UNIVERSITÄT
LEIPZIG

Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik
Schwarmintelligenz und Komplexe Systeme

Entwicklung eines ACO basierten Algorithmus zur Lösung eines multikriteriellen Optimierungsproblems in der Auftragsplanung

Masterarbeit

Leipzig, Mai, 2021

vorgelegt von

Falk Müller

Studiengang: MSc. Informatik

Betreuender Hochschullehrer: Prof. Dr. Martin Middendorf

Danksagung

An dieser Stelle möchte ich mich bei Prof Dr. Martin Middendorf für die Betreuung der Masterarbeit bedanken. Des weiteren Danke ich Ronny Kind für die Geduld und Unterstützung während der Zeit des Schreibens und Korrekturen dieser Arbeit.

Besonderer Dank gilt Hoang Thanh Le für die Betreuung und die vielen Hinweise zur Verbesserung dieser Arbeit. Er hatte immer ein offenes Ohr für Rückfragen.

Inhaltsverzeichnis

Symbole

J	Set aller Aufträge
|J|	Anzahl der Aufträge
M	Set aller Bearbeiter
|M|	Anzahl der Bearbeiter
p_{ij}	Die Wahrscheinlichkeit für den Pfad ij
τ_{ij}	Pheromon-Wert auf dem Pfad ij
$\Delta\tau_{ij}$	Pheromon-Wert für die Erhöhung des Pheromons auf Pfad ij
η_{ij}	Heuristischer Wert für den Pfad ij
α	Exponentialfaktor für den Pheromon-Wert
β	Exponentialfaktor für den Heuristik-Wert
K	Anzahl der Ameisen
N	Anzahl der Iterationen
d	Länge einer Wegstrecke
C	Kosten

1. Einführung

1.1. Motivation

Diese Arbeit soll helfen, ein reales Problem in der Auftragsverteilung zu lösen. Da sich Ameisenalgorithmen bereits für eine Vielzahl von ähnlichen Problemen als nützlich erwiesen haben, wie bei kombinatorischen Optimierungsproblemen, wie dem Job Shop Scheduling Problem [22], dem Parallel-Machine-Scheduling-Problem [23] oder dem Quadratic Assignment Problem [25], soll auch dieses Optimierungsproblem mithilfe von Ameisen gelöst werden.
Im Folgenden wird zuerst das zu lösende Problem genauer erörtert. Kapitel 2 gibt eine Einweisung in die Grundlagen des Ameisenalgorithmus und erörtert diese. Anschließend wird in Kapitel 3 auf die Implementierungs-Details eingegangen und es werden verschiedene Variationen des Algorithmus vorgestellt. In den Tests (Kapitel 4) werden zuerst die Konfigurations-Parameter bestimmt und dann die verschiedenen Algorithmen durch Messungen miteinander verglichen.

1.2. Aufgabenstellung

Gesucht wird ein Algorithmus zur Auftragsverteilung. Die genauere Definition der zu lösenden Aufgabe entstammt der Auftragsverteilung einer bundesweit agierenden Immobilienfirma, deren Geschäft im Besichtigen und Bewerten des Wertes von Immobilien besteht. Bedingt durch das Geschäftsgeheimnis sind die in dieser Arbeit verwendeten Zahlen (von Mitarbeitern, Aufträgen, ...) zur Modellierung des Problems teils frei gewählt oder verändert. Der zu implementierende Algorithmus soll Aufträge an Bearbeiter zuweisen und dabei die Kosten minimieren, welche unter Anderem aus Lohnkosten, Fahrtkosten, Verspätungskosten und einem Unzufriedenheits-Wert der Mitarbeiter bestehen. Im Kapitel zur Implementierung wird dabei formal genauer auf das Problem eingegangen.

1.3. Ausgangslage

Als **Auftrag** wird die Besichtigung einer Immobilie durch einen Außendienstmitarbeiter bezeichnet. Die bundesweite Verteilung dieser Aufträge entspricht in etwa der Bevölkerungsverteilung. Je nach Objekttyp der Immobilie (Etagenwohnung, Mehrfamilienhaus, ... bis hin zum Krankenhaus) kann eine solche Besichtigung zwischen einer Stunde und mehreren Tagen dauern. Die Einnahmen je Auftrag variieren, genau wie die Dringlichkeit eines Auftrages und die damit verbunden Preisabschläge.

Die Firma verfügt über mehrere hundert **Außendienstmitarbeiter**, auch als Besichtiger, Bearbeiter oder Mitarbeiter bezeichnet. Diese wohnen und arbeiten bundesweit verteilt. Die Verteilung entspricht in etwa dem aufkommenden Auftragsvolumen in den jeweiligen Regionen und der Bevölkerungsverteilung. Diese Außendienstmitarbeiter sind regionalen Niederlassungen je Bundesland zugeordnet.
Zusätzlich gibt es freie Mitarbeiter, welche sich von den festen Mitarbeitern im Wesentlichen durch höhere Kosten unterscheiden. Diese sollten deshalb nur in einem sinnvollen Kosten/Nutzen Verhältnis hinzugezogen werden.
Für einen Teil der Besichtigungen sind Zertifizierungen vorgeschrieben, über welche ca. die Hälfte der Außendienstmitarbeiter verfügen. Dies ist also eine Eigenschaft von Aufträgen, wodurch nicht jeder Mitarbeiter, jeden Auftrag bearbeiten kann.
Zum Anderen gibt es optionale Eigenschaften von Aufträgen und Mitarbeitern, welche eine Bevorzugung der Bearbeitung eines Auftrages durch einen bestimmten Mitarbeiter bewirken. Wenn ein Mitarbeiter viele Aufträge eines Objekttyps besichtigt hat, ist er besser geeignet für den nächsten Auftrag des Objekttyps, als ein Mitarbeiter mit wenig Erfahrung im Bezug auf diesen Typ. Selbiges gilt für andere Eigenschaften, wie Auftragsart (Wertindikation, Außenbesichtigung, Innenbesichtigung, ...) und andere Eigenschaften, auf welche hier nicht genauer eingegangen werden soll.

Bei der **Auftragsverteilung** müssen also verschiedene Faktoren berücksichtigt werden. Zum Einen Kriterien, die einen Mitarbeiter von der Bearbeitung des Auftrags ausschließen, zum Anderen Kriterien, welche die Attraktivität des Auftrags für einen Mitarbeiter beeinflussen. Einer der wichtigsten Faktoren ist die Distanz zwischen dem Mitarbeiter und dem Auftrag, weshalb die Mitarbeiter und die Aufträge in etwa gleich verteilt im Bundesgebiet auftreten.

Weitere Faktoren sind die Kosten für die Mitarbeiter und die Verspätungskosten für verfristete Aufträge. Ziel ist es, die Aufträge möglichst optimal auf die Mitarbeiter zu verteilen, sodass die Kosten minimal sind.

2. Grundlagen

Es handelt sich um ein multikriterielles Optimierungsproblem. Das Problem ist ein NP hartes Problem, wie beim Job-Shop-Problem [13]. Bei einem Problem dieser Art, mit nur 10 Aufträgen und 10 Bearbeitern, würde sich bereits ein möglicher Suchraum von
$(n!)^m = 3.9594087e+65$
= 39594087000000000000000000000000000000000000000000000000000000000000000000000000
möglichen Lösungs-Variationen ergeben [6].

2.1. Ameisenalgorithmen

Bei dem Verfahren wird die Kommunikation von Ameisen mittels Pheromonen zum Ermitteln der kürzesten Strecke zur Nahrungsquelle imitiert [26]. Die Grundlage bilden Beobachtungen blinder Tiere, wie Ameisen, und der Frage, wie diese einen Pfad von ihrer Kolonie zur Futterquelle und zurück bilden. Es zeigte sich, dass Ameisen Pheromone auf ihren Laufwegen verteilen und sich an diesen orientieren. Eine Ameise würde prinzipiell eine zufällige Richtung einschlagen. Wenn sie einen Pheromon-Pfad findet, entscheidet sie sich zu einer hohen Wahrscheinlichkeit, diesem zu folgen und verteilt dabei selbst Pheromone auf diesen Wegen. Je mehr Ameisen einen Pfad laufen, desto mehr Pheromone werden auf diesem verteilt und desto attraktiver wird dieser Pfad für nachfolgende Ameisen.

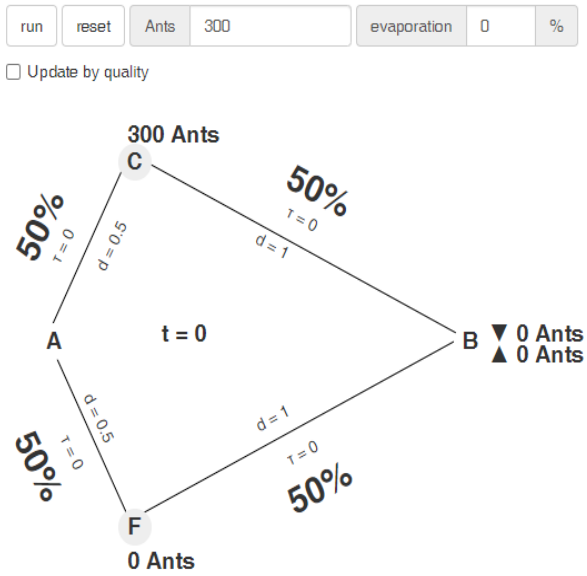

Abbildung 1: Beispiel-Pfad zum Zeitpunkt t = 0

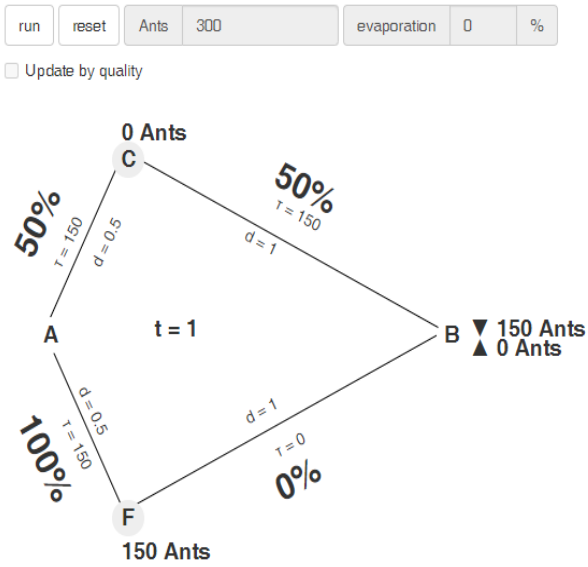

Abbildung 2: Beispiel-Pfad zum Zeitpunkt t = 1

Als Beispiel sei der Pfad aus Abbildung 1 gegeben. Die Ameisen laufen von der Kolonie (C) zum Futter F und wieder zurück. Da zum Zeitpunkt t = 0 noch kein Pheromon verteilt ist, wählen die ersten Ameisen gleichmäßig zufällig zu 50% den Pfad CAF und zu 50% den Pfad CBF. Der Pfad CBF ist doppelt so lang wie der Pfad CAF. Da die Ameisen alle die selbe Geschwindigkeit haben, benötigen die Ameisen auf Pfad CBF doppelt so lang, wie die Ameisen auf Pfad CAF. Die Ameisen bewegen sich mit der Geschwindigkeit von einer Entfernungseinheit d pro einer Zeiteinheit t. Zum Zeitpunkt t = 1 (Abbildung 2) kommen 50% der Ameisen bereits über den kurzen Pfad beim Futter an und die anderen 50% erreichen den Punkt B. Als Pheromon wird vereinfacht eine Einheit je Ameise gezählt. Die 150 Ameisen am Futter laufen nun geschlossen über A zurück zu C, da auf dem Pfad FB noch kein Pheromon liegt, während die 150 Ameisen von Punkt B weiter zum Futter laufen (Abbildung 3).

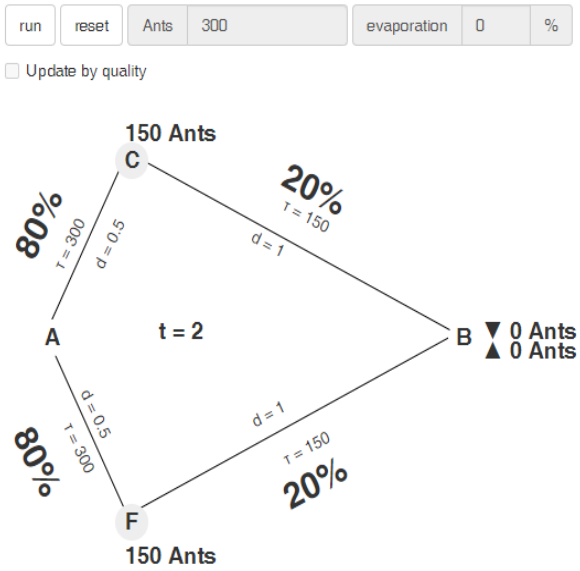

Abbildung 3: Beispiel-Pfad zum Zeitpunkt t = 2

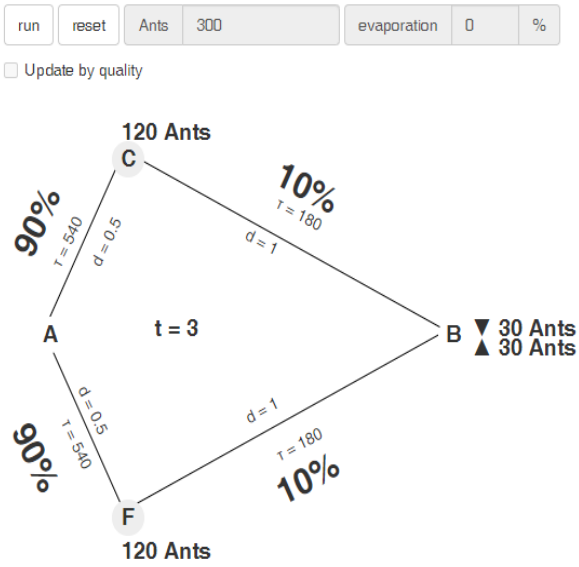

Abbildung 4: Beispiel-Pfad zum Zeitpunkt t = 3

Im nächsten Zeit-Schritt (Abbildung 4) betrachten die 150 Ameisen, die von der Kolonie C zum Futter F wollen, das Pheromon auf den Routen. Zu B sind bis jetzt 150 Ameisen gelaufen, zu A sind bereits 150 Ameisen hingelaufen und 150 Ameisen zurückgekommen, also ist das Pheromon zu A höher wie zu B. Demnach laufen nun weniger Ameisen (30 Ameisen) zu B und mehr Ameisen entscheiden sich für den Weg über A (120). Äquivalent verhält es sich bei den Ameisen, welche vom Futter zurück zum Nest wollen.
Wenn man dieses einfache Beispiel nun fortführt, wird man sehen, dass sich nach nur wenigen Iterationen fast alle Ameisen für den kürzeren Pfad CAF entscheiden.

2.2. ACO

Aus dem Modell der realen Welt kann nun ein ähnlich agierendes Optimierungstool entwickelt werden [24]. Im Unterschied zur realen Welt haben die digitalen Ameisen nun ein Gedächtnis, also einen Speicher. Sie merken sich ihre gelaufene Route und können so ihr Pheromon anhand der Qualität ihres Ergebnisses positionieren. Desweiteren müssen die Ameisen nicht komplett blind sein und können für Ihre Entscheidung eine Heuristik benutzen, wie die Distanz.

2.2.1. Pheromon-Update

Im realen Beispiel verteilt eine Ameise ihr Pheromon gleichmäßig auf der von ihr zurückgelegten Strecke. In der Demonstration war dies eine Pheromon-Einheit je Zeit-Einheit. Adaptiert auf den Ameisenalgorithmus muss dies nun nicht mehr blind geschehen. Jede Ameise merkt sich ihre zurückgelegte Strecke. Anschließend verteilt sie ihr Pheromon anhand der Qualität ihres Ergebnisses. Dies kann zum Beispiel geschehen, indem auf jedem von der Ameise benutzen Streckenabschnitt der Pheromon-Wert nicht um 1 erhöht wird, sondern um 1/d [17], der von der Ameise zurückgelegten Distanz. Dadurch wird weniger Pheromon auf langen Strecken verteilt.

