JOHANNES KEPLER
UNIVERSITÄT LINZ

Experimentelle Evaluierung heuristischer Suchalgorithmen zur Optimierung von Abflugreihenfolgen im Air Traffic Flow Management

Eingereicht von
Samuel Liam Jaburek, BSc

Angefertigt am
Institut für Wirtschaftsinformatik - Data & Knowledge Engineering

Beurteiler
Assoz.-Prof. Mag. Dr.
Christoph Schütz

Oktober 2021

Masterarbeit

zur Erlangung des akademischen Grades

Master of Science

im Masterstudium

Wirtschaftsinformatik

JOHANNES KEPLER UNIVERSITÄT LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Masterarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, 7. Oktober 2021

Kurzfassung

Das Projekt SlotMachine verfolgt das Ziel, die Zuteilung von Flugverkehrsmanagement-Slots im Falle von Kapazitätsengpässen zu optimieren. Insbesondere beschäftigt sich das Projekt mit der Frage, wie die Abflugreihenfolge von Flügen im Falle von Kapazitätsengpässen optimiert werden kann. Um die Vertraulichkeit der Daten zu gewährleisten werden die Suche nach optimalen Lösungen und die Evaluierung der gefundenen Lösungen von getrennten Komponenten durchgeführt. Die Evaluierung gefundener Lösungen erfolgt mittels Multi-Party-Computation. Die Suche nach optimalen Lösungen erfolgt mittels heuristischer Suche.

In dieser Arbeit werden verschiedene Methoden zur heuristischen Suche von optimalen Abflugreihenfolgen in Bezug auf ihre Eignung für das Projekt SlotMachine experimentell untersucht. Experimente sollen zeigen, inwiefern verschiedene Optimierungsalgorithmen für die Optimierung von Abflugreihenfolgen geeignet sind. Insbesondere werden genetische Algorithmen und diverse lokale Suchalgorithmen untersucht. In den Experimenten werden existierende Open-Source-Frameworks mit unterschiedlichen Einstellungen verwendet, um optimale Abflugreihenfolgen in unterschiedlichen Szenarien in Bezug auf die Beschaffenheit der Präferenzen der Flüge zu ermitteln.

Die in dieser Arbeit durchgeführten Experimente zeigen, dass die mittels heuristischer Suche gefundenen Ergebnisse im Allgemeinen nahe der optimalen Lösung sind, wie sie von der (exakten) Ungarischen Methode gefunden wird.

Abstract

The SlotMachine project aims to optimize the allocation of air traffic management slots in the event of capacity bottlenecks. In particular, the project addresses the question of how the departure sequence of flights can be optimized in the event of capacity bottlenecks. To ensure the confidentiality of the data, the search for optimal solutions and the evaluation of the solutions are carried out by separate components. The evaluation of the found solutions is performed by means of multi-party computation. The search for optimal solutions is performed using heuristic search.

In this work, different methods for the heuristic search of optimal departure sequences are experimentally investigated with respect to their suitability for the SlotMachine project. Experiments determine to what extent different optimization algorithms are suitable for the optimization of departure sequences. In particular, genetic algorithms and various local search algorithms will be investigated. In the experiments, existing open source frameworks with different settings are used to determine optimal departure sequences in different scenarios with respect to the nature of the preferences of the flights.

The experiments conducted in this work show that the results found using heuristic search are generally close to the optimal solution as found by the (exact) Hungarian method.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Codeverzeichnis

1 Einleitung

Diese Arbeit wurde als Teil des Projekts SlotMachine durchgeführt, welches durch das SESAR Joint Undertaking im Rahmen des Forschungs- und Innovationsprogramms Horizon 2020 der Europäischen Union gefördert wird. Im Projekt-Konsortium vertreten sind EUROCONTROL, SWISS, AIT, Frequentis sowie das Institut für Wirtschaftsinformatik - Data & Knowledge Engineering der Johannes Kepler Universität Linz. Das Projekt SlotMachine beschäftigt sich mit der Entwicklung einer Plattform für die Optimierung von Abflugreihenfolgen im Flugverkehrsmanagement. Im Folgenden wird das Projekt SlotMachine kurz vorgestellt, gefolgt von einer Beschreibung der Zielsetzung und dem Aufbau dieser Masterarbeit.

1.1 Projekt SlotMachine

Das Projekt SlotMachine verfolgt das Ziel, die Zuteilung von Flugverkehrsmanagement-Slots im Falle von Kapazitätsengpässen zu optimieren. Insbesondere beschäftigt sich das Projekt mit der Frage wie Abflugzeiten von Flugzeugen an Flughäfen im Falle von Kapazitätsengpässen optimiert werden können.

Aktuell kann im Flugverkehrsmanagement ein Slot-Austausch nur zwischen Flügen der gleichen Fluglinie abgewickelt werden [27, 28]. Da Flugkostenstrukturen vertrauliche Daten beinhalten, wie Zuschlagszahlungen bei Verspätungen oder Überstundenpauschalen, wollen Fluglinien die gewünschten Abflugszeiträume geheimhalten. Bei der Kostenstruktur handelt es sich typischerweise nicht um eine lineare Funktion, sondern um eine Stufenfunktion mit signifikanten Sprüngen, wenn ein bestimmter Zeitpunkt für den Abflug verpasst wird. Diesen Zeitpunkt bezeichnet man auch als Verspätungsziel (engl. delay target). Flüge können auch mehrere Verspätungsziele haben.

Im Falle von Verspätungen sollen Flüge idealerweise nicht später als ihr(e) Verspätungsziel(e) starten. Im Projekt SlotMachine soll dementsprechend ein System entwickelt werden, welches genutzt werden kann, um die Abflugreihenfolge zu optimieren, unter Berücksichtigung der Verspätungsziele beziehungsweise der Kostenfunktionen. Dafür werden Flüge zwischen verschiedenen Fluglinien getauscht, die Abflugreihenfolge wird geändert. Das Finden der optimalen Abflugreihenfolge erfolgt dabei in einer Art und Weise, dass sensible Daten, d.h, die Verspätungsziele, geheim bleiben.

Um die Vertraulichkeit der Daten zu gewährleisten wird die Suche nach optimalen Lösungen und die Evaluierung der gefundenen Lösungen von getrennten Komponenten durchgeführt. Die Evaluierung gefundener Lösungen erfolgt mittels Multi-Party-Computation [11] auf vertraulichkeitsbewahrende Weise. Die Suche nach optimalen Lösungen erfolgt mittels heuristischer Suche. Dabei werden gefundene Zwischenlösungen iterativ auf Basis der vertraulichkeitsbewahrenden Evaluierung der Lösungen verbessert.

1.2 Zielsetzung der Arbeit

In dieser Arbeit sollen verschiedene Methoden zur heuristischen Suche von optimalen Abflugreihenfolgen von Flügen in Bezug auf ihre Eignung für das Projekt SlotMachine experimentell untersucht werden. Experimente sollen zeigen, ob und mit welchen Einstellungen unterschiedliche Metaheuristiken für die Optimierung von Abflugreihenfolgen geeignet sind. Insbesondere sollen genetische Algorithmen und diverse lokale Suchalgorithmen auf ihre Eignung hin untersucht werden. Dabei sollen unterschiedliche Szenarien betreffend der Präferenzen der Fluglinien in Bezug auf die Abflugreihenfolge berücksichtigt werden. Um die Experimente durchführen zu können soll ein Prototyp der Optimierungskomponente für das SlotMachine-System erstellt werden, wobei die vertraulichkeitsbewahrende Evaluierung der Lösungen nicht umgesetzt werden soll. Ein konfigurierbarer Testdatengenerator soll entwickelt werden um Daten für die Experimente zu generieren.

1.3 Aufbau der Arbeit

Im nächsten Abschnitt der Masterarbeit wird der State of the Art betrachtet und vorhandene Quellen zu Metaheuristik-Algorithmen und zu Frameworks werden zusammengefasst dargestellt. Zusätzlich werden verwandte Arbeiten zur Optimierung von Prozessen bei Abflügen betrachtet. In Abschnitt 3 wird ein kurzer Überblick über die Gesamtarchitektur die implementierten Prototyps gegeben. Die Datenverarbeitung wird im Abschnitt 4 betrachtet. Hierfür wird das Excel-Workbook, die Komponente Excel I/O und der Testdatengenerator als Bestandteile des Prototyps erläutert und die Funktionen der Einzelteile erklärt. In Abschnitt 5 wird das Kernstück des Prototyps betrachtet, der heuristische Optimierer. Die Schnittstellen des REST-Servers werden offengelegt und es wird erklärt, wie die Fitnessberechnung durchgeführt wird. Außerdem werden die Einstellungsmöglichkeiten der Frameworks Jenetics und OptaPlanner und der Ungarischen Methode gezeigt. In Abschnitt 6 werden die ersten Ergebnisse aufgelistet, anhand derer die Konfigurationen für die Hauptstudie ausgewählt und eingestellt wurden. In Abschnitt 7 wird die Hauptstudie präsentiert und die Ergebnisse werden zusammengefasst. Abschließend folgt das Fazit über die Masterarbeit. Der Anhang gibt einen detaillierten Überblick über die Hauptstudie in Bezug auf die erreichten Fitnesswerte.

2 State of the Art

In dieser Masterarbeit werden unterschiedliche Begriffsdefinitionen, Konzepte und Tools verwendet. Diese werden in den folgenden Abschnitten genauer und einsteigend beschrieben.

2.1 Metaheuristiken

In diesem Abschnitt wird eine Auswahl heuristischer und nicht heuristischer Algorithmen gegeben, die geeignet sind das SlotMachine-Optimierungsproblem zu lösen. Die Einteilung der Algorithmen in verschiedene Untergruppen erfolgt anhand der Systematik des OptaPlanner-Handbuchs [26]. In der Literatur gibt es auch andere Einteilungen, welche bei den Algorithmen kurz angeführt werden. Die der Algorithmen in die verschiedenen Untergruppen erfolgt anhand der späteren Einteilung von OptaPlanner [26].

2.1.1 Definitionen

Heuristiken sind Entscheidungsregeln, welche vom Benutzer an den Algorithmus übermittelt werden. Zu beachten ist, dass diese Regeln falsch sein können und nur als Richtlinie bedeutsam sind [24]. Der englische Begriff „Heuristics“ ist grob übersetzt ebenso Heuristik. Algorithmen für Optimierung können in zwei Gruppen unterteilt werden:

- Exakte Algorithmen garantieren die optimale Lösung zu finden. Dafür kann die Laufzeit unter Umständen zu hoch für bestimmte Anwendungsfälle sein, da sie beweisen müssen, dass ihre Lösung tatsächlich die beste ist [34].

- Heuristiken garantieren jedoch nur, dass sie eine möglichst optimale Lösung finden, diese dafür jedoch in einer wesentlich kürzeren Zeit als bei exakten Algorithmen. Sie sind der Algorithmentyp, welche bei einer großen Zahl von Optimierungsproblemen überhaupt eine Lösung zurückgeben, da exakte Algorithmen hierfür zu lange benötigen [34].

Metaheuristiken basieren auf den gleichen Prinzipien wie Heuristiken zu den Anfangszeiten, indem sie Lösungen ähnlich suchen, wie die menschliche Intuition es vorgibt (siehe als Beispiel Tabu Search und andere) [34]. Ein aktueller Ansatz ist es, dass vorhandene Metaheuristiken untersucht und die besten für das jeweilige Problemfeld ausgesucht und verwendet werden. Der Begriff Metaheuristik wurden zuerst 1986 in Kombination mit Tabu Search betrachtet, jedoch nicht genauer definiert [16]. Sörenson und Glover [35] definierten es als problemunabhängiges Framework, welches einen Satz an Strategien vorgibt, welche genutzt werden können, um heuristische Optimierungsalgorithmen zu entwickeln. Metaheuristiken existieren unabhängig von Problemen und können für viele Felder angewendet werden. Sörenson beschreibt eine Metaheuristik auch als eine Art Kochstil [34].

2.1.2 Optimierungsalgorithmen

Die in den folgenden Abschnitten benannten Algorithmen werden genauer untersucht und für spätere Experimente mit dem Prototyp herangezogen. Die Unterteilung wird hier ähnlich der vom OptaPlanner-Handbuch [26] vollzogen.

Exakte Algorithmen

Exakte Algorithmen als Algorithmentyp entspricht im näheren Sinne keinem heuristischen Algorithmus, wie die anderen genauer betrachteten Algorithmen. Im Gegenzug zu den anderen untersuchten Algorithmen wird das globale Optimum gefunden, jedoch skalieren sie generell nicht gut und sind daher für größere Datenmengen nicht zu gebrauchen[26].

Ungarische Methode (Hungarian Algorithm) Die Ungarische Methode, auch als Hungarian Algorithm bekannt, kann Zuteilungsprobleme zwischen Flugzeugen und offenen Slots lösen [23]. Es ein Algorithmus zur Zuteilung von Ressourcen zu Einheiten, worin versucht wird die Kosten zu minimieren. Angepasst kann es auch für Planungsprobleme verwendet werden, wenn Zeiten Kosten zugewiesen bekommen. Jedoch ist er nicht für die Nutzung mit definierten Restriktionen entwickelt worden [37]. Der Algorithmus ist so konzipiert, dass er eine Laufzeit von $O(n^3)$ hat [37] und verwendet mathematische Konstrukte wie Matrizen [43]. Details sind im Paper von Kuhn [23] nachlesbar. Ein beispielhafter Ablaufprozess kann von Zhou [43] nachgelesen werden.

Evolutionäre Algorithmen

Algorithmen von diesem Typ arbeiten mithilfe einer Bevölkerung an Lösungen und der Evolution dieser Bevölkerung [26]. In diesem Sinne zählt der genetische Algorithmus dazu, da hier eine Bevölkerung verwendet wird, die sich mittels diverser Strategien verbessern soll [24].

Genetischer Algorithmus Der genetische Algorithmus ist ein auf Zufall basierender Algorithmus, welcher mit den Prinzipien der natürlichen Auswahl und Genetik arbeitet [12, 18]. Im Suchprozess wird dadurch bewirkt, dass die optimalsten Individuen die durchlaufenen Generationen überstehen und somit eine möglichst optimale Lösung nach möglichst wenigen Generationen gefunden wird. Die Individuen werden mit einem „Fitnesswert“ bewertet. Umso höher dieser Wert ist, umso näher befindet sich das Individuum dem Optimum. Der genetische Algorithmus benötigt als Ausgangspunkt eine Initialbevölkerung mit möglichen Lösungen [12, 18, 24].

Als genetische Mechanismen werden der Prozess der Auswahl und der Operationen an dieser Auswahl verstanden. Üblich als Operationen für die Rekombinationsphase sind hier Fortpflanzung [18], Crossover und Mutation [12, 18, 24]. Die Auswahl wird anhand gemessener Fitnesswerte der alten Bevölkerung durchgeführt. Zuerst wird ein Teil der alten Bevölkerung in die nächste Generation übernommen. Danach ändern sich zusätzlich einige Individuen in der Rekombinationsphase, um neue Individuen zu formen und dadurch neue potenzielle Lösungen zu erzeugen [18]. Verschiedene Selektoren

und Rekombinationsoperationen werden von Luke [24] beschrieben. Eine mögliche Implementierung wird in Algorithmus 1 dargestellt.

Drei der möglichen Änderungen sind wie folgt:

- **Fortpflanzung**: Mit dieser Operation werden alte Individuen lediglich in ihrer Form in ein neues Individuum für die neue Bevölkerung kopiert [18].
- **Crossover**: Für zwei ausgewählte Individuen wird zufällig ein Crossover-Punkt bestimmt. Informationen vom ersten Individuum werden bis zu diesem Punkt für das neue Individuum übernommen. Nach diesem Punkt erhält das neue Individuum die Daten vom zweiten Individuum [18]. Es gibt auch andere Mechaniken zum Vermischen der Informationen von zwei Individuen, wie die Nutzung von zwei Crossover-Punkten [24].
- **Mutation**: Um neue Informationen zu erlangen, werden hier von einem Individuum zufällige Änderungen in dessen Informationen durchgeführt. So kann der genetische Algorithmus auch lokalen Optima entkommen [18, 24].

Nachteil des genetischen Algorithmus ist die Notwendigkeit einer Initialbevölkerung [18] sowie der Tatsache, dass nicht immer die optimalste Lösung gefunden wird [12]. Details zum Algorithmus können in den referenzierten Quellen [12, 18, 24] nachgeschlagen werden.

Konstruktionsheuristiken

Konstruktionsheuristiken sind meistens gut darin, schnell initiale Lösungen zu finden. Diese können von anderen Algorithmen verbessert werden [17]. Initiallösungen müssen nicht unbedingt alle geforderten Einschränkungen erfüllen, da sie ohnehin nur als Basis für andere Algorithmen verwendet werden, die zu besseren Endlösungen führen [26]. Basierend auf den Konstruktionsheuristiken, die OptaPlanner [26] unterstützt, werden diese kurz beschrieben. Die „Fit“-Algorithmen basieren auf der „First-Fit“-Heuristik, die grob Folgendes aussagt: Die Behälter, wo die Elemente eingefügt werden, und die Elemente sollen nach definierten Charakteristiken sortiert werden. Danach werden die Elemente zuerst in den ersten Behälter eingefügt. Wenn dieser voll ist oder kein Platz mehr ist, werden die Elemente in den zweiten Behälter eingefügt. Dies geschieht so lange, bis

```
1  populationSize ← gewünschte Bevölkerungsgröße
                    /* populationSize sollte bestenfalls eine gerade Zahl sein */
2  for populationSize-mal do
3  |  P ← P ∪ {neues zufälliges Individuum}
4  end
5  Best wird ohne Wert initialisiert
6  repeat
7  |  foreach Individuum P_i ∈ P do
8  |  |  Fitness von P_i berechnen (Fitness(P_i))
9  |  |  if Best ist ohne Wert oder Fitness(P_i) > Fitness(Best) then
10 |  |  |  Best ← P_i
11 |  |  end
12 |  end
13 |  Q ← {}
14 |  for (populationSize/2)-mal do Fortpflanzung, Crossover und Mutation
15 |  |  Elternteil P_a ← AuswahlMitErsetzen(P)
16 |  |  Elternteil P_b ← AuswahlMitErsetzen(P)
17 |  |  Kinder C_a, C_b ← Crossover(Kopie(P_a), Kopie(P_b))
18 |  |  Q ← Q ∪ {Mutation(C_a), Mutation(C_b)}
19 |  end
20 |  P ← Q
21 until Best eine ideale Lösung ist oder die Zeit abgelaufen ist
22 return Lösung Best
```

Algorithmus 1 : Genetischer Algorithmus [24]

alle Elemente eingefügt wurden oder alle Behälter voll sind [21, 41]. Keller [21] bezeichnet die Heuristik als „First Fit Decreasing“. Kim [22] bezeichnet es ebenso und beschreibt den Algorithmus zusammengefasst so: Jedes Element einer sortierten Liste wird sequenziell dem Behälter mit dem kleinsten Index zugeordnet. Wenn nicht mehr genug Platz in dem Behälter ist, wird das aktuelle Element einem neuen Behälter zugeordnet. Nachfolgend werden die untersuchten Algorithmen, die sich vom beschriebenen Konzept ableiten, in ihrer Funktionsweise und Abweichungen näher beschrieben.

- **First Fit**: Jedes Element wird in der gegebenen Sortierung abgearbeitet und initialisiert mit den besten noch verfügbaren Werten. Nachdem alle Elemente einen zugewiesenen Wert vorweisen können, ist der Algorithmus abgeschlossen [26].
- **First Fit Decreasing**: Der First-Fit-Decreasing-Algorithmus sortiert alle Elemente anhand ihrer definierten Schwierigkeit. Somit sollen möglichst schwer zu planende Elemente zuerst Werte zugewiesen bekommen. Danach bekommen die leichter planbare Elemente Werte zugewiesen. Es werden immer nur noch verfügbare Werte zugewiesen [26].
- **Weakest Fit**: Unterschiedlich zum First-Fit-Algorithmus arbeitet der Weakest-Fit-Algorithmus nicht mit der Planbarkeit, sondern mit dem Konzept der Schwäche. Ein Element gilt als schwach, wenn er wahrscheinlich nicht für viele Elemente verwendbar ist. Dieser Algorithmus teilt den Elementen zuerst die verfügbaren schwächsten Elemente zu [26].
- **Weakest Fit Decreasing**: Dieser Algorithmus kombiniert First Fit Decreasing und Weakest Fit. Die Elemente werden demnach anhand ihrer Schwierigkeit sortiert. Möglichst schwer zu planende Elemente bekommen zuerst die schwächsten möglichen Werte zugewiesen [26].
- **Strongest Fit**: Im Gegensatz zu Weakest Fit weist der Strongest Fit zuerst die stärksten Werte den Elementen zu. Ein Wert ist stark, wenn er für möglichst viele Elemente verwendbar ist [26].
- **Strongest Fit Decreasing**: Dieser Algorithmus kombiniert First Fit Decreasing und Weakest Fit. Die Elemente werden demnach anhand ihrer Schwierigkeit sortiert. Möglichst schwer zu planende Elemente bekommen zuerst die stärksten möglichen Werte zugewiesen [26].

Cheapest Insertion Dieser Algorithmus arbeitet die verfügbaren Werte der Reihenfolge ab und weist diese den bestmöglichen Elementen zu. Hierbei werden die bereits zugewiesen Elemente beachtet. Kein Element wird doppelt zugewiesen und der Algorithmus beendet sich, nachdem alle Elemente einen Wert zugewiesen bekommen haben [26]. Es ist ein schneller Algorithmus, welcher auch mit vielen Eingaben stabil arbeitet [29]. Alssager [1] beschreibt den Cheapest-Insertion-Algorithmus ebenso im Detail und zeigt einen Schwachpunkt auf: Die erstellte Lösung kann so konstruiert sein, dass sie zu hohe Kosten hat oder nicht alle Einschränkungen beachtet und daher nicht gültig ist. Durch die Nutzung eines anderen Algorithmus, der als Basis die hier generierte Lösung nutzt, hat dieser Schwachpunkt keine größere Auswirkung.

Lokale Suchalgorithmen

Laut OptaPlanner-Handbuch [26] werden die in diesem Abschnitt als lokale Suchalgorithmen behandeln. Laut Simon [32] können einige der lokalen Suchalgorithmen (beispielsweise Tabu Search) auch als evolutionären Algorithmen betrachtet werden. Gemeinsamkeiten der Algorithmen bestehen in der Notwendigkeit einer Initiallösung und dass nur ein einziger Suchpfad verfolgt wird im Gegensatz zur Möglichkeit eines Suchbaums [26].
Lokale Suche arbeitet mit Suchschritten. Pro Suchschritt werden eine definierte Anzahl an Bewegungen anhand ihrer Tauglichkeit getestet. Danach wird die beste Bewegung für den Suchschritt ausgewählt. Basierend auf der neuen Position werden nun die nächsten Bewegungen für den nächsten Suchschritt durchgeführt. Die Bewegung wird nur dann weiter untersucht, wenn sie ausgewählt wird, wodurch kein Suchbaum sondern ein Suchpfad entsteht. Für die endgültige Lösung werden üblicherweise eine hohe Anzahl an Suchschritten ausgeführt [26]. Im Feld der heuristischen Algorithmen nutzen metaheuristische Algorithmen eine Vorgehensweise, die auf viele andere Problemfelder anwendbar ist und nicht für das Problem spezifisch entwickelt wurden. Die meisten sind inspiriert von Prinzipien aus der Natur, nutzen Zufallselemente und haben einige einstellbare Parameter. Durch die Parameter können diese Algorithmen den Problemen angepasst werden [7].

Hill Climbing Hill Climbing ist einer der ersten Vertreter der lokalen Suchalgorithmen. Gestartet wird mit einer stochastischen Lösung. Danach wird der Hügel erklommen, indem nur Bewegungen für den aktuellen Lösungsschritt akzeptiert werden, die optimaler als die aktuelle ist [6]. Als Bewegung zählt beispielsweise das Bewegen einer Figur am Schachbrett. Wie in Algorithmus 2 ersichtlich ist, erfolgen Bewegungen in der Nachbarschaft der aktuellen Lösung. Pro Lösungsschritt wird eine definierte Anzahl an halbzufällig ausgewählten Bewegungen mittels Funktion bezüglich der Bewertung geprüft. Nach der Prüfung aller Bewegungen gewinnt die optimalste [24, 26].

```
1 s ← Initiallösung (zufällig gewählt oder vorher erstellt)
2 repeat
3 |   Wähle eine Lösung s' ∈ Nachbarschaft(s)
4 |   if f(s') > f(s) then
  |   |                                  /* f(s) bewertet die Qualität der Lösung */
5 |   |   s ← s'
6 |   end
7 until s eine ideale Lösung ist oder die Zeit abgelaufen ist
8 return Lösung s
```

Algorithmus 2 : Hill Climbing Algorithmus [24]

Da für Hill Climbing immer nur Bewegungen im aktuellen Suchschritt gewinnen, bleibt der Algorithmus möglicherweise in lokalen Optima hängen. Dies passiert, wenn die aktuelle Lösung optimaler als alle anderen Lösungen ist, die in der Nachbarschaft erreichbar sind. Daher werden andere Algorithmen, die Hill Climbing verbessern, wie Tabu Search, Simulated Annealing, zur Nutzung statt Hill Climbing empfohlen [6, 24, 26].

Tabu Search Dieser Metaheuristik-Algorithmus sucht die Nachbarschaft ab. Es speichert die Bewertungen von vergangenen Bewegungen, basierend auf der aktuellen Lösung, je nach Information unterschiedlich lange ab und kontrolliert die Bewegungen, die basierend auf der aktuellen Lösung durchgeführt werden. Je nach Einstellung werden alte Bewegungen in Bezug auf Werte oder Elemente für eine bestimmte Anzahl an Bewegungen auf eine Tabuliste gesetzt, die für die nächsten Bewegungen nicht verwendet werden dürfen [2, 24, 26]. Dies wird ebenso im Algorithmus 3 als kurze, grobe Übersicht über einen möglichen Ablauf von Tabu Search dargestellt. Die Aktualisierung der *TabuList* bezieht sich auf das Löschen des ältesten Elements und dem Hinzufügen des neuesten Elements, womit die *TabuList* durchgehend die korrekte Anzahl an Elementen beinhaltet.

Somit wird verhindert, dass die akzeptierten Bewegungen sich im Kreis bewegen. Dies führt jedoch dazu, dass auch schlechtere Bewegungen als die aktuelle Lösung für den aktuellen Schritt akzeptiert werden [7]. In Fällen, wo es sehr viele bis unendlich viele Lösungsmöglichkeiten gibt, wie es bei reellen Zahlen der Fall ist, ist es unwahrscheinlich die exakt gleiche Lösung mehrfach auszuwählen. Daher ist hier ein anderer Algorithmus zu bevorzugen. Ebenso besteht die Gefahr bei vielen Lösungsmöglichkeiten in der nahen Nachbarschaft zu bleiben. Die Gefahr kann verringert werden, indem man andere Aspekte nutzt, die mithilfe der Tabu Liste für die nächsten Schritte verboten werden [24].

1 $s \leftarrow$ Initiallösung (zufällig gewählt oder vorher erstellt)
2 $TabuList \leftarrow \emptyset$
3 **while** *die Abbruchbedingung nicht erfüllt ist* **do**
4 | Wähle die beste Lösung $s' \in Nachbarschaft(s) \setminus TabuList$
5 | $s \leftarrow s'$
6 | Aktualisiere *TabuList*
7 **end**
8 **return** *die beste gefundene Lösung*

Algorithmus 3 : Tabu Search Algorithmus [7]

Tabu Search kann ebenso wie Hill Climbing in lokalen Optima hängen bleiben, wenn die Tabuliste mit zu wenigen Elementen definiert ist [26].

Simulated Annealing Im Gegenzug zu den vorigen Algorithmen bewertet dieser Algorithmus grob gesehen nur wenige Bewegungen pro Suchschritt, bevor die gewinnende Bewegung bestimmt wird. Die erste Bewegung, die eine optimalere Lösung als die vorige hat, gewinnt. Durch einen Zufallscheck können jedoch auch Bewegungen gewinnen, die nicht optimaler als die vorige Lösung sind [26]. Im Detail (siehe Boussaid [7]) arbeitet der Algorithmus (siehe Algorithmus 4) mit einer fiktiven Temperatur. Er holt seine Inspiration vom gezielten Abkühlen von Metall [24]. Pro Bewegung wird eine potenzielle Lösung aus der Nachbarschaft (N) der aktuellen Lösung (s) ausgewählt und von dieser der Wert anhand einer objektiven Funktion bestimmt. Sofern dieser Wert niedriger oder gleich dem Wert der vorig ausgewählten Lösung ist, wird die potenziellen Lösung zur neuen Lösung. Ansonsten wird anhand einer Wahrscheinlichkeit ebenso eine Lösung, die schlechter ist und demnach einen höheren Wert hat, zur neuen Lösung bestimmt. Die Wahrscheinlichkeit ist von der sinkenden Temperatur abhängig und wird mit dem Fortschritt der Suche geringer, wie in Zeile 6 vom Algorithmus 4 gezeigt wird [7]. Dem-

nach, sofern die Temperatur sehr gering ist, ähnelt dieser Algorithmus sehr dem Hill Climbing Algorithmus [24].

```
1  s ← Initiallösung (zufällig gewählt oder vorher erstellt)
2  t ← Temperatur (initial eine hohe Zahl)
3  Aktuell beste Lösung l ← s
4  repeat
5  |  Wähle s' ∈ N(s) aus
6  |  if f(s') > f(s) oder Zufallszahl zwischen 0 und 1 < e^((f(s')−f(s))/t) then
   |  |                                  /* f(s) bewertet die Qualität der Lösung */
7  |  |  s ← s'
8  |  end
9  |  Verringere t
10 |  if f(s) > f(l) then
11 |  |  l ← s
12 |  end
13 until Lösung l ist eine ideale Lösung, Zeit ist abgelaufen oder t ≤ 0
14 return die Lösung l
```

Algorithmus 4 : Simulated Annealing Algorithmus [7, 24]

Great Deluge Ähnlich zum Simulated Annealing Algorithmus bewertet Great Deluge nur einige Bewegungen pro Suchschritt. Die erste Bewegung, welche optimaler als das aktuelle Wasserlevel ist, gewinnt. Das Wasserlevel steigt pro Runde um einen vordefinierten Wert. Dadurch soll dem Algorithmus Zeit gegeben werden, dass er lokalen Optima entkommen kann [26]. Laut Baykasoglu [5] kann der Algorithmus auch so beschrieben werden, dass eine Person (entspricht der gewählten Lösung) so hochklettert, dass die Füße nicht nass werden, während das Wasser ansteigt. Baykasoglu [5] implementiert Great Deluge etwas anders (siehe Algorithmus 5 als mögliche Implementierung eines Great Deluge Algorithmus). Hier wird eine Lösung akzeptiert, die mit der Funktion $f()$ geringere Werte hat als die aktuelle Lösung. Außerdem sinkt das Level l um Δl, wodurch der Wasserlevel bestimmt wird. Lösungen s' werden akzeptiert, die unter dem aktuellem Level l sind. Eine Lösung s ist besser, wenn $f()$ geringer ist. Für Details siehe referenzierte Quellen [5, 26].

Late Acceptance Late Acceptance, auch bekannt als Late Acceptance Hill Climbing [26], ist ähnlich zum Hill Climbing Algorithmus [8, 13, 42]. Im Unterschied zu diesem

```
1  s ← Initiallösung (zufällig gewählt oder vorher erstellt)
2  Berechne initiale Kostenfunktion f(s) und setze Level l ← f(s)
3  repeat
4  |  s' ∈ Nachbarschaft(s)
5  |  if f(s') ≤ f(s) then
6  |  |  s ← s'
7  |  else if f(s') ≤ l then
8  |  |  s ← s'
9  |  end
10 |  l ← l − Δl              /* Δl entspricht dem Anstieg des Wasserspiegels */
11 until eine Abbruchbedingung erfüllt ist
12 return Lösung s'
```

Algorithmus 5 : Great Deluge Algorithmus [5]

werden aktuelle Bewegungen mit der Lösung, die vor einer definierten Anzahl an Suchschritten aktuell war, verglichen. Die Länge der Liste, die vergangene Bewertungen von akzeptierten Lösungen beinhaltet, kann vom Nutzer eingestellt werden. Dadurch kann die aktuelle Bewegung mit der Bewertung der Bewegung verglichen werden, die vor der definierten Anzahl geschah [8, 26]. Der Ablauf von Late Acceptance wird bei Algorithmus 6 dargestellt, der eine beispielhafte Implementierung zeigt. Genauere Erklärungen finden sich in den referenzierten Arbeiten [8, 13, 42], die ebenso beispielhafte Implementierungen zeigen.

Step Counting Hill Climbing Dieser Algorithmus wurde mit Inspiration vom Late Acceptance (oder auch Late Acceptance Hill Climbing) Algorithmus und Hill Climbing Algorithmus entwickelt. Er fügt einen Zählmechanismus ein, welcher die Qualität verbessern soll. Eine Bewegung wird nur dann akzeptiert, wenn sie optimaler oder gleich optimal wie ein bestimmter Grenzwert ist. In Algorithmus 7 entspricht dieser Wert b. Der Grenzwert b wird immer nach c Schritten mit dem aktuellen Wert aktualisiert. Dann wird auch die Zählvariable auf 0 gesetzt [9, 31]. Es werden nur wenige Bewegungen pro Suchschritt bewertet [26].

Strategic Oscillation Dieser Algorithmus wird von OptaPlanner verwendet und entspricht einer Erweiterung diverser Algorithmen, wie den Tabu Search Algorithmus. Es

```
1  s ← Initiallösung (zufällig gewählt oder vorher erstellt)
2  L_h ← Anzahl vergangener Schritte, die gespeichert werden sollen
3  Berechne C(s)                    /* C(s) entspricht den Kosten der Lösung s */
4  Speichere alle Kosten der vergangenen Schritte f initial mit Initialkosten C(s)
     ∀k ∈ 0...L_h − 1 f_k := C(s)
5  i ← 0
6  while solange die Abbruchbedingung nicht erfüllt ist do
7  |   Konstruiere Lösung s'        /* beispielsweise aus der Nachbarschaft(s) */
8  |   Berechne C(s')
9  |   Berechne den virtuellen Anfang v := I mod L_h
10 |   if C(s') ≤ f_v then
   |   |    /* vergleiche die aktuelle Lösung mit den Kosten der akzeptierten
   |   |     Lösung vor einer definierten Anzahl an Schritten */
11 |   |   s ← s'
12 |   |   Aktualisiere f (f_v ← C(s))
13 |   end
14 |   i ← i + 1
15 end
16 return Lösung s
```

Algorithmus 6 : Late Acceptance Algorithmus [8, 13, 42]

wird die Wahl der Bewegung beeinflusst, die akzeptiert werden soll [26]. Strategic Oscillation wird auch als Prozess verstanden, welcher versucht die Richtung der Suche nach der besten Lösung zu steuern. So sollen Regionen durchquert werden können, die nur unpassende oder ungültige Lösungen beinhalten. Außerdem soll so garantiert werden, dass mehr Regionen und unterschiedlichere Bewegungen bezüglich ihrer Qualität geprüft wurden, als es sonst der Fall ist [15]. Dieser Prozess ist nicht nur auf Tabu Search beschränkt und kann ebenso beispielsweise für Varianten vom Simulated Annealing Algorithmus angewendet werden [3].

```
1  s ← Initiallösung (zufällig gewählt oder vorher erstellt)
2  c ← Updateintervall
3  s* ← s
4  b ← f(s)
5  i ← 0
6  while Abbruchbedingung nicht erfüllt ist do
7  |  s' ← zufälliger Nachbar einer zufälligen Nachbarschaft k (s' ∈ N_k(s))
8  |  if f(s') ≤ f(s) oder f(s) < b then
9  |  |  s ← s'
10 |  |  if f(s) < f(s*) then
11 |  |  |  s* ← s
12 |  |  end
13 |  end
14 |  if i ≥ c then
15 |  |  b ← f(s)
16 |  |  i ← 0
17 |  end
18 |  i ← i + 1
19 end
20 return Lösung s*
```

Algorithmus 7 : Step Counting Hill Climbing Algorithmus [31]

2.2 Frameworks

Es gibt mehrere verschiedene Frameworks, die es ermöglichen mit geringem Aufwand unterschiedliche heuristische Algorithmen zu implementieren. Fokussiert wurde auf die zwei Frameworks Jenetics und OptaPlanner, da diese leicht in Java einzubinden sind, die Möglichkeit bieten die Fitnesswertberechnung auszulagern und die gewünschten Einschränkungen auf den gültigen Lösungsbereich einstellbar machen. Auf andere Frameworks wird nicht näher eingegangen.

2.2.1 Jenetics

Jenetics ist ein Java Framework, welches eine Implementierung des genetischen Algorithmus mit verschiedenen Einstellungsmöglichkeiten bereitstellt. Es wird von Franz Wilhelmstötter betreut und wird 2021 weiterhin aktiv weiterentwickelt. Das Framework wird in vielen Forschungsprojekten und akademischen Veröffentlichungen verwendet, wie auf der Webseite des Frameworks gezeigt wird [39]. Es wird Jenetics in der Version 6.1.0 verwendet [38, 40].

Im Framework sind folgende Aspekte, die ein Genetischer Algorithmus benötigt, voneinander abgetrennt implementiert: Gene, Chromosomen, Genotypen, Phenotypen, Bevölkerung und die Fitnessfunktion. Die Gene beschreiben die Parameter des Optimierungsproblems gemeinsam mit den möglichen Werten und den aktuellen Werten. Ein Chromosom ist eine Sammlung von mindestens einem Gen. Ein Genotyp entspricht strukturell einem Individuum, welches aus mindestens einem Chromosom besteht. Die Chromosomen eines Genotyps müssen alle den gleichen Gentyp haben. Ein Phenotyp entspricht jedoch tatsächlich einem Individuum und enthält den Genotypen, die Generation, wo es erstellt wurde und optional den dazugehörigen Fitnesswert vom Phenotypen. Genotyp und Phenotyp können nach Erstellung nicht mehr verändert werden. Die Bevölkerung entspricht einer Sammlung von Phenotypen. Die Fitnessfunktion, welche aussagt wie optimal ein Individuum ist, gibt den Fitnesswert anhand der Genotypen zurück [14, 38].

