Fachhochschul-Bachelorstudiengang
SOFTWARE ENGINEERING
A-4232 Hagenberg, Austria

Routing in der Logistik

Praktische
Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science in Engineering

Eingereicht von

David Baumgartner

Begutachtet von Prof.(FH) Priv.-Doz. DI Dr. Stephan Dreiseitl

Hagenberg, Juni 2017

Inhaltsverzeichnis

Kurzfassung

Routing ist allgegenwärtig in Form von Navigationssystemen wie zum Beispiel Google Maps. Diese Navigationslösungen besitzen nur die Fähigkeit einfache *point-to-point* Routen zu finden. Dabei können Zwischenstopps vorhanden sein, aber mit einer fixen Reihenfolge. Routing und im speziellen Transportlogistik beinhalten mehr als nur Navigationsoptimierung. So müssen im Falle von UPS Paketauslieferungen optimiert werden, um Einsparungen zu ermöglichen.
Im Rahmen dieser Arbeit wurden die Problemstellungen wie *Traveling Salesman Problem* und weitere erarbeitet und erklärt. Des Weiteren wurde auf mögliche Lösungsansätze für solche Probleme eingegangen. Die Abläufe solcher Lösungsmöglichkeiten stellen einen weiteren Inhalt dar. Im Speziellen wird genauer auf den *Savings-Algorithmus* eingegangen und eine Beispielimplementierung mit Zeitfenster durchgeführt.

Abstract

Routing is a daily demand and is used within our navigation-systems like *Google Maps*. The navigation solution contains the functionality to find *point-to-point* routes with fixed sequence stops. The field of transport systems/logistics requires a more sophisticated stop optimiziation. As an example the package delivery service UPS uses stop und routing optimiziation to save millions of costs.
This thesis contains the basics about the *Traveling Salesman Problem* and its offsprings. Additionally it includes the basics to the solving approaches for these problems and how they work. There is an implementation of the *Savings-Algorithm* with the time window extension.

Kapitel 1

Einleitung

1.1 Motivation

Der Gütertransport existiert seit Waren transportiert werden. Nicht nur der Personenverkehr nimmt immer mehr zu, sondern auch der Gütertransport. Es werden immer mehr Güter weltweit erzeugt, dafür müssen zum Abnehmer meist weite Strecken zurückgelegt werden. Dies führt dazu, dass Straßen ausgebaut und Hauptverkehrsrouten adaptiert werden müssen. Vom Jahr 2014 auf 2015 nahm zum Beispiel das Güteraufkommen in Deutschland um $1,9\,\%$ zu.[1] Damit die Kosten konstanter bleiben, müssen Transportrouten optimierter erledigt werden. Dadurch werden Kosteneinsparungen erzielt. So wird auch der Umwelt etwas Gutes getan.

1.2 Problemstellung

Ein Problem in der Logistik stellt die Optimierung von Routen dar. Optimierungen von Routen bieten eine Möglichkeit, um effektiver Kunden zu erreichen. Dabei kann nicht nur Zeit gespart werden, sondern auch Fahrzeuge besser ausgenützt werden. Zum Beispiel erspart sich UPS durch intelligente Optimierung Millionen an Kilometern. Der Paketdienstleister UPS entwickelte sich unter anderem auch zu einem Technikunternehmen mit tausenden Servern. Diese lösen und analysieren ununterbrochen Routen, damit am Morgen sofort optimierte Touren zur Verfügung stehen. Das dafür entwickelte System nennt sich *ORION* und analysiert in Realzeit anhand der Verkehrslage. Dafür wurde seit 2003 an diesem System gearbeitet und erst im Jahr 2012 mit dem Betatesting begonnen.[2] Solche Probleme betreffen nicht nur Auslieferungsdienste wie UPS, sondern auch lokale Produzenten mit eigener Auslieferung. Diese sind meistens zusätzlich auf Zeitfenster bei

[1] Bundesverband Güterkraftverkehr; http://bgl-ev.de

[2] UPS ORION; https://www.pressroom.ups.com

den Kunden angewiesen, um einen Mehrwert für die Kunden zu bieten. Mit diesen Zeitfenstern wird die Problemstellung um einiges komplexer und schwerer lösbar.

1.3 Zielsetzung

Diese Arbeit soll die Grundlagen für Optimierungen in der Logistik näher bringen und einen Einstieg darstellen. Hierbei wird auch auf mögliche Kostenberechnungen und Kostenmatrizen eingegangen und Probleme aufgezeigt. Am Ende wird das Gebiet der Zeitfenster anhand eines Beispiels aufgearbeitet und eine mögliche Lösung präsentiert.

Kapitel 2

IT|PRO

2.1 Allgemein

Das Unternehmen IT|PRO wurde 1999 mit der Vision *Lösungen für Menschen* gegründet. Im Unternehmen werden aktuelle Technologien eingesetzt, um bestmögliche und wirtschaftliche Lösungen zu kreieren. Diese Technologien reichen vom *.NET Framework* bis hin zum *Angular 2/4 Framework* und zusätzlicher Hardwareentwicklung. Das Unternehmen bietet dabei Gesamtlösungen sowie spezielle Kundenlösungen und Standardprodukte. Es wird ein Komplettservice angeboten mit Beratung, Analyse der bestehenden Geschäftsprozesse mit Projektentwicklung und Inbetriebnahme bis hin zum Support. Im Hintergrund stehen engagierte Entwickler und Partnerbetriebe, damit eine Lösung aus einer Hand geliefert werden kann. Aktuell besteht das Team aus den beiden Geschäftsführern und rund 20 Mitarbeitern/Innen.

2.2 Arbeitsgebiet

Ein Teil der eigenen Softwarelandschaft im Unternehmen IT|PRO widmet sich der Routenplanung und Optimierung. Diese Applikation wurde in einem Softwareprojekt mit der FH-Hagenberg auf eine neue Version gebracht und dabei weitgehend neu implementiert. Im Rahmen des Praktikums wurde eine weitere Hauptumstellung durchgeführt. So wurde das Karten-Framework *Bing Maps* durch das freie Framework *Leaflet* ersetzt. Des Weiteren wurde das letzte fehlende Feature, ein Disponierungstool, aus der vorhergehenden Version neu implementiert. Eine weitere Aufgabe im Praktikum war die Überarbeitung und teilweise Neuimplementierung des Routing-Algorithmus. Dieser beruht auf einem Savings-Algorithmus mit der Erweiterung um Zeitfenster. In Kapitel 3 werden die Grundlagen im Bereich der Logistik erklärt und erläutert.

Kapitel 3

Logistik

3.1 Einsatzgebiet & Problemstellung

Logistik gehört wie die Datenanalyse im Bereich von BigData zu den Problemgebieten, welche sich nicht so einfach lösen lassen. Grundsätzlich sind sie exakt mit genügend Zeit lösbar, aber nicht in der benötigten oder vorhandenen Zeit. Dieses Problem wirkt sich im 21. Jahrhundert stark aus, da immer mehr Güter transponiert werden und diese über die gesamte Welt. Am Beispiel des Paketdienstes UPS lässt sich dies erkennen. Im Jahr 2006 war das tägliche Zustellvolumen von Paketen und Dokumenten bei 14.1 Millionen, im Jahr 2014 bereits bei 18.0 Millionen[1]. Dies entspricht einer Steigerung von $\sim 27\,\%$, bei einer gleichzeitigen Erweiterung der Fahrzeugflotte um $\sim 13\,\%$. Dies betrifft nicht nur Logistikunternehmen wie UPS, sondern auch zum Beispiel Brauereien, welche die Auslieferung ihrer Produkte nicht komplett ausgelagert haben.
Logistik beinhaltet mehr als nur Auslieferungen:

- Planung: Ein Disponent erstellt Listen an Aufträgen, die erledigt werden müssen;
- Organisation: Die Auftragslisten müssen einem Fahrzeug zugeteilt werden, wobei überprüft werden muss, ob die maximale Kapazität des Fahrzeugs nicht überschritten wird;
- Steuerung: Die geplante Route kann vom Disponenten manuell oder durch ein Informationssystem automatisiert optimiert werden;
- Abwicklung: Erfolgt durch den Fahrer mit dem zugeteilten Fahrzeug;
- Kontrolle: Durch Analyse der Touren und Fahrten kann ein Informationsrückfluss erfolgen, um den gesamten Prozess zu verbessern;

Ein weiteres Problem betrifft die Anzahl der Stopps. Wie in Tabelle 3.1 ersichtlich, steigt die Anzahl der möglichen Routen mit der Anzahl der Stopps.