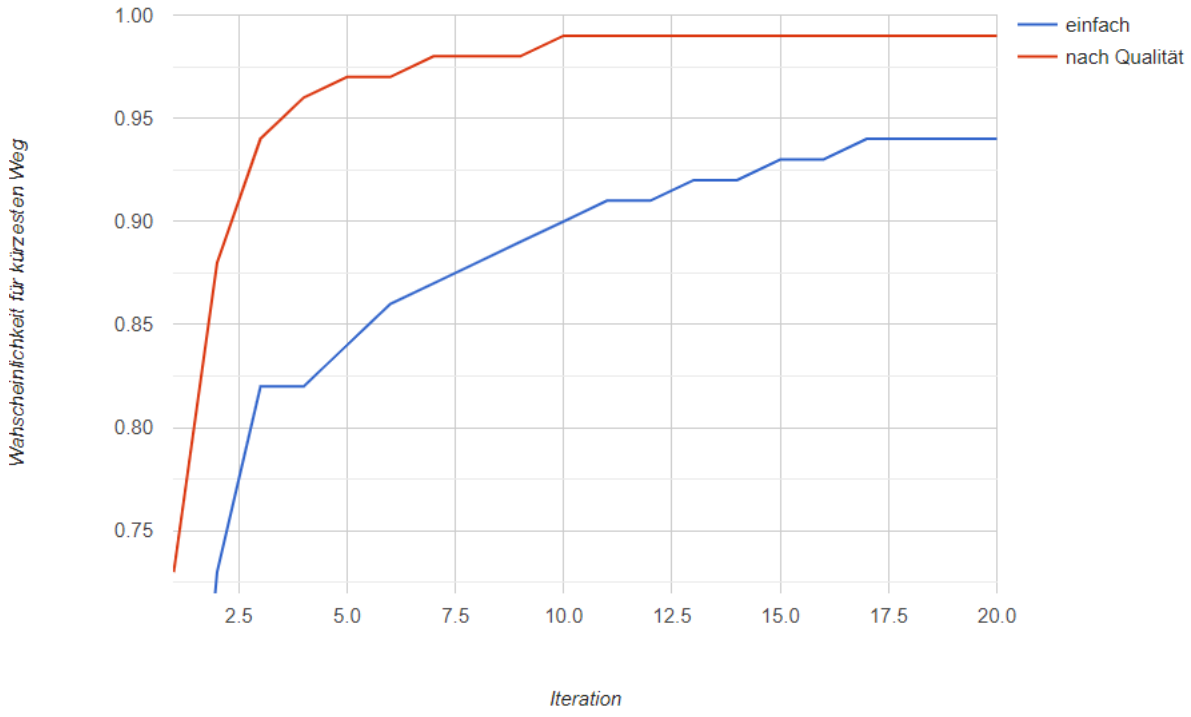

Abbildung 5: Update: einfache Pheromon-Verteilung und Verteilung d_{best}/d (Wertetabelle siehe Anhang 1)

Abbildung 5 zeigt auf 20 Iterationen hin (entspricht 20 Zeit-Einheiten im Beispiel aus Abschnitt 2.1) die Entwicklung der Wahrscheinlichkeit, dass sich eine Ameise, welche an der Kolonie startet, für den kürzesten Weg entscheidet. Die eine Linie entspricht der einfachen Pheromon-Verteilung, bei der anderen Linie wurde nach Qualität des Ergebnisses, d_{best}/d

geupdated [24]. d_{best} ist die bis Dato kürzeste gefundene Strecke und d die von der Ameise zurückgelegte Distanz. Je größer die Distanz ist, desto weniger Pheromon wird von der Ameise verteilt. Bei Update nach Qualität entscheiden sich mehr Ameisen eher für die kürzeste Route.

2.2.2. Verdunstung

Verdunstung der Pheromone auf den Pfaden bewirkt, dass der Einfluss unattraktiver Strecken nach und nach sinkt, aber auch, dass auf Strecken mit viel Pheromon, vieles davon verdunstet und so eine leichte Glättung eintritt, was die Exploration des Suchraumes begünstigt.

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} \text{ [16]} \quad (1)$$

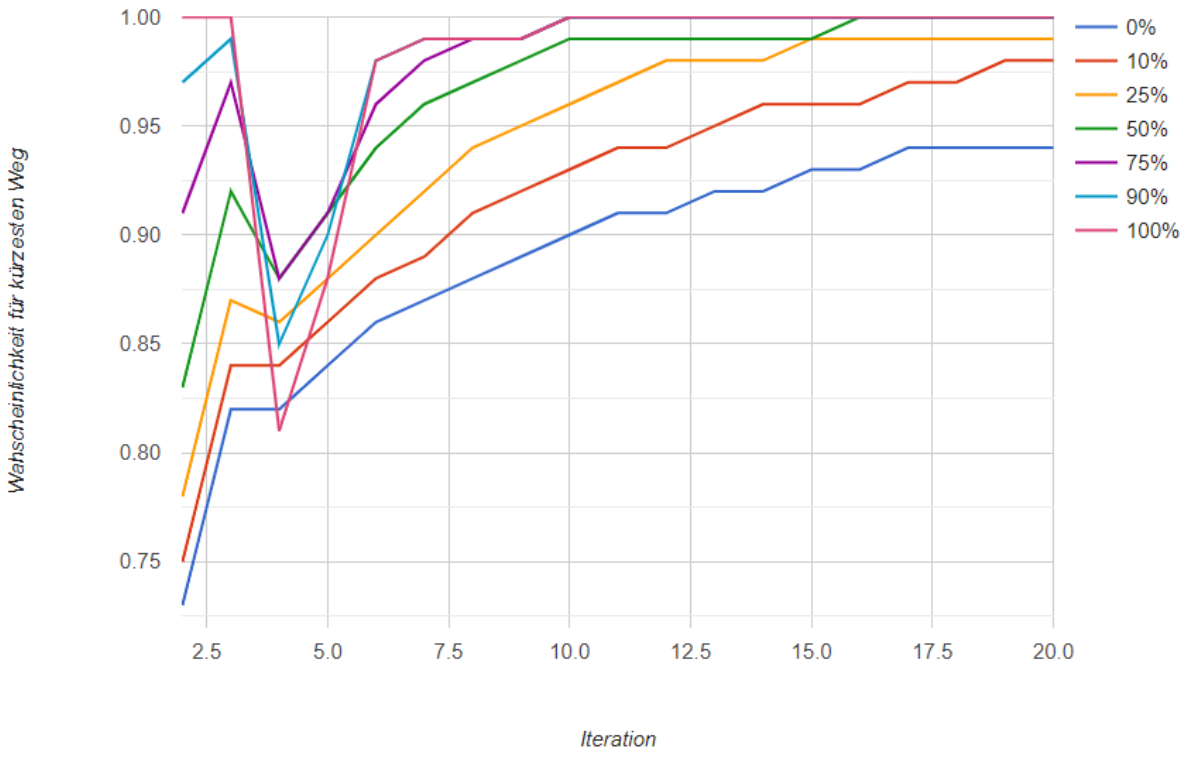

Abbildung 6: Einfluss von Verdunstung (Wertetabelle siehe Anhang 1)

In jeder Iteration wird jeder Pheromon-Wert um einen definieren Anteil reduziert. Abbildung 6 zeigt dies für ρ = {0, 0.1, 0.25, 0.5, 0.75, 0,9, 1}. In der Ausgangssituation ist die Wahrscheinlichkeit für die kürzere Strecke 50%, da noch kein Pheromon auf den Strecken

liegt. Bei höheren Verdunstungs-Werten legen sich die Ameisen früher auf eine Route fest, was je nach Problem gewünscht ist oder sich ungünstig auswirken kann. In der 4. Iteration ist ein Abfallen der Wahrscheinlichkeit zu beobachten. Dies liegt daran, dass die Ameisen, welche sich an der Futterquelle für den längeren Rückweg entschieden haben, nun zum Zeitpunkt t = 4 zur Kolonie C zurückkehren und dabei ihr Pheromon auf dem Rückweg verteilen. Bei t = 0 sind 50% der Ameisen den langen Weg über B gegangen, bei t = 2 (Abbildung 3) entscheiden sich 20 % dieser Ameisen für den langen Rückweg. Bei t=4 sind diese zurück am Nest und haben ihr Pheromon auf dem Wegabschnitt CB verteilt, wodurch dieser Pfad kurzfristig wieder etwas mehr Bedeutung gewinnt.

2.2.3 Heuristik

Im realen Beispiel war die Ameise blind. Sie entschied sich zufällig, wenn kein Pheromon vorhanden war oder bezog Pheromon-Werte mit in ihre Entscheidung ein, wenn bereits welches vorhanden war. Bei der Adaption des Verhaltens im Algorithmus muss die Ameise nicht blind sein. Sie kann außer dem Pheromon-Wert auch andere Faktoren mit in Ihre Entscheidung einbeziehen. Zum Beispiel die Heuristik $\eta = 1/d$ [4] [5], welche bewirkt, dass kürzere Streckenabschnitte bevorzugt werden.

Eine klassische Formel für die Entscheidung einer Ameise wäre [5]:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{j \in \text{mögliche Knoten}} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}} \qquad (2)$$

- p_{ij} ist die Übergangswahrscheinlichkeit vom aktuellen Standort i der Ameise zu Punkt j.
- τ_{ij} ist der Pheromon-Wert auf der Strecke von i nach j
- η_{ij} entspricht dem Heuristikwert, wie 1/d (Länge der Strecke i zu j)
- α und β sind Faktoren zur Beeinflussung des Einflusses von Pheromon und Heuristik in der Entscheidung der Ameise

Sie bestimmt von ihrem Standort die Wahrscheinlichkeiten aller möglichen Pfade und entscheidet sich dann entsprechend dieser Wahrscheinlichkeiten.

2.3. ähnliche Probleme und Arbeiten

Das Job-Shop-Scheduling-Problem (kurz JSSP) behandelt die Verteilung von (Produktions-)Aufträgen auf verschiedene Maschinen. Dabei besteht ein Auftrag aus mehreren Teilschritten, welche nacheinander auf verschiedenen Maschinen abgearbeitet werden. Es gilt meist die Produktions-Endzeit MIN(C_{max}) zu minimieren (C wird in dieser Arbeit als Variable für die Kosten benutzt und nur in diesem Kapitel für Zeit verwendet). Teschemacher et al. [5] beschrieben dabei, ähnlich dem Problem dieser Arbeit, weiche und harte Kriterien. Harte Kriterien müssen voll erfüllt sein, da sonst die komplette Lösung invalide wäre. Als Beispiel sei hier die korrekte Reihenfolge der einzelnen Arbeitsschritte eines Auftrags genannt. Weiche Kriterien beeinflussen lediglich die Qualität des Ergebnisses. Chaouch et al. [4] versuchten die Lücken (inaktive Zeit der Maschinen) zu füllen, indem sie die zuletzt fertiggestellten Aufträge auf diese verteilen. Nazif et al. [1] benutzten bei der Pheromon-Matrix minimale und maximale Werte, um den Pheromon-Wert zu begrenzen. Desweiteren wurde versucht, das Ergebnis mit einer lokaler Suche zu verbessern, indem in der Lösung zufällig Aufträge bzw. Auftragsschritte miteinander vertauscht wurden, und dann geprüft wurde, ob dies das Ergebnis verbesserte. Rondon et al. [6] versuchten neben der Minimierung der Produktionszeit MIN(C_{max}) auch die Summe aller Endzeitpunkte der Produktion der einzelnen Aufträge $\sum C_i$ zu minimieren und untersuchten unter Anderem verschiedene Heuristiken für das Problem. Udomsakdigool et al. [11] nutzen mehrere Kolonien, welche jeweils eine eigene Pheromon-Matrix benutzten und eine globale Pheromon-Matrix, welche nach jeder Iteration von der besten Ameise je Kolonie geupdated werden durfte.

Unter der Gruppe der Multi-Objective Optimization Problems (kurz Moop) verstehen sich Optimierungsprobleme mit mehreren, sich teils entgegenstehenden Optimierungs-Kriterien. Alaya et al. [12] verglichen in ihrer Arbeit verschiedene Ansätze mit mehreren Kolonien. Als geeignetste Lösung schnitt jedoch eine Lösung mit nur eine Kolonie ab, welche je Kriterium eine eigene Pheromon-Matrix nutze. Dabei wählt jede Ameise vor ihrem Lauf zufällig ein Kriterium und die zugehörige Matrix aus. Als Heuristik wurde die Summe der Heuristiken aller Kriterien verwendet. Die beste Ameise je Kriterium durfte dann jeweils die dazugehörige Matrix updaten. Das Update geschieht in Relation zur bislang besten gefunden Lösung

(Formel siehe Kapitel 4.5. Parameter). Ling et al. [9] untersuchten ebenfalls dieses Problem mit mehreren Kolonien und einem Update entlang der dominanten Front an Lösungen.

Das Parallel Machine Scheduling Problem ist ähnlich dem JSSP, jedoch werden die Aufträge im Ganzen an eine Maschine zugewiesen. Jeder Auftrag hat dabei einen definieren Fertigstellungstermin. Es gilt, die Verspätung der Aufträge zu verhindern bzw. zu minimieren. Neto et al. [2] untersuchten dies im Hinblick darauf, ob eine Fabrik einen Auftrag outsourcen soll oder nicht. Dazu wird zuerst ein Auftrag mithilfe einer Pheromon-Matrix und ohne Heuristik gewählt. Im nächsten Schritt wird die Maschine gewählt, welche den Auftrag bearbeiten soll, falls dieser nicht ausgelagert wird. Es wird eine Pheromon-Matrix genutzt und als Heuristik wird die Maschine bevorzugt, welche am wenigsten gelaufen ist. Im dritten Schritt wird anhand einer Formel mit gewichtetem Zufall bestimmt, ob der Auftrag ausgelagert werden soll oder nicht. Berrichi et al. [14] untersuchten dies unter Einbeziehung möglicher Ausfall-/Reparaturzeiten von Maschinen. Liang at al [17] wählten zur Lösung des Problems erst die Maschine und dann den Auftrag. Anschließend wählt die Ameise erneut eine andere Maschine und prüft, ob dies das Ergebnis verbessert. Zum Schluss wird eine lokale Suche durchgeführt, bei der zum Einen Aufträge auf der selben Maschine vertauscht werden und zum Anderen Aufträge zwischen Maschinen getauscht werden.

3. Formales Problem

3.1. Beschreibung

Einfach beschrieben geht es darum, n Aufträge auf m Bearbeiter zu verteilen. Dabei müssen obligatorische Bedingungen erfüllt werden. Es gilt, die Summe der Kosten zu minimieren. Im Folgenden werden die verschiedenen Modelle wie „Auftrag“ und „Bearbeiter“ genauer definiert. Kosten werden dabei immer als Ganz-Zahl (Datentyp „int“ oder „integer“) definiert, aber in der Benutzung oft als Kommazahl (Datentyp „float“) gehandhabt, um zu große Rundungsdifferenzen bei Berechnungen zu vermeiden. Die Werte der Kosten sind fiktiv, aber so gewählt, dass sie im Verhältnis zueinander das Problem repräsentieren können. Als Zeitangaben werden ebenfalls Ganz-Zahlen genutzt. Diese repräsentieren die Stunden in einem vereinfachten Kalender mit dem Startpunkt 0.

3.2 Variablen

3.2.1. Aufträge

Als Gegeben wird ein Set von Aufträgen $J = \{J1, J2, J3, \ldots, J_n\}$ gesehen sowie folgende Eigenschaften der Aufträge:

- **PJ_j**: Die geografische Position des Auftrags bzw. der zu besichtigenden Immobilie.
- **DUE_j**: Das Abgabedatum, bis zu welchem Zeitpunkt die Besichtigung spätestes erfolgt sein sollte.
- **PR_j**: Die Dauer, die ein Bearbeiter vor Ort für die Besichtigung benötigt.
- **TC_j**: Sollte ein Bearbeiter erst nach dem vereinbarten Abgabetermin DUE_j mit der Besichtigung fertig sein, fallen Verspätungskosten an. Um diese berechnen zu können, gibt es zu jedem Auftrag eine festgelegte Höhe der Verspätungskosten je Zeiteinheit, um welche der Auftrag jeweils in Verzug ist.

- **$PROB_j$**: Wie einleitend erwähnt, gibt es obligatorische Eigenschaften, welche ein Mitarbeiter haben muss, um einen Auftrag bearbeiten zu können. $PROB_j$ ist die Liste der obligatorischen Eigenschaften des Auftrags J_j.
- **$PROP_j$**: Liste der optionalen Eigenschaften, welche ein Bearbeiter haben kann, um Auftrag J_j für ihn attraktiver zu machen. $PROB_j$ und $PROP_j$ sind einfache Wort-Listen.

3.2.2. Bearbeiter

Ein Set von Bearbeitern M = {M1, M2, M3, …, M_m} wird als gegeben angenommen mit den Eigenschaften:

- **PM_i**: Die geografische Position des Bearbeiters. Ob dies sein privater Wohnsitz oder eine Firmenniederlassung ist, spielt hierbei keine Rolle.
- **CPH_i**: Der Stundenlohn des Bearbeiters wird für die Kostenberechnung benötigt und kann auch benutzt werden, um externe Mitarbeiter abzubilden, welche sich von Angestellten im Wesentlichen nur durch einen höheren Stundenlohn unterscheiden.
- **PRO_i**: Eigenschaften, wie Zertifizierungen oder bevorzugte Objektarten werden als einfache Worte („Strings“) in der Liste zu jedem Mitarbeiter geführt.

3.2.3. Konstanten

- **HPK**: Eine Konstante. Diese gibt an, wie viel Streckeneinheiten ein Bearbeiter in einer Zeiteinheit zurücklegt.

