

Diplomarbeit

**Evaluation von Optimierungsalgorithmen
zur Tourplanung im Hamburger Hafen**

von

Cheng Wang

November 2011

Betreut von

Prof. Dr. Sibylle Schupp

Rainer Marrone

Technische Universität Hamburg-Harburg

Institut für Softwaresysteme

Eidesstattliche Erklärung

Ich, CHENG WANG (Student im Studiengang Informatik-Ingenieurwesen an der Technischen Universität Hamburg-Harburg, Matr.-Nr. 29074), versichere an Eides statt, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Hamburg, den 2. November 2011

Cheng Wang

Inhaltsverzeichnis

1 Einleitung	3
1.1 Motivation.....	3
1.2 Ziel der Diplomarbeit.....	4
2 Optimierungsalgorithmen	7
2.1 Optimierungsproblem.....	7
2.2 Ameisenalgorithmus.....	8
2.2.1 Grundidee	8
2.2.2 Algorithmen mit künstlichen Ameisen	9
2.3 Genetischer Algorithmus.....	14
2.4 Erschöpfende Suche	22
2.5 NSGA2.....	23
3 Analyse der Tourplanung.....	31
3.1 Problemstellung	31
3.2 Optimierungskriterien	33
3.2.1 Schnellstes Kriterium	33
3.2.2 Ökonomisches Kriterium.....	34
3.3 Singleobjektive versus multiobjektive Optimierung.....	34
3.4 Generierung von Testdaten.....	35
3.4.1 Testdaten von Schiffen	35
3.4.2 Testdaten von Terminals	36
3.5 OPT4J Framework	36
3.6 Implementierung	38

4 Parameterbestimmung der Optimierungsalgorithmen	41
4.1 Parameter von Ameisenalgorithmus	41
4.2 Parameter von genetischem Algorithmus	43
4.3 Parameter von NSGA2	44
4.4 Methode der Parameterbestimmung	44
4.5 Simulation	45
4.5.1 Erschöpfende Suche	46
4.5.2 Ameisenalgorithmus.....	47
4.5.3 Genetischer Algorithmus.....	54
4.5.4 NSGA2.....	61
5 Evaluation.....	63
5.1 Metriken für die Evaluation.....	63
5.2 Methodiken zur Evaluation	64
5.3 Evaluation	65
5.3.1 Erschöpfende Suche	65
5.3.2 Heuristiken	66
5.4 Vergleich der Optimierungsalgorithmen	71
6 Zusammenfassung und Ausblick	75
Literaturverzeichnis.....	77

1 Einleitung

1.1 Motivation

Logistik ist ein wichtiger Schlüssel für den globalen Wettbewerb. Weil der Warentransport auf dem Wasser meistens günstig ist, ist der Transport über Seeweg sehr wichtig für Logistik. Ein anderer Vorteil ist, dass ein Frachtschiff sehr viel Mengen von Volumen und Gewicht transportieren kann. Deswegen hat Seefracht in den globalisierten Märkten eine sehr große Bedeutung. Aus Sicht der Logistik ist die Seefracht kostengünstig, nur wenn Schiffe auf dem Meer fahren. Auf diesem Grund sollen die Be- und Entladezeit und die Liegezeit des Schiffes im Hafen verringert werden, um die Effizienz der Seefracht zu erhöhen. Als ein wichtiger Teil der Logistik ist die Hafententwicklung, wie zum Beispiel im Bereich der Automatisierungstechnik, Informationstechnologie und künstlicher Intelligenz, nicht nur eine wichtige Methode für die Steigerung der Wettbewerbsfähigkeit des Hafens, sondern auch der Schlüssel für die Kostensenkung und die Effizienzerhöhung.

Im Hafen gibt es viele verschiedene Terminals und Liegeplätze. Wegen Beladen und Entladen muss ein Schiff normalerweise nicht nur ein Terminal besuchen. Für die Terminals haben verschiedene Schiffe unterschiedliche Prioritäten. Um die Be- und Entladezeit und die Liegezeit des Schiffs im Hafen zu verringern, soll zuerst eine effiziente Tour gefunden werden. Deswegen wird eine gute Reihenfolge der zu bedienenden Terminals beim Be- und Entladen gesucht, damit alle Schiffe möglichst kurze Zeit warten müssen. Früher wurde alles manuell gemacht. Weil es zu viele unterschiedliche mögliche Touren gibt, braucht die Tourplanung viel Zeit. Und wenn ein Schiff viele Terminals besuchen möchte, kann nicht immer eine gute Tour gefunden werden. Aber mit dem Computer kann alles schnell und einfach gemacht werden. Eine gute Tour kann auch mit dem Computer immer gefunden

werden. Durch künstliche Intelligenz kann die Tourplanung effizienter gelöst werden.

1.2 Ziel der Diplomarbeit

Zurzeit gibt es keine Unterstützung für die Optimierung dieser Tätigkeiten. Schiffe rufen zuerst die Terminals an, um ihre Tourplanung manuell zu bestimmen. Die Mitarbeiter in den Terminals verarbeiten verschiedene Schiffstypen mit unterschiedlicher Priorität und geben dem kommenden Schiff einige Vorschläge.

Der gesamte Plan ist die Implementierung eines Multiagentensystems, um die Tourplanung effizient zu lösen. Jeder Agent plant seine Tour allein anhand der Informationen, die von den Terminals gegeben werden.

Ziel dieser Diplomarbeit ist verschiedene Optimierungsalgorithmen zu finden und zu evaluieren, um die Tourplanung und die Tour selber effizienter zu gestalten. Optimierungsalgorithmen unterscheiden sich in der Anzahl der Kriterien, die für die Optimierung berücksichtigt werden, die Größe des rechtzeitig behandelten Suchraums und die Qualität der Ergebnisse. Deswegen müssen zuerst die Optimierungskriterien bestimmt werden. Danach sollen die Algorithmen für die Tourplanung simuliert werden. Anschließend werden die besten Parameter für die Algorithmen bestimmt. Abschließend werden die Algorithmen mit Hilfe von zwei Metriken, wie die Laufzeit der Algorithmen und Qualität der Ergebnisse bewertet.

Einleitend in die Thematik wird in dem Kapitel 2 ein Überblick über den aktuellen Stand der Technik bezüglich Optimierungsalgorithmen gegeben. Hier erfolgt eine kurze Einführung von Optimierungsproblem. Anschließend wird auf die Thematik der Optimierungsalgorithmen detailliert eingegangen. Im Kapitel 3 wird zuerst das Problem von der Tourplanung analysiert. Danach werden die

Optimierungskriterien bestimmt. Dann wird die Generierung von Testdaten beschrieben, damit Algorithmen getestet werden können. Anschließend wird das verwendete Java-Framework vorgestellt. Zuletzt wird die Implementierung erklärt. Zu Beginn werden in Kapitel 4 die Parameter von den Optimierungsalgorithmen erläutert. Anschließend werden die Parameter mit verschiedenen Werten getestet. Und der optimale Wert wird bestimmt. Den Anfang von Kapitel 5 bilden Methodiken zur Evaluation der Optimierungsalgorithmen und den dafür notwendigen Metriken. Im weiteren Verlauf wird die Fähigkeit zur Simulation und Evaluation von Optimierungsalgorithmen dargestellt. Eine Zusammenfassung der wichtigsten Ergebnisse dieser Diplomarbeit liefert das Kapitel 6 und gibt einen Ausblick auf mögliche weiterführende Arbeiten.

2 Optimierungsalgorithmen

Elementares Wissen über die Thematik Optimierungsalgorithmen wird in diesem Kapitel vermittelt. Beginnend wird im Abschnitt 2.1 auf Optimierungsproblem eingegangen. Die Definition und die Klassifikation des Optimierungsproblems werden erklärt. Anschließend werden im Abschnitt 2.2, 2.3, 2.4 und 2.5 der Ameisenalgorithmus, genetischer Algorithmus, erschöpfende Suche und NSGA2 vorgestellt, durch die ein Problem optimiert werden kann.

2.1 Optimierungsproblem

Manche Probleme haben nicht nur eine Lösung sondern mehrere Lösungen. Optimierungsproblem ist solches Problem. Optimierungsproblem hat immer eine oder mehrere Zielfunktionen und vielleicht auch Nebenbedingungen. Eine Zielfunktion wird unter die Nebenbedingungen minimiert. [KS]

$$\begin{aligned} \min f(x) \\ \text{s.d. } g_j(x) &= 0, \quad j = 1, \dots, m_e \\ g_j(x) &\geq 0, \quad j = m_e + 1, \dots, m \\ x_l &\leq x \leq x_u, \\ x &\in \mathfrak{R}^n \end{aligned}$$

lineares Optimierungsproblem:

Wenn die Zielfunktion linear ist, heißt das Optimierungsproblem linear wie zum Beispiel $f(x) = C^T x$. Die minimale Lösung kann einfach mit Hilfe von den Nebenbedingungen berechnet werden.

nichtlineares Optimierungsproblem:

Im Vergleich zu den linearen Optimierungsproblemen sind nichtlineare Optimierungsprobleme noch schwieriger. Ein nichtlineares Optimierungsproblem besteht darin, dass seine Zielfunktion nichtlinear ist. Und es gibt viele lokale Optima und ein globales Optimum. Weil die Zielfunktion als auch die Nebenbedingungen von Tourplanungsproblem nichtlinear sind, ist in dieser Arbeit dargestelltes Problem ein nichtlineares Optimierungsproblem. Ein nichtlineares Optimierungsproblem ist schwer zu lösen. Um solches Problem zu lösen, müssen einige Heuristiken und auch exakter Algorithmus verwendet werden.

2.2 Ameisenalgorithmus

2.2.1 Grundidee

In der Natur verstreuen die Lebensmittel rund um das Nest. Und die Ameisen können immer einen kürzesten Weg zwischen Futterstelle und Nest finden. Wenn eine Ameise keine Futterstelle gefunden hat, läuft sie in eine zufällige Richtung. Und gleichzeitig werden die Pheromone von der Ameise abgesondert. Die Pheromone sind nicht immer auf Boden. Nach einer gewissen Zeit verflüchtigen die Pheromone sich langsam. Wenn die Ameise einen kürzesten Weg gefunden hat, kann sie in einer kürzesten Zeit zwischen Futterstelle und Nest hin und her laufen.

Im Vergleich zu den anderen Wegen werden die Pheromone von dieser Ameise auf den kürzesten Weg mehr und mehr angesammelt. Die Ameisen suchen die Futterstelle nicht nur zufällig. Sie wählen meistens einen Weg, auf dem bereits mehr Pheromone als andere Wege hinterlassen wurden [BD00].

Für eine Ameisenkolonie ist diese Fähigkeit sehr wichtig. Obwohl eine einzelne Ameise allein auch zufällig den kürzesten Weg zwischen Futterstelle und Nest finden kann, können viele Ameisen schneller mittels den Pheromone den kürzesten Weg finden. Mehr Ameisen lassen die Pheromone auf dem kürzesten Weg schneller ansammeln. Das heißt, dass der kürzeste Weg immer stärker markiert wird. Die nachfolgenden Ameisen können mit einer höheren Wahrscheinlichkeit den stärker markierten Weg wählen. Dadurch kann im Zeitablauf der kürzere Weg immer stärker markiert werden, bis ein direkter Weg zwischen Futterstelle und Nest gefunden wird.

2.2.2 Algorithmen mit künstlichen Ameisen

Die Futtersuche der Ameisen hat dem italienische Mathematiker Marco Dorigo Inspiration gegeben. In 1991 wurde ein grundlegendes Modell der Ameisenalgorithmen von Coloni, Dorigo und Maniezzo entwickelt und erstmals präsentiert [CDM91]. Der erste Algorithmus war Ant System. Ant System wurde basierend auf dem Verhalten der Ameisen entwickelt. Und bei dem Algorithmus läuft die Kommunikation zwischen den Ameisen auf der Basis von Pheromonen ab. Danach erklärte Marco Dorigo die Kernidee der Ameisenalgorithmen in seiner Doktorarbeit in 1992 [D92]. Die Ameisenalgorithmen realisieren das Verhalten von Ameisen bei der Futtersuche.

Künstliche Ameisen bilden das Verhalten ihrer natürlichen Vorbilder nach. Aber natürlich gibt es auch Unterschiede.

Gleichheit:

1. Künstliche Ameisen und natürliche Ameisen können die Informationen auf den Wegen legen. Die natürlichen Ameisen können die chemische Sekrete

(Pheromone) absondern. Neben den Pheromonen existieren die digitalen Informationen für die künstlichen Ameisen. Die digitalen Informationen bezeichnen die Qualität der Lösungen und können von allen nachfolgenden Ameisen geändert werden. Mittels den digitalen Pheromone kann die gesuchte Lösung sukzessive analog einer Wegsuche festlegen.

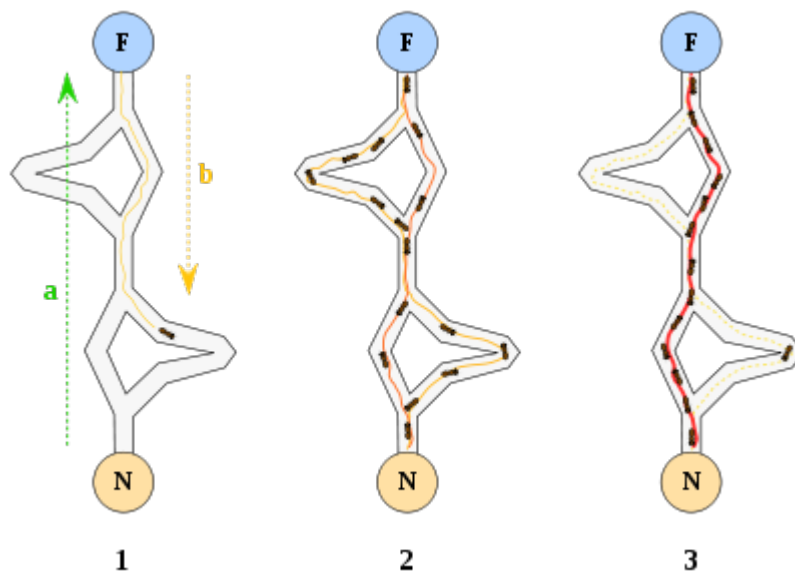


Abbildung 2.1: Wegsuche von Ameisen [BDG99]

2. Die digitalen Pheromone können sich auch wie chemische Pheromone langsam mit einer Verflüchtigungsrate ρ verflüchtigen. Wegen der Verflüchtigung können natürliche Ameisen und künstliche Ameisen die historische Informationen langsam vergessen, damit können die Ameisen sich nicht immer streng die mit Pheromonen markierten Wege entscheiden, aber auch neue Wege erforschen.

3. Der Übergang von einem Knoten zu anderem Knoten wird immer von natürlichen Ameisen und künstlichen Ameisen mit einer Wahrscheinlichkeit realisiert. Die Wahrscheinlichkeit kann durch die aktuellen Informationen bestimmt werden.

Unterschied:

Die künstlichen Ameisen haben auch andere Eigenschaften, die die natürlichen Ameisen nicht besitzen. Bei der Wegentscheidung ziehen die künstlichen Ameisen nicht nur die Pheromonmenge hinzu, sondern orientieren sich zusätzlich an einer heuristischen Information, eine relative wichtige lokale Informationen.

Ameisenalgorithmen beschränken sich darauf, dass es eine gewisse Anzahl m an Ameisen gibt, die eine Tour vom Nest zur Futterstelle machen. Eine Iteration endet dann, wenn alle Ameisen ihre Tour beendet haben. Mit der Variable t_{\max} wird festgelegt, wie viele Iterationen durchgeführt werden, ehe der Algorithmus beendet wird.

Übergangsregel:

Einige Regeln von den Übergang von Knoten i zu Knoten j müssen für jeden Ameise in jeder Iteration beachtet werden:

1. Eine Ameise darf keine zyklischen Wege gehen, das heißt, ein Knoten i darf nicht in einer gleichen Tour zweimal besucht werden. Um diese Situation zu vermeiden, existiert eine Liste J_i^k für jede Ameise k und jeden Knoten i . Diese Liste enthält alle Knoten, die die Ameise k in der momentanen Iteration von Knoten j aus noch nicht besucht hat. Nach jedem Übergang muss die entsprechende Liste aktualisiert werden. Natürlich am Anfang jeder Iteration müssen alle Listen mit den erreichbaren Knoten außer dem Startknoten, neu gefüllt werden.