[1] UPS - Weltweit, https://www.ups.com/content/at/de/about/facts/worldwide.html

Anzahl der Haltepunkte	Anzahl der möglichen Routen	Berechnungszeit in Sekunden bei 30 Routen/Sekunde
1	1	0.030
2	2	0.067
3	6	0.200
4	24	0.800
5	120	4.000
6	720	24.000
7	5.040	168.000
8	40.320	1344.000
9	362.880	12096.000
10	3.628.800	120960.000
11	39.916.800	1330560.000
12	479.001.600	15966720.000
13	6.227.020.800	207567360.000

Tabelle 3.1: Aufschlüsselung der Anzahl an möglichen Routen, gegeben durch die Berechnung der Fakultät der Anzahl an Knoten

Eine Route besteht hierbei aus einem Startdepot am Anfang und einem Enddepot am Ende. Zwischen diesen Punkten befinden sich mindestens ein Haltepunkt bis n viele. Diese Anzahl lässt sich durch die Zahl der Stopps und dessen Fakultät berechnen. Daraus lässt sich ableiten, dass bis zu einem Grad alle Möglichkeiten testbar sind, abhängig von der Rechenleistung. Angenommen ein Rechner berechnet 30 Möglichkeiten pro Sekunde, so würde man bei 8 Stopps eine Zeit von 1344 Sekunden bzw. 22, 4 Minuten benötigen.

3.1.1 Graph

Graphen im Sinne der Logistik sind anders definiert als im Bereich der neuronalen Netzwerke. In der Graphentheorie ist ein Graph ein System an Knoten und Kanten. Dabei kann dieser als Paar $G := (V, E)$ interpretiert werden, welches aus zwei Mengen besteht. Eine Menge mit den Knoten V und die zweite mit den Kanten E zwischen den Knoten. Im Sinne der Graphentheorie kann entlang der Kanten durch den Graph navigiert werden.
Eine Spezialisierung bilden gerichtete Graphen, bei welchen die Kanten zusätzlich eine Richtung beinhalten. Des Weiteren existieren noch gewichtete Graphen mit Gewichtungen an den Kanten. Sie beschreiben, wie groß die Kosten sind, wenn die Kante zum Navigieren verwendet wird. Ein Beispiel dazu befindet sich in der Abbildung 3.1, mit 4 Knoten und einem voll vernetzten System zwischen diesen. Die Spezialisierungen der einzelnen Graphen-Typen können kombiniert werden, sowie um weitere Bedürfnisse

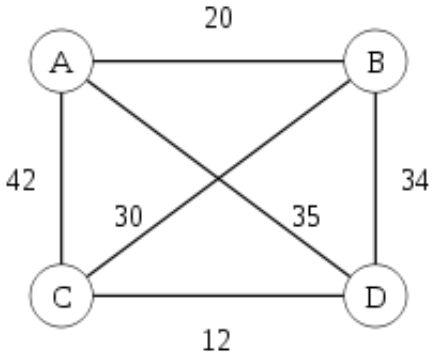

Abbildung 3.1: Beispiel für einen Graphen mit gewichteten Kanten und 4 Knoten als Kunden; Die Kosten sind dabei als ganze Zahlen an den Kanten angegeben; Bild: www.wikipedia.org

abgeändert werden [10].
Um durch einen Graphen zu navigieren existieren Algorithmen, welche zum Beispiel die kürzeste Route von einem Punkt zu allen anderen im Graphen finden. Hierzu zählt zum Beispiel der Dijkstra-Algorithmus, der als Standardalgorithmus im Gebiet der kürzesten Routenfindung eingesetzt wird.

3.1.2 Traveling Salesman Problem (TSP)

Das TSP oder auch unter dem Begriff *Vehicel Routing Problem (VRP)* bekannt, stellt ein Grundproblem der Logistik dar. Bei dieser Problemstellung sollen geografische Positionen einmal angefahren werden und dies mit dem geringsten Aufwand. Dies lässt sich in der Abbildung 3.2 erkennen. Dabei werden die größten Städte Deutschlands angefahren, der Start- und Zielpunkt sind dabei identisch. Dieses Problem kann mit einer Laufzeitkomplexität von $O((n-1)!/2)$ exakt gelöst werden, wobei mit zunehmender Größe des Faktors n, die Berechnungszeit wesentlich größer wird. Aus diesem Grund wurden heuristische Algorithmen entwickelt, welche eine mögliche Lösung finden, aber sehr wahrscheinlich nicht die effizienteste Route. Heuristische Lösungen werden eingesetzt, da sie zeit- und ressourcensparender sind und dabei verwendbare Lösungen liefern [6, 10].
Weiters werden einige spezialisierte Formen des VRP kurz erklärt und beschrieben.

Capacity Vehicle Routing Problem (CVRP): In dieser Erweiterung von VRP spielt besonders die Kapazität der Fahrzeuge eine Rolle, denn diese sollte so effizient wie möglich ausgenützt werden. Hierbei starten alle Fahrzeuge vom selben Depot, alle Kunden sind vorab bekannt. In diesem

Abbildung 3.2: Beispiel für eine Route durch Deutschland; Bild: www.wikipedia.org

Fall besitzen alle Fahrzeuge dieselbe Kapazität, welche in der Route nicht überschritten werden darf [6].
Lösungsalgorithmen für CVRP:

- Savings-Algorithmus: wird in Sektion 3.2.2 erklärt
- Der Sweep-Algorithmus verwendet ein *Cluster-first* Vorgehen. Hierbei werden Stopps einem Fahrzeug solange hinzugefügt, bis dessen Kapazität oder die maximale Routenlänge erreicht wird. Die Zuteilung zu einem Fahrzeug wird durch Polarkoordinaten festgestellt und dessen Winkel zum aktuell ausgewählten Punkt. Im Anschluss wird eine TSP Optimierung auf den erstellten Cluster an Haltepunkten ausgeführt und so eine optimierte Route generiert.
- Ein *Route-first* Vorgehen, wie von Beasley 1983 beschrieben, beginnt mit einer optimierten Route ohne Berücksichtigung der Nebenbedingungen. Im Anschluss wird diese Route auf Fahrzeuge aufgeteilt, sodass die Nebenbedingungen eingehalten werden.

Periodic Vehicle Routing Problem (PVRP): PVRP stellt wie CVRP eine Erweiterung des Basisproblems dar. In diesem Fall sollen Kunden nicht einmal, sondern mehrmals angefahren werden. Dieses Problem tritt zum Beispiel bei einem Hausbau mit mehreren Betonlieferungen auf, über eine längere Zeit abhängig von der Nachfrage [6].

Vehicle Routing Problem with Backhauls (VRPB): Das VRPB erweitert das CVRP um die Möglichkeit, Kunden nicht nur zu beliefern, sondern auch etwas entgegenzunehmen. Dabei entstehen zwei Teilmengen. Kunden die beliefert werden und Kunden von welchen abgeholt werden soll. Im Grunde erweitert sich das Basismodell um eine Vorrangregel, bei welcher zu beliefernde Kunden bevorzugt werden. Ansonsten könnte der Fall eintreten, dass ein Fahrzeug keine Zusatzladung mehr aufnehmen kann [10].

