinreichend gute Lösung finden zu können.

4.7.2.5 Test 5

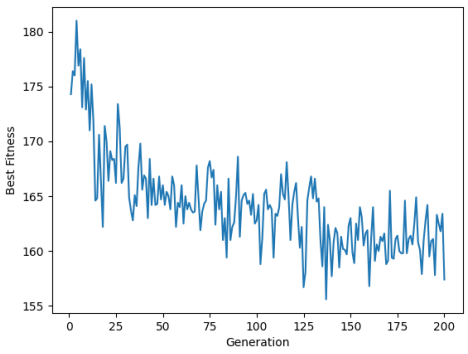

Figure 13: Best Fitness Test 5

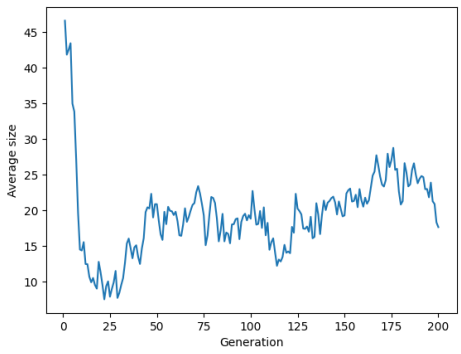

Figure 14: Average Size Test 5

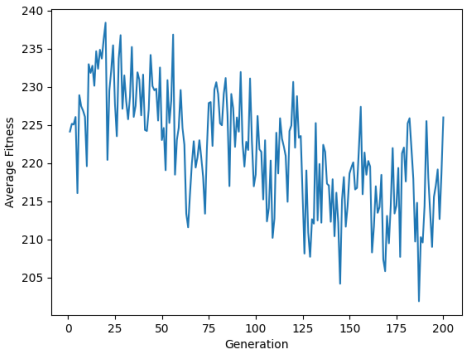

Figure 15: Average Fitness Test 5

Depth-Limit	20
Tarp-p	True
Tarp-Perc	20
Lppp-p	False

In diesem Test wird die Kombination aus einer Tiefenlimitierung mit der Tarpeian Methode untersucht. Von dieser Kombination wird erwartet, ein gedeckeltes und verzögertes Wachstum zu haben, wodurch die Eigenschaften der einzelnen Methoden widerspiegelt werden würden.

Die beste Fitness (s. Figure 13) sinkt innerhalb der ersten 25 Generationen noch deutlich. Im Verlauf flacht die Kurve weiter ab. Die starken Schwankungen sorgen weiterhin für einige Abstiege, allerdings ist als deutliche Tendenz zu verzeichnen, dass sich die beste Fitness auf einem gleichbleibenden Niveau stabilisiert.
Die durchschnittliche Größe (s. Figure 14) fällt innerhalb der ersten 25 Generationen rapide auf eine verschwindend geringe Anzahl an Knoten ab. Danach wächst diese allerdings wieder ein wenig an. Sie pendelt sich zwischen 15 und 30 Knoten ein. An der Skala der y-Achse lässt sich erkennen, dass sich die durchschnittliche Größe in diesem Test auf einem sehr niedrigen Niveau befindet.
Die durchschnittliche Fitness (s. Figure 15) bewegt sich in einem recht kleinen Raum zwischen 240 und 205 Knoten. Die Kurve unterliegt in diesem Raum großen Schwankungen. Innerhalb des gesamten Tests lässt sich nur ein leichter stufenweiser Abstieg der Fitness erkennen.

Durch den soeben vorgestellten Test wird deutlich, dass eine zu große Kontrolle der Größe der Individuen den Ergebnissen des Algorithmus schaden kann. Die durchschnittliche Größe ist am Anfang entgegen der Erwartung so drastisch gefallen, dass dies zu einer mangelhaften Diversität der Population und somit zu weniger Fortschritten im weiteren Verlauf führen kann. Nur durch Mutation kann diese Diversität zurückgewonnen werden. Das beschriebene Szenario birgt eine Gefahr vor allem in denjenigen Verfahren, die ausschließlich mit Crossover arbeiten.

