

**Effiziente Lokale Suche für
Vehicle Routing und Scheduling Probleme
mit Ressourcenbeschränkungen**

Von der Fakultät für Wirtschaftswissenschaften der
Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors der
Wirtschafts- und Sozialwissenschaften genehmigte Dissertation

vorgelegt von

Diplomlehrer für Mathematik und Physik

Birger Funke, M.O.R.

aus Bad Dübén

Berichter: Univ.-Prof. Dr.rer.nat.habil. Hans-Jürgen Sebastian
Univ.-Prof. Dr.rer.pol.habil. Michael Bastian

Tag der mündlichen Prüfung: 2. April 2003

Diese Dissertation ist auf den Internetseiten
der Hochschulbibliothek online verfügbar

Vorwort

Als ich im Januar 1992 unter der Leitung von Prof. Hans-Jürgen Sebastian meine Mitarbeit im Forschungsprojekt PROKON begann, fand ich am Türschild meines neuen Arbeitsplatzes die Aufschrift „Operations Research“, mit der ich damals nichts anzufangen wußte. Zu diesem Zeitpunkt ahnte ich nicht, daß ich etwa zehn Jahre später in diesem Wissenschaftszweig eine Dissertation einreichen würde.

Meine erste intensive Beschäftigung mit einigen Optimierungsmodellen und -verfahren geschah folgerichtig im Rahmen des PROKON-Projektes. Daraus entstanden Interesse und Motivation, das Zusatzstudium Operations Research an der RWTH Aachen aufzunehmen. Innerhalb eines Software-Projektes bei der ELITE Foundation habe ich mich mit vielen praktischen Aspekten bei der Planung von Brieftransporten bei der Deutschen Post AG und der Entwicklung geeigneter Heuristiken beschäftigt. Immer wieder mußten dabei Kompromisse gefunden werden, weil es nicht möglich war, alle realen Anforderungen innerhalb der verwendeten Algorithmen zu berücksichtigen. Dabei entstand die Idee, daß es möglich sein müßte, Algorithmen zu entwickeln, denen ein allgemeineres Modell zugrundeliegt.

Die vorliegende Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehr- und Forschungsgebiet Operations Research und Logistik Management der RWTH Aachen entstanden. Auch in dieser Zeit hat mich Prof. Sebastian betreut. Für seine langjährige Unterstützung und Begleitung möchte ich ihm an dieser Stelle danken. Er hat meine Entwicklung über viele Jahre gefördert und mir Gelegenheit zur Mitarbeit an interessanten Projekten und zur Teilnahme an internationalen wissenschaftlichen Tagungen gegeben. Ein Dank gilt auch Herrn Prof. Michael Bastian, der das Koreferat meiner Arbeit übernommen hat.

Besonders herzlich möchte ich mich bei meinen beiden Kollegen, Dr. Tore Grünert und Dr. Stefan Irnich bedanken. Beide haben nicht nur mein Manu-

skript sorgfältig und kritisch gelesen und mit vielen Vorschlägen und hilfreichen Kommentaren zu dessen Verbesserung beigetragen, vielmehr hätte ich diese Dissertation ohne die zahlreichen intensiven und äußerst gewinnbringenden fachlichen Diskussionen, die wir gemeinsam geführt haben, ohne die vielen Anregungen und Ideen, die dabei entstanden sind, nicht in dieser Form verfassen können.

Bei meiner Arbeit haben mir außerdem geholfen: Inge Buss, durch eine akribische Suche nach orthographischen Fehlern in meinem Manuskript, Gilberto von Sparr, der diverse Ausfälle meines Rechners schnell behob und stets für lauffähige Programme sorgte und die Kollegen des Lehr- und Forschungsgebietes, die mir ihre Computer nachts und an Wochenenden für meine umfangreichen Testrechnungen zur Verfügung stellten. Ihnen allen sei dafür ebenso herzlich gedankt wie den Kollegen, Freunden und Verwandten sowie meiner Freundin Annette, die mich in den letzten zwei Jahren moralisch unterstützt und meine geringe zeitliche Abkömmlichkeit verstanden haben.

Schließlich möchte ich in meinen Dank auch meine Eltern, Helga und Dr. Wilhelm Funke, einschließen, denn sie haben großen Anteil an meiner gesamten Entwicklung und deshalb auch daran, daß diese Arbeit entstehen konnte.

Aachen, im März 2003

Birger Funke

Inhaltsverzeichnis

Einleitung	1
1 Vehicle Routing und Scheduling Probleme	7
1.1 Ein Klassifikationsschema	8
1.2 Modelle	10
1.2.1 Beziehungen zwischen speziellen VRSP	10
1.2.2 Ein Modell für das m-TSPR	13
1.2.3 Erweiterungen des Modells	15
1.2.4 Einige spezielle Problemtypen	17
2 Verbesserungsheuristiken	25
2.1 Kombinatorische Optimierungsprobleme	26
2.2 Austausch-Nachbarschaften für das VRSP	31
2.2.1 Knotenorientierte Austausch-Nachbarschaften	32
2.2.2 Kantenorientierte Austausch-Nachbarschaften	39
2.2.3 GENIUS	50
2.3 Meta-Heuristiken	52
2.3.1 Variation der Nachbarschaft	55
2.3.2 Multi-Start Verfahren	61
2.3.3 Populationsbasierte bzw. evolutionäre Verfahren	63
2.3.4 Verfahren mit Gedächtnis	68
3 Lokale Suche in der k-opt-Nachbarschaft	71
3.1 Lokale Suche für das TSP	71
3.1.1 Einige Begriffe der Graphentheorie	72
3.1.2 Ein Modell für das TSP	75
3.1.3 Ein Modell für die k -opt-Nachbarschaft	78
3.1.4 Ein Algorithmus für die Lexikographische Suche	85

3.1.5	Effizienzbetrachtungen für die Lexikographische Suche	86
3.2	Probleme mit beschränkten Ressourcen	91
3.2.1	Berücksichtigung von Zeitfenstern (TSPTW)	93
3.2.2	Mengen beschränkter Ressourcen	97
3.2.3	Abhängige Ressourcen	99
3.3	Multiple TSP	102
3.3.1	Das Giant-Route-Modell für das m-TSP	102
3.3.2	k -opt-Schritte auf der Giant-Route	105
3.3.3	Berücksichtigung beschränkter Ressourcen beim m-TSP	107
3.3.4	k -opt*-Nachbarschaft	111
3.4	Suche in großen k -opt-Nachbarschaften	114
3.4.1	Reduktion der k -opt-Nachbarschaft	115
4	Cyclic-Transfer-Nachbarschaften	121
4.1	Formale Beschreibung der Cyclic Transfers	123
4.2	Aufbau eines Transfergraphen	127
4.2.1	Repräsentation von Cyclic Transfers im Transfergraphen	128
4.2.2	Behandlung von leeren Kundenteilmengen	129
4.2.3	Auswahl der Transfermengen	131
4.2.4	Wahl des Austauschoperators	136
4.3	Suche nach verbessernden Cyclic-Transfer-Schritten	139
4.4	Effiziente Lokale Suche in der Cyclic-Transfer-Nachbarschaft	146
4.5	Auflösen von Touren	148
5	Erweiterungen der Konzepte	153
5.1	Erweiterte Modelle	154
5.1.1	Heterogener Fuhrpark und mehrere Depots	154
5.1.2	Modelle mit Strafkosten und weichen Restriktionen . .	155
5.2	Effizienzverbesserungen und Integration mit Meta-Strategien und CP	155
5.2.1	Verbesserungen der Effizienz	156
5.2.2	Meta-Heuristiken für die Diversifikation der Suche . . .	157
5.2.3	Fixierung von Bögen und Einsatz von Makro-Knoten für k -opt-Nachbarschaften	159
5.2.4	Integration mit Constraint Programming	160

6	Ergebnisse	161
6.1	Qualität von heuristisch bestimmten Lösungen	162
6.2	Auswahl der Probleminstanzen	165
6.2.1	TSP mit Zeitfenstern	165
6.2.2	VRP	166
6.2.3	VRP mit Zeitfenstern	168
6.3	Validierung der k -opt-Nachbarschaften	169
6.3.1	Eine VND für die Lokale Suche	169
6.3.2	Leistungsfähigkeit der k -opt-Nachbarschaften	172
6.3.3	Auswirkungen effizienzsteigernder Maßnahmen	176
6.3.4	Einsatz von k -opt*-Nachbarschaften	187
6.3.5	k -opt-Nachbarschaften für das VRPTW	189
6.4	Cyclic-Transfer-Nachbarschaften für das VRP und das VRPTW	192
6.4.1	Durchführung der Suche in den Cyclic-Transfer-Nachbarschaften	193
6.4.2	Ergebnisse für das VRP	197
6.4.3	Ergebnisse für das VRPTW	200
6.4.4	Auflösung von Touren beim VRPTW	202
6.5	Eine einfache Diversifikation mit Kick-Schritten	206
6.5.1	Diversifikation mit Kicks	206
6.5.2	Kick-Schritte für das TSPTW	208
6.5.3	Kick-Schritte für VRP und VRPTW	211
	Zusammenfassung	215
	Literaturverzeichnis	221
A	Herleitung der Verbindungstheoreme	237

Abbildungsverzeichnis

1.1	Beziehungen zwischen VRSP	11
1.2	Ressourcenfenster beim VRPTWPD	19
1.3	Ressourcen für das VRPTWPD	22
2.1	Lokale Suche mit verschiedenen großen Nachbarschaften	30
2.2	Relocate- und Exchange-Schritt beim VRP	35
2.3	Node Ejection Chain	37
2.4	HyperOpt-Schritt beim TSP	39
2.5	Umkehrung der Orientierung eines Teilpfades beim 2-opt-Schritt	40
2.6	Eine Nachbarlösung, die durch verschiedene k -opt-Schritte erzeugt werden kann.	42
2.7	Verschiedene 3-opt-Schritte	44
2.8	„stem-and-cycle“-Struktur bei der sequentiellen Suche	44
2.9	Sequentielle Suche beim TSP	45
2.10	Nichtsequentielle 4-opt-Schritte („Double-Bridge-Move“)	46
2.11	Beispiele für k -opt*-Schritte	49
2.12	GENI – Type-I-Insertion	50
2.13	GENI – Type-II-Insertion	51
2.14	„Lineare Kombinationen“ beim Scatter Search	67
3.1	Überführen eines Hamiltonkreises in einen Hamiltonpfad	77
3.2	Vier Teilschritte eines k -opt-Schritts	79
3.3	Lexikographische Suche in der 2-opt-Nachbarschaft	96
3.4	Giant-Route für ein m -TSP	103
3.5	Pfaddarstellung einer Giant-Route ($m + 1$ Depotknoten)	103
3.6	Pfaddarstellung einer Giant-Route mit getrennten Depotein- und Depotausgängen	104
3.7	2-opt-Schritt $[aBa]$ auf der Giant-Route	106

3.8	3-opt-Schritt $[acba]$ auf der Giant-Route	107
3.9	3-opt-Schritt „ $aCba$ “ auf der Giant-Route	107
3.10	Fallunterscheidung bei der Bestimmung des Ressourcenver- brauchs für Segmente der Giant-Route	110
3.11	3-opt*-Schritte vom Typ $[a b c]$ und $[aC b]$	111
3.12	Durch Nächste-Nachbarn-Listen reduzierte Nachbarschaft beim TSP.	116
4.1	Cyclic Transfer der Stufe drei.	124
4.2	Transfergraph	130
4.3	Transfergraph mit zusätzlichem Knoten S_{0*}	131
4.4	Zyklisches Vertauschen von drei Tourendstücken	134
6.1	Vergleich verschiedener Cyclic-Transfer-Nachbarschaften beim VRP	198
6.2	Vergleich verschiedener Cyclic-Transfer-Nachbarschaften beim VRPTW	202
6.3	Ergebnisse beim Einsatz von Cyclic-Transfer-Nachbarschaften und zehn Kick-Schritten beim VRP	211
6.4	Ergebnisse beim Einsatz von Cyclic-Transfer-Nachbarschaften und zehn Kick-Schritten beim VRPTW	212

Tabellenverzeichnis

6.1	Ergebnisse einiger Tabu-Search Verfahren für die VRP-Instanzen von Christofides et. al.	167
6.2	Lokale Suche mit vollständigen k -opt-Nachbarschaften beim asymmetrischen TSPTW	173
6.3	Lokale Suche mit Intra-Tour-Nachbarschaften beim VRP . . .	174
6.4	Lokale Suche in vollständigen k -opt-Nachbarschaften beim VRP	176
6.5	Aufwandsreduktion durch Kostenabschätzung beim TSPTW .	180
6.6	Aufwandsreduktion durch Kostenabschätzung beim VRP . . .	181
6.7	k -opt-Nachbarschaften mit höchstens einem „langen“ Segment beim TSPTW	182
6.8	k -opt-Nachbarschaften mit höchstens einem „langen“ Segment beim VRP	183
6.9	Auswirkungen der Reduktion auf „nächste Nachbarn“ beim TSPTW	185
6.10	Auswirkungen der Reduktion auf „nächste Nachbarn“ beim VRP	186
6.11	k -opt*-Nachbarschaften für das VRPTW	188
6.12	k -opt-Nachbarschaften mit höchstens einem „langen“ Segment beim VRPTW	190
6.13	Vergleich der 4-opt-Suche mit Meta-Heuristiken für die RC1- und RC2-Instanzen von Solomon	191
6.14	Vergleich der 4-opt-Suche mit Meta-Heuristiken für die R1- und R2-Instanzen von Solomon	192
6.15	Anzahl der Touren beim VRPTW	204
6.16	Vergleich der k -opt-Suche mit Kick-Schritten mit den Ergebnissen von Balas und Simonetti	210

Einleitung

In den modernen Industriegesellschaften spielt der Transport von Rohstoffen, Teilfertig- und Fertigprodukten, aber auch die Entsorgung von Abfällen, Umverpackungen oder Altprodukten eine wichtige und stetig wachsende Rolle.

Das nationale und internationale Transportaufkommen bei Gütertransporten hat sich in den letzten Jahren ständig erhöht. Gleichzeitig führt die wachsende Mobilität in der Gesellschaft zu einer stetigen Zunahme der Personentransporte. So stieg z.B. der individuelle und öffentliche Personenverkehr in der Bundesrepublik Deutschland zwischen 1996 und 1999 um ca. drei Prozent. Gleichzeitig nahm der Güterverkehr sogar um etwa 14 Prozent zu.¹

Getragen von der wirtschaftlichen Bedeutung der Transportlogistik im privatwirtschaftlichen wie im öffentlichen Sektor wurden in den letzten vierzig Jahren tausende Artikel zu diesem Themengebiet veröffentlicht. Schon in dem 1983 von Bodin et. al. zusammengestellten Literaturüberblick [BGAB83], werden etwa 700 Artikel zitiert. In einer von Gutin und Punnen im Jahr 2002 herausgegebenen Monographie [GP02], welche sich ausschließlich dem Traveling Salesman Problems widmet, werden bereits mehr als 800 Referenzen angegeben. Mehrere wissenschaftliche Zeitschriften widmen sich ausschließlich Transport- und Tourenplanungsproblemen.

Fragestellungen, die zur Klasse der Vehicle Routing und Scheduling Probleme (VRSP) gehören, nehmen in der Transportlogistik einen breiten Rahmen ein. Die Entwicklung leistungsfähiger Optimierungsverfahren und Heuristiken für diese Problemklasse ist auch heute Gegenstand intensiver Forschungstätigkeit auf dem Gebiet des Operations Research.

¹Die Personentransporte in Deutschland erhöhten sich zwischen 1996 und 1999 von 927,3 Milliarden auf 955,5 Milliarden Personenkilometer. Die im Güterverkehr bewegten Ladungen stiegen im gleichen Zeitraum von 429,6 auf 491,4 Milliarden Tonnenkilometer. [DIZ01]

Vehicle Routing und Scheduling Probleme

VRSP beinhalten eine oder mehrere der folgenden Aufgaben:

- die Zuordnung von Transportaufträgen zu Touren oder Fahrzeugen („clustering“),
- die Planung der einzelnen Touren, d.h. die Festlegung der Reihenfolge der Aufträge innerhalb einer Tour („routing“), und
- die Planung der Fahrzeugumläufe für die Durchführung der Transporte („vehicle scheduling“).

Durch die Formulierung zusätzlicher Bedingungen, wie z.B. die Angabe einer maximalen Tourdauer oder von Reihenfolgeabhängigkeiten zwischen den Aufträgen innerhalb einer Tour, kann eine große Zahl spezieller VRSP gebildet werden.

Zu den VRSP gehören viele bekannte kombinatorische Optimierungsprobleme, wie z.B. das Traveling Salesman Problem (TSP) und das Vehicle Routing Problem (VRP). Diese und fast alle übrigen VRSP sind $\mathcal{NP}$ -hart. Das bedeutet, daß der Einsatz von optimierenden Verfahren nur für relativ kleine Probleminstanzen realisierbar ist und für praktische Aufgabenstellungen häufig ausgeschlossen werden muß. Dementsprechend gibt es ein großes Interesse an leistungsfähigen Heuristiken.

Anforderungen für die Planung realer Transportaufgaben

Trotz der rasanten Entwicklung der Rechentechnik und wesentlicher methodischer Fortschritte in den letzten Jahrzehnten stellt die Entwicklung von Heuristiken, die bei der Lösung realer Planungsaufgaben erfolgreich eingesetzt werden können, noch immer eine große Herausforderung dar.

Zu den wesentlichen Anforderungen, die für einen erfolgreichen Einsatz heuristischer Verfahren in der wirtschaftlichen Praxis entscheidend sind, zählen

1. die Möglichkeit, große Probleme (mit vielen Transportaufträgen) lösen zu können,²

²Was „viele Transportaufträge“ sind, ist von der konkreten Anwendung abhängig. Teilweise umfassen praktische Planungsaufgaben auch deutlich mehr Aufträge als die in der Literatur untersuchten Benchmark-Probleme. Wenn die Leistungsfähigkeit der Algorithmen nicht ausreicht, kann man immer versuchen, das Problem künstlich zu dekomponieren. Dadurch können aber ggf. große Potentiale verschenkt werden.

2. die Gewährleistung einer hohen Lösungsqualität, weitgehend unabhängig von den zugrundeliegenden Daten,
3. akzeptable Rechenzeiten (abhängig vom jeweiligen Einsatzgebiet),
4. die Berücksichtigung zusätzlicher Restriktionen (z.B. Zeitfenster oder Reihenfolgebeschränkungen) und Erweiterungen des Modells (z.B. mehrere Fahrzeugdepots, heterogener Fuhrpark oder Planung über mehrere Perioden) und
5. die Robustheit der Verfahren gegenüber Änderungen der Modellstruktur.

