Masterarbeit

Automatisierte Tourenplanung in der Transportlogistik

eingereicht an der

Montanuniversität Leoben

erstellt am

Lehrstuhl Angewandte Mathematik

Vorgelegt von:

Klaus Pichler
m0935032

Betreuer:

Ao. Univ.Prof.Dr. Norbert Seifter

Leoben, am 30. September 2015

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient habe.

Affidavit

I declare in lieu of oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Leoben,am _____ _____

Datum Unterschrift

Kurzfassung

In Hinblick auf die steigenden Transportkosten wird in Zukunft die Rolle der Transportlogistik immer wichtiger. Um Umweltbelastungen und Transportkosten so niedrig wie möglich zu halten, müssen die Fahrzeuge eines Unternehmens bei der Verteilung ihrer Waren äußerst effizient eingesetzt werden. Um dies zu erreichen, kann in vielen Fällen eine gezielte automatisierte Tourenplanung in der unternehmensinternen Disposition eingesetzt werden.
Diese Arbeit beschäftigt sich mit dem sogenannten VRP[1], einem kombinatorischen Optimierungsproblem. Bei dem Vehicle Routing Problem geht es darum, Kunden mittels Lieferfahrzeugen, von einem oder mehreren Depots aus, möglichst effizient zu beliefern, sodass die dabei entstehenden Transportkosten möglichst gering gehalten werden. Dabei muss entschieden werden, welche Transportaufträge den einzelnen Fahrzeugen zugeordnet werden, und in welcher Reihenfolge die Kunden innerhalb einer Tour beliefert werden. In der Arbeit wird aufgezeigt, mit welchen mathematischen Algorithmen dieses Problem näherungsweise gelöst werden kann, und wie diese im Detail funktionieren. Zur Lösung dieser VRPs wurde eine eigene auf JAVA basierende Software entwickelt, mit der auch eine umfangreiche Visualisierung der Probleminstanzen möglich ist. Des Weiteren wird auch die Herkunft der Kartendaten besprochen, welche zur Lösung der Tourenplanungsprobleme herangezogen werden. Dabei ist natürlich zu beachten, dass es sich hierbei um riesige Datenmengen handelt, da Distanzen, Geschwindigkeitsbeschränkungen, Einbahnregelungen, Fahrverbote usw. beachtet werden müssen.
Abschließend werden die mit verschiedenen Verfahren gelösten VRPs bezüglich ihrer Lösungsqualität und benötigten Laufzeit diskutiert. Daraus kann abgeleitet werden, ob bei vorhandenen realen Aufgabenstellungen ein Einsparungspotential vorhanden und daher der Einsatz einer automatisierten Tourenplanung gerechtfertigt ist.

[1] Vehicle Routing Problem

Abstract

In the future the rising transport costs will become more important in the transport logistics. To keep environmental pollution and transport costs as low as possible, the vehicles of the company for distributing its products must be used extremely efficiently. To achieve this, a targeted automated route planning can be used in many cases in the company's internal disposition.

This work deals with the so-called Vehicle Routing Problem, a combinatorial optimization problem. In the VRP it comes to supplying the customers using delivery vehicles of one or more depots as efficiently as possible, so that the resulting transportation costs are kept to a minimum. It has to be decided, which transport orders will be allocated to individual vehicles, and in which order the customers within a tour will be delivered. In this work it is shown, by which mathematical algorithms this problem can be solved as an approximation, and how these work in detail. To solve this, a JAVA based software was developed, which also makes a comprehensive visualization of the problem instances possible. Furthermore, the origin of the map data is discussed, which are used for solving the VRPs. It should also be noted, that here we deal with a huge amount of data, because the distances, velocities restrictions, one-way restrictions and driving bans need to be considered.

Finally, VRPs are discussed by various methods with respect to their solution quality and required runtime. From this can be deduced, whether a savings potential is available and therefore the use of an automated route planning for real problems is justified.

Inhaltsverzeichnis

Abbildungsverzeichnis

Acronym

VRP	Vehicle Routing Problem
TSP	Travelling Salesman Problem
CVRP	Capacitated Vehicle Routing Problem
DCVRP	Distance-Constrained Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
VRPB	Vehicle Routing Problem with Backhauls
VRPPD	Vehicle Routing Problem with Pickup and Delivery
PDP	Pickup and Delivery Problem
TS	Tabu Search
SA	Simulated Annealing
GA	Genetische Algorithmen
OSM	Open Street Map
CR	Crossover Rate
RAR	Ruin and Recreate
RFCS	Route First Cluster Second

1 Einführung

Viele Unternehmen in den unterschiedlichsten Branchen auf der ganzen Welt beschäftigen sich täglich mit einer Aufgabe die als Tourenplanung bezeichnet wird.
Die Aufgabenstellung der Tourenplanung besteht darin, innerhalb einer vorgegebenen Planungsperiode Transportaufträge mit bestimmten Sendungsmengen zu Touren zusammenzufassen, so dass jede Tour mit einem am Depot positionierten Fahrzeug durchgeführt werden kann. Für jede Tour ist außerdem eine optimale Bearbeitungsreihenfolge der Aufträge zu finden. Als mögliche Zielvorgaben sind die Minimierung der insgesamt zurücklegenden Wegstrecke, der benötigten Fahrzeit oder der einzusetzenden Fahrzeuge vorzusehen.
Für die Tourenplanung gibt es eine Vielzahl von Anwendungsgebieten:

- Speditionsunternehmen, die Güter bei verschiedenen Kunden ausliefern und/oder einsammeln müssen
- Müllentsorgungsunternehmen
- Schüler- und Krankentransporte
- Innerbetriebliche Werksverkehre

Die einzelnen Problemstellungen unterscheiden sich in Abhängigkeit von der zugrunde liegenden Branche und dem Anwendungsbereich, wodurch sich zahlreiche zu berücksichtigende Kriterien ergeben. [29]
Da die Transportkosten in den letzten Jahren, unter anderem ausgelöst durch teurer werdenden Treibstoff, immer weiter ansteigen, ist eine effiziente Tourenplanung von großer Bedeutung. In vielen Unternehmen wird die Tourenplanung aufgrund von Erfahrungswerten der jeweiligen Disponenten händisch erstellt. Hierbei kann jedoch keine gesicherte Aussage getroffen werden, ob diese Lösungen wirklich kosteneffizient sind. Deshalb kann in vielen Fällen eine automatisierte Tourenplanung eingesetzt werden, welche auf mathematischen Algorithmen basiert, und bessere Lösungen erzielt. Das der Tourenplanung zugrundeliegende kombinatorische Optimierungsproblem wird als VRP bezeichnet, und wurde erstmals von Dantzig und Ramser quantitativ untersucht.[6]
Der Schwerpunkt dieser Arbeit liegt in der Beschreibung der Möglichkeiten zur Lösung des VRP, die im Zuge dieser Arbeit auch in das entwickelte Programm integriert werden. Des Weiteren werden verschiedene Probleminstanzen des VRP gelöst, und hinsichtlich ihrer Lösungsqualität

und benötigten Laufzeit diskutiert. Auch die Herkunft der benötigten Kartendaten wird beschrieben.

2 Allgemeine Grundlagen

Hier sollen zunächst einige wichtige Grundlagen und Definitionen der Graphentheorie sowie des TSP[2] und des VRP näher beschrieben werden, um das vorliegende kombinatorische Problem besser zu verstehen.

2.1 Graph

Graphen sind mathematische Modelle für netzartige Strukturen die vielseitig angewendet werden. Darunter fallen unter anderem folgende Beispiele (siehe [37]):

- Straßennetze
- Computernetze
- elektrische Schaltungen
- Programmabläufe
- Wasser- und Gasleitungsnetze
- chemische Moleküle
- wirtschaftliche Verflechtungsbeziehungen

Für diese Arbeit ist speziell die Modellierung von Straßennetzen von besonderer Bedeutung. Eine allgemeine Definition eines Graphen kann beispielsweise in [8] gefunden werden: Ein Graph ist ein Paar $G = (V, E)$ disjunkter Mengen mit $E \subseteq [V]^2$; die Elemente von E sind also 2-elementige Teilmengen von V. Die Elemente von $V = \{v_1, ..., v_n\}$ nennt man *Knoten* des Graphen, die Elemente von $E = \{e_1, ..., e_n\}$ heißen *Kanten*. Bildlich kann man einen Graphen darstellen, indem man seine Knoten als Punkte zeichnet und zwei dieser Punkte immer dann durch eine Linie verbindet, wenn zwischen diesen beiden Knoten eine Kante existiert. Wie man diese Punkte und Linien zeichnet, ob gerade oder geschwungen, disjunkt oder überkreuz, ist eine Frage der Zweckmäßigkeit und der Ästhetik. Die formale Definition eines Graphen ist jedenfalls von seiner bildlichen Darstellung unabhängig.
Die Abbildung 2.1 zeigt einen einfachen ungerichteten Graphen in dem die Knoten als graue Punkte und die Kanten als grüne Linien dargestellt werden.

[2] Travelling Salesman Problem

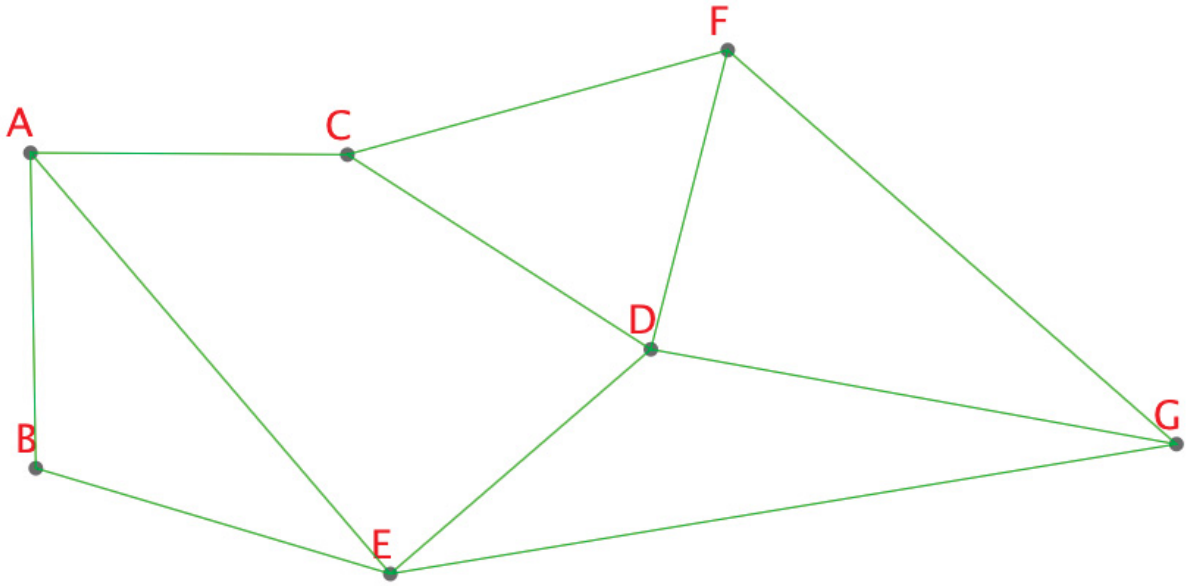


Abbildung 2.1: Darstellung eines einfachen ungerichteten Graphs

Wenn zwei Knoten $u, v \in V(G)$ durch eine Kante $e = \{u, v\}$ verbunden sind, dann heißen diese *adjazent* in G. Aus Abbildung 2.1 ist ersichtlich, dass zum Beispiel die Knoten A und C adjazent (benachbart) sind. Der Grad deg v eines Knotens $v \in V(G)$ ist die Anzahl der zu v benachbarten Knoten. Der Knoten A hat daraus folgend den Grad 3.
Wenn eine Kante in beide Richtungen durchlaufen werden kann, so wird diese als *ungerichtet* bezeichnet, andernfalls bezeichnet man sie als *gerichtet*. Eine gerichtete Kante kann man sich zum Beispiel in der Realität als eine Einbahnstraße vorstellen. Ein Graph, in dem alle Kanten gerichtet sind, wird als *Digraph* bezeichnet und wird in Abbildung 2.2 veranschaulicht. Gerichtete Kanten werden üblicherweise mit einem Pfeilsymbol versehen.

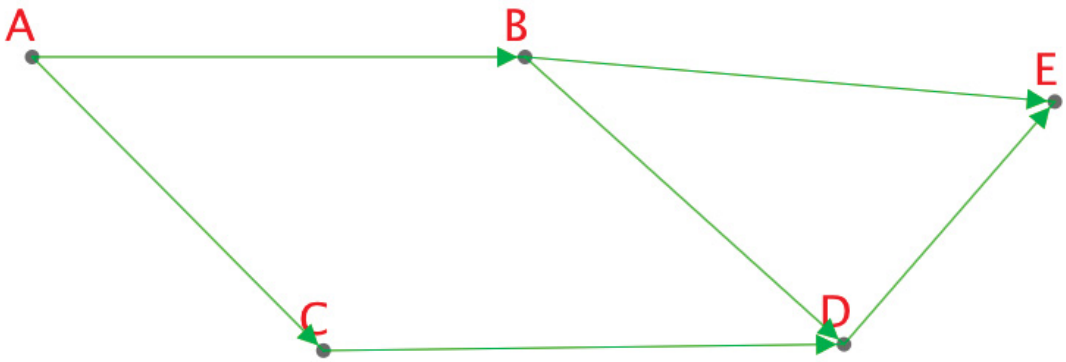


Abbildung 2.2: Gerichteter Graph (Digraph)

Sind in einem Graphen wie in Abbildung 2.3 alle Knoten miteinander verbunden, sodass es für alle Knotenpaare $(v_i, v_j), v_i, v_j \in V, i \neq j$ eine verbindende Kante gibt, so handelt es sich um einen *vollständigen* Graph.

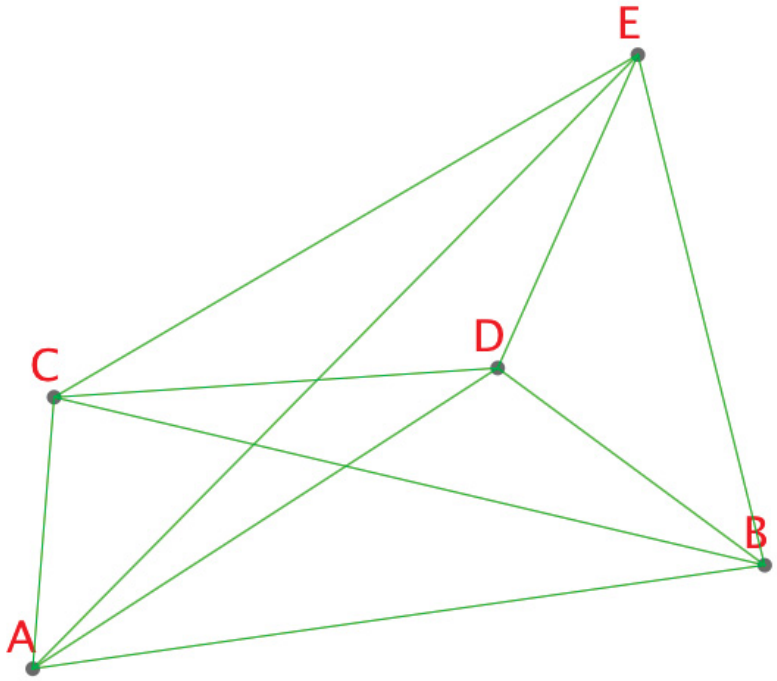


Abbildung 2.3: Vollständiger Graph

Ein Graph heißt *einfach*, wenn es zwischen zwei Knoten $v_i, v_j \in V$ maximal eine verbindende Kante $e_i \in E$ gibt und der Graph keine Schlingen (Kanten der Form (v_i, v_i)) enthält.
In der Graphentheorie wird eine Folge von Knoten und Kanten $W = \{v_0, e_0, v_1, ..., v_k, e_k, v_{k+1}\}$ als *Weg* bezeichnet. Ist der betrachtete Graph einfach, so genügt es, den Weg als eine Folge von Knoten $W = \{v_0, v_1, ..., v_k, , v_{k+1}\}$ zu beschreiben. Wenn alle Knoten in der Folge $W = \{v_0, v_1, ..., v_{k+1}\}$ paarweise verschieden sind, so handelt es sich um einen *Pfad*. Ist der Pfad zusätzlich geschlossen, das heißt $v_{k+1} = v_0$, so spricht man von einem *Kreis*. [8]

2.1.1 Bewertete Graphen

In den praktischen Anwendungsproblemen werden die Kanten eines Graphen in den meisten Fällen mit reellen Zahlen bewertet.
Ein gerichteter oder ungerichteter Graph $G = (V, E)$ wird *kantenbewertet* genannt, falls jeder Kante $e \in E(G)$ eine reelle Zahl zugeordnet ist, falls es also eine Abbildung $c : E \to \mathbb{R}$ gibt. Die Bewertung der Kante $\{u, v\}$ wird mit $c(u, v)$ bezeichnet. [43]
Zur Visualisierung der Kantenbewertungen können die Bewertungen neben den entsprechenden Kanten wie in Abbildung 2.4 im Graph platziert werden. Eine andere Art und Weise,

Kantenbewertungen zu visualisieren, besteht in der Färbung der Kanten. Hierbei werden die Zahlen mit Farben indentifiziert.

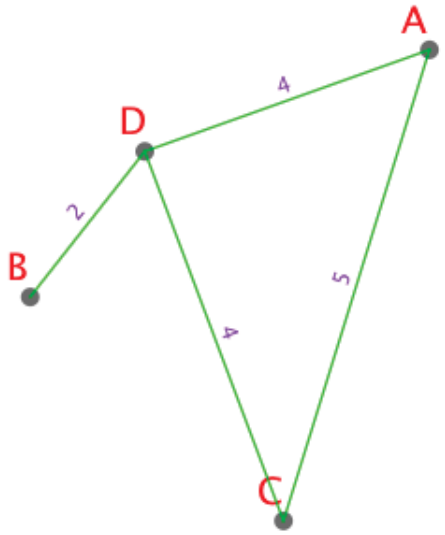


Abbildung 2.4: Bewerteter Graph

Wird ein Straßennetz als Graph modelliert, so kann die Kantenbewertung als Zeit, die das Fahrzeug für diesen Straßenabschnitt benötigt, oder als Länge des Straßenabschnitts interpretiert werden. Angenommen, die Bewertungen aus Abbildung 2.4 stehen für Minuten, und der kürzeste Pfad zwischen den Knoten B und A ist gesucht. Dann kann am Graphen abgelesen werden, dass der Weg über die Knoten B, D und A führt und dessen Länge 6 Minuten beträgt. Die Länge eines Weges $W(v_0, v_n)$ von v_0 nach v_n wird mit $l(W) = \sum_{k=0}^{n-1} c(v_k, v_{k+1})$ berechnet. Bewertete Graphen, die darüber hinaus auch vollständig sind, können eine wichtige Eigenschaft besitzen, welche im späteren Verlauf der Arbeit benötigt wird: Gilt $c(u, w) \leq c(u, v) + c(v, w)$ für alle Knoten $u, v, w \in V(G)$ so heißt dieser Graph *metrisch*. Das heißt, der Weg von u über v nach w darf nicht kostengünstiger sein als der direkte Weg von u nach w.

2.1.2 Kürzeste Pfade in Graphen

Ist man an dem kürzesten Pfad zwischen zwei Knoten $v_0, v_1 \in V$ eines Graphen $G = (V, E)$ interessiert, kann das effektiv nicht durch den Vergleich aller möglichen Pfade zwischen diesen Knoten bestimmt werden. Man nehme an, $G = (V, E)$ repräsentiere das Straßennetz von Österreich und gesucht sei der kürzeste Pfad von Leoben nach Salzburg. Es wäre eine riesige Anzahl von Möglichkeiten zu untersuchen, von denen es auch viele einfach nicht wert sind, betrachtet zu werden. Zum Beispiel ist eine Route von Leoben nach Salzburg, die über Wien geht, ganz sicher eine schlechte Wahl.
Folgende Algorithmen können bei dem kürzeste-Pfade-Problem eingesetzt werden

- Dijkstras Algorithmus

- Bellman-Ford-Algorithmus
- Floyd-Warshall-Algorithmus
- Johnson's Algorithmus

Da in dieser Arbeit nur der Algorithmus von Dijkstra von Relevanz ist, wird dieser als einziger hier näher beschrieben.
Der Algorithmus von Dijkstra löst das kürzeste-Pfade-Problem mit einem Startknoten $s \in V$ auf einem bewerteten zusammenhängenden Graphen $G = (V, E)$ für den Fall, dass alle Kantenbewertungen nichtnegativ sind. Es wird daher im folgenden vorausgesetzt, dass $c(u, v) \geq 0$ für alle Kanten $(u, v) \in E$ gilt.
In Algorithmus 1 wird die Vorgangsweise dieses Verfahrens schrittweise veranschaulicht. Dabei werden folgende Attribute verwendet:

- Das Knotenattribut d beschreibt die Länge des bisher kürzesten gefundenen Pfades von s zu dem jeweiligen Knoten
- Das Knotenattribut pi merkt sich, über welchen Vorgängerknoten der bisher kürzeste gefundene Pfad verbunden ist
- Die Menge S verwaltet die Knoten, zu denen bereits der kürzeste Pfad gefunden wurde
- Die Menge Q verwaltet die Knoten, zu denen noch kein kürzester Pfad gefunden wurde.

Algorithmus 1 Dijkstra

Input: (G, s, c)
for all $v \in G.V$ **do**
$v.d = \infty$
$v.\pi = NULL$
end for
$s.d = 0$
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$ **do**
$u = EXTRACT\text{-}MIN(Q)$
$S = S \cup \{u\}$
for all $v \in G.Adj[u] \wedge v \notin S$ **do**
if $v.d > u.d + c(u, v)$ **then**
$v.d = u.d + c(u, v)$
$v.\pi = u$
end if
end for
end while

Der Algorithmus wird beendet, wenn die Menge Q leer wird. Dann wurde zu jedem Knoten ein kürzester Pfad gefunden. [4]
Um jetzt also speziell den kürzesten Pfad zwischen zwei Knoten $v_0, v_1 \in V$ eines Graphen $G = (V, E)$ zu berechnen, muss zunächst der Dijkstra Algorithmus mit einem der beiden Knoten als Startknoten s gestartet werden. Danach wird über die π Attribute beginnend beim anderen Knoten der Pfad rückwärts rekonstruiert. Hierbei soll aber noch angemerkt werden, dass in diesem Fall der Algorithmus 1 auch vorzeitig abgebrochen werden kann, sobald $v_1 \in S$. Durch diese Maßnahme kann unnötige Rechenzeit vermieden werden.
Zur Veranschaulichung eines kürzesten Pfades, wird in Abbildung 2.5 das Straßennetz von Graz und naher Umgebung durch einen Graphen modelliert. Zwischen den beiden Knoten v_0: Hauptplatz in Gratkorn und v_1: Murfelderstraße 1 in Graz wurde der kürzeste Pfad, welcher mit Hilfe des Algorithmus von Dijkstra errechnet wurde, eingezeichnet.

Abbildung 2.5: Kürzester Pfad zwischen zwei Knoten

2.2 Das Traveling Salesman Problem

Aufgrund der einfachen Modellierbarkeit und Anschaulichkeit wird üblicherweise das TSP als Ausgangspunkt zu Betrachtungen über Tourenplanungsprobleme gewählt. Das TSP beschäftigt sich mit einem Handelsreisenden, der eine Anzahl von Städten zu besuchen hat und anschlie-

ßend zum Ausgangspunkt zurückkehrt. Das Ziel dabei ist die Minimierung der zurückgelegten Wegstrecke. Das Wegnetz kann durch einen Graphen $G = (V, E), E \subseteq [V]^2$, mit einer Kostenfunktion $c : E \to \mathbb{R}, (i, j) \mapsto c_{ij}$, abgebildet werden. Häufig wird dabei ein vollständiger metrischer[3] Graph eingesetzt, um die Lösbarkeit des Problems sicherzustellen. Das TSP kann als ein VRP verstanden werden, in dem alle zu beliefernden Kunden einem einzigen Fahrzeug zugeordnet werden. Das Problem lässt sich auch in prägnanter Form als ganzzahliges lineares Programm formulieren:

$$Min! \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ij} \tag{2.1}$$

unter den Nebenbedingungen

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in [V]^2 \tag{2.2}$$

$$\sum_{j \in V} x_{ji} = 1 \qquad \forall i \in V \tag{2.3}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in V \tag{2.4}$$

$$\sum_{i \in S, j \notin S} x_{ij} \geq 1 \qquad \forall S \subset V \tag{2.5}$$

$$\sum_{i \in S, j \notin S} x_{ji} \geq 1 \qquad \forall S \subset V \tag{2.6}$$

Die Entscheidungsvariablen x_{ij} geben an, ob zwischen den Knoten i und j eine direkte Verbindung besteht. Nach 2.2 handelt es sich hierbei um binäre Variablen. Während die Nebenbedingungen 2.3 und 2.4 sicherstellen, dass jeder Knoten nur einmal besucht, beziehungsweise verlassen wird, schließen die Nebenbedingungen 2.5 und 2.6 Konfigurationen aus dem Lösungsraum aus, die mehrere Zyklen anstelle eines einzelnen zusammenhängenden Zyklus ergeben. [23]

Bereits 1972 wurde in [17] die NP-Vollständigkeit des TSP nachgewiesen. Das bedeutet, dass kein Verfahren bekannt ist, das dieses Problem immer und nachweislich optimal mit einem Zeitaufwand löst, der höchstens polynomiell mit der Problemgröße anwächst. In diesem Fall entspricht die Problemgröße der Anzahl der Knoten.

Zur Lösung des TSP stehen viele Heuristiken und exakte Verfahren zur Verfügung, von denen einige im Laufe dieser Arbeit noch näher beschrieben werden.

[3] Metrisch: siehe Kapitel 2.1.1

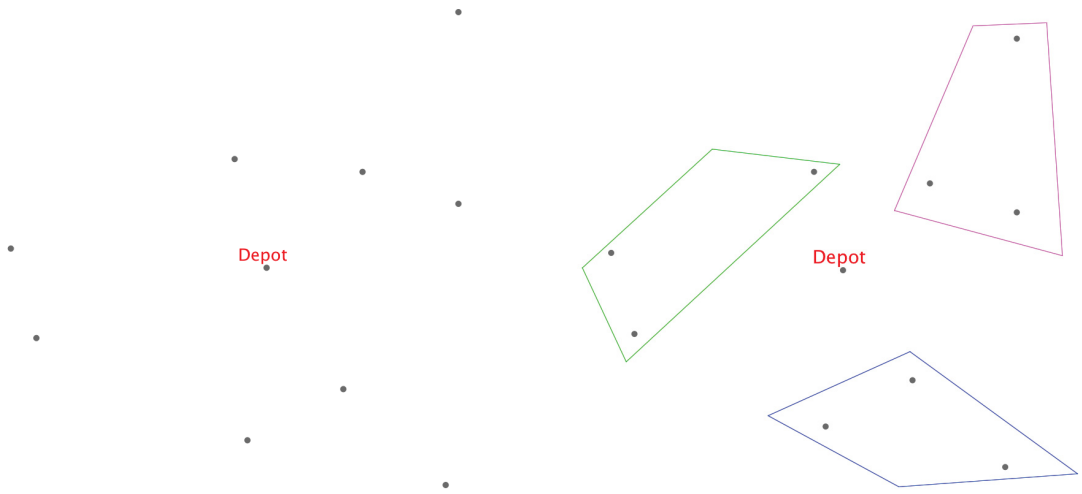
2.3 Das Vehicle Routing Problem

Wie bereits in Kapitel 1 beschrieben, liegt bei dem VRP eine Ausgangssituation vor, in der eine Flotte von Fahrzeugen mit Kapazitätsbeschränkungen eine Anzahl von Kunden von einem Depot aus beliefern. Es müssen die einzelnen Rundreisen so gestaltet werden, dass alle Restriktionen wie zum Beispiel:

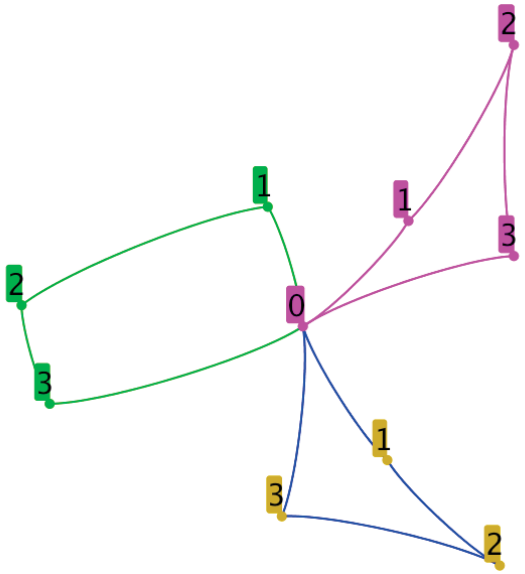
- Kapazität der Fahrzeuge.
- Maximale Lenkzeiten der Fahrer.

eingehalten werden. Probleme, die als einzige Restriktion die Kapazität der Fahrzeuge haben, werden in der Literatur als CVRP[4] bezeichnet.
Anhand dieser kurzen Beschreibung kann bereits eine wichtige Eigenschaft aller Vehicle Routing Probleme erkannt werden, nämlich das Vorhandensein zweier grundlegender Problemstellungen, dem *Clustering* und dem *Routing*. Beim Clustering geht es um die Entscheidung, auf welche Touren die einzelnen Aufträge disponiert werden, während das Routing die Reihenfolge der einzelnen Aufträge innerhalb einer Tour festlegt. Dadurch ist ersichtlich, dass das Routing auf ein TSP zurückgeführt werden kann, wodurch das TSP zu einem Teilproblem des VRP wird. In Abbildung 2.6 wird der Zusammenhang zwischen Clustering und Routing veranschaulicht.
Es soll nun an dieser Stelle das CVRP als ganzzahliges lineares Optimierungsproblem, wie schon zuvor das TSP, beschrieben werden. Dabei wird im Folgenden auf die Formulierungen von [23] zurückgegriffen.
Ausgehend vom TSP entsteht das CVRP, indem einer der Knoten als Depot ausgezeichnet wird[5], während die übrigen als Kundenknoten angesehen werden, denen jeweils eine Nachfrage d_i zugeordnet wird. Die Mengen sind bekannt und dürfen nicht auf mehrere Anlieferungen verteilt werden. Schließlich wird die Kapazität des Transportmittels durch den Parameter q begrenzt. Gilt $\sum_{i \in V} d_i \leq q$, so entspricht das Problem dem TSP, da alle Kunden auf einer einzigen Tour beliefert werden können. Andernfalls ergibt sich die Notwendigkeit, mehrere Fahrzeuge für die Belieferung der Kundenknoten einzusetzen.
Vorrangiges Ziel ist häufig die Reduzierung der Anzahl der eingesetzten Fahrzeuge. Damit wird unterstellt, dass jedes eingesetzte Fahrzeug hohe Fixkosten verursacht, deren Einsparung ein beliebiges Verlängern der Touren der übrigen Fahrzeuge rechtfertigt. Oft wird aber auch die Transportkostenminimierung aus dem TSP übernommen.
Die Menge K repräsentiert im Folgenden die Menge der Fahrzeuge, beziehungsweise Touren. Die Entscheidungsvariablen aus 2.2 werden nun um einen Index hierfür erweitert. Somit gibt x_{ijk} an, ob auf Tour k der Knoten j als direkter Nachfolger von i ist, oder nicht. Das Depot wird durch die zwei Knoten 0 und $n+1$ abgebildet, von denen der erste als Start- und der

[4] Capacitated Vehicle Routing Problem
[5] in der Regel der Knoten 0



((a)) Ausgangssituation ((b)) Clustering



((c)) Routing

Abbildung 2.6: Zusammenhang zwischen Clustering und Routing

zweite als Zielpunkt der Fahrzeuge angesehen wird. Die Kosten der Wegstrecke c_{ij} entsprechen denen des TSP. w_{ik} gibt den Zeitpunkt an, zu dem die Tour k den Knoten i erreicht, $t_{ij} > 0$ bezeichnet die Fahrzeit zwischen den Knoten. Hiermit lässt sich das Problem als gemischt-ganzzahliges lineares Programm wie folgt darstellen:

$$Min! \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ijk} \tag{2.7}$$

unter den Nebenbedingungen

$$x_{ijk} \in \{0,1\} \qquad \forall (i,j) \in [V]^2, \forall k \in K \tag{2.8}$$

$$\sum_{k \in K} \sum_{j \in V} x_{ijk} = 1 \qquad \forall i \in V \setminus \{0\} \tag{2.9}$$

$$\sum_{i \in V \setminus \{0\}} d_i \sum_{j \in V} x_{ijk} \leq q \qquad \forall k \in K \tag{2.10}$$

$$\sum_{j \in V} x_{0jk} = 1 \qquad \forall k \in K \tag{2.11}$$

$$\sum_{i \in V} x_{ihk} - \sum_{j \in V} x_{hjk} = 0 \qquad \forall h \in V \setminus \{0\}, \forall k \in K \tag{2.12}$$

$$\sum_{i \in V} x_{i,n+1,k} = 1 \qquad \forall k \in K \tag{2.13}$$

$$w_{ik} + t_{ij} - M(1 - x_{ijk}) \leq w_{jk} \qquad \forall (i,j) \in [V]^2, \forall k \in K \tag{2.14}$$

$$w_{ik} \geq 0 \qquad \forall i \in V, \forall k \in K \tag{2.15}$$

2.7 zeigt die Zielfunktion, in der die Gesamtwegstrecke minimiert wird, wie es bei dem CVRP üblicherweise angewendet wird. Die erweiterten Binärvariablen werden in 2.8 festgelegt.
Die restlichen Nebenbedingungen können wie folgt verstanden werden:

- Gleichung 2.9 stellt sicher, dass jeder Kunde genau einmal verlassen wird.
- Durch 2.10 wird die Einhaltung der Fahrzeugkapazitäten gewährleistet
- Durch die Gleichungen 2.11 und 2.13 wird sichergestellt, dass jede Tour beim Depot beginnt, beziehungsweise auch dort endet.
- Mit 2.12 wird sichergestellt, dass ein Kunde gleich oft besucht und verlassen wird.
- Gleichungen 2.14 und 2.15 schließen Kurzzyklen aus.