Vehicle Routing Problem with Time Windows (VRPTW): Das VRPTW erweitert ebenfalls das CVRP, sodass Kunden nicht zu jeder Zeit angefahren werden können. Dieses Problem ist aus der Praxis heraus gewachsen, denn manche Kunden und Lager sind nur zu bestimmten Zeiten anzutreffen. Zusätzlich könnten Straßensperren berücksichtigt werden, wie zum Beispiel eine Innenstadt mit Zustellungserlaubnis am Vormittag. Das Basismodell muss hierbei um Zeitintervalle erweitert werden, weiters müssen die Abfahrtszeiten vom Depot, Fahrzeiten für alle Teilstrecken und Servicezeit bei jedem Kunden vorhanden sein. Im Gesamten muss festgelegt werden wie optimiert werden soll, denn es besteht die Möglichkeit, dass Fahrzeuge warten müssen, bis der Kunde anzutreffen ist. Im Falle einer Wartezeit könnte zuerst ein anderer Kunde angefahren werden. Dies bedeutet, dass wahrscheinlich eine längere Strecke zurückgelegt werden muss [10].
Lösungsalgorithmen für VRPTW:

- Savings-Algorithmus
- Route-first, Cluster-second (Solomen 1986)
- Sequentielle Einfügeheuristik (Solomon 1987)
- Zeitorientierte Nearest-Neighbor Heuristik
- Tabu-Search
- Evolutionäre Algorithmen

Pick-up-and-delivery Problem (PDP): Das PDP stellt eine zusätzliche Spezialisierung des VRP dar. In diesem Fall müssen Abholungen und Lieferungen in einer Route erledigt werden. Dabei ist die Kapazität der Fahrzeuge sehr zu berücksichtigen, da diese erst wieder zusätzliche Ladungen aufnehmen können, wenn genügend Kapazität dafür vorhanden ist. Außerdem muss berücksichtigt werden, dass eine Ladungsaufnahme in dieser

Route später auch zugestellt wird. Diese Ladungen dürfen nicht als Auslieferung für eine weitere Route übernommen werden.

Diese Beschreibungen stellen nur eine Teilmenge der Problemdomänen dar, weshalb für genauere Informationen Fachlektüre hinzugezogen werden sollte. Des Weiteren existieren Meta-Heuristische Algorithmen, welche eine geringe Laufzeit besitzen, dafür kein exaktes Ergebnis liefern, sondern nur eine Annäherung.

3.2 Lösungsverfahren

3.2.1 Exakte Verfahren

Exakte Verfahren liefern die beste Lösung, aber nicht ohne Nachteile. Einer dieser Nachteile ist, dass die benötigte Zeit mit jedem weiteren Knoten im System stark ansteigt, da die zu überprüfenden Möglichkeiten steigen.

Branch-and-Cut: *Branch-and-Cut* vereint zwei Verfahren, welche ebenfalls im Gebiet der Logistik eingesetzt werden: Das Schnittebenenverfahren und das *Branch-and-Bound* Verfahren. Ziel dieser ganzzahligen linearen Optimierung stellt eine lineare Zielfunktion dar, welche gesucht wird. Diese Funktion wird mit Hilfe von linearen Ungleichungen gesucht, welche eine Menge an Punkten umschließen. Durch das Schnittebenenverfahren werden weitere Ungleichungen in das System eingefügt, welche von allen zulässigen Punkten erfüllt werden. Durch ein weiteres Lösen des Systems muss eine neue Lösung entstehen, welche näher am gesuchten Optimum liegt. Das *Branch-and-Bound* wird im Anschluss ausgeführt, wenn keine Schnittebenen mehr gefunden werden. Die genauere Beschreibung dieses Algorithmus befindet sich in der Sektion 3.2.2, da dieses Verfahren in der Fachliteratur den Übergruppen *exakte Verfahren* und *heuristische Verfahren* zugeordnet wird.

3.2.2 Heuristik

Heuristik bedeutet, mit wenig Wissen sowie mit begrenzter Zeit eine wahrscheinliche Lösung zu liefern. Heuristiken werden oft im Gebiet der Optimierungen eingesetzt. Zum exakten Lösen eines Logistik-Problems wird sehr viel Zeit und Speicher benötigt. Deshalb werden hier häufig Heuristiken eingesetzt. Ein Vorteil der Heuristik ist es, dass in kurzer Zeit gute Lösungen für schwierige Probleme gefunden können. Die Nachteile dieser Methoden sind aber nicht zu vernachlässigen, sie geben nämlich:

- keine Garantie für die optimale Lösung;
- keine Garantie für eine Lösung und für die Güte einer Lösung;

Branch-and-Bound: Im Grunde wird bei diesem Algorithmus ein Baum im Sinne der Informationstechnologie aufgebaut, in welchem anhand der Möglichkeiten gesucht und verglichen wird. Dabei werden Schwellenwerte für die Kosten festgelegt, welche bei einer Route nicht über- und unterschritten werden dürfen.

- *Branchen* (Verzweigung) ist der erste Schritt in diesem Algorithmus. Hierbei wird das Problem in mehrere Teilprobleme aufgeteilt und in einer Baumstruktur abgelegt.
- *Bounden* (Beschränkung) repräsentiert den zweiten Schritt, in welchem die Lösungsmenge beschränkt wird. Dabei wird überprüft, welche Zweige suboptimal sind und als mögliche Lösungen verworfen werden.

Eine Lösung wird verworfen, wenn das Gewicht eines *1-tree* größer ist als das Gewicht einer zuvor gefundenen möglichen Lösung. Ein *1-tree* beschreibt dabei einen Weg von der Hauptwurzel bis zu einem Endblatt. Diese Art der Routenfindung liefert rasch brauchbare Ergebnisse [9].

Nearest-Neighbor (NN): Die NN-Heuristik stellt ein mögliches Eröffnungsverfahren dar. Diese liefert zwar eine Approximation der Lösungen, aber meist nicht sehr gute Resultate. Der Algorithmus kann dazu verwendet werden, um eine Beispiellösung zu generieren. Diese Lösung kann als Ausgangszustand für Verbesserungsheuristiken eingesetzt werden, welche selbst keine Grundlösung finden würden. Der Grundalgorithmus wählt zufällig einen Knoten aus dem System und sucht sich anhand diverser Kriterien einen weiteren Knoten, welcher als *Nearest-Neigbor* bezeichnet wird. Von diesem ausgehend wird ein weiterer Knoten zur Route hinzugefügt, bis diese vollständig ist. Ein Nachteil dieses Algorithmus lässt sich am Ende erkennen, wenn die Route vollständig ist. Start und Endpunkt liegen oft sehr weit auseinander. Dieses Verhalten ist meist nicht gewünscht, da eine Rundreise erwartet wird und nicht eine Route im Sinne einer Linie und einem langen Rückweg [8].
Von diesem Algorithmus aus existieren noch einige weitere Spezialisierungen, wie zum Beispiel der *Variable Neighbourhood Search* Algorithmus. Dieser nützt den Vergleich von lokalen Optima zu globalen Optima, um effizientere Routen zu finden.

Savings-Algorithmus (SA): Der SA von Clarke und Wright gehört nicht zur Kategorie der Eröffnungsverfahren, sondern zu den Optimierungsalgorithmen. Bei diesem wird hauptsächlich mit einem Saving gearbeitet. Dieses Saving repräsentiert, wie viel gespart werden kann, wenn der Kunde B nach dem Kunden A angefahren wird und somit zwischen diesen Kunden nicht zum Depot zurückgekehrt werden muss. Dabei spielt die Berechnung des

Savings aus den Kosten eine wichtige Rolle, auf welche später noch genauer eingegangen wird.
Der Algorithmus beruht darauf, in jedem Schritt eine Route zu vergrößern und dabei einige andere zu entfernen. Der grundsätzliche Ablauf sieht wie folgt aus:

1. Zu Beginn müssen Werte zur Berechnung der Ersparnisse initialisiert werden, wie eine Zeit- oder Distanzmatrix. In diesem Schritt wird aus jedem Knoten/Kunde eine triviale Route erstellt, bestehend aus *Depot - Knoten - Depot*.
2. Im nächsten Schritt wird jede Route mit jeder anderen Route gepaart und der Savings-Wert berechnet. In diesem Schritt muss auch überprüft werden, ob die Kombinationen die Nebenbedingungen einhalten. Sollte dies nicht der Fall sein, muss diese Möglichkeit verworfen werden.
3. Die Kombination von zwei Routen mit dem höchsten Saving, wird im Anschluss zu einer neuen Route vereint.
4. Nach der Kombination müssen alle Möglichkeiten entfernt werden, welche die zweite Teilroute beinhalten. Zusätzlich müssen alle Teilrouten mit dem ersten Teil durch die neue Route ersetzt werden.
5. Infolgedessen wird bei Punkt 2 fortgesetzt, wenn sich noch Kombinationen an Routen in der Savingsliste befinden. Sollte dies nicht der Fall sein, so wurde eine optimierte Route gefunden oder mehrere Teilrouten, welche nicht mehr kombiniert werden können.