Bei der Entwicklung neuer Verfahren konzentrierte man sich mehrere Jahrzehnte lang vorwiegend auf die ersten drei Aspekte, das heißt, man versuchte, relativ große Instanzen eines fest vorgegebenen Problems möglichst schnell und in hoher Qualität zu lösen. Dabei wurden zum Teil hervorragende Ergebnisse erzielt. So können heute z.B. TSP-Instanzen mit mehreren zehntausend Kunden mit hoher Qualität in akzeptabler Zeit gelöst werden.³ Der direkte praktische Einsatz eines solchen Verfahrens für eine Transportplanung ist jedoch nur selten möglich, weil fast immer zusätzliche Restriktionen einzuhalten sind. Dazu können Beschränkungen der Tourlänge oder der Fahrzeit, Gewichts- und Volumenrestriktionen, Reihenfolgebeschränkungen beim Besuch von Kunden, kombinierte Einsammel- und Auslieferungstouren etc. gehören. Auch kompliziertere Kostenfunktionen können von vielen Heuristiken nicht berücksichtigt werden.

Mittlerweile gibt es auch zahlreiche Publikationen, die sich mit der Lösung von Problemen mit bestimmten Typen zusätzlicher Restriktionen beschäftigen (z.B. VRP mit Zeitfenstern, Pickup & Delivery Probleme). Die meisten dieser Heuristiken sind aber auf die jeweilige spezielle Modellstruktur zugeschnitten. Es bleibt daher offen, ob und wie die angewandte Methodik auf Probleme übertragen werden kann, bei denen veränderte oder zusätzliche Restriktionen berücksichtigt werden müssen.

Für die praktische Anwendung stellt die fehlende Robustheit der verwendeten Heuristiken gegenüber Änderungen der Modellstruktur ein ernsthaft-

³Applegate et. al. [ACR00] beschreiben Heuristiken, mit denen sie Probleme mit bis zu 85000 Kunden in weniger als zehn Minuten lösen können. Der Abstand ihrer Lösungen zum Optimum beträgt dabei weniger als 0,5 Prozent. Lösungen, die nicht mehr als 1 Prozent vom Optimum abweichen, können sie sogar innerhalb einer Minute bestimmen.

tes Problem dar. Es kommt häufig vor, daß erst während der ersten praktischen Tests oder sogar noch später neue Restriktionen „entdeckt“ werden oder durch geänderte Rahmenbedingungen tatsächlich erst später entstehen. Dann ist die Entscheidung über das eingesetzte Verfahren jedoch bereits gefallen und kann in der Regel nicht mehr problemlos geändert werden. Hohe Folgekosten für das Redesign der Software, die Beschränkung auf nur teilweise verwendbare Lösungen, die mit vollkommen unbekannten Einbußen „per Hand“ korrigiert werden, oder sogar der vollständige Verzicht auf eine optimierungsbasierte Planung sind dann häufige Konsequenzen.

Der einzige bislang verfolgte Ansatz, der den obigen Forderungen 4 und 5 in hohem Maß gerecht werden kann, ist die Verknüpfung heuristischer Suchverfahren mit Methoden des Constraint Programming, wie es in unterschiedlicher Form von Shaw [Sha98] und de Backer et. al. [dFS⁺00] bzw. von Pesant und Gendreau [PG99] vorgeschlagen wurde. Dabei wird der Vorteil der hohen Flexibilität aber durch eine merkliche Einschränkung bei der Effizienz der Verfahren erkaufte. Potentielle Anwender stehen deshalb vor der unbefriedigenden Alternative, entweder ein sehr flexibles oder ein wesentlich effizienteres Verfahren für die Planung einsetzen zu können.

Zielstellung

Gegenstand dieser Arbeit ist die Entwicklung von effizienten und leistungsfähigen Verbesserungsheuristiken, die für nahezu alle VRSP eingesetzt werden können. Dafür wird ein sehr allgemeines Ressourcenmodell verwendet, mit denen die VRSP beschrieben werden. Restriktionen können darin durch Einschränkungen der Zulässigkeitsbereiche für Ressourcenvariablen formuliert werden.

Der Fokus dieser Dissertation liegt in der Entwicklung von Verbesserungsheuristiken auf der Basis von Nachbarschaften und jeweils geeigneter, auf diese Nachbarschaften abgestimmter Suchstrategien. Mit der Lexikographischen Suche für k -opt-Nachbarschaften und der Suche in Cyclic-Transfer-Nachbarschaften werden zwei sehr unterschiedliche Strategien entwickelt, mit denen die Nachbarschaften auch unter Berücksichtigung von Ressourcenbeschränkungen effizient untersucht werden können. Eine darüber hinausgehende Einbindung der Nachbarschaften in eine Meta-Heuristik ist die zugrundeliegende Intention der Untersuchungen, steht aber nicht selbst im Zentrum der Betrachtungen.

Gliederung der Arbeit

Die vorliegende Arbeit umfaßt insgesamt sechs Kapitel. Diese lassen sich den folgenden drei Schwerpunkten zuordnen:

Modelle und Heuristiken aus der Literatur Das Kapitel 1 dient der Beschreibung der VRSP anhand eines Klassifikationsschemas von Desrochers et. al. [DLS90] und der Einführung eines Ressourcenmodells für die Klasse der VRSP. Dieses ist das Basismodell, das allen Betrachtungen der Kapitel 3 und 4 zugrunde liegt. Zur Veranschaulichung des abstrakten Ressourcenbegriffs wird gezeigt, wie mehrere klassische VRSP mit Hilfe von einer oder mehreren beschränkten Ressourcen modelliert werden können.

Im Kapitel 2 wird ein Überblick der umfangreichen Literatur zu heuristischen Methoden für VRSP gegeben. Dabei liegt der Schwerpunkt bei den klassischen kanten- und knotenorientierten Austausch-Nachbarschaften. Die aus Sicht des Autors bestimmenden Ideen der zahlreichen publizierten Meta-Heuristiken sind in einem eigenen Abschnitt zusammengefaßt.

Neue Konzepte auf der Basis des Ressourcenmodells Das Kapitel 3 widmet sich vorwiegend der effizienten Suche in k -opt-Nachbarschaften für Probleme mit Ressourcenbeschränkungen. Dabei wird gezeigt, wie die von Savelsbergh [Sav85] für Probleme mit Zeitfensterrestriktionen vorgeschlagene Durchführung einer Lexikographischen Suche auf eine Menge von beschränkten Ressourcen übertragen werden kann. Die Vorgehensweise wird formal für k -opt-Nachbarschaften mit beliebigem Wert von k beschrieben und auf Probleme mit mehreren Touren (m-TSPR) sowie auf k -opt*-Nachbarschaften ausgedehnt. Außerdem werden effizienzsteigernde Maßnahmen in Form von Kostenabschätzungen und gezielten Einschränkungen der k -opt-Nachbarschaften diskutiert.

Im Kapitel 4 werden die Cyclic-Transfer-Nachbarschaften von Thompson und Psaraftis [TP93] aufgegriffen. Diese werden mit einem Ansatz von Ahuja, Orlin und Sharma zur effizienten Suche in sehr großen Nachbarschaften [AOS01] in Verbindung gebracht. Es wird dargestellt, wie sich die Überprüfung der Ressourcenbeschränkungen mit Hilfe der in Kapitel 3 beschriebenen Konzepte auch für die Cyclic-Transfer-Nachbarschaften verwenden läßt.

Die Darstellung einiger Möglichkeiten zur Fortsetzung und Erweiterung der beschriebenen Konzepte ist Gegenstand des Kapitels 5. Dabei wird einerseits auf weitere Problemtypen eingegangen, die mit Hilfe beschränkter

Ressourcen modelliert werden können. Andererseits werden mögliche Verbesserungen und Erweiterungen sowie Anknüpfungspunkte für weitergehende methodische Entwicklungen vorgestellt.

Validierung der Konzepte Im Kapitel 6 wird die Wirksamkeit und die Effizienz der vorgeschlagenen Methoden anhand von umfangreichen Testrechnungen nachgewiesen. Dafür werden Benchmark-Probleminstanzen zum TSP mit Zeitfenstern sowie zum VRP mit bzw. ohne Zeitfensterrestriktionen verwendet. Anhand zahlreicher Vergleichsrechnungen mit unterschiedlichen vollständigen und künstlich reduzierten Nachbarschaften wird deren Beitrag zur Verbesserung der Lösungen abgeschätzt und dem notwendigen Rechenaufwand gegenübergestellt. Dadurch können wertvolle Rückschlüsse für eine sinnvolle Gestaltung von hybriden Verbesserungsheuristiken auf der Basis mehrerer Nachbarschaften gezogen werden.

Kapitel 1

Vehicle Routing und Scheduling Probleme

Vehicle Routing und Scheduling Probleme (VRSP) bilden eine große Klasse kombinatorischer Optimierungsprobleme. Zu dieser Klasse gehören viele bekannte Probleme, u.a. das Traveling Salesman Problem (TSP), das Vehicle Scheduling Problem (VSP) und das Vehicle Routing Problem (VRP). Durch die Berücksichtigung zusätzlicher Nebenbedingungen wie Zeitfenster oder Reihenfolgeabhängigkeiten der Kunden etc. ergeben sich viele speziellere Problemstellungen, etwa das TSP mit Zeitfenstern (TSPTW), das Vehicle Routing Problem mit Zeitfenstern (VRPTW) oder das Pickup & Delivery Problem (PDP).

Die große Zahl von Publikationen, die innerhalb der letzten Jahrzehnte zur Beschreibung mehr oder weniger stark spezialisierter Modelle für Vehicle Routing und Scheduling Probleme (und darauf basierender Algorithmen) erschienen sind, deutet auf die Notwendigkeit einer integrierenden Modellbildung hin. Erste Ansätze dazu waren der Artikel [BG81] von Bodin und Golden und das Klassifikationsschema für VRSP, das von Desrochers et al. in [DLS90] publiziert wurde. 1995 stellten Desrosiers et al. in [DDSS95] ein allgemeines Modell für Vehicle Routing und Scheduling Probleme vor. Dieses Modell wurde in [DDI⁺98] nochmals erweitert.

Gegenstand dieses Kapitels ist die Klassifizierung und Modellierung von VRSP. Die wesentlichen Ideen der Modellbildung sind dem Beitrag [DDI⁺98] von Desaulniers et al. entnommen, der eine Vielzahl zuvor veröffentlichter Modelle aufgreift und vereinheitlicht. Den Autoren gelingt es, ein allgemeines Modell für VRSP anzugeben, das insbesondere eine sehr große Vielfalt un-

terschiedlicher Zielfunktionen und Nebenbedingungen berücksichtigen kann. Dies ist für die vorliegende Arbeit von besonderer Bedeutung, da gerade auf die Anwendbarkeit der später behandelten Suchverfahren auf einen großen Teil dieser Probleme großer Wert gelegt wird.

Um die bereits angesprochene Vielfalt der Modelle zu illustrieren, wird zunächst anhand des Klassifikationsschemas für VRSP von Desrochers et al. dargelegt, welche Merkmale zur Unterscheidung der einzelnen Modelle herangezogen werden können. Anschließend wird in Abschnitt 1.2 in Anlehnung an [DDI⁺98] ein Modell für das m-TSP¹ mit Ressourcenbeschränkungen (m-TSPR) entwickelt. Es umfaßt u.a. das Pickup & Delivery Problem, das Multiple Trip VRPTW und das VRPTW mit simultanen Pickups & Deliveries als Spezialfälle (siehe Unterabschnitt 1.2.4). Alle algorithmischen Betrachtungen dieser Arbeit beziehen sich letztendlich auf das m-TSP mit Ressourcenbeschränkungen.

1.1 Ein Klassifikationsschema für VRSP

Die Ausgangssituation bei einem Vehicle Routing und Scheduling Problem wird bei Desrochers et. al. wie folgt charakterisiert:

„A number of vehicles, stationed at one or more depots, have to serve a collection of customers in such a way that given constraints are respected and a given objective function is optimized.“²

Die einzelnen Problemtypen unterscheiden sich danach, wie die Depots und Kunden verteilt sind, welche Fahrzeuge benutzt werden dürfen bzw. müssen, was die vorgegebenen Restriktionen beinhalten und wie die Zielfunktion gegeben ist. Das Klassifikationsschema von Desrochers et al. umfaßt daher die vier Kriterienklassen: „Adressen“, „Fahrzeuge“, „Problem-Charakteristika“ und „Ziele“. Diese beinhalten (etwas gekürzt) die folgenden Unterscheidungsmerkmale.

¹Multiple Traveling Salesman Problem

²aus [DLS90]: Eine Zahl von Fahrzeugen, die sich an einem oder mehreren Depots befinden, muß eine Menge von Kunden derart bedienen, daß bestimmte vorgegebene Restriktionen eingehalten werden und eine gegebene Zielfunktion optimiert wird.

Adressen werden in einem Graphen $G=(V,A)$ mit den Knoten V und den (ggf. gerichteten) Kanten A dargestellt. Eine Unterscheidung wird jeweils getroffen nach

- Anzahl der Depots
- Art der Aufträge:
 - zugeordnet nur zu Knoten, nur zu Kanten oder zu Knoten und Kanten
 - reines Einsammeln bzw. Ausliefern oder gemischt
 - kein, ein oder mehrere Zeitfenster oder fixe Zeitpunkte an jedem Knoten
- Beschränkungen der Adressenauswahl
 - alle Adressen müssen besucht werden
 - eine Teilmenge der Adressen muß besucht werden
 - mindestens eine Adresse aus jeder Teilmenge einer Partition der Adressen muß besucht werden
 - mehrere Pläne für einen gegebenen Zeitraum (z.B. Wochenpläne)

Fahrzeuge werden klassifiziert nach:

- ihrer (maximalen) Anzahl
- ihrer Kapazität
- der Einsetzbarkeit für verschiedene Güter (z.B. Kühlfahrzeuge, Teilbarkeit des Laderaums)
- Verfügbarkeit (Einsatz-Zeitfenster)
- Beschränkungen der Tourdauer

Kapazität und Verfügbarkeit der Fahrzeuge können jeweils für alle Fahrzeuge einheitlich oder für jedes Fahrzeug individuell vorgegeben sein. Die Tourdauer kann ebenfalls abhängig von der jeweiligen Tour unterschiedlich nach unten bzw. nach oben beschränkt sein.

Problem-Charakteristika sind eine Kollektion weiterer Eigenschaften. Diese umfassen

- den Netzwerktyp,
- den Strategie-Typ,
- Adressen-Adressen-Beschränkungen,
- Adressen-Fahrzeug-Beschränkungen und
- Fahrzeug-Fahrzeug-Beschränkungen.

Netzwerktypen unterscheiden sich danach, ob die Kosten die Dreiecksungleichung einhalten bzw. ob ungerichtete, gerichtete oder gemischte Netzwerke betrachtet werden. Von den Strategie-Typen ist z.B. abhängig, ob ein Aufteilen der Nachfrage erlaubt ist oder ob ein Fahrzeug auch mehrere Touren bedienen kann. Zu den Adressen-Adressen-Beschränkungen zählen u.a. Reihenfolge-Beschränkungen oder Adressen-Inkompatibilitäten usw.

Ziele können abhängig sein von der reinen Reisezeit, der Tourdauer, der zurückgelegten Strecke, von den eingesetzten Fahrzeugen sowie ggf. von verschiedenen Strafkosten. Mehrere Ziele können z.B. durch Summation oder einen Minimum- bzw. Maximum-Operator miteinander verknüpft sein.

1.2 Modelle

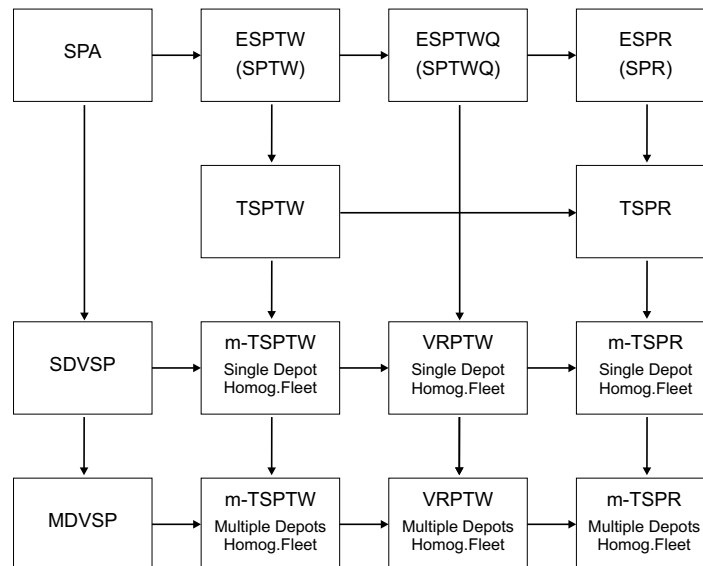
In [DDI⁺98] werden explizit zeitlich beschränkte VRSP betrachtet. Die Autoren beginnen mit relativ einfachen Modellen, die dann systematisch erweitert werden. So wird schrittweise ein komplexes Mehrgüter-Netzflußmodell entwickelt, das sehr viele Varianten des VRSP abdecken kann (siehe Abb. 1.1).

1.2.1 Beziehungen zwischen speziellen Vehicle Routing und Scheduling Problemen

Ausgangspunkt der Betrachtungen ist das Kürzeste-Wege-Problem bzw. Shortest Path Problem auf einem azyklischen Graphen (SPA). Das SPA kann in polynomialer Zeit gelöst werden [Ful72].

Als erste Erweiterung werden Kundenzeitfenster hinzugenommen, wodurch das SPA in ein Shortest Path Problem mit Zeitfenstern (SPTW) bzw. ein Elementary Shortest Path Problem mit Zeitfenstern (ESPTW) überführt

Es sei darauf hingewiesen, daß sich das SPTW stets auf einen azyklischen Graphen transformieren läßt, wenn jeder Kreis des Graphen einen positiven Zeitverbrauch besitzt. Wegen der beschränkten Zeitfenster an den Knoten ist dann jeder zulässige Pfad endlich. Das Problem läßt sich somit äquivalent auf einen azyklischen Graphen abbilden, der endlich viele Kopien der Originalknoten mit unterschiedlichen Zeitfenstern besitzt.

Abbildung 1.1: Beziehungen zwischen VRSP⁴

⁴Diese Darstellung wurde aus [DDI⁺98] übernommen.

Kapazitätsrestriktionen (SPTWQ bzw. ESPTWQ). Schließlich werden auch Probleme mit einer Menge beschränkter Ressourcen betrachtet, die als Shortest Path Problem mit Ressourcenbeschränkungen (SPR, ESPR) bezeichnet werden. Diese umfassen alle vorgenannten Probleme.