2. Wenn eine Ameise sich in Knoten i befindet, gibt es eine lokale Information, welche darlegt, welcher Knoten j aus lokaler Sicht für einen Übergang am besten geeignet ist. Diese Information wird als η_{ij} bezeichnet. In dieser Arbeit wird die kürzeste Servicezeit gesucht. Weil die Bearbeitungszeit nach Annahme bestimmt

ist, wird nur die kleinste Summe von der Fahrzeit und der Wartezeit gesucht. Als die lokale Information η_{ij} wird die Kehrzahl der Summe von Fahrzeit von einem Terminal i nach nächstes Terminal j und Wartezeit in Terminal j definiert.

$$\eta_{ij} = \frac{1}{(t_{wait\ j} + t_{drive\ ij})}$$

3. Es gibt noch eine globale Information, die in Form der Pheromonmenge $\tau_{ij}(t)$ auf der Kante (i, j) in der Tour t darstellt.

Nach den Faktoren kann die Wahrscheinlichkeit p , für welchen Knoten die Ameise als nächstes wählt, wie folgend definiert werden:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{k \in allowed_k} [\tau_{ik}(t)]^\alpha \cdot [\eta_{ik}]^\beta} \quad (2.1)$$

Diese Formel kann nur dann verwendet werden, wenn j in der Liste J_i^k enthalten ist, das heißt, wenn der Knoten noch nicht besucht wurde, sonst ist die Wahrscheinlichkeit p_{ij}^k gleich null. Die Variablen α und β können frei gewählt werden und stellen die Einflussgröße von lokaler und globaler Information dar. Wenn α gleich null ist, existieren keine Information durch das Pheromon. Wenn hingegen β gleich null ist, entscheidet die Ameise sich nur den Weg, der am häufigsten benutzt wird.

Pheromon-Update-Regel:

Jede Ameise, die ihre Tour beendet hat, kann jene Kanten (i, j) , die sie benutzt hat, um eine gewisse Pheromonmenge $\Delta\tau_{ij}^k(t)$ verstärken. Es gibt auch verschiedene Modelle in [DMC91]. Ihr Unterschied ist nur $\Delta\tau_{ij}^k(t)$.

◆ Ameisen-Zyklus-Algorithmus

$$\Delta \tau_{ij}^k(t) = \frac{Q}{(t_{wait}^k + t_{drive}^k + t_{process}^k)} \quad (2.2)$$

$t_{wait}^k, t_{drive}^k, t_{process}^k$ bezeichnen die Wartezeit, Fahrzeit und Bearbeitungszeit der Tour t der Ameise k . Mit Q wird ein Parameter bezeichnet, dessen Wert bestimmt, wie viel Pheromon eine Ameise verteilen darf. Wenn eine Kante (i, j) nicht in Tour t der Ameise k enthalten ist, ist $\Delta \tau_{ij}^k(t) = 0$. Erst nachdem alle Ameisen gelaufen sind, werden antiproportional $\frac{Q}{(t_{wait}^k + t_{drive}^k + t_{process}^k)}$ die Pheromonmenge auf allen Kanten gleichzeitig verändert.

◆ Ameisen-Mengen-Algorithmus

$$\Delta \tau_{ij}^k(t) = \frac{Q}{(t_{wait\ ij}^k + t_{drive\ ij}^k)} \quad (2.3)$$

$t_{drive\ ij}^k$ bezeichnet die Fahrzeit zwischen Terminal i und j , die von Ameise k besucht wurden. Und $t_{wait\ ij}^k$ bedeutet die Wartezeit von Ameise k im Terminal j . Die Pheromonmenge auf einer Kante wird mit dem Faktor $\frac{Q}{(t_{wait\ ij}^k + t_{drive\ ij}^k)}$ verändert.

Praktisch sind Bearbeitungszeiten in verschiedenen Terminals auch nicht gleich. Das Tourplanungsproblem sucht die minimale Servicezeit. Aber die Bearbeitungszeit ist konstant. Das heißt, dass die optimale Lösung die minimale Summe von Wartezeit und Fahrzeit besitzt. Deswegen wird der Ameisen-Mengen-Algorithmus in dieser Arbeit benutzt.

Eine Verflüchtigungsrate des Pheromons von 0 bis 1 wird auch definiert. Einerseits bezeichnet Verflüchtigungsrate, wie lange die Pheromone wirken können. Und andererseits wird die Pheromonmenge anhand der Qualität der Lösung bestimmt. Damit können zwei Probleme gelöst werden:

- Die Verflüchtigungsrate beeinflusst das Erkunden neuer Wege. Je schneller die Pheromone verflüchtigen, desto mehr Wege werden erforscht.
- Während die Pheromonmenge anhand der Qualität der Lösung bestimmt werden, muss vermieden werden, dass sich eventuelle Wege durchsetzen, die nicht den Kürzesten darstellen.

Die Pheromon-Update-Regel lautet daher:

$$\tau_{ij}(t) \leftarrow (1 - \rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t) \quad (2.5)$$

$\Delta\tau_{ij}(t)$ ergibt sich seinerseits aus der Formel:

$$\Delta\tau_{ij}(t) = \sum_{k=1}^m \Delta\tau_{ij}^k(t) \quad (2.6)$$

Am Anfang werden die einzelnen Kanten mit einem kleinen, positiven Wert τ_0 initialisiert. Das bedeutet, dass im Zeitpunkt $t=0$ die Pheromonmenge homogen verteilt wird.

2.3 Genetischer Algorithmus

Genetischer Algorithmus ist eine andere sehr wichtige Optimierungsmethode. Der Algorithmus basiert auf die biologische Evolution und die Genetik. Durch Evolution verändert sich die vererbare Gene einer Population von Lebewesen von Generation zu Generation. Die angepasste Gene werden bei der Fortpflanzung an den Nachkommen weitergegeben. Dabei sind drei grundlegende Operatoren Selektion, Rekombination und Mutation sehr wichtige für die Evolution. Durch die drei Operatoren kann die Population sich über Tausende von Generationen an die Umwelt- und Lebensbedingungen anpassen. Die Evolution löst damit ein schwieriges Optimierungsproblem.

Die grundlegende Idee ist, dass zuerst verschiedene zufällige Lösungen von dem Algorithmus erzeugt werden. Eine ausreichende gute Lösung wird durch Rekombination und Mutation der anfänglichen Lösungsvorschläge gefunden. Folgende Prozedur zeigt die Hauptschritte der genetischen Algorithmus.

Algorithmus: Genetischer Algorithmus
Wähle problemspezifische Individuencodierung Initialisiere Individuum der Startpopulation zufällig repeat Bewerte Individuum mit Fitnessfunktion Selektiere Elternpaare nach Fitnesswerte Erzeuge Nachkommen durch Rekombination Mutiere die erzeugte Nachkommen until Abbruchbedingung

Individuum:

Der erste Schritt des Algorithmus besteht darin, dass die Individuen mit geeigneter Methode kodiert werden. Eine geeignete Kodierung der Individuen ist sehr wichtig, da sonst der Algorithmus nicht erfolgreich ist. Ein Individuum kann als Zeichenkette, Zahlenfolge oder Folge von Variablen binär kodiert werden. Das Chromosom, das repräsentativ für das "Lebewesen" ist, wird auch statt vom Individuum gesprochen. Ein Individuum beziehungsweise Chromosom entspricht einer Lösung aus dem Suchraum S des Problems. In dieser Arbeit sind die Touren beziehungsweise die Permutationen der Terminal-IDs die Lösungen. Deswegen werden die Chromosomen als Zahlenfolge wie zum Beispiel 7,12,5,3,6,2 kodiert.

Startpopulation:

Und dann wird die erste Generation der Population (Startpopulation) mit zufälligen Chromosomen aus dem Suchraum S erzeugt. Prinzipiell ist es auch möglich, die Startpopulation mit den Chromosomen mit größerem Fitnesswert zu belegen. Dabei wird jedoch das Risiko eingegangen, dass mit solchen Chromosomen wird ein lokales Optimum oder eine suboptimale Lösung ausliefern. Dieser Effekt heißt vorzeitige Konvergenz. Und die Größe der ersten Population muss so gewählt werden, dass der Suchraum gut abgedeckt wird. Die Größe der Population wird danach durch Methode der Parameterbestimmung bestimmt.

Fitnessfunktion:

Im nächsten Schritt wird mit Hilfe der Fitnessfunktion für jedes Chromosom c aus $P = \{c_1, c_2, c_3, \dots, c_n\}$ die Fitness $f(c)$ gerechnet. Die Fitnessfunktion entspricht der Zielfunktion des Optimierungsproblems. Fitnessfunktion entscheidet, dass mit welcher Wahrscheinlichkeit ein Chromosom an der nächsten Rekombination teilnimmt. Aus den Fitnesswerten können die Wahrscheinlichkeiten $p(c_i)$ der Chromosomen $c_i \in P$ nach der Formel

$$p(c_i) = \frac{f(c_i)}{\sum_{j=1}^n f(c_j)}$$

berechnet werden. Und die Summe der $p(c_i)$ ist natürlich gleich 1. Je größer $f(c_i)$ gegenüber der Fitness anderer Chromosomen ist, desto größer ist auch $p(c_i)$. Der Kehrwert der Summe von der Servicezeiten $t_{wait}^{c_i} + t_{drive}^{c_i} + t_{process}^{c_i}$ in allen Terminals gilt in dieser Arbeit als Fitnesswert des entsprechenden Chromosoms.

Selektion:

Als nächstes wird Selektion angewendet. Die Chromosomen werden von der Population als Eltern ausgewählt, um Rekombination oder Mutation durchzuführen. Nach Evolutionstheorie werden die Chromosomen mit gutem Fitnesswert ausgewählt. Es gibt auch viele unterschiedliche Methoden. In dieser Arbeit werden Roulette-Wheel-Selektion und Elitistselektion verwendet.

◆ Roulette-Wheel-Selektion

Am meisten wird Roulette-Wheel-Schema (Monte-Carlo) verwendet. Durch der Methode werden zufällig einige Chromosomen ausgewählt, um die nächste Generation der Population zu bilden. Dabei ist die oben gerechnete Wahrscheinlichkeit $p(c)$ des Chromosoms c sehr wichtig. Mit der Wahrscheinlichkeit wird ein Chromosom ausgewählt. Und das Roulette-Wheel-Schema funktioniert so:

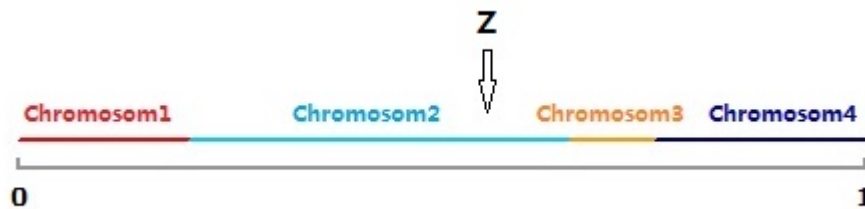


Abbildung 2.2: Roulette-Wheel-Selektion

Auf einem Intervall $[0,1]$ wird die Wahrscheinlichkeit $p(c)$ der Chromosomen nacheinander markiert. In Abbildung 2.6 ist diese Markierung für die Beispielwerte $p(c_1)=0.2$, $p(c_2)=0.45$, $p(c_3)=0.1$ und $p(c_4)=0.25$ dargestellt. Eine anschließend generierte, reelle Zufallszahl z zwischen 0 und 1 zeigt wie in Abbildung 2.2 auf ein Chromosom, das für die nächste Generation ausgewählt wird. Der Vorgang wird genau n -mal wiederholt. Jedes ausgewähltes Chromosom wird in die nächste Generation weitergegangen. Die Anzahl der Chromosomen von neuer Generation

ist dann genau gleich wie die Vorgängergeneration. Es wird nicht garantiert, dass das Chromosom mit größtem Fitnesswert in die nächste Generation weiter geht, aber es besteht eine sehr gute Chance, dass das Chromosom mit größtem Fitnesswert in die nächste Generation benutzt wird. Und weil durch dieses Verfahren die schwachen Chromosomen noch Chancen haben, in die nächste Generation weiter zu gehen, ist es geeignet, die genetische Vielfalt aufrechtzuerhalten.

◆ Elitistselektion

Wenn die neuen Chromosomen von Rekombination oder Mutation erzeugt werden, gibt es eine große Möglichkeit, dass das beste Chromosom verloren wird. Mit Hilfe von dieser Selektion wird zuerst das beste Chromosom in nächste Generation kopiert. Danach werden die anderen Chromosomen von klassischer Methode ausgewählt. Die Chromosomen mit besserem Fitnesswert werden einfach ausgewählt. Elitistselektion kann die Performance von GA sehr schnell inkrementieren, weil es verhindert, das beste Chromosom zu verlieren.

Rekombination (Crossover):

Nachdem die Selektion wird die Rekombination durchgeführt. Durch Rekombination werden die Chromosomen verändert, um neue und insbesondere Chromosomen mit noch größerem Fitnesswert zu erzeugen. Bei der Rekombination werden aus der Population P paarweise Chromosomen ausgewählt. Und die ausgewählten Chromosomen werden dann mit einer Wahrscheinlichkeit P_c rekombiniert. Es gibt viele Rekombinationsmethoden. Zwei Rekombinationen werden hier definiert: Order Crossover (OX) von Davis [D85] und One Point Crossover mit Rotation.

◆ OX

Es gibt drei Schritten, um die Nachkommen $N1$ und $N2$ aus zwei Eltern $E1$ und $E2$ zu erzeugen.

1. Ein Teil aus den Elternchromosomen wird ausgewählt. Und der Teil wird direkt in die Nachkommen kopiert:

$$E1=(1-2-3-4-5-6-7-8-9)$$

$$E2=(4-5-2-1-8-7-6-9-3)$$

$$N1=(x-x-x-1-8-7-6-x-x)$$

$$N2=(x-x-x-4-5-6-7-x-x)$$

2. Danach wird es vom zweiten Schneidpunkt angefangen und werden die Gene in derselben Ordnung kopiert:

$$\text{Sequenz von } E1=(8-9-1-2-3-4-5-6-7)$$

$$\text{Sequenz von } E2=(9-3-4-5-2-1-8-7-6)$$

3. Die restlichen Gene werden von den Eltern übernommen, wobei schon vorhandene Gene weggelassen werden. Die Ersetzung beginnt hinter den übernommenen Chromosomenstücken:

$$\text{Restliche Gene von } E1=(9-2-3-4-5)$$

$$\text{Restliche Gene von } E2=(9-3-2-1-8)$$

$$N1=(3-4-5-1-8-7-6-9-2)$$

$$N2=(2-1-8-4-5-6-7-9-3)$$

◆ One Point Crossover mit Rotation

Der One-Point-Crossover-Operator hat vier Schritten, um die Nachkommen zu erzeugen.

1. Ein beliebiger Rotationspunkt aus den beiden Elternchromosomen wird ausgewählt. Danach werden die Elternchromosomen von dieser Stelle rotiert:

$$E1=(1-2-3-4-5-6-7-8-9)$$

$$E2=(4-2-1-8-7-6-9-3-5)$$

$$E1'=(6-7-8-9-1-2-3-4-5)$$

$$E2'=(6-9-3-5-4-2-1-8-7)$$

2. Ein beliebiger Schneidpunkt aus den ersten Elternchromosomen wird ausgewählt. Danach wird der Elternchromosomen von dieser Stelle zwei Teile zerlegt:

$$E1'=(\overline{6-7-8-9}-1-2-3-4-5)$$

$$E2'=(\overline{6-9-3-5}-4-2-1-8-7)$$

3. Der erste Teil wird von den Eltern übernommen:

$$N1=(6-9-3-5-x-x-x-x-x)$$

$$N2=(6-7-8-9-x-x-x-x-x)$$

4. Eine beliebige Stelle wird ausgewählt. Ab dieser Stelle werden die restlichen Gene von den Eltern übernommen, wobei schon vorhandene Gene weggelassen werden:

$$\text{Restliche Gene von } E1' = (\overline{7-8-1-2-4})$$

$$\text{Restliche Gene von } E2' = (\overline{3-5-4-2-1})$$

$$N1=(6-9-3-5-1-2-4-7-8)$$

$$N2=(6-7-8-9-4-2-1-3-5)$$

Mutation:

Um die genetische Vielfalt aufrechtzuerhalten, ist die Mutation nach der Rekombination auch notwendig. Ein oder mehrere Gene eines Chromosoms werden durch Mutation zufällig stark verändert. Sie können zu schlechteren Ergebnissen führen, aber auch bessere Ergebnisse ausliefern. Für jedes Gen von jedem Chromosom aus P bestimmt eine gegebene Mutationswahrscheinlichkeit

P_m , ob dieses mutiert oder nicht. Mutationen treten mit geringer Wahrscheinlichkeit auf. Für die als Zahlenfolge kodierte Chromosomen gibt es viele Mutationsverfahren. Eine einfache Möglichkeit wird kurz vorgestellt:

◆ SWAP

Zwei zufällig ausgesuchte Gene wechseln ihren Platz.