Mittels Jenetics kann das Ergebnis mittels gegebener Fitnessfunktion sowohl maximale oder minimale Werte für die Optimierung erlangen. In der Implementierung nutzt Jenetics einen „EvolutionStream“ für die Ausführung der Evolutionsschritte des genetischen Algorithmus. Das Framework wurde so entwickelt, dass es für eine große Anzahl an

verschiedenen Problemfeldern nutzbar ist. Jenetics bietet bereits getestete „Encodings“ für die gängigeren Probleme, wie Permutation und Matching-Probleme. Ebenso werden unterschiedliche Arten unterstützt, wie Einschränkungen definiert werden. So können ungültige Lösungen berücksichtigt werden. Daher kann mit der Nutzung von Jenetics leicht eine Vielzahl an unterschiedlichen Implementierungen eines genetischen Algorithmus getestet werden [38].

Zusätzlich wurden bereits verschiedene „Alterer“ für Rekombination und Mutation integriert, wie sie für den genetischen Algorithmus verwendet werden können. So kann sichergestellt werden, dass genetische Vielfalt erstellt werden kann. „Alterer“ für Rekombinationen kombinieren bereits vorhandene Lösungen, um zu neuen Lösungen zu gelangen. Für die Mutation wird in kleineren Schritten der Lösungsraum durchsucht und die Vielfalt aufrechterhalten. Durch Mutation entstehen neue Lösungen, die bewertet werden können. Ebenso stellt Jenetics verschiedene „Selectors“ bereit, die dafür sorgen, welcher Teil in der Bevölkerung Nachkomme und welcher Teil Überlebender ist. So wählt ein „Offspring Selector“ den Anteil aus, welcher Nachkomme sein wird und für Rekombination ausgewählt wird. Der „Survivors Selector“ wählt den Anteil aus, welcher überlebt [38].

Mehr Details zu den Einstellungen finden sich in Abschnitt 5.3.

2.2.2 OptaPlanner

OptaPlanner [10] ist ebenso ein Java Framework, welches verschiedene Implementierungen von diversen Optimierungsalgorithmen im Bereich Exhaustive Search, Konstruktionsheuristiken und Metaheuristiken bereitstellt. Es strebt an, eine „Constraint Satisfaction Engine“ zu sein, welche Planungsprobleme optimiert. Ein ausführliches Benutzerhandbuch und eine Dokumentation sind ebenso verfügbar [26], sowie der Code [25]. OptaPlanner wird in Version 8.1.0.Final verwendet.

OptaPlanner arbeitet mit drei Konzepten: Planungsprobleme, Variablen und Lösungen [26].

Es versucht die gegebenen Planungsprobleme mit einem definierten Ziel zu lösen, indem es die gegebenen Einschränkungen beachtet. So kann ein Problem beispielsweise darin bestehen, den Profit eines Fließbandes mit optimierter Zeitplanung zu erhöhen. Mittels der vorimplementierten Algorithmen kann es diese Planungsprobleme lösen, indem es

einen Wert der Lösung berechnet. Anhand dieses Wertes wird die beste Lösung bestimmt. Die Lösungsberechnung kann mittels Drools oder Java erfolgen. Die Konfiguration erfolgt wiederum mittels XML-Datei oder direkt im Java Programmcode [26].

Planungsprobleme folgen in OptaPlanner drei Regeln [26]:

- Die Lösung kann in einer angemessenen Zeit verifiziert werden, aber die optimale Lösung kann nicht in einer akzeptablen Zeit gefunden werden. In den meisten Fällen benötigen die Exakten Algorithmen zu viel Zeit.
- Das Problem hat Einschränkungen, welche den Wert einer Lösung als Zahlenwert bestimmen. Diese können in verschiedenen Arten von Einschränkungen getrennt werden, wie beispielsweise eine harte und eine weiche Einschränkung.
- Das Problem hat viele mögliche Lösungen, weswegen der Suchraum viele mögliche und viele optimale Lösungen hat.

OptaPlanner unterstützt folgende Algorithmen: Brute Force, Branch and Bound, First Fit, First Fit Decreasing, Weakest Fit, Weakest Fit Decreasing, Strongest Fit, Strongest Fit Decreasing, Cheapest Insertion, Regret Insertion, Hill Climbing, Tabu Search, Simulated Annealing, Great Deluge, Late Acceptance, Step Counting Hill Climbing, Strategic Oscillation und Variable Neighborhood Descent. Eine Auswahl davon wurde in vorigen Abschnitten genauer beschrieben. Brute Force und Branch and Bound sollen laut OptaPlanner-Handbuch selbst [26] wegen fehlender Skalierung für größere Probleme nicht verwendet werden.

Es werden von OptaPlanner vier verschiedene Arten zur Ermittlung des Werts der Lösung unterstützt [26]:

- „Easy score“ Berechnung: Alle Einschränkungen werden in einer Methode mittels Java-Code berechnet, wodurch der resultierende Wert im existierenden Code bestimmt werden kann. Jedoch ist diese Methode langsam.
- „Constraint streams score“ Berechnung: Die Einschränkungen werden als individuelle `ConstraintStreams` in Java implementiert. Diese Methode ist schnell und kann gut für für größere Probleme skalieren.

- „Incrementatal Java score“ Berechnung: Einige Methoden im tieferen Code von Java müssen implementiert werden. Der Nachteil dieser Variante ist, dass sie schwer implementierbar und wartbar ist. Dafür ist diese Methode schnell.
- Drools Wertberechnung: Mittels Verwendung von Werteregeln mit Unterstützung von Drools lassen sich die Resultate leicht berechnen. Einschränkungen können separat voneinander angelegt werden.

Mehr Details zu OptaPlanner und seinen Einstellungen finden sich in Abschnitt 5.4.

2.3 Verwandte Arbeiten

Barnhart [4] beschreibt die Situation in den Vereinigten Staaten von Amerika, wo die Verzögerungen und Behinderungen im Luftverkehr auf einen Kostenpunkt von 31,2 Milliarden Dollar für das Jahr 2007 gerechnet wurde. Der Bau von neuen Flughäfen oder neuen Lande- und Startbahnen an den meistbenutzten Flughäfen der Welt ist vielerorts aufgrund politischen Klimas, Landverfügbarkeit oder aus anderen Gründen nicht möglich. Zusätzlich wird laut Hafidi [19] die Anzahl an Flugpassagieren in den kommenden Jahren stark ansteigen, woraufhin mehr Flugzeuge unterwegs sein werden und vorhandene Routen und der Verkehr an den Flughäfen stärker optimiert werden muss, um den Anstieg bewältigen zu können. Daher müssen andere Systeme eingeführt werden, um die bestehenden Kapazitäten an den Flughäfen effizienter zu nutzen. Da Flugkosten nicht linear ansteigen, sondern einige sprungweise Anstiege haben, ist es von Vorteil verspätete Flüge vor dem jeweils nächstem Kostensprung abfliegen zu lassen. Einige Minuten Verspätung bereiten oft keine Probleme. Die Kostensprünge können von Überstunden, Zahlungen an Passagiere wegen großen Verspätungen oder anderen Faktoren abhängen, die nur den Fluglinien selbst bekannt sind [27].

Die Problematik die Abflugreihenfolgen am Flughafen zu optimieren, um Kosten zu sparen und um für mehr Effektivität zu sorgen, wurde bereits in ähnlichen Arbeiten ebenso untersucht. Allerdings wurden 2013 nur 0,2% aller möglichen Flüge in Europa untereinander getauscht, um größere Kosten von priorisierten Flügen zu vermeiden [28]. Im ATFM-System können Fluglinien zwei ihrer Flüge aufgrund Verspätungen tauschen. Erweiterte ATFM-Systeme, wie ESS (enhanced ATFM slot swapping), können

zu Ersparnissen von etwa 4900 € an nicht zustande gekommen Verlusten pro Tausch führen [28].

Eine der Möglichkeiten besteht in der besseren Zuteilung von Abflugslots. Außerdem können Tausch von Slots zwischen zwei verschiedenen Flügen dazu führen, dass priorisierte Flüge weniger Verspätungen haben und weniger relevante Flüge in den ungenutzteren Abflugzeiten abfliegen [4]. Pilon [28] zeigte verschiedene Ansätze bezüglich Flugpriorisierungen anhand anderer Arbeiten auf: Flüge können basierend auf Punkten priorisiert werden, die anhand geflogener Distanzen und anderen Faktoren berechnet werden. Ebenso können sogenannte „Credits“ in einem System mit menschlichen Aktoren verwendet werden oder anhand Priorisierungen alternative Routen zugewiesen werden. Ebenso lassen sich die Ansätze in Kombination verwenden.

So existiert das System UDPP, User Driven Prioritization Process (benutzergesteuerter Priorisierungsprozess). Dieses versucht, dass Aktionen einer Fluglinie keine Flüge anderer Fluglinien beeinflussen, als Haupteinschränkung umzusetzen. UDPP erlaubt es Fluglinien auftretende Verspätungen innerhalb der eigenen Abflüge zu verteilen. Ein Tausch zwischen zwei Abflugzeiten ist nur innerhalb einer Fluglinie möglich. So können wirtschaftlich wichtige Flüge priorisiert pünktlicher abfliegen [27, 28]. Eine detaillierte Erklärung von UDPP kann in der Arbeit von Pilon [28] nachgelesen werden.

Andere Systeme und Möglichkeiten den Flugverkehr effizienter zu gestalten, wurden von Hafidi [19] untersucht. Es wurden verschiedene Bereiche voneinander abgetrennt. Zuerst wurden Systeme und Forschungen für Flugverkehrskontrolle basierend von Flugplätzen aufgezeigt. Es wird nur eine Auswahl beschrieben: So wurden in vergangener Zeit Systeme entwickelt, um die Planung des Rollzeitraums zu optimieren anhand von drei Algorithmen. So konnten Verspätungen am Rollfeld von 20% auf 2% verringert werden, wodurch auch der Spritverbrauch während dem Rollen verringert werden konnte [33]. Andere von Hafidi [19] untersuchte Systeme arbeiten mit Prognosemodellen oder anderen Modellierungsmethoden. Ein anderer Ansatz ist die Einführung von „Remote Tower Center“ (ferngesteuerter Flugverkehrskontrollturm). Hier wird es möglich, Aufgaben für den jeweiligen Flughafen aus einer entfernten Flugverkehrskontrollstelle wahrzunehmen und so die Kapazitäten der Fluglotsen und Flugleiter besser auszunutzen [20].

Zusätzlich dazu untersuchte Hafidi [19] Forschungen im Bereich der „Terminal Control Area“, dem Luftraum über dem Flughafen. In diesem Rahmen werden in der Arbeit

von Hafidi mehrere Optimierungsstrategien vorgestellt, die sich mit der Nutzung von Multilevel-Optimierung, automatisierter Planung oder der Nutzung eines optimierungsbasierten Entscheidungsunterstützungssystem beschäftigen. Ebenso wurde untersucht, wie Fluglotsen Flüge während ihrem Flug möglichst optimal unterstützen können.

3 Architektur

Der entstandene Prototyp befasst sich mit der Verwirklichung eines heuristischen Optimierers, der zurzeit die vertrauenswürdigen Daten der Fluglinien nicht schützt, weil die Gewichtstabellen direkt im Optimierer vorhanden sind und Fitness Berechnungen dort ebenso durchgeführt werden. Die einzelnen Bestandteile des Prototyps sind in Abbildung 3.1 ersichtlich. Excel I/O und Heuristic Optimizer sind voneinander unabhängig ausführbar.

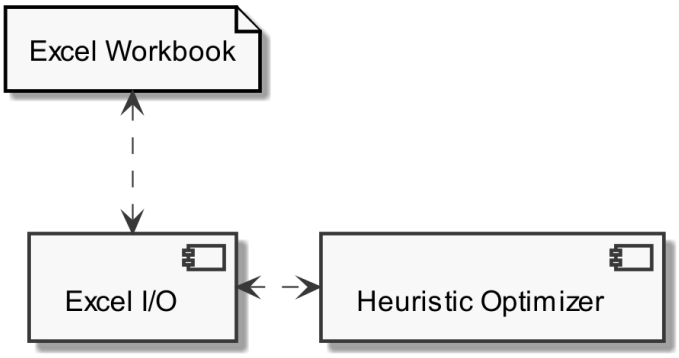

Abbildung 3.1: Gesamtarchitektur

Excel Workbook Das Excel Workbook beinhaltet die Grundinformationen zu den Flügen und zur Optimierung. Ebenso können dort die Optimierungsergebnisse mit angezeigt werden. Es handelt sich hier um einfache Exceldokumente, die von Excel I/O gelesen und geschrieben werden können.

Excel I/O Das Kommandozeilentool Excel I/O ermöglicht die Konvertierung zwischen den Daten im Exceldateiformat und den Daten im JSON-Format, welches der heuristische Optimierer benötigt. Excel I/O liest die Grundinformationen zu Flügen und

Optimierungen aus dem Excel Workbook. Anhand dieser Informationen werden die Optimierungssessions-Daten erzeugt. Diese können daraufhin vom heuristischen Optimierer verwendet werden. Excel I/O kann ebenso die generierten Gewichtstabellen in seperate Arbeitsblätter eines Exceldokuments schreiben. Nachdem die empfohlene Flugreihenfolge bekannt ist, kann diese auch in ein vorbereitetes Exceldokument geschrieben werden.

Heuristic Optimizer Im Kernstück des Prototyps werden die Optimierungssessions-Daten verwendet, um die Abflugreihenfolgen den Prioritäten nach zu optimieren. Der heuristische Optimierer (Heuristic Optimizer) benötigt die Daten für die Optimierungssessions, die vom Excel I/O generiert werden. Die gefundene optimale Lösung kann wie beschrieben von Excel I/O verwendet werden.

4 Datenverarbeitung

In diesen Abschnitten wird die Datenverarbeitung behandelt. Daher werden zuerst Flüge, Optimierungssessions und Margins definiert. Ein Flug hat Informationen zur Identifikation, eine geplante Abflugzeit, eine gewünschte Zeit, bevor er nicht abfliegen will, eine gewünschte Zeit, zu der er abfliegen will, eine gewünschte Zeit, nach der er nicht abfliegen will, eine Priorität und eine zugewiesene Abflugzeit. Die zugewiesene Abflugzeit muss nicht bekannt sein. Eine Optimierungssession hat eine definierte Anzahl an Flügen, eine Start- und Endzeit, eine Identifikations-Nummer und andere Faktoren zur Berechnung der optimalen Abflug-Slots und der Gewichtstabellen, wie sie in den folgenden Abschnitten bestimmt werden. Margins betreffen die gewünschte Zeit, bevor er nicht abfliegen will, eine gewünschte Zeit, zu der er abfliegen will und eine gewünschte Zeit, nach der er nicht abfliegen will. Daher sind die Margin-Werte die folgenden drei Werte: `TimeNotBefore`, `TimeWished` und `TimeNotAfter`. Margins beeinflussen maßgeblich die zu generierende Gewichtstabelle und daher die zugewiesene Abflugzeit. Ein Flug, der innerhalb der Margin-Werte ist, beschreibt einen Flug, der nicht vor `TimeNotBefore` und nicht nach `TimeNotAfter` eine zugewiesene Abflugzeit hat.

4.1 Excel-Workbook

Im Excel-Workbook werden die Grundinformationen zu den Flügen und den Optimierungssessions gespeichert, um sie mittels Excel I/O konvertieren zu können. Es werden zwei Excel Dokumente verwendet. Das Excel-Quelldokument beinhaltet zwei Tabellenblätter: `Flights` und `Optimizations`.

In `Flights` werden die Informationen zu Flügen gespeichert und in `Optimizations` die Informationen zu den Optimierungssessions. Tabelle 4.1 zeigt das Blatt `Flights`, welches die Spalten `FlightId` (abgekürzt `FID`), `ScheduledTime` (abgekürzt `ST`), `TimeNotBefore`

(abgekürzt TNB), TimeWished (abgekürzt TW), TimeNotAfter (abgekürzt TNA), priority (abgekürzt prio) und AssignedSlot (abgekürzt AS) beinhaltet. Die Spalten ScheduledTime, TimeNotBefore, TimeWished, TimeNotAfter beinhalten auch eine Datumsangabe (beispielsweise 2021-06-10 12:15:00). Das Datum wird in der Tabelle jedoch nicht angezeigt, um sie kompakt zu halten. Es werden Beispieldaten gezeigt, die auch für die Findung der Parameter 6 verwendet werden.

FID	ST	TNB	TW	TNA	prio	AS
F01	03:00:00	11:55:00	12:15:00	12:26:00	2,11	
F02	03:03:00	12:07:00	12:18:00	12:28:00	1,80	
F03	03:06:00	12:11:00	12:24:00	12:36:00	0,86	
F04	03:09:00	12:28:00	12:30:00	12:36:00	2,18	
F05	03:12:00	12:21:00	12:33:00	12:46:00	0,77	
F06	03:15:00	12:26:00	12:36:00	12:51:00	0,83	
F07	03:18:00	12:24:00	12:39:00	12:45:00	1,79	
F08	03:21:00	12:40:00	12:42:00	12:46:00	2,45	
...	...	...	...	...	...	

Tabelle 4.1: Excel-Workbook Tabelle (Excel-Quelldokument, Tabellenblatt Flights)

Das Tabellenblatt Flights (siehe Tabelle 4.1) hat folgende Spalten:

- **FlightId**: Die Flugidentifikation, in diesem Fall bestehend aus F und einer Zahl. Ein möglicher Wert ist „F01“.
- **ScheduledTime**: Die ursprünglich geplante Abflugzeit. Der Flug kann nicht vor dieser Zeit abfliegen. Beinhaltet ein Datum und eine Uhrzeit. Möglich als Wert ist beispielsweise „2021-06-10 03:00:00“.
- **TimeNotBefore**: Einer der drei Margin-Werte. Der Flug soll nicht vor diesem Zeitpunkt abfliegen. Beinhaltet ein Datum und eine Uhrzeit. Ein möglicher Wert ist „2021-06-10 11:55:00“.
- **TimeWished**: Ebenso einer der Margin-Werte. Der Flug soll, wenn möglich, zu diesem Zeitpunkt abfliegen oder möglichst nahe an diesem Zeitpunkt. Beinhaltet ein Datum und eine Uhrzeit, wie der beispielhafte Wert: „2021-06-10 12:15:00“.

- **TimeNotAfter**: Der letzte der Margin-Werte. Der Flug soll nicht nach diesem Zeitpunkt abfliegen. Beinhaltet ein Datum und eine Uhrzeit. Ein möglicher Wert ist „2021-06-10 12:26:00“.
- **priority**: Die Priorität des Fluges. Ein höherer Wert zeigt einen Flug mit höherer Priorität an im Vergleich zu einem Flug mit niedrigerer Priorität. Einer der möglichen Werte ist beispielweise „2,11“.
- **AssignedSlot**: Diese Spalte ist im Quelldokument leer und kann später mit den zugewiesenen Abflugzeiten pro Flug ausgefüllt werden.

Zu beachten sind die Margins. Diese sind schwache Einschränkungen, die eingehalten werden sollen, aber nicht in jedem Fall. Falls eine Optimierung einen besseren Fitnesswert hat mit einigen Flügen, die vor `TimeNotBefore` fliegen ist diese Lösung dennoch besser als eine, wo alle Flüge innerhalb der Margin-Werte abfliegen. Außerdem werden diese Werte von den Fluglinien vorgegeben und können bei kurzen Abständen zwischen `TimeNotBefore` und `TimeNotAfter` nicht für jeden Flug eingehalten werden, wenn alle zu einer ähnlichen Zeit abfliegen wollen, da es nur eine begrenzte Anzahl an Slots zwischen den Zeitpunkten gibt.

Das Tabellenblatt `Optimizations` wird in Tabelle 4.2 dargestellt, wobei eine oder mehrere Optimierungssessions eintragbar sind. Die Spalte `OptId` (beispielsweise `063e75cf-1e46-4dcb-b204-d833bcd289ab`) wird auf „063“ abgekürzt und die Spalten `StartTime` und `EndTime` beinhalten auch ein Datum (beispielsweise `2021-06-10 12:15:00`). Für die Übersichtlichkeit werden diese ebenso abgekürzt. Spaltennamen `StartTime` und `EndTime` werden auf `StartT` und `EndT` abgekürzt. Der Spaltennamen `Interval [sec]` wird auf `Int` abgekürzt. Spaltennamen `minValue`, `maxValue` und `dropValue` werden auf `maxV`, `minV` und `dropV` abgekürzt.

OptId	StartT	EndT	Int	Framework	minV	maxV	dropV
063…	12:15:00	17:00:00	180	OPTAPLANNER	−10 000	10 000	6000
…	…	…	…	…	…	…	…

Tabelle 4.2: Excel-Workbook Tabelle (Excel-Quelldokument, Tabellenblatt `Optimizations`)

Im Tabellenblatt `Optimization` (siehe Tabelle 4.2) sind die Daten zu den Optimierungssessions gespeichert. Es gibt folgende Spalten:

- **OptId**: Die Identifikation der Optimierungssessions. Beinhaltet eine UUID (Universally Unique Identifier), die wie folgt lauten kann: „063e75cf-1e46-4dcb-b204-d833bcd289ab“.
- **StartTime**: Der Zeitpunkt, an dem der erste zuweisbare Slot für die Optimierungssession ist. Beinhaltet ein Datum und eine Uhrzeit, wie beispielsweise „2021-06-10 12:15:00“.
- **EndTime**: Der Zeitpunkt, an dem der letzte zuweisbare Slot für die Optimierungssession ist. Beinhaltet ein Datum und eine Uhrzeit. Ein möglicher Wert ist „2021-06-10 17:00:00“.
- **Interval [sec]**: Gibt an, wie lange der Zeitraum zwischen zwei Slots in Sekunden ist. Möglich sind daher Zahlen wie „180“.
- **Framework**: Hier wird das Framework angegeben, welches in dieser Optimierungssession verwendet wird. Man kann „JENETICS“, „OPTAPLANNER“ oder „HUNGARIAN“ auswählen.
- **minValue**: Gibt an, welches der kleinste mögliche Wert für die zu generierende Gewichtstabelle der Optimierungssession ist. Möglich ist beispielsweise „-10000“.
- **maxValue**: Gibt an, welches der größte mögliche Wert für die zu generierende Gewichtstabelle der Optimierungssession ist. Dieser Wert kann beispielsweise „10000“ sein.
- **dropValue**: Gibt an, um wie viel das Gewicht bei `TimeNotAfter` und `TimeNotBefore` fällt, beziehungsweise steigt. Ein möglicher Wert kann „6000“ sein.

Das Excel-Quelldokument stellt im Prototyp die graphische Oberfläche dar, weil sie leicht bedienbar ist, mittels Java Bibliotheken ausgelesen und mit den üblichen Tabellenbearbeitungsprogrammen geöffnet werden kann. Daher wird keine graphische Oberfläche implementiert.

Im weiteren Verlauf werden Excel-Quelldokument und Excel-Zieldokument unterschieden. Ein Excel-Quelldokument wird so aufgebaut, wie in den vorigen Absätzen beschrieben wurde. Ein Excel-Zieldokument beinhaltet im Gegensatz zum Quelldokument die Informationen zu den Optimierungen (Optimierungssessions-Identifikation, Optimierungsframework, Startzeit, Endzeit, Intervalldauer, Mindestwert, Maximalwert, Fallhöhe) direkt

im Tabellenblatt „Flights“. Zusätzlich dazu gibt es pro Flug ein eigenes Tabellenblatt, wo die Gewichtstabelle gespeichert ist. Diese Tabellenblätter haben zwei Spalten: „Slot“ und „Weight“. In der Spalte „Slot“ wird die Zeit des Slots angegeben und in der anderen Spalte das Gewicht, was der jeweilige Flug für die bestimmte Spalte definiert hat. Mögliche Werte wurden in Tabelle 4.3 abgebildet.

Slot	**Weight**
10.06.2021 12:15:00	-7725
10.06.2021 12:18:00	-3105
10.06.2021 12:21:00	1347
...	...

Tabelle 4.3: Excel Workbook Tabelle (Excel-Zieldokument, Tabellenblatt `F05`)

4.2 Excel I/O

Excel I/O ist ein Kommandozeilentool, welches erstellt wurde, um mittels Kommandozeilenargument die Daten vom Excel-Workbook in ein für den heuristischen Optimierer lesbares Format zu konvertieren. Je nach benutztem Befehl liest Excel I/O zum Beispiel das Excel-Quelldokument aus oder konvertiert die Daten in ein passendes JSON-Dateiformat. Excel I/O unterstützt eine Vielzahl an Befehlen, wobei die wichtigsten `cmtj`, `cmwetj`, `wr` und `tdg` sind. Diese werden später noch genauer beschrieben, wobei `tdg` in einem eigenen Abschnitt beim Testdatengenerator beschrieben wird.

Das Tool wurde als Java Projekt implementiert, mit eigenen Klassen, die die Logik zum Lesen und Schreiben der verwendeten JSON- und Excel-Dateien beinhalten. Außerdem wurden Klassen angelegt mit den Attributen der einzelnen Objekte, mit denen Excel I/O arbeitet, wie die Flüge, Slots und andere Objekte. Zur Generierung der Gewichtstabellen und der einzelnen Flüge wurde ebenso eine eigene Klasse für die Logik angelegt. Der Code ist über ein GitHub-Repository[1] verfügbar.

[1] https://github.com/jku-win-dke/mt2106-slma-excelio

4.2.1 JSON-Konfigurationen erzeugen

Der Befehl `cmtj` erzeugt eine JSON-Datei für den heuristischen Optimierer, worin die Gewichtstabelle anhand der Excel-Quelldatei generiert wurde. Es benötigt als erstes Argument des Kommandozeilenbefehls den Dateiort der Excel-Quelldatei und als zweites Argument den Dateiort der zu erstellenden JSON-Datei. Falls das zweite Argument nicht angegeben wird, werden die Informationen im JSON Format direkt auf die Konsole ausgegeben. Der Befehl kann wie folgt genutzt werden:

```
1        -cmtj flights.xlsx optSession.json
```

Code 4.1: Startbefehl für Optimierer

Hierbei ist `flights.xlsx` eine Excel-Quelldatei. Die generierte Datei beinhaltet `optSession.json`, jedoch wird auch die Optimierungs-Identifikation im Dateinamen inkludiert.

Der Befehl `cmwetj` ist ähnlich zum Befehl `cmtj`. Jedoch wird zusätzlich eine Excel-Zieldatei generiert, die mit dem `wr`-Befehl für das Schreiben der Ergebnisse in eine Exceldatei verwendet werden kann. Der Befehl kann wie folgt genutzt werden:

```
1        -cmwetj flights.xlsx optSession.json
```

Code 4.2: Startbefehl für Optimierer

Ähnlich zu `cmtj` ist `flights.xlsx` eine Excel-Quelldatei und die generierte Datei beinhaltet `optSession.json`. Die Quelldatei und Zieldatei können beide zur Mitanzeige von Ergebnissen verwendet werden. Mit Excel I/O selbst ist der Befehl `wr` vorgesehen.

Da beide Befehle sehr ähnlich funktionieren, wird im Folgenden nur `cmwetj` genauer beschrieben. Nachdem der Befehl gestartet ist, wird zuerst die Excel-Quelldatei gelesen und die Margins und die Optimierungsssessions-Informationen gespeichert. Dazu zählen die Flug-Identifikations-Nummer (`FlightId`), die ursprünglich geplante Abflugzeit (`ScheduledTime`), `TimeNotBefore`, `TimeWished`, `TimeNotAfter`, die Flugpriorität (`priority`), Optimierungssessions-Identifikations-Nummer (`optId`), Start- und Endzeit der Optimierungssession (`StartTime`, `EndTime`), Intervalllänge pro Slot (`Interval`), Optimierungsframework (`Framework`). Zusätzlich wird der minimale und maximale Wert für die zu generierende Gewichtstabelle (`minValue`, `maxValue`) angegeben. Dazu gibt es noch

die Angabe der Fallhöhe (`dropValue`). Es werden alle angegebenen Flüge und Optimierungssessions verarbeitet.

Danach werden pro Optimierungssession die Flüge anhand gegebener Informationen inklusive der Gewichtstabelle erstellt. Für die Gewichtstabellen sind die Margins (`TimeNotBefore`, `TimeWished`, `TimeNotAfter`), die ursprüngliche Abflugzeit (`ScheduledTime`) und die Flugpriorität (`priority`) relevant. Zusätzlich ist der Beginn der Slots (`SlotBegin`) auf den gleichen Wert wie `StartTime` zu setzen und `SlotEnd` auf `EndTime` für die Ende der Slots in Bezug auf die Abbildung. Wie in Abbildung 4.1 ersichtlich ist, werden die Gewichte (`w`) anhand der gegebenen Zeit (`t`) jeweils anders berechnet. In der referenzierten Abbildung wird `minValue` auf $-10\,000$, `dropValue` auf 6000 und `maxValue` auf $10\,000$ gesetzt. Folgend werden die Gewichte `w` für die möglichen Zeiten `t` dargestellt.

Abbildung 4.1: Gewichtstabelle

Alle Parameter von Zeiten, wie `ScheduledTime` sind mindestens null, selbst wenn sie vor `StartTime` sind. Die Formeln zur Errechnung der aktuellen Gewichte `w` sind wie folgt für

- Slots zwischen `StartTime` (inklusiv) und `ScheduledTime` (exklusiv):

$$w(t) = minValue$$

- Slots zwischen `ScheduledTime` (inklusiv) und `TimeNotBefore` (exklusiv):

$$k = \frac{(0 - minValue)}{(TimeNotBefore - ScheduledTime)}$$

$$w(t) = (k * t + minValue - k) * priority$$

- Slots zwischen `TimeNotBefore` (inklusiv) und `TimeWished` (exklusiv):

$$k = \frac{(maxValue - dropValue)}{(TimeWished - TimeNotBefore)}$$

$$w(t) = (k * t + dropValue - k * TimeNotBefore) * priority$$

- Slots zwischen `TimeWished` (inklusiv) und `TimeNotAfter` (inklusiv):

$$k = \frac{(dropValue - maxValue)}{(TimeNotAfter - TimeWished)}$$

$$w(t) = (k * t + maxValue - k * TimeWished) * priority$$

- Slots zwischen `TimeNotAfter` (exklusiv) und `EndTime` (inklusiv):

$$k = \frac{(minValue - 0)}{(SlotEnd - TimeNotAfter)}$$

$$w(t) = (k * t + minValue - k * EndTime) * priority$$

Wie ersichtlich ist, werden die Gewichte anhand gleichmäßiger Funktionen zwischen zwei definierten Punkten berechnet. Das Gewicht bei Slot `ScheduledTime`, `StartTime` und `EndTime` ist 0. Bei Slot `TimeNotBefore` und `TimeNotAfter` beträgt das Gewicht den Wert der Fallhöhe (`dropValue`). Beim Slot `TimeWished` beträgt das Gewicht den maximalen Wert von `maxValue`. All diese Werte werden anschließend mit der definierten Priorität multipliziert.

Die Gewichtstabelle bestimmt später für die Optimierungen den Fitnesswert, da dieser als Summe der Gewichte der Flüge zu den zugewiesenen Abflugzeiten berechnet wird (siehe Abschnitt 5.2). Nachdem die Gewichtstabellen generiert wurden, werden diese in die entsprechenden Flüge übertragen und eine Liste an Flügen wird in eine Exceldatei

geschrieben, sofern eine Excel-Zieldatei angegeben wurde. Wenn eine JSON-Datei erzeugt werden soll, wird diese anhand der generierten Informationen ebenso generiert und gespeichert. Die Excel-Zieldatei sieht so aus wie sie unter Abschnitt 4.1 beschrieben wurde. Die generierte JSON-Datei (Code 7.1) beinhaltet folgende Informationen:

- `optId`: Identifikation der Optimierungssession. Beinhaltet eine UUID (Universally Unique Identifier).
- `initialFlightSequence`: Informationen zu einer vorgeschlagenen initialen Flugsequenz.
- `flights`: Beinhalten Informationen zu den Flügen, wobei pro Flug die Daten zu `flightId` (Flug-Identifikation), `scheduledTime` (ursprünglich geplante Abflugzeit) und die `weightMap` (Gewichtstabelle) gespeichert werden.
- `slots`: Zusätzlich dazu werden die Zeitpunkte der verfügbaren Slots gespeichert.
- `optimizationFramework`: Gibt das zu verwendende Framework an (Hungarian Algorithm, OptaPlanner oder Jenetics).
- `margins`: Außerdem werden die Margins (`timeNotBefore`, `timeWished`, `timeNotAfter`) inklusive der `scheduledTime` (ursprünglich geplante Abflugzeit) und die `flightId` (Flug-Identifikation) gespeichert für eine verbesserte Auswertung im Anschluss an die Optimierung.
- `parameters`: Hier werden die nötigen weiteren Parameter für die Optimierung gespeichert, die für OptaPlanner und Jenetics möglich sind.

```
1          {
2            "optId" : "1fe567d4-1f5f-47d4-af00-4a3004ee626f",
3            "initialFlightSequence" : [ "F001", "F002", "F003", "F004", "F005", "F006",
                 "F007", "F008", "F009", "F010", "F011", "F012", "F013", "F014", "F015",
                 "F016", "F017", "F018", "F019", "F020", "F021", "F022", "F023", "F024",
                 "F025", "F026", "F027", "F028", "F029", "F030", "F031", "F032", "F033",
                 "F034", "F035", "F036", "F037", "F038", "F039", "F040", "F041", "F042",
                 "F043", "F044", "F045", "F046", "F047", "F048", "F049", "F050", "F051",
                 "F052", "F053", "F054", "F055", "F056", "F057", "F058", "F059", "F060",
                 "F061", "F062", "F063", "F064", "F065", "F066", "F067", "F068", "F069",
                 "F070", "F071", "F072", "F073", "F074", "F075", "F076", "F077", "F078",
                 "F079", "F080", "F081", "F082", "F083", "F084", "F085", "F086", "F087",
                 "F088", "F089", "F090", "F091", "F092", "F093", "F094", "F095", "F096",
                 "F097", "F098", "F099", "F100" ],
4            "flights" : [ {
```

```
      "flightId" : "F1",
      "scheduledTime" : 1626791743.808514700,
      "weightMap" : [ 10474, 10138, 9803, 9468, 9133, 8798, 8463, 8127, 7792,
          7457, 7122, 6787, 6452, -60, -181, -302, -423, -544, -665, -787, -908,
           -1029, -1150, -1271, -1392, -1513, -1634, -1755, -1876, -1997, -2119,
           -2240, -2361, -2482, -2603, -2724, -2845, -2966, -3087, -3208, -3329,
           -3450, -3572, -3693, -3814, -3935, -4056, -4177, -4298, -4419, -4540,
           -4661, -4782, -4904, -5025, -5146, -5267, -5388, -5509, -5630, -5751,
           -5872, -5993, -6114, -6236, -6357, -6478, -6599, -6720, -6841, -6962,
           -7083, -7204, -7325, -7446, -7567, -7689, -7810, -7931, -8052, -8173,
           -8294, -8415, -8536, -8657, -8778, -8899, -9021, -9142, -9263, -9384,
           -9505, -9626, -9747, -9868, -9989, -10110, -10231, -10352, -10474 ]
    }, {
      "flightId" : "F2",
      "scheduledTime" : 1626791743.808514700,
      "weightMap" : [ 4972, 8287, 8022, 7757, 7492, 7227, 6961, 6696, 6431,
          6166, 5901, 5635, 5370, 5105, -48, -145, -242, -339, -436, -533, -630,
           -727, -823, -920, -1017, -1114, -1211, -1308, -1405, -1502, -1599,
          -1696, -1793, -1890, -1987, -2084, -2181, -2277, -2374, -2471, -2568,
          -2665, -2762, -2859, -2956, -3053, -3150, -3247, -3344, -3441, -3538,
          -3635, -3732, -3828, -3925, -4022, -4119, -4216, -4313, -4410, -4507,
          -4604, -4701, -4798, -4895, -4992, -5089, -5186, -5282, -5379, -5476,
          -5573, -5670, -5767, -5864, -5961, -6058, -6155, -6252, -6349, -6446,
          -6543, -6640, -6736, -6833, -6930, -7027, -7124, -7221, -7318, -7415,
          -7512, -7609, -7706, -7803, -7900, -7997, -8094, -8191, -8287 ]
    }, {
    [...]
    }, {
      "flightId" : "F100",
      "scheduledTime" : 1626791743.808514700,
      "weightMap" : [ -8744, -8644, -8543, -8442, -8340, -8239, -8138, -8037,
          -7936, -7835, -7734, -7633, -7532, -7431, -7330, -7228, -7127, -7026,
          -6925, -6824, -6723, -6622, -6521, -6420, -6319, -6218, -6116, -6015,
          -5914, -5813, -5712, -5611, -5510, -5409, -5308, -5207, -5106, -5004,
          -4903, -4802, -4701, -4600, -4499, -4398, -4297, -4196, -4095, -3994,
          -3892, -3791, -3690, -3589, -3488, -3387, -3286, -3185, -3084, -2983,
          -2881, -2780, -2679, -2578, -2477, -2376, -2275, -2174, -2073, -1972,
          -1871, -1769, -1668, -1567, -1466, -1365, -1264, -1163, -1062, -961,
          -860, -759, -657, -556, -455, -354, -253, -152, -51, 5386, 5666, 5946,
           6226, 6505, 6785, 7065, 7345, 7625, 7905, 8184, 8464, 8744 ]
    } ],
    "slots" : [ {
      "time" : 1626791743.808514700
    }, {
      "time" : 1626791863.808514700
    }, {
    [...]
    }, {
      "time" : 1626803623.808514700
```

```
27              } ],
28              "optimizationFramework" : "JENETICS",
29              "margins" : [ {
30                "flightId" : "F1",
31                "scheduledTime" : 1626791743.808514700,
32                "timeNotBefore" : 1626790243.808514700,
33                "timeWished" : 1626791743.808514700,
34                "timeNotAfter" : 1626793243.808514700
35              }, {
36                "flightId" : "F2",
37                "scheduledTime" : 1626791743.808514700,
38                "timeNotBefore" : 1626790363.808514700,
39                "timeWished" : 1626791863.808514700,
40                "timeNotAfter" : 1626793363.808514700
41              }, {
42              [...]
43              }, {
44                "flightId" : "F100",
45                "scheduledTime" : 1626791743.808514700,
46                "timeNotBefore" : 1626802123.808514700,
47                "timeWished" : 1626803623.808514700,
48                "timeNotAfter" : 1626805123.808514700
49              } ],
50              "parameters" : {
51                "offspringFraction" : 0.7,
52                "crossover" : "PARTIALLY_MATCHED_CROSSOVER",
53                "maximalPhenotypeAge" : 80,
54                "crossoverAlterProbability" : 0.9,
55                "survivorsSelectorParameter" : 50,
56                "survivorsSelector" : "TOURNAMENT_SELECTOR",
57                "populationSize" : 500,
58                "mutator" : "SWAP_MUTATOR",
59                "terminationConditions" : {
60                  "BY_EXECUTION_TIME" : 10
61                },
62                "offspringSelectorParameter" : 50,
63                "offspringSelector" : "TOURNAMENT_SELECTOR",
64                "mutatorAlterProbability" : 0.15
65              }
66              }
```

Code 4.3: Auszug aus generierter JSON-Datei

4.2.2 Excel-Ergebnisse schreiben

Der Befehl `wr` zeigt Excel I/O an, dass es die gegebenen Ergebnisse einer JSON-Datei in eine bestimmte Excel-Zieldatei schreiben soll. Beispielsweise kann es mit dem folgenden Befehl ausgeführt werden:

```
1          -wr flights.xlsx flightSequenceResult.json flights-e48aad2f-a746-4787-9f20-493
               a2286d900.xlsx
```

Code 4.4: Startbefehl für Optimierer

Die Excel-Quelldatei `flights.xlsx` kann weggelassen werden. Die Ergebnisse als JSON-Datei (`flightSequenceResult.json`) können in der Form verwendet werden, wie sie der heuristische Optimierer zurückgibt. Die Excel-Zieldatei (`flights-e48aad2f-a746-4787-9f20-493a2286d900.xlsx`) wird benötigt, um die Ergebnisse in diese schreiben zu können. Das Format der Excel-Zieldatei wurde bereits in einem früheren Abschnitt beschrieben. Es werden in der Zieldatei die leeren Felder der Spalte „Assigned Slot“ mit den tatsächlich zugewiesenen Slots befüllt.