M aus Gleichung 2.14 entspricht einer sehr großen Zahl mit $M > \sum_{(i,j)\in[V]^2} t_{ij}$. M wird dafür verwendet, um die Nebenbedingung

$$w_{ik} + t_{ij} \leq w_{jk} \quad \forall (i,j,k) \in \{(i,j,k) \mid x_{ijk} = 1\} \Leftrightarrow (w_{ik} + t_{ij}) \cdot x_{ijk} \leq w_{jk}$$

zu linearisieren. Die Herstellung einer zeitlichen Beziehung erfolgt dadurch nur für direkt aufeinanderfolgende Fahrten. Hierdurch wird mit $t_{ij} > 0$ das Entstehen von Kurzzyklen verhindert, die nicht am Depot beginnen und enden.
Im Gegensatz zu der beschriebenen Drei-Index-Formulierung, wird in [39] eine alternative Zwei-Index-Formulierung für das VRP vorgestellt.

2.3.1 Erweiterungsformen des klassischen Vehicle Routing Problem

Je nach Ausgangssituation und Anforderung des betrachteten Unternehmen, können verschiedene Formen bzw. Erweiterungen des Problems unterschieden werden. Laut [39] können die Basisprobleme des VRP nach Abbildung 2.7 eingeteilt werden. Das klassische CVRP wird dabei je nach den gewünschten Anforderungen erweitert.

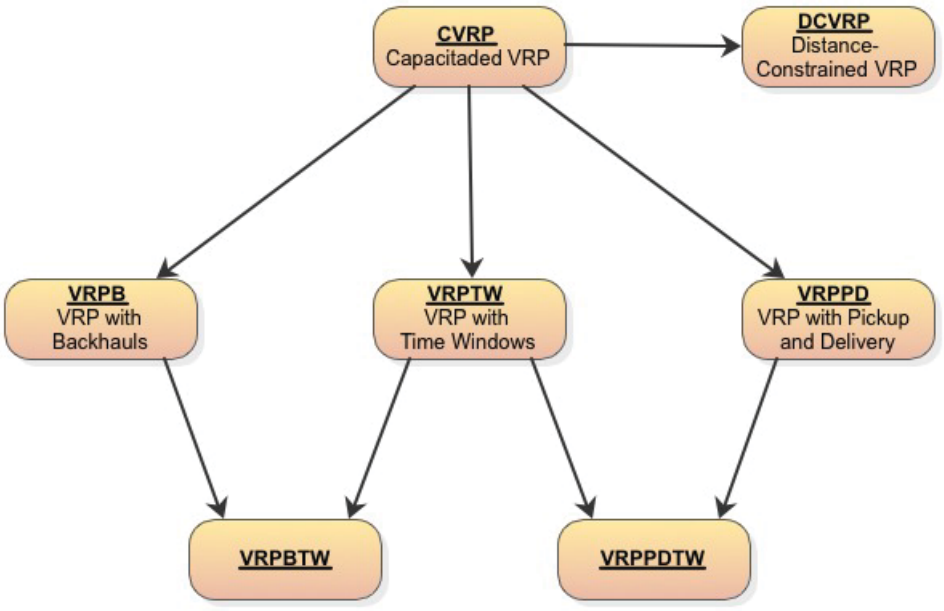


Abbildung 2.7: Basisprobleme des VRP

2.3.1.1 Vehicle Routing Problem with Backhauls

Das VRPB[6] ist eine Erweiterung des CVRP in dem die Menge der Kunden $V \setminus \{0\}$ in 2 Teilmengen aufgeteilt wird. Die erste Menge L beinhaltet Kunden, denen eine bestimmte Menge

[6]Vehicle Routing Problem with Backhauls

an Nachfrage $d_i > 0$ zugeordnet ist, während die zweite Menge B die Kunden verwaltet, bei denen Waren abgeholt werden müssen. Letzteren werden Nachfragen $d_i < 0$ zugeordnet. Die Kunden der Menge B dürfen in der ursprünglichen Problemdefinition auf einer Tour erst dann angefahren werden, nachdem die Kunden aus Menge L von der jeweiligen Tour bedient wurden. Eine Rechtfertigung für diese Forderung ist die allgemeine Schwierigkeit der Umpositionierung der Waren auf einem Fahrzeug, um den Zugriff auf die restlichen Zustellungen zu wahren. Gilt $B = \emptyset$, so entspricht das VRPB dem CVRP.

2.3.1.2 Vehicle Routing Problem with Time Windows

Oft wird die Anforderung gestellt, dass Kunden innerhalb eines definierten Zeitfensters beliefert werden müssen. In dem VRPTW[7] wird jeder Kunde mit einem Zeitintervall $[a_i, b_i]$ versehen. Dabei definiert a_i die frühest- und b_i die spätestmöglichste Zustellungszeit $a_i \leq b_i \quad \forall v_i \in V \setminus \{v_0\}$.
Zur Ergänzung des ursprünglichen CVRP soll auch noch jedem Kunden i eine Servicezeit s_i zugeordnet werden, welche die benötigte Zustellzeit bei diesem festlegt. Das Fahrzeug muss mit der Entladung innerhalb des gegebenen Zeitfensters beginnen und darf erst nach Ablauf der Servicezeit seine Weiterfahrt antreten. Dadurch ergibt sich die zusätzliche Nebenbedingung:

$$a_i \sum_{j \in V} x_{ijk} \leq w_{ik} \leq b_i \sum_{j \in V} x_{ijk} \qquad \forall i \in V, \forall k \in K \tag{2.16}$$

Zusätzlich muss bei dieser Erweiterung die bisher verwendete Nebenbedingung 2.14 aufgrund der Servicezeiten s_i folgendermaßen modifiziert werden (siehe: [23])

$$w_{ik} + s_i + t_{ij} - M(1 - x_{ijk}) \leq w_{jk} \qquad \forall (i,j) \in [V]^2, \forall k \in K \tag{2.17}$$

2.3.1.3 Vehicle Routing Problem with Pickup and Delivery

Bei dem VRPPD[8] handelt es sich um eine Erweiterung des Problems, in der ein Transportauftrag aus einem Startknoten O_i, einem Zielknoten D_i sowie einer Transportmenge d_i besteht. Daraus ergibt sich, dass neben der Einhaltung der Fahrzeugkapazitäten zusätzlich auch die Startknoten vor den Zielknoten auf der selben Tour angefahren werden müssen. Das VRPPD stellt eine starke Modifikation des Ausgangsproblems dar, und wird in der Literatur auch oft als PDP[9] bezeichnet.

[7] Vehicle Routing Problem with Time Windows
[8] Vehicle Routing Problem with Pickup and Delivery
[9] Pickup and Delivery Problem

2.3.1.4 Distance-Constrained Vehicle Routing Problem

In der Variante des DCVRP[10] wird bei jeder Tour die kapazitive Restriktion durch eine maximale Längen- bzw. Kostenrestriktion c_{max} ersetzt. Dadurch ergibt sich folgende Nebenbedingung:

$$\sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ijk} \leq c_{max} \qquad \forall k \in K \tag{2.18}$$

Mithilfe dieses Modells können maximal erlaubte Fahr- und Lenkzeiten bei der Disposition berücksichtigt werden. Natürlich kann die maximale Längenrestriktion zusätzlich auf die verschiedenen Formen des VRP angewendet werden, ohne die Nebenbedingung 2.10 entfernen zu müssen.

[10] Distance-Constrained Vehicle Routing Problem

3 Lösungsansätze für das Traveling Salesman Problem

Wie bereits in Kapitel 2.2 erwähnt, werden einige bekannte Verfahren zur Lösung des TSP beschrieben. Dabei wird von einem vollständigen metrischen Graphen K_n ausgegangen, wobei n für die Anzahl der Knoten V steht, welche von dem Handelsreisenden besucht werden müssen. Aufgrund der Tatsache, dass das TSP ein Teilproblem des VRP ist und benötigt wird, um das Routing der Fahrzeuge vorzunehmen, sollen vor allem Algorithmen mit kürzerer Laufzeit eingesetzt werden.

3.1 Exakte Lösungsverfahren

Das TSP kann mit exakten Verfahren optimal gelöst werden. Jedoch sind diese aufgrund der hohen Rechenzeit für Tourenoptimierungen meist schlecht geeignet. Eventuell könnte bei der Lösung eines VRP eine exakte Optimierung der einzelnen Touren am Schluss vorgenommen werden. Daher sollen diese kurz erwähnt werden.
Zu den exakten Lösungsverfahren zählen:

Brute Force Methode

Es werden alle möglichen Permutationsfolgen der Knoten erstellt, um so die Rundreise mit der kürzesten Länge zu identifizieren. Da es hierbei bei einem ungerichteten vollständigen Graphen $(n-1)!/2$ verschiedene zu untersuchende Möglichkeiten gibt, kann diese Methode nur bei sehr kleinen Problemen eingesetzt werden.

Branch and Bound

Branch and Bound ist ein Verfahren, mit dem eine vollständige Enumeration aller Lösungen simuliert werden kann, ohne sie alle einzeln betrachten zu müssen. Für viele NP-schwere kombinatorische Optimierungsprobleme ist Branch and Bound der beste Ansatz, um zu einer optimalen Lösung zu gelangen.
Bei der folgenden allgemeinen Formulierung des Branch and Bound Verfahrens wird auf die Anleitung von [18] zurückgegriffen.

Um den Branch and Bound Ansatz auf ein kombinatorisches Optimierungsproblem anzuwenden, werden zwei Schritte benötigt:

- Der Branch Schritt: Eine gegebene Teilmenge der möglichen Lösungen kann in mindestens zwei nichtleere Teilmengen partitioniert werden.
- Der Bound Schritt: Für eine durch Branch-Iteration gewonnene Teilmenge kann eine untere Schranke für die Kosten der in ihr liegenden Lösung berechnet werden.

Das allgemeine Verfahren wird in Algorithmus 2 beschrieben. Der Algorithmus erzeugt einen *gewurzelten Baum* T[11], dessen Knoten $v \in T$ Lösungsmengen der betrachteten Probleminstanz darstellen.

Algorithmus 2 Branch and Bound

Input: Eine Instanz eines Minimierungsproblems.
Output: Eine optimale Lösung S^*.

1:
Setze den Anfangsbaum $T := (\{S\}, \emptyset)$ fest, wobei S die Menge der zulässigen Lösungen ist.
Markiere S als aktiv
Setze die anfängliche obere Schranke $U := \infty$ fest (oder wende eine Heuristik an, um einen besseren Wert für die obere Schranke zu erhalten).
2:
Wähle einen aktiven Knoten v des Baumes T (gibt es keinen solchen dann **stop**).
Markiere v als nicht aktiv.
Branch: Bestimme eine Partition $v = v_1 \dot\cup ... \dot\cup v_t$.
3:
for $i := 0$ **to** t **do**
  Bound: Bestimme eine untere Schranke L für die Kosten einer jeden Lösung in v_i.
  if $|v_i| = 1$ und $cost(v_i) < U$ **then**
    Setze $U := cost(v_i)$ und $S^* := v_i$
  end if
  if $|v_i| > 1$ und $L < U$ **then**
    Setze $T := (V(T) \cup \{v_i\}, E(T) \cup \{\{v, v_i\}\})$ und markiere v_i als aktiv.
  end if
end for
4: Go to **2**

Abbildung 3.1 zeigt einen Branch and Bound Baum, so wie er durch das Verfahren erstellt werden könnte. Die blauen Knoten sind jene, die als aktiv gesetzt wurden, und noch von Algorithmus 2 im zweiten Schritt abgearbeitet werden.
Das Branch and Bound Verfahren findet immer eine beste Lösung für das vorliegende Minimierungsproblem. Die Implementierungen und die Effizienz hängen natürlich immer von dem vorhandenen Problem ab.

[11] Ein gewurzelter Baum ist ein zusammenhängender kreisfreier Graph mit einem ausgezeichneten Knoten der als Wurzel bezeichnet wird.

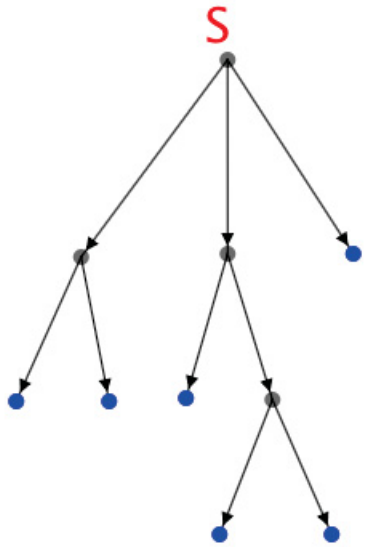


Abbildung 3.1: Branch and Bound Baum

Abbildung 3.2 zeigt anhand einer vollständigen Partitionierung, wie der Partitionierungsvorgang aus Algorithmus 2 (Schritt 2) für ein TSP durchgeführt werden kann. Dabei wird eine einfache Variante des Branch Schrittes vorgestellt. Die dabei betrachtete Probleminstanz besteht aus einem Depot v_0 und 3 weitere Knoten v_1, v_2 und v_3, welche zu besuchen sind. Die einzelnen Knoten in diesem Baum stehen für die jeweiligen Lösungsmengen. Über den Knoten wird immer derjenige Kunde angezeigt, welcher in die Lösungsmenge an der entsprechenden Position fixiert wird. Die Wurzel in Ebene 1 enthält alle 6 verschiedenen möglichen Lösungen.
Der linke Knoten aus Ebene 2 enthält die Lösungen: $\{\{v_0, v_1, v_2, v_3, v_0\}, \{v_0, v_1, v_3, v_2, v_0\}\}$. Der mittlere Knoten aus Ebene 2: $\{\{v_0, v_2, v_1, v_3, v_0\}, \{v_0, v_2, v_3, v_1, v_0\}\}$. Der rechte Knoten aus Ebene 2: $\{\{v_0, v_3, v_1, v_2, v_0\}, \{v_0, v_3, v_2, v_1, v_0\}\}$. Die Knoten aus Ebene 3 bestehen jeweils nur aus einer einzigen Lösung. Wie aus dem Baum ersichtlich ist, repräsentiert der Knoten ganz rechts in Ebene 3 den Lösungsraum: $\{\{v_0, v_3, v_2, v_1, v_0\}\}$. Diese Knoten der untersten Ebene können nicht weiter in Teilmengen partitioniert werden, und werden als Blätter des Baumes T bezeichnet.
Der Bound Schritt für ein TSP kann mit verschiedenen Verfahren aus [18] durchgeführt werden. Dabei wird eine untere Schranke für die Kosten der entsprechenden Teilmenge berechnet.

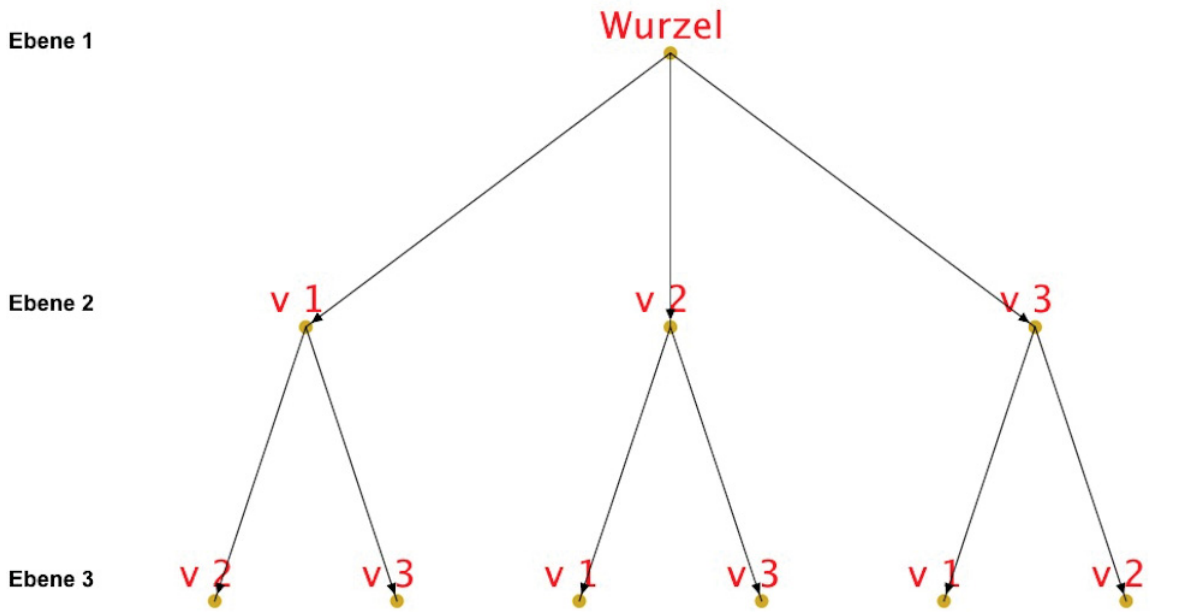


Abbildung 3.2: Vollständige TSP Partitionierung für eine gegebene Probleminstanz

3.2 Heuristiken

Der große Vorteil der Heuristiken gegenüber den exakten Verfahren liegt darin, dass in kurzen Rechenzeiten brauchbare Lösungen erzielt werden können. Daher werden diese für das Routing gegenüber den exakten Verfahren bevorzugt. Die dabei erzeugten Lösungen können natürlich auch weit von der optimalen Lösung entfernt sein.
Für das TSP können prinzipiell zwei Haupttypen von Heuristiken unterschieden werden (siehe [31]):

- Eröffnungsheuristiken
- Verbesserungsheuristiken (Lokale Suche)

3.2.1 Eröffnungsheuristiken für das Traveling Salesman Problem

Eröffnungsheuristiken haben die Aufgabe eine Startlösung $T = \{v_0, v_1, ..., v_{n-1}, v_0\}$ zu bilden, welche im Anschluss noch mit *lokalen Suchverfahren* verbessert werden kann.

3.2.1.1 Nearest Neighbor Heuristik

Die Nearest Neighbor Heuristik ist eines der einfachsten Verfahren um eine Startlösung für das TSP zu erhalten. Der Ablauf kann in Algorithmus 3 gefunden werden. Zusätzlich zeigt

Abbildung 3.3 eine TSPLIB[12] Instanz, auf die das Nearest Neighbour Verfahren angewendet wurde. Die euklidischen Abstände zwischen den einzelnen Knoten repräsentieren die Kantenbewertungen.
Die offensichtliche Schwäche dieser Heuristik ist, dass die Distanz zwischen der Ausgangsstadt und der letzten besuchten Stadt bis zuletzt nicht berücksichtigt wird.
Nach [31] gilt für die mit der Nearest Neighbor Heuristik gefundenen Lösung $l(T)$ und die optimale Lösung $l(T_{opt})$ die Schranke:

$$\frac{l(T)}{l(T_{opt})} \leq \frac{(\log_2 n)}{2}$$

Algorithmus 3 Nearest Neighbor Heuristik

Input: $G(V, E)$
Output: Eine Rundreise T

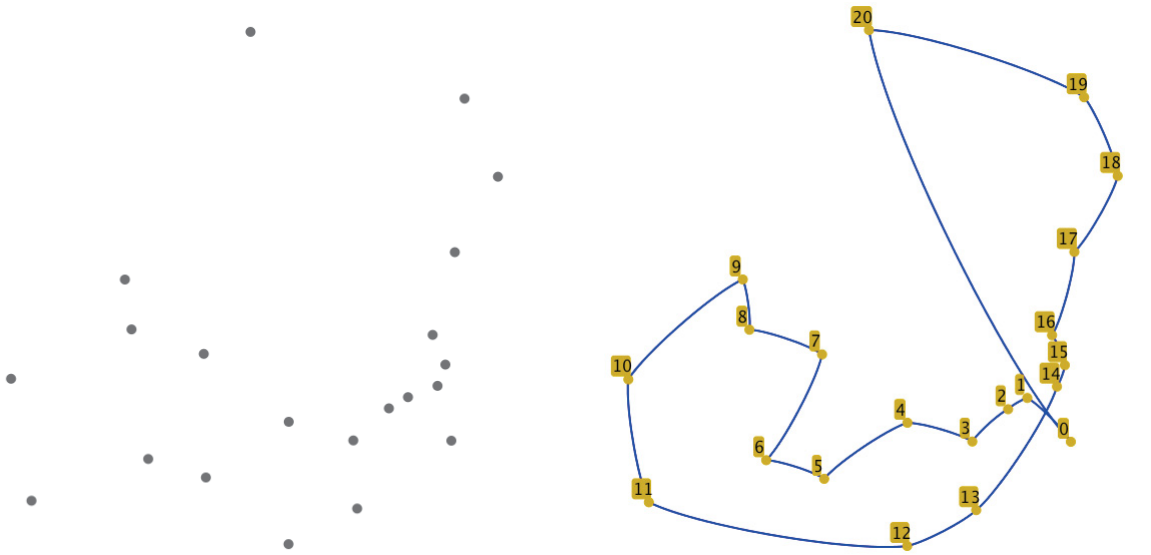
1:
Wähle zufällig einen Startknoten v_0
Zu Besuchende Knoten $O := V \setminus \{v_0\}$
Besuchte Knoten $B := \{v_0\}$
2:
while $O \neq \emptyset$ **do**
Bestimme den Knoten v, der am nächsten zu dem zuletzt hinzugefügten Knoten in B liegt.
$O := O \setminus \{v\}$
$B := B \cup \{v\}$
end while
3:
Rundreise $T := B \cup \{v_0\}$

3.2.1.2 Nearest Insertion Heuristik

Algorithmus 4 zeigt die Funktionsweise der Nearest Insertion Heuristik. Diese weist im Allgemeinen bessere Ergebnisse als die Nearest Neighbor Heuristik auf, welches durch Abbildung 3.4 für diese Instanz bestätigt werden kann.

$$\text{Obere Schranke: } \frac{l(T)}{l(T_{opt})} \leq 2$$

[12] TSPLIB ist eine Sammlung von TSP Instanzen, welche von der Ruprecht Karl Universität in Heidelberg online zur Verfügung gestellt wird



((a)) Instanz ((b)) Lösung mit Nearest Neighbor: $l(T) = 3333$

Abbildung 3.3: Lösung der TSP Instanz Gr21 mit dem Nearest Neighbor Verfahren

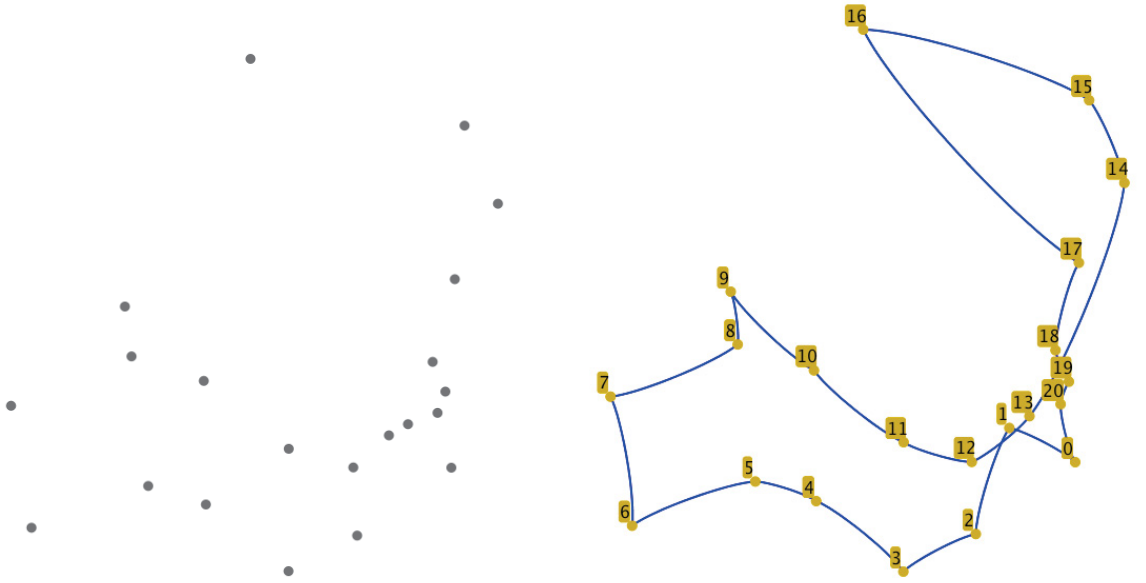
Algorithmus 4 Nearest Insertion Heuristik

Input: $G(V, E)$
Output: Eine Rundreise T

1:
Wähle zufällig einen Startknoten v_0
Zu besuchende Knoten $O := V \setminus \{v_0\}$
Rundreise $T := \{v_0, v_0\}$
2:
while $O \neq \emptyset$ **do**
Wähle den Knoten $v_k \notin T$, welcher die geringste Entfernung zu einem der Knoten aus T aufweist.
$O := O \setminus \{v_k\}$
Finde jene Kante $(v_i, v_j) \in T$, sodass $c_{ik} + c_{kj} - c_{ij}$ minimal wird. Füge v_k zwischen v_i und v_j ein.
end while

3.2.1.3 Farthest Insertion Heuristik

Die Farthest Insertion Heuristik unterscheidet sich lediglich durch die Auswahl des nächsten Knotens v von der Nearest Insertion Heuristik aus Algorithmus 3.4. Hierbei wird nun der Knoten v gewählt, welcher zu den Knoten der Tour T die größte Distanz aufweist. Abbildung 3.5 zeigt, dass mit dieser Heuristik ein besseres Ergebnisse als mit der Nearest Insertion Heuristik für die Testinstanz erzielt wird.
Nach den Testergebnissen von [5] für Probleme mit 50 Knoten und euklidischen Daten im Einheitsquadrat liegen die Ergebnisse der Nearest Insertion Heuristik um 7 bis 22% oberhalb



((a)) Instanz ((b)) Lösung mit Nearest Insertion: $l(T) = 3121$

Abbildung 3.4: Lösung der TSP Instanz Gr21 mit dem Nearest Insertion Verfahren

der Lösungen, die mit der Farthest Insertion Heuristik erzielt werden.

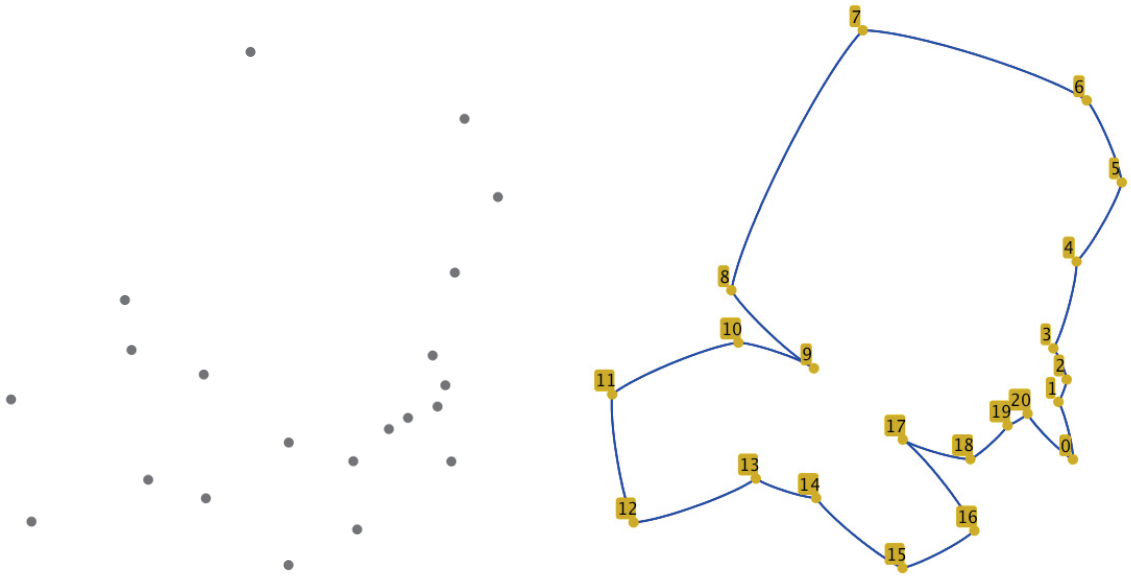
$$\text{Obere Schranke: } \frac{l(T)}{l(T_{opt})} \leq \log_2 n$$

3.2.1.4 Random Insertion Heuristik

Bei der Random Insertion Heuristik wird im Gegensatz zu der Nearest Insertion Heuristik und der Farthest Insertion Heuristik immer ein zufälliger nächster Knoten v für den Einfügeschritt gewählt. Die Testergebnisse von [5] zeigen, dass für ein großes TSP mit 2000 Knoten im Einheitsquadrat die Random Insertion Heuristik kaum schlechtere Ergebnisse als die Farthest Insertion Heuristik erzielt. Die Random Insertion Heuristik besitzt aber zusätzlich noch den Vorteil, dass für die Auswahl eines Knotens im Unterschied zu der Farthest Insertion Heuristik eine geringe, von n unabhängige Rechenzeit benötigt wird.

