Karlsruhe Institute of Technology

Algorithm Engineering in der Praxis am Fallbeispiel eines VRP

Diplomarbeit
von

Hanno Kersting

An der Fakultät für Informatik
Institut für Theoretische Informatik (ITI)

Erstgutachter:	Prof. Dr. rer. nat. D. Wagner
Zweitgutachter:	Prof. Dr. rer. nat. P. Sanders
Betreuende Mitarbeiter:	Dipl.-Math. R. Bauer
	Dipl.-Inform. M. Krug

Bearbeitungszeit: 16. September 2009 – 15. März 2010

Inhaltsverzeichnis

1. Einleitung und Problemstellung

1.1 Einleitung

Das Tourenplanungsproblem der Informatik findet viele Anwendungen in der realen Welt. Wenn eine optimale Reihenfolge für eine Menge von Aufträgen gefunden werden und eine Zuordnung von Fahrzeugen zu Touren erfolgen soll, kann man meist von einem Tourenplanungsproblem sprechen. Ganz klar lässt sich diese Problemstellung bei großen Logistikunternehmen erkennen. Hier spielt es eine große Rolle, dass die Ergebnisse von Planungsalgorithmen möglichst nah an die optimale Lösung heranreichen, da Verbesserungen zu enormen Kosteneinsparungen führen können.

Diesem Umstand ist es sicherlich auch zu verdanken, dass Tourenplanungsprobleme bereits relativ bekannt und gut erforscht sind. Dennoch besteht immer noch eine gewisse Diskrepanz zwischen der akademischen Sicht auf dieses Problem und den Anforderungen, die aus der realen Welt entstammen. Rein akademische Algorithmen sind meist nur bedingt einsetzbar oder erfordern einen hohen Anpassungsaufwand.

Im Rahmen meiner Diplomarbeit habe ich mit Algorithmen für Tourenplanungsprobleme gearbeitet, die auch tatsächlich zur Lösung von Realweltproblemen verwendet werden. Im Verlauf dieser Arbeit möchte ich einen Einblick in die Besonderheiten und Unterschiede geben, die sich stellen, wenn kommerziell verwendbare Algorithmen entwickelt werden sollen, und welche Kompromisse und Entscheidungen bei solch praxisnaher Entwicklung getroffen werden müssen.

1.2 Die Aufgabenstellung

Diese Diplomarbeit wurde bei dem Unternehmen *Planung Transport Verkehr AG*, abkürzend PTV genannt, mit Hauptsitz in Karlsruhe geschrieben, das sich intensiv mit der Lösung von Tourenplanungsproblemen beschäftigt. Aufgabenstellung dieser Diplomarbeit war es, bestehende Ansätze zu analysieren, zu verbessern und auf spezielle Probleme anzupassen. Dabei wurden folgende drei Punkte wurden genauer betrachtet:

1. Der *Savingsalgorithmus* als ein Konstruktionsverfahren, das erste gültige Lösungen liefert.
2. Die *Granulare Tabusuche* als Verbesserungsverfahren, das bereits bestehende, gültige Lösungen weiter verbessern kann.
3. Mögliche Qualitätssteigerung durch die Nutzung variabler Fahrzeiten, die von der Tageszeit abhängen.

Der Savingsalgorithmus

Der Savingsalgorithmus eignet sich gut für real anfallende Probleme, da kaum Anpassungen bei wechselnden Nebenbedingungen oder Restriktionen vorgenommen werden müssen und er gute Lösungen bei geringem zeitlichen Rechenaufwand liefert. Daher wird er in vielen kommerziellen Programmen der PTV verwendet. Gewünscht war eine Analyse und Verbesserung des bisher bestehenden Algorithmus. Im Vordergrund stand eine qualitative Verbesserung der Lösungen, ein zeitlicher Mehraufwand war - in gewissen Grenzen - tolerabel. Alle neu entworfenen Varianten des Savingsalgorithmus sollten ausführlich auf ihre Ergebnisqualität und Robustheit getestet werden. Dazu gehörte ein Feintuning der vorhandenen Parameter, ein Vergleich der Ergebnisqualität der Varianten untereinander und ein Vergleich mit den Lösungen des bereits implementierten Algorithmus der PTV.

Die Granulare Tabussuche

Bei der PTV existiert ein Framework zur Verbesserung bereits bestehender Lösungen von Tourenplanungsproblemen. Dementsprechend galt es, das Verhalten der neuen Savingsvarianten auch in Bezug auf die Nachoptimierung zu betrachten und mit dem bereits bestehenden Algorithmus der PTV zu vergleichen. Hier war vor allem von Interesse,

- ob sich ein Konstruktionsverfahren besser für eine anschließende Nachoptimierung eignet,
- ob nach der Nachoptimierung aller Ergebnisse der Konstruktionsverfahren manche Lösungen besonders hervorstechen,
- ob die Lösungen des Konstruktionsverfahrens mit den besten Ergebnissen vor der Nachoptimierung auch nach deren Ausführung zu den besten Lösungen gehören.

Bei internen Tests der PTV wurde festgestellt, dass es eine Reihe von Probleminstanzen gibt, die sich mit der aktuell existierenden Nachoptimierung nicht oder nur kaum verbessern lassen. Hier sollte eine Erweiterung der bereits implementieren Granularen Tabusuche vorgenommen werden, um auf diese Fälle besser reagieren zu können. Hierbei war eine obere Schranke für die zur Verfügung stehende Laufzeit vorgegeben, die beachtet werden musste. Ausgehend von der neu integrierten Erweiterung sollten die Veränderungen der Ergebnisqualität der Nachoptimierung untersucht werden. Zu vergleichen waren die Ergebnisse der Nachoptimierung *ohne* die Erweiterung mit den Ergebnissen, die *mit* der Erweiterung erzielt wurden, jeweils unter Beachtung der gleichen zeitlichen Schranken.

Zeitabhängige Fahrzeiten

Die PTV hat sich auch mit dem Lösen von zeitabhängigen VRPs beschäftigt. Hier hat sich gezeigt, dass speziell für dieses Problem entwickelte Algorithmen nicht unbedingt bessere Ergebnisse liefern, als der bereits vorhandene Savingsalgorithmus. Dies erscheint verwunderlich, da der Savingsalgorithmus die zusätzliche Information der Zeitabhängigkeit nicht sinnvoll nutzen kann. Diese Beobachtung sollte weiter untersucht und eine Erklärung für dieses Verhalten gefunden werden. Dazu war eine genaue Analyse der zeitabhängigen Fahrzeiten notwendig. Auf diesem Weg gewonnene Erkenntnisse sind jedoch nicht beliebig zu verallgemeinern, da alle Untersuchungen nur auf einem sehr begrenzten Datensatz vorgenommen werden konnten und somit allenfalls Stichprobencharakter haben. Auch die vorliegenden zeitabhängigen Daten waren noch nicht endgültig und sollten in Zukunft aktualisiert werden.

1.3 Überblick über verwandte Probleme

Zum ersten Mal erwähnt wurde das *Vehicle Routing Problem* (VRP) von Dantzig und Ramser (1959) unter der Bezeichnung *Truck Dispatching Problem*. Als Beispiel wurde die Treibstoffauslieferung von einem zentralen Depot aus mit einer Flotte von Tankwagen genannt. Kerninhalt eines jeden VRPs ist, dass eine Zuordnung einer Menge von Fahrzeugen auf eine Menge von Stopps geschickt erfolgen soll, die dann in möglichst guter Reihenfolge besucht werden. Bis heute wurde ein Vielzahl von Verallgemeinerungen und auch Spezialisierungen des VRPs untersucht. Hier soll ein kurzer Überblick über verwandte Probleme gegeben werden.

Traveling Salesman Problem

Das *Traveling Salesman Problem* (TSP) lässt sich als Spezialfall des VRPs mit nur einem Fahrzeug auffassen (siehe Bellmore und Nemhauser (1968)). Da das TSP ein NP-vollständiges Problem ist, gehört somit auch das VRP zu der Klasse der NP-vollständigen Probleme.

Capacitated VRP

Das *Capacitated VRP* (CVRP) ist eine Verallgemeinerung des VRP. Jeder Stopp muss nicht nur besucht, sondern auch mit einer gewissen Menge beliefert werden. Jedes Fahrzeug verfügt aber nur über eine gewisse Kapazität, die nicht überschritten werden darf. Als Einführung bietet sich Toth und Vigo (2001) an.

VRP with Time Windows

Eine weitere Verallgemeinerung ist das *VRP with Time Windows* (VRPTW). Bei diesem Problem müssen die Stopps in individuellen zeitlichen Intervallen besucht werden. Trifft ein Fahrzeug zu früh ein, muss es bis zum Beginn des Intervalls warten. Hier ist der Startzeitpunk jeder Tour Teil der Lösung. Auch hier sei für weitere Informationen Toth und Vigo (2001) empfohlen.

Time Dependent VRP

Bei allen bisherigen Varianten des VRPs wurde davon ausgegangen, dass die Fahrzeit von einem Stopp zum nächsten ein konstanter Wert ist. Eine weitere Verallgemeinerung ist, statt diesem Wert, der sich als konstante Funktion auffassen lässt, auch andere Funktionen zu erlauben. So lassen sich tageszeitbedingte Schwankungen der Fahrzeit modellieren. Eine sehr gute Einführung, die ihre Untersuchungen anhand des Berliner Straßennetzes anstellt, bieten B. Fleischmann und Gnutzmann (2004). Einen eher theoretischen Blickpunkt vermitteln Malandraki und Daskin (1992).

Weitere Verallgemeinerungen

Neben den bisher erwähnten, bekannten Varianten des VRP existieren noch eine Vielzahl weiterer Verallgemeinerungen. Hier sei aber lediglich noch auf folgende Arbeiten verwiesen, die es sich als Ziel gesetzt haben, eine Vielzahl dieser Probleme zu sammeln und zu katalogisieren: Toth und Vigo (2001), B. Golden und Wasil (2008) und Domschke (1997).

1.4 Etablierte Lösungsverfahren

In den vorherigen Abschnitten wurde eine Reihe von unterschiedlichen Problemstellungen kurz vorgestellt. Diese unterscheiden sich an einigen Stellen, haben aber doch so viele Gemeinsamkeiten, dass viele Lösungsstrategien auf alle diese Probleme angewandt werden können. So können Verfahren, die schon zur Lösung des TSP eingesetzt wurden, auch sehr gewinnbringend bei der Lösung von den meisten Abarten des VRP eingesetzt werden (siehe Domschke (1997)).

Hier wird jetzt eine Auswahl der bekannten und schon gut untersuchten Lösungsstrategien vorgestellt, die sich für eine ganze Reihe von Tourenplanungsproblemen eignen.

1.4.1 Exakte Lösungsverfahren

Im Rahmen dieser Diplomarbeit wurden nur NP-vollständige Probleme betrachtet. Ein Beweis wird dazu nicht erbracht, aber auf die ausführliche Untersuchung von König (1995) verwiesen. Daher wurde es für wenig erfolgversprechend gehalten, sich ausführlich mit exakten Lösungsansätzen zu beschäftigen.

Diesbezüglich wird lediglich auf ein sehr bekanntes Verfahren, das Erstellen und Lösen von ganzzahligen linearen Programmen (ILPs), verwiesen. Zu diesem Thema bieten N. Azi und Potvin (2006), N. Azi und Potvin (2008) als auch Domschke (1997) gute Einführungen mit Verweisen auf weiterführende Literatur.

1.4.2 Heuristiken

Es gibt viele Möglichkeiten, Heuristiken für das VRP zu entwerfen, aber ein bestimmtes Verfahren ist in der Literatur sehr häufig anzutreffen. Es ist sehr universell einsetzbar, einfach strukturiert und wurde an einer Vielzahl von Problemstellungen erfolgreich eingesetzt. Bei diesem Verfahren wird zuerst mit einer *Eröffnungsheuristik* eine Menge gültiger Lösungen für die Probleminstanz gesucht. Diese Lösungen werden daraufhin mit einer oder mehreren unterschiedlichen Verbesserungsheuristiken optimiert. Diese sind meist lokale Suchverfahren, so dass die Gefahr besteht, nur lokale Optima zu finden. Um dem zu entgehen, kann die Menge der gültigen Startlösungen entsprechend verteilt und groß gewählt werden, oder es kommen Metaheuristiken zum Einsatz, die es ermöglichen, ein lokales Optimum wieder zu verlassen.

Besonders zu empfehlen sind in diesem Zusammenhang die beiden Übersichtsartikel von Bräysy und Gendreau (2005a) und Bräysy und Gendreau (2005b).

1.4.2.1 Eröffnungsheuristiken

Eröffnungsheuristiken liefern eine erste gültige Lösung für die gegebene Probleminstanz. Diese Lösung wird gemeinhin von nachfolgenden Vefahren noch verbessert, wobei deren Ergebnisqualität meist sehr stark von der ersten Lösung abhängt.

Die Savings-Heuristik

Eine der ersten Standardheuristiken zur Erstellung von Initiallösungen ist die *Savings-Heuristik* von Clarke und Wright aus dem Jahre 1964 (siehe Clarke und Wright (1964)). Bis heute wurden viele Modifikationen und Spezialisierungen für dieses Verfahren entwickelt, aber die ursprüngliche Funktionsweise ist relativ einfach.

Bei der Savings-Heuristik beginnt man mit der maximalen Anzahl möglicher Touren (für jeden Stopp eine eigene Tour) und versucht diese zu verkleinern. Dazu werden fortlaufend die beiden Touren aneinandergehängt, die zu der größten Kosteneinsparung für den neuen, so gewonnenen Tourenplan führen. Der Algorithmus endet, wenn es nicht weiter möglich ist, die Tourenanzahl zu verringern, ohne dass der Tourenplan ungültig wird.

Sequentieller Savingsalgorithmus

Umgekehrt ist es auch möglich, ohne Touren zu beginnen und diese sequentiell aufzubauen. Es wird ein, vom Depot möglichst weit entfernt liegender, Knoten ausgewählt, der noch in keiner Tour verplant ist. Mit ihm wird eine neue Tour begonnen und schrittweise die Knoten mit jeweils minimalem Abstand zum zuletzt hinzugefügten Knoten angehängt. Gibt es keine solchen Knoten mehr, die gültig eingefügt werden können, wird eine neue Tour begonnen (siehe Yellow (1970)).

Der Sweep-Algorithmus

Der *Sweep-Algorithmus* bildet ein weiteres heuristisches Verfahren zum Finden einer Initiallösung. Vorgestellt wurde er 1974 von Gillet und Miller (siehe Domschke (1997) und G. Laporte und Semet (2000)). Voraussetzung für seine Anwendung ist, dass jedem Knoten Koordinaten zugeordnet werden können, die den relativen Abstand zum Depot angeben. Arbeitet man auf Grundlage von Kartenmaterial, stehen diese Werte zur Verfügung. Ausgehend von einem beliebigen Knoten als Start werden allen weiteren Knoten Polarwinkel in Bezug auf das Depot zugeordnet. Wenn diese Werte gegeben sind, verläuft der Algorithmus folgenderweise:

Als erster Knoten einer neuen Tour wird der Knoten mit dem niedrigsten Polarwinkel gewählt, der noch in keiner anderen Tour vorkommt. Sukzessive, den ansteigenden Polarwinkeln nach, werden zu dieser Tour weitere Knoten hinzugefügt. Lässt sich ein Knoten nicht mehr hinzufügen, weil dadurch eine Nebenbedingung verletzt wird, so wird mit einer neuen Tour begonnen. Dies wird wiederholt, bis alle Knoten zu genau einer Tour gehören.

Für weitere Eröffnungsheuristiken, Tests und Vergleiche untereinander empfiehlt sich Bräysy und Gendreau (2005a).

1.4.2.2 Verbesserungsheuristiken

Liegt bereits ein gültiger Tourenplan vor, der durch eine Eröffnungsheuristik gewonnen wurde, so kann dieser meist noch verbessert werden. Typischerweise geschieht dies durch eine weitere Art von Heuristiken. Bewährt als Verbesserungsheuristiken haben sich Algorithmen, die eine lokale Suche durchführen (siehe Abschnitt 3.3.1).

Lokale Suchalgorithmen unterscheiden sich in der Wahl der entsprechenden Nachbarschaftsfunktion. Wählt man die Nachbarschaft zu klein, kann die Wahrscheinlichkeit sehr hoch sein, dass man nur lokale Optima findet. Wählt man sie dagegen zu groß, im Extremfall alle Lösungen, dann degeneriert der Algorithmus zu einer erschöpfenden Suche mit exponentieller Laufzeit. Einen Überblick über erfolgreiche Verbesserungsheuristiken findet man in in dem Artikel von Bräysy und Gendreau (2005a).

1.4.2.3 Metaheuristiken

Wird eine Lokale Suche verwendet und lässt sich nach einigen Iterationen keine bessere Lösung in der aktuellen Nachbarschaft finden, endet die Nachoptimierung. Es wurde ein lokales Optimum gefunden. Es ist aber gut möglich, dass außerhalb der zuletzt betrachteten Nachbarschaft noch bessere Lösungen als die des lokalen Optimums existieren. Solche Lösungen können von einer Lokalen Suche nicht unterschieden werden.

Um dieses Problem zumindest abzuschwächen, wurde eine Reihe von Metaheuristiken entwickelt, die das Entkommen aus lokalen Optima zum Ziel haben. Kurz vorgestellt werden hier lediglich die *Tabusuche* (siehe Kapitel 3.3.1) und das *Simulated Annealing* Verfahren. Eine weitaus gründlichere Einführung in Metaheuristiken bieten Aarts und Lenstra (1997). Einen sehr guten Überblick über Metaheuristiken, die speziell auf das VRP angepasst sind, bieten Bräysy und Gendreau (2005b).

Tabusuche

Die Kernidee der Tabusuche ist, dass nach dem Finden eines lokalen Optimums der Verbesserungsvorgang nicht abgebrochen wird, sondern stattdessen weitergesucht wird. Nach dem Finden eines lokalen Optimums befinden sich in der aktuellen Nachbarschaft keine besseren Lösungen, es müssen also auch Verschlechterungen zugelassen werden. Damit nach einer verschlechternden Lösung nicht im nächsten Zug wieder zu dem ursprünglichen lokalen Optimum zurückgesprungen wird, muss eine Auswahl von Lösungen verboten, also tabu gesetzt werden, um ein Kreiseln zu verhindern. Dies können beispielsweise alle bereits betrachteten Lösungen sein. Dies geschieht mit Hilfe einer Tabuliste. Diese Tabuliste muss regelmäßig aktualisiert werden und schränkt so die Auswahl verfügbarer Lösungen ein.

Ausnahmen können unter bestimmten Bedingungen, den sogenannten Aspirationskriterien, auftreten. Ein übliches Aspirationskriterium ist, dass eine Lösung gewählt werden darf, obwohl sie in der Tabuliste vermerkt ist, wenn ihre Qualität die Qualität der bisher besten Lösung übertrifft.

Es gibt mehrere Strategien, welche Züge oder Lösungen man auf die Tabuliste setzt. Ein erster Ansatz wäre, alle Lösungen, die bereits durchlaufen wurden, auf die Liste zu sitzen. So wird verhindert, dass der Algorithmus in einen Zyklus gerät. Diese Möglichkeit ist aber wenig effektiv, da es sehr lange dauern kann, bis sich der Algorithmus Lösungen zuwendet, die nicht mehr in der unmittelbaren Umgebung des letzten Optimums liegen. Auch kann die Liste sehr lang werden, was ihre Verwaltung und die Prüfoperationen verlangsamt.

Ein Ansatz, dem zu entgehen, ist, nicht mehr Lösungen in der Tabuliste zu speichern, sondern nur noch Eigenschaften von Lösungen, die verboten werden. Oder umgekehrt können auch Eigenschaften gespeichert werden, die bei den weiteren Lösungen vorhanden sein müssen. Meist werden diese Eigenschaften für eine begrenzte Anzahl von Iterationsschritten in der Tabuliste gespeichert, oder die Liste selbst hat nur eine begrenzte Größe, so dass alte Einträge mit der Zeit verdrängt werden.

Simulated Annealing

Das Simulated Annealing Verfahren ist ein weiterer Ansatz, um die Ergebnisqualität von Lokalen Suchen zu erhöhen. In jedem Schritt der Lokalen Suche wird nicht mehr zwangsläufig die beste Lösung aus der Nachbarschaft ausgewählt, sondern die Auswahl findet mehr oder weniger zufällig statt.

Zu Beginn des Verfahrens ist die Auswahl der nächsten Lösung nahezu ziellos. Im weiteren Verlauf werden jedoch die Lösungen mit einer höheren Wahrscheinlichkeit gewählt, die zu einer stärkeren Verbesserung der Lösungsqualität führen. Über die Zeit werden also vermehrt bessere Lösungen ausgewählt. Es gibt mehrere Möglichkeiten, wie die Wahrscheinlichkeiten für die Wahl einer neuen Lösung aus der Nachbarschaft sinnvoll bestimmt werden können. Für genaue Verfahren wird das Buch von Aarts und Lenstra (1997) empfohlen.

Anschaulich wird so zuerst ein günstig erscheinender Bereich in der Menge der Lösungen gefunden, da man viele verschiedenen Lösungen „besuchen“ kann, die recht unterschiedliche Eigenschaften aufweisen. Erst gegen Ende der Verbesserung wird ein bestimmter Bereich der Lösungen genauer durchsucht und von diesem Bereich das Lokale (und vielleicht auch Globale) Optimum gefunden.

Nachempfunden ist dieses Verfahren physikalischen Abkühlungsprozessen. Dabei werden Stoffe auf eine maximale Temperatur erhitzt, bis sie schmelzen. Dann wird die Temperatur sehr langsam gesenkt. Dabei organisieren sich die Teilchen des erhitzten Stoffes so, dass die Energie des Stoffes nach dem Erstarren minimal ist. Die Teilchen bewegen sich zu Beginn des Verfahrens zufällig, dennoch liegt nach dem Abkühlungsprozess eine gleichmäßige Gitterstruktur vor (siehe Aarts und Lenstra (1997)).

2. Eine Diplomarbeit bei der PTV-AG

Die PTV-AG ist ein Softwareunternehmen für Reise-, Transport- und Verkehrsplanung. Es arbeiten - verteilt auf allen Kontinenten - rund 700 Mitarbeiter, wobei der Hauptsitz in Karlsruhe das Entwicklungs- und Innovationszentrum darstellt (siehe PTVAG (2009)).

Eine Diplomarbeit bei solch einem Unternehmen unterscheidet sich in mancherlei Hinsicht von einer Betreuung, die ausschließlich von der Universität ausgeht. Da diese Unterschiede starken Einfluss auf den Verlauf und das Ergebnis haben, sollen sie hier näher geschildert werden.

2.1 Unterschiede in der Zielsetzung

Aufgabenstellung der Diplomarbeit war es, kommerzielle Algorithmen zu entwickeln und zu erweitern. Im Fokus standen immer Realweltprobleme und von Kunden gewünschte Nebenbedingungen und Restriktionen. Dies ist ein großer Unterschied zu einer rein akademischen Betrachtung des gleichen Problems.

Man ist nicht so frei in der Bearbeitung des Problems, da alle Entscheidungen im Hinblick auf eine kommerzielle Verwertbarkeit des Produkts getroffen werden müssen. Besonders starken Einfluss übt dies auf die implementierten Algorithmen aus. Hier ist es nicht möglich, für jeden Kunden einen eigenen Algorithmus zu entwickeln, sondern die vorliegenden Algorithmen müssen mit wenig Aufwand an ein möglichst großes Feld von möglichen Problemstellungen angepasst werden können. Folgende Punkte haben für alle Algorithmen zu gelten:

- Es muss leicht möglich sein, eine beliebige Auswahl von Nebenbedingungen zu betrachten.
- Neue Nebenbedingungen müssen leicht integriert werden können.
- Der interne Ablauf des Algorithmus muss weitestgehend entkoppelt sein von allen möglichen Nebenbedingungen.

Dies wurde durch eine logische Trennung des Algorithmus von den Nebenbedingungen erreicht. Die Nebenbedingungen stehen lediglich als Black-Box-Funktionen zur Verfügung.

2.2 Unterschiede in der Arbeitsweise

Obwohl sehr viel wissenschaftliche Literatur zur Lösung des VRPs besteht, war das verfügbare Wissen nur sehr eingeschränkt nutzbar. Zu groß ist der Unterschied zwischen den

Realweltanforderungen und den zwingend erforderlichen Verallgemeinerungen und Abstraktionen, die in den wissenschaftlichen Artikeln vorgenommen werden. Dies betrifft im besonderen Maße Annahmen über die zur Verfügung stehende Laufzeit und die Beschränkung auf eine lediglich kleine Auswahl von Nebenbedingungen. Bestehende Ansätze können oft nur als Inspiration oder Lösungsansatz und nicht als ausgereiftes Konzept, das ohne Änderungen übernommen werden kann, Verwendung finden.

Eine Diplomarbeit bei einem Unternehmen führt auch dazu, dass bereits vorhandene Software genutzt werden konnte, aber auch musste. Die implementierten Algorithmen waren nicht ohne das kommerzielle Programm *Intertour* (siehe Kapitel 2.4) nutzbar, das das Einlesen der Daten und die Visualisierung der Ergebnisse übernahm. So konnte Zeit gespart werden, da keine Implementierung dieser Funktionalitäten nötig war. Auch die Prüffunktionen standen als Black-Box-Funktion zur Verfügung und mussten nicht neu implementiert werden. Diese Umstände machten jedoch eine Einarbeitungsphase notwendig und führten zu Komplikationen bei den abschließenden Auswertungen (siehe Kapitel 2.3).

2.3 Unterschiede bei der Analyse

Ein wichtiger Bestandteil der Diplomarbeit waren Tests und Analysen der neu entwickelten und implementierten Algorithmen. Optimale Ergebnisse für die einzelnen Probleminstanzen waren nicht bekannt und konnten auch nicht berechnet berechnet werden, aber es stand ein bereits implementiertes Verfahren der PTV zur Lösung von Tourenplanungsproblemen zur Verfügung. Nach Angaben der PTV liefert dieses Verfahren gute Ergebnisse. Hier konnte ein sinnvoller Vergleich der Lösungen der neuen Algorithmen mit den Ergebnissen dieses Verfahrens durchgeführt werden und somit qualitative Aussagen über die neuen Algorithmen getroffen werden.

Leider konnten im Rahmen dieser Diplomarbeit nicht alle angedachten Tests durchgeführt werden. Dies lag teilweise an den verwendeten hochsensiblen Kundendaten, die nicht kopiert werden durften, um die Infrastruktur der Universität zu nutzen. Aber auch mehr Rechenkapazität hätte dieses Problem nicht grundlegend beseitigt, da zum Testen vorhandene Programme genutzt werden mussten, so dass dies nur bedingt skriptgesteuert möglich war. Viele Tests mussten „von Hand“ angestoßen werden, die Ausgaben der Testläufe mussten erst nachbearbeitet werden, bevor sie sinnvoll genutzt werden konnten. Auch war es nur mit viel Aufwand möglich, Tests zu wiederholen oder zu modifizieren, wenn die Ergebnisse anders als erwartet ausfielen.

2.4 PTV Intertour

PTV Intertour ist ein kommerzielles Programm für operationale Tourenplanung. Mit ihm werden, unter Berücksichtigung von Planungsrestriktionen, Tourenpläne ermittelt. Es stellt eine grafische Benutzeroberfläche zur Verfügung, so dass man die Planungsergebnisse direkt sehen und leicht den Überblick behalten kann (siehe PTV-AG (2007)). Auch das Einlesen der Probleminstanzen und die Aufbereitung der Daten in eine für die Algorithmen nutzbare Form geschieht durch Intertour.

Intertour war ein für diese Diplomarbeit sehr wichtiges Programm, das das Testen und Ausführen der entwickelten Algorithmen erst ermöglichte. Da es ein kommerzielles Programm für den Kundengebrauch ist, eignet es sich nur bedingt für ausführliche Tests der verwendeten Algorithmen. Es ist nur rudimentär möglich, skriptgesteuert zu arbeiten und flexibel zwischen verschiedenen Probleminstanzen zu wechseln.

In Abbildung 2.1 ist der typische Aufbau bei der Planung zu erkennen. Auf der linken Seite ist eine Karte eingeblendet, die alle geplanten Touren darstellt. Auf der rechten Seite

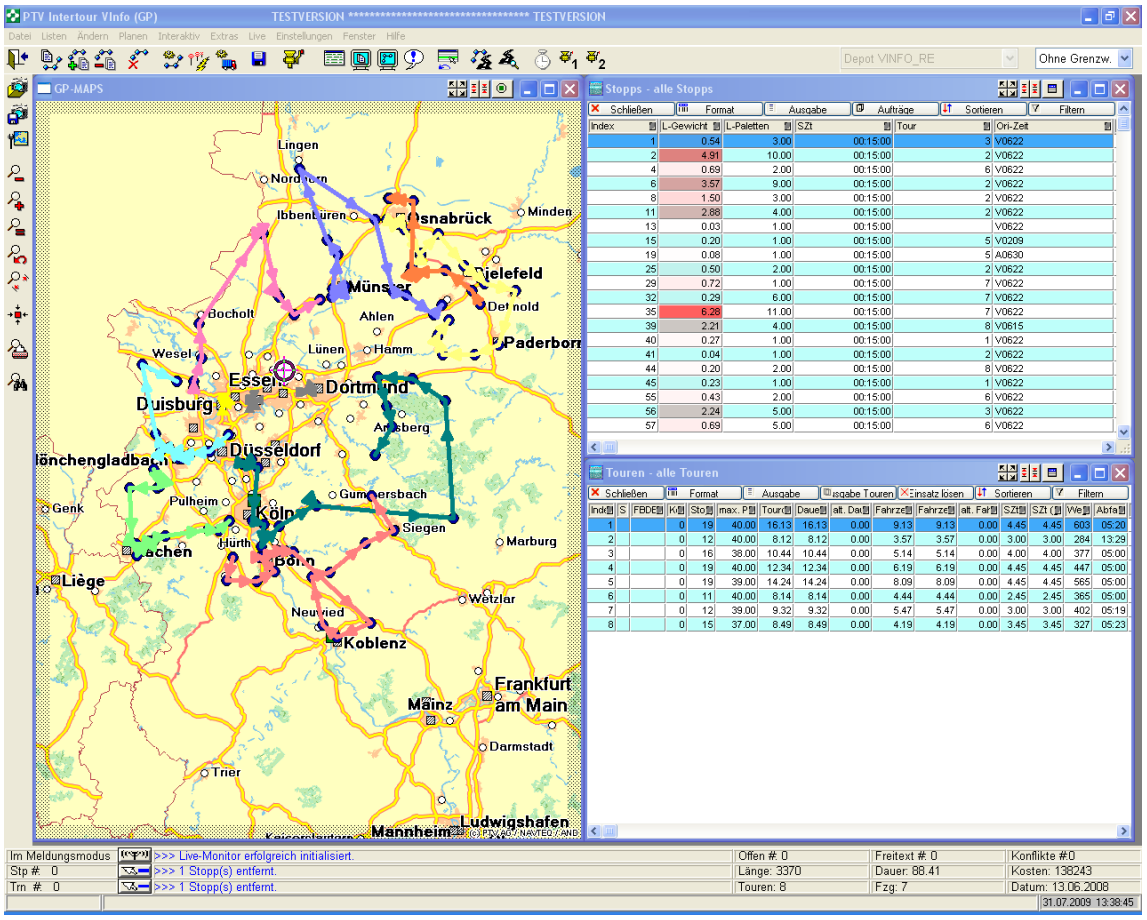


Abbildung 2.1: Die grafische Oberfläche von Intertour.

können benutzerdefiniert eine Reihe von Tabellen angezeigt werden. Üblicherweise sind dies detaillierte Informationen zu allen Stopps und den geplanten Routen. Weiter unten sind wichtige Angaben zu dem aktuellen Tourenplan zu erkennen. Dazu gehören Kosten, Zeit, Fahrzeuge und Anzahl der Routen.

Über eine grafische Benutzeroberfläche war es direkt möglich, die Tourenplanung anzustoßen und Parameter für die Algorithmen zu verändern (siehe Abbildung 2.2). Dieser Teil war nur für den internen Gebrauch bestimmt. Kunden werden lediglich auf eine sehr begrenzte Zahl von Algorithmen und Parameterkombinationen Zugriff haben, die sich aus den internen Tests ergeben haben.