4.3 Testdatengenerator

Für den Befehl `tdg` gibt es verschiedene Einstellungsmöglichkeiten, die von der mitgegebenen JSON-Einstellungsdatei abhängen. Es kann auch das gewählte Framework eingestellt werden. Dies wird jedoch aufgrund der aktuellen Einstellungen und der Notwendigkeit, dass Tests von allen drei Frameworks (Jenetics, OptaPlanner und Hungarian Algorithm) getestet werden sollen, ignoriert. In Einstellungsdatei (siehe Code 4.5) sind folgende Attribute änderbar:

- Fluganzahl (`flightCount`) und Slotanzahl (`slotCount`): Hierbei wird darauf geachtet, dass es gleiche viele Flüge wie Slots gibt. Es sind alle positiven Zahlen möglich.
- Flugkürzel (`flightPrefix`): Hier kann das Flügkürzel eingestellt werden, welches vor den Flugnummern angezeigt wird, wie „F“ für „F14“.
- Startzeit (`slotStartTime`): Hier kann der Startzeitpunkt der Slots eingestellt werden.

- Slotzeit (`slotLengthSec`): Es kann ebenso eingestellt werden, wie lange ein Slot dauert.
- Margin-Breite (`marginWindowLength`): Die zeitliche Entfernung zwischen `TimeNotBefore` und `TimeNotAfter` in Sekunden kann eingestellt werden, wo `TimeWished` in der Mitte ist. Mögliche Werte sind beispielsweise 300 oder 600 Sekunden.
- Flugverteilung (`distributionSetting`): Hierzu kann zwischen verschiedenen Einstellungen, wie die Flüge innerhalb der gegebenen Slotanzahl verteilt sind, ausgewählt werden. Es gibt mehrere Möglichkeiten. Der Wert 0 entspricht der gleichverteilten Einstellung, wo pro Slot ein Flug bezüglich `TimeWished` zugeteilt ist. Der Wert 1 wird für eine extreme Häufung als Einstellung verwendet. Die meisten Flüge haben ein `TimeWished` in der Mitte der möglichen Slots und die Randslots haben oft keine Flüge in Bezug auf `TimeWished`. Der Wert 2 ergibt eine Flugverteilung, wo die meisten Flüge in Bezug auf `TimeWished` am Rand sind. Kaum Flüge haben hier ihre `TimeWished` in der Mitte der möglichen Slots. Der Wert 3 ergibt eine normalere Verteilung, wobei hier weiterhin ein großer Teil in der Mitte abfliegen will und nur einige Flüge am Rand bezüglich Slots abfliegen will. Der Wert 4, normale Häufung, weist zwar auch mehr Flüge in der Mitte bezüglich `TimeWished` zu, jedoch weniger als die meisten anderen Einstellungen. Daher werden auch am Rand einige Flüge zugewiesen. Einstellungen 1 und 4 wurden auf 100 Flüge und Slots angepasst. Die Abbildungen zu Einstellung 0, 1 und 4 sind in Abschnitt 7.2.1 als Abbildungen 7.1, 7.2 und 7.3 zu finden.
- Prioritätseinstellungen (`prioritySettings`): Für diesen Parameter kann folgendes eingestellt werden: Welche Bereiche bei den Slots unterschiedliche Prioritäten zwischen zwei Randwerten zufällig verteilt haben. Hierfür wird ein Array verwendet. Für diese Einstellung werden die Slots in Perzentile aufgeteilt. Das erste Perzentil ist am Anfang der Slots. Es folgt folgendem Format: „[[a,b,x,y],...]“, wobei `a` für das erste Perzentil steht, wo die aktuellen Prioritätseinstellungen gelten und `b` das erste Perzentil ist, wo die aktuellen Einstellungen nicht mehr gelten. Der Wert `x` gibt den möglichen untersten Prioritätswert an und `y` den höchstmöglichen. Die Priorität für jeden Flug innerhalb der angegebenen Perzentile entspricht einer zufällig gewählten Priorität zwischen `x` und `y`. Eine mögliche Einstellung ist „[[0,20,0.8,1.2],[20,80,1.0,1.0],[80,100,2.0,2.0]]“, wobei hier die Priorität der

ersten zwanzig Perzentile zwischen 0,8 und 1,2 liegt, der nächsten sechzig Perzentile genau auf 1 und der letzten zwanzig Perzentile auf 2 ist.

- Mindestwert (`minValue`): Der minimale Wert für Gewichte bei einer Priorität von eins. Es wird empfohlen, dass dieser Wert negativ ist.
- Maximalwert (`maxValue`): Der maximale Wert für Gewichte bei einer Priorität von eins. Es wird empfohlen, dass dieser Wert positiv und größer als der Änderungswert ist.
- Fallhöhe (`dropValue`): Die Fallhöhe für Gewichte bei einer Priorität von eins. Dies ist der Sprung an Gewichtswert vor `TimeNotBefore` und nach `TimeNotAfter`. Es wird empfohlen, dass dieser Wert kleiner als der Maximalwert ist.
- maximale Zeit (`maxTime`): Die maximale Zeit, die die Algorithmen für das Ergebnis brauchen dürfen. Beeinflusst nicht die Berechnungsdauer bei Nutzung des Ungarischen Methode.

```
{
  "flightCount": 100,
  "flightPrefix": "F",
  "slotCount": 100,
  "slotLengthSec": 120,
  "marginWindowLength": 3000,
  "distributionSetting": 1,
  "optimizationFramework": "OPTAPLANNER",
  "prioritySettings": [
    [0,35,0.8,1.1],[35,65,2.0,3.0],[65,100,0.8,1.1]
  ],
  "minValue": -10000,
  "maxValue": 10000,
  "dropValue": 6000,
  "maxTime": 10
}
```

Code 4.5: Einstellungsdatei für Testdatengenerator (JSON)

Anhand dieser Einstellungen werden die Testdaten bezüglich Flüge generiert. Danach werden für alle vorhandenen Algorithmus-Einstellungen separate Optimierungs-Identifikations-Nummern erstellt und Dateien erzeugt. Laut dem Feintuning werden pro Einstellung 88 Konfigurationen erschaffen. Davon werden 16 verschiedene Jenetics-Einstellungen je fünfmal hergestellt, was 80 Konfigurationen sind. Zusätzlich werden 7 OptaPlanner-Einstellungen als Konfigurationen erstellt und eine Konfiguration für die Ungarische

Methode. Der Testplan wurde anhand der ersten Testdurchläufe mitsamt möglichen Einstellungen für die Frameworks bestimmt und ist unter Abschnitt 7.3 zu finden. Dort werden auch die tatsächlich umgesetzten Einstellungen im Detail erläutert.

5 Heuristischer Optimierer

Das Kernstück des Prototyps ist der Heuristische Optimierer. Er nimmt die Daten für zukünftige Optimierungssessions an, optimiert die Abflugreihenfolge und gibt diese zurück. Zur Nutzung wird ein Springboot-Server genutzt, auf welchen über REST-Zugriffe möglich sind. Änderungen an den Konfigurationen der benutzten Algorithmen sind während dem Betrieb pro Optimierungssession möglich. Für die Optimierungen werden folgende Algorithmentypen unterstützt: exakter Algorithmus (Hungarian Algorithm), Metaheuristiken und Konstruktionsheuristiken (OptaPlanner) und zusätzlich dazu noch der genetische Algorithmus (Jenetics). Der heuristische Optimierer setzt zusätzlich für die Hauptstudie die Ungarische Methode, auch Hungarian Algorithm genannt, um. Der Code ist über ein GitHub-Repository[1] verfügbar.

5.1 REST-Server

Die Nutzung eines REST-Servers ermöglicht die einfache Nutzung der folgenden Schnittstellen: Optimierungen initialisieren, starten und die Ergebnisse auslesen. Außerdem gibt es weitere Schnittstellen für Statistiken und zur Löschung von Optimierungssessions. Diese werden folgend beschrieben.

5.1.1 Optimierungen initialisieren

Mit dieser Schnittstelle können Optimierungssessions erstellt und initialisiert werden. Dafür wird eine JSON-Datei mit Werten erwartet, die hier wiederholt werden. Diese können von Excel I/O generiert werden (siehe auch Code 7.1 und Abschnitt 4.2.1):

[1] https://github.com/jku-win-dke/mt2106-slma-optimizer

- `optId`: Identifikation der Optimierungssession. Beinhaltet eine UUID (Universally Unique Identifier).
- `initialFlightSequence`: Informationen zu einer vorgeschlagenen initialen Flugsequenz.
- `flights`: Beinhalten Informationen zu den Flügen, wobei pro Flug die Daten zu `flightId` (Flug-Identifikation), `scheduledTime` (ursprünglich geplante Abflugzeit) und die `weightMap` (Gewichtstabelle) gespeichert werden.
- `slots`: Zusätzlich dazu werden die Zeitpunkte der verfügbaren Slots gespeichert.
- `optimizationFramework`: Gibt das zu verwendende Framework an (Hungarian Algorithm, OptaPlanner oder Jenetics).
- `margins`: Außerdem werden die Margins (`timeNotBefore`, `timeWished`, `timeNotAfter`) inklusive der `scheduledTime` (ursprünglich geplante Abflugzeit) und die `flightId` (Flug-Identifikation) gespeichert für eine verbesserte Auswertung im Anschluss an die Optimierung. Diese Werte sind optional.
- `parameters`: Hier werden die nötigen weiteren Parameter für die Optimierung gespeichert, die für OptaPlanner und Jenetics möglich sind.

Nachdem die Optimierungssession erstellt und initialisiert wird, weiß der Heuristische Optimierer welche Einstellungen diese Session benötigt. Über das Framework sind, wie bereits zuvor beschrieben, drei Möglichkeiten gegeben: Hungarian Algorithm, OptaPlanner und Jenetics. Der Hungarian Algorithm benötigt keine Parameter. Jenetics und OptaPlanner können mit einstellbaren Parametern arbeiten. Detailliertere Einstellungsmöglichkeiten der drei Frameworks werden in späteren Abschnitten erläutert.

Für Jenetics werden die Parameter in definierten Rahmen genau vorgegeben. Es kann das maximale Alter von Phenotypen eingestellt werden, der Anteil von Nachkommen in Relation zu Überlebenden und die Bevölkerungsgröße. Außerdem sind genauere Einstellungen zum Mutator (Swap Mutator) und Crossover (Partially Matched Crossover) möglich bezüglich Rekombinationswahrscheinlichkeit. Es können Selektoren für Nachkommen und für Überlebende mitsamt Parameter eingestellt werden. Folgende Selektoren sind möglich: Boltzmann Selektor, Exponential Rank Selektor, Linear Rank Selektor, Roulette

Wheel Selektor, Stochastic Universal Selektor, Tournament Selektor und Truncation Selektor. Zusätzlich sind Abbruchbedingungen einstellbar, welche sich auf Fitnesswert, Zeit, Generationenanzahl oder andere Werte beziehen können. Diese Einstellungen werden in Abschnitt 5.3 genauer erläutert.

Für OptaPlanner wiederum können nur vorgegebene Konfigurationen ausgewählt und die Ausführungszeit festgelegt werden. Mögliche Konfigurationen sind: Great Deluge, Hill Climbing, Late Acceptance, Simulated Annealing, Step Counting Hill Climbing, Strategic Oscillation und Tabu Search. Die genaueren Definitionen werden in den folgenden Abschnitten zu OptaPlanner, Findung der Parameter und zur Hauptstudie erläutert.

Zusätzlich dazu ist die Auswahl des Hungarian Algorithm, auch Ungarische Methode genannt, möglich. Dieser Algorithmus verwendet keine weiteren Parameter und ignoriert vorgegebene Ausführungszeiten.

5.1.2 Optimierungen starten

Über diese Schnittstelle wird die Optimierungssession asynchron gestartet. Daher können auch verschiedene Sessionen parallel ausgeführt werden. Diese dauert dann so lange, wie es von der Konfiguration des bestimmten Frameworks vorbestimmt ist. Für den Start wird die Optimierungssessions-Identifikation (UUID) benötigt. Je nach Einstellung wird dann das Jenetics-, OptaPlanner- oder Hungarian Algorithm-Framework verwendet.

5.1.3 Optimierungsresultate auslesen

Mit dieser Schnittstelle kann unter Angabe der Optimierungssessions-Identifikations-Nummer das Ergebnis der Optimierungssession zurückgegeben werden. Hierbei werden die vorgeschlagene Flugsequenz und die Identifikations-Nummer übermittelt. Außerdem gibt die Schnittstelle die Margins zurück, sofern sie bei der Erstellung der Session mitgegeben wurden. Ein Beispiel wird in Code 5.1 abgekürzt dargestellt. Wie hier sichtbar ist, werden auch der erreichte Fitnesswert der empfohlenen Abflugreihenfolge übermittelt. Zusätzlich wird die Anzahl der Aufrufe der Fitnessfunktion für Jenetics- und OptaPlanner-Optimierungen zurückgegeben.

```
{
  "optId":"05eb80e9-562e-43b6-b02b-43aa1ad3b498",
  "optimizedFlightSequence":["F1","F2",[...],"F99","F100"],
  "slots":[
    "2021-07-20T14:35:43.808514700Z","2021-07-20T14:37:43.808514700Z",[...],"
        2021-07-20T17:53:43.808514700Z"
  ],
  "margins":[
    {
      "flightId":"F1",
      "scheduledTime":"2021-07-20T14:35:43.808514700Z",
      "timeNotBefore":"2021-07-20T14:10:43.808514700Z",
      "timeWished":"2021-07-20T14:35:43.808514700Z",
      "timeNotAfter":"2021-07-20T15:00:43.808514700Z"
    },
    {
      "flightId":"F2",
      "scheduledTime":"2021-07-20T14:35:43.808514700Z",
      "timeNotBefore":"2021-07-20T14:12:43.808514700Z",
      "timeWished":"2021-07-20T14:37:43.808514700Z",
      "timeNotAfter":"2021-07-20T15:02:43.808514700Z"
    },{
    [...]
    },{
      "flightId":"F100",
      "scheduledTime":"2021-07-20T14:35:43.808514700Z",
      "timeNotBefore":"2021-07-20T17:28:43.808514700Z",
      "timeWished":"2021-07-20T17:53:43.808514700Z",
      "timeNotAfter":"2021-07-20T18:18:43.808514700Z"
    }
  ],
  "fitness":1407993,
  "fitnessFunctionInvocations":250099
}
```

Code 5.1: Auszug aus generierter JSON-Datei

5.1.4 Weitere Schnittstellen

Zusätzlich gibt es noch eine Schnittstelle zur synchronen Ausführung von Optimierungssessions. In diesem Fall kann nur eine Session zu einer Zeit ausgeführt werden. Weiter gibt es eine Schnittstelle zur Löschung von Sessions-Daten. Bei Eingabe der jeweiligen Optimierungssessions-Identifikation (UUID) werden sowohl die Eingabedaten als auch

Ergebnisse gelöscht, sofern sie vorhanden sind. Ebenso sind im Laufe des Projektes Methoden für Statistiken geplant, die jedoch nicht Teil der Masterarbeit sind.

5.2 Fitnesswert-Berechnung

Der Fitnesswert wird über die Gewichtstabelle berechnet. Wie die Gewichtstabelle generiert wird und die einzelnen Gewichte bestimmt werden, kann in Abschnitt 4.2.1 nachgelesen werden. In Tabelle 5.1 wird eine beispielhafte Gewichtstabelle von den Flügen A, B, C, D, E, F, G und H für die Slots 1, 2, 3, 4, 5, 6, 7 und 8 angegeben. Die Gewichte sind von 0 bis 50 eingetragen, wodurch beispielsweise der Flug B bei Slot 1 ein Gewicht von 50 hat.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A	0	20	50	30	20	10	0	0
B	50	40	30	20	10	0	0	0
C	20	40	50	35	20	5	0	0
D	10	25	50	35	30	25	20	15
E	0	0	10	30	50	30	10	0
F	0	0	0	0	25	50	30	10
G	5	30	50	40	30	20	10	0
H	0	25	35	45	50	45	40	35

Tabelle 5.1: Gewichtstabelle

In Tabelle 5.2 werden beispielhafte Zuteilungen angegeben. Wie zu sehen ist, ergibt die Zuteilung in Variante 1 nicht die beste Lösung. Die beste Zuteilung ist in Variante 3 zu finden. Der Fitnesswert der Optimierung nach Variante 3 ist 335, was die Summe der Gewichte bei den zugewiesenen Slots ist. Flug B wird hier zu Slot 1 zugewiesen und hat ein Gewicht von 50. Flug C wird dem zweiten Slot zugewiesen mit einem Gewicht von 40.

Durch die Gewichte und die Fitnessberechnung soll eine möglichst optimale Flugreihenfolge ermittelt werden. Folgende Ziele sollen von den Optimierungssessions eingehalten werden:

- Der Abstand zwischen dem zugeteilten Slot und `TimeWished` soll minimiert werden.

	1	2	3	4	5	6	7	8	Fitnesswert
Zuteilungsvariante 1:	A	B	C	D	E	F	G	H	
Gewicht bei zugewiesenem Slot:	0	40	50	35	50	50	10	35	$\rightarrow$ **270**
Zuteilungsvariante 2:	B	C	A	H	E	F	D	G	
Gewicht bei zugewiesenem Slot:	50	40	50	45	50	50	20	0	$\rightarrow$ **305**
Zuteilungsvariante 3:	B	C	A	G	E	F	D	H	
Gewicht bei zugewiesenem Slot:	50	40	50	40	50	50	20	35	$\rightarrow$ **335**

Tabelle 5.2: Gewichtstabelle

- Der zugeteilte Slot soll möglichst nicht vor `TimeNotBefore` und nicht nach `TimeNotAfter` zugewiesen werden.
- Sofern für einen Slot mehrere Flüge als Kandidaten möglich sind, soll der Flug mit dem größten Gewicht bei diesem Slot priorisiert werden.

5.3 Jenetics

Jenetics ist das Framework zur Nutzung des genetischen Algorithmus [39]. Nachfolgend wird ein kurzer Überblick gegeben, sowie mögliche Einstellungsmöglichkeiten erklärt. Jenetics wurde in Abschnitt 2.2.1 vorgestellt und bietet verschiedenste Einstellungen und Implementierungen des genetischen Algorithmus an. Im Rahmen der Implementierung wurden die Fitnessberechnungen und Einschränkungen in den restlichen Code vom heuristischen Optimierer integriert.

Es gibt verschiedene Einstellungsmöglichkeiten. Nachfolgend werden diese mitsamt Variablennamen für die Verwendung in `parameters` aufgeführt. Eine mögliche Einstellung wird ebenso in Code 5.2 dargestellt, was ein Auszug aus einer Optimierungssessions-Datei ist.

```
1        [...]
2        "parameters" : {
3          "offspringFraction" : 0.7,
4          "crossover" : "PARTIALLY_MATCHED_CROSSOVER",
5          "maximalPhenotypeAge" : 80,
6          "crossoverAlterProbability" : 0.9,
7          "survivorsSelectorParameter" : 50,
```

```
8           "survivorsSelector" : "TOURNAMENT_SELECTOR",
9           "populationSize" : 500,
10          "mutator" : "SWAP_MUTATOR",
11          "terminationConditions" : {
12            "BY_EXECUTION_TIME" : 10
13          },
14          "offspringSelectorParameter" : 50,
15          "offspringSelector" : "TOURNAMENT_SELECTOR",
16          "mutatorAlterProbability" : 0.15
17        }
18        [...]
```

Code 5.2: Auszug aus Jenetics-Einstellungen einer JSON-Datei

- **maximales Alter von Phenotypen** (`maximalPhenotypeAge`): Hierüber wird das maximale Alter von Phenotypen über einen positiven Zahlenwert eingestellt.
- **Bevölkerungsgröße** (`populationSize`): Bestimmt die maximale Größe jeder Generation und wird mit einem positiven Zahlenwert eingestellt.
- **Mutator** (`mutator`): Obwohl hier keine Wahlmöglichkeit besteht, muss der Wert „SWAP_MUTATOR“ eingetragen werden.
- **Rekombinationswahrscheinlichkeit (Mutator)** (`mutatorAlterProbability`): Hierzu kann ein Wert zwischen 0 und 1 (exklusiv 1) verwendet werden.
- **Crossover** (`crossover`): Hier besteht ebenso keine Wahlmöglichkeit. Es muss der Wert „PARTIALLY_MATCHED_CROSSOVER“ eingestellt werden.
- **Rekombinationswahrscheinlichkeit (Crossover)** (`crossoverAlterProbability`): Hierzu kann ein Wert zwischen 0 und 1 (exklusiv 1) verwendet werden.
- **Selektoren** (`offspringSelector` und `survivorsSelector`): Die Selektoren für Nachkommen und für Überlebende können individuell voneinander eingestellt werden. Mögliche Werte sind „BOLTZMANN_SELECTOR“, „EXPONENTIAL_RANK-_SELECTOR“, „LINEAR_RANK_SELECTOR“, „ROULETTE_WHEEL_SELECTOR“, „STOCHASTIC_UNIVERSAL_SELECTOR“, „TOURNAMENT_SELECTOR“ und „TRUNCATION_SELECTOR“.

- **Selektorparameter** (`survivorsSelectorParameter` und `offspringSelectorParameter`): Je nach Selektor sind andere Parameter notwendig. „BOLTZMANN_SELECTOR“ erlaubt positive und negative Zahlen. „EXPONENTIAL_RANK_SELECTOR“ erlaubt Zahlen zwischen 0 und 1 (exklusiv 1). „LINEAR_RANK_SELECTOR“ benötigt als Parameter Zahlen größer oder gleich Null. Die Selektoren „ROULETTE_WHEEL_SELECTOR“ und „STOCHASTIC_UNIVERSAL_SELECTOR“ benötigen keine Parameter. „TOURNAMENT_SELECTOR“ benötigt eine Zahl größer oder gleich 2. „TRUNCATION_SELECTOR“ benötigt eine positive Zahl als Parameter.
- **Fraktion der Nachkommen** (`offspringFraction`): Zeigt das Verhältnis von Nachkommen und Überlebenden in Generationen an.
- **Abbruchbedingungen** (`terminationConditions`): Für die Abbruchbedingungen muss wie in Code 5.2 ein „BY_EXECUTION_TIME“ mit einem Parameter angegeben werden. Im gezeigtem Beispiel wird eine Optimierungssession 10 Sekunden ausgeführt.

5.4 OptaPlanner

OptaPlanner bietet eine Auswahl an Konstruktionsheuristiken und dazu Metaheuristiken, die in Kombination verwendbar sind. Für die Verwendung in der Hauptstudie wurden die in den nächsten Abschnitten beschriebenen Einstellungsmöglichkeiten erstellt, die in der letzten Version der Implementierung finalisiert sind. Auch für OptaPlanner gibt es verschiedene Einstellungsmöglichkeiten. Generell kann die Art der Konstruktionsheuristik und der Metaheuristik definiert werden. In der durchgeführten Implementierung lassen sich jedoch nur der Konfigurationsname (`configurationName`) und die Dauer der Optimierungssession (`secondsSpentLimit`) auswählen. Ein Beispiel ist in Code 5.3 ersichtlich. Über den Konfigurationsnamen werden vordefinierte Einstellungen für die Session übernommen. Der Wert `acceptedCountLimit` gibt an, wie viele akzeptierte Bewegungen pro Schritt evaluiert werden sollen [26].

```
1        [...]
2        "parameters" : {
3          "secondsSpentLimit" : 10,
4          "configurationName" : "TABU_SEARCH"
```

```
5            }
6            [...]
```

Code 5.3: Auszug aus OptaPlanner-Einstellungen einer JSON-Datei

Für jede Einstellung wird First Fit als Konstruktionsheuristik in der Standardeinstellung verwendet, da nur die Wahl des Metaheuristik-Algorithmus eine größere Auswirkung auf das Endergebnis hat. Die später übernommenen Voreinstellungen sind wie folgt:

- Hill Climbing (`HILL_CLIMBING`): Für Hill Climbing wird die vorgegebene Standardeinstellung vom Framework verwendet.
- Tabu Search (`TABU_SEARCH`): Die Standardeinstellung wird für Tabu Search ebenso verwendet.
- Simulated Annealing (`SIMULATED_ANNEALING`): Mit Abweichung der Starttemperatur (0hard/500soft) wird hier auch die Standardeinstellung verwendet.
- Late Acceptance (`LATE_ACCEPTANCE`): Late Acceptance nutzt eine `lateAcceptanceSize` von 50 und ein `acceptedCountLimit` von 4.
- Great Deluge (`GREAT_DELUGE`): Für Great Deluge wird der Wert des schrittweise steigenden Wasserlevels auf 0hard/100soft gesetzt und ein `acceptedCountLimit` von 1 benutzt.
- Step Counting Hill Climbing (`STEP_COUNTING_HILL_CLIMBING`): Es wird eine `stepCountingHillClimbingSize` von 20 und ein `acceptedCountLimit` von 1 verwendet.
- Strategic Oscillation (`STRATEGIC_OSCILLATION`): Strategic Oscillation wird in Verbindung mit Tabu Search verwendet. Es wird eine Tabu Ratio von 0,04, ein `acceptedCountLimit` von 1000 verwendet. Die Optimierung ist anders als die übliche Einstellung eingestellt bezüglich Auswahl der Bewegung für den aktuellen Schritt. Wenn es eine akzeptierte Bewegung gibt, wird diese verwendet. Sofern keine Bewegung ausgewählt werden konnte, soll eine Bewegung benützt werden, die einen besseren Fitnesswert als der aktuelle Höchstwert hat [26]. Daher wird hier „STRATEGIC_OSCILLATION_BY_LEVEL_ON_BEST_SCORE“ bei `finalistPodiumType` verwendet.

5.5 Hungarian Algorithm

Für die Ungarische Methode (siehe Abschnitt 2.1.2) gibt es keine Möglichkeiten zur Einstellung. Die Implementierung des Algorithmus selbst wurde von der referenzierten Quelle [36] übernommen. Im Projekt SlotMachine kann die Ungarische Methode jedoch nicht verwendet werden, da sie keine Möglichkeit bietet die Suche nach einer optimalen Lösung von der Evaluierung einer gefundenen Lösung zu trennen. Dadurch ist eine vertraulichkeitsbewahrende Optimierung der Abflugsreihenfolgen mit der im SlotMachine entwickelten Architektur nicht möglich. Die Ungarische Methode wird jedoch in dieser Arbeit zur Ermittlung des Referenzwerts der optimalen Lösung verwendet.

6 Findung der Parameter

In diesem Abschnitt werden zuerst die Grunddaten für die folgenden Experimente dargestellt. Danach wird die Vorgehensweise der Parametertests beschrieben. Abschließend werden die Ergebnisse und deren Auswirkungen auf die Hauptstudie besprochen.

6.1 Grunddaten

In der Vorstudie wurden Daten basierend auf der Arbeit von Ruiz [30] erstellt, um möglichst realistische Flugdaten nachahmen zu können. Hierfür wurden die Zeitabstände zwischen `TimeNotBefore` und `TimeNotAfter` dementsprechend anhand eines Zufallsgenerator bestimmt, der Werte zwischen 0 und 40 Minuten ausgibt. Die Prioritäten wurden ebenso anhand eines Zufallsgenerators als Zahlen zwischen 0,75 und 2,5 gesetzt. Der Wert `TimeWished` wurde anhand von Daten in der referenzierten Arbeit in Bezug auf Verspätung zum Slot gesetzt. Die Flüge können in Tabelle 6.1 untersucht werden, die die Daten so darstellt, wie sie im Excel Quelldokument auffindbar sind. Die Optimierungsdaten dazu sind wie folgt:

- Optimierungssessions-Identifikation: 063e75cf-1e46-4dcb-b204-d833bcd289ab
- Startzeit: 2021-6-10 12:15:00
- Endzeit: 2021-6-10 17:00:00
- Intervallzeit pro Slot in Sekunden: 180
- Framework: OptaPlanner *(Das Framework wird für manche Parametertests auf Jenetics oder Hungarian (für Hungarian Algorithm) gestellt.)*

- Mindestwert: $-10\,000$
- Maximalwert: $10\,000$
- Fallhöhe: 6000

FlightId	ScheduledTime	TimeNotBefore	TimeWished	TimeNotAfter	priority	AssignedSlot
F01	2021-06-10 03:00:00	2021-06-10 11:55:00	2021-06-10 12:15:00	2021-06-10 12:26:00	2,11	
F02	2021-06-10 03:03:00	2021-06-10 12:07:00	2021-06-10 12:18:00	2021-06-10 12:28:00	1,80	
F03	2021-06-10 03:06:00	2021-06-10 12:11:00	2021-06-10 12:24:00	2021-06-10 12:36:00	0,86	
F04	2021-06-10 03:09:00	2021-06-10 12:28:00	2021-06-10 12:30:00	2021-06-10 12:36:00	2,18	
F05	2021-06-10 03:12:00	2021-06-10 12:21:00	2021-06-10 12:33:00	2021-06-10 12:46:00	0,77	
F06	2021-06-10 03:15:00	2021-06-10 12:26:00	2021-06-10 12:36:00	2021-06-10 12:51:00	0,83	
F07	2021-06-10 03:18:00	2021-06-10 12:24:00	2021-06-10 12:39:00	2021-06-10 12:45:00	1,79	
F08	2021-06-10 03:21:00	2021-06-10 12:40:00	2021-06-10 12:42:00	2021-06-10 12:46:00	2,45	
F09	2021-06-10 03:24:00	2021-06-10 12:35:00	2021-06-10 12:45:00	2021-06-10 12:55:00	2,42	
F10	2021-06-10 03:27:00	2021-06-10 12:43:00	2021-06-10 12:48:00	2021-06-10 13:06:00	1,24	
F11	2021-06-10 03:30:00	2021-06-10 12:51:00	2021-06-10 12:51:00	2021-06-10 12:59:00	1,07	
F12	2021-06-10 03:33:00	2021-06-10 12:35:00	2021-06-10 12:54:00	2021-06-10 13:04:00	1,10	
F13	2021-06-10 03:36:00	2021-06-10 12:55:00	2021-06-10 12:57:00	2021-06-10 12:58:00	0,97	
F14	2021-06-10 03:39:00	2021-06-10 13:11:00	2021-06-10 13:12:00	2021-06-10 13:15:00	2,20	
F15	2021-06-10 03:42:00	2021-06-10 13:08:00	2021-06-10 13:14:00	2021-06-10 13:34:00	1,38	
F16	2021-06-10 03:45:00	2021-06-10 13:13:00	2021-06-10 13:24:00	2021-06-10 13:24:00	1,19	
F17	2021-06-10 03:48:00	2021-06-10 13:18:00	2021-06-10 13:28:00	2021-06-10 13:32:00	0,85	
F18	2021-06-10 03:51:00	2021-06-10 13:08:00	2021-06-10 13:23:00	2021-06-10 13:36:00	0,86	
F19	2021-06-10 03:54:00	2021-06-10 13:18:00	2021-06-10 13:31:00	2021-06-10 13:39:00	1,67	
F20	2021-06-10 03:57:00	2021-06-10 13:22:00	2021-06-10 13:31:00	2021-06-10 13:49:00	2,46	
F21	2021-06-10 04:00:00	2021-06-10 13:26:00	2021-06-10 13:27:00	2021-06-10 13:37:00	2,14	
F22	2021-06-10 04:03:00	2021-06-10 13:37:00	2021-06-10 13:44:00	2021-06-10 13:55:00	1,29	
F23	2021-06-10 04:06:00	2021-06-10 13:27:00	2021-06-10 13:46:00	2021-06-10 13:47:00	1,03	
F24	2021-06-10 04:09:00	2021-06-10 13:57:00	2021-06-10 14:08:00	2021-06-10 14:16:00	2,38	
F25	2021-06-10 04:12:00	2021-06-10 13:37:00	2021-06-10 13:50:00	2021-06-10 13:52:00	1,74	
F26	2021-06-10 04:15:00	2021-06-10 13:40:00	2021-06-10 13:48:00	2021-06-10 14:07:00	2,50	
F27	2021-06-10 04:18:00	2021-06-10 13:19:00	2021-06-10 13:38:00	2021-06-10 13:53:00	1,58	

FlightId	ScheduledTime	TimeNotBefore	TimeWished	TimeNotAfter	priority	AssignedSlot
F28	2021-06-10 04:21:00	2021-06-10 13:25:00	2021-06-10 13:43:00	2021-06-10 14:02:00	1,20	
F29	2021-06-10 04:24:00	2021-06-10 14:03:00	2021-06-10 14:09:00	2021-06-10 14:25:00	1,60	
F30	2021-06-10 04:27:00	2021-06-10 13:55:00	2021-06-10 13:55:00	2021-06-10 14:14:00	2,42	
F31	2021-06-10 04:30:00	2021-06-10 13:25:00	2021-06-10 13:43:00	2021-06-10 13:48:00	2,48	
F32	2021-06-10 04:33:00	2021-06-10 13:48:00	2021-06-10 13:51:00	2021-06-10 13:58:00	1,71	
F33	2021-06-10 04:36:00	2021-06-10 14:09:00	2021-06-10 14:26:00	2021-06-10 14:43:00	1,96	
F34	2021-06-10 04:39:00	2021-06-10 13:49:00	2021-06-10 13:53:00	2021-06-10 14:12:00	1,45	
F35	2021-06-10 04:42:00	2021-06-10 13:58:00	2021-06-10 13:58:00	2021-06-10 14:10:00	1,74	
F36	2021-06-10 04:45:00	2021-06-10 13:54:00	2021-06-10 13:56:00	2021-06-10 14:02:00	1,80	
F37	2021-06-10 04:48:00	2021-06-10 13:45:00	2021-06-10 13:58:00	2021-06-10 14:03:00	0,94	
F38	2021-06-10 04:51:00	2021-06-10 14:12:00	2021-06-10 14:14:00	2021-06-10 14:34:00	1,74	
F39	2021-06-10 04:54:00	2021-06-10 14:14:00	2021-06-10 14:25:00	2021-06-10 14:43:00	1,21	
F40	2021-06-10 04:57:00	2021-06-10 14:36:00	2021-06-10 14:42:00	2021-06-10 14:58:00	1,75	
F41	2021-06-10 05:00:00	2021-06-10 14:02:00	2021-06-10 14:09:00	2021-06-10 14:12:00	2,28	
F42	2021-06-10 05:03:00	2021-06-10 14:43:00	2021-06-10 14:49:00	2021-06-10 14:55:00	1,44	
F43	2021-06-10 05:06:00	2021-06-10 14:09:00	2021-06-10 14:14:00	2021-06-10 14:26:00	1,04	
F44	2021-06-10 05:09:00	2021-06-10 14:01:00	2021-06-10 14:08:00	2021-06-10 14:08:00	1,10	
F45	2021-06-10 05:12:00	2021-06-10 13:55:00	2021-06-10 14:14:00	2021-06-10 14:21:00	1,37	
F46	2021-06-10 05:15:00	2021-06-10 14:12:00	2021-06-10 14:18:00	2021-06-10 14:23:00	1,64	
F47	2021-06-10 05:18:00	2021-06-10 14:43:00	2021-06-10 14:54:00	2021-06-10 15:12:00	1,88	
F48	2021-06-10 05:21:00	2021-06-10 14:12:00	2021-06-10 14:23:00	2021-06-10 14:25:00	1,81	
F49	2021-06-10 05:24:00	2021-06-10 13:50:00	2021-06-10 14:08:00	2021-06-10 14:28:00	0,87	
F50	2021-06-10 05:27:00	2021-06-10 14:35:00	2021-06-10 14:54:00	2021-06-10 15:11:00	1,72	
F51	2021-06-10 05:30:00	2021-06-10 14:47:00	2021-06-10 15:01:00	2021-06-10 15:15:00	1,73	
F52	2021-06-10 05:33:00	2021-06-10 14:43:00	2021-06-10 14:51:00	2021-06-10 15:07:00	1,12	
F53	2021-06-10 05:36:00	2021-06-10 14:45:00	2021-06-10 14:47:00	2021-06-10 14:54:00	2,11	
F54	2021-06-10 05:39:00	2021-06-10 14:47:00	2021-06-10 14:48:00	2021-06-10 14:54:00	1,51	