3.2.2 Verbesserungsheuristiken für das Traveling Salesman Problem

Im Allgemeinen sind Verbesserungsheuristiken bzw. *lokale Suchverfahren* die in der Praxis erfolgreichsten Verfahren, um gute Lösungen für Instanzen des TSP zu erhalten. Die Idee der lokalen Suche ist die folgende: Begonnen wird mit einer durch eine Eröffnungsheuristik erstellten Tour. Dann wird versucht, diese durch lokale Änderungen zu verbessern.
Der in Algorithmus 5 vorgestellte allgemeine Grundrahmen lokaler Suche ist unabhängig von einer speziellen Problemstellung verwendbar, während eine geeignete Wahl der Operatoren zur Erzeugung der Nachbarschaftslösungen von problemspezifischen Aspekten abhängt. [42]



((a)) Instanz ((b)) Farthest Insertion: $l(T) = 2707$

Abbildung 3.5: Lösung der TSP Instanz Gr21 mit dem Farthest Insertion Verfahren

Algorithmus 5 Lokale Suche

Input: Ausgangslösung x

```
while Abbruchkriterium nicht erreicht do
  Erzeuge Nachbarlösungen N(x)
  Wähle x' ∈ N(x) so, dass f(x') ≤ f(x*) ∀x* ∈ N(x)
  if f(x') < f(x) then
    x ← x'
  else
    Abbruch
  end if
end while
```

In Algorithmus 5 werden zunächst alle möglichen Nachbarlösungen bezüglich eines Nachbarschaftsoperators erstellt, und dann dasjenige $x^{'} \in N(x)$ gewählt, welches die beste Lösung erzielt. Gilt $f(x^{'}) < f(x)$ so wird $x^{'}$ zur neuen Ausgangslösung. Diese Vorgangsweise wird als *Best Improvement* Strategie bezeichnet.
Andererseits kann in einem sukzessiven Vorgehen schrittweise ein Nachbar x^* von x erzeugt werden, und direkt der Vergleich vorgenommen werden. Gilt $f(x^*) < f(x)$ so wird x^* zur neuen Ausgangslösung. Dieses sukzessive Vorgehen wird als *First Improvement* Strategie bezeichnet. Während Best Improvement Strategien bezüglich der erzielten Lösungsqualität tendenziell favorisiert werden, kommen First Improvement Strategien vermehrt zum Einsatz, wenn die Evaluation aller Nachbarn zu rechenintensiv erscheint.
Sobald keine neue beste Lösung aus der Nachbarschaft $N(x)$ gefunden werden kann, bricht der Algorithmus ab. Zusätzlich können lokale Suchverfahren auch abbrechen, sobald eine festgelegte maximale Rechenzeit überschritten wurde.

3.2.2.1 k-Opt Heuristik

Bei der k-Opt Heuristik [20] werden Nachbarschaftslösungen erzeugt, indem k Kanten aus einer gegebenen Tour entfernt und durch k andere Kanten so ersetzt werden, dass sich wieder eine gültige Tour ergibt.
Algorithmus 6 zeigt eine k-Opt Heuristik Umsetzung, bei der eine First Improvement Strategie verfolgt wird. Der Abbruch erfolgt, nachdem bei einer gegebenen Tour T keine Verbesserung mehr erzielt werden kann. [18]

Algorithmus 6 k-Opt Heuristik

Input: Ausgangslösung T

```
S ← Familie der k-elementigen Teilmengen vonE(T)
for all s ∈ S do
  X ← Menge aller gültiger Touren T' mit E(T') ⊇ E(T) \ s
  for all T' ∈ X do
    if c(E(T')) < c(E(T)) then
      T ← T'
      Starte Algorithmus von vorne.
    end if
  end for
end for
```

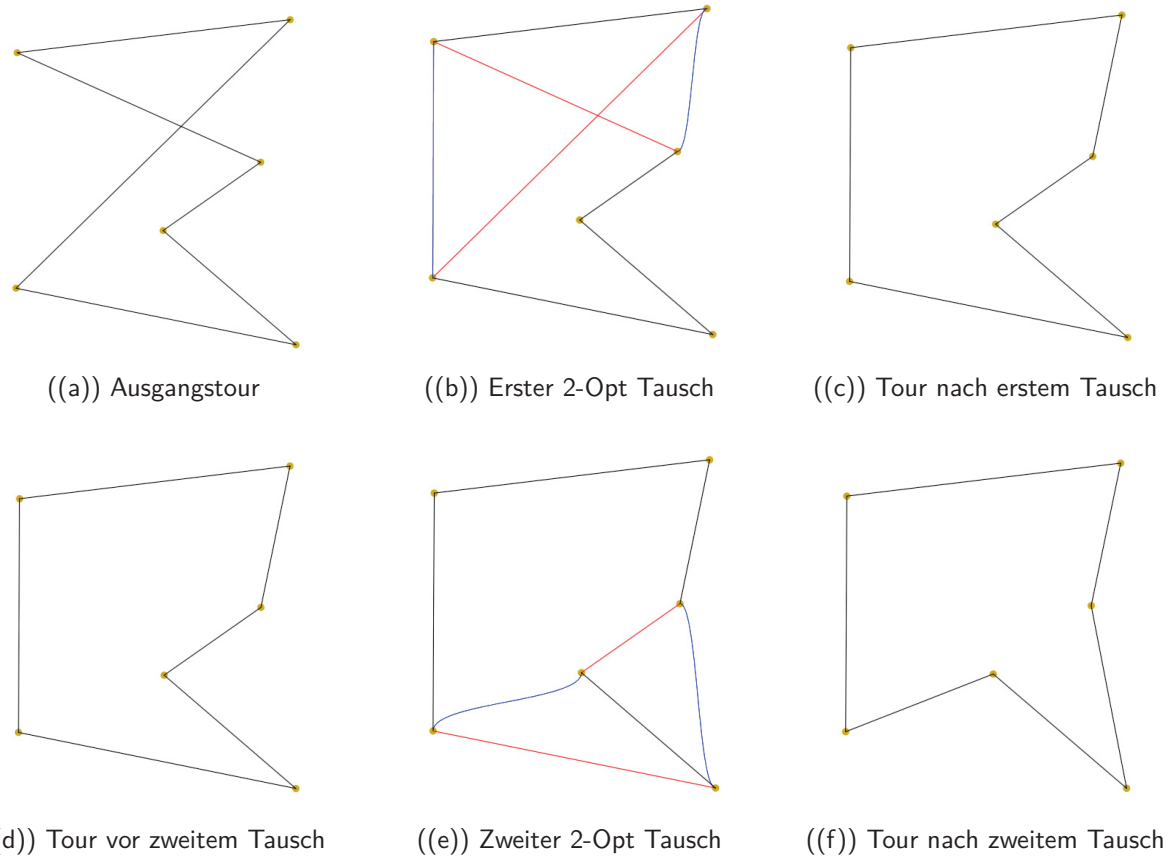
Im folgenden wird eine 2-Opt Heuristik betrachtet um dieses lokale Suchverfahren besser zu beschreiben. In Algorithmus 7 wird dabei die Vorgangsweise veranschaulicht, in der die Kosten zwei vorhandener Kanten $s \in S$ mit den Kosten zwei neuer Kanten $s^* \in S$ verglichen werden. Sind die Kosten von s^* geringer, so wird ein Kantenaustausch vorgenommen und die lokale Suche startet von vorne. Dieser Austausch wird in Abbildung 3.6 anhand eines Beispiels dargestellt. Dabei stellen die roten Kanten die Menge s und die blauen Kanten die Menge s^* dar. Kann durch einen Tausch zweier Kanten keine weitere Verbesserung erzielt werden, so wird der Algorithmus abgebrochen. Die daraus gewonnene Tour wird als 2-Opt bezeichnet.
Obere Schranke für die 2-Opt Heuristik laut [2]:

$$\frac{l(T)}{l(T_{opt})} \leq 4 \cdot \sqrt{n}$$

Algorithmus 7 2-Opt Heuristik

Input: Ausgangslösung T

$S \leftarrow$ alle möglichen Kantenpaare von $E(T)$ die in T nicht direkt aufeinanderfolgen.
for all $s \in S$ **do**
  $s^* \leftarrow$ Erzeuge neues Kantenpaar durch Vertauschung der Start- und Endknoten der Kanten von s
  if $c(s^*) < c(s)$ **then**
    $E(T) \leftarrow E(T) \dot{\cup} s^* \setminus s$
    Starte Algorithmus von vorne.
  end if
end for

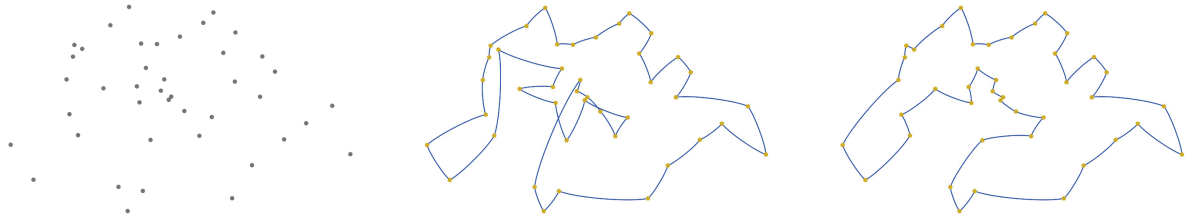


((a)) Ausgangstour ((b)) Erster 2-Opt Tausch ((c)) Tour nach erstem Tausch

((d)) Tour vor zweitem Tausch ((e)) Zweiter 2-Opt Tausch ((f)) Tour nach zweitem Tausch

Abbildung 3.6: 2-Opt Tausch anhand einer Beispieltour

Die 2-Opt Heuristik liefert bei kleinen TSP bereits sehr gute Ergebnisse. Natürlich sind diese aber auch immer von der jeweiligen Startlösung T abhängig. Abbildung 3.7 zeigt eine TSPLIB Instanz, die mithilfe der 2-Opt Heuristik gelöst wurde. Dabei wurde die Starttour mithilfe der Nearest Insertion Heuristik erzeugt.
Ein bedeutender Nachteil der k-Opt Heuristik ist, dass k im Vorhinein gewählt werden muss. Es ist schwer zu beurteilen, welches k den besten Kompromiss hinsichtlich der Rechenzeit und der Qualität der Lösung bietet. Laut [20] ist k = 3 häufig eine günstige Wahl, die die besten Ergebnisse im Verhältnis zum Rechenaufwand erzielt.



((a)) Instanz

((b)) Tour nach Nearest Insertion: $l(T) = 1522$

((c)) Tour nach 2-Opt optimierung: $l(T) = 1302$

Abbildung 3.7: Lösung der TSP Instanz Swiss 42 mit dem 2-Opt Verfahren

3.2.2.2 Lin-Kernighan Heuristik

Die *Lin-Kernighan Heuristik* [33] modifiziert die k-Opt Heuristik, indem ein variables k eingesetzt wird. Anstatt k im vorhinein festzulegen, entscheidet die Heuristik in jeder Iteration, welches k zu wählen ist. Das bedeutet, dass den beiden zu vertauschenden Kantenmengen jeweils eine neue Kante hinzugefügt wird, bis eine Verbesserung der Tour gefunden werden kann, oder das maximale k erreicht wird. Dabei kann die benötigte Rechenzeit noch vermindert werden, indem die möglichen Kanten auf die nächsten fünf Nachbarn beschränkt werden. Eine allgemeine Beschreibung der Lin-Kernighan Heuristik kann beispielsweise in [16] gefunden werden: Der Algorithmus versucht in jedem Iterationsschritt zwei Kantenmengen $X = \{x_1, ..., x_k\} \quad x_i \in E(T)$ und $Y = \{y_1, ..., y_k\} \quad y_i \notin E(T)$ für $1 \leq i \leq k$ zu bilden, sodass durch Vertauschung der Kantenmengen eine gültige und kürzere Tour entsteht. Dabei müssen die beiden Mengen folgende Kriterien erfüllen:

- $x_i \in X$ und $y_i \in Y$ müssen einen gemeinsamen Knoten haben.
- $x_{i+1} \in X$ und $y_i \in Y$ müssen ebenfalls einen gemeinsamen Knoten haben.
- Es ist notwendig y_i so zu wählen, sodass ein positives G_i entsteht. $G_i = g_1 + g_2 + ... + g_i$. Die Differenz $g_i = c(x_i) - c(y_i)$ definiert die Kosteneinsparung, wenn x_i durch y_i ersetzt wird.
- X und Y müssen disjunkt sein.

Allgemein gilt: $x_i = \{v_{2i-1}, v_{2i}\}$, $y_i = \{v_{2i}, v_{2i+1}\}$ und $x_{i+1} = \{v_{2i+1}, v_{2i+2}\}$ für $i \geq 1$. Für $i = 1$ ergibt sich daher $x_1 = \{v_1, v_2\}$, $y_1 = \{v_2, v_3\}$ und $x_2 = \{v_3, v_4\}$. Mit v_i werden während des Algorithmus beliebige Knoten definiert. Diese Knotennummerierung hat nichts mit der Knotenfolge der Ausgangstour T gemeinsam. Algorithmus 8 beschreibt den Grundablauf der Lin-Kernighan Heuristik, für welchen wieder eine Ausgangstour T benötigt wird.

Algorithmus 8 Lin-Kernighan Heuristik

Input: Ausgangslösung T

1: $i \leftarrow 1$. Wähle einen beliebigen Knoten $v_1 \in V(T)$.
2: Wähle eine Kante $x_1 = \{v_1, v_2\} \in E(T)$. (Hierbei gibt es 2 verschiedene Möglichkeiten)
3: Wähle eine Kante $y_1 = \{v_2, v_3\} \notin E(T)$. Sodass $G_1 > 0$. Ist dies nicht möglich, so gehe zu Schritt 11.
4: $i \leftarrow i + 1$.
5: Wähle eine Kante $x_i = \{v_{2i-1}, v_{2i}\} \in E(T)$, sodass $x_i \neq y_s$ für alle $s < i$.
$e \leftarrow \{v_{2i}, v_1\}$
$T' \leftarrow$ Ersetze in $E(T)$ die Kanten X durch $Y \dot{\cup} e$
if $l(T') < l(T)$ **then**
$T \leftarrow T'$
Gehe zurück zu Schritt 1. (Hier wurde eine Verbesserung der Tour gefunden, wodurch der Algorithmus mit der neuen Ausgangslösung von vorne startet).
end if
6: Wähle eine Kante $y_i = \{v_{2i}, v_{2i+1}\} \notin E(T)$, sodass:
$G_i > 0$
$y_i \neq x_s$ für alle $s \leq i$.
x_{i+1} existiert.
Wenn ein y_i unter diesen Voraussetzungen existiert, dann gehe zu Schritt 4.
7: Gibt es noch eine offene Alternative für y_2, dann setze $i \leftarrow 2$ und gehe zu Schritt 6.
8: Gibt es noch eine offene Alternative für x_2, dann setze $i \leftarrow 2$ und gehe zu Schritt 5.
9: Gibt es noch eine offene Alternative für y_1, dann setze $i \leftarrow 1$ und gehe zu Schritt 3.
10: Gibt es noch eine offene Alternative für x_1, dann setze $i \leftarrow 1$ und gehe zu Schritt 2.
11: Gibt es noch eine offene Alternative für v_1, dann gehe zu Schritt 1.
12: Ende.

Der Algorithmus terminiert mit einer Lösung T, wenn für alle möglichen Knoten v_1 keine Verbesserung mehr erzielt werden kann.
Bis heute ist die Lin-Kernighan Heuristik eine der effektivsten Methoden um optimale beziehungsweise fast optimale Lösungen für das TSP zu erhalten.
In [16] wird des Weiteren auch noch im Detail gezeigt, durch welche Maßnahmen die Lin-Kernighan Heuristik in Bezug auf Rechenzeit optimiert werden kann.

3.2.2.3 Insertion Search Heuristik

Die *Insertion Search* Heuristik ist ein lokales Suchverfahren, dass sich aufgrund der niedrigen Rechenzeit für das Routing der Touren besonders gut eignet. Dieses Verfahren versucht durch Verschiebung einzelner Knoten in einem definierten Einsetzungsintervall eine Verbesserung der Tourenlänge zu erzielen. Algorithmus 9 veranschaulicht den Ablauf dieser Heuristik, bei der die Nachbarschaftslösungen durch eine stark eingeschränkte 3-Opt Änderung erzeugt werden. Da bei Verschiebung eines Knoten s auf eine andere Position drei Kanten entfernt und durch drei neue ersetzt werden.

Algorithmus 9 Insertion Search

Input: (Ausgangslösung T, Reichweite α)

$S \leftarrow$ definiere eine Suchliste, die aus allen zu besuchenden Knoten von T besteht. (n Knoten)
$l_a = l(T)$
while $S \neq \emptyset$ **do**
  $s \leftarrow$ wähle zufällig einen Knoten $s \in S$
  $S \leftarrow S \setminus s$
  $i \leftarrow$ Index des Knoten s in T
  $T \leftarrow$ Bestimme die beste Nachbarschaftslösung von T, die durch Verschiebung des Knoten s innerhalb des Einsetzungsintervall $[max\{i-\alpha, 1\}, min\{i+\alpha, n\}]$ erzeugt werden kann. (Im schlechtesten Fall bleibt T unverändert)
end while
if $l(T) < l_a$ **then**
  Starte Algorithmus von vorne.
else
  Ende.
end if

4 Lösungsansätze für das Vehicle Routing Problem

4.1 Exakte Verfahren für das Vehicle Routing Problem

Eine optimale Lösung eines VRP durch exakte Verfahren ist laut [39] nur für kleine Probleminstanzen (Kundenanzahl kleiner gleich 50) effektiv möglich. Zu deren Lösung können Branch and Bound Verfahren angewendet werden, welche bereits in Kapitel 3.1 vorgestellt wurden. Da sich diese Arbeit mit Probleminstanzen beschäftigt, die höhere Kundenanzahlen als 50 aufweisen, sollen die exakten Verfahren nicht näher untersucht werden. In [39] können exakte Verfahren für die verschiedenen VRP Formen aus Kapitel 2.3.1 gefunden werden.

4.2 Heuristische Verfahren für das Vehicle Routing Problem

In Kapitel 2.3 wurde gezeigt, dass das VRP aus zwei Teilproblemen besteht:

- Ein Zuordnungsproblem von Kunden zu einer Tour: Clustering
- Ein Routing Problem für jede Tour, dass heißt die Bestimmung einer Rundreise für jedes Fahrzeug.

Heuristische Verfahren zur Lösung des VRP können somit in *Sukzessiv-* und *Parallelverfahren* unterteilt werden. [35]

Sukzessiv-Verfahren

Grundsätzlich werden bei den Sukzessiv-Verfahren zwei Typen unterschieden:

- *Route-First-Cluster-Second:* Zuerst wird mit allen Kundenknoten ein TSP gelöst, und so eine Giant Tour erstellt. Im nächsten Schritt wird diese Giant Tour in zulässige Fahrzeug Touren aufgeteilt.
- *Cluster-First-Route-Second:* Kunden werden unter Beachtung der Kapazität zu Touren zusammengefasst, und anschließend wird für jede Tour ein TSP Verfahren angewandt.

Parallelverfahren

Bei den Parallelverfahren erfolgt das Routing und das Clustering, wie der Name schon andeutet, parallel.
Dabei werden zwei Arten von Verfahren unterschieden:

- Konstruktionsverfahren: Erstellung einer Ausgangslösung.
- Verbesserungsverfahren: Verbesserungen des gesamten Tourenplans

4.3 Eröffnungsheuristiken für das Vehicle Routing Problem

4.3.1 Das Clark and Wright Savings-Verfahren

Das *Clarke and Wright Savingsverfahren* wurde bereits 1964 vorgestellt [40]. Das Verfahren zählt zu den am häufigsten eingesetzten heuristischen Lösungsverfahren für das VRP. Es zählt zu den Parallelverfahren, weil die Zuordnung zum Fahrzeug und die Reihenfolgebildung für das Besuchen der Kunden simultan erfolgen.
Das Verfahren beginnt mit einer Anfangslösung $L = \{T_1, T_2, .., T_n\}$, bei der jeder Kunde i durch eine Tour $T_i = \{0, i, 0\}$ einzeln vom Depot aus angefahren wird. Diese Tour wird als Pendeltour bezeichnet. Im Anschluss werden dann die *Saving* Werte für $i, j = 1, ..., n$ und $i \neq j$ berechnet:

$$s_{ij} = c_{i0} + c_{0j} - c_{ij}$$

Der Saving Wert s_{ij} definiert die Kosten, die durch Vereinigung der beiden Touren, welche am Knoten i endet beziehungsweise am Knoten j startet, eingespart werden. Die Formel soll anhand der folgenden Beispiele verdeutlicht werden. Abbildung 4.1(a) zeigt eine Ausgangssituation, in der die beiden Kunden durch die Pendeltouren $T_1 = \{0, 1, 0\}$ und $T_2 = \{0, 2, 0\}$ beliefert werden. In Abbildung 4.1(b) wurde T_1 mit T_2 vereint, wodurch die beiden Kunden durch die Tour $T_1 = \{0, 1, 2, 0\}$ bedient werden. Bei der Vereinigung der beiden Touren kann beobachtet werden, dass T_1 nun nicht mehr die Kante $\{1, 0\}$ durchläuft[13], wodurch die Kosten c_{10} eingespart werden. Des Weiteren wird die Kante $\{0, 2\}$ auch nicht mehr benützt, womit c_{02} eingespart wird. Zusätzlich muss nun jedoch $\{1, 2\}$ durchlaufen werden, welche in Abbildung 4.1(a) rot eingezeichnet ist, und somit werden die Einsparungen um c_{12} vermindert. Aus diesem Beispiel ist nun ersichtlich, dass der Saving Wert s_{12} wie folgt berechnet wird:

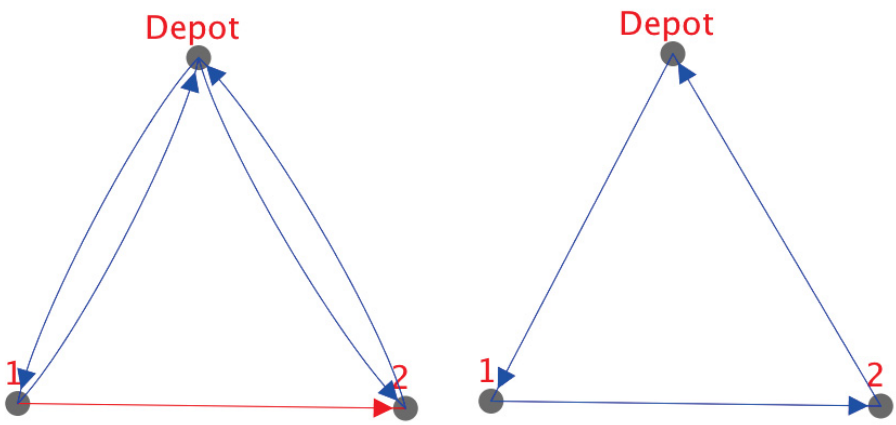
$$s_{12} = c_{10} + c_{02} - c_{12}$$

[13]Die Kante wird nicht mehr in der Richtung von 1 auf 0 durchlaufen.

Abbildung 4.2 zeigt wie zwei Touren mit mehreren Kunden vereint werden, so wie es im weiteren Verlauf während des Algorithmus 10 mit den Touren T_k und T_l durchgeführt wird. Dies ist natürlich nur dann erlaubt, wenn keine der Fahrzeugrestriktionen verletzt wird. Aus der Abbildung kann abgelesen werden, dass der Saving Wert sich folgendermaßen berechnet:

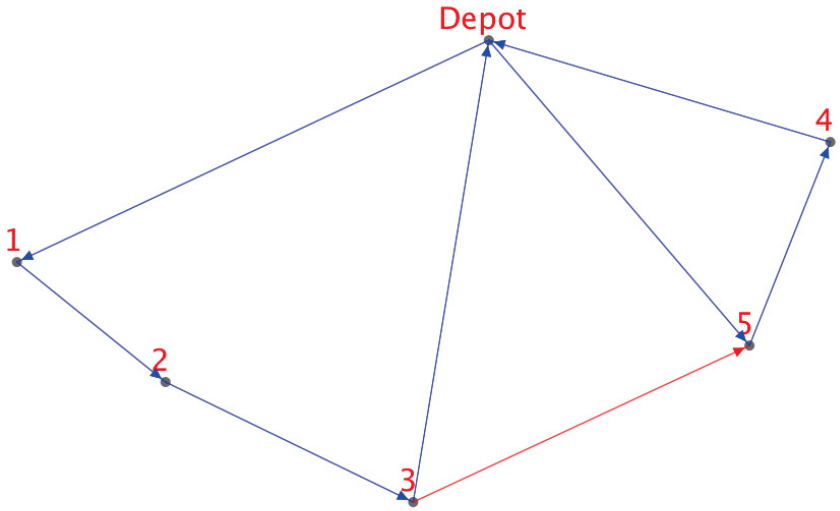
$$s_{35} = c_{30} + c_{05} - c_{35}$$

Da die Graphen der Probleminstanzen metrisch sind, gilt für alle Saving Werte: $s_{ij} \geq 0$.
Prinzipiell gibt es zwei verschiedene Arten des Savings Algorithmus, eine parallele (Algorithmus 10) und eine sequentielle Version (Algorithmus 11). In [39] wurde das parallele sowie das sequentielle Verfahren für die 14 VRP Instanzen von Christofides, Mingozzi und Toth getestet. Die Ergebnisse zeigten, dass das parallele Verfahren deutlich bessere Ergebnisse als das sequentielle erzielt. In keiner der getesteten Instanzen konnte die sequentielle Variante ein besseres Ergebnis erzielen.

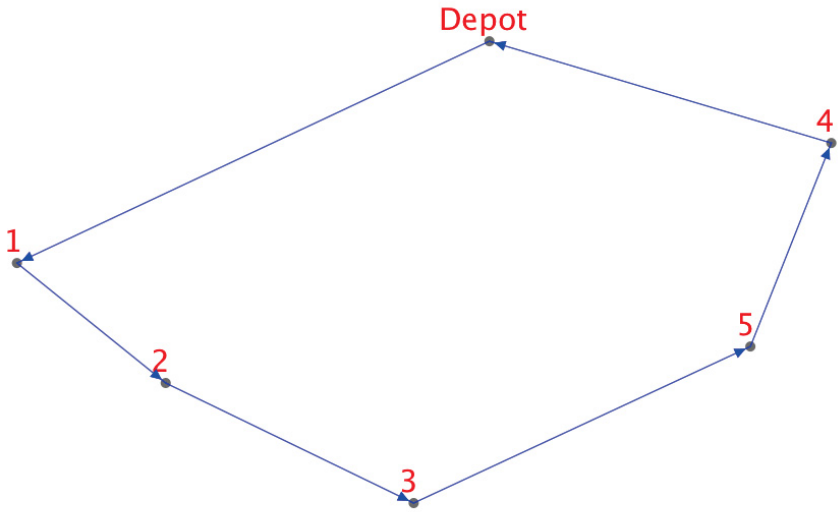


((a)) T_1 und T_2 vor Vereinigung ((b)) T_1 nach Vereinigung

Abbildung 4.1: Savings Verfahren: Vereinigung der Touren



((a)) T_k und T_l vor Vereinigung. Hierbei wird der Savingwert s_{35} betrachtet, da die erste Tour am Knoten 3 endet (enthält die Kante $\{3, 0\}$) und die zweite Tour am Knoten 5 beginnt (enthält die Kante $\{0, 5\}$.



((b)) T_k nach Vereinigung. Die Tour T_l wird hierbei aus L entfernt, da diese keine Kunden mehr bedient. Natürlich muss angemerkt werden, dass eine Vereinigung nur dann erlaubt ist, wenn keine der Fahrzeugrestriktionen wie zum Beispiel die maximale Kapazität verletzt wird.

Abbildung 4.2: Savings Verfahren: Vereinigung von Touren mit mehreren Knoten

Algorithmus 10 Paralleles Savings Verfahren

Input: $G(V, E)$
Output: Tourenplan L

1: $L \leftarrow$ bilde Pendeltouren.
2: $S \leftarrow$ berechne die Saving Werte für $i, j = 1, ..., n\ i \neq j$.
3: sortiere die Saving Werte absteigend.
for all $s_{ij} \in S$ **do**
  if existiert eine Tour $T_k \in L$ sodass $\{i, 0\} \in E(T_k)$ und existiert eine Tour $T_l \in L$ sodass $\{0, j\} \in E(T_l)$ und können T_k und T_l bezüglich der vorhandenen Restriktionen wie zum Beispiel der maximalen Ladekapazität vereint werden **then**
    Vereine die Touren T_k und T_l. (T_k wird um die Knoten von T_l erweitet, während T_l aus L entfernt wird)
  end if
end for

Algorithmus 11 Sequentielles Savings Verfahren

Input: $G(V, E)$
Output: Tourenplan L

1: $L \leftarrow$ bilde Pendeltouren.
2: $S \leftarrow$ berechne die Saving Werte für $i, j = 1, ..., n\ i \neq j$.
3: sortiere die Saving Werte absteigend.
for all $T \in L$ **do**
  4:
  for all $s_{ij} \in S$ **do**
    if $\{i, 0\} \in E(T)$ und $j \notin T$ und T kann mit $\{0, j, 0\}$ vereint werden **then**
      vereine T mit $\{0, j, 0\}$ (Knoten j wird an das Ende von T angehängt, während die Tour $\{0, j, 0\}$ aus L entfernt wird).
      springe zu Punkt 4.
    end if
    if $\{0, j\} \in E(T)$ und $i \notin T$ und T kann mit $\{0, i, 0\}$ vereint werden **then**
      vereine T mit $\{0, i, 0\}$ (Knoten i wird zu Beginn der Tour T eingefügt, während die Tour $\{0, i, 0\}$ aus L entfernt wird).
      springe zu Punkt 4.
    end if
  end for
end for

Zu den Algorithmen 10 und 11 muss zusätzlich erwähnt werden, dass sich die Abläufe auf Probleminstanzen beziehen, bei der ein gerichteten Graph vorliegt. Bei einem ungerichteter Graph kann die Reihenfolge der Knoten innerhalb einer Tour umgekehrt werden, ohne dass dies eine Auswirkung auf die Länge der Tour hätte. Somit kann zum Beispiel bei Algorithmus 10 nicht nur eine Tour T_k gesucht werden, welche am Knoten i endet, sondern auch dort startet. Das selbe gilt dann natürlich auch für die Tour T_l in Bezug auf den Knoten j.
Das Savings-Verfahren wurde in [11] erweitert, wonach die Saving Werte wie folgt berechnet werden:

$$s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}$$

Dabei ist λ ein Parameter mit dem die Form der Touren beeinflusst werden kann. Je größer λ ist, umso mehr werden die Kosten zwischen den zu verbindenden Knoten gewichtet. $\lambda = 0.4$ und $\lambda = 1.0$ sollen bei diesem Vorgehen laut [39] die besten Ergebnisse erzielen.
Trotz der von den Pendeltouren repräsentierten äußerst schlechten Startlösung gelangt das Savingsverfahren zu akzeptablen Lösungen. Die Touren sind wie Blätter einer Blüte um das Depot aufgebaut. Abbildung 4.3 zeigt einen Tourenplan, der mit dem parallelen Savingsverfahren erstellt wurde ($\lambda = 1.0$). Dafür wurde die Christofides, Mingozzi und Toth Instanz 4 verwendet.