(2-3-0-4-1) → (2-3-1-4-0)

Abbruchkriterium:

Die vier Schritte Bewertung der Individuen, Selektion, Rekombination und Mutation können beliebig oft wiederholt werden. Jede neue Generation erzeugt unterschiedliche Chromosomen. Und die neuen Lösungen können in Form von Chromosomen hervorbringen. Es gibt einige Beispiele für Abbruchkriterien.

1. Die Evolution der Chromosomen wird nach einer gegebenen Anzahl von Generationen durchgeführt.
2. Im Vergleich zur vorherigen Generation ist keine Verbesserung aufgetreten.

Leider ist es nicht garantiert, dass die neuen Lösungen immer besser als die Vorherigen sind. Der Algorithmus endet in dieser Arbeit deshalb nach einer bestimmten Anzahl, damit die beste Lösung nicht verpasst wird. Für jede Generation wird das Chromosom, welches den größten Fitnesswert der Population besitzt, als Ergebnis gespeichert. Als Ergebnis gibt das Chromosom zurück, welches den größten Fitnesswert aller Ergebnisse besitzt.

Wie Ameisenalgorithmus ist die Performance des genetischen Algorithmus auch abhängig von den Parametern. Für unterschiedliche Optimierungsprobleme werden unterschiedliche Parameterwerte benutzt. Wenn die Parameterwerte dem Optimierungsproblem nicht passen, liefert der Algorithmus vielleicht schlechtere Lösung oder läuft er langsamer. Für den genetischen Algorithmus müssen noch die

passenden Parameter bestimmt werden. Die Parameter sind die Größe der Startpopulation, die Rekombinations- und die Mutationswahrscheinlichkeit.

In dieser Arbeit werden zwei genetische Algorithmen getestet. Erster genetischer Algorithmus benutzt das Roulette-Wheel-Schema als Selektion, Order Crossover und die oben gezeigte Mutationsmethode als andere Operatoren. Der zweite genetische Algorithmus benutzt andere Selektion und Rekombinationsmethode. Elitisteselektion und One-Point-Crossover mit Rotation werden verwendet.

2.4 Erschöpfende Suche

Für viele Probleme in der Informatik sind keine effizienten Algorithmen bekannt. Der natürlichste und einfachste Ansatz zu einer algorithmischen Lösung eines Problems besteht darin, einfach alle potenziellen Lösungen durchzuprobieren, bis die richtige gefunden ist. Diese Methode nennt man auch "Brute-Force-Suche". Der größte Nachteil ist die extrem hohe Komplexität.

Das Optimierungsproblem kann auch durch die "Brute-Force-Suche" gelöst werden. Die Methode sucht alle Möglichkeiten der Touren und dann gibt ein Optimum zuletzt aus. Allerdings steigt der Aufwand an Rechenoperationen proportional zur Anzahl der zu probierenden, möglichen Lösungen. Wenn ein Schiff 20 Terminals besuchen möchte, gibt es $20!$ Permutationen möglicher Touren. Der Vergleich wird sehr lange Zeit dauern. Trotzdem ist dieser Algorithmus sehr wichtig, weil er immer ein exaktes globales Optimum ausliefern kann.

Brute-Force-Suche ist ein einfaches Verfahren. Alle Möglichkeiten werden durchprobiert. Das Verfahren sucht eine beliebige Permutation. Brute-Force werden alle möglichen Permutationen durchlaufen. Wenn die Servicezeit kleiner

als das Vorherige ist, wird diese neue Permutation als beste Permutation gespeichert. Schließlich wird die beste Permutation gefunden. Eine Herausforderung ist die Bestimmung aller Permutationen. Folgender Algorithmus zeigt die benutzte Methode. Zuerst wird die Prozedur mit einer beliebige Permutation $Perm[]$ und k gleich 0 angefangen. Folgend wird die erste Stelle der Permutation mit anderer Stelle vertauscht. Danach läuft die Prozedur immer iterativ bis k gleich n . Schließlich werden alle Permutationen ausgegeben.

<p>Eingabe: erste beliebige Permutation $Perm[]$; Anfangswert k; Länge der Permutation n;</p> <p>Ausgabe: alle Permutationen;</p>
<pre> Procedure Iterator (Perm[], k, n) { if k=n then Permutation ausgeben; else for i=k; i<n; i++ do Perm[k] mit Perm[i] vertauschen; Iterator (Perm[], k+1, n); end for end if } </pre>

2.5 NSGA2

Multiobjektiver Optimierungsalgorithmus ist auch sehr wichtig für das Optimierungsproblem. In dieser Arbeit werden Wartezeit und Fahrzeit optimiert. Deswegen wird auch ein multiobjektiver Optimierungsalgorithmus NSGA2

verwendet. Srinivas und Deb haben in 1994 erstmals die Non-Dominated Sorting Technik mit NSGA [SD94] vorgeschlagen. Danach haben sie den Algorithmus verbessert und NSGA2 in 2002 in [DPAM02] vorgestellt.

Algorithmus: Non-Dominated Sorting Genetic Algorithm (NSGA2) [DPAM02]

```

t = 0
Erzeuge Startpopulation Pt
Fast-Nondominated-Sort Pt
Erzeuge Qt aus Pt
repeat
    Rt = Pt ∪ Qt
    Fast-Nondominated-Sort Rt
    i:=1
    while (|Pt+1| + |Fi|) < pop do
        Pt+1 ∪ Fi
        i:=i+1
    end while
    Sort(Fi, ≥n)
    Pt+1 = Fi[0: pop - |Pt+1|]
    Erzeuge Qt+1 aus Pt+1
    t = t + 1
until Abbruchbedingung

```

Individuum und Startpopulation:

Wie oben gezeigten genetischen Algorithmus werden die Individuen und die Startpopulation von NSGA2 definiert.

Fitnesswert:

Bei NSGA2 wird die Population in einzelne Subpopulationen (in NSGA2 als Fronten genannt) zerlegt. Und die Fitnesswerte der Lösungen werden anhand der Front zugewiesen. Die Fitnesswerte der Individuen werden mittels eines Gewichtungsvektors für einzelne Zielkriterien wie zum Beispiel Wartezeit oder Fahrzeit von Schiffen auf einen skalaren Fitnesswert (als gewichtete Summe) umgerechnet. Die pareto-basierte Fitnesszuweisung hat einen Vorteil, dass bei der Bewertung von Fitnesswert eines Individuums alle Zielfunktionswerte immer gleichzeitig betrachtet werden und das Individuum immer mit allen anderen Individuen in der Population verglichen wird.

Genetischer Operator:

Wie genetischen Algorithmus werden die Nachkommen von NSGA2 durch Selektion, Rekombination und Mutation erzeugt. Und die Selektion wird hier Selektionsturnier (Tournament Selection) verwendet. Selektionsturnier hat Goldberg in 1989 in [GKD89] vorgeschlagen. Diese Methode gibt momentan schlechter bewerteten Individuen eine Chance, um die Vielfalt der Population aufrechtzuerhalten. Es gibt zwei Schritten:

1. Einige Individuen werden ausgewählt.
2. Die ausgewählte Individuen werden miteinander verglichen und die Sieger in die Population von Nachkommen aufgenommen.

Das Turnier wird solange fortgeführt, bis die Anzahl der Individuen für die Population von Nachkommen genug ist.

Fast-Nondominated Sort:

Aus Pseudocode ist deutlich, dass Fast-Nondominated-Sort eine wichtige Funktion für NSGA2 ist. Ziel dieser Methode ist es, die Lösung in Population P

hinsichtlich der Dominanz über andere Lösungen zu sortieren. Die Population P wird durch diese Methode in vielen Subpopulationen (F_1, F_2, \dots, F_n) zerlegt. Die Subpopulation wird als Front genannt. Das Verfahren geht dabei wie folgt vor:

1. Alle Lösungen der Population werden verglichen. Die Anzahl der Lösungen, die von einer Variable n_{y_i} , welche die Lösung y_i dominieren, gezählt, gespeichert. Und die Lösungen, welche durch die Lösung y_i dominiert werden, werden in einer Menge S_{y_i} auch gleichzeitig gespeichert.
2. Die Lösung mit $n_{y_i} = 0$ sind Pareto-Front und werden in der Subpopulation F_1 gespeichert. Und dann werden die Lösungen in F_1 temporär aus der Lösungsmenge entfernt.
3. Für andere Lösungen wird der Wert n_{y_i} angepasst.
4. Schritt 2 und 3 werden wiederholt, bis alle Lösungen temporär aus der Lösungsmenge entfernt werden.
5. Schließlich werden F_1, F_2, \dots, F_n gebildet.

F_1 ist optimale Pareto-Front. Und für alle andere Fronten gilt:

1. Die Lösungen der F_i ($i > 1$) werden von den Lösungen der F_{i-1} dominiert.
2. Die Lösungen der F_i ($i > 1$) dominieren die Lösungen der F_{i+1} .

Folgende Abbildung 2.3 zeigt ein Beispiel von Fast-Nondominated Sort.

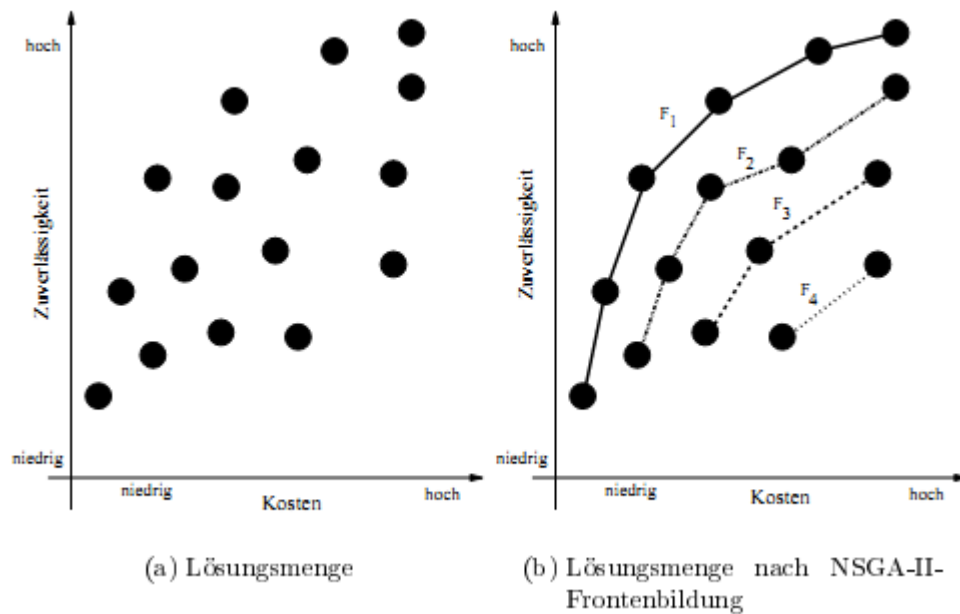


Abbildung 2.3: Beispiel Frontenbildung NSGA2 [R06]

Die Abbildung 2.3 (a) zeigt die Eingabedaten Population P . Und in der Abbildung 2.3 (b) sind deutlich, dass die Population P in vier Subpopulationen zerlegt wird. [R06]

Crowding-Distanz Berechnung:

Eine Methode wird angewendet, um die Vielfalt der Lösungen aufrechtzuhalten. Das ist Bestimmung der Lösungsdichte. Um die Lösungsdichte zu bestimmen, wird Crowding-Distanz von Deb eingeführt. Es gibt vier Schritten, um die Crowding-Distanz zu rechnen:

1. Die Distanzen für alle Lösungen $I[y_i]_{Distanz}$ werden gleich null initialisiert.
2. Die Distanz zu beiden benachbarten Lösungen wird für alle Lösungen y_i und für alle Zielfunktionen f_m gerechnet. Und nach jeder Zielfunktion werden die Lösungen aufsteigend sortiert.

3. Für die Lösungen mit kleinstem und größtem Zielfunktionswert werden die Distanz Unendlichkeit definiert.

4. Die Distanz für alle andere Lösung y_i bei $I = F_1$ in Abbildung 2.4 wird wie folgend gerechnet:

$$I[y_i]_{Distanz} = f_{Kosten}(I[y_{i+1}]) - f_{Kosten}(I[y_{i-1}]) + f_{Zuverlässigkeit}(I[y_{i+1}]) - f_{Zuverlässigkeit}(I[y_{i-1}])$$

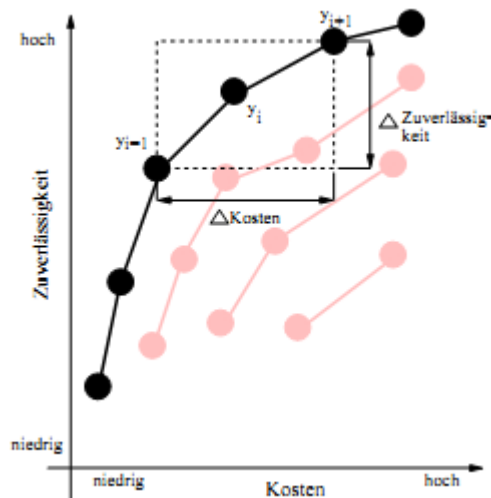


Abbildung 2.4: Beispiel Crowding Distanz-Berechnung mit NSGA2 [R06]

Nach der Bestimmung der Crowding-Distanz für alle Lösungen kann Crowding-Comparator angewendet werden, um die Vielfalt der Lösungen aufrechtzuhalten. Um zwei Lösungen zu vergleichen, müssen nicht nur die Crowding-Distanz, sondern auch der Rang benutzt werden. Es gilt:

$$y_i \geq_n y_j, \text{ falls } rang_{y_i} < rang_{y_j}$$

$$\text{oder } rang_{y_i} = rang_{y_j} \text{ und } I[y_i]_{Distanz} > I[y_j]_{Distanz}$$

Der Rang $rang_{y_n}$ einer Lösung y_n bedeutet die Nummer der Front F , in der die Lösung sich befindet. Das heißt, wenn die Lösung y_i einen höheren Rang als Lösung y_j besitzt, wird die Lösung y_i ausgewählt. Oder wenn beide Lösungen gleichen Rang besitzen, wird die Lösung mit größerer

Crowding-Distanz ausgewählt. Größere Crowding-Distanz bedeutet eine geringere Dichte, weil es in deren Umgebung weniger andere Lösungen gibt.

[R06]

Wie bei der singleobjektiven Optimierungsalgorithmen können die pareto-optimale Lösungen bei der multiobjektiven Optimierungsalgorithmen gefunden werden. In dieser Arbeit werden die Fahrzeit und die Wartezeit von NSGA2 minimiert.

3 Analyse der Tourplanung

In diesem Kapitel wird eine sehr umfassende Analyse der Ziele gezeigt. Beginnend wird im Abschnitt 3.1 der Hamburger Hafen kurz vorgestellt und die Tourplanung beschrieben. Im folgenden Abschnitt 3.2 werden die unterschiedlichen Optimierungskriterien definiert. Im Abschnitt 3.3 werden singleobjektive und multiobjektive Optimierungsalgorithmen verglichen. Danach wird Generierung von Testdaten im Unterkapitel 3.4 erklärt. Im letzten Unterkapitel 3.5 wird das in dieser Arbeit angewendetem Framework vorgestellt.

3.1 Problemstellung

Der Hamburger Hafen ist der größte Seehafen in Deutschland und auch den drittgrößte in Europa. Viele Schiffe besuchen jeden Tag verschiedene Terminals im Hamburger Hafen.

Im Hamburger Hafen gibt es heutzutage viele unterschiedliche Terminals für Seeschiffe: [WWW2]

- 4 große Container-Terminals
- 5 Multipurposeterminals
- 8 Massengutterminals

In 2007 sind insgesamt 12217 Seeschiffe im Hamburger Hafen angekommen, davon 7492 Containerschiffe, 933 konventionelle Stückgutfrachter, 1759 Schüttgutfrachter, 1495 Tankschiffe und so weiter. In 2009 ist alle statistische Zahl zurückgegangen. Aber in 2010 sind Anzahl der kommenden Seeschiffe und das gesamte Gewicht des Umschlags wieder gestiegen.

Jahr	2007	2008	2009	2010
Gesamtumschlag in Mio. [t]	140,4	140,4	110,4	121,1

Tabelle 3.1: Umschlagstatistik [WWW1]

Deswegen ist die Planung sehr wichtig, sonst gibt es auch Stau und können die Schiffe nicht pünktlich be- und entladen und dann abfahren. Manuell kann die Planung bestimmt werden, aber es dauert längere Zeit und gibt wirklich sehr viel Arbeit. Mit dem Computer kann die Planung leicht und schnell lösen.

Schiffe:

Das Ziel der Planung ist eine Tour zu bestimmen, entlang der ein Schiff schnell wie möglich alle zu bedienende Terminals nur einmal besuchen kann. Hier ist die Zeit sehr wichtig. Für Schiffe gibt es Be- und Entladezeit, Fahrzeit und vielleicht auch Wartezeit. Für alle Schiffe sind Be- und Entladezeit unterschiedlich. Die Bearbeitungszeit ist schwer zu berechnen. In dieser Arbeit wird die Bearbeitungszeit einfach wie folgend angenommen. Die Bearbeitungszeit in jedem Terminal ist 100 Minuten.