FlightId	ScheduledTime	TimeNotBefore	TimeWished	TimeNotAfter	priority	AssignedSlot
F55	2021-06-10 05:42:00	2021-06-10 14:44:00	2021-06-10 14:53:00	2021-06-10 14:54:00	0,86	
F56	2021-06-10 05:45:00	2021-06-10 14:46:00	2021-06-10 15:01:00	2021-06-10 15:14:00	2,39	
F57	2021-06-10 05:48:00	2021-06-10 14:52:00	2021-06-10 15:09:00	2021-06-10 15:28:00	1,10	
F58	2021-06-10 05:51:00	2021-06-10 14:48:00	2021-06-10 14:54:00	2021-06-10 14:55:00	1,95	
F59	2021-06-10 05:54:00	2021-06-10 14:31:00	2021-06-10 14:51:00	2021-06-10 15:00:00	2,08	
F60	2021-06-10 05:57:00	2021-06-10 14:43:00	2021-06-10 14:44:00	2021-06-10 14:48:00	0,79	
F61	2021-06-10 06:00:00	2021-06-10 15:02:00	2021-06-10 15:08:00	2021-06-10 15:25:00	0,98	
F62	2021-06-10 06:03:00	2021-06-10 15:08:00	2021-06-10 15:08:00	2021-06-10 15:09:00	1,13	
F63	2021-06-10 06:06:00	2021-06-10 14:40:00	2021-06-10 14:59:00	2021-06-10 15:09:00	2,23	
F64	2021-06-10 06:09:00	2021-06-10 14:56:00	2021-06-10 15:15:00	2021-06-10 15:31:00	1,68	
F65	2021-06-10 06:12:00	2021-06-10 15:16:00	2021-06-10 15:28:00	2021-06-10 15:42:00	2,41	
F66	2021-06-10 06:15:00	2021-06-10 15:08:00	2021-06-10 15:24:00	2021-06-10 15:29:00	0,93	
F67	2021-06-10 06:18:00	2021-06-10 15:19:00	2021-06-10 15:31:00	2021-06-10 15:40:00	1,16	
F68	2021-06-10 06:21:00	2021-06-10 15:04:00	2021-06-10 15:23:00	2021-06-10 15:25:00	0,81	
F69	2021-06-10 06:24:00	2021-06-10 15:15:00	2021-06-10 15:31:00	2021-06-10 15:47:00	2,24	
F70	2021-06-10 06:27:00	2021-06-10 15:47:00	2021-06-10 16:06:00	2021-06-10 16:14:00	1,56	
F71	2021-06-10 06:30:00	2021-06-10 15:44:00	2021-06-10 15:57:00	2021-06-10 16:02:00	2,31	
F72	2021-06-10 06:33:00	2021-06-10 15:56:00	2021-06-10 15:56:00	2021-06-10 15:57:00	1,63	
F73	2021-06-10 06:36:00	2021-06-10 15:38:00	2021-06-10 15:48:00	2021-06-10 15:57:00	1,52	
F74	2021-06-10 06:39:00	2021-06-10 15:53:00	2021-06-10 15:53:00	2021-06-10 16:07:00	1,05	
F75	2021-06-10 06:42:00	2021-06-10 15:48:00	2021-06-10 16:01:00	2021-06-10 16:16:00	1,78	
F76	2021-06-10 06:45:00	2021-06-10 15:49:00	2021-06-10 16:00:00	2021-06-10 16:10:00	2,06	
F77	2021-06-10 06:48:00	2021-06-10 16:03:00	2021-06-10 16:09:00	2021-06-10 16:20:00	1,54	
F78	2021-06-10 06:51:00	2021-06-10 16:10:00	2021-06-10 16:21:00	2021-06-10 16:39:00	1,14	
F79	2021-06-10 06:54:00	2021-06-10 16:06:00	2021-06-10 16:25:00	2021-06-10 16:39:00	0,87	
F80	2021-06-10 06:57:00	2021-06-10 15:59:00	2021-06-10 16:05:00	2021-06-10 16:14:00	2,33	
F81	2021-06-10 07:00:00	2021-06-10 16:21:00	2021-06-10 16:24:00	2021-06-10 16:27:00	1,79	

FlightId	ScheduledTime	TimeNotBefore	TimeWished	TimeNotAfter	priority	AssignedSlot
F82	2021-06-10 07:03:00	2021-06-10 16:04:00	2021-06-10 16:23:00	2021-06-10 16:30:00	2,16	
F83	2021-06-10 07:06:00	2021-06-10 16:13:00	2021-06-10 16:20:00	2021-06-10 16:29:00	1,05	
F84	2021-06-10 07:09:00	2021-06-10 16:59:00	2021-06-10 17:02:00	2021-06-10 17:09:00	2,44	
F85	2021-06-10 07:12:00	2021-06-10 16:38:00	2021-06-10 16:48:00	2021-06-10 17:02:00	1,51	
F86	2021-06-10 07:15:00	2021-06-10 16:24:00	2021-06-10 16:30:00	2021-06-10 16:43:00	1,75	
F87	2021-06-10 07:18:00	2021-06-10 16:39:00	2021-06-10 16:39:00	2021-06-10 16:47:00	2,01	
F88	2021-06-10 07:21:00	2021-06-10 16:27:00	2021-06-10 16:41:00	2021-06-10 16:56:00	2,18	
F89	2021-06-10 07:24:00	2021-06-10 16:48:00	2021-06-10 17:02:00	2021-06-10 17:14:00	1,80	
F90	2021-06-10 07:27:00	2021-06-10 16:34:00	2021-06-10 16:48:00	2021-06-10 16:52:00	1,82	
F91	2021-06-10 07:30:00	2021-06-10 16:40:00	2021-06-10 16:45:00	2021-06-10 16:55:00	1,64	
F92	2021-06-10 07:33:00	2021-06-10 16:56:00	2021-06-10 17:02:00	2021-06-10 17:13:00	2,14	
F93	2021-06-10 07:36:00	2021-06-10 16:44:00	2021-06-10 17:02:00	2021-06-10 17:14:00	1,48	
F94	2021-06-10 07:39:00	2021-06-10 16:43:00	2021-06-10 17:02:00	2021-06-10 17:13:00	1,50	
F95	2021-06-10 07:42:00	2021-06-10 16:44:00	2021-06-10 17:04:00	2021-06-10 17:19:00	1,02	
F96	2021-06-10 07:45:00	2021-06-10 16:49:00	2021-06-10 17:06:00	2021-06-10 17:23:00	2,24	

Tabelle 6.1: Daten für die Findung der Parameter

6.2 Parametertests

Um die zu testenden Einstellungen von den Algorithmen von Jenetics und OptaPlanner auf ein kleines Maß zu reduzieren, wurden mit den Daten der Vorstudie diverse Parameter getestet. Die Tests wurden mit unterschiedlichen Laufzeiten durchgeführt. Es wurde darauf geachtet, dass unter den Frameworks selbst eine Vergleichbarkeit bestehen bleibt. Aufgrund einer Umstellung der Kostenfunktion in der Vorstudienphase wurden die OptaPlanner-Vorstudientests mit der alten Kostenfunktion durchgeführt und die Jenetics-Vorstudientests mit der neuen. Daher sollen die Ergebnisse von Jenetics-Tests nicht mit den Ergebnissen der OptaPlanner-Tests verglichen werden. Für die geplanten Tests mit dem Hungarian Algorithm, auch Ungarische Methode genannt, sind keine Parametertests notwendig, da es hier keine einstellbaren Parameter gibt.

6.2.1 Jenetics: Auswahl der Selektoren

Für Jenetics wurde folgende Standardeinstellung für die Tests der Selektoren verwendet, welche aus dem Handbuch von Jenetics [38] entnommen wurden:

- maximales Alter der Phenotypen: 70 Generationen
- Bevölkerungsgröße pro Generation: 50
- Fraktion der Nachkommen: 0,6
- Mutator: Swap Mutator, mit einer Rekombinationswahrscheinlichkeit von 0,2
- Crossover: Partially Matched Crossover, mit einer Rekombinationswahrscheinlichkeit von 0,35
- Abbruchbedingung: beende die Optimierung nach 10 Sekunden

Die Standardeinstellungen stammen aus der Dokumentation von Jenetics [39]. Die Abbruchbedingung wurde gesetzt, um ähnlich wie die Hauptstudie zu agieren, wo Testläufe 5 oder 10 Sekunden Zeit bekommen.

Es werden die folgenden Selektoren geprüft: Tournament-Selektor, Boltzmann-Selektor, Exponential-Rank-Selektor, Roulette-Wheel-Selektor, Stochastic-Universal-Selektor, Linear-Rank-Selektor, Truncation-Selektor. Danach werden die Bevölkerungsgröße, Fraktion

der Nachkommen, Rekombinationswahrscheinlichkeit vom Mutator, Rekombinationswahrscheinlichkeit vom Crossover und maximales Alter der Phenotypen als einstellbare Parameter getestet. Jeder Test wird mindestens fünfmal durchgeführt, da bei Jenetics die Ergebnisse bei gleichen Einstellungen leicht variieren können. Genauere Erklärungen der Funktionsweisen der Selektoren finden sich im Handbuch [38].

Tournament-Selektor

Hier wird die Anzahl der ausgewählten Individuen (`size`) getestet. Eine Anzahl von keinem oder nur einem Individuum ergibt einen Fehler. Getestet wurden Werte in der Nähe der empfohlenen Standardeinstellung (2) [40] und mit größeren Abständen auch größere Werte. Ergebnisse sind sichtbar in Tabelle 6.2 und Abbildung 6.1. Wie man in der Abbildung erkennen kann, sind Werte zwischen 3 und etwa 50 ausgewählten Individuen (`size`) zu bevorzugen, da hier ein Fitnesswert im Durchschnitt von über einer Million erreicht werden kann.

`size`	**Fitness** *(Durchschnitt)*	`size`	**Fitness** *(Durchschnitt)*	`size`	**Fitness** *(Durchschnitt)*
0	—	7	1 018 014,4	50	1 008 958,2
1	—	8	1 035 676,2	100	996 494,2
2	963 152,5	9	1 040 381,0	500	963 237,6
3	1 024 998,3	10	1 029 006,4	1000	916 286,2
4	1 040 932,4	13	1 016 603,0	5000	752 742,6
5	1 003 452,2	16	1 019 590,2	10 000	677 004,6
6	1 027 015,0	20	1 028 401,0	100 000	555 860,8

Tabelle 6.2: Vorstudie, Jenetics, Tournament-Selektor

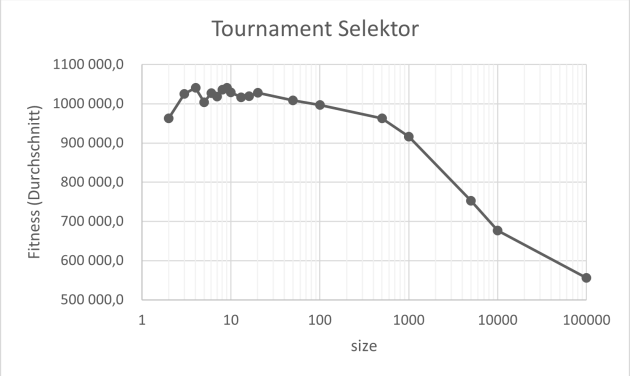

Abbildung 6.1: Vorstudie, Jenetics, Tournament-Selektor

Boltzmann-Selektor

Hier wird die Auswahlintensität mittels Wert `b` getestet. Der Parameter kann beliebige positive und negative Werte annehmen. Getestet wurden Werte in der Nähe der empfohlenen Standardeinstellung (4) [40] und mit größeren Abständen auch weiter entfernte Werte. Ergebnisse sind in Tabelle 6.3 und Abbildung 6.2 sichtbar. Wie man sehen kann, geben negative Werte immer sehr niedrige Fitnesswerte im Durchschnitt, aber auch Werte über 750. Der Wert `b` ergibt die besten Fitnesswerte (über 995 000) zwischen 100 und 700.

`b`	**Fitness** *(Durchschnitt)*	`b`	**Fitness** *(Durchschnitt)*	`b`	**Fitness** *(Durchschnitt)*
−100	530 547,3	1	531 012,2	250	1 047 474,0
−50	529 408,0	2	529 408,0	300	1 015 324,0
−20	529 408,0	3	625 566,0	400	1 039 747,2
−10	532 147,6	4	746 980,0	500	1 048 172,2
−7	529 408,0	5	782 218,6	600	1 027 253,4
−5	529 898,0	7	799 887,0	700	1 035 557,8
−4	529 408,0	10	854 743,2	750	529 408,0
−3	529 494,0	20	938 854,8	800	529 459,6
−2	529 408,0	50	978 905,6	900	529 408,0
−1	533 236,0	100	1 001 828,2	1000	529 408,0
0	529 408,0	200	996 009,8		

Tabelle 6.3: Vorstudie, Jenetics, Boltzmann-Selektor

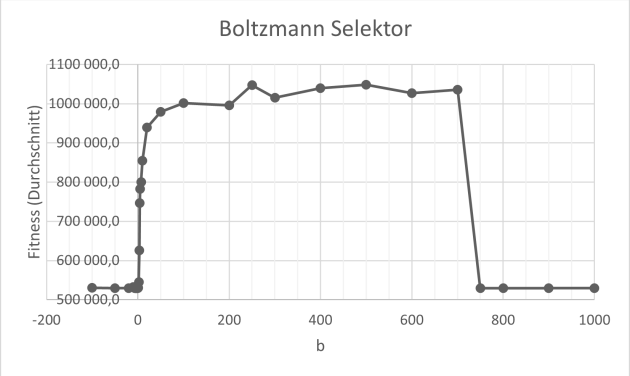

Abbildung 6.2: Vorstudie, Jenetics, Boltzmann-Selektor

Exponential-Rank-Selektor

Es kann der Parameter `c` verändert werden, je nachdem können gute oder schlechte Optimierungen generiert werden. Getestet wurden Werte zwischen 0 und 1, wobei 1 einen Fehler hervorruft. Die empfohlene Standardeinstellung ist 0,975 [40]. In der Nähe dieses Wertes wurden mehr Werte getestet. Ergebnisse sind in Tabelle 6.4 und Abbildung 6.3 sichtbar.

`c`	**Fitness** *(Durchschnitt)*	`c`	**Fitness** *(Durchschnitt)*	`c`	**Fitness** *(Durchschnitt)*
0	1 050 343,4	0,6	1 044 684,8	0,925	1 027 468,4
0,1	1 060 700,4	0,7	1 036 466,0	0,95	1 038 729,8
0,2	1 032 162,2	0,75	1 040 190,2	0,975	536 249,6
0,3	1 032 086,0	0,8	1 031 230,0	1	—
0,4	1 027 481,0	0,85	1 022 078,4		
0,5	1 029 114,6	0,9	1 021 081,2		

Tabelle 6.4: Vorstudie, Jenetics, Exponential-Rank-Selektor

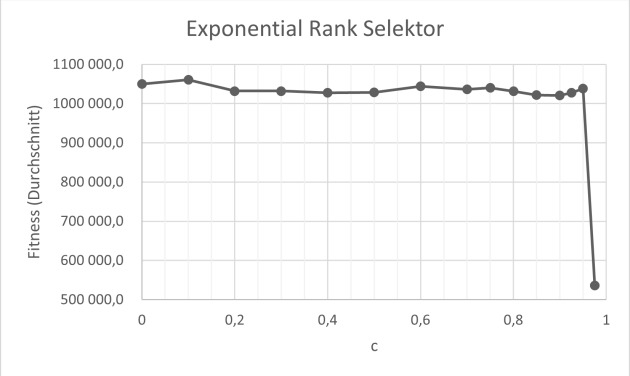

Abbildung 6.3: Vorstudie, Jenetics, Exponential-Rank-Selektor

Roulette-Wheel-Selektor

Für diesen Selektor gibt es keinen einstellbaren Parameter [40]. Daher wurde dieser Selektor zwanzigmal getestet. Der durchschnittliche Fitnesswert ist 540 382,5. Im Vergleich zu einigen anderen getesteten Selektoren ist dies ein schlechter Wert. Daher wird dieser Selektor nicht weiter untersucht und verwendet.

Stochastic-Universal-Selektor

Dieser Selektor hat ebenso keinen einstellbaren Parameter [40]. Daher wurde dieser Selektor auch zwanzigmal getestet. Der durchschnittliche Fitnesswert ist hier 534 710,7. Im Vergleich zu einigen anderen getesteten Selektoren ist dies ein schlechter Wert und noch schlechter als beim Roulette Wheel Selektor. Daher wird dieser Selektor nicht weiterverwendet.

Linear-Rank-Selektor

Hier kann der Parameter n^- verändert werden, welches eine Auswirkung auf die Wahrscheinlichkeit zur Auswahl der Individuen hat. Standardmäßig ist n^- 0,5 [40]. Es sind nur positive Werte und die Null möglich. Daher wurden Werte nahe 0 genauer getestet mit einigen größeren Werten Richtung 100, wie in Tabelle 6.5 und Abbildung

6.4 ersichtlich ist. In Abbildung 6.4 ist zu beachten, dass aufgrund der logarithmischen Darstellung der Nullwert nicht eingezeichnet wird. Man sieht hier eindeutig, dass keine Fitness über einer Million erreicht wird, jedoch ein n^--Wert von 0 noch den besten Fitnesswert im Durchschnitt erreicht.

n^-	**Fitness** *(Durchschnitt)*	n^-	**Fitness** *(Durchschnitt)*	n^-	**Fitness** *(Durchschnitt)*
0	978 615,4	0,5	531 240,8	4	529 408,0
0,05	957 015,8	0,75	529 408,0	5	529 408,0
0,1	897 667,2	1	533 076,6	7	529 408,0
0,15	772 907,4	1,5	529 432,2	10	529 408,0
0,2	585 543,2	2	531 322,4	100	529 408,0
0,25	555 538,6	3	529 408,0		

Tabelle 6.5: Vorstudie, Jenetics, Linear-Rank-Selektor

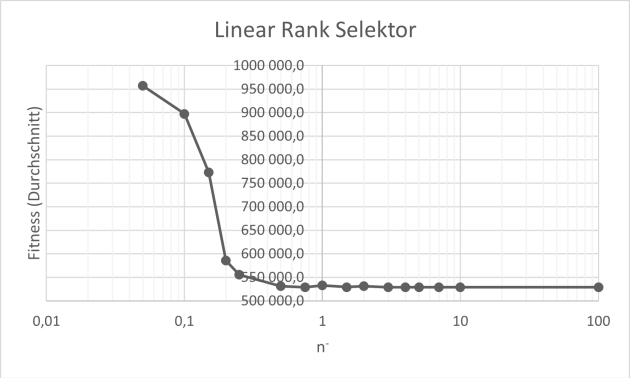

Abbildung 6.4: Vorstudie, Jenetics, Linear-Rank-Selektor

Truncation-Selektor

Hier werden die Individuen nach ihrer Fitness sortiert und die besten `n` Individuen werden ausgewählt. Je nach Einstellung werden mehr oder weniger Individuen ausgewählt. Die Standardeinstellung ist, dass `n` der größtmögliche Wert eines Integer-Objekts ist, 2 147 483 647, für eine möglichst große Auswahl [40]. Der Parameter muss größer oder gleich 1 sein. Es wurden Werte absteigend vom Maximalwert getestet. Ein `n`-Wert von 0

erzeugt einen Fehler. Ergebnisse sind in Tabelle 6.3 und Abbildung 6.2 dargestellt. Wie ersichtlich ist, hat der `n`-Wert kaum eine Auswirkung auf den Fitnesswert.

`n`	**Fitness** *(Durchschnitt)*	`n`	**Fitness** *(Durchschnitt)*	`n`	**Fitness** *(Durchschnitt)*
2 147 483 647	1 043 802,6	250 000 000	1 031 432,0	50	1 051 831,2
2 147 483 646	1 059 621,8	25 000 000	1 026 518,4	20	1 015 798,4
2 100 000 000	1 042 852,2	2 500 000	1 020 583,0	10	1 032 826,0
2 000 000 000	1 038 971,0	250 000	1 037 160,0	7	1 014 101,8
1 750 000 000	1 055 550,8	25 000	1 054 163,0	5	1 032 129,8
1 500 000 000	1 028 237,8	2500	1 040 598,6	4	1 047 473,0
1 250 000 000	1 026 770,8	1000	1 050 369,8	3	1 038 918,8
1 000 000 000	1 036 486,6	500	1 065 767,0	2	1 021 507,4
750 000 000	1 035 882,0	250	1 046 293,2	1	1 040 335,2
500 000 000	1 034 691,4	100	1 047 415,2	0	—

Tabelle 6.6: Vorstudie, Jenetics, Truncation-Selektor

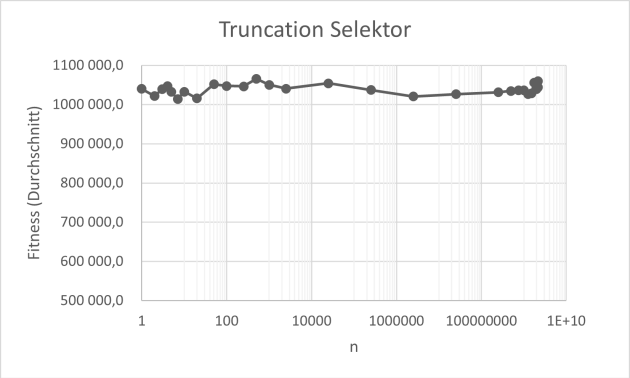

Abbildung 6.5: Vorstudie, Jenetics, Truncation-Selektor

Zusammenfassung Selektoren

In den vorigen Abschnitten wurden die verschiedenen Selektoren mit diversen Parametern getestet. Anhand der Testdaten, der minimalen und maximalen Fitnesswerte pro einzelne Einstellungen und der empfohlenen Standardeinstellung wurde die Selektoren-Auswahl für die folgenden Parametertests getroffen: Truncation-Selektor mit einem Parameter von 500 mit einem Fitnesswert von 1 065 767 im Durchschnitt. Für die Hauptstudie

wird jedoch der Tournament-Selektor mit einem Parameter von 50, 10 oder 3 verwendet, da dieser Selektor auch der empfohlene Selektor laut Handbuch [38] ist. Somit sollen die unterschiedlich generierten Daten der Hauptstudie bessere Ergebnisse liefern. Der Tournament-Selektor mit dem Parameter von 50 ergibt einen Fitnesswert von 1 008 958,2. Ein Parameter von 10 beim gleichen Selektor erreicht 1 029 006,4 und der Parameter 3 1 024 998,3. Diese Parameter ergeben Fitnesswerte im Durchschnitt von über einer Million und unterscheiden sich nicht viel voneinander. Durch die Auswahl an Tournament-Selektor-Parameter soll im Verlauf der Hauptstudie ausgelesen werden können, welche Einstellung tatsächlich zu bevorzugen ist.

6.2.2 Jenetics: Auswahl der anderen Parameterwerte

Mit der vorhin besprochenen Einstellung, die schon für die Selektoren-Tests verwendet wurde, werden zusätzlich die Selektoren auf den Truncation-Selektor mit dem Parameter 500 eingestellt. Es werden die Bevölkerungsgröße, Fraktion der Nachkommen, Rekombinationswahrscheinlichkeit für Mutator (`mutatorAlterProbability`), Rekombinationswahrscheinlichkeit für Crossover (`crossOverAlterProbability`) und maximales Alter der Phenotypen getestet, wobei wiederum mindestens fünf Tests pro Einstellung durchgeführt werden.

Bevölkerungsgröße

Es werden verschiedene Bevölkerungsgrößen (`popSize`) für die einzelnen Generationen getestet, wobei nur positive Werte möglich sind und Null einen Fehler liefert. Die Ergebnisse sind in Tabelle 6.7 und Abbildung 6.6 ersichtlich. Zu beachten ist, dass bei großen Bevölkerungsgrößen die eingestellte Laufzeit pro Durchlauf oft nicht mehr eingehalten wird. So dauert der Testdurchlauf für eine Bevölkerungsgröße von 100 000 etwa fünf Minuten. Bevölkerungsgrößen zwischen 30 und 500 sind daher zu bevorzugen.

popSize	**Fitness** *(Durchschnitt)*	popSize	**Fitness** *(Durchschnitt)*	popSize	**Fitness** *(Durchschnitt)*
0	—	40	1 031 054,4	150	1 020 064,4
1	529 408,0	50	1 039 444,6	250	1 055 219,8
2	529 408,0	60	1 016 937,6	500	1 066 533,8
5	543 103,0	70	1 052 236,6	1000	1 013 868,0
10	643 069,8	80	1 017 885,8	5000	1 014 999,0
20	969 451,8	90	1 016 064,8	10 000	934 060,4
30	1 036 111,2	100	1 047 034,2	100 000	663 821,6

Tabelle 6.7: Vorstudie, Jenetics, Bevölkerungsgröße

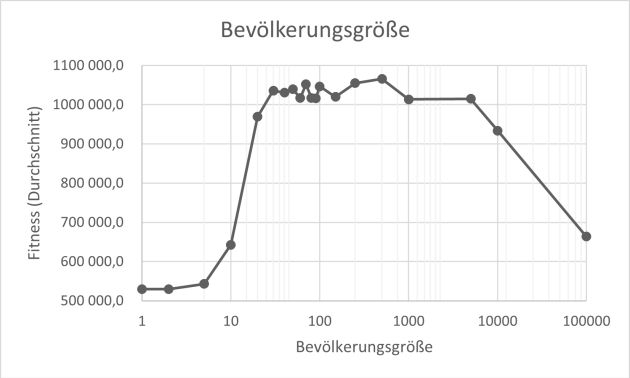

Abbildung 6.6: Vorstudie, Jenetics, Bevölkerungsgröße

Fraktion der Nachkommen

Hierüber wird eingestellt, wie viel Prozent der Nachkommen für die nächste Generation überleben. Daher sind nur Werte zwischen 0 und 1 möglich, wobei 1 einen Fehler liefert. Die Ergebnisse sind in Tabelle 6.8 und Abbildung 6.7 zu sehen. Der Fitnesswert ist im Durchschnitt bei einer Fraktion von 0,7 am höchsten. Dieser Wert wird für die weitere Nutzung in der Hauptstudie empfohlen. Die zwei höchsten Parameterwerte (0,6 und 0,7) wurden zusätzlich mit einer Bevölkerungsgröße von 70 (einem der empfohlenen Werte) getestet und hier erzeugt der Fraktionswert 0,7 einen höheren Fitnesswert im Durchschnitt.

popSize	**Fitness** *(Durchschnitt)*	popSize	**Fitness** *(Durchschnitt)*	popSize	**Fitness** *(Durchschnitt)*
0	529 408,0	0,4	1 031 723,0	0,8	991 927,6
0,1	868 127,4	0,5	1 038 241,4	0,9	543 660,8
0,2	929 677,8	0,6	1 043 201,6	1	—
0,3	1 000 587,2	0,7	1 087 419,6		

Tabelle 6.8: Vorstudie, Jenetics, Bevölkerungsgröße

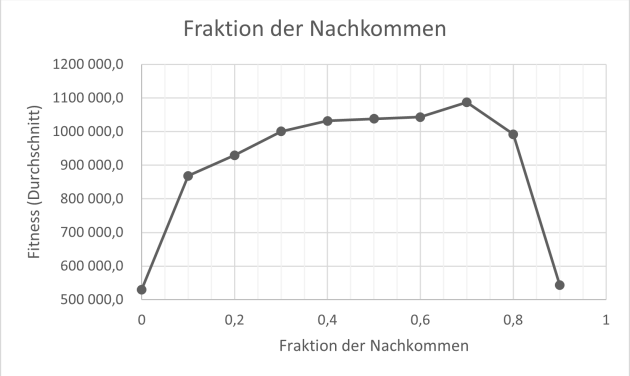

Abbildung 6.7: Vorstudie, Jenetics, Fraktion der Nachkommen

Rekombinationswahrscheinlichkeit (Mutator)

Der Parameter `mutatorProbability` (abgekürzt `mutProb`) für den Swap Mutator bestimmt die Wahrscheinlichkeit der Rekombination im Swap Mutator. Es sind Werte zwischen 0 und 1 möglich, wobei bei 1 ein Fehler generiert wird. Die Standardeinstellung ist 0,2 [38]. Die getesteten Werte sind in Tabelle 6.9 und Abbildung 6.8 ersichtlich. Ein Wert zwischen 0,05 und 0,3 für `mutatorProbability` ergibt Fitnesswerte im Durchschnitt von über einer Million. Die Kernwerte (0,1 bis 0,2) wurden mit einer Bevölkerungsgröße von 70 und einer Fraktion von Nachkommen von 0,7 erneut getestet. Hier wurde ebenso der Wert von 0,15 getestet, welcher den höchsten Fitnesswert im Durchschnitt bringt.

mutProb	**Fitness** *(Durchschnitt)*	mutProb	**Fitness** *(Durchschnitt)*	mutProb	**Fitness** *(Durchschnitt)*
0	608 199,4	0,25	1 024 257,8	0,6	805 477,4
0,05	1 047 623,8	0,3	1 020 590,8	0,7	728 659,0
0,1	1 049 253,2	0,35	989 245,2	0,8	703 185,4
0,15	1 057 336,4	0,4	973 716,8	0,9	537 993,8
0,2	1 039 226,4	0,5	911 673,0	1	—

Tabelle 6.9: Vorstudie, Jenetics, mutatorProbability

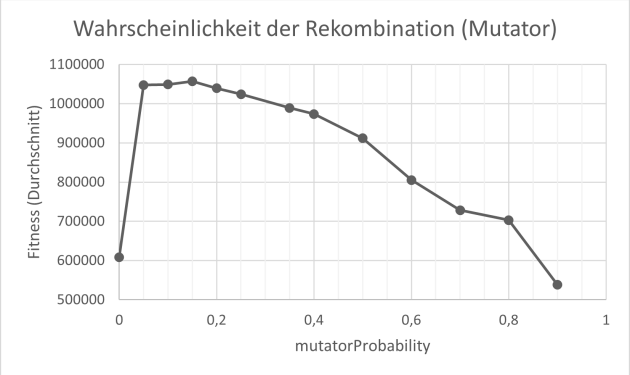

Abbildung 6.8: Vorstudie, Jenetics, Wahrscheinlichkeit der Rekombination im Swap Mutator

Rekombinationswahrscheinlichkeit (Crossover)

Für Partially Matched Crossover wird ebenso die Wahrscheinlichkeit für die Rekombination `crossoverProbability` (abgekürzt `crosProb`) eingestellt. Es sind, ähnlich zur Rekombinationswahrscheinlichkeit vom Mutator, nur Werte zwischen 0 und 1 möglich, wobei 1 einen Fehler generiert. Der Wert 0,35 wurde extra getestet, da es die empfohlene Einstellung laut Handbuch [38] ist. Die Ergebnisse sind wiederum in Tabelle 6.10 und Abbildung 6.10 abgebildet. Zwischen 0,8 und 0,95 für `crossoverProbability` erreicht der Fitnesswert im Durchschnitt Werte über 1 110 000. Diese wurden separat mit einer Bevölkerungsgröße von 70, eine Fraktion für Nachkommen von 0,7 und einer Rekombinationswahrscheinlichkeit vom Mutator von 0,15 getestet. Hier ist der Fitnesswert ebenso für `crossoverProbability` von 0,9 der beste.

crosProb	**Fitness** *(Durchschnitt)*	crosProb	**Fitness** *(Durchschnitt)*	crosProb	**Fitness** *(Durchschnitt)*
0	529 408,0	0,4	1 044 600,4	0,85	1 115 162,8
0,1	902 448,8	0,5	1 074 363,4	0,9	1 131 784,2
0,2	998 754,6	0,6	1 094 508,0	0,95	1 115 160,6
0,3	1 027 731,4	0,7	1 108 004,6	1	—
0,35	1 024 257,8	0,8	1 128 835,2		

Tabelle 6.10: Vorstudie, Jenetics, crossoverProbability

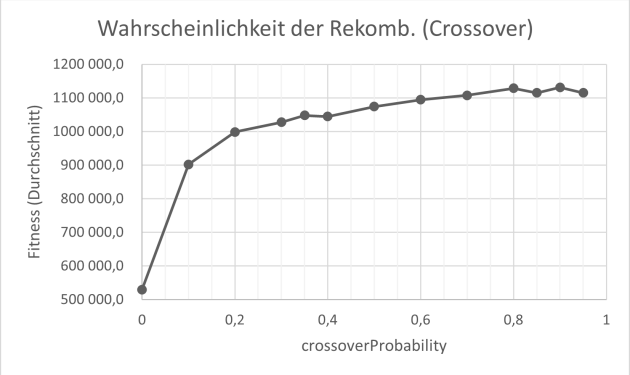

Abbildung 6.9: Vorstudie, Jenetics, Wahrscheinlichkeit der Rekombination im Partially Matched Crossover

maximales Alter der Phenotypen

Hierfür wurden die Einstellungen anhand der vorigen Testdurchläufe angepasst. Es wird eine Bevölkerungsgröße von 70, Fraktion an Nachkommen von 0,7, Rekombinationswahrscheinlichkeit vom Mutator von 0,15 und Rekombinationswahrscheinlichkeit vom Crossover von 0,9 verwendet. Der Parameter für das maximale Alter der Phenotypen `maxAgePhenotype` (abgekürzt `maxAgePh`) muss positiv sein, 0 erzeugt einen Fehler. Die Standardeinstellung für `maxAgePhenotype` beträgt 70 [38]. Ergebnisse sind in Tabelle 6.11 und Abbildung 6.11 ersichtlich. Ein maximales Alter von Phenotypen von 80 ergibt den größten Durchschnittswert an Fitness.

maxAgePh	**Fitness** *(Durchschnitt)*	maxAgePh	**Fitness** *(Durchschnitt)*	maxAgePh	**Fitness** *(Durchschnitt)*
0	—	40	1 134 580,8	150	1 156 163,0
1	540 675,0	50	1 166 548,2	250	1 133 610,6
2	534 598,0	60	1 142 284,8	500	1 152 967,4
5	578 342,2	70	1 129 205,8	1000	1 163 654,8
10	1 114 382,4	80	1 170 404,2	5000	1 158 992,2
20	1 154 745,0	90	1 145 552,6		
30	1 163 052,0	100	1 157 110,4		

Tabelle 6.11: Vorstudie, Jenetics, maximales Alter der Phenotypen

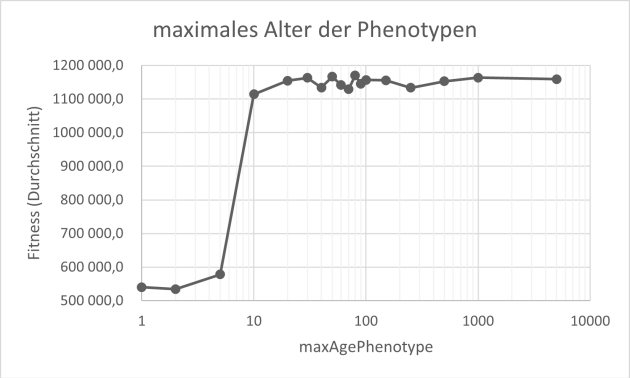

Abbildung 6.10: Vorstudie, Jenetics, maximales Alter der Phenotypen

Zusammenfassung anderer Parameter

Für die Bevölkerungsgröße sollen zwei Werte für die Tests ausgewählt werden: Jeweils ein Wert unter 100 und ein Wert über 100. Ein Wert von 70 ergibt einen Fitnesswert von 1 052 236,6 im Durchschnitt und ein Wert von 500 1 066 533,8, was jeweils die größten getesteten Werte unter und über 100 sind. Es wird nur ein Wert für die Fraktion der Nachkommen für die Hauptstudie verwendet. Der Wert von 0,7 ergab bei den vorigen Tests den höchsten Fitnesswert, weswegen er für alle Hauptstudientests verwendet wird, sowie es beim maximalen Alter von Phenotypen von 80 ist. Bei den Rekombinationswahrscheinlichkeiten werden jedoch auch zwei Werte getestet, um den besseren Wert für die Hauptstudie herauszufinden. Dabei soll jeweils ein Wert unter und über 0,5 sein.

Für die Rekombinationswahrscheinlichkeit vom Mutator ergibt der Wert 0,15 einen Fitnesswert von 1 057 336,4. Da die Fitness bei höheren Parameterwerten auf niedrigere Fitnesswerte abfällt, wird der niedrigste getestete Wert über 0,5 (0,6) als zweiter Wert verwendet. Hierbei wurde ein Fitnesswert von 805 477,4 erzielt. Die empfohlene Standardeinstellung vom Mutator liegt bei 0,2 und wird nicht für die Hauptstudie verwendet, weil als Wert unter 0,5 bereits 0,15 gewählt wurde. Bei den Wahrscheinlichkeiten für das Crossover ergibt ein Wert von 0,9, den größten Fitnesswert mit den anderen gewählten Einstellungen. Dies wurde bei den weiter oben beschriebenen Testergebnissen erläutert. Als zweiter Wert wird 0,35 verwendet, weil dies die vorige empfohlene Standardeinstellung ist.