Sweep-Algorithmus: Der Sweep-Algorithmus verwendet wie zuvor schon erwähnt ein *Cluster-first* Vorgehen. Dieser verwendet ein geometrisches Verfahren zum Erstellen des Clusters, weshalb dieser nur in planaren Instanzen angewendet werden kann. Jeder Kunde im System wird in einem Polarkoordinatensystem repräsentiert. Jeder besitzt einen Winkel und einen direkten Abstand zum zentralen Depot.

- Zu Beginn wird ein beliebiger Knoten/Kunde als erster Haltepunkt in der Route ausgewählt.
- Im Anschluss wird zu diesem Ausgangspunkt ein weiterer Kunde hinzugefügt und zwar dieser mit dem geringsten Winkelabstand zum letzten Punkt in der Route.
- Dieser Prozess wird solange durchgeführt, bis eine Route eine maximale Anzahl an Punkten erreicht hat oder ein Fahrzeug keine Kapazität mehr zur Verfügung hat.
- Damit wurde ein Cluster an Stopps für ein Fahrzeug erstellt, welcher im einem zweiten Schritt zu einer optimierten Route entwickelt werden muss.

Somit gehört auch dieser Algorithmus zur Kategorie der Eröffnungsverfahren [8].

3.2.3 Meta-Heuristiken

Meta-Heuristik wird in der Informatik als Bezeichnung verwendet, wenn eine Lösung näherungsweise erzeugt wird. Diese sind nicht stark problemspezifisch orientiert, im Gegensatz zu Heuristiken. Diese Art von Algorithmen wird im Bereich der Logistik deshalb eingesetzt, weil sie lokalen Optima in schwereren und größeren Problemen entwischen können und eher zu einem globalen Optima tendieren oder konvergieren.

Tabu-Search (TS): Der *Tabu-Search* baut auf keiner bisher beschriebenen Methode auf. Im Unterschied zu anderen Algorithmen sind beim TS auch ungültige Lösungen zugelassen, welche die Nebenbedingungen verletzen. Es muss dafür eine Kostenfunktion existieren, mit welcher die Kosten einer Lösung festgestellt werden können. Für jede gefundene Lösung wird evaluiert, ob dies eine verwendbare Lösung ist oder nicht. Hierbei fließen die einzelnen Nebenbedingungen, mit einem hinzumultiplizierten Faktor, eine Rolle. So kann festgestellt werden, ob es sich um eine bessere Lösung handelt oder nicht. Eine Kernfunktionalität stellt die Tabu-Liste dar. In dieser werden Unmöglichkeiten gespeichert, welche nicht durchgeführt werden dürfen, solange sie sich in der Liste befinden. Zum Beispiel darf von einem Ort aus nicht nach Osten zum nächsten gefahren werden, da dieser Zug in der Iteration vorher durchgeführt worden ist.
Der Hauptalgorithmus besteht aus einer Hauptschleife, welche an ein Abbruchkriterium geknüpft ist. Zu Beginn wird dabei eine mögliche Lösung initialisiert und als aktuell beste Lösung angenommen. Im Anschluss werden weitere mögliche Lösungen erzeugt und die Beste daraus ausgewählt. Diese alternative Lösung wird mit der aktuell besten Lösung verglichen und als neue Beste definiert, wenn dies der Fall ist. Daraufhin wird die Tabu-Liste aktualisiert. Dieser Ablauf wird solange durchgeführt, bis das Abbruchkriterium eintritt.
Bei diesem Algorithmus wird nicht garantiert, dass eine gültige Lösung gefunden wird. So kann aber eine Lösung entstehen, welche weit geringere Kosten mit sich bringt als eine Lösung ohne Verletzung der Nebenbedingungen. Im Falle einer Applikation muss durch die Evaluierungsfunktion definiert sein, in wie weit eine Nichteinhaltung toleriert wird oder nicht [8].

Simulated Annealing (SA): SA ist für sich im Detail kein Algorithmus, sondern bezeichnet eine Klasse von Algorithmen, welche versuchen global zu optimieren. Die Idee und die Bezeichnung stammen hierbei aus der Härtung von Stahl. In der Stahlproduktion wird dieser *erwärmt* und wieder *abgekühlt*, was als *härten* bezeichnet wird. Dieser Prozess des Abkühlens führt

dazu, dass sich die Teilchen wie in einer Gitterstruktur anordnen. Zu Beginn kann von einer stochastischen Suche gesprochen werden und zum Ende hin von einer deterministischen Suche. Zum Ende des Prozesses werden die Schritte immer kleiner, wie bei der Härtung kühlen dabei die letzten Temperaturen sehr langsam ab. Im Gesamten wird eine global optimale Konfiguration gefunden, in dem dieser Prozess simuliert wird. Am Anfang wird deshalb von einer stochastischen Suche gesprochen, da das System Möglichkeiten annimmt, welche zu einer Verschlechterung des Ergebnisses führen. Wird die sogenannte Temperatur geringer, minimiert sich die Akzeptanz für Möglichkeiten, was wiederum deterministisch ist. Der stochastische Anteil ermöglicht dem lokalen Minima zu entkommen und dem globalen Minima sich anzunähern. Der Grundalgorithmus ähnelt dabei sehr dem *Tabu-Search* mit der Hauptschleife. Beim SA wird solange optimiert, bis das System einen erstarrten Zustand erreicht [5].

Ant Colony Optimierung (ACO): ACO repräsentiert eine Simulation von sozialen Interaktionen in der Natur. So wurde für diesen Algorithmus das Verhalten und die Interaktionen von Ameisen, Bienen und weiteren Insekten analysiert. Jedes Individuum dieser Insekten erledigt eine eigene Aufgabe, unabhängig von anderen in der Kolonie. Ein funktionierendes System kommt dadurch zustande, dass die Arbeiten jedes einzelnen Individuums auf den Arbeiten eines anderen aufbauen oder mit anderen verbunden sind. Durch die Kooperation in der Kolonie, entsteht die Möglichkeit, komplexe Probleme zu lösen und Überlebensstrategien auszuführen [1, 7].
Damit dieser Algorithmus eingesetzt werden kann, müssen einige Grundvoraussetzungen erfüllt sein:

- Eine Strategie zum Überprüfen einer gültigen Lösung, damit nur solche akzeptiert werden;
- Eine heuristische Funktion η zum Messen der Qualitäten der Teile, welche sich noch nicht in der Teillösung befinden;
- Eine Funktion zum Aktualisieren der Pheromone τ, welche die Wertigkeit der Kanten repräsentiert;
- Eine Funktion ϕ, welche auf der heuristischen Funktion η und den Pheromonen aus τ basiert und iterativ mögliche Lösungen konstruiert;

Der grobe Ablauf des Grundalgorithmus:

- Zu Beginn wird die Ameise an einem Startpunkt ausgesetzt;
- Diese besitzt nun die Möglichkeit sich mit einer bestimmte Anzahl von Zügen durch das Netzwerk zu bewegen unter Berücksichtigung der Funktion ϕ; dabei wird eine Tabu-Liste für diese Ameise erstellt;
- Im Anschluss wird die Tour der Ameise bewertet und die Pheromone der Kanten berechnet; eine weitere Ameise wird ausgesetzt bis die maximal Anzahl der Ameisen erreicht ist;

- Nachdem alle Ameisen einmal im Netzwerk waren, werden alle Pheromone an den Kanten aktualisiert;
- Diese entstandene Lösung wird nun mit der letzten verglichen und ein weiterer Durchlauf wird unter Berücksichtigung eines Abbruchkriteriums mit den neuen Pheromonen eingeleitet;