3.3. Funktionen

getDistance(i,j)

Gibt die Distanz zwischen der Position des Bearbeiters PM_i und der Position des Auftrages PJ_j zurück. Prinzipiell könnte hier ein externer Service verwendet werden, welcher zum Beispiel unter Anderem auch die Verkehrslage berücksichtigen könnte. Zur Vereinfachung wird die Entfernung anhand der gedachten Luftlinie berechnet. Die Formel der Berechnungsmethode

ist äquivalent in der Berechnung der Funktion GeoCoordinate.GetDistanceTo des System.Device Paketes des Microsoft .Net Frameworks.

```
public float getDistance(PointModel startPoint, PointModel endPoint)
{
    var d1 = startPoint.Lat * (Math.PI / 180.0);
    var num1 = startPoint.Lng * (Math.PI / 180.0);
    var d2 = endPoint.Lat * (Math.PI / 180.0);
    var num2 = endPoint.Lng * (Math.PI / 180.0) - num1;
    var d3 = Math.Pow(Math.Sin((d2 - d1) / 2.0), 2.0)
             + Math.Cos(d1) * Math.Cos(d2) * Math.Pow(Math.Sin(num2 / 2.0), 2.0);

    return Constants.EarthRadius
            * (2.0 * Math.Atan2(Math.Sqrt(d3), Math.Sqrt(1.0 - d3)));
}
```

getTravilingTime

Gibt die Fahrtzeit eines Bearbeiters zu einem Auftrag zurück. Für die benötigte Zeit wird die Entfernung mit einem definierten Faktor multipliziert. Hier wird vorerst 0,02 Stunden je Kilometer angenommen. Dies entspricht einer Fahrtzeit von einer Stunde für 50 Kilometer. Hier könnte aber auch zukünftig ein externer Dienst eingesetzt werden, welcher beispielsweise Geschwindigkeitsbegrenzungen mit einbezieht.

$$\text{getTravilingTime}(i,j) \rightarrow \text{getDistance}(i,j) * HPK \tag{3}$$

GetTardinessCosts: Gibt die Verspätungskosten zu einem Auftrag zurück, wenn dieser zum Zeitpunkt t fertiggestellt wird. Je Auftrag ist ein Datum für die letztmögliche Abgabe definiert (DUE_j). Wenn ein Auftrag erst nach dieser Frist fertiggestellt wird, fallen Verspätungskosten an, welche je Auftrag und je Stunde definiert sind (TC_j). Die Funktion „GetTardinessCosts“ berechnet diese Kosten für einen Auftrag anhand der angegeben Zeit des Startes der Bearbeitung.

$$\text{GetTardinessCosts}(i, t) \rightarrow MAX(TC_j * (t - DUE_j + PR_j),0) \tag{4}$$

getStaffOrderCosts: Des Weiteren fallen Personalkosten während der Bearbeitung des Auftrags vor Ort an. „getStaffOrderCosts“ Berechnet diese Anhand der Arbeitszeit vor Ort (PR_j) multipliziert mit dem Stundenlohn des Bearbeiters (CPH_i).

$$\text{getStaffOrderCosts}(i, j) \rightarrow CPH_i * PR_j \quad (5)$$

getStaffTravelCosts: Die Lohnkosten während der Fahrtzeit berechnet die Funktion „getStaffTravelCosts“ anhand des Stundenlohns des zugewiesenen Bearbeiters multipliziert mit der beim Routing bestimmten Fahrtzeit (Funktion „getTravilingTime“).

$$\text{getStaffTravelCosts}(t, i) \rightarrow CPH_i * t \quad (6)$$

getStaffOrderPreference: Neben den obligatorischen Qualifikationen, welche ein Bearbeiter haben muss, um einen Auftrag annehmen zu können, hat ein Bearbeiter auch weitere Qualifikationen und Eigenschaften. Wie eingangs beschrieben, können dies zum Beispiel Vorlieben für eine bestimmte Objektart oder Auftragsart sein. Aufträge haben ebenfalls neben den obligatorischen Eigenschaften auch optionale Eigenschaften, welche ein Bearbeiter nicht erfüllen muss. Diese Eigenschaften sind als einfache Worte in der Liste der Eigenschaften PRO_i angegeben. Es wird im Modell angenommen, dass ein Mitarbeiter Aufträge bevorzugt, deren Eigenschaften er umfänglich abdeckt und Unzufriedenheit entsteht, wenn er Aufträge bearbeitet, deren Eigenschaften er nur wenig abdeckt. Wenn als Beispiel ein Arbeiter nur 2 von 10 Eigenschaften eines Auftrags abdeckt, wird ein Wert von 0.2 für die Zufriedenheit angenommen. Hat er die selben 10 Eigenschaften, die der Auftrag hat, dann wird mit 1 eine volle Zufriedenheit angenommen. Sollte ein Auftrag überhaupt keine Anforderungen stellen, also keine obligatorischen und optionalen Eigenschaften haben, wird ein Zufriedenheits-Wert von 0.5 angenommen.

$$\text{getStaffOrderPreference}(i, j) \rightarrow (|PROB_j \cap PRO_i| + |PROP_j \cap PRO_i|) / (|PROB_j| + |PROP_j|), \text{ wenn } |PROB_j| + |PROP_j| > 0$$
$$\text{getStaffOrderPreference}(i, j) \rightarrow 0.5, \text{ wenn } |PROB_j| + |PROP_j| = 0 \quad (7)$$

3.4. Bedingungen

Jeder Auftrag kann obligatorische Qualifikationen des Bearbeiters voraussetzen. Wenn ein Bearbeiter eine dieser Eigenschaften nicht erfüllt, ist er für den Auftrag ungeeignet. Die Funktion „canStaffProcessOrder“ prüft dies im Bezug auf einen Arbeiter und einen spezifischen Auftrag.

$$\text{canStaffProcessOrder}(i,j) \rightarrow \text{PROB}_j \subseteq \text{PRO}_i \quad (8)$$

3.5. Ziel der Optimierung

Das Ergebnis der Ameisen besteht aus einer Liste, welche |M| lang ist. Jeder Index der Liste repräsentiert den Bearbeiter, der den selben Index in der Liste der Bearbeiter der Seed-Daten hat. Jedem Bearbeiter wird eine sortierte Liste an Aufträgen zuordnet.
$R_i = \{J_x, J_y, ...\}$

getTotalCosts: Die Funktion „getTotalCosts“ berechnet für jeden Mitarbeiter dessen Tour, verschiedene Kosten und Summiert diese auf. Die Kosten für das Fahrzeug (Benzin, Verschleiß, …) werden dabei mit Länge der Strecke * 0,1 berechnet. Der Unzufriedenheitswert je Mitarbeiter und Tour berechnet sich aus 1 - getStaffOrderPreference(i, j). Er geht multipliziert mit einem Faktor von 100 Geldeinheiten in die Berechnung der Gesamtkosten ein. Die Verspätungskosten je Tour R_i werden durch folgende Funktionen berechnet:

```
public float getTourTardinessCosts(List<int>[] R, int i)
{
    var time = 0;
    var costs = 0;
    foreach (var j in R) {
        costs += GetTardinessCosts(i, time)
        time += getTravilingTime(i,j) + PRj;
    }

    return costs;
}
```

So ergibt sich als Formel für die Gesamtkosten einer Lösung:

`getTotalCosts(R)=`

$$\sum_{i=0}^{|M|} getTour\ TardinessCosts\,(R_i, i) + \sum_{j \in R_i} \begin{array}{l} getDistance(i,j) * 0.1 \\ + getStaffOrderCosts(i,j) \\ + (1 - getStaffOrderPreference(i,j)) * 100 \\ + getStaffTravelCosts(i, getTravilingTime(i,j)) \end{array} \quad (9)$$

Das Formale Problem dieser Arbeit ist es, eine gültige Lösung mit minimalen Kosten zu finden:

MIN(getTotalCosts(R))

4. Implementierung

4.3. Algorithmen

4.3.1. ALG1: Vergleichsalgorithmus

Im ersten Schritt wurde ein Zufallsalgorithmus implementiert, welcher aus einem Set an Aufträgen per Zufall nach und nach Aufträge einzeln auf die Bearbeiter aufteilte. Da der Suchraum aber enorm groß ist, brachte dieser Algorithmus entsprechend unbefriedigende Ergebnisse, auch, wenn man den Zufallsalgorithmus hundertfach mehr Lösungen hat bauen lassen, als durch die Ameisenalgorithmen. Deshalb wurde ein Greedy-Algorithmus implementiert. Dieser verfährt ähnlich einem Ameisenalgorithmus, welcher ohne Pheromon, rein auf Heuristik arbeitet [13], also $\alpha = 0$. Der Algorithmus ordnet jedem Auftrag dem ihm lokal am naheliegendsten Bearbeiter zu und sortiert auch dessen Tour entsprechend so, dass er von seinem nächsten Auftrag den Auftrag anfährt, der diesem wiederum am nahesten liegt.

4.3.2. ALG2: Einfacher Ameisenalgorithmus

Sowohl Habibeh Nazif [1] als auch Huang & Yang [8] beschrieb in einem Paper über das Job-Shop-Scheduling Problem, wie ein möglicher Graph aufgespannt sein müsste, welchen die Ameisen ablaufen könnten, um das Problem zu lösen. Alle Ameisen Starten dabei von einem gedachten Start-Knoten 0 [13]. Die Knoten des Graphen $G_{i,j}$ repräsentieren dabei, dass der Auftrag j von dem Mitarbeiter i bearbeitet werden soll. Die Kanten zwischen den Knoten bestimmen die Reihenfolge der Wahl. Auf ihnen liegt das Pheromon. Die Pheromon Werte können als Gleitkommazahl in einem Wörterbuch gespeichert werden. Der Pheromon-Wert im Eintrag i_1j_1-i_2j_2 sagt etwas über die Wahrscheinlichkeit der Wahl der Kante aus, die von dem Knoten G_{i1j1} zu dem Knoten G_{i2j2} führt.
Die Ameise läuft den Graphen von G0 ausgesehen so lange ab, bis alle Aufträge ausgewählt wurden. Sie betrachtet für ihre Lösung von dem Knoten aus, auf dem sie sich befindet, alle

Kanten, die zu Knoten führen, deren Aufträge noch nicht bearbeitet sind. Anschließend folgen die üblichen Schritte im Bezug auf Verdunstung und Pheromon-Update.
In ersten Versuchen zeigte sich, dass ein Wörterbuch zur Speicherung der Pheromon-Werte sehr ineffizient im Bezug auf Speicherplatz und Performance ist. Deshalb wurde ein vierdimensionaler Array $D[i_1][j_1][i_2][j_2]$ genutzt, da dieser auch leichter zu initialisieren und zu updaten war.

4.3.3. ALG3: Minimierter einfacher Ameisenalgorithmus

ALG2 benötigte durch die vierdimensionale Pheromon-Matrix sehr hohe Ressourcen. Deshalb war es naheliegend, zu überlegen, diese auf drei Dimensionen zu reduzieren. Dabei kodieren die ersten beiden Ebenen weiterhin einen Knoten G_{ij} im Graphen, während die dritte Dimension (mit der Länge der Anzahl an Aufträgen) die Information über die Reihenfolge beinhaltet.
Die Ameisen werden dabei den selben gedachten Graphen ablaufen, lediglich die zugrundeliegende Pheromon-Matrix wird anders repräsentiert und somit anders gelesen und geschrieben.

4.3.4. ALG4: erst Reihenfolge, dann Zuordnung

Der vierte Algorithmus orientiert sich an dem Algorithmus von Roberto Fernandes Tavares Neto et al. [2] zur Lösung des Parallel Machine Scheduling Problem. Dabei wird zuerst die Reihenfolge der Aufträge bestimmt und hierfür eine eigene Pheromon-Matrix benutzt. Anschließend wird eine zweite Pheromon-Matrix benutzt, um die zuvor sortieren Jobs nun in dieser Reihenfolge auf die Mitarbeiter zu verteilen.

4.3.5. ALG5: erst Zuordnung, dann Reihenfolge je Maschine

Jean-Paul Arnaout et al. [3] untersuchten ebenfalls das Parallel Machine Scheduling Problem. Ihr Algorithmus ähnelt ALG4, allerdings wird erst die Zuordnung der Aufträge zu den Arbeitern bestimmt und anschließend die Reihenfolge, in der der jeweilige Mitarbeiter die

ihm zugeordneten Aufträge abarbeitet. Dabei wird auf der zweiten Matrix kodiert, ob Auftrag i1 nach Auftrag i2 abgearbeitet werden soll. Im Paper von Jean-Paul Arnaout et al. wird dies je Arbeiter getan. Somit würde sich hier eine dreidimensionale Matrix [i, j1, j2] zur Speicherung anbieten.

4.3.6. ALG6: erst Zuordnung, dann Reihenfolge global

In ALG5 wurde die Reihenfolge der Aufträge mit einer separaten Pheromon-Matrix je Arbeiter bestimmt. Dies bedeutet zum Einen einen höheren initialen Aufwand und mehr Speicherbedarf, da je Arbeiter eine j x j Matrix initialisiert werden muss. Bei 4 Byte je Pheromon-Wert, 100 Arbeitern und 1000 Aufträgen wären dies bereits ein Verbrauch von 400 MB Arbeitsspeicher, welcher quadratisch mit der Anzahl der Aufträge wächst. Des Weiteren wird pro Arbeiter eine größtenteils leere, bzw. unnütze Pheromon-Matrix geführt, da jeder Mitarbeiter nur einen entsprechenden Anteil an Aufträgen abarbeitet. Dies bedeutet auch, dass ein Auftrag, welcher dem Mitarbeiter im Verlauf immer mit einem minimalen Pheromon-Wert neu zugewiesen wird, wahrscheinlich am Schluss angereiht wird.
Der sechste Algorithmus, der untersucht werden soll, löst dies, in dem er nur eine globale j x j Pheromon-Matrix für Reihenfolge benutzt. Diese sagt global für den Lösungsraum aus, dass Auftrag j_1 vor Auftrag j_2 bearbeitet werden sollte.

4.4. Heuristiken

Zur Evaluation der Pfadwahl bezieht die Ameise bei jeder Entscheidung zum Pheromon-Wert zusätzlich einen Heutistik-Wert ein. Rondon et al. [6] haben sich bei der Befassung mit dem Job-Shop-Problem ebenfalls mit folgenden Heuristiken beschäftigt:

- Kürzeste Bearbeitungszeit zuerst: Von den ausstehenden Aufträgen wird der gewählt, der am schnellsten abgearbeitet ist.
- Frühestes Abgabedatum zuerst: Der ausstehende Auftrag, der das früheste Abgabedatum hat, wird gewählt. Dabei sollte die Bearbeitungszeit mit berücksichtigt werden. Holloway [7] nennt dies den spätesten Startzeitpunkt, welcher sich aus dem Abgabedatum minus der Bearbeitungszeit berechnet.

- In Reihenfolge der Erstellung: Ein Auftrag, der vor einem anderen Auftrag erstellt wird, wird auch vor diesem abgearbeitet.
- Gewichtete kürzeste Bearbeitungszeit zuerst: Außer der Bearbeitungszeit wird als zweiter Faktor eine Wichtung der Aufträge hinzugezogen. Diese Wichtung ist bei Randon et al. zu ein zu jedem Auftrag bereits vorgegebener Wert.
- Längste Bearbeitungszeit zuerst: Der noch ausstehende Auftrag mit der längsten Bearbeitungszeit wird bevorzugt.
- Kürzeste Einrichtungszeit zuerst: Der Auftrag, bei dem man die geringste Zeit benötigt, um die Maschine entsprechend umzustellen, wird gewählt. Im Falle des in dieser Arbeit behandelten Problems entspricht dies der Anfahrtszeit, bzw. Anfahrtsstrecke.

Des Weiteren sind Kombinationen aus diesen Varianten möglich.

Da die Bearbeitungszeit der Aufträge, bei dem spezifischen untersuchten Problem dieser Arbeit, im Mittel nicht stark variiert, werden Heuristiken, wie „Längste Bearbeitungszeit zuerst“, nicht weiter untersucht.

Bei der Betrachtung des einfacheren Problems, reduziert auf die Distanz, trifft man häufig die Heuristik 1/d an [4] [5]. Das Reziproke der Entfernung bevorzugt bzw. belohnt kleinere Distanzen. Dies entspricht der kürzesten Einrichtungszeit bei dem von Rondon et al. [6] untersuchten Problem.

Zur Normalisierung ist folgende Formel denkbar: 1 - (d / 1053), da bei den verwendeten Koordinaten die maximale Distanz 1053 km betragen kann.

Während bei 1/d der Wert exponentiell sinkt, wäre $1 – (d/d_{max})$ ein monoton sinkender Wert (Abbildung 7):

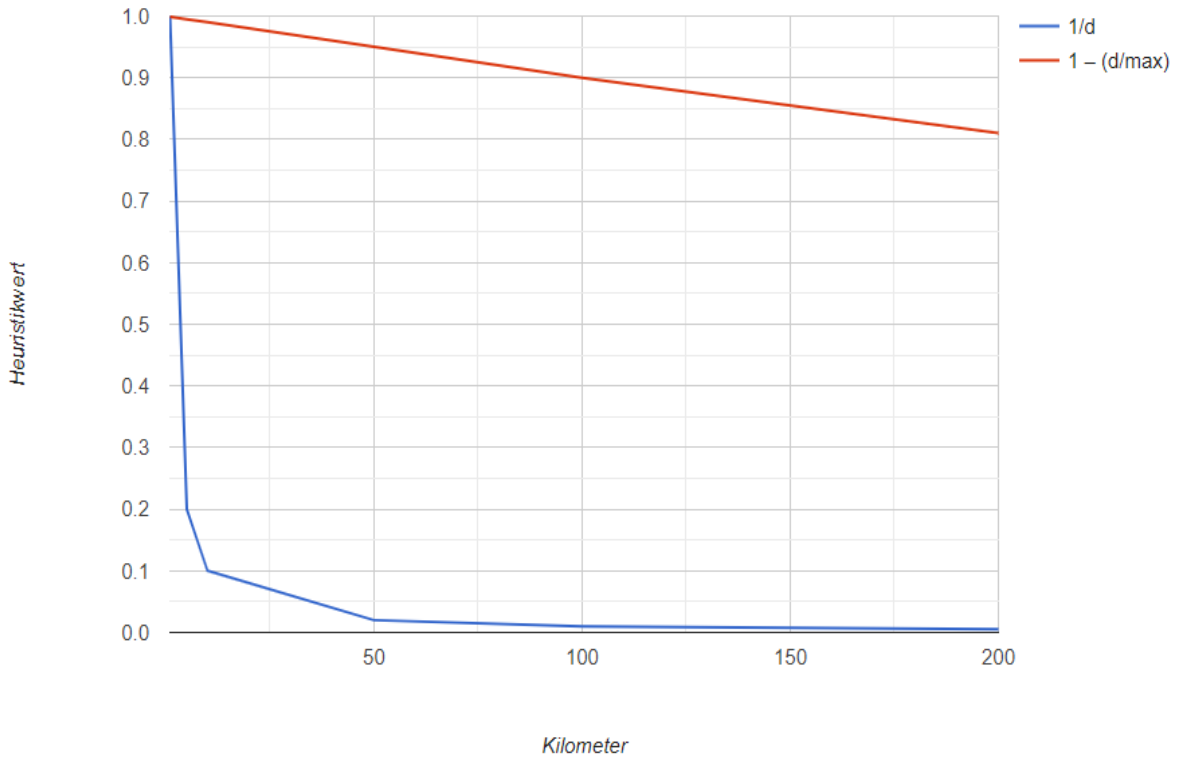

Abbildung 7: 1/d im Vergleich zu 1 – (d/d_{max}), Wertetabelle siehe Anhang 2

Da die zu berücksichtigenden Distanzen in der Regel aber klein sein werden (<100km), wird in dieser Arbeit 1/d als Heuristik für das Problem betrachtet, da die exponentielle Heuristik hier kürzere Distanzen stärker bevorzugt, als die monotone Variante.