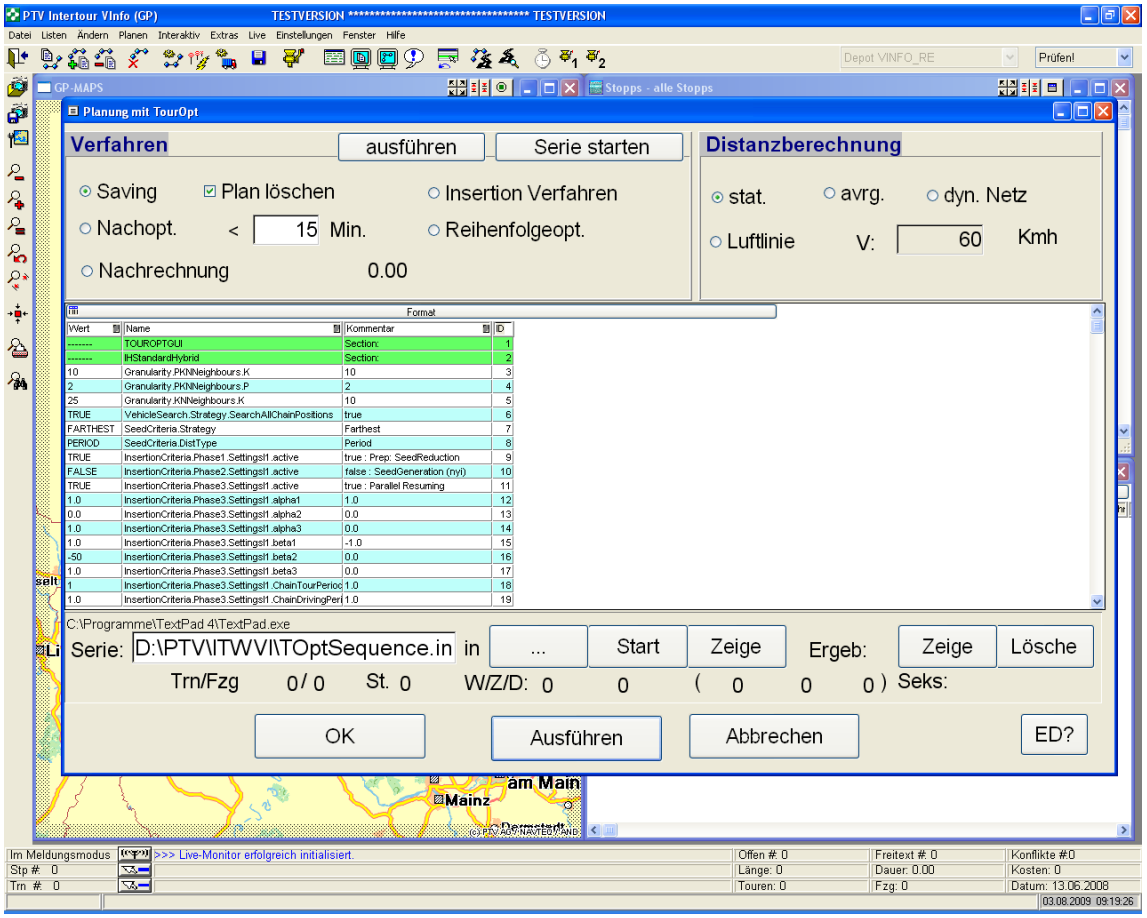


Abbildung 2.2: Die Oberfläche zur Steuerung der Planungsalgorithmen.

3. Lösungsstrategien

3.1 Grundlagen und Begriffsklärung

Die folgenden Definitionen bilden die Grundlage für alle folgenden Kapitel der Diplomarbeit.

Definition 1 (Graph) *Ein* ungerichteter Graph *G ist ein Tupel (V, E), wobei V eine endliche Menge von* Knoten *und $E \subseteq \{\{u, v\} \mid u, v \in V\}$ eine Menge von Paaren von Knoten ist, genannt* Kanten*. Die Menge $\{u, v\}$ bezeichnet eine ungerichtete Kante zwischen $u, v \in V$.*
Im gerichteten *Fall ist $E \subseteq V \times V$ eine Menge von gerichteten Paaren. Das Paar (u, v) bezeichnet eine gerichtete Kante von u nach v.*

Definition 2 (Rundtour) *Eine* Rundtour *in einem Graphen $G = (V, E)$ ist eine Folge $(v_1, v_2, \ldots v_k)$ mit $v_1, v_2, \ldots v_k \in V$, $v_n \neq v_m$ für $n \neq m$, $1 \leq n, m < k$ und $v_1 = v_k$. Zusätzlich muss bei ungerichteten Graphen G die Kante $\{v_i, v_{i+1}\}$ in E liegen für $1 \leq i < k$ und bei gerichteten Graphen die Kante (v_i, v_{i+1}). Der Knoten v_1 und v_k wird als* Depot *der Rundtour bezeichnet. Für eine Rundtour r bezeichne der Wert k die* Stopplänge *der Rundtour, abgekürzt mit $l(r)$. Für einen Knoten v und eine Rundtour t bedeutet $v \in t$, dass ein v_i mit $1 \leq i \leq k$ existiert und $v = v_i$. Abkürzend wird statt Rundtour auch der Begriff* Tour *gebraucht.*

Definition 3 (Tourenplan) *Ein* Tourenplan *zu einem Graphen $G = (V, E)$ und einem Knoten $v_{depot} \in V$ ist eine Menge T von Rundtouren in G und es gilt:*

- ***Einheitliches Depot:** Das Depot von allen $t \in T$ ist v_{depot}.*
- ***Disjunktheit der anderen Knoten:** Für $t_1, t_2 \in T$, $v_1 \in t_1$ und $v_2 \in t_2$ gilt $v_1 \neq v_2$, wenn $t_1 \neq t_2$, $v_1 \neq v_{depot}$ und $v_2 \neq v_{depot}$.*

Definition 4 (Vehicle Routing Problem)

- *Eine* Kostenfunktion *für Tourenpläne ist eine Funktion $K : A \to \mathbb{R}$, wobei A die Menge aller Tourenpläne ist.*
- *Eine* Prüffunktion *für Tourenpläne ist eine Funktion $D : A \to \{0, 1\}$, wobei A die Menge aller Tourenpläne ist. Ein Tourenplan T wird als* gültig *bezeichnet, wenn $D(T) = 1$.*

- *Zu einem Graphen $G = (V, E)$ wird $c : E \to \mathbb{R}_+$* Kantengewichtsfunktion *genannt. Sie repräsentiert den zeitlichen oder räumlichen Abstand zwischen den Knoten.*
- *Gegeben ist ein vollständiger, ungerichteter Graph $G = (V, E)$, eine Kantengewichtsfunktion c, ein $v_{depot} \in V$, eine Prüffunktion D und eine Kostenfunktion K.*
- ***Vehicle Routing Problem**: Gesucht ist ein gültiger Tourenplan T, mit $B(T)$ minimal.*
- *Wird im weiteren Verlauf von einem VRP gesprochen, gehören dazu auch die Funktionen c, D, K, der Graph $G = (V, E)$ und der Knoten $v_{depot} \in V$ mit den hier gewählten Bezeichnern.*

3.2 Der Savingsalgorithmus

Eine der ersten Standardheuristiken zur Erstellung von Initiallösungen ist die *Savings-Heuristik* von Clarke und Wright aus dem Jahre 1964 (siehe Clarke und Wright (1964)). Darauf aufbauend wurden weitere Verfahren entwickelt, die eine gültige Startlösung für ein gegebenes VRP generieren können.

3.2.1 Definitionen und Bemerkungen

Definition 5 (Stichtour) *Eine* Stichtour *in einem Graphen $G = (V, E)$ ist eine Rundtour der Stopplänge 3.*

Feststellung 1 *Zu einem gegebenen Graphen $G = (V, E)$ und einem $v_{depot} \in V$ hat jede Stichtour t die Form $t = (v_{depot}, v, v_{depot})$ mit $v \in V$. Die Stichtour t ist also eindeutig über v definiert.*

Definition 6 (Randknoten) *Zu einem gegebenen Tourenplan T zu einem Graphen $G = (V, E)$ und einem $v_{depot} \in V$ ist ein Knoten $v \in V$ genau dann* Randknoten*, wenn eine Rundtour $t = (v_{depot}, v_2, \ldots v_{k-1}, v_{depot})$ existiert mit $v = v_2$ oder $v = v_{k-1}$. Der Knoten v wird dann Randknoten der Tour t genannt. Für $v = v_2$, wird v* Startrandknoten *und für $v = v_{k-1}$, wird v* Endrandknoten *genannt.*

Definition 7 (Aneinanderhängen) *Gegeben seien die Touren $t_1 = (v_{depot}, v_2, \ldots v_{k-1}, v_{depot})$ und $t_2 = (v_{depot}, p_2, \ldots p_{l-1}, v_{depot})$. Wir schreiben:*

- $t_1 \blacktriangleright\blacktriangleright t_2 = (v_{depot}, v_2, \ldots v_{k-1}, p_2, \ldots p_{l-1}, v_{depot})$
- $t_1 \blacktriangleleft\blacktriangleright t_2 = (v_{depot}, v_{k-1}, \ldots v_2, p_2, \ldots p_{l-1}, v_{depot})$
- $t_1 \blacktriangleright\blacktriangleleft t_2 = (v_{depot}, v_2, \ldots v_{k-1}, p_{l-1}, \ldots p_2, v_{depot})$
- $t_1 \blacktriangleleft\blacktriangleleft t_2 = (v_{depot}, v_{k-1}, \ldots v_2, p_{l-1}, \ldots p_2, v_{depot})$

Dieses Bilden einer neuen Tour aus den zwei vorhandenen Touren t_1 und t_2 wird als Aneinanderhängen *bezeichnet.*

Definition 8 (Saving) *Gegeben sei eine Instanz des VRP und ein zugehöriger Tourenplan. Sei v Randknoten einer Tour t_1 und u Randknoten einer anderen Tour t_2. Dann sei*

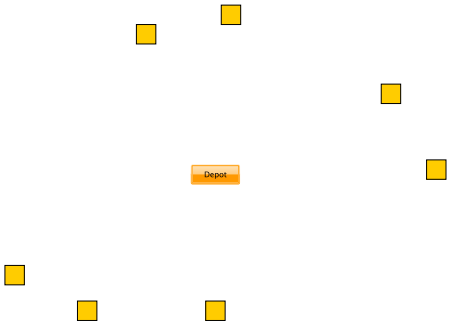
$$\text{saving}_{uv} = c(v, v_{depot}) + c(v_{depot}, u) - c(v, u)$$

3.2.2 Der Savingsalgorithmus von Clarke und Wright

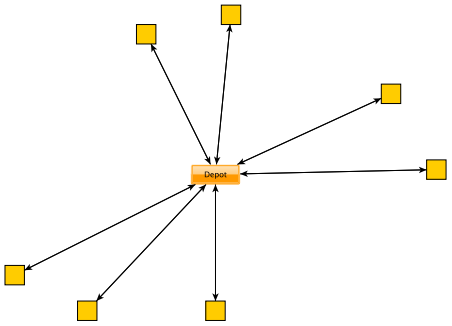
Die Kernidee des Algorithmus besteht darin, dass mit einer maximalen Menge von Stichtouren begonnen wird, die sukzessiv miteinander verschmolzen werden, um die Kosten des Tourenplans zu reduzieren. Es werden immer die Touren miteinander verschmolzen, die zu der maximalen Kantengewichtsreduktion führen. Anders ausgedrückt, werden jeweils zwei Randknoten u und v gesucht mit saving_{uv} maximal, wobei der Tourenplan auch nach dem Aneinanderhängen der beiden zugehörigen Touren gültig bleibt. Diese Vorgehensweise sei in der Bilderserie 3.2 noch einmal dargestellt.

Hierzu sei noch angemerkt, dass die Kostenfunktion K von dem ursprünglichen Savingsalgorithmus nicht genutzt wird. Es wird nur davon ausgegangen, dass sie stark mit der Kantengewichtsfunktion korreliert. Die Summe der Kantengewichte der implizit im Tourenplan enthaltenen Kanten soll minimal sein. Hierbei wird die Erkenntnis genutzt, dass das Aneinanderhängen von Touren diese Summe verringern kann, da zwei Kanten zum Depot wegfallen und nur eine Kante zwischen den zwei Randknoten in den Tourenplan aufgenommen wird (siehe Abbildung. 3.1).

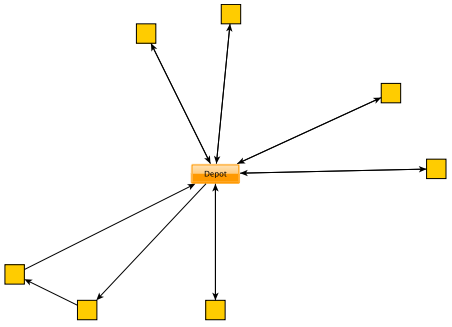
Abbildung 3.1: Die zuerst getrennt existierenden zwei Touren werden zu einer Tour zusammengefasst.



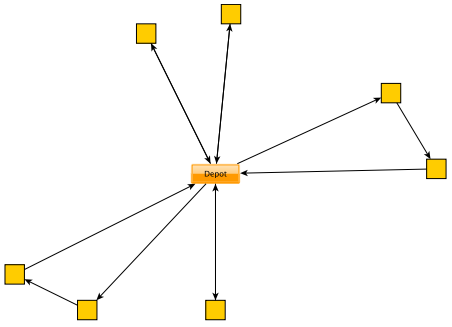
(a) Hier soll ein Tourenplan gebildet werden.



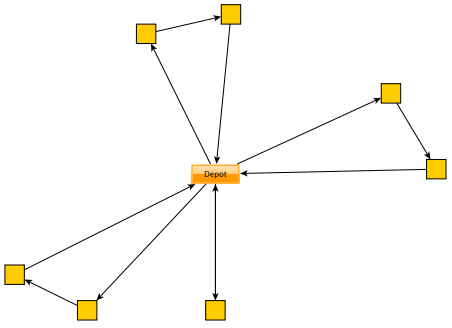
(b) Die Stichtouren werden gebildet.



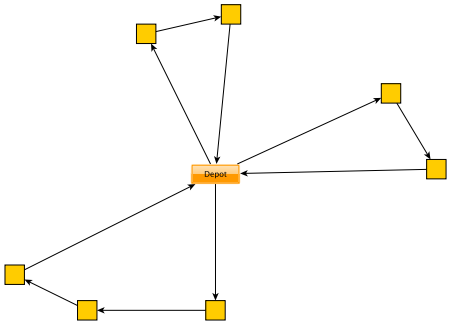
(c) Das Verschmelzen beginnt.



(d) Solange keine ...



(e) ... Nebenbedingungen verletzt werden ...



(f) ... wird weiter verschmolzen.

Abbildung 3.2: Beispielhafter Ablauf des Savingsalgorithmus. Zuerst werden maximal viele Stichtouren gebildet. In jedem nachfolgenden Schritt werden die zwei Touren miteinander verschmolzen, die die höchste Kosteneinsparung versprechen und zu keiner Verletzung der Nebenbedingungen führen.

Der nachfolgende Pseudocode zeigt die genaue Arbeitsweise des Savingsalgorithmus:

Algorithmus 1 : Der Savingsalgorithmus

Eingabe : Eine Instanz des VRP
Ausgabe : T als Tourenplan

```
1  T ← ∅;
   /* Für jeden Stopp wird eine Stichtour gebildet.                    */
2  für alle s ∈ V tue
3  |  T ← T ∪ {(v_depot, s, v_depot)};
4  Ende
   /* Alle Savings werden berechnet und sortiert.                      */
5  savingListe ← ∅;
6  für alle s_1 ∈ V \ {v_depot} tue
7  |  für alle s_2 ∈ V \ {v_depot, s_1} tue
8  |  |  savingListe ← savingListe ∪ {saving_{s_1 s_2}};
9  |  Ende
10 Ende
11 sortiereAbsteigend (savingListe);
12 solange savingListe ≠ ∅ tue
13 |  maxSaving ← erstes Element saving_uv der savingListe;
14 |  t_1 ← eindeutige Tour, die u enthält ;
15 |  t_2 ← eindeutige Tour, die v enthält ;
16 |  savingListe ← savingListe \ { maxSaving };
   |  /* Pruefen, ob die zwei, dem Saving zugehoerigen Touren, verkettet
   |     werden dürfen.                                                */
17 |  wenn (t_1 ≠ t_2) und u Endknoten und v Startknoten dann
18 |  |  wenn (T ∪ {t_1 ►► t_2}) \ {t_1, t_2} gültig dann
19 |  |  |  T ← (T ∪ {t_1 ►► t_2}) \ {t_1, t_2};
20 |  |  Ende
21 |  sonst
22 |  |  wenn (t_1 ≠ t_2) und u Startknoten und v Startknoten dann
23 |  |  |  wenn (T ∪ {t_1 ◄► t_2}) \ {t_1, t_2} gültig dann
24 |  |  |  |  T ← (T ∪ {t_1 ◄► t_2}) \ {t_1, t_2};
25 |  |  |  Ende
26 |  |  Ende
27 |  |  sonst
28 |  |  |  wenn (t_1 ≠ t_2) und u Endknoten und v Endknoten dann
29 |  |  |  |  wenn (T ∪ {t_1 ►◄ t_2}) \ {t_1, t_2} gültig dann
30 |  |  |  |  |  T ← (T ∪ {t_1 ►◄ t_2}) \ {t_1, t_2};
31 |  |  |  |  Ende
32 |  |  |  Ende
33 |  |  |  sonst
34 |  |  |  |  wenn (t_1 ≠ t_2) und u Startknoten und v Endknoten dann
35 |  |  |  |  |  wenn (T ∪ {t_1 ◄◄ t_2}) \ {t_1, t_2} gültig dann
36 |  |  |  |  |  |  T ← (T ∪ {t_1 ◄◄ t_2}) \ {t_1, t_2};
37 |  |  |  |  |  Ende
38 |  |  |  |  Ende
39 |  |  |  |
40 |  |  |
41 |  |
42 Ende
43 return T;
```

Feststellung 2 (Laufzeit des Savingsalgorithmus) *Sei $n = |V|$. Es werden $\mathcal{O}(n^2)$ Savings berechnet und sortiert. Für jedes Saving muss geprüft werden, ob die beiden zugehörigen Knoten noch Randknoten sind und ob der Tourenplan, der nach dem Verschmelzen von den beiden zugehörigen Touren entsteht, noch gültig ist. Die Laufzeit für eine solche Prüfoperation liege in $\mathcal{O}(p(n))$. Insgesamt liegt die Laufzeit des Savingsalgorihtmus in $\mathcal{O}(n^2 \log(n) + n^2 \cdot p(n))$.*

Feststellung 3 (Unabhängigkeit des Savingsalgorithmus) *Der Savingsalgorithmus ist unabhängig von der Prüffunktion D und der Kostenfunktion K. Damit ist gemeint, dass der logische Programmablauf nicht durch besondere Eigenschaften von D und K beeinflusst wird. Die Prüffunktion hat sehr wohl Einfluss auf das Endergebnis des Algorithmus, da bei unterschiedlichen Prüffunktionen auch unterschiedliche Tourenpläne als ungültig deklariert werden können. Diese Information über bestimmte Eigenschaften der Prüffunktion wird jedoch nicht in dem internen Ablauf des Algorithmus verwendet.*

3.2.3 Probleme mit gerichteten Touren und Graphen

Der von Clarke und Wright (1964) vorgestellte Savingsalgorithmus arbeitet auf vollständigen, ungerichteten Graphen mit ungerichteten Touren und Tourenplänen. Dies ist bei den bei der PTV verwendeten Algorithmen nicht möglich, da die Prüffunktion aus einer Vielzahl von Gründen intern mit gerichteten Touren arbeiten muss. Hier wären u.a. zu nennen:

- Die Zeit, die benötigt wird, um eine Tour zu befahren, kann davon abhängig sein, in welcher Richtung man sie befährt.
- Zeitfenster an Knoten können eine Richtung der Tour ausschließen.
- Abholmengen an Knoten können dazu führen, dass die maximale Beladungskapazität eines Fahrzeugs in der einen Richtung überschritten wird.
- Die Vorgabe, dass Knoten u vor Knoten v besucht werden muss, schließt eine Richtung aus.

Hierdurch ergeben sich zwei grundlegende Probleme:

1. Es muss nicht gelten: $\text{saving}_{uv} = \text{saving}_{vu}$.
2. Es ist nicht klar, wie gerichtete Touren aneinandergehängt werden sollen.

Problem 1

Die beiden Werte saving_{uv} und saving_{vu} unterscheiden sich in ihrer Semantik. Einmal wird die Kostenersparnis angegeben, wenn die beim Verschmelzen entstandene Tour von u zu v befahren wird, und im anderen Fall von v zu u. Hier ist nicht klar, ob man zwischen saving_{uv} und saving_{vu} unterscheiden soll und beide Werte berechnet, oder ob man nur für jedes Knotenpaar (u, v) einen Wert berechnet.

Verwendete Lösung

Um die Funktionsweise des ursprünglichen Savingsalgorithmus beibehalten zu können, bei der für jedes Knotenpaar nur ein Saving berechnet wird, wird für zwei Knoten u und v das folgende Saving berechnet: $\max(\text{saving}_{uv}, \text{saving}_{vu})$.

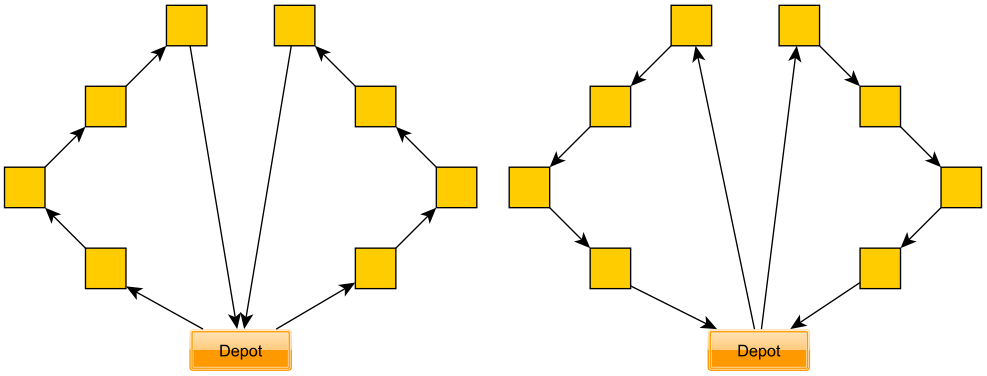


Abbildung 3.3: In diesen Fällen hängte man die entsprechenden Touren gerne aneinander, dies ist aber aufgrund der verschiedenen Richtungen nicht möglich. Das Verändern der Richtung einer Tour kann oft Abhilfe schaffen.

Problem 2

Der Savingsalgorithmus versucht sukzessive Touren aneinanderzuhängen. Sind die Touren jedoch gerichtet, ist dies nicht immer möglich (siehe Abbildung 3.3). Hier ist nicht klar, wie man reagieren soll. Entscheidet man sich, die Touren aufgrund ihrere unterschiedlichen Richtungen nicht aneinanderzuhängen, verschenkt man viel Potential. Der Aufbau der Touren geschieht bei der Verwendung des Savingsalgorithmus völlig ungeplant. Es wird nicht darauf geachtet, dass sie eine gewisse Richtung aufweisen. Jeder Schritt ist äußerst kurzsichtig, es wird immer nur die momentane größte Kosteneinsparung beachtet. Bei so einem Vorgehen ist es gut möglich, dass Touren entstehen, die ähnliche Kosten verursachen, wenn sie gedreht werden. Dementsprechend kann es auch zu großen Kosteneinsparungen kommen, wenn man beim Zusammenhängen von Touren prüft, ob eine Drehung die Kosteneinsparung vergrößern kann.

Andererseits werden durch das Drehen von Touren alle vorherigen Entscheidungen, die zu der Entstehung genau dieser Tour geführt haben, teilweise revidiert. Das ist insbesondere dann der Fall, wenn für jedes Knotenpaar (u, v) zwischen saving_{uv} und saving_{vu} unterschieden wird. In solch einem Fall hängt die Höhe der Kosteneinsparung sehr wohl von der Richtung der Tour ab. Auch kann das Drehen von Touren dazu führen, dass ein bestimmtes Saving, das vor der Drehung die höchste Kosteneinsparung versprach, nach der Drehung nur noch zu mäßigen Kosteneinsparungen führt und andere Savingswerte besser geeignet wären, um zwei Touren miteinander zu verschmelzen.

Verwendete Lösung

Wenn zwei Touren t_1 und t_2 aufgrund der unterschiedlichen Richtungen nicht aneinandergehängt werden können, so wird die Prüfung auf Gültigkeit für t_1 ►► t_2, t_1 ►◄ t_2, t_1 ◄► t_2 und t_1 ◄◄ t_2 ausgeführt. Sollten mehrere Kombinationen gültig sein, wird die mit den geringsten Kosten verwendet. Dieser Ansatz scheint gerechtfertigt, da er keine größeren Modifikationen des Savingsalgorithmus notwendig macht und der Algorithmus bei der Verschmelzung von Touren nicht zwischen saving_{uv} und saving_{vu} unterscheidet.

3.2.4 Der verwendete Savingsalgorithmus bei der PTV

Bei der PTV wurde nicht nur der naive Savingsalgorithmus (siehe Kapitel 3.2.2) implementiert, sondern eine veränderte Version, die sich an den Vorschlägen von Paessens (1988) orientiert.

Eine wichtige Veränderung ist, dass der Wert

$$\text{saving}_{uv} = c(v, v_{depot}) + c(v_{depot}, u) - c(v, u)$$

um 2 Tuningparameter f und g, wie folgt, erweitert wurde:

$$\text{saving}_{uv} = c(v, v_{depot}) + c(v_{depot}, u) - f \cdot c(v, u) + g \cdot |(c(v, v_{depot}) - c(v_{depot}, u))|$$

Wenn die Parameter f und g geschickt gewählt werden, dann führt diese neue Formel zu besseren Ergebnissen. Die genaue Wahl von f und g ist empirisch erfolgt. Intuitiv verändern beide Parameter die Relevanz der Entfernung zum Depot bei der Berechnung eines Savings. Bei der ursprünglichen Berechnung hatte die Entfernung der beiden beteiligten Knoten zum Depot einen starken Einfluss auf das Endergebnis, da sie zweimal in der Formel berücksichtigt wird. Durch den f-Parameter kann auch die Entfernung der beiden Stopps voneinander stärker gewichtet werden, der g-Parameter macht es möglich, Paare von Stopps gezielt zu berücksichtigen, die einen sehr unterschiedlichen Abstand zum Depot aufweisen.

In dem naiven Savingsalgorithmus wird zu Beginn für alle u und v das Saving saving_{uv} berechnet, diese Werte sortiert und dann nacheinander abgearbeitet. Viele werden jedoch im Verlauf des Algorithmus nicht benötigt, da ein saving_{uv} nur relevant ist, wenn sowohl u als auch v Randknoten sind. Daher werden hier nicht zu Beginn alle Savings einmalig berechnet, sondern schrittweise, nur bei Bedarf und in einem Heap gespeichert. Damit sich die Reihenfolge der betrachteten Savings nicht ändert, muss zu jedem Zeitpunkt das größte, noch zu betrachtende Saving im Heap vorhanden sein. In Paessens (1988) ist ein Weg angegeben, wie viele Knoten von vornherein, mit wenig Aufwand, ausgeschlossen werden können. Als Voraussetzungen müssen gelten:

1. $\text{saving}_{uv} \geq 0$ für alle Knoten u, v, einschließlich v_{depot}.
2. Die Dreiecksungleichung soll gelten, insbesondere $c(v_{depot}, v) \leq c(v_{depot}, u) + c(v, u)$.
3. Die Kostenfunktion c ist symmetrisch.

Die beiden ersten Bedingungen sind in unserem Fall erfüllt. Nur die Symmetrie der Kostenfunktion ist nicht gewährleistet. Hier wird eine neue Kantengewichtsfunktion $c^* = \max(c(u, v), c(v, u))$ verwendet. Für die Schwierigkeiten, die sich durch dieses Vorgehen ergeben, siehe Kapitel 3.2.3. Aus Übersichtsgründen wird c^* weiterhin als c bezeichnet. Da die Dreiecksungleichung erfüllt ist, gilt demnach auch

$$c(v, u) \geq c(v_{depot}, u) - c(v_{depot}, v) \tag{3.1}$$

Wenn man nun zwei Knoten u und v finden möchte, so dass saving_{uv} größer als die maximale, im Moment im Heap befindliche Ersparnis saving_{max} ist, so muss gelten:

$$\text{saving}_{uv} = c(v, v_{depot}) + c(v_{depot}, u) - c(v, u) > \text{saving}_{max}$$

Sollte der Heap zu Beginn noch leer sein, so wird $\text{saving}_{max} = 0$ gesetzt. Hieraus folgt unter Verwendung von Bedingung 3.1:

$$c(v, v_{depot}) > \frac{\text{saving}_{max}}{2}$$

und

$$c(u, v_{depot}) > \frac{\text{saving}_{max}}{2}$$

Es reicht also aus, nur die Knoten v_i zu betrachten, bei denen $c(v_i, v_{depot})$ einen gewissen Wert übersteigt. Dieser Wert kann für alle Knoten vorberechnet und die Knoten dann, nach diesem Wert sortiert, in einer Liste gehalten werden.

Sollte nun ein Wert aus dem Heap während des Ablaufs des Algorithmus entfernt worden sein, so kann es notwendig sein, den Heap weiter aufzufüllen. Dies geschieht analog zu der ersten Berechnung. Dabei lässt sich Rechenzeit sparen, wenn man sich merkt, welche Kombinationen von Knoten man bereits in den Heap eingefügt hat. Dies ist einfach möglich, wenn man für jeden Knoten noch einen zusätzlichen Index speichert, der auf die sortierte Liste der Knoten verweist. Eine weitere leicht herleitbare Eigenschaft aus dem Artikel von Paessens (1988) garantiert, dass man die Knoten genau in dieser Reihenfolge abarbeiten kann. Es reicht also aus, sich einen Knoten zu merken, ab dem die Berechnung zu einem späteren Zeitpunkt wieder aufgenommen werden muss.

3.2.5 Begründung für die Wahl des Savingsalgorithmus

Es gibt eine Reihe von Gründen, die für die Wahl des Savingsalgorithmus als Konstruktionsverfahren sprechen. Es ist ein einfacher, gut erforschter Algorithmus, der sich an einer Vielzahl von Beispielen bereits bewährt hat. Besonders zeichnet ihn aber aus, dass:

- seine Funktionsweise unabhängig von vorliegenden Nebenbedingungen und Kostenfunktionen ist
- seine Laufzeit sehr niedrig ist

Unabhängigkeit

Gerade bei der Betrachtung von VRPs, die ihren Ursprung in Realweltproblemen haben, gibt es eine Vielzahl von Nebenbedingungen, die betrachtet werden müssen. Auch ist diese Menge von Nebenbedingungen im ständigen Wandel begriffen. Für jede Probleminstanz ist ein gewisser Anpassungsprozess notwendig, da zu ihr individuelle Nebenbedingungen gehören. Der Savingsalgorithmus erlaubt es, dass solch ein Anpassungsprozess lediglich die Prüf- und Kostenfunktion betrifft, nicht jedoch den Algorithmus selbst. Der Algorithmus verwendet die gegebenen Funktionen als reine „Black-Box“ -Funktionen, ohne bestimmte Eigenschaften von ihnen zu berücksichtigen. Eine Änderung der Prüf- oder Kostenfunktion zieht demnach keine zwingende Änderung des Algorithmus nach sich.

Dies ist ein wichtiger Punkt, der gerade in letzter Zeit von immer mehr Autoren angesprochen wird, die sich mit verschiedenen Varianten des VRPs beschäftigen. Exemplarisch sind drei Zitate aus dem Nachwort von bekannten Artikeln genannt:

1. „On the algorithmic side, time has probably come to concentrate on the development of faster, simpler (with fewer parameters) and more robust algorithms, even if this causes a small loss in solution quality. These attributes are essential if we want to see more of our algorithms impemented in commercial packages.“ (siehe G. Laporte und Semet (2000))
2. „What is now needed is a greater emphasis on simplicity and flexibility.“ (siehe J. Cordeau und Sormany (2005))
3. „. . . the research on simpler and more flexible, yet effective, metaheuristics will also increase.“ (siehe Bräysy und Gendreau (2005b))

Laufzeit

Der Savingsalgorithmus ist von seinem Ablauf her sehr simpel. Dies führt zu einer sehr geringen Laufzeit (siehe Kapitel 2).

3.2.6 Varianten des Savingsalgorithmus

3.2.6.1 Motivation

Der Savingsalgorithmus der PTV weist eine geringe Laufzeit auf. Im Vergleich zu einer eventuell folgenden Nachoptimierung ist die benötigte Rechenzeit verschwindend gering. Hier wäre man bereit, mehr Rechenzeit für ein neues Konstruktionsverfahren zu investieren, das bessere Ergebnisse liefert. Ein naheliegender Ansatz ist, den Savingsalgorithmus mehrfach auszuführen und die beste Lösung aller Durchläufe zurückzugeben. Da es ein deterministischer Algorithmus ist, muss er so abgewandelt werden, dass sich die Durchläufe voneinander unterscheiden und unterschiedliche Ergebnisse liefern können.

Erfolgreich getestet wurde solch ein Ansatz bereits von Holmes und Parker (1976). In ihrem modifizierten Savingsalgorithmus werden in jedem Durchlauf nicht alle vorberechneten Savings betrachtet, sondern ein gewisser Teil wird in jeder Iteration, abhängig von den Ergebnissen der vorherigen Iterationen, gesperrt. So werden in jeder Iteration potentiell andere Tourenpläne mit unterschiedlichen Kosten gebildet.

Dieser Ansatz ließ sich nicht für den Savingsalgorithmus der PTV übernehmen, da er mit sehr vielen Nebenbedingungen und Einschränkungen arbeitet und daher eine sehr viel höhere Laufzeit als die meisten akademischen Varianten aufweist. Als sehr grober Richtwert kann eine obere Schranke von maximal 100 Iterationen angegeben werden, die von dem Savingsalgorithmus der PTV in vertretbarer Zeit bearbeitet werden können. Der Ansatz von Holmes und Parker (1976) benötigt sehr viel mehr Durchläufe, und es sah nicht nach

einem sinnvollen Kompromiss aus, das Verfahren nach einer festen Anzahl Iterationen abzubrechen.