Zusammengefasst werden für die Hauptstudie folgende Werte verwendet:

- Bevölkerungsgröße: 70 und 500
- Fraktion an Nachkommen: 0,7
- Rekombinationswahrscheinlichkeit vom Mutator: 0,15 und 0,6
- Rekombinationswahrscheinlichkeit vom Crossover: 0,35 und 0,9
- maximales Alter von Phenotypen: 80

Für Bevölkerungsgröße soll in der Hauptstudie mit untersucht werden, inwiefern diese sich auf das Ergebnis auswirken. Zwei verschiedene Werte bei beiden Rekombinationswahrscheinlichkeiten werden auch in der Hauptstudie dazu verwendet, um herauszufinden, inwiefern sich ein Wert unter und über 0,5 auf das Ergebnis auswirkt. Eine genaue Aufschlüsselung, welche Konfigurationen in der Hauptstudie verwendet werden, ist in Abschnitt 7.3 aufgelistet.

6.2.3 OptaPlanner

Für die Algorithmen von OptaPlanner wurden zuerst die einstellbaren Parameter von Hill Climbing, Tabu Search, Simulated Annealing, Late Acceptance, Great Deluge, Step Counting Hill Climbing und Strategic Oscillation manuell durchgetestet. Danach wurde noch der Parameter `acceptedCountLimit` angepasst. Die Tests erfolgten auf Basis der

empfohlenen Werte, wie sie das Handbuch von OptaPlanner nutzt [26]. Die Testdurchläufe wurden mit jeweils 60 Sekunden durchgeführt.

Ebenso erfolgten die Tests auf Basis einer anderen Kostenfunktion, als in der Hauptstudie und in der Vorstudie von Jenetics verwendet wurde. Hierbei hat der Slot zum ersten Slot den geringsten möglichen Wert. Die Slots zwischen dem vorigen Zeitpunkt und dem Zeitpunkt `ScheduledTime` liegen auf einer linearen Funktion zwischen dem geringsten Wert und dem geringsten Wert plus ein Fünftel der Distanz zwischen dem geringsten und dem größten Wert. Der Slot zum Zeitpunkt `ScheduledTime` hat den Wert geringster Wert plus zwei Fünftel der Distanz zwischen dem geringsten und größten Wert. Die Slots zwischen `ScheduledTime` und `TimeNotBefore` liegen auf einer linearen steigenden Funktion zwischen dem geringsten Wert plus zwei Fünftel der Distanz zwischen dem geringsten und größten Wert und dem geringsten Wert plus drei Fünftel der Distanz zwischen dem geringsten und größten Wert. Der Slot zum Zeitpunkt `TimeNotBefore` hat den Wert geringster Wert plus vier Fünftel der Distanz zwischen dem geringsten und größten Wert. Die Slots zwischen `TimeNotBefore` und `TimeWished` liegen auf einer linearen Funktion zwischen dem geringsten Wert plus vier Fünftel der Distanz zwischen dem geringsten und größten Wert und dem größten Wert. Der Slot zum Zeitpunkt `TimeWished` hat den größtmöglichen Wert. Die Slots zwischen `TimeWished` und `TimeNotAfter` liegen auf einer abfallenden linearen Funktion zwischen dem größten Wert und dem geringsten Wert plus vier Fünftel der Distanz zwischen dem geringsten und größten Wert. Der Slot zum Zeitpunkt `TimeNotAfter` hat den Wert geringster Wert plus vier Fünftel der Distanz zwischen dem geringsten und größten Wert. Die Slots zwischen `TimeNotAfter` und dem letztem Slot liegen auf einer linearen fallenden Funktion zwischen dem geringsten Wert plus drei Fünftel der Distanz zwischen dem geringsten und größten Wert und dem geringsten Wert plus zwei Fünftel der Distanz zwischen dem geringsten und größten Wert.

In den folgenden Abschnitten werden jeweils die getesteten Parameter und ihre Ergebnisse dargestellt. Zusätzlich werden die gewählten Einstellungen für die Hauptstudie aufgezeigt. Jeder Algorithmustyp wird in der Hauptstudie nur einmal getestet mit einer Einstellung der Parameter.

Hill Climbing

Für Hill Climbing gibt es keine einstellbaren Parameter. Daher wurde direkt der Parameter `acceptedCountLimit` mit drei verschiedenen Einstellungen getestet, wie in Tabelle 6.12 und Abbildung 6.11 ersichtlich ist.

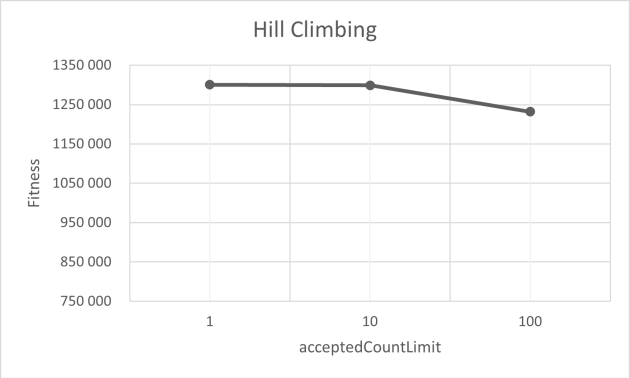

Abbildung 6.11: Vorstudie, OptaPlanner, Hill Climbing, acceptedCountLimit

`acceptedCountLimit`	*Notiz*	**Fitness**
1		1 300 634
10		1 298 940
100		1 231 962
—	*Default Einstellung*	1 300 634

Tabelle 6.12: Vorstudie, OptaPlanner, Hill Climbing

Wie man hier sehen kann, sinkt der Fitnesswert von 1 300 634 bei `acceptedCountLimit` von 1 auf 1 231 962 bei `acceptedCountLimit` von 100 ab. Die Default-Einstellung, wo `acceptedCountLimit` nicht direkt definiert wird, schafft ebenso einen Fitnesswert von 1 300 634. Daher wird bei Hill Climbing für die Hauptstudie die Default-Einstellung ohne weitere eingestellte Parameter verwendet.

Tabu Search

Der Algorithmus Tabu Search hat bei OptaPlanner viele verschiedene Einstellungsmöglichkeiten. Es wurden die Parameter `entityTabuSize`, `entityTabuRatio`, `valueTabuSize`

, moveTabuSize, undoMoveTabuSize alleine und nicht in Kombination untereinander getestet. Der Parameter valueTabuRatio wurde nicht getestet, da über valueTabuSize und entityTabuRatio ähnliche Testbereiche abgedeckt sind. Es werden bei den TabuSize-Werten jeweils die Werte 0, 1, 2, 3, 4, 5, 6, 7, 10, 15, 25, 50, 100, 1000 und teilweise noch 100 000 getestet. Der Default-Wert ist jeweils 7 laut OptaPlanner-Handbuch [26], weswegen kleine Werte in kürzerem Abstand getestet werden, größere Werte jedoch in größerem Abstand, um eventuelle Tendenzen erkennen zu können. Bei entityTabuRatio werden die Werte wie in Tabelle 6.14 verwendet. Die Werte zwischen 0,01 und 0,1 werden genauer untersucht, da der Default-Wert 0,02 ist. Ebenso wird hier acceptedCountLimit mit dem empfohlenen Default-Wert untersucht, um hier bessere und schlechtere Werte für acceptedCountLimit herauszufinden. Bei keiner Einstellung bezüglich acceptedCount-Limit und auch keiner weiteren Einstellung bezüglich weiterer Parameter ergibt sich ein Fitnesswert von 1 302 214.

Die Tests für entityTabuSize (abgekürzt entTSize) sind wie in Tabelle 6.13 und Abbildung 6.12 abgebildet (getestet mit acceptedCountLimit von 1000).

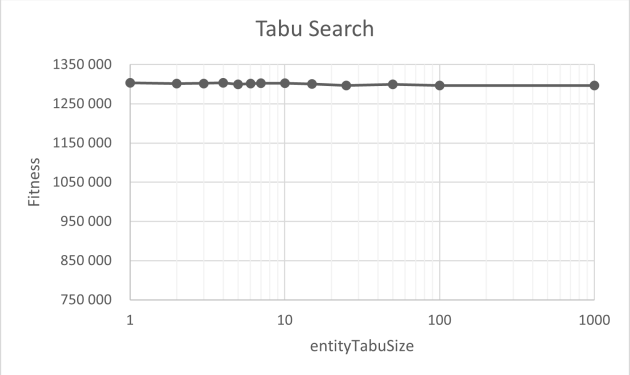

Abbildung 6.12: Vorstudie, OptaPlanner, Tabu Search, entityTabuSize

Der entityTabuSize-Wert von Null ergab einen Fehler. Zwischen dem besten und schlechtestem Fitnesswert der Tabelle 6.13 gibt es einen Unterschied von 0,55%. Daher wird auch davon ausgegangen, dass dieser Parameter nicht so bedeutsam ist. Sollte entityTabuSize als Parameter für Tabu Search eingestellt werden, wird der entityTabu-Size-Wert 4 empfohlen, weil dies hier der größte Fitnesswert ist.

entTSize	Fitness	entTSize	Fitness	entTSize	Fitness
0	—	5	1 299 988	25	1 296 792
1	1 303 046	6	1 301 512	50	1 299 231
2	1 301 271	7	1 302 779	100	1 296 798
3	1 302 010	10	1 302 214	1000	1 296 294
4	1 303 494	15	1 300 307		

Tabelle 6.13: Vorstudie, OptaPlanner, Tabu Search, `entityTabuSize`

Die Tests für `entityTabuRatio` (abgekürzt `entTRatio`) sind in Tabelle 6.14 und Abbildung 6.13 abgebildet. Die Tests wurden mit `acceptedCountLimit` von 1000 durchgeführt.

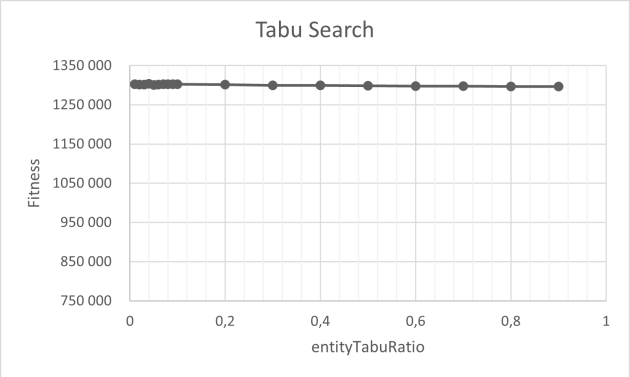

Abbildung 6.13: Vorstudie, OptaPlanner, Tabu Search, entityTabuRatio

entTRatio	Fitness	entTRatio	Fitness	entTRatio	Fitness
0	—	0,07	1 302 779	0,5	1 298 499
0,01	1 303 046	0,08	1 302 252	0,6	1 297 961
0,02	1 301 271	0,09	1 302 482	0,7	1 297 402
0,03	1 302 010	0,1	1 302 214	0,8	1 296 492
0,04	1 303 494	0,2	1 301 408	0,9	1 296 798
0,05	1 300 268	0,3	1 299 730	1	—
0,06	1 301 512	0,4	1 299 244		

Tabelle 6.14: Vorstudie, OptaPlanner, Tabu Search, `entityTabuRatio`

Ein `entityTabuRatio`-Wert von Null oder Eins ergibt einen Fehler. Zwischen dem besten und schlechtestem Fitnesswert der Tabelle 6.14 gibt es einen Unterschied von 0,54%.

Daher wird auch davon ausgegangen, dass dieser Parameter nicht so bedeutsam ist. Sollte `entityTabuRatio` als Parameter für Tabu Search eingestellt werden, wird der `entityTabuRatio`-Wert 0,04 empfohlen, weil das der größte Fitnesswert ist.

Die Tests für `valueTabuSize` (abgekürzt `valTSize`) sind in Tabelle 6.15 und Abbildung 6.14 abgebildet. Die Tests wurden mit `acceptedCountLimit` von 1000 durchgeführt.

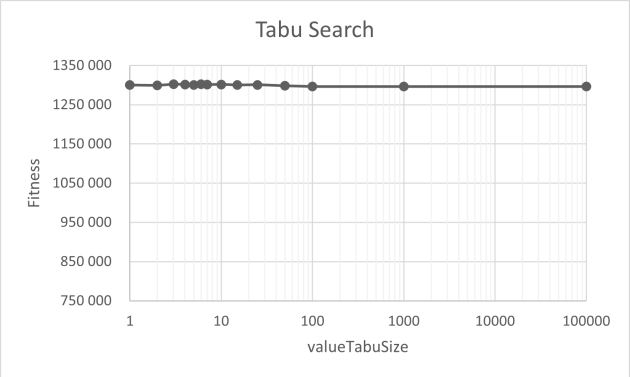

Abbildung 6.14: Vorstudie, OptaPlanner, Tabu Search, valueTabuSize

`valTSize`	**Fitness**	`valTSize`	**Fitness**	`valTSize`	**Fitness**
0	—	5	1 299 926	25	1 300 748
1	1 300 597	6	1 301 995	50	1 298 565
2	1 299 271	7	1 301 335	100	1 296 537
3	1 302 792	10	1 301 798	1000	1 296 735
4	1 301 837	15	1 300 345	100 000	1 296 537

Tabelle 6.15: Vorstudie, OptaPlanner, Tabu Search, `valueTabuSize`

Der `valueTabuSize`-Wert von Null ergab einen Fehler. Zwischen dem besten und schlechtestem Fitnesswert der Tabelle 6.15 gibt es einen Unterschied von 0,48%. Daher wird auch davon ausgegangen, dass dieser Parameter nicht so bedeutsam ist. Sollte `valueTabuSize` als Parameter für Tabu Search eingestellt werden, wird der `valueTabuSize`-Wert 3 empfohlen, da das hier der größte Fitnesswert ist.

Die Tests für `moveTabuSize` (abgekürzt `movTSize`) sind in Tabelle 6.16 und Abbildung 6.15 abgebildet. Die Tests wurden mit `acceptedCountLimit` von 1000 durchgeführt.

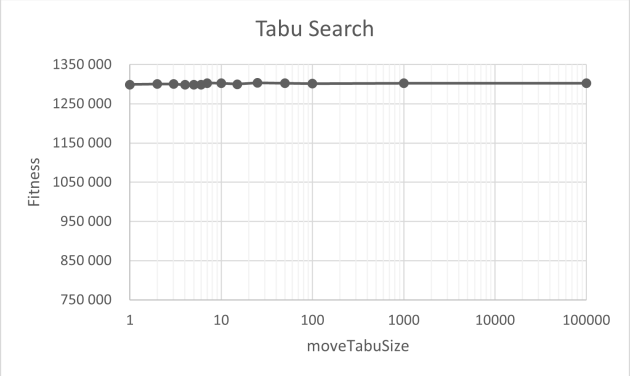

Abbildung 6.15: Vorstudie, OptaPlanner, Tabu Search, moveTabuSize

movTSize	**Fitness**	movTSize	**Fitness**	movTSize	**Fitness**
0	—	5	1 298 959	25	1 303 718
1	1 298 965	6	1 298 930	50	1 302 684
2	1 300 183	7	1 302 902	100	1 301 447
3	1 300 170	10	1 302 047	1000	1 302 714
4	1 298 940	15	1 300 027	100 000	1 302 714

Tabelle 6.16: Vorstudie, OptaPlanner, Tabu Search, `moveTabuSize`

Der `moveTabuSize`-Wert von Null ergab einen Fehler. Zwischen dem besten und schlechtestem Fitnesswert der Tabelle 6.16 gibt es einen Unterschied von 0,37%. Daher wird auch davon ausgegangen, dass dieser Parameter nicht so bedeutsam ist. Sollte `moveTabuSize` als Parameter für Tabu Search eingestellt werden, wird der `moveTabuSize`-Wert 3 empfohlen, da hier der größte Fitnesswert erreicht werden konnte.

Die Tests für `undoMoveTabuSize` (abgekürzt `undMTSize`) sind in Tabelle 6.17 und Abbildung 6.16 abgebildet. Die Tests wurden mit `acceptedCountLimit` von 1000 durchgeführt.

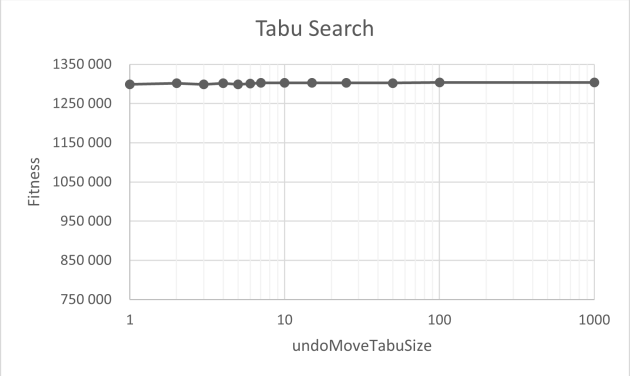

Abbildung 6.16: Vorstudie, OptaPlanner, Tabu Search, undoMoveTabuRatio

undMTSize	**Fitness**	undMTSize	**Fitness**	undMTSize	**Fitness**
0	—	5	1 298 926	25	1 302 700
1	1 298 753	6	1 301 134	50	1 302 326
2	1 301 897	7	1 302 489	100	1 303 972
3	1 298 956	10	1 302 470	1000	1 303 641
4	1 301 646	15	1 302 995		

Tabelle 6.17: Vorstudie, OptaPlanner, Tabu Search, `undoMoveTabuSize`

Der `undoMoveTabuSize`-Wert von Null ergab einen Fehler. Zwischen dem besten und schlechtestem Fitnesswert der Tabelle 6.17 gibt es einen Unterschied von 0,40%. Daher wird auch davon ausgegangen, dass dieser Parameter nicht so bedeutsam ist. Sollte `undoMoveTabuSize` als Parameter für Tabu Search eingestellt werden, wird der `undoMoveTabuSize`-Wert 3 empfohlen, da das der größte Fitnesswert ist.

Zusätzlich wurde noch `acceptedCountLimit` (abgekürzt `accCouLim`) getestet, wie in Tabelle 6.18 und Abbildung 6.17 ersichtlich ist. Hierfür wurde „valueTabuRatio“ mit dem empfohlenen Default-Wert (0,02)[26] verwendet.

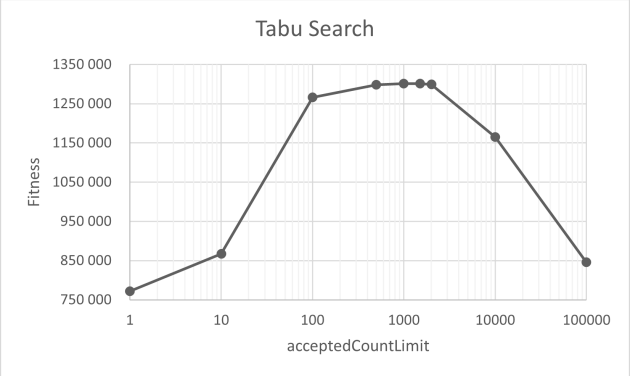

Abbildung 6.17: Vorstudie, OptaPlanner, Tabu Search, acceptedCountLimit

accCouLim	**Fitness**	accCouLim	**Fitness**	accCouLim	**Fitness**
1	772 315	500	1 298 153	2000	1 299 519
10	867 480	1000	1 301 271	10 000	1 165 357
100	1 266 557	1500	1 301 249	100 000	845 623

Tabelle 6.18: Vorstudie, OptaPlanner, Tabu Search, `acceptedCountLimit`

Da der `acceptedCountLimit`-Wert einen größeren Einfluss auf den Fitnesswert hat, er jedoch zwischen `acceptedCountLimit`-Wert von 500 und 2000 einen stabilen Wert um etwa 1 300 000 schafft, wird der empfohlene `acceptedCountLimit`-Wert von 1000 (laut Handbuch von OptaPlanner [26]) tatsächlich für die Hauptstudie verwendet. Es werden keine weiteren Einstellungen getätigt für die Hauptstudie und daher bleiben die empfohlenen Standardeinstellungen eingestellt.

Simulated Annealing

Der Algorithmus Simulated Annealing benötigt auch in der einfachen Einstellung den Parameter `simulAnnealStartTemp`. Hier wurden nur unterschiedliche Werte für den soft-Wert von HardSoftScore getestet und der hard-Wert war durchgängig auf 0. Der empfohlene Wert laut Handbuch [26] war 0hard/100soft. Nach den Tests (siehe Tabelle 6.19 und Abbildung 6.18) zeigt sich, dass die Werte zwischen 0hard/100soft und 0hard/2000soft stabil sind (Unterschiede von nur 0,26%). Daher wird der beste getestet Wert von 0hard/500soft für die Hauptstudie verwendet. Außerdem zeigt sich bei Tests mit

`acceptedCountLimit` (abgekürzt `accCouLim`, siehe Tabelle 6.20 und Abbildung 6.19), das hier kaum Unterschiede zwischen den Werten 1, 10, 50, 100, 500, 1000, 100 000 erkennbar sind (0,02% Unterschied zwischen bestem und schlechtestem Wert). Daher wird hier der empfohlene Wert 1 (laut Handbuch [26]) auch für die Hauptstudie verwendet.

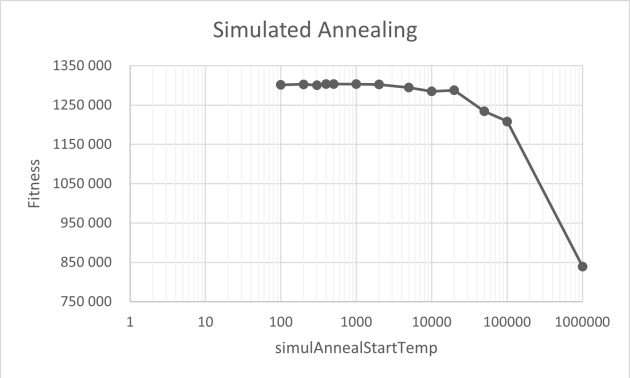

Abbildung 6.18: Vorstudie, OptaPlanner, Simulated Annealing, simulAnnealStartTemp

`simulAnnealStartTemp`	**Fitness**	`simulAnnealStartTemp`	**Fitness**
0hard/0soft	1 290 809	0hard/2000soft	1 302 594
0hard/100soft	1 301 603	0hard/5000soft	1 294 988
0hard/200soft	1 302 991	0hard/10000soft	1 284 972
0hard/300soft	1 300 617	0hard/20000soft	1 287 880
0hard/400soft	1 304 014	0hard/50000soft	1 234 209
0hard/500soft	1 304 020	0hard/100000soft	1 208 617
0hard/1000soft	1 303 746	0hard/1000000soft	838 825

Tabelle 6.19: Vorstudie, OptaPlanner, Simulated Annealing, `simulAnnealStartTemp`

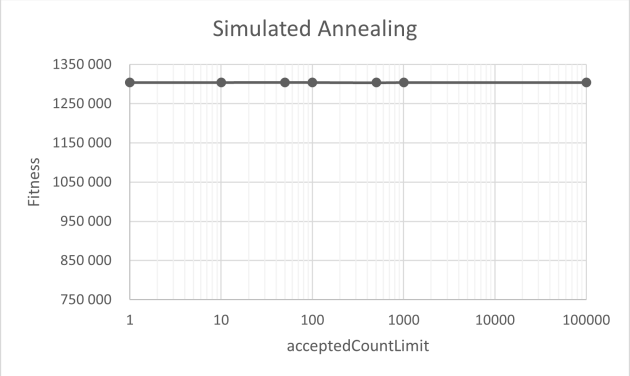

Abbildung 6.19: Vorstudie, OptaPlanner, Simulated Annealing, acceptedCountLimit

accCouLim	**Fitness**	accCouLim	**Fitness**	accCouLim	**Fitness**
1	1 304 018	100	1 304 016	100 000	1 304 020
10	1 304 085	500	1 303 776		
50	1 304 087	1000	1 304 051		

Tabelle 6.20: Vorstudie, OptaPlanner, Simulated Annealing, `acceptedCountLimit`

Late Acceptance

Bei Late Acceptance fällt bei den Tests mit unterschiedlichen Werten von `lateAcceptanceSize` (abgekürzt `latAccSize`) auf, dass hier Fitnesswerte bis zu 1 303 234 möglich sind, obwohl die Default-Einstellung (ohne eingestellte Parameter) nur einen Fitnesswert von 1 035 441 erreicht. Die Tests werden in Tabelle 6.21 und Abbildung 6.20 abgebildet. Bei einem Wert von 0 wird ein Fehler generiert. Nach den Tests ist klar, dass der größte Wert zwischen `lateAcceptanceSize` von 10 und 60 erreicht wird (Unterschiede von nur 0,49%). Bei höheren Werten (70 bis 10000) zeigt sich, dass der Fitnesswert bis auf 917265 absinkt. Für die Hauptstudie wird daher der `lateAcceptanceSize`-Wert von 50 verwendet. Zusätzlich wird `acceptedCountLimit` getestet, wie in Tabelle 6.22 und Abbildung 6.21 ersichtlich ist. Es zeigt sich bei den Tests mit `acceptedCountLimit` (abgekürzt `accCouLim`), dass hier Unterschiede zwischen den Werten erkennbar sind (38,5% Unterschied zwischen bestem und schlechtestem Wert). Daher wird hier der empfohlene Wert 1 (laut Handbuch [26]) nicht für die Hauptstudie verwendet. Zwischen

den Werten 3 und 9 sind die Werte stabil (0,41% Unterschied zwischen bestem und schlechtestem Wert) mit dem besten Fitnesswert bei `acceptedCountLimit`-Wert von 4. Daher wird dieser Wert auch für die Hauptstudie verwendet.

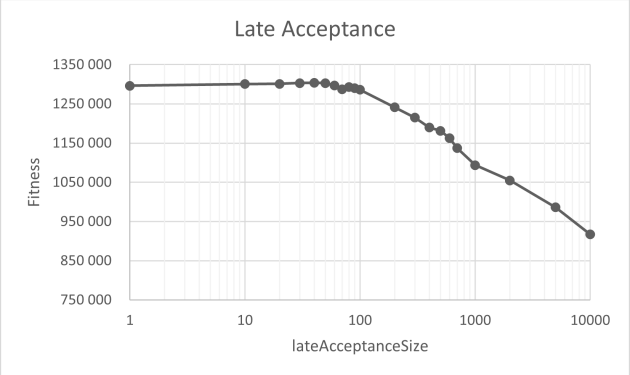

Abbildung 6.20: Vorstudie, OptaPlanner, Late Acceptance, lateAcceptanceSize

`latAccSize`	**Fitness**	`latAccSize`	**Fitness**	`latAccSize`	**Fitness**
0	—	70	1 287 671	600	1 162 629
1	1 296 231	80	1 293 348	700	1 137 316
10	1 300 847	90	1 289 713	1000	1 093 817
20	1 301 197	100	1 285 964	2000	1 054 908
30	1 303 067	200	1 241 286	5000	986 480
40	1 303 309	300	1 215 203	10000	917 265
50	1 303 234	400	1 189 698		
60	1 296 923	500	1 181 482		

Tabelle 6.21: Vorstudie, OptaPlanner, Late Acceptance, `lateAcceptanceSize`

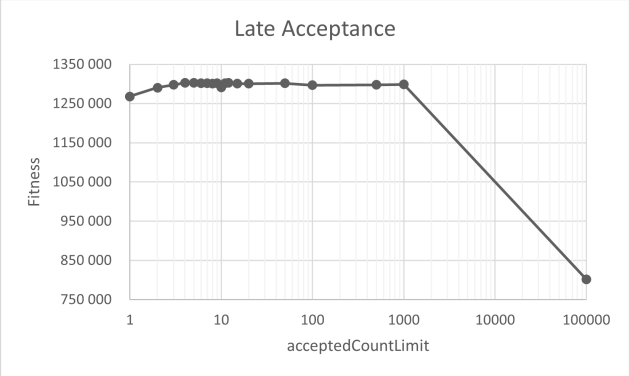

Abbildung 6.21: Vorstudie, OptaPlanner, Late Acceptance, acceptedCountLimit

accCouLim	**Fitness**	accCouLim	**Fitness**	accCouLim	**Fitness**
1	1 268 076	8	1 301 274	50	1 301 869
2	1 290 423	9	1 302 259	100	1 296 865
3	1 297 842	10	1 291 308	500	1 298 019
4	1 303 234	11	1 301 848	1000	1 298 893
5	1 302 474	12	1 302 795	100 000	801 452
6	1 301 797	15	1 300 535		
7	1 302 131	20	1 300 524		

Tabelle 6.22: Vorstudie, OptaPlanner, Late Acceptance, `acceptedCountLimit`

Great Deluge

Beim Algorithmus Great Deluge gibt es drei einstellbare Parameter: `acceptedCountLimit`, wie bei den anderen Algorithmen. Ebenso kann `grDelWaterLevelIncrRatio` und `grDelWaterLevelIncrScore` eingestellt werden. Die Default-Einstellung, ohne eingestellte Parameter, ergibt einen Fitnesswert von 826 867. Für den Parameter `grDelWaterLeveLIncr-Ratio` (abgekürzt `gDWIncrRat`) wurden die Ergebnisse in Tabelle 6.23 und Abbildung 6.22 dargestellt. Wie hier sichtbar ist, gelang es nicht eine Ratio zu finden, mit der Ergebnisse über eine Million (bezüglich Fitnesswert) erreichbar sind. Daher wurde auch der Parameter `grDelWaterLeveLIncrScore` (abgekürzt `gDWIncrSco`, betrifft nur den „soft“-Wert vom 0hard/0soft Score) getestet, wie in Tabelle 6.24 und Abbildung 6.23 gezeigt wird. Wie hier ersichtlich ist, ergibt ein `grDelWaterLeveLIncrScore`-Wert von 0hard/100soft in

der mittleren Position zwischen 0hard/80soft und 0hard/110soft Werte über 1 300 000. Zusätzlich wird nun für 0hard/100soft der ocVaracceptedCountLimit-Wert getestet in Tabelle 6.25 und Abbildung 6.24, wobei `accCouLim` für `acceptedCountLimit` steht. Da die Unterschiede zwischen Werten 1 und 1000 kaum vorhanden sind, wird weiterhin die empfohlene Standardeinstellung [26] von `acceptedCountLimit` von 1 für die Hauptstudie verwendet.

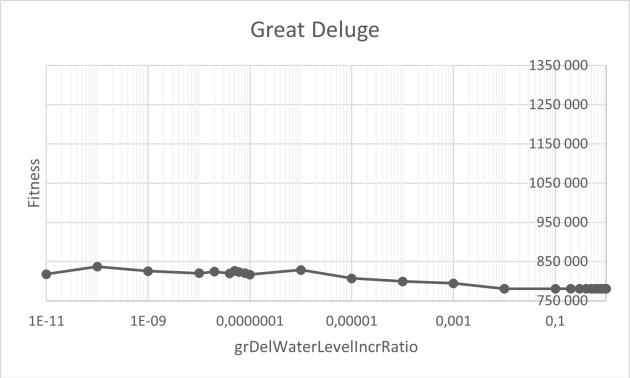

Abbildung 6.22: Vorstudie, OptaPlanner, Great Deluge, grDelWaterLeveLIncrRatio

gDWIncrRat	**Fitness**	gDWIncrRat	**Fitness**	gDWIncrRat	**Fitness**
$1 \cdot 10^{-11}$	818 056	0,000 000 1	816 937	0,4	780 653
$1 \cdot 10^{-10}$	837 579	0,000 001	828 537	0,5	780 653
0,000 000 001	825 713	0,000 01	807 324	0,6	780 653
0,000 000 01	820 385	0,0001	799 474	0,7	780 653
0,000 000 02	824 043	0,001	794 365	0,8	780 653
0,000 000 04	819 763	0,01	780 653	0,9	780 653
0,000 000 05	826 867	0,1	780 653	1	780 653
0,000 000 06	823 532	0,2	780 653		
0,000 000 08	820 363	0,3	780 653		

Tabelle 6.23: Vorstudie, OptaPlanner, Great Deluge, `grDelWaterLeveLIncrRatio`

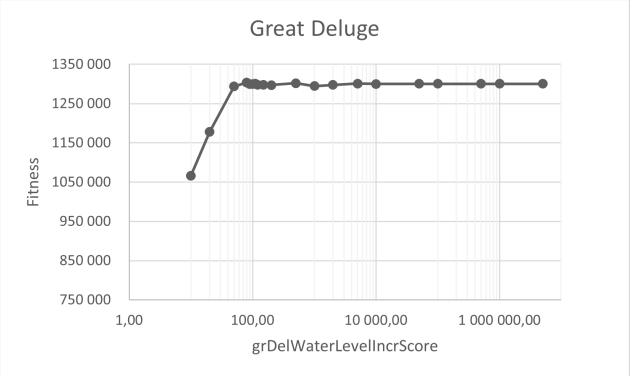

Abbildung 6.23: Vorstudie, OptaPlanner, Great Deluge, grDelWaterLeveLIncrScore

gDWIncrSco	**Fitness**	gDWIncrSco	**Fitness**	gDWIncrSco	**Fitness**
0	824 973	110	1 300 984	5000	1 301 059
10	1 066 556	120	1 297 973	10 000	1 300 209
20	1 178 174	150	1 298 265	50 000	1 300 634
50	1 293 965	200	1 297 338	100 000	1 300 634
80	1 303 912	500	1 302 135	500 000	1 300 634
90	1 300 139	1000	1 294 873	1 000 000	1 300 634
100	1 300 072	2000	1 297 649	5 000 000	1 300 634

Tabelle 6.24: Vorstudie, OptaPlanner, Great Deluge, `grDelWaterLeveLIncrScore`

Abbildung 6.24: Vorstudie, OptaPlanner, Great Deluge, acceptedCountLimit

accCouLim	Fitness	accCouLim	Fitness	accCouLim	Fitness
1	1 300 072	5	1 301 805	100	1 295 865
2	1 302 456	6	1 302 470	1000	1 298 892
3	1 302 476	7	1 302 043		
4	1 303 697	10	1 297 080		

Tabelle 6.25: Vorstudie, OptaPlanner, Great Deluge, `acceptedCountLimit`

Step Counting Hill Climbing

Für diesen Algorithmus gibt es zwei einstellbare Parameter: `stepCountHillClimbSize` und `acceptedCountLimit`. Ein `stepCountHillClimbSize` von Null ergibt einen Fehler. Die Tests von `stepCountHillClimbSize` (abgekürzt `stCHClSize`) wurden mit `acceptedCountLimit` von 1 ausgeführt und sind in Tabelle 6.26 und Abbildung 6.25 angeführt. Die Tests zu `acceptedCountLimit` (abgekürzt `accCouLim`) werden in Tabelle 6.27 und Abbildung 6.26 dargestellt. Wie ersichtlich ist, wird für die Hauptstudie ein `stepCountHillClimbSize`-Wert von 20 verwendet, sowie ein `acceptedCountLimit` von 1, da das für `stepCountHillClimbSize` der größte Wert ist und für `acceptedCountLimit` die empfohlene Standardeinstellung [26] und die Veränderung bei Werten zwischen 1 und 4 gering bleibt.

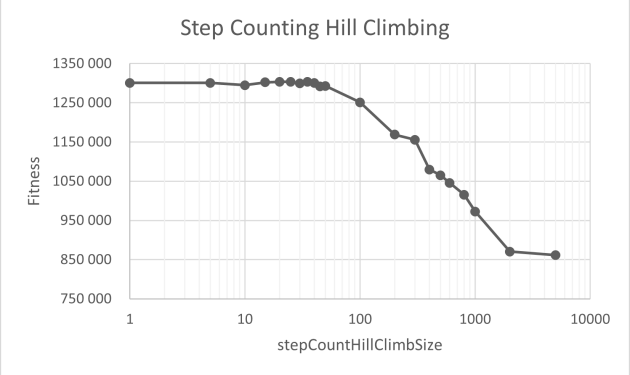

Abbildung 6.25: Vorstudie, OptaPlanner, Step Counting Hill Climbing, stepCountHillClimbSize

stCHClSize	Fitness	stCHClSize	Fitness	stCHClSize	Fitness
0	—	35	1 303 182	500	1 065 474
1	1 300 634	40	1 300 418	600	1 045 912
5	1 300 876	45	1 291 617	800	1 015 751
10	1 295 052	50	1 293 226	1000	972 771
15	1 302 208	100	1 251 078	2000	870 656
20	1 303 624	200	1 169 018	5000	862 089
25	1 303 500	300	1 155 888		
30	1 299 261	400	1 080 013		

Tabelle 6.26: Vorstudie, OptaPlanner, Step Counting Hill Climbing, `stepCountHillClimbSize`

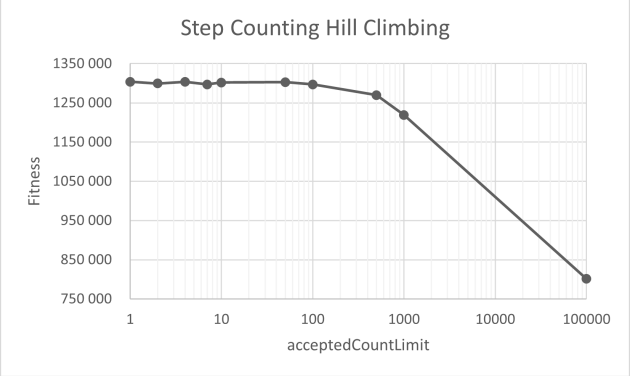

Abbildung 6.26: Vorstudie, OptaPlanner, Step Counting Hill Climbing, acceptedCountLimit

accCouLim	Fitness	accCouLim	Fitness	accCouLim	Fitness
1	1 303 624	10	1 301 980	1000	1 219 286
2	1 299 442	50	1 302 711	100 000	801 452
4	1 304 028	100	1 296 884		
7	1 296 806	500	1 269 730		

Tabelle 6.27: Vorstudie, OptaPlanner, Great Deluge, `acceptedCountLimit`

Strategic Oscillation

Zuletzt wurde die Erweiterung Strategic Oscillation zusammen mit Tabu Search getestet. Hierfür wird Tabu Search mit `acceptedCountLimit` von 1000 und einer `entityTabuRatio`

von 0,04 verwendet. Für Strategic Oscillation gibt es vier verschiedene `finalistPodiumType`-Werte, die in Tabelle 6.28 dargestellt werden. Anzumerken ist, dass `Highest Score` die ursprüngliche Standardeinstellung ist, sofern `finalistPodiumType` nicht eingestellt wird. Demnach ist dies die normale Tabu Search Einstellung, weswegen es nicht zur Auswahl für die Hauptstudie steht. Für die Hauptstudie wird ein `finalistPodiumType`-Wert von `STRATEGIC_OSCILLATION_BY_LEVEL_ON_BEST_SCORE` verwendet.