4.5. Parameter

Anzahl der Ameisen (K): Hier gibt es den Ansatz, statisch eine Zahl zu bestimmen, wie zum Beispiel 10 [9] oder 30 [3]. Ein anderer Ansatz wäre, die Anzahl der benötigten Ameisen zu berechnen, wie Beispielsweise durch K = |J|/2 [13].
J ist hierbei das Set an Aufträgen und |J| die Anzahl der im Set befindlichen Aufträge.

Anzahl Iterationen (N_{Cmax}): Der Algorithmus benötigt eine Abbruchbedingung. Im Experiment könnte dies der Fall sein, wenn die Änderung des besten Ergebnisses zum vorherigen Ergebnis über mehrere Iterationen annähernd Konstant bleibt. Oft wird eine feste Anzahl an Iterationen verwendet und der Algorithmus beendet, wenn die maximale Anzahl

der Iterationen erreicht ist.

$N_C = N_{Cmax}$

Diese Werte liegen in vergleichbaren Algorithmen, wenn angegeben, um die 100 bis 1000 [12][13] Iterationen. Aber auch darunter, wie Beispielsweise nur 20 Iterationen [10].

Pheromon Verdunstung (ρ): Die Pheromon-Werte werden nach jeder Iteration reduziert, um den Einfluss vergangener Iterationen nach und nach zu minimieren.

$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$ [16] (10)

In der Literatur werden für den Faktor ρ ($0 < \rho < 1$) unterschiedliche Werte von 0.01 [2], 0.1, 0.7 [4] bis 0.95 [9] angegeben. Es ist auch ein reziproker Zusammenhang zwischen der Anzahl der Iteration N_{Cmax} und der Verdunstung ρ erkennbar, sodass bei wenigen Iterationen eher mit einer hohen Verdunstung gerechnet wird.

Pheromon Update $\Delta\tau_{ij}$: Am Ende jeder Iteration erfolgt eine Erhöhung der Pheromon-Werter auf den Kanten des Graphen.

$\tau_{ij} = \tau_{ij} + \Delta\tau_{ij}$ (11)

Oft erfolgt das Pheromon-Update in Kombination mit der Verdunstung

$(1-\rho) * \tau_{ij} + \rho * \Delta\tau_{ij}$ [3][4][17] (12)

Dies bewirkt, das der Pheromon-Wert τ_{ij} kleiner 1 bleibt, wenn $0 <= \Delta\tau_{ij} <= 1$. Es gibt verschiedenste Methoden, den Wert des Updates $\Delta\tau_{ij}$ zu berechnen.

Liang et al., [17] addieren den reziproken Objektwert der besten Lösung zum Pheromon-Wert, wenn die jeweilige Kante Teil der besten Lösung ist.

$$\Delta\tau_{ij} = \frac{1}{L_{best}} \quad wenn (i,j) \in bester\ Lösung \qquad (13)$$
$$\Delta\tau_{ij} = 0 \quad ansonsten$$

Berrichi et al. [14] berechnen das Update über eine gewählte Konstante Q geteilt durch den Rank der Ameise. Dieser Rank richtet sich allerdings nach einer Ordnung, bezogen auf eine Nicht-dominante Front [18].

$$\Delta\tau_{ij}^{f} = \frac{Q}{Rank_f} \quad wenn (i,j) \in bester\ Lösung\ der\ Armeise\ mit\ Rang\ f \qquad (14)$$
$$\Delta\tau_{ij}^{f} = 0 \quad ansonsten$$

Alaya et al. [12] erhöhen einen Pheromon-Wert τ_{ij}, wenn dieser Teil der Lösung ist, wie folgt:

$$\Delta\tau_{ij} = \frac{1}{1+\mathrm{f}(S)-\mathrm{f}(S_{best})} \quad wenn(i,j) \in bester\ Lösung$$
$$\Delta\tau_{ij} = 0 \quad ansonsten \tag{15}$$

Dies ähnelt der Lösung von Liang et al. [17], normalisiert wird dies aber mittels Abzug der Kosten der aktuell besten Lösung. Des Weiteren bietet diese Lösung die Möglichkeit, mehrere Ameisen Updaten zu lassen, da diese entsprechend der Güte ihrer Lösung ihr Update platzieren. Bessere Lösungen platzieren mehr Pheromon.

Udomsakdigool et al. [11] erhöhen um einen festen Wert, wenn die Kante Teil der besten Lösung ist.

$$\Delta\tau_{ij} = 1 \quad wenn(i,j) \in bester\ Lösung$$
$$\Delta\tau_{ij} = 0 \quad ansonsten \tag{16}$$

In dieser Arbeit wird das Update mit einem festen Wert untersucht, da nur die beste Ameise updaten soll.

Initialer Pheromon-Wert τ_0: Initial werden alle Werte der Pheromon-Matrix mit einen initialen Wert belegt. Dieser kann zum Beispiel 1 [9] sein. Erste Tests haben im Vergleich zum Greedy Algorithmus gezeigt, dass das Ergebnis sehr viel schlechter ist. Dies zeigte, dass die Parameter falsch gewählt waren. So schien der initiale Depot-Wert im Vergleich zum Update zu hoch zu sein, sodass die Ameisen eher zufällig wählten und einen zu geringen Lern-Effekt zeigen. Dies kann aber auch durch eine höhere Verdunstung kompensiert werden, welche den initialen Pheromon-Wert schneller sinken lässt.

Einfluss von Pheromon(α) und Heuristik (β):

Allgemein wird die Wahrscheinlichkeit, dass eine Ameise einen Pfad wählt, wie folgt berechnet [5]:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{j \in mögliche\ Knoten} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}} \tag{17}$$

Der Einfluss der Heuristik β kann dabei abhängig vom Einfluss des Pheromon α ($0 < \alpha < 1$) berechnet werden:

$$\beta = (1 - \alpha) \text{ [4][13]} \tag{18}$$

Die Werte für α können ein höheren Einfluss des Pheromons (α = 0.8 [4]) oder einen geringeren (α = 0.2 [13]) bewirken. Allerdings kann α und β auch unabhängig betrachtet werden und auch Werte weit größer 1 ausweisen [12][9].

Übergangsfunktion-Weiche Q: Die ersten Versuche machten deutlich, dass die Ameisen im Vergleich zur reinen Heuristik bzw. dem Greedy-Algorithmus schlechtere Ergebnisse (ca. 200% schlechter) lieferten. Dies wird darauf zurückzuführen sein, dass der Suchraum sehr groß ist und bei geringer Anzahl von Ameisen diese diesen nur sehr ungenügend explorieren können. Deshalb wäre es von Vorteil, über einen weiteren Parameter steuern zu können, ob die oben beschriebene klassische Übergangsregel angewandt werden soll oder MAX($\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}$) zur Entscheidung des Pfades verwendet werden soll. Bei der Max-Regel MAX($\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}$) wird strikt der Pfad mit dem höchsten Produkt gewählt. Udomsakdigool et al., [11] schlagen hierfür eine Konstante Q vor (0 <= Q <= 1). Eine Ameise wählt für sich dabei jeder Entscheidung einen Zufallswert q (0 <= q <= 1). Ist q <= Q verfährt sie nach der Max-Regel, andernfalls nach der klassischen proportionalen Übergangsregel. Im Test von Udomsakdigool et al. [11] haben sich hier 0.1, 0.3 und 0.5 als gute Werte ergeben.

5. Tests und Messungen

Ziel ist die möglichst optimale Entscheidung für einen Algorithmus, eine Heuristik und die Festlegung der Parameter.

5.1. Allgemeine Vorgehensweise

Gesucht ist die Kombination aus Algorithmus, Heuristik und Parametern (A,H,P) für ein möglichst optimales Ergebnis. Es wird ein schrittweises Vorgehen und Wählen durchgeführt. Hierfür wird eine Referenzkonfiguration benötigt, mit welcher im weiteren Vorgehen verglichen wird.

Zunächst wird ein Referenz-Algorithmus A_R gewählt. Mit dessen Hilfe werden anschließend die Parameter P aus verschiedenen Kombinationen, durch Messungen und Vergleichen festgelegt.

Nachdem das Referenz-Set (A_R, H, P) festgelegt ist, werden die Algorithmen durchgetestet, um den geeignetsten Algorithmus A_B zu bestimmen. Dabei werden die Sets (A_1, H, P), (A_2, H, P), (A_3, H, P), ... getestet, um den geeignetsten zu bestimmen.

Das hierbei gewählte beste Set ist die gesuchte End-Konfiguration.

5.2. Testdaten (Seed)

Das Problem wurde mit 200 Bearbeitern und 3000 zuzuordnenden Aufträgen definiert. Ein Problem Set besteht also aus 200 Bearbeitern und 3000 Aufträgen, welche zufällig verteilt innerhalb folgender Koordinaten (die in etwa Deutschland einschließen) platziert werden: latMin = 47.437836; latMax = 54.939949; lngMin = 5.909917; lngMax = 15.180695. Jeder Bearbeiter bekommt einen fiktiven Stundenlohn,der zufällig zwischen 10 und 20 Währungseinheiten gewählt wird. Die Bearbeitungsdauer eines Auftrags wird jedem Auftrag

zufällig zwischen 6 und 16 Stunden zugewiesen. Die Verspätungskosten je Stunde werden jedem Auftrag individuell und zufällig zwischen 100 und 1000 Geldeinheiten zugeordnet.
Der späteste Abgabetermin (gesamt) berechnet sich aus der mittleren Bearbeitungszeit eines Auftrags multipliziert mit der Anzahl der Aufträge. Alle Aufträge bekommen einen Abgabetermin zugeteilt, der zufällig zwischen 0 (Startzeitpunkt der Berechnung) und dem spätest Abgabezeitraum (gesamt) liegt.
Mit einer Wahrscheinlichkeit von je 50% bekommt jeder Auftrag eine optionale Anforderung {A,B,C,D,E,F} zugeordnet sowie zu 50% Wahrscheinlichkeit jeweils eine der obligatorischen Anforderungen {Ob_1, Ob_2}. Genau so werden jedem Bearbeiter zufällig mit einer Wahrscheinlichkeit von je 50% Qualifikationen {A,B,C,D,E,F,Ob_1,Ob_2} zugeordnet.
Zu vergleichende Messungen werden jeweils mit dem selben Testdatenset durchgeführt. Für das Testen verschiedener Parameter wird das selbe Test-Datenset verwendet. Für die Messungen zum Vergleich der Algorithmen wird ebenfalls jeweils der selbe Testdatensatz verwendet, jedoch ein anderer, als für das Vergleichen der Unterschiedlichen Parameter. So wurden insgesamt 5 verschiedene Problem-Instanzen genutzt. Da zum Vergleich keine absoluten Werte, sondern jeweils die Abweichungen von der Untergrenze zur aktuellen Messreihe herangezogen werden, sind die Statistiken meist trotzdem miteinander vergleichbar.

5.3. Messverfahren

Um spätere Auswertungen zu vereinfachen, soll zu jeder Messung eine JSON-Datei erstellt werden, welche alle zu der Messung relevanten und alle erfassten Daten vorhält.

Im Verlauf einer Messung soll erfasst werden:

- das Ergebnis der Kosten C
- C je einzelner Iteration
- der maximal benötigte Arbeitsspeicher in der Spitze
- das Endergebnis (Mitarbeiter-Auftrags-Zuweisung)
- die Pheromon-Matrizen nach 5%, 10%, 20%, 40%, 60%, 80% und 100% der Iterationen

Diese werden zusammen mit allen relevanten Parametern zum Ablauf der Messung gespeichert:

- ausgeführter Algorithmus und Heuristik

- den Parametern (Anzahl Ameisen, Verdunstung, ...)
- genutzter Testdatensatz

Die Messungen werden durchgeführt für 200 Mitarbeiter und 3000 Aufträge. Dies entspricht dem der Arbeit zugrunde liegenden realen Problem. Die Messungen pro zu messendem Set werden 10 mal wiederholt.

Zur Auswertung der jeweiligen Ergebnisse (Kosten) werden der beste, schlechteste und durchschnittliche Wert [9] geteilt durch die durchschnittliche Abweichung [3] von der Untergrenze betrachtet:

$$\frac{C-LB}{LB}\times 100\,\% \tag{19}$$

LB (Lower Bound) ist die Untergrenze, der niedrigste Wert unter den zu betrachtenden Kosten. Wenn C als Beispiel doppelt so hoch ist, wie die minimal gefunden Kosten (LB), so ist die Abweichung zu diesen minimalen Kosten 100%.

5.4. Mögliche Konfigurationen

Es werden folgende Algorithmen aus Abschnitt 4.3. untersucht:

- ALG1: Greedy Algorithmus
- ALG4: erst Reihenfolge, dann Zuordnung
- ALG5: erst Zuordnung, dann Reihenfolge je Maschine
- ALG6: erst Zuordnung, dann Reihenfolge global

Der einfache Ameisenalgorithmus hat sich im Test als ungeeignet heraus gestellt, da eine Ameise durch die mehrdimensionale Pheromon-Matrix bei jeder einzelnen Entscheidung |I| * |J| * |J| Wahlmöglichkeiten hat, wodurch die Berechnungszeit für jede einzelne Ameise in Vergleich zu den anderen Algorithmen wesentlich höher war. Der Greedy Algorithmus wird nur zum Vergleich heran gezogen und mit betrachtet.

Als Heuristik wird die mehrfach in der Literatur [4][5][6][15] erwähnte kürzeste Distanz 1/d genutzt.

Die folgenden Parameter werden untersucht und festgelegt:

- Anzahl der Ameisen (K)
- Anzahl Iterationen (N_{Cmax})
- Pheromon-Verdunstung (ρ)
- Einfluss von Pheromon(α) und Heuristik (β): β = 1-α
- Übergangsfunktion-Weiche Q

Das Pheromon Update $\Delta\tau_{ij}$ und der Initiale Pheromon-Wert τ_0 werden beide mit 1 festgelegt.

5.5. Referenz Konfiguration festlegen

Zur Vereinfachung der Messung und der Vergleichbarkeit wird eine Referenzkonfiguration festgelegt. An dieser werden dann jeweils die verschiedenen Algorithmen und Heuristiken gemessen.

Als Referenz-Algorithmus wurde sich für den ALG5 „erst Maschinenzuordnung, dann Reihenfolge je Maschine“ entschieden. Die Zuordnung zur Maschine (bzw. Bearbeiter im untersuchten Problem) vor der Bestimmung der Reihenfolge in der Tour erscheint schlüssig und ALG6 ist eine vermutete Vereinfachung (und eventuell auch Verbesserung), resultierend aus ALG5.

Als Heuristik wird die „kürzeste Entfernung“ 1/d verwendet, da diese häufig in der Literatur zu Basis-Implementierungen oder untersuchen von Algorithmen anklang findet [4][5][6][15].

Als Referenz Konfiguration wurden in ersten Tests folgende Werte ermittelt und festgelegt:

$K = 10$, $N_{Cmax} = 10$, $\rho = 0.4$, $\alpha = 0.3$, $Q = 0.1$, $Q_a = 0$

5.6. Durchführung der Messungen

5.6.1. Bestimmung der Parameter

In diesem Abschnitt sollen die Parameter Anzahl der Ameisen (K), Anzahl Iterationen (N_{Cmax}), Pheromon Verdunstung (ρ), Einfluss von Pheromon(α) und Heuristik (β) und Übergangsfunktion-Weiche Q bestimmt werden. Pheromon Update $\Delta\tau_{ij}$ und Initialer Pheromon-Wert τ_0 werden mit 1 festgelegt. Der Einfluss der Heuristik (β) wird definiert als $\beta = 1 - \alpha$;
Alle Messungen werden hierbei mit $N_{Cmax} = 100$ durchgeführt, wodurch später am Verlauf der Kosten N_{Cmax} abgeleitet werden kann. Sollten die Kosten früher stagnieren, wird N_{Cmax} entsprechend verringert oder erhöht, falls bei $N_{Cmax} = 100$ noch keine Stagnation erkenntlich sein sollte.

Die Referenz-Konfiguration ist: $K = 10$, $\rho = 0.4$, $\alpha = 0.3$, $Q = 0$. Diese Werte liegen im üblichen Bereich der im Abschnitt „Parameter“ referenzierten Quellen und haben in ersten Tests Ergebnisse ähnlich des Greedy Algorithmus geliefert.

Im ersten Schritt sollen die Parameter K, ρ und α durch Messungen näher bestimmt werden. Dabei werden folgende Werte betrachtet, die in etwa um die Referenzkonfiguration liegen:

- K: 10, 30, 60
- ρ: 0.2, 0.4, 0.7
- α: 0.3, 0.5, 0.7

Da alle möglichen Kombination gemessen werden, ergibt dies 3*3*3 = 27 Messungen (je 10 Mal wiederholt).

Abbildung 8: Messung der durchschnittlichen Abweichung der Parameter-Konfigurationen am Referenz Algorithmus, Wertetabelle siehe Anhang 3

Die durchschnittliche Abweichung zur Untergrenze (Abbildung 8) zeigt genauer Betrachtet, dass die niedrigsten Kosten bei ρ = 0.4, 0.7 und α = 0.3 lagen. Die Anzahl der Ameisen schien eher einen geringeren Einfluss zu haben, der beste Wert lag hierbei bei K = 30.