Dennoch schien es ein erfolgversprechender Weg zu sein, gewisse Savings in jeder Iteration zu sperren und so unterschiedliche Lösungen für jede Iteration zu erhalten. Es ist jedoch nicht offensichtlich, welche Savings man in jeder Iteration sperren soll, wenn die Anzahl verfügbarer Iterationen begrenzt ist. Alle getesteten deterministischen Varianten erschienen durchweg unbefriedigend. Ein starker Anstieg der Ergebnisqualität war jedoch zu beobachten, als randomisierte Varianten genutzt wurden.

In diesem Abschnitt werden jetzt zwei intuitive Varianten vorgestellt, die auch ohne systematische Tests, um die besten Parameterkonfigurationen herauszufinden, bereits vielversprechende Ergebnisse geliefert haben.

3.2.6.2 Definitionen und Bemerkungen

Definition 9 (Zufallszahlen) *Seien $a, b \in \mathbb{R}$. Dann hat*

$$c \leftarrow \text{rand}(a, b)$$

nach dem Aufruf der Methode rand *einen Wert in $\mathbb{R}$, der zufällig und gleichverteilt aus dem Intervall $[a, b]$ gezogen wurde.*

3.2.6.3 Funktionsweise der Varianten

Um neue Tourenpläne bei jedem Durchlauf des Savingsalgorithmus erzielen zu können, wird die Abarbeitungsreihenfolge der vorberechneten Savings verändert. Dies geschieht in allen Fällen randomisiert. Dadurch kann potentiell jeder gültige Tourenplan erstellt werden.

Die neuen Savingsvarianten wurden um folgende Punkte erweitert:

- Einen Wert iteration $\in \mathbb{N}$, der angibt, wie viele Iterationen durchlaufen werden sollen.
- Einen Wert rnd $\in \mathbb{N}$, der das Maß der Auswirkung des Zufalls steuert.

3.2.6.4 Variante 1

In der ersten Variante des Savingsalgorithmus wird die Reihenfolge verändert, in der die vorberechneten Savings betrachtet werden. Nach jeder Iteration werden die Savingwerte zufällig, aber immer noch abhängig von dem Parameter rnd, verändert. Jeder Savingwert wird nach folgender Vorschrift verändert:

$$s \leftarrow s + \text{rand}(-\frac{s}{\text{rnd}}, \frac{s}{\text{rnd}})$$

Dann werden die Savings erneut sortiert und in der kommenden Iteration absteigend betrachtet. So ändert sich ihre Reihenfolge und in jeder Iteration können unterschiedliche Lösungen gefunden werden.

Man beachte, dass nach jeder Iteration die ehemals sortierte Savingliste weiter durchmischt wird. Es wird nicht zu Beginn einer jeden Iteration mit einer sortierten Savingliste begonnen, die zufällig verändert wird, sondern alle Veränderungen werden auf der gleichen

Savingliste durchgeführt, deren Elemente so immer weiter durchmischt werden. Diese Entscheidung wurde getroffen, da so bessere Ergebnisse geliefert wurden. Dies wurde jedoch nur stichprobenartig getestet und nicht systematisch untersucht.

Algorithmus 2 : Variante 1 des Savingsalgorithmus

Eingabe : Eine Instanz des VRP, ein Parameter iteration $\in \mathbb{N}$, der die Anzahl Iterationen angibt und ein Parameter rnd $\in \mathbb{N}$ zur Zufallssteuerung
Ausgabe : T als Tourenplan

```
1  T ← ∅;
2  bestTourplan ← ∅;
3  savingListe ← fuelleSavingsListe();
4  sortiereAbsteigend (savingListe);
   /* Jetzt sind alle Savings berechnet und liegen sortiert vor.        */
5  i ← 0;
6  solange i < iteration tue
   |  /* Die aktuelle Savings-Liste wird abgearbeitet.                  */
7  |  kopieDerListe ← savingListe;
8  |  T ← arbeiteSavingsAb ();
9  |  savingListe ← kopieDerListe;
10 |  wenn K(T) < K(bestTourplan ) dann
11 |  |  bestTourplan ← T;
12 |  Ende
   |  /* Jetzt wird die Savings-Liste neu durchmischt.                  */
13 |  für alle s ∈ savingListe tue
14 |  |  s ← s + rand (−s/rnd, s/rnd);
15 |  Ende
16 |  sortiereAbsteigend (savingListe);
17 |  i ++;
18 Ende
19 T ← bestTourplan;
20 return T;
```

Feststellung 4 *Zu der Variante 1 des Savingsalgorithmus sind die beiden Tuningparameter* iteration $\in \mathbb{N}$ *und* rnd $\in \mathbb{N}$ *neu hinzugekommen.*

Feststellung 5 *Die Laufzeit der Variante 1 des Savingsalgorithmus liegt in* $\mathcal{O}(\text{iteration} \cdot g(n))$*, wenn* $g(n)$ *die Laufzeit des originalen Savingsalgorithmus angibt.*

3.2.6.5 Variante 2

In der zweiten Variante des Savingsalgorithmus wird ebenfalls die Reihenfolge verändert, in der die vorberechneten Savings betrachtet werden. Während jeder Iteration wird nicht immer das Saving, das die höchste Kosteneinsparung verspricht, betrachtet, sondern es wird zufällig und gleichverteilt eines der rnd-höchsten Savings ausgewählt. Das führt erneut dazu, dass in jeder Iteration andere Tourenpläne gebildet werden, von denen der beste

zurückgegeben wird.

Algorithmus 3 : Variante 2 des Savingsalgorithmus

Eingabe : Eine Instanz des VRP, ein Parameter iteration $\in \mathbb{N}$, der die Anzahl Iterationen angibt und ein Parameter rnd $\in \mathbb{N}$ zur Zufallssteuerung
Ausgabe : T als Tourenplan

```
1  T ← ∅;
2  bestTourplan ← ∅;
3  savingListe ← fuelleSavingsListe();
4  sortiereAbsteigend (savingListe);
   /* Jetzt sind alle Savings berechnet und liegen sortiert vor.        */
5  i ← 0;
6  solange i < iteration tue
   |  /* Alle Elemente der Savings-Liste werden abgearbeitet.          */
7  |  kopieDerListe ← savingListe;
8  |  solange savingListe ≠ ∅ tue
9  |  |  c ← ⌊ rand (1, rnd) ⌋ ;
10 |  |  maxSaving ← c-tes Element der savingListe;
11 |  |  savingListe ← savingListe \ { maxSaving };
12 |  |  T ← bearbeiteSaving (maxSaving);
13 |  Ende
14 |  savingListe ← kopieDerListe;
15 |  wenn K(T) < K(bestTourplan ) dann
16 |  |  bestTourplan ← T;
17 |  Ende
18 |  i ++;
19 Ende
20 T ← bestTourplan;
21 return T;
```

Feststellung 6 *Zu der Variante 2 des Savingsalgorithmus sind die beiden Tuningparameter* iteration $\in \mathbb{N}$ *und* rnd $\in \mathbb{N}$ *im Vergleich zu dem originalen Savingsalgorithmus neu hinzugekommen.*

Feststellung 7 *Die Laufzeit der Variante 2 des Savingsalgorithmus liegt in* $\mathcal{O}(\text{iteration} \cdot g(n))$*, wenn* $g(n)$ *die Laufzeit des originalen Savingsalgorithmus angibt.*

3.2.7 Das Nearest-Neighbor-Verfahren

Wichtiger Bestandteil dieser Diplomarbeit war der Vergleich der neu entwickelten Algorithmen mit den bestehenden Verfahren der PTV. Um bei diesen Tests mehr Vergleichsmöglichkeiten zu haben und eventuell bessere Aussagen über die Qualität der einzelnen Verfahren treffen zu können, wurde noch ein weiteres Verfahren implementiert.

Dieses neue Verfahren ähnelt dem Savingsverfahren, aber es werden nicht die Touren miteinander verschmolzen, die die größte Kosteneinsparung versprechen, sondern immer die beiden Touren, deren Enden am nächsten beieinander liegen.

Feststellung 8 *Das Nearest-Neighbor-Verfahren lässt sich als Savingsalgorithmus mit folgender modifizierter Kostenersparnisfunktion auffassen:*

$$\text{saving}_{uv} = (-1) \cdot c(v, u)$$

Solch ein Vorgehen scheint intuitiv sinnvoll zu sein. Auch rein optisch sahen die als Ergebnis gelieferten Tourenpläne gut aus. Dazu kam, dass sich dieses neue Verfahren schnell implementieren ließ, es eine geringe Laufzeit aufweist und es keine Parameter gibt, die variiert werden können, so dass es sich mit geringem Aufwand in die Testreihen eingliedern ließ.

3.3 Der SmartSwap

Bei der PTV wurde bereits eine voll funktionsfähige Metaheuristik zur Verbesserung von existierenden Tourenplänen zu einem beliebigen VRP implementiert. Es ist eine *Granulare Tabusuche*, abkürzend GTS genannt, die sich sehr stark an dem Artikel von Toth und Vigo (2003) orientiert. Das vorhandene Framework wurde um eine weitere Funktionalität erweitert. Das verwendete Verfahren und die Erweiterung seien hier vorgestellt.

3.3.1 Einführung Lokale Suchen und Tabusuche

In den folgenden Kapiteln wird eine *Granulare Tabusuche*, die auf das VRP zugeschnitten ist, verwendet. Hier sollen die Grundlagen und Voraussetzungen für ihre Definition geschaffen werden. Es werden schrittweise Verfahren und deren Verallgemeinerungen vorgestellt, bis es möglich ist, die Tabusuche zu beschreiben, die im Rahmen dieser Diplomarbeit implementiert und getestet wurde.

Definition 10 (Lokaler Suchalgorithmus) *Ein* Lokaler Suchalgorithmus *oder eine* Lokale Suche *zu einem Problem* Π *und einer Probleminstanz* π *ist eindeutig über folgende Komponenten definiert:*

- *Den* Suchraum $S(\pi)$*, eine endliche Menge von Lösungskandidaten, abkürzend* S *genannt.*
- *Eine* Prüffunktion $D : S \to \{0,1\}$*. Ein* $s \in S$ *wird als* gültig *bezeichnet, wenn* $D(s) = 1$*.*
- *Eine* Kostenfunktion $K : S \to \mathbb{R}$*.*
- *Eine* Nachbarschaftsfunktion $N : S \to \mathcal{P}(S)$*. Zu einem* $s \in S$ *wird* $N(s)$ *als* Nachbarschaft *von* s *bezeichnet.*

Definition 11 (Lokales Optimum) *Zu einem gegebenen Suchraum* S*, einer Prüffunktion* D*, einer Kostenfunktion* K *und einer Nachbarschaftsfunktion* N *ist ein* Lokales Optimum *ein gültiges* $s \in S$*, so dass für alle gültigen* $s^* \in N(s)$ *gilt:* $K(s) \leq K(s^*)$*.*

Die Arbeitsweise jeder Lokalen Suche ist durch folgenden Algorithmus eindeutig bestimmt:

Algorithmus 4 : Funktionsweise einer Lokalen Suche

Eingabe : Suchraum S, Prüffunktion D, Kostenfunktion K, Nachbarschaftsfunktion N und gültige Startlösung $s \in S$
Ausgabe : Eine gültige Lösung $s_{min} \in S$

```
1 s_min ← s;
2 solange s_min ist kein Lokales Optimum tue
3 |  s_min ← gültiges s_neu ∈ N(s_min) mit K(s_neu) minimal;
4 Ende
5 gib zurück s_min;
```

Definition 12 (Schlichte Tabusuche) *Eine* Schlichte Tabusuche *zu einem Problem* Π *und einer Probleminstanz* π *ist über folgende Komponenten definiert:*

- *Eine Lokale Suche zu* Π *und* π*.*
- *Zwei Abbildungen* $h_1 : S \times S \to \mathcal{P}(B)$ *und* $h_2 : S \times S \to \mathcal{P}(B)$ *mit* B *eine Menge von* Eigenschaften*. Die Menge* B *repräsentiert die Eigenschaften, die Bestandteil einer neu ausgewählten Lösung sein können und die eventuell mittels der Tabuliste verboten sind.*
- *Ein Abbruchkriterium.*
- *Eine Menge* Z *von inneren Zuständen und eine Übergangsfunktion* $\delta : Z \to Z$*.*
- *Ein innerer Zustand* $z \in Z$ *besteht aus:*
  - *Einer bisherigen besten Lösung* $s_{best} \in S$
  - *Einer* Tabuliste $(b_1, b_2 \dots b_k)$ *mit allen* $b_i \in B$
  - *Einem* $l_{max} \in \mathbb{N}$ *als maximale Länge der Tabuliste*

Die Übergangsfunktion δ jeder Schlichten Tabusuche ist durch folgenden Algorithmus eindeutig bestimmt:

Algorithmus 5 : Funktionsweise einer Schlichten Tabusuche

Eingabe : Suchraum S, Prüffunktion D, Kostenfunktion K, Nachbarschaftsfunktion N, Tabuliste t, maximale Länge der Tabuliste l_{max}, zwei Abbildungen $h_{1,2} : S \times S \to B$, ein Abbruchkriterium und eine gültige Startlösung $s \in S$

Ausgabe : Eine gültige Lösung $s_{best} \in S$

```
1  s_best ← s;
2  t ← ∅;
3  solange Abbruchkriterium ist nicht erfüllt tue
4  |  s_min ← gültiges s_min ∈ N(s) mit K(s_min) minimal und h_1(s, s_min) ∩ t = ∅;
   |  /* Aktualisieren der Tabuliste                                          */
5  |  für alle b ∈ (h_2(s, s_min) \ t) tue
6  |  |  t = (b_1, b_2 ... b_k) ← (b_1, b_2 ... b_k, b);
7  |  |  wenn k ≥ l_max dann
8  |  |  |  t ← (b_2 ... b_k, b);
9  |  |  Ende
10 |  Ende
   |  /* Aktualisieren der bisherigen besten Lösung                           */
11 |  wenn s_min < s_best dann
12 |  |  s_best ← s_min;
13 |  Ende
14 |  s ← s_min;
15 Ende
16 gib zurück s_best;
```

Definition 13 (Tabusuche) *Eine* Tabusuche *ist eine Verallgemeinerung der Schlichten Tabusuche. Sie ist über folgende,* ***zusätzliche*** *Komponenten definiert:*

- *Eine Menge* L *von Lokalen Suchen und einem* $l_{start} \in L$
- *Ein Aspirationskriterium*
- *Ein Diversifikationskriterium und eine Abbildung* $\mathrm{div} : L \to L$
- *Ein Intensifikationskriterium und eine Abbildung* $\mathrm{int} : L \to L$

Die Arbeitsweise jeder Tabusuche ist durch folgenden Algorithmus eindeutig bestimmt:

Algorithmus 6 : Funktionsweise einer Tabusuche

Eingabe : Suchraum S, Prüffunktion D, Kostenfunktion K, Nachbarschaftsfunktion N, Tabuliste t, maximale Länge der Tabuliste l_{max}, zwei Abbildungen $h_{1,2} : S \times S \to B$, Menge von Lokalen Suchen L, ein $l_{start} \in L$, ein Abbruchkriterium, ein Aspirationskriterium, ein Diversifikationskriterium, Abbildung div : $L \to L$, ein Intensifikationskriterium, Abbildung int : $L \to L$ und gültige Startlösung $s \in S$

Ausgabe : Eine gültige Lösung $s_{best} \in S$

```
1  s_best ← s;
2  t ← ∅;
3  aktuelleSuche ← l_start;
4  solange Abbruchkriterium ist nicht erfüllt tue
       /* Wählen der minimalen nächsten Lösung, die nicht durch die
          Tabuliste gesperrt ist, oder bei der das Aspirationskriterium
          erfüllt ist                                                  */
5      s_min ← gültiges s_min ∈ N(s) mit K(s_min) minimal und (h_1(s, s_min) ∩ t = ∅ oder
       Aspirationskriterium ist erfüllt);
       /* Aktualisieren der Tabuliste                                  */
6      für alle b ∈ (h_2(s, s_min) \ t) tue
7          t = (b_1, b_2 ... b_k) ← (b_1, b_2 ... b_k, b);
8          wenn k ≥ l_max dann
9              t ← (b_2 ... b_k, b);
10         Ende
11     Ende
       /* Aktualisieren der bisherigen besten Lösung                   */
12     wenn s_min < s_best dann
13         s_best ← s_min;
14     Ende
       /* Überprüfung auf Intensivierung oder Diversifizierung         */
15     wenn Diversifikationskriterium erfüllt dann
16         aktuelleSuche ← div(aktuelleSuche);
17     Ende
18     wenn Intensifikationskriterium erfüllt dann
19         aktuelleSuche ← int(aktuelleSuche);
20     Ende
21     s ← s_min;
22 Ende
23 gib zurück s_best;
```

Abbruchkriterium

Da eine Tabusuche nicht abbricht, wenn ein Lokales Optimum gefunden wurde, muss ein anderes Kriterium angegeben sein, bei dem der Algorithmus abbricht. Übliche Kriterien sind:

- Es wurde eine festgelegte Anzahl Iterationen durchlaufen.
- Es wurde für eine festgelegte Anzahl von Iterationen kein neues s_{best} gefunden.
- Eine bestimmte Rechenzeit wurde überschritten.

Diversifizierung

Scheint die aktuell verwendete Lokale Suche zu keiner Verbesserung mehr zu führen, weil ihre Nachbarschaft zu begrenzt ist, kann eine *Diversifizierung* stattfinden. Bei der Diversifizierung wird eine neue Lokale Suche ausgewählt, deren Nachbarschaft Elemente enthält, die sich stärker als zuvor von der aktuell betrachteten Lösung unterscheiden.

Intensivierung

Wurde nach einer oder mehreren Diversifizierungen eine neue beste Lösung gefunden, kann es gewünscht sein, dass die Elemente, die der neuen besten Lösung stark ähneln, genauer untersucht werden. Es wird also eine andere Lokale Suche mit entsprechender Nachbarschaft verwendet. Diesen Wechsel nennt man *Intensivierung*.

Aspirationskriterium

Es kann sein, dass ein gültiger Tourenplan nicht akzeptiert wird, da er in Konflikt mit den Elementen der Tabuliste steht. Dies ist in den meisten Fällen gewollt, um ein Kreiseln der Lösungen zu verhindern. Es kann aber Einzelfälle geben, in denen man solche Lösungen dennoch akzeptieren will. Dies ist der Fall, wenn das *Aspirationskriterium* erfüllt ist. Üblicherweise soll eine Lösung immer akzeptiert werden, wenn sie geringere Kosten als die bisherige beste Lösung aufweist.

3.3.2 Konkrete Tabusuche für das VRP

Betrachten wir nun eine Schlichte Tabussuche, die an das VRP angepasst ist. Hier gilt:

- Π ist die Menge aller VRPs und π ist eine VRP-Instanz.
- Sei der Graph G und $v_{depot} \in G$ zu π gehörend, dann ist $S(\pi)$ die Menge der Tourenpläne zu G und v_{depot}.
- D ist die zu π gehörende Prüffunktion und K ist die Kostenfunktion von π.
- Als Abbruchkriterium wird eine maximale Anzahl von Iterationen verwendet.
- Für N wird eine der folgenden Nachbarschaftsfunktionen gewählt:
  1. Die *Two-Exchange-Nachbarschaft* eines konkreten Tourenplans enthält alle anderen Tourenpläne, die sich bilden lassen, indem man die beliebig langen Enden von zwei verschiedenen Touren vertauscht (siehe Abbildung 3.4).
  2. Bei allen Elementen der *Or1-Nachbarschaft* wird ein Knoten aus einer bestehenden Tour entfernt und in eine andere Tour eingefügt (siehe Abbildung 3.5).
  3. Analog zu der Or1-Nachbarschaft wird auch eine *Or2-Nachbarschaft* verwendet. Bei ihr werden zwei aufeinander folgende Knoten aus einer Tour gelöscht und in eine andere eingefügt.
  4. Bei der *Swap-Nachbarschaft* tauschen zwei Knoten ihre Position (siehe Abbildung 3.6).

  Es ist auch möglich mehrere Nachbarschaftsfunktionen $N_1, N_2 \dots N_k$ zu kombinieren, indem man N definiert als $N(s) = N_1(s) \cup N_2(s) \cup \dots \cup N_k(s)$.
- Die Menge von Eigenschaften B ist eine Menge von Kanten. Sei $G = (V, E)$ der zu π gehörende Graph, dann gilt $B = E$.

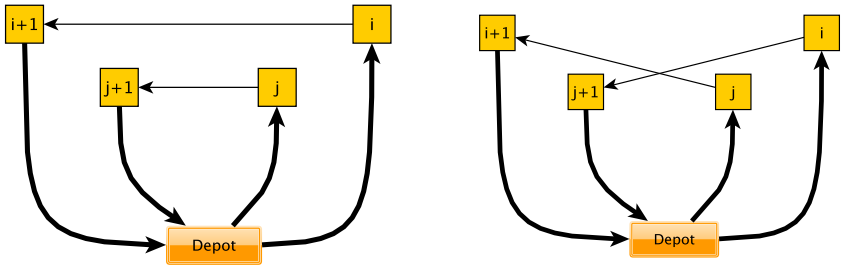


Abbildung 3.4: Two-Exchange: Alle Knoten, die nach dem Knoten i in der einen Tour liegen, werden mit allen Knoten aus der anderen Tour getauscht, die hinter dem Knoten j liegen.

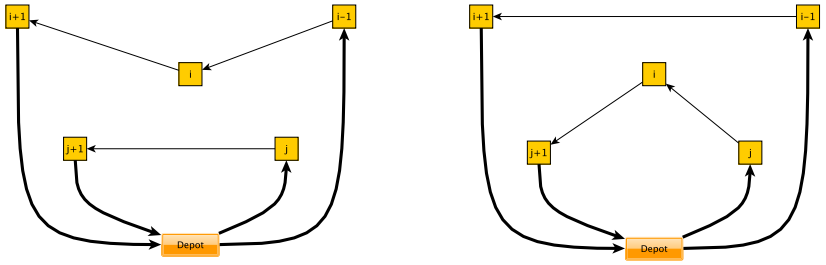


Abbildung 3.5: Or1: Der Knoten i wird aus der einen in eine andere Tour versetzt.

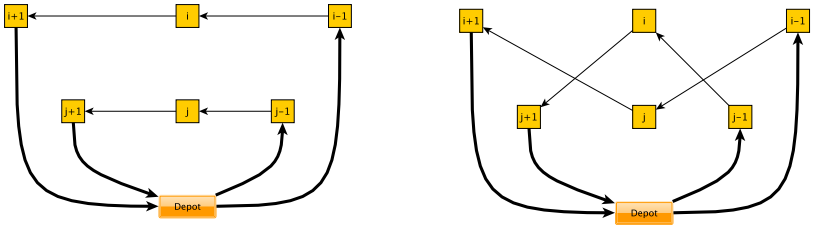


Abbildung 3.6: Swap: Zwei Knoten aus zwei verschiedenen Touren tauschen ihre Plätze.

- Die Abbildung $h_1(t_1, t_2)$ gibt die Menge von Kanten zurück, die beim Übergang von Tourenplan t_1 zu t_2 gelöscht werden.
- Die Abbildung $h_2(t_1, t_2)$ gibt die Menge von Kanten zurück, die beim Übergang von Tourenplan t_1 zu t_2 hinzugefügt werden.

3.3.3 Die Granulare Tabusuche

Die Granulare Tabusuche (GTS) ist eine Erweiterung der an das VRP angepassten Tabusuche. Intuitiv scheint es nicht nötig zu sein, im Verlauf einer Tabusuche in jeder Iteration alle Elemente der verwendeten Nachbarschaft auf Gültigkeit zu prüfen. Knoten, die geografisch oder zeitlich sehr weit voneinander entfernt liegen, treten in optimalen Lösungen fast nie direkt hintereinander in einer Tour auf. Um diesem Umstand Rechnung zu tragen, wurde die Tabusuche um einen *Granularitätsparameter* $k \in \mathbb{N}$ und ein *Granularitätskriterium* erweitert.

Definition 14 (Granularitätsnachbarschaft) *Die* Granularitätsnachbarschaft *eines bestimmten Knotens enthält nur die Knoten, die am nächsten zu ihm liegen. Formal lässt sich dies folgendermaßen formulieren: Gegeben ist ein VRP mit $G = (V, E)$, $c : E \to \mathbb{R}_+$, ein Granularitätsparameter k und die Abbildung $N_G : V \to \mathcal{P}(V)$ mit $N_G(v) = \{v_1, v_2, \ldots, v_k\}$ wobei für jedes $u \notin N_G(v)$ gilt:*

$$c(\{v, u\}) \geq \max_{1 \leq i \leq k} c(\{v, v_i\})$$

Dann ist $N_G(v)$ die Granularitätsnachbarschaft von v.

Granularitätskriterium

Sei N eine Nachbarschaftsfunktion, k der Granularitätsparameter, T ein Tourenplan und $N(T)$ die Menge von Tourenplänen, aus der ein gültiger Tourenplan für die nächste Iteration der GTS gewählt wird. Dann besagt das Granularitätskriterium, dass nur die $T^* \in N(T)$ auf Gültigkeit untersucht werden müssen, für die gilt: In T^* dürfen zwei Knoten u und v, die nicht gegenseitig in ihrer Granularitätsnachbarschaft enthalten sind, in einer Tour nur direkt aufeinander folgen, wenn sie dies bereits in T tun.

3.3.4 Motivierendes Beispiel

Die vier bisher vorgestellten Nachbarschaftsfunktionen können zusammen mit der GTS einen Tourenplan oft erstaunlich gut verbessern. Dennoch gibt es eine Reihe von Fällen, gerade in Verbindung mit sehr strengen Nebenbedingungen, bei denen recht offensichtliche Verbesserungen noch möglich wären, die mit den bisherigen Nachbarschaftsfunktionen jedoch nicht gefunden werden können. Hierzu ein kurzes Beispiel:

- In Abbildung 3.7 ist der Ausschnitt aus einem Tourenplan zu erkennen, der optimiert werden soll. Der Tourenplan ließe sich verbessern, wenn Knoten 2 von der unteren in die obere und Knoten e von der oberen in die untere Tour wanderte.
- Der Swap kann die beiden Knoten ihre Touren wechseln lassen, jedoch tauschen sie exakt ihre Positionen. Der so entstehende Tourenplan (siehe Abbildung 3.8) ist nicht optimal. Solch eine Vertauschung kann auch zu einer Verschlechterung des bisherigen Tourenplans führen und wird daher oft nicht ausgewählt.
- Auch kann das Einfügen von zwei Knoten an zunächst sehr ungünstigen Positionen dazu führen, dass der Tourenplan ungültig wird. Solche Operationen sind nie erlaubt, auch wenn nachfolgende Operationen zu einem gültigen und besseren Tourenplan führen könnten.

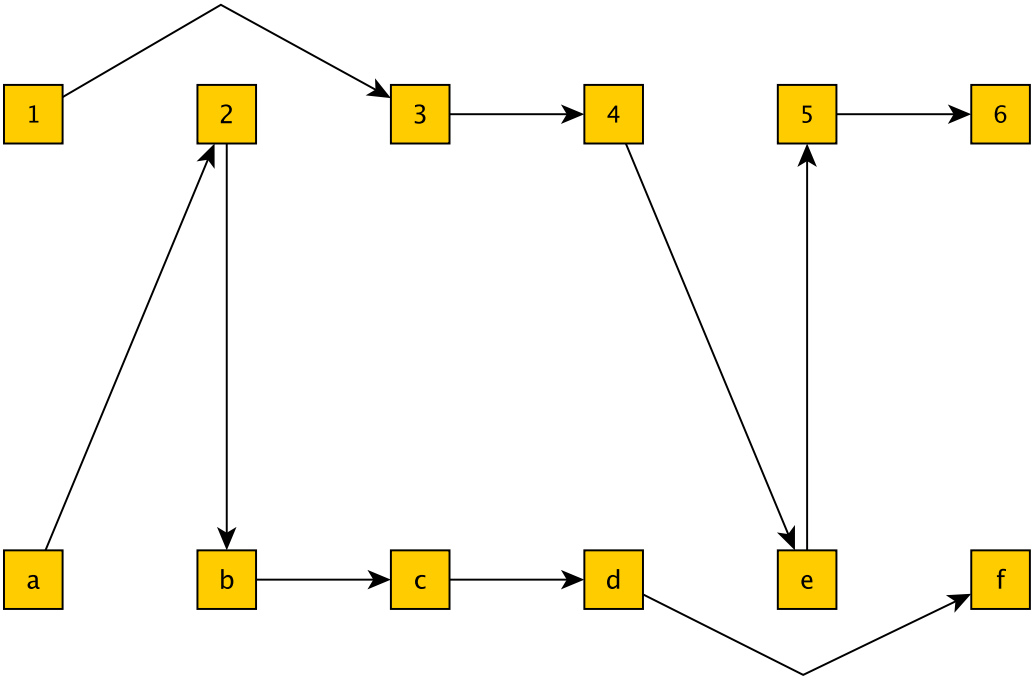


Abbildung 3.7: Hier sind zwei Touren dargestellt, die optimiert werden sollen.

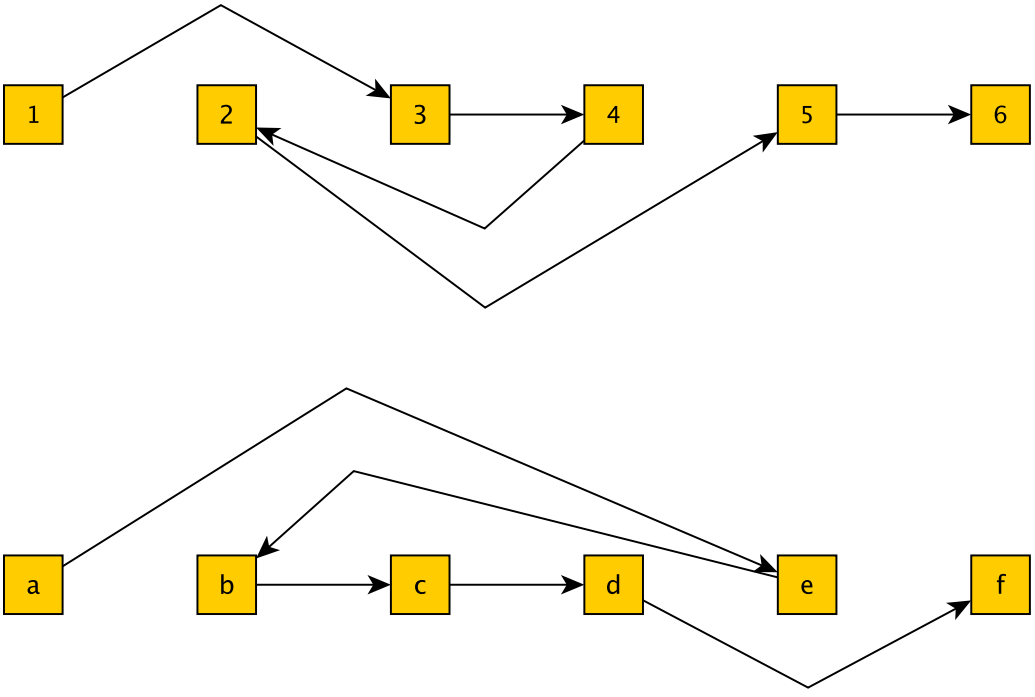


Abbildung 3.8: Der Swap kann in einer Iteration lediglich zu diesem neuen Tourenplan führen.

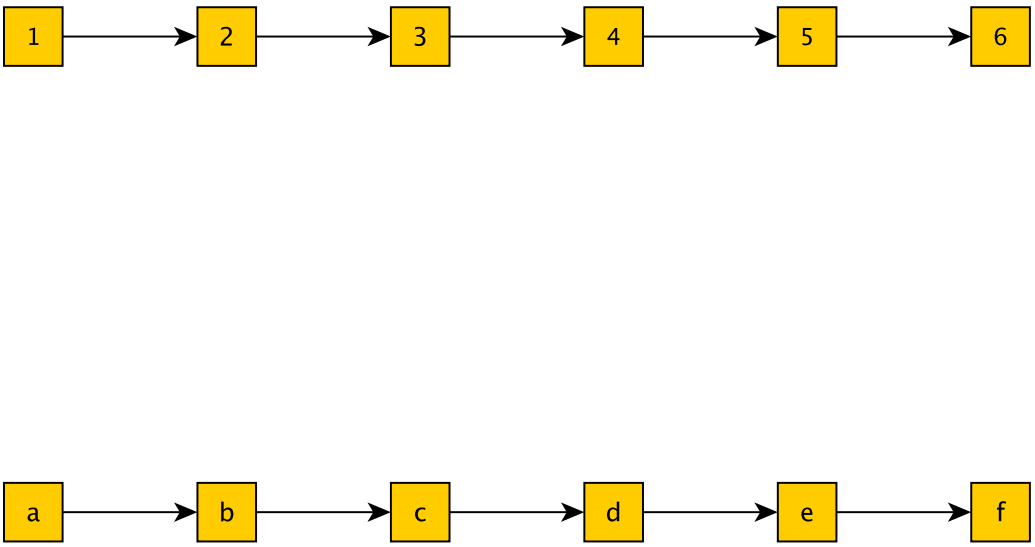


Abbildung 3.9: Der SmartSwap dagegen kann die zu vertauschenden Knoten direkt an der besten Position einfügen.