Terminals:

Obwohl jedes Terminal viele Liegeplätze hat, sind die Terminals nicht immer frei. Wenn das Terminal schon besetzt ist, muss das Schiff immer warten bis vorheriges Schiff fertig ist. Wenn das Terminal jetzt frei ist, kann das Schiff sofort bearbeitet werden. Aber vielleicht ist das Terminal schon reserviert. Wenn das Schiff A, das das Terminal reserviert hat, angekommen ist, muss das Schiff B, das jetzt bearbeitet wird, warten, bis Schiff A fertig ist.

Wegen des Wasserwiderstand und Fließrichtung vom Wasser kann die Fahrzeit von zwei gleich langen Distanzen aber ungleich sein. Deswegen wird die Distanzmatrix der Terminals in dieser Arbeit asymmetrisch definiert.

Aktualisierung der Terminal-Informationen:

Nachdem die beste Tour gefunden wird, muss die Terminal-Informationen aktualisiert werden. Der Zeitraum, in dem das Schiff im Terminal gearbeitet wird, wird markiert. In dem markierten Zeitraum können andere Schiffe nicht mehr in dem Terminal bearbeitet werden. Es gibt noch einen wichtigen Schritt, dass alle markierten Zeiträume nach Zeit sortieren müssen.

3.2 Optimierungskriterien

Um eine beste Tour zu finden werden auch geeignete Optimierungskriterien verwendet. Im Folgenden werden einige Kriterien erklärt, damit die Schiffe, die verschiedene Probleme lösen möchte, wie zum Beispiel Einsparung von Kraftstoffe oder Expressdienst, verschiedene Kriterien auswählen können.

3.2.1 Schnellstes Kriterium

Für die Schiffes, die schnell wie möglich be- und entladen und vom Hafen abfahren möchten, wie zum Beispiel die Expressdienste, wird schnellstes Kriterium benutzt. Für die Tour werden Bearbeitungszeit, Fahrzeit und auch Wartezeit berechnet. Die Tour mit kürzester Servicezeit wird ausgewählt. Bearbeitungszeit wird wie oben berechnet. Ein Schiff fährt immer mit im Hafen

erlaubter maximaler Geschwindigkeit nach nächstem Terminal. In dieser Arbeit ist diese maximale Geschwindigkeit mit 12 Seemeilen pro Stunde angenommen.

3.2.2 Ökonomisches Kriterium

Aber normalerweise wird ökonomisches Kriterium für die Schiffe benutzt. Verbrauch von Kraftstoffe kostet auch viel Geld für Schiffe. Zum Beispiel ein Auto, das bei Geschwindigkeit 90 km/h 9 Liter und bei 160 km/h 15 Liter verbraucht. Für die Schiffe ist auch so, wenn ein Schiff zu schneller oder zu langsamer fährt, werden mehr Kraftstoffe verbraucht. Wie beim Auto hat ein Schiff auch die Höchstgeschwindigkeit und die empfohlene Reisegeschwindigkeit. Wenn die Schiffe mit der Höchstgeschwindigkeit fahren, verbrauchen sie sehr viele Menge von Kraftstoffe wegen überproportional steigenden Wasserwiderstandes und zwar wird die Umweltverschmutzung stark zugenommen. Deswegen wird eine auf Verbrauch von Kraftstoffe optimierte Reisegeschwindigkeit definiert. Das ist so genannt ECO-Speed. Bei ökonomischem Kriterium fahren alle Schiffe mit ECO-Speed. In dieser Arbeit wird es angenommen, dass ECO-Speed 8 Seemeilen pro Stunde ist. Das ist ungefähr 14.82 km/h. Und Die Tour wird auch wie oben schnellstes Kriterium ausgewählt.

3.3 Singleobjektive versus multiobjektive Optimierung

Das Optimierungsproblem kann mit singleobjektiver Optimierung gelöst werden. Aber in der realen Welt braucht ein Optimierungsproblem vielleicht nicht nur ein Objekt. In dieser Arbeit wird eine Tour mit der kürzesten Servicezeit bestimmt und müssen gleichzeitig die Kraftstoffe wenig wie möglich verbraucht werden. Deswegen kann multiobjektive Optimierung verwendet werden.

Für singleobjektive Optimierung können schnellstes Kriterium oder ökonomisches Kriterium verwendet werden. Und die Servicezeit wird optimiert. Eine Tour mit minimaler Servicezeit wird geliefert. Für die Schiffe kostet jede Wartezeit viel Geld, deswegen ist die Tour mit minimaler Servicezeit häufig eine gute Tour.

Aber es gibt auch Ausnahme, deshalb wird multiobjektive Optimierung benutzt. Multiobjektive Optimierung kann Tourplanungsproblem mit mehreren Zielen lösen. Minimale Fahrzeit bedeutet einen minimalen Verbrauch von Kraftstoffe und minimale Wartezeit bedeutet, dass das Schiff immer arbeitet. Mit Hilfe von dieser multiobjektiven Optimierung können eine Tour mit sowohl kurzer Fahrzeit als auch kurzer Wartezeit gefunden. Aber manchmal widersprechen die kurze Fahrzeit und kurze Wartezeit sich. Deswegen muss ein Trade-Off gelöst werden. Wenn die Wartezeit viel länger als Fahrzeit ist, ist die Tour schlecht.

3.4 Generierung von Testdaten

Um die Evaluation zu ermöglichen, müssen zuerst die Testdaten generiert werden. Die Testdaten in dieser Arbeit sind die Daten von Schiffen und auch die Informationen von Terminals. Die in dieser Arbeit benötigten Testdaten werden in diesem Kapitel vorgestellt.

3.4.1 Testdaten von Schiffen

Weil die Geschwindigkeiten von Schiffe und die Bearbeitungszeit schon wie oben angenommen werden, sind die Schiff-ID, die Ankunftszeit und die ID-Nummer der zu bedienenden Terminals die notwendig Eingangsdaten.

Für die Simulation werden einige Daten für unterschiedliche Schiffe, die unterschiedliche Anzahl der zu bedienenden Terminals haben, erzeugt. Die unterschiedlichen Schiffe haben natürlich auch die unterschiedliche Ankunftszeit. In dieser Arbeit werden insgesamt 13 Schiffe initialisiert. Die Anzahl der zu bedienenden Terminals von jedem Schiff ist nicht gleich. In Hamburger Hafen gibt es insgesamt 17 Terminals. Deswegen ist die Anzahl der zu bedienenden Terminals von 3 bis 15 definiert.

3.4.2 Testdaten von Terminals

Für Terminals müssen Anzahl der Terminals und Anzahl der Liegeplätze von Terminals definiert werden. Natürlich sind die Distanzen zwischen Terminals auch eine wichtige bekannte Bedingung.

Für unterschiedliche Situationen wird ein paar Testdatengruppen von Terminals generiert. Die Testdaten von Terminals haben 3 Gruppen.

1. Geringer Verkehr: Für jeden Liegeplatz gibt es nur eine Zeitraum besetzt.
2. Normaler Verkehr: Jeder Liegeplatz wurde schon von zwei bis drei Schiffe besetzt.
3. Hoher Verkehr: Jeder Liegeplatz wurde schon von vier bis fünf Schiffe reserviert.

Die besetzte Zeiträume von jedem Liegeplatz in jedem Terminals werden zufällig erzeugt.

3.5 OPT4J Framework

In dieser Arbeit wird ein Framework OPT4J benutzt. OPT4J ist ein Framework für die Anwendung der meta-heuristische Optimierungsalgorithmen, um beliebige

Optimierungsprobleme in Java zu programmieren. Das Framework ist sehr erweiterbar, so dass die Probleme (reale Welt und auch Benchmark) sowie Optimierer bequem implementiert werden können. OPT4J bietet noch eine grafische Benutzeroberfläche für die Konfiguration sowie eine Visualisierung der Optimierungsaufgaben. Weil das Framework schon ein multiobjektiver Evolutionärer Algorithmus (inklusive SPEA2 und NSGA2) und auch singleobjektiver Evolutionärer Algorithmus wie zum Beispiel genetischer Algorithmus mit unterschiedliche Selektionen, Rekombinations- und Mutationsoperatoren einschließt, passt das Framework sehr gut zu dieser Arbeit.

Opt4J besteht aus vielen Interfaces. Diese Interfaces wurden schon durch vordefinierte Classes implementiert. Allerdings kann der Benutzer selber auch maßgeschneiderte Implementierungen von diesen Interfaces implementieren. Die Abbildung zeigt die schematische Darstellung von OPT4J Interfaces und Classes.

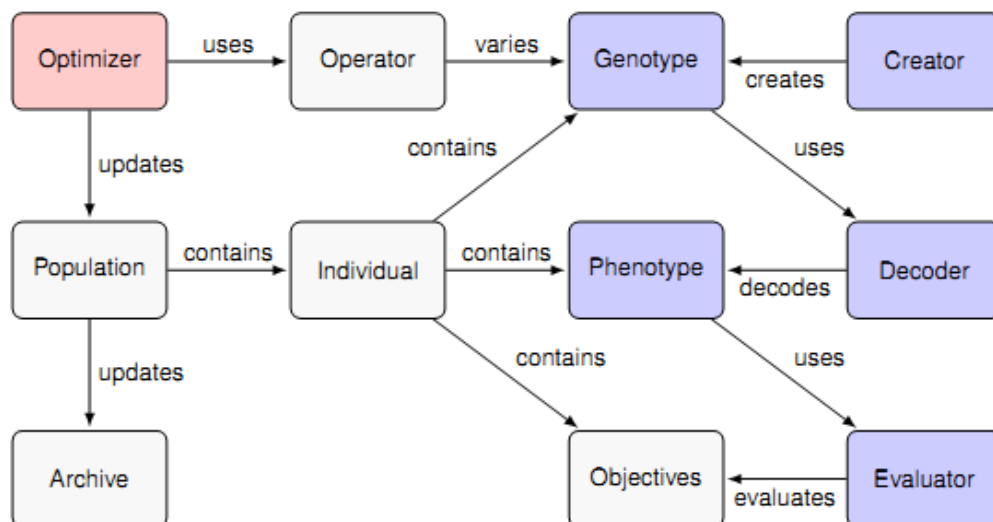


Abbildung 3.1: schematische Darstellung von OPT4J Interfaces und Classes

[LGRT11]

In dieser Arbeit müssen nur die in Abbildung 3.1 gezeigten Interfaces Creator, Decoder, Evaluator, Genotype und Phenotype von dem singleobjektiven und auch multiobjektiven Tourplanungsproblem definiert werden. Dann können diese Interfaces mit anderen vordefinierten Interfaces zusammenarbeiten, um das Problem zu lösen.

3.6 Implementierung

Um die Tour mit minimaler Servicezeit zu finden, werden nicht nur Optimierungsalgorithmen sondern auch eine Methode zur Berechnung der Servicezeit verwendet. Erschöpfende Suche, genetischer Algorithmus und NSGA2 suchen zuerst eine mögliche Tour. Danach wird die gesamte Servicezeit von dieser Tour berechnet. Durch Vergleich der Servicezeiten wird die optimale Tour gefunden. Ameisenalgorithmus ist aber etwas anderes. Ameisenalgorithmus sucht das nächste optimale Terminal eins nach dem anderen. Schließlich wird eine ganze Tour gefunden. Aber die Methoden zur Berechnung der Servicezeit sind gleich.

In dieser Arbeit sind Bearbeitungszeit und die Geschwindigkeit von den Schiffen bereits definiert. Die Distanzen zwischen allen Terminals sind auch bekannt. Deswegen kann die Fahrzeit einfach berechnet werden. Der Schwerpunkt ist nur Berechnung von Wartezeit. In [DSS09] wird ein Warteprofil gezeigt, um die Wartezeit zu berechnen. Ein Warteprofil für einzelnen Liegeplatz wird in dieser Arbeit benutzt.

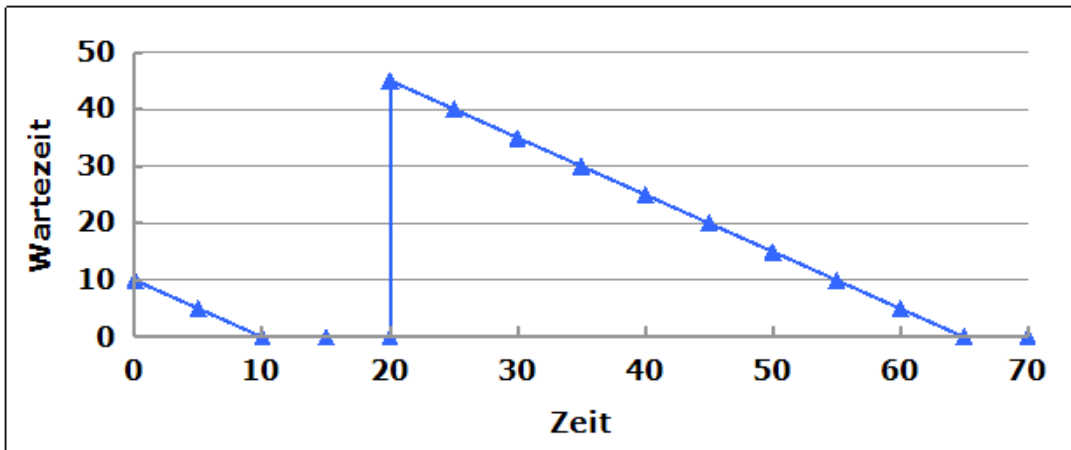


Abbildung 3.2: Wartezeit-Profil für einzelnen Liegeplatz [DSS08]

In der Abbildung 3.2 wird ein Beispiel von einem Wartezeit-Profil für einzelnen Liegeplatz gegeben. In diesem Liegeplatz ist die Bearbeitungszeit für das Schiff 15 Minuten. Der Liegeplatz ist in dem Zeitraum von $t=30$ bis $t=50$ geschlossen. Und der Liegeplatz wird schon von zwei Schiffen in dem Zeitraum von $t=0$ bis $t=10$ und von $t=55$ bis $t=65$ reserviert. Das angekommene Schiff muss immer warten, bis die anderen Schiffe fertig sind. Deswegen verlängert die Wartezeit in dem Zeitpunkt $t=20$.

<p>Eingabe:</p> <p>Ankunftszeit AZ; gewählte Tour $Tour[]$;</p> <p>Ausgabe:</p> <p>minimale gesamte Wartezeit</p>
<p>alle $Wartezeit[i][j]$ gleich 0 initialisieren;</p> <p>for $i=0$; $i<Anzahl$ der zu bedienenden Terminals; $i++$ do</p> <p> for $j=0$; $j<Anzahl$ der Liegeplätze im Terminal $Tour[i]$; $j++$ do</p> <p> $Wartezeit[i][j]$ an Liegeplatz j im Terminal $Tour[i]$ mit Hilfe von Information des Terminal $Tour[i]$ und Ankunftszeit AZ berechnen;</p> <p> end for</p> <p> minimale $Wartezeit[Tour[i]]$ im Terminal $Tour[i]$ wählen;</p> <p> Ankunftszeit AZ für nächstes Terminal aktualisieren;</p> <p>end for</p> <p>minimale gesamte Wartezeit berechnen;</p>

Folgende Prozedur zeigt, wie wird gesamte minimale Wartezeit für eine gewählte Tour berechnet. In jedem Liegeplatz wird Ankunftszeit mit besetzten Zeiträumen verglichen, um die Wartezeit zu berechnen. Wartezeiten an jedem Liegeplatz in einem bestimmten Terminal werden berechnet und verglichen. Die minimale Wartezeit in diesem Terminal wird als Wartezeit gewählt. Durch Addition von allen minimalen Wartezeiten kann die minimale gesamte Wartezeit bekommen werden.

4 Parameterbestimmung der Optimierungsalgorithmen

Die Performance von Ameisenalgorithmus und genetischem Algorithmus sind für die Parameter sehr sensitiv. Wenn die Parameterwerte nicht geeignet sind, können keine guten Ergebnisse bekommen. Für die unterschiedlichen Problem-Typen müssen unterschiedliche Parameterwerte bestimmt werden. Und für den gleichen Typ von Problem mit unterschiedlichen Größen müssen auch unterschiedliche Parameterwerte gewählt werden. In diesem Kapitel werden zuerst alle Parameter von den oben gezeigte Heuristiken erklärt. Danach wird die Methode zur Parameterbestimmung definiert. Schließlich werden die Heuristiken simuliert und die Parameterwerte bestimmt.