In der nächsten Messung wurde nun das Umfeld der beiden besten Messungen (30/0.04/0.3) und (30/0.7/0.3) genauer Betrachtung. Folgende Werte wurden dabei betrachtet:

- K: 20, 30, 40
- ρ: 0.3, 0.4, 0.5, 0.6, 0.7, 0.8
- α: 0.2, 0.3, 0.4

Im ersten Durchlauf zeigte sich dass α = 0.2 gute Ergebnisse erzielte, weshalb α = 0.1 zur Messung hinzugenommen wurde.

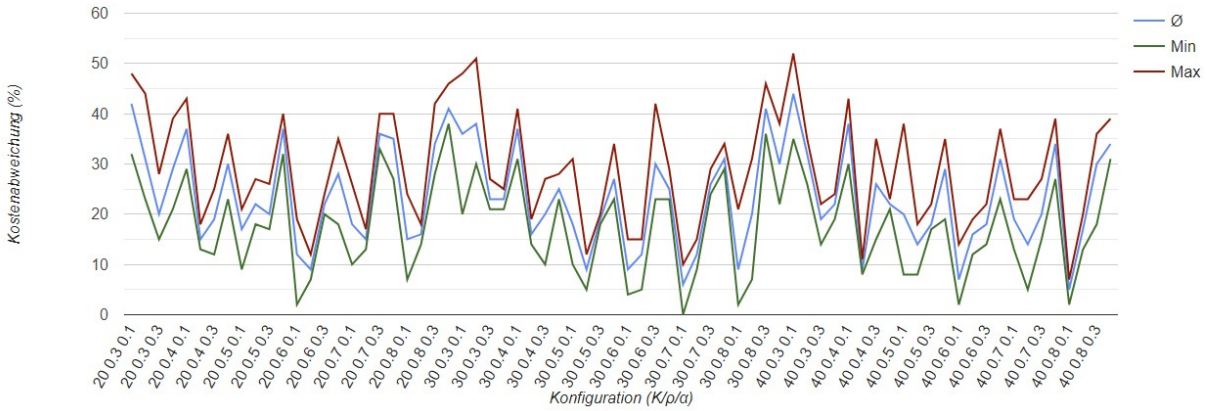

Abbildung 9: zweite Messung der durchschnittlichen Abweichung der Kosten zur Untergrenze, Wertetabelle siehe Anhang 4

Die Messung der durchschnittlichen Abweichung zur Untergrenze mit allen möglichen Konfigurationen (Abbildung 9) zeigte, dass die besten Werte bei p = 0.7 bis 0.8 und a = 0.1 bis 0.2 liegen, wobei K > 30 keine Verbesserung brachte. Die gewählte Konfiguration ist p = 0.7, a = 0.1, K = 30.

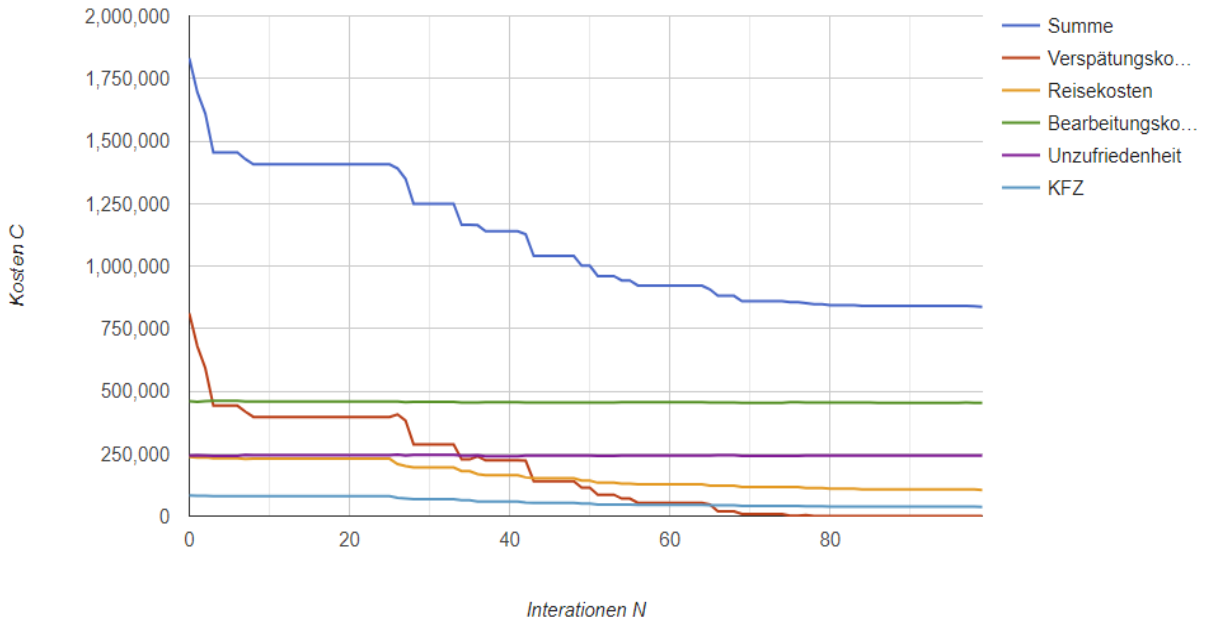

Abbildung 10: Kostenverlauf der gewählten Konfiguration

Abbildung 10 zeigt den Kostenverlauf der besten Messung. Die Kosten sind dabei wie folgt aufgeschlüsselt:

- KFZ: Benzin-Kosten

- Verspätungskosten: bei Verfristung des Abgabetermins entstandene Kosten
- Reisekosten: Stundenlohn des Fahrers * Fahrtzeit
- Bearbeitungskosten: Stundenlohn des Fahrers * Bearbeitungszeit vor Ort
- Unzufriedenheit: fiktive Kosten durch Verteilung von Aufträgen an Mitarbeiter, die deren Qualifikation nur ungenügend abdecken

Die Messung zeigt, dass N_{Cmax} = 80 festgelegt werden kann, da keine signifikanten Verbesserungen der Kosten bei N_{Cmax} > 80 zu erwarten sind. Dies wurde auch durch die Betrachtung der anderen Kosten-Verläufe ersichtlich und ist in der Abbildung nur exemplarisch demonstriert.

Im nächsten Schritt soll geprüft werden, ob ein Einstreuen der Max-Regel eine Verbesserung bewirkt. Bei der Max-Regel MAX(τ_{ij}^{α} * η_{ij}^{β}) wird strikt der Pfad mit dem höchsten Produkt aus Pheromon und Heuristik gewählt. 10 Messungen (je 10 mal wiederholt) werden hierfür für den Parameter Q durchgeführt, welcher die Wahrscheinlichkeit angibt, mit der eine Ameise bei der aktuellen Entscheidung nach der Max-Regel verfährt.

- Q: 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

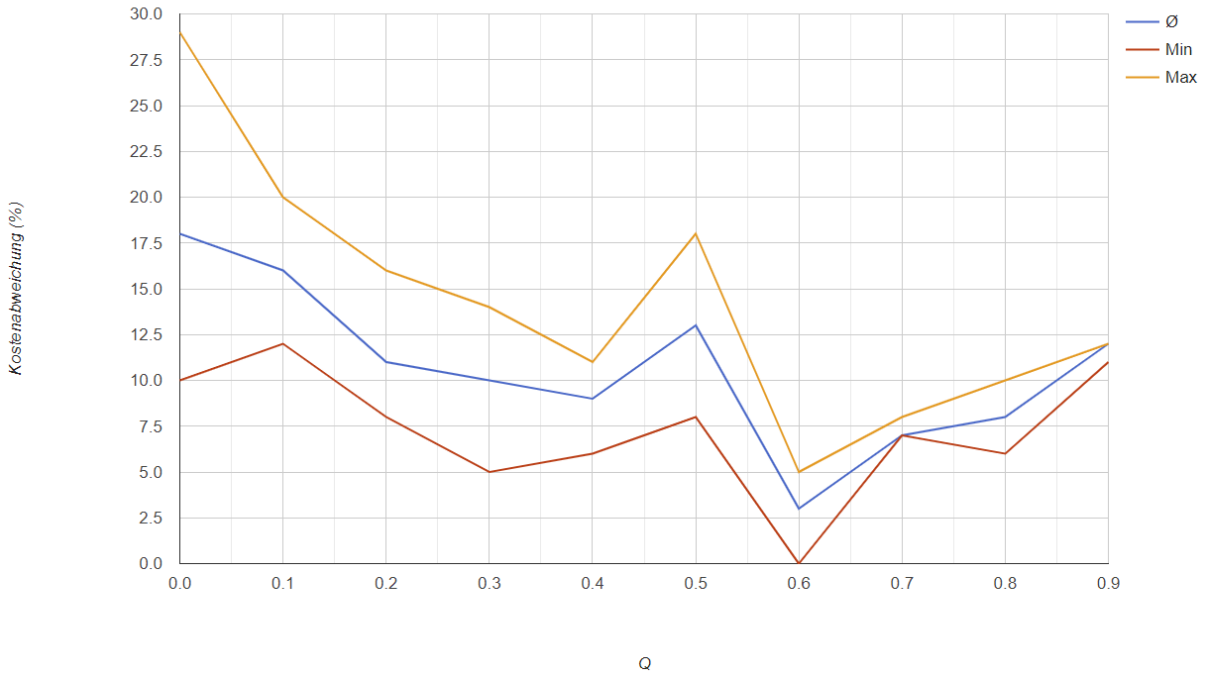

Abbildung 11: Kostenmessung bei unterschiedlichen Q Werten, Wertetabelle siehe Anhang 5

Die durchschnittlich geringsten Kosten wurden bei Q = 0.6 erzielt. Eine Verbesserung von Durchschnittlich 18% konnte erreicht werden (Abbildung 11).

Auf das selbe Problemset (Mitarbeiter und Aufträge) bezogen, ergibt sich nun folgender Kostenverlauf (Abbildung 12):

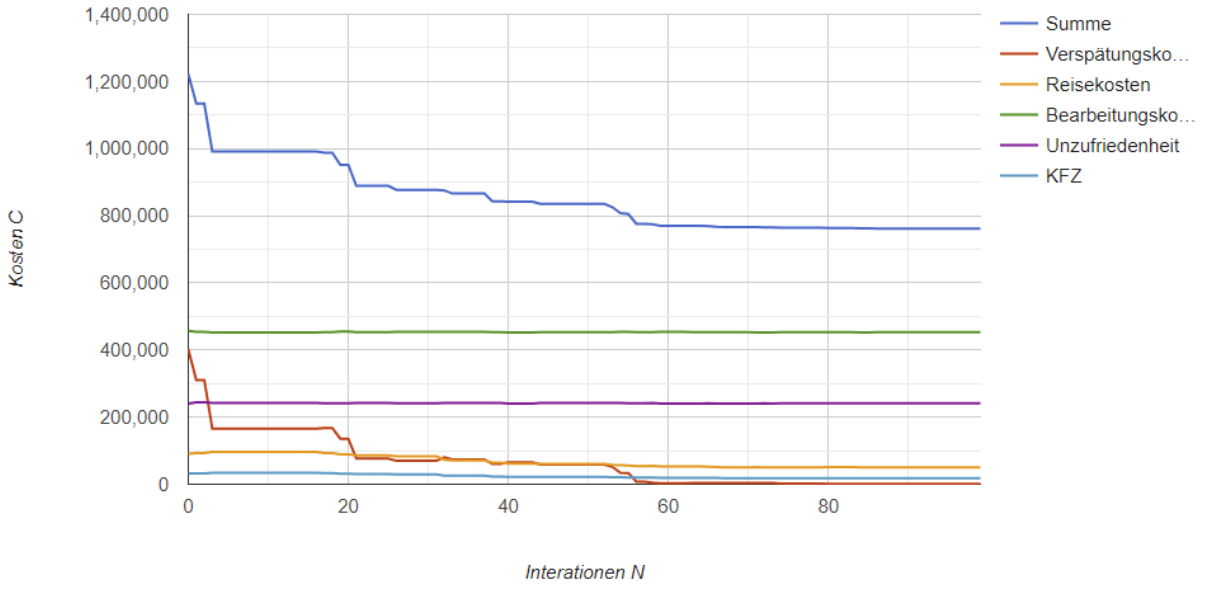

Abbildung 12: Kostenverlauf bei Q = 0.6

Die beste Messung bei (Q= 0) mit einen Ergebnis von 836246 konnte um 10% auf 761716 verbessert werden.

Daraus ergibt sich folgende gewählte Konfiguration:

- N = 80
- P = 0.7
- a = 0.1
- b = 1 - a
- K = 30

- $\Delta\tau_{ij} = 1$
- $\tau_0 = 1$
- $Q = 0.6$

5.6.2. Bestimmung des Algorithmus

Nachdem nun die Konfiguration festgelegt ist, wird diese an den Algorithmen getestet.

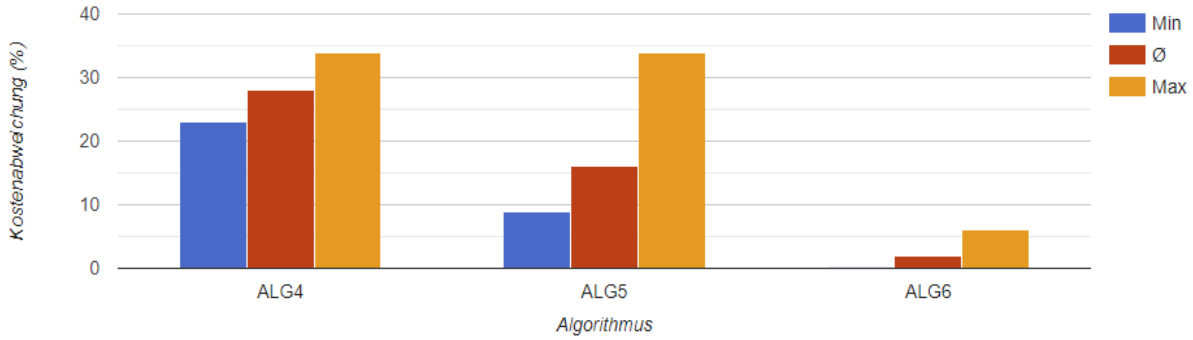

Abbildung 13: durchschnittliche Abweichung der Kosten unterschiedlicher Algorithmen, Wertetabelle siehe Anhand 6

Wie in Abbildung 13 ersichtlich ist, hatte der ALG6 die mit Abstand geringsten Kosten, weshalb dieser der gewählte Algorithmus ist. Der Greedy Algorithmus ergab zum Vergleich eine durchschnittliche Kostenabweichung von +23% und liegt dabei in etwa mit ALG4 gleichauf. Im Folgenden wird nun die Kostenentwicklung der drei Algorithmen betrachtet, gemittelt über jeweils 10 Messungen auf dem selben Problemsatz.