- Es wäre also wünschenswert, zusätzlich eine Lokale Suche zu verwenden, die direkt die gewünschte Lösung aus Abbildung 3.9 in ihrer Nachbarschaft enthält.

Solche Fälle waren die Motivation für die Entwicklung einer neuen Nachbarschaftsfunktion, des *SmartSwaps*. Sein Verhalten ist angelehnt an den Swap. In dieser Nachbarschaft sollen zu einem gegebenen Tourenplan T all die Tourenpläne enthalten sein, die man erhält, wenn zwei Knoten u und v aus unterschiedlichen Touren ihre Tourzugehörigkeit tauschen. Knoten u soll aber nicht wie beim Swap an die ehemalige Position von v eingefügt werden, sondern direkt an die bestmögliche Position in der Tour. Gleiches soll für v gelten. Mit solch einer Nachbarschaft ist es möglich, direkt von dem Tourenplan aus Abbildung 3.7 zu dem aus Abbildung 3.9 überzugehen. Aus implementationstechnischen Gründen (siehe Kapitel 3.3.7), wurde die Nachbarschaft so gewählt, dass sie *alle* Tourenpläne enthält, die sich bilden lassen, wenn zwei Knoten aus unterschiedlichen Touren ihre Tourzugehörigkeit tauschen und an eine *beliebige* Stelle in der neuen Tour eingefügt werden.

3.3.5 Formale Definition des SmartSwaps

Der SmartSwap ist über eine Nachbarschaftsfunktion $N : M \to \mathcal{P}(M)$ definiert.

- Sei $G = (V, E)$ ein Graph, $v_{depot} \in V$, R die Menge aller Rundtouren in G und M die Menge aller Tourenpläne zu G und v_{depot}.
- Seien $T \in M$, $t_1 = (v_{\phi(1)}, v_{\phi(2)}, \dots v_{\phi(m)}) \in T$, $t_2 = (v_{\sigma(1)}, v_{\sigma(2)}, \dots v_{\sigma(k)}) \in T$ beliebig, mit ϕ und σ Permutationen und $t_1 \neq t_2$.
- Seien $a_1, a_2, b_1, b_2 \in \mathbb{N}$ mit $1 < a_1 < m$, $1 < a_2 < k$, $1 < b_1 \leq k$ und $1 < b_2 \leq m$.
- Sei $\otimes : (M \times R \times R \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}) \to M$ eine Hilfsfunktion mit $\otimes(T, t_1, t_2, a_1, a_2, b_1, b_2) = (T \setminus \{t_1, t_2\}) \cup \{t_1^*, t_2^*\}$, wobei

$$t_1^* = (v_{\phi(1)}, v_{\phi(2)}, \dots v_{\phi(a_1-1)}, v_{\phi(a_1+1)}, \dots v_{\phi(b_2-1)}, v_{\sigma(a_2)}, v_{\phi(b_2)}, \dots v_{\phi(m)})$$

und

$$t_2^* = (v_{\sigma(1)}, v_{\sigma(2)}, \dots v_{\sigma(a_2-1)}, v_{\sigma(a_2+1)}, \dots v_{\sigma(b_1-1)}, v_{\phi(a_1)}, v_{\sigma(b_1)}, \dots v_{\sigma(k)})$$

(Um Fallunterscheidungen zu vermeiden, wurde bei der Definition von t_1^* und t_2^* angenommen, dass $b_2 > a_1$ und $b_1 > a_2$, alle anderen Fälle verlaufen analog.)

Definition 15 *Der SmartSwap ist definiert über die obige Nachbarschaftsfunktion N mit*

$$N(T) = \bigcup_{\substack{t_i,t_j \in T \\ t_i \neq t_j \\ a_q,a_w,b_e,b_r \in \mathbb{N} \\ 1<a_q<l(t_i) \\ 1<a_q<l(t_j) \\ 1<b_e \leq l(t_i) \\ 1<b_r \leq l(t_j)}} \otimes(T, t_i, t_j, a_q, a_w, b_e, b_r)$$

3.3.6 Laufzeituntersuchung des SmartSwaps

Die Laufzeit bei der Nachoptimierung wird von den ausgeführten Gültigkeitsprüfungen dominiert. Es ist jedoch nicht möglich, scharfe Laufzeitschranken für diese Prüfungen anzugeben, da ihre Implementation nicht bekannt ist und sie lediglich als „Black-Box“ -Methoden verwendet werden. Daher beschränkt sich die Analyse darauf, die Anzahl notwendiger Prüfungen als Maß für die zeitliche Komplexität der betrachteten Algorithmen zu verwenden. Die Nachoptimierung besteht aus beliebig vielen, hintereinander ausgeführten Iterationen. Hier wird die Laufzeit für eine einzelne Iteration, ohne Betrachtung der Granularitätsbedingungen betrachtet.

Gegeben sei ein Tourenplan T zu einem Graphen $G = (V, E)$, mit $|V| = n$, n_T sei die Anzahl aller vorkommenden Knoten in T ohne v_{depot} und $l_{max} = \max_{t \in T} \{l(t)\}$. Die Größe der Nachbarschaft zu dem gegebenen Tourenplan T lässt sich wie folgt abschätzen:

$$\begin{aligned}
|N(T)| &= \sum_{t_1 \in T} \sum_{\substack{t_2 \in T \\ t_2 \neq t_1}} \sum_{1<a<l(t_1)} \sum_{1<b<l(t_2)} \sum_{1<c \leq l(t_1)} \sum_{1<d \leq l(t_2)} 1 \\
&< \sum_{t_1 \in T} \sum_{t_2 \in T} \sum_{1<a<l(t_1)} \sum_{1<b<l(t_2)} l(t_1) \cdot l(t_2) \\
&\leq \sum_{t_1 \in T} \sum_{t_2 \in T} \sum_{1<a<l(t_1)} \sum_{1<b<l(t_2)} l_{max}^2 \\
&= \sum_{t_1 \in T} \sum_{t_2 \in T} (l(t_1) - 2) \cdot (l(t_2) - 2) \cdot l_{max}^2 \\
&= \sum_{t_1 \in T} (l(t_1) - 2) \cdot n_T \cdot l_{max}^2 \\
&= n_T^2 \cdot l_{max}^2 \\
&< n^2 \cdot l_{max}^2
\end{aligned}$$

Jedes Element der Nachbarschaft wird generiert und auf Gültigkeit geprüft. Daraus ergibt sich folgende Feststellung:

Feststellung 9 *Zu einem gegebenen Graphen $G = (V, E)$, mit $|V| = n$ und einem beliebigen Tourenplan T zu G, dessen Touren eine maximale Stopplänge l_{max} haben, müssen während einer Iteration des SmartSwaps $\mathcal{O}(n^2 \cdot l_{max}^2)$ Prüfoperationen durchgeführt werden.*

Wenn man das gegebene Problem etwas näher betrachtet, fällt auf, dass eine Reihe von getätigten Prüfoperationen unabhängig voneinander sind. Ob ein Tourenplan T gültig bleibt, wenn man einen Knoten u an der Position k in der Tour t_u einfügt, ist unabhängig davon, ob ein anderer Stopp v an der Position l in einer anderen Tour t_v eingefügt wird.

Die einzige Ausnahme bildet hier das Vorkommen von *Mehrfacheinsätzen*. Ein Mehrfacheinsatz liegt immer dann vor, wenn ein Fahrzeug zwei oder mehr Touren bedient. Dann kann das Verändern einer Tour sehr wohl Auswirkungen auf die folgenden Touren haben und somit auch die Gültigkeit beeinflussen. Typischerweise kann sich der Startzeitpunkt für spätere Touren verschieben, so dass enge Zeitfenster nicht mehr eingehalten werden können.

Wenn zwei Touren t_1 und t_2 in einem Tourenplan T von dem gleichen Fahrzeug bedient werden, müssen also weiterhin die Tourenpläne

$$\bigcup_{\substack{a_q,a_w,b_e,b_r\in\mathbb{N}\\ 1<a_q<l(t_i)\\ 1<a_q<l(t_j)\\ 1<b_e\leq l(t_i)\\ 1<b_r\leq l(t_j)}} \otimes(T,t_1,t_2,a_q,a_w,b_e,b_r)$$

einzeln auf Gültigkeit überprüft werden. Sollten t_1 und t_2 jedoch von unterschiedlichen Fahrzeugen bedient werden, so reicht es bei jedem Knotenpaar $\{u,v\}$ mit $u \in t_1$ und $v \in t_2$ die Einfügepositionen für Knoten u unabhängig von den Positionen für Knoten v auf Gültigkeit zu überprüfen. Anstelle des Produkts beider Tourstopplängen muss so nur noch die Summe an Prüfungen durchgeführt werden. Dennoch kann mit den so gewonnenen Informationen wieder die ursprüngliche Nachbarschaft aufgebaut werden. Die Anzahl der Prüfoperationen lässt sich also auch folgendermaßen abschätzen:

$$\begin{aligned}
|\text{Prüfoperationen}| &= \sum_{t_1\in T}\sum_{\substack{t_2\in T\\ t_2\neq t_1}}\sum_{1<a<l(t_1)}\sum_{1<b<l(t_2)}\Big(\sum_{1<c\leq l(t_1)}1+\sum_{1<d\leq l(t_2)}1\Big)\\
&< \sum_{t_1\in T}\sum_{t_2\in T}\sum_{1<a<l(t_1)}\sum_{1<b<l(t_2)} l(t_1)+l(t_2)\\
&\leq \sum_{t_1\in T}\sum_{t_2\in T}\sum_{1<a<l(t_1)}\sum_{1<b<l(t_2)} 2\cdot l_{max}\\
&= \sum_{t_1\in T}\sum_{t_2\in T} 2\cdot(l(t_1)-2)\cdot(l(t_2)-2)\cdot l_{max}\\
&= \sum_{t_1\in T} 2\cdot n_T\cdot(l(t_1)-2)\cdot l_{max}\\
&= 2\cdot n_T^2\cdot l_{max}\\
&< 2\cdot n^2\cdot l_{max}
\end{aligned}$$

Feststellung 10 *Zu einem gegebenen Graphen $G=(V,E)$, mit $|V|=n$ und einem Tourenplan T zu G ohne Mehrfacheinsätze, dessen Touren eine maximale Stopplänge l_{max} haben, müssen während einer Iteration des SmartSwaps $\mathcal{O}(n^2\cdot l_{max})$ Prüfoperationen durchgeführt werden.*

Eine Nachoptimierung mit der GTS besteht normalerweise nicht nur aus einer, sondern aus vielen Iterationen. Die meisten Gültigkeitsprüfungen einer Iteration wurden auch schon in der vorherigen Iteration durchgeführt, ohne dass sich etwas an den Faktoren, die die Gültigkeit beeinflussen können, änderte. In jeder Iteration werden maximal zwei verschiedene

Touren des Tourenplans verändert. In der nachfolgenden Iteration ist es hinreichend, nur Gültigkeitsprüfungen für die Knoten anzustoßen, die in veränderte Touren eingefügt werden sollen. Die Ergebnisse aller anderen Prüfungen können aus der vorherigen Iteration beibehalten werden. Drei Sonderfälle gibt es, die beachtet werden müssen:

1. Bei der ersten Iteration existieren noch keine Prüfergebnisse, die weiter verwendet werden können. Es müssen also alle Prüfungen durchgeführt werden.
2. Nach einer Intensivierung ändert sich die Granularitätsbedingung, und Tourenpläne, die zuvor gültig waren, können ungültig geworden sein. Hier werden auch alle Prüfungen erneut durchgeführt.
3. Analog zur Intensivierung ändert sich auch bei der Diversifizierung die Granularitätsbedingung. Ungültige Tourenpläne können gültig werden. Auch hier werden alle Tourenpläne erneut geprüft.

Es lässt sich also eine weitere Abschätzung vornehmen. t_1 und t_2 bezeichnen hier die beiden veränderten Touren aus der vorherigen Iteration.

$$\begin{aligned}
|\text{Prüfoperationen}| &= \sum_{1<a<l(t_1)} \sum_{1<b<l(t_2)} \Big(\sum_{1<c\leq l(t_1)} 1 + \sum_{1<d\leq l(t_2)} 1 \Big) \\
&< \sum_{1\leq a\leq l(t_1)} \sum_{1\leq b\leq l(t_2)} l(t_1) + l(t_2) \\
&\leq \sum_{1\leq a\leq l(t_1)} \sum_{1\leq b\leq l(t_2)} 2 \cdot l_{max} \\
&= 2 \cdot l(t_1) \cdot l(t_2) \cdot l_{max} \\
&\leq 2 \cdot l_{max}^3
\end{aligned}$$

Feststellung 11 *Zu einem gegebenen Graphen $G = (V, E)$, mit $|V| = n$ und einem beliebigen Tourenplan T zu G, dessen Touren eine maximale Stopplänge l_{max} haben, müssen während einer Iteration des SmartSwaps $\mathcal{O}(l_{max}^3)$ Prüfoperationen durchgeführt werden, wenn:*

- *es nicht die erste Iteration der Nachoptimierung ist,*
- *nach der vorherigen Iteration keine Intensivierung stattgefunden hat,*
- *nach der vorherigen Iteration keine Diversifizierung stattgefunden hat.*

Die hier grob skizzierten Möglichkeiten der Beschleunigung führen auch in der Praxis zu einer erheblichen Verringerung der Laufzeit. Exemplarisch ist die Auswirkung auf die Laufzeit in dem Diagramm 3.10 aufgetragen. Sie liegt ca. bei einem Faktor 20. Es wurde die Testinstanz 1 aus der Tabelle 4.1 verwendet.

3.3.7 Anforderungen an den SmartSwap

Der SmartSwap musste in ein bereits vorhandenes Framework eingefügt werden. Das führte zu einer Reihe von Einschränkungen bei dem Entwurf und der Implementation. Folgende Punkte waren fest vorgegeben:

- Ein Zugriff auf die Elemente der Tabuliste war nicht möglich.
- Es konnte nicht überprüft werden, ob das Aspirationskriterium erfüllt ist.

Somit war es nötig, die Nachbarschaft für den SmartSwap sehr groß werden zu lassen. Es war nicht ausreichend, dass sie nur die Tourenpläne enthält, die entstehen, wenn beide zu vertauschenden Knoten an die beste Position in der neuen Tour eingefügt werden, sondern

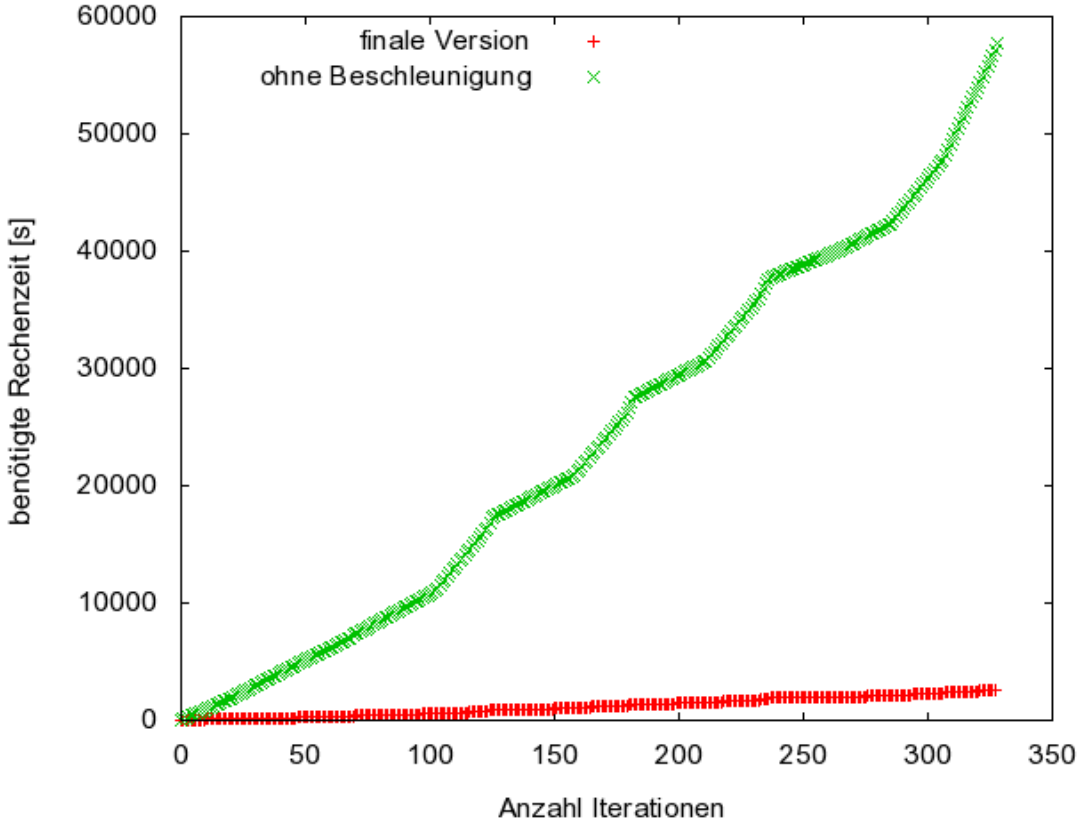


Abbildung 3.10: Vergleich der Laufzeit des SmartSwaps vor und nach der Beschleunigung.

sie musste für alle möglichen Positionen die entstehenden Tourenpläne enthalten. Wenn der Tourenplan mit der besten Positionierung der Knoten nicht ausgewählt werden darf, weil er in Konflikt mit der Tabuliste steht, dann sollen nacheinander die nächstbesten Positionierungen geprüft werden. Dies ist nur möglich, wenn die zu prüfenden Elemente auch in der Nachbarschaft vorkommen.

Zusätzlich zu der Kenntnis über die Tabuliste wäre auch noch ein Zugriff auf das Aspirationskriterium notwendig gewesen, da es ja möglich sein kann, dass ein Tourenplan, der eigentlich ausgeschlossen ist, aufgrund des Aspirationskriteriums dennoch ausgewählt wird. Es müsste also sehr viel Verwaltungsarbeit von der GTS in die Lokale Suche wandern. Dies unterwanderte den konzeptuellen Aufbau des Frameworks und erschwerte spätere Änderungen.

Es müssen aber dennoch die Überprüfungen auf Granularität und Gültigkeit von Touren vorgenommen werden. Es soll eine Filterung der Elemente der Nachbarschaft stattfinden, so dass die GTS zur weiteren Prüfung nur gültige Tourenpläne erhält, bei denen die Granularitätsbedingung erfüllt ist. Dies war im Konzept des Frameworks vorgesehen. Gerade durch den frühzeitigen Test auf Granularität kann viel Rechenzeit gespart werden, da so viele teure Prüfoperationen ausgeschlossen werden.

4. Tests und Auswertungen

Ein zentraler Bestandteil der Diplomarbeit war das Entwerfen und die Implementation von Algorithmen zur Tourenplanung. Eng damit verknüpft ist aber auch eine ausführliche Test- und Analysephase. Zum einen weisen die Algorithmen mehrere Freiheitsgrade in der Parameterwahl auf, die durch anschließende Tests möglichst gut bestimmt werden sollen. Zum anderen sollen auch Aussagen über die Qualität der neuen Algorithmen getroffen werden können. Da keine optimalen Lösungen der verwendeten Probleminstanzen vorlagen oder berechnet werden konnten, fand hier ein Vergleich mit einem kommerziellen Algorithmus der PTV statt, von dem bekannt war, dass er gute Ergebnisse liefert.

Insgesamt lassen sich die durchgeführten Tests thematisch in vier verschiedene Bereiche trennen:

1. Die beiden neu erstellten Savingsvarianten wurden untersucht (siehe Kapitel 4.3). Es wurden gute Belegungen für freie Parameter gefunden, und es fanden Vergleiche der Varianten untereinander und mit dem Savingsalgorithmus der PTV statt.
2. Im Anschluss wurde der Einfluss der Nachoptimierung auf die verschiedenen Savingsvarianten untersucht (siehe Kapitel 4.4). Es sollte untersucht werden, ob die Lösungen bestimmter Konstruktionsverfahren besonders gut geeignet sind für eine anschließende Nachoptimierung und ob hier Unterschiede zwischen den Konstruktionsverfahren bestehen.
3. Es wurde untersucht, ob sich die Ergebnisse der Nachoptimierung verbessern, wenn als zusätzliche Nachbarschaft der SmartSwap verwendet wird (siehe Kapitel 4.4).
4. Unabhängig von diesen Untersuchungen werden in dem separaten Kapitel 5 die Auswirkungen von zeitabhängigen Fahrzeiten bei der Tourenplanung untersucht. Ziel der Tests war es, den Einfluss auf die Ergebnisse zu quantifizieren und so Aussagen über die Relevanz der Zeitabhängigkeit treffen zu können.

4.1 Der Versuchsaufbau

4.1.1 Die Testinstanzen

Alle Testdurchläufe wurden auf 43 verschiedenen Testinstanzen durchgeführt. Da es sich um vertrauliche Daten seitens der PTV handelt, können diese Instanzen leider nicht vollständig veröffentlicht werden und sollen im weiteren Verlauf dieser Diplomarbeit lediglich als *Test1* ... *Test43* bezeichnet werden.

4.1.1.1 Eigenschaften der Testinstanzen

In der Tabelle 4.1 sind für jede Testinstanz die Anzahl der Knoten und der zur Verfügung stehenden Fahrzeuge angegeben. Weitere exakte Eigenschaften dürfen leider nicht angegeben werden. Es handelt sich jedoch um Planungsdaten, die fast das gesamte Straßennetz Deutschlands umfassen (siehe Abbildung 4.1.1.1).

4.1.1.2 Die verwendete Kosten- und Prüffunktion

Bestandteil eines jeden VRPs ist eine zugehörige Kostenfunktion K und eine Prüffunktion D (siehe Kapitel 4).

Seien T_1 und T_2 zwei Tourenpläne. Die verplanten Stopps werden mit s_1 und s_2, die benötigten Fahrzeuge mit v_1 und v_2 und die benötigte Zeit mit t_1 und t_2 bezeichnet. Dann ist die Kostenfunktion durch folgende Punkte fest vorgegeben:

- Aus $s_1 > s_2$ folgt $K(T_2) > K(T_1)$
- Aus $s_1 = s_2$ und $v_1 < v_2$ folgt $K(T_2) > K(T_1)$
- Aus $s_1 = s_2$, $v_1 = v_2$ und $t_1 < t_2$ folgt $K(T_2) > K(T_1)$

Die Prüffunktion D kann für jede Probleminstanz verschieden sein und wurde als reine Black-Box-Funktion aufgefasst. Sie ist äußerst komplex und berücksichtigt je nach Probleminstanz eine Vielzahl von Nebenbedingungen. Hier soll nur ein kleiner Ausschnitt der möglichen Nebenbedingungen angeben werden:

- Eine individuelle Maximaltourdauer darf nicht überschritten werden.
- Die gesetzlich vorgeschriebenen Lenk- und Ruhezeiten müssen eingehalten werden (siehe UNION (2006)).
- Zeitfenster müssen eingehalten werden.
- Kapazitäten der Fahrzeuge dürfen nicht überschritten werden.
- Es muss eine gültige Verteilung der Fahrzeuge auf die vorhandenen Touren existieren.
- Mehrfacheinsätze der Fahrzeuge können erlaubt oder verboten sein.
- Es kann Paare von Stopps geben, die in einer vorgegebenen Reihenfolge besucht werden müssen.

4.1.2 Die getesteten Algorithmen

Folgende Algorithmen wurden in den Testdurchläufen beachtet:

- der Savingsalgorithmus der PTV (siehe Kapitel 3.2.4)
- die Tabusuche der PTV *ohne* dem SmartSwap (siehe Kapitel 3.3.3)
- die Tabusuche der PTV *mit* dem SmartSwap (siehe Kapitel 3.3.3)
- beide Varianten des Savingsalgorithmus (siehe Kapitel 3.2.6.4 und 3.2.6.5)
- das Nearest-Neighbor-Verfahren (siehe Kapitel 3.2.7)

Die beiden getesteten Savingsvarianten werden im weiteren Verlauf der Arbeit als *Variante 1* und *Variante 2* bezeichnet. Hierbei bezeichne Variante 1 den Algorithmus, in dessen Verlauf die berechneten Savings zufällig modifiziert werden (siehe Kapitel 3.2.6.4), und Variante 2 den Algorithmus, der zufällig eines der obersten k Savings wählt (siehe Kapitel 3.2.6.5).

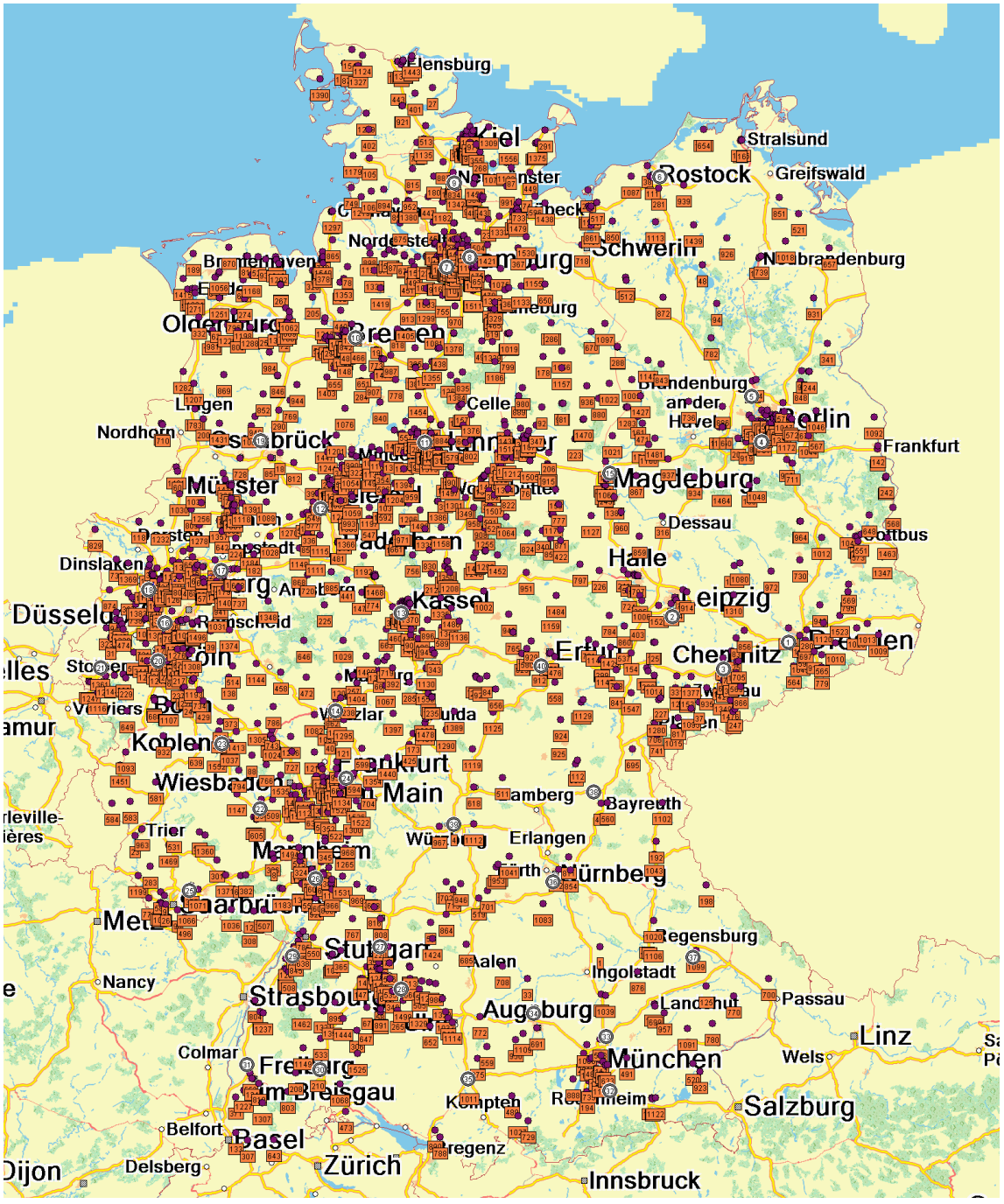


Abbildung 4.1: Die Knoten der verwendeten Testinstanzen.

Testinstanz	Knotenanzahl	Fahrzeuganzahl
1	223	44
2	128	22
3	104	44
4	40	3
5	37	5
6	34	5
7	57	3
8	20	2
9	36	3
10	138	4
11	64	5
12	77	2
13	109	3
14	137	5
15	81	3
16	74	5
17	35	4
18	61	4
19	34	3
20	54	5
21	42	5
22	26	5
23	33	3
24	15	5
25	15	2
26	22	3
27	47	2
28	29	5
29	31	5
30	25	3
31	23	2
32	20	8
33	14	3
34	13	3
35	17	3
36	12	2
37	9	2
38	9	2
39	12	9
40	13	3
41	9	2
42	10	4
43	25	2

Tabelle 4.1: Alle 43 getesteten Instanzen mit der zugehörigen Anzahl Knoten und Fahrzeuge.

4.1.2.1 Die Parameter

Für die Algorithmen der PTV existiert bereits eine Standardbelegung der Parameter, die durch interne Tests gewählt wurde. Für alle Tests wurde diese Belegung verwendet. Bei dem Nearest-Neighbor-Verfahren gibt es keine frei zu wählenden Parameter. Nur bei den beiden Savingsvarianten können folgende Parameter variiert werden:

- die f- und g-Parameter (siehe Kapitel 3.2.4)
- die beiden Parameter iteration und rnd (siehe Kapitel 4 und 6)

Um die gleichnamigen Parameter beider Savingsvarianten unterscheiden zu können, wird folgende Notation mit Indizes verwendet: f_1, f_2, iteration_1, rnd_2, ...

4.1.3 Bestehende Zeitrestriktionen

Die getesteten Algorithmen wurden ausschließlich auf Unterschiede bezüglich ihrer Ergebnisqualität untersucht. Die benötigte Rechenzeit war wenig relevant. Es war lediglich gefordert, dass die Konstruktionsverfahren zum Finden einer Lösung für eine durchschnittliche Probleminstanz wenig Zeit (maximal eine Minute) benötigen und die Nachoptimierung nie länger als 15 Minuten rechnet.

Die Angaben sind nicht als scharfe Schranken zu verstehen, da alle Kunden potentiell unterschiedliche Hardware zur Verfügung haben, und auch die Rechenzeit von den indiviuellen Nebenbedingungen und Restriktionen der Kunden beeinflusst wird. Sie drücken lediglich die Wünsche und Erwartungen der Kunden aus, denen die Verfahren zu entsprechen haben.

Alle im Rahmen dieser Diplomarbeit entwickelten Algorithmen entsprechen diesen Vorgaben. Bei allen Tests haben die Konstruktionsverfahren weniger als eine Minute Rechenzeit benötigt, die Nachoptimierung wurde beim Überschreiten der 15-Minuten-Marke abgebrochen.

4.1.4 Vergleichbarkeit von Lösungen

Aufgrund der dreigestaffelten Kostenfunktion ist es nicht so leicht möglich, die Qualität von Lösungen zu vergleichen. Aussagen wie „Lösung A ist 5% besser als Lösung B.“ können nicht gemacht werden. Um dennoch Lösungen hinsichtlich ihrer Qualität vergleichen zu können, wurde der Begriff der *Toleranz* eingeführt. Zwei Lösungen sollen als „gleich gut“ gelten, wenn sie sich *nur* in ihrer benötigten Zeit unterscheiden und dieser Unterschied innerhalb des Toleranzbereichs liegt. Liegt die Toleranz bei 0, werden nur Lösungen als gleich gewertet, die auch exakt die gleiche Zeit benötigen, bei einem Wert von 0.05 darf die Zeit um 5% voneinander abweichen und bei einer Toleranz von 1 hat die benötigte Zeit keinen Einfluss.

Motivation für die Verwendung der Toleranz war die folgende Überlegung: Wird keine Toleranz (oder eine Toleranz von 0) verwendet, ist es möglich, dass eine bestimmte Parameterkonfiguration in vielen Fällen das beste Ergebnis liefert und daher als sehr gut angesehen wird. Eine andere Parameterkonfiguration liefert in diesen Fällen minimal schlechtere Ergebnisse, ist aber in vielen anderen Fällen noch sehr erfolgreich. In so einem Fall ist es schwierig zu berücksichtigen, dass auch die zweite Parameterkonfiguration ein aussichtsreicher Kandidat sein könnte.

4.2 Beste f- und g-Parameter

Sowohl Variante 1 als auch Variante 2 benutzen die modifizierte Formel

$$\text{saving}_{uv} = c(v, v_{depot}) + c(v_{depot}, u) - f \cdot c(v, u) + g \cdot |(c(v, v_{depot}) - c(v_{depot}, u))|$$

zur Berechnung der Savingswerte. Für die Tests wurde $f \in [0.5, 1.5]$ und $g \in [-0.5, 0.5]$ gewählt. Wenn man f und g jeweils in 0.1-er Schritten erhöht ergeben sich dadurch 121 zu testende Kombinationen. Da es aus mangelnder Rechenkapazität nicht möglich war, alle nachfolgenden Tests um einen Faktor 121 zu verlangsamen, wurden die 121 Kombinationen nur mit dem Standard-Savingsalgorithmus getestet unter der Annahme, dass sich die Ergebnisse auf die modifizierten Varianten übertragen lassen.