4.1 Parameter von Ameisenalgorithmus

Bei Ameisenalgorithmus gibt es insgesamt 5 Parametern: die Verflüchtigung ρ , die Anzahl der Ameisen m , die Menge des Pheromon, die eine Ameise verteilen darf, Q und die Exponenten α und β . Die verschiedenen Parameterwerte können den Algorithmus vielfältig steuern. Mit den „richtigen“ Parametern können die guten Lösungen effizient erzeugt werden. Die Beziehungen zwischen den Parametern werden mit Hilfe von Gleichungen (2.1), (2.3) und (2.5) definiert.

In Gleichung (2.1) gibt es nur 2 Parametern α und β . Und dabei sind α und β Parameter, welche die Gewichtung der Pheromonwerten der Kante und der Sichtweite der Ameise angibt. $\alpha > \beta$ bedeutet, dass die durch Iterationen erzeugten Lösungen eine wichtigere Bedeutung hat. Das heißt, dass die Ameisen häufig die vorher gewählten Touren nochmal wählen. Aber der Ameisenalgorithmus ist

lokales Suchverfahren (eine zufällige Lösung wird iterativ optimiert), das durch sogenannte Metaheuristik unterstützt werden. Deswegen, wenn α groß ist, erzeugt der Algorithmus nicht so viele zufällige Lösungen. Die Lösung ist oft nur ein lokales Optimum. $\alpha < \beta$ bedeutet hingegen, dass die lokale Information über die Länge der Kante höheren Einfluss erhält. Obwohl der Algorithmus schnell konvergieren kann, ist die Lösung häufig auch nur ein lokales Optimum. [PFM06]

In Gleichung (2.3) gibt es noch einen Parameter Q . Q bedeutet die Menge Pheromon, die eine Ameise verteilen darf. Je größer Q ist, desto schneller die Pheromone inkrementieren. Wenn Q zu klein ist, konvergiert der Algorithmus langsam. Aber wenn Q zu groß ist, bekommt der Algorithmus oft nur ein lokales Optimum wegen zu konzentrierten Pheromonen.

In Gleichung (2.5) ist deutlich, dass ρ zeigt, wie schnell das Pheromon verflüchtigt. Die Verflüchtigung ρ beeinflusst die Konvergenzgeschwindigkeit und die globale Erkundungsfähigkeit. Wenn ρ sehr groß ist, werden die Pheromone der noch nie markierten Lösungen wegen der Verflüchtigungsrate nahe 0 reduziert und hat die markierte Lösung eine große Möglichkeit, nochmal markiert zu werden. Deswegen verringert die Erkundungsfähigkeit des Algorithmus und wird immer ein lokales Optimum liefert. Und wenn die Verflüchtigungsrate zu klein ist, kann die globale Erkundungsfähigkeit erhöhen, aber reduziert die Konvergenzgeschwindigkeit.

Beim Ameisenalgorithmus wählen die Ameisen neue Wege zufällig nach der vorhandenen Pheromonmenge. Und die Pheromone werden von Ameisen abgesondert. Wie oben gesagt, dass auf dem kürzere Weg mehr Pheromone ablagern. Deswegen liefern kooperative Ameisen das beste Ergebnis. Es ist deutlich, dass die Größe m der Ameisenpopulation die größeren Auswirkungen auf das Ergebnis hat. Je mehr Ameisen es gibt, desto mehr Touren werden erkundet. Aber wenn die Ameisenpopulation zu groß ist, sind die Änderungen der Pheromonmenge von vielen markierten Lösungen fast gleich. Wenn die Ameisenpopulation klein und das Optimierungsproblem groß ist, werden die

Pheromone von den nicht markierten Lösungen nahe 0 verflüchtigen. Obwohl die Lösungen schnell konvergieren können, reduziert die Stabilität des Algorithmus. Und der Algorithmus kann zu früh zur Stagnation kommen. Das heißt, dass die Wahl des Parameterwerts von m ähnlich wie verflüchtigungsrate ist. Der Konflikt zwischen Konvergenzgeschwindigkeit und globaler Erkundungsfähigkeit ist auch ein Problem. Und zwar die Anzahl von Ameisen hat auch Auswirkung auf die Wahl des Parameterwerts von Q .

4.2 Parameter von genetischem Algorithmus

Beim genetischen Algorithmus sind die Parameter nicht so kompliziert wie Ameisenalgorithmus. Es gibt nur 3 Parametern: Größe der Population, Rekombinationswahrscheinlichkeit und Mutationswahrscheinlichkeit.

Die Größe der Population ist sehr wichtig für genetischen Algorithmus. Je mehr Chromosomen es gibt, desto mehr verschiedene Allelen in der Population vorkommen. Damit kann der Algorithmus nicht wegen zu wenig genetischer Vielfalt zu früh in einem lokalen Optimum konvergieren. Aber wenn eine Population mehr Chromosomen besitzt, dauern die Rekombination und Mutation natürlich auch länger Zeit.

Rekombination ist einen wichtigen Operator für genetischen Algorithmus. Der Operator erzeugt verschiedene Chromosomen, um die Lösung zu optimieren. Wegen der steigenden Rekombinationswahrscheinlichkeit haben die Chromosomen mehre Chancen mit anderen Chromosomen zu rekombinieren. Damit werden mehr verschiedene Chromosomen erzeugt und verglichen und ist es klar, dass mehr Zeit gebraucht wird.

Durch Mutation können die Chromosomen stark geändert werden, damit der Algorithmus nicht immer nach lokalem Optimum sucht. Wenn die Mutationswahrscheinlichkeit zu hoch ist, werden die Chromosomen, die bereits nahe einem Optimum gefunden werden, wieder wegen Mutation weggedrängt. Deswegen soll die Mutationswahrscheinlichkeit immer klein sein, um die Chromosomen, die bereits nahe einem Optimum gefunden werden, nicht wegen Mutation wegzubringen.

4.3 Parameter von NSGA2

NSGA2 basiert auf genetischen Algorithmus. Außer Rekombinations- und Mutationswahrscheinlichkeit muss noch ein Parameter für Turnier bestimmt werden. Für jedes Turnier werden n Chromosomen ausgewählt und verglichen. Wenn n zu klein ist, werden vielleicht nicht die besten Chromosomen ausgewählt. Aber wenn n zu groß ist, werden normalerweise die Chromosomen mit besserem Fitnesswert ausgewählt. Das ist nicht gut für die Aufrechterhaltung der Vielfalt von der Population. Ein lokales Optimum wird häufig ausgegeben.

4.4 Methode der Parameterbestimmung

Für meiste Optimierungsalgorithmen gibt es keine gute Methode, um die Parameter zu bestimmen. Parameterbestimmung für die Optimierungsalgorithmen kann auch in einem Optimierungsproblem umgewandelt werden. Und dieses Problem kann mit Optimierungsalgorithmen gelöst werden. Aber das Problem ist, dass die schlechten Parameter wieder eine schlechte Auswirkung auf die Ergebnisse von Optimierungsalgorithmen haben.

Eine andere Möglichkeit zur Parameterbestimmung ist mit mehreren verschiedenen Parameterwerten der Algorithmus durchzuführen. Danach werden die Ergebnisse verglichen und der beste Parameterwert bestimmt. Aber es dauert lange Zeit.

Aus der Erfahrung ist der Wertebereich der guten Parameterwerte durch ein Intervall begrenzt. Deswegen müssen nicht alle Parameterwerte getestet werden. In dieser Arbeit werden die Parameter mit einigen guten Werten getestet und verglichen. Zuerst wird ein Parameter mit verschiedenem Wert testen und werden gleichzeitig alle anderen Parameterwerte festgehalten. Durch Vergleich kann ein bester Parameterwert bestimmt werden. Danach können andere Parametern genau gleich bestimmt werden. Wenn die Beziehung zwischen einige Parametern sehr stark ist, müssen die Parameter mit ein paar Werten gleichzeitig getestet werden. Durch diese Methode können die besten Parameterwerte gefunden werden. Und zwar die Probleme mit unterschiedlichen Größen (z.B. 10, 15 Terminals) werden einzeln getestet. Nach dem Vergleich können wir die besten Parameterwerte für die Probleme mit unterschiedlichen Größen bekommen. Mittels der Beziehung zwischen besten Parameterwerten und Größen der Probleme kann eine Kurve oder Linie skizziert werden. Mit Hilfe von der Kurve oder Linie kann eine mathematische Funktion bestimmt werden. Dann können die besten Parameterwerte für mehrere Probleme mit unterschiedlichen Größen bestimmt werden.

4.5 Simulation

Bevor der Parameterbestimmung müssen zuerst die erschöpfende Suche testen. Wenn das Problem durch die erschöpfende Suche schnell gelöst werden kann, müssen die Heuristiken nicht verwendet werden. Deswegen müssen die erschöpfende Suche mit unterschiedlichen Testdaten simuliert werden, danach

können die Ergebnisse der Probleme, die durch erschöpfende Suche nicht schnell gelöst werden, mit Hilfe von Heuristiken berechnet werden.

4.5.1 Erschöpfende Suche

Zuerst wird erschöpfende Suche simuliert. Im Vergleich zu den Heuristiken ist erschöpfende Suche einfach und stabil, weil es keinen weiteren Parameter gibt und immer eine beste Lösung erzeugt werden kann. Aber erschöpfende Suche läuft sehr langsam, wenn das Problem groß ist. Aber wenn das Problem nicht so groß ist, das heißt, dass das Schiff nicht so viele Terminals besuchen möchte, kann erschöpfende Suche schnell das Problem lösen. Auf diesem Fall kann nur erschöpfende Suche verwendet werden und müssen die schnellere und komplexere Heuristiken nicht benutzt werden.

Deswegen muss zuerst der Algorithmus mit oben gezeigten Eingangsdaten getestet werden. Die Ergebnisse werden im Tabelle 5.1 gezeigt.

Anzahl der zu bedienenden Terminals	geringer Verkehr	normaler Verkehr	hoher Verkehr
3	0.1s	0.1s	0.1s
4	0.46s	0.5s	0.5s
5	1.8s	1.7s	1.7s
6	12.7s	13.4s	13.5s
7	125.7s	126.8s	127.3s
8	833.7s	832.9s	841.3s

Tabelle 4.1: Laufzeit der erschöpfenden Suche

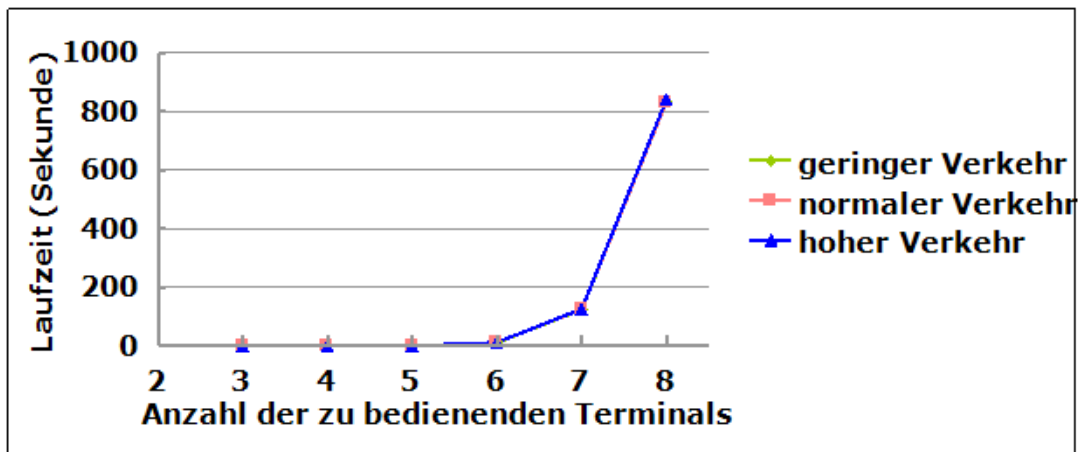


Abbildung 5.1: Laufzeit der erschöpfenden Suche

In der Abbildung 5.1 ist deutlich, dass die Laufzeit mit höherer Anzahl der zu bedienenden Terminals sehr schnell wächst. Und die Dichte der ankommenden Schiffe hat fast keine Auswirkung auf die Laufzeit des Algorithmus. Wenn ein Schiff weniger als 8 Terminals besuchen möchte, kann die Brute-Force-Suche in weniger als einer Minute die optimale Lösung finden. Aber wenn ein Schiff mehr als 7 zu bedienenden Terminals hat, dauert die Berechnung lange Zeit. Deswegen werden die Heuristiken auch verwendet.

4.5.2 Ameisenalgorithmus

Ameisenalgorithmus ist ein solcher Algorithmus. Der Algorithmus kann sehr schnell eine optimale Lösung für ein sehr großes Problem finden. Aber es gibt viele Parameter. Wenn die Parameterwerte nicht so gut bestimmt werden, kann der Algorithmus nicht eine gute optimale Lösung finden. Deswegen müssen die Parameterwerte für die Probleme auch optimiert werden.

In diesem Test werden die Testdaten, mit dem erschöpfende Suche nicht in kurzer Zeit schaffen kann, benutzt. Die Anzahl der Iterationen für den Algorithmus wird 50 definiert, damit der Algorithmus nicht zu lange Zeit läuft. Für jede Testdaten

wird der Algorithmus 10 Mal wiederholt. Das beste Ergebnis bedeutet die kürzeste Servicezeit aus 10 Teste. Und der zugehörige Parameterwert ist den optimale Parameterwert. Die längste Servicezeit ist das schlechteste Ergebnis. Und die durchschnittlichen Ergebnisse werden verglichen.

Anzahl der Ameisen

Bei Ameisenalgorithmus kann die Anzahl der Touren T mit Anzahl der Ameisen m und Anzahl der Iterationen S bestimmen. Eine Ameise kann eine Tour in einer Iteration finden. Das heißt, dass m Ameisen in S Iterationen $m \times S$ Touren bestimmen können. Das bedeutet auch, je mehr Ameisen gibt, desto mehr Touren werden erkundet. Es ist deutlich, dass die Größe m der Ameisenpopulation die größeren Auswirkungen auf das Ergebnis und die Laufzeit hat und muss zuerst bestimmt werden

In der Test werden andere Parametern wie folgend angenommen:

$$\rho = 0.6, \alpha = 0.9, \beta = 1.0, Q = 1.0$$

Folgende Tabelle zeigt die Ergebnisse der Simulation.

m		3	4	5	6	7
8 Terminals	ST _{best}	979	975	975	975	975
	ST _{worst}	983	983	983	983	983
	ST _{ave.}	981.4	977.8	977.8	978.8	980
	T	15.7s	21.2s	26.5s	31.1s	36.6s
10 Terminals	ST _{best}	1209	1209	1197	1209	1205
	ST _{worst}	1237	1225	1225	1221	1217
	ST _{ave.}	1221	1217	1208.2	1214.6	1212.2
	T	28.3s	37.7s	47.3s	57.1s	65.7s
12 Terminals	ST _{best}	1463	1455	1455	1447	1463
	ST _{worst}	1483	1479	1479	1471	1471
	ST _{ave.}	1471.8	1467.8	1469.4	1464.6	1466.2
	T	36.4s	49.3s	60.9s	73.3s	86.1s
15 Terminals	ST _{best}	1838	1838	1822	1830	1818
	ST _{worst}	1866	1862	1854	1850	1846
	ST _{ave.}	1850	1849.2	1838.8	1840.4	1836.4
	T	59.4s	80.2s	99.5s	118.4s	133.2s

Tabelle 4.2: Ergebnisse und Laufzeit des Ameisenalgorithmus mit unterschiedlicher Anzahl der Ameisen

Tabelle zeigt die Ergebnisse, wenn die Anzahl der Ameisen zwischen 3 und 7 liegt. Wie erwartet, wenn die Anzahl der Ameisen zu klein oder zu groß ist, kann nicht ein gutes Ergebnis bekommen werden. In der Tabelle ist deutlich, wenn die Anzahl der Ameisen gleich halbe Anzahl der Terminals ist, kann der Algorithmus eine bessere Lösung ausgeben. Aber wenn die Anzahl der Terminal größer als 12 und die Anzahl der Ameisen größer als 6 ist, kann der Algorithmus nicht in kurzer Zeit das Problem schaffen. Deswegen wird die Anzahl der Ameisen für das Problem mit kleiner Anzahl der Terminals immer gleich halbe Anzahl der

Terminals angenommen. Für die Probleme mit der Anzahl der Terminals 13, 14, 15 definieren wir die Anzahl der Ameisen 6, 5, 4. Damit die Lösung in ein und halb Minuten ausgegeben werden kann.

Verflüchtigungsrate

Die Verflüchtigung ρ beeinflusst die Konvergenzgeschwindigkeit und die globale Erkundungsfähigkeit. In diesem Test wird die Anzahl der Ameisen wie oben definiert. Und $\alpha = 0.9, \beta = 1.0, Q = 1.0$ sind die andere Parameterwerte.