In der Abb. 1.1 sind alle Kürzeste-Wege-Probleme in der obersten Zeile dargestellt. Diese können nun zu Vehicle Scheduling Problemen bzw. Vehicle Routing Problemen erweitert werden. Die weiter links stehenden Probleme können stets als Spezialfall der rechts stehenden Probleme erhalten werden.

Durch die Erweiterung der Modelle um die sogenannten Überdeckungsbedingungen (engl. covering constraints), die dafür sorgen, daß alle Kunden bedient werden müssen, geht man vom ESPTW zum TSP mit Zeitfenstern (TSPTW) bzw. vom ESPR zum Traveling Salesman Problem mit Ressourcenbeschränkungen (TSPR) über.

Läßt man außerdem mehrere Touren zu, erhält man das Multiple TSP mit Zeitfenstern (m-TSPTW), das Vehicle Routing Problem mit Zeitfenstern (VRPTW) und das Multiple TSP mit Ressourcenbeschränkungen (m-TSPR). Durch den Übergang zu einem Mehrgüter-Netzflußmodell können auch VRSP mit heterogenem Fuhrpark bzw. mehreren Depots dargestellt werden.⁵

Prinzipiell können alle nachfolgenden Betrachtungen zur Lokalen Suche auf einem Mehrgüter-Netzflußmodell basieren, wie es in [DDI⁺98] angegeben wird. Der Fokus dieser Arbeit soll aber nicht auf der Lösung von Problemen mit heterogenen Fuhrparks bzw. Multi-Depot Problemen liegen. Deshalb kann ein etwas einfacheres Modell zugrunde gelegt werden, d.h. wir betrachten formal das Netzflußmodell, das sich durch die Beschränkung auf einen Fahrzeugtyp bzw. ein Depot ergibt. Mit diesem Modell können das m-TSPR und alle Spezialfälle davon (also TSPTW, VRPTW, PDP, ...) beschrieben werden.

⁵Modelle für VRSP mit heterogenem Fuhrpark und VRSP mit mehreren Depots sind oft zueinander äquivalent. Einen heterogenen Fuhrpark kann man z.B. dadurch modellieren, daß die unterschiedlichen Fahrzeuge verschiedenen (fiktiven) Depots zugewiesen werden. Die Fahrzeugeigenschaften können dann über die mit dem Depot verbundenen Kanten bzw. Bögen unterschiedlich abgebildet werden (z.B. Fixkosten, Kapazitäten). Bei der Modellierung mit Hilfe eines Mehrgüter-Netzflußmodells können sich alle Kanten und Bögen je nach Fahrzeug(typ) unterscheiden. So sind auch unterschiedliche Geschwindigkeiten, Fahrtkosten etc. modellierbar.

1.2.2 Ein Modell für das m-TSPR

Sei $G = (V, A)$ ein vollständiger gerichteter Graph mit der Knotenmenge $V = N \cup D$. Dabei repräsentiert N die Menge der Aufträge bzw. Kunden. Die dazu disjunkte Depotmenge D besteht aus den Knoten o und d , die den Ausgang bzw. den Eingang des Fahrzeugdepots bezeichnen. Jedem Bogen $(i, j) \in A$ sind die Kosten c_{ij} zugeordnet.

Außerdem existiere eine Menge von Ressourcen R . Ist r eine Ressource von R , so bezeichnet T_i^r die Variable, die den Wert der Ressource r bei der Ankunft am Knoten $i \in V$ angibt. d_{ij}^r ist der Verbrauch der Ressource r entlang des Bogens (i, j) . An jedem Knoten $i \in V$ muß der Wert der Ressourcenvariablen T_i^r im Intervall $[a_i^r, b_i^r]$ liegen, das als Ressourcenfenster bezeichnet wird. Ein Pfad im Graphen G heißt zulässig, wenn die Ressourcenfenster aller Ressourcen an jedem Knoten eingehalten werden.

Das m-TSPR läßt sich dann schreiben als

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \quad (1.1a)$$

so daß

$$\sum_{j: (i,j) \in A} x_{ij} = 1, \quad \forall i \in N \quad (1.1b)$$

$$\sum_{j: (i,j) \in A} x_{ij} - \sum_{j: (j,i) \in A} x_{ji} = 0, \quad \forall i \in N \quad (1.1c)$$

$$\sum_{j: (o,j) \in A} x_{oj} = \sum_{i: (i,d) \in A} x_{id} \leq m, \quad (1.1d)$$

$$x_{ij}(T_i^r + d_{ij}^r - T_j^r) \leq 0, \quad \forall r \in R, \forall (i, j) \in A \quad (1.1e)$$

$$a_i^r \leq T_i^r \leq b_i^r, \quad \forall r \in R, \forall i \in V \quad (1.1f)$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \quad (1.1g)$$

Die binären Variablen x_{ij} geben an, ob der Bogen (i, j) Teil der Lösung ist. Die Zielfunktion (1.1a) minimiert die Summe der Kosten der verwendeten Bögen. (1.1b) enthält die bereits erwähnten Überdeckungsbedingungen. Sie gewährleisten, daß jeder Kundenknoten genau einmal besucht wird. Die Gleichungen (1.1c) stellen die Netzflußbedingungen dar. Durch sie wird gesichert, daß an jedem Kundenknoten gleich viele Fahrzeuge ankommen und abfahren,

so daß diese Knoten weder Quellen noch Senken für die Fahrzeuge darstellen können. Gemäß (1.1d) verlassen das Startdepot ebenso viele Fahrzeuge wie auch am Zieldepot ankommen. Die maximale Anzahl der Fahrzeuge ist auf m begrenzt.⁶ Durch die Formulierung in (1.1e) wird erreicht, daß sich die Werte der Ressourcenvariablen T_i^r und T_j^r für jeden benutzten Bogen (i, j) um d_{ij}^r voneinander unterscheiden. Dabei sind Abweichungen nach unten erlaubt. Dadurch läßt sich z.B. modellieren, daß trotz der Existenz von Zeitfenstern eine zu frühe Ankunft am Knoten j zulässig ist, weil die Möglichkeit besteht, bis zum Beginn des Zeitfensters zu warten. Zwingt man z.B. die Ressourcenvariablen am Depotausgangsknoten o durch entsprechende Wahl des Ressourcenfensters auf null und betrachtet einen zulässigen Pfad von o zum Knoten i , so gibt die Variable T_i^r die Summe der Ressourcenveränderungen auf diesem Pfad an. Die Restriktionen (1.1f) sorgen schließlich für die Einhaltung der Ressourcenfenster für jede Ressource an allen Knoten des Graphen.

Eine Veränderung der Ressourcen beim Besuch eines Knotens braucht nicht explizit modelliert zu werden. Wenn knotenabhängige Ressourcenänderungen berücksichtigt werden sollen, so können diese auf alle aus dem betreffenden Knoten ausgehenden Bögen aufgeschlagen werden.

Die in Abschnitt 1.2.1 genannten Spezialfälle des m-TSPR können mit dem Modell (1.1) durch folgende Festlegungen erhalten werden:

m-TSPTW Die Ressourcenmenge R besteht nur aus der Ressource r_{time} , mit deren Hilfe die zeitlichen Einschränkungen modelliert werden. $a_i^{r_{time}}$ und $b_i^{r_{time}}$ geben das Zeitfenster am Knoten i an.⁷ Der Ressourcenverbrauch $d_{ij}^{r_{time}}$ des Bogens (i, j) wird gleich der Fahrtdauer von i nach j zuzüglich der Servicezeit am Knoten j gesetzt.

(kapazitiertes) VRPTW Die Ressourcenmenge R besteht aus den Ressourcen r_{time} (Zeit) und r_{load} (Fahrzeugbeladung). Die Zeit wird genauso wie beim TSPTW modelliert. Die Beschränkungen der Fahrzeugbeladung r_{load} sind an allen Knoten gleich: $a^{r_{load}} = 0$ und $b^{r_{load}} = Q$, wobei Q die gesamte Kapazität eines Fahrzeugs ist. Der Ressourcenverbrauch $d_{ij}^{r_{load}}$ auf dem Bogen (i, j) wird gleich dem Bedarf q_j des Kunden j gesetzt.⁸

⁶Statt (1.1d) würde es auch genügen, $\sum_{j:(o,j) \in A} x_{oj} \leq m$ bzw. $\sum_{i:(i,d) \in A} x_{id} \leq m$ zu fordern, da wegen (1.1c) die Gleichheit in (1.1d) immer erfüllt ist.

⁷Darunter wird, wie üblich, die früheste und späteste *Ankunftszeit* beim Kunden verstanden.

TSPTW Das Modell ist identisch zum m-TSPTW, jedoch wird zusätzlich die maximale Zahl der Fahrzeuge m gleich eins gesetzt.

TSPR Im Unterschied zum m-TSPR ist die maximale Zahl der Fahrzeuge gleich eins.

1.2.3 Erweiterungen des Modells

In [DDI⁺98] werden mehrere Erweiterungen betrachtet, mit denen zusätzliche Problemeigenschaften modelliert werden können. Die erste Erweiterung – das Verwenden einer Menge von Ressourcen – ist bereits im Modell (1.1) enthalten. Mit Hilfe einer Ressourcenmenge ist es möglich, eine Vielfalt von Restriktionen zu formulieren, die jeweils einen einzelnen Pfad betreffen. Ein Modell, das diese Möglichkeiten nutzt, ist das Vehicle Routing Problem mit Zeitfenstern und mit Pickups & Deliveries, das in Abschnitt 1.2.4 beschrieben wird.

Wir gehen nachfolgend auf die Verallgemeinerung der Ressourcenverbräuche mittels einer „Resource Extension Function“ sowie eine verallgemeinerte Zielfunktion ein.

Resource Extension Function Desaulniers et al. [DDI⁺98] ersetzen die Ungleichung (1.1e), die die Veränderung der Ressource r entlang des Bogens (i, j) durch einen fixen Wert d_{ij}^r beschreibt, durch

$$x_{ij} \cdot (f_{ij}^r(T_i^R, T_j^R) - T_j^r) \leq 0, \quad \forall r \in R, \forall (i, j) \in A \quad (1.2)$$

T_i^R und T_j^R bezeichnen Vektoren der Ressourcenvariablen an den Knoten i bzw. j . Die Funktion f_{ij}^r ist i.a. abhängig von T_i^R und T_j^R und wird als Resource Extension Function (REF) bezeichnet. Mit ihrer Hilfe wird der Wert der Ressourcenvariablen T_j^r gemäß (1.2) eingeschränkt.

An dieser Stelle genügt es, die Berechenbarkeit der REF zu fordern. Später werden jedoch weitere Anforderungen an die REF formuliert werden, um eine gegebene Lösung effizient auf ihre Zulässigkeit prüfen zu können.⁹

⁸Diese Modellierung wird durch folgende Überlegung plausibel: Die Beladung des Fahrzeugs wird durch die Bedarfe der besuchten Kunden festgelegt. Die Entscheidung, den Bogen (i, j) zu benutzen, führt unweigerlich dazu, daß der Kunde j bedient und die entsprechende Ladung aufgenommen wird (bei Auslieferungstouren vor der Abfahrt am Depot und bei Einsammeltouren am Kundenstandort). Deshalb wird jedem Bogen, der zum Kunden j führt, der Bedarf dieses Kunden als Verbrauch der Ressource r_{load} zugerechnet.

⁹vgl. Abschnitt 3.2.2

Die Gleichung (1.1e) ist als Spezialfall in (1.2) enthalten, bei dem die REF nur von T_i^r abhängig ist:

$$f_{ij}^r(T_i^R, T_j^R) = T_i^r + d_{ij}^r.$$

Die Verwendung von (1.2) ermöglicht es, voneinander abhängige Ressourcen zu modellieren und so auch komplexere Zusammenhänge zu modellieren. Das VRPTW mit simultanen Pickups & Deliveries¹⁰ ist ein Beispiel für eine Problemklasse, die sich mit mehreren voneinander abhängigen Ressourcen modellieren läßt.

Verallgemeinerung der Zielfunktion Die Zuhilfenahme der REF macht Ressourcen zu einem sehr flexiblen Modellierungswerkzeug. Bei der Verallgemeinerung der Zielfunktion macht man sich diesen Umstand zunutze, indem eine spezielle Ressource r_0 für die Modellierung der Kosten einer Lösung eingeführt wird.

Als Beispiel nehmen wir an, daß die Zielfunktion (1.1a) mit Hilfe der Ressource r_0 modelliert werden soll. Man setzt dazu die REF

$$f_{ij}^{r_0}(T_i^R, T_j^R) = T_i^{r_0} + c_{ij}, \forall i, j \in A. \quad (1.3)$$

Abhängig davon, ob das Problem nur eine oder auch mehrere Touren erlaubt, kann die Zielfunktion unterschiedlich dargestellt werden. Im ersten Fall, d.h. beim TSPR kann man den Wert von r_0 am Depoteingang optimieren. Durch die Wahl der REF werden die Kosten c_{ij} entlang der (einzigen) Tour vom Depotausgang o bis zum Knoten i in der Variablen $T_i^{r_0}$ summiert. Dies funktioniert beim TSPR, weil jeder Knoten genau einmal besucht wird und es zu jedem Knoten i eine eigene Ressourcenvariable $T_i^{r_0}$ gibt. Der Wert von $T_d^{r_0}$ repräsentiert den Wert der Zielfunktion nach dem kompletten Durchlaufen der Tour und somit der gesamten Lösung. Ersetzt man also die Zielfunktion (1.1a) durch

$$\min T_d^{r_0}, \quad (1.4)$$

so ist das TSPR-Modell vollständig.

Im allgemeinen Fall werden die Knoten o und d beim m-TSPR von *jeder Tour* einmal (also maximal m -fach) besucht. Am Depotausgang o genügt

¹⁰siehe Seite 20

dennoch eine einzige Ressourcenvariable $T_o^{r_0}$, die auf einen für alle Touren gemeinsam gültigen Wert von r_0 initialisiert werden kann. Am Depoteingang d müssen die Kosten aller eingehenden Touren summiert werden. Dies kann durch die folgende Formulierung der Zielfunktion erreicht werden:¹¹

$$\min \sum_{(i,d) \in A} x_{id} T_i^{r_0} + x_{id} c_{id}. \quad (1.5)$$

Die Flexibilität der Resource Extension Functions ermöglicht es nun, die im Klassifikationsschema¹² angedeutete Vielfalt unterschiedlicher Zielfunktionen abzubilden.

Abschließend sei darauf hingewiesen, daß es trotz der genannten Erweiterungen auch eine Reihe von praktischen Problemen gibt, die mit dem Modell (1.1) nicht beschrieben werden können. Dazu zählen z.B. solche Probleme, bei denen der Verbrauch einer Ressource an einem Knoten oder auf einem Bogen von dem aktuellen Wert dieser oder anderer Ressourcen abhängig ist.¹³

1.2.4 Einige spezielle Problemtypen

Es wurde bereits gezeigt, daß z.B. das TSPTW oder das VRPTW mit Hilfe von Modell (1.1) dargestellt werden können. Um die Flexibilität des Modells und die Verwendung der Ressourcen zu illustrieren, werden in diesem Unterabschnitt für einige weitere häufig untersuchte Klassen des VRSP eine Modellierung als m-TSPR angegeben. Dazu wird teilweise die REF gemäß Gleichung (1.2) verwendet.

VRP mit Zeitfenstern und Pickups & Deliveries Eine wichtige Klasse der VRSP sind die Pickup & Delivery Probleme (PDP). Bei einem PDP muß eine Menge bestimmter Frachten oder Ladungen von ihren Ausgangsorten

¹¹Bei der Verwendung des Giant-Route-Modells für das m-TSPR (siehe Abschnitt 3.3.1 auf Seite 102) werden die Depotknoten künstlich vervielfältigt, so daß am letzten Depoteingang ebenfalls nur eine Tour endet. Dann entfällt die hier notwendige Unterscheidung zwischen TSPR und m-TSPR.

¹²siehe Abschnitt 1.1 auf Seite 10

¹³Man kann sich z.B. vorstellen, daß die Servicezeit bei einem Kunden von der Ankunftszeit abhängig ist, etwa, weil für die Entladung von Waren zu bestimmten Zeiten mehr Personal verfügbar ist o.ä. Auch die Fahrtdauer zwischen zwei Kunden kann stark von der Tageszeit abhängen (z.B. wegen Berufsverkehr, unterschiedlichen Ampelschaltungen, Wartezeiten an Bahnübergängen).

zum jeweiligen Zielort transportiert werden. Ein Transportauftrag w umfaßt die zu transportierende Menge q_w (Ladungsmenge) und die Standorte p_w und d_w , an denen die Ladung aufgenommen (pickup) bzw. abgeliefert werden muß (delivery). Jede Ladung muß von einem Fahrzeug ohne Umladen vom Pickup-Knoten zum Delivery-Knoten transportiert werden. Die Menge aller Pickup-Knoten sei mit $\mathcal{P}$ und die Menge aller Delivery-Knoten mit $\mathcal{D}$ bezeichnet. $\mathcal{P}$ und $\mathcal{D}$ sind disjunkt und enthalten keine Depotknoten.

Wenn die Fahrzeugkapazität auf Q beschränkt ist, können die Aufträge nicht mehr in beliebiger Reihenfolge abgewickelt werden, weil es z.B. notwendig sein kann, eine eingesammelte Ladung erst abzuliefern, bevor eine weitere Ladung aufgenommen werden kann. Sind zusätzlich auch Zeitfenster zu berücksichtigen, kann es notwendig sein, die Aufträge auf mehrere Fahrzeuge zu verteilen. Im allgemeinen Fall erhält man also ein Vehicle Routing Problem mit Zeitfenstern mit Pickups & Deliveries (VRPTWPD).

Für das VRPTWPD ergeben sich folgende logische Konsequenzen: Seien p_w der Pickup-Knoten und d_w der Delivery-Knoten eines Auftrages w , dann müssen p_w und d_w durch ein und dieselbe Tour bedient werden. Dabei muß p_w stets vor d_w besucht werden. Beim Besuch des Pickup-Knotens erhöht sich die Beladung des Fahrzeugs um die Auftragsmenge $q(w)$. Durch das Aufsuchen des zugehörigen Delivery-Knotens wird die Transportkapazität $q(w)$ wieder freigesetzt.