4.7.2.6 Test 6

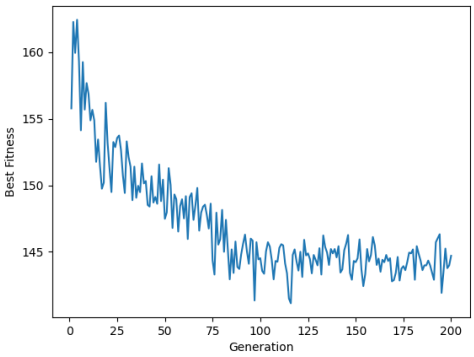

Figure 16: Best Fitness Test 6

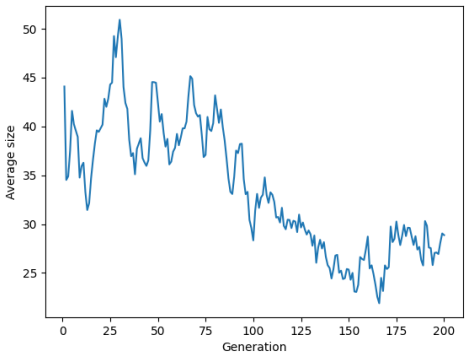

Figure 17: Average Size Test 6

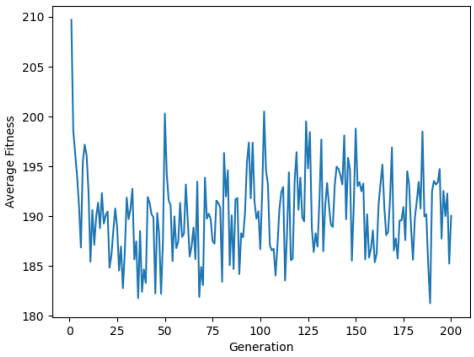

Figure 18: Average Fitness Test 6

Depth-Limit	20
Tarp-p	False
Lppp-p	True
Lppp-x	0.9
Lppp-y	0.1

In diesem Test wird die Kombination aus einer Tiefenlimitierung mit der Linear Parametric Parsimony Pressure Methode untersucht. Die Fitness wird in diesem Test wieder durch ein 90 zu 10 Verhältnis von Fitness und Größe der Individuen dargestellt und kann daher nicht unmittelbar mit den der anderen verglichen werden. Da der Test zur Linear Parametric Parsimony Pressure Methode allein bereits eine sehr Schwankende durchschnittliche Größe aufwies, wird in diesem Test eine schwankende Wachstumskurve mit weniger extremen Ausschlägen nach oben erwartet.

Die durchschnittliche Größe (s. Figure 17) bewegt sich auf einem niedrigen Niveau zwischen 0 und 50 Knoten. Zu Beginn ist ein Größenwachstum zu verzeichnen. So schwankt die durchschnittliche Größe in der ersten Hälfte noch zwischen 30 und 45 Knoten mit einem Ausreißer, der bis zu 50 Knoten geht. In der zweiten Hälfte fällt sie

allerdings stark und bleibt auf einem Niveau zwischen 0 und 30 Knoten bis zum Ende des Tests bestehen.

4.7.2.7 Analyse

Der im Verhältnis zu den Tests, bei denen mit nur einer Kontrollmethode und ohne die maximale Tiefe gearbeitet wird, starke Abfall der durchschnittlichen Größe in Test 5 und 6 lässt sich auf den ersten Blick nicht mit den erwarteten Ergebnissen in Einklang bringen. Da die Kontrollmethoden den einzigen direkten Druck auf die Größe der Individuen ausüben, wurde ein ähnliches Verhalten wie im jeweiligen Test ohne Tiefenbeschränkung erwartet. In Ergänzung dazu wurde angenommen, dass ein Unterschied darin besteht, dass die Größe gedeckelt ist. Zu erklären ist die Kluft zwischen Annahme und Ergebnis einerseits durch die Natur eines durchschnittlichen Wertes. Es wäre denkbar, dass in den vorherigen Tests einige sehr große Individuen den Durchschnitt stark beeinflusst haben. Andererseits steht die Größe der nächsten Generation in sehr starkem Zusammenhang mit der Elterngeneration. Um Korrelationen zwischen dem Wachstum der durchschnittlichen Größe der Generation und der Größe der Elterngeneration verstehen zu können, muss untersucht werden, wie ein Individuum im Verlauf der Generationen wachsen kann. Die erste Möglichkeit für das Wachstum eines Individuums ist die Mutation. Durch die Mutation wächst ein Individuum genau dann, wenn der generierte Unterbaum größer ist als der ausgetauschte Unterbaum. Die zweite Möglichkeit besteht im Crossover. Hierbei wird je ein Unterbaum zweier Individuen miteinander vertauscht. In diesem Prozess kann genau ein Individuum an Größe gewinnen, während das andere jene verliert. Somit verändert sich die durchschnittliche Größe nicht. Da nur 10 Prozent der Individuen durch Mutation gebildet werden, ist ein Zuwachs der durchschnittlichen Größe sehr langsam. Ohne jegliche Größenkontrolle gibt es demnach auch genauso viele Möglichkeiten, an Größe zu verlieren wie an Größe zu gewinnen. Durch das

Hinzufügen wachstumshemmender Methoden wird dieses Gleichgewicht unweigerlich beeinflusst. Auch wenn eine Deckelung der Größe für das Wachstum bis zu dieser maximalen Größe irrelevant erscheint, so wirkt sich jede verhinderte Mutation, die zu einem Größenzuwachs der Population beitragen würde, auf das Wachstumspotential der gesamten Population aus.

4.7.3 Selektion

In diesem Test soll der Unterschied zwischen den drei zuvor vorgestellten Selektionsmethoden (s. Kapitel 4.3) untersucht werden.
Die für die folgenden Versuche gleichbleibenden Parameter wurden wie folgt festgelegt:

Location Number	64
Article Number	55
Generationen	200
Max-Creation-Depth	4
Bad-Fitness	300
Direct-Candidates	10
Max-Population	100
Depth-Limit	20
Tarp-p	False
Lppp-p	False