ρ		0.3	0.4	0.5	0.6	0.7	0.8
8 Terminals	ST _{best}	975	975	971	975	971	975
	ST _{worst}	987	979	979	983	983	983
	ST _{ave.}	978.3	976.4	974.8	979.2	976.6	978.4
	T	21.2s	21.3s	21.1s	20.6s	22.4s	21.6s
12 Terminals	ST _{best}	1455	1447	1455	1447	1455	1459
	ST _{worst}	1471	1471	1467	1471	1475	1479
	ST _{ave.}	1464.6	1462.2	1462.2	1464.6	1463.8	1468.6
	T	73.8s	72.9s	74.6s	74.1s	73.2s	73.9s
15 Terminals	ST _{best}	1830	1822	1822	1818	1818	1822
	ST _{worst}	1850	1850	1850	1846	1858	1850
	ST _{ave.}	1838	1835.6	1834.8	1836.4	1838	1837.2
	T	89s	88.9s	88.6s	88.2s	88.9s	89s

Tabelle 4.3: Ergebnisse und Laufzeit des Ameisenalgorithmus mit unterschiedlichem Verflüchtigungsrate

Tabelle zeigt die Simulationsergebnisse. Während der Änderung der Verflüchtigungsrate sind alle Ergebnisse fast gleich. Aber wenn Verflüchtigungsrate gleich 0.5 ist, kann eine gute Lösung bekommen werden. Mit unterschiedlicher Verflüchtigungsrate ist die Laufzeit immer gleich. Deswegen wird für weitere Arbeit 0.5 als Verflüchtigungsrate benutzt.

Exponenten α und β

Durch zwei Teste werden Anzahl der Ameisen und Verflüchtigungsrate schon optimal bestimmt. Weil α und β eine starke Beziehung haben, werden die beide Parameter gleichzeitig getestet. Dieses Mal werden die Parameter α und β mit ein paar Werten wie folgende Tabelle getestet.

α, β	ST_{best}	ST_{worst}	ST_{ave}	T
0.5, 1	967	979	973.4	22.4s
0.5, 2	967	982	974	22.7s
0.5, 5	966	971	969.2	22.6s
1, 1	967	991	978.4	22.7s
1, 2	971	979	974	22.8s
1, 5	966	971	970	22.6s
2, 1	971	984	977.4	22.7s
2, 2	978	982	978.4	22.8s
2, 5	966	978	972.8	22.6s

Tabelle 4.4: Ergebnisse und Laufzeit des Ameisenalgorithmus mit unterschiedliche α und β von dem Problem mit 8 Terminals

α, β	ST _{best}	ST _{worst}	ST _{ave}	T
0.5, 1	1459	1479	1467	74.9s
0.5, 2	1447	1459	1454.2	76.1s
0.5, 5	1431	1443	1439	76s
1, 1	1459	1491	1472.8	77.4s
1, 2	1459	1471	1462.4	76.3s
1, 5	1431	1447	1442.2	77.3s
2, 1	1455	1483	1472.6	77s
2, 2	1451	1467	1461.2	75.9s
2, 5	1439	1455	1449.4	74s

Tabelle 4.5: Ergebnisse und Laufzeit des Ameisenalgorithmus mit unterschiedliche α und β von dem Problem mit 12 Terminals

α, β	ST _{best}	ST _{worst}	ST _{ave}	T
0.5, 1	1854	1865	1857.8	78.7s
0.5, 2	1818	1838	1829.2	78.6s
0.5, 5	1790	1810	1796.2	79.5s
1, 1	1842	1861	1851	78.9s
1, 2	1818	1843	1829.2	78.7s
1, 5	1794	1810	1802	77.8s
2, 1	1842	1859	1849.6	80.3s
2, 2	1802	1850	1835.8	78.5s
2, 5	1802	1818	1808.4	77.8s

Tabelle 4.6: Ergebnisse und Laufzeit des Ameisenalgorithmus mit unterschiedliche α und β von dem Problem mit 15 Terminals

In der oben gezeigten drei Tabelle ist deutlich, dass α und β auf den Algorithmus eine große Auswirkung haben. Wenn α groß ist, kann der Algorithmus schnell konvergieren. Aber die Lösung ist immer lokal optimal, deswegen sind die Ergebnisse schlecht. Wenn β klein ist, sucht der

Algorithmus die Lösung zufällig. Deswegen wenn β groß ist, sind die Lösungen immer besser. Für alle Probleme sind 0.5 und 5 gute Parameterwerte für α und β .

Menge des Pheromon

Schließlich wird für Ameisenalgorithmus die Menge des Pheromons, die eine Ameise verteilen darf Q bestimmt. In diesem Test wird der Algorithmus wie immer 10 Mal durchgeführt, während Q gleich 1, 10, 100, 1000 und 10000 sind. Die andere Parametern werden die durch vorherige Teste bekommende beste Werte verwendet.

Q		1	10	100	1000	10000
8 Terminals	ST_{best}	966	966	967	967	971
	ST_{worst}	971	979	971	975	971
	$ST_{ave.}$	969.2	970.8	970.2	971	971
	T	22.6s	22.6s	22.5s	22.7s	22.8s
12 Terminals	ST_{best}	1431	1439	1431	1427	1435
	ST_{worst}	1443	1451	1443	1443	1447
	$ST_{ave.}$	1439	1443.6	1439.3	1439	1441.4
	T	76s	75.5s	76.2s	76.9s	75.7s
15 Terminals	ST_{best}	1790	1798	1794	1798	1794
	ST_{worst}	1810	1806	1811	1806	1818
	$ST_{ave.}$	1796.2	1802	1801.4	1802.8	1808
	T	79.5s	81.2s	79.7s	78s	78.8s

Tabelle 4.7: Ergebnisse und Laufzeit des Ameisenalgorithmus mit unterschiedlicher Q

Wenn der Parameter Q gleich verschiedene Werte ist, sind die Ergebnisse und Laufzeit fast gleich. Das bedeutet, dass Q fast keine Auswirkung auf den Algorithmus hat. Deswegen wird normaler Wert 1 als Parameterwert benutzt.

4.5.3 Genetischer Algorithmus

Der genetische Algorithmus ist für die Parameter auch sensitiv. Mit verschiedenen Parametern können auch die unterschiedlichen Lösungen ausgegeben. Für einzelnes Problem müssen geeignete Parameterwerte bestimmt werden, um das Problem effizient und gut zu lösen. Wie die Methode der Parameterbestimmung für Ameisenalgorithmus wird genetischer Algorithmus mit verschiedenem Parameterwerte mehrmals durchgeführt. Durch Vergleich der Ergebnisse können die passenden Parameterwerte bestimmt werden. In dieser Arbeit werden zwei unterschiedliche Rekombinationsmethoden angewendet. Der genetische Algorithmus mit unterschiedlichen Rekombinationsmethoden hat auch unterschiedliche Performance. Deswegen müssen die Parameter auch für zwei genetische Algorithmen separat bestimmt.

4.5.3.1 Genetischer Algorithmus mit OX-Rekombinationsoperator

In diesem Test werden die Testdaten von Schiffen mit mehr als 8 zu bedienenden Terminals benutzt. Der genetische Algorithmus mit OX-Rekombinationsoperator (GA1) wird genau gleich wie Ameisenalgorithmus 10-mal durchgeführt. Und die durchschnittlichen Ergebnisse werden verglichen. Genetischer Algorithmus sucht zufällige Lösungen. Deswegen muss die Abbruchsbedingung so definiert werden, dass der Algorithmus 100-mal wiederholt werden, damit die beste Lösung nicht

verpasst werden kann. Wie Ameisenalgorithmus ist die kürzere Servicezeit immer die bessere Lösung.

Zuerst müssen die entsprechende Rekombinations- und Mutationswahrscheinlichkeit mit einer angenommenen Größe der Population bestimmt werden.

Rekombinationswahrscheinlichkeit

Die Folgen der steigenden Rekombinationswahrscheinlichkeit sind die mehr Chancen Chromosomen zu rekombinieren. Damit werden mehr verschiedene Chromosomen erzeugt und verglichen. In dem Test werden für die Population, die 30 Chromosomen besitzt, die Rekombinationswahrscheinlichkeit zwischen 0.5 und 0.9 und die Mutationswahrscheinlichkeit 0.005 angenommen. Die Testdaten mit 8, 12, 15 zu bedienenden Terminals werden benutzt. Die folgende Tabelle fasst die Ergebnisse zusammen.

P_c		0.5	0.6	0.7	0.8	0.9
8 Terminals	ST_{best}	979	971	971	971	975
	ST_{worst}	983	975	979	976	983
	$ST_{ave.}$	980	973	975.2	973.8	978.4
	T	19s	19.4s	19.2s	19.3s	19.1s
12 Terminals	ST_{best}	1463	1463	1475	1463	1469
	ST_{worst}	1479	1475	1483	1485	1483
	$ST_{ave.}$	1472.2	1471.4	1477.4	1472.2	1473.2
	T	25.8s	26s	26s	26.1s	26.3s
15 Terminals	ST_{best}	1835	1830	1846	1850	1857
	ST_{worst}	1862	1854	1858	1860	1867
	$ST_{ave.}$	1850.2	1845.4	1854	1853	1860.4
	T	33.1s	33.6s	33.8	32.9	33.9

Tabelle 4.8: Ergebnisse und Laufzeit des GA1 mit unterschiedlicher Rekombinationswahrscheinlichkeit

In der Tabelle ist deutlich, dass mit der Wahrscheinlichkeit zwischen 0.5 und 0.9 gute Lösung bekommen werden kann. Und die Wahrscheinlichkeit hat fast keine Auswirkung auf die Laufzeit. Wenn Rekombinationswahrscheinlichkeit 0.6 ist, können gute Ergebnisse bekommen werden. Dieser Wert wird für weitere Arbeit benutzt.

Mutationswahrscheinlichkeit

Eine Mutation sollen idealerweise nur mit eine kleine Wahrscheinlichkeit hervorrufen, um das Chromosom, das sich bereits nahe eins Optimums gefunden wurde, nicht von diesem Optimum wegzubringen. In oben gezeigtem Test wird die passende Rekombinationswahrscheinlichkeit schon bekommen. Mit der Wahrscheinlichkeit 0.6 wird in diesem Experiment Mutationswahrscheinlichkeit bestimmt. Und die Anzahl der Population wird wie vorherigem Test 30 angenommen.

P_m		0.002	0.004	0.006	0.008	0.01
8 Terminals	ST_{best}	972	975	974	972	971
	ST_{worst}	979	979	983	976	980
	$ST_{ave.}$	976	976.8	976.6	974.6	975.2
	T	18.7s	18.7s	18.9s	18.7s	19.2s
12 Terminals	ST_{best}	1463	1459	1459	1463	1463
	ST_{worst}	1479	1479	1475	1475	1484
	$ST_{ave.}$	1472	1471.2	1470.2	1469.4	1473.6
	T	26.5s	26.6s	26.6s	26.2s	26.5s
15 Terminals	ST_{best}	1832	1838	1842	1823	1834
	ST_{worst}	1860	1862	1863	1849	1862
	$ST_{ave.}$	1846.2	1850.8	1854.8	1838.4	1847.8
	T	31.2s	31.1s	31.3s	32.7s	31.6s

Tabelle 4.9: Ergebnisse und Laufzeit des GA1 mit unterschiedlicher Mutationswahrscheinlichkeit

Die Ergebnisse zeigt, dass wenn die Mutationswahrscheinlichkeit in Intervall 0.002 bis 0.01 liegt, sind die Ergebnisse und Laufzeit fast gleich. Wenn 0.008 als die Wahrscheinlichkeit benutzt wird, sind die Ergebnisse ein bisschen besser als andere. Deswegen wird der Algorithmus in dieser Arbeit mit dem Wert 0.008 verwendet.

Größe der Population

Wie Kapitel 4 gesagt, ist die Größe der Population sehr wichtig für genetischen Algorithmus. In der Test werden Rekombinations- und Mutationswahrscheinlichkeit $p_c = 0.6$, $p_m = 0.008$ definiert. Die Größe der Population wird wie folgende Tabelle jeweilig geprüft.

S		20	40	60	80	100
8 Terminals	ST _{best}	971	971	971	971	971
	ST _{worst}	980	982	967	975	979
	ST _{ave.}	974.4	975.6	974.6	973.4	974.2
	T	22.7s	41s	60.5s	82.8s	102.3s
12 Terminals	ST _{best}	1460	1463	1447	1447	1455
	ST _{worst}	1483	1471	1471	1476	1472
	ST _{ave.}	1472	1467	1460.4	1460.2	1462
	T	34.4s	73.2s	104.5s	145.6s	176.5s
15 Terminals	ST _{best}	1842	1842	1845	1834	1838
	ST _{worst}	1862	1866	1858	1849	1850
	ST _{ave.}	1855	1855.2	1850.2	1842.6	1842.8
	T	48.1s	96.8s	148.8s	196.4s	248.1s

Tabelle 4.10: Ergebnisse und Laufzeit des GA1 mit unterschiedlicher Größe der Population

Die Tabelle zeigt die Ergebnisse, die mit unterschiedlicher Größe der Population bekommen werden. Wie erwartet, dass je größer die Population ist, desto bessere Lösung ausgegeben werden kann. Aber die Laufzeit ist auch schnell wachsend. Deswegen kann die große Population nicht helfen, das Problem schnell zu lösen. Für die Probleme mit unterschiedlicher Größe werden unterschiedliche Parameterwerte benutzt. Obwohl der Algorithmus mit einer Größe der Population 80 die beste Lösung liefern kann, kann die Größe der Population wegen der Laufzeit leider nicht für alle Situationen 80 gewählt werden. Für das Problem mit 7 und 8 zu bedienenden Terminals wird 80 als Größe der Population benutzt, weil das Ergebnis besser ist. Aber wenn die Anzahl der zu bedienenden Terminals mehr als 8 sind, wird die Größe der Population gemäß der Laufzeit gewählt. Deswegen werden die Größe der Population 40 für die Probleme mit 9,10,12,13 zu bedienenden Terminals und 30 für die Probleme mit 14 und 15 zu bedienenden Terminals angenommen.

4.5.3.2 Genetischer Algorithmus mit One-Point-Crossover

In diesen Test werden die Parameter für genetischen Algorithmus mit One-Point-Crossover optimieren. Wegen der Laufzeit wird die Größe der Population wie oben definiert. Hier müssen nur Rekombinations- und Mutationswahrscheinlichkeit bestimmt werden.

Rekombinationswahrscheinlichkeit

Wie letztem Test werden die Rekombinationswahrscheinlichkeit zwischen 0.5 und 0.9 und die Mutationswahrscheinlichkeit 0.005 angenommen. Die Größe der Population wird den von letztem Test bekommenen optimale Wert benutzt.

P_c		0.5	0.6	0.7	0.8	0.9
8 Terminals	ST_{best}	939	939	939	941	939
	ST_{worst}	947	953	947	953	947
	$ST_{ave.}$	943.8	945	941.8	945.8	941.4
	T	82s	80.7s	83.7s	81.8s	82.3s
12 Terminals	ST_{best}	1427	1423	1423	1427	1415
	ST_{worst}	1451	1451	1443	1451	1443
	$ST_{ave.}$	1440.6	1437.4	1432.6	1435.8	1431.8
	T	62.8s	64.2s	65.1s	65.6s	64.7s
15 Terminals	ST_{best}	1750	1774	1770	1738	1750
	ST_{worst}	1818	1798	1786	1806	1802
	$ST_{ave.}$	1788.4	1788.4	1782	1778.8	1777.2
	T	71.4s	68.4s	69.8s	69.3s	68.5s

Tabelle 4.11: Ergebnisse und Laufzeit des GA2 mit unterschiedlicher Rekombinationswahrscheinlichkeit

Aus der Tabelle wird deutlich, wenn die Rekombinationswahrscheinlichkeit zwischen 0.5 und 0.9 ist, haben die Ergebnisse nicht so großes Unterschiedmaß.

Aber wenn die Wahrscheinlichkeit gleich 0.9 ist, kann der Algorithmus bessere Lösung ausgeben. Aus diesem Grund wird 0.9 als Rekombinationswahrscheinlichkeit in weiterer Simulation verwendet.

Mutationswahrscheinlichkeit

Folgend wird die Mutationswahrscheinlichkeit optimiert. In diesem Test wird oben von dem Test bekomme Rekombinationswahrscheinlichkeit benutzt.