Evolutionäre Algorithmen (EA): EA sind wie SA kein Algorithmus, sondern ein Überbegriff für eine Klasse von Algorithmen und Methoden. Das Prinzip dieser Algorithmen beruht auf dem Vorgang einer biologischen Evolution. Die Natur einer biologischen Evolution ist es, sich zu verbessern und zu überleben, was äquivalent zu einem Optimierungsprozess ist. In der Natur sind die Vorgänge der Optimierung bei einem Menschen sehr langsam und für ein einzelnes Individuum nicht wahrnehmbar. Dabei muss sich jede Generation aufs Neue behaupten und beweisen, dass sie mit der aktuellen Umgebung besser zurechtkommt. Diese Selektion geschieht ganz nach *Charles Darwins* Worten *survival of the fittest*, sodass der Angepassteste/Fitteste sich am stärksten verbreitet beziehungsweise vermehrt. Die Veränderungen entstehen dabei durch einfache Fortpflanzung im Sinne einer Rekombination und durch Mutationen. So wird bei der Mutation nur ein Individuum benötigt, welches verändert wird und bei der Rekombination zwei.
Ein EA besitzt genau solche Eigenschaften und versucht hier die Natur nachzuahmen. Im Sinne eines Algorithmus werden die Rekombination und Mutation schneller durchgeführt, sodass immer Mengen an potentiellen Lösungen zur Verfügung stehen. Alle Lösungen werden evaluiert und ihre Fitness bestimmt. Unbrauchbare Lösungen werden aussortiert, wenn sie nicht fit genug sind, bis eine geeignete Lösung gefunden wurde oder ein Abbruchkriterium wirksam wird [2, 4].
Der grundsätzliche Ablauf sieht wie folgt aus:

1. Zu Beginn muss eine Beispiellösung initialisiert und bewertet werden, um einen Ausgangszustand zu erlangen;
2. Im Anschluss wird eine Selektion durchgeführt;
3. Die Selektierten werden rekombiniert und so Nachkommen erzeugt;
4. Auf die gesamte Menge an Lösungen wird eine Mutation angewendet;
5. Zum Abschluss werden alle Lösungen bewertet und ihre Fitness wird festgestellt;
6. Sollte kein Abbruchkriterium erfüllt sein, so wird ab Punkt 2 wieder fortgesetzt;

In diesem Kapitel wurden grundlegende Eigenschaften in der Logistik erklärt und Problemgebiete aufgezeigt. Es wurden einige Lösungsansätze beschrieben, um deren Grundalgorithmus zu verstehen. Im nächsten Kapitel wird auf Kostenfunktionen eingegangen und Probleme darin aufgezeigt.

Kapitel 4

Kostenberechnung

Die Kostenfunktion stellt in jedem VRP und dessen Ablegern eine Kernkomponente dar. Diese muss an das Problem angepasst werden und ist somit problem- und zielspezifisch. Mit dieser Funktion wird definiert wie und was optimiert werden soll. Zum Beispiel kann diese so ausgelegt sein, dass die Strecke oder Fahrzeit minimal sein soll oder beides gemeinsam. Im weiteren Verlauf dieses Kapitels wird auf einige Eigenschaften der Kostenfunktion für den Savings-Algorithmus eingegangen.

4.1 Distanz-Basierend

Rein auf Distanz-Basierende-Kostenfunktionen bringen meist nicht das gewünschte Ergebnis zu Tage. Dies lässt sich auf die Einschränkung der vorhandenen Daten zurückführen. Alle Ergebnisse basieren dabei rein auf den Distanzen, welche gefahren werden. In diesem Fall könnte ein Kunde einige wenige Kilometer entfernt sein, direkt aber nur über einen Feldweg. Dies würde zu einer kurzen Distanz führen ohne die Beschaffenheit der Strecke und der dabei benötigten Zeit zu berücksichtigen. Sollte trotzdem nur die Distanz zur Verfügung stehen, so ist dies besser als überhaupt keine Informationen zu haben.
Die Berechnung für die Ersparnis im SA ist in der Gleichung 4.1 zu finden. Dabei stellen die Punkte A und B zwei Kunden dar und DE das Depot, von welchem aus losgefahren wird. Die Variable *dis* repräsentiert die Distanzmatrix, welche alle Distanzen zwischen den Punkten beinhaltet. Die Ersparnis ist hierbei definiert als das, was eingespart wird, wenn zwischen Kunde A und B nicht zum Depot zurückgekehrt, sondern direkt von A nach B gefahren wird [3].

$$sav_{a,b} := dis_{a,z} + dis_{z,b} - dis_{a,b} \tag{4.1}$$

In der Abbildung 4.1 ist ein einfaches Netzwerk mit Kunden und einem Depot abgebildet. Die Abkürzung AB steht in diesem Beispiel für Autobahn,

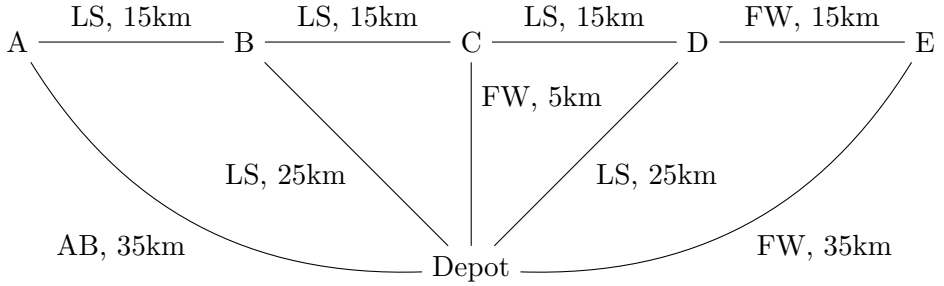

Abbildung 4.1: Ein einfaches Netzwerk mit 4 Kunden und einem Depot

LS für Landstraße und *FW* für Feldweg. Wie in dieser Abbildung festgestellt werden kann, existieren zwei Kanten mit einem Feldweg und eine mit einer Autobahn. An diesem Beispiel lässt sich erkennen, dass die Beschaffenheit des Fahrwegs ignoriert und vernachlässigt wird. In Tabelle 4.1 befinden sich die Ergebnisse der Ersparnisse zu Beginn des Savings-Algorithmus nach der Initialisierung. Im Gesamten lässt sich hier erkennen, dass lange Strecken bevorzugt werden. In diesem Fall würden zuerst die Routen *A* - *B* oder *D* - *E* zusammengefügt werden, abhängig von der Sortierung der Savings-Liste.

Route	Ersparnis
A $\leftrightarrow$ B	$35 + 25 - 15 = 45$
B $\leftrightarrow$ C	$25 + 5 - 15 = 15$
C $\leftrightarrow$ D	$5 + 25 - 15 = 15$
D $\leftrightarrow$ E	$25 + 35 - 15 = 45$

Tabelle 4.1: Aufschlüsselung der Ersparnisse anhand der Teilrouten zu Beginn der Optimierung

An diesem Beispiel lässt sich erkennen, dass die Beschaffenheit einer Straße ignoriert wird. Für den Algorithmus sieht dies so aus, als ob dies eine gute Kombination wäre. Im Grund stimmt dies, zumal die Ersparnis größer ist.

4.2 Zeit-Basierend

Zeit-Basierende-Kostenfunktionen beinhalten einen Vorteil gegenüber der Distanz-Basierten-Version, aber bergen auch Probleme. Der Vorteil der Zeitinformation liegt darin, dass die Beschaffenheit der Strecke versteckt mit einfließt. So wird auf einer Strecke mit Autobahn weniger Zeit benötigt, als auf derselben Distanz, aber mit Feldweg als Beschaffenheit.
Die Berechnung für die Ersparnis im Savings-Algorithmus ist in der Gleichung 4.2 zu finden. Dabei ist diese identisch mit der Gleichung aus 4.1 mit

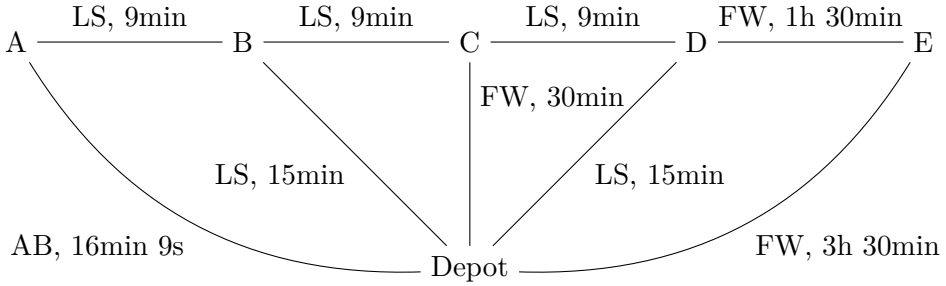

Abbildung 4.2: Derselbe Graph aus 4.1 mit 4 Kunden und einem Depot mit den benötigten Zeiten

dem Unterschied, dass hier eine Zeitmatrix *time* verwendet wird.

$$sav_{a,b} := time_{a,z} + time_{z,b} - time_{a,b} \tag{4.2}$$

In Abbildung 4.2 befindet sich das gleiche Netzwerk aus 4.1, aber mit Zeiten an den Kanten. Die Abkürzung AB steht für Autobahn mit $130\,km/h$, LS für Landstraße mit $100\,km/h$ und FW für Feldweg mit $10\,km/h$. In der Tabelle 4.2 befinden sich die Ergebnisse der Ersparnisse zu Beginn des Savings-Algorithmus nach der Initialisierung. Erkennbar ist hier, dass nicht die Strecke mit der Autobahn bevorzugt wird, sondern der langsame Feldweg. Dieses Verhalten führt unweigerlich dazu, dass der Feldweg als erste Kombination verwendet wird.