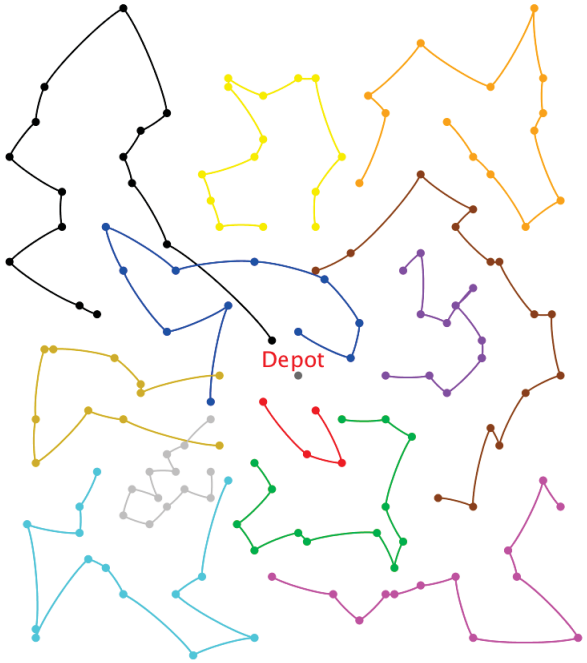


Abbildung 4.3: Erzeugter Tourenplan mit dem Savingsverfahren

4.3.2 Sweep Algorithmus

Der *Sweep Algorithmus* von [13] ist ein koordinatenorientiertes Verfahren, bei dem die Knoten in Form von kartesischen Koordinaten vorliegen. Der Standort des Depots liegt dabei im Ursprung des Koordinatensystems. Algorithmus 12 beschreibt den simplen Ablauf.
Das Verfahren zählt zu den sukzessiven Verfahren. In den Schritten 1-3 wird zuerst das Clustering durchgeführt, während Schritt 4 das Routing für die einzelnen Touren durchführt. Dieses geschieht mit den Lösungsverfahren für das TSP. Hier können die Heuristiken von Kapitel 3 eingesetzt werden. Das Ergebnis des Sweep Algorithmus ist auch von dem jeweiligen Startwinkel abhängig. Dadurch ist es möglich, verschiedene Tourenpläne durch verschiedene Startwinkel zu erzeugen. Im ersten Schritt von Algorithmus 12 ist ersichtlich, wie der Polarwinkel ϕ berechnet

wird. Dieser wird durch den Startwinkel beeinflusst, wodurch die Reihenfolge der Kundenknoten nach dem Sortieren unterschiedlich ist. Somit werden durch unterschiedliche Startwinkel unterschiedliche Tourenpläne erzeugt.
Abbildung 4.4 zeigt einen Tourenplan der mit dem Sweep Algorithmus erzeugt wurde (Startwinkel $= 0°$). Die verwendete Testinstanz ist die selbe wie in Abbildung 4.3.
Der Sweep Algorithmus erzielt gute Ergebnisse, wenn das Depot relativ zentral zur Kundenstruktur liegt und mit möglichst wenig Touren jeweils relativ viele Kunden pro Tour angefahren werden sollen [40]. Nachteil dieses Verfahrens ist, dass keine überlappenden Touren erstellt werden. Bei überlappenden Touren gibt es Kanten unterschiedlicher Fahrzeuge, welche sich gegenseitig überschneiden, bzw. gibt es gemeinsame Schnittmengen der Flächen die durch die Polygon Form der Touren aufgespannt werden. In Abbildung 4.5 wird dies unter anderem veranschaulicht. Des Weiteren wird beim Clustering auch keine Rücksicht auf Distanzunterschiede der einzelnen Kunden genommen.

Algorithmus 12 Sweep Algorithmus

Input: ($G(V, E), Startwinkel$**)**
Output: Tourenplan L

1: Berechne zu allen Knoten $v \in V \setminus \{0\}$ den dazugehörigen Polarwinkel:
$\phi = \arctan(\frac{v.y-0.y}{v.x-0.x}) - Startwinkel$
falls $\phi < 0$ dann: $\phi = \phi + 360$
2: Sortiere die Kundenknoten aufsteigend nach ihren Polarwinkel ϕ.
3: $T_c \leftarrow \{0\}$, $L \leftarrow \emptyset$
for all $v \in V \setminus \{0\}$ **do**
  if v kann zu T_c hinzugefügt werden, ohne eine Restriktion zu verletzen **then**
    $T_c \leftarrow T_c \dot{\cup} \{v\}$
  else
    $T_c \leftarrow T_c \cup \{0\}$
    $L \leftarrow L \dot{\cup} \{T_c\}$ Es wird eine Kopie von T_c zum Tourenplan L hinzugefügt.
    $T_c \leftarrow \{0\}$
  end if
end for
$T_c \leftarrow T_c \cup \{0\}$
$L \leftarrow L \dot{\cup} \{T_c\}$
4:
for all $T \in L$ **do**
  Löse das TSP für T
end for

4.3.3 Petal Algorithmus

Der Petal Algorithmus, der erstmals in [10] vorgestellt wurde, ist eine Erweiterung des Sweep Algorithmus. Dabei soll der Nachteil beseitigt werden, dass die Distanzen zwischen den Kundenknoten nicht berücksichtigt werden. Anstatt lediglich größtmögliche Touren von aufein-

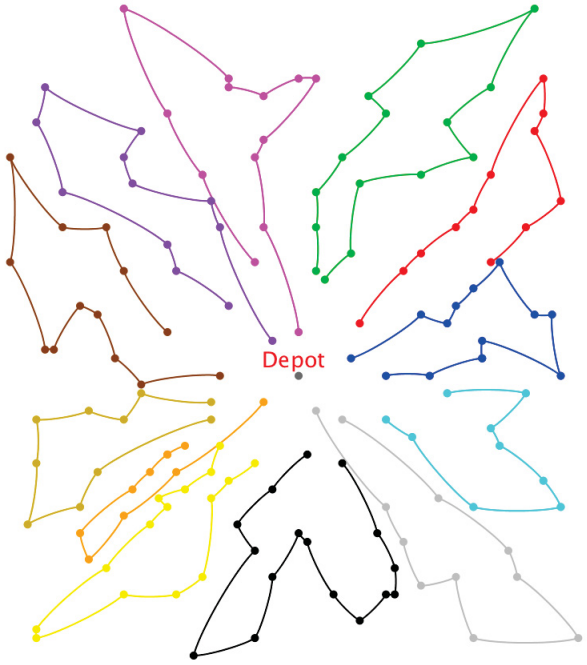
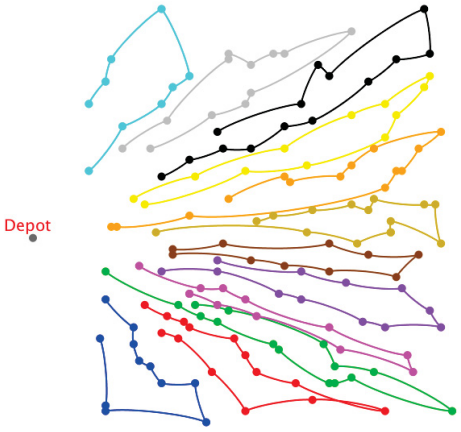
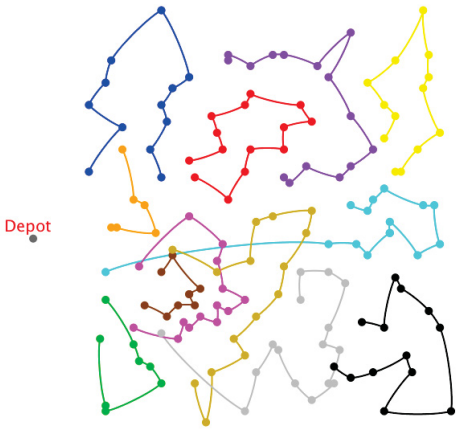


Abbildung 4.4: Erzeugter Tourenplan mit dem Sweep Algorithmus



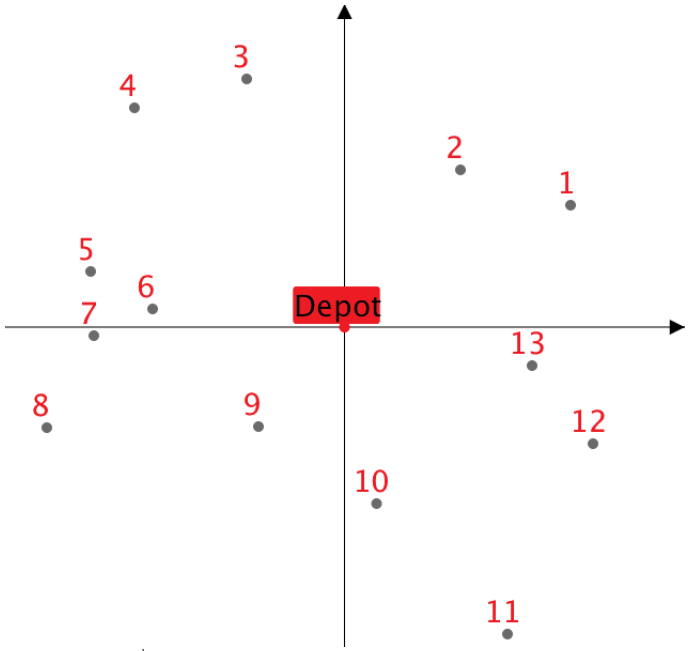
((a)) Tourenplan erzeugt mit dem Sweep Algorithmus (Kosten: 1736.97). In den erzeugten Touren gibt es keine Überlappungen.



((b)) Tourenplan erzeugt durch das Savings Verfahren (Kosten: 1602.08). Hier gibt es überlappende Touren.

Abbildung 4.5: Nachteil des Sweep Algorithmus
Das Depot befindet sich links außerhalb der Kundenstruktur. In dieser Situation erzeugt das Savings Verfahren bessere Tourenpläne als der Sweep Algorithmus.

anderfolgenden Kundenknoten innerhalb der gegebenen Reihenfolge[14] zu erstellen (siehe Algorithmus 12 Schritt 3), werden bei dem Petal Algorithmus alle möglichen Touren von aufeinanderfolgenden Kunden dieser Reihenfolge erstellt. Die durch dieses Vorgehen erzeugten Touren werden als *Petals* bezeichnet. Alle möglichen Petals bezüglich einer Reihenfolge bilden das *Petal-Set*. Abbildung 4.6 zeigt, wie bei einer gegebenen CVRP Instanz das dazugehörige Petal-Set erstellt wird. Hierbei sind die Knoten bereits geordnet nach ihren Polarwinkeln numeriert. Die Ladekapazität der Fahrzeuge ist mit $q = 14$ festgelegt. In der Tabelle können alle möglichen Touren (Petals) gefunden werden, die ohne Verletzung der Kapazitätsbeschränkung gebildet werden können.



Knoten	Nachfrage d_i	Petals ($q = 14$)			
1	5	$\{1\}$	$\{1,2\}$	$\{1,2,3\}$	
2	3	$\{2\}$	$\{2,3\}$	$\{2,3,4\}$	
3	4	$\{3\}$	$\{3,4\}$		
4	4	$\{4\}$	$\{4,5\}$		
5	9	$\{5\}$	$\{5,6\}$		
6	5	$\{6\}$	$\{6,7\}$	$\{6,7,8\}$	$\{6,7,8,9\}$
7	3	$\{7\}$	$\{7,8\}$	$\{7,8,9\}$	
8	5	$\{8\}$	$\{8,9\}$		
9	1	$\{9\}$	$\{9,10\}$	$\{9,10,11\}$	
10	10	$\{10\}$	$\{10,11\}$		
11	3	$\{11\}$	$\{11,12\}$	$\{11,12,13\}$	
12	5	$\{12\}$	$\{12,13\}$	$\{12,13,1\}$	
13	4	$\{13\}$	$\{13,1\}$	$\{13,1,2\}$	

Abbildung 4.6: Erzeugung des Petal-Sets

Algorithmus 13 zeigt den gesamten Ablauf des Petal Algorithmus. In Schritt 4 werden die Kos-

[14] In diesem Fall ist die Reihenfolge durch die Polarwinkel der Knoten festgelegt

ten eines jeden Petals durch lösen des TSP der dazugehörigen Route ermittelt. Anschließend muss in Schritt 5 ein *spannendes Petal-Set* gefunden werden. Hierbei handelt es sich um ein Set, in dem jeder Kunde genau einmal in einem Petal vorkommt. Im optimalen Fall soll das spannende Petal-Set mit den geringsten Kosten gesucht werden.

Algorithmus 13 Petal Algorithmus

Input: $G(V, E)$

Output: Tourenplan L

1: Berechne zu allen Knoten $v \in V \setminus \{0\}$ den dazugehörigen Polarwinkeln:
$\phi = \arctan(\frac{v.y-0.y}{v.x-0.x})$

2: Sortiere die Kundenknoten aufsteigend nach ihren Polarwinkeln.

3: $S \leftarrow$ bilde das Petal-Set zu der gegebenen Reihenfolge.

4:

for all $T \in S$ **do**

Berechne die Kosten des Petals T durch lösen des TSP.

end for

5: $L \leftarrow$ finde ein spannendes Petal-Set zu S

In [32] wird das Auffinden eines optimalen spannenden Petal-Sets wie folgt beschrieben: Das Petal-Set wird als bewerteter *zyklischer Digraph* $G = (V, E)$ dargestellt, wobei die Knoten $V = \{v_1, ..., v_n\}$ die Kunden repräsentieren, währenddessen die Kanten $E \subset \{(v_i, v_j) | v_i, v_j \in V, i \neq j\}$ die Petals darstellen. Der Startknoten einer Petalkante ist der erste Knoten in dem betrachteten Petal, währenddessen der Endknoten der erste Knoten ist, der nicht mehr in dem betrachteten Petal vorkommt. Die Bewertungen der Kanten entsprechen den Kosten die in Schritt 4 von Algorithmus 13 ermittelt wurden. Der zyklische Digraph zu dem vorhandenem Beispiel aus Abbildung 4.6 wird in Abbildung 4.7 veranschaulicht.

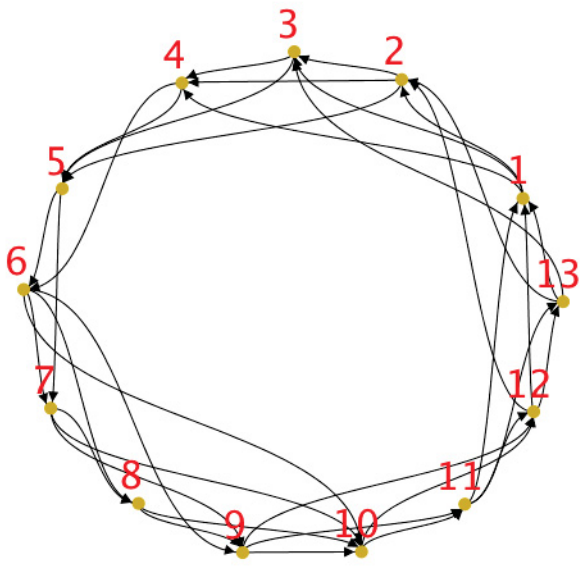


Abbildung 4.7: Petal-Set als zyklischer Digraph

In dem Digraphen wird nun der Kreis mit den geringsten Kosten gesucht. Aus diesem Kreis kann das optimale spannende Petal-Set unmittelbar abgelesen werden, da die Kanten die einzelnen Petals bzw. Touren für den Tourenplan darstellen.
Um den Kreis mit den geringsten Kosten zu identifizieren wird jetzt wie folgt vorgegangen:
Der zyklische Digraph wird bei Knoten k aufgebrochen. Dies wird bewerkstelligt, indem der Knoten k dupliziert wird, wobei der Originalknoten alle ausgehenden Kanten behält, während das Duplikat alle eingehenden zugeordnet bekommt. Alle Kanten, die an k vorbeiführen, werden aus dem Digraph entfernt. Der daraus entstandene Graph wird als *azyklischer Graph* bezeichnet und für $k = 4$ in Abbildung 4.8 dargestellt. Nun wird der kürzeste Pfad zwischen k und dessen Duplikat mithilfe des in Kapitel 2.1.2 gezeigten Algorithmus von Dijkstra bestimmt. Dadurch erhält man den kürzesten Kreis, der durch den Knoten k geht. Wenn der Kreis an allen n verschiedenen Stellen aufgebrochen wird, kann der optimale Kreis gefunden werden. Dadurch erhält man das optimale spannende Petal-Set, welches den Tourenplan darstellt.
Es ist jedoch nicht notwendig, den zyklischen Digraphen an allen n verschiedenen Stellen aufzubrechen. Es reicht, wenn der Knoten s^* gesucht wird, der am wenigsten ausgehende Kanten besitzt. Nun wird der Digraph an den Knoten V^* aufgebrochen, zu denen die ausgehenden Kanten von s^* führen. Dies genügt, um das optimale spannende Petal-Set zu finden, da eine gültige Lösung mindestens einen Knoten von V^* enthalten muss. Es ist nicht möglich, einen gültigen Kreis zu konstruieren, der keinen Knoten von V^* besitzt.
Für den zyklischen Digraph aus Abbildung 4.7 gilt nun also:

$$s^* = 3 \qquad V^* = \{4, 5\}$$

Daraus folgt, dass der Digraph an den Knoten 4 und 5 aufgebrochen werden muss, um das spannende Petal-Set mit den geringsten Kosten zu finden.
In [32] wird außerdem auch gezeigt, dass dieses Verfahren nicht nur für die Reihenfolge der Knoten bezüglich der Polarwinkel eingesetzt werden kann, sondern für jede beliebige Reihenfolge. Daher kann der Petal Algorithmus z.B. auch dafür eingesetzt werden, ein optimales spannendes Petal-Set für die Reihenfolge der Knoten aus einer Giant Tour zu finden. In diesem Fall müssen in Algorithmus 13 die Schritte 1 und 2 dementsprechend ersetzt werden.
Abbildung 4.9 zeigt den direkten Vergleich der Lösungen des Sweep- und Petal Algorithmus[15]. Der Petal Algorithmus liefert dabei klar das bessere Ergebnis.

4.4 Verbesserungsheuristiken für das Vehicle Routing Problem

Die Verbesserungsheuristiken für das VRP können in *Single-Route* und *Multi-Route* Verbesserungen unterteilt werden [39]. Bei den Single-Route Verbesserungen werden nur einzelne

[15] Christofides, Mingozzi und Toth Instanz 1

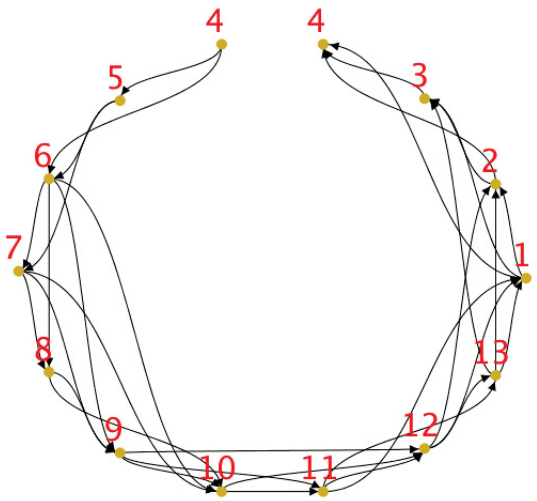
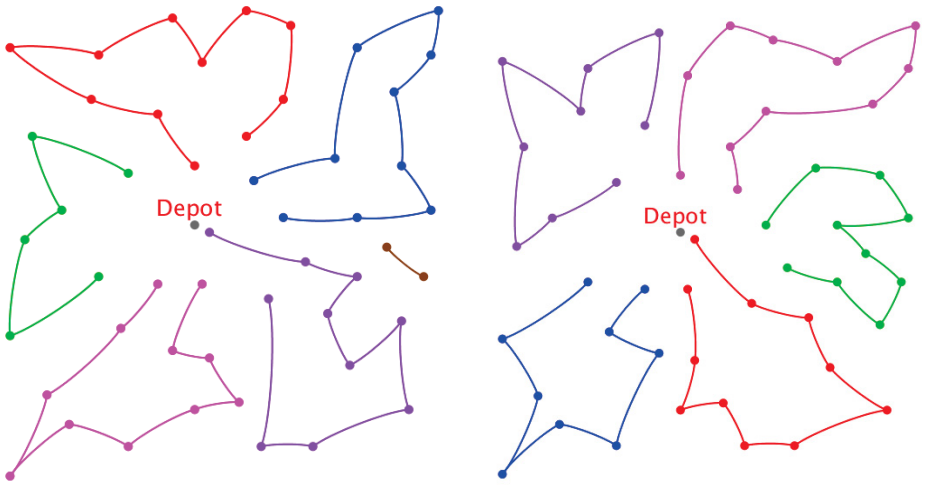


Abbildung 4.8: Aufgebrochener azyklischer Digraph



((a)) Tourenplan mit Sweep Algorithmus. Fahrzeuge: 6 Kosten: 598.39

((b)) Tourenplan mit Petal Algorithmus. Fahrzeuge: 5 Kosten: 531.9

Abbildung 4.9: Vergleich zwischen Sweep und Petal Algorithmus

Touren verbessert. Dies kann mit den TSP Heuristiken von Kapitel 3 durchgeführt werden. Währenddessen werden bei den Multi-Route Verbesserungen Nachbarlösungen untersucht, in denen tourenübergreifende Änderungen vorgenommen werden.

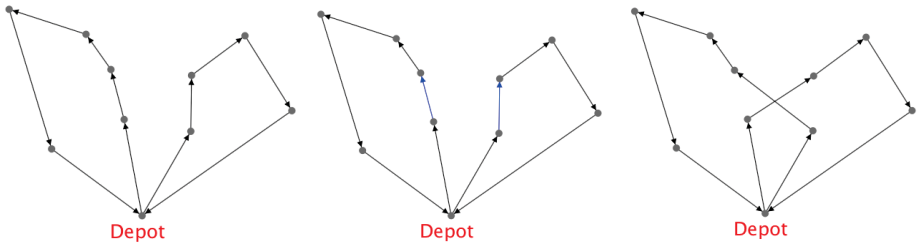
4.4.1 Knotenorientierte Austausch-Nachbarschaften

In [41] werden folgende tourenübergreifende Verbesserungsoperatoren für zwei Fahrzeuge definiert:

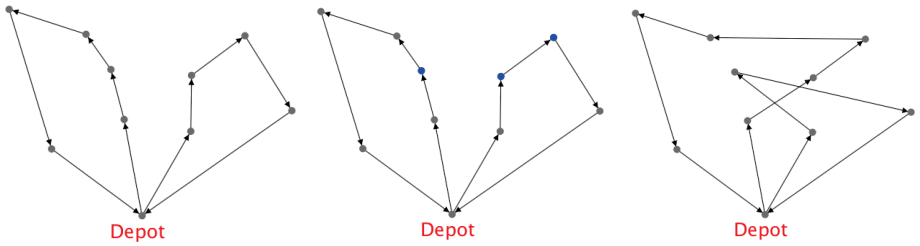
- *String cross*: Zwei Sequenzen von Kunden werden ausgetauscht, indem zwei Kanten von zwei verschiedenen Touren gekreuzt werden. (Abbildung 4.10(a))
- *String exchange*: Zwei Sequenzen von bis zu k Kunden werden zwischen zwei Touren ausgetauscht. Die Länge der beiden Sequenzen muss nicht notwendigerweise gleich sein. (Abbildung 4.10(b))
- *String relocation*: Eine Sequenz von bis zu k Kunden wird von einer Tour auf eine andere versetzt, typischerweise mit $k = 1$ oder 2. Mit diesem Operatoren kann möglicherweise die Anzahl der Touren vermindert werden. (Abbildung 4.10(c))
- *String mix*: Es werden für 2 Touren die Nachbarschaftslösungen mithilfe von String exchange und String relocation bezüglich eines vorgegebenen k erzeugt. Die beste wird daraus gewählt.

Mit Hilfe dieser Nachbarschaftsoperatoren kann zum Beispiel Algorithmus 5 für das VRP eingesetzt werden.
Ein Verbesserungsoperator, in dem zwei oder mehr Touren eines Tourenplans betrachtet werden, wird in [27] vorgestellt. Die Autoren beschreiben ein generelles b-*cyclic,* k-*transfer* Schema, um eine Nachbarschaftslösung $L^{'}$ ausgehend aus einer Lösung L zu erzeugen. Mit dem Parameter b wird festgelegt, wie viele Touren eines Tourenplans $L = \{T_1, T_2, .., T_n\}$ gleichzeitig betrachtet werden. k definiert die Menge an Kunden, welche zwischen den betrachteten Touren zyklisch ausgetauscht werden. Um eine Nachbarschaftslösung zu erzeugen, muss zuerst eine zyklische Permutation $\rho \subseteq \{1, ..., n\} \quad |\rho| = b$, zum Beispiel $\rho = \{2, 5, 3\}$ definiert werden. Anhand dieses ρ werden k Kunden von T_2 nach T_5, von T_5 nach T_3 bzw. von T_3 nach T_2 versetzt. In diesem Beispiel werden 3 Touren gleichzeitig verändert ($b = 3$). Die k Kunden sollen jeweils so in die nächste Tour eingesetzt werden, dass die resultierenden Kosten minimal werden. Die b veränderten Touren aus $L^{'}$ müssen aufgrund ihrer Restriktionen[16] überprüft werden, da sonst keine gültige Lösung $L^{'}$ vorliegt. Natürlich ist ersichtlich, dass es viele verschiedene Möglichkeiten gibt, k Kunden aus einer Tour T auszuwählen, sodass k üblicherweise niedrig gehalten wird. Laut [27] eignen sich $b = 2$ bzw. $b = 3$ und $k = 1$ bzw. $k = 2$ am besten für die

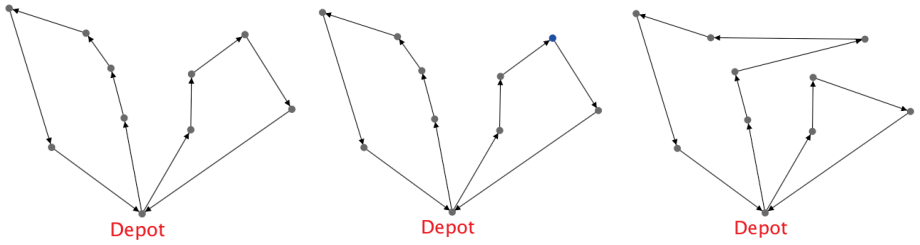
[16] Maximale Ladekapazitäten oder maximal erlaubte Lenkzeiten.



((a)) String cross



((b)) String Exchange



((c)) String Relocation

Abbildung 4.10: Van Breedam Nachbarschaft

Wahl der Parameter. Abbildung 4.11 zeigt wie ein 3-cyclic, 1-transfers bei einem gegebenen Beispiel durchgeführt werden kann, um eine Nachbarschaftslösung $L^{'}$ zu erzeugen.

Tourenplan L	
Tour 1	5,3,9,7
Tour 2	1,16,14,8
Tour 3	2,15,12,11
Tour 4	4,6,10,13
Tourenplan $L^{'}$	
Tour 1	5,9,7,15
Tour 2	1,16,14,8
Tour 3	2,12,11,6
Tour 4	3,4,10,13

Abbildung 4.11: b-cyclic, k-transfer Beispiel.
$\rho = \{3, 1, 4\}$ $(b = 3)$ und $k = 1$. Für die Menge der auszutauschenden Knoten wurde $\{\{15\}, \{3\}, \{6\}\}$ gewählt.

Das zyklische Austauschverfahren von [27] hat den Nachteil, dass immer gleich viele Kunden unter den betrachteten Touren ausgetauscht werden. Daher wird dem Verfahren in [36] ein zusätzlicher Vektor α hinzugefügt, welcher die Kundenmengen der einzelnen Austauschschritte festlegt. Bei einem $\rho = \{2, 5, 3\}$ und einem $\alpha = \{1, 0, 2\}$ würden zum Beispiel 1 Kunde von T_2 nach T_5, kein Kunde von T_5 nach T_3 bzw. 2 Kunden von T_3 nach T_2 verschoben werden. Durch Variation des Vektors α können mehr Nachbarschaftslösungen als bei dem bisherigen b-cyclic, k-transfer erzeugt werden. Wenn der Austauschvektor α aus identischen Elementen besteht, wie zum Beispiel $\alpha = \{2, 2, 2\}$, würde ein klassischer b-cyclic, k-transfer mit $k = 2$ durchgeführt werden. Abbildung 4.12 zeigt, wie eine Nachbarschaftslösung $L^{'}$ mit $\rho = \{1, 3, 4\}$ und $\alpha = \{2, 0, 1\}$ aus einem gegebenen Beispiel erzeugt wird.

Tourenplan L	
Tour 1	5,3,9,7
Tour 2	1,16,14,8
Tour 3	2,15,12,11
Tour 4	4,6,10,13
Tourenplan $L^{'}$	
Tour 1	4,5,7
Tour 2	1,16,14,8
Tour 3	2,15,12,3,9,11
Tour 4	6,10,13

Abbildung 4.12: b-cyclic, α-transfer Beispiel.
$\rho = \{1, 3, 4\}$ $(b = 3)$ und $\alpha = \{2, 0, 1\}$. Für die Menge der auszutauschenden Knoten wurde $\{\{3, 9\}, \{\}, \{4\}\}$ gewählt.

Ein weiterer wichtiger Operator, der λ-*Interchange*, wird in [24] erstmals vorgestellt: Die Bezeichnung λ-Interchange steht für alle Austausch-Schritte, die jeweils bis zu λ Knoten zwi-

schen zwei Touren verschieben. Das bedeutet, es handelt sich hierbei um einen 2-cyclic Austausch. Ein 1-Interchange beinhaltet alle Austauschschritte mit den α Vektoren: $\{1,0\}$,$\{0,1\}$ und $\{1,1\}$. Zum 2-Interchange gehören bereits alle Austauschschritte mit den α Vektoren: $\{1,0\}$,$\{0,1\}$,$\{1,1\}$,$\{0,2\}$,$\{2,0\}$,$\{1,2\}$,$\{2,1\}$ und $\{2,2\}$. Wegen der schnell wachsenden Größe der λ-Interchange-Nachbarschaften werden in der Praxis häufig nur Nachbarschaften für $\lambda \leq 2$ betrachtet. Natürlich kann der λ-Interchange auch mehr als 2 Touren $b > 2$ betrachten. Jedoch ist hierbei ebenfalls eine schnell anwachsende Laufzeit zu beachten.

4.4.2 Ruin and Recreate Prinzip

Ein alternativer Ansatz zu den knotenorientierten Austausch-Nachbarschaften wird in [34] vorgestellt. Das *Ruin and Recreate* Prinzip verfolgt die Grundidee, einen gewissen Teil der Lösung zu zerstören (*Ruin*), um diesen Teil anschließend auf eine unterschiedliche Art wieder aufzubauen (*Recreate*). Mit diesem Vorgehen kann in jedem Iterationsschritt eine Nachbarschaftslösung erzeugt werden. Es sei angemerkt, dass sich dieses Algorithmen Schema nicht nur auf das VRP sondern auch auf andere kombinatorische Optimierungsprobleme, wie zum Beispiel dem TSP, anwenden lässt.
Das RAR[17] Prinzip für das VRP wird in [34] folgendermaßen beschrieben: Zunächst werden in dem Ruin Schritt k Kunden aus dem Tourenplan L entfernt und der Menge B hinzugefügt, sodass ein unvollständiger Tourenplan L^* entsteht. Die Kunden aus der Menge B werden zunächst nicht mehr beliefert, und werden erst anschließend im Recreate Schritt in den Tourenplan L^* eingefügt. Folgende Vorgehensweisen für den Ruin Schritt können angewendet werden:

- Radial Ruin: Ein zufällig ausgewählter Kunde s, sowie alle Kunden innerhalb einer definierten Entfernung r zu s werden aus L entfernt.
- Jeder Kunde wird mit einer definierten Wahrscheinlichkeit p wird aus L entfernt.
- Zufällig ausgewählte Ketten[18] werden aus L entfernt.
- Time Deletion: Jeder Kunde, der innerhalb eines definierten Zeitfensters $[a_i, b_i]$ beliefert wird, wird aus L entfernt.
- Volume Deletion: Jeder Kunde, dessen Bedarf d innerhalb einer definierten Reichweite $[d_{min}, d_{max}]$ liegt, wird aus L entfernt.