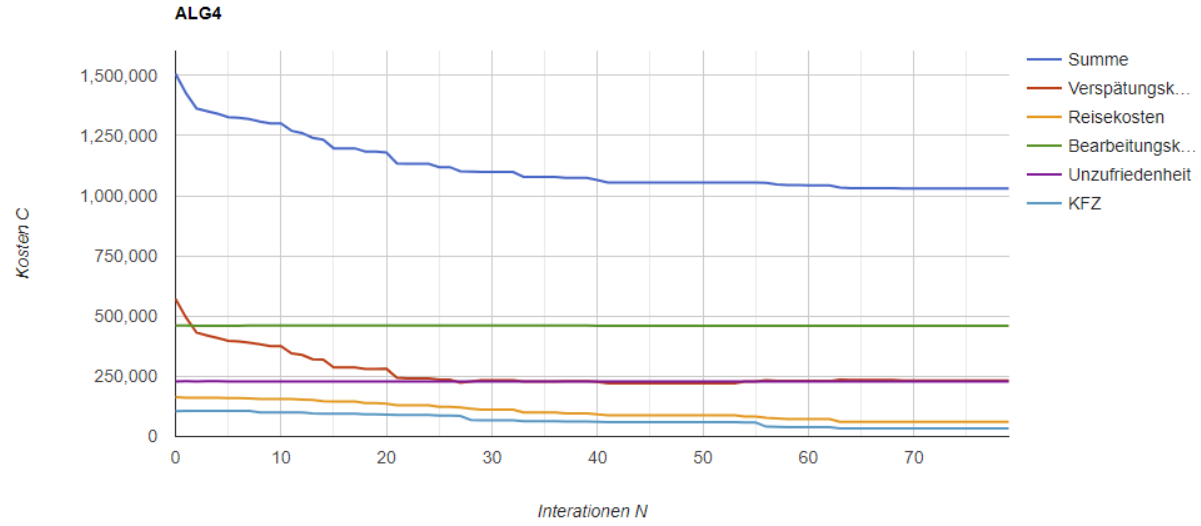

Abbildung 14: Kostenentwicklung ALG4

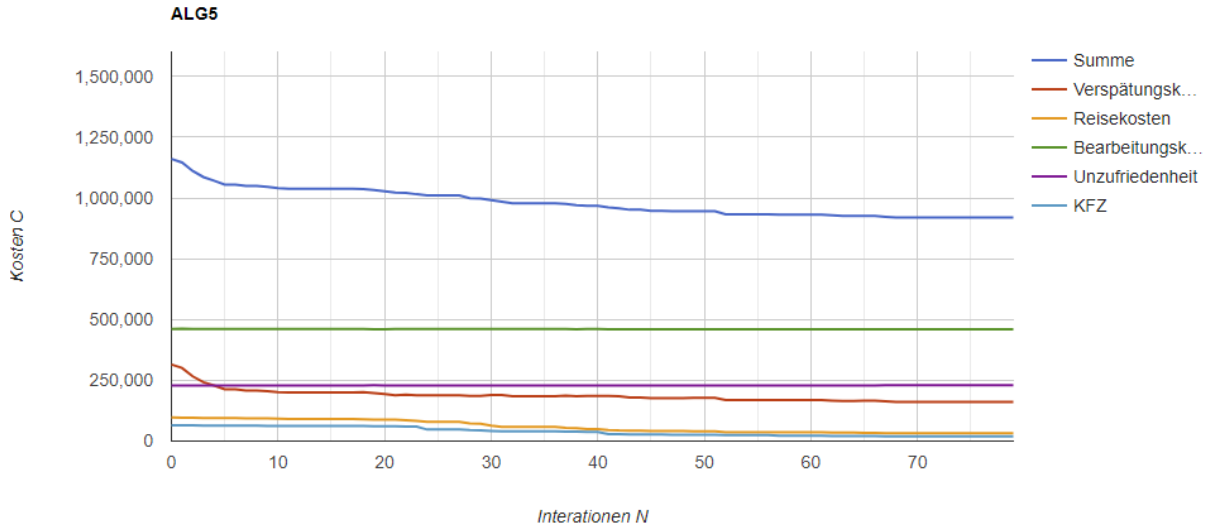

Abbildung 15: Kostenentwicklung ALG 5

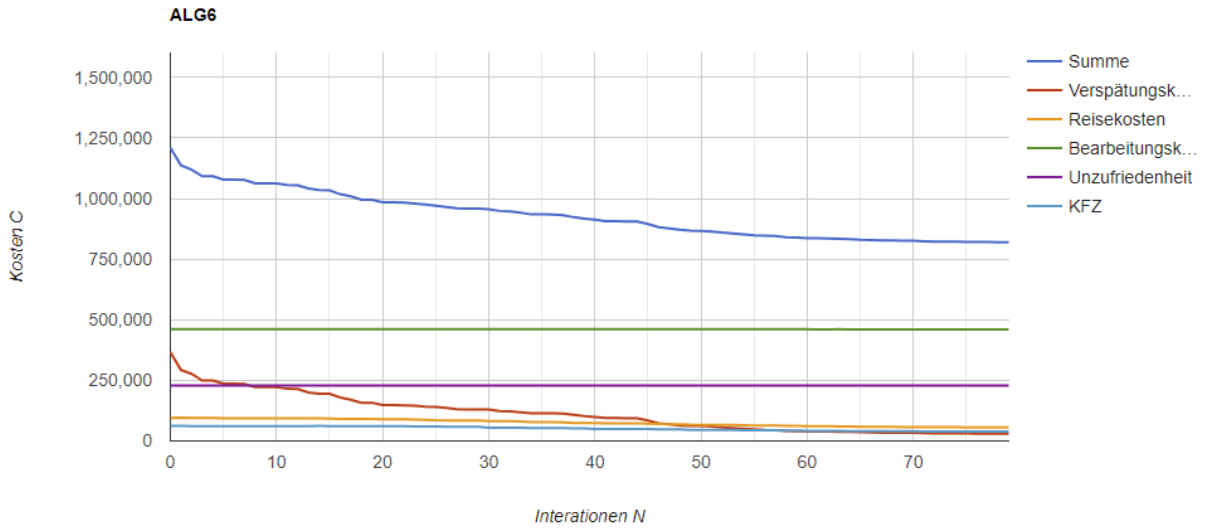

Abbildung 16: Kostenentwicklung ALG 6

In Abbildung 14 ist ersichtlich, dass ALG 4 in den ersten Iterationen wesentlich höhere Kosten ermittelt, als die anderen beiden Algorithmen (Abbildung 15, Abbildung 16), die hier in etwa gleichauf liegen. Algorithmus 6 reduziert die Verspätungskosten wesentlich effektiver als Algorithmus 5.

Die nächsten Grafiken zeigen die Zuordnungen der Aufträge zu den Bearbeitern, für den Greedy-Algorithmus (Abbildung 17) und ALG6 (Abbildung 18). Dabei wurde jedem Bearbeiter zufällig eine Farbe zugeordnet. Bei der Entwicklung der Zuteilung über die Iterationen des Algorithmus (Abbildung 19) sind die Farbzuordnungen gleich geblieben. Da die Aufträge in der Regel im Umkreis des Bearbeiters liegen, entstehen auf der Karte sternartige Formationen.

Der Greedy-Algorithmus (Abbildung 17) ordnet die Aufträge nach der Nähe zum Mitarbeiter zu. In der Abbildung ist zu erkennen, dass nicht immer der naheste Bearbeiter einen Auftrag zugeordnet ist. Dies ist darin begründet, dass der Arbeiter jeweils nicht die Qualifikationen für den Auftrag besitzt und diesen deshalb nicht anfahren kann.

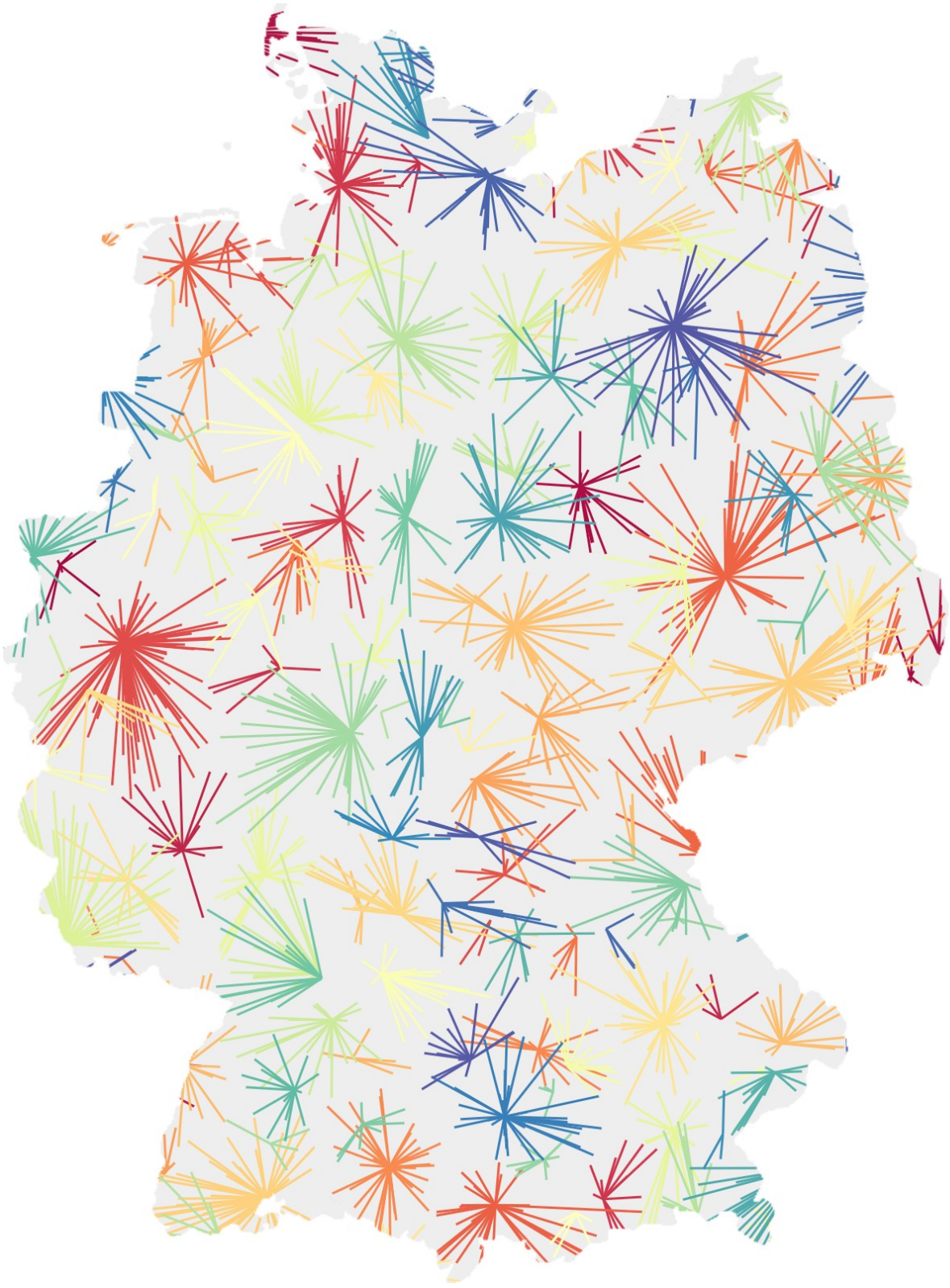

Abbildung 17: Auftragsverteilung Greedy Algorithmus

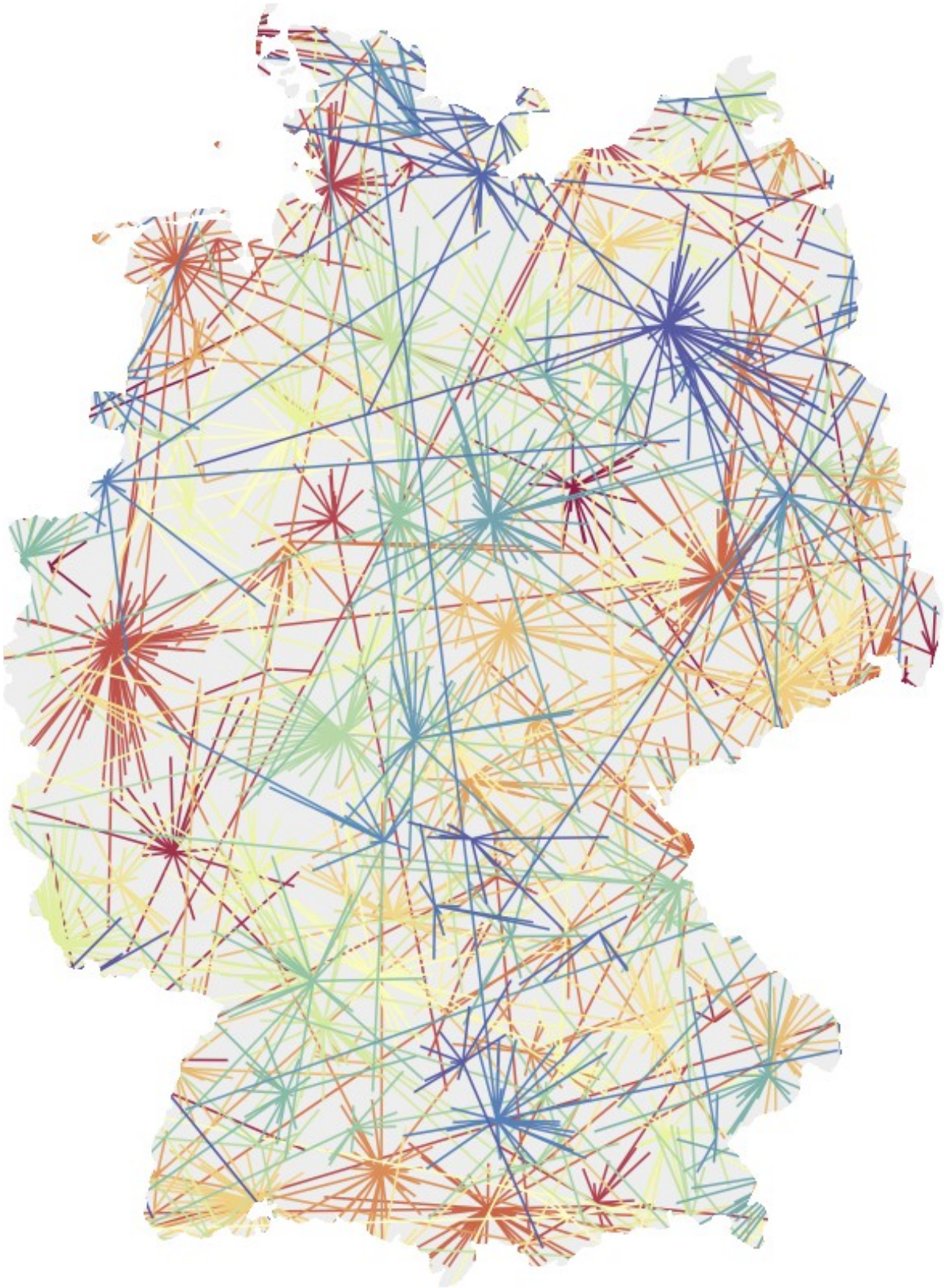

Abbildung 18: Auftragsverteilung ALG 6 (80 Iterationen)

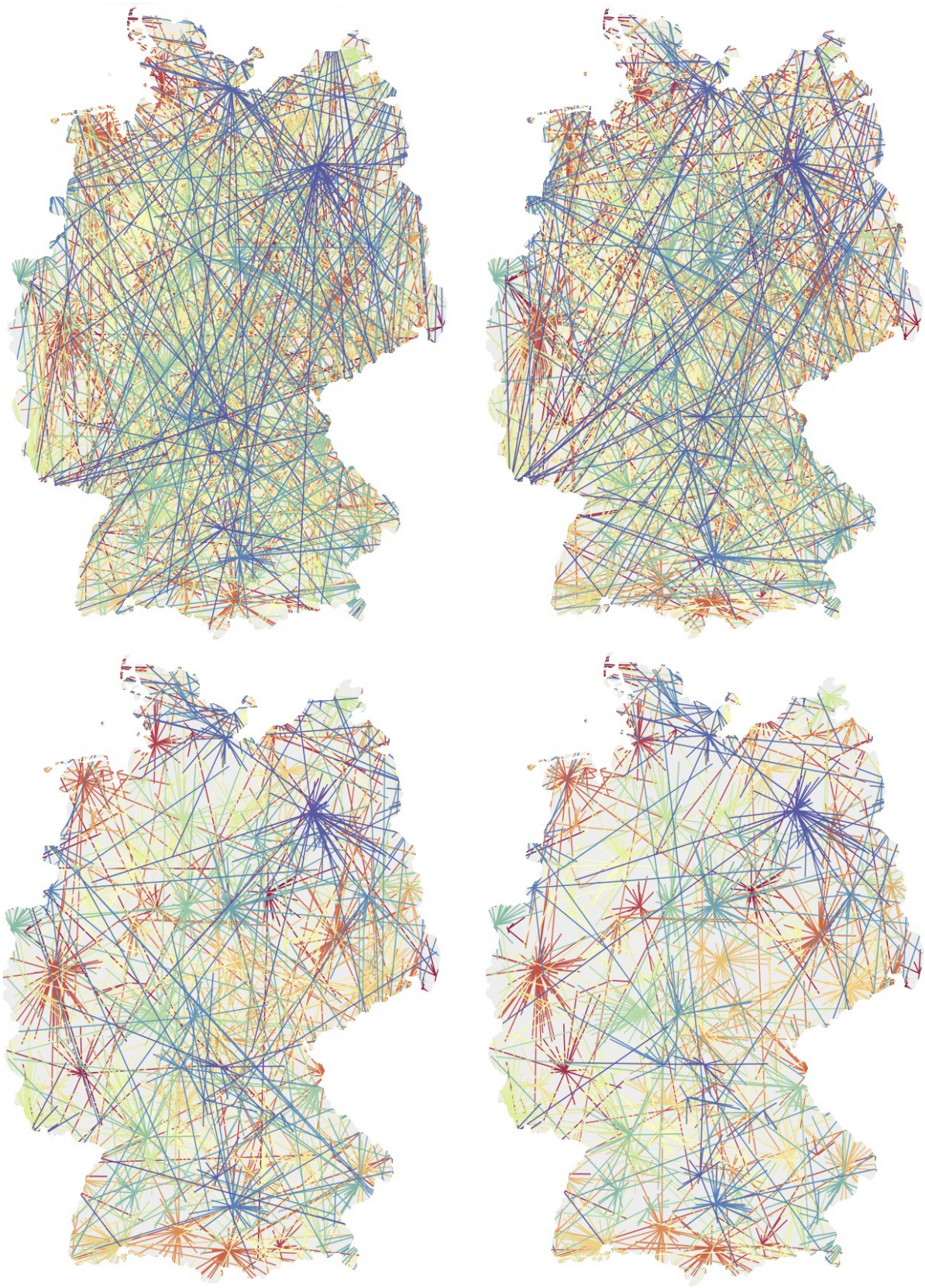

Abbildung 19: Auftragsverteilung ALG 6 mach 1, 20, 40, 60 Iterationen

Algorithmus 6 teilt Aufträge zwar meist regional, teilweise aber auch weit entfernten Bearbeitern zu. (Abbildung 18) Dies wird darin zu begründen sein, dass im gewählten Szenario verspätete Abgaben sehr teuer sind (um ein Vielfaches des Stundenlohns), weshalb auch größere Strecken in Kauf genommen werden. Die Entwicklung der Auftragsverteilung über die Iterationen des Algorithmus (Abbildung 19) zeigt, dass die eher regionale Zuordnung erst im Laufe der Iteration entsteht und in den ersten Iterationen noch sehr willkürlich wirkt.

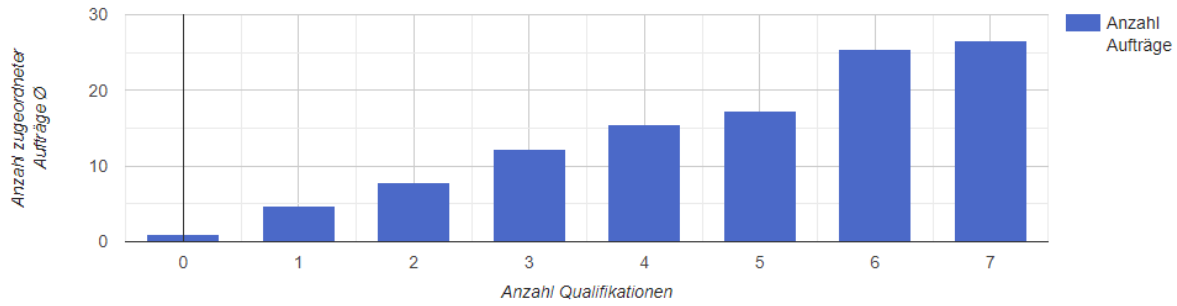

Abbildung 20: Verhältnis der Anzahl der Qualifikationen des Bearbeiters zur Anzahl der ihm zugeordneten Aufträge

Wie in Abbildung 18 zu sehen, fahren einige Mitarbeiter weit entfernte Aufträge an. Es ist anzunehmen, dass Mitarbeiter mit vielen Qualifikationen mehr Aufträge anfahren können und deshalb auch anfahren müssen. Abbildung 20 zeigt, dass diese Vermutung sich bewahrheitet. Dadurch steigen die Unzufriedenheits-Kosten. Diese sind relativ hoch aber über den Verlauf der Iterationen auch kaum verändert (Abbildung 16). Dies ist wiederum darin begründbar, dass es nur sehr wenige Mitarbeiter mit vielen Qualifikationen gibt (Abbildung 21) und die Unzufriedenheits-Kosten je Mitarbeiter berechnet werden und zwischen 0 und 100 Kosteneinheiten liegen.