4.7.3.1 Tournament Selection

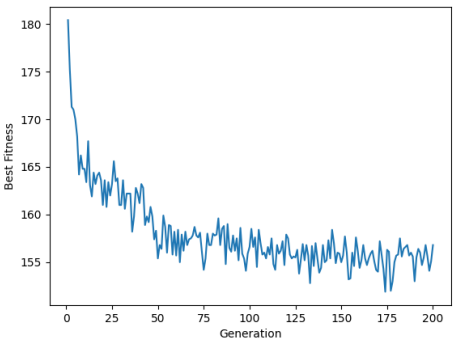

Figure 19: Best Fitness Tournament Selection

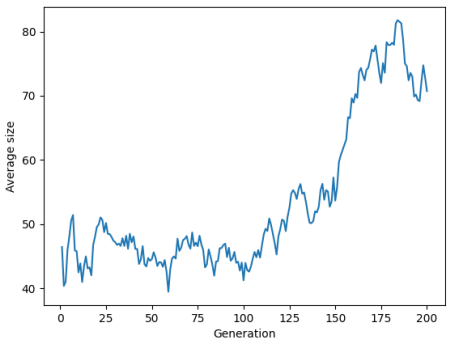

Figure 20: Average Size Tournament Selection

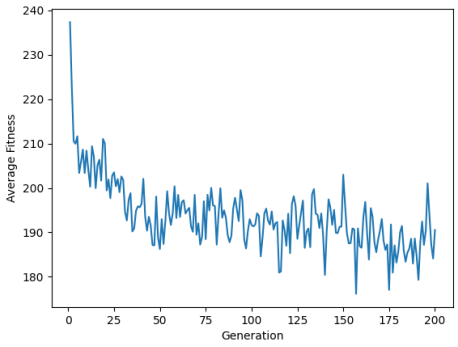

Figure 21: Average Fitness Tournament Selection

Tournament Size	2

In diesem Test wird die im Kapitel 4.3.1 vorgestellte Tournament Selektion mit einer Tournament Size von 2 untersucht.
Die beste Fitness fällt zu Beginn stark ab, woraufhin die Kurve langsam abflacht. Über den gesamten Verlauf sind Schwankungen zwischen einzelnen Generationen zu erkennen.
Bei der durchschnittlichen Größe der Individuen zeichnen sich Schwankungen während des gesamten Verlaufs ab. In der ersten Hälfte ist kein signifikantes Wachstum zu vermerken. Die Kurve schwankt zwischen 40 und 50 Knoten. In der zweiten Hälfte beginnt ein stärkeres Wachstum bis hin zu 80 Knoten.
Die durchschnittliche Fitness beschreibt eine ähnliche generelle Kurve wie die beste Fitness. Allerdings unterliegt diese weitaus stärkeren Schwankungen.

In diesem Versuch ist durch die Fitness ein Lernprozess des Algorithmus zu erkennen. Im Verlauf der 200 Generationen verbessert sich das beste Individuum um 30. Hierdurch wird bereits die Fähigkeit der Tournament Selection ersichtlich, die Optimierung einer Population vorzunehmen.

4.7.3.2 Fitness-Proportionate Selection

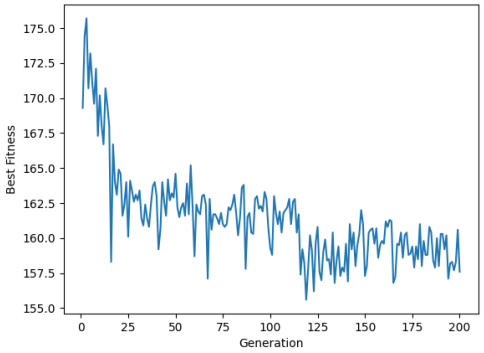

Figure 22: Best Fitness Fitness-Proportionate Selection

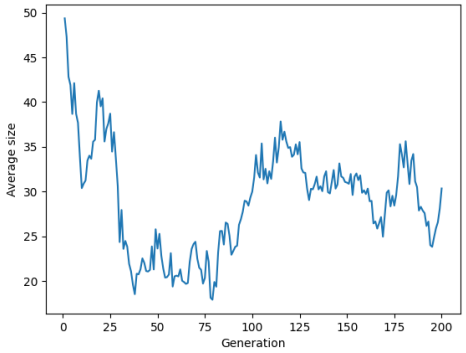

Figure 23: Average Size Fitness-Proportionate Selection

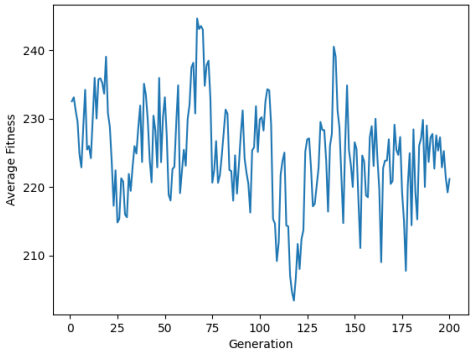

Figure 24: Average Fitness Fitness-Proportionate Selection

In diesem Test wird die in Kapitel 4.3.2 vorgestellte Fitness-Proportionate Selection verwendet.

Die beste Fitness verfolgt eine ähnliche Kurve wie die des vorherigen Tests. Zu Beginn ist die Kurve stark abfallend. Im weiteren Verlauf flacht sie ein wenig ab. In diesem Fall ist die Entwicklung allerdings in der Mitte durch eine weitere stufenweise Verringerung der Fitness erweitert.

Die durchschnittliche Größe bewegt sich auf niedrigerem Niveau. Zunächst fällt diese auf einen Tiefpunkt von 20 Knoten und bewegt sich von da an zwischen 20 und 25 Knoten, bis sie schließlich wieder auf 35 Knoten anwächst.

Die durchschnittliche Fitness ist im Gegensatz zu dem vorherigen Test sehr schwankend und bewegt sich auf einem sehr hohen Niveau zwischen 210 und 240 Punkten. Abgesehen von einigen Ausreißern lässt sich kaum eine Tendenz zu einer Verringerung der Fitness erkennen.

Die beste Fitness dieses Tests lässt immer noch darauf schließen, dass die Selektionsmethode in der Lage ist die Population zu einer guten Lösung zu kultivieren.

In Hinblick auf die durchschnittliche Größe kann durch den starken Abfall zu Beginn des Testes eine mangelhafte Diversität der Population erahnt werden.