Route	Ersparnis
A $\leftrightarrow$ B	$16,15 + 15 - 9 = 22,15$
B $\leftrightarrow$ C	$15 + 30 - 9 = 36$
C $\leftrightarrow$ D	$30 + 15 - 9 = 36$
D $\leftrightarrow$ E	$15 + 210 - 90 = 135$

Tabelle 4.2: Aufschlüsselung der Ersparnisse anhand der Teilrouten zu Beginn der Optimierung mit den benötigten Zeiten als Basis

4.3 Distanz-Zeit-Basierend

Eine Kostenberechnung mit Distanz- und Zeitwerten bietet die Möglichkeit, die Beschaffenheit der Strecke zu berücksichtigen und so zum Beispiel Feldwege zu ignorieren. Diese Daten erzeugen ein Problem. Es wirft die Frage auf, wie die Distanzen und die Zeiten in eine Relation gesetzt werden sollen. Einfaches Addieren würde dazu führen, dass eine Strecke von $45\,km$ mit einer Geschwindigkeit von $10\,km/h$ zu einer ungewollt hohen Ersparnis

führen würde. Damit dieses Verhalten nicht auftritt, muss ein Wert durch den anderen dividiert werden.
Eine für diesen Fall akzeptable Verhältnisrechnung wäre eine Division aus der Strecke in Metern und der Zeit in Minuten. In Tabelle 4.3 sind hierzu die Wertigkeiten der Kanten aus den Abbildungen 4.1 und 4.2 zu finden. An diesem Beispiel lässt sich erkennen, dass die Kante vom *Depot* zum Knoten *A* mit $2786,37$ besser bewertet wird, im Gegensatz zu der Kante vom *Depot* zum Knoten *E*. Im Grunde ist so ein Verhalten das Gewünschte, denn eine Strecke mit einer schlechten Streckenbeschaffenheit wird schwächer bewertet. Für die weitere Savingsberechnung kann die Basisberechnung verwendet

Kante	Wertigkeit
A $\leftrightarrow$ B, B $\leftrightarrow$ C, C $\leftrightarrow$ D	$15000m/9min = 1666,67$
D $\leftrightarrow$ E	$15000m/90min = 166,67$
Depot $\leftrightarrow$ A	$45000m/16,15min = 2786,37$
Depot $\leftrightarrow$ B, Depot $\leftrightarrow$ D	$25000m/15min = 1666,67$
Depot $\leftrightarrow$ C	$5000m/30min = 166,679$
Depot $\leftrightarrow$ E	$45000m/210min = 214,29$

Tabelle 4.3: Aufschlüsselung der Kantenwertigkeit anhand der Strecke und der dafür benötigten Zeit

werden, mit der berechneten Distanz-Zeit-Matrix. Diese Matrix beinhaltet dabei die Werte wie zuvor beschrieben.
Basierend auf den in der Tabelle 4.3 berechneten Kantenwertigkeiten beinhaltet die Tabelle 4.4 die daraus resultierenden Ersparnisse. Dafür wird derselbe Graph aus 4.1 mit den neuen Kantenwertigkeiten verwendet. An diesem Beispiel lässt sich erkennen, dass die Autobahn vom *Depot* zum Knoten *A* bevorzugt wird.

Route	Ersparnis
A $\leftrightarrow$ B	$2786,37 + 1666,67 - 1666,67 = 2786,37$
B $\leftrightarrow$ C	$1666,67 + 166,67 - 1666,67 = 166,67$
C $\leftrightarrow$ D	$166,67 + 1666,67 - 1666,67 = 166,67$
D $\leftrightarrow$ E	$1666,67 + 214,29 - 166,67 = 1724,29$

Tabelle 4.4: Ersparnisse für die Kombinationen mit den berechneten Werten aus Distanz und Zeit

Die Findung der passenden Kostenberechnung stellt den Entwickler des Algorithmus vor ein größeres Problem, als die Implementierung des grundsätzlichen Algorithmus. So sollten mehrere Überlegungen festgehalten und getestet werden. So kann das Verhalten festgestellt werden und auch in welche Richtung optimiert wird. Das Überprüfen und Testen kann einfach in

einem Script mit einem Graph aus der Praxis erfolgen, bei welchem die optimale Route bekannt ist. Auf diese Art kann sichtbar gemacht werden, wie gut der Algorithmus und dessen Kostenfunktion optimieren und Lösungen finden.

Kapitel 5

Problemgebiete

5.1 Kostenberechnung

Die Kostenberechnung und somit im Falle des SA die Savings-Funktion stellt den Entwickler dieser Funktion vor eine größere Herausforderung. Dabei können Fehler entstehen, die meist nur sehr schwer zu finden sind und zwar nur, wenn ein gesamter Prozess durchgesteppt wird. Hier empfiehlt es sich eine Liste der zeitlichen Werte zu notieren, um im Nachhinein die Historie des Ablaufes noch einmal aufarbeiten zu können.
Wie in Sektion 4.3 erkennbar ist, führt eine Verhältnismatrix dazu, dass unterschiedliche lange Strecken vom selben Typ zum selben Wert führen. Das Ergebnis führt dazu, dass die Knoten, welche an eine Autobahn angebunden sind, zu Beginn verknüpft werden. Im Anschluss folgt die nächsten Klasse. Das Ergebnis wirkt hier auf den ersten Blick zufriedenstellend, aber nicht, wenn dies weiter verfolgt wird. Eine Abhilfe schafft dabei das Einbeziehen der Distanz in die Berechnung. Wie in der Gleichung 5.1 festgehalten, kann auf diese Weise eine Differenzierung erreicht werden.

$$kost_{a,b} := dis_{a,b} * (\frac{dis_{a,b}}{time_{a,b}}) \tag{5.1}$$

Diese Gleichung ermöglicht es Kanten mit großen Werten und unbrauchbaren Eigenschaften diese zu neutralisieren.

5.2 Nebenbedingung

Nebenbedingungen bieten die Möglichkeit, die konkrete Problemstellung in den Algorithmus einfließen zu lassen. Bei diesen muss beachtet werden, dass diese immer überprüft werden. Sollte eine Verletzung eintreten, muss zusätzlich entschieden werden, ob diese Möglichkeit verworfen werden soll oder doch toleriert wird. Im Falle einer Tolerierung kann zusätzlich mit Strafen gearbeitet werden, sodass eine solche Möglichkeit an Gewichtung verliert.

Time Windows (Zeitfenster): Mit der Verwendung von Zeitfenstern besteht die Erweiterung der Knoten um Zeitbereiche, in welchen ein Kunde anzutreffen ist. Für diesen Fall müssen alle Knoten mit dieser Information ausgestattet werden oder eine Standardzeit angenommen werden. Bei der Verwendung in einem Savings-Algorithmus muss bei jeder Berechnung der neuen Ersparnisse überprüft werden, ob diese Nebenbedingung eingehalten wird. Der Grundalgorithmus geht anlässlich einer Verletzung von einer scharfen Nichteinhaltung aus. Dies führt zur Entfernung einiger Möglichkeiten. Eine weitere Möglichkeit bietet das Herabstufen einer solchen Möglichkeit, was zu einer sogenannten weichen Nebenbedingung führt.
Es kann definitiv die Aussage getroffen werden, dass Routing-Optimierungen kein triviales Problem darstellen. In diesem Sinne gehören sie auch zu der Kategorie *NP-vollständig*. Dies bedeutet, dass sie sich nichtdeterministisch in Polynomialzeit lösen lassen. So gibt es viele Möglichkeiten, um das Problem zu lösen, aber nicht ohne entsprechenden zeitlichen Aufwand. Die Polynomialzeit bildet dabei eine Grenze zwischen dem Bereich des *praktisch lösbaren* sowie *praktisch nicht lösbaren*.