Das VRPTWPD kann mit Hilfe beschränkter Ressourcen modelliert werden. Dazu wird wie beim VRPTW je eine Ressource benötigt, die die Abfahrtszeit bzw. die Beladung des Fahrzeuges an einem Knoten angeben.¹⁴ Als Zeitverbrauch eines Bogens $(i, j) \in A$ wird die Fahrtdauer zwischen den Knoten i und j zuzüglich der Servicezeit am Zielknoten j berücksichtigt. Der Verbrauch der Ressource r_{load} ist abhängig davon, ob ein Pickup-Knoten oder ein Delivery-Knoten besucht wird:

$$d_{ij}^{r_{load}} = \begin{cases} +q_j, & \text{falls } j \in \mathcal{P}, \\ -q_j, & \text{falls } j \in \mathcal{D}, \\ 0, & \text{sonst.} \end{cases}$$

Das Ressourcenfenster von r_{load} ist an allen Knoten gleich:

$$a^{r_{load}} = 0 \text{ und } b^{r_{load}} = Q.$$

Zusätzlich muß gesichert werden, daß die jeweils zu einem Auftrag gehörigen Knoten innerhalb einer Tour und in der richtigen Reihenfolge bedient

¹⁴siehe Seite 14

werden. Dazu wird für jeden Transportauftrag w eine eigene Ressource r_w angelegt. $d_{ij}^{r_w}$ gibt den Verbrauch der Ressource r_w auf dem Bogen $(i, j) \in A$ an. Er wird folgendermaßen festgesetzt:

$$d_{ij}^{r_w} = \begin{cases} -1, & \text{falls } j = p_w \\ +1, & \text{falls } j = d_w \\ 0, & \text{sonst.} \end{cases}$$

Die unteren und oberen Schranken am Knoten $i \in V$ werden wie folgt festgelegt (vgl. Abb. 1.2):

$$a_i^{r_w} = \begin{cases} 1, & \text{falls } i \in \{o, d, p_w\} \\ 0, & \text{sonst} \end{cases} \quad \text{und} \quad b_i^{r_w} = \begin{cases} 0, & \text{falls } i = d_w \\ 1, & \text{sonst.} \end{cases}$$

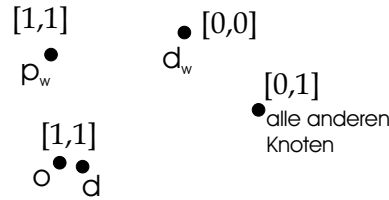


Abbildung 1.2: Ressourcenfenster von r_w für ein VRPTWPD

Durch diese Festlegungen wird erreicht, daß kein Delivery-Knoten d_w besucht werden kann, solange die Ressource des zugehörigen Auftrags r_w noch nicht den Wert eins angenommen hat. Dies kann jedoch nur durch das vorherige Besuchen des Pickup-Knotens p_w erreicht werden.

Multiple Trip VRPTW Beim Multiple Trip VRPTW (MTVRPTW) ist es erlaubt, ein und dasselbe Fahrzeug mehrfach nacheinander einzusetzen. Dies entspricht in vielen Fällen einem realistischen Szenario, bei dem ein Fahrzeug nach seiner Rückkehr zum Depot entladen bzw. erneut beladen, evtl. der Fahrer gewechselt und danach auf eine weitere Tour geschickt werden kann. Gegebenenfalls ist eine Servicezeit für das Umladen etc. zu berücksichtigen. Um die Zeitfenster der Kunden einhalten zu können, müssen evtl. auch beim MTVRPTW mehrere Fahrzeuge eingesetzt werden.

Das Multiple Trip VRPTW kann modelliert werden, indem eine ausreichend große Zahl von Pseudodepotknoten als zusätzliche Kundenknoten eingeführt werden.¹⁵ Diese Pseudodepotknoten sind spezielle Knoten, die eine zwischenzeitliche Rückkehr zum Depot modellieren sollen. Die räumlichen und zeitlichen Entfernungen der Pseudodepots zu den übrigen Kundenknoten sind daher identisch zur Entfernung zum Depotausgang o bzw. dem Depoteingang d . Alle Abstände zwischen Depotausgang, Depoteingang und den Pseudodepots sind null.

Beim Besuch eines Pseudodepots können einzelne Ressourcenvariablen (z.B. die Fahrzeugbeladung) erneut initialisiert werden. Andere Ressourcenvariablen (z.B. die Zeit) bleiben unverändert oder erhöhen sich um einen bestimmten Wert (Servicezeit für das Umladen). Außerdem müssen die Werte aller Ressourcenvariablen unabhängig davon sein, ob direkt nach der Abfahrt vom Depot (oder direkt vor der Rückkehr zum Depot) ein oder mehrere Pseudodepots besucht werden oder nicht. Nicht benötigte Pseudodepotknoten können somit am Beginn (oder am Ende) von Touren positioniert werden, ohne die Zulässigkeit oder den Zielfunktionswert der Lösung zu beeinflussen. All diese Forderungen können durch eine geeignete Wahl der REF und der Ressourcenfenster an den Pseudodepotknoten erfüllt werden.

Beispiel 1.1. Sei P die Menge aller Pseudodepotknoten und p ein Element von P . Die Ressourcenvariable $T_p^{r_{load}}$ gibt die Beladung des Fahrzeugs am Knoten p an. Diese soll auf den Wert 0 zurückgesetzt werden, was wie folgt erreicht wird: Zunächst setzt man das Ressourcenfenster am Knoten p auf $a_p^{r_{load}} = b_p^{r_{load}} = 0$. Damit ist 0 der einzig zulässige Wert von $T_p^{r_{load}}$. Da die Rückkehr zu einem Pseudodepot immer (kapazitiv) zulässig sein soll, darf die REF, die wegen (1.2) eine untere Schranke für $T_p^{r_{load}}$ liefert, auf jedem Bogen, der von einem Knoten i zu einem Pseudodepot p führt, höchstens null sein. Die REF für die Beladung des Fahrzeugs r_{load} kann deshalb gewählt werden als:

$$f_{ij}^{r_{load}}(T_i^R, T_j^R) = \begin{cases} 0, & j \in P, \\ T_i^{r_{load}} + q_j, & j \in N \setminus P. \end{cases}$$

Die REF der übrigen Ressourcen können analog formuliert werden.

¹⁵Die Zahl $n-1$ ist immer ausreichend groß: Falls $n-1$ Pseudodepotknoten zur Verfügung stehen, kann mit einem einzigen Fahrzeug nach jedem Kundenbesuch ein (Pseudo-)Depot bzw. am Ende der Tour der Depoteingang d angesteuert werden. In vielen Fällen wird eine solche Pendeltour-Lösung jedoch nicht zulässig bzw. nicht optimal sein, weshalb auch eine geringere Anzahl ausreichend sein kann. Je nach Wahl des Lösungsverfahrens kann die Zahl der Pseudodepots auch während der Berechnungen dynamisch angepaßt werden.

VRPTW mit simultanen Pickups & Deliveries Beim VRPTW mit simultanen Pickups & Deliveries (VRPTWSPD) müssen die Kunden in zweifacher Hinsicht versorgt werden. Einerseits sind Waren vom Depot zu den Kunden zu liefern, gleichzeitig müssen aber auch (andere) Waren bei den Kunden abgeholt und zum Depot zurückgebracht werden. Jeder Kunde soll trotzdem nur einmal besucht werden, wobei Auslieferung und Abholung simultan stattfinden.¹⁶ Die Fahrzeuge müssen bei der Abfahrt am Depot alle auszuliefernden Waren geladen haben. Durch die Zustellung der Ware bei den Kunden wird zusätzliche Ladekapazität des Fahrzeugs freigestellt, die für die einzusammelnden Waren genutzt werden kann.

Das VRPTW mit simultanen Pickups & Deliveries (VRPTWSPD) wurde erstmals in [Min89] explizit formuliert,¹⁷ aber bislang nur von wenigen Autoren untersucht. Im Zusammenhang mit der Verwendung von Mehrwegverpackungen und der Rücknahme von Umverpackungen bzw. verbrauchten Industriegütern für die Demontage, Aufbereitung oder andere Formen des Recycling gewinnt das VRPTWSPD aber zunehmend an Bedeutung.

Die meisten Veröffentlichungen beziehen sich auf die Spezialfälle TSP bzw. VRP mit Rückholung (TSP/VRP with backhauling).¹⁸ Bei diesen Problemen sind alle Kunden entweder Auslieferungskunden (linehaul customers) oder Abholungskunden (backhaul customers). Bei Auslieferungskunden wird *ausschließlich* Ware zugestellt und bei Abholungskunden *ausschließlich* abgeholt. Zusätzlich wird beim VRPB häufig festgelegt, daß die Abholungen nur am Ende der Tour, d.h. nach den Auslieferungen durchgeführt werden dürfen. Dadurch ist die Zulässigkeit einer Tour nur von den Bedarfen der zugeordneten Kunden abhängig, nicht aber von der (eingeschränkten) Reihenfolge der Kunden, wodurch die Lösung des Problems ggf. deutlich vereinfacht wird.

Ein häufig angeführtes praktisches Beispiel für die strikte Unterteilung der Kunden in Auslieferungs- und Abholungskunden sind Distributionsprobleme in der Lebensmittelbranche. Die Auslieferungskunden sind dort Supermärkte und Geschäfte, Abholungen werden dagegen bei Lebensmittel-Lieferanten durchgeführt. Die Vorgabe der Reihenfolge zwischen Auslieferungs- und Abholungskunden wird damit begründet, daß Auslieferungskunden oft mit hö-

¹⁶„Simultan“ bedeutet hier, daß an ein und demselben Knoten zuerst die auszuliefernden Waren ausgeladen und anschließend die abzuholenden Waren in das Fahrzeug eingeladen werden.

¹⁷Diese Annahme wird in [DDI⁺98] und [Det01] geäußert.

¹⁸siehe u.a. zum TSPB: [AM94], [GHL96] und [GLH97] und zum VRPB: [GJ89], [BM97], [TV99], [MCS92] (multiple Depot), [MGB99] und [GDDS95] (exakte Verfahren)

herer Priorität bedient werden müssen und eine Mischung von Auslieferung und Abholung schwierig ist, weil die Fahrzeuge oft nur von hinten be- und entladen werden können.¹⁹

Das VRPTWSPD kann auch in seiner allgemeinen Form mit Hilfe der REF modelliert werden.²⁰ Gegeben sei ein VRPTW mit der Kundenmenge N . An jedem Kunden $i \in N$ muß die Ladung d_i abgeliefert und die Ladung p_i aufgenommen werden.

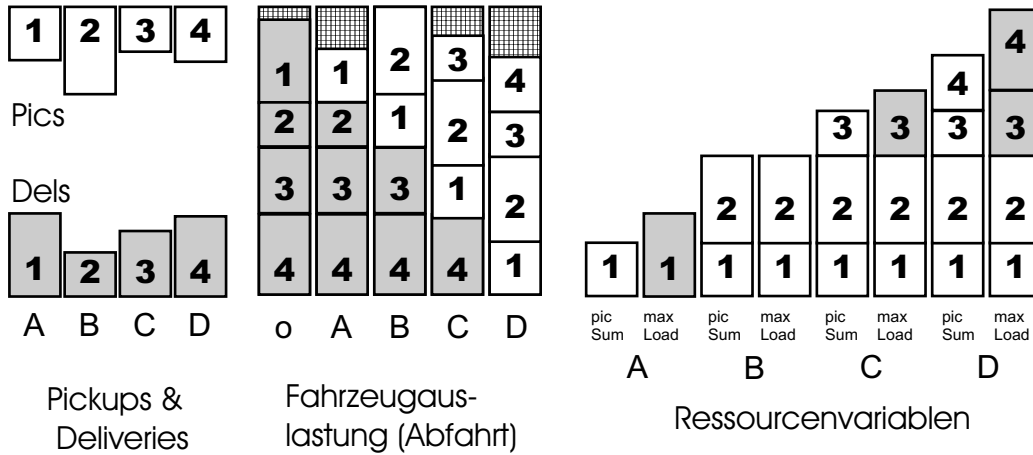


Abbildung 1.3: Ressourcenvariablen T^{picSum} und $T^{maxLoad}$ am Beispiel einer Tour $o \rightarrow A \rightarrow B \rightarrow C \rightarrow D \rightarrow \dots$

Die kapazitive Zulässigkeit einer Tour setzt voraus, daß die Summe der bereits abgeholt zuzüglich der Summe der noch auszuliefernden Waren die Kapazität des Fahrzeugs an keinem Punkt der Tour übersteigt. Für die Überprüfung genügt es nicht, die Summe der eingesammelten und die Summe der ausgelieferten Waren in je einer Ressourcenvariablen zu akkumulieren, weil daraus die Beladung des Fahrzeugs nicht abgeleitet werden kann. Dazu müßte man die Summe der auf der weiteren Tour auszuliefernden Waren kennen. Das ist z.B. beim VRP aber nicht möglich, sofern die Aufteilung der Kunden auf die einzelnen Touren nicht vorher fixiert wird. Deshalb werden zwei Ressourcen verwendet. Die Ressourcenvariable T_i^{picSum} soll die Summe der abgeholt Waren vom Tourbeginn am Depotausgang o bis zum Kunden i

¹⁹siehe [TV99, S. 529], [Det01, S. 83]

²⁰Eine analoge Modellierung wird in [DDI⁺98, S. 71] angegeben.

beinhalten. Die Ressourcenvariable $T_i^{maxLoad}$ gibt (eine untere Schranke für) die maximale Beladung des Fahrzeugs seit der Abfahrt am Depot an, wenn nur die bis zum Kunden i durchgeführten Pickups und Deliveries berücksichtigt werden (siehe Abb. 1.3).

Um nur zulässige Touren zu erhalten, werden die Ressourcenvariablen beschränkt durch

$$\begin{aligned} 0 \leq T_i^{picSum} &\leq Q \quad \text{für alle } i \in V \quad \text{und} \\ 0 \leq T_i^{maxLoad} &\leq Q \quad \text{für alle } i \in V. \end{aligned}$$

Die maximale Beladung $maxLoad$ des Fahrzeugs auf einer Tour, die die Knoten $(v_r, v_{r+1}, \dots, v_{s-1}, v_s)$ miteinander verbindet, kann bestimmt werden durch

$$maxLoad_{(v_r, \dots, v_s)} = \max_{k=r, \dots, s} \left\{ \sum_{h=r}^k p_{v_h} + \sum_{h=k+1}^s d_{v_h} \right\}$$

bzw. rekursiv formuliert

$$maxLoad_{(v_r, \dots, v_s)} = \max \left\{ maxLoad_{(v_r, \dots, v_{s-1})} + d_{v_s}, \sum_{h=r}^s d_{v_h} \right\}$$

Durch die Wahl der REF gemäß

$$\begin{aligned} f_{ij}^{picSum} &= T_i^{picSum} + p_i \\ f_{ij}^{maxLoad} &= \max(T_i^{maxLoad} + d_j, T_i^{picSum} + p_j) \end{aligned}$$

kann daher sichergestellt werden, daß die Fahrzeugkapazität an keinem Knoten überschritten wird.²¹

²¹Eine alternative Modellierung für ein TSP mit simultanen Pickups & Deliveries, wird in [KS92] vorgeschlagen. Die Autoren setzen in ihrem Modell voraus, daß es möglich ist, eine abgeholte Ware auch bei einem anderen Kunden abliefern zu können. Savelsbergh verwendet drei „globale Variablen“, die nach einem speziellen Schema aktualisiert werden.

Kapitel 2

Verbesserungsheuristiken

Viele kombinatorische Optimierungsprobleme sind $\mathcal{NP}$ -schwer, d.h. unter der bis heute nicht bewiesenen, aber gemeinhin vertretenen Annahme $\mathcal{P} \neq \mathcal{NP}$ kann der Aufwand zur Bestimmung einer optimalen Lösung bei Verwendung eines deterministischen Algorithmus nicht durch ein Polynom in der Größe des Problems abgeschätzt werden (siehe [GJ79]). Daher müssen für die Lösung großer Probleme oft heuristische Verfahren anstelle von optimierenden Verfahren eingesetzt werden. Diese erlauben es, mit vertretbarem Rechenaufwand „gute“ Lösungen zu finden.

Meistens ist es nicht möglich, Heuristiken anzugeben, die in der Lage sind, „direkt“ sehr gute Lösungen zu erzeugen. Deshalb werden heuristische Verfahren für kombinatorische Optimierungsprobleme fast immer zweibzw. mehrstufig angelegt. Man beginnt mit einem konstruktiven Verfahren (Eröffnungsverfahren) und erzeugt heuristisch eine zulässige Lösung. In einer zweiten Stufe wird versucht, die bereits gefundene Lösung systematisch zu verbessern. Dazu bedient man sich eines sogenannten Verbesserungsverfahrens, das als Eingabe eine zulässige Lösung erwartet und daraus eine bessere Lösung zu bestimmen versucht. Es ist oft sinnvoll, mehrere Verbesserungsverfahren miteinander zu kombinieren, so daß sich dadurch auch drei- oder mehrstufige Verfahren ergeben.

Die Nachbarschaftssuche bzw. Lokale Suche¹ kann als ein Prinzip zur Entwicklung eines Verbesserungsverfahrens aufgefaßt werden. Ein großer Teil der in der Literatur veröffentlichten Verbesserungsverfahren für Routing Probleme basiert ganz oder teilweise auf diesem Prinzip.

¹in der englischsprachigen Literatur „local search“ oder auch „descent heuristic“

Mit diesem Kapitel wird versucht, eine Übersicht über die wichtigsten auf der Lokalen Suche basierender Verbesserungsverfahren aus einer vom konkreten Problem noch weitgehend losgelösten Sicht zu geben. Dies ermöglicht es einerseits, gemeinsame Schemata und Strukturen vieler (in der Literatur teilweise unterschiedlich dargestellter) Ansätze deutlich zu machen. Andererseits läßt sich so erkennen, daß einige Konzepte durchaus verallgemeinert bzw. fortgesetzt werden könnten, obwohl dies bislang nicht verfolgt wurde.

2.1 Nachbarschaften für kombinatorische Optimierungsprobleme

Um das Vorgehen der Lokalen Suche beschreiben zu können, sollen zunächst die zentralen Begriffe „Kombinatorisches Optimierungsproblem“ und „Nachbarschaft“ definiert werden. Dabei sind die nachfolgenden Definitionen 2.1–2.4 zum Teil aus [AL97, S. 3f.] übernommen bzw. erfolgen in enger Anlehnung an die dort vorgenommene Modellierung. Im Gegensatz zu [AL97] wird im Rahmen dieser Arbeit eine explizite Unterscheidung zwischen der Menge aller Lösungen und der Menge der zulässigen Lösungen eines Problems benötigt. Dies macht auch eine entsprechende Erweiterung des Nachbarschaftsbegriffs (Definition 2.5) notwendig.