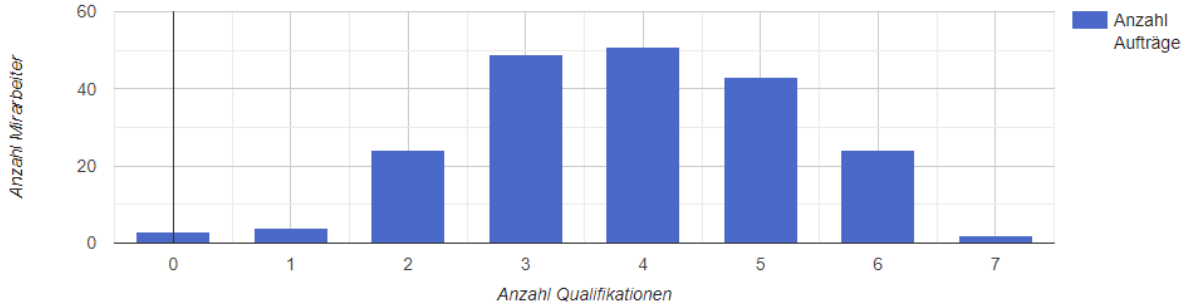

Abbildung 21: Verhältnis von Anzahl der Qualifikationen zu den Mitarbeitern

5.6.3. Lokale Suche

Desweiteren kann versucht werden, das Ergebnis durch lokale Suche zu verbessern. Dadurch soll der Suchraum weiter exploriert werden. Ahmadizar et al. [19] benutzen bei ihrem Sortierungsproblem folgendes Verfahren:

- Das Ergebnis liegt vor
- Generiere für jeden Auftrag eine Zufallszahl R [0,1]
- Wenn R <= T, T ist eine vordefinierte Konstante
    - Für jede Position i (i = 1, … N)
        - Platziere den Auftrag an Position i
        - Bestimme die Güte der neuen Lösungen

Die Konstante T bestimmt die Bedeutung der lokalen Suche im Vergleich zum Ameisenalgorithmus.
Der Ansatz kann genutzt werden, um die Sortierung innerhalb der einzelnen Fahrer weiter zu explorieren. Er kann ebenfalls in der Verteilung genutzt werden, indem man alle Aufträge iteriert und jeweils mit der Ursprungsposition des Auftrags tauscht.

Chen et al. [20] nutzen zur lokalen Suche den NEH Algorithmus, welcher 1983 von Nawaz [21] zur Lösung von Frequenz-Problemen beschrieben wurde. Dieser sieht das schrittweise Neu-Einsortieren der Elemente in einer Liste vor. Als Beispiel dient die von den Ameisen gefundene Lösung {a3, a2, a4, a1}. Die Neue Liste wird nun wie folgt aufgebaut:

- Das erste Element der Lösung wird eingefügt: L^{new} = { a3 }

- Das nächste Element der ursprünglichen Lösung (a2) wird eingefügt, indem es an allen |L| + 1 Positionen getestet wird. Die neue Liste mit der besten Güte gewinnt. Getestet werden also {a2, a3} und {a3, a2}. In diesem Beispiel nehmen wir an, dass L^{new} = {a2, a3} das bessere Ergebnis hat.
- Das nächste Element der ursprünglichen Lösung (a4) wird nun nach dem selben Verfahren in die neue Liste einsortiert. Getestet werden die Lösungen: {a4, a2, a3}, {a2, a4, a3}, {a2, a3, a4}.
- Dies wird wiederholt, bis alle Elemente der ursprünglichen Lösung neu einsortiert sind.

Desweiteren könnten Mutationen [6], durch zufälliges Vertauschen von Aufträgen innerhalb einer Lösung dazu beitragen, den zu schnellen Fall in ein lokales Optimum zu verhindern. Oder es werden Teile zweier Lösungen kombiniert (Kreuzung [6]), um weitere Lösungen zu finden.

Ahmadizar [19] und der NEH Algorithmus [20] permutieren das Problem auf eine ähnliche Weise, wobei beim von Ahmadizar beschrieben Algorithmus durch eine Konstante der Grad der Exploration gesteuert wird. Untersucht wird daher der Algorithmus von Ahmadizar und ob dieser den aktuellen Lösungsansatz verbessern kann. Im Paper wird als Konstante T ein Wert um 10/N vorgeschlagen, welcher in dem hier Beschrieben Problem 10/3000 = 0.3% betragen würde. Diese Zahl ist bewusst klein gewählt, da diese Exploration sonst einen zu großen Umfang annehmen würde. Er werden zusätzlich T * N * (N -1) Lösungen untersucht. Dies entspricht bei dem gegebenen Problem, bei T = 0.003 und N = 3000, zusätzlichen 26991 Lösungen, welche Untersucht werden. Der Auftrag wird dabei jeweils von seiner ursprünglichen Position entfernt und an der neuen Position eingefügt.
Eine Kreuzung von zwei Lösungen ist schwierig, da jeder Auftrag nur einmal vergeben werden kann, was wiederum das Tauschen ganzer Teile von Lösungen schwierig macht. Da der Algorithmus von Ahmadizar bereits eine Mutation des Ergebnisses darstellt, wird auf die Untersuchung anderer Mutationsarten verzichtet.

Bei 80 Iterationen werden im Schnitt zusätzliche 2,2 Millionen Lösungen exploriert. Ohne lokale Suche werden von 30 Ameisen in 80 Iterationen lediglich 2400 Lösungen untersucht. Da die Ameisen allerdings zur Lösungsfindung viele Pheromon-Werte in Relation zu einander

betrachten müssen, ist die Ausführungszeit nicht im selben Maße gestiegen, sondern hat sich im Mittel nur verzehnfacht.

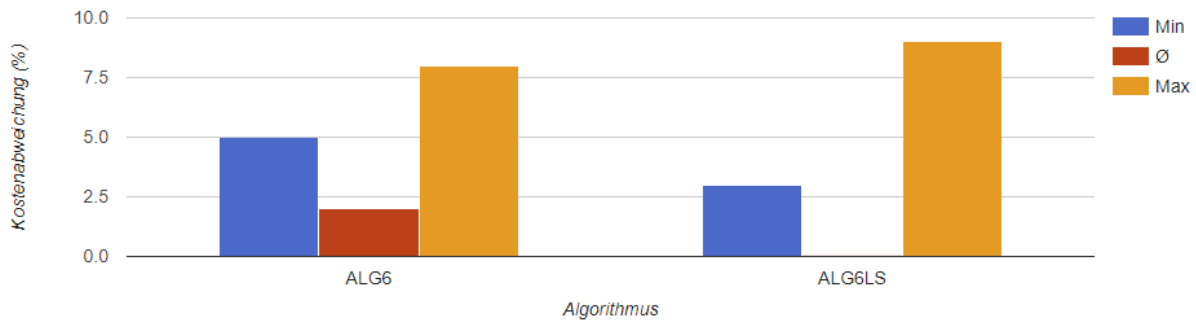

Abbildung 22: Kostenvergleich ALG6 mit und ohne lokale Suche, Wertetabelle siehe Anhang 7

Die Kosten sind mit lokaler Suche im Schnitt um 2% gesunken (Abbildung 22). Abbildung 23 zeigt die Auftragsverteilung der besten Lösung, welche mit lokaler Suche gefunden wurde. Es zeigt sich, dass viel längere Wegstrecken in Kauf genommen werden.

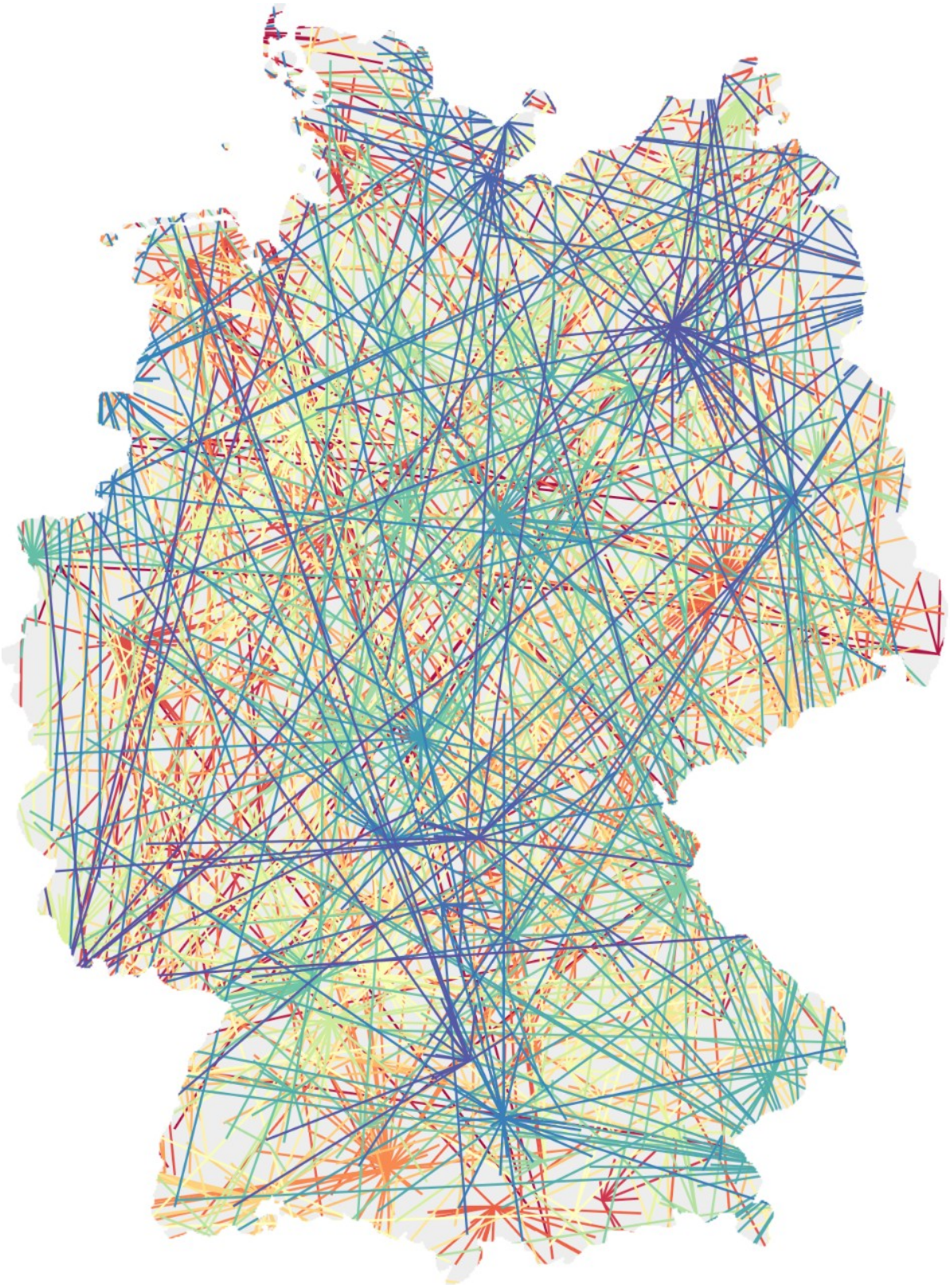

Abbildung 23: Auftragsverteilung bei ALG6 mit lokaler Suche

Der Grund wird bei der Kostenentwicklung deutlich. Der Algorithmus mit lokaler Suche ist der einzige, welcher die Bearbeitungskosten im Vergleich zu den anderen getesteten Algorithmen ohne lokale Suche um ca. 20% minimiert (Abbildung 24).

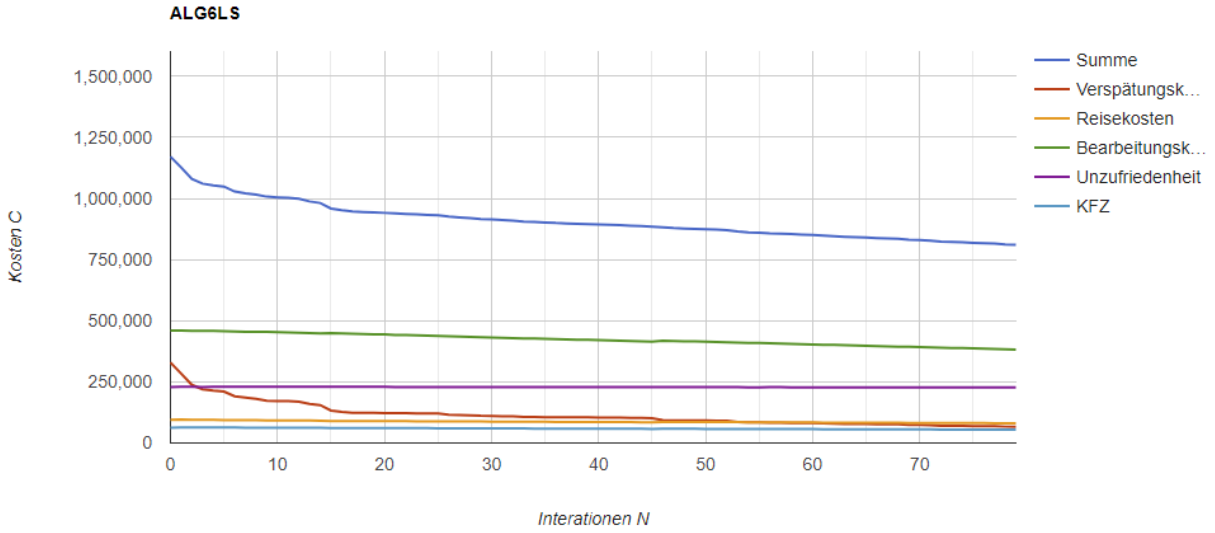

Abbildung 24: Durchschnittliche Kostenentwicklung bei ALG6 mit lokaler Suche

Die lokale Suche hat also Bearbeitern mit geringerem Stundenlohn wesentlich weiter entfernte Aufträge mit längeren Bearbeitungszeiten zugeordnet, wenn dadurch geringfügig Kosten gespart werden konnten.

6. Fazit und Ausblick

Es wurde ein Ameisenalgorithmus entwickelt und konfiguriert, welcher das Problem ca. 50% besser löst, als der zum Vergleich herangezogene Greedy-Algorithmus. Die lokale Suche brachte nur eine geringfügige Verbesserung unter wesentlich höherem Aufwand und wird daher nicht zur Lösung dieses Problems empfohlen.
Die Ergebnisse haben gezeigt, dass die Verteilung ausgewogener gestaltet werden könnte. Beispielsweise könnte man Anpassen:

- Die Heuristik oder die Fitness-Funktion könnte belohnen, dass die Anzahl der wenigsten und meisten Aufträge je Bearbeiter nicht zu weit vom Mittel abweicht
- Die längste Strecken minimieren

Desweiteren können bekannte Verfahren, wie das Benutzen mehrerer Pheromon Matrizen, mehrerer Kolonien und dominanter Fronten zum Update ([9],[11],[12]) genutzt werden, um die verschiedenen Kriterien und Kosten einzeln zu Optimieren.