`finalistPodiumType`	**Fitness**
`HIGHEST_SCORE`	1 303 494
`STRATEGIC_OSCILLATION`	1 290 044
`STRATEGIC_OSCILLATION_BY_LEVEL`	1 290 044
`STRATEGIC_OSCILLATION_BY_LEVEL_ON_BEST_SCORE`	1 297 363

Tabelle 6.28: Vorstudie, OptaPlanner, Strategic Oscillation, `finalistPodiumType`

7 Hauptstudie

Nachfolgend werden zuerst die Methodik und danach die Parameter der Hauptstudie erklärt. Anschließend darauf wird der Testplan erläutert und die Ergebnisse dargestellt und beschrieben.

7.1 Methodik

Die Experimente wurden auf einer OpenVZ basierten virtuellen Maschine mit einer Intel Xeon CPU E5-2640 mit 2,4 GHz ausgeführt. Die virtuelle Maschine hat Zugriff auf 4 GB Arbeitsspeicher und kann bis zu 40 Kerne der physischen CPU nutzen. Das genutzte Betriebssystem ist CentOS Linux 7. Zur Ausführung des implementierten heuristischen Optimierers wurde OpenJDK 16 verwendet.

Zur Durchführung der Hauptstudie wurde zuerst die Vorstudie zur Findung von Einstellungsparameter (siehe Abschnitt 6) für die gewählten Algorithmen durchgeführt. Anhand der Ergebnisse wurden die gewählten 24 Konfigurationen bestimmt: Eine Konfiguration für die Ungarische Methode, auch Hungarian Algorithm genannt, sieben Konfigurationen für OptaPlanner Algorithmen und 16 Konfigurationen für verschiedene Einstellungen des genetischen Algorithmus im Jenetics-Framework. Diese Konfigurationen sind unter Abschnitt 7.3 ersichtlich.

Zur Durchführung der Tests werden zuerst die Testdaten anhand des Testplans, welcher ebenso im Abschnitt 7.3 ist, erstellt. Hierfür wird der Testdatengenerator (siehe Abschnitt 4.3) verwendet. Nach Erstellung der Daten mit dem Testdatengenerator werden diese auf die virtuelle Maschine hochgeladen. Der heuristische Optimierer mit der REST-Schnittstelle (siehe Abschnitt 5) wird auf der virtuellen Maschine mit dem folgenden Befehl gestartet:

```
1 nohup java -Xms2048m -Xmx2048m -XX:MaxPermSize=2048m -XX:MaxNewSize=2048m -jar
    optimizer-0.5.jar &
```

Code 7.1: Startbefehl für Optimizer

Mittels eines Skripts werden die Optimierungen ausgeführt. Das Skript arbeitet folgende Befehle in einer Schleife pro Testdatensatz ab:

- Initialisierung der Optimierungssession: `POST /optimizations` mit Weitergabe des aktuellen Testdatensatzes
- Starten der Optimierungssession: `PUT /optimizations/{optId}/start/wait` anhand gegebener Optimierungsidentifikation
- Speichern der Ergebnisse: `GET /optimizations/{optId}/result` anhand gegebener Optimierungsidentifikation als JSON-Datei
- Löschen der aktuellen Optimierungssession: `GET /optimizations/{optId}/remove` anhand gegebener Optimierungsidentifikation

7.2 Parameter

Für die Erstellung der Testdaten, die in der Hauptstudie verwendet werden, wurden verschiedene Parameter konfiguriert und anhand folgender Beschreibungen eingestellt. Die Szenarien lehnen sich an die Arbeit von Ruiz [30] an, welche zur Findung der Parameter als Inspiration verwendet worden ist.

7.2.1 Flugverteilung

Hierbei geht es um verschiedene Verteilungen der `TimeWished`-Zeitpunkte innerhalb des verfügbaren Slot-Bereichs. Dadurch soll getestet werden, inwieweit unterschiedliche Flugverteilungen das Ergebnis beeinflussen. Die folgenden Abbildungen wurden auf 100 Flüge angepasst. Bei 50 Flügen fehlen einige der Täler und Hügel in den Diagrammen. Die gleichverteilte Flugverteilung wird voraussichtlich zu den wenigsten Problemen bei den Optimierungen führen.

- **gleichverteilt**: Hierbei wird pro möglichen Slot ein Flug verteilt. Es gibt daher immer nur einen Flug, der über `TimeWished` zu einem gewissen Slot abfliegen will. Ein Slot hat daher auch nur einen Flug, der hier über `TimeWished` den Slot-Zeitpunkt will. Die Verteilung ist in Abbildung 7.1 ersichtlich.
- **extreme Häufung**: Hierbei wird eine extreme Häufung in der Verteilungsfunktion angestrebt. Die meisten Flüge wollen zu Slots, die in der Mitte der verfügbaren Slots sind, abfliegen. Die Slots an den Rändern der möglichen Slots haben nur wenige Flüge, die hier abfliegen wollen. Das ist in Abbildung 7.2 dargestellt.
- **normale Häufung**: Im Unterschied zur extremen Häufung wollen hier weiterhin viele Flüge zu Slots in der Mitte abfliegen, jedoch ist die Anzahl nicht so groß wie bei der anderen Häufung. An den Rändern gibt es auch mehr Flüge, die hier gewünschte Abflugzeiten haben. Dies ist in Abbildung 7.3 ersichtlich. Bei 50 Flügen ähnelt die normale Häufung mehr der gleichverteilten Verteilung als bei 100 Flügen.

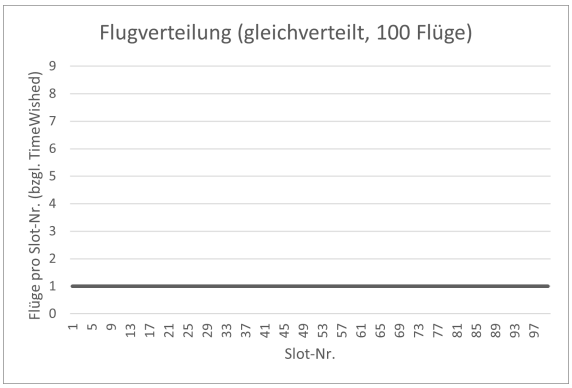

Abbildung 7.1: Flugverteilung (gleichverteilt)

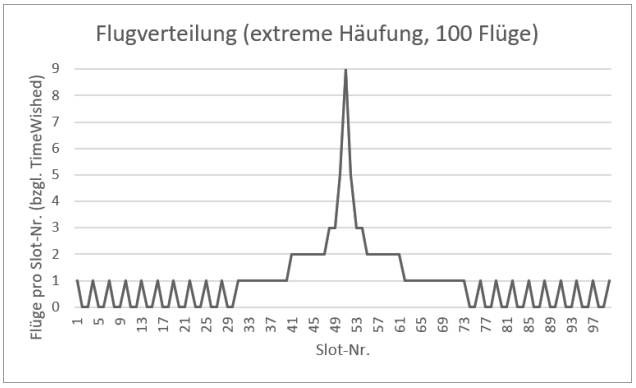

Abbildung 7.2: Flugverteilung (extreme Häufung)

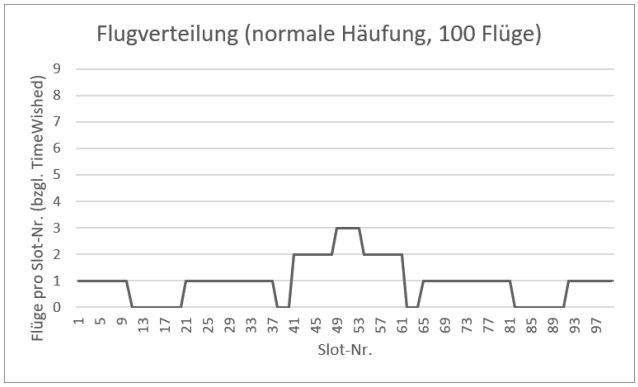

Abbildung 7.3: Flugverteilung (normale Häufung)

7.2.2 Priorität

Neben der unterschiedlichen Verteilung von `TimeWished` bei den generierten Flügen wird auch die Priorität konfiguriert. Die Priorität wirkt sich auf die Funktion aus, welche die Gewichtstabellen generiert, da jedes Grundgewicht mit der Priorität des aktuellen Flugs multipliziert wird. Eine hohe Priorität wird als zufällige Priorität zwischen 2 und 3 definiert. Für Flüge mit niedriger Priorität wird eine Zahl zwischen 0,8 und 1,1 vergeben. Je nach Flugverteilung können niedrige Prioritäten bei den mittleren Flügen zu weniger Problemen bei den Optimierungen führen.

- **Mitte hohe Priorität**: In dieser Konfiguration haben die Flüge, die im mittleren Bereich der zeitlich geordneten Flüge sind, eine höhere Priorität als die Flüge, welche zeitlich am Rand einzuordnen sind. Die Verteilung ist in Abbildung 7.4 ersichtlich.
- **Rand hohe Priorität**: Umgekehrt zur vorigen Konfiguration haben die zeitlich ersten und letzten Flüge eine höhere Priorität als die mittleren Flüge. Die Verteilung ist in Abbildung 7.5 ersichtlich.
- **gleichmäßige Priorität**: In dieser Einstellung wird allen Flügen zufällig eine Zahl zwischen 0,8 und 3 als Priorität zugewiesen. Im Durchschnitt werden hier die Flüge eine Priorität von 1,9 haben. Die Verteilung ist in Abbildung 7.6 ersichtlich.

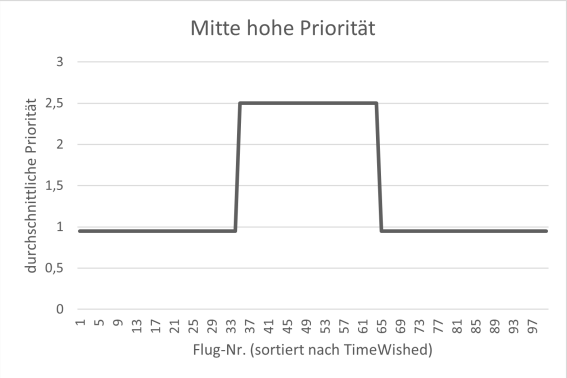

Abbildung 7.4: Priorität (Mitte hohe Priorität)

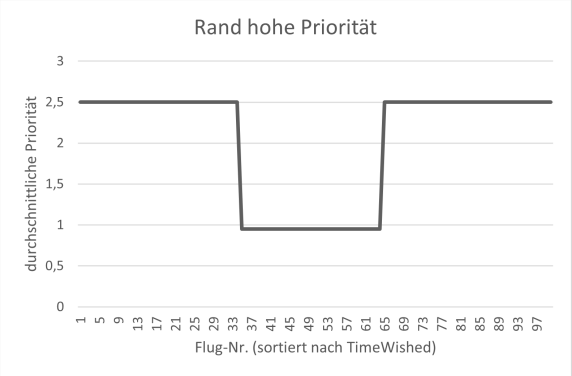

Abbildung 7.5: Priorität (Rand hohe Priorität)

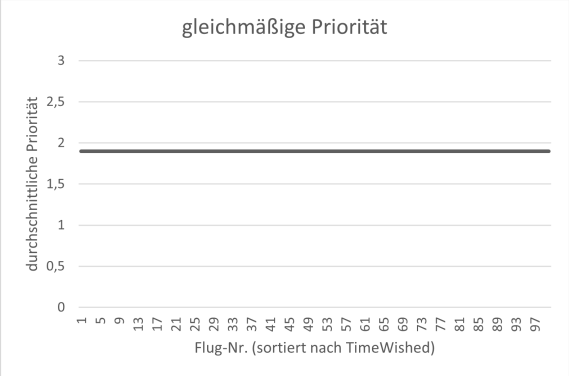

Abbildung 7.6: Priorität (gleichmäßige Priorität)

7.2.3 Margins

Als Margins wird hier der Zeitunterschied zwischen den `TimeNotBefore` und `TimeWished` Werten der einzelnen Flüge bezeichnet. Hiermit wird getestet, wie der Optimizer mit verschiedenen Margins zurechtkommt. Breitere Margins führen zu weniger Problemen bei den geplanten Optimierungssessions, da jeder Flug mehr mögliche Slots innerhalb der Margins hat als bei schmäleren Margins. Es gilt dabei für alle Varianten, dass `TimeWished` vor `TimeNotAfter` und nach `TimeNotBefore` ist.

- **breite Margins**: Margins von 50 Minuten. `TimeNotBefore` und `TimeWished`, sowie `TimeNotAfter` und `TimeWished` haben einen Abstand von jeweils 25 Minuten.
- **normale Margins**: Margins von 30 Minuten. `TimeNotBefore` und `TimeWished`, sowie `TimeNotAfter` und `TimeWished` haben einen Abstand von jeweils 15 Minuten.
- **schmale Margins**: Margins von 10 Minuten. `TimeNotBefore` und `TimeWished`, sowie `TimeNotAfter` und `TimeWished` haben einen Abstand von jeweils 5 Minuten.

7.2.4 Fluganzahl

Anhand der Fluganzahl wird getestet, wie der Optimizer mit unterschiedlicher Anzahl an Flügen und Slots zurechtkommt. Die Tests werden mit 100 und mit 50 Flügen durchgeführt.

Dadurch soll geprüft werden, ob der heuristische Optimierer mit verschiedener Anzahl an Flügen zurechtkommt. Es ist zu erwarten, dass 50 Flüge leichter optimierbar sind.

7.2.5 Mindest-, Maximalwert, Fallhöhe

Der Mindestwert, Maximalwert und die Fallhöhe beeinflussen die Berechnung der Gewichte, die über die Gewichtstabelle für die Optimierung bedeutend sind. Jedoch wird hierfür nur eine Variante eingestellt mit einem Mindestwert von $-10\,000$, einem Maximalwert von $10\,000$ und einer Fallhöhe von 6000, wie in Abbildung 4.1 dargestellt wurde.

7.2.6 Zeit

Die Zeit, die der Optimizer pro Testdurchlauf benötigen darf. Ein Testdurchlauf bedeutet hier ein Test einer Konfiguration. Es gibt 80 Jenetics-Konfigurationstests, 7 OptaPlanner-Konfigurationstests und einen Hungarian-Algorithm-Test pro Testnummer. Die Tests werden mit 5 und mit 10 Sekunden Dauer für die Optimierungssessions durchgeführt, um zu sehen, ob es bei längeren Sessionen bessere Ergebnisse gibt.

7.3 Testplan

Die vorgegebenen Parameter in den einstellbaren Werten ergeben 108 einzelne Testnummern mit jeweils 88 Konfigurationen pro Testnummer, die getestet werden. Dies ergibt 9504 einzelne Testdurchläufe. Zur besseren Darstellung werden „extreme Häufung“ mit „extr. Häuf.“, „normale Häufung“ mit „norm. Häuf.“, „Mitte hohe Priorität“ mit „Mitte h. P.“, „Rand hohe Priorität“ mit „Rand h. P.“ und „gleichmäßige Priorität“ mit „gleichm. P.“ abgekürzt. Die Einstellungen der Testnummern werden in Tabelle 7.1 dargestellt.

Test-Nr.	Flugverteilung	Priorität	Margins	Fluganzahl	Zeit
1	gleichverteilt	Mitte h. P.	breit	100	10
2	extr. Häuf.	Mitte h. P.	breit	100	10
3	norm. Häuf.	Mitte h. P.	breit	100	10

Test-Nr.	Flugverteilung	Priorität	Margins	Fluganzahl	Zeit
4	gleichverteilt	Rand h. P.	breit	100	10
5	extr. Häuf.	Rand h. P.	breit	100	10
6	norm. Häuf.	Rand h. P.	breit	100	10
7	gleichverteilt	gleichm. P.	breit	100	10
8	extr. Häuf.	gleichm. P.	breit	100	10
9	norm. Häuf.	gleichm. P.	breit	100	10
10	gleichverteilt	Mitte h. P.	mittel	100	10
11	extr. Häuf.	Mitte h. P.	mittel	100	10
12	norm. Häuf.	Mitte h. P.	mittel	100	10
13	gleichverteilt	Rand h. P.	mittel	100	10
14	extr. Häuf.	Rand h. P.	mittel	100	10
15	norm. Häuf.	Rand h. P.	mittel	100	10
16	gleichverteilt	gleichm. P.	mittel	100	10
17	extr. Häuf.	gleichm. P.	mittel	100	10
18	norm. Häuf.	gleichm. P.	mittel	100	10
19	gleichverteilt	Mitte h. P.	schmal	100	10
20	extr. Häuf.	Mitte h. P.	schmal	100	10
21	norm. Häuf.	Mitte h. P.	schmal	100	10
22	gleichverteilt	Rand h. P.	schmal	100	10
23	extr. Häuf.	Rand h. P.	schmal	100	10
24	norm. Häuf.	Rand h. P.	schmal	100	10
25	gleichverteilt	gleichm. P.	schmal	100	10
26	extr. Häuf.	gleichm. P.	schmal	100	10
27	norm. Häuf.	gleichm. P.	schmal	100	10
28	gleichverteilt	Mitte h. P.	breit	50	10
29	extr. Häuf.	Mitte h. P.	breit	50	10
30	norm. Häuf.	Mitte h. P.	breit	50	10
31	gleichverteilt	Rand h. P.	breit	50	10
32	extr. Häuf.	Rand h. P.	breit	50	10
33	norm. Häuf.	Rand h. P.	breit	50	10
34	gleichverteilt	gleichm. P.	breit	50	10
35	extr. Häuf.	gleichm. P.	breit	50	10
36	norm. Häuf.	gleichm. P.	breit	50	10

Test-Nr.	Flugverteilung	Priorität	Margins	Fluganzahl	Zeit
37	gleichverteilt	Mitte h. P.	mittel	50	10
38	extr. Häuf.	Mitte h. P.	mittel	50	10
39	norm. Häuf.	Mitte h. P.	mittel	50	10
40	gleichverteilt	Rand h. P.	mittel	50	10
41	extr. Häuf.	Rand h. P.	mittel	50	10
42	norm. Häuf.	Rand h. P.	mittel	50	10
43	gleichverteilt	gleichm. P.	mittel	50	10
44	extr. Häuf.	gleichm. P.	mittel	50	10
45	norm. Häuf.	gleichm. P.	mittel	50	10
46	gleichverteilt	Mitte h. P.	schmal	50	10
47	extr. Häuf.	Mitte h. P.	schmal	50	10
48	norm. Häuf.	Mitte h. P.	schmal	50	10
49	gleichverteilt	Rand h. P.	schmal	50	10
50	extr. Häuf.	Rand h. P.	schmal	50	10
51	norm. Häuf.	Rand h. P.	schmal	50	10
52	gleichverteilt	gleichm. P.	schmal	50	10
53	extr. Häuf.	gleichm. P.	schmal	50	10
54	norm. Häuf.	gleichm. P.	schmal	50	10
55	gleichverteilt	Mitte h. P.	breit	100	5
56	extr. Häuf.	Mitte h. P.	breit	100	5
57	norm. Häuf.	Mitte h. P.	breit	100	5
58	gleichverteilt	Rand h. P.	breit	100	5
59	extr. Häuf.	Rand h. P.	breit	100	5
60	norm. Häuf.	Rand h. P.	breit	100	5
61	gleichverteilt	gleichm. P.	breit	100	5
62	extr. Häuf.	gleichm. P.	breit	100	5
63	norm. Häuf.	gleichm. P.	breit	100	5
64	gleichverteilt	Mitte h. P.	mittel	100	5
65	extr. Häuf.	Mitte h. P.	mittel	100	5
66	norm. Häuf.	Mitte h. P.	mittel	100	5
67	gleichverteilt	Rand h. P.	mittel	100	5
68	extr. Häuf.	Rand h. P.	mittel	100	5
69	norm. Häuf.	Rand h. P.	mittel	100	5

Test-Nr.	Flugverteilung	Priorität	Margins	Fluganzahl	Zeit
70	gleichverteilt	gleichm. P.	mittel	100	5
71	extr. Häuf.	gleichm. P.	mittel	100	5
72	norm. Häuf.	gleichm. P.	mittel	100	5
73	gleichverteilt	Mitte h. P.	schmal	100	5
74	extr. Häuf.	Mitte h. P.	schmal	100	5
75	norm. Häuf.	Mitte h. P.	schmal	100	5
76	gleichverteilt	Rand h. P.	schmal	100	5
77	extr. Häuf.	Rand h. P.	schmal	100	5
78	norm. Häuf.	Rand h. P.	schmal	100	5
79	gleichverteilt	gleichm. P.	schmal	100	5
80	extr. Häuf.	gleichm. P.	schmal	100	5
81	norm. Häuf.	gleichm. P.	schmal	100	5
82	gleichverteilt	Mitte h. P.	breit	50	5
83	extr. Häuf.	Mitte h. P.	breit	50	5
84	norm. Häuf.	Mitte h. P.	breit	50	5
85	gleichverteilt	Rand h. P.	breit	50	5
86	extr. Häuf.	Rand h. P.	breit	50	5
87	norm. Häuf.	Rand h. P.	breit	50	5
88	gleichverteilt	gleichm. P.	breit	50	5
89	extr. Häuf.	gleichm. P.	breit	50	5
90	norm. Häuf.	gleichm. P.	breit	50	5
91	gleichverteilt	Mitte h. P.	mittel	50	5
92	extr. Häuf.	Mitte h. P.	mittel	50	5
93	norm. Häuf.	Mitte h. P.	mittel	50	5
94	gleichverteilt	Rand h. P.	mittel	50	5
95	extr. Häuf.	Rand h. P.	mittel	50	5
96	norm. Häuf.	Rand h. P.	mittel	50	5
97	gleichverteilt	gleichm. P.	mittel	50	5
98	extr. Häuf.	gleichm. P.	mittel	50	5
99	norm. Häuf.	gleichm. P.	mittel	50	5
100	gleichverteilt	Mitte h. P.	schmal	50	5
101	extr. Häuf.	Mitte h. P.	schmal	50	5
102	norm. Häuf.	Mitte h. P.	schmal	50	5

Test-Nr.	Flugverteilung	Priorität	Margins	Fluganzahl	Zeit
103	gleichverteilt	Rand h. P.	schmal	50	5
104	extr. Häuf.	Rand h. P.	schmal	50	5
105	norm. Häuf.	Rand h. P.	schmal	50	5
106	gleichverteilt	gleichm. P.	schmal	50	5
107	extr. Häuf.	gleichm. P.	schmal	50	5
108	norm. Häuf.	gleichm. P.	schmal	50	5

Tabelle 7.1: Testplan, Hauptstudie, Testnummern

Die einzelnen Konfigurationen, die pro Testnummer getestet werden, sind 80 Jenetics-Tests, 7 OptaPlanner-Tests und ein Hungarian-Algorithm-Test. Die Jenetics-Tests werden in Tabelle 7.2 genauer erläutert.Die Eingabedateien und Einstellungsdaten des Testdatengenerators sind unter folgendem Link[1] online verfügbar. Jeder Jenetics-Test wird fünfmal pro Konfiguration durchgeführt und hat folgende Einstellungen:

- Crossover: Partially Matched Crossover mit Parameter `c.A.P.` (`crossoverAlterProbability`)
- Mutator: Swap Mutator mit Parameter `m.A.P.` (`mutatorAlterProbability`)
- maximales Alter von Phenotypen: 80
- Fraktion der Nachkommen: 0,7
- Selektor für Nachkommen: Tournament Selector mit Parameter `o.S.P.` (`offspringSelectorParameter`)
- Selektor für Überlebende: Tournament Selector mit Parameter `s.S.P.` (`survivorsSelectorParameter`)
- Bevölkerungsgröße pro Generation: einstellbarer Parameter `pop.S.` (`populationSize`)

[1] http://files.dke.uni-linz.ac.at/theses/jaburek

Konf.-Nr.	pop.S.	o.S.P.	s.S.P.	m.A.P.	c.A.P.
1	500	50	50	0,15	0,35
2	500	50	50	0,15	0,9
3	500	50	50	0,6	0,35
4	500	50	50	0,6	0,9
5	500	10	10	0,15	0,35
6	500	10	10	0,15	0,9
7	500	10	10	0,6	0,35
8	500	10	10	0,6	0,9
9	500	3	3	0,15	0,35
10	70	3	3	0,15	0,9
11	70	3	3	0,6	0,35
12	70	3	3	0,6	0,9
13	70	10	10	0,15	0,35
14	70	10	10	0,15	0,9
15	70	10	10	0,6	0,35
16	70	10	10	0,6	0,9

Tabelle 7.2: Testplan, Hauptstudie, Konfigurationsnummern, Jenetics

Die sieben OptaPlanner-Tests nutzen für die „Construction Heuristic Phase“ einen First-Fit-Algorithmus und für die „Local Search Phase“ den Algorithmus, wie in Tabelle 7.3 angegeben ist. Der Hungarian Algorithm nutzt keine weiteren Konfigurationen.

Konf.-Nr.	Algorithmustyp	Einstellung
17	Hill Climbing	*Default-Einstellung*
18	Tabu Search	*Default-Einstellung*
19	Simulated Annealing	`simulAnnealStartTemp` = `0hard/500soft`
20	Late Acceptance	`lateAcceptanceSize` = 50 `acceptedCountLimit` = 4
21	Great Deluge	`grDelWaterLeveLIncrScore` = `0hard/100soft` `acceptedCountLimit` = 1
22	Step Counting Hill Climbing	`stepCountHillClimbSize` = 20 `acceptedCountLimit` = 1
23	Strategic Oscillation (mit Tabu Search)	`finalistPodiumType` = `STRATEGIC_OSCILLATION_BY_LEVEL_ON_BEST_SCORE` `entityTabuRatio` = 0,04 `acceptedCountLimit` = 1000
24	Hungarian Algorithm	

Tabelle 7.3: Testplan, Hauptstudie, Konfigurationsnummern, OptaPlanner und Hungarian Algorithm

7.4 Ergebnisse

Folgend werden die analysierten Ergebnisse besprochen. Zusammengefasste Ergebnisse der erreichten Fitnesswerte im größeren Detailgrad als im Fließtext finden sich im Anhang 9. Die Ergebnisse ohne Kürzung sind online verfügbar. Unter dem Link[2] sind neben den Einstellungsdateien des Testdatengenerators und den Eingabedateien vom Optimierer auch die Ergebnisdateien ersichtlich. Strukturiert zusammengefasste Ergebnisse in Bezug auf Fitnesswerte und Margin-Verletzungen finden sich in den folgenden Abschnitten. Es wird nicht versucht herauszufinden, welches Szenario die besten Fitnesswerte erzielt. Der Fokus liegt in der Analyse, welche Konfiguration für welches Szenario zu bevorzugen ist. Hierfür werden die Anzahl der Flüge, die nicht nach dem jeweiligen `TimeNotBefore` beziehungsweise nicht vor dem `TimeNotAfter` abfliegen und die erzielten Fitnesswerte in Relation zum erzielten Fitnesswert der Ungarischen Methode begutachtet. Die Ungarische Methode wird nur als Vergleichsalgorithmus für die erreichten Fitnesswerte verwendet.

[2]http://files.dke.uni-linz.ac.at/theses/jaburek

7.4.1 Überblick

Jenetics-Durchläufe betreffen Konfigurationsnummern 1 bis 16. OptaPlanner-Durchläufe betreffen Konfigurationsnummern 17 bis 23 und die Ungarische Methode wird mit Konfigurationsnummer 24 durchgeführt. Die Prozentwerte basieren auf der Ungarischen Methode. 100% entsprechen dem Fitnesswert der Ungarischen Methode. Überblicksergebnisse sind in Tabelle 7.4 zu finden.

Mit Blick auf alle durchgeführten Testdurchläufe und dem erreichtem Fitnesswert sind folgende Beobachtungen erwähnenswert:

- Das schlechteste Ergebnis von Jenetics-Durchläufen liegt bei -93,8% vom Fitnesswert der Ungarischen Methode, da der erzielte Fitnesswert negativ ist.
- Das beste Jenetics-Ergebnis pro Testnummer liegt bei 100%.
- Das schlechteste Ergebnis von OptaPlanner-Durchläufen liegt bei 88,4% vom Fitnesswert der Ungarischen Methode.
- Das beste OptaPlanner-Ergebnis pro Testnummer liegt bei 100%.
- In 29,6% der Testnummern ist das beste Ergebnis von Jenetics besser als das schlechteste Ergebnis von OptaPlanner.

Die Reihenfolge der besten Konfigurationen bezüglich dem durchschnittlich erreichtem Fitnesswert in Bezug auf die Ungarischen Methode ist wie in Tabelle 7.4 dargestellt.

Bei zu früh und zu spät abfliegenden Flugzeugen schneiden die Konfigurationen von OptaPlanner besser ab als die von Jenetics. Es geht hier jedoch um Unterschiede von weniger als einem oder zwei Prozent, wenn die Konfigurationsnummern 11 und 12 hierfür nicht beachtet werden, da diese eindeutig schlechter abschneiden. Bei OptaPlanner fliegen zwischen 4,57% und 4,78% der Flüge, je nach Algorithmus, zu früh ab und verletzen dadurch die Margins in Bezug auf `TimeNotBefore`. Zwischen 4,79% und 11,55% an Flügen fliegen bei Konfigurationen von Jenetics zu früh ab. 4,16% bis 4,38% beträgt der Anteil an Flügen bei OptaPlanner-Konfigurationen, die nach `TimeNotAfter` abfliegen. Bei Jenetics gibt es je nach Einstellung zwischen 4,43% und 11,08% zu spät abfliegende Flugzeuge. In Bezug auf zu spät und zu früh abfliegende Flüge empfehlen sich Tabu Search und Hill

Platz	% Fitnesswert	Konf.-Nr.	Framework	Algorithmustyp
1	100,00%	24	Hungarian	Hungarian Algorithm
2	99,67%	22	OptaPlanner	Step Counting Hill Climbing
3	99,64%	19	OptaPlanner	Simulated Annealing
4	99,60%	20	OptaPlanner	Late Acceptance
5	99,57%	21	OptaPlanner	Great Deluge
6	99,55%	23	OptaPlanner	Strategic Oscillation
7	99,41%	18	OptaPlanner	Tabu Search
8	99,03%	17	OptaPlanner	Hill Climbing
9	97,42%	2	Jenetics	Genetischer Algorithmus
10	97,39%	6	Jenetics	Genetischer Algorithmus
11	97,12%	14	Jenetics	Genetischer Algorithmus
12	96,72%	10	Jenetics	Genetischer Algorithmus
13	95,99%	5	Jenetics	Genetischer Algorithmus
14	95,77%	1	Jenetics	Genetischer Algorithmus
15	95,31%	13	Jenetics	Genetischer Algorithmus
16	95,08%	9	Jenetics	Genetischer Algorithmus
17	92,26%	7	Jenetics	Genetischer Algorithmus
18	92,01%	4	Jenetics	Genetischer Algorithmus
19	91,92%	3	Jenetics	Genetischer Algorithmus
20	91,72%	8	Jenetics	Genetischer Algorithmus
21	91,22%	16	Jenetics	Genetischer Algorithmus
22	91,13%	15	Jenetics	Genetischer Algorithmus
23	74,23%	11	Jenetics	Genetischer Algorithmus
24	67,95%	12	Jenetics	Genetischer Algorithmus

Tabelle 7.4: Ergebnisse, Hauptstudie, durchschnittlich erreichter Fitness relativ zum Hungarian Algorithm

Climbing als OptaPlanner-Einstellungen oder Konfigurationsnummern 6, 2 und 10 als Jenetics-Einstellung.

Diesen Ranglisten nach schneiden Variationen vom OptaPlanner besser als Variationen von Jenetics ab. Jedoch sind die Unterschiede nicht groß. Anhand vom durchschnittlichen Fitnesswert sind Step Counting Hill Climbing und Simulated Annealing zu bevorzugen. Von Jenetics aus sind die Konfigurationsnummern 2 und 6 zu bevorzugen. Konfigurationsnummern 11 und 12 sollen nicht verwendet werden, da sie sehr viel schlechtere Ergebnisse liefern als andere Konfigurationen.

7.4.2 Flugverteilung

Es wurden Experimente mit drei verschiedene Flugverteilungen getestet: gleichverteilte Flüge, Flüge mit extremer Häufung und Flüge mit normaler Häufung in der Mitte (gleichverteilt, extreme Häufung, normale Häufung). Die Ergebnisse sind in einer Übersicht in Tabelle 7.4.2 zusammengefasst. In der Tabelle 7.4.2 wird gleichverteilt mit „glv.“, extreme Häufung mit „extr. H.“ und normale Häufung mit „n. H.“ abgekürzt, sowie Konfigurationsnummer mit „KonfNr.“ abgekürzt.

Bei den Tests mit gleichverteilten Flügen ergeben sich perfekte Lösungen. Der erreichte durchschnittliche Fitnesswert jeder Konfiguration entspricht genau den der Ungarischen Methode, die ausgewählt wurde, die optimalsten Lösungen zu finden. Ebenso zeigt sich in Tabelle 7.4.2, dass kein Flug vor `TimeNotBefore` und kein Flug nach `TimeNotAfter` abfliegt. Somit erfüllt jeder Flug die Margins und es gibt keine Unterschiede zwischen den einzelnen Konfigurationen.

Die extreme Häufung soll mit den dazugehörigen Tests zeigen, dass perfekte Lösungen nicht immer möglich sind. Im Vergleich zur Ungarischen Methode erreichen die Konfigurationen 31,23% bis 99,61% des optimalen Fitnesswertes im Durchschnitt. Es ist dabei zu beachten, dass Konfigurationsnummern 11 und 12 (Jenetics) beide schlechte Fitnesswerte im Durchschnitt erreichen. Die drittschlechteste Konfiguration erreicht 85,48%. Auffällig ist die Tatsache, dass die schlechteste OptaPlanner-Konfiguration (Hill Climbing) mit 98,69% etwa 2 Prozentpunkte besser ist als die beste Jenetics-Konfiguration (KonfNr. 2) mit 96,32%. In Bezug auf die Verletzung von `TimeNotBefore`, bei denen Flugzeuge zu früh abfliegen, schneiden Optimierungen mit Konfigurationen vom OptaPlanner-Framework etwas besser im Schnitt ab (9,97% bis 10,28%) als die Jenetics-Konfigurationen (10,27% bis 12,22% ohne Konfigurationsnummern 11 und 12), wobei eine Jenetics-Konfiguration (KonfNr. 2) mit 10,27% zu früh abfliegenden Flügen besser abschneidet als Late Acceptance (OptaPlanner) und Strategic Oscillation (OptaPlanner) mit jeweils 10,28%. Bei Flügen, die zu spät abfliegen und daher `TimeNotAfter` verletzen, schneiden alle OptaPlanner-Konfigurationen (8,53% bis 8,78%) besser als Jenetics-Konfigurationen (9,02% bis 10,86% ohne Konfigurationsnummern 11 und 12) ab. Bei beiden Margin-Werten (`TimeNotBefore`, `TimeNotAfter`), sowie beim erreichten durchschnittlichen Fitnesswert sind die Jenetics-Konfigurationen 11 und 12 jeweils um einiges schlechter als die anderen Konfigurationen. Die besten Jenetics-Konfigurationen sind für die extreme Häufung Konfigurationsnum-

mern 2, 6 und 14, die in den drei untersuchten Bereichen die besten Ergebnisse für Jenetics erzielt haben. Weil bei den verschiedenen OptaPlanner-Konfigurationen die Ergebnisse sehr dicht aneinander liegen, ist hier kein klarer Favorit zu erkennen. Strategic Oscillation und Late Acceptance schneiden bei den Verletzungen der Margins als schlechteste ab. Hill Climbing erzielt die schlechtesten Fitnesswerte bei den OptaPlanner-Konfigurationen.

Als normale Häufung wird die Variante bezeichnet, die realistischere Ergebnisse liefern soll. Die Ergebnisse sind in Tabelle 7.4.2 ersichtlich. Hier erreichen die verschiedenen Einstellungen ohne Konfigurationsnummern 11 und 12 im Vergleich zur Ungarischen Methode 87,92% bis 99,63%. Die schlechteste OptaPlanner-Konfiguration (Hill Climbing) erreicht mit 98,39% ebenso mehr als die beste Jenetics-Konfiguration (95,91%) mit der Konfigurationsnummer 6. Bei Verletzungen von `TimeNotBefore` erreichen alle OptaPlanner-Konfigurationen (3,69% bis 4,06%) bessere Fitnesswerte im Durchschnitt als die Jenetics-Konfigurationen (4,09% bis 5,91% ohne Konfigurationsnummern 11 und 12). Dies ist jedoch nicht bei Flügen der Fall, die nach `TimeNotAfter` abfliegen. OptaPlanner-Konfiguration 23 (Strategic Oscillation) erreicht mit 4,50% schlechtere Werte als Jenetics-Konfigurationen 2, 6, 10 und 14 (4,26% bis 4,44%). In Summe schneiden die Konfigurationen vom OptaPlanner wiederum besser als die vom Jenetics-Framework ab. Innerhalb der OptaPlanner-Konfigurationen werden ähnliche Werte erreicht, weswegen kein klarer Verlierer oder Sieger bestimmbar ist. Bei den Jenetics-Konfigurationen schneiden die Konfigurationsnummern 2, 6, 10 und 14 am besten ab.