Definition 2.1. (Kombinatorisches Optimierungsproblem)

Ein kombinatorisches Optimierungsproblem wird durch die Angabe einer Menge von Probleminstanzen dargestellt und ist entweder ein Minimierungs- oder ein Maximierungsproblem.

Ohne Beschränkung der Allgemeinheit betrachten wir nachfolgend ausschließlich Minimierungsprobleme.

Definition 2.2. (Instanz eines kombinatorischen Optimierungsproblems)

Eine Instanz $\mathcal{P}$ eines kombinatorischen Optimierungsproblems ist ein Paar (Y, c) . Dabei ist Y die Menge der zulässigen Lösungen von $\mathcal{P}$ und die Kostenfunktion c ist eine Abbildung $c : Y \rightarrow \mathbb{R}$.

Das Problem besteht darin, eine global optimale Lösung zu bestimmen, d.h., ein

$y^* \in Y$ mit $c(y^*) \leq c(y)$ für alle $y \in Y$:

$$\mathcal{P} : \quad \min \quad c(y) \quad (2.1a)$$

so daß

$$y \in Y \quad (2.1b)$$

Die Kosten $c^* = c(y^*)$ heißen optimale Kosten. Die Menge $Y^* = \{y \in Y : c(y) = c^*\}$ ist die Menge aller optimalen Lösungen von $\mathcal{P}$.

Die Instanz eines kombinatorischen Optimierungsproblems wird i.d.R. nur implizit angegeben, d.h. es existiert eine kompakte Datenrepräsentation x und ein Algorithmus mit polynomialem Laufzeitverhalten, um festzustellen, ob x eine zulässige Lösung von $\mathcal{P}$ repräsentiert.²

Wir bezeichnen die Menge X , die alle Datenrepräsentationen x umfaßt, als Lösungsmenge von $\mathcal{P}$. Die Menge Y der zulässigen Lösungen von $\mathcal{P}$ ist in X enthalten: $Y \subseteq X$.

Die Lösungsmenge X wird oft mit Hilfe einer Menge von Entscheidungsvariablen dargestellt. Eine Lösung $x \in X$ wird dann durch die Zuordnung von Werten zu den einzelnen Variablen repräsentiert. Die Größe der Repräsentation einer Lösung (z.B. gemessen an der Zahl der benötigten Bits für die Darstellung auf einem Computer) wird auch als Größe der Problem Instanz interpretiert.²

Definition 2.3. (Nachbarschaft)

Sei $\mathcal{P} = (Y, c)$ eine Instanz eines kombinatorischen Optimierungsproblems. Eine Nachbarschaftsfunktion ist eine Abbildung $\mathcal{N} : Y \rightarrow \text{Pot}(Y)$ ³, die für jede zulässige Lösung $y \in Y$ eine Menge $\mathcal{N}(y) \subseteq Y$ definiert, deren Elemente in einem bestimmten Sinne „in der Nähe“ von y liegen.

Die Menge $\mathcal{N}(y)$ heißt die Nachbarschaft von y und jede Lösung $y' \in \mathcal{N}(y)$ heißt Nachbar von y .

Wir gehen davon aus, daß jede zulässige Lösung in ihrer eigenen Nachbarschaft enthalten ist, d.h. es gilt $y \in \mathcal{N}(y)$ für alle $y \in Y$.

Definition 2.4. (Lokale Optimalität)

Sei $\mathcal{P} = (Y, c)$ eine Instanz eines kombinatorischen Optimierungsproblems

²vgl. [AL97, S.3f.]

³Pot(Y) bezeichnet die Potenzmenge von Y .

und $\mathcal{N}$ eine Nachbarschaftsfunktion. Eine zulässige Lösung $y \in Y$ heißt lokal optimal in Bezug auf die Nachbarschaft $\mathcal{N}$, falls gilt

$$c(y) \leq c(y') \quad \text{für alle } y' \in \mathcal{N}(y).$$

Auch die Nachbarschaftsfunktion $\mathcal{N}$ wird häufig nicht explizit gemäß der Definition 2.3 angegeben. Stattdessen bezieht man sich auf die bereits angesprochene Datenrepräsentation x einer Lösung und gibt eine Menge von Operatoren vor, die als Nachbarschaftsschritte bezeichnet werden. Dadurch läßt sich eine erweiterte Nachbarschaft $\hat{\mathcal{N}}(x)$ beschreiben, die aus der Menge aller Lösungen besteht, die durch Anwendung eines Nachbarschaftsschritts auf die Lösung x erzeugt werden können.

Definition 2.5. (Erweiterte Nachbarschaft)

Sei $\mathcal{P} = (Y, c)$ eine Instanz eines kombinatorischen Optimierungsproblems und $\mathcal{N}$ eine Nachbarschaftsfunktion. Dann heißt $\hat{\mathcal{N}} : X \rightarrow \text{Pot}(X)$ eine erweiterte Nachbarschaftsfunktion von $\mathcal{N}$, wenn für alle $y \in Y$ gilt: $\hat{\mathcal{N}}(y) = \mathcal{N}(y)$. Die Menge $\hat{\mathcal{N}}(x) \subseteq X$ heißt erweiterte Nachbarschaft von x .

Bemerkung 2.6. Seien $y \in Y$ und $\hat{\mathcal{N}}(y)$ bekannt, so läßt sich die Menge $\mathcal{N}(y)$ bestimmen als

$$\mathcal{N}(y) = \hat{\mathcal{N}}(y) \cap Y. \quad (2.2)$$

Zu einer Nachbarschaft $\mathcal{N}(y)$ kann es i.a. verschiedene erweiterte Nachbarschaften geben. Da letztlich nur zulässige Lösungen von $\mathcal{P}$ interessieren, betrachtet man meistens eine möglichst kleine erweiterte Nachbarschaft $\hat{\mathcal{N}}$ und kann die Menge $\mathcal{N}(y)$ dadurch bestimmen, daß man die Zulässigkeit der Elemente von $\hat{\mathcal{N}}(y)$ explizit überprüft.

Definition 2.7. (Nachbarschaftsschritt)

Gegeben sei eine Instanz $\mathcal{P} = (Y, c)$ eines kombinatorischen Optimierungsproblems, $\mathcal{N}$ eine Nachbarschaftsfunktion von Y und $\hat{\mathcal{N}}$ eine Erweiterung von $\mathcal{N}$. Ein Nachbarschaftsschritt ist ein Operator $\mathbf{m}^{\hat{\mathcal{N}}} : X \rightarrow X$, mit

$$\mathbf{m}^{\hat{\mathcal{N}}}(x) \in \hat{\mathcal{N}}(x) \quad \text{für alle } x \in X.$$

Ausgehend von einer Startlösung $y_0 \in Y$ kann das Vorgehen einer iterativen Lokalen Suche durch folgenden Algorithmus beschrieben werden:

Algorithmus 2.1. Lokale Suche

1. $y^{akt} := y_0$
2. Bestimme eine verbessernde Lösung y' mit $y' \in \mathcal{N}(y^{akt})$.
3. Falls keine verbessernde Lösung y' gefunden werden konnte $\rightarrow$ STOPP.
4. $y^{akt} := y'$, gehe zu 2.

Die letzte gefundene Verbesserung y^{akt} ist die Lösung der Lokalen Suche. Diese Lösung ist lokal optimal in bezug auf die Nachbarschaft $\mathcal{N}$.⁴

Da das Konzept der Lokalen Suche sehr allgemein ist, können auf dessen Basis sehr verschiedene Verbesserungsverfahren konstruiert werden. Diese unterscheiden sich in erster Linie in der Definition der Nachbarschaft $\mathcal{N}$ (bzw. $\hat{\mathcal{N}}$). Darüber hinaus gibt es mehrere Möglichkeiten, wie der Schritt 2 des obigen Algorithmus realisiert werden kann. So ist einerseits eine „Erstensuche“ möglich, d.h. der Schritt wird abgebrochen, sobald eine einzige verbessernde Lösung in $\mathcal{N}(y^{akt})$ gefunden wurde. Alternativ können aber auch mehrere oder alle verbessernden Lösungen der Nachbarschaft bestimmt werden, um daraus die beste Nachbarlösung auszuwählen. Dieses Vorgehen wird auch als „Bestensuche“ bezeichnet.⁵

Bei der Wahl der Nachbarschaft muß ein Kompromiß zwischen der gewünschten Qualität der Lösungen und der Rechendauer der Lokalen Suche gefunden werden. Tendenziell gilt:

- je größer die Nachbarschaft, desto besser die Qualität der Lösungen
- je kleiner die Nachbarschaft, desto geringer der Rechenaufwand für die Durchführung der Lokalen Suche.

Es sei explizit darauf hingewiesen, daß es sich hierbei lediglich um *tendenzielle* Aussagen handelt. Insbesondere kann man nicht davon ausgehen, daß eine

⁴ Der Begriff „Lokale Suche“ wird von einigen Autoren noch allgemeiner gefaßt. Danach muß im Schritt 2 nicht unbedingt eine verbessernde Lösung aus der Nachbarschaft ausgewählt werden, und die Abbruchbedingung im Schritt 3 kann auch anders formuliert werden. Dann können Meta-Heuristiken wie z.B. Simulated Annealing ebenfalls als Lokale Suche aufgefaßt werden. Dennoch ist es üblich, den Begriff der Lokalen Suche auf ein reines Abstiegsverfahren zu beziehen, wie es hier auch getan wird.

⁵Bei vielen Suchverfahren erweist es sich als effizient, bereits die erste gefundene Verbesserung zu benutzen. Man beachte, daß es in diesem Fall von Bedeutung ist, in welcher Reihenfolge die Nachbarschaft der aktuellen Lösung y^{akt} abgesucht wird. Insbesondere für die Reproduzierbarkeit von Rechenergebnissen ist es zwingend erforderlich, daß diese Reihenfolge *genau* angegeben wird.

Vergrößerung der Nachbarschaften grundsätzlich zu einer Verbesserung der Ergebnisse führen muß. Es kann im Gegenteil sogar eine Verschlechterung der Ergebnisse zur Folge haben. Wie das folgende Beispiel zeigt, gilt dies selbst dann, wenn die größere Nachbarschaft in jedem Schritt so gewählt ist, daß die kleinere Nachbarschaft darin enthalten ist.

Beispiel 2.1. Wir betrachten drei unterschiedlich große Nachbarschaften in einem eindimensionalen Lösungsraum (siehe Abb. 2.1). Die Größe der Nachbarschaften drückt sich in diesem Beispiel in der *maximalen* Schrittweite aus. Es wird in jedem Schritt die beste Nachbarlösung gesucht.

Von der Startlösung 0 ausgehend, führt die Lokale Suche mit ausschließlich kleinen Schritten zum lokalen Minimum b. Die zweite Nachbarschaft erlaubt eine größere Schrittweite. Deshalb wird ein Weg mit stärkerem Abstieg gefunden, der aber nur zum (relativ schlechten) lokalen Minimum a führt. Erst unter Verwendung einer ausreichend großen maximalen Schrittweite kann auch das lokale Minimum c gefunden werden.

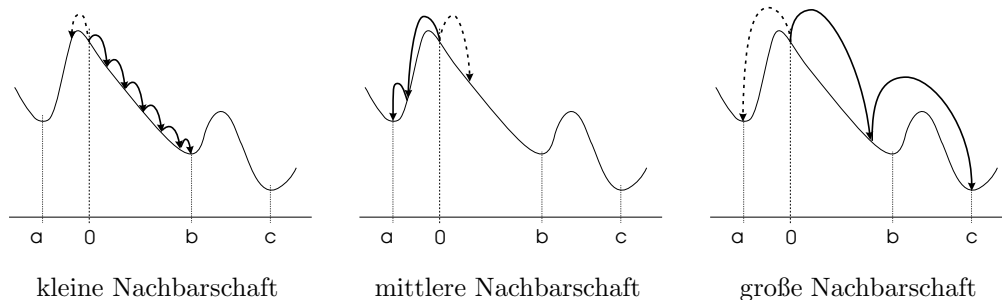


Abbildung 2.1: Beispiel für eine Lokale Suche mit verschiedenen großen Nachbarschaften in einem eindimensionalen Lösungsraum

Die Wahl einer geeigneten Nachbarschaft stellt deshalb oft eine schwierige Aufgabe dar. Aarts und Lenstra schreiben dazu:

*„Neighborhoods depend on the problem under consideration, and finding efficient neighborhood functions that lead to high-quality local optima can be viewed as one of the challenges of local search. So far, no general rules are available and each situation has to be considered separately. ...“*⁶

⁶aus [AL97, S. 4]: Nachbarschaften hängen vom betrachteten Problem ab, und effiziente

Dieser Aussage folgend und dem Fokus dieser Arbeit entsprechend, konzentrieren wir uns im folgenden auf spezielle Nachbarschaften für Vehicle Routing und Scheduling Probleme.

2.2 Austausch-Nachbarschaften für Vehicle Routing und Scheduling Probleme

In diesem Abschnitt wird das Konzept einer Austausch-Nachbarschaft erläutert und eine systematische Darstellung der für VRSP erfolgreichsten Austausch-Nachbarschaften vorgenommen.

Alle in dieser Arbeit betrachteten VRSP lassen sich auf der Basis eines Graphen $G = (V, A, c)$ formulieren, der aus einer Knotenmenge V und einer Menge von Bögen A aufgebaut ist. Knoten stehen dabei für die Standorte von Kunden oder (Fahrzeug-)Depots. Bögen repräsentieren die Verbindungen zwischen den Knoten und können z.B. durch eine (Weg-)Länge, eine Fahrtdauer und Kosten charakterisiert sein. Die Lösungen eines VRSP können deshalb entweder in Form einer Sequenz von Knoten oder durch eine Sequenz von Kanten bzw. Bögen angegeben werden.⁷

Bei einer Austausch-Nachbarschaft werden die Nachbarn dadurch gebildet, daß k Elemente dieser Sequenz (also k Knoten oder k Bögen) aus der aktuellen Lösung entfernt und durch k (andere) Knoten bzw. Bögen ersetzt werden. Prinzipiell ist es zugelassen, daß ein oder mehrere der k ausgewählten Elemente auch mit sich selbst getauscht (also praktisch nicht verändert) werden. Somit sind in einer k -Austausch-Nachbarschaft auch die kleineren Nachbarschaften enthalten, die den Austausch von $1, 2, \dots, (k - 1)$ Elementen zulassen. Soll dies ausgeschlossen werden, so sprechen wir von einer *echten* k -Austausch-Nachbarschaft. Handelt es sich bei den Elementen um Bögen, so werden diese normalerweise gegen solche ausgetauscht, die zuvor nicht in der Lösung enthalten waren.

Bei einer Sequenzlänge von n gibt es $\binom{n}{k}$ Möglichkeiten, k Elemente für einen Austausch auszuwählen. Daher beträgt der Aufwand für das Enume-

Nachbarschaftsfunktionen zu finden, die zu lokalen Optima hoher Qualität führen, kann als eine der Herausforderungen der Lokalen Suche betrachtet werden. Bislang sind keine allgemeinen Regeln verfügbar und jede Situation muß separat untersucht werden. ...

⁷Eine explizite Modellierung erfolgt in den Abschnitten 1.2.2 und 3.1.3.

rieren aller Lösungen einer Austausch-Nachbarschaft mindestens $\mathcal{O}(n^k)$. Deshalb werden in der Praxis fast ausschließlich Nachbarschaften mit kleinem k betrachtet.

Sind die k Elemente einmal ausgewählt, so gibt es i.d.R. immer noch viele Möglichkeiten, *wie* der Austausch dieser Elemente vorgenommen werden kann, was zu einer weiteren Vergrößerung der betrachteten Nachbarschaften führt. Deshalb werden für den Austausch dieser Elemente häufig Einschränkungen gemacht, die die Vielfalt der möglichen Austausch-Schritte und damit die Größe der Nachbarschaft erheblich reduzieren können. Je nach Art dieser Einschränkungen können sehr unterschiedliche Nachbarschaften entstehen, deren wichtigste Vertreter nachfolgend dargestellt werden.

Wir unterscheiden die Nachbarschaften in erster Linie danach, ob Kanten oder Knoten getauscht werden. Wir sprechen dann von kantenorientierten Nachbarschaften bzw. von knotenorientierten Nachbarschaften.⁸ Alternativ werden Austausch-Nachbarschaften in der Literatur oft danach charakterisiert, ob Austausche nur innerhalb einer Tour bzw. auch zwischen zwei oder mehr Touren betrachtet werden.⁹ Man spricht dann von Intra-Tour-Austauschen bzw. von Inter-Tour-Austauschen.¹⁰ Diese Einteilung entspricht aber weitgehend der Unterteilung in kanten- und knotenorientierte Nachbarschaften, denn viele erfolgreiche knotenorientierte Nachbarschaften tauschen Knoten zwischen zwei oder mehr Touren aus. Kantenaustausch-Nachbarschaften orientieren sich dagegen häufig an einer einzelnen Tour. Daß dies jedoch nicht zwingend so sein muß, wird u.a. im Kapitel 3 gezeigt werden. Wir ziehen deshalb die Unterscheidung nach kantenorientierten bzw. knotenorientierten Nachbarschaften vor.

2.2.1 Knotenorientierte Austausch-Nachbarschaften

Bisher wurde nur sehr allgemein davon gesprochen, daß bei einem knotenorientierten Nachbarschaftsschritt k Knoten *ausgetauscht* werden. Da in einer

⁸Prinzipiell kann man jeden Schritt einer knotenorientierten Nachbarschaft durch einen äquivalenten Schritt einer kantenorientierten Nachbarschaft erzeugen (und umgekehrt). Dennoch ist diese Unterscheidung sinnvoll, denn bereits beim Austauschen weniger Bögen kann man z.B. Lösungen erzeugen, bei denen viele Knoten gleichzeitig vertauscht werden.

⁹Unter einer Tour wird ein Pfad verstanden, der an einem Depotknoten beginnt und an einem Depotknoten endet. Eine Lösung eines VRSP setzt sich i.a. aus mehreren Touren zusammen. Eine Definition des Tourbegriffs erfolgt im Abschnitt 3.1.

¹⁰vergleiche [KS97], [Gie94, S.26]

Lösung eines VRSP stets alle Kunden genau einmal bedient werden müssen (Überdeckungsbedingung (1.1b)), kann eine Lösung durch eine Knotensequenz repräsentiert werden, die alle Knoten beinhalten muß, wobei Depotknoten ggf. mehrfach auftreten können. Unter dem Austausch eines Knotens soll das Transferieren des Knotens an eine neue Position innerhalb der Knotensequenz verstanden werden.