Leider ließ sich so keine beste Konfiguration finden und auch keine *kleine* Menge von Konfigurationen, so dass immer mindestens eine Konfiguration bei jeder Testinstanz zu einer sehr guten oder optimalen Lösung führt. Daraufhin wurde untersucht, ob Korrelationen zwischen gewissen Eigenschaften der Probleminstanzen und der besten Parameterkonfiguration bestehen. Folgende Eigenschaften wurden untersucht:

- durchschnittlicher Abstand der Knoten zum Depot
- maximaler Abstand der Knoten zum Depot
- durchschnittlicher Abstand der Knoten untereinander
- Anzahl der verfügbaren Fahrzeuge
- Anzahl der Knoten
- das Verhältnis der Anzahl Knoten zur Anzahl verfügbarer Fahrzeuge

Leider konnte keinerlei Korrelation festgestellt werden. Die erstellten Diagramme befinden sich im Anhang (siehe Diagramm A.1). Daher wurde beschlossen, die Parameterkonfiguration der PTV zu verwenden und keine Zeit mehr für weitere Tests mit anderen Eigenschaften zu investieren. Für alle weiteren Tests gilt: $f_1 = f_2 = 1.1$ und $g_1 = g_2 = 0.2$.

4.3 Die zwei Savingsvarianten

4.3.1 Der Iterationsparameter

Ursprünglich sollte auch untersucht werden, wie viele Iterationen bei beiden Varianten sinnvollerweise ausgeführt werden sollten, ob es Parameterkonfigurationen gibt, bei denen wenig Iterationen ausreichen, oder ob keine Unterschiede festzustellen sind und ob sich beide Varianten darin unterscheiden. Aufgrund mangelnder Rechenkapazität musste leider auf diese Tests verzichtet werden. Bei allen folgenden Tests wurde immer mit genau 20 Iterationen gearbeitet, es gilt also $\text{iteration}_1 = \text{iteration}_2 = 20$. Dieser Wert erschien mir sinnvoll, da meiner bisherigen Erfahrung nach eine Erhöhung der Iterationszahl über 20 zu keiner weiteren Verbesserung führt, aber 20 Iterationen immer noch in akzeptablem Zeitaufwand bearbeitet werden können (kürzer als eine Minute pro Testinstanz).

4.3.2 Der Zufallsparameter für Variante 1

Es sollte bestimmt werden, welche Belegung des Parameters rnd_1 bei der Variante 1 des Savingsalgorithmus zu den besten Ergebnissen führt. Dazu wurden alle 43 Testinstanzen mit folgenden Belegungen für rnd_1 getestet: 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048. Aufgrund mangelnder Rechenkapazität konnte jeder Durchlauf nur sieben Mal wiederholt werden. Insgesamt wurden für jede Parameterkonfiguration demnach 301 Testergebnisse gesammelt.

In der Tabelle 4.2 ist angegeben , wie oft ein bestimmter rnd_1-Parameter, abhängig von einer gewählten Toleranz, zu der besten, überhaupt gefundenen Lösung führte. Der beste Wert ist hervorgehoben.

Es ist deutlich zu erkennen, dass der Parameter $\text{rnd}_1 = 32$ die besten Ergebnisse liefert, unabhängig davon welche Toleranz gewählt wurde. Größere Toleranzwerte als 5% wurden

Toleranz	4	8	16	32	64	128	256	512	1024	2048
0	11	25	49	**62**	51	26	18	13	15	4
0.01	68	96	116	**124**	110	90	79	72	68	59
0.02	92	106	125	**131**	123	107	93	91	90	83
0.03	102	116	136	**142**	136	122	105	101	98	89
0.04	125	136	150	**154**	152	138	129	128	124	116
0.05	140	144	155	**163**	162	148	135	135	133	126

Tabelle 4.2: Anzahl bester Ergebnisse, die mit einer bestimmten Parameterkonfiguration bei Variante 1 des Savingsalgorithmus gefunden wurden.

Toleranz	0	0.01	0.02	0.03	0.04	0.05
∅ Parameteranzahl	1.3	4.1	5.1	5.6	6.5	7.1

Tabelle 4.3: Durchschnittliche Anzahl unterschiedlicher Parameter, die zur der besten Lösung führen.

nicht betrachtet, da Ergebnisse mit solch starken Abweichungen intuitiv nicht mehr als gleichwertig angesehen werden.

Es ist möglich, dass verschiedene Parameterbelegungen zu gleich guten Lösungen führen. An den steigenden Werten in Tabelle 4.2 ist zu erkennen, dass dies bei steigender Toleranz immer häufiger der Fall ist. Genauer dargestellt ist dies noch einmal in Tabelle 4.3. In Abhängigkeit von einer Toleranz ist angegeben, wie viele verschiedene Parameterbelegungen durchschnittlich zu der besten Lösung führen.

Eine weitere Korrelation lässt sich aber auch mit der Knotenanzahl der einzelnen Probleminstanzen erkennen. Bei kleinen Probleminstanzen führen sehr viele Parameterbelegungen zu guten Lösungen, während es bei größer werdenden Instanzen immer weniger Parameter werden. Bei kleinen Probleminstanzen liegen die besten Parameterbelegungen der sieben unabhängig voneinander durchgeführten Testläufe auch nahe zusammen. Besonders deutlich ist dies bei den Probleminstanzen 1,2,7,10 und 11 zu erkennen. Bei allen sieben unabhängigen Testdurchläufen gibt es auch bei einer Toleranz von 5% durchschnittlich weniger als 1.5 verschiedene Parameterkonfigurationen, die zu der besten Lösung führen. Alle diese Testinstanzen weisen mehr als 100 Knoten auf.

4.3.3 Der Zufallsparameter für Variante 2

Es sollte bestimmt werden, welche Belegung des Parameters rnd_2 bei der Variante 2 des Savingsalgorithmus zu den besten Ergebnissen führt. Dazu wurden alle 43 Testinstanzen mit folgenden Belegungen für rnd_2 getestet: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Man beachte, dass für $\text{rnd}_2 = 1$ der Standard-Savingsalgorithmus ausgeführt wird, der nicht abhängig von Zufallszahlen ist. Aufgrund mangelnder Rechenkapazität konnte jeder Durchlauf nur sieben Mal wiederholt werden. Insgesamt wurden für jede Parameterkonfiguration demnach 301 Testergebnisse gesammelt.

In der Tabelle 4.4 ist angegeben, wie oft ein bestimmter rnd_2-Parameter, abhängig von einer gewählten Toleranz, zu der besten, überhaupt gefundenen Lösung führte. Der beste Wert ist hervorgehoben.

Es ist deutlich zu erkennen, dass der Parameter $\text{rnd}_2 = 5$ die besten Ergebnisse liefert, unabhängig davon welche Toleranz gewählt wurde. Größere Toleranzwerte als 5% wurden nicht betrachtet, da Ergebnisse mit solch starken Abweichungen intuitiv nicht mehr als gleichwertig angesehen werden.

Toleranz	1	2	3	4	5	6	7	8	9	10
0	11	37	36	37	**39**	**39**	26	21	20	13
0.01	50	104	118	**119**	**119**	115	93	85	85	68
0.02	74	120	127	134	**135**	134	120	111	112	98
0.03	83	131	138	149	**153**	150	141	137	128	116
0.04	108	147	151	155	**168**	163	151	154	149	136
0.05	119	156	161	166	**172**	169	162	163	157	148

Tabelle 4.4: Anzahl bester Ergebnisse, die mit einer bestimmten Parameterkonfiguration bei Variante 2 des Savingsalgorithmus gefunden wurden.

Toleranz	0	0.01	0.02	0.03	0.04	0.05
∅ Parameteranzahl	1.4	4.7	5.8	6.6	7.3	7.5

Tabelle 4.5: Durchschnittliche Anzahl unterschiedlicher Parameter, die zur der besten Lösung führen.

Es ist möglich, dass verschiedene Parameterbelegungen zu gleich guten Lösungen führen. An den steigenden Werten in Tabelle 4.4 ist zu erkennen, dass dies bei steigender Toleranz immer häufiger der Fall ist. Genauer dargestellt ist dies noch einmal in Tabelle 4.5. In Abhängigkeit von einer Toleranz ist angegeben, wie viele verschiedene Parameterbelegungen durchschnittlich zu der besten Lösung führen.

Eine weitere Korrelation lässt sich aber auch mit der Knotenanzahl der einzelnen Probleminstanzen erkennen. Bei kleinen Probleminstanzen führen sehr viele Parameterbelegungen zu guten Lösungen, während es bei größer werdenden Instanzen immer weniger Parameter werden. Bei kleinen Probleminstanzen liegen die besten Parameterbelegungen der sieben, unabhängig voneinander durchgeführten Testläufe auch nahe zusammen. Besonders deutlich ist dies bei den Probleminstanzen 7 und 11 zu erkennen. Bei allen sieben unabhängigen Testdurchläufen gibt es auch bei einer Toleranz von 5% durchschnittlich weniger als 1.5 verschiedene Parameterkonfigurationen, die zu der besten Lösung führen. Bei den Testinstanzen 9 und 10 gibt es durchschnittlich höchstens zwei Parameterkonfigurationen, die zu der besten Lösung führen. Alle diese Testinstanzen weisen mehr als 100 Knoten auf.

4.3.4 Vergleich der beiden Varianten untereinander

Es wurde untersucht, welche der beiden Savingsvarianten potentiell in der Lage ist, bessere Ergebnisse zu liefern. Dazu wurde bei jeder Testinstanz die beste Lösung beider Varianten miteinander verglichen, die mit einem beliebigen Parameter erzielt werden konnte. Die Parameter rnd_1 und rnd_2 waren also *nicht* fest vorgegeben, sondern es wurden alle Belegungen aus dem vorherigen Abschnitt verwendet. Getestet wurden 41 Probleminstanzen mit jeweils sieben unabhängigen Durchläufen. Zwei Probleminstanzen konnten aus technischen Problemen nicht berücksichtigt werden.

In der Tabelle 4.6 ist abhängig von einem Toleranzwert aufgetragen, wie oft die beiden Varianten gleich gute oder qualitativ unterschiedliche Lösungen geliefert haben.

Beide Varianten scheinen sich hinsichtlich der erzielten Ergebnisqualität nicht groß zu unterscheiden. Beide Verfahren wurden demnach als gleichberechtigt angesehen und in allen weiteren Tests beide verwendet.

In einem weiteren Test sollte untersucht werden, ob sich diese Ergebnisse ändern, wenn $\text{rnd}_1 = 32$ und $\text{rnd}_2 = 5$ fest gewählt werden. Das sind die Belegungen aus dem vorherigen

Toleranz	Variante 1 besser	Variante 2 besser	gleich gut
0	116	97	74
0.01	40	38	209
0.02	29	34	224
0.03	23	28	236
0.05	12	24	251
1	12	24	251

Tabelle 4.6: Vergleich der Anzahl der besten Ergebnisse, die mit den beiden Savingsvarianten erzielt werden können.

Toleranz	Variante 1 besser	Variante 2 besser	gleich gut
0	110	19	172
0.01	47	19	235
0.02	36	19	246
0.03	29	19	253
0.04	25	19	257
0.05	22	19	260

Tabelle 4.7: Vergleich der Anzahl der besten Ergebnisse der beiden Savingsvarianten mit $\text{rnd}_1 = 32$ und $\text{rnd}_2 = 5$.

Abschnitt, die die besten Testergebnisse geliefert haben. Es wurden erneut 41 Testinstanzen mit sieben unabhängigen Durchläufen getestet. Verglichen wurden in jedem Durchlauf aufgrund der festen Parameterwahl nur zwei Ergebnisse miteinander.

In der Tabelle 4.7 ist abhängig von einem Toleranzwert aufgetragen, wie oft die beiden Varianten gleich gute oder qualitativ unterschiedliche Lösungen geliefert haben.

Auch hier scheinen beide Varianten ähnlich gute Ergebnisse zu liefern, wenn man eine Toleranz von mindestens 1% zulässt. Erstaunlich ist, dass bei einer Toleranz von 0 die Variante 1 klar überlegen scheint. Hier handelt es sich aber nur um sehr kleine zeitliche Abweichungen. Beide Verfahren liefern also auch bei festem rnd_1 und rnd_2 vergleichbar gute Ergebnisse. Variante 2 scheint aber leicht führend in der Ergebnisqualität zu sein.

In allen weiteren Tests, in denen die beiden Varianten des Savingsalgorithmus verwendet werden, gilt ab hier $\text{rnd}_1 = 32$ und $\text{rnd}_2 = 5$.

4.3.5 Vergleich mit dem Savingsalgorithmus der PTV

Um Aussagen über die Lösungsqualität der beiden Savingsvarianten treffen zu können, wurden ihre Ergebnisse mit denen des bisher verwendeten Savingsalgorithmus der PTV verglichen (siehe Kapitel 3.2.4). Dieser Algorithmus wird kommerziell vertrieben und liefert nach Aussagen der PTV gute Ergebnisse.

Diese Aussage bestätigte sich auch in einem Vergleich mit dem Nearest-Neighbor-Verfahren (siehe Kapitel 3.2.7). In allen nachfolgenden Tests lieferte dieses Verfahren nie bessere Ergebnisse als der Algorithmus der PTV. Bei einem Test auf 42 Testinstanzen hat es unabhängig von einer gewählten Toleranz in 28 Fällen gleich gute Ergebnisse, in 14 Fällen jedoch schlechtere Ergebnisse geliefert. Dies ist ein Indikator dafür, dass das Nearest-Neighbor-Verfahren entweder sehr schlechte Ergebnisse liefert, oder das Verfahren der PTV gut funktioniert.

Auf 42 Testinstanzen wurde geprüft, wie oft die neuen Savingsvarianten ein besseres Ergebnis als der Algorithmus der PTV liefern. In der Tabelle 4.8 ist dargestellt, wie oft

Toleranz	Variante 1 besser	gleich gut	Variante 1 schlechter
0	32	9	1
0.01	25	16	1
0.02	19	22	1
0.03	15	26	1
0.04	13	28	1
0.05	8	33	1

Tabelle 4.8: Vergleich der Variante 1 des Savingsalgorithmus mit dem Savingsalgorithmus der PTV.

Toleranz	Variante 2 besser	gleich gut	Variante 2 schlechter
0	33	8	1
0.01	24	17	1
0.02	22	19	1
0.03	14	27	1
0.04	11	30	1
0.05	11	30	1

Tabelle 4.9: Vergleich der Variante 2 des Savingsalgorithmus mit dem Savingsalgorithmus der PTV.

Variante 1 ein besseres, schlechteres oder gleich gutes Ergebnis liefert, in der Tabelle 4.9 sind die Ergebnisse für die zweite Variante aufgetragen.

Zuletzt wurde noch getestet, ob sich die beiden Savingsvarianten mit ihren Ergebnissen ergänzen, ob es Fälle gibt, in denen die eine Variante bessere Ergebnisse, während die andere schlechtere Ergebnisse als der Savingsalgorithmus der PTV liefert. Dazu wurde jeweils das bessere Ergebnis der beiden Varianten mit dem Ergebnis des Savingsalgorithmus der PTV verglichen. Die Ergebnisse sind in Tabelle 4.10 dargestellt.

Beide Varianten liefern durchweg bessere Ergebnisse als der PTV-Savingsalgorithmus. Es war zu erwarten, dass dieser Unterschied bei steigender Toleranz kleiner wird, aber dennoch können in ca. 85% der Fälle Verbesserungen beobachtet werden, in ca. einem Viertel der Fälle liegen sie über 5% bei der benötigten Zeit, oder es konnte sogar ein Fahrzeug eingespart oder ein Stopp mehr besucht werden. Auffällig ist, dass im direkten Vergleich mit dem PTV-Savingsalgorithmus Variante 2 leicht besser als Variante 1 ist, obwohl im direkten Vergleich der beiden Varianten untereinander ein konträres Ergebnis zu beobachten ist. Auch lässt sich nicht feststellen, dass die Verwendung des besten Ergebnisses beider Verfahren zu einer großen Steigerung der Ergebnisqualität führt. Allenfalls bei sehr klein

Toleranz	Variante 1 oder 2 besser	gleich gut	Variante 1 und 2 schlechter
0	36	5	1
0.01	30	11	1
0.02	22	19	1
0.03	19	22	1
0.04	15	26	1
0.05	11	30	1

Tabelle 4.10: Vergleich der beiden Varianten des Savingsalgorithmus mit dem Savingsalgorithmus der PTV.

Instanz	max Knotenanzahl	Knoten der Varianten	Knoten des PTV-Algorithmus
2	128	125	124
3	104	87	101
12	77	45	41
13	109	77	70
14	137	132	108
18	61	61	60

Tabelle 4.11: Probleminstanzen, bei denen die Savingsvarianten Lösungen geliefert haben, die sich in ihrer Knotenanzahl von den Lösungen des Savingsalgorithmus der PTV unterscheiden.

Instanz	max Fahrzeuganzahl	Fzge der Varianten	Fzge des PTV-Algorithmus
2	22	21	20
3	44	5	7
20	5	3	4

Tabelle 4.12: Probleminstanzen, bei denen die Savingsvarianten Lösungen geliefert haben, die sich in ihrer Fahrzeuganzahl von den Lösungen des Savingsalgorithmus der PTV unterscheiden.

gewählter Toleranz werden so mehr Ergebnisse gefunden, die besser als das Ergebnis des Savingsalgorithmus der PTV sind.

Genauer betrachtet wurden die Unterschiede der beiden Savingsvarianten im Vergleich mit dem Savingsalgorithmus der PTV, die eine größere Abweichung als 5% bei der benötigten Zeit aufweisen. Bei den elf Verbesserungen und der einen Verschlechterung aus Tabelle 4.10 konnten sieben Fälle festgestellt werden, bei denen sich nicht nur die Fahrzeit verändert hat. In sechs dieser Fälle kam es zu einer Veränderung der Anzahl besuchter Knoten, in drei Fällen zu einer Veränderung der Anzahl benötigter Fahrzeuge. Diese Veränderungen sind in den Tabellen 4.11 und 4.12 dargestellt.

Es ist zu erkennen, dass es durchaus zu starken Unterschieden bei den verschiedenen Lösungen kommen kann. Als maximaler Wert wurde ein Unterschied von 24 verplanten Knoten gemessen. Bei sieben verschiedenen Testinstanzen konnte eine signifikante Verbesserung der Ergebnisqualität durch die Verwendung der beiden Savingsvarianten erzielt werden, die nicht nur die benötigte Zeit betrifft. Lediglich bei der Testinstanz 3 wurde eine Verschlechterung der Ergebnisqualität durch die Verwendung der Varianten beobachtet. Dieser Fall ist verblüffend und es konnte keine Erklärung gefunden werden. Es ließen sich keine besonderen Eigenschaften der Testinstanz identifizieren, die zu diesem stark abweichenden Ergebnis führten. Allerdings ist dies auch die einzige Testinstanz, bei der überhaupt eine Verschlechterung im Vergleich mit den neuen Varianten festgestellt werden konnte, sie kann daher als Sonderfall angesehen werden.

Als weiterer Punkt wurde untersucht, in welchem Maße es bei den vier unterschiedlichen Konstruktionsverfahren zu „Ausreißern “ bezüglich der Ergebnisqualität kommt. Die bisherigen Experimente zeigten, dass die beiden Savingsvarianten meist bessere Ergebnisse als der Savingsalgorithmus der PTV und damit auch bessere Ergebnisse als das Nearest-Neighbor-Verfahren liefern. Jetzt soll gezeigt werden, dass die Ergebnisse nicht nur quantitativ besser sind, sondern dass auch in den Fällen, in denen nicht das beste Ergebnis gefunden wird, man „nicht weit davon entfernt “ ist.

Dazu wurden bei allen Testinstanzen die Lösungen aller vier Verfahren miteinander vergli-

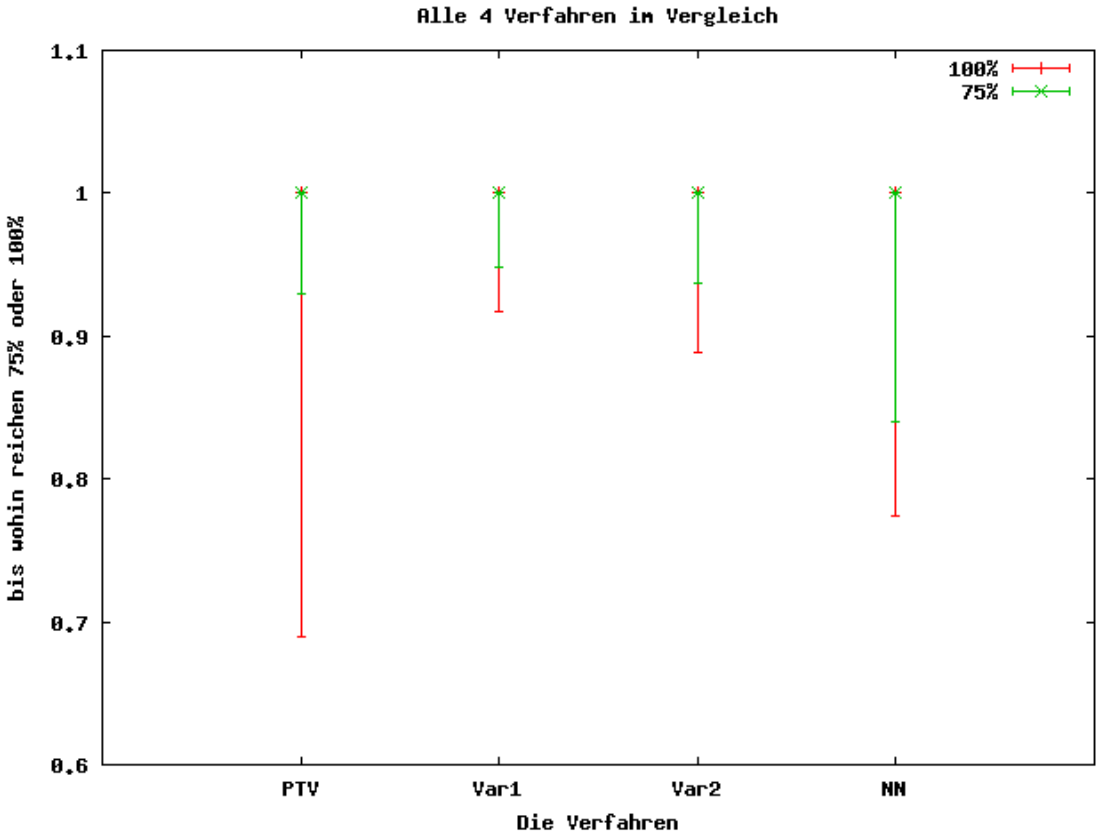


Abbildung 4.2: Vergleich der Konstruktionsverfahren hinsichtlich der Abweichung der benötigten Zeit ihrer Ergebnisse von der minimal benötigten Zeit, die durch eines der Verfahren erreicht wurde. „PTV“ steht für den Savingsalgorithmus der PTV, „Var1“ für die erste Variante des Savingsalgorithmus, „Var2“ für die zweite Variante und „NN“ für das Nearest-Neighbor-Verfahren.

chen. Die jeweils beste Lösung wurde als Referenzwert verwendet, für die anderen Lösungen wurde berechnet, wie weit sie relativ von der besten Lösung abweichen. Da eine hierarchische Kostenfunktion vorliegt, wurde sowohl die beste Lösung hinsichtlich der Knoten, der Fahrzeuge und der Zeit betrachtet. Aufgetragen sind die Ergebnisse in den Diagrammen 4.2, 4.3, 4.4 und 4.5.

Es ist zu erkennen, dass auch die eventuell auftretenden Abweichungen von der besten gefundenen Lösung bei den zwei Savingsvarianten am kleinsten ist. Bei den anderen beiden Verfahren treten bei allen drei Kriterien weitaus stärkere Abweichungen auf.

Abschließend lässt sich festhalten, dass beide Savingsvarianten eine Verbesserung des bisher verwendeten Verfahrens darstellen. In vielen Fällen konnte die Lösung des Savingsalgorithmus der PTV verbessert werden, die Lösungsqualität ist stabiler. Es kommt bei ihrer Verwendung nicht zu so großen Abweichungen von der besten bekannten Lösung. Der zeitliche Mehraufwand war bei den getesteten Probleminstanzen aus Anwendersicht zu vernachlässigen. Bei weitaus größeren Probleminstanzen wären hier weitere Analysen und Abwägungen nötig.

4.3.6 Untersuchung auf Robustheit

Bisher wurden alle Tests auf Realweltdaten durchgeführt. Zu jeder Testinstanz gehörte auch immer eine zugehörige Menge von Restriktionen und Nebenbedingungen. Der folgende Test soll einen Eindruck vermitteln, wie robust sich die vier verschiedenen Algorithmen

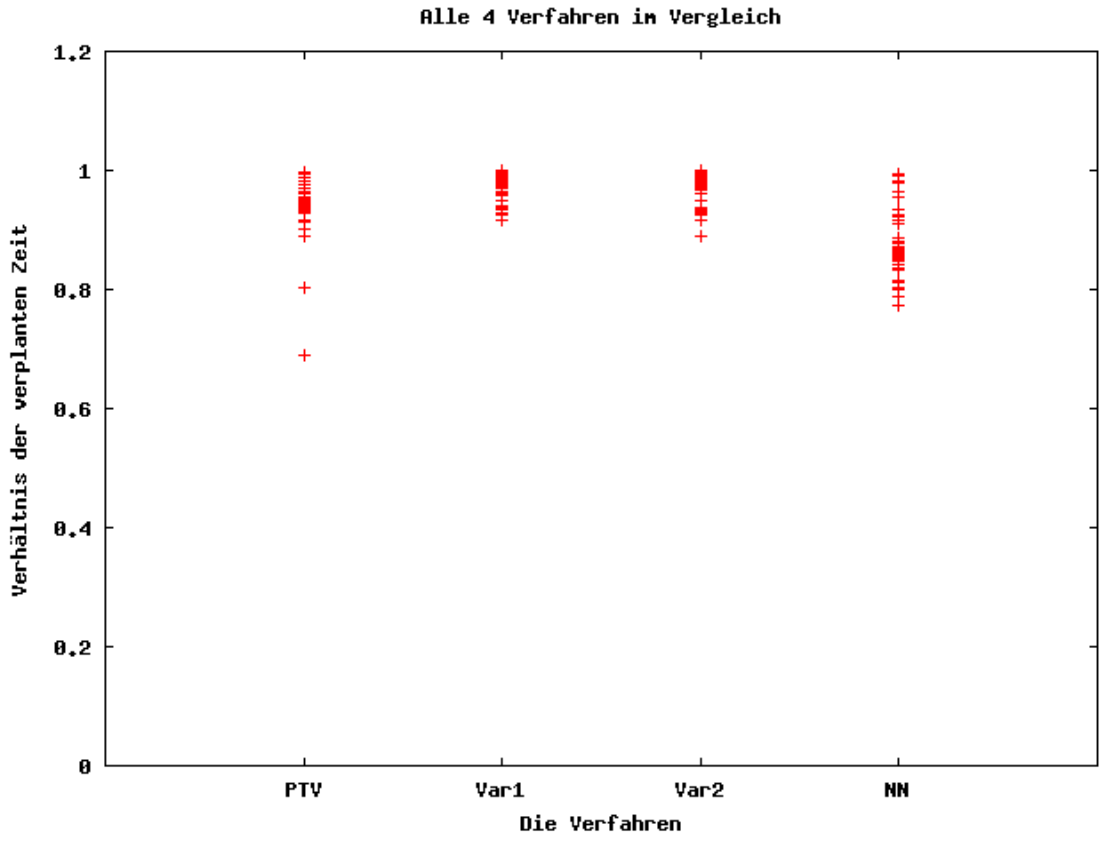


Abbildung 4.3: Vergleich der Konstruktionsverfahren hinsichtlich der Abweichung der benötigten Zeit ihrer Ergebnisse von der minimal benötigten Zeit, die durch eines der Verfahren erreicht wurde. „PTV“ steht für den Savingsalgorithmus der PTV, „Var1“ für die erste Variante des Savingsalgorithmus, „Var2“ für die zweite Variante und „NN“ für das Nearest-Neighbor-Verfahren.

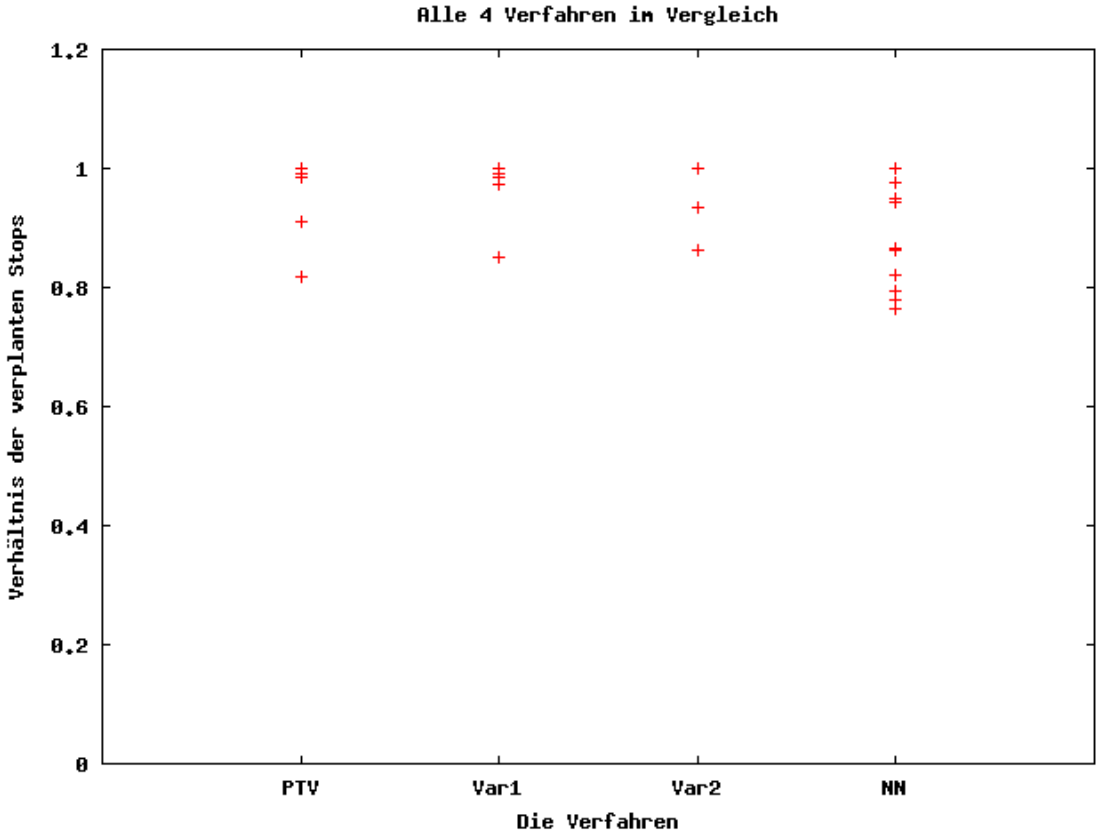


Abbildung 4.4: Vergleich der Konstruktionsverfahren hinsichtlich des Verhältnisses ihrer verplanten Knoten zu der maximalen Anzahl verplanter Knoten. „PTV“ steht für den Savingsalgorithmus der PTV, „Var1“ für die erste Variante des Savingsalgorithmus, „Var2“ für die zweite Variante und „NN“ für das Nearest-Neighbor-Verfahren.

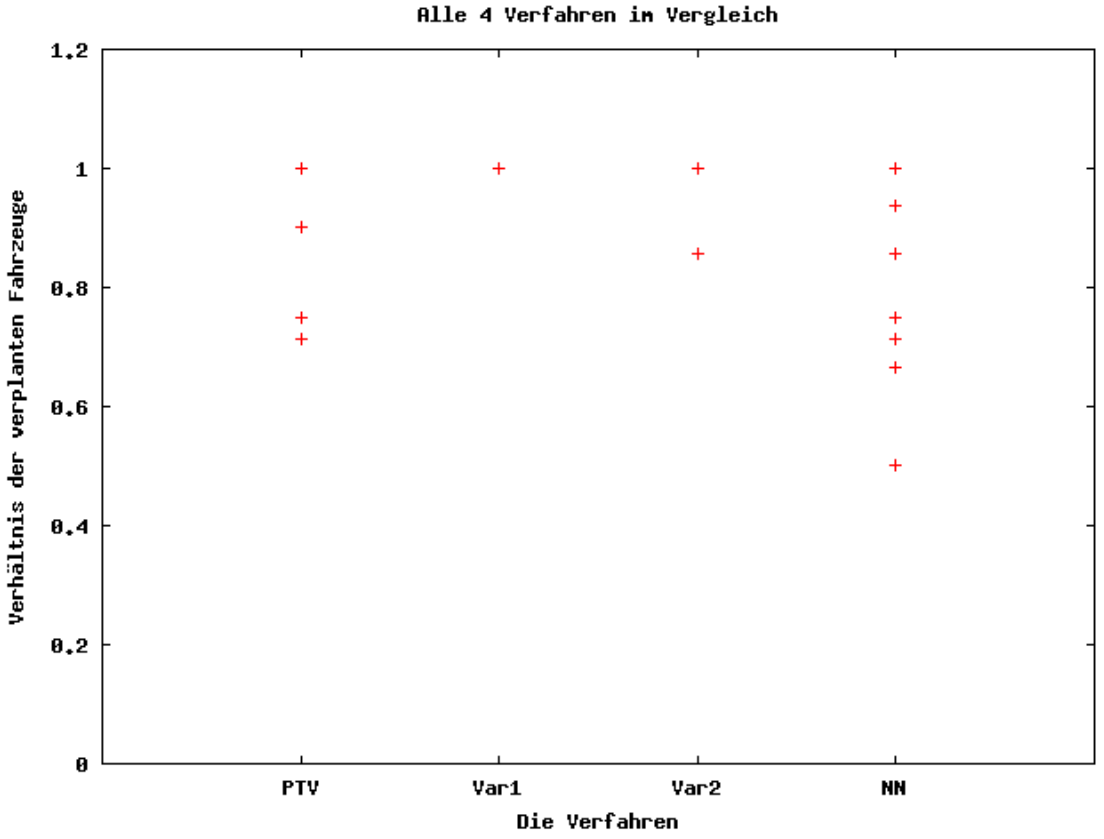


Abbildung 4.5: Vergleich der Konstruktionsverfahren hinsichtlich des Verhältnisses ihrer benötigten Fahrzeuge zu der minimalen Anzahl benötigter Fahrzeuge. „PTV“ steht für den Savingsalgorithmus der PTV, „Var1“ für die erste Variante des Savingsalgorithmus, „Var2“ für die zweite Variante und „NN“ für das Nearest-Neighbor-Verfahren.