Anschließend werden die Kunden aus B mit einem bestimmten Verfahren wieder in den Tourenplan L^* eingefügt, sodass der Tourenplan $L^{'}$ entsteht. Dies wurde in [34] mit einem *Best Insertion* Verfahren gelöst. Dabei werden die Kunden aus B einzeln zufällig aus B entfernt

[17] Ruin and Recreate
[18] Eine Kette definiert eine Sequenz von Kunden, die innerhalb einer Tour hintereinander beliefert werden.

und so in den Tourenplan L^* eingefügt, dass die zusätzlich entstehenden Kosten minimal werden. Zusätzlich muss überprüft werden, ob bei dem Einfügen keine der Fahrzeugrestriktionen verletzt wird. Ist es nicht möglich, einen Kunden in L^* einzufügen, ohne eine Restriktion zu verletzen, muss ein zusätzliches Fahrzeug T_z dem Tourenplan L^* hinzugefügt werden. Wenn alle Kunden aus B wieder beliefert werden, erhält man die gültige Nachbarschaftslösung $L^{'}$. Algorithmus 14 veranschaulicht das Grundprinzip in einem lokalen Suchverfahren. Dieses Verfahren führt laut [34] zu äußerst guten Ergebnissen. In vielen der getesteten VRP Instanzen konnten neue beste Lösungen erzielt werden.

Algorithmus 14 Ruin and Recreate Prinzip

Input: Tourenplan L

1: Wähle ein Ruin-Verfahren. (Ein Verfahren kann zufällig gewählt werden.)
2: $L^*, B \leftarrow$ Ruin(L).
3: $L^{'} \leftarrow$ Recreate(L^*, B)
4: Entscheide ob L durch $L^{'}$ ersetzt werden soll.
5: Wenn Abbruchkriterium erreicht wurde: stop. Sonst gehe zu Schritt 1.

5 Methaheuristiken

Im Gegensatz zu den problemspezifischen Heuristiken, definieren Metaheuristiken eine Folge von Schritten, welche für beliebige kombinatorische Optimierungsprobleme eingesetzt werden können. In Abhängigkeit von der vorhandenen Problemstellung, müssen die einzelnen Schritte problemspezifisch implementiert werden. Zu den wichtigsten Methaheuristiken zählen unter anderem:

- Simulated Annealing
- Tabu Search
- Ameisenalgorithmen
- Genetische Algorithmen

Mit den Methaheuristiken werden für das VRP im Allgemeinen bessere Ergebnisse als mit den klassischen Heuristiken erzielt. Jedoch benötigen diese mehr Rechenzeit. [39] Da im Rahmen dieser Arbeit ein genetischer Algorithmus für die entwickelte JAVA Software implementiert wurde, sollen vor allem diese in diesem Kapitel näher beschrieben werden.

5.1 Simulated Annealing

Die Grundidee von Simulated Annealing liegt darin, den Annealingprozess aus der Metallurgie nachzubilden. Dabei wird Metall zuerst erhitzt, und anschließend langsam abgekühlt, damit die Atome genügend Zeit haben sich zu ordnen und stabile Kristalle zu bilden. Simulated Annealing wird zum Auffinden von Minima von Zielfunktionen eingesetzt und eignet sich daher zur Lösung des VRP. Der Hauptvorteil dieser Methaheuristik liegt darin, dass der Algorithmus lokale Optima überwinden kann, indem innerhalb eines iterativen Schrittes auch eine schlechtere Lösung mit einer bestimmten Wahrscheinlichkeit akzeptiert werden kann. [3]
Das Verfahren wird in [39] folgend beschrieben:
In jedem Iterationsschritt t von SA[19] wird eine zufällige Nachbarschaftslösung L aus $N(L_t)$ erzeugt. Gilt $f(L) \leq f(L_t)$ so wird L_{t+1} mit L gleichgesetzt. Sonst gilt:

[19] Simulated Annealing

$$L_{t+1} := \begin{cases} L & \text{mit der Wahrscheinlichkeit } p_t \\ L_t & \text{mit der Wahrscheinlichkeit } 1 - p_t \end{cases}$$

p_t wird in Abhängigkeit von t und $f(L) - f(L_t)$ mit:

$$p_t = e^{\frac{f(L)-f(L_t)}{\Theta_t}}$$

berechnet. Wobei Θ_t die Temperatur bei der Iteration t kennzeichnet. Üblicherweise wird Θ_t durch eine fallende Stufenfunktion in Abhängigkeit von t definiert: Zu Beginn wird Θ_t mit einem Startwert $\Theta_1 > 0$ initialisiert. Nach jeweils T Iterationen wird Θ_t mit einem Faktor $\alpha(0 < \alpha < 1)$ multipliziert, sodass die Wahrscheinlichkeit der Akzeptanz einer schlechteren Lösung mit steigendem t sinkt. Die üblichen Abbruchkriterien sind:

- Der Wert f^* von L^* hat sich in den letzten k_1 aufeinanderfolgenden Zyklen von T Iterationen nicht um $\pi_1(0 < \pi_1 < 100)\%$ gesenkt.
- Die Anzahl an akzeptierten neuen Lösungen während den letzten k_2 aufeinanderfolgenden Zyklen von T Iterationen ist kleiner als $\pi_2(0 < \pi_2 < 100)\%$ von T.
- k_3 Zyklen von T Iterationen wurden durchgeführt.

Für die Erzeugung der Nachbarschaftslösungen $N(L_t)$ können die in Kapitel 4.4 vorgestellten Verfahren eingesetzt werden. Die benötigte Startlösung L_1 kann mit einem beliebigen Eröffnungsverfahren erstellt werden. Abbildung 5.1 zeigt einen Flowchart zu dem SA Algorithmus, wie er in [3] abgebildet wird.
In [24] wird eine effektive SA Implementierung für das VRP vorgestellt, welche bei der Erzeugung der Nachbarschaftslösungen auf den λ-Interchange Operator zurückgreift.

5.2 Tabu Search

Das grundlegende Paradigma von Tabu Search liegt in der Nutzung von Information über den Verlauf des Suchprozesses, um lokale Suchverfahren derart zu steuern, dass lokale Optima überwunden werden können. Entsprechende Informationen werden in einem abstrakten Gedächtnis gespeichert, welches als Tabu Liste bezeichnet wird. TS[20] ist eine lokale Suchstrategie, die mit einer Ausgangslösung L startet. Aus der Menge der zulässigen Nachbarlösungen $N(L)$ wird in jeder Iteration der bestmögliche Zug ausgeführt. Der Prozess wird beendet, sobald ein zu spezifizierendes Abbruchkriterium erfüllt ist. Das Abbruchkriterium kann zum Beispiel durch die maximale Anzahl an Iterationen definiert sein.
Da das Hauptziel darin liegt, das erneute Aufsuchen bereits untersuchter Lösungen zu verhindern, werden Lösungen oder Züge tabuisiert. Entsprechende Lösungen oder Züge werden somit

[20] Tabu Search

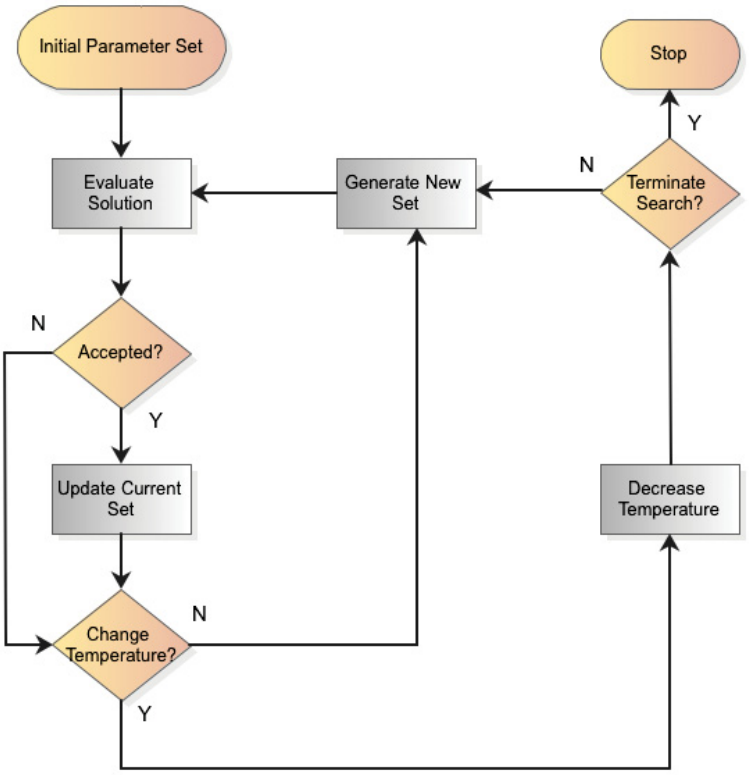


Abbildung 5.1: Simulated Annealing Flowchart

für eine gewisse Anzahl an Iterationen nicht mehr betrachtet, dass heißt, die Nachbarschaft wird um diese Lösungen bzw. Züge (dynamisch) reduziert. Die tabuisierten Lösungen werden in einer Tabu Liste T abgelegt. Die Reduktion der zulässigen Nachbarschaft entspricht einer Diversifikation der Suche, wobei Bereiche des Lösungsraums mit bereits besuchten Lösungen tendenziell verlassen werden. [15]

TS wird in [14] erstmals systematisch beschrieben. Algorithmus 15 zeigt den darin allgemein dargestellten Ablauf. Bei der Aktualisierung von T in Schritt 4 wird die aktuelle Lösung L der Liste hinzugefügt und ältere Lösungen $L^{'} \in T$ können entfernt werden, wenn diese bereits eine definierte Anzahl an k Iterationen in T liegen.

Um Rechenzeit sowie benötigten Speicher möglichst gering zu halten, werden in T nicht die kompletten Lösungen eines Problems gespeichert, sondern üblicherweise nur bestimmte Attribute dieser Lösungen [39]. In [24] wird eine VRP TS Implementierung vorgestellt, in welcher die Tabu Liste T in Form von Zügen verwaltet wird. Wenn ein Kunde i von einer Tour T_u zu einer anderen Tour T_v verschoben wird, so wird die Rückkehr von i nach T_u für eine gewisse Anzahl an k Iterationen tabuisiert. Zur Erzeugung der Nachbarschaftslösungen wird in dieser Implementierung ebenfalls der λ-Interchange Operator verwendet.

In [39] werden weitere effektive TS Implementierungen für das VRP vorgestellt.

Algorithmus 15 Tabu Search

1: Wähle eine Startlösung $L \in X$ und setze:
$L^* := L$, den Iterationszähler $k = 0$, $T = \emptyset$
2:
if $N(L) \setminus T = \emptyset$ **then**
  Gehe zu Schritt 4
else
  $k := k + 1$ und wähle $L_k \in N(L) \setminus T$ sodass $f(L_k) \leq f(L^*) \forall L^* \in N(L) \setminus T$
end if
3: Setze $L := L_k$
if $f(L) < f(L^*)$ wobei L^* die bis jetzt beste gefundene Lösung darstellt **then**
  $L^* := L$
end if
4:
if $N(L) \setminus T = \emptyset$ oder eines der Abbruchkriterien wurde erreicht **then**
  Stop.
else
  Führe für T ein entsprechendes Update durch und gehe zu Schritt 2.
end if

5.3 Ameisenalgorithmen

Die Ameisenalgorithmen, oder auch als *Ant Colony Optimization* bezeichnet, basieren auf dem modellhaften Verhalten von realen Ameisen bei ihrer Futtersuche. Prinzipiell kann beobachtet werden, dass Ameisen für den Fall, dass ihnen mehrere Wege unterschiedlicher Länge von ihrem Nest zur Futterstelle zur Verfügung stehen, nach kurzer Zeit mit Mehrheit den kürzesten Weg wählen. Die Ursache für dieses kollektiv intelligente Verhalten liegt in der Abgabe des *Pfadpheromons* der Ameisen, während sie sich auf der Futtersuche befinden. Dieses Abgegebene Pheromon verdunstet langsam mit der Zeit. Am Anfang wählen die Ameisen ihren Weg bei jeder Gabelung zufällig aus und jeder Weg wird mit der selben Menge an Pheromon markiert. Eine Ameise die einen längeren Weg gewählt hat, benötigt für diesen eine längere Zeit. Wenn diese wieder zur selben Gabelung zurückkehrt, ist auf dem längeren Weg bereits mehr Pfadpheromon verdunstet als auf den kürzeren. Dadurch steigt die Wahrscheinlichkeit, dass nachfolgende Ameisen einen kürzeren Weg wählen, da Ameisen Wege mit höherer Pheromonkonzentration favorisieren.
Inspiriert von diesem Verhalten wurde der Ameisenalgorithmus als Methaheuristik erstmals 1992 in [9] vorgestellt, um mit diesem kürzeste Pfade innerhalb eines Graphen zwischen zwei Knoten zu finden. Da bis jetzt kaum Veröffentlichungen von Ameisenalgorithmen zur Lösung des VRP vorhanden sind, sollen diese in dieser Arbeit nicht näher beschrieben werden. An dieser Stelle soll lediglich an die VRP Implementierung in [28] verwiesen werden, bei der es sich laut [38] um eine der besten Ameisenalgorithmen Umsetzungen für das VRP handelt.

5.4 Genetische Algorithmen

In diesem Abschnitt soll die grundlegende Terminologie genetischer Algorithmen eingeführt werden und deren Gebrauch am Beispiel von TSP und VRP veranschaulicht werden. Der Ursprung der genetischen Algorithmen liegt in der Theorie der biologischen Evolution begründet. Das Schema der Evolutionslehre *C. Darwin*, basierend auf dem Gesetz *Survival of the Fittest* wird dabei direkt auf ein Problem angewandt. Dieses Algorithmen-Schema wird in [30] wie folgt beschrieben:

1. Eine Lösung des Problems, welche im Folgenden als Individuum bezeichnet wird, muss als ein *Chromosom* darstellbar sein[21].

2. Erschaffe eine Generation von Individuen durch zufällige Erzeugung von Chromosomen.

3. Fortpflanzung:

   a) *Selektion:*

   Aus der Elterngeneration werden zwei Individuen durch ein Selektionsverfahren ausgewählt. Die beiden Individuen repräsentieren Vater und Mutter. Die Leistungsfähigkeit der beiden mit Blick auf das zu lösende Problem wird mit einer *Fitness*-Funktion gemessen. Individuen mit einem höheren Fitnesswert besitzen eine größere Wahrscheinlichkeit als Elternteil ausgewählt zu werden.

   b) *Kombination(Kreuzung):*

   Die Gene von Vater und Mutter werden vermischt und ergeben so die Gene des Kindes oder der Kinder.[22]

   c) *Mutation:*

   Mit einer bestimmten Wahrscheinlichkeit tritt eine spontane Veränderung in dem Chromosom des Kindes auf.

Algorithmus 16 veranschaulicht den Ablauf systematisch.
Da dieses Verfahren, wie die Evolution selbst, unendlich lange läuft, wird eine maximale Anzahl an Generationen k als Abbruchkriterium festgelegt. Das beste Individuum I_b das jemals existierte stellt die Lösung dieses Verfahrens dar. Es sei natürlich angemerkt, dass die Individuen der ersten Generation nicht zufällig erstellt werden müssen, sondern auch mit einer Eröffnungsheuristik erzeugt werden können.
In Abhängigkeit der vorliegenden Problemstellung muss der Kreuzungsprozess sowie die Mutation problemspezifisch implementiert werden. Für die Selektion der Eltern stehen eine Reihe von allgemeinen Selektionsverfahren zur Verfügung, welche beispielsweise in [12] und [21] beschrieben werden. Die Selektion ist sehr wesentlich dafür, einen genetischen Algorithmus in

[21] Ein Individuum besteht aus einem Chromosom, welches aus mehreren einzelnen Genen besteht.
[22] In den meisten Kreuzungsverfahren werden zwei Kinder erzeugt.

Algorithmus 16 Genetischer Algorithmus

$P \leftarrow$ Erzeuge eine Startpopulation.
Wähle I_b sodass $f(I_b) \geq f(I^*) \forall I^* \in P$. Die Funktion f definiert die Fitness der Individuen.

for 0 **to** k **do**
  $P^* \leftarrow \emptyset$
  $P' \leftarrow$ Bilde einen Vermehrungspool mit Hilfe eines definierten Selektionsverfahren.
  while $|P^*| < |P|$ **do**
    Entnimm aus P' die Eltern I_1 und I_2.
    $M \leftarrow$ Erzeuge die Kinder durch Kreuzung von I_1 und I_2.
    for all $I_c \in M$ **do**
      $I_c \leftarrow$ Führe mit einer bestimmten Wahrscheinlichkeit p eine Änderung des Chromosoms aus. (Mutation)
    end for
    $P^* \leftarrow P^* \cup M$
  end while
  $P \leftarrow P^*$
  Wähle I_b^* sodass $f(I_b^*) \geq f(I^*) \forall I^* \in P$
  Ersetze I_b durch I_b^* falls $f(I_b^*) > f(I_b)$
end for

Richtung bessere Lösungen zu treiben. Je stärker der Selektionsdruck, desto rascher konvergiert der Algorithmus. Jedoch kann es auch passieren, dass die genetische Vielfalt schnell verlorengeht und das Verfahren gegen ein schwächeres lokales Optimum konvergiert. Abbildung 5.2 zeigt einen typischen Konvergenzverlauf von GA[23].

Fitnessproportionale Selektion

Bei diesem Selektionsverfahren, welches in der Literatur auch als *Rouletterad-Selektion* bezeichnet wird, wird zunächst für jedes Individuum $I_i \in P$ eine relative Fitness r_i wie folgt berechnet:

$$r_i = \frac{f(I_i)}{\sum_{j=1}^{|P|} f(I_j)}$$

Klarerweise gilt:

$$\forall I_i \in P: \quad 0 \leq r_i \leq 1$$

$$\sum_{j=1}^{|P|} r_j = 1$$

Die Auswahl der Individuen, welche in den sogenannten Vermehrungspool kommen, kann anschließend mit dem Verfahren aus Algorithmus 17 durchgeführt werden. Bei diesem Selektions-

[23] Genetische Algorithmen

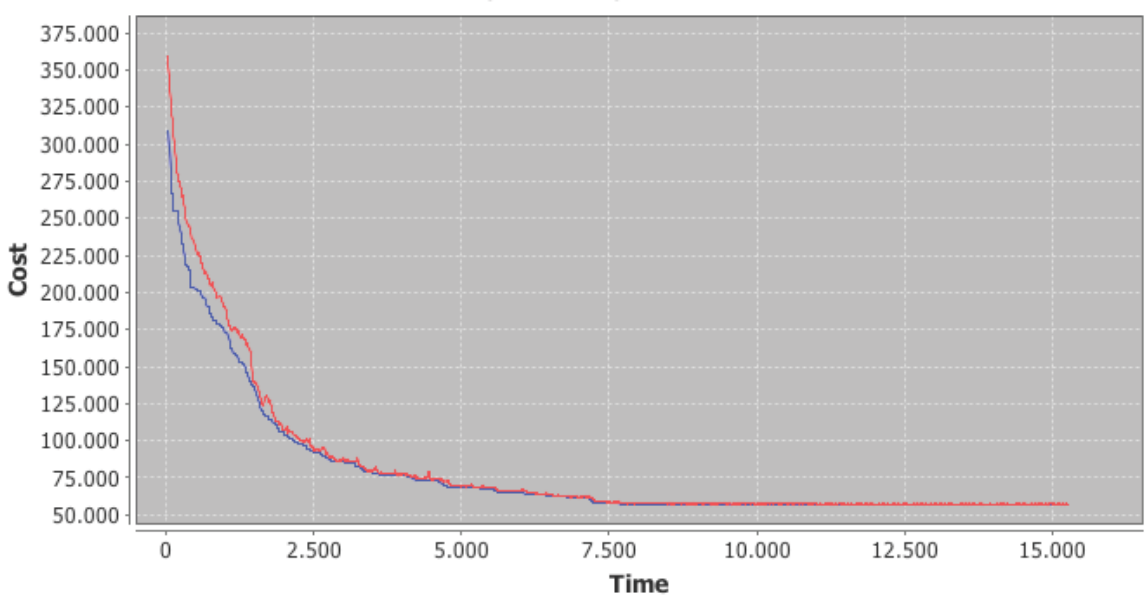


Abbildung 5.2: Konvergenzverhalten von genetischen Algorithmen
Dieses Diagramm wurde anhand eines Testlaufs der TSPLIB Instanz Gr96 erstellt. Die rote Linie entspricht den durchschnittlichen Tourenkosten der Population, während die blaue Linie die Kosten der momentan besten Tour anzeigt. Das Verfahren konvergiert hierbei gegen einen Kostenwert von 56.628.

Algorithmus 17 Auswahl der Individuen für den Vermehrungspool

$P' \leftarrow \emptyset$ definiere einen leeren Vermehrungspool.
for 1 **to** $|P|$ **do**
  $z \leftarrow$ bestimme eine Zufallszahl $z \in [0, 1)$.
  $sum \leftarrow 0$
  $j \leftarrow 0$
  while $sum < z$ **do**
    $j \leftarrow j + 1$
    $sum \leftarrow sum + r_j$
  end while
  $P' \leftarrow P' \cup \{I_j\}$
end for

verfahren kommt es zu Problemen, falls es einzelne Individuen gibt, die eine sehr hohe Fitness besitzen. Diese setzen sich im Laufe des genetischen Algorithmus bei der Selektion besonders häufig durch, wodurch die genetische Vielfalt rasch verloren gehen kann. Ohne eine ausreichend große genetische Vielfalt besteht die Gefahr, dass das Verfahren gegen ein schlechtes lokales Optimum konvergiert.
Andererseits kann ein zu geringer Selektionsdruck entstehen, wenn die Fitnesswerte der Individuen $I \in P$ zu gering streuen, wodurch das Verfahren nur äußerst langsam konvergiert.

Rangbasierte Selektion

Hier werden zunächst die Individuen nach ihren Fitnesswerten sortiert. Anschließend bekommt das beste Individuum die künstliche Fitness $|P|$, während das schlechteste Individuum die Fitness 1 zugeordnet bekommt. Anhand dieser künstlichen Fitnesswerte wird zunächst ein vorläufiger Vermehrungspool erstellt. Um anschließend einen endgültigen Vermehrungspool zu erstellen wird ein sogenanntes *Turnier* durchgeführt, welches auf folgende Arten umgesetzt werden kann:

- Es werden zufällig zwei Individuen aus dem vorläufigen Vermehrungspool entnommen und eine Zufallszahl $z \in [0, 1)$ wird bestimmt. Im Vorhinein muss noch ein fixer Parameter r definiert werden.

  $$\text{In den endgültigen Vermehrungspool kommt: } \begin{cases} \text{das bessere Individuum falls } z < r \\ \text{das schlechtere Individuum falls } z \geq r \end{cases}$$

  Dieser Vorgang wird so lange wiederholt, bis der Vermehrungspool die gewünschte Größe erreicht hat.

- Es werden zufällig zwei Individuen aus dem vorläufigen Vermehrungspool entnommen. Das Individuum mit der größeren Fitness wird in den endgültigen Vermehrungspool aufgenommen. Dieser Vorgang wird so lange wiederholt, bis der Vermehrungspool die gewünschte Größe erreicht hat.

Dieses Selektionsverfahren verhindert, dass sich Individuen mit einer vergleichsweise großen Fitness schnell durchsetzen. Üblicherweise konvergiert der genetische Algorithmus bei dieser Vorgehensweise eher langsamer. Im Gegensatz dazu besteht aber eine geringere Chance, in einem schlechteren lokalen Optima hängen zubleiben.

Reine Turnierwahl

Bei der reinen Turnierwahl, oder auch als *Tournament Selection* bezeichnet, werden $q > 1$ Individuen zufällig und unabhängig von ihrer Fitness gewählt. Das Individuum mit der größten

Fitness kommt in den Vermehrungspool. Dieser Vorgang wird so oft wiederholt, bis der Vermehrungspool ausreichend viele Eltern enthält. Mit steigendem q wird hierbei logischerweise der Selektionsdruck erhöht. Alternativ kann auch eine gewichtete Turnierwahl ausgeführt werden. Bei dieser Vorgehensweise werden die q Individuen gemäß ihrer Fitnesswerte sortiert und anschließend ein Individuum anhand einer fix vorgegebenen Wahrscheinlichkeitsverteilung $p_1, ..., p_q$ ausgewählt. Die Wahrscheinlichkeiten müssen so gewählt werden, dass $\sum_{j=1}^{q} p_j = 1$ eingehalten wird. Die Auswahl des Individuums für den Vermehrungspool kann mit dem bereits beschriebenen Verfahren aus Algorithmus 17 durchgeführt werden.

Elitismus

Bei Verfolgung einer *Elitismus-Strategie* werden die q besten Individuen $\in P$ direkt für die Population der nächsten Generation unverändert übernommen. Es ist ersichtlich, dass Elitismus eine Zeitersparnis erwirtschaftet, da die genetischen Operatoren Selektion, Kreuzung und Mutation auf weniger Individuen angewendet werden müssen. Des Weiteren zeigen viele Untersuchungen, dass durch gezielten Einsatz einer Elitismus-Strategie eine signifikante Verbesserung der Performance der GA erzielt werden kann.[21]

5.5 Anwendung genetischer Algorithmen für das Traveling Salesman Problem

Wie in allen Methaheuristiken, muss die konkrete Implementierung des Algorithmus in Abhängigkeit der vorliegenden Problemstellung erfolgen. Bei GA für das TSP wird das Chromosom eines Individuums durch eine Rundreise repräsentiert, welche alle Städte passiert und in der Stadt endet, in welcher sie begann. Ein mögliches Chromosom könnte also wie folgt aussehen: $[0\ 5\ 1\ 4\ 3\ 2]$. Die einzelnen Gene dieses Chromosoms repräsentieren die Städte, die besucht werden. Die Berechnung der Fitness wird hierbei in Abhängigkeit der Gesamtdistanz $l(T)$ durchgeführt. Da Touren mit kürzer Länge favorisiert werden, berechnet sich die Fitness der Individuen üblicherweise durch:

$$f(T) = \frac{1}{l(T)}$$

5.5.1 Kreuzungsverfahren für das Traveling Salesman Problem

Für die Kombination bzw. die Kreuzung von Individuen gibt es mehrere verschiedene Verfahren die angewendet werden können. In [12] werden unter anderem die wichtigsten Kreuzungsoperatoren welche bei dem TSP angewendet werden können veranschaulicht. Einige der wichtigeren sollen an dieser Stelle näher beschrieben werden:

Partially Matched Crossover

Bei dem *Partially Matched Crossover* wird wie folgt vorgegangen:

- Bestimme zufällig zwei Kreuzungspunkte.
- Die Gensequenzen innerhalb dieser Kreuzungspunkte werden vertauscht, wodurch die Proto-Children entstehen. Diese können doppelte Städte beinhalten, wodurch ungültige Lösungen vorliegen können.
- Anhand eines Abbildungsverfahrens zwischen den Parents werden doppelte Städte, welche sich außerhalb des Kreuzungsbereichs befinden, aus den Proto-Children ersetzt.

Beispiel:

Parent 1: $1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$

Parent 2: $6 \quad 2 \quad 1 \quad 3 \quad 5 \quad 4$

Für die Kreuzungspunkte werden $p_1 = 2$ und $p_2 = 5$ zufällig ausgewählt.

Parent 1: $1 \quad 2 \quad |3 \quad 4 \quad 5| \quad 6$

Parent 2: $6 \quad 2 \quad |1 \quad 3 \quad 5| \quad 4$

Proto-Child 1: $1 \quad 2 \quad |1 \quad 3 \quad 5| \quad 6$

Proto-Child 2: $6 \quad 2 \quad |3 \quad 4 \quad 5| \quad 4$

Im Proto-Child 1 kommt die Stadt 1 redundant im Chromosom vor. Das Abbildungsverfahren wird wie folgt durchgeführt: Stadt 1 in Parent 2 verweist auf Stadt 3 in Parent 1 $\rightarrow$ Stadt 3 in Parent 2 verweist auf Stadt 4 in Parent 1. Dadurch ergibt sich die Abbildung: $1 \rightarrow 3 \rightarrow 4$, womit Stadt 1 durch Stadt 4 ersetzt wird.

Child 1: $4 \quad 2 \quad 1 \quad 3 \quad 5 \quad 6$

Proto-Child 2 beinhaltet zweimal die Stadt 4. Hier wird das Abbildungsverfahren wie folgt abgewickelt: Stadt 4 in Parent 1 verweist auf Stadt 3 in Parent 2 $\rightarrow$ Stadt 3 in Parent 1 verweist auf Stadt 1 in Parent 2. Die Abbildung lautet: $4 \rightarrow 3 \rightarrow 1$, womit Stadt 4 durch Stadt 1 ersetzt wird.

Child 2: $6 \quad 2 \quad 3 \quad 4 \quad 5 \quad 1$

Cycle Crossover

Der Ablauf für das *Cycle Crossover* lautet wie folgt:

- Die Stadt an der Position 1 in Parent 1 wird für Child 1 übernommen.
- Anschließend wird die Stadt unterhalb der zuvor übernommenen Stadt an der selben Stelle für den Child 2 übernommen.
- Nun werden diese ersten beiden Schritte wiederholt. Aber diesmal wird in Child 1 jene Stadt aus dem Parent 1 übernommen, welche zuvor in Child 2 übernommen wurde.
- Sobald in Child 2 eine Stadt übernommen wird, welche in Child 1 bereits vorhanden ist, werden diese Schritte nicht mehr wiederholt, und die restlichen leeren Stellen werden mit den übrigen Städten des jeweils anderen Parent befüllt.

Beispiel:

Parent 1: $1\ 2\ 3\ 4\ 5\ 6$

Parent 2: $3\ 6\ 4\ 1\ 2\ 5$

$$\rightarrow \begin{matrix} 1 & * & * & * & * & * \\ 3 & * & * & * & * & * \end{matrix} \rightarrow \begin{matrix} 1 & * & 3 & * & * & * \\ 3 & * & 4 & * & * & * \end{matrix} \rightarrow \begin{matrix} 1 & * & 3 & 4 & * & * \\ 3 & * & 4 & 1 & * & * \end{matrix}$$

Child 1: $1\ 6\ 3\ 4\ 2\ 5$

Child 2: $3\ 2\ 4\ 1\ 5\ 6$

Order Crossover

Der Ablauf des Verfahrens ist folgendermaßen:

- Bestimme zufällig zwei Kreuzungspunkte, um den dazwischenliegenden Abschnitt der beiden Chromosomen auszutauschen.
- Ersetze im Parent 1 alle Städte, die von Parent 2 nach Parent 1 verschoben werden sollen, durch Platzhalter. Analog für Parent 2.
- Beginne ab dem zweiten Kreuzungspunkt damit, die Platzhalter auf dem Ring nach links zu schieben, bis sich alle im Bereich zwischen den Kreuzungspunkten befinden.