In der Regel beziehen sich knotenorientierte Nachbarschaften auf das Verschieben von Knoten zwischen zwei oder mehreren verschiedenen Touren der Lösung, so daß praktisch ein Austausch von Knoten zwischen diesen Touren stattfindet.

Prinzipiell lassen sich alle derartigen Nachbarschaften dahingehend erweitern, daß anstelle verschiedener Touren auch disjunkte Tourabschnitte betrachtet werden, zwischen denen Knoten ausgetauscht werden. Darauf wird in der Literatur jedoch kaum eingegangen, was durch folgende Aspekte begründet werden könnte: Erstens stellt die Auswahl geeigneter Tourabschnitte eine zusätzliche schwierige Aufgabe dar, die durch die Beschränkung auf *komplette* existierende Touren vereinfacht bzw. soweit reduziert wird, daß normalerweise einfach alle Varianten ausprobiert werden können. Zweitens werden knotenorientierte Nachbarschaften häufig beim VRP eingesetzt, um eine Reduktion der Tourenzahl dadurch zu erreichen, daß alle Knoten einer Tour auf andere Touren verteilt werden. Das Verschieben von Knoten innerhalb einer Tour kann dazu aber nicht direkt beitragen.

α^* -Notation für Knotenaustausch-Nachbarschaften In der Literatur existieren bislang keine einheitlichen Bezeichnungen, und auch die Form der Beschreibung von knotenorientierten Nachbarschaften ist sehr unterschiedlich. Bevor ein Überblick der häufig verwendeten knotenorientierten Nachbarschaften gegeben wird, wird daher zunächst eine einfache Notation vorgestellt, die es erlaubt, eine große Klasse knotenorientierter Nachbarschaftsschritte einheitlich und kompakt zu beschreiben. Diese Notation orientiert sich einerseits an der von Taillard in [Tai93] beschriebenen (M, P) -Notation und andererseits an der Idee der Cyclic Transfers¹¹. Viele der in der Literatur verwendeten Nachbarschaftsschritte lassen sich damit darstellen.

Wir beschreiben einen Knotenaustausch zwischen l Touren bzw. disjunkten Tourabschnitten anhand eines Vektors $\alpha = (\alpha_1, \dots, \alpha_l)$ mit $\alpha_i \in \mathbb{N}$ für alle $i \in \{1, \dots, l\}$. Ein Nachbarschaftsschritt α wird folgendermaßen durch-

¹¹siehe Seite 37

geführt: Man wählt l Touren (oder disjunkte Tourabschnitte) der Ausgangslösung, die wir als $\sigma_1, \dots, \sigma_l$ bezeichnen. Dann werden gleichzeitig jeweils $\min(\alpha_i, |\sigma_i|)$ Knoten von σ_i in den Nachfolger σ_{i+1} verschoben. σ_l hat dabei den Nachfolger σ_1 .

Oft werden Nachbarschaften betrachtet, bei denen α_i nur die maximale Zahl der Knoten angibt, die aus σ_i in den Nachfolger verschoben werden dürfen. Dies wird durch einen zusätzlichen $*$ gekennzeichnet, d.h. im Vektor steht α_i^* anstelle von α_i . Eine häufige Einschränkung der Nachbarschaften ist außerdem, daß die Knoten, die aus σ_i nach σ_{i+1} verschoben werden, die vorherigen Positionen der Knoten einnehmen müssen, die aus σ_{i+1} verschoben werden. Wir notieren dies durch einen Unterstrich unter α_i .

Eine Austausch-Nachbarschaft kann durch Angabe eines Vektors oder durch das Aufzählen mehrerer Vektoren dieser Notation beschrieben werden.

Beispiel 2.2. α^* -Notation für Knotenaustausch-Nachbarschaften

Notation	Bedeutung
$(3,2,1,0)$	Verschiebe drei Knoten von σ_1 nach σ_2 , zwei Knoten von σ_2 nach σ_3 , einen Knoten von σ_3 nach σ_4 aber keinen Knoten von σ_4 nach σ_1 .
$(3,\underline{3}^*)$	Verschiebe drei Knoten von σ_1 nach σ_2 . Diese seien mit v_1, v_2 und v_3 bezeichnet. Parallel werden <i>maximal</i> drei Knoten von σ_2 nach σ_1 verschoben. Diese Knoten dürfen aber nur an den Positionen eingefügt werden, die zuvor von v_1, v_2 oder v_3 eingenommen wurden.
$(\underline{1},\underline{1}),(\underline{2},\underline{2})$	Es werden entweder je genau ein Knoten aus σ_1 und σ_2 oder je genau zwei Knoten gegeneinander getauscht. Beim Einfügen muß jeder Knoten an die Position eines gerade entfernten Knotens gesetzt werden.

Relocation Die einfachste Form einer Knotenaustausch-Nachbarschaft beinhaltet alle Lösungen, die durch den Austausch bzw. das Transferieren genau eines Knotens entstehen. Dieser Austausch-Schritt wird als Relocation bezeichnet.¹² In der α^* -Notation wird ein Relocation-Schritt durch $(1,0)$ bzw. durch $(0,1)$ beschrieben.

Van Breedam gibt in [vB01] eine Erweiterung des einfachen Relocation-Schritts an, die er String-Shift nennt. Ein String-Shift verschiebt anstelle eines einzelnen Knotens einen String, d.h. eine Kette benachbarter Knoten einer Tour. Es kann die maximale Länge α_1 des zu verschiebenden Strings vorge-

¹²siehe [AL97]

geben werden. Aarts und Lenstra bezeichnen denselben Schritt in [AL97] als Path-Relocation.

Exchange Unter einem Exchange-Schritt wird häufig der Austausch je eines Knotens zwischen zwei Touren verstanden, d.h. ein Nachbarschaftsschritt der Form (1,1). Oft wird jedoch zusätzlich gefordert, daß die gegeneinander getauschten Knoten nur die vorherige Position des jeweils anderen Knotens einnehmen dürfen, d.h. es werden nur Schritte der Form (1,1) erlaubt.

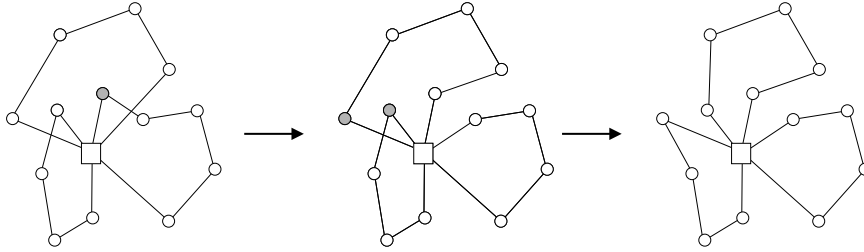


Abbildung 2.2: Ein Relocate- und ein Exchange-Schritt beim VRP

Analog zum String-Relocation kann man auch einen String-Exchange betrachten. Dabei werden zwei Strings gegeneinander ausgetauscht. Van Bredam fordert in [vB01] auch für den String-Exchange, daß die getauschten Strings nur die Position des jeweils anderen Strings einnehmen dürfen und in der neuen Tour nicht frei positioniert werden können.

λ -Interchange Der λ -Interchange wurde erstmalig von Osman in [Osm93] beschrieben und nachfolgend von einer Reihe weiterer Autoren eingesetzt (z.B. [Tha93], [TOVS93], [CR96],[OS96]). Die Bezeichnung λ -Interchange steht für alle Austausch-Schritte, die jeweils bis zu λ Knoten zwischen zwei Touren verschieben, d.h. für Schritte der Form (λ^*, λ^*) .

Zum 1-Interchange gehören also die Schritte $(1^*, 1^*)$, d.h. die Relocation-Schritte $(1,0)$ und $(0,1)$ und die Exchange-Schritte $(1,1)$. Der 2-Interchange umfaßt bereits acht verschiedenartige Nachbarschaftsschritte: $(1,0)$, $(0,1)$, $(1,1)$, $(0,2)$, $(2,0)$, $(1,2)$, $(2,1)$ und $(2,2)$. Wegen der schnell wachsenden Größe der λ -Interchange-Nachbarschaften ($\mathcal{O}(n^{4\lambda})$) werden in der Praxis häufig nur Nachbarschaften für $\lambda \leq 2$ betrachtet.

Russells Interchange Russell beschreibt in [Rus95b] eine beschränkte Austausch-Nachbarschaft zwischen mehreren Touren. Seine Intension ist, eine relativ große Nachbarschaft zuzulassen, die Suche aber auf „aussichtsreiche“ Austausche zu beschränken.

Betrachtet wird zunächst nur eine Tour t_j . Innerhalb dieser Tour werden l direkt benachbarte Knoten ($l = 0, 1, 2$) ausgewählt. Dies seien der Knoten i und seine l direkten Nachfolger bis zum Knoten $i+l$. Zusätzlich werden zwei weitere Knoten aus den übrigen Touren ausgewählt (t_j ausgeschlossen), bei denen die Summe der (euklidischen) Distanzen zu $i-1$ (dem Vorgänger von i) und $i+l+1$ (dem Nachfolger von $i+l$) am geringsten ist. Für jeden der nun insgesamt $l+2$ gewählten Knoten werden jetzt zwei bestgeeignete Touren für die Einfügung dieses Knotens bestimmt. Schließlich werden alle Lösungen untersucht, die sich bilden lassen, indem die $l+2$ gewählten Knoten entfernt und in eine der ihnen zugeordneten Einfüge-Touren eingebaut werden.

Node Ejection Chains Das Konzept der Ejection Chains wurde von Glover in [Glo92a] bzw. [Glo92b] beschrieben und nachfolgend für die Lösung verschiedener kombinatorischer Optimierungsprobleme eingesetzt.

Rego wendet die Ejection Chains in [Reg01] auf einzelne Knoten beim VRP an und nennt die so erhaltenen Nachbarschaften Node Ejection Chains (NEC). Bei einer NEC wird ein beliebiger Knoten von seiner aktuellen Position vertrieben (ejection) und an die Position eines zweiten Knotens verschoben. Dieser Knoten wird dadurch ebenfalls von seiner aktuellen Position vertrieben und an die Position eines dritten Knotens verschoben usw. Rego unterscheidet zwei Typen von Ejection Chains, die er „multi-node exchange process“ und „multi-node insert process“ nennt. Beide unterscheiden sich ausschließlich in der letzten Stufe der Ejection Chain. Entweder wird der letzte vertriebene Knoten in eine noch nicht betrachtete Position eingefügt oder an der Position des ersten vertriebenen Knotens (siehe die Abb. 2.3).

Die einzelnen verschobenen Knoten müssen nicht notwendig in verschiedenen Touren enthalten sein. Stattdessen wird eine „legitimacy condition“ formuliert. Diese beinhaltet sinngemäß:

- kein Knoten darf mehrfach verschoben werden und
- die direkten Vorgänger und Nachfolger eines verschobenen Knotens dürfen nicht verschoben werden.

Wählt man die Knoten aus Tourabschnitten, die der „legitimacy condition“ genügen, so kann man beide Typen der Ejection Chain auch mit der

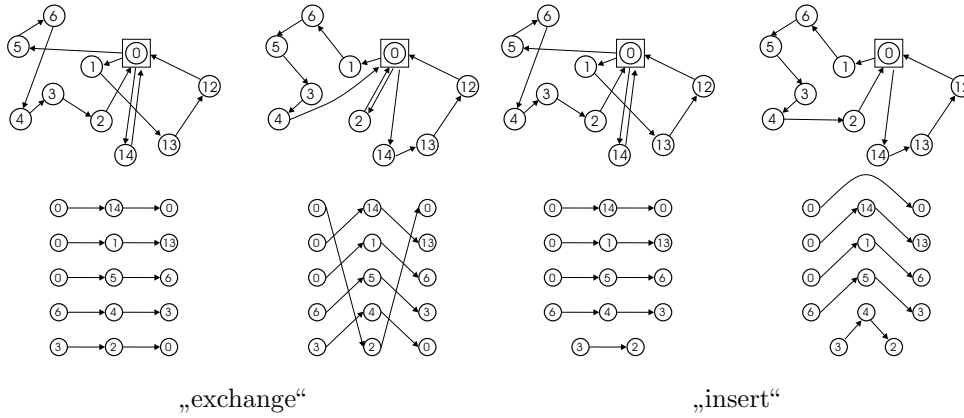


Abbildung 2.3: Node Ejection Chain

α^* -Notation beschreiben. Ein Nachbarschaftsschritt α einer NEC der Stufe l hat dann die Form

$$\alpha = (\alpha_1, \dots, \alpha_l) = (\underline{1}, \dots, \underline{1}, \underline{1}^*).$$

Für $\alpha_l = \underline{1}$ erhält man den „multi-node exchange process“, und $\alpha_l = 0$ beschreibt den „multi-node insert process“. Wie das Beispiel in Abb. 2.3 zeigt, kann mit einem „insert“-Schritt ggf. eine Pendeltour eingespart werden. Durch mehrere aufeinanderfolgende „insert“-Schritte können prinzipiell auch längere Touren Knoten für Knoten verkürzt und schließlich ganz aufgelöst werden.

Rego schränkt die Auswahl der zu verschiebenden Knoten zusätzlich ein, indem er fordert, daß die in der Ejection Chain aufeinanderfolgenden Knoten zu den k nächsten Nachbarn gehören müssen.

Eine Erweiterung der NEC's sind die „Subpath Ejection Chains“, die ebenfalls von Rego in [Reg96] für das TSP bzw. in [Reg98] für das VRP beschrieben wurden. Anstelle von einzelnen Knoten werden hier Teilpfade verschoben. Dies geschieht, indem die entsprechenden Kanten ersetzt werden, weshalb dieses Verfahren eher als Kantenaustausch-Nachbarschaft aufgefaßt werden kann.

Cyclic Transfers Der Begriff der Cyclic Transfers wurde 1993 von Thompson und Psaraftis eingeführt [TP93]. Mit Cyclic Transfers läßt sich eine relativ große Klasse von knotenorientierten Nachbarschaften beschreiben. Cyclic

Transfer-Schritte sind dadurch gekennzeichnet, daß ein oder mehrere Knoten zyklisch zwischen verschiedenen Touren verschoben werden.

Sei $b, m \in \mathbb{N}$ und T die Menge aller Touren der Ausgangslösung und $T^b = \{t_1, \dots, t_b\}$ eine b -elementige Teilmenge von T . Thompson und Psaraftis nennen einen Nachbarschaftsschritt einen „ b -cyclic m -transfer“, falls dieser je m Knoten der Tour t_i in die Tour $t_{(i+1)}$ bzw. von Tour t_b nach Tour t_1 verschiebt.¹³

Ein b -cyclic m -transfer ist demnach ein Austausch von $b * m$ Knoten und läßt sich schreiben als $\alpha = (\alpha_1, \dots, \alpha_b) = (m, \dots, m)$. Die Autoren erlauben auch das Verschieben von „dummy demands“. Damit wird ermöglicht, daß auch weniger als m Knoten aus einer Tour herausgeschoben werden, was sich schreiben läßt als $(m^*, \dots, m^*)$.

Zunächst wird gefordert, daß die verschobenen Knoten in der neuen Tour optimal eingebaut werden müssen. Dadurch lassen sich die Kosten der Einfügung eindeutig festlegen. Da das TSP aber ein $\mathcal{NP}$ -schweres Problem ist, verwenden die Autoren bei der praktischen Realisierung eine Approximation, mit deren Hilfe die Kosten beim Einfügen eines einzelnen Knotens in seine neue Tour abgeschätzt werden.

Da die zugehörige Cyclic-Transfer-Nachbarschaft sehr schnell wächst, können praktisch auch nur sehr kleine Werte für b und m gewählt werden. Sonst würde allein das Absuchen der Cyclic-Transfer-Nachbarschaft einen sehr hohen Aufwand verursachen. Die Autoren verwenden deshalb zusätzliche Beschränkungen der Nachbarschaft und einen algorithmischen Ansatz, der es erlaubt, m variabel zu halten. So gelingt es ihnen, eine praktikable Lokale Suche auf der Basis der Cyclic Transfers zu implementieren.

Eine alternative Implementierung und Durchführung der Lokalen Suche mit einer Cyclic-Transfer-Nachbarschaft ist Gegenstand von Kapitel 4.

HyperOpt Eine weitere Nachbarschaft wird durch den von Burke, Cowling und Keuthen in [BCK01] beschriebenen HyperOpt-Schritt definiert, den die Autoren beim TSP einsetzen. Mit einem HyperOpt-Schritt können maximal $2 * (l - 1)$ Knoten gleichzeitig transferiert werden. Diese werden wie folgt bestimmt:

¹³Genaugenommen betrachten die Autoren sogar das Verschieben von m Aufträgen, die jeweils auch durch mehrere Knoten repräsentiert werden können (z.B. zwei Knoten beim Pickup & Delivery Problem). Wir gehen hier vereinfachend davon aus, daß ein Auftrag durch einen Knoten repräsentiert werden kann.

Man wählt zwei „Hyper-Kanten“ der Länge l , darunter werden zwei Ketten von je $l + 1$ aufeinanderfolgenden Knoten verstanden. Anschließend werden die inneren Kanten der „Hyperkanten“ gelöscht, so daß die $2 * (l - 1)$ inneren Knoten vollkommen frei liegen. Diese Knoten werden nun unter Verwendung der Dynamischen Programmierung optimal untereinander getauscht und verschoben, wobei alle übrigen Kanten und Knoten der Tour unverändert bleiben.¹⁴

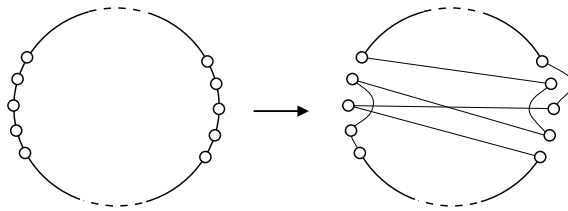


Abbildung 2.4: Beispiel für einen HyperOpt-Schritt mit $l = 4$ beim TSP

2.2.2 Kantenorientierte Austausch-Nachbarschaften

Die wohl wichtigste Nachbarschaft für Routing Probleme ist die Kantenaustausch-Nachbarschaft bzw. k -opt-Nachbarschaft, die 1965 von Lin [Lin65] für das Traveling Salesman Problem formuliert wurde. Diese Nachbarschaft beinhaltet alle Lösungen, die aus der aktuellen Lösung y^{akt} dadurch hervorgehen, daß k Kanten eliminiert und durch k neue Kanten ersetzt werden.¹⁵

¹⁴Die Zuordnung des HyperOpt zu den knotenorientierten Nachbarschaften ist sicher nicht zwingend. Beim HyperOpt werden bis zu $2 * l$ Kanten bzw. $2 * (l - 1)$ Knoten ausgetauscht. Die Autoren selbst betrachten diese Nachbarschaft offenbar eher vom Standpunkt einer kantenorientierten Nachbarschaft, was sich u.a. in der Definition des Parameters l ausdrückt.