	PTV-Saving	Variante 1	Variante 2	Nearest-Neighbor
Zeit	10	10	7	6
Fahrzeuge	1	1	1	2
Knoten	4	4	4	2

Tabelle 4.13: Auswirkungen der Deaktivierung von Nebenbedingungen auf die Qualität von Tourenplänen. Differenziert wurde zwischen Verbesserungen, die die Zeit, Fahrzeuganzahl und die beinhalteten Knoten betreffen.

gegenüber geänderten Nebenbedingungen verhalten. Es ist möglich, dass ein Verfahren besonders sensibel auf geänderte Nebenbedingungen reagiert und ein anderes kaum beeinflusst wird.

Es wurden erneut alle 43 verschiedenen Testinstanzen mit allen vier Konstruktionsverfahren getestet. Dabei wurden folgende Nebenbedingungen deaktiviert:

- die maximale Tourdauer
- die maximale Tourfahrzeit
- die maximale Wartezeit pro Tour
- die maximale Wartezeit pro Knoten
- die maximale Tourlänge
- die maximale Anzahl Knoten pro Tour

Die so erhaltenen Ergebnisse wurden untereinander und mit den Ergebnissen, die mit aktivierten Nebenbedingungen erzielt wurden, verglichen.

Es lässt sich leider kein eindeutiges Ergebnis aus den Testresultaten ablesen. Bei 13 der 43 Testinstanzen konnte bei mindestens einem Verfahren eine Verbesserung durch die Deaktivierung der Nebenbedingungen festgestellt werden. Diese Zahl mag sehr niedrig erscheinen, da die deaktivierten Nebenbedingungen sehr zentral erscheinen und man annehmen kann, dass sie viele Gültigkeitsprüfungen der Konstruktionsverfahren beeinflussen. Man könnte dies als Robustheit der Verfahren deuten, es kann aber auch sein, dass die Maximalwerte der Nebenbedingungen bei den meisten Testinstanzen so großzügig gewählt sind, dass sie keine praktische Restriktion darstellen. Dies ist von außen kaum zu beurteilen.

Auffällig ist jedoch, dass bei allen Testinstanzen mit mehr als 100 Knoten solche Verbesserungen festgestellt werden konnten. Wenn man annimmt, dass nicht genau bei diesen Instanzen die Nebenbedingungen restriktiver sind, dann sieht es so aus, als hätten die Nebenbedingungen nur bei größeren Instanzen Einfluss auf die Konstruktionsverfahren.

Bisher wurde nicht zwischen den einzelnen Konstruktionsverfahren differenziert. In der Tabelle 4.13 ist aufgeführt, wie oft die Deaktivierung der Nebenbedingungen bei den einzelnen Verfahren zu Verbesserungen des Tourenplans führte. Es ist angegeben, wie oft sich die benötigte Zeit oder die Fahrzeuganzahl verringerte und wie oft mehr Knoten in dem Tourenplan enthalten waren. Die Deaktivierung führte bis auf einen Fall nie zu einer Verschlechterung des Tourenplans. In der einen Ausnahme verschlechterte sich das Ergebnis, das mit dem Savingsalgorithmus der PTV erzielt wurde, zeitlich ganz leicht.

Man erkennt, dass sich alle Werte ähneln. Bei allen Verfahren kam es ähnlich oft zu Verbesserungen der Ergebnisqualität. Lediglich das Nearest-Neighbor-Verfahren liegt etwas zurück, obwohl seine Testergebnisse auch bei aktivierten Nebenbedingungen vergleichsweise schlecht waren.

Testinstanz	PTV-Saving	Variante 1	Variante 2	Nearest-Neighbor
1	Z, F	Z, F	Z, F	Z
2	Z, K	Z, K	Z, K	Z
3	Z	Z	Z	Z, F
4	-	-	-	-
5	-	-	-	Z, F
6	Z	-	-	-
7	-	-	-	-
8	-	-	-	-
9	-	-	-	-
10	Z, K	Z, K	Z, K	-
11	Z	Z	Z	-
12	-	-	-	-
13	Z, K	Z, K	Z, K	Z, K
14	Z, K	Z, K	Z, K	Z, K
15	V	Z	-	-
16	Z	-	-	-
17	-	-	-	-
18	Z	Z	-	-
19	-	-	-	-
20	-	-	-	-
21	-	-	-	-
22	-	-	-	-
23	-	-	-	-
24	-	-	-	-
25	-	-	-	-
26	-	-	-	-
27	-	Z	-	-
28	-	-	-	-
29	-	-	-	-
30	-	-	-	-
31	-	-	-	-
32	-	-	-	-
33	-	-	-	-
34	-	-	-	-
35	-	-	-	-
36	-	-	-	-
37	-	-	-	-
38	-	-	-	-
39	-	-	-	-
40	-	-	-	-
41	-	-	-	-
42	-	-	-	-
43	-	-	-	-

Tabelle 4.14: Für alle 43 getesteten Instanzen ist angegeben, wie sich die Ergebnisse nach dem Deaktivieren einer Auswahl von Nebenbedingungen geändert haben. „-“ steht für keine Veränderung, „Z“ für eine Verringerung der benötigten **Z**eit, „F“ für weniger benötigte **F**ahrzeuge, „K“ für mehr verplante **K**noten und „V“ für eine zeitliche **V**erschlechterung des Tourenplans.

In der Tabelle 4.14 sind die Testergebnisse aller Verfahren auf allen Testinstanzen noch einmal detaillierter aufgelistet.

Hier ist auch zu erkennen, dass die Verbesserungen der einzelnen Verfahren oft gemeinsam auftreten. Das könnte ein Hinweis darauf sein, dass es eher eine Eigenschaft der Testinstanzen ist, ob die Deaktivierung von Nebenbedingungen zu besseren Ergebnissen führt, anstatt eine besondere Eigenschaft der einzelnen Algorithmen zu sein. Andererseits ähneln sich alle getesteten Algorithmen sehr stark, es sind alles Savingsvarianten. Daher muss es nicht unbedingt verwunderlich sein, dass sie bei den gleichen Testinstanzen zu Verbesserungen führen.

Auch wenn es sich nicht klären lässt, ob die Auswirkung der Deaktivierung einiger Nebenbedingungen eher auf die besonderen Eigenschaften der Testinstanzen oder der Algorithmen zurückzuführen ist, lässt sich dennoch festhalten, dass dieser Effekt erst bei größeren Testinstanzen auftritt. Die deaktivierten Nebenbedingungen scheinen erste Auswirkungen zu haben, wenn die Algorithmen mit vielen verschiedenen Knoten arbeiten müssen.

Als wichtiger Punkt lässt sich jedoch festhalten, dass die Ergebnisqualität der Lösungsalgorithmen untereinander nicht von der Veränderung der Nebenbedingungen beeinflusst wurde. Ein Algorithmus, der bei einem Test mit aktivierten Nebenbedingungen die beste Lösung geliefert hat, hat dies auch bei deaktivierten Nebenbedingungen getan. Steht man also vor der Wahl, einen bestimmten Algorithmus zu benutzen bei einer Probleminstanz, über deren Nebenbedingungen und Restriktionen nur wenig bekannt ist, bieten sich immer noch die beiden Savingsvarianten an.

4.4 Die Nachoptimierung ohne Verwendung des SmartSwaps

Die bereits implementierte Nachoptimierung der PTV ist in der Lage, bestehende Lösungen weiter zu verbessern. Nach Aussagen der PTV soll dieses Verfahren gut funktionieren. Dies hat sich in eigenen Tests bestätigt. Bei 168 Testdurchläufen konnte die Nachoptimierung in 150 Fällen das Ergebnis noch verbessern. Die 18 Fälle, in denen keine Verbesserung gefunden werden konnte, traten nur bei sehr kleinen Testinstanzen auf. Jede dieser Testinstanzen besaß weniger als 15 Knoten und bot somit wenig Spielraum für Verbesserungen. In 20 Fällen konnte bei der Nachoptimierung die Knotenanzahl verbessert werden, in 9 Fällen die benötige Anzahl Fahrzeuge. Ansonsten traten nur Verbesserungen bei der benötigten Zeit auf.

Wichtiger als die Qualität des Nachoptimierungsverfahrens war jedoch der Einfluss der neu implementierten Konstruktionsverfahren auf die Nachoptimierung. Es ist denkbar, dass bestimmte Konstruktionsverfahren Lösungen liefern, die besonders geeignet für eine anschließende Nachoptimierung sind. Hier soll verglichen werden, ob die besten Ergebnisse, die von Konstruktionsverfahren geliefert wurden, auch nach der Nachoptimierung noch am besten sind. Oder ob es Lösungen gibt, die nicht auffällig gut sind, jedoch die Eigenschaft besitzen, dass die Nachoptimierung „gut“ auf ihnen arbeiten kann und so sehr gute endgültige Lösungen findet.

4.4.1 Der Versuchsaufbau

Es wurde die in Kapitel 3.3.2 vorgestellte Tabusuche benutzt. Als Parameterbelegung wurde die Standardvorgabe der PTV verwendet. Es gilt:

- Die *Größe der Tabuliste* beträgt 25. Es werden jeweils die ältesten Elemente verdrängt.
- Als *Nachbarschaft* wird die Vereinigung der Or1-, Or2-, Swap- und Two-Exchange-Nachbarschaft verwendet.

Toleranz	Variante 1 besser	gleich gut	Variante 1 schlechter
0	20	20	2
0.01	10	30	2
0.02	7	33	2
0.03	7	33	2
0.04	6	34	2
0.05	5	35	2

Tabelle 4.15: Vergleich der Variante 1 des Savingsalgorithmus mit anschließender Nachoptimierung mit dem Savingsalgorithmus der PTV mit anschließender Nachoptimierung.

Toleranz	Variante 2 besser	gleich gut	Variante 2 schlechter
0	21	19	2
0.01	9	31	2
0.02	8	32	2
0.03	8	32	2
0.04	6	34	2
0.05	6	34	2

Tabelle 4.16: Vergleich der Variante 2 des Savingsalgorithmus mit anschließender Nachoptimierung mit dem Savingsalgorithmus der PTV mit anschließender Nachoptimierung.

- Es gibt drei separate *Abbruchkriterien* Die Tabusuche wird beendet, wenn eines der drei Kriterien erfüllt ist.
  1. Es wurden 150 Iterationen durchgeführt.
  2. Es kam zu drei Diversifizierungen ohne Intensivierung
  3. Die Nachoptimierung dauert länger als 15 Minuten.
- Es findet eine *Diversifizierung* statt, wenn innerhalb von 20 Iterationen keine Verbesserung der Lösung gefunden wird.
- Sobald eine Verbesserung der Lösung gefunden wird, findet eine *Intensivierung* statt.
- Der Granularitätsparameter k wird zu Beginn auf $k = 25$ gesetzt.
- Bei jeder Diversifizierung wird k um 15 erhöht.
- Bei jeder Intensivierung wird k auf 25 zurückgesetzt.

4.4.2 Analyse der Ergebnisse

Analog zu Abschnitt 4.3.5 werden hier die Ergebnisse der beiden Savingsvarianten mit den Ergebnissen des Savingsalgorithmus der PTV verglichen. Vor dem Vergleich wurde jedes Ergebnis der Konstruktionsverfahren einmal mit der Tabusuche verbessert. Wieder werden die Ergebnisse abhängig von einer Toleranz dargestellt. In der Tabelle 4.15 wird Variante 1 mit dem Savingsalgorithmus der PTV verglichen, in der Tabelle 4.16 die Variante 2 und in der Tabelle 4.17 wird das jeweils bessere Ergebnis der beiden Varianten verglichen. Getestet wurden 42 Probleminstanzen. In den Tabellen ist jeweils eingetragen, wie oft der entsprechende Algorithmus bessere, gleich gute oder schlechtere Ergebnisse im Anschluss an die Nachoptimierung lieferte.

Auch im Anschluss an die Nachoptimierung sind die Ergebnisse der beiden Savingsvarianten besser als die des Savingsalgorithmus der PTV. Die Ergebnisse ähneln denen ohne anschließende Nachoptimierung. Wieder wurden die Lösungen aus Tabelle 4.17, die Abweichungen oberhalb der 5%-Toleranzgrenze aufweisen, genauer betrachtet. In der Tabelle

Toleranz	Variante 1 oder 2 besser	gleich gut	Variante 1 und 2 schlechter
0	23	18	1
0.01	12	29	1
0.02	10	31	1
0.03	10	31	1
0.04	9	32	1
0.05	8	33	1

Tabelle 4.17: Vergleich der beiden Varianten des Savingsalgorithmus mit anschließender Nachoptimierung mit dem Savingsalgorithmus der PTV mit anschließender Nachoptimierung.

Probleminstanz	Knotenanzahl	Knoten der Varianten	Knoten des PTV-Algorithmus
2	128	126	125
12	77	70	69
13	109	96	89
14	137	137	134

Tabelle 4.18: Probleminstanzen, bei denen die Savingsvarianten Lösungen lieferten, die sich in ihrer Knotenanzahl von den Lösungen des Savingsalgorithmus der PTV unterscheiden.

4.18 sind alle Testinstanzen angegeben, bei denen es zu Abweichungen bei der benötigten Knotenanzahl kam.

Es fällt auf, dass in der Tabelle 4.18 keine neuen Testinstanzen auftreten. Alle Testinstanzen waren schon in der Tabelle 4.11 vertreten. Die Unterschiede in der Knotenanzahl zwischen den neuen Savingsvarianten und dem Savingsalgorithmus der PTV verringerten sich. Die Nachoptimierung kann bereits sehr gute Lösungen nicht in dem Maße verbessern wie weniger gute Lösungen. Der Savingsalgorithmus der PTV profitiert also stärker von einer anschließenden Nachoptimierung, da sich die Ergebnisse der Konstruktionsverfahren einander annähern. Besonders gut zu erkennen ist dies bei der Testinstanz 3, dem Sonderfall, in dem der Savingsalgorithmus der PTV einmalig ein besseres Ergebnis geliefert hat als die neuen Varianten. Ohne Verwendung der Nachoptimierung konnte er 24 Knoten mehr verplanen, eine anschließende Nachoptimierung führte jedoch dazu, dass auch die neuen Varianten jeweils 101 Knoten, genau so viele wie der Algorithmus der PTV, verplanen konnten. Hier hob die Nachoptimierung den Unterschied bei der Knotenanzahl auf. Es unterschied sich nur noch die Anzahl benötigter Fahrzeuge. Der Savingsalgorithmus der PTV benötigt sieben, die beiden Varianten 20 bzw. 19 Fahrzeuge.

Ein weiterer interessanter Punkt ist, dass die Lösungen des Nearest-Neighbor-Verfahrens im Zuge der Nachoptimierung stark verbessert werden konnten. Als reines Konstruktionsverfahren lieferte es von allen getesteten Verfahren die schlechtesten Ergebnisse, insbesondere waren die Ergebnisse nie echt besser als die des Savingsalgorithmus der PTV. Nach der Nachoptimierung liefert das Nearest-Neighbor-Verfahren jedoch öfter bessere Ergebnisse als das PTV-Verfahren. Bei größer werdender Toleranz wird dieser quantitative Unterschied immer kleiner, aber nie liefert das PTV-Verfahren mehr bessere Lösungen. In der Tabelle 4.19 sind die Messergebnisse noch einmal dargestellt. Es ist angegeben, wie oft das Nearest-Neighbor-Verfahren, abhängig von einer Toleranz und einer anschließenden Nachoptimierung, eine bessere, gleich gute oder schlechtere Lösung als das PTV-Verfahren fand.

In zwei der 42 getesteten Fälle waren die Ergebnisse des Nearest-Neighbor-Verfahrens sogar

Toleranz	Nearest-Neighbor besser	gleich gut	Nearest-Neighbor schlechter
0	11	27	4
0.01	8	30	4
0.02	6	32	4
0.03	4	34	4
0.04	4	34	4
0.05	4	34	4

Tabelle 4.19: Vergleich des Nearest-Neighbor-Verfahrens mit anschließender Nachoptimierung mit dem Savingsalgorithmus der PTV mit anschließender Nachoptimierung.

besser als die der beiden Savingsvarianten. Hierbei handelte es sich jedoch um Lösungen, die sich lediglich in der benötigten Zeit unterschieden. Dieser Unterschied lag in einem Fall unter 1% und im anderen Fall unter 2%.

4.4.3 Ergebnis der Untersuchungen

Abschließend lässt sich festhalten, dass beide Varianten des Savingsalgorithmus als Konstruktionsverfahren bessere Ergebnisse liefern als der Savingsalgorithmus der PTV. Das gilt auch, wenn zusätzlich eine Nachoptimierung aller Ergebnisse stattfindet, jedoch nähert sich die Qualität der Lösungen einander an. Auffällig ist das Nearest-Neighbor-Verfahren. Als Konstruktionsverfahren hat es die Ergebnisse mit den höchsten Kosten geliefert, diese Ergebnisse scheinen im Anschluss an eine Nachoptimierung denen des PTV-Verfahrens jedoch überlegen.

Beide in dieser Diplomarbeit entwickelten Varianten eignen sich also als Konstruktionsverfahren. Auch wenn eine Nachoptimierung stattfindet, sind sie zu bevorzugen. Der erforderliche zeitliche Mehraufwand ist im Vergleich zu der benötigten Zeit für die Nachoptimierung zu vernachlässigen.

4.5 Die Nachoptimierung mit Verwendung des SmartSwaps

Ein zentraler Bestandteil der Diplomarbeit war die Implementation, Beschleunigung und Analyse des SmartSwaps. Der SmartSwap ist eine neue Nachbarschaft, die von der Tabusuche bei der Nachoptimierung genutzt werden kann. In diesem Abschnitt soll untersucht werden, ob die Verwendung des SmartSwaps bei der Nachoptimierung zu einer Verbesserung der Ergebnisse führt und wie stark etwaige Verbesserungen ausgeprägt sind.

Um einen fairen Vergleich der beiden Nachoptimierungen zu ermöglichen, wurde die verfügbare Rechenzeit in beiden Fällen auf 15 Minuten begrenzt. Die SmartSwap-Nachbarschaft ist größer als die anderen Nachbarschaften, dementsprechend steigt die benötigte Rechenzeit pro Iteration und es können nur weniger Iterationen in einem festen Zeitrahmen ausgeführt werden. In den folgenden Tests soll untersucht werden, ob sich diese Verschiebung von Rechenzeit auszahlt.

4.5.1 Versuchsaufbau

Es wurde erneut die in Kapitel 3.3.2 vorgestellte Tabusuche benutzt. Für den SmartSwap selbst gibt es keine Parameter, die variiert werden können. Er kann lediglich aktiviert oder deaktiviert werden. Bei den restlichen Parametern wurde versucht, möglichst nah an den Standardvorgaben der PTV zu bleiben.

Unverändert blieben folgende Punkte:

- Die *Größe der Tabuliste* beträgt 25. Es werden jeweils die ältesten Elemente verdrängt.
- Es findet eine *Diversifizierung* statt, wenn innerhalb von 20 Iterationen keine Verbesserung der Lösung gefunden wird.
- Sobald eine Verbesserung der Lösung gefunden wird, findet eine *Intensivierung* statt.
- Der Granularitätsparameter k wird zu Beginn auf $k = 25$ gesetzt.

Folgende Punkte wurden geringfügig angepasst:

- Als *Nachbarschaft* wird die Vereinigung der Or1-, Or2-, Swap-, Two-Exchange- und *SmartSwap*-Nachbarschaft verwendet.
- Es gibt drei separate *Abbruchkriterien* Die Tabusuche wird beendet, wenn eines der drei folgenden Kriterien erfüllt ist.
  1. Es wurden 150 Iterationen durchgeführt.
  2. Es kam zu vier (statt drei) Diversifizierungen ohne Intensivierung
  3. Die Nachoptimierung dauert länger als 15 Minuten.
- Bei der ersten Diversifizierung zu Beginn des Verfahrens oder nach einer Intensivierung wird der SmartSwap aktiviert. Bei jeder weiteren Diversifizierung ohne Intensivierung wird der Granularitätsparameter k um 15 erhöht.
- Bei jeder Intensivierung wird der SmartSwap deaktiviert und der Granularitätsparameter k auf 25 zurückgesetzt.

Um Rechenzeit zu sparen, wird die SmartSwap-Nachbarschaft nicht während der gesamten Nachoptimierung verwendet. Erst bei der ersten Diversifizierung wird sie aktiviert. Eine eventuell folgende Intensivierung deaktiviert die SmartSwap-Nachbarschaft wieder bis zur nächsten Diversifizierung. Um dennoch die drei Stufen aus der ursprünglichen Nachoptimierung beibehalten zu können, wurde in diesem Fall das Abbruchkriterium so angepasst, dass vier anstatt drei Diversifizierungen ohne Intensivierung nötig sind, bevor die Tabusuche abbricht.

4.5.2 Analyse der Ergebnisse

Es wurden die Ergebnisse der Nachoptimierung *mit* SmartSwap mit denen der Nachoptimierung *ohne* SmartSwap verglichen. Getestet wurden die Ergebnisse der vier verschiedenen Konstruktionsverfahren (Variante 1, Variante 2, Savingsalgorithmus der PTV und das Nearest-Neighbor-Verfahren), angewandt auf 42 Probleminstanzen. Verglichen wurden jeweils die zwei Ergebnisse, die im Anschluss an eine Nachoptimierung gewonnen wurden, einmal ohne Verwendung des SmartSwaps und einmal mit ihm.

In der Tabelle 4.20 ist angegeben, in wie vielen Fällen die Verwendung des SmartSwaps zu einer Verbesserung oder Verschlechterung führte oder keinen Einfluss auf das Ergebnis hatte. Erneut sind die Ergebnisse abhängig von einer Toleranz aufgetragen.

In sieben Fällen konnten durch die Verwendung des SmartSwaps mehr Knoten verplant werden. Die maximale Steigerung der Knotenanzahl lag bei 5. Dagegen kam es nur in einem Fall zu einer Verminderung der Knotenanzahl um lediglich einen Knoten. Veränderungen in der Knotenanzahl wurden in den Testinstanzen 12, 13 und 14 festgestellt. Die Anzahl benötigter Fahrzeuge war nie von der Verwendung des SmartSwaps betroffen.

Es zeigt sich, dass die Verwendung des SmartSwaps zu einer leichten Verbesserung der Ergebnisse führt. Rein quantitativ werden so mehr bessere Lösungen gefunden, auch die Anzahl verplanter Knoten nimmt zu. Lediglich in einem einzigen Fall wurden weniger

Toleranz	|Verbesserungen|	Das Ergebnis blieb gleich	|Verschlechterungen|
0	23	135	10
0.01	9	155	4
0.02	7	159	2
0.05	7	159	2

Tabelle 4.20: Vergleich der Ergebnisse der Nachoptimierung mit aktiviertem und mit deaktiviertem SmartSwap.

Knoten verplant, alle negativen zeitlichen Abweichungen sind sehr gering. Jedoch ist auffallend, dass in den meisten Fällen das Ergebnis unverändert blieb, ab einer Toleranz von 2% blieben ca. 95% der Testergebnisse in ihrer Qualität unverändert. Auch bei einer Toleranz von 0% liegt dieser Wert immer noch bei über 80%. Der SmartSwap scheint also einen relativ geringen Einfluss auf die Ergebnisse der Nachoptimierung auszuüben.

4.5.3 Deutungsversuch der Ergebnisse

Die Idee für den SmartSwap als neue Nachbarschaft entstand innerhalb der PTV, als bemerkt wurde, dass bestimmte Tourenpläne durch die Nachoptimierung nicht verbessert werden konnten, obwohl durch bloßes Ansehen dieser Tourenpläne schnell offensichtliche Verbesserungen gefunden wurden. Die bestehenden Nachbarschaften schienen nicht ausreichend zu sein, um diese Probleme zu lösen.

Nun ist es so, dass jeder Tourenplan T, der in einer SmartSwap-Nachbarschaft eines Tourenplans T_{start} liegt, auch in der Or1-Nachbarschaft eines Tourenplans T^* liegt, wobei T^* Element der Or1-Nachbarschaft von T_{start} ist. Anders ausgedrückt, lässt sich jede Veränderung eines Tourenplans, die auf einem Element der SmartSwap-Nachbarschaft beruht, auch durch zwei separate Änderungen mit Elementen aus Or1-Nachbarschaften simulieren. Anstatt dass zwei Knoten in einem Schritt ihre Tourzugehörigkeit tauschen und direkt an die beste Position eingefügt werden, wird erst der eine Knoten an die beste Position versetzt und in einem weiteren Schritt der zweite Knoten.

Aber gerade bei sehr „vollen“ Tourenplänen mit sehr restringenten Nebenbedingungen ist es oft nicht möglich, eine SmartSwap-Veränderung durch zwei oder mehr separate Änderungen zu simulieren, ohne dass zwischenzeitlich ein ungültiger Tourenplan entsteht. Beispielsweise kann es zu Überschreitungen der maximalen Knotenanzahl pro Tour oder der Beladungskapazität von Fahrzeugen kommen, wenn nicht beide Knoten in einem Schritt ihre Tourzugehörigkeit tauschen.

Solche Testinstanzen standen im Rahmen dieser Diplomarbeit jedoch nicht zur Verfügung. Ein Großteil der Testinstanzen schien „harmlos“ zu sein. Der Fuhrpark sah verhältnismäßig groß aus, es wurden keine Fälle erkannt, in denen bestimmte Nebenbedingungen offensichtlich zu Situationen führen, die von der ursprünglichen Tabusuche nicht verbessert werden können.

Ich gehe davon aus, dass die relativ geringe Auswirkung des SmartSwaps auf die Ergebnisqualität auf die Wahl der Testinstanzen zurückzuführen ist, und dass durchaus andere praxisrelevante Probleminstanzen existieren, bei denen er zu weitaus größeren Verbesserungen führen kann.

5. Analyse der zeitabhängigen Kantengewichte

5.1 Motivation

Das Savingsverfahren der PTV (siehe Kapitel 3.2.4) kann zeitabhängige Information nicht sinnvoll berücksichtigen, da zu dem Berechnungszeitpunkt der Savings und auch zu dem Zeitpunkt der Verschmelzung von zwei Touren noch nicht feststeht, zu welchem genauen Zeitpunkt diese neue Verbindung befahren wird. Daher wird bei dem Savingsansatz immer der Mittelwert der zeitabhängigen Kantengewichte verwendet und das Problem so, unter Informationsverlust, wieder auf den bekannten Fall reduziert. Ein Ansatz, diesem Problem zu entgegnen, war die Implementation eines neuen Insertion-Verfahrens, das diese Information besser nutzen kann. Die Ergebnisse dieses neuen Verfahrens waren nach PTV-internen Tests jedoch schlechter.

In diesem Kapitel soll eine Erklärung gegeben werden, weshalb der Savingsalgorithmus auch bei der Verwendung von zeitabhängigen Informationen durchaus ein geeignetes Konstruktionsverfahren ist. Es wird gezeigt, dass die Reduzierung der zeitabhängigen Funktionen auf ihren Mittelwert kaum zu einem Verlust der Ergebnisqualität führt.

Um dies zu zeigen, werden die zur Verfügung stehenden zeitabhängigen Kantengewichte auf starke Schwankungen untersucht. Hier zeigt sich, dass diese Schwankungen von so geringem Ausmaß sind, dass sie die Planungsergebnisse kaum beeinflussen.

5.2 Beschaffenheit der Daten

Zu einem VRP ist auch eine Kantengewichtsfunktion $c : E \to \mathbb{R}_+$ gegeben (siehe Kapitel 4). Bei einem zeitabhängigen VRP hat sie die Form $c : E \times T \to \mathbb{R}_+$, wobei $T = \{0, \ldots, 86399\}$ die Zeit angibt. Es wird sekundengenau mit 86400 Sekunden pro Tag gerechnet. Beliebige Funktionen dieser Form können nicht verlustfrei gespeichert werden. Bei der PTV wurde folgende Näherung gewählt:

- Für jedes $e \in E$ wird eine Funktion $c_e : T^* \to \mathbb{N}$ gespeichert.
- $T^* = \{t_1, \ldots, t_{18}\} \subset T$ ist eine Menge von Stützstellen.
- $c(e,t) = c_e(t_i) + \frac{(t-t_i)}{t_{i+1}-t_i} \cdot (c_e(t_{i+1}) - c_e(t_i)))$, wobei $t_i \leq t < t_{i+1}$. Es wird zwischen den Stützstellen also linear interpoliert. Bei Randfällen mit $t < t_1$ oder $t \geq t_{18}$ wird ein t_0 bzw. ein t_{19} verwendet mit $t_0 = t_{18}$ und $t_{19} = t_1$.

Man beachte, dass lediglich sekundengenau gerechnet wird. Es werden nur ganzzahlige Werte und keine Fließkommazahlen verwendet. Die Zahl 18 als Menge der Stützstellen wurde als Kompromiss zwischen Genauigkeit der zeitabhängigen Funktion und Speicherbedarf gewählt. Hier ist zu bemerken, dass die Stützstellen *nicht* äquidistanten Abstand zueinander haben. Nach Aussagen der PTV finden starke zeitliche Schwankungen nur in einem kleinen Bereich des gesamten Planungszeitraums, nämlich zu den Hauptverkehrszeiten, statt. Hier liegen die Stützstellen enger zusammen, ansonsten, gerade zu Nachtzeiten, weit auseinander. So lässt sich eine sehr viel genauere Repräsentation der tatsächlich vorliegenden zeitabhängigen Daten erstellen, als es die bloße Anzahl an Stützstellen vermuten lässt.

Es standen zeitabhängige Daten für eine große Instanz aus dem Ruhrgebiet mit 674 Knoten zur Verfügung. Die Knoten decken fast das gesamte Ruhrgebiet ab, in einigen Bereichen treten lokal begrenzte Ballungen von Knoten auf. Die verwendeten Daten repräsentieren demnach sowohl weite Verbindungen von Knoten, die sich durch das gesamte Ruhrgebiet ziehen, als auch sehr kurze Verbindungen von nahe liegenden Knoten.

Zu beachten ist außerdem, dass zwischen Knoten und geografischen Koordinaten unterschieden werden kann. Es kann mehrere Knoten mit den gleichen Koordinaten geben. Dementsprechend können sich auch die Ergebnisse unterscheiden, je nachdem ob die Fahrzeiten für alle möglichen Paare von Knoten oder von Koordinaten untersucht werden. Beide Betrachtungen scheinen plausibel. Da sich die Ergebnisse kaum unterscheiden, werden nur die Ergebnisse der Knoten betrachtet

Als Repräsentation des zeitabhängigen Netzes wurde für jedes verfügbare Paar von unterschiedlichen Stopps (u, v) die Fahrzeit in Abhängigkeit von der Zeit betrachtet. Es wurde für jede dieser Relationen die minimale, die maximale und die durchschnittliche Fahrzeit berechnet. Diese Werte werden mit $t(u,v)_{min}$, $t(u,v)_{max}$ und $t(u,v)_{avg}$ bezeichnet. Insgesamt wurden 453602 Paare betrachtet. In dem Diagramm 5.2 ist die Verteilung der durchschnittlichen Fahrzeiten für alle Paare von Knoten zu erkennen.

5.3 Testergebnisse

In diesem Abschnitt werden die Messergebnisse vorgestellt und gedeutet. Zuerst wird gezeigt, dass kaum hohe absolute Abweichungen bei den Fahrzeiten für die einzelnen Knotenpaare vorkommen. Dann werden die relativen Abweichungen betrachtet. Auch hier kann gezeigt werden, dass es nur sehr wenige Paare mit hohen Abweichungen gibt. Ausgehend von diesen beiden Ergebnissen zeigt ein weiteres Experiment, dass die wenigen hohen, relativen Abweichungen nur bei Knotenpaaren mit sehr geringer Durchschnittsfahrzeit vorkommen, dass die dadurch entstehenden absoluten Abweichungen also gering sind. Abschließend wird eine konkrete Lösung eines VRPs mit einer zeitabhängigen Kostenfunktion auf mögliche Verbesserungen durch Verschieben der Startzeitpunkte an den Knoten untersucht. Die maximal mögliche Verbesserung liegt unter einem Prozent und ist somit sehr gering.