Im folgenden Kapitel wird ein Beispiel mit dem Savings-Algorithmus aufgearbeitet. Anhand dessen wird die Auswirkung der letzten Kosten/Savings-Funktion mit der Gleichung 5.1 genauer überprüft.

Kapitel 6

Logistik mit Zeitfenster

6.1 Saving mit Zeitfenster in Python

Dieses Beispiel stellt eine vereinfachte Welt dar, welche mit 5 Knoten/Kunden und 1 Depot ausgestattet ist. Das gesamte Netzwerk wurde nicht voll vernetzt, wie in Abbildung 6.1 ersichtlich ist. Aus diesem Grund muss der Knoten D von A oder B aus über E angefahren werden. Dies gilt auch für die Strecke vom Depot aus. Die Zeit- und Distanzmatrix sind in diesem Beispiel vollständig. So beinhalten sie auch die kombinierte Strecke von $A \rightarrow E \rightarrow D$ mit den aufsummierten Kosten. In Tabelle 6.1 sind alle benötigten Werte des Graphen 6.1 aufgeschlüsselt. Hierbei muss beachtet werden, dass die Zeit in Minuten und Sekunden angegeben ist. Das Beispiel beinhaltet den gesamten Algorithmus 6.1, welcher mit der Skriptsprache Python implementiert ist. Wie in einer Sektion zuvor schon beschrieben muss mit der Initialisierung begonnen werden. Hierzu müssen die Kostenmatrix sowie die trivialen Routen erstellt werden. Bei der Distanzmatrix ist zu beachten, dass diese in Kilometern beschrieben ist und intern auf Meter umgerechnet wird. Des Weiteren beinhaltet die Zeitmatrix die Zeiten bereits in Sekunden.

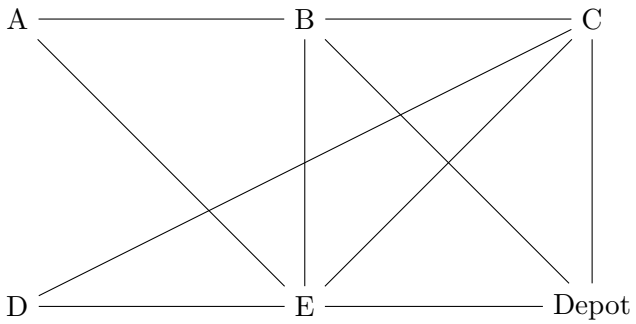

Abbildung 6.1: Beispiel Graph mit 5 Knoten/Kunden und 1 Depot

Kante	km/h	km	Zeit (mm:ss)
A ↔ B	80	10	07:30
A ↔ E	100	14,14	08:29
B ↔ C	90	10	06:40
B ↔ E	110	10	05:27
B ↔ DE	100	14,14	08:29
C ↔ D	130	22,36	10:19
C ↔ E	70	14,14	12:07
C ↔ DE	100	10	06:00
D ↔ E	90	10	06:40
E ↔ DE	100	10	06:00

Tabelle 6.1: Aufschlüsselung der Distanzen, Durchschnittsgeschwindigkeiten und der benötigten Zeiten

In dieser Implementierung des Algorithmus werden alle Zeiten in Sekunden gerechnet inklusive der Wartezeiten.
Eine triviale Route beinhaltet nur einen Knoten, sodass die Strecke *Depot → Knoten/Kunde → Depot* abgebildet wird. Weiters werden die möglichen Kombinationen erstellt und deren Ersparnis berechnet. Bei der Berechnung des Savings-Wertes muss die Nebenbedingung der Zeitfenster überprüft werden. Dieser erste Schritt erstreckt sich von Zeile 23 bis Zeile 70.
Im zweiten Schritt werden Routen kombiniert, bis keine Kombinationen mehr durchführbar sind. Dabei müssen nach der Kombination zweier Routen alle Einträge aus der Saving-Liste aktualisiert werden. Zu diesen Aktualisierungsaufgaben zählen:

- Entfernen aller Kombinationen, welche die zweite Route als Teil beinhalten;
- Ersetzen aller Routen in Kombinationen, welche die erste Route als Teil beinhalten;

Im Abschluss des zweiten Teils müssen die Ersparnisse mit überarbeiteten Routen neu berechnet und dabei die Zeitfenster geprüft werden.

Listing 6.1: Basis Implementierung des Saving-Algorithmus mit Zeitfenster

```
1  #!/usr/bin/env python
2  """
3  Saving-Algorithmus
4  Routing in Pyhton
5
6  Step 1:
7      - Init costmatrix
8      - Init base routes
9      - Init savings-list + check time windows
10     - Order savings-list desc
11 Step 2:
```

```
12      - Select top saving
13      - Merge Routes
14      - Check time windows
15      - Clear no more available savings
16      - Replace route a with new route a in savings
17      - Calc savings-list + order desc
18  """
19  import copy
20  from route import *
21  from util import *
22
23  node_de = node(0, 'DE', time_window(7, 19)) # -> Depot
24  node_a = node(1, 'A', time_window(10, 13)) # -> A
25  node_b = node(2, 'B', time_window(7, 19)) # -> B; Depot time window
26  node_c = node(3, 'C', time_window(8, 12)) # -> C
27  node_d = node(4, 'D', time_window(7, 19)) # -> D; Depot time window
28  node_e = node(5, 'E', time_window(8, 11)) # -> E
29
30
31  """      DE,     A,       B,       C,       D,       E"""
32  d_mx = [[0,      24.14,   14.14,   10,      20,      10      ], # DE
33          [24.14,  0,       10,      20,      24.14,   14.14   ], # A
34          [14.14,  10,      0,       10,      20,      10      ], # B
35          [10,     20,      10,      0,       22.36,   14.14   ], # C
36          [20,     24.14,   20,      22.36,   0,       10      ], # D
37          [10,     14.14,   10,      14.14,   10,      0       ]] # E
38
39  """      DE,     A,       B,       C,       D,       E"""
40  t_mx = [[0,      869,     509,     360,     760,     360     ], # DE
41          [869,    0,       450,     850,     909,     509     ], # A
42          [509,    450,     0,       400,     727,     327     ], # B
43          [350,    850,     400,     0,       619,     727     ], # C
44          [760,    909,     727,     619,     0,       400     ], # D
45          [360,    509,     327,     727,     400,     0       ]] # E
46
47  matrix = cost_matrix(d_mx, t_mx, 6)
48
49  print(matrix)
50
51  routes = []
52  routes.append(route([node_a], 0, node_de, t_mx))
53  routes.append(route([node_b], 1, node_de, t_mx))
54  routes.append(route([node_c], 2, node_de, t_mx))
55  routes.append(route([node_d], 3, node_de, t_mx))
56  routes.append(route([node_e], 4, node_de, t_mx))
57
58  savings = []
59  for item_a in routes:
60      for item_b in routes:
61          if item_a is not item_b:
62              s = saving(
63                      copy.deepcopy(item_a),
64                      copy.deepcopy(item_b),
65                      matrix)
```

```
66              if s is not -1:
67                  savings.append(s)
68
69 savings = util.sort_savings(savings)
70 util.print_savings(savings)
71
72 while len(savings) > 0:
73     saving = savings.pop(0)
74     print('\nSelected Saving')
75     util.print_savings([saving])
76
77     route_b_id = saving.r_b.r_id
78     route = saving.r_a
79     route.add_stops(saving.r_b)
80
81     print('\nMerged Routes')
82     print(route)
83
84     savings = [x for x in savings if x.r_b.r_id != route_b_id]
85     savings = [x for x in savings if x.r_a.r_id != route_b_id]
86
87     routes = [r for r in routes if r.r_id != route_b_id]
88
89     for i in range(len(routes)):
90         if routes[i].r_id == route.r_id:
91             routes[i] = route
92
93     for saving in savings:
94         if saving.r_a.r_id == route.r_id:
95             saving.r_a = route
96         elif saving.r_b.r_id == route.r_id:
97             saving.r_b = route
98
99     savings = [x for x in savings if x.calc(matrix) is not -1]
100
101    savings = util.sort_savings(savings)
102
103    print('\nCurrentSaving & CurrentRoutes')
104    util.print_savings(savings)
105
106 print('\n\t ---- Result ---- ')
107 print(routes)
```

6.2 Ergebnis

Gesamtheitlich kann festgestellt werden, dass eine grundsätzlich optimierte Route gefunden wurde. Trotzdem sieht es danach aus, als ob es noch bessere Routen geben müsste, wie im Ergebnis 6.2 ersichtlich. Dabei muss wiederholt werden, dass es sich bei diesem Algorithmus um eine heuristische Lösung handelt.