4.7.3.3 Greedy Over-Selection

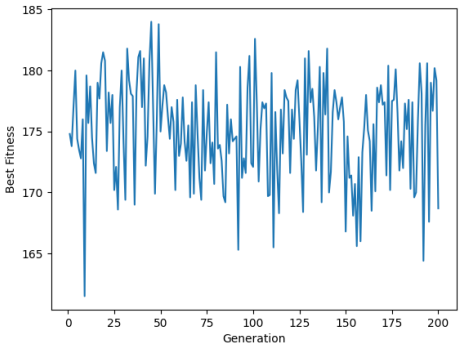

Figure 25: Best Fitness Greedy Over-Selection

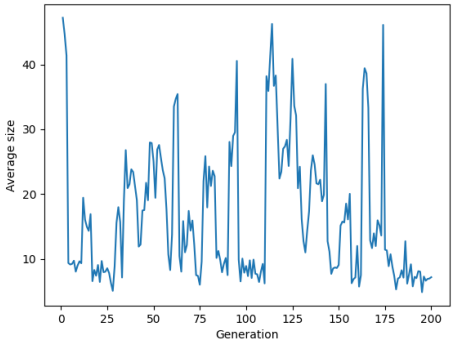

Figure 26: Average Size Greedy Over-Selection

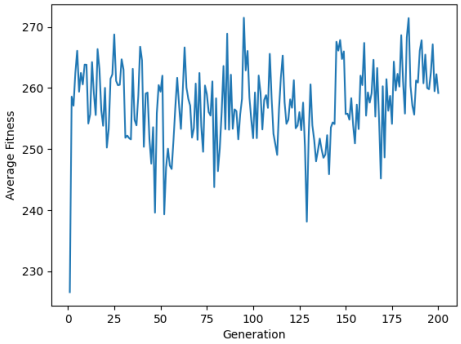

Figure 27: Average Fitness Greedy Over-Selection

In diesem Test wird die in Kapitel 4.3.3 vorgestellte Greedy Over-Selection verwendet. Die beste Fitness weist keine Tendenz zu einer Verringerung auf. Abgesehen von einigen Ausreißern schwankt die Kurve über die gesamte Dauer des Tests zwischen 170 und 180 Punkten.
Die durchschnittliche Größe fällt zu Beginn sehr stark bis unter 10 Knoten. Im weiteren Verlauf schwankt diese zwischen 40 und 10 Knoten.
Die durchschnittliche Fitness steigt zu Beginn stark an und schwank ab diesem Punkt zwischen 240 und 270 Punkten.

Die schlechte Leistung von der Greedy Over-Selection ist durch die Größe der Individuen zu erklären. Zu Beginn fällt die durchschnittliche Größe sehr stark, dieser Trend setzt sich fort.
Die Schwankungen zu fast 50 Knoten liegen in der Einführung neuer Unterbäume durch die Mutation begründet, welche durch den RHH-Ansatz Unterbäume erzeugt

werden. Da die durchschnittliche Größe zumeist zwischen 40 und 50 Knoten startet und die Individuen durch die Erzeugung mittels des RHH-Ansatzes entstanden sind, bereitet dies Grund zur Annahme, dass die maximale Größe durch die Max-Creation-Depth von 4 in diesem Bereich liegt. Die Schwankungen bis zu 50 Knoten und der darauffolgende Abfall der Größe legen die Vermutung nahe, dass die Greedy Over-Selection eine Tendenz hat, kleinere Individuen zu bevorzugen. Dies könnte der Fall sein, wenn kleiner Individuen zu Beginn eine bessere Fitness hervorbringen. Der Selektionsdruck dieser Methode war zu stark, um schlechteren Individuen mit einer größeren Genvielfalt die Chance zu geben, sich zu entwickeln.

4.7.3.4 Vergleich

Die durchgeführten Versuche stellen heraus, dass die Tournament Selection in der Konfiguration, die als Grundlage für die Tests dient, die besten Ergebnisse erbracht hat. Die Greedy Over Selection unterlag einem zu starken Selektionsdruck und hat somit das Überwinden eines lokalen Optimums unmöglich gemacht. Die Fitness-Propotionate Selection hat zwar bessere Ergebnisse geliefert, ist jedoch im Vergleich zur Tournament Selection, vor allem im Hinblick auf die durchschnittliche Fitness, die schlechtere Methode. Ein Vorteil der Tournament Selection liegt darin, dass die Methode sehr flexibel ist, da der Selektionsdruck direkt durch das Anpassen der Tournamentgröße anpassbar ist. Dadurch wird ermöglicht, diese Selektionsmethode für jeden Anwendungsfall schnell und intuitiv einzustellen. Aus diesen Gründen wird im finalen Test dieser Arbeit die Tournament Selection verwendet.

4.7.4 Finale Tests

In diesem letzten Abschnitt des Kapitels „Experimente“ werden die Erkenntnisse der vorherigen Versuche genutzt, um einen finalen Test mit einem verbesserten Algorithmus durchzuführen. Es werden zwei Versuche mit je 500 Generationen durchgeführt. Um die Ausführbarkeit in einem realistischen Rahmen zu halten, wird eine Tiefenlimitierung von 20 verwendet.

Location Number	64
Article Number	55
Generationen	500
Max-Creation-Depth	4
Bad-Fitness	300
Direct-Candidates	10
Max-Population	200