¹⁵In der Originalarbeit [Lin65] wird die Bezeichnung λ für die Anzahl der ausgetauschten Kanten gewählt. Einige Autoren beziehen sich auf diese Arbeit und sprechen deshalb auch von λ -opt-Nachbarschaften. Zur besseren Unterscheidung von der λ -Interchange-Nachbarschaft verwenden wir die auch sonst übliche Bezeichnung k -opt-Nachbarschaft.

k -opt-Nachbarschaft

Die k -opt-Nachbarschaft wird zunächst anschaulich am Beispiel des TSP erklärt. Als aktuelle Lösung y^{akt} betrachten wir deshalb eine Tour t , die ausgehend vom Standort des Handelsreisenden (Depot) alle n Kundenknoten des Graphen besucht und schließlich zum Depot zurückkehrt. Die k -opt-Nachbarschaft von t beinhaltet genau die Touren, die aus t dadurch entstehen, daß k Bögen entfernt und durch k (nicht notwendig andere) Bögen so ersetzt werden, daß wieder genau eine Tour entsteht. Dabei kann es vorkommen, daß die Orientierung der Bögen in einem Toursegment der Ausgangslösung umgekehrt werden muß. Als Beispiel hierfür zeigt Abb. 2.5 die Umkehrung der Orientierung einer Teiltour beim 2-opt-Schritt.

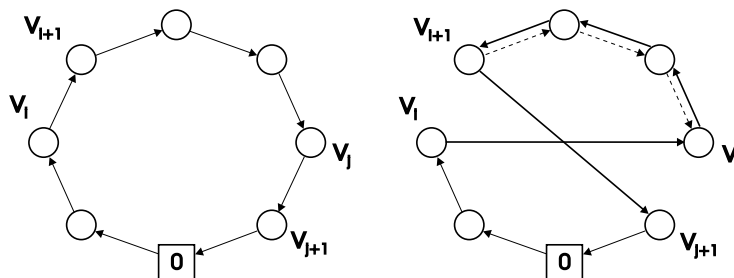


Abbildung 2.5: Umkehrung der Orientierung eines Teilpfades beim 2-opt-Schritt (alte Orientierung gestrichelt)

Da für $k = 1$ nur die Möglichkeit besteht, den einzigen eliminierten Bogen an derselben Stelle wieder einzufügen, betrachten wir ausschließlich Nachbarschaften mit $k \geq 2$.¹⁶

¹⁶ Ein einzelner Bogen könnte nur entweder in gleicher oder in umgekehrter Richtung an seiner eigenen Position wieder eingesetzt werden. Im ersten Fall entsteht eine identische Tour. Im zweiten Fall wird erzwungen, daß sich die Orientierung aller übrigen Bögen ebenfalls umkehrt. Beide Nachbarlösungen sind z.B. auch in der echten 2-opt-Nachbarschaft enthalten. Man erhält sie, wenn zwei benachbarte Bögen der Tour für den 2-opt-Schritt gewählt werden: Seien (v_i, v_{i+1}) und (v_j, v_{j+1}) die gelöschten Bögen und (v_i, v_j) und (v_{i+1}, v_{j+1}) die eingefügten Bögen (Abb. 2.5). Dann erhält man die Identität für $v_{i+1} = v_j$ und die inverse Tour für $v_{j+1} = v_i$.

Größe der k -opt-Nachbarschaft

Es gibt $\binom{n+1}{k}$ verschiedene Möglichkeiten, die k zu eliminierenden Bögen auszuwählen. Durch das Entfernen der k Bögen entstehen k Toursegmente bzw. isolierte Knoten. Diese Segmente bzw. Knoten müssen durch die hinzugefügten neuen Bögen wieder zu einer neuen Tour zusammengesetzt werden, wobei die Reihenfolge der Segmente beliebig permutiert werden kann. Es gibt demzufolge $(k-1)!$ verschiedene Reihenfolgen (ein Segment wird festgehalten). Gleichzeitig können die Segmente jeweils unabhängig voneinander in ihrer Orientierung umgekehrt werden oder nicht ($2^{(k-1)}$ Möglichkeiten). Insgesamt kann die Größe der Nachbarschaft $\mathcal{N}(t)$ wie folgt nach oben abgeschätzt werden:

$$|\mathcal{N}(t)| \leq \binom{n+1}{k} (k-1)! 2^{(k-1)} \quad (2.3)$$

Der Ausdruck auf der rechten Seite liefert nur eine obere Schranke der Anzahl unterschiedlicher Nachbarlösungen. Dafür gibt es mehrere Gründe:

1. Bei der Zerlegung der Tour in Toursegmente können auch isolierte Knoten (d.h. Pfade mit nur einem Knoten) entstehen. Für diese macht eine Umkehrung der Orientierung keinen Sinn.
2. Die k -opt-Nachbarschaft beinhaltet für $k > 2$ auch die l -opt-Nachbarschaften für $l \in \{2, \dots, k-1\}$. Schritte dieser Nachbarschaften erhält man z.B. immer dann, wenn ein oder mehrere Bögen durch sich selbst ersetzt werden. Da es dafür i.d.R. jedoch eine Vielzahl von Möglichkeiten gibt, werden diese identischen Nachbarschaftsschritte auf der rechten Seite der Ungleichung (2.3) mehrfach gezählt.

Beispiel 2.3. Wir betrachten eine Tour $t = (0, 1, 2, 3, 4, 5, 6, 7, 8)$ und ihre Nachbartour $t' = (0, 1, 2, 6, 5, 4, 3, 7, 8)$, die in Abb. 2.6 dargestellt sind. Die Nachbartour t' erhält man durch einen 2-opt-Schritt

- 2-opt-Schritt:
 - lösche die Bögen (2,3) und (6,7)
 - füge die Bögen (2,6) und (3,7) ein

oder durch einen 3-opt-Schritt der einen Bogen identisch ersetzt, also z.B. so:

- 3-opt-Schritt:
 - lösche die Bögen (1,2),(2,3) und (6,7)
 - füge die Bögen (1,2),(2,6) und (3,7) ein

3. Einige identische Touren der Nachbarschaft können auf unterschiedliche Weise entstehen. In (2.3) werden diese Touren dann mehrfach gezählt. Dies läßt sich am besten anhand eines Beispiels erklären.

Beispiel 2.4. (Fortsetzung von Beispiel 2.3)

Die Nachbartour t' kann durch eine Reihe weiterer k -opt-Schritte erzeugt werden, z.B. durch folgende 3-opt-Schritte:

- 3-opt-Schritt:
 - lösche die Bögen $(2,3)$, $(3,4)$ und $(6,7)$
 - füge die Bögen $(2,6)$, $(4,3)$ und $(3,7)$ ein
- 3-opt-Schritt:
 - lösche die Bögen $(2,3)$, $(4,5)$ und $(6,7)$
 - füge die Bögen $(2,6)$, $(5,4)$ und $(3,7)$ ein

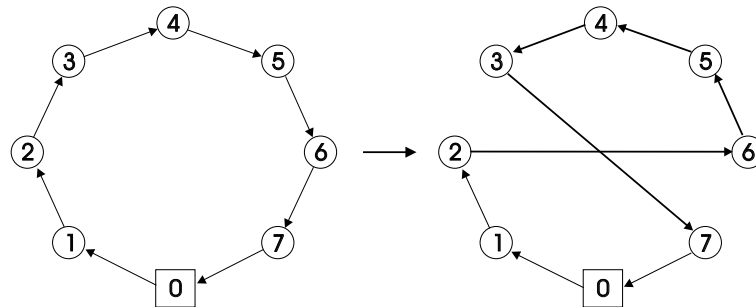


Abbildung 2.6: Die Tour t (links) kann durch verschiedene k -opt-Schritte in die Nachbartour t' (rechts) überführt werden.

Lexikographische Suche

Die lexikographische Suche ist ein Algorithmus, der eine systematische Suche innerhalb einer k -opt-Nachbarschaft durchführt und dafür sorgt, daß *alle* Kantenaustausche untersucht werden und dabei *kein* Kantenaustausch mehrfach betrachtet wird.

Man stelle sich die Bögen der aktuellen Lösung y^{akt} aufsteigend nummeriert vor. Bei der lexikographischen Suche werden schrittweise unterschiedliche Lösungen der Nachbarschaft von y^{akt} untersucht. Die zu eliminierenden Bögen werden in jedem neuen Schritt so gewählt, daß die Nummern der

Bögen eine lexikographisch geordnete Folge bilden. In jedem Schritt werden alle Nachbarlösungen untersucht, die dadurch entstehen können, daß durch das Wiedereinsetzen von k Bögen eine neue Rundtour gebildet wird. Davon gibt es eine feste Zahl, die ausschließlich von k , aber nicht von den aktuell vorliegenden Segmenten abhängig ist.¹⁷

Beispiel 2.5. Wir betrachten eine 3-opt-Nachbarschaft. Die Lösung y^{akt} sei eine Rundtour mit n Knoten und n Bögen. Die Bögen von y^{akt} seien aufsteigend nummeriert. Die in den aufeinanderfolgenden Schritten der Lexikographische Suche zu eliminierenden Bögen bilden eine lexikographisch geordnete Folge:

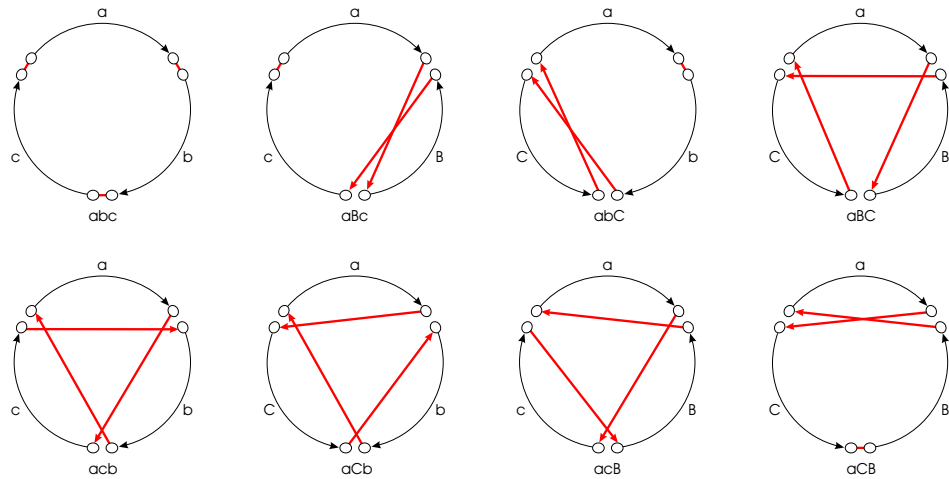
$$\begin{array}{ccccccc}
 (1, 2, 3), & (1, 2, 4), & (1, 2, 5), & \dots & , & (1, 2, n) \\
 & (1, 3, 4), & (1, 3, 5), & \dots & , & (1, 3, n) \\
 & & (1, 4, 5), & \dots & , & (1, 4, n) \\
 & & & \ddots & & \vdots \\
 & & & & & \vdots \\
 (2, 3, 4), & (2, 3, 5), & \dots & , & (2, 3, n) \\
 & (2, 4, 5), & \dots & , & (2, 4, n) \\
 & & \ddots & & \vdots \\
 & & & & \vdots \\
 & & & & & \vdots \\
 & & & & & (n-2, n-1, n)
 \end{array}$$

Nach dem Eliminieren der drei Bögen bleiben jeweils drei Ketten bzw. Segmente übrig. Um eine neue Roundtour zu bilden, müssen diese wieder miteinander verbunden werden. Unabhängig davon, wie lang die einzelnen Segmente sind, gibt es dafür bei der 3-opt-Nachbarschaft prinzipiell acht verschiedene Möglichkeiten (siehe Abb. 2.7). Eine davon ist die Identität, drei weitere Nachbarlösungen, setzen einen eliminierten Bogen unverändert wieder ein, so daß es sich eigentlich um 2-opt-Schritte handelt. Die übrigen vier Ersetzungen stellen echte 3-opt-Schritte dar.

Die lexikographische Suche ist vor allem dann interessant, wenn zusätzliche Restriktionen wie z.B. Zeitfenster zu berücksichtigen sind. Normalerweise wächst der Aufwand für die Überprüfung der Zeitfenster linear mit der Anzahl der Knoten. Savelsbergh hat aber in [Sav90] gezeigt, daß es bei einer lexikographischen Suchreihenfolge möglich ist, die notwendigen Überprüfungen mit Aufwand $\mathcal{O}(1)$ durchzuführen. Darauf wird im Abschnitt 3.1 noch ausführlich eingegangen.

¹⁷Damit tatsächlich *keine* Schritte mehrfach betrachtet werden, müssen ggf. manche Schritte ausgeschlossen werden, falls ein oder mehrere Segmente nur aus einem einzelnen Knoten bestehen.

¹⁸Die Notation $[abc]$, $[acb]$, ... wird später verwendet werden. Sie wird auf Seite 84 erklärt.


 Abbildung 2.7: Verschiedene 3-opt-Schritte¹⁸

Sequentielle Suche

Bei der Sequentiellen Suche werden die Nachbarschaftsschritte durch abwechselndes Löschen und Einfügen zueinander adjazenter Kanten aufgebaut. Die gelöschten und hinzugefügten Kanten bilden daher immer einen alternierenden Pfad.

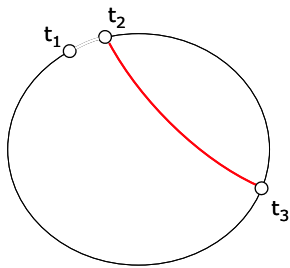


Abbildung 2.8: „stem-and-cycle“-Struktur bei der sequentiellen Suche

Sei (t_1, t_2) die erste gelöschte Kante, so muß die erste eingefügte Kante von einem der inzidenten Knoten (z.B. vom Knoten t_2) ausgehen. Man fügt also eine Kante (t_2, t_3) ein, löscht anschließend die Kante (t_3, t_4) , usw. Beim Einfügen einer Kante entsteht eine sogenannte „stem-and-cycle“-Struktur (Abb. 2.8). Die anschließend zu löschende Kante muß immer so gewählt werden, daß der Kreis aufgelöst wird und somit wieder ein Pfad hergestellt

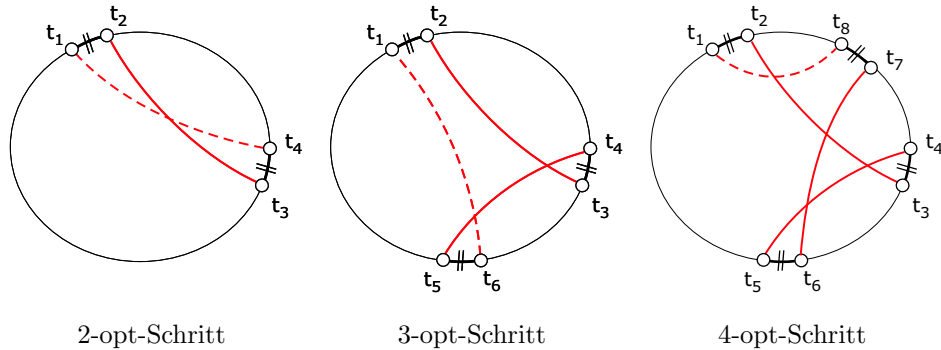


Abbildung 2.9: Sequentielle Suche beim TSP:

Die gelöschten und hinzugefügten Kanten bilden einen alternierenden Pfad. Durch die gestrichelte Kante würde der alternierende Pfad geschlossen werden und die „stem-and-cycle“-Struktur zu einem Kreis entarten, der eine neue Tour darstellt.

wird. Wird der alternierende Pfad mit der k -ten eingefügten Kante geschlossen, d.h. die k -te eingefügte Kante ist (t_{2k}, t_1) , dann entsteht eine Lösung der k -opt-Nachbarschaft (Abb. 2.9).

Für $k \geq 4$ können mit der Sequentiellen Suche nicht alle k -opt-Schritte durch Austausch von k Kanten erzeugt werden. Ein bekanntes Beispiel dafür ist der sogenannte „Double-Bridge-Move“ (siehe Abb. 2.10), ein 4-opt-Schritt, bei dem die gelöschten und eingefügten Kanten keinen alternierenden Pfad bilden. Dieser Schritt kann nur durch den sequentiellen Austausch von fünf Kanten erhalten werden, wobei eine eingefügte Kante wieder gelöscht wird.

Glover zeigt in [Glo92a, S. 24], daß mit alternierenden Pfaden für $k \geq 4$ nur dann alle Nachbarlösungen einer k -opt-Nachbarschaft gefunden werden können, wenn es erlaubt ist, eine eingefügte Kante wieder zu löschen. Das Einfügen bereits gelöschter Kanten kann dagegen generell verboten sein.

Das Verfahren von Lin & Kernighan: Das Verfahren von Lin & Kernighan (LK) wurde für das symmetrische TSP konzipiert und 1973 in [LK73] veröffentlicht. Der Algorithmus basiert auf der Sequentiellen Suche und einem Gewinnkriterium, das die Suche steuert. Eine Übertragung auf das m-TSPR ist nicht ohne weiteres möglich. Neuere Versionen wie der Iterated LK stellen auch heute noch die besten bekannten Heuristiken für das TSP dar.

Zum besseren Verständnis des Vorgehens ist es sinnvoll, die wesentlichen

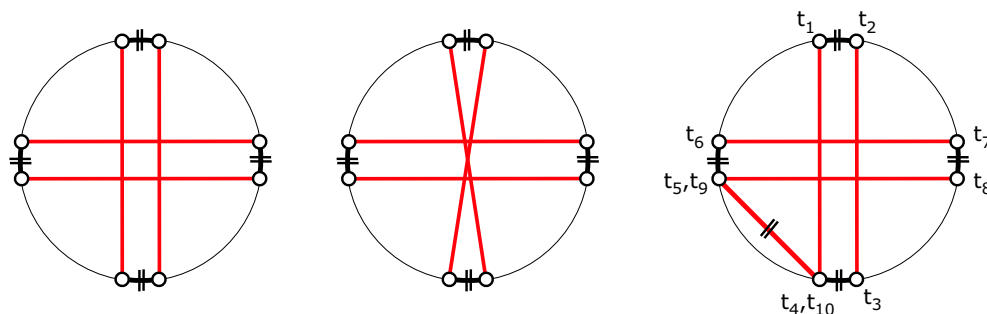


Abbildung 2.10: Der „Double-Bridge-Move“ (links) und der 4-opt-Schritt in der Mitte können sequentiell nicht mit vier Kantentauschen realisiert werden. Die Abb. rechts zeigt, wie man den Double-Bridge-Move mit fünf Kantentauschen erzeugen kann, wobei die Kante (t_4, t_5) wieder gelöscht wird.