P_m		0.002	0.004	0.006	0.008	0.01
8 Terminals	ST_{best}	939	939	939	939	939
	ST_{worst}	955	951	951	951	951
	$ST_{ave.}$	945.4	941.8	943.4	941.4	942.2
	T	81.6s	82.9s	83.8s	82.5s	81.7s
12 Terminals	ST_{best}	1415	1415	1423	1407	1423
	ST_{worst}	1455	1459	1443	1427	1455
	$ST_{ave.}$	1441.4	1437.4	1431.8	1418.2	1434.2
	T	64.1s	64.9s	68.1s	64.7s	65.9s
15 Terminals	ST_{best}	1742	1734	1746	1726	1734
	ST_{worst}	1778	1778	1766	1762	1774
	$ST_{ave.}$	1766.8	1762.8	1750	1746	1756.3
	T	67.7s	67.2s	68.3s	71.1s	68.7s

Tabelle 4.12: Ergebnisse und Laufzeit des GA2 mit unterschiedlicher Mutationswahrscheinlichkeit

Die Tabelle zeigt die Ergebnisse, die durch den genetischen Algorithmus mit unterschiedlicher Mutationswahrscheinlichkeit gerechnet werden. Wenn die Mutationswahrscheinlichkeit zu klein ist, wird immer das lokale Optimum ausgegeben. Wenn der Algorithmus den Wert 0.008 als

Mutationswahrscheinlichkeit benutzt, kann ein besseres Ergebnis bekommen werden.

4.5.4 NSGA2

NSGA2 ist einen multiobjektiven Optimierungsalgorithmus. Wie oben gesagt basiert NSGA2 auch auf genetischem Algorithmus. Wegen die Laufzeit wird die Population gleich wie genetischem Algorithmus definiert. Weil NSGA2 auch einen One-Point-Crossover-Operator benutzt, müssen die Rekombinations- und Mutationswahrscheinlichkeit nicht noch mal optimieren. Für den Tournament-Selektor muss noch ein Parameter bestimmen.

Anzahl der Chromosomen in jedem Turnier

In diesem Test wird der Parameter mit den Werten 2, 4, 8, 16 getestet. Andere Parameterwerte werden die von oben gezeigtem Test bekomme optimale Werte benutzt.

n		2	4	8	16
8 Terminals	ST _{best}	935	935	935	935
	ST _{worst}	935	935	935	935
	ST _{ave.}	935	935	935	935
	T	81.1s	81.8s	82.1s	81.4s
12 Terminals	ST _{best}	1407	1407	1407	1407
	ST _{worst}	1419	1423	1427	1427
	ST _{ave.}	1411.8	1413.4	1415.8	1415.8
	T	64.8s	64.4s	65.1s	64.2s
15 Terminals	ST _{best}	1730	1738	1730	1746
	ST _{worst}	1754	1754	1774	1766
	ST _{ave.}	1744.6	1746	1748.4	1754
	T	68.4s	67.6s	66.5s	66.9s

Tabelle 4.13: Ergebnisse und Laufzeit des NSGA2 mit unterschiedlicher Anzahl der Akteure

In der Tabelle werden die Ergebnisse gezeigt. Für die kleinen Probleme kann der Algorithmus immer gute Lösung ausgeben, obwohl der Parameter unterschiedliche Werte benutzt. Aber wenn Problem groß ist, wird gutes Ergebnis von dem Algorithmus mit Parameterwert 2 bekommen. Deswegen wird der Parameterwert 2 als Anzahl der Chromosomen in jedem Turnier für alle Probleme verwendet.

5 Evaluation

In diesem Kapitel werden die Optimierungsalgorithmen simuliert und bewertet. Zuerst werden im Abschnitt 5.1 relevante Metriken definiert. Danach werden die Methodiken zur Evaluation im Abschnitt 5.2 erklärt. Es folgt im Abschnitt 5.3 die eigentliche Auswertung der Ergebnisse. Eine kurze Zusammenfassung der Ergebnisse wird abschließend im Abschnitt 5.4 vorgenommen.

5.1 Metriken für die Evaluation

Um die Optimierungsalgorithmen zu vergleichen, werden passende Metriken zur Bemessung benötigt. Die wichtige Metriken für die Optimierungsalgorithmen sind Laufzeitkomplexität und Richtigkeit.

Laufzeitkomplexität:

Die Laufzeitkomplexität ist ein Maß für die Geschwindigkeit der Berechnung. Die Laufzeiten von den Algorithmen werden verglichen. Und Bei der kurzen Laufzeit liegt eine niedrige Komplexität vor. In dieser Arbeit ist Zeit ein Stichwort. Wenn die Rechenzeit zu lang ist, muss das Schiff immer auf Tourplanung warten. Und noch schlimmer ist, wenn andere Schiffe gleichzeitig im Hafen angekommen sind, müssen die Schiffe längere Zeit auf der Tourplanung warten. Deswegen ist lange Rechenzeit nicht geeignet für diese Arbeit.

Richtigkeit:

Und die Richtigkeit wird in diesem Kontext das Unterschied zwischen beste Tour und von den Algorithmen gerechnete beste Tour. Weil die Heuristiken nicht

immer die beste Tour berechnen kann. Die beste Tour kann mit Hilfe von erschöpfender Suche gefunden werden. Die gesamte Servicezeit wird berechnet und verglichen. Je größer Unterschied von Servicezeit gibt, desto schlechter das Ergebnis ist.

5.2 Methodiken zur Evaluation

Die Optimierungsalgorithmen werden im Hafen in vielen verschiedenen Situationen verwendet. Für unterschiedliche Situationen ist die Performance der Optimierungsalgorithmen nicht gleich. Deswegen müssen alle oben gezeigte Optimierungsalgorithmen für unterschiedliche Situationen testen, um den beste Algorithmus auszuwählen.

Anzahl der Terminals:

Ein Schiff möchte normalerweise nicht nur ein Terminal besuchen. Je mehr Terminals das Schiff besuchen möchte, desto mehr mögliche Touren es gibt. Manche Algorithmen sind für großen Suchraum geeignet und andere sind nur für kleinen Suchraum durchführbar. In dieser Arbeit werden unterschiedliche Testdaten generiert. Die unterschiedlichen Schiffe haben 3 bis 15 zu bedienende Terminals.

Anzahl der ankommenden Schiffe pro Stunde:

Während ein Schiff im Hafen angekommen ist, sind die Terminals manchmal meistens schon besetzt oder reserviert. Die Algorithmen werden in drei Situationen getestet.

1. Jede Stunde kommen nur wenige Schiffe in den Hafen an. Das heißt, meiste Terminals sind frei und nicht reserviert. Wenn ein Schiff angekommen ist, kann das Schiff ohne Wartezeit sofort bearbeitet werden.
2. Die Dichte der kommenden Schiffe ist normal. Wegen kommenden Schiffen sind manche Terminals besetzt oder reserviert. Wenn anderes Schiff schon im Terminal ist, muss das Schiff manchmal warten.
3. Sehr viele Schiffe kommen in den Hafen an. Das bedeutet, meiste Terminals sind besetzt oder reserviert. Ein kommendes Schiff muss immer warten bis andere Schiffe fertig sind.

5.3 Evaluation

Für die Evaluation wird jeder Optimierungsalgorithmus mehrmals mit unterschiedlichen Testdaten in drei Situationen simuliert. Die Simulation wird zuerst von erschöpfender Suche durchgeführt. Damit können die optimalen Lösungen bekommen werden. Danach wird die Simulation von vier Heuristiken durchgeführt. Durch Simulation können wir wissen, welcher Algorithmus passt welchen Testdaten.

5.3.1 Erschöpfende Suche

Obwohl die erschöpfende Suche lange Zeit laufen muss, kann sie immer die optimale Lösung finden. Das ist sehr wichtig für die Evaluation. Die folgende Tabelle zeigt die Servicezeiten der optimalen Touren als die Ergebnisse. K1 bedeutet ökonomisches Kriterium. K2 bedeutet schnellstes Kriterium.

Anzahl der zu bedienenden Terminals	7	8	9
geringer Verkehr (K1)	842	935	1052
normaler Verkehr (K1)	850	935	1052
hoher Verkehr (K1)	848	948	1056
geringer Verkehr (K2)	795	891	1001
normaler Verkehr (K2)	801	891	1002
hoher Verkehr (K2)	801	904	1006
Laufzeit	3 Minuten	0.4 Stunden	3.5 Stunden

Tabelle 5.1: minimale Servicezeit der Tourplanung

Leider können nicht alle Tourplanungsprobleme von erschöpfender Suche gelöst werden. Wenn ein Schiff 10 zu bedienenden Terminals hat, muss der Algorithmus mehr als einen Tag laufen, um das Problem zu lösen. Wenn das Problem noch größer ist, ist die Laufzeit noch viel länger. Deswegen werden in dieser Arbeit nur die oben gezeigten Lösungen gerechnet.

Wenn das Problem groß ist, das heißt, dass ein Schiff mehr als 7 zu bedienenden Terminals hat, kann erschöpfende Suche das Problem nicht in kurzer Zeit lösen. Deswegen werden die Heuristiken verwendet.

5.3.2 Heuristiken

Für die Heuristiken wird die Simulation dreimal mit oben gezeigte drei unterschiedliche Datengruppen von Testdaten durchgeführt. Heuristiken suchen immer zufällige Lösungen. Deswegen sind die Ergebnisse vielleicht jedes Mal nicht gleich. Aus diesem Grund wird jeder Optimierungsalgorithmus 10 Mal durchgeführt. Die durchschnittlichen Ergebnisse werden verglichen. Die von dem

Experiment bekommen optimalen Parameterwerte werden in weiterem Test benutzt.

Wie im Kapitel 2 gezeigt werden ein Ameisenalgorithmus (ACO), zwei genetischer Algorithmus und ein multiobjektiver Optimierungsalgorithmus NSGA2 getestet. GA1 ist der genetische Algorithmus mit Roulette-Wheel-Selektion, OX Rekombinationsoperator und SWAP Mutationsoperator. GA2 ist der genetische Algorithmus mit Elitistselektion, One-Point-Crossover und SWAP Mutationsoperator.

In diese Arbeit werden die Be- und Entladezeit konstant angenommen. Deswegen werden die minimale Summe von Wartezeit und Fahrzeit fokussiert. Für weitere Simulation werden nur die minimale Summe von Wartezeit und Fahrzeit gezeigt.

5.3.2.1 Simulation mit Testdaten für geringen Verkehr

Zuerst werden die Optimierungsalgorithmen mit Testdaten für geringen Verkehr simuliert.

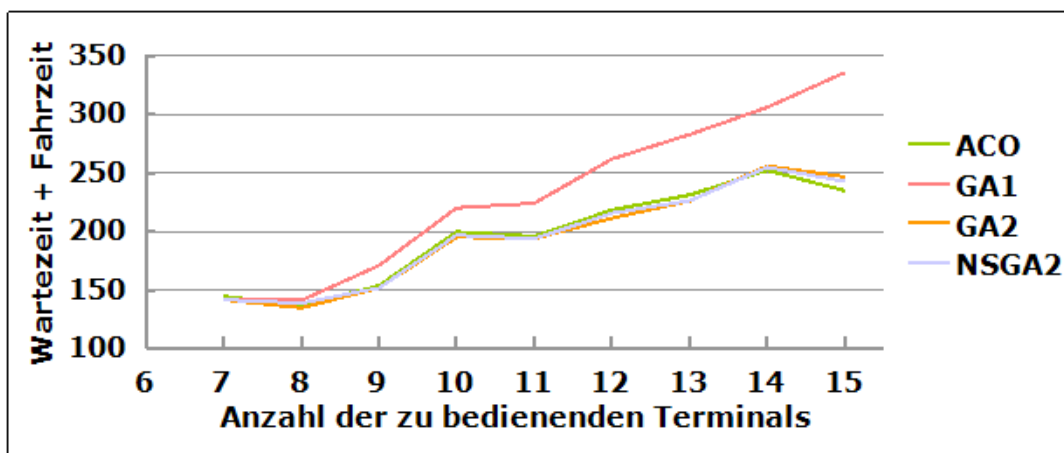


Abbildung 5.1: Lösungen für geringen Verkehr mit ökonomischem Kriterium

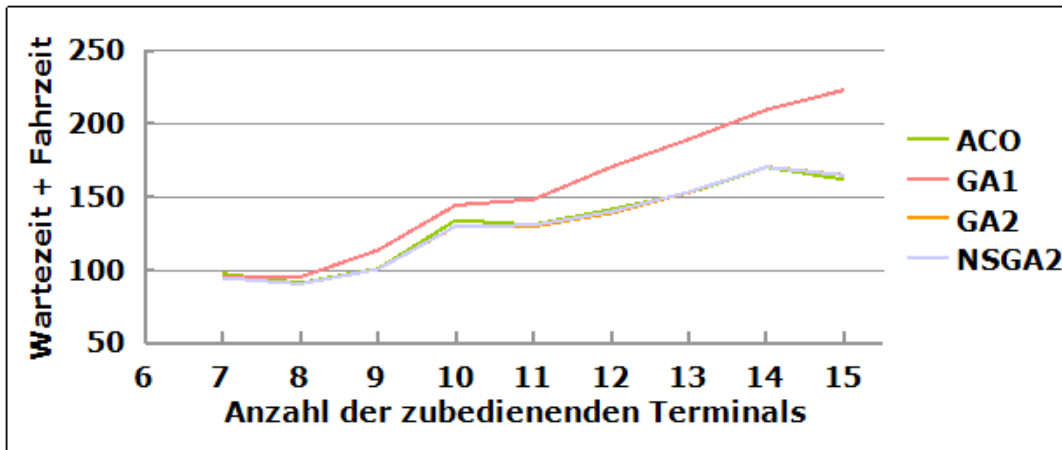


Abbildung 5.2: Lösungen für geringen Verkehr mit schnellstem Kriterium

In den Abbildung 5.1 und 5.2 werden die Kurven von den Lösungen für geringen Verkehr mit ökonomischem und schnellstem Kriterium gezeigt. In den Abbildungen wird deutlich, dass die dargestellten Kurven von ACO, GA2 und NSGA2 einen fast identischen Verlauf zeigen. Die durch GA2 bekommenen Lösungen sind ein bisschen besser als NSGA2. Aber GA1 kann nur eine schlechtere Lösung ausgeben. Wenn das Problem klein ist, zum Beispiel es gibt nur 7, 8 oder 9 zu bedienende Terminals, können ACO, GA2 und NSGA2 eine sehr gute Lösung bekommen. Und diese Lösungen sind die globalen Optima. Wenn das Problem nicht so groß ist, wie zum Beispiel die Anzahl der zu bedienenden Terminals ist größer als 9 und kleiner als 15, kann GA2 ein besseres Ergebnis liefern. Aber wenn die Anzahl der zu bedienenden Terminals gleich 15 ist, kann ACO eine bessere Lösung ausgeben. Für diese Arbeit ist die Laufzeit begrenzt. Und für genetischen Algorithmus kann nur die Population wegen der Laufzeit reduzieren. Deswegen kann genetischer Algorithmus nicht so gut Ergebnis bekommen.

5.3.2.2 Simulation mit Testdaten für normalen Verkehr

In diesem Unterkapitel wurden die Optimierungsalgorithmen mit Testdaten für normalen Verkehr simuliert. Folgende Abbildung 5.3 und 5.4 zeigen die Kurven von den Lösungen für normalen Verkehr mit zwei unterschiedlichen Kriterien.