4.7.4.1 Mit Tarpeian

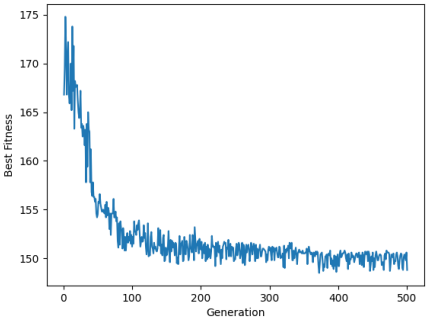

Figure 28: Best Fitness mit Tarpeian

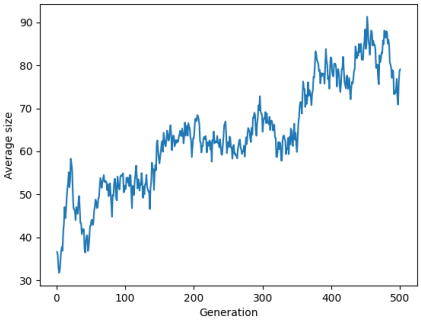

Figure 29: Average Size mit Tarpeian

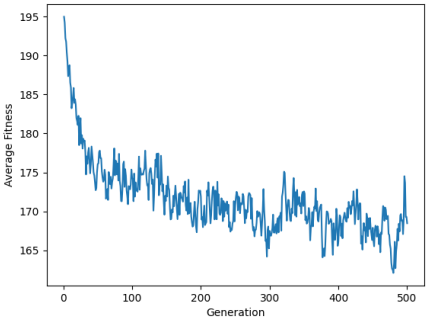

Figure 30: Average Fitness mit Tarpeian

Depth-Limit	20
Tarp-Perc	10
Tournament Size	2

In diesem Test werden die Tournament Selection, die Tarpeian Methode und eine Tiefenlimitierung verwendet. Von diesem Test wird eine stetige Optimierung der Lösung sowie ein moderates Wachstum der durchschnittlichen Größe erwartet. Es besteht die Hoffnung, dass durch eine größere Population und eine geringer Größenkontrolle, als in dem Test zur Kombination von Tiefenlimitierung und Tarpeian, die Diversität der Population gesteigert und eine bessere Lösung gefunden werden kann.

Die beste Fitness dieses Tests fällt innerhalb der ersten 100 Generationen stetig ab. Die typischen Schwankungen sind auch in diesem Test vorhanden. Nach dieser starken Verbesserung zu Beginn flacht die Kurve deutlich ab. Im restlichen Verlauf der Kurve ist eine leichte Tendenz zum Fallen der Fitness zu erkennen, allerdings auf einem deutlich geringeren Niveau als zuvor.

Die durchschnittliche Größe wächst stufenweise an. Zu Beginn gibt es ein starkes Wachstum innerhalb der ersten 25 Generationen gefolgt von einem Abfall in den

nächsten 25 Generationen. Im weiteren Verlauf sind die Stufen klar zu erkennen. Zwischen den Generationen gibt es einige Schwankungen, allerdings hält sich die durchschnittliche Größe auf einem konstanten Niveau, bis die nächste Wachstumsstufe erreicht wird. Zum Ende wird eine maximale Größe von ungefähr 90 Knoten erlangt, wonach ein Absinken zu erkennen ist.
Die durchschnittliche Fitness weist den stärksten Abstieg innerhalb der ersten 25 Generation auf. Danach flacht die Kurve ab. Es sind außerdem deutliche Schwankungen erkennbar.

Dieser Test weist vielversprechende Ergebnisse auf. Durch die zunächst sehr starke Verbesserung der besten Fitness und der im späteren Verlauf abflachenden Kurve kann ist das Konvergieren des Algorithmus sichtbar. Auch das durchschnittliche Wachstum der Individuen hält sich in Grenzen, sodass der Algorithmus keine unrealistischen Leistungsanforderungen stellt. Die starken Schwankungen in den Kurven der besten und durchschnittlichen Fitness lassen Grund zur Annahme, dass dem Algorithmus durch die Selektion und die genetischen Operationen die Möglichkeit inne wohnt, lokale Optima zu überwinden. Um die Verbesserung der Individuen besser darstellen zu können, werden mehrere Validierungsdatensätze der Bestellungen, welche zur Bildung der Fitness genutzt werden, angewandt und durch ein Balkendiagram visualisiert. Hierbei ist zu beachten, dass alle Bestellungsdatensätze durch einen wahrscheinlichkeitsbasierten Algorithmus erzeugt wurden. Sie weisen daher eine ähnliche Frequenz der meistbestellten Artikel auf.