Zu Beginn wurde geprüft, ob es überhaupt Paare von Knoten gibt, die starke Schwankungen bei der Fahrzeit aufweisen. Dafür wurde die Differenz zwischen maximaler und minimaler Fahrzeit betrachtet. In der Tabelle 5.1 und dem Diagramm 5.3 ist aufgeführt, wie groß die Anzahl von Paaren (u, v) ist, bei denen die Differenz aus $t(u,v)_{max}$ und $t(u,v)_{min}$ einen gewissen Wert übersteigt, und welchen Prozentwert der Gesamtanzahl von Paaren das ausmacht.

Alle absoluten Abweichungen liegen unter zehn Minuten. Über die Hälfte aller Schwankungen liegen unter 30 Sekunden. Das Einsparungspotential scheint demnach nur begrenzt groß zu sein, wenn man Strecken gezielt zu einer bestimmten Uhrzeit befährt. Messungen

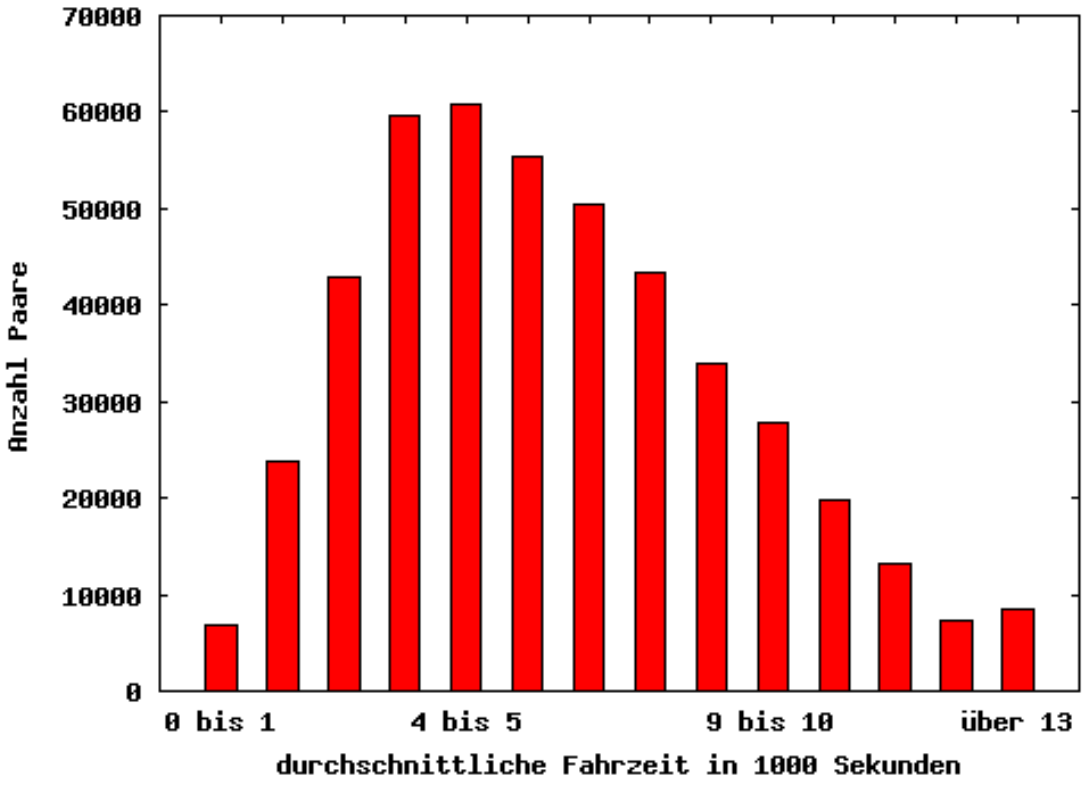


Abbildung 5.1: Häufigkeitsverteilung verschiedener Durchschnittsgeschwindigkeiten.

Zeitdifferenz in Sekunden	absolute Anzahl Paare	relative Anzahl Paare
> 30	233832	52 %
> 60	153511	34 %
> 120	66149	15 %
> 180	32469	7 %
> 300	2972	1 %
> 500	0	0 %

Tabelle 5.1: Häufigkeitsverteilung über die Abweichung $t(u,v)_{max} - t(u,v)_{min}$ für alle Knotenpaare (u,v).

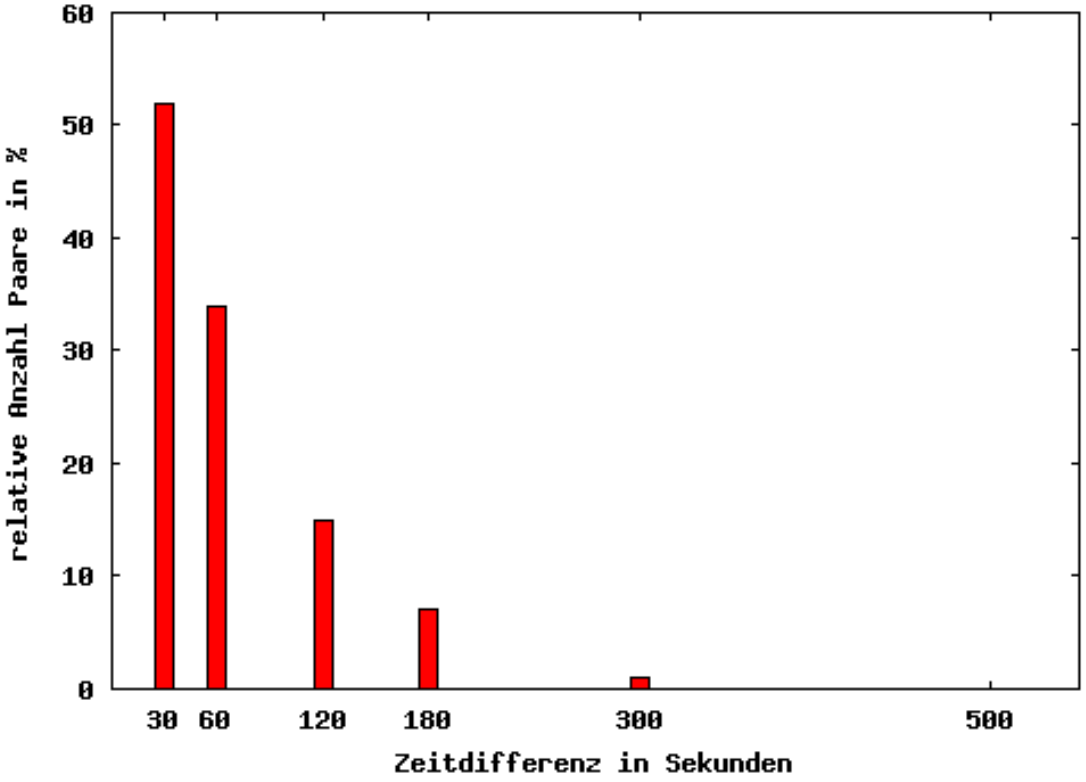


Abbildung 5.2: Häufigkeitsverteilung über die Abweichung $t(u,v)_{max} - t(u,v)_{min}$ für alle Knotenpaare (u,v).

für kleinere Werte als 30 Sekunden wurden nicht durchgeführt, da sie nicht mehr relevant erscheinen.

Absolute Werte können eventuell nur begrenzt aussagekräftig sein. Gerade bei Strecken mit nur sehr kurzer Fahrzeit können Abweichungen von bis zu zehn Minuten einen erheblichen Unterschied ausmachen. Daher wurden auch relative Abweichungen betrachtet. Es wurde gezählt, wie oft die maximale und die minimale Fahrzeit einen bestimmten Prozentsatz von der Durchschnittsfahrzeit abweicht. Die Ergebnisse sind in Tabelle 5.2 aufgelistet.

Auch hier ist zu erkennen, dass starke Abweichungen nur sehr begrenzt auftreten. Auffällig ist, dass starke Abweichungen bei der minimalen sehr viel seltener als bei der maximalen Fahrzeit auftreten. Für die maximalen Abweichungen sind die Werte in Diagramm 5.3 noch eimal grafisch dargestellt.

Auch starke relative Abweichungen können wenig über die absoluten Fahrzeitunterschiede

Zeitdifferenz in %	Abweichungen $t(u,v)_{min}$	in %	Abweichungen $t(u,v)_{max}$	in %
> 1	51240	11	156815	34
> 2	2079	0	59241	13
> 3	222	0	19153	4
> 5	22	0	429	0
> 10	0	0	3	0
> 15	0	0	0	0

Tabelle 5.2: Häufigkeitsverteilung über die relative Abweichung von $t(u,v)_{max}$ und $t(u,v)_{min}$ für alle Knotenpaare (u,v).

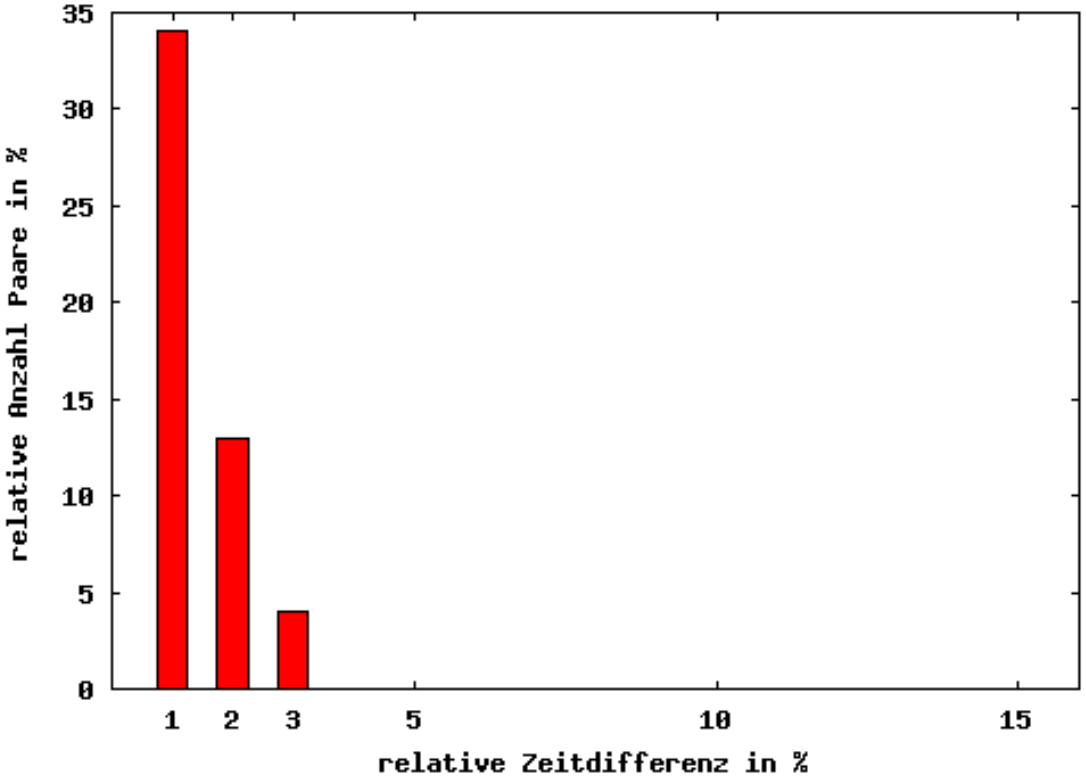


Abbildung 5.3: Häufigkeitsverteilung über die relative Abweichung von $t(u,v)_{max}$ und $t(u,v)_{min}$ für alle Knotenpaare (u,v).

aussagen. Hier sollen die Aussagen der beiden vorherigen Abschnitte zusammengebracht werden. Bei dem nächsten Experiment wurden die Knotenpaare (u,v) zusammengefasst, bei denen $t(u,v)_{max}$ oder $t(u,v)_{min}$ um einen gewissen Prozentwert von $t(u,v)_{avg}$ abweicht. Von jeder dieser Mengen wurde dann die durchschnittliche Fahrzeit bestimmt.

Aufgetragen sind die Ergebnisse in Tabelle 5.3. Die erste Zeile der Tabelle gibt mit einer Abweichung von 0 % und einer durschnittlichen Fahrzeit von 5976 Sekunden die durchschnittliche Fahrzeit *aller* Paare von Knoten an. Diese Werte sind auch in Diagramm 5.3 noch einmal dargestellt.

Man erkennt, dass hohe prozentuale Schwankungen nur bei kurzen Strecken auftreten. Die absolute Zeitdifferenz bleibt also trotz der recht hohen prozentualen Abweichung gering. Ab einem Prozent wurde aufgehört zu messen und keine noch kleineren Prozentwerte betrachtet.

Abweichungen in %	Durchschnittliche Fahrzeit in Sekunden
≥ 0	5976
> 1	4692
> 2	4046
> 3	3639
> 5	2128
> 10	907
> 15	0

Tabelle 5.3: Abhängigkeit von $t(u,v)_{avg}$ zu der relativen Abweichung von $t(u,v)_{max}$ und $t(u,v)_{avg}$ bzw. $t(u,v)_{min}$ und $t(u,v)_{avg}$.

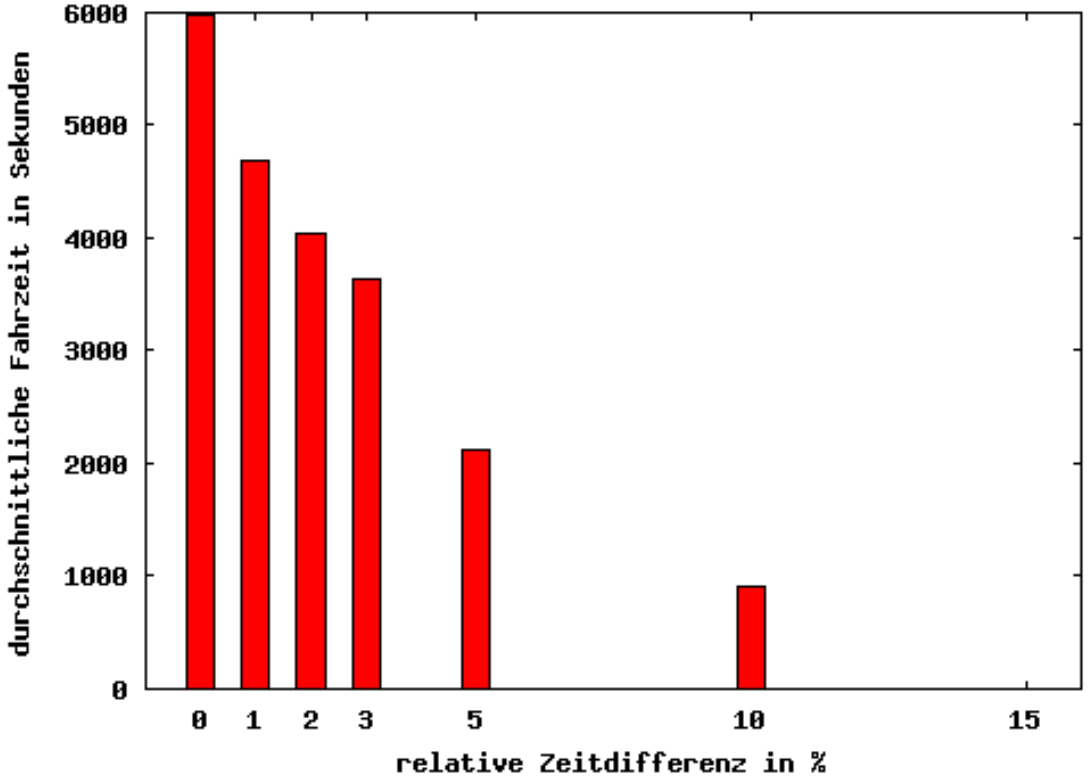


Abbildung 5.4: Abhängigkeit von $t(u,v)_{avg}$ zu der relativen Abweichung von $t(u,v)_{max}$ und $t(u,v)_{avg}$ bzw. $t(u,v)_{min}$ und $t(u,v)_{avg}$.

Zu erkennen ist dies auch in Abbildung 5.3. Für jedes Knotenpaar (u,v) wurde ein Punkt eingetragen mit den Koordinaten $((u,v)_{max} - (u,v)_{min} \mid (u,v)_{avg})$. Es wird also die Durchschnittsgeschwindigkeit mit der maximalen zeitlichen Abweichung in Verbindung gesetzt.

Bei diesem Datensatz lässt sich tatsächlich eine untere Schranke für die Abweichungen in Abhängigkeit von der Durchschnittsgeschwindigkeit angeben. Bei größer werdenden absoluten Abweichungen erhöht sich auch der Wert für die minimale Durchschnittsfahrzeit aller zugehörigen Knotenpaare. Man weiß also, dass eine hohe absolute Abweichung der Fahrzeit nur bei entsprechend langen Verbindungen auftreten kann.

In dem Diagramm 5.3 wurde für jedes Knotenpaar ein Punkt in Abhängigkeit von seinen relativen Abweichungen zur Durchschnittsfahrzeit eingetragen. Auch hier ist zu erkennen, dass es nur wenige Paare mit hohen prozentualen Abweichungen gibt, diese weisen alle eine niedrige durchschnittliche Fahrzeit auf.

5.3.1 Auswirkung der Zeitabhängigkeit auf Tourenpläne

Alle vorherigen Tests legen nahe, dass die Zeitabhängigkeit in den zugrunde liegende Daten nicht sehr ausgeprägt ist. Dennoch ist es schwer, ein Gefühl für die tatsächlich möglichen Auswirkungen der Zeitabhängigkeit zu bekommen. Hierzu soll der letzte Test beitragen.

Mit dem Savingsalgorithmus der PTV wurde ein Tourenplan erstellt. Hierbei wurden nicht die zeitabhängigen Daten verwendet, sondern es wurde immer mit den Durchschnittsfahrzeiten gerechnet. Der so entstandene Tourenplan sah rein optisch gut aus und wies keinerlei auffällige Besonderheiten auf.

Für diesen Tourenplan sollte nun getestet werden, ob Potential zur Zeiteinsparung vorhanden wäre, indem man die implizit vorgegebenen Verbindungen genauer untersucht.

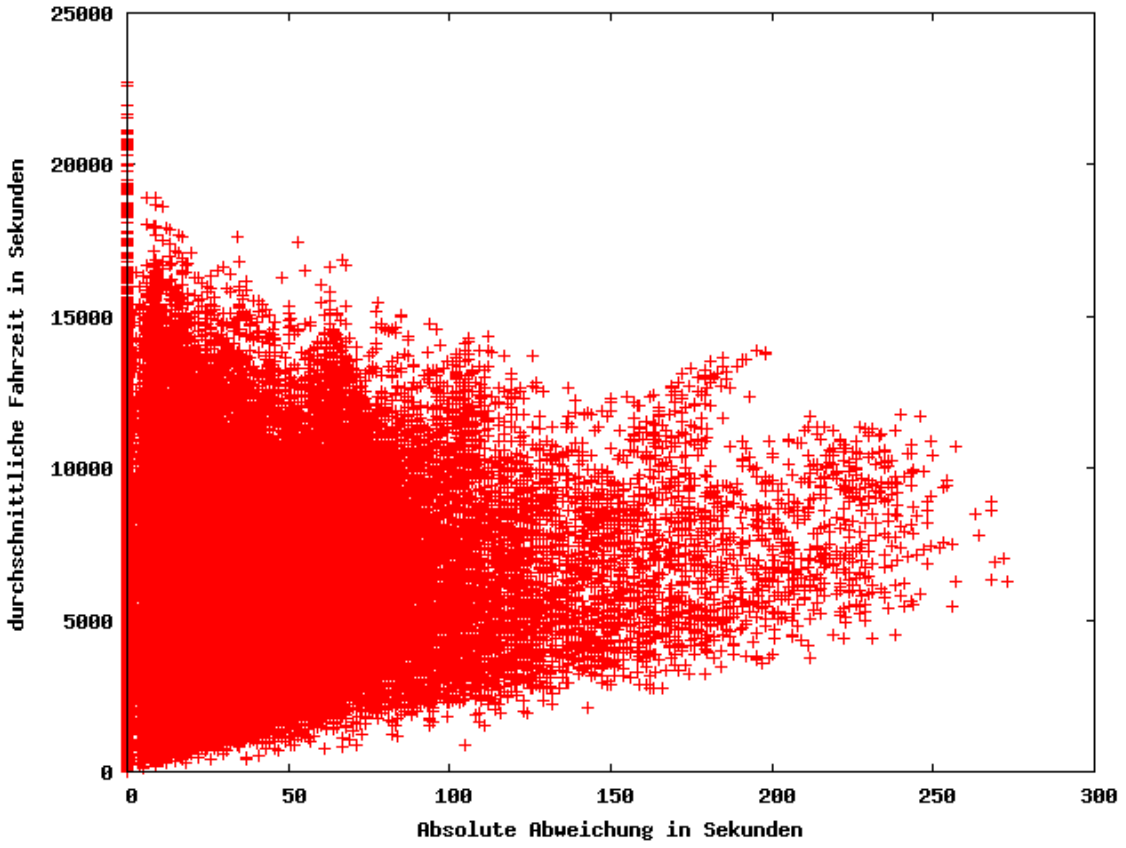


Abbildung 5.5: Für jedes Knotenpaar (u, v) ist ein Punkt markiert mit den Koordinaten $((u, v)_{max} - (u, v)_{min} \mid (u, v)_{avg})$.

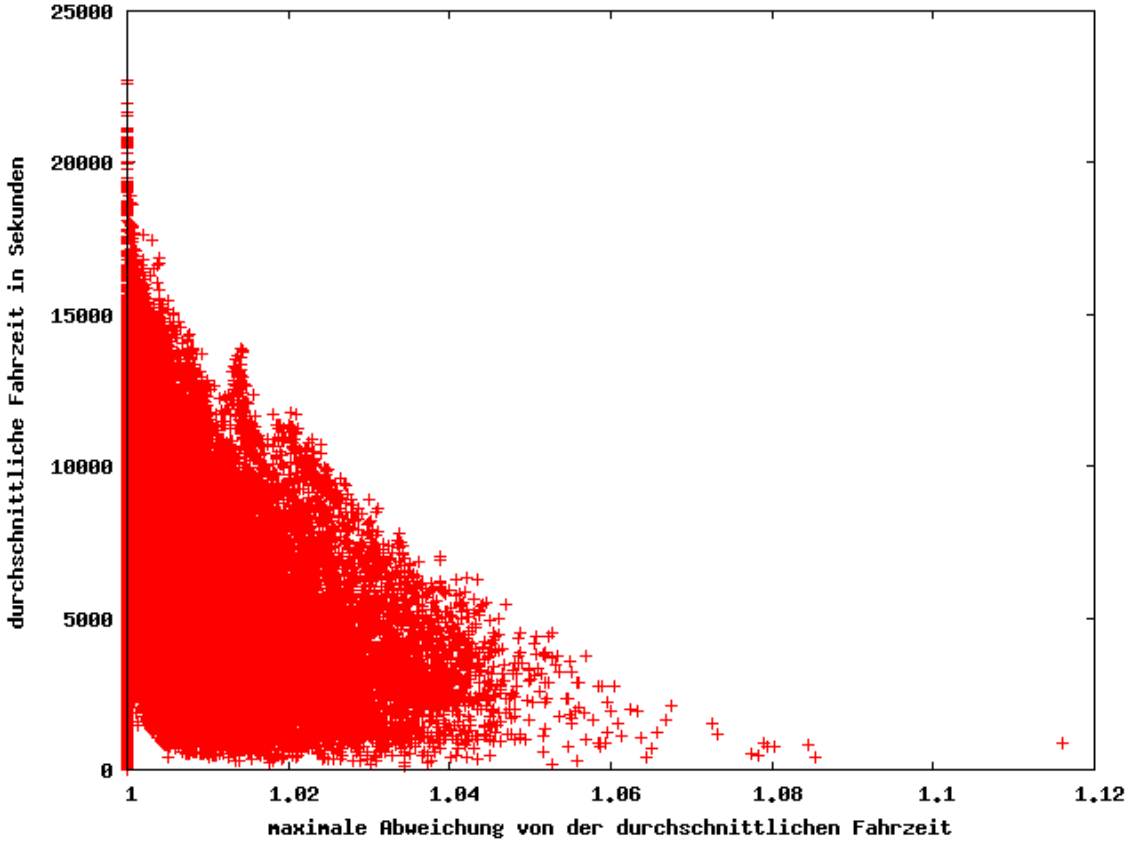


Abbildung 5.6: Für jedes Knotenpaar (u, v) ist ein Punkt markiert. Seine Koordinaten bestehen aus $(u, v)_{avg}$ und der maximalen prozentualen Abweichung von $(u, v)_{max}$ und $(u, v)_{min}$ von$(u, v)_{avg}$.

Verglichen wurde ein Tourenplan, bei dem jede Verbindung zu dem Zeitpunkt mit der höchsten Fahrzeit befahren wird, mit einem anderen Tourenplan, bei dem immer der Zeitpunkt mit der niedrigsten Fahrzeit gewählt wurde. Es wurde also eine Summe über alle Differenzen von $(u, v)_{max}$ und $(u, v)_{min}$ gebildet für alle im Tourenplan enthaltenen Verbindungen (u, v).

Natürlich müssen beide Tourenpläne nicht gültig sein, da die Startzeitpunkte an den Knoten nicht zeitlich chronologisch aufeinander folgen müssen, aber sie geben eine obere und untere Schranke für die Fahrzeit auf diesem konkreten Tourenplan an. Dieser ist auf jeden Fall gültig, da er mit einem Konstruktionsverfahren erstellt wurde, das in allen Gültigkeitsprüfungen die zeitabhängigen Fahrzeiten berücksichtigt.

Wenn man den Tourenplan mit Durchschnittsfahrzeiten durchrechnet, beträgt die Gesamtfahrzeit 416,26 Stunden. Der zeitliche Unterschied zwischen den beiden neu berechneten extremen Tourenplänen beträgt 1 Stunde, 30 Minuten und 20 Sekunden. Das ist die maximale Einsparung, die durch Verschiebung der Startzeitpunkte zu erreichen wäre. Sie liegt bei ca. 0,35 %, ist also äußerst niedrig.

Als Vergleichswert mag vielleicht die reine Beladungszeit dienlich sein. Jedes Fahrzeug muss aufgrund von Be- und Entladevorgängen an jedem Knoten eine feste Zeit warten. Diese Zeit liegt bei jedem Knoten in der verwendeten Testinstanz bei mindestens 15 Minuten. Diese Wartezeiten können durch gesetzliche Vorgaben oder dem Eintreffen vor der Öffnungszeit des jeweiligen Knoten noch beliebig verlängert werden. Als untere Schranke ergibt sich so eine minimale Wartezeit von fast 170 Stunden. Dagegen erscheinen die 1,5 Stunden Zeitunterschied, die durch zeitabhängige Fahrzeiten entstehen können, sehr gering.

5.4 Fazit

Die Experimente zeigen, dass der Einfluss der Zeitabhängigkeit beim Lösen eines VRPs nicht sehr groß ist. Es ist also durchaus gerechtfertigt, bei dem Erstellen von Tourenplänen mit der Durchschnittsfahrzeit zu rechnen, ohne große Qualitätsverluste in Kauf nehmen zu müssen. Hierbei ist jedoch zu beachten, dass die Gültigkeitsprüfungen sehr wohl mit den zeitabhängigen Daten erfolgen müssen, damit es nicht zu ungültigen Tourenplänen kommen kann.

Es sei aber noch darauf hingewiesen, dass die hier getroffenen Ergebnisse nicht beliebig zu verallgemeinern sind. Sie wurden auf Grundlage einer einzigen Probleminstanz gewonnen, deren Knoten alle im Ruhrgebiet liegen. Auch das zu Grunde liegende zeitabhängige Kartenmaterial kann Aktualisierungen unterliegen und in Zukunft eventuell stärkere zeitabhängige Eigenschaften aufweisen. Dennoch geben die gewonnenen Ergebnisse eine erste Erklärung ab, weshalb traditionelle Konstruktionsverfahren sehr gute Ergebnisse liefern, die speziell für diese Problemstellung entwickelte, neue Verfahren oft übertreffen. Ebenso können die hier erbrachten Tests als Vorlage für neue Untersuchungen an unbekanntem neuen Kartenmaterial dienen.

6. Rekapitulation und Ausblick

Zielsetzung dieser Diplomarbeit war die Analyse und Erweiterung von kommerziellen Verfahren zur Lösung von Tourenplanungsproblemen. Ziel war es, mögliche Verbesserungen der bestehenden Verfahren zu finden, ohne deren Flexibilität zu verlieren, die es ihnen erlaubt, auf sehr heterogenen Probleminstanzen zu arbeiten. Hier lässt sich eine Trennung in folgende drei Teilgebiete vornehmen:

1. Analyse und Entwicklung von Konstruktionsverfahren.
2. Betrachtung von zeitabhängigen Fahrzeiten.
3. Erweiterung und Test der Nachoptimierung.

Analyse und Entwicklung von Konstruktionsverfahren

Eine Literaturrecherche über Konstruktionsverfahren war nur bedingt hilfreich, da viele Verfahren sehr genau auf eine bestimmte Auswahl von Nebenbedingungen und Restriktionen zugeschnitten sind und somit nicht für ständig wechselnde Problemfälle genutzt werden können. Ein vielversprechender Ansatz schien jedoch zu sein, den Savingsalgorithmus so zu modifizieren, dass er mehrfach mit kleinen Veränderungen im Programmablauf ausgeführt wird, so dass viele gültige Lösungen gebildet werden, von denen die beste zurückgegeben wird. In dem Artikel von Holmes und Parker (1976) wurde ein solches deterministisches Verfahren vorgestellt. Leider konnte es nicht übernommen werden, da es von einer sehr viel geringeren Laufzeit des originalen Savingsalgorithmus ausging. Die Prüfroutinen des Savingsalgorithmus der PTV benötigen aufgrund der Komplexität der möglichen Nebenbedingungen verhältnismäßig viel Zeit. Unabhängig davon war es nicht gewünscht, dass die Laufzeit exponentiell mit wachsender Instanzgröße ansteigt.

Aber auf diese grundsätzliche Idee aufbauend, wurde versucht, eine Savingsvariante zu finden, deren Laufzeit gut steuerbar ist und die bessere Ergebnisse liefert. Beide Anforderungen konnten erfolgreich durch die Entwicklung von zwei randomisierten Algorithmen gelöst werden. Nachfolgende Tests haben Belegungen für die Tuningparameter zur Steuerung des Zufalls erbracht, die gute Ergebnisse erwarten lassen. Aus Zeitgründen konnte jedoch nicht der Tuningparameter, der die Laufzeit des Algorithmus beeinflusst, genauer untersucht werden. Aufgrund der Ergebnisse einer kleinen Testreihe wurde er auf einen Wert gesetzt, der bei allen nachfolgenden Tests nicht mehr verändert wurde.

Hier wären weitere Testläufe wünschenswert. Es wäre hilfreich zu untersuchen, in wie weit die Ergebnisqualität weiter ansteigt, wenn dem Algorithmus mehr Rechenzeit zur Verfügung gestellt wird. Vielleicht gibt es hier Unterschiede bei den verschiedenen Algorithmen.

Die Ergebnisse der bisherigen Tests zeigen, dass beide entwickelten Varianten des Savingsalgorithmus ähnlich gute Ergebnisse liefern. Hier wäre aber interessant zu wissen, ob vielleicht die eine Variante gleiche Ergebnisqualität mit weniger Laufzeit erzielen kann, oder ob bei mehr Laufzeit noch bessere Ergebnisse generiert werden können.

Ausführliche Vergleiche mit den bereits vorhandenen, kommerziellen Verfahren der PTV ergaben jedoch, dass dieser neue Ansatz sehr gut geeignet ist, um effektiv das VRP lösen zu können. Nach Aussagen der PTV liefert ein bereits implementiertes Verfahren gute Ergebnisse. Im direkten Vergleich mit diesem liefern die neuen Algorithmen jedoch qualitativ bessere Lösungen und stellen somit eine Verbesserung dar.

Betrachtung von zeitabhängigen Fahrzeiten

Seit einiger Zeit gewinnt das VRP mit zeitabhängigen Fahrzeiten immer mehr an Bedeutung. Die Berücksichtigung der Uhrzeit verspricht genauere Ergebnisse, da die Eigenschaften der realen Welt genauer modelliert werden. Dieser Ansatz bringt jedoch auch eine Reihe von Problemen bei der Anpassung bereits bestehender Algorithmen mit sich. Es ist von einem Anstieg des Rechen- und Speicherbedarfs auszugehen.

Unabhängig von der Entwicklung und Verfeinerung allgemeiner Verfahren zur Tourenplanung, wurde auch der Ansatz der Berücksichtigung von zeitabhängigen Fahrzeiten näher untersucht. Hier hat sich herausgestellt, dass die zeitabhängigen Fahrzeiten, die gegenwärtig verwendet werden, keine relevanten Schwankungen aufweisen. Es kann darauf verzichtet werden, sie bei der Tourenplanung gezielt zu berücksichtigen.

Diese Aussage darf jedoch nur als Hinweis oder Aufforderung zu mehr Analysen und Tests verstanden werden. Die bisherigen Untersuchungen haben lediglich den Charakter eines Fallbeispiels und sollen Anhaltspunkte für eigene Überlegungen liefern. Es wäre sehr interessant, wenn in diesem Bereich weitere Untersuchungen folgten, die auf Grundlage von weitaus mehr Testinstanzen eine Bewertung der Relevanz der zeitabhängigen Fahrzeiten lieferten. Zusätzlich ist es sehr wahrscheinlich, dass sich die zu Grunde liegenden Daten noch ändern werden. Es ist gut möglich, dass in Zukunft andere und vielleicht auch genauere Daten zur Verfügung stehen, die dann auch stärkere zeitliche Schwankungen aufweisen.