- Übernimm bei beiden Chromosomen die Originalstädte vom anderen Chromosom auf die Platzhalter.

Beispiel:

$$\begin{array}{ll} \text{Parent 1:} & 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \\ \text{Parent 2:} & 6 \quad 3 \quad 5 \quad 2 \quad 1 \quad 4 \end{array}$$

Für die Kreuzungspunkte werden $p_1 = 2$ und $p_2 = 4$ zufällig ausgewählt.

$$\begin{array}{ll} \text{Parent 1:} & 1 \quad 2 \quad |3 \quad 4| \quad 5 \quad 6 \\ \text{Parent 2:} & 6 \quad 3 \quad |5 \quad 2| \quad 1 \quad 4 \end{array}$$

$$\rightarrow \begin{array}{cccccc} 1 & . & |3 & 4| & . & 6 \\ 6 & . & |5 & 2| & 1 & . \end{array} \rightarrow \begin{array}{cccccc} & & 3 & 4 & |. & .| & 6 & 1 \\ & & 5 & 2 & |. & .| & 1 & 6 \end{array} \rightarrow$$

$$\begin{array}{ll} \text{Child 1:} & 3 \quad 4 \quad 5 \quad 2 \quad 6 \quad 1 \\ \text{Child 2:} & 5 \quad 2 \quad 3 \quad 4 \quad 1 \quad 6 \end{array}$$

In [1] werden verschiedene TSP Kreuzungsverfahren bezüglich ihrer Performance getestet, wobei das Order Crossover Verfahren die besten Ergebnisse liefert.

5.5.2 Mutationsverfahren für das Traveling Salesman Problem

In einem GA hat der Mutationsoperator die Aufgabe, die in den Genen enthaltenen Informationen mit einer geringen Wahrscheinlichkeit zu verändern, um so zu verhindern, dass wichtiges Genmaterial ausstirbt oder nie der Population zugeführt wird. Eine gängige Mutationswahrscheinlichkeit, liegt bei $p = 1/n$, wobei n für die Anzahl der Gene eines Chromosoms steht [22]. Je nach Spezifikation des Mutationsoperators lässt sich

- Mutation wie eine *Hill-Climbing-Variante*[24] verwenden, die durch das Erzeugen kleiner Änderungen in der Umgebung der durch ein Chromosom dargestellten Lösung zielgerichtet auf das Optimum zuläuft (lokale Optimierung).
- durch weitläufige Mutation die Vielfältigkeit in der Population garantieren und dafür zu sorgen, dass lokale Optima wieder verlassen werden können, um das globale Optimum zu erreichen. [12]

[24] Bei Hill-Climbing wird ausgehend von einer Lösung ein zufälliger Schritt in Richtung eines Nachbarn ausgeführt. Ist die Bewertung der neuen Lösung besser als die vorhergehende, so wird die neue Lösung behalten. Ansonsten wird mit der alten Lösung weitergearbeitet.

Die häufigst verwendeten Mutationsoperatoren für Reihenfolgenprobleme werden in [7] beschrieben. Zur Mutation eines Chromosoms eignet sich beispielsweise eine *Inversion* einer zufällig ausgewählten Teilsequenz:

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$$

$$1 \quad 5 \quad 4 \quad 3 \quad 2 \quad 6$$

Diese Veränderung entspricht dem Nachbarschaftsoperator der 2-Opt Heuristik, da zwei bestehende Kanten entfernt und durch zwei neue Kanten ersetzt werden.
Die Verschiebung einer zufällig ausgewählten Teilsequenz an eine andere Position eignet sich ebenfalls als Mutationsoperator:

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$$

$$1 \quad 5 \quad 6 \quad 2 \quad 3 \quad 4$$

Diese Verschiebung ist eine Veränderung der 3-Opt Heuristik.
Eine weitere gängige Mutation für Chromosomen dieser Art ist der Austausch zweier zufällig ausgewählter Gene:

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$$

$$1 \quad 2 \quad 5 \quad 4 \quad 3 \quad 6$$

5.6 Anwendung genetischer Algorithmen für das Vehicle Routing Problem

Während für GA bezüglich des TSP reichlich Literatur vorhanden ist, gibt es bisher noch nicht viele Artikel die sich dem VRP widmen. Die Formulierung eines Modells zum lösen des VRP mit GA stellt eine Herausforderung dar, da es schwierig ist, ein Individuum (Tourenplan) als ein Chromosom von Genen darzustellen. Des Weiteren stellt sich anschließend die Frage, wie eine sinnvolle Kreuzung zweier Tourenpläne durchzuführen ist, um akzeptable Nachfahren zu erzeugen.
In [19] wird ein Modell für einen CVRP GA vorgestellt, in welchem die genetischen Operatoren des TSP aus Kapitel 5.5 angewendet werden können, indem die Tourenpläne als Giant Tour Chromosomen dargestellt werden. Der Tourenplan:

Tourenplan L

Tour 1	1,8,5
Tour 2	4,9,3,10
Tour 3	2,7,6

könnte als Chromosom folgend aussehen:

Chromosom von L: $4 \quad 9 \quad 3 \quad 10 \quad 1 \quad 8 \quad 5 \quad 2 \quad 7 \quad 6$

Die einzelnen Zahlen repräsentieren die Kunden die beliefert werden. Dabei ist zu beachten, dass in dieser Darstellung der Individuen keine Informationen über die Zuordnung der Kunden zu den einzelnen Fahrzeugen gespeichert wird, da keine Trennzeichen vorhanden sind.
Um die Fitness eines Chromosoms dieser Art zu bestimmen, muss zuerst eine optimale Spaltung des Chromosoms mit Berücksichtigung der Fahrzeugrestriktionen durchgeführt werden. Dies kann beispielsweise mit dem Petal Algorithmus aus Kapitel 4.3.3 durchgeführt werden. Nach der Aufteilung der Kunden:

$$|4 \quad 9 \quad 3 \quad 10| \quad |1 \quad 8 \quad 5| \quad |2 \quad 7 \quad 6|$$

können die Kosten des Tourenplans mit $c(L) = \sum_{i=0}^{k} c(T_i)$ berechnet werden. Da Tourenpläne mit geringeren Kosten bessere Lösungen darstellen berechnet sich die Fitness dieser Individuen durch:

$$f(L) = \frac{1}{c(L)}$$

Ein Vorteil dieser Chromosommodellierung ist, dass mittels Kreuzung der Individuen durch TSP Kreuzungsoperatoren keine ungültigen Lösungen erezugt werden können, da erst bei der Berechnung der Fitness eine gültige Aufteilung der Giant Tour erfolgt. Ausgehend von dieser Chromosomdarstellung wird in [19] ein Experiment durchgeführt, in dem sieben Instanzen von Christofides Mingozzi und Toth getestet werden. Das Design des Experiments wird folgendermaßen beschrieben:

- Ein GA wird jeweils für mehrere bekannte TSP Kreuzungsverfahren einzeln durchgeführt, von denen drei bereits in Abschnitt 5.5.1 beschrieben wurden.
- In einer weiteren Variante wird ein Mix aus allen Kreuzungsverfahren verwendet, wobei jedes Kreuzungsverfahren mit einer gleichverteilten Wahrscheinlichkeit zum Einsatz kommt.
- Für die Mutationwahrscheinlichkeit wird $p = 0.01$ gewählt.
- Als Mutationsoperatoren werden die drei Verfahren aus Abschnitt 5.5.2 mit gleichen Wahrscheinlichkeiten verwendet.

- Die Populationsgröße $|P|$ wird mit 30 Individuen festgelegt.
- Die Startpopulation wird mit zufälligen Chromosomen erzeugt.
- Die Versuche wurden einmal mit, einmal ohne Mutation durchgeführt.

Folgende Fakten konnten aufgrund der Resultate beobachtet werden:

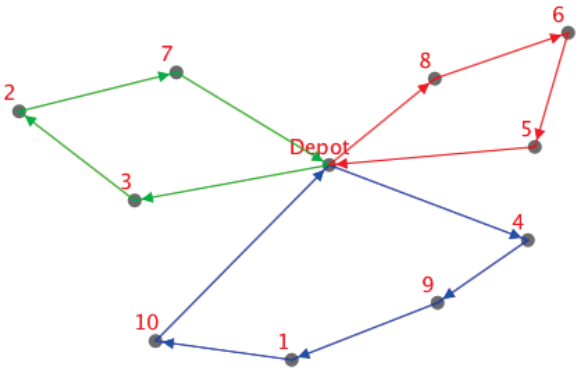
- Mutation mit einer Wahrscheinlichkeit von $p = 0.01$ verbessert immer die Performance von GA
- Die Verwendung mehrerer Kreuzungsoperatoren erzeugt bessere Resultate, als bei Verwendung nur eines Kreuzungsverfahrens erzielt werden.

Der GA mit den gemischten Kreuzungsverfahren konnte auf den Testinstanzen durchaus akzeptable Tourenpläne erzeugen. Der positive Synergieeffekt multipler Kreuzungsoperatoren stellt eine wichtige Erkenntnis dar. Jedoch konnte auf keiner Instanz das bisher beste bekannte Resultat erzielt werden. Ein großer Nachteil dieser GA Implementierung ist die Rechenzeit, da die Berechnung der Fitness der Individuen aufgrund der optimalen Aufsplittung der Chromosomen äußerst teuer ist.
In [26] wird eine ähnliche Implementierung, auf der selben Modellierung der Chromosomen basierend, zur Lösung der selben Testinstanzen verwendet. Jedoch wird nur das Order Crossover Verfahren, statt multiple Kreuzungsverfahren, für die Kreuzung der Individuen verwendet. Des Weiteren wird für die Mutation ein lokales Suchverfahren auf die Individuen mit einer Wahrscheinlichkeit von $p = 0.05$ angewandt. Somit handelt es sich hierbei um einen hybridisierten GA. Die Ergebnisse dieser Implementierung sind sehr gut und schlagen die, die in [19] erzielt wurden. Die guten Ergebnisse können auf die Hybridisierung zurückgeführt werden, wodurch sich die benötigte Rechenzeit natürlich erhöht. Bei einer Instanz mit 199 Kunden betrug die Rechenzeit für den GA beispielsweise bereits fast eine Stunde. Für größere Instanzen wird dieses Verfahren daher in der Praxis nur schwer anwendbar sein.

5.6.1 Genetischer Algorithmus von Pereira

Eine Alternative Repräsentation eines GA für ein CVRP wird von Pereira in [25] vorgestellt. Hierbei wird ein Chromosom aus mehreren Gensträngen dargestellt, wobei ein einzelner Strang eine Tour repräsentiert. Abbildung 5.3 zeigt wie ein solches Chromosom aus einem gegebenem Tourenplan gebildet wird.
Diese Chromosomdarstellung erlaubt unter Umständen, dass die Ladekapazitäten der Fahrzeuge überschritten werden könnten. Deshalb gibt es für die Chromosomen eine Reparaturfunktion. Angenommen bei dem Chromosom:

$$T_1: \ 7\ 8\ 1 \quad T_2: \ 4\ 2\ 5\ 3\ 6$$



Chromosom: $T_1: \ 3\ 2\ 7 \quad T_2: \ 4\ 9\ 1\ 10 \quad T_3: \ 8\ 6\ 5$

Abbildung 5.3: Darstellung eines Tourenplans als Chromosom nach Pereira

wird ab Kunde 5 die Kapazität von T_2 überschritten. So wird dieser Genstrang wie folgt aufgeteilt:

$$T_1: \ 7\ 8\ 1 \quad T_2: \ 4\ 2 \quad T_3: 5\ 3\ 6$$

Als genetischer Kreuzungsoperator wird ein Verfahren verwendet, bei dem die Genfragmente eines Individuums an ein anderes Individuum vererbt wird. Algorithmus 18 beschreibt im Detail wie dieses Verfahren aus den Eltern I_1 und I_2 einen Nachfahren I_c erzeugt. Indem I_1 und I_2 vertauscht werden, kann anschließend ein zweites Kind gebildet werden.

Algorithmus 18 Pereria Crossover

Input: I_1, I_2

$s \leftarrow$ Wähle aus I_2 eine zufällige Subroute aus einer Tour. $(s = \{v_1, v_2, ..., v_n\})$
$c \leftarrow$ Finde den Kunden c, der am nächsten zu v_1 ist, unter der Bedingung: $c \notin s$.
$I_c \leftarrow I_1$
Entferne aus I_c alle Kunden die in s vorkommen.
Füge s in I_c direkt nach c ein, sodass die Subroute s und c nun zu der selben Tour gehören.

Wende die Reparaturfunktion auf I_c an.

Abbildung 5.4 soll dieses Vorgehen anhand eines Beispiels näher veranschaulichen. Dabei wurde noch nicht die Reparaturfunktion auf das erzeugte Kind I_c angewendet.
Für die Mutation der erzeugten Kinder werden folgende Operatoren in Betracht gezogen:

- *Swap:* Wähle zufällig zwei Kunden und vertausche diese. Die ausgewählten Kunden können zur selben oder unterschiedlichen Touren gehören.
- *Inversion:* Wähle eine Subroute einer Tour und invertiere die Reihenfolge dieser.
- *Insertion:* Wähle einen Kunden und füge ihn an einer anderen zufälligen Stelle ein. Die Einfügestelle kann entweder in der selben, oder einer anderen Tour liegen. Mit einer

Individuum I_2

Tour 1	3	2	7	
Tour 2	4	9	1	10
Tour 3	8	6	5	

Individuum I_1

Tour 1	1	4			
Tour 2	2	9	3	8	7
Tour 3	10				
Tour 3	6	5			

Child I_c

Tour 1	4				
Tour 2	2	3	8	7	
Tour 3	6	9	1	10	5

Abbildung 5.4: Beispiel für Pereira Crossover
Für die Subroute wurde $s = \{9, 1, 10\}$ gewählt. Des Weiteren ist der Kunde 6 dem Kunden 9 am nächsten: $c = 6$.

Wahrscheinlichkeit von $p = \frac{1}{2 \cdot k}$ (k repräsentiert die Anzahl der Fahrzeuge) soll der Kunde auf einer neuen Tour eingefügt werden.

- *Displacement:* Gleich wie der Insertion Operator, nur wird hier im Gegensatz zu einem einzelnen Kunden eine Subroute gewählt.

Falls die Mutationsoperatoren innerhalb einer Tour angewendet werden, sind diese mit den bereits beschriebenen Verfahren aus Abschnitt 5.5.2 ident. Es ist zu beachten, dass nach einer erfolgten Mutation eines Tourenplans die Reparaturfunktion eingesetzt werden muss, da Kapazitätsverletzungen auftreten könnten.
Basierend auf diesem Modell wurden in [25] Testläufe mit bekannten Instanzen durchgeführt.[25] Die Autoren verwendeten folgende Parameter:

- Anzahl an Generationen: $k = 50000$
- Populationsgröße: $|P| = 200$
- Als Selektionsmethode wurde die Wettkampf-Selektion mit $q = 5$ gewählt.
- 25% der besten Individuen aus $P(t)$ werden direkt für $P(t+1)$ übernommen $\rightarrow$ Elitismus Strategie.
- Mutationsraten:
  1. Swap: $p = 0.05$
  2. Inversion: $p = [0.1, 0.15]$
  3. Insertion: $p = 0.05$
  4. Displacement: $p = [0.15, 0.2]$
- Die Startpopulation wurde mit zufälligen Chromosomen erzeugt.

[25] Augerat und "Christofides and Eilon" Testinstanzen

Die Testresultate zeigen, dass dieser GA eine effektive Implementierung ist. In manchen der getesteten Instanzen konnten sogar neue beste Lösungen erzielt werden. Das Verfahren hat des Weitern den Vorteil, dass die genetischen Operatoren in Bezug auf Rechenzeit billig sind, was zu einer kurzen Laufzeit führt. Dennoch muss hier abschließend erwähnt werden, dass in [25] nur kleine Instanzen[26] getestet wurden, wodurch keine Aussage über die Performance der Implementierung für größere Instanzen getroffen werden kann.

[26] Die größte Instanz beinhaltet 80 Kunden

6 Entwickelte Software

Wie bereits zu Beginn dieser Arbeit erwähnt, wurde eine auf JAVA basierende Software für die Lösung von VRPs entwickelt. Dieses Kapitel soll eine Übersicht geben, welche Algorithmen für diese Anwendung implementiert wurden und auf welchen Testinstanzen beziehungsweise Problemstellungen deren Effizienz untersucht wurde. Anschließend werden die Resultate der durchgeführten VRP Testläufe in Kapitel 7 präsentiert. Abbildung 6.1 zeigt das Hauptfenster der Software, in der die geographischen Daten von Leoben mittels OSM[27] geladen wurden. Zunächst soll eine kurze Beschreibung der importierten geographischen Daten erfolgen.

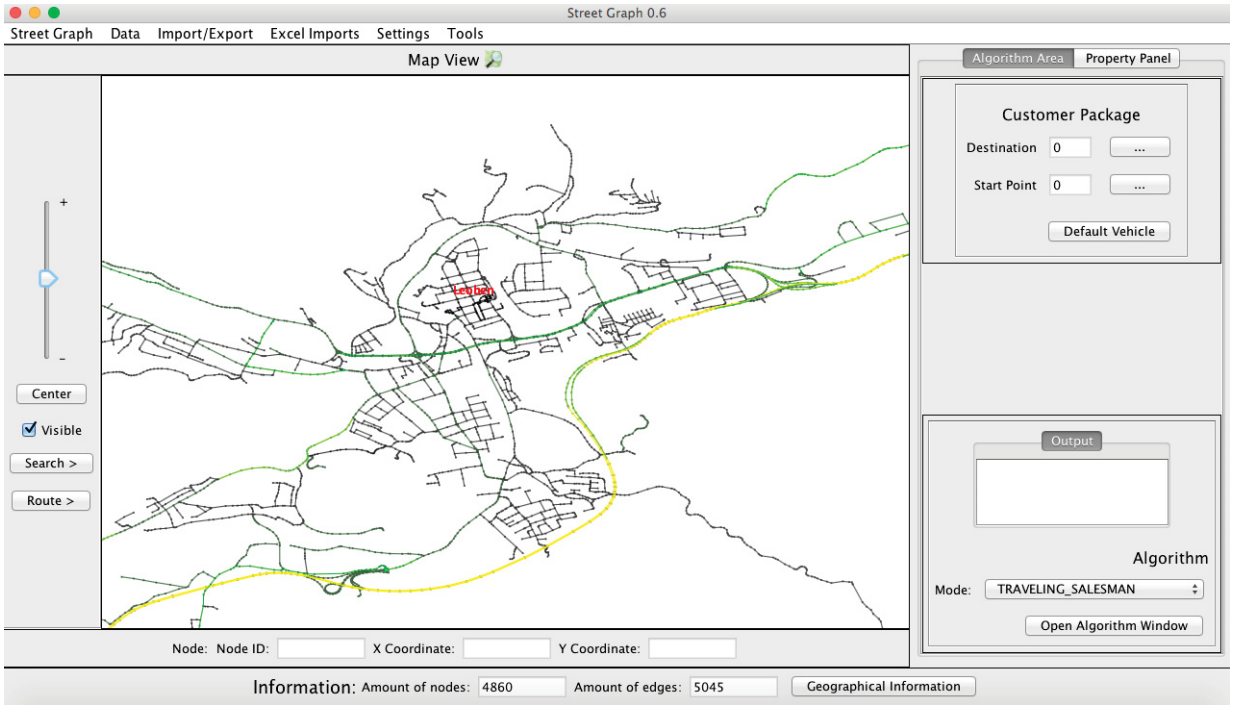


Abbildung 6.1: Hauptfenster der Software

6.1 Open Street Map

Für die Software wurde eine Schnittstelle entwickelt, mittels der geographische Kartendaten insbesondere Straßennetze geladen und als Graph dargestellt werden können.
Bei OSM handelt es sich um ein Projekt, das für jeden frei nutzbare Geodaten sammelt. Auf der offiziellen Homepage können geographische Daten in XML Form für beliebig wählbare Gebiete

[27] Open Street Map

heruntergeladen werden. Anhand dieser Daten kann die Anwendung einen Graphen konstruieren und diesen visualisieren, so wie er in Abbildung 6.1 in der Map View dargestellt wird. Basierend auf diesen Daten können VRPs sowie TSPs mit konfigurierbaren Kunden- und Fahrzeugdaten gelöst werden. Abbildung 6.2 zeigt wie diese OSM Daten aufgebaut sind. In jedem dieser Files gibt es einen Abschnitt, in dem die einzelnen Knoten aufgelistet werden und einen weiteren Abschnitt, in dem die Wege definiert werden. Die Knoten beinhalten neben einer eindeutigen ID auch Längen- und Breitengrade. Zusätzlich können auch Adressinformationen wie Stadt, Postleitzahl, Straße und Hausnummer hinterlegt sein, so wie beispielsweise in Abbildung 6.2(a).

```
<node id="1667308473" visible="true" lat="47.1995445" lon="15.3359489"/>
<node id="1667308477" visible="true" lat="47.1995753" lon="15.3352584"/>
<node id="308563165" visible="true" lat="47.1961539" lon="15.3359986"/>
- <node id="285861268" visible="true" lat="47.1993653" lon="15.3370693">
    <tag k="addr:city" v="Deutschfeistritz"/>
    <tag k="addr:housenumber" v="147"/>
    <tag k="addr:postcode" v="8121"/>
    <tag k="addr:street" v="Deutschfeistritz"/>
  </node>
```

((a)) Knoten der OSM Daten

```
- <way id="57065660" visible="true">
    <nd ref="328054124"/>
    <nd ref="328054131"/>
    <nd ref="2427878066"/>
    <nd ref="328054138"/>
    <nd ref="2427878050"/>
    <nd ref="328054143"/>
    <nd ref="328054148"/>
    <nd ref="2427878076"/>
    <nd ref="328054156"/>
    <nd ref="712568035"/>
    <tag k="highway" v="residential"/>
    <tag k="is_in" v="Deutschfeistritz,Graz-Umgebung,Steiermark,Österreich,Europe"/>
    <tag k="maxspeed" v="30"/>
    <tag k="name" v="Schießstattgasse"/>
  </way>
```

((b)) Wege der OSM Daten

Abbildung 6.2: OSM XML Daten

Die Wege bestehen aus einer Reihe von Knoten, sowie zusätzlichen Attributen. Der Weg aus Abbildung 6.2(b) setzt sich beispielsweise aus 10 aufeinanderfolgenden Knoten zusammen. Das Schlüsselwort "highway" bildet das Hauptattribut für Straßen und Pfade. Andere Wege könnten Flüsse oder Grenzen darstellen, welche für diese Problemstellung nicht benötigt werden und bereits im Vorhinein rausgefiltert werden. Die zulässige Höchstgeschwindigkeit in

km/h, Straßenname und geographische Lage werden ebenfalls angegeben. Optional können noch viele weitere Attribute mit zusätzlichen Informationen aufgelistet werden, wie beispielsweise "maxweight", welches das maximal erlaubte Gesamtgewicht in Tonnen vorschreibt, oder "oneway", das eine Einbahnstraße kennzeichnet. Neben dem Schlüsselwort "highway" wird auch noch der Straßentyp angegeben. Folgende Tabelle beinhaltet die wichtigsten OSM Straßentypen und ihre Definitionen:

Straßentypen:

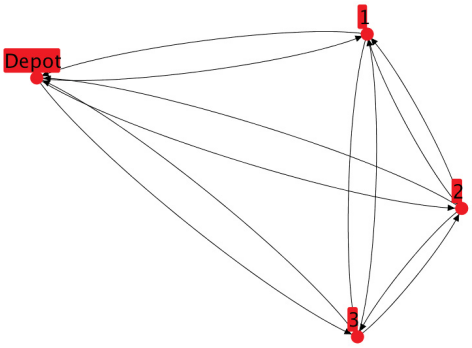
Schlüsselwort	**Beschreibung**
motorway	Autobahn
trunk	Autobahnähnliche Straße
primary	Bundesstraße
secondary	Gut ausgebaute Landstraße
tertiary	Nicht so gut ausgebaute Landstraße
unclassified	Nebenstraße (schmale Landstraße)
residential	Straße in einem Wohngebiet
service	Erschließungsweg
motorway-link	Autobahn-Zubringer

Die entwickelte OSM Schnittstelle der Software liest die Daten zeilenweise und erstellt dadurch den Graphen, wobei die Straßen in Abhängigkeit ihrer zulässige Höchstgeschwindigkeit mit unterschiedlichen Farben dargestellt werden. Dabei kann der Anwender zu Beginn wählen, welche Straßentypen importiert werden sollen. Die Bewertung der Kanten erfolgt in Form der benötigten Zeit, die das Fahrzeug für den jeweiligen Straßenabschnitt benötigt: $t = \frac{s}{v}$. Um den importierten OSM Graph für den Anwender übersichtlicher zu gestalten, werden die Namen der jeweiligen Städte in roter Schrift in der Map View angezeigt.

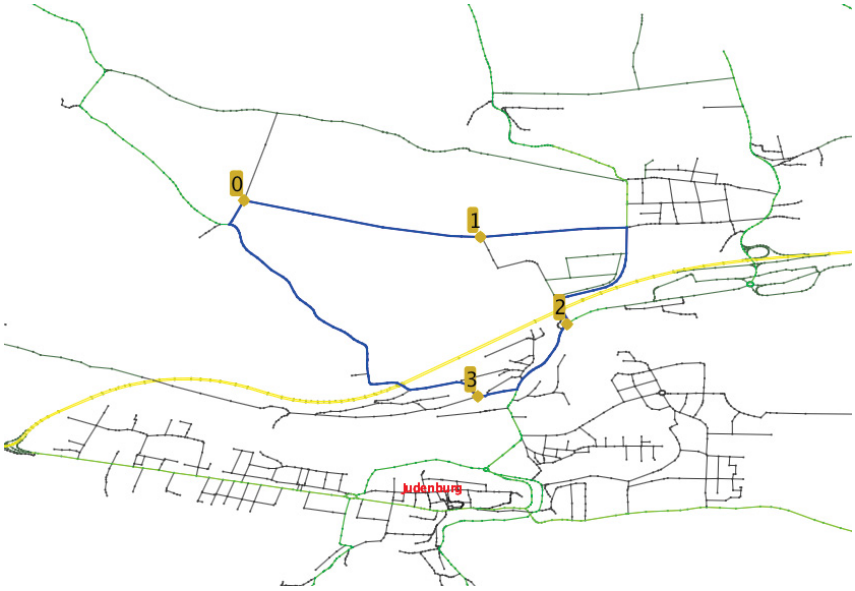
6.2 Implementierte Algorithmen

Mit Hilfe der entwickelten Software können sowohl VRP als auch TSP Algorithmen gestartet werden. Diese können entweder auf mittels OSM erstellten Graphen mit konfigurierbaren Kunden- und Fahrzeugdaten, oder auf Graphen aus Testinstanzen mit gegebenen Kunden- bzw. Fahrzeugdaten aus dem Internet, angewendet werden.
Ausgangsbasis für alle implementierten Algorithmen ist immer ein vollständiger metrischer Graph K_n, in dem jeder Knoten mit allen anderen verbunden ist und $c(u, w) \leq c(u, v)+c(v, w)$ für alle Knoten $u, v, w \in V(K_n)$ gilt. Die einzelnen Knoten repräsentieren dabei das Depot und die Kunden. Daher müssen geographische Graphen wie der aus Abbildung 6.3(a) zuerst in einen vollständigen metrischen Graphen wie in Abbildung 6.3(b) überführt werden, der nur noch die relevanten zu besuchenden Knoten und das Depot enthält. Um die Kanten dieses vollständigen Graphen K_n zu bilden, müssen auf G die kürzesten Pfade zwischen allen Kunden und dem

((a)) OSM Graph G



((b)) Umgewandelter vollständiger gerichteter Graph K_n



((c)) TSP Lösung auf dem OSM Graph. $T = \{0, 1, 2, 3, 0\}$.

Abbildung 6.3: Umwandlung: OSM Graph zu vollständigen Graph

Depot in beide Richtungen gebildet werden.[28] Die kürzesten Pfade werden in der Software mithilfe des Algorithmus von Dijkstra aus Kapitel 2.1.2 berechnet. Die Bewertungen der Kanten von K_n entsprechen den Längen der kürzesten Pfade zwischen den jeweils betrachteten Knoten. Nachdem ein TSP- oder VRP-Algorithmus auf K_n angewandt wurde, kann anschließend deren Ergebnis auf dem OSM Graph wie in Abbildung 6.3(c) visualisiert werden, da sich die Anwendung merkt, welche Kantenmenge $E' \subset E(G)$ Bestandteil der Kante $e \in E(K_n)$ ist.

6.2.1 Implementierte Algorithmen für das Travelling Salesman Problem

Die im Zuge dieser Arbeit implementierten TSP-Algorithmen können entweder für das Routing innerhalb eines VRP, oder auch direkt zur Lösung von TSP Aufgaben verwendet werden. Die Software verfügt über eine TSPLIB Schnittstelle, mit dessen Hilfe die verschiedenen Instanzen importiert werden können, von denen bereits einige in Kapitel 3 dargestellt wurden. Folgende Algorithmen wurden in der Software implementiert:

- Die beschriebenen Eröffnunsheuristiken von Abschnitt 3.2.1.
- Von den Verbesserungensheuristiken aus Abschnitt 3.2.2 wurden:
  - Eine 2-Opt Implementierung mit einer First Improvement Strategie
  - und die Insertion Search Heuristik umgesetzt.
- Ein Branch and Bound Verfahren, wie es in Kapitel 3.1 beschrieben wurde. Jedoch eignet sich dieser Algorithmus aufgrund der hohen Rechenzeit kaum für das Routing eines VRP.

Des Weiteren können TSPs auch mit einem implementierten GA gelöst werden. Hierfür wurde für die Software ein GA Framework entwickelt, welches sich ebenfalls für die Lösung von VRPs eignet. Dieses Framework, welches in Algorithmus 19 beschrieben wird, benötigt folgende Input-Parameter:

- Eine Startpopulation P
- Maximale Anzahl an Generationen
- CR[29]: Definiert in % wie viele Individuen von P in den Vermehrungspool P' gelangen. (Wenn CR $< 100\% \rightarrow$ Elitismus. Da die restlichen $|P| - |P^*|$ freien Plätze der nächsten Generation mit den besten Individuen von P befüllt werden.)
- Eine Selektionsmethode: selectionMethod

[28] Da aufgrund von Einbahnstraßen die kürzesten Wege von v_i nach v_j und v_j nach v_i unterschiedlich sein können.