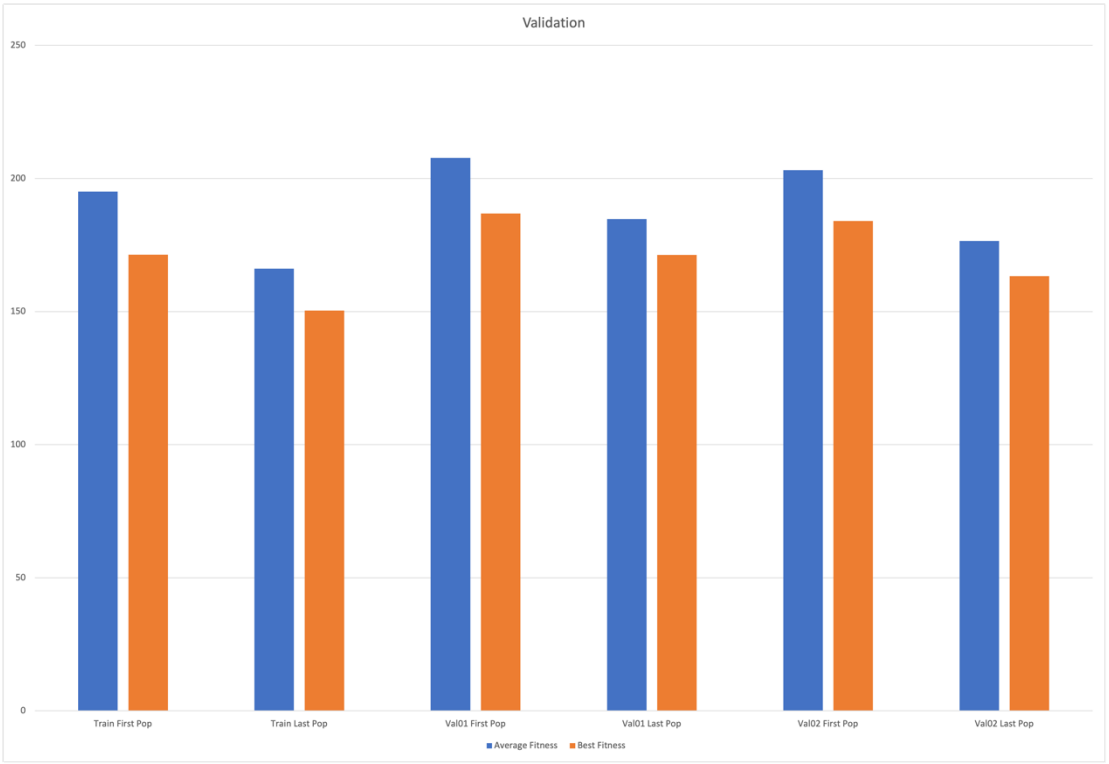

Figure 31: Validation Balkendiagramm mit Tarpeian

Die Validierung zeigt einige Evaluationen der ersten und der letzten Population des Testlaufes mit verschiedenen Datensätzen. Die y-Achse beschreibt den Fitness-Score. Auf der x-Achse sind pro Population zwei Balken für alle drei Datensätze. Jeder Balken stellt eine Metrik dar. So lässt sich für alle Datensätze erkennen, dass die letzte Generation merklich bessere Resultate liefert. Dadurch ist gesichert, dass der Algorithmus vom Trainingsdatensatz abstrahiert hat, sodass auch bei anderen Bestellungen gute Lösungen gefunden werden können.

4.7.4.2 Ohne Tarpeian

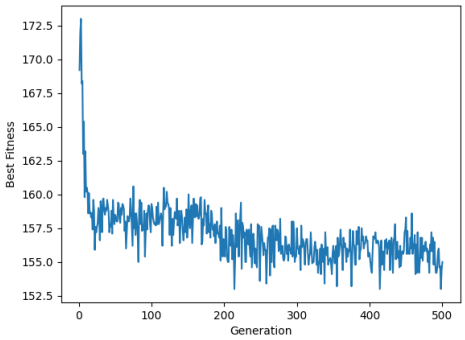

Figure 32: Best Fitness ohne Tarpeian

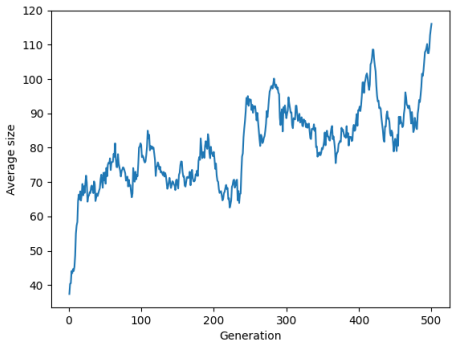

Figure 33: Average Size ohne Tarpeian

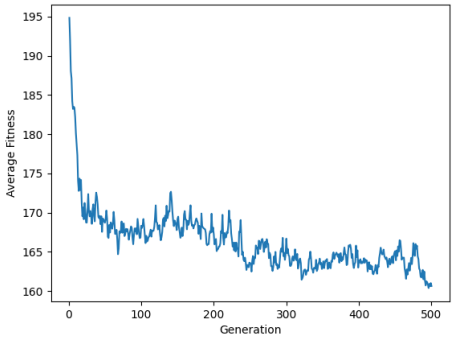

Figure 34: Average Fitness ohne Tarpeian

Depth-Limit	20
Tournament Size	2

In diesem Test werden die Tournament Selection und eine Tiefenlimitierung verwendet. Es wird eine nennenswerte Optimierung der Fitness mit einer soweit kontrollierten durchschnittlichen Größe, dass der Algorithmus in einer realistischen Zeit ausgeführt werden kann, erwartet.

Die beste Fitness fällt innerhalb der ersten Generationen sehr stark ab. Im weiteren Verlauf ist eine stufenweise Verringerung der Fitness zu sehen. Dies findet unter starke Schwankung statt.

Die durchschnittliche Größe wächst zu Beginn sehr stark an. Dieser Trend setzt sich stufenweise über den gesamten Verlauf fort.

Die durchschnittliche Fitness hat ein sehr ähnliches Verhalten wie die beste Fitness allerdings mit weniger Schwankungen.

Genauso wie der vorrangehende Test erzielt dieser Versuch gute Resultate. Die Fitness bewegt sich auf einem ähnlichen Niveau allerdings mit stärkeren Schwankungen. Die durchschnittliche Fitness weist bessere Ergebnisse auf, allerdings ist dies unter dem

Aspekt, dass im vorherigen Test die Tarpeian Methode einige schlechte Fitnesswerte für die überdurchschnittlich großen Individuen verteilt, zu vernachlässigen. Trotz dessen lässt sich ein starker Lerneffekt erkennen. Die Ergebnisse werden durch eine Validierung geprüft.