Listing 6.2: Ergebnis des Beispiels

```
1    ---- Result ----
2 [id: 4
3 5: E [8.0 : 11.0] | 3240
4 4: D [7.0 : 19.0] | 0
5 1: A [10.0 : 13.0] | 3771
6 2: B [7.0 : 19.0] | 0
7 3: C [8.0 : 12.0] | 0
8 ]
```

Die Route $Depot \rightarrow E \rightarrow D \rightarrow A \rightarrow B \rightarrow C \rightarrow Depot$ spiegelt zusätzlich eine Eigenheit des Savings-Algorithmus wieder. Dazu muss die Historie genauer betrachtet werden. Wie im Listing 6.3 zu erkennen ist, werden in der ersten Vereinigung die Routen mit den Knoten A und B kombiniert. Der Grund liegt in der Ersparnis, da Knoten, die weiter vom Depot entfernt sind und nahe beieinander liegen, einen hohen Wert erlangen.

Listing 6.3: Erste Kombination von zwei Teilrouten

```
1 Selected Saving
2 SavingsCount: 1
3 841172.84
4    r_a: 1
5    r_b: 2
6
7 Merged Routes
8 id: 0
9 1: A [10.0 : 13.0] | 9931
10 2: B [7.0 : 19.0] | 0
```

Im Vergleich dazu bekommt die Vereinigung $C \leftrightarrow D$ nur auf einen sehr niedrigen Wert. Dies kommt mit der großen Entfernung zwischen den Punkten zustande und der geringeren Entfernung zum Depot.
In einem direkten Vergleich der generierten Tour und einer manuell zusammengestellten Tour lässt sich feststellen, dass es eine bessere Route gibt. In der Tabelle 6.2 befinden sich die Zeit- und Distanzkosten der generierten Tour. Die Tabelle 6.3 beinhaltet die Zeit- und Distanzkosten der manuellen

	E	D	A	B	C	DE	Summe
Zeit	360	400	400 + 509	450	400	360	2879 s
Distanz	10	10	10 + 14,14	10	10	10	74,14 km

Tabelle 6.2: Benötigte Zeit und Strecke der generierten Tour

erstellten Tour. Im Vergleich lässt sich erkennen, dass die Kante $C \leftrightarrow D$ mehr Zeit und Distanz mit sich bringt. Dafür wird die Kante $D \leftrightarrow E$ nicht mehrfach benötigt. Der zeitliche Unterschied liegt bei 32 Sekunden ohne Berücksichtigung der Wartezeiten. In Bezug auf den realen Straßenverkehr, kann dieser Wert vernachlässigt werden. Die berechnete Kostenmatrix führt

	C	D	E	A	B	DE	Summe
Zeit	360	619	400	509	450	509	2847 s
Distanz	10	22,36	10	14,14	10	14,14	60,64 km

Tabelle 6.3: Benötigte Zeit und Strecke der manuellen Tour

in diesem Fall dazu, dass die Strecke und die Zeit optimiert werden. Sollte aber die Distanz minimiert werden, so müsste eine Distanzmatrix als Kostenmatrix verwendet werden. Eine andere Möglichkeit wäre die Berechnung der Kosten abzuändern, sodass sich die Distanzen stärker widerspiegeln.

Kapitel 7

Zusammenfassung

Routing in der Logistik bietet die Möglichkeit effizienter Auslieferungen zu erledigen. Dabei existieren allgegenwärtig Navigationssysteme wie Google Maps. Diese Lösungen liefern meist nur die Möglichkeit einfache *point-to-point* Optimierungen durchzuführen. Diese Konsumentenlösungen verwenden ohne Internetzugang nur Offline-Daten ohne aktuelle Verkehrslagen zu berücksichtigen. Im Falle von Belieferungen wird eine höhere Abstraktion benötigt. So muss nicht nur die Strecke zwischen den Haltepunkten optimiert werden, sondern auch die Reihenfolge dieser. Dies führt somit von *point-to-point* Optimierungen, wie der Dijkstra-Algorithmus, zu Knoten/-Kunden Optimierungen, wie dem *Tabu-Search*.
Im Rahmen der Arbeit wurden einige Problemstellungen der Logistik aufgezeigt. Es wurden auch mögliche Lösungsansätze für solche Probleme gezeigt und beschrieben. Der Savings-Algorithmus mit seinen Eigenheiten wurde zudem genauer beschrieben. Des Weiteren stellen Kostenfunktionen eine zentrale Komponente in Routing-Algorithmen dar. Durch diese wird definiert, wie und nach was optimiert wird.
Der Einsatz von Optimierungen in der Logistik benötigt spezielle Algorithmen. Die Komplexität solcher Routing-Probleme stellt eine besondere Herausforderung dar. So kann in begrenzter Zeit nicht einfach das Optimum gefunden werden. Eine Ausnahme besteht, wenn die Anzahl an Stopps so gering ist, dass alle Kombinationen getestet werden können. Aus dieser zeitlichen Einschränkung heraus werden meistens heuristische und metaheuristische Lösungen verwendet. Somit werden Routing-Probleme, wie das *Traveling Salesman Problem*, die Menschheit noch weiter beschäftigen.

Quellenverzeichnis

Literatur

[1] Ajith Abraham, He Guo und Hongbo Liu. „Swarm intelligence: foundations, perspectives and applications“. In: *Swarm Intelligent Systems*. Springer, 2006, S. 3–25 (siehe S. 67).

[2] Hans-Georg Beyer u. a. *Evolution are Algorithmen Begriffe und Definitionen*. Techn. Ber. (siehe S. 68).

[3] Geoff Clarke und John W Wright. „Scheduling of vehicles from a central depot to a number of delivery points“. *Operations research* 12.4 (1964), S. 568–581 (siehe S. 69).

[4] Christian Fischer. *Intelligente Systeme in der Logistik*. 2008. URL: http://www.informatik.uni-ulm.de (siehe S. 68).

[5] Scott Kirkpatrick, Mario P Vecchi u. a. „Optimization by simulated annealing“. *science* 220.4598 (1983), S. 671–680 (siehe S. 67).

[6] Gilbert Laporte. „The vehicle routing problem: An overview of exact and approximate algorithms“. *European journal of operational research* 59.3 (1992), S. 345–358 (siehe S. 60–62).

[7] Antonio Mucherino, Stefka Fidanova und Maria Ganzha. „Ant colony optimization with environment changes: an application to GPS surveying“. In: *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*. IEEE. 2015, S. 495–500 (siehe S. 67).

[8] Jakob Puchinger. *Optimierungsverfahren in der Transportlogistik*. 2010. URL: https://www.ads.tuwien.ac.at/teaching/ss10/Transportlogistik/OTL_10.pdf (siehe S. 64, 66).

[9] Richard Wiener. „Branch and Bound Implementations of the Traveling Salesperson Problem - Part 4: Distributed processing solution using RMI.“ *Journal of Object Technology* 2.6 (2003), S. 51–65 (siehe S. 64).

[10] Bernhard Wurzer. „Fallbeispiele zur Tourenplanung mit Hilfe spezieller Tourenplanungssoftware am Beispiel eines Grazer Kleinunternehmens“. Diss. uniwien, 2010 (siehe S. 60, 62).