Zusammengefasst schneiden die Algorithmen, die vom OptaPlanner-Framework bereitgestellt werden besser als die Jenetics-Konfigurationen ab, wobei in einigen Testbereichen ein bis zwei OptaPlanner-Konfigurationen schlechter als die besten Jenetics-Konfigurationen abschneiden. In Bezug auf den erreichten Fitnesswert können von OptaPlanner aus alle Algorithmen außer Hill Climbing empfohlen werden. Bei den Verletzungen der Margins ist das Ergebnis nicht mehr so eindeutig. Es gibt jedoch auch nur Unterschiede von bis zu 0,75 Prozentpunkte zwischen den einzelnen Ergebnissen. Bei Analyse der Ergebnisse aus Sicht der Jenetics-Konfigurationen schneiden die Konfigurationsnummern 11 und 12 eindeutig schlechter als die anderen Einstellungen ab und sollten daher nicht verwendet werden. Die Einstellungen 2, 6, 10 und 14 jedoch zeigen auf alle Testbereiche die besten Ergebnisse von Jenetics-Konfigurationen, wobei Konfigurationsnummern 2 und 6 stetig die besseren sind und manchmal 10 statt einer der anderen Konfigurationen bei den besten zwei Einstellungen von Jenetics dabei sind.

	Fitness im Vergleich zu KonfNr. 24 Flugverteilung				Abflüge vor `TimeNotBefore` Flugverteilung				Abflüge nach `TimeNotAfter` Flugverteilung			
KonfNr.	alle	glv.	extr. H.	n. H.	alle	glv.	extr. H.	n. H.	alle	glv.	extr. H.	n. H.
1	95,77%	100,00%	93,59%	93,74%	5,09%	0,00%	10,74%	4,51%	4,72%	0,00%	9,47%	4,68%
2	97,42%	100,00%	96,32%	95,95%	4,81%	0,00%	10,27%	4,15%	4,48%	0,00%	9,11%	4,34%
3	91,92%	100,00%	86,67%	89,08%	5,93%	0,00%	11,96%	5,83%	5,49%	0,00%	10,61%	5,86%
4	92,01%	100,00%	86,88%	89,16%	5,77%	0,00%	11,72%	5,59%	5,55%	0,00%	10,68%	5,97%
5	95,99%	100,00%	93,96%	94,01%	5,08%	0,00%	10,66%	4,58%	4,73%	0,00%	9,43%	4,74%
6	97,39%	100,00%	96,27%	95,91%	4,79%	0,00%	10,29%	4,09%	4,43%	0,00%	9,02%	4,26%
7	92,26%	100,00%	87,25%	89,53%	5,74%	0,00%	11,72%	5,51%	5,42%	0,00%	10,56%	5,70%
8	91,72%	100,00%	86,18%	88,99%	5,95%	0,00%	12,05%	5,81%	5,51%	0,00%	10,62%	5,93%
9	95,08%	100,00%	92,44%	92,79%	5,26%	0,00%	11,05%	4,74%	4,87%	0,00%	9,67%	4,93%
10	96,72%	100,00%	95,22%	94,95%	4,96%	0,00%	10,54%	4,33%	4,57%	0,00%	9,27%	4,44%
11	74,23%	100,00%	49,32%	73,37%	9,93%	0,00%	19,23%	10,56%	9,60%	0,00%	18,13%	10,66%
12	67,95%	100,00%	31,23%	72,63%	11,55%	0,00%	23,72%	10,93%	11,08%	0,00%	22,26%	10,98%
13	95,31%	100,00%	93,08%	92,85%	5,19%	0,00%	10,76%	4,82%	4,85%	0,00%	9,68%	4,86%
14	97,12%	100,00%	95,89%	95,47%	4,83%	0,00%	10,29%	4,21%	4,51%	0,00%	9,19%	4,33%
15	91,13%	100,00%	85,48%	87,92%	5,95%	0,00%	11,94%	5,91%	5,64%	0,00%	10,86%	6,07%
16	91,22%	100,00%	85,55%	88,12%	6,02%	0,00%	12,22%	5,84%	5,59%	0,00%	10,85%	5,92%
17	99,03%	100,00%	98,69%	98,39%	4,57%	0,00%	10,00%	3,72%	4,16%	0,00%	8,72%	3,75%
18	99,41%	100,00%	99,03%	99,21%	4,58%	0,00%	10,06%	3,69%	4,19%	0,00%	8,58%	3,97%
19	99,64%	100,00%	99,61%	99,31%	4,76%	0,00%	10,25%	4,03%	4,24%	0,00%	8,64%	4,08%
20	99,60%	100,00%	99,18%	99,62%	4,78%	0,00%	10,28%	4,06%	4,33%	0,00%	8,75%	4,25%
21	99,57%	100,00%	99,11%	99,59%	4,64%	0,00%	9,97%	3,94%	4,25%	0,00%	8,53%	4,22%
22	99,67%	100,00%	99,40%	99,60%	4,68%	0,00%	10,14%	3,89%	4,29%	0,00%	8,78%	4,08%
23	99,55%	100,00%	99,01%	99,63%	4,77%	0,00%	10,28%	4,03%	4,38%	0,00%	8,64%	4,50%
24	100,00%	100,00%	100,00%	100,00%	4,70%	0,00%	9,94%	4,17%	4,39%	0,00%	8,44%	4,72%

Tabelle 7.5: Hauptstudie, Ergebnisse, Flugverteilung (Fitness im Vergleich zu Hungarian Algorithm, Abflüge für `TimeNotBefore`, Abflüge nach `TimeNotAfter`)

7.4.3 Priorität

Es wurden Experimente mit drei verschiedene Prioritäten getestet: Mitte hohe Priorität, Rand hohe Priorität und gleichmäßige Priorität. In Tabelle 7.4.3 werden die Ergebnisse zusammengefasst. Mitte hohe Priorität wird mit „M. h. P.“, Rand hohe Priorität mit „R. h. P.“, gleichmäßige Priorität mit „g. P.“ und Konfigurationsnummer mit „KonfNr.“ abgekürzt.

Bei Optimierungen, wo die Prioritäten in der Mitte hoch sind, zeigt sich in Bezug auf Fitnesswerte, dass OptaPlanner Werte zwischen 99,49% und 99,08% erreicht. Jenetics erreicht ohne Konfigurationsnummern 11 und 12 Fitnesswerte im Durchschnitt von 97,21% und 90,65%. Konfigurationsnummer 11 erreicht 73,52% und Nummer 12 schafft einen Fitnesswert im Durchschnitt von 68,35% vom Fitnesswert der Ungarischen Methode. Bei dem Anteil an Flügen, die vor `TimeNotBefore` abfliegen, erreichen OptaPlanner-Konfigurationen Werte zwischen 4,25% und 4,56%. Jenetics-Einstellungen sind hier schlechter, als die OptaPlanner-Algorithmen und erreichen ohne Konfigurationsnummer 11 und 12 4,78% bis 6,27%. Bei Abflügen nach `TimeNotAfter` erreichen OptaPlanner-Konfigurationen Anteile zwischen 3,67% und 4,8% und Jenetics-Konfigurationen zwischen 4,28% und 5,74%, wenn Konfigurationsnummern 11 (9,86%) und 12 (11,25%) nicht mitgezählt werden. Generell schneiden hier alle OptaPlanner-Konfigurationen und Jenetics-Konfigurationen 2, 6 und 14 gut ab.

Bei Prioritäten am Rand zeigt sich bei den Fitnesswerten ein ähnliches Bild, wo OptaPlanner-Konfigurationen besser als die von Jenetics abschneiden. Jedoch ändert sich das Bild bei dem Anteil an Abflügen, die vor `TimeNotBefore` abfliegen. OptaPlanner-Konfigurationen erreichen 4,69% bis 5,36% und Jenetics Einstellungen Anteile zwischen 4,83% und 5,86% (ohne Konfigurationsnummern 11 und 12), wie in Tabelle 7.4.3 ersichtlich ist. Ähnlich ist es für Abflüge, die nach `TimeNotAfter` abfliegen. In beiden Fällen ist Hill Climbing die beste Einstellung, und danach folgen Tabu Search und einige verschiedene Einstellungen von Jenetics, wie in der Tabelle 7.4.3 zu sehen ist.

Wenn die Prioritäten über alle Slots gleichmäßig und zufällig verteilt sind, ergibt sich ein ähnliches Bild wie bei einer höheren Priorität in der Mitte. OptaPlanner-Konfigurationen (außer Late Acceptance beim Anteil an Flügen, die vor `TimeNotBefore` abfliegen) schneiden besser ab als die Jenetics-Konfigurationen. Konfigurationsnummern 11 und 12

belegen in den drei Bereichen die letzten zwei Plätze. Die verschiedenen OptaPlanner-Konfigurationen schneiden jeweils ähnlich ab. Bei den Jenetics-Konfigurationen gibt es jedoch größere Unterschiede. Empfehlenswert sind Nummern 2, 6, 10 und 14, die jeweils bessere Ergebnisse, als andere Einstellungen bieten.

Die großen Unterschiede fallen beim Vergleich der Tests mit einer hohen Priorität am Rand mit den anderen Tests auf. Hill Climbing, Tabu Search, Konfigurationsnummern 2, 6, 10 und 14 von Jenetics bieten aus beiden Frameworks die besseren Ergebnisse.

	Fitness im Vergleich zu KonfNr. 24 Priorität				Abflüge vor `TimeNotBefore` Priorität				Abflüge nach `TimeNotAfter` Priorität			
KonfNr.	alle	M. h. P.	R. h. P.	g. P.	alle	M. h. P.	R. h. P.	g. P.	alle	M. h. P.	R. h. P.	g. P.
1	95,77%	95,45%	96,18%	95,69%	5,09%	5,21%	4,99%	5,05%	4,72%	4,67%	4,79%	4,69%
2	97,42%	97,18%	97,60%	97,49%	4,81%	4,80%	4,83%	4,79%	4,48%	4,38%	4,70%	4,37%
3	91,92%	91,14%	92,66%	91,96%	5,93%	6,19%	5,77%	5,82%	5,49%	5,70%	5,37%	5,39%
4	92,01%	91,48%	92,82%	91,74%	5,77%	5,98%	5,58%	5,75%	5,55%	5,72%	5,49%	5,43%
5	95,99%	95,82%	96,26%	95,88%	5,08%	5,11%	5,11%	5,02%	4,73%	4,70%	4,84%	4,64%
6	97,39%	97,21%	97,52%	97,46%	4,79%	4,80%	4,87%	4,71%	4,43%	4,28%	4,59%	4,40%
7	92,26%	91,60%	93,15%	92,02%	5,74%	6,08%	5,47%	5,67%	5,42%	5,72%	5,22%	5,32%
8	91,72%	91,19%	92,71%	91,27%	5,95%	6,27%	5,63%	5,96%	5,51%	5,71%	5,38%	5,46%
9	95,08%	94,56%	95,57%	95,10%	5,26%	5,39%	5,24%	5,16%	4,87%	4,90%	4,88%	4,82%
10	96,72%	96,54%	96,93%	96,70%	4,96%	5,01%	4,96%	4,91%	4,57%	4,54%	4,63%	4,53%
11	74,23%	73,52%	76,07%	73,11%	9,93%	10,10%	9,54%	10,14%	9,60%	9,86%	9,43%	9,51%
12	67,95%	68,35%	68,09%	67,41%	11,55%	11,67%	11,48%	11,50%	11,08%	11,25%	10,87%	11,12%
13	95,31%	95,03%	95,73%	95,17%	5,19%	5,28%	5,16%	5,13%	4,85%	4,83%	4,90%	4,81%
14	97,12%	97,14%	97,24%	96,99%	4,83%	4,78%	4,93%	4,78%	4,51%	4,30%	4,72%	4,51%
15	91,13%	90,65%	91,98%	90,77%	5,95%	6,15%	5,74%	5,96%	5,64%	5,74%	5,53%	5,66%
16	91,22%	90,81%	92,00%	90,84%	6,02%	6,14%	5,85%	6,06%	5,59%	5,71%	5,46%	5,60%
17	99,03%	99,08%	98,84%	99,16%	4,57%	4,47%	4,69%	4,56%	4,16%	3,81%	4,44%	4,22%
18	99,41%	99,26%	99,47%	99,51%	4,58%	4,42%	4,81%	4,53%	4,19%	3,78%	4,64%	4,14%
19	99,64%	99,49%	99,78%	99,65%	4,76%	4,25%	5,36%	4,67%	4,24%	3,67%	4,97%	4,08%
20	99,60%	99,32%	99,78%	99,71%	4,78%	4,36%	5,17%	4,81%	4,33%	3,78%	4,94%	4,28%
21	99,57%	99,31%	99,62%	99,77%	4,64%	4,28%	5,03%	4,61%	4,25%	3,75%	4,81%	4,19%
22	99,67%	99,46%	99,79%	99,75%	4,68%	4,25%	5,11%	4,67%	4,29%	3,67%	4,92%	4,28%
23	99,55%	99,27%	99,75%	99,63%	4,77%	4,56%	5,19%	4,56%	4,38%	4,08%	4,86%	4,19%
24	100,00%	100,00%	100,00%	100,00%	4,70%	4,08%	5,56%	4,47%	4,39%	3,78%	5,33%	4,06%

Tabelle 7.6: Hauptstudie, Ergebnisse, Priorität (Fitness im Vergleich zu Hungarian Algorithm, Abflüge für `TimeNotBefore`, Abflüge nach `TimeNotAfter`)

7.4.4 Margin-Breite

Es wurden Experimente mit drei verschiedene Margin-Breiten gemessen: schmale Margins mit 10 Minuten Abstand, normale Margins mit 30 Minuten Abstand und breite Margins mit 50 Minuten Abstand zwischen `TimeNotBefore` und `TimeNotAfter`. Die Ergebnisse werden zusammengefasst in Tabelle 7.4.4 dargestellt. Konfigurationsnummer wird mit „KonfNr.“ abgekürzt.

Bei schmalen Margins werden von Jenetics-Konfigurationen (ohne Konfigurationsnummer 11 (54,44%) und 12 (41,81%)) Fitnesswerte zwischen 83,32% und 95,42% vom Fitnesswert der Ungarischen Methode erreicht. OptaPlanner-Algorithmen erreichen Werte zwischen 98,53% und 99,70%. Die beste OptaPlanner-Konfiguration verwendet den Step Counting Hill Climbing Algorithmus. Die beste Jenetics-Konfiguration ist Nummer 2. Bezüglich Flügen, die zu früh abfliegen, schneiden Algorithmen von OptaPlanner (7,25% bis 7,61%) auch besser als die Konfigurationen von Jenetics (7,72% bis 9,96% ohne Konfigurationsnummer 11 (16,10%) und 12 (19,21%)) ab. Ein ähnliches Bild ergibt sich auch bei Flügen, die zu spät abfliegen zwischen Jenetics (7,22% bis 9,34% ohne die Ausreißer der Konfigurationsnummern 11 und 12) und OptaPlanner (6,53% bis 7,06%). Von Jenetics Seite sind wiederum Nummern 2, 6 und 14 zu empfehlen. Konfigurationsnummern 11 und 12 haben schlechte Ergebnisse. Bei OptaPlanner schneiden alle Frameworks ähnlich gut ab.

Wenn normale Margins in den Optimierungssessions verwendet werden, ergibt sich bei Fitnesswerten wiederum, dass die OptaPlanner-Konfigurationen (99,54% bis 99,11%) besser als die Konfigurationen von Jenetics (98,14% bis 93,20% ohne die zwei schlechtesten Konfigurationen (Nummer 11 und 12)) sind. Bei Flügen, die vor `TimeNotBefore` abfliegen, sind die Einstellungen vom OptaPlanner-Framework nicht mehr eindeutig vorne, weil Simulated Annealing und Strategic Oscillation von Jenetics-Konfigurationsnummern 2 und 14 geschlagen werden. Beim Anteil an Flügen, die zu spät abflogen, gibt es wiederum eine Jenetics-Konfiguration (Nummer 6), welche mit 4,06% besser als manche OptaPlanner-Konfigurationen ist. In Summe schneiden die meisten OptaPlanner-Konfigurationen gut ab, wobei einige Ausreißer dabei sind, wie in der Tabelle nachlesbar ist. Bei den Jenetics-Konfigurationen erreichen die Nummern 2, 6, 14 und in den meisten Fällen auch 10 gute Lösungen.

Mit Verwendung von breiten Margins ergeben sich wiederum Fitnesswerte, wo die OptaPlanner-Konfigurationen (99,89% bis 99,44%) zumindest 0,56 Prozentpunkte besser

sind als die Jenetics-Konfigurationen (98,88% bis 96,76% ohne Konfigurationsnummer 11 (90,29%) und 12 (88,56%)). Bei Abflügen vor `TimeNotBefore` und nach `TimeNotAfter` zeigt sich, dass einige Jenetics-Konfigurationen besser als die OptaPlanner-Einstellungen sind. Insbesondere Late Acceptance und Great Deluge schneiden hier schlechter ab als üblich. Es werden Werte von bis zu 2,14% (OptaPlanner (Tabu Search)) und 2,23% (Jenetics-Konfigurationsnummer 6) erreicht. Bei OptaPlanner sind Tabu Search, Simulated Annealing und Strategic Oscillation zu empfehlen. Bei Jenetics sind, wie üblich, Konfigurationen 2, 6, 10 und 14 zu empfehlen, wobei 6 ähnlich gut ist wie die OptaPlanner-Konfigurationen.

Besonders bei den schlechter abschneidenden Jenetics-Konfigurationen zeigt sich, dass breitere Margins zu besseren Fitnesswerten im Vergleich zur Ungarischen Methode führen. In den meisten Fällen sind Ergebnisse bei Tests mit normalen Margins besser als bei schmalen Margins. Die breiten Margins schneiden hier im Schnitt am besten ab, da hier die Flüge die meisten Slots als noch akzeptable Ausweichmöglichkeiten innerhalb der Margins anbieten, im Gegensatz zu schmalen Margins. Am eindeutigsten sieht man es bei Jenetics-Konfigurationsnummern 11 und 12 in Tabelle 7.4.4. Je nach Einstellung der Margins verbessern die Konfigurationen von Jenetics ihre Ergebnisse.

	Fitness im Vergleich zu KonfNr. 24 Marginbreite				Abflüge vor `TimeNotBefore` Marginbreite				Abflüge nach `TimeNotAfter` Marginbreite			
KonfNr.	alle	schmal	normal	breit	alle	schmal	normal	breit	alle	schmal	normal	breit
1	95,77%	92,00%	97,01%	98,32%	5,09%	8,23%	4,62%	2,41%	4,72%	7,74%	4,37%	2,03%
2	97,42%	95,42%	98,02%	98,84%	4,81%	7,72%	4,43%	2,27%	4,48%	7,22%	4,26%	1,97%
3	91,92%	85,07%	93,67%	97,02%	5,93%	9,71%	5,41%	2,68%	5,49%	9,11%	5,02%	2,33%
4	92,01%	85,19%	93,83%	97,02%	5,77%	9,47%	5,23%	2,61%	5,55%	9,02%	5,19%	2,43%
5	95,99%	92,40%	97,14%	98,42%	5,08%	8,18%	4,65%	2,40%	4,73%	7,72%	4,43%	2,03%
6	97,39%	95,16%	98,14%	98,88%	4,79%	7,68%	4,47%	2,23%	4,43%	7,30%	4,06%	1,92%
7	92,26%	85,68%	94,01%	97,09%	5,74%	9,34%	5,27%	2,61%	5,42%	8,96%	4,97%	2,33%
8	91,72%	84,72%	93,53%	96,92%	5,95%	9,69%	5,42%	2,74%	5,51%	9,04%	5,16%	2,35%
9	95,08%	90,71%	96,43%	98,10%	5,26%	8,49%	4,84%	2,46%	4,87%	7,98%	4,50%	2,12%
10	96,72%	94,01%	97,58%	98,58%	4,96%	8,00%	4,56%	2,31%	4,57%	7,46%	4,28%	1,97%
11	74,23%	54,44%	77,96%	90,29%	9,93%	16,10%	9,39%	4,29%	9,60%	15,73%	9,21%	3,85%
12	67,95%	41,81%	73,49%	88,56%	11,55%	19,21%	10,68%	4,76%	11,08%	18,58%	10,41%	4,24%
13	95,31%	91,21%	96,58%	98,13%	5,19%	8,44%	4,76%	2,37%	4,85%	7,93%	4,46%	2,16%
14	97,12%	94,70%	97,92%	98,74%	4,83%	7,79%	4,43%	2,28%	4,51%	7,24%	4,26%	2,02%
15	91,13%	83,32%	93,32%	96,76%	5,95%	9,84%	5,37%	2,64%	5,64%	9,34%	5,17%	2,42%
16	91,22%	83,70%	93,20%	96,76%	6,02%	9,96%	5,36%	2,74%	5,59%	9,08%	5,28%	2,41%
17	99,03%	98,53%	99,11%	99,44%	4,57%	7,31%	4,25%	2,17%	4,16%	6,53%	3,92%	2,03%
18	99,41%	99,24%	99,31%	99,68%	4,58%	7,44%	4,17%	2,14%	4,19%	6,72%	4,03%	1,81%
19	99,64%	99,49%	99,54%	99,89%	4,76%	7,25%	4,75%	2,28%	4,24%	6,64%	4,08%	2,00%
20	99,60%	99,65%	99,45%	99,71%	4,78%	7,42%	4,44%	2,47%	4,33%	6,78%	4,11%	2,11%
21	99,57%	99,67%	99,33%	99,71%	4,64%	7,25%	4,22%	2,44%	4,25%	6,72%	3,94%	2,08%
22	99,67%	99,70%	99,48%	99,82%	4,68%	7,25%	4,42%	2,36%	4,29%	6,69%	4,19%	1,97%
23	99,55%	99,58%	99,40%	99,67%	4,77%	7,61%	4,47%	2,22%	4,38%	7,06%	4,25%	1,83%
24	100,00%	100,00%	100,00%	100,00%	4,70%	7,28%	4,61%	2,22%	4,39%	6,97%	4,17%	2,03%

Tabelle 7.7: Hauptstudie, Ergebnisse, Marginbreite (Fitness im Vergleich zu Hungarian Algorithm, Abflüge für `TimeNotBefore`, Abflüge nach `TimeNotAfter`)

7.4.5 Fluganzahl

Es wurden Tests mit jeweils 50 und 100 Flügen und Slots durchgeführt. Die zusammengefassten Ergebnisse sind in Tabelle 7.4.5 ersichtlich. Konfigurationsnummer wird mit „KonfNr.“, `TimeNotBefore` mit „`TNB`“ und `TimeNotAfter` mit „`TNA`“ abgekürzt.

Bei 50 Flügen in den Tests erreicht der Hill Climbing Algorithmus als einzige Konfiguration vom OptaPlanner schlechtere Ergebnisse als Jenetics-Konfigurationsnummern 2, 6, 10 und 14. Es werden Werte zwischen 99,81% und 97,85% im Vergleich zur Ungarischen Methode erreicht (ohne Konfigurationsnummern 11 und 12). Bei zu früh abfliegenden Flügen sind die meisten Konfigurationen von Jenetics, außer Konfigurationsnummern 3, 6, 9, 11, 12 und 16, besser als die von OptaPlanner. Den besten Wert erreicht Konfigurationsnummer 14 mit 1,27%. Bei Flügen, die zu spät abfliegen bietet sich ein ähnliches Bild. Mit Ausnahme vom Hill Climbing Algorithmus sind alle Jenetics-Konfigurationen (außer Konfigurationsnummern 11 und 12) besser als die von OptaPlanner.

Wenn pro Optimierungssession 100 Flüge getestet werden, schneiden OptaPlanner-Konfigurationen beim Fitnesswert (99,62% bis 99,15%) besser ab als die von Jenetics (95,78% bis 84,41%). Ähnlich ist es bei Flügen, die vor `TimeNotBefore` abfliegen, wo Jenetics wiederum hinter OptaPlanner ist. Bei zu spät abfliegenden Flügen zeigen OptaPlanner-Einstellungen ebenso bessere Ergebnisse. Bei OptaPlanner erreichen alle Algorithmen in Bezug auf die drei getesteten Bereiche ähnliche Werte und daher sind alle zu empfehlen. Für Jenetics zeigen Konfigurationsnummern 11 und 12 sehr schlechte Ergebnisse. Empfohlen werden dort die Konfigurationen 2, 6 und 14.

Zwischen den Tests mit 50 und mit 100 Flügen gibt es einige Unterschiede bei den Verletzungen der Margins. Bei den Fitnesswerten ist in beiden Fällen eine OptaPlanner-Konfiguration zu empfehlen. Jedoch zeigt sich bei 50 Flügen, dass Jenetics-Konfigurationen weniger Verletzungen der Margins generieren als Konfigurationen von OptaPlanner. Bei 100 Flügen ist es genau umgekehrt. Nur Hill Climbing gibt bei beiden Testvarianten gute Ergebnisse.

	Fitness			Abflüge vor TNB			Abflüge nach TNA		
		Fluganzahl			Fluganzahl			Fluganzahl	
KonfNr.	alle	50	100	alle	50	100	alle	50	100
1	95,77%	98,74%	92,81%	5,09%	1,32%	8,85%	4,72%	1,13%	8,30%
2	97,42%	99,07%	95,78%	4,81%	1,31%	8,30%	4,48%	1,16%	7,81%
3	91,92%	98,06%	85,77%	5,93%	1,36%	10,50%	5,49%	1,20%	9,77%
4	92,01%	98,22%	85,81%	5,77%	1,28%	10,26%	5,55%	1,15%	9,95%
5	95,99%	98,79%	93,19%	5,08%	1,30%	8,86%	4,73%	1,16%	8,29%
6	97,39%	99,04%	95,75%	4,79%	1,34%	8,25%	4,43%	1,14%	7,71%
7	92,26%	98,11%	86,41%	5,74%	1,29%	10,19%	5,42%	1,15%	9,69%
8	91,72%	98,23%	85,22%	5,95%	1,28%	10,62%	5,51%	1,09%	9,94%
9	95,08%	98,68%	91,47%	5,26%	1,34%	9,19%	4,87%	1,14%	8,59%
10	96,72%	98,95%	94,50%	4,96%	1,33%	8,58%	4,57%	1,13%	8,01%
11	74,23%	91,17%	57,30%	9,93%	2,63%	17,23%	9,60%	2,39%	16,80%
12	67,95%	86,67%	49,23%	11,55%	3,81%	19,29%	11,08%	3,38%	18,78%
13	95,31%	98,74%	91,88%	5,19%	1,30%	9,08%	4,85%	1,16%	8,53%
14	97,12%	98,95%	95,29%	4,83%	1,27%	8,39%	4,51%	1,16%	7,85%
15	91,13%	97,85%	84,41%	5,95%	1,28%	10,62%	5,64%	1,11%	10,17%
16	91,22%	97,87%	84,57%	6,02%	1,35%	10,69%	5,59%	1,18%	10,00%
17	99,03%	98,90%	99,15%	4,57%	1,33%	7,81%	4,16%	1,07%	7,24%
18	99,41%	99,45%	99,37%	4,58%	1,41%	7,76%	4,19%	1,22%	7,15%
19	99,64%	99,67%	99,62%	4,76%	1,67%	7,85%	4,24%	1,37%	7,11%
20	99,60%	99,68%	99,52%	4,78%	1,59%	7,96%	4,33%	1,33%	7,33%
21	99,57%	99,60%	99,54%	4,64%	1,37%	7,91%	4,25%	1,22%	7,28%
22	99,67%	99,76%	99,58%	4,68%	1,52%	7,83%	4,29%	1,41%	7,17%
23	99,55%	99,81%	99,29%	4,77%	1,74%	7,80%	4,38%	1,56%	7,20%
24	100,00%	100,00%	100,00%	4,70%	1,85%	7,56%	4,39%	1,74%	7,04%

Tabelle 7.8: Hauptstudie, Ergebnisse, Fluganzahl (Fitness im Vergleich zu Hungarian Algorithm (Konfigurationsnummer 24), Abflüge für `TimeNotBefore`, Abflüge nach `TimeNotAfter`)

7.4.6 Zeitdauer

Zusätzlich zu den vorigen Tests wurde auch noch überprüft, ob Unterschiede feststellbar sind, wenn Tests 5 oder 10 Sekunden lang durchgeführt werden. Die gewonnenen Ergebnisse sind in Tabelle 7.4.6 dargestellt. Konfigurationsnummer wird mit „KonfNr.“, Sekunden mit „Sek.“, `TimeNotBefore` mit „`TNB`“ und `TimeNotAfter` mit „`TNA`“ abgekürzt.

Bei Optimierungssessions mit 5 Sekunden Dauer zeigt sich beim erreichten Fitnesswert, dass OptaPlanner-Konfigurationen (99,61% bis 99,03%) um einiges besser abschneiden als Jenetics-Einstellungen (96,73% bis 89,41% ohne Konfigurationsnummern 11 und 12). Die besten acht Einstellungen bei Verletzungen von `TimeNotBefore`, bei denen die Flüge zu früh abfliegen, sind die OptaPlanner-Konfigurationen und Jenetics Konfigurationsnummer 6 mit Werten zwischen 4,57% und 4,93%. Jenetics-Konfigurationen 2 und 14 erreichen ebenso noch Werte unter 5%. Beim Anteil der zu spät abfliegenden Flugzeuge erreichen die besten acht Konfigurationen (alle OptaPlanner-Konfigurationen und Jenetics-Konfigurationsnummer 6) Werte zwischen 4,17% und 4,55%. Bei beiden Margin-Verletzungen sind Konfigurationsnummern 11 und 12 die schlechtesten mit Werten über 9%.

Längere Optimierungssessions (10 Sekunden Dauer) erzeugen in den meisten Fällen bessere Ergebnisse. In Bezug auf erreichte Fitnesswerte werden von OptaPlanner-Konfigurationen Werte zwischen 99,73% und 99,03% vom erreichten Optimum (dem Ergebnis der Ungarischen Methode) erreicht. Konfigurationen von Jenetics (ohne Nummern 11 und 12) erreichen nur zwischen 98,15% und 92,67%. In Bezug auf zu früh abfliegende Flüge gibt es einige Jenetics-Konfigurationen, welche unter den besten sind. OptaPlanner-Konfigurationen erreichen 4,54% bis 4,76%. Jenetics-Konfigurationen erreichen ohne Nummer 11 und 12 4,68% bis 5,61%. Das Bild ist ähnlich bei zu spät abfliegenden Flugzeugen, die `TimeNotAfter` verletzen. Hier ist eine Jenetics-Konfiguration (Nummer 6) besser als die schlechteren OptaPlanner-Konfigurationen.

In Summe zeigt sich bei den Tests, dass bei 5 Sekunden die Konfigurationen von OptaPlanner eindeutig besser als die von Jenetics sind. Bei beiden Zeitdauern sind vor allem Hill Climbing und Tabu Search in Bezug auf Margin-Verletzungen zu empfehlen. Bei den Fitnesswerten ist jedoch Strategic Oscillation die bessere Wahl.

	Fitness			Abflüge vor TNB			Abflüge nach TNA		
		Zeitdauer (Sek.)			Zeitdauer (Sek.)			Zeitdauer (Sek.)	
KonfNr.	alle	5	10	alle	5	10	alle	5	10
1	95,77%	94,76%	96,79%	5,09%	5,28%	4,89%	4,72%	4,90%	4,53%
2	97,42%	96,73%	98,12%	4,81%	4,94%	4,68%	4,48%	4,58%	4,39%
3	91,92%	90,41%	93,43%	5,93%	6,30%	5,56%	5,49%	5,78%	5,20%
4	92,01%	90,39%	93,63%	5,77%	6,14%	5,40%	5,55%	5,90%	5,20%
5	95,99%	95,03%	96,94%	5,08%	5,30%	4,86%	4,73%	4,88%	4,57%
6	97,39%	96,63%	98,15%	4,79%	4,90%	4,69%	4,43%	4,55%	4,30%
7	92,26%	90,74%	93,77%	5,74%	6,11%	5,37%	5,42%	5,76%	5,09%
8	91,72%	89,92%	93,52%	5,95%	6,44%	5,46%	5,51%	5,86%	5,17%
9	95,08%	93,96%	96,20%	5,26%	5,53%	5,00%	4,87%	5,11%	4,63%
10	96,72%	95,88%	97,57%	4,96%	5,14%	4,78%	4,57%	4,74%	4,40%
11	74,23%	73,60%	74,86%	9,93%	10,03%	9,83%	9,60%	9,76%	9,43%
12	67,95%	68,07%	67,83%	11,55%	11,66%	11,44%	11,08%	11,08%	11,07%
13	95,31%	94,21%	96,41%	5,19%	5,43%	4,96%	4,85%	5,07%	4,63%
14	97,12%	96,34%	97,90%	4,83%	4,96%	4,71%	4,51%	4,67%	4,35%
15	91,13%	89,60%	92,67%	5,95%	6,32%	5,58%	5,64%	5,93%	5,35%
16	91,22%	89,41%	93,03%	6,02%	6,42%	5,61%	5,59%	5,99%	5,19%
17	99,03%	99,03%	99,03%	4,57%	4,57%	4,57%	4,16%	4,19%	4,13%
18	99,41%	99,33%	99,50%	4,58%	4,63%	4,54%	4,19%	4,17%	4,20%
19	99,64%	99,58%	99,70%	4,76%	4,76%	4,76%	4,24%	4,20%	4,28%
20	99,60%	99,59%	99,62%	4,78%	4,93%	4,63%	4,33%	4,31%	4,35%
21	99,57%	99,59%	99,55%	4,64%	4,69%	4,59%	4,25%	4,31%	4,19%
22	99,67%	99,61%	99,73%	4,68%	4,76%	4,59%	4,29%	4,35%	4,22%
23	99,55%	99,53%	99,57%	4,77%	4,81%	4,72%	4,38%	4,43%	4,33%
24	100,00%	100,00%	100,00%	4,70%	4,78%	4,63%	4,39%	4,43%	4,35%

Tabelle 7.9: Hauptstudie, Ergebnisse, Zeitdauer pro Optimierungsvorgang (Fitness im Vergleich zu Hungarian Algorithm (Konfigurationsnummer 24), Abflüge für `TimeNotBefore`, Abflüge nach `TimeNotAfter`)

8 Fazit

Die in dieser Arbeit durchgeführten Experimente mit den Frameworks Jenetics und OptaPlanner zeigen, dass die mittels heuristischer Suche gefundenen Ergebnisse im Allgemeinen nahe der optimalen Lösung sind, wie sie von der (exakten) Ungarischen Methode gefunden wird. Generell lässt sich sagen, dass die lokalen Suchalgorithmen in OptaPlanner etwas bessere Ergebnisse liefern als die mittels Jenetics implementierten genetischen Algorithmen. Die verschiedenen Algorithmen können aber in den untersuchten Konfigurationen prinzipiell (mit einigen wenigen Ausnahmen) gut für die Optimierung von Abflugreihenfolgen verwendet werden. Bei der gleichmäßigen Verteilung der Präferenzen der einzelnen Flüge ist die Wahl des Optimierungsalgorithmus wie erwartet nicht von Bedeutung, da alle Einstellungen das optimale Ergebnis erzielen. Probleme bereiten den Algorithmen wie erwartet eine extreme Konzentration der Präferenzen von Flügen auf einige wenige Slots. Wie zu erwarten war erreichen die Optimierungsalgorithmen bei breiteren Margin-Breiten bessere Ergebnisse als bei schmalen Margin-Breiten. Bei einer Optimierung von 50 Flügen, im Gegensatz zu 100 Flügen, zeigt sich, dass von Jenetics implementierte genetische Algorithmen etwas bessere Ergebnisse erzielen als von OptaPlanner implementierte lokale Suchalgorithmen. Die unterschiedliche Zeitdauer pro Optimierung wirkt sich nicht darauf aus, wie nahe die Ergebnisse am Optimum sind.

Die Ergebnisse der experimentellen Analyse und die umgesetzte Implementierung der Optimierungskomponente können als Grundlage für weitere Arbeiten im Projekt SlotMachine verwendet werden. So gibt es bereits aufbauend auf den Ergebnissen dieser Arbeit erste Experimente, wie man relative Fitnesswerte für die Evaluierung der Zwischenlösungen beim genetischen Algorithmus anstelle von absoluten Fitnesswerten verwenden kann. Die tatsächlichen Fitnesswerte für die einzelnen Lösungen sollen dabei nicht an die Optimierungskomponente zurückgeliefert werden. Stattdessen erhält die Optimierungskomponente eine Rangliste der in jeder Iteration gefundenen Lösungen sowie den maximalen (und eventuell minimalen) Fitnesswert. Dadurch kann die Vertraulichkeit besser gewahrt

werden. Erste Experimente deuten darauf hin, dass genetische Algorithmen nach wie vor gute Ergebnisse liefern.

9 Anhang

In diesem Anhang wird ein Überblick über die Ergebnisse der Hauptstudie in Bezug auf die erreichten Fitnesswerte gegeben. Die Fitnesswerte werde im Vergleich zur Ungarischen Methode, die die Optimallösung darstellt, angezeigt.