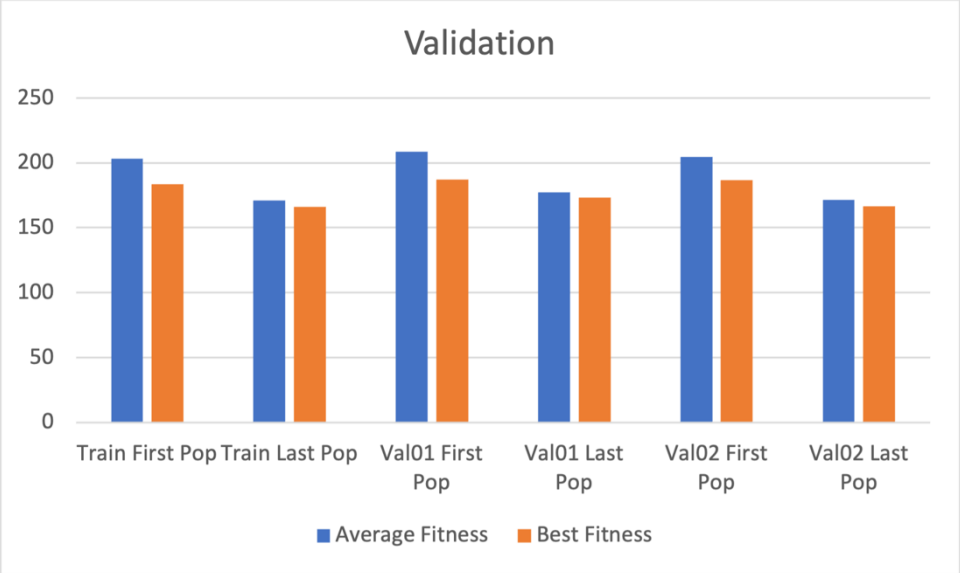

Figure 35: Validation Balkendiagramm ohne Tarpeian

Die Validation verhält sich in diesem Test genauso wie in dem vorherigen. Die letzte Population erzielt eine merkliche Verbesserung der ersten Generation gegenüber. Dies bestätigt die Erkenntnisse des ersten Tests.

5 Fazit

Abschließend sollen die gewonnenen Erkenntnisse nun in einem Fazit zusammengefasst werden: In der vorliegenden Arbeit wurde die Einsetzbarkeit der genetischen Programmierung zur Lösung des Storage Location Assignment Problems in Anwendung auf das Lager der APS Glass and Bar Supply GmbH thematisiert. Um das genannte Thema umfassend untersuchen zu können, wurden zunächst einige bereits auf dieses Problem angewandte Verfahren sowie die genetische Programmierung vorgestellt und in Hinblick auf ihr Potential zur Lösung des Problems diskutiert. Anschließend wurde eine Implementierung des Algorithmus umgesetzt. Diese wurde mit Hilfe verschiedener Versuche zu Lösungsansätzen grundlegender Probleme des Algorithmus verbessert. Ein finaler Test mit der überarbeiteten praktischen Implementierung soll die Forschungsfrage klären.

Die im Rahmen der Implementierung durchgeführten praktischen Versuche, die mit Hilfe der Daten der APS Glass und Bar Supply GmbH durchgeführt wurden, haben vielversprechende Ergebnisse geliefert. Diese untermauern die im Hauptteil aufgestellte Hypothese, dass die genetische Programmierung ein großes Potential zur Verbesserung der Lagersortierung birgt. In den Tests wurde mit Hilfe einer Fitnessfunktion eine Funktion zum Einsortieren von Artikeln beim Wareneingang kultiviert. Die letztendliche Fitness hat sich aus den Kommissionierungskosten einiger, basierend auf historischen Daten generierter, Bestellungen ergeben. Für den finalen Test dieser Arbeit wurde ein weiterer Validierungsdatensatz verwendet, um die Leistungsentwicklung aufzuzeigen.

Des Weiteren konnten wertvolle Erkenntnisse über verschiedene Komponenten des Algorithmus in Form der Testergebnisse zu den grundlegenden Problemen der genetischen Programmierung gewonnen werden. So wurden einige Bloat-Kontrollmethode in einem praxisnahem Versuchsaufbau miteinander verglichen. Daraus ergab sich der Schluss, dass die zu starke Kontrolle des Wachstums der

Individuen einer Population, die Möglichkeit nimmt, bessere Lösungen zu finden. Das Maß, in dem jenes Wachstum kontrolliert wird, ist entscheidend für die Entwicklung einer leistungsfähigen Population. Da in diesem Versuch durch wenig Mutation und keine Neueinführung ein geringes Wachstumspotential besteht, bedarf es keiner starken Kontrolle. Die Limitierung der maximalen Tiefe ist in diesem konkreten Fall ausreichend, um effizient gute Lösungen in einer realistischen Zeit finden zu können.
Des Weiteren konnte herausgestellt werden, dass die Tournament Selection die flexibelste und effizienteste Methode zur Kultivierung einer leistungsfähigen Population für diesen Anwendungsfall darstellt. Durch die Tournamentgröße lässt sich jene sehr leicht und intuitiv einstellen.
Abschließende Tests mit Algorithmen, die durch die gesammelten Erkenntnisse verbessert wurden, ergeben, dass die genetische Programmierung ein durchaus großes Potential zur Optimierung eines dynamischen Lagers durch eine Matching Funktion innehält. Dies prüft die in der Einleitung aufgestellte These.
In zukünftigen Arbeiten könnten die Auswirkungen der in dieser Arbeit erarbeiteten Lösung auf ein reales Lager in einer Langzeitstudie untersucht und mit der vorherigen Leistungsfähigkeit dessen verglichen werden. Dies könnte das Potential einer Funktion zur Kontrolle des Wareneinganges mit realen Daten bewerten. Außerdem wäre ein ähnlicher Versuch durch eine Simulation eines Lagers basierend auf historischen Daten des Unternehmens ebenso denkbar. Ein weiterer Ansatz könnte mehr lagerplatzbezogene Daten enthalten, sodass der Algorithmus komplexere Zusammenhänge von Artikeln zu Lagerplätzen lernen könnte.