Literaturverzeichnis

[1] Nazif, H. (2015). Solving Job Shop Scheduling Problem Using an Ant Colony Algorithm. Journal of Asian Scientific Research, 5(5), 261-268. doi:10.18488/journal.2/2015.5.5/2.5.261.268

[2] Tavares Neto, R F. (2015). An ant colony optimization approach for the parallel machine scheduling problem with outsourcing allowed. Journal of Intelligent Manufacturing, 26(3), 527-538. doi:10.1007/s10845-013-0811-5

[3] Arnaout, J., Musa, R., & Rabadi, G. (2008). Ant colony optimization algorithm to parallel machine scheduling problem with setups. 2008 IEEE International Conference on Automation Science and Engineering, 578-582. doi:10.1109/COASE.2008.4626566

[4] Chaouch, I., Driss, O B., & Ghedira, K. (2017). A Modified Ant Colony Optimization algorithm for the Distributed Job shop Scheduling Problem. Procedia Computer Science, 112, 296-305. doi:10.1016/j.procs.2017.08.267

[5] Teschemacher, U., & Reinhart, G. (2016). Enhancing Constraint Propagation in ACO-based Schedulers for Solving the Job Shop Scheduling Problem. 48th CIRP Conference on manufacturing systems. doi:10.1016/j.procir.2015.12.071

[6] Rondon, R L A., & Carvalho, A S. (2009). Solving a real job shop scheduling problem. 2009 35th Annual Conference of IEEE Industrial Electronics, 2494-2498. doi:10.1109/IECON.2009.5415226

[7] Holloway, C A., & Nelson, R T.(1973). Alternative Formulation of the Job Shop Problem with Due Dates. Management Science, 20(1), 65-75. doi:10.1287/mnsc.20.1.65

[8] Huang, R. (2008). Ant colony system for job shop scheduling with time windows. The International Journal of Advanced Manufacturing Technology, 39(1), 151-157. doi:10.1007/s00170-007-1203-9

[9] Ning, J., Zhang, C., Sun, P., & Feng, Y. (2018). Comparative Study of Ant Colony Algorithms for Multi-Objective Optimization. Information, 10(1), 11. doi:10.3390/info10010011

[10] Eric Hsueh-Chan Lu, , Ya-Wen Yang, , & Zeal Li-Tse Su. (2016). Ant Colony Optimization solutions for logistic route planning with pick-up and delivery. 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 000808-000813. doi:10.1109/SMC.2016.7844340

[11] Udomsakdigool, A., & Kachitvichyanukul, V. (2008). Multiple colony ant algorithm for job-shop scheduling problem. International Journal of Production Research, 46(15), 4155-4175. doi:10.1080/00207540600990432

[12] Alaya, I., Solnon, C., & Ghedira, K. (2007). Ant Colony Optimization for Multi-Objective Optimization Problems. 19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007), 1, 450-457. doi:10.1109/ICTAI.2007.108

[13] Florez, E., Gomez, W., & Lola Bautista, M. (2013). An Ant Colony Optimization Algorithm For Job Shop Scheduling Problem. International Journal of Artificial Intelligence & Applications, 4(4), 53-66. doi:10.5121/ijaia.2013.4406

[14] Berrichi, A. (2013). Efficient bi-objective ant colony approach to minimize total tardiness and system unavailability for a parallel machine scheduling problem. The International Journal of Advanced Manufacturing Technology, 68(9), 2295-2310. doi:10.1007/s00170-013-4841-0

[15] Sun, B., Wang, H., & Fang, Y. (2009). Application of Ant Colony Algorithm in Discrete Job-Shop Scheduling. 2009 Second International Symposium on Knowledge Acquisition and Modeling, 2, 29-32. doi:10.1109/KAM.2009.71

[16] Turguner, C., & Sahingoz, O K. (2014). Solving job shop scheduling problem with Ant Colony Optimization. 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI), 385-389. doi:10.1109/CINTI.2014.7028706

[17] Liang, P., Yang, H., Liu, G., & Guo, J. (2015). An Ant Optimization Model for Unrelated Parallel Machine Scheduling with Energy Consumption and Total Tardiness. Mathematical Problems in Engineering, 2015, 1-8. doi:10.1155/2015/907034

[18] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T.(2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), 182-197. doi:10.1109/4235.996017

[19] Ahmadizar ,F. (2012). A new ant colony algorithm for makespan minimization in permutationflow shops, Elsevier Computers & Industrial Engineering jounal 63, 355-361 doi:10.1016/j.cie.2012.03.015

[20] Chen, R., Fu-Ren Hsieh, , & Di-Shiun Wu.(2012). Heuristics based ant colony optimization for vehicle routing problem. *2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA),* 1039-1043. doi:10.1109/ICIEA.2012.6360876

[21] Nawaz, M., Enscore, E E., & Ham, I.(1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega, 11*(1), 91-95. doi:10.1016/0305-0483(83)90088-9

[22] Luo, Y., & Waden, Y P.(2017). Processing time tolerance-based ACO algorithm for solving job-shop scheduling problem. *IOP Conference Series: Earth and Environmental Science, 69*, 012181. doi:10.1088/1755-1315/69/1/012181

[23] Li, L., Qiao, F., & Wu, Q.(2009). ACO-based scheduling of parallel batch processing machines to minimize the total weighted tardiness. *2009 IEEE International Conference on Automation Science and Engineering,* 280-285. doi:10.1109/COASE.2009.5234191

[24] Dorigo, M., Maniezzo, V., & Colorni, A.(1996). Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26*(1), 29-41. doi:10.1109/3477.484436

[25] Maniezzo, V., & Colorni, A.(1999). The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering, 11*(5), 769-778. doi:10.1109/69.806935

[26] Goss, S., Aron, S., Deneubourg, J L., & Pasteels, J M.(1989). Self-organized shortcuts in the Argentine ant. *Naturwissenschaften, 76*(12), 579-581. doi:10.1007/bf00462870

Abbildungsverzeichnis

Abbildungsverzeichnis

Anhang

Anhang 1: Einfluss Verdunstung und Pheromon-Update auf die Qualität des Ergebnisses

Die Tabelle zeigt für 20 Iteration eines Beispiels des Brücken-Experiments (Die Ameisen können zwischen 2 Wegen wählen, wobei der Eine doppelt so lang ist, wie der Andere), wie häufig sich die Ameisen für den kürzeren Weg entschieden haben (von 0.0 bis 1.0). Dies jeweils mit 0%, 10%, 25%, 50%, 75%, 90%, 100% Verdunstung der Pheromone bei jeder Iteration. In der letzten Spalte wurde keine Verdunstung untersucht, dafür wurde das Pheromon nach der Formel d_{best}/d positioniert, wodurch Ameisen auf der langen Route also nur halb soviel Pheromon je Streckenabschnitt verteilt haben, wie Ameisen auf der kurzen Route.

	0%	10%	25%	50%	75%	90%	100%	0%, update by quality
1	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.73
2	0.73	0.75	0.78	0.83	0.91	0.97	1.0	0.88
3	0.82	0.84	0.87	0.92	0.97	0.99	1.0	0.94
4	0.82	0.84	0.86	0.88	0.88	0.85	0.81	0.96
5	0.84	0.86	0.88	0.91	0.91	0.9	0.88	0.97
6	0.86	0.88	0.9	0.94	0.96	0.98	0.98	0.97
7	0.87	0.89	0.92	0.96	0.98	0.99	0.99	0.98
8	0.88	0.91	0.94	0.97	0.99	0.99	0.99	0.98
9	0.89	0.92	0.95	0.98	0.99	0.99	0.99	0.98
10	0.9	0.93	0.96	0.99	1.00	1.00	1.00	0.99
11	0.91	0.94	0.97	0.99	1.00	1.00	1.00	0.99
12	0.91	0.94	0.98	0.99	1.00	1.00	1.00	0.99
13	0.92	0.95	0.98	0.99	1.00	1.00	1.00	0.99
14	0.92	0.96	0.98	0.99	1.00	1.00	1.00	0.99
15	0.93	0.96	0.99	0.99	1.00	1.00	1.00	0.99
16	0.93	0.96	0.99	1.00	1.00	1.00	1.00	0.99
17	0.94	0.97	0.99	1.00	1.00	1.00	1.00	0.99
18	0.94	0.97	0.99	1.00	1.00	1.00	1.00	0.99
19	0.94	0.98	0.99	1.00	1.00	1.00	1.00	0.99

20	0.94	0.98	0.99	1.00	1.00	1.00	1.00	0.99

Anhang 2: Vergleich der Entwicklung von 1/d zu 1 – (d/d_{max})

Zeigt die Verlaufs-Entwicklung des Heuristik Wertes einmal mit 1/d und einmal mit 1 - (d/d_{max}) berechnet.

d	1/d	1 - d/1055
1	1	0,999
5	0,2	0,995
10	0,1	0,990
50	0,02	0,95
100	0,01	0,90
200	0,005	0,81
500	0,002	0,52
1000	0,001	0,05

Anhang 3: Parameter Vergleichsmessung 1

Es wurden mit ALG5 für alle Kombinaten aus K, p, a jeweils 10 Messungen durchgeführt, um die Kosten zu ermitteln (Durchschnitt, minimalisierte gemessene Wert und höchster gemessener Wert) für alle Kombinationen aus K (10,30,60), p (0.2, 0.4, 0.7), a (0.3, 0.5, 0.7). Die letzten drei Spalten zeigen die Abweichung der Kosten mit niedrigstem Wert (LB = 973750) berechnet via: (C – LB) / LB * 100%

K	p	a	C Ø	C min	C max	C Ø LB	C min LB	C max LB
60	0.2	0.3	1143708	1038611	1218157	17,45	6,66	25,10
60	0.2	0.5	1131387	1109854	1148098	16,19	13,98	17,90
60	0.4	0.7	1158931	1114954	1222481	19,02	14,50	25,54
60	0.4	0.5	1123094	1065881	1194481	15,34	9,46	22,67
60	0.4	0.3	1081827	1054276	1130791	11,10	8,27	16,13
60	0.2	0.7	1153564	1122698	1191772	18,47	15,30	22,39
60	0.7	0.5	1186136	1155547	1246808	21,81	18,67	28,04
60	0.7	0.3	1071001	1022517	1099688	9,99	5,01	12,93
60	0.7	0.7	1227472	1199891	1255482	26,06	23,22	28,93

30	0.2	0.7	1166851	1153427	1188241	19,83	18,45	22,03
30	0.4	0.7	1229780	1201449	1261658	26,29	23,38	29,57
30	0.7	0.3	1026762	973750	1081486	5,44	0,00	11,06
30	0.4	0.5	1136185	1107660	1174415	16,68	13,75	20,61
30	0.2	0.5	1138337	1054058	1206474	16,90	8,25	23,90
30	0.4	0.3	1050994	1018168	1081693	7,93	4,56	11,09
30	0.7	0.7	1278249	1188078	1352701	31,27	22,01	38,92
30	0.2	0.3	1215171	1181005	1242398	24,79	21,28	27,59
30	0.7	0.5	1171482	1095808	1275795	20,31	12,53	31,02
10	0.2	0.7	1215220	1193145	1239945	24,80	22,53	27,34
10	0.7	0.3	1192242	1069412	1320676	22,44	9,82	35,63
10	0.7	0.7	1318468	1213986	1522060	35,40	24,67	56,31
10	0.4	0.7	1237777	1191102	1325048	27,11	22,32	36,08
10	0.7	0.5	1332979	1299362	1398230	36,89	33,44	43,59
10	0.4	0.5	1194799	1128880	1302991	22,70	15,93	33,81
10	0.2	0.5	1159953	1142377	1176966	19,12	17,32	20,87
10	0.4	0.3	1127986	1116135	1137113	15,84	14,62	16,78
10	0.2	0.3	1235729	1142629	1297308	26,90	17,34	33,23

Anhang 4: Parameter Vergleichsmessung 2

Es wurden mit ALG5 für alle Kombinaten aus K, p, a jeweils 10 Messungen durchgeführt, um die Kosten zu ermitteln (Durchschnitt, minimalisierte gemessene Wert und höchste gemessene Wert) für alle Kombinationen aus K (20,30,40), p (0.3, 0.4, 0.5, 0.6, 0.7, 0.8), a (0.1, 0.2, 0.3, 0.4). Die letzten drei Spalten zeigen die Abweichung der Kosten mit niedrigstem Wert (LB = 836246) berechnet via: (C – LB) / LB * 100%

K	p	a	C Ø	C min	C max	C Ø LB	C min LB	C max LB
20	0.3	0.1	1189571	1104616	1239069	42	32	48
20	0.3	0.2	1099143	1028307	1200594	31	23	44
20	0.3	0.3	1006124	961859	1069345	20	15	28
20	0.3	0.4	1081904	1008694	1159376	29	21	39
20	0.4	0.1	1143273	1075836	1193182	37	29	43
20	0.4	0.2	960644	940867	982789	15	13	18

20	0.4	0.3	998442	938868	1046306	19	12	25
20	0.4	0.4	1086536	1030967	1135930	30	23	36
20	0.5	0.1	976012	908539	1015825	17	9	21
20	0.5	0.2	1022025	990619	1064993	22	18	27
20	0.5	0.3	1006602	978263	1056479	20	17	26
20	0.5	0.4	1148973	1101968	1174723	37	32	40
20	0.6	0.1	939874	856314	996460	12	2	19
20	0.6	0.2	912372	896425	934598	9	7	12
20	0.6	0.3	1016539	999897	1035772	22	20	24
20	0.6	0.4	1073544	988581	1130020	28	18	35
20	0.7	0.1	988781	916470	1053412	18	10	26
20	0.7	0.2	963506	946477	977952	15	13	17
20	0.7	0.3	1140490	1109901	1172691	36	33	40
20	0.7	0.4	1130785	1058341	1172813	35	27	40
20	0.8	0.1	960241	890733	1033754	15	7	24
20	0.8	0.2	972765	957048	986808	16	14	18
20	0.8	0.3	1119419	1071474	1187372	34	28	42
20	0.8	0.4	1179704	1155081	1219759	41	38	46
30	0.3	0.1	1139635	1001875	1241562	36	20	48
30	0.3	0.2	1152686	1083372	1265615	38	30	51
30	0.3	0.3	1030367	1007894	1059384	23	21	27
30	0.3	0.4	1030559	1007881	1045510	23	21	25
30	0.4	0.1	1146165	1095870	1177275	37	31	41
30	0.4	0.2	973477	957410	993663	16	14	19
30	0.4	0.3	1004205	918412	1065911	20	10	27
30	0.4	0.4	1049480	1025148	1073372	25	23	28
30	0.5	0.1	988442	920536	1093466	18	10	31
30	0.5	0.2	912209	874167	937494	9	5	12
30	0.5	0.3	992289	983664	1004447	19	18	20
30	0.5	0.4	1062028	1029153	1119280	27	23	34
30	0.6	0.1	914281	869828	960654	9	4	15
30	0.6	0.2	934247	881635	964263	12	5	15
30	0.6	0.3	1089640	1028291	1183454	30	23	42
30	0.6	0.4	1046810	1031038	1077085	25	23	29
30	0.7	0.1	884501	836246	922994	6	0	10

30	0.7	0.2	935785	912850	958522	12	9	15
30	0.7	0.3	1055828	1035755	1075661	26	24	29
30	0.7	0.4	1094731	1079032	1120113	31	29	34
30	0.8	0.1	911985	849614	1008825	9	2	21
30	0.8	0.2	1006193	894812	1096935	20	7	31
30	0.8	0.3	1179763	1138359	1219660	41	36	46
30	0.8	0.4	1084501	1024351	1150099	30	22	38
40	0.3	0.1	1204329	1130071	1272298	44	35	52
40	0.3	0.2	1100712	1057270	1132799	32	26	35
40	0.3	0.3	992959	955337	1024137	19	14	22
40	0.3	0.4	1024226	999109	1040777	22	19	24
40	0.4	0.1	1151441	1083750	1195722	38	30	43
40	0.4	0.2	914126	903853	926541	9	8	11
40	0.4	0.3	1051239	962969	1132729	26	15	35
40	0.4	0.4	1020492	1014325	1026601	22	21	23
40	0.5	0.1	1004141	904682	1154070	20	8	38
40	0.5	0.2	951533	907054	986386	14	8	18
40	0.5	0.3	990919	974260	1018163	18	17	22
40	0.5	0.4	1082812	999009	1132135	29	19	35
40	0.6	0.1	896280	852659	953725	7	2	14
40	0.6	0.2	967978	940032	992509	16	12	19
40	0.6	0.3	990707	955607	1023849	18	14	22
40	0.6	0.4	1098359	1032247	1147182	31	23	37
40	0.7	0.1	992998	944927	1030706	19	13	23
40	0.7	0.2	955144	879215	1028212	14	5	23
40	0.7	0.3	1001708	964766	1061256	20	15	27
40	0.7	0.4	1121322	1064524	1162476	34	27	39
40	0.8	0.1	875496	853544	893715	5	2	7
40	0.8	0.2	982013	943254	1002717	17	13	20
40	0.8	0.3	1088091	983783	1140473	30	18	36
40	0.8	0.4	1121962	1093856	1163692	34	31	39

Anhang 5: Q Parameter Messung

Es wurden mit ALG5 Messungen für Verschiedene Q Werte durchgeführt. Q gibt die Wahrscheinlichkeit an, mit der sich eine Ameise bei ihrer aktuellen Entscheidung fest für den Pfad mit dem höchsten Wert aus Pheromon und Heuristik entscheidet. Die anderen Parameter waren dabei fest vergeben (p = 0.7, a = 0.1, K = 30). Die letzten drei Spalten zeigen die Abweichung der Kosten mit niedrigstem Wert (LB = 824366) berechnet via: (C – LB) / LB * 100%

Q	C ø	C min	C max	C ø LB	C min LB	C max LB
0	972378	903393	1064133	18	10	29
0,05	983312	917150	1073142	19	11	30
0,1	956592	922190	988513	16	12	20
0,2	919155	889210	957955	11	8	16
0,3	906905	866429	938521	10	5	14
0,4	897733	871124	913620	9	6	11
0,5	932807	893961	969314	13	8	18
0,6	845120	824366	865799	3	0	5
0,7	884468	882624	887073	7	7	8
0,8	886363	876503	903846	8	6	10
0,9	922489	916050	925782	12	11	12

Anhang 6: Algorithmus Vergleichsmessung

Es werden die Algorithmen 4, 5 und 6 aus der Arbeit miteinander verglichen, dazu wurden jeweils 10 Messungen jedes Algorithmus mit den festgelegten Parametern (N = 80, p = 0.7, a = 0.1, b = 1 – a, K = 30, $\Delta\tau_{ij} = 1$, $\tau_0 = 1$, Q = 0.6) durchgeführt. Zum Vergleich die Kosten, welche der Greedy-Algorithmus ergeben hat. Die letzten drei Spalten zeigen die Abweichung der Kosten mit niedrigstem Wert (LB = 799606) berechnet via: (C – LB) / LB * 100%

ALG	C ø	C min	C max	C ø LB	C min LB	C max LB
ALG4	1026604	979762	1072072	28	23	34
ALG5	923605	870480	976826	16	9	22
ALG6	819243	799606	845438	2	0	6
greedy	1604531			23		

Anhang 7: Algorithmus Vergleichsmessung mit lokaler Suche

Vergleich der Kosten (Durchschnitt, minimale und maximale Kosten auf 10 Durchläufe) von Algorithmus 6 mit und ohne lokaler Suche. Die letzten drei Spalten zeigen die Abweichung der Kosten mit niedrigstem Wert (LB = 783824) berechnet via: (C – LB) / LB * 100%

ALG	C ø	C min	C max	C ø LB	C min LB	C max LB
ALG6	819243	799606	845438	5	2	8
ALG6LS	808633	783824	856600	3	0	9

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

_____	_____	_____
Ort	Datum	Unterschrift