[29] Crossover Rate

- Liste von Kreuzungsverfahren: crossoverMethods
- Liste von Mutationsmethoden: mutationMethods

Algorithmus 19 Framework für genetischen Algorithmus

Input: (P, Anzahl an Generationen, CR, selectionMethod, crossoverMethods, mutationMethods)

1: **for** 0 **to** Anzahl an Generationen **do**
2: $P^{'} \leftarrow$ selectionMethod.selectIndividuals(P, CR)
3: $P^{*} \leftarrow \emptyset$ Menge der Kinder
4: **while** $P^{'} \neq \emptyset$ **do**
5: Entnimm aus $P^{'}$ die Eltern I_1 und I_2.
6: $M \leftarrow$ Erzeuge die Kinder durch Kreuzung von I_1 und I_2 mit einem zufälligen Kreuzungsverfahren von crossoverMethods.
7: $P^{*} \leftarrow P^{*} \cup M$
8: **end while**
9: **for all** $m \in$ mutationMethods **do**
10: **for all** $I_c \in P^{*}$ **do**
11: $I_c \leftarrow$ Führe mit einer konfigurierten Mutationswahrscheinlichkeit p eine Änderung gemäß der Mutationsmethode m an dem Chromosoms durch.
12: **end for**
13: **end for**
14: $P \leftarrow P^{*} \cup \{|P| - |P^{*}|$ besten $I \in P\}$
15: **end for**

Erläuterung von Zeile 6 aus Algorithmus 19: Für die übergebenen Kreuzungsmethoden muss eine Wahrscheinlichkeitsverteilung angegeben werden, mit welcher vor jeder Kreuzung eins dieser Verfahren ausgewählt wird
Erklärung für Zeile 9 bis 13 von Algorithmus 19: Für jede übergebene Mutationsmethode kann eine Mutationswahrscheinlichkeit $p(0 < p < 1)$ konfiguriert werden. Zusätzlich ist es auch möglich, jeder Mutationsmethode ein Generationsintervall zuzuordnen, in dem diese aktiv ist. Die Startpopulation kann vom Anwender frei konfiguriert werden, wodurch entweder mit zufälligen und oder durch Eröffnungsheuristiken erzeugten Individuen gearbeitet werden kann. Folgende GA Operatoren sind in der Software für das TSP implementiert:

- Kreuzungsverfahren
  - Partially Matched Crossover
  - Order Crossover
- Mutationsverfahren
  - Displacement
  - Inversion
  - Swap

  - Local Search

- Selektionsverfahren
  - Reine Turnierauswahl

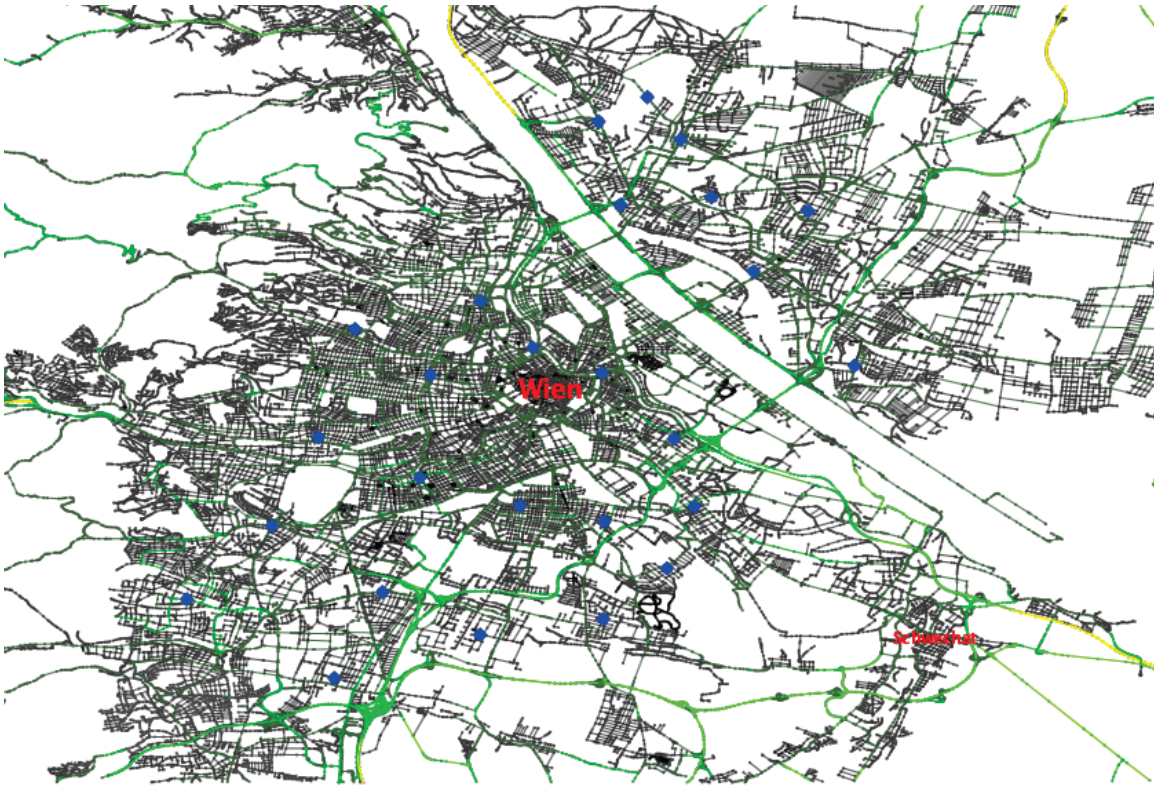
Durch Hinzufügen einer Local Search Mutationsmethode kann der GA durch eine 2-Opt oder Insertion Search Heuristik hybridisiert werden.
Folgender TSP Testlauf soll die Effektivität der GA verdeutlichen:
In Abbildung 6.4 wird eine erstellte TSP Aufgabe, welche mithilfe des GA in der Software gelöst wurde veranschaulicht. Die erzeugte Tour besitzt eine Gesamtlänge von 91,21km und eine benötigte Reisezeit von 1 Stunde und 53 Minuten. Bei dieser Reisezeit werden die Stehzeiten, welche in Wien unumgänglich sind, jedoch nicht mitberücksichtigt. Eine anschließende rechenintensive Branch and Bound Berechnung konnte bestätigen, dass es sich bei dieser Rundreise um eine optimale Tour handelt. Doch im Gegensatz zu dem exakten Branch and Bound Verfahren konnte der GA das optimale Ergebnis bereits nach 8,94s erreichen, während das exakte Verfahren mehrere Minuten für die Berechnung benötigte. Folgende Parameter wurden hierbei für den Algorithmus 19 verwendet:

- Startpopulation: $|P| = 300$ zufällige Touren.
- Anzahl an Generationen: 2500
- $CR = 80\%$
- Kreuzungsverfahren: Order Crossover : Partially Matched Crossover $\rightarrow$ 4 : 1
- Mutationsmethoden:
  - Inversion $p = 0.05$
  - Displacement $p = 0.03$
  - Swap $p = 0.01$
  - 2-Opt Hybridisierung $p = 0.02$ von Generation: $2000 - 2500$
- Selektionsverfahren: Reine Turnierauswahl $q = 2$

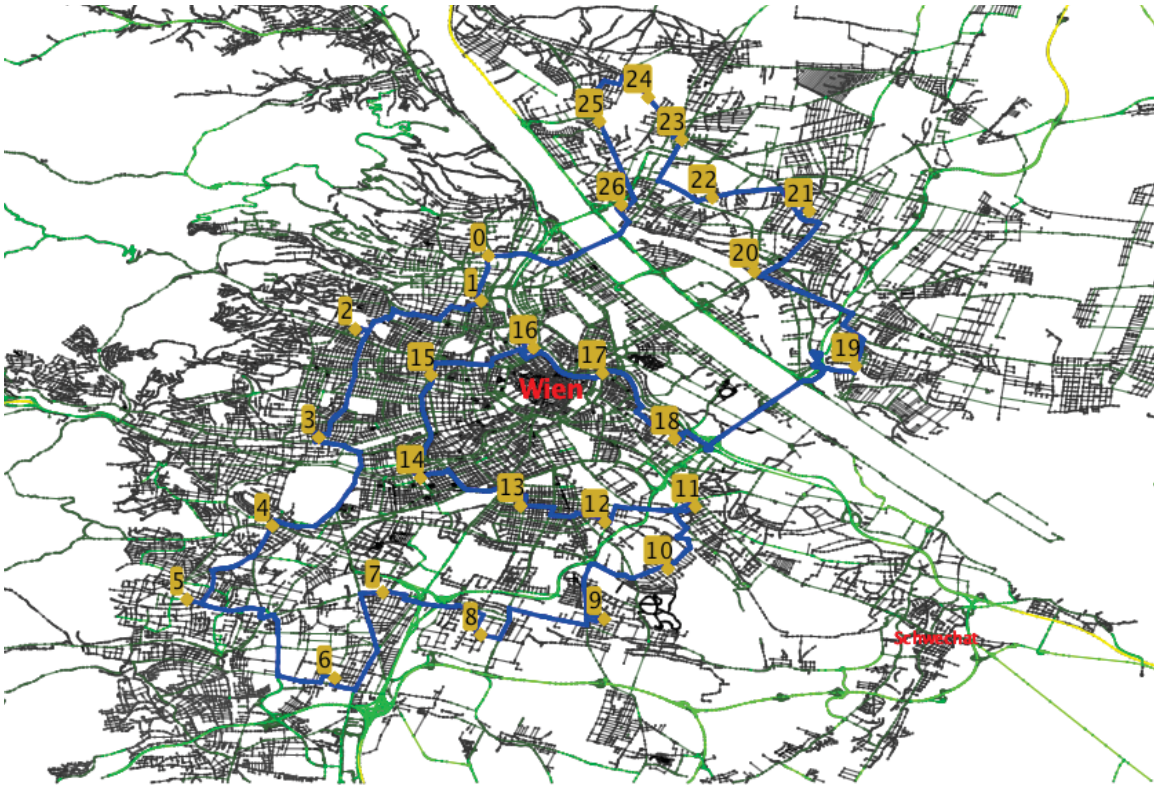
6.2.2 Implementierte Algorithmen für das Vehicle Routing Problem

Die in der Software implementierten VRP Algorithmen stellen das Herzstück der entwickelten Anwendung dar. Folgende Eröffnungsalgorithmen, welche bereits in Kapitel 4 beschrieben wurden, stehen dem Anwender der Software zur Verfügung:

- Ein paralleles Clark and Wright Savings-Verfahren mit konfigurierbarem λ Parameter.



((a)) Wien TSP Aufgabe



((b)) TSP Lösung

Abbildung 6.4: Gelöste TSP Instanz mit Hilfe des genetischen Algorithmus

- Sweep Algorithmus mit beliebigen Startwinkeln.
- Petal Algorithmus.
- RFCS[30]: Die Giant Tour kann mit einem konfigurierbaren TSP Verfahren gelöst werden, während die anschließende Aufteilung zu gültigen Touren entweder auf optimalen Wege mit Hilfe des in Abschnitt 4.3.3 beschriebenen Verfahrens, oder durch Erstellung größtmöglicher aufeinander folgender Touren erfolgen kann.

Zu den implementierten VRP Verbesserungsheuristiken gehören:

- Lokale Suche mit dem λ-*Interchange* Operator für beliebig viele Touren.
- Lokale Suche mit RAR. Dabei wird für den Ruin Schritt das Radial Ruin Verfahren verwendet und anschließend der Tourenplan durch ein Best Insertion Verfahren wiederhergestellt.[31]
- Single-Route Verbesserung.

Die Multi-Route Verbesserungsverfahren λ-*Interchange* und RAR wurden mit einer First Improvement Strategie verwirklicht, da diese kürzere Laufzeiten als Best Improvement Strategien benötigen.
Der Anwender kann durch Hilfe des implementierten *Tour Plan Creator* für die Lösung von VRPs eine beliebige Eröffnungsheuristik wählen und beliebig viele Verbesserungsheuristiken in beliebiger Reihenfolge anhängen. Alternativ steht der GA von Algorithmus 19 zur Lösung von VRPs zur Verfügung. Für die GA Operatoren werden jene von Abschnitt 5.6.1 verwendet:

- Kreuzungsverfahren: Pereira Crossover.
- Mutationsverfahren:
  - Swap
  - Insertion
  - Displacement

Zusätzlich wurde ein *Route* Mutationsoperator eingeführt, mit welchem die einzelnen Touren eines Tourenplans mit den bereits beschriebenen TSP Mutationsoperatoren mutiert werden. Bei dieser Mutationsmethode werden die einzelnen Touren eines Tourenplans als Individuen betrachtet und mit einer vorgegebenen Wahrscheinlichkeit mutiert.
Des Weiteren können als Mutationsoperatoren die implementierten VRP Verbesserungsheuristiken eingesetzt werden, wodurch der GA hybridisiert werden kann.

[30] Route First Cluster Second
[31] Siehe Abschnitt 4.4.2

Als Selektionsmethode wird hier ebenfalls die reine Turnierauswahl eingesetzt. Schlussendlich kann die benötigte Startpopulation durch den Tour Plan Creator erzeugt werden. Hier ist es durchaus möglich, die Startpopulation durch gemischte Verfahren zusammenzusetzen. Es empfiehlt sich, die Startpopulation durch randomisierte Verfahren zu erzeugen, um eine möglichst große Vielfalt an Tourenplänen für den GA bereitzustellen. Als Algorithmen zur Erzeugung der Startindividuen eignet sich daher der Sweep Algorithmus mit zufälligen Startwinkel, sowie das RFCS Verfahren bei dem die Giant Tour durch die Random Insertion Heuristik gebildet wird. Da der entwickelte GA für das VRP vielversprechende Ergebnisse liefert, werden in Kapitel 7 ausführliche Testläufe mit diesem durchgeführt. Zur Ermöglichung der Testläufe, wurde in der Software eine Schnittstelle entwickelt, mit der die "Christofides, Mingozzi and Toth"-Instanzen importiert werden können. Einige dieser Instanzen wurden bereits in Kapitel 4 abgebildet um verschiedene Tourenpläne von unterschiedlichen Verfahren zu veranschaulichen.

7 Testresultate

7.1 Christofides, Mingozzi und Toth Instanzen

Um die Qualität des entwickelten GA zu testen, wurden Testläufe auf den 14 "Christofides, Mingozzi and Toth"-Instanzen durchgeführt. Bei den Instanzen 1-5 und 11-12 handelt es sich um CVRPs, da die einzelnen Touren nur durch die vorgegebene Ladekapazität beschränkt werden. Für die restlichen Instanzen wird zusätzlich eine maximale Länge der Touren vorgegeben, wodurch diese Instanzen eine Mischung von CVRPs und DCVRPs darstellen.
In den folgenden Tabellen werden die Resultate der Testläufe, sowie deren Inputparameter für den GA präsentiert. Für jede Instanz wurden hierbei zwei Testläufe durchgeführt. Im ersten Durchlauf wurde dabei jeweils kein λ-*Interchange* Operator als Mutationsverfahren eingesetzt, während dieser bei dem zweiten Durchlauf angewendet wird. Dieses Vorgehen soll veranschaulichen, dass durch den zusätzlichen Einsatz des λ-*Interchange* Operator als Mutationsverfahren bessere Ergebnisse erzielt werden können. Natürlich muss berücksichtigt werden, dass diese besseren Resultate durch eine längere Rechenzeit erkauft werden. Als Selektionsmethode wurde für alle Instanzen eine reine Turnierauswahl mit $q = 2$ eingesetzt.
Folgender Rechner wurde zur Lösung der Instanzen verwendet:

- MacBook Pro
- Prozessor: 2,7 GHz Intel Core i7
- Speicher: 8 GB 1333 MHz DDR3

Schlussendlich wird in Abschnitt 7.2 ein Fazit über die erzielten Resultate gezogen.

Instanz	Durch-läufe	ØZeit s	Beste Lösung	Schwächste Lösung	Ø Lösung	Startpopulation	Mutations Methoden	Anzahl an Generationen	CR %
vrpnc1	10	49.09	524.61	533.65	526.16	Heuristik: Sweep; #: 300	Insertion: p 0.05, ab 0, bis 5000 Displacement: p 0.05, ab 0, bis 5000 Swap: p 0.05, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.05, ab 0, bis 5000	5000	90
vrpnc1	10	132.71	524.61	524.61	524.61	Heuristik: Sweep; #: 300	Insertion: p 0.05, ab 0, bis 5000 Displacement: p 0.05, ab 0, bis 5000 Swap: p 0.05, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.05, ab 0, bis 5000 λ-Interch. ($\lambda = 2$): p 0.01, ab 4600, bis 5000	5000	90
vrpnc2	10	189.28	851.23	885.75	863.62	Heuristik: RFCS; #: 470 Heuristik: Sweep; #: 470	Insertion: p 0.01, ab 0, bis 5000 Displacement: p 0.01, ab 0, bis 5000 Swap: p 0.01, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.02, ab 0, bis 5000	5000	90
vrpnc2	10	261.86	835.26	852.72	844.93	Heuristik: RFCS; #: 470 Heuristik: Sweep; #: 470	Insertion: p 0.01, ab 0, bis 5000 Displacement: p 0.01, ab 0, bis 5000 Swap: p 0.01, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.02, ab 0, bis 5000 λ-Interch. ($\lambda = 2$): p 0.002, ab 4750, bis 5000	5000	90

<table>
<tr><th>Instanz</th><th>Durch-
läufe</th><th>ØZeit
s</th><th>Beste
Lösung</th><th>Schwächste
Lösung</th><th>Ø
Lösung</th><th>Startpopulation</th><th>Mutations
Methoden</th><th>Anzahl an
Generationen</th><th>CR
%</th></tr>
<tr><td>vrpnc3</td><td>10</td><td>306.89</td><td>835.83</td><td>873.66</td><td>850.35</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>300</td></tr><tr><td>Sweep</td><td>750</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>5000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>5000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>5000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>5000</td></tr><tr><td>RAR</td><td>0.16</td><td>0</td><td>5000</td></tr></table></td><td>5000</td><td>90</td></tr>
<tr><td>vrpnc3</td><td>10</td><td>526.61</td><td>826.14</td><td>842.57</td><td>834.19</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>300</td></tr><tr><td>Sweep</td><td>750</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>5000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>5000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>5000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>5000</td></tr><tr><td>RAR</td><td>0.16</td><td>0</td><td>5000</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.003</td><td>4750</td><td>5000</td></tr></table></td><td>5000</td><td>90</td></tr>
<tr><td>vrpnc4</td><td>5</td><td>864.04</td><td>1047.14</td><td>1087.18</td><td>1064.26</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr><tr><td>Sweep</td><td>2000</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3700</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
</table>

<table>
<tr><th>Instanz</th><th>Durch-
läufe</th><th>ØZeit
s</th><th>Beste
Lösung</th><th>Schwächste
Lösung</th><th>Ø
Lösung</th><th>Startpopulation</th><th>Mutations
Methoden</th><th>Anzahl an
Generationen</th><th>CR
%</th></tr>
<tr><td>vrpnc4</td><td>5</td><td>2596.45</td><td>1031.1</td><td>1044.4</td><td>1035.23</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr><tr><td>Sweep</td><td>2000</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.002</td><td>3400</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.0015</td><td>3680</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3700</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
<tr><td>vrpnc5</td><td>3</td><td>1338.87</td><td>1349.36</td><td>1372.64</td><td>1357.77</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>1000</td></tr><tr><td>Sweep</td><td>2200</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.008</td><td>3725</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
<tr><td>vrpnc5</td><td>3</td><td>3995.54</td><td>1306.3</td><td>1320.71</td><td>1312.56</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>1000</td></tr><tr><td>Sweep</td><td>2200</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.0015</td><td>3400</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.001</td><td>3680</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.008</td><td>3725</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
</table>

Instanz	Durch-läufe	ØZeit s	Beste Lösung	Schwächste Lösung	Ø Lösung	Startpopulation	Mutations Methoden	Anzahl an Generationen	CR %
vrpnc6	10	51.81	556.68	560.24	559.11	Heuristik: Sweep; #: 300	Insertion: p 0.05, ab 0, bis 5000 Displacement: p 0.05, ab 0, bis 5000 Swap: p 0.05, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.05, ab 0, bis 5000	5000	90
vrpnc6	10	176.23	555.43	555.43	555.43	Heuristik: Sweep; #: 300	Insertion: p 0.05, ab 0, bis 5000 Displacement: p 0.05, ab 0, bis 5000 Swap: p 0.05, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.05, ab 0, bis 5000 λ-Interch. ($\lambda = 2$): p 0.01, ab 4600, bis 5000	5000	90
vrpnc7	10	270.42	917.07	954.18	938.19	Heuristik: RFCS; #: 470 Heuristik: Sweep; #: 470	Insertion: p 0.05, ab 0, bis 5000 Displacement: p 0.05, ab 0, bis 5000 Swap: p 0.05, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.05, ab 0, bis 5000	5000	90
vrpnc7	10	345.78	909.67	920.72	911.4	Heuristik: RFCS; #: 470 Heuristik: Sweep; #: 470	Insertion: p 0.05, ab 0, bis 5000 Displacement: p 0.05, ab 0, bis 5000 Swap: p 0.05, ab 0, bis 5000 Route: p 1.0, ab 0, bis 5000 RAR: p 0.05, ab 0, bis 5000 λ-Interch. ($\lambda = 2$): p 0.004, ab 4750, bis 5000	5000	90

<table>
<tr><th>Instanz</th><th>Durch-
läufe</th><th>ØZeit
s</th><th>Beste
Lösung</th><th>Schwächste
Lösung</th><th>Ø
Lösung</th><th>Startpopulation</th><th>Mutations
Methoden</th><th>Anzahl an
Generationen</th><th>CR
%</th></tr>
<tr><td>vrpnc8</td><td>10</td><td>323.42</td><td>874.33</td><td>899.68</td><td>883.61</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>200</td></tr><tr><td>Sweep</td><td>800</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.03</td><td>0</td><td>5000</td></tr><tr><td>Displacement</td><td>0.03</td><td>0</td><td>5000</td></tr><tr><td>Swap</td><td>0.03</td><td>0</td><td>5000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>5000</td></tr><tr><td>RAR</td><td>0.1</td><td>0</td><td>5000</td></tr><tr><td>Single Route Opt.</td><td>0.008</td><td>4965</td><td>5000</td></tr></table></td><td>5000</td><td>95</td></tr>
<tr><td>vrpnc8</td><td>10</td><td>996.98</td><td>865.94</td><td>879.99</td><td>872.67</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>200</td></tr><tr><td>Sweep</td><td>800</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.03</td><td>0</td><td>5000</td></tr><tr><td>Displacement</td><td>0.03</td><td>0</td><td>5000</td></tr><tr><td>Swap</td><td>0.03</td><td>0</td><td>5000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>5000</td></tr><tr><td>RAR</td><td>0.1</td><td>0</td><td>5000</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.003</td><td>4750</td><td>5000</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.005</td><td>4940</td><td>5000</td></tr><tr><td>Single Route Opt.</td><td>0.008</td><td>4965</td><td>5000</td></tr></table></td><td>5000</td><td>95</td></tr>
<tr><td>vrpnc9</td><td>5</td><td>1399.35</td><td>1200.52</td><td>1232.38</td><td>1215.61</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr><tr><td>Sweep</td><td>2000</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3700</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
</table>

<table>
<tr><th>Instanz</th><th>Durch-
läufe</th><th>ØZeit
s</th><th>Beste
Lösung</th><th>Schwächste
Lösung</th><th>Ø
Lösung</th><th>Startpopulation</th><th>Mutations
Methoden</th><th>Anzahl an
Generationen</th><th>CR
%</th></tr>
<tr><td>vrpnc9</td><td>5</td><td>5301.53</td><td>1168.52</td><td>1182.99</td><td>1171.68</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr><tr><td>Sweep</td><td>2000</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.002</td><td>3400</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.0015</td><td>3680</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3700</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
<tr><td>vrpnc10</td><td>3</td><td>1715.35</td><td>1456.81</td><td>1484.35</td><td>1469.18</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>1000</td></tr><tr><td>Sweep</td><td>2200</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.008</td><td>3700</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
<tr><td>vrpnc10</td><td>3</td><td>6473.2</td><td>1411.40</td><td>1420.69</td><td>1414.92</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>1000</td></tr><tr><td>Sweep</td><td>2200</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Displacement</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Swap</td><td>0.02</td><td>0</td><td>3800</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>3800</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.0015</td><td>3400</td><td>3800</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.001</td><td>3680</td><td>3800</td></tr><tr><td>Single Route Opt.</td><td>0.008</td><td>3700</td><td>3800</td></tr></table></td><td>3800</td><td>95</td></tr>
</table>

<table>
<tr><th>Instanz</th><th>Durch-
läufe</th><th>ØZeit
s</th><th>Beste
Lösung</th><th>Schwächste
Lösung</th><th>Ø
Lösung</th><th>Startpopulation</th><th>Mutations
Methoden</th><th>Anzahl an
Generationen</th><th>CR
%</th></tr>
<tr><td>vrpnc11</td><td>10</td><td>284.18</td><td>1044.74</td><td>1075.29</td><td>1056.46</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
<tr><td>vrpnc11</td><td>10</td><td>565.95</td><td>1042.12</td><td>1048.07</td><td>1044.38</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>4000</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.003</td><td>3750</td><td>4000</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.003</td><td>3940</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
<tr><td>vrpnc12</td><td>10</td><td>166.76</td><td>824.78</td><td>863.92</td><td>838.22</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>750</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.1</td><td>0</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.02</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
</table>

<table>
<tr><th>Instanz</th><th>Durch-
läufe</th><th>ØZeit
s</th><th>Beste
Lösung</th><th>Schwächste
Lösung</th><th>Ø
Lösung</th><th>Startpopulation</th><th>Mutations
Methoden</th><th>Anzahl an
Generationen</th><th>CR
%</th></tr>
<tr><td>vrpnc12</td><td>10</td><td>499.95</td><td>819.56</td><td>843.03</td><td>822.43</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>750</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.1</td><td>0</td><td>4000</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.005</td><td>3700</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.02</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
<tr><td>vrpnc13</td><td>10</td><td>376.1</td><td>1546.31</td><td>1560.16</td><td>1552.91</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>600</td></tr><tr><td>Sweep</td><td>300</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
<tr><td>vrpnc13</td><td>10</td><td>1263.55</td><td>1546.14</td><td>1550.98</td><td>1547.59</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>600</td></tr><tr><td>Sweep</td><td>300</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>4000</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.003</td><td>3750</td><td>4000</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.003</td><td>3940</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
</table>

<table>
<tr><th>Instanz</th><th>Durch-
läufe</th><th>ØZeit
s</th><th>Beste
Lösung</th><th>Schwächste
Lösung</th><th>Ø
Lösung</th><th>Startpopulation</th><th>Mutations
Methoden</th><th>Anzahl an
Generationen</th><th>CR
%</th></tr>
<tr><td>vrpnc14</td><td>10</td><td>190.6</td><td>866.37</td><td>890.73</td><td>875.96</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
<tr><td>vrpnc14</td><td>10</td><td>525.57</td><td>866.37</td><td>866.37</td><td>866.37</td><td><table><tr><td>Heuristik</td><td>#</td></tr><tr><td>RFCS</td><td>900</td></tr></table></td><td><table><tr><td>Name</td><td>p</td><td>ab</td><td>bis</td></tr><tr><td>Insertion</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Displacement</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Swap</td><td>0.01</td><td>0</td><td>4000</td></tr><tr><td>Route</td><td>1.0</td><td>0</td><td>4000</td></tr><tr><td>RAR</td><td>0.12</td><td>0</td><td>4000</td></tr><tr><td>λ-Interch. ($\lambda = 2$)</td><td>0.003</td><td>3750</td><td>4000</td></tr><tr><td>λ-Interch. ($\lambda = 3$)</td><td>0.003</td><td>3940</td><td>4000</td></tr><tr><td>Single Route Opt.</td><td>0.01</td><td>3900</td><td>4000</td></tr></table></td><td>4000</td><td>95</td></tr>
</table>

Der eingesetzte Route Mutationsoperator wendet folgende Mutationsverfahren auf die einzelnen Touren an:

- Displacement
- Inversion
- Swap

Dabei wird jede einzelne Tour des betrachteten Tourenplans mit $p = 0.001$ von einem dieser Operatoren verändert.
Bei der Single Route Optimierung werden alle Touren innerhalb eines Tourenplans mit der 2-Opt und der Insertion Search Heuristik verbessert.
Folgende Tabelle soll die jeweils besten erzielten eigenen Lösungen mit den bisher bekannten besten Lösungen vergleichen, und deren prozentuelle Abweichung veranschaulichen.

Instanz	n	Beste bekannte Lösung	Eigene Lösung	% Abweichung zur besten Lösung
vrpnc1	50	524.61	524.61	0%
vrpnc2	75	835.26	835.26	0%
vrpnc3	100	826.14	826.14	0%
vrpnc4	150	1028.42	1031.10	0.26%
vrpnc5	199	1291.29	1306.30	1.16%
vrpnc6	50	555.43	555.43	0%
vrpnc7	75	909.68	909.68	0%
vrpnc8	100	865.94	865.94	0%
vrpnc9	150	1162.55	1168.52	0.51%
vrpnc10	199	1395.85	1411.40	1.11%
vrpnc11	120	1042.11	1042.12	0.001%
vrpnc12	100	819.56	819.56	0%
vrpnc13	120	1541.14	1546.14	0.32%
vrpnc14	100	866.37	866.37	0%

7.2 Fazit der Testresultate

Anhand der Ergebnisse kann deutlich gezeigt werden, dass der GA mit λ-*Interchange* Hybridisierung hervorragende Resultate erzielt. Bei 8 von 14 Instanzen konnten die bisher besten bekannten Lösungen erreicht werden, bei denen es sich vermutlich um die optimalen Lösungen handelt. Keiner der aus Kapitel 4 vorgestellten Algorithmen erreicht auch nur annähernd eine derartige Lösungsqualität.
Die Input Parameter des GA, die bei den Testläufen verwendet wurden, wurden aufgrund der Menge und Anordnung der Kunden angenommen, um möglichst gute Tourenpläne zu

erzeugen. Aufgrund der großen Anzahl an möglichen Parametern ist es jedoch schwer, optimale Einstellungen für die jeweiligen Instanzen zu finden, die die besten Zeit-Leistungs-Verhältnisse garantieren. Es sei angemerkt, dass die erzielte Lösungsqualität höchstwahrscheinlich auch mit geringerer Rechenzeit durch effizientere Parametrisierung erreicht werden kann.

8 Praktische Anwendung der Software

Im Rahmen eines Universitätsprojektes wurde die Software zur Tourenplanoptimierung eines österreichischen Unternehmens eingesetzt. Aufgrund einer vorhandenen Geheimhaltungserklärung dürfen jedoch weder Name noch Branche des Unternehmens in dieser Arbeit erwähnt werden. In diesem abschließenden Kapitel wird ein grober Überblick über die vorhandene Aufgabenstellung und deren Lösung gegeben.
Abbildung 8.1 zeigt den gesamten Graphen auf dem die Tourenplanoptimierung erfolgt.