Literaturverzeichnis

Wannenwetsch, H. (2014). Lagermanagement. In: Integrierte Materialwirtschaft, Logistik und Beschaffung. Springer-Lehrbuch. Springer Vieweg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-45023-5_9

Pfohl, HC. (2018). Lagerhaus. In: Logistiksysteme. Springer Vieweg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-56228-4_6

Homberger, Jörg, Harald Bauer, und Gabi Preissler. (2019). Operations Research und Künstliche Intelligenz. utb GmbH.

Koop, A., Moock, H. (2018). Lineare Optimierung – eine anwendungsorientierte Einführung in Operations Research. Springer Spektrum, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-56141-6

PulvericH, MicHael, and Jörg ScHietinger. (2009) "Handbuch Kommissionierung." Effizient picken und packen

Koch, S. (2014). Genetische Algorithmen für das Order Batching-Problem in manuellen Kommissioniersystemen. Produktion und Logistik. Springer Gabler, Wiesbaden. https://doi.org/10.1007/978-3-658-05346-8

Burkhard Schwenker, Stefan Bötzel (2006). Wachstum und ständige Optimierung — die Formel für den nachhaltigen Erfolg von Unternehmen. In: Auf Wachstumskurs. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-29349-3_2

Charles J. Malmborg, Krishnakumar Bhaskaran, A revised proof of optimality for the cube-per-order index rule for stored item location, Applied Mathematical Modelling, Volume 14, Issue 2, 1990, Pages 87-95, ISSN 0307-904X, https://doi.org/10.1016/0307-904X(90)90076-H.

J. Xie, Y. Mei, A. T. Ernst, X. Li and A. Song, "A genetic programming-based hyper-heuristic approach for storage location assignment problem," 2014 IEEE Congress on Evolutionary Computation (CEC), 2014, pp. 3000-3007, doi: 10.1109/CEC.2014.6900604.

Harald Gleißner, J. Christian Femerling (2008). Lager-, Umschiags- und Kommissioniersysteme. In: Logistik. Gabler. https://doi.org/10.1007/978-3-8349-9547-6_5

Schuur, Peter C., "The Cube Per Order Index Slotting Strategy, How Bad Can It Be?" (2014). 13th IMHRC Proceedings (Cincinnati, Ohio. USA – 2014). 26.
https://digitalcommons.georgiasouthern.edu/pmhr_2014/26

Mantel, Ronald & Schuur, Peter & Heragu, Sunderesh. (2007). Order oriented slotting: A new assignment strategy for warehouses. European Journal of Industrial Engineering. 1. 301-316. 10.1504/EJIE.2007.014689.

Reyes, J., Solano-Charris, E & Montoya-Torres, J. (2019). The storage location assignment problem: A literature review. International Journal of Industrial Engineering Computations , 10(2), 199-224.

Glover F., Future paths for integer programming and links to artificial intelligence, Computers & Operations Research, Volume 13, Issue 5, 1986, Pages 533-549, ISSN 0305-0548, https://doi.org/10.1016/0305-0548(86)90048-1.

Harbich, Steffen. "Einführung genetischer algorithmen mit anwendungsbeispiel." Universität Magdeburg, December (2007).

Koza, J.R., Poli, R. (2005). Genetic Programming. In: Burke, E.K., Kendall, G. (eds) Search Methodologies. Springer, Boston, MA. https://doi.org/10.1007/0-387-28356-0_5

S. Luke, L. Panait, "A Comparison of Bloat Control Methods for Genetic Programming," in Evolutionary Computation, vol. 14, no. 3, pp. 309-344, Sept. 2006, doi: 10.1162/evco.2006.14.3.309.

Poli, R. (2003). A Simple but Theoretically-Motivated Method to Control Bloat in Genetic Programming. In: Ryan, C., Soule, T., Keijzer, M., Tsang, E., Poli, R., Costa, E. (eds) Genetic Programming. EuroGP 2003. Lecture Notes in Computer Science, vol 2610. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-36599-0_19

Fang, Yongsheng, and Jun Li. "A review of tournament selection in genetic programming." Advances in Computation and Intelligence: 5th International Symposium, ISICA 2010, Wuhan, China, October 22-24, 2010. Proceedings 5. Springer Berlin Heidelberg, 2010.

Xie, H., Zhang, M., Andreae, P., Johnson, M. (2008, July). An analysis of multi-sampled issue and no-replacement tournament selection. In Proceedings of the 10th annual conference on Genetic and evolutionary computation (pp. 1323-1330).

Walker, M. (2001). Introduction to genetic programming. Tech. Np: University of Montana.

Desale, S., Rasool, A., Andhale, S., & Rane, P. (2015). Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. Int. J. Comput. Eng. Res. Trends, 351(5), 2349-7084.

Luke, S., & Spector, L. (1997). A comparison of crossover and mutation in genetic programming. Genetic Programming, 97, 240-248.

Poli, R., & Langdon, W. B. (1998). On the search properties of different crossover operators in genetic programming. Genetic Programming, 293-301.

Selzam, B. (2003). Genetische Algorithmen. In Uni Dortmund.

Piszcz, Alan & Soule, Terence. (2006). A survey of mutation techniques in genetic programming. GECCO 2006 - Genetic and Evolutionary Computation Conference. 1. 951-952. 10.1145/1143997.1144165.

Malkauthekar, M. D. (2013, October). Analysis of Euclidean distance and Manhattan distance measure in Face recognition. In Third International Conference on Computational Intelligence and Information Technology (CIIT 2013) (pp. 503-507). IET.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorstehende Bachelorthesis mit dem Titel

Einsatz der Genetischen Programmierung zu Lösung des Storage Location Assignment Problems anhand eines konkreten Anwendungsfalls

selbstständig ohne fremde Hilfe gefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quelle kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum, Unterschrift