Test. Nr.	**Fitnesswerte im Vergleich zum Hungarian Algorithm**							
Konf. Nr. >	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durchschnitt >	**95,77%**	**97,42%**	**91,92%**	**92,01%**	**95,99%**	**97,39%**	**92,26%**	**91,72%**
Rang >	**14**	**9**	**19**	**18**	**13**	**10**	**17**	**20**
1	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
2	92,52%	94,46%	86,09%	85,96%	93,45%	95,61%	84,80%	86,23%
3	99,40%	99,57%	98,99%	98,99%	99,33%	99,57%	98,97%	98,92%
4	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
5	96,94%	98,12%	93,15%	92,31%	96,86%	98,03%	94,00%	93,09%
6	95,13%	96,34%	94,18%	94,20%	95,43%	96,17%	94,06%	94,18%
7	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
8	94,29%	96,28%	88,96%	89,38%	94,68%	95,89%	89,80%	88,50%
9	97,65%	98,52%	96,36%	96,29%	97,91%	98,75%	96,43%	96,50%
10	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durchschnitt >	**95,77%**	**97,42%**	**91,92%**	**92,01%**	**95,99%**	**97,39%**	**92,26%**	**91,72%**
11	90,93%	93,34%	81,54%	81,13%	89,85%	93,38%	81,76%	81,03%
12	93,53%	96,07%	86,48%	86,59%	94,67%	96,68%	85,72%	87,07%
13	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
14	96,11%	97,97%	88,86%	90,21%	96,46%	98,16%	89,73%	88,82%
15	95,55%	97,09%	89,59%	88,30%	95,61%	97,57%	90,98%	89,93%
16	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
17	93,46%	96,33%	84,40%	86,16%	93,77%	96,88%	85,08%	83,11%
18	94,55%	96,91%	87,28%	86,44%	95,07%	97,24%	87,15%	86,69%
19	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
20	83,08%	91,33%	55,32%	61,35%	82,76%	92,16%	62,11%	57,39%
21	84,64%	91,50%	68,78%	69,00%	87,93%	92,38%	73,16%	71,28%
22	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
23	90,49%	95,19%	73,95%	76,08%	90,47%	94,31%	76,56%	75,67%
24	88,06%	94,90%	72,53%	73,10%	85,76%	93,14%	74,50%	73,14%
25	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
26	84,97%	94,08%	67,27%	67,27%	86,17%	93,28%	69,63%	66,88%
27	84,43%	91,01%	70,82%	71,09%	84,63%	92,18%	71,67%	68,96%
28	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
29	100,00%	100,00%	99,97%	99,98%	100,00%	100,00%	99,96%	100,00%
30	99,97%	99,96%	99,92%	99,96%	100,00%	100,00%	99,93%	99,94%
31	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
32	99,35%	99,60%	99,38%	99,41%	99,61%	99,60%	99,30%	99,49%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durchschnitt >	**95,77%**	**97,42%**	**91,92%**	**92,01%**	**95,99%**	**97,39%**	**92,26%**	**91,72%**
33	99,70%	99,74%	99,49%	99,45%	99,62%	99,75%	99,25%	99,59%
34	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
35	99,84%	99,88%	99,78%	99,80%	99,92%	99,86%	99,77%	99,86%
36	99,93%	99,94%	99,64%	99,72%	99,93%	99,89%	99,80%	99,82%
37	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
38	99,99%	99,98%	99,95%	99,91%	100,00%	99,99%	99,93%	99,96%
39	99,97%	99,96%	99,91%	99,87%	99,99%	99,98%	99,91%	99,90%
40	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
41	96,87%	97,47%	96,08%	96,13%	97,00%	97,41%	95,68%	96,04%
42	99,09%	99,20%	98,92%	99,10%	99,04%	99,13%	98,95%	98,85%
43	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
44	99,10%	99,16%	98,95%	99,13%	99,15%	99,17%	98,99%	99,14%
45	99,90%	99,94%	99,51%	99,63%	99,92%	99,97%	99,74%	99,78%
46	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
47	98,72%	99,20%	97,62%	98,59%	99,38%	99,07%	97,12%	98,67%
48	96,33%	96,34%	96,31%	96,32%	96,31%	96,36%	96,32%	96,33%
49	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
50	99,93%	99,70%	98,26%	98,90%	99,85%	99,85%	97,76%	97,93%
51	89,92%	95,04%	89,68%	89,29%	91,73%	95,04%	88,35%	89,02%
52	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
53	99,22%	99,86%	98,61%	98,16%	99,41%	99,69%	98,15%	97,93%
54	93,28%	94,59%	88,50%	89,00%	93,16%	94,16%	88,82%	90,70%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durchschnitt >	**95,77%**	**97,42%**	**91,92%**	**92,01%**	**95,99%**	**97,39%**	**92,26%**	**91,72%**
55	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
56	88,37%	92,12%	79,92%	80,19%	90,45%	93,28%	80,65%	79,44%
57	99,02%	99,19%	98,69%	98,73%	99,01%	99,23%	98,62%	98,63%
58	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
59	94,88%	96,37%	88,91%	89,43%	94,82%	96,11%	88,92%	88,85%
60	94,96%	95,87%	93,92%	94,19%	94,92%	95,55%	93,84%	94,18%
61	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
62	92,46%	95,52%	83,90%	82,46%	92,00%	95,69%	84,89%	80,40%
63	96,26%	97,46%	95,00%	95,28%	96,46%	97,24%	95,16%	94,81%
64	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
65	85,82%	89,30%	72,48%	74,88%	87,63%	89,41%	75,48%	72,55%
66	88,26%	92,70%	76,86%	78,27%	88,19%	93,48%	77,63%	77,35%
67	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
68	91,92%	95,24%	83,35%	83,95%	92,26%	95,05%	85,61%	82,82%
69	93,00%	95,86%	80,75%	83,76%	93,36%	94,95%	85,01%	84,21%
70	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
71	89,61%	93,60%	76,92%	73,18%	88,83%	94,22%	75,13%	72,04%
72	90,13%	94,02%	79,62%	79,61%	91,82%	95,38%	79,95%	76,40%
73	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
74	65,67%	83,98%	36,73%	37,96%	71,06%	80,38%	38,55%	32,64%
75	74,82%	83,37%	55,39%	54,17%	75,50%	83,49%	56,72%	54,26%
76	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durchschnitt >	**95,77%**	**97,42%**	**91,92%**	**92,01%**	**95,99%**	**97,39%**	**92,26%**	**91,72%**
77	80,84%	88,39%	63,74%	57,73%	80,70%	90,68%	63,29%	59,45%
78	79,53%	87,74%	63,02%	63,83%	79,89%	86,03%	66,52%	60,50%
79	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
80	73,93%	87,04%	55,19%	50,42%	77,31%	84,94%	51,81%	46,39%
81	76,75%	85,32%	62,92%	61,85%	77,18%	84,11%	62,04%	59,63%
82	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
83	99,99%	100,00%	99,90%	99,91%	100,00%	100,00%	99,86%	99,90%
84	99,98%	99,95%	99,71%	99,79%	99,96%	99,97%	99,65%	99,73%
85	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
86	99,40%	99,46%	99,01%	99,04%	99,35%	99,67%	99,34%	98,89%
87	99,76%	99,91%	99,24%	99,34%	99,68%	99,79%	99,22%	99,09%
88	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
89	99,92%	99,95%	99,32%	99,50%	99,87%	99,97%	99,53%	99,59%
90	99,82%	99,96%	99,19%	99,41%	99,95%	99,98%	99,33%	99,46%
91	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
92	99,98%	99,97%	99,83%	99,86%	99,99%	100,00%	99,87%	99,85%
93	99,95%	99,95%	99,73%	99,84%	99,91%	99,98%	99,88%	99,83%
94	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
95	96,61%	96,36%	95,31%	95,78%	96,65%	96,89%	95,65%	95,53%
96	99,03%	99,11%	98,38%	98,57%	98,96%	99,07%	98,87%	98,72%
97	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
98	99,26%	99,16%	98,26%	98,55%	99,13%	99,26%	98,40%	98,36%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Durchschnitt >	**95,77%**	**97,42%**	**91,92%**	**92,01%**	**95,99%**	**97,39%**	**92,26%**	**91,72%**
99	99,68%	99,92%	99,03%	99,12%	99,82%	99,93%	99,22%	99,11%
100	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
101	98,68%	99,77%	94,64%	95,40%	97,48%	98,51%	94,49%	95,40%
102	96,60%	96,55%	96,44%	96,54%	96,56%	96,57%	96,60%	96,57%
103	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
104	98,63%	100,00%	91,32%	95,16%	99,25%	99,85%	94,12%	95,88%
105	86,90%	88,76%	84,63%	84,22%	88,20%	88,81%	83,87%	83,65%
106	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
107	97,34%	99,27%	93,36%	94,40%	96,92%	99,56%	95,29%	94,61%
108	89,10%	92,08%	87,63%	86,95%	88,80%	91,21%	87,07%	87,05%

Tabelle 9.1: Hauptstudie, Ergebnisse, Überblick (durchschnittlicher Fitnesswert im Vergleich zum Hungarian Algorithm (Konfigurationsnummer 24)), Jenetics (Teil 1)

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Durchschnitt >	**95,08%**	**96,72%**	**74,23%**	**67,95%**	**95,31%**	**97,12%**	**91,13%**	**91,22%**
Rang >	**16**	**12**	**23**	**24**	**15**	**11**	**22**	**21**
1	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
2	90,69%	93,69%	49,14%	48,50%	92,78%	94,66%	83,76%	84,39%
3	99,19%	99,43%	98,51%	98,51%	99,24%	99,54%	98,95%	98,89%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Durchschnitt >	**95,08%**	**96,72%**	**74,23%**	**67,95%**	**95,31%**	**97,12%**	**91,13%**	**91,22%**
4	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
5	96,26%	97,65%	60,80%	34,50%	96,65%	97,74%	93,36%	92,32%
6	95,10%	95,62%	92,72%	92,72%	94,98%	96,05%	93,97%	93,64%
7	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
8	93,20%	95,18%	51,68%	46,24%	93,24%	96,10%	87,69%	87,92%
9	97,29%	97,90%	95,11%	95,11%	97,26%	98,38%	96,10%	96,13%
10	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
11	88,51%	91,16%	21,25%	5,43%	89,67%	93,27%	80,73%	80,95%
12	92,36%	95,46%	55,37%	54,63%	92,49%	95,94%	86,87%	85,23%
13	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
14	95,31%	96,97%	45,61%	3,96%	95,27%	97,27%	89,05%	88,78%
15	93,60%	96,48%	43,12%	39,48%	94,26%	96,93%	88,16%	86,76%
16	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
17	92,32%	95,18%	25,56%	-6,79%	92,60%	96,34%	84,92%	83,63%
18	93,29%	95,82%	42,96%	41,35%	94,14%	96,44%	83,03%	84,90%
19	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
20	76,11%	89,55%	-45,80%	-88,96%	80,92%	91,08%	55,21%	60,72%
21	80,89%	90,38%	11,36%	9,52%	81,80%	91,44%	65,18%	72,71%
22	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
23	87,16%	92,02%	7,23%	-32,09%	88,56%	94,00%	70,43%	72,28%
24	85,81%	91,35%	22,75%	18,62%	84,98%	93,59%	70,68%	70,57%
25	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Durchschnitt >	**95,08%**	**96,72%**	**74,23%**	**67,95%**	**95,31%**	**97,12%**	**91,13%**	**91,22%**
26	83,89%	89,40%	-21,09%	-56,81%	83,49%	92,41%	64,03%	62,42%
27	83,94%	88,97%	15,82%	14,84%	82,56%	89,50%	64,39%	67,30%
28	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
29	100,00%	100,00%	98,67%	98,42%	100,00%	100,00%	99,96%	99,97%
30	99,98%	100,00%	98,75%	98,57%	99,97%	99,99%	99,95%	99,88%
31	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
32	99,43%	99,63%	94,67%	93,80%	99,56%	99,61%	99,37%	99,41%
33	99,69%	99,78%	96,48%	96,41%	99,71%	99,61%	99,35%	99,37%
34	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
35	99,89%	99,84%	96,00%	95,69%	99,83%	99,84%	99,75%	99,79%
36	99,93%	99,94%	96,89%	96,63%	99,94%	99,94%	99,50%	99,61%
37	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
38	99,97%	99,98%	98,84%	98,83%	99,94%	99,95%	99,91%	99,91%
39	99,95%	99,98%	98,63%	98,62%	99,96%	99,99%	99,80%	99,89%
40	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
41	97,05%	97,25%	92,11%	91,96%	96,84%	96,92%	95,27%	96,05%
42	98,96%	99,16%	96,26%	95,78%	98,96%	99,19%	98,81%	98,96%
43	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
44	99,20%	99,17%	95,66%	95,59%	99,10%	99,13%	98,83%	98,89%
45	99,82%	99,89%	96,63%	96,52%	99,89%	99,93%	99,33%	99,70%
46	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
47	98,98%	99,26%	54,82%	9,61%	99,21%	99,67%	98,06%	97,96%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Durchschnitt >	**95,08%**	**96,72%**	**74,23%**	**67,95%**	**95,31%**	**97,12%**	**91,13%**	**91,22%**
48	96,33%	96,34%	95,62%	95,59%	96,31%	96,33%	96,28%	96,34%
49	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
50	99,74%	99,78%	51,25%	9,91%	99,91%	99,85%	96,78%	97,00%
51	90,19%	92,07%	80,94%	80,73%	91,74%	91,83%	84,24%	86,02%
52	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
53	99,44%	99,23%	44,77%	8,09%	99,55%	99,90%	94,68%	97,04%
54	91,18%	95,00%	83,41%	83,39%	90,69%	94,30%	87,63%	88,29%
55	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
56	87,92%	90,97%	55,69%	54,91%	86,80%	91,75%	79,86%	79,53%
57	98,94%	99,08%	98,40%	98,37%	98,87%	99,13%	98,53%	98,60%
58	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
59	93,79%	95,90%	54,44%	35,04%	94,07%	96,76%	88,18%	87,87%
60	94,74%	95,38%	93,21%	93,20%	94,72%	95,45%	93,85%	93,87%
61	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
62	91,31%	92,98%	45,71%	38,84%	90,52%	94,28%	80,65%	81,05%
63	96,04%	96,81%	94,26%	94,20%	96,09%	96,90%	95,15%	95,17%
64	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
65	85,02%	88,08%	18,37%	13,91%	85,40%	90,99%	71,61%	71,11%
66	85,43%	91,21%	56,35%	54,44%	87,39%	92,30%	77,41%	76,09%
67	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
68	91,07%	94,39%	37,50%	3,14%	91,30%	95,31%	82,95%	81,32%
69	91,75%	93,77%	43,32%	40,98%	90,16%	94,44%	81,07%	81,45%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Durchschnitt >	**95,08%**	**96,72%**	**74,23%**	**67,95%**	**95,31%**	**97,12%**	**91,13%**	**91,22%**
70	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
71	85,93%	91,51%	18,69%	0,31%	87,60%	92,55%	75,02%	74,34%
72	88,17%	92,87%	44,80%	42,91%	88,01%	93,49%	77,10%	76,76%
73	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
74	59,41%	77,30%	-65,97%	-93,81%	66,23%	82,40%	28,50%	30,53%
75	70,54%	78,39%	6,64%	6,12%	70,00%	83,13%	52,68%	48,28%
76	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
77	76,51%	85,86%	0,61%	-31,81%	79,63%	87,08%	57,68%	58,94%
78	74,27%	82,82%	21,86%	17,59%	74,90%	85,41%	58,41%	60,31%
79	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
80	70,23%	82,96%	-20,65%	-46,70%	71,12%	80,04%	45,22%	45,56%
81	73,61%	80,48%	18,84%	18,20%	74,08%	83,61%	57,96%	56,49%
82	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
83	99,97%	100,00%	98,40%	98,28%	99,99%	100,00%	99,86%	99,89%
84	99,89%	99,99%	98,62%	98,49%	99,88%	99,92%	99,75%	99,53%
85	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
86	99,18%	99,44%	94,80%	94,37%	99,48%	99,23%	98,76%	98,78%
87	99,59%	99,75%	96,19%	96,21%	99,63%	99,86%	98,69%	99,11%
88	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
89	99,85%	99,97%	94,95%	94,78%	99,86%	99,94%	99,37%	99,43%
90	99,79%	99,93%	96,52%	96,30%	99,76%	99,95%	99,02%	99,20%
91	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Durchschnitt >	**95,08%**	**96,72%**	**74,23%**	**67,95%**	**95,31%**	**97,12%**	**91,13%**	**91,22%**
92	99,95%	99,99%	99,20%	99,14%	99,98%	99,98%	99,88%	99,79%
93	99,88%	99,91%	98,71%	98,59%	99,86%	99,98%	99,71%	99,58%
94	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
95	96,32%	96,60%	92,45%	92,40%	96,33%	96,64%	95,09%	95,45%
96	98,82%	98,98%	95,69%	95,67%	99,03%	99,03%	98,29%	98,49%
97	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
98	98,99%	99,24%	93,94%	93,51%	99,05%	99,28%	97,84%	98,17%
99	99,72%	99,89%	95,65%	95,24%	99,79%	99,89%	98,79%	98,93%
100	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
101	97,85%	98,82%	51,47%	9,11%	98,08%	98,98%	94,57%	93,14%
102	96,50%	96,59%	95,76%	95,75%	96,34%	96,55%	96,36%	96,36%
103	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
104	99,02%	99,40%	43,43%	7,94%	98,74%	99,73%	95,15%	91,73%
105	87,09%	89,42%	80,92%	80,82%	86,94%	89,02%	83,54%	83,67%
106	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
107	98,44%	99,44%	41,43%	4,99%	99,47%	99,48%	95,33%	93,59%
108	88,27%	89,45%	84,44%	84,46%	88,42%	89,94%	86,46%	86,05%

Tabelle 9.2: Hauptstudie, Ergebnisse, Überblick (durchschnittlicher Fitnesswert im Vergleich zum Hungarian Algorithm (Konfigurationsnummer 24)), Jenetics (Teil 2)

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Durchschnitt >	**99,03%**	**99,41%**	**99,64%**	**99,60%**	**99,57%**	**99,67%**	**99,55%**	**100,00%**
Rang >	**8**	**7**	**3**	**4**	**5**	**2**	**6**	**1**
1	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
2	96,39%	97,35%	99,20%	97,57%	97,50%	99,22%	97,21%	100,00%
3	99,92%	99,92%	100,00%	99,93%	99,93%	99,93%	99,86%	100,00%
4	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
5	99,49%	99,49%	99,89%	99,57%	99,42%	99,78%	99,29%	100,00%
6	98,85%	98,91%	99,77%	99,76%	99,33%	99,44%	99,04%	100,00%
7	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
8	96,87%	98,46%	99,20%	96,78%	98,34%	99,33%	98,60%	100,00%
9	99,58%	99,70%	99,73%	99,47%	99,82%	99,75%	99,72%	100,00%
10	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
11	95,96%	97,28%	96,60%	98,51%	97,55%	96,17%	96,27%	100,00%
12	98,31%	99,67%	98,23%	99,02%	99,04%	100,00%	99,52%	100,00%
13	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
14	99,84%	99,89%	99,86%	99,65%	99,56%	99,76%	99,81%	100,00%
15	98,77%	99,57%	100,00%	100,00%	99,55%	99,22%	99,91%	100,00%
16	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
17	99,08%	97,72%	98,84%	99,50%	99,49%	99,52%	98,71%	100,00%
18	99,30%	99,48%	99,76%	99,58%	99,69%	99,81%	98,69%	100,00%
19	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
20	99,24%	99,26%	99,79%	99,79%	98,22%	100,00%	96,05%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Durchschnitt >	**99,03%**	**99,41%**	**99,64%**	**99,60%**	**99,57%**	**99,67%**	**99,55%**	**100,00%**
21	98,42%	99,96%	100,00%	100,00%	100,00%	99,97%	99,89%	100,00%
22	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
23	99,96%	99,98%	99,99%	99,99%	100,00%	100,00%	99,98%	100,00%
24	99,71%	100,00%	100,00%	100,00%	100,00%	99,94%	99,99%	100,00%
25	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
26	98,53%	99,95%	99,96%	99,80%	99,63%	99,29%	99,51%	100,00%
27	99,64%	98,85%	99,91%	99,87%	99,64%	99,99%	99,51%	100,00%
28	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
29	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	99,99%	100,00%
30	100,00%	100,00%	100,00%	100,00%	99,95%	100,00%	100,00%	100,00%
31	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
32	99,69%	99,68%	100,00%	99,66%	99,76%	99,99%	99,63%	100,00%
33	99,78%	100,00%	100,00%	100,00%	99,56%	100,00%	99,87%	100,00%
34	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
35	99,85%	99,98%	99,98%	99,96%	99,94%	99,93%	99,84%	100,00%
36	99,67%	99,94%	100,00%	100,00%	100,00%	100,00%	99,87%	100,00%
37	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
38	99,93%	99,98%	100,00%	100,00%	99,93%	99,96%	99,98%	100,00%
39	99,98%	100,00%	100,00%	100,00%	99,90%	100,00%	99,96%	100,00%
40	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
41	96,07%	96,99%	100,00%	96,74%	96,25%	98,76%	98,70%	100,00%
42	99,08%	99,36%	99,36%	99,28%	99,21%	99,32%	100,00%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Durchschnitt >	**99,03%**	**99,41%**	**99,64%**	**99,60%**	**99,57%**	**99,67%**	**99,55%**	**100,00%**
43	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
44	99,00%	99,38%	100,00%	99,45%	99,11%	99,90%	99,38%	100,00%
45	99,87%	99,95%	99,97%	99,91%	99,90%	100,00%	99,97%	100,00%
46	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
47	99,17%	98,64%	100,00%	99,17%	99,17%	99,98%	99,17%	100,00%
48	96,30%	97,49%	97,49%	96,36%	96,30%	96,36%	98,99%	100,00%
49	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
50	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
51	88,42%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
52	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
53	99,01%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
54	93,82%	96,19%	96,30%	100,00%	100,00%	100,00%	100,00%	100,00%
55	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
56	97,15%	99,77%	99,98%	99,46%	98,65%	98,60%	99,28%	100,00%
57	99,99%	99,99%	99,99%	99,99%	99,99%	99,99%	99,90%	100,00%
58	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
59	99,23%	99,52%	99,91%	99,71%	99,42%	99,93%	99,44%	100,00%
60	97,74%	97,98%	99,87%	99,83%	99,91%	99,86%	98,92%	100,00%
61	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
62	96,95%	98,82%	98,96%	98,14%	98,78%	98,18%	98,68%	100,00%
63	99,49%	99,28%	99,50%	99,88%	99,93%	99,91%	99,16%	100,00%
64	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Durchschnitt >	**99,03%**	**99,41%**	**99,64%**	**99,60%**	**99,57%**	**99,67%**	**99,55%**	**100,00%**
65	92,77%	94,15%	95,53%	92,85%	94,44%	95,47%	94,94%	100,00%
66	99,92%	99,43%	98,33%	99,19%	99,19%	99,19%	98,51%	100,00%
67	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
68	99,79%	99,74%	99,96%	99,86%	99,80%	99,69%	99,77%	100,00%
69	99,13%	99,03%	99,25%	99,52%	99,98%	98,77%	99,02%	100,00%
70	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
71	98,32%	98,70%	99,56%	99,38%	98,79%	98,81%	98,61%	100,00%
72	99,20%	99,84%	99,17%	99,53%	99,91%	99,66%	99,17%	100,00%
73	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
74	97,99%	97,36%	99,74%	99,17%	99,78%	99,37%	97,13%	100,00%
75	100,00%	98,35%	100,00%	99,92%	100,00%	100,00%	99,24%	100,00%
76	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
77	99,98%	99,94%	99,98%	99,99%	100,00%	99,99%	99,98%	100,00%
78	100,00%	100,00%	99,99%	100,00%	100,00%	100,00%	100,00%	100,00%
79	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
80	99,44%	99,81%	99,98%	99,43%	99,74%	99,44%	99,38%	100,00%
81	99,41%	98,91%	98,88%	99,50%	100,00%	99,27%	99,00%	100,00%
82	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
83	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
84	99,87%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
85	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
86	99,69%	99,87%	100,00%	99,94%	99,61%	99,76%	99,76%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Durchschnitt >	**99,03%**	**99,41%**	**99,64%**	**99,60%**	**99,57%**	**99,67%**	**99,55%**	**100,00%**
87	99,76%	100,00%	100,00%	99,98%	99,84%	99,79%	99,99%	100,00%
88	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
89	99,89%	99,99%	100,00%	99,84%	99,75%	99,99%	99,94%	100,00%
90	100,00%	100,00%	99,94%	100,00%	100,00%	100,00%	99,99%	100,00%
91	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
92	99,90%	99,96%	100,00%	99,93%	99,90%	99,82%	99,90%	100,00%
93	99,89%	100,00%	100,00%	99,77%	99,77%	99,89%	99,99%	100,00%
94	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
95	95,61%	96,39%	99,78%	99,10%	96,15%	99,18%	98,00%	100,00%
96	98,91%	99,37%	99,44%	99,40%	98,96%	99,36%	99,92%	100,00%
97	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
98	99,29%	99,39%	99,95%	99,91%	99,88%	99,23%	99,88%	100,00%
99	99,95%	100,00%	100,00%	99,99%	99,94%	99,99%	99,94%	100,00%
100	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
101	99,16%	98,25%	100,00%	98,24%	99,62%	99,97%	98,17%	100,00%
102	96,63%	96,65%	96,65%	96,65%	96,51%	96,63%	99,62%	100,00%
103	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
104	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
105	88,79%	95,14%	95,14%	100,00%	99,99%	100,00%	100,00%	100,00%
106	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
107	99,47%	99,47%	99,47%	99,56%	99,91%	99,47%	99,52%	100,00%

Test. Nr.	Fitnesswerte im Vergleich zum Hungarian Algorithm							
Konf. Nr. >	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
Durchschnitt >	**99,03%**	**99,41%**	**99,64%**	**99,60%**	**99,57%**	**99,67%**	**99,55%**	**100,00%**
108	94,11%	98,48%	98,48%	100,00%	99,67%	99,67%	99,67%	100,00%

Tabelle 9.3: Hauptstudie, Ergebnisse, Überblick (durchschnittlicher Fitnesswert im Vergleich zum Hungarian Algorithm (Konfigurationsnummer 24)), OptaPlanner

Quellenverzeichnis

[1] Mansour Alssager, Zulaiha Ali Othman, and Masri Ayob. "Cheapest Insertion Constructive Heuristic based on Two Combination Seed Customer Criterion for the Capacitated Vehicle Routing Problem". en-gb. In: *International Journal on Advanced Science, Engineering and Information Technology* 7.1 (2017). Publisher: INSIGHT - Indonesian Society for Knowledge and Human Development, pp. 207–214. ISSN: 2088-5334. URL: http://ijaseit.insightsociety.org/index.php?option=com_content&view=article&id=9&Itemid=1&article_id=1792.

[2] A. Amuthan and K. Deepa Thilak. "Survey on Tabu Search meta-heuristic optimization". In: *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. Oct. 2016, pp. 1539–1543. DOI: 10.1109/SCOPES.2016.7955697.

[3] Aris Anagnostopoulos et al. "A simulated annealing approach to the traveling tournament problem". en. In: *Journal of Scheduling* 9.2 (Apr. 2006), pp. 177–193. ISSN: 1099-1425. DOI: 10.1007/s10951-006-7187-8. URL: https://doi.org/10.1007/s10951-006-7187-8.

[4] Cynthia Barnhart et al. "Demand and capacity management in air transportation". en. In: *EURO Journal on Transportation and Logistics* 1.1 (June 2012), pp. 135–155. ISSN: 2192-4384. DOI: 10.1007/s13676-012-0006-9. URL: https://doi.org/10.1007/s13676-012-0006-9.

[5] Adil Baykasoglu. "Design optimization with chaos embedded great deluge algorithm". en. In: *Applied Soft Computing* 12.3 (Mar. 2012), pp. 1055–1067. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2011.11.018. URL: https://www.sciencedirect.com/science/article/pii/S1568494611004601.

[6] Mohammed Azmi Al-Betar et al. "β-Hill Climbing Algorithm for Sudoku Game". In: *2017 Palestinian International Conference on Information and Communication Technology (PICICT)*. May 2017, pp. 84–88. DOI: 10.1109/PICICT.2017.11.

[7] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. "A survey on optimization metaheuristics". In: *Information Sciences* 237 (2013), pp. 82–117. DOI: 10.1016/j.ins.2013.02.041. URL: https://doi.org/10.1016/j.ins.2013.02.041.

[8] Edmund K. Burke and Yuri Bykov. "The late acceptance Hill-Climbing heuristic". en. In: *European Journal of Operational Research* 258.1 (Apr. 2017), pp. 70–78. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2016.07.012. URL: https://www.sciencedirect.com/science/article/pii/S0377221716305495.

[9] Yuri Bykov and Sanja Petrovic. "A Step Counting Hill Climbing Algorithm applied to University Examination Timetabling". en. In: *Journal of Scheduling* 19.4 (Aug. 2016), pp. 479–492. ISSN: 1099-1425. DOI: 10.1007/s10951-016-0469-x. URL: https://doi.org/10.1007/s10951-016-0469-x.

[10] *Constraint satisfaction solver (Java™, Open Source)*. en. URL: https://www.optaplanner.org/ (visited on 09/20/2021).

[11] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. ISBN: 9781107043053. URL: http://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB%5C&isbn=9781107043053.

[12] Caichang Ding, Lin Chen, and Baorong Zhong. "Exploration of intelligent computing based on improved hybrid genetic algorithm". In: *Cluster Computing* 22.Supplement (2019), pp. 9037–9045. DOI: 10.1007/s10586-018-2049-7. URL: https://doi.org/10.1007/s10586-018-2049-7.

[13] George H. G. Fonseca, Haroldo G. Santos, and Eduardo G. Carrano. "Late acceptance hill-climbing for high school timetabling". en. In: *Journal of Scheduling* 19.4 (Aug. 2016), pp. 453–465. ISSN: 1099-1425. DOI: 10.1007/s10951-015-0458-5. URL: https://doi.org/10.1007/s10951-015-0458-5.

[14] Wolfgang Geithner et al. "Genetic Algorithms for Machine Optimization in the Fair Control System Environment". en. In: JACOW Publishing, Geneva, Switzerland, June 2018, pp. 4712–4715. ISBN: 978-3-95450-184-7. DOI: 10.18429/JACoW-IPAC2018-THPML028. URL: https://accelconf.web.cern.ch/ipac2018/doi/JACoW-IPAC2018-THPML028.html.

[15] Fred Glover, James P. Kelly, and Manuel Laguna. “Genetic algorithms and tabu search: Hybrids for optimization”. en. In: *Computers & Operations Research*. Genetic Algorithms 22.1 (Jan. 1995), pp. 111–134. ISSN: 0305-0548. DOI: 10.1016/0305-0548(93)E0023-M. URL: https://www.sciencedirect.com/science/article/pii/0305054893E0023M.

[16] Fred W. Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & Operations Research* 13.5 (1986), pp. 533–549. DOI: 10.1016/0305-0548(86)90048-1. URL: https://doi.org/10.1016/0305-0548(86)90048-1.

[17] Fred W. Glover et al. “Construction heuristics for the asymmetric TSP”. In: *Eur. J. Oper. Res.* 129.3 (2001), pp. 555–568. DOI: 10.1016/S0377-2217(99)00468-3. URL: https://doi.org/10.1016/S0377-2217(99)00468-3.

[18] Pengfei Guo, Xuezhi Wang, and Yingshi Han. “The enhanced genetic algorithms for the optimization design”. In: *2010 3rd International Conference on Biomedical Engineering and Informatics*. Vol. 7. Oct. 2010, pp. 2990–2994. DOI: 10.1109/BMEI.2010.5639829.

[19] Meryem Hafidi, Mohamed Benaddy, and Salah-ddine Krit. “Review of optimization and automation of air traffic control systems”. In: *Proceedings of the Fourth International Conference on Engineering & MIS 2018*. ICEMIS ’18. New York, NY, USA: Association for Computing Machinery, June 2018, pp. 1–7. ISBN: 978-1-4503-6392-1. DOI: 10.1145/3234698.3234708. URL: https://doi.org/10.1145/3234698.3234708.

[20] Billy Josefsson et al. “Scheduling air traffic controllers at the remote tower center”. In: *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. Sept. 2017, pp. 1–10. DOI: 10.1109/DASC.2017.8102018.

[21] Gastón Keller et al. “An analysis of first fit heuristics for the virtual machine relocation problem”. In: *8th International Conference on Network and Service Management, CNSM 2012, Las Vegas, NV, USA, October 22-26, 2012*. IEEE, 2012, pp. 406–413. URL: https://ieeexplore.ieee.org/document/6380049/.

[22] Byung-In Kim and Juyoung Wy. “Last two fit augmentation to the well-known construction heuristics for one-dimensional bin-packing problem: an empirical study”. en. In: *The International Journal of Advanced Manufacturing Technology* 50.9-12 (Oct. 2010), pp. 1145–1152. DOI: 10.1007/s00170-010-2572-z. URL: http://link.springer.com/10.1007/s00170-010-2572-z.

[23] H. W. Kuhn. “The Hungarian method for the assignment problem”. en. In: *Naval Research Logistics (NRL)* 52.1 (2005), pp. 7–21. ISSN: 1520-6750. DOI: 10.1002/nav.20053. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.20053 (visited on 08/26/2021).

[24] Sean Luke. *Essentials of Metaheuristics*. en. second. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/. Lulu, 2013. ISBN: 978-1-300-54962-8.

[25] *OptaPlanner*. Sept. 2021. URL: https://github.com/kiegroup/optaplanner/tree/8.1.0.Final (visited on 09/20/2021).

[26] *OptaPlanner User Guide*. URL: https://docs.optaplanner.org/latestFinal/optaplanner-docs/html_single/ (visited on 04/03/2021).

[27] Nadine Pilon, Laurent Guichard, and Katherine Cliff. “Reducing Impact of Delays using Airspace User- Driven Flight Prioritisation”. en. In: (2019), p. 8.

[28] Nadine Pilon et al. “Improved Flexibility and Equity for Airspace Users During Demand-capacity Imbalance”. en. In: (2016), p. 8.

[29] Dian Rachmawati and Wilyanto. “Implementation of Modified Cheapest Insertion Heuristic on Generating Medan City Tourism Route”. en. In: *Journal of Physics: Conference Series* 1566 (June 2020). Publisher: IOP Publishing, p. 012076. ISSN: 1742-6596. DOI: 10.1088/1742-6596/1566/1/012076. URL: https://doi.org/10.1088/1742-6596/1566/1/012076.

[30] Sergio Ruiz et al. “A New Air Traffic Flow Management User-Driven Prioritisation Process for Low Volume Operator in Constraint: Simulations and Results”. en. In: *Journal of Advanced Transportation* 2019 (Apr. 2019), e1208279. ISSN: 0197-6729. DOI: 10.1155/2019/1208279. URL: https://www.hindawi.com/journals/jat/2019/1208279/.

[31] Haroldo G. Santos et al. “Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem”. en. In: *International Transactions in Operational Research* 26.2 (2019), pp. 707–724. ISSN: 1475-3995. DOI: 10.1111/itor.12316. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12316.

[32] Azin Shamshirgaran, Hamed Javidi, and Dan Simon. “Evolutionary Algorithms for Multi-Objective Optimization of Drone Controller Parameters”. In: *CoRR* abs/2105.08650 (2021). arXiv: 2105.08650. URL: https://arxiv.org/abs/2105.08650.

[33] J W Smeltink, M J Soomer, and P R de Waal. “An Optimisation Model for Airport Taxi Scheduling”. en. In: (June 2004), p. 25.

[34] Kenneth Sörensen. “Metaheuristics - the metaphor exposed”. In: *International Transactions in Operational Research* 22.1 (2015), pp. 3–18. DOI: 10.1111/itor.12001. URL: https://doi.org/10.1111/itor.12001.

[35] Kenneth Sörensen and Fred W. Glover. “Metaheuristics”. en. In: *Encyclopedia of Operations Research and Management Science*. Ed. by Saul I. Gass and Michael C. Fu. Boston, MA: Springer US, 2013, pp. 960–970. ISBN: 978-1-4419-1153-7. DOI: 10.1007/978-1-4419-1153-7_1167. URL: https://doi.org/10.1007/978-1-4419-1153-7_1167 (visited on 09/20/2021).

[36] Kevin Stern. *KevinStern/software-and-algorithms*. Sept. 17, 2021. URL: https://github.com/KevinStern/software-and-algorithms/blob/34c07b6ac3030422c71f7b57e693c56e87de1b61/src/main/java/blogspot/software_and_algorithms/stern_library/optimization/HungarianAlgorithm.java (visited on 09/22/2021).

[37] Shinsuke Tamura et al. “Feasiblity of Hungarian algorithm based Scheduling”. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Istanbul, Turkey, 10-13 October 2010*. IEEE, 2010, pp. 1185–1190. DOI: 10.1109/ICSMC.2010.5642375. URL: https://doi.org/10.1109/ICSMC.2010.5642375.

[38] Franz Wilhelmstötter. *Jenetics - Library User's Manual 6.2*. URL: https://jenetics.io/manual/manual-6.2.0.pdf (visited on 08/26/2021).

[39] Franz Wilhelmstötter. *Jenetics: Java Genetic Algorithm Library*. en. Apr. 2021. URL: https://jenetics.io/ (visited on 04/03/2021).

[40] Franz Wilhelmstötter. *Release v6.1.0 · jenetics/jenetics*. en. 2020. URL: https://github.com/jenetics/jenetics/releases/tag/v6.1.0 (visited on 09/20/2021).

[41] Amina El Yaagoubi, Ahmed El Hilali Alaoui, and Jaouad Boukachour. “A Heuristic Approach For Solving Container-on-Barge Stowage Planning Problem Based On Bin-Packing First-Fit Algorithm”. In: *2020 5th International Conference on Logistics*

Operations Management (GOL). Oct. 2020, pp. 1–6. DOI: 10.1109/GOL49479.2020.9314748.

[42] Biao Yuan, Chaoyong Zhang, and Xinyu Shao. "A late acceptance hill-climbing algorithm for balancing two-sided assembly lines with multiple constraints". en. In: *Journal of Intelligent Manufacturing* 26.1 (Feb. 2015), pp. 159–168. ISSN: 1572-8145. DOI: 10.1007/s10845-013-0770-x. URL: https://doi.org/10.1007/s10845-013-0770-x.

[43] Ning Zhou et al. "Application of an improved Hungarian algorithm in semantic WEB service discovery". In: *2008 7th World Congress on Intelligent Control and Automation*. June 2008, pp. 5422–5427. DOI: 10.1109/WCICA.2008.4593813.