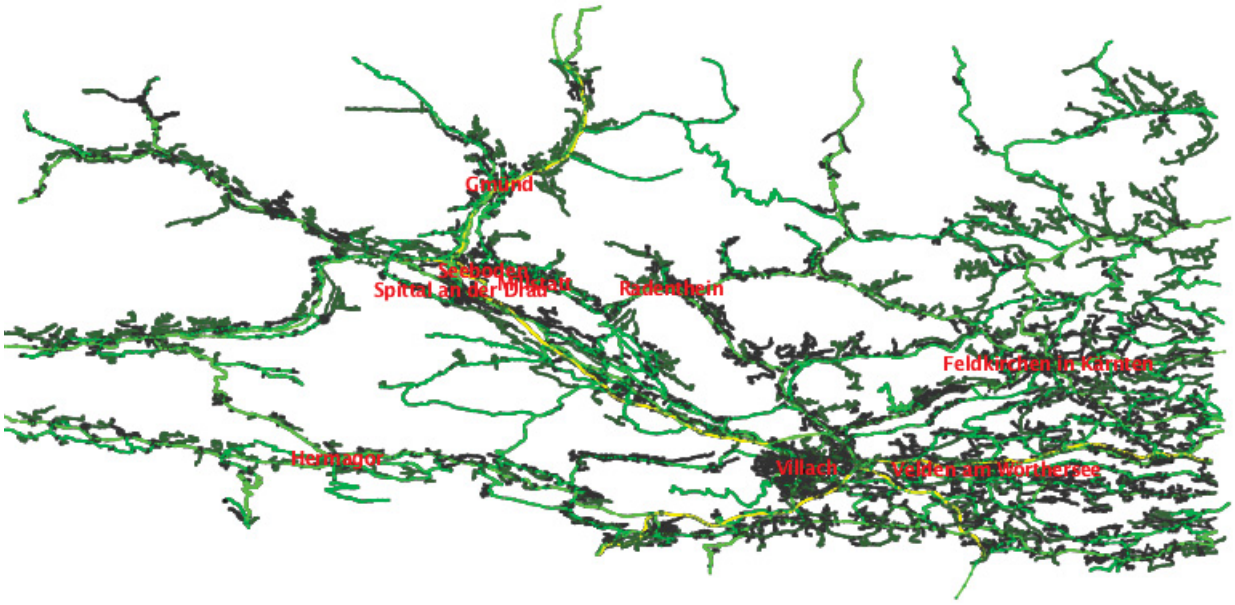


Abbildung 8.1: Graph des Optimierungsgebiets der praktischen Aufgabe

Die zu beliefernden Kunden, inklusive der zu beliefernden Mengen, sind für einen langen Zeitraum bereits vorgegeben, und ändern sich bei den unterschiedlichen Auslieferungszyklen nicht. Deshalb kann ein vorhandener Tourenplan für einen längeren Zeitraum verwendet werden. Bestimmte Kunden des Unternehmens werden an geraden Kalenderwochen beliefert, während andere Kunden an ungeraden Kalenderwochen bedient werden. Für die Belieferung der Kunden steht nur ein einziges Fahrzeug zur Verfügung, welches jede Woche jeweils Montag, Dienstag, Donnerstag und Freitag eine Tour absolviert. Bei dem vorhandenem VRP welches eine Mischung aus dem CVRP und dem DCVRP darstellt, wird die Kundenmenge einer gesamten Kalenderwoche betrachtet, und anhand dieser durch Lösung des VRP ein Tourenplan erstellt. Die einzelnen Touren dieses Tourenplans repräsentieren jeweils eine bestimmte Wochentag-

Tour des Fahrzeuges. Da jedoch Kunden, die an ungeraden Kalenderwochen beliefert werden, nicht mit Kunden, die an geraden Kalenderwochen beliefert werden, vermischt werden dürfen, ergeben sich daher zwei voneinander unabhängige VRPs.

8.1 Ausgangssituation der Problemstellung

Zunächst soll anhand von Abbildung 8.2 veranschaulicht werden, wie die Touren bisher disponiert wurden. Hierbei wurde die Abbildung mithilfe von protokollierten GPS Daten erstellt. Die beiden Gebäude markieren den Standort des Hauptquartiers. Die unterschiedlichen Farben sind folgendermaßen definiert:

- Rot → Montagtour
- Gold → Dienstagtour
- Blau → Donnerstagtour
- Magenta → Freitagtour

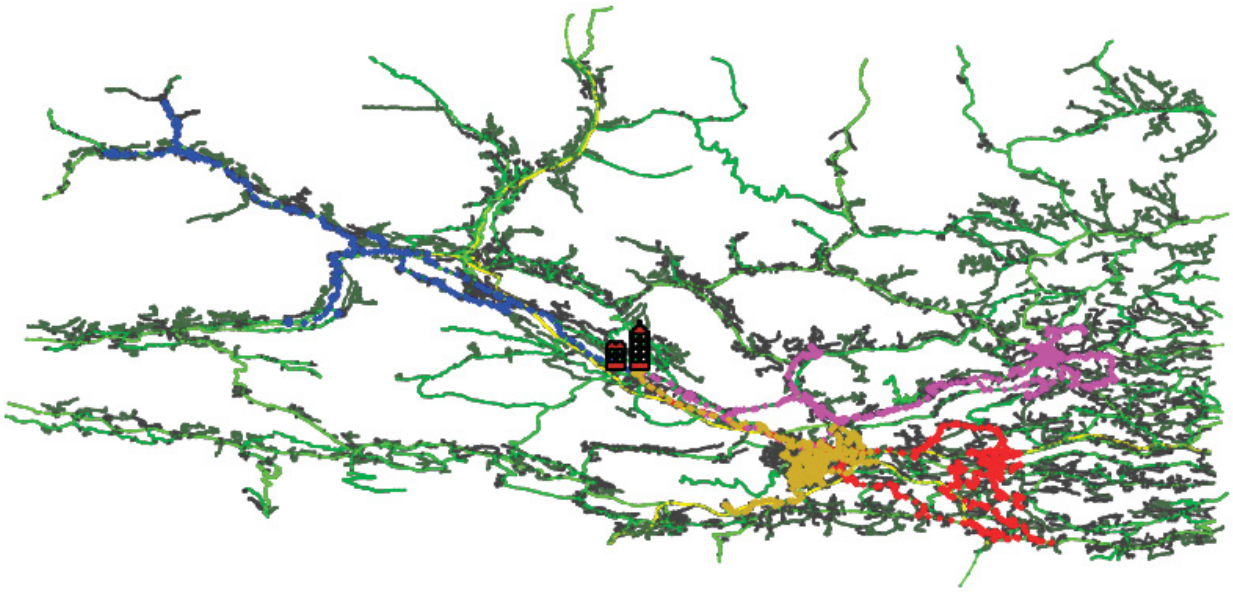
Unter Berücksichtigung dieser Touren werden durch die Software folgende Ausgangsdaten errechnet, welche auch mit den Unternehmensunterlagen übereinstimmen:

Gerade Kalenderwoche

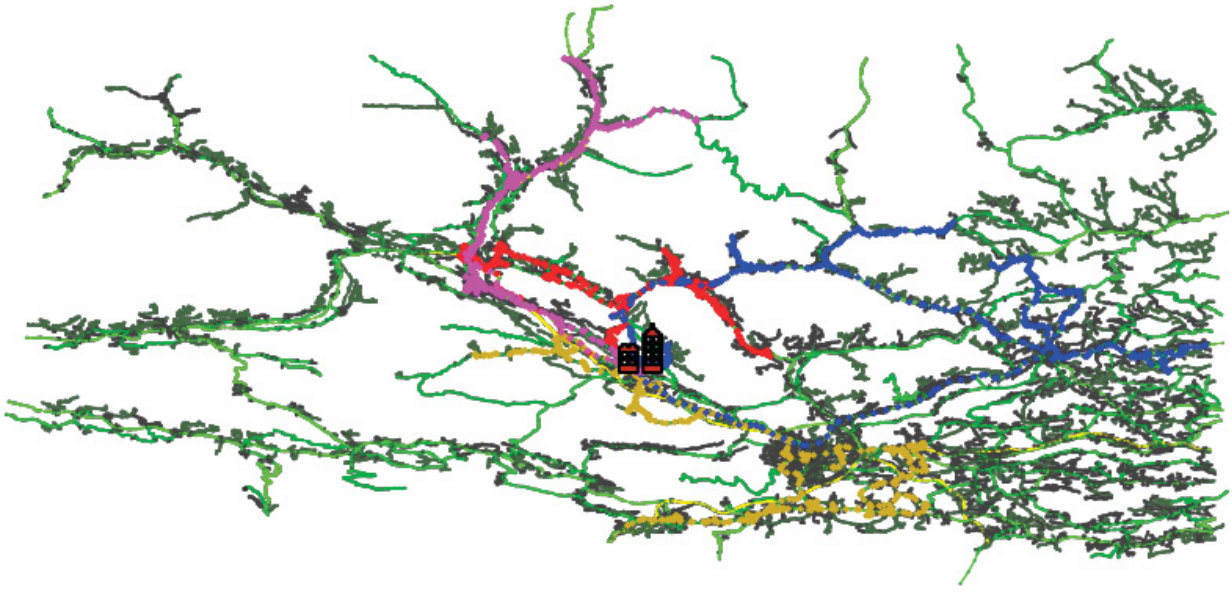
Tour	**Benötigte Fahrzeit**	**Zurückgelegte Kilometer**	Kunden	Verwendetes Volumen $[dm^3]$
Montagtour	07:43:24	189.3km	130	14922
Dienstagtour	07:54:08	167.85km	141	18315
Donnerstagtour	08:40:27	219.27km	137	16778.25
Freitagtour	05:20:32	176.41km	74	9191.25
Summe	29:38:31	752.83km	482	59206.5

Ungerade Kalenderwoche

Tour	**Benötigte Fahrzeit**	**Zurückgelegte Kilometer**	Kunden	Verwendetes Volumen $[dm^3]$
Montagtour	08:23:12	166.35km	163	14150.25
Dienstagtour	12:32:42	277.41km	216	20801.25
Donnerstagtour	07:23:57	213.45km	110	9921.38
Freitagtour	07:27:35	222.65km	103	12672
Summe	35:47:26	879.86km	592	57544.88
Gesamtsumme	65:25:57	1632.69km	1074	116751.38



((a)) GPS Touren der geraden Kalenderwoche

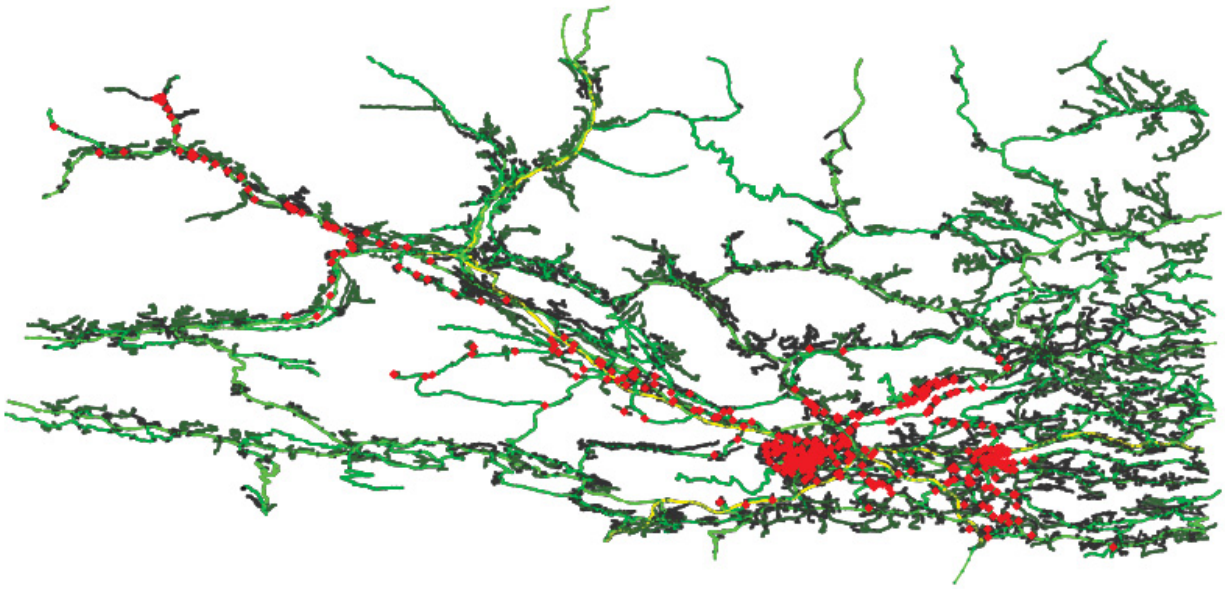


((b)) GPS Touren der ungeraden Kalenderwoche

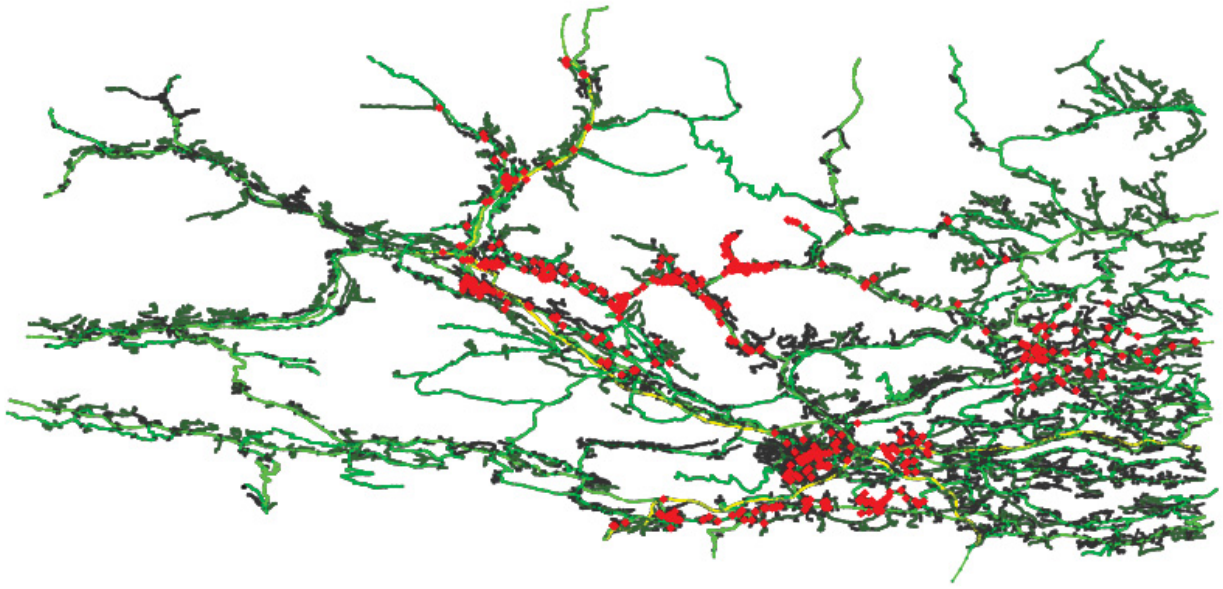
Abbildung 8.2: GPS Touren

8.2 Lösung der Problemstellung

Es sollen nun beide VRPs gelöst werden, sodass sich eine Einsparung, in Bezug auf die benötigte Fahrzeit bzw. zurückgelegten Kilometer, zur vorhandenen Ist Situation ergibt. Zunächst war es durch Absprache mit dem Unternehmen möglich, einige Kunden zwischen den beiden Kalenderwochen zu vertauschen, um so eine bessere Ausgangssituation zu ermöglichen. Abbildung 8.3 veranschaulicht die zu lösende Ausgangssituation des VRP, wobei die roten Knoten die zu beliefernden Kunden markieren.



((a)) Vehicle Routing Problem der geraden Kalenderwoche



((b)) Vehicle Routing Problem der ungeraden Kalenderwoche

Abbildung 8.3: Ausgangssituation des Vehicle Routing Problems

Bei der Lösung der VRPs wurde als Restriktion eine maximale Ladekapazität von $24000dm^3$ angenommen, und für die maximale Fahrzeit 12 Stunden vorgeschrieben. Als Lösungsverfahren wurde hierbei zunächst das Clark and Wright Savings-Verfahren eingesetzt, um einen vorläufigen Tourenplan zu erstellen. Anschließend wurden die daraus gebildeten Touren einzeln durch eine Reihe von TSP Verfahren optimiert. Zu diesem Zweck wurde die Random Insertion Heuristik gefolgt von der 2-Opt- und der Insertion Search-Heuristik angewendet. Multi-Route Verbesserungsheuristiken und der GA konnten aufgrund der starken geographischen Restriktionen und der doch untypischen hohen Anzahl an Kunden pro Tour keine besseren Ergebnisse erzielen.
In der folgenden Tabelle werden die Ergebnisse der berechneten Tourenpläne aufgelistet, welche des Weiteren in Abbildung 8.4 am OSM Graph veranschaulicht werden.

Gerade Kalenderwoche

Tour	**Benötigte Fahrzeit**	**Zurückgelegte Kilometer**	Kunden	Verwendetes Volumen $[dm^3]$
Rot	09:23:24	205.76km	169	20124.00
Blau	10:43:18	226.38km	195	23294.25
Magenta	07:27:10	231.98km	102	12264.75
Summe	27:33:52	664.12km	466	55683.00

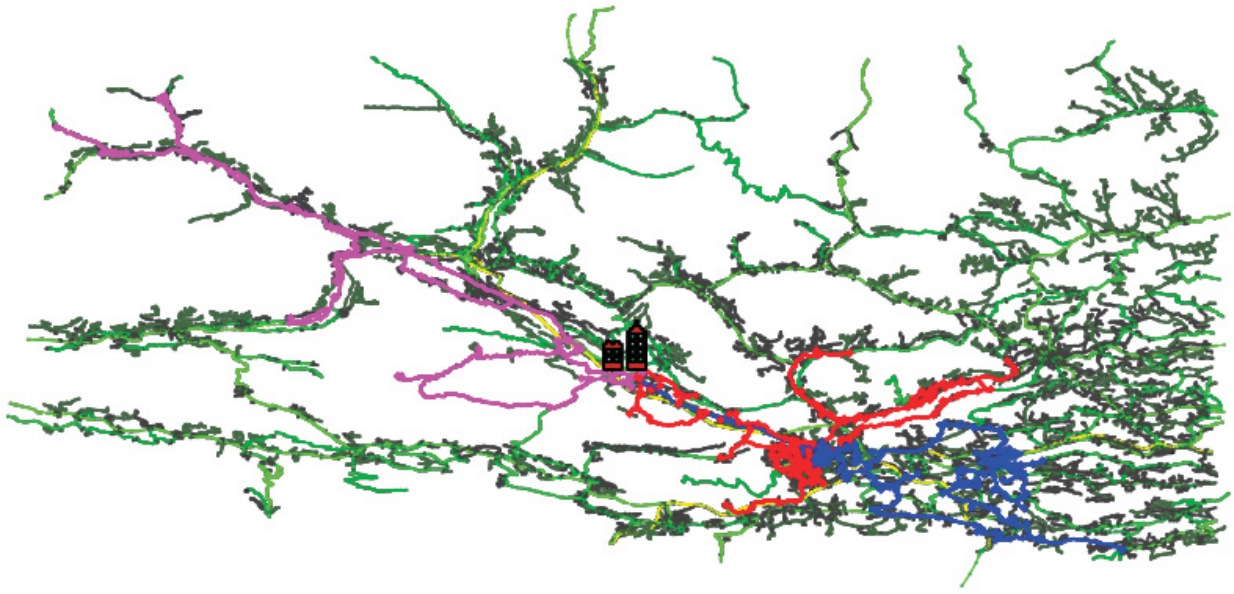
Ungerade Kalenderwoche

Tour	**Benötigte Fahrzeit**	**Zurückgelegte Kilometer**	Kunden	Verwendetes Volumen $[dm^3]$
Rot	08:05:41	155.32km	157	13601.25
Blau	10:29:29	271.61km	162	17782.88
Gold	09:39:31	201.03km	179	16463.25
Magenta	06:38:14	174.50km	110	13221.00
Summe	34:52:55	802.46km	608	61068.38
Gesamtsumme	62:26:47	1466.58km	1074	116751.38

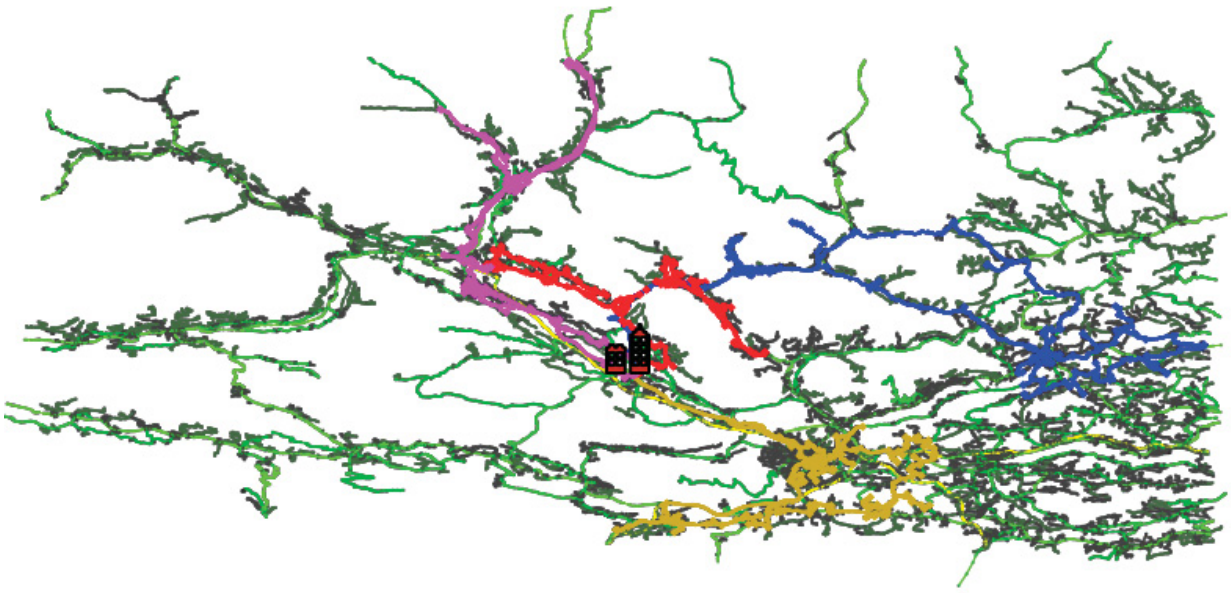
Anhand dieser Daten lässt sich ablesen, dass eine Verbesserung zur Ist Situation erzielt werden konnte. Folgende Tabelle stellt die Kennzahlen der alten und neuen Tourenplänen direkt gegenüber:

Vergleich der Tourenpläne

Tourenplan	**Benötigte Fahrzeit**	**Zurückgelegte Kilometer**
Alter Tourenplan	65:25:57	1632.69km
Neuer Tourenplan	62:26:47	1466.58km
Einsparung	02:59:10	166.11km



((a)) Lösung der geraden Kalenderwoche



((b)) Lösung der ungeraden Kalenderwoche

Abbildung 8.4: Lösung des Vehicle Routing Problems

Trotz der großen geographischen Einschränkungen, die aufgrund der Gebirge gegeben waren, konnte in diesem Projekt dennoch eine akzeptable Einsparung erzielt werden. Dabei muss besonders hervorgehoben werden, dass eine komplette Wochentag-Tour des Fahrzeuges eingespart werden konnte, wodurch dieses zum Vorteil des Unternehmens für alternative Aufgaben an diesem Tag eingesetzt werden kann.

Literaturverzeichnis

Buch- und Journalquellen

[1] Otman Abdoun und Jaafar Abouchabaka. „A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem“. In: *International Journal of Computer Applications* 31.11 (2011)

[2] Barun Chandra, Howard Karloff und Craig Tovey. „New results on the old k-opt algorithm for the traveling salesman problem“. In: *SIAM Journal on Computing* 28.6 (1999)

[3] Rui Chibante. *Simulated Annealing, Theory with Applications*. Hrsg. von Rui Chibante. Global Optimization. Sciyo, 08/2010. ISBN: 978-953-307-134-3. http://www.intechopen.com/books/show/title/simulated-annealing--theory-with-applications

[4] T H Cormen. *Algorithmen: eine Einführung*. Oldenbourg, 2013. ISBN: 9783486582628. http://books.google.at/books?id=curIpfL84boC

[5] II Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis. „An Analysis of Several Heuristics for the Traveling Salesman Problem“. In: *SIAM Journal on Computing* (1977), S. 563–581

[6] G.B Dantzig und J.H Ramser. „The Truck Dispatching Problem“. In: *Management Science 6* (1959), S. 80–91

[7] Kusum Deep und Hadush Mebrahtu. „Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem“. In: *International Journal of Combinatorial Optimization Problems and Informatics*. 2.3 (2011), S. 1–23

[8] Reinhard Diestel. *Graphentheorie*. Springer-Lehrbuch Masterclass. Springer, 2006. ISBN: 9783540213918.

https://books.google.at/books?id=imMyNFYQussC

[9] Marco Dorigo. „Optimization, learning and natural algorithms“. Diss. Politecnico di Milano Italy, 1992

[10] B. A. Foster und D. M. Ryan. „An Integer Programming Approach to the Vehicle Scheduling Problem“. In: *Operational Research Quarterly* 27.2 (1976), S. 367–384

[11] T. J. Gaskell. „Bases for Vehicle Fleet Scheduling“. In: *Operational Research Quarterly* 18.3 (1967), S. 281–295

[12] I Gerdes, F Klawonn und R Kruse. *Evolutionäre Algorithmen: Genetische Algorithmen — Strategien und Optimierungsverfahren — Beispielanwendungen*. Computational Intelligence. Vieweg+Teubner Verlag, 2013. ISBN: 9783322868398.
https://books.google.at/books?id=TBd6rWMLCqcC

[13] B. Gillett und L. Miller. „A Heuristic for the Vehicle Dispatching Problem. Operations Research“. In: *Operations Research* 22 (1974), S. 340–349

[14] Fred Glover. „Tabu Search—Part I“. In: *ORSA Journal on Computing* 1.3 (1989), S. 190–206. DOI: 10.1287/ijoc.1.3.190.
http://dx.doi.org/10.1287/ijoc.1.3.190

[15] K Gutenschwager. *Online-Dispositionsprobleme in der Lagerlogistik: Modellierung - Lösungsansätze - praktische Umsetzung*. Wirtschaftswissenschaftliche Beitr{ä}ge. Physica-Verlag HD, 2013. ISBN: 9783642574948.
https://books.google.at/books?id=_aWcBgAAQBAJ

[16] Keld Helsgaun. „An effective implementation of the Lin–Kernighan traveling salesman heuristic“. In: *European Journal of Operational Research* 126.1 (2000), S. 106–130

[17] Richard M. Karp. *Complexity of Computer Computations: Proceedings*. TheIBM Research Symposia Series. Basic Books, 1972, S. 85–103. ISBN: 9780306307072.
http://books.google.at/books?id=XINQAAAAMAAJ

[18] B Korte und J Vygen. *Kombinatorische Optimierung: Theorie und Algorithmen*. Springer-Lehrbuch Masterclass. Springer Berlin Heidelberg, 2012. ISBN: 9783642254017. http://books.google.at/books?id=Rj0lBAAAQBAJ

[19] Krunoslav Puljic und Robert Manger. „Comparison of eight evolutionary crossover operators for the vehicle routing problem“. In: *MATHEMATICAL COMMUNICATIONS* (2013), S. 359–375

[20] Shen Lin. „Computer solutions of the traveling salesman problem“. In: *Bell System Technical Journal* 44.10 (1965), S. 2245 –2269

[21] M Mitchell. *An Introduction to Genetic Algorithms*. A Bradford book. Bradford Books, 1998. ISBN: 9780262631853. https://books.google.ca/books?id=0eznlz0TF-IC

[22] Heinz Mühlenbein. „How Genetic Algorithms Really Work: Mutation and Hillclimbing“. In: *Parallel Problem Solving from Nature*. Bd. 2. 1992, S. 15–26

[23] C Ohrt. *Tourenplanung im Straßengüterverkehr*. Betriebswirtschaftliche Forschung zur Unternehmensf{ü}hrung. Gabler Verlag, 2008. ISBN: 9783834997111. https://books.google.at/books?id=HdAhBAAAQBAJ

[24] Ibrahim Hassan Osman. „Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem“. In: *Annals of Operations Research* 41 (1993), S. 421–451

[25] Francisco B Pereira u. a. „GVR: a new genetic representation for the vehicle routing problem“. In: *Problem, Proceedings of the 13th Irish Conference on Artificial Intelligence and Cognitive Science*. Springer-Verlag, 2002, S. 95–102

[26] Christian Prins. „A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem“. In: *COMPUTERS AND OPERATIONS RESEARCH* 31 (2001)

[27] H.N. Psaraftis, P.M. and Thompson. „Cyclic transfer algorithms for multi-vehicle routing and scheduling problems“. In: *Operations Research* 41 (1993), S. 935–946

[28] Marc Reimann, Karl Doerner und Richard F Hartl. „D-Ants: Savings Based Ants divide and conquer the vehicle routing problem“. In: *Computers & Operations Research* 31.4 (2004), S. 563–591. ISSN: 0305-0548. DOI: http://dx.doi.org/10.1016/S0305-0548(03)00014-5.
http://www.sciencedirect.com/science/article/pii/S0305054803000145

[29] Julia Rieck. *Tourenplanung mittelständischer Speditionsunternehmen*. 1. Aufl. Gabler Edition Wissenschaft. Wiesbaden: Gabler, 2008. ISBN: 978-3-8349-1398-2.
http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA\&SRT=YOP\&IKT=1016\&TRM=ppn+581048881\&sourceid=fbw_bibsonomy

[30] M Rimscha. *Algorithmen kompakt und verständlich: Lösungsstrategien am Computer*. Springer Fachmedien Wiesbaden, 2014. ISBN: 9783658056186.
https://books.google.at/books?id=nb0hBAAAQBAJ

[31] F Rothlauf. *Design of Modern Heuristics: Principles and Application*. Natural Computing Series. Springer, 2011. ISBN: 9783540729624.
https://books.google.at/books?id=rqEI_u6PEToC

[32] D. M. Ryan, C. Hjorrin und F. Glover. „Extensions of the Petal Method for Vehicle Routeing“. In: *The Journal of the Operational Research Society* 44.3 (1993), S. 289–296

[33] B. W. Kernighan S. Lin. „An Effective Heuristic Algorithm for the Traveling-Salesman Problem“. In: *Operations Research* 21.2 (1973), S. 498 –516

[34] Gerhard Schrimpf u. a. „Record Breaking Optimization Results Using the Ruin and Recreate Principle“. In: *Journal of Computational Physics* 159 (2000), S. 139–171

[35] L Suhl und T Mellouli. *Optimierungssysteme: Modelle, Verfahren, Software, Anwendungen*. Springer-Lehrbuch. Springer Berlin Heidelberg, 2013. ISBN: 9783642389375.
http://books.google.at/books?id=GV0NAAAAQBAJ

[36] E Taillard. „Parallel iterative search methods for vehicle routing problems“. In: *Networks* 23.8 (1993), S. 661–673

[37] P Tittmann. *Graphentheorie: eine anwendungsorientierte Einführung ; mit zahlreichen Beispielen und 80 Aufgaben*. Mathematik-Studienhilfen. Fachbuchverl. Leipzig im Carl-Hanser-Verlag, 2003. ISBN: 9783446223431.
http://books.google.at/books?id=EV6qGBCuSe4C

[38] P Toth und D Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. MOS-SIAM Series on Optimization. Society for Industrial und Applied Mathematics, 2014. ISBN: 9781611973587.
https://books.google.at/books?id=VUzUBQAAQBAJ

[39] Paolo Toth und Daniele Vigo. *The vehicle routing problem*. Siam, 2001

[40] R Vahrenkamp und D C Mattfeld. *Logistiknetzwerke: Modelle für Standortwahl und Tourenplanung*. Springer Fachmedien Wiesbaden, 2014. ISBN: 9783834969125.
http://books.google.at/books?id=_US4AQAAQBAJ

[41] A. Van Breedam. *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and timerelated constraints*. Antwerp: University of Antwerp, 1994

[42] W Wenger. *Multikriterielle Tourenplanung*. Gabler Research : Produktion und Logistik. Gabler Verlag, 2010. ISBN: 9783834922571.
http://books.google.at/books?id=rPyP5K5H5FIC

[43] M Wessler und H Röpcke. *Graphen und Netzwerktheorie: Grundlagen - Methoden - Anwendungen*. Carl Hanser Verlag GmbH & Company KG, 2014. ISBN: 9783446441842.
http://books.google.at/books?id=BFstBQAAQBAJ