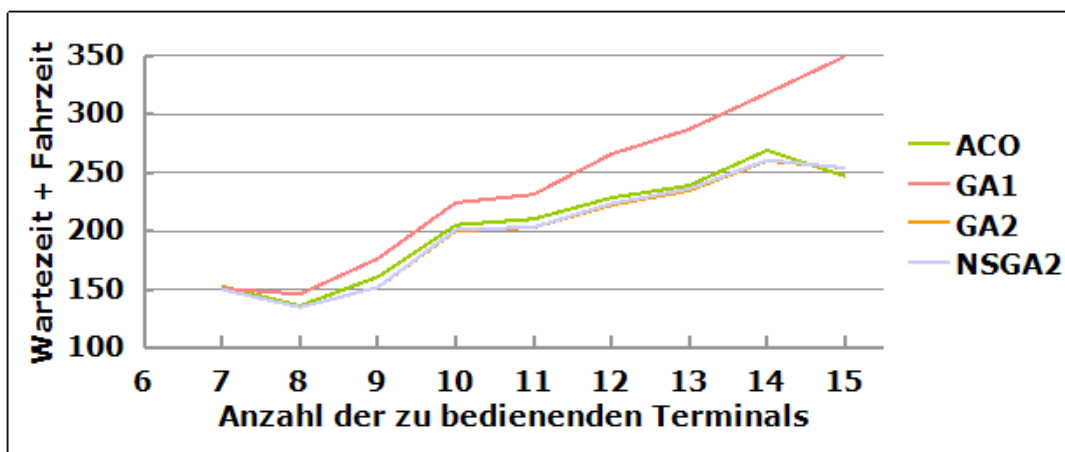


Abbildung 5.3: Lösungen für normalen Verkehr mit ökonomischem Kriterium

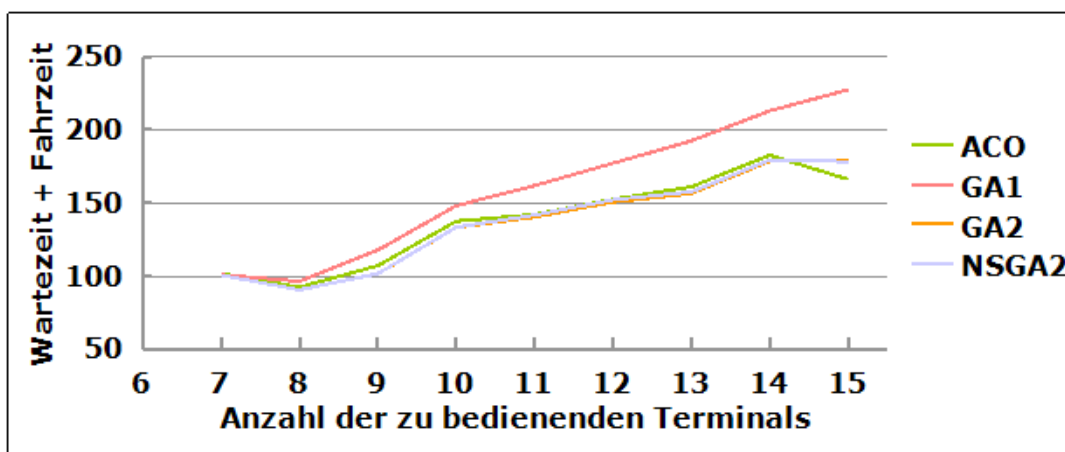


Abbildung 5.4: Lösungen für normalen Verkehr mit schnellstem Kriterium

Für die Testdaten mit normalem Verkehr sind die Kurven von den Ergebnissen ähnlich wie letztem Test. Durch GA1 werden immer die schlechtesten Lösungen ausgeliefert. Für die Probleme mit 7, 8 oder 9 zu bedienenden Terminals werden immer die globale Optima durch GA2 bekommen. Und wenn ein Schiff nicht

mehr als 14 Terminals besuchen möchte, können die gute Lösungen durch GA2 erzeugt. Für ein großes Problem, wie zum Beispiel mit 15 zu bedienenden Terminals, kann ACO die beste Lösung in kurzer Zeit ausgeben.

5.3.2.3 Simulation mit Testdaten für hohen Verkehr

In der Hauptsaison kommen mehr Schiffe im Hamburger Hafen an. In diesem Zeitraum sind vielleicht die Terminals immer reserviert. Die Optimierungsalgorithmen müssen mehr rechnen. In den Abbildung 5.5 und 5.6 werden die Ergebnisse für hohen Verkehr gezeigt.

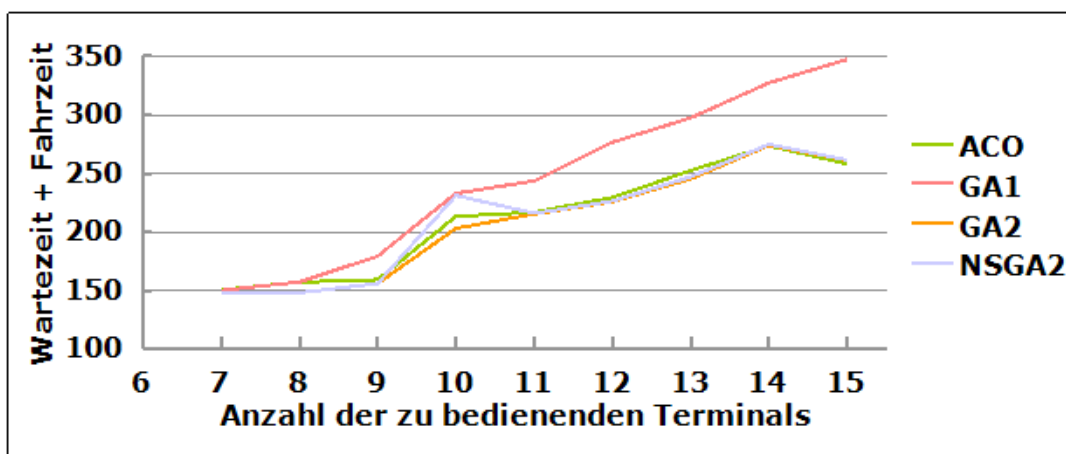


Abbildung 5.5: Lösungen für hohen Verkehr mit ökonomischem Kriterium

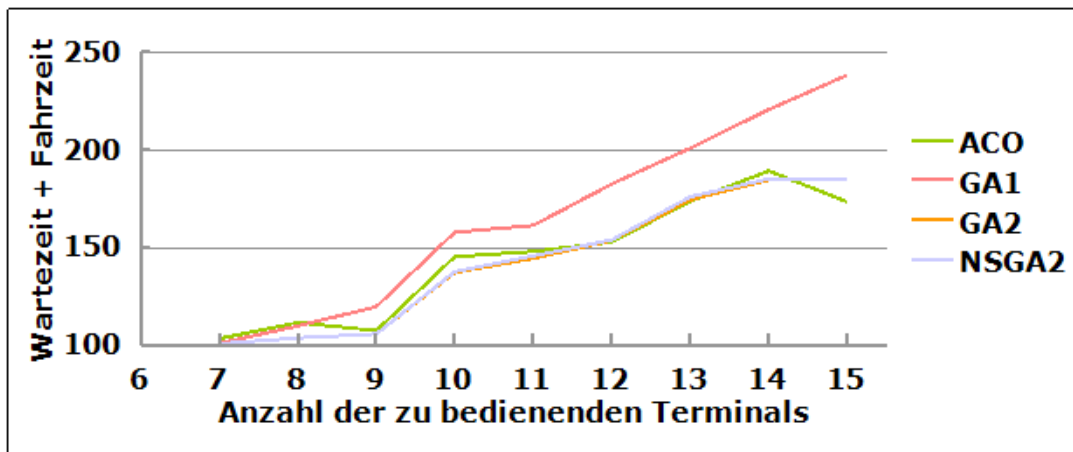


Abbildung 5.6: Lösungen für hohen Verkehr mit schnellstem Kriterium

Die Verläufe der Kurven von den Ergebnisse sind wie letzten zwei Teste immer identisch. Für hohen Verkehr sind die minimale Summe von Wartezeit und Fahrzeit offensichtlich länger als andere Situationen. GA2 hat immer die beste Performance für ein nicht großes Problem. ACO ist auch wie immer für großes Problem geeignet.

5.4 Vergleich der Optimierungsalgorithmen

Die drei unterschiedlichen Situationen werden schon simuliert und vier Heuristiken werden auch getestet. Aus dem Unterkapitel 5.3 wird deutlich, dass erschöpfende Suche kann kleines Problem von der Tourplanung sehr gut und schnell lösen. Aber die Laufzeit der erschöpfenden Suche wächst mit höherer Anzahl der zu bedienenden Terminals sehr schnell. Deswegen wird Heuristiken verwendet. Für die nicht großen Probleme kann genetischer Algorithmus (GA2) mit Elitistselektion, One-Point-Crossover mit Rotation und Swap-Mutation schnell und gut lösen. Durch GA2 wird die Lösung eines großen Problems auch lange Zeit berechnet. Obwohl kann die Laufzeit des GA2 verringert werden, wird eine schlechte Lösung ausgeliefert. Deshalb wird Ameisenalgorithmus für eine

große Tourplanung benutzt. Folgende Tabelle zeigt die passenden Algorithmen mit optimalem Parameterwert. Für genetischen Algorithmus muss die Größe der Population gleich 40 sein, falls die Anzahl der zu bedienenden Terminals nicht mehr als 13 ist. Und wenn die Anzahl der zu bedienenden Terminals gleich 14 ist, muss die Größe der Population wegen der Laufzeit ein bisschen verkleinert werden.

Anzahl der zu bedienenden Terminals	Optimierungsalgorithmus	optimaler Parameterwert
≤ 6	erschöpfende Suche	kein Parameter
7 ~ 14	Genetischer Algorithmus (Elitistselektion, One-Point-Crossover mit Rotation, Swap-Mutation)	$S = 40$ (7 ~ 13), 30 (14), $p_c = 0.9$, $p_m = 0.008$
15	Ameisenalgorithmus	$m = 4$, $\rho = 0.5$, $\alpha = 0.5$, $\beta = 5$, $Q = 1$

Tabelle 5.2: passende singleobjektive Optimierungsalgorithmen und Konfiguration für Tourplanung

Ein singleobjektiver Optimierungsalgorithmus kann die minimale Servicezeit ausrechnen. Aber manchmal möchte die Reederei den Verbrauch von Seefracht verringern. Aus diesem Grund kann multiobjektiver Optimierungsalgorithmus wie zum Beispiel NSGA2 verwendet werden. Durch NSGA2 kann minimale Fahrzeit und minimale Wartezeit ausgerechnet werden. Je nach den unterschiedlichen Situationen kann eine von solche Lösungen ausgewählt. Aber NSGA2 basiert auf genetischen Algorithmus. Deswegen ist er für großes Problem

nicht geeignet. Wie genetischem Algorithmus muss die Größe der Population des NSGA2 für unterschiedliche Anzahl der zu bedienenden Terminals unterschiedlich definiert werden.

Anzahl der zu bedienenden Terminals	Optimierungsalgorithmus	optimaler Parameterwert
≤ 14	NSGA2	$S = 40 (\leq 13), 30(14),$ $n = 2,$ $P_c = 0.9,$ $P_m = 0.008$

Tabelle 5.3: passende Konfiguration von NSGA2

6 Zusammenfassung und Ausblick

In diese Diplomarbeit wurde anhand verschiedener Literaturquellen ein Einblick in das Gebiet Optimierungsalgorithmen zur Tourplanung gegeben. Hierbei wurde gezeigt, dass durch Optimierungsalgorithmen die beste Tour im Hafen bestimmt wird. Zuerst wird die Tourplanung analysiert. Damit wird die Anforderung bzw. die notwendigen Informationen von Terminals und Schiffen gezeigt. Danach werden sowohl die singleobjektive als auch multiobjektive Optimierungsalgorithmen erklärt und verglichen. Dann werden die Testdaten nach aktueller Situation im Hamburger Hafen generiert. Anschließend werden die passenden Parameterwerte für die Optimierungsalgorithmen bestimmt. Schließlich werden die Optimierungsalgorithmen mit drei Gruppen von Testdaten simuliert und am Ende bewertet. Es wird gezeigt, welcher Algorithmus für welche Situation geeignet ist.

Durch Simulation von Parameterbestimmung wurden verschiedene Lösungen mit unterschiedlichen Parameterwerten bekommen. Das beweist, dass die Optimierungsalgorithmen mit unterschiedlichen Parameterwerten unterschiedliche Performance haben. Und der gleiche Algorithmus mit unterschiedlichen Operatoren kann auch unterschiedliche Lösungen ausliefern. Durch Simulation von Heuristiken wurden gute Ergebnisse erreicht. Das bedeutet, dass die Heuristiken auch eine gute Lösung für die Tourplanung sind. Als Ergebnis dieser Diplomarbeit ist eine wertvolle Referenz für die weitere Forschung zum Thema Tourplanung im Hafen.

In dieser Arbeit wurde Wartezeit-Profil für einzelnen Liegeplatz verwendet. In weiterer Arbeit kann noch andere Profile wie zum Beispiel das Wartezeit-Profil für ein Terminal mit mehreren Liegeplätzen in [DSS09] oder auch Servicezeit-Profil in [DSS08] verwendet werden. Die anderen Profile sind vielleicht effizienter.

Im Vergleich zur erschöpfenden Suche kann manchmal durch Heuristik wie zum Beispiel Ameisenalgorithmus nicht perfektes Ergebnis ausgerechnet werden. Vielleicht wurden die Parameter nicht sehr gut optimiert. Die Methode der Parameterbestimmung kann in weiterer Arbeit verbessert werden. Es gibt viele andere Methoden. Parameterbestimmung von Ameisenalgorithmus ist auch ein Optimierungsproblem. Im Vergleich zum Ameisenalgorithmus hat genetischer Algorithmus nicht so viele Parameter. Und es ist nicht so schwer, die Parameter von genetischem Algorithmus zu bestimmen. Aus diesem Grund kann genetischer Algorithmus angewendet werden, um die Parameter von Ameisenalgorithmus zu optimieren.

Andererseits können die Heuristiken in weitere Arbeit auch verbessert werden. Genetischer Algorithmus und Ameisenalgorithmus haben eigene Vorteile und Nachteile. Eine Lösung ist Hybrid-Algorithmus. Ein Hybrid-Algorithmus kann Ameisenalgorithmus als Hauptteil verwenden und mit genetischem Algorithmus zusammenarbeiten. Die Rekombination von genetischem Algorithmus kann in Ameisenalgorithmus einbetten. Durch Rekombination kann neue Tour erzeugt werden, um die globale Suchfunktion zu verbessern. Eine andere Methode ist, dass für großes Problem die Ergebnisse von genetischem Algorithmus als Startwert von Ameisenalgorithmus benutzt werden können, um die Lösungen von Ameisenalgorithmus schnell und gut zu konvergieren.

Literaturverzeichnis

[BD00]: E. Bonabeau, M. Dorigo und G. Theraulaz. Inspiration for optimization from social insect behaviour, in: Nature 406 2000, S.39-42

[BDG99]: E. Bonabeau, M. Dorigo und G. Theraulaz. Swarm Intelligence: From Nature to Artificial Systems, 1999

[BM01]: E. Bonabeau und C. Meyer. Schwarm-Intelligenz: Unternehmen lernen von Bienen und Ameisen, in: Harvard Business manager 2001 Heft 6, S.38-48

[CDM91]: A. Colomi, M. Dorigo und V. Maniezzo. Distributed optimization by ant colonies, 1991

[D85]: L. Davis. Applying Adaptive Algorithms to Epistatic Domains, Proc. 9th International Joint Conference on Artificial Intelligence (S. 162-164), 1985

[D92]: Marco Dorigo. Optimization, learning and natural algorithms. Ph. D. Thesis, Department of Electronics, Politecnico di Milano, Italy, 1992

[DM91]: M. Dorigo und V. Maniezzo. Ant system: An autocatalytic optimizing process, Technical Report No. 91-016 Revised, Politecnico di Milano, Italy, 1991

[DMC91]: Marco Dorigo, Vittorio Maniezzo und Alberto Colomi. Positive Feedback as a Search Strategy, Technical report, Department of Electronics, Milan Polytechnic Institute, June 1991

[DPAM02]: K. Deb, A. Pratap, S. Agarwal und T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA2. In: IEEE Transactions on Evolutionary Computing 6 (2002), Nr. 2, S. 182-197

[DSS08]: A. Douma, M. Schutten und P. Schuur. Using service-time profiles for distributed planning of container barge rotations along terminals. University of Twente, The Netherlands, 2008

[DSS09]: A. Douma, M. Schutten und P. Schuur. Waiting profiles: An efficient protocol for enabling distributed planning of container barge rotations along terminals in the port of Rotterdam. *Transportation Research, Part C: Emerging technologies*, 17(2). pp. 133-148.

[G89]: D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989

[GKD89]: D.E. Goldberg, B. Korb und K. Deb. Messy Genetic Algorithms: Motivation, Analysis, and First Results, *Complex Systems* 3 (1989), S. 493-539

[KS]: K. Schittkowski. *Script: Mathematische Grundlagen von Optimierungsverfahren*

[LGRT11]: M. Lukasiwycz, M. Glaß, F. Reimann und J. Teich. Opt4J - A Modular Framework for Meta-heuristic Optimization. *Proc. GECCO2011, Dublin, Ireland 2011*

[PFM06]: Paola Pellegrini, Daniela Favaretto und Elena Moretti. On MAX-MIN Ant System's Parameters, in: Marco Dorigo, et al. *Ant Colony Optimization and Swarm Intelligence*. Brüssel, Belgien: Springer, 2006

[R06]: Dirk Reichelt. *Kommunikationsnetzwerkplanung unter Kosten- und Zuverlässigkeitsgesichtspunkten mit Hilfe von evolutionären Algorithmen*. Dissertation, Fakultät für Wirtschaftswissenschaften, Technische Universität Ilmenau, Germany, 2006

[SD94]: N. Srinivas und K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. In: Evolutionary Computation 2 (1994), Fall, Nr. 3, S. 221-248

[WWW1]: Wikipedia. Hamburger Hafen. http://de.wikipedia.org/wiki/Hamburger_Hafen abgerufen am 26.06.2011

[WWW2]: hafen-hamburg.de <http://www.hafen-hamburg.de/> abgerufen am 04.07.2011

[ZLB04]: E. Zitzler, M. Laumanns und S. Bleuler. A Tutorial on Evolutionary Multiobjective Optimization, in: Metaheuristics for Multiobjective Optimisation 535 2004, S. 3-38