Erweiterung und Test der Nachoptimierung

Ein weiteres Ziel war die Erweiterung des bereits vorliegenden Frameworks zur Nachoptimierung von bestehenden Tourenplänen. Dieses Verfahren basiert auf dem Prinzip der *Lokalen Suche* und benutzt daher verschiedene Mengen, genannt *Nachbarschaften*, in denen nach besseren Lösungen gesucht wird. Es wurde festgestellt, dass bei speziellen Problemfällen die bisher genutzten Nachbarschaften nicht ausreichen, um das Ergebnis wie gewünscht zu verbessern. Um diesem Problem zu entgegnen, sollte eine neue Nachbarschaft entwickelt werden, die die gewünschten Lösungen enthält. Solch eine Nachbarschaft wurde im Rahmen dieser Diplomarbeit erstellt und in das vorhandene Framework integriert.

Da die neue Nachbarschaft sehr viele Elemente enthält und die Nachoptimierung dementsprechend länger dauerte, wurde viel Zeit darauf verwandt, Rechenzeit einsparen zu können. Dies war durch geschicktes Speichern von Ergebnissen, wodurch viele Neuberechnungen vermieden wurden, möglich.

Abschließende Tests zeigten jedoch, dass sich die Ergebnisse der Nachoptimierung unter Verwendung der neuen Nachbarschaft nur leicht verbesserten. Die Ursache hierfür konnte aus Zeitgründen nicht genauer untersucht werden. Es ist jedoch zu bedenken, dass die neue Nachbarschaft für einen speziellen Problemfall entwickelt wurde und auch nur für diesen verwendet werden soll. Es ist nicht klar, ob dieser Problemfall überhaupt in den getesteten

Probleminstanzen auftrat und mit welcher Häufigkeit. Hier wären genauere Analysen der vorhandenen Testinstanzen sinnvoll und nach Möglichkeit die Verwendung von Instanzen, bei denen bekannt ist, dass genau dieses Problem vorliegt. Es ist zu erwarten, dass dann stärkere Verbesserungen messbar sind.

Als letzter Punkt soll noch einmal darauf hingewiesen werden, dass Unterschiede zwischen einer rein akademischen Betrachtung eines Problems und der zwingenden Aufgabe, ein kommerzielles Produkt zu entwerfen, bestehen. Diese Unterschiede hatten Einfluss auf alle Bereiche dieser Diplomarbeit. Angefangen bei der Planung von Algorithmen und deren Implementierung, bis hin zu der anschließenden Test- und Vergleichsphase. Es liegt überraschend viel Arbeit in der Umsetzung eines akademischen Problems samt Lösung hin zu einem marktreifen Produkt. Gerade aufgrund der sehr viel komplexeren Problemstellungen, die sich aus Alltagsanforderungen ergeben, stellt dieser Übersetzungsvorgang einen nicht zu unterschätzenden Arbeitsaufwand dar.

A. Weitere Testergebnisse

A.1 Auswertungen der optimalen f- und g-Parameter

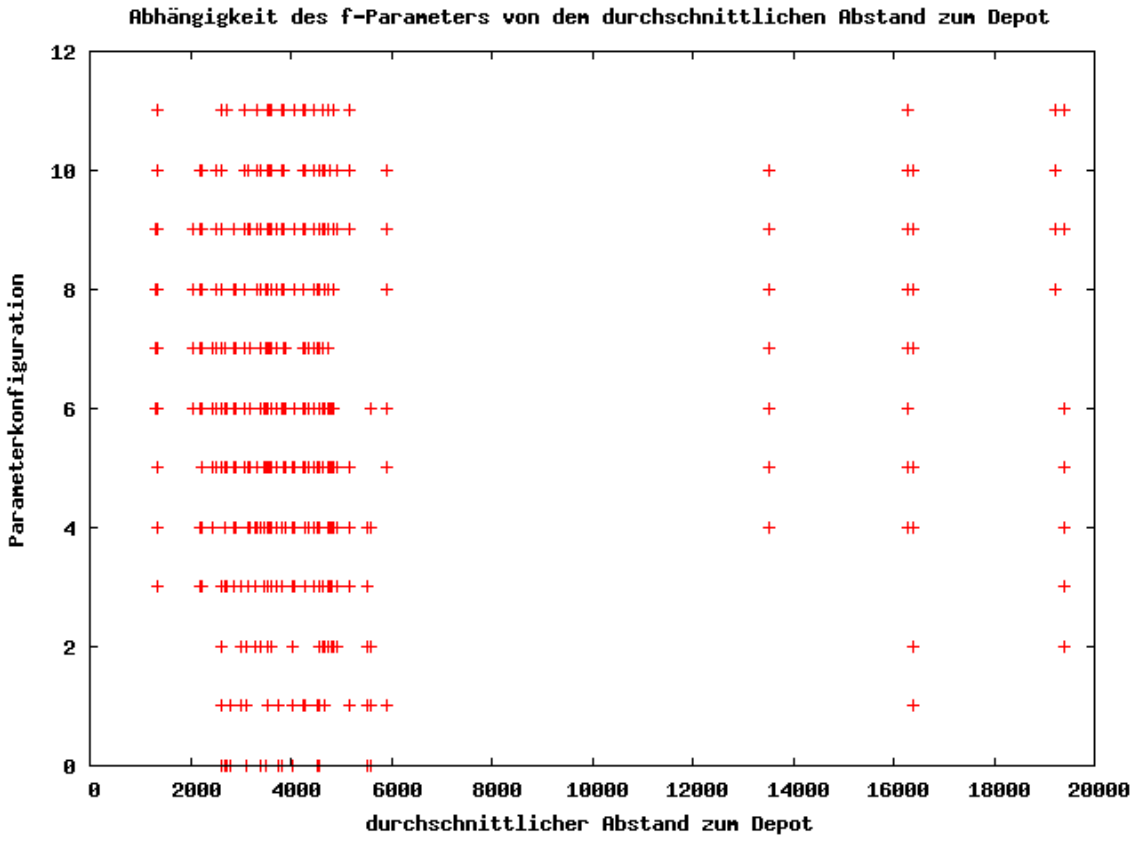


Abbildung A.1: Auf der X-Achse ist der durchschnittliche Abstand zum Depot aufgetragen. Die Y-Achse gibt den verwendeten f-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten f-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

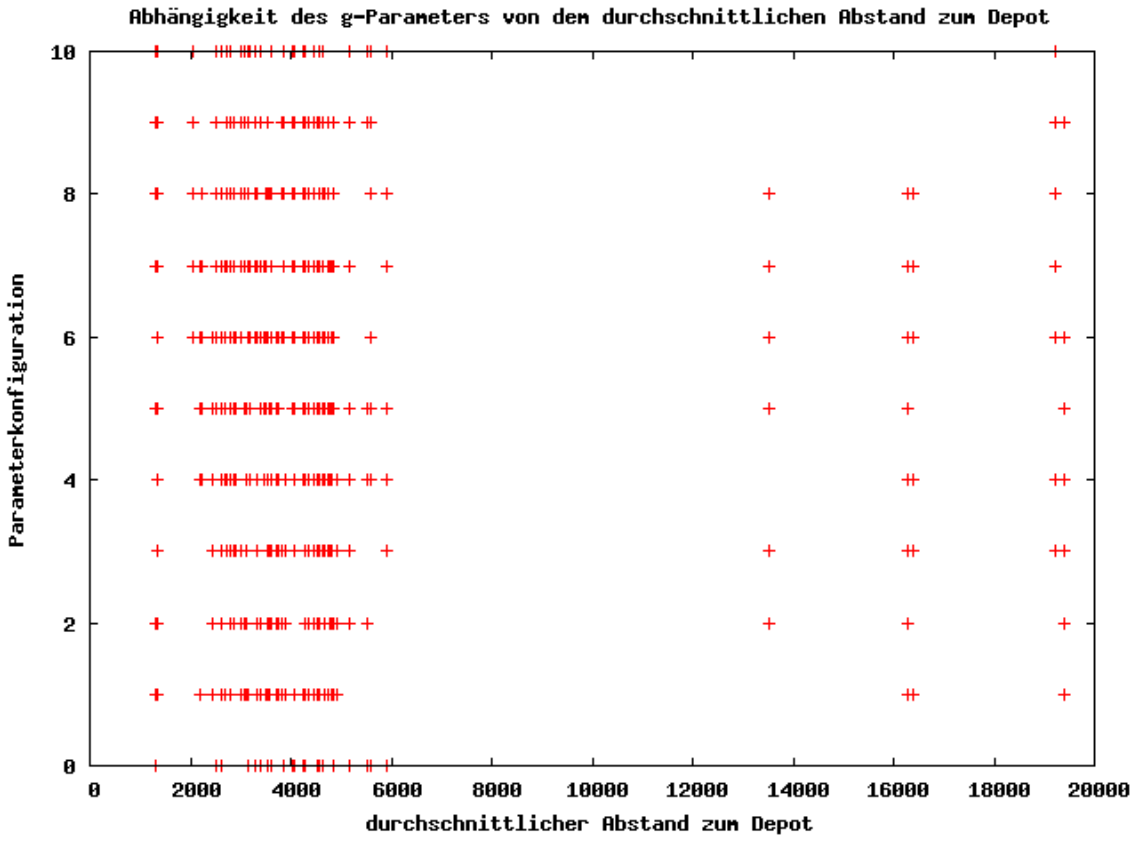


Abbildung A.2: Auf der X-Achse ist der durchschnittliche Abstand zum Depot aufgetragen. Die Y-Achse gibt den verwendeten g-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten g-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

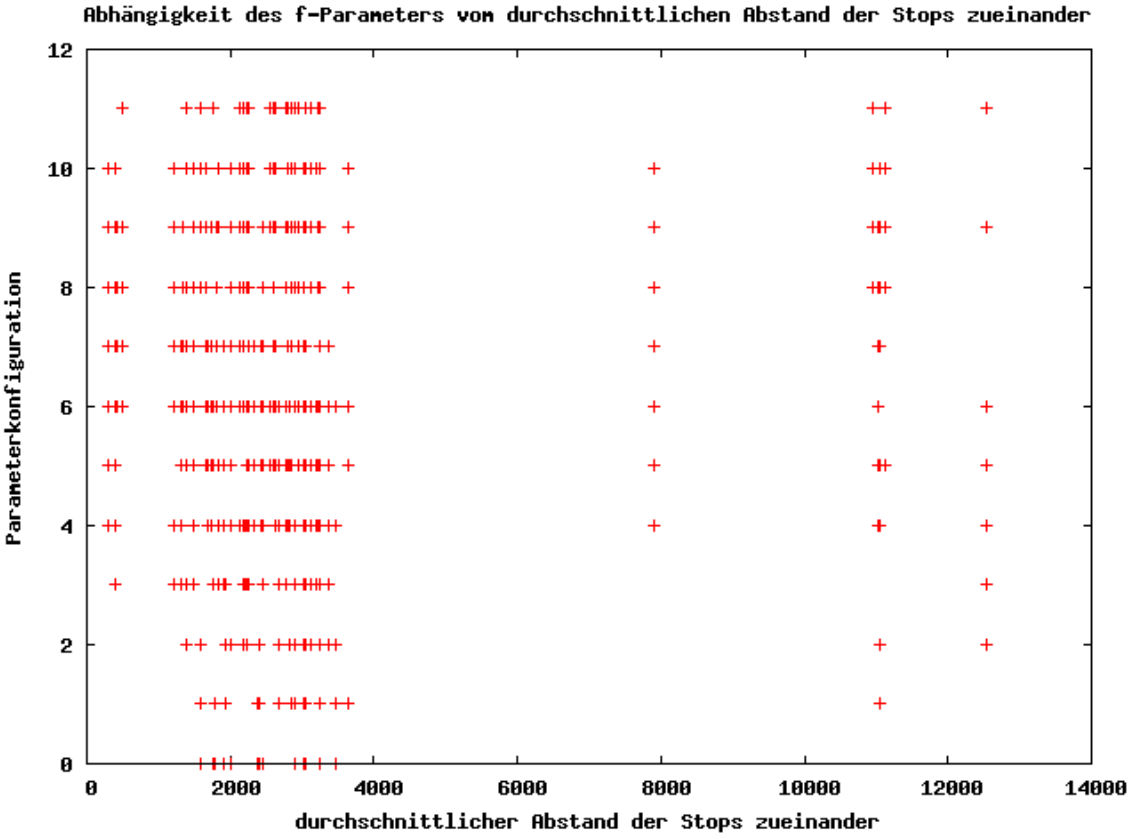


Abbildung A.3: Auf der X-Achse ist der durchschnittliche Abstand der Knoten zueinander aufgetragen. Die Y-Achse gibt den verwendeten f-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten f-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

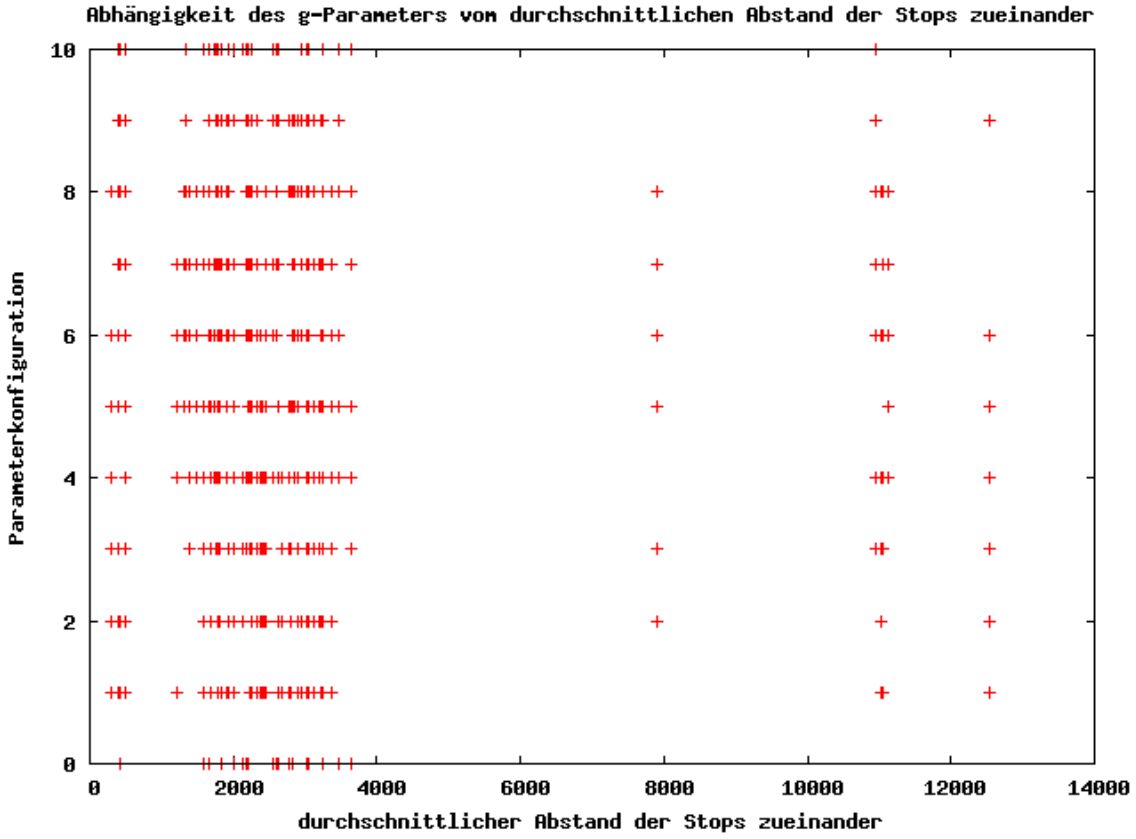


Abbildung A.4: Auf der X-Achse ist der durchschnittliche Abstand der Knoten zueinander aufgetragen. Die Y-Achse gibt den verwendeten g-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten g-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

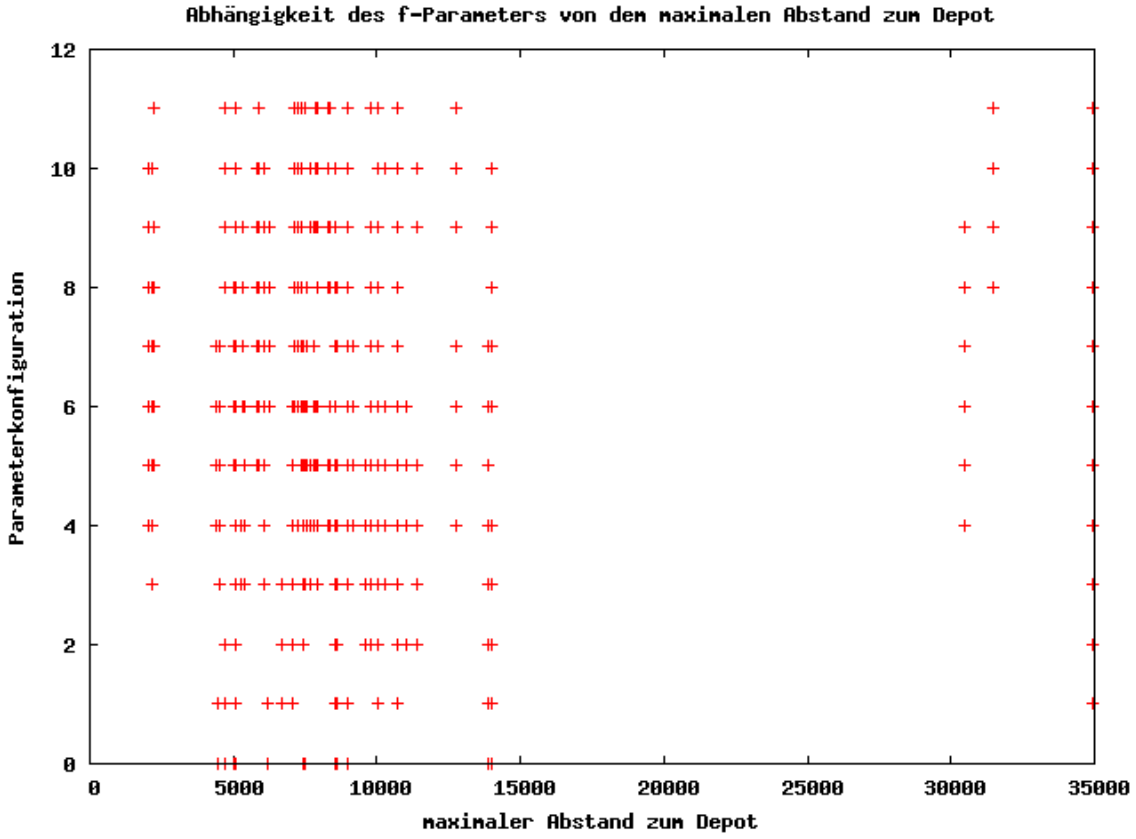


Abbildung A.5: Auf der X-Achse ist der maximale Abstand der Knoten zum Depot pro Testinstanz aufgetragen. Die Y-Achse gibt den verwendeten f-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten f-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

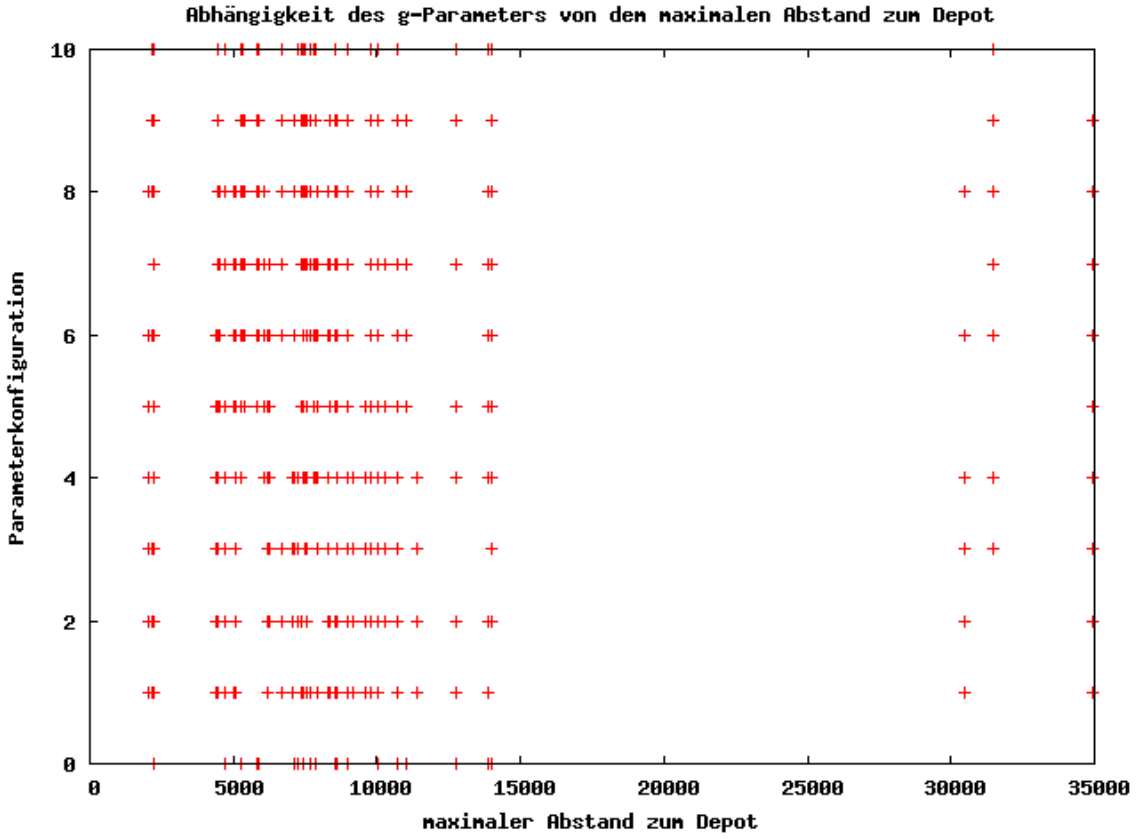


Abbildung A.6: Auf der X-Achse ist der maximale Abstand der Knoten zum Depot pro Testinstanz aufgetragen. Die Y-Achse gibt den verwendeten g-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten g-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

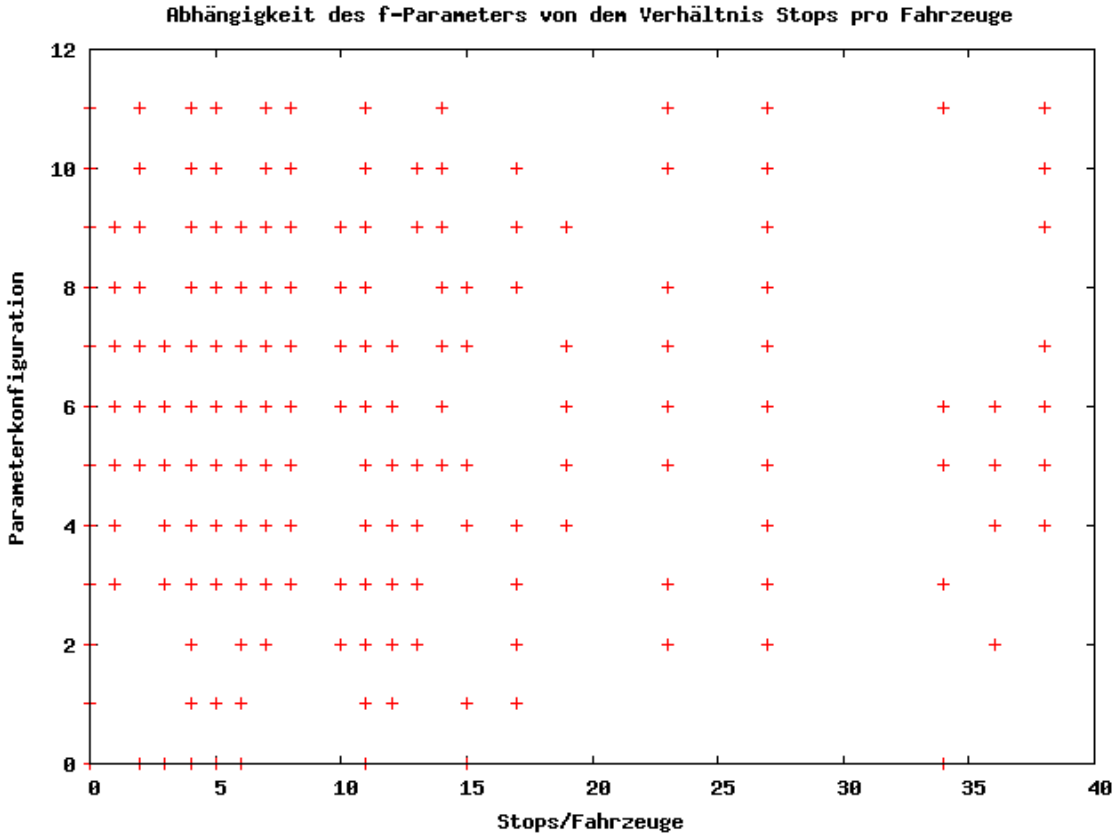


Abbildung A.7: Auf der X-Achse ist das Verhältnis von Knoten zu der Anzahl zur Verfügung stehender Fahrzeuge pro Testinstanz aufgetragen. Die Y-Achse gibt den verwendeten f-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten f-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

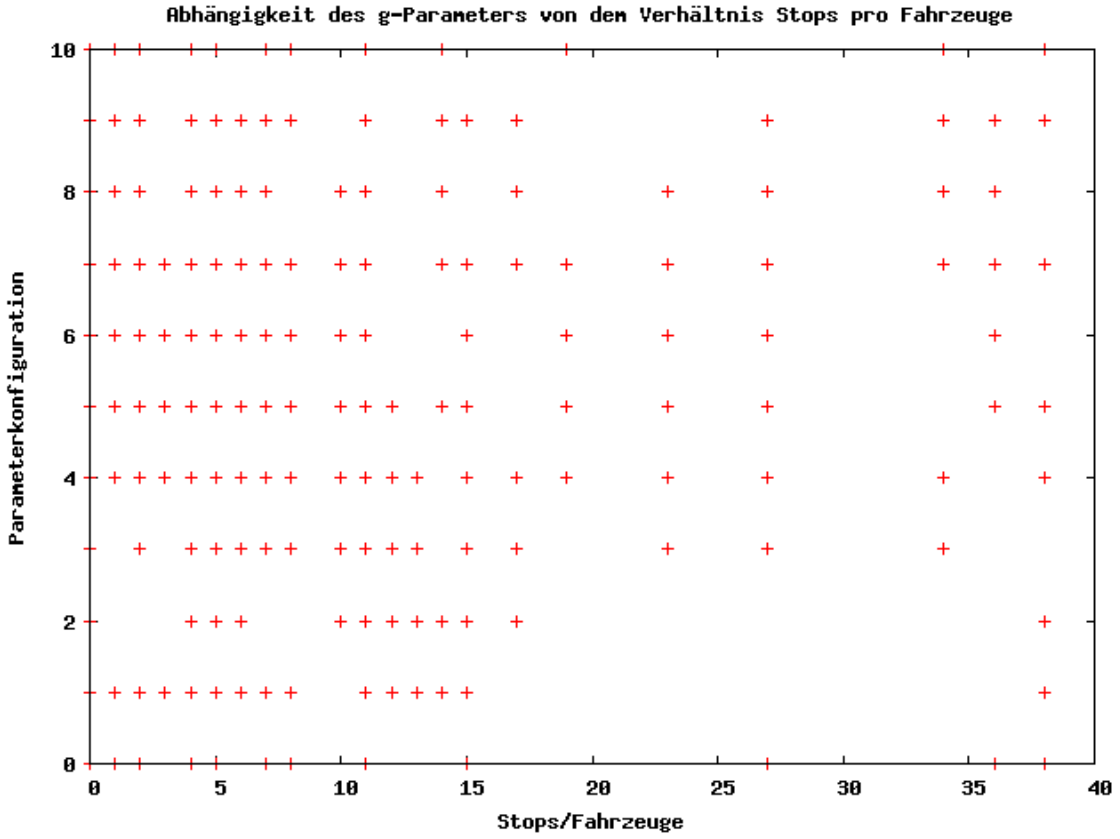


Abbildung A.8: Auf der X-Achse ist das Verhältnis von Knoten zu der Anzahl zur Verfügung stehender Fahrzeuge pro Testinstanz aufgetragen. Die Y-Achse gibt den verwendeten g-Parameter an. Markiert wurden für jede getestete Probleminstanz die zehn besten Ergebnisse und die dabei verwendeten g-Parameter. Insgesamt wurden pro Instanz 121 Parameterkombinationen getestet.

Literaturverzeichnis

[PTVAG 2009] : *Die PTV AG und ihre Geschäftsfelder*. December 2009. – URL `http://www.ptv.de/unternehmen/ueber-ptv/geschaeftsfelder/`

[Aarts und Lenstra 1997] Aarts, E. ; Lenstra, J.: *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., 1997. – ISBN 0471948225

[B. Fleischmann und Gnutzmann 2004] B. Fleischmann, M. G. ; Gnutzmann, S.: Time-Varying Travel Times in Vehicle Routing. In: *Transportation Science* 38 (2004), S. 160–173

[B. Golden und Wasil 2008] B. Golden, S. R. ; Wasil, E.: *The Vehicle Routing Problem, Latest Advances and New Challenges*. Springer, 2008. – ISBN 978-0-387-77777-1

[Bellmore und Nemhauser 1968] Bellmore, M. ; Nemhauser, G. L.: The traveling salesman problem: a survey. In: *Operations Research* 16 (1968), Nr. 3, S. 538–558

[Bräysy und Gendreau 2005a] Bräysy, O. ; Gendreau, M.: Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. In: *Transportation Science* 39 (2005), Nr. 1, S. 104–118

[Bräysy und Gendreau 2005b] Bräysy, O. ; Gendreau, M.: Vehicle Routing Problem with Time Windows, Part II: Metaheuristics. In: *Transportation Science* 39 (2005), Nr. 1, S. 119–139

[Clarke und Wright 1964] Clarke, G. ; Wright, J. W.: Scheduling of vehicles from a central depot to a number of delivery points. In: *Operations Research* 12 (1964), Nr. 4, S. 568–581

[Dantzig und Ramser 1959] Dantzig, G. B. ; Ramser, J. H.: The Truck Dispatching Problem. In: *Management Science* 6 (1959), Nr. 1, S. 80–91

[Domschke 1997] Domschke, W.: *Logistik: Rundreisen und Touren*. Oldenbourg, 1997. – ISBN 3-486-24273-3

[G. Laporte und Semet 2000] G. Laporte, J. P. ; Semet, F.: Classical and Modern Heuristics for the Vehicle Routing Problem. In: *International Transactions in Operational Research* 7 (2000), S. 285–300

[Holmes und Parker 1976] Holmes, R. A. ; Parker, R. G.: A Vehicle Scheduling Procedure Based Upon Savings and a Solution Perturbation Scheme. In: *Palgrave Macmillan Journals* 27 (1976), Nr. 1, S. 83–92

[J. Cordeau und Sormany 2005] J. Cordeau, A. Hertz G. Laporte J.Sormany J. Cordeau M. G. ; Sormany, J.: New heuristics for the vehicle routing problem. In: *Logistics Systems: Design and Optimization*, Public Health Service, 2005, S. 279–297

[König 1995] König, Barbara: *Heuristiken zur Ein-Depot-Tourenplanung.* Technische Universität München, Institut für Informatik, Diplomarbeit, 1995

[Malandraki und Daskin 1992] Malandraki, C. ; Daskin, M. S.: Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms. In: *Transportation Science* 26 (1992), S. 185–200

[N. Azi und Potvin 2006] N. Azi, M. G. ; Potvin, J.: An Exact Algorithm for a Single Vehicle Routing Problem with Time Windows and Multiple Routes. In: *European Journal of Operational Research* 178 (2006), Nr. 3, S. 755–766

[N. Azi und Potvin 2008] N. Azi, M. G. ; Potvin, J.: An Exact Algorithm for a Vehicle Routing Problem with Time Windows and Multiple Use of Vehicles. In: *European Journal of Operational Research* 202 (2008), Nr. 3, S. 756–763

[Paessens 1988] Paessens, H.: The savings algorithm for the vehicle routing problem. In: *European Journal of Operational Research* 34 (1988), Nr. 3, S. 336–344

[PTV-AG 2007] PTV-AG: *PTV Intertour 5.9, Handbuch.* 2007

[Toth und Vigo 2001] Toth, P. ; Vigo, D.: *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematics, 2001. – ISBN 0-89871-498-2

[Toth und Vigo 2003] Toth, P. ; Vigo, Daniele: The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. In: *INFORMS JOURNAL ON COMPUTING* 15 (2003), Nr. 4, S. 333–346

[UNION 2006] UNION, DAS EUROPÄISCHE PARLAMENT UND DER RAT DER E.: VERORDNUNG (EG) Nr. 561. In: *Amtsblatt der Europäischen Union* (2006)

[Yellow 1970] Yellow, P. C.: A Computational Modification to the Savings Method of Vehicle Scheduling. In: *Palgrave Macmillan Journals* 21 (1970), Nr. 2, S. 281–283