Ideen zu kennen, von denen sich Lin und Kernighan der Entwicklung des Algorithmus leiten ließen:

Idee des Verfahrens Wir können beim TSP davon ausgehen, daß alle Lösungen dieselbe Tourlänge (Anzahl von Kanten) besitzen. Die aktuelle Lösung y^{akt} unterscheide sich von der optimalen Lösung y^* in k Kanten. Dann kann man durch Austausch dieser k Kanten, d.h. durch einen k -opt-Schritt, direkt zu der Lösung y^* übergehen. Dabei gibt es jedoch zwei Probleme:

- Die Größe der k -opt-Nachbarschaft wächst mit $\mathcal{O}(n^k)$ und ist daher ggf. viel zu groß, um sie komplett abzusuchen.
- Die Zahl k ist unbekannt.

Um der optimalen Lösung dennoch möglichst schnell nahezukommen, bestimmt man dynamisch ein k . Dazu wird ein Nachbarschaftsschritt sequentiell aufgebaut. Dabei wird eine Folge von Schritten erzeugt, die sich um je eine weitere ausgetauschte Kante voneinander unterscheiden. Das Generieren weiterer Schritte wird abgebrochen, wenn der zusammengesetzte Nachbarschaftsschritt ein bestimmtes Qualitätskriterium nicht mehr erfüllen kann. Schließlich führt man den besten zwischenzeitlich gebildeten Schritt durch.

Umsetzung beim TSP Beim LK werden die Nachbarschaftsschritte entsprechend der sequentiellen Suche iterativ aufgebaut. Zuerst wird eine

Kante (t_1, t_2) gelöscht, so daß ein Pfad entsteht. Danach wird in jeder Iteration i eine Swap-Operation durchgeführt, d.h. eine Kante (t_{2i}, t_{2i+1}) wird eingefügt und eine Kante (t_{2i+1}, t_{2i+2}) wird gelöscht. Der Knoten t_{2i+2} ist immer so zu wählen, daß nach jeder weiteren Iteration wieder ein Pfad entsteht (Referenzstruktur). Dieser Pfad könnte durch Einfügen der fehlenden Kante (t_{2i+2}, t_1) eindeutig in eine Rundtour überführt werden („closing-up“), was einem k -opt-Schritt mit $k = i + 1$ entspräche.

Da erst durch das closing-up ein vollständiger Nachbarschaftsschritt entsteht, werden dessen Kosten für die Bewertung der Referenzstruktur mit herangezogen.

Folgende zusätzliche Festlegungen werden von Lin & Kernighan getroffen:

- Es wird immer eine vollständige 2-opt-Suche oder 3-opt-Suche durchgeführt.
- Die Auswahl der Swap-Operation wird durch eine Greedy-Strategie gesteuert, d.h. es wird immer derjenige Swap berücksichtigt, der den kumulierten Gewinn

$$G(i) = c(t_1, t_2) + \sum_{l=1}^i \left[c(t_{2l+1}, t_{2l+2}) - c(t_{2l}, t_{2l+1}) \right]$$

maximiert. Dies ist gleichbedeutend damit, daß der aktuelle Zuwachs des Gewinns

$$\Delta G = G(i) - G(i-1) = c(t_{2i+1}, t_{2i+2}) - c(t_{2i}, t_{2i+1})$$

maximiert wird.

- Es ist nicht erlaubt, eine gelöschte Kante wieder einzufügen oder eine eingefügte Kante wieder zu löschen. Dadurch wird sichergestellt, daß die Länge der Sequenz entsprechend der Anzahl der Kanten der Tour beschränkt wird.
- Die Suche wird abgebrochen, wenn der kumulierte Gewinn negativ wird bzw. wenn der aktuelle kumulierte Gewinn kleiner ist als der höchste Nettogewinn G^{net} , der innerhalb der aktuellen Sequenz bestimmt wurde. Dabei berücksichtigt der Nettogewinn die „closing-up-costs“, d.h. die Kosten, die entstehen würden, wenn der Pfad nach der i -ten Iteration wieder zu einer Rundtour geschlossen wird:

$$G^{net}(i) = G(i) - c(t_{2i}, t_1)$$

Für eine ausführliche Beschreibung des LK sei auf [LK73] verwiesen. Aktuelle Verfahren beruhen auf dem Iterated LK (siehe [MOF92], [JM97], [HRM97] oder [ABCC98]). Eine spezielle Erweiterung für Probleme, bei denen die Kunden in Cluster aufgeteilt sind, wird in [Net99] vorgeschlagen.

***Or-opt*-Nachbarschaft**

Wie bei den knotenorientierten Nachbarschaften gibt es auch bei den kantenorientierten Nachbarschaften eine Vielzahl speziellerer Nachbarschaften, die durch gezielte Einschränkungen entstehen. Eine der wichtigsten und meistzitierten dieser Nachbarschaften ist die *Or-opt*-Nachbarschaft, die erstmalig in [Or76] beschrieben wurde. Die *Or-opt*-Nachbarschaft beinhaltet alle Lösungen, die durch das Verschieben einer Kette von 1, 2 oder 3 Knoten erhalten werden können. Sie ist daher eine Teilmenge der 3-opt-Nachbarschaft, die u.a. Verschiebungen beliebig langer Segmente erlaubt. Der Aufwand zum Absuchen der *Or-opt*-Nachbarschaft beträgt jedoch nur $\mathcal{O}(n^2)$.

Die *Or-opt*-Nachbarschaft wurde von vielen Autoren mit Erfolg eingesetzt. Der Vorteil besteht gegenüber der 3-opt-Nachbarschaft darin, daß sie mit wesentlich geringerem Aufwand abgesucht werden kann, aber häufig dennoch einen großen Teil der möglichen verbessernden 3-opt-Schritte beinhaltet. So muß der große Rechenzeitgewinn meistens nur mit einem geringen Qualitätsverlust bezahlt werden. Dies gilt insbesondere für Probleme mit engen Zeitfenstern oder Reihenfolgebeschränkungen, bei denen viele 3-opt-Schritte, die eine Invertierung eines Segments erfordern, ohnehin unzulässig sind. Auch die zusätzliche Beschränkung der Länge der verschobenen Knotenkette auf maximal drei Knoten (die die wesentliche Reduktion des Rechenaufwands bewirkt) führt i.d.R. zu keinen oder nur geringen Qualitätseinbußen. Dies kann man sich auch dadurch erklären, daß eine Verschiebung längerer Knotenkette oft auch durch mehrere aufeinanderfolgende *Or-opt*-Schritte erreicht werden kann. Voraussetzung dafür ist allerdings, daß eine Folge von *Or-opt*-Schritten gefunden wird, bei der jeder einzelne Schritt zulässig und verbessernd ist.

***k-opt**-Nachbarschaft**

Die *k-opt**-Nachbarschaft ist eine Erweiterung der *k-opt*-Nachbarschaft, die immer dann betrachtet werden kann, wenn die Lösung eines Problems aus mehreren Touren besteht (wie z.B. beim VRP). Die Lösungen können dann

in Form einer großen Rundtour (Giant-Route) repräsentiert werden, die das Depot mehrfach enthält (siehe auch Abschnitt 3.3.1).

Bei der k -opt-Nachbarschaft wurde stets verlangt, daß nach Ausführung eines Schritts *genau eine* Rundtour entstehen muß. Beim Betrachten einer Giant-Route, führen aber ggf. auch solche Schritte zu zulässigen Lösungen, die die Lösung in zwei oder mehr (Kurz-)Zyklen aufteilen. Entscheidend ist dabei, daß in jedem einzelnen Zyklus mindestens ein Depot enthalten sein muß. In der k -opt*-Nachbarschaft werden diese zusätzlich möglichen Schritte berücksichtigt (siehe Abb. 2.11).

Potvin, Lapalme und Rousseau beschreiben in [PLR89] die Anwendung der k -opt*-Nachbarschaft auf das m-TSP mit Zeitfenstern (m-TSPTW). Sie stellen fest, daß insbesondere die 2-opt*-Nachbarschaft für Probleme mit Zeitfenstern sehr gut geeignet ist und eine Kombination der 2-opt*-Nachbarschaft mit der 2-opt-Nachbarschaft oft bessere Ergebnisse liefert als die Lokale Suche mit der (i.a. wesentlich größeren) 3-opt-Nachbarschaft.

Die 2-opt*-Nachbarschaft wurde von mehreren Autoren insbesondere für Probleme mit Zeitfenstern eingesetzt. Trotz der guten Erfolge mit der 2-opt*-Nachbarschaft wurde der Einsatz von k -opt*-Nachbarschaften mit $k > 2$ bislang scheinbar nicht systematisch untersucht. Im Abschnitt 3.3.4 wird beschrieben, wie eine Lokale Suche in der k -opt*-Nachbarschaft für m-TSPR durchgeführt werden kann.

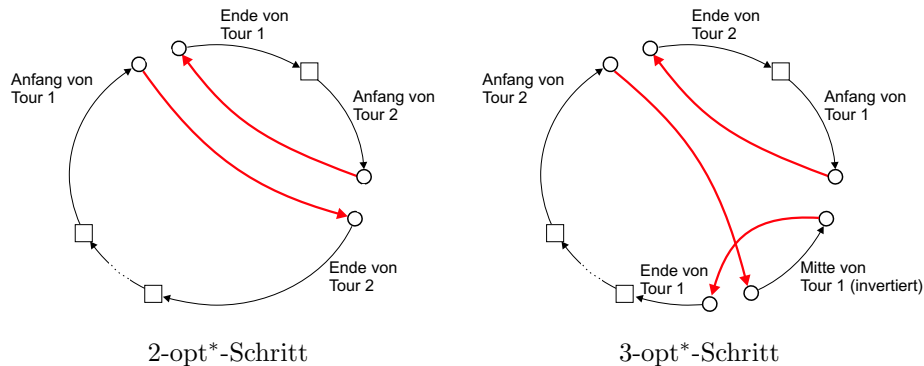


Abbildung 2.11: Beispiele für k -opt*-Schritte:

Beim 2-opt*-Schritt wird der Anfang von Tour 1 mit dem Ende von Tour 2 verbunden und umgekehrt. Dadurch besteht eine gute Chance, daß auch bei Existenz von Zeitfenstern eine zulässige Lösung gebildet wird.

2.2.3 GENIUS

1992 schlugen Gendreau, Hertz und Laporte in [GHL92] ein spezielles Konstruktions- und Verbesserungsverfahren für das TSP vor. GENIUS basiert auf einer verallgemeinerten Einfügun gsvorschrift (engl. Generalized Insertion – GENI). Diese Vorschrift wird einerseits zur Konstruktion einer Startlösung und andererseits auch für ein spezielles Verbesserungsverfahren (Unstringing and Stringing – US) benutzt. US entfernt einen Knoten aus der aktuellen Lösung, indem ein „inverser GENI-Schritt“ durchgeführt wird. Anschließend wird derselbe Knoten durch einen weiteren GENI-Schritt erneut in die Lösung eingebaut. Durch die Kombination von GENI und US erhält man das Verfahren GENIUS.

Generalized Insertion – GENI Die Einfügung eines Knotens in eine Tour ist bei GENI mit einem gleichzeitig durchgeführten k -opt-Schritt verbunden. Der erste gelöschte Bogen ist stets der Bogen, der vom eingefügten Knoten ausgeht. Dadurch wird erreicht, daß der eingefügte Knoten nicht unbedingt zwischen zwei zuvor benachbarten Knoten plazi ert wird. Die einfache Einfügung eines Knotens (ohne weitere Modifikation der Tour) wird als Sonderfall erhalten.

Die Autoren unterscheiden zwei Varianten des verallgemeinerten Einfügungsschritts, die sie als Type-I-Insertion bzw. Type-II-Insertion bezeichnen.

Type-I-Insertion Es wird ein 3-opt-Schritt durchgeführt, der zwei Teilstücke (Segmente) der Originaltour invertiert, aber die Reihenfolge der Segmente untereinander nicht ändert. Die Einfügung des Knotens erfolgt am ersten eingefügten Bogen:

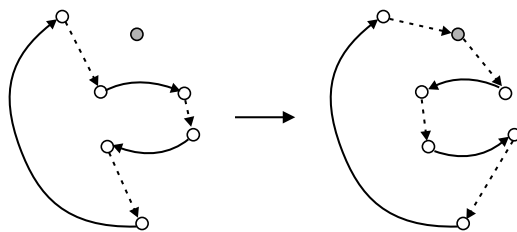


Abbildung 2.12:
GENI – Type-I-Insertion

Type-II-Insertion Es wird ein 4-opt-Schritt durchgeführt, der ebenfalls zwei Teilstücke der Originaltour invertiert. Das genaue Vorgehen ist Abb. 2.13 zu entnehmen.

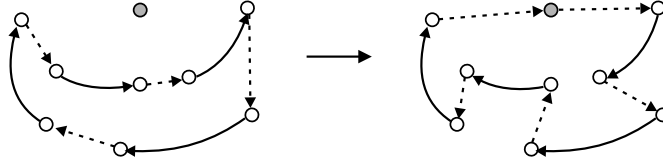


Abbildung 2.13: GENI – Type-II-Insertion

GENI berücksichtigt immer beide möglichen Orientierungen der Tour. Um die Größe der Nachbarschaft zu reduzieren, werden nicht alle möglichen Einfügungen betrachtet. Die Autoren definieren eine p -Nachbarschaft $N_p(v)$ für alle Knoten v der Tour. $N_p(v)$ beinhaltet p der zu v nächstgelegenen Knoten der Tour.

Die Einfügeheuristik GENI stellt nicht direkt ein Verbesserungsverfahren dar. Dennoch kann GENI als Nacheinanderausführung einer Knoten-Einfügung und einem 3-opt-Schritt bzw. 4-opt-Schritt aufgefaßt werden.¹⁹ Durch die Einfügung des Knotens bedingt, wird insgesamt ein Bogen mehr eingefügt als gelöscht.

Die Schritte bei der US-Prozedur sind Verbesserungsschritte. Die Nacheinanderausführung von „unstringing“- und „stringing“-Schritt stellt insgesamt einen speziellen k -opt-Schritt dar. Dies ist entweder ein 7-opt-Schritt oder ein 9-opt-Schritt:

- Type-I-Insertion: Beim unstringing werden vier Bögen gelöscht und drei Bögen eingefügt (inverser GENI-Schritt) und beim stringing werden drei Bögen gelöscht und vier Bögen eingefügt.
- Type-II-Insertion: Zunächst werden fünf Bögen gelöscht und vier Bögen eingefügt (unstringing) und dann vier Bögen gelöscht und fünf Bögen eingefügt (stringing).

GENIUS wurde später weiterentwickelt, um das Verfahren auch auf andere Problemklassen anwenden zu können. Die erste Erweiterung ist das von Gendreau et al. in [GHLS95] beschriebene GENIUS-TW, das die Berücksichtigung von Zeitfenstern und somit die Anwendung auf TSPTW ermöglicht. In [PGR97] wird die Verknüpfung von GENIUS mit dem Einsatz von Constraint

¹⁹Die Reihenfolge spielt keine Rolle, d.h. man kann GENI ebenso als Nacheinanderausführung eines 3-opt-Schritts bzw. 4-opt-Schritts und einer Knoten-Einfügung ansehen.

Programming (CP) beschrieben. Bei diesem Ansatz wird die Zulässigkeit eines Nachbarschaftsschritts mit Hilfe des CP überprüft. Sobald eine Entscheidung getroffen wurde, die zur Unzulässigkeit einer (Teil-)Lösung führt, wird ein Backtracking ausgelöst. Bei dem Verfahren GENIUS-CP können prinzipiell beliebige Restriktionen berücksichtigt werden, so daß das Verfahren allgemein auf m-TSPR (ggf. auch darüber hinaus) anwendbar ist. Ein wesentlicher Vorteil des Ansatzes liegt darin, daß die Restriktionen für das CP deklarativ formuliert werden können, weshalb eine Anpassung des Algorithmus beim Hinzufügen weiterer Nebenbedingungen nicht notwendig ist. In [RGP02a] erweitern die gleichen Autoren den zuvor beschriebenen Ansatz. Hier setzen sie GENI für die Einfügung von Knoten im Rahmen einer Large Neighborhood Search ein, wie sie von Shaw [Sha98] vorgeschlagen wurde. Für eine Nachoptimierung der gefundenen Touren am Ende der Suche wird erneut GENIUS-CP eingesetzt.

In [PG99] wird ein alternativer Ansatz zur Verbindung zwischen lokaler Suche und CP untersucht. Dabei wird das CP-Modell benutzt, um die zulässigen Nachbarschaftsschritte zu bestimmen. Die Autoren wenden den neuen Ansatz auch auf GENIUS an.

GENIUS wurde auch bei der Lösung von VRPTW mit heterogenem Fuhrpark eingesetzt. In [OVD98] benutzen die Autoren eine Kombination von einem Genetischen Algorithmus und Scatter Search. Ein Gen repräsentiert ein Cluster von Kunden, die gemeinsam bedient werden müssen. Die Bewertung eines Gens wird ermittelt, indem mit Hilfe von GENIUS das zugehörige TSP gelöst wird. In [GLMT99] findet GENIUS als Teil einer Adaptive Memory Procedure²⁰ Verwendung. Auch hier wird GENIUS in erster Linie dazu benutzt, einzelne Touren zu verbessern.

2.3 Meta-Heuristiken

Trotz der großen Bandbreite verschiedener Nachbarschaften besitzt die Lokale Suche eine generelle Schwäche. Die Suche nach verbessernden Lösungen bleibt sehr oft auf einen relativ kleinen Teilbereich des gesamten Lösungsraums beschränkt und bricht ggf. frühzeitig mit einer lokal optimalen Lösung ab. So können viele interessante Lösungen nicht gefunden werden. Die Qualität der gefundenen Lösungen ist daher häufig sehr stark von der Wahl der

²⁰Die Adaptive Memory Procedure (AMP) ist eine spezielle Meta-Strategie, die von Rochat und Taillard für das VRPTW entwickelt und in [RT95] beschrieben wurde.

Startlösung abhängig und ggf. sehr unbefriedigend.

Um bessere Lösungen finden zu können, wurde in den letzten zwei Jahrzehnten eine große Zahl von Meta-Heuristiken entwickelt. Mit deren Hilfe soll in erster Linie erreicht werden, daß

1. der gesamte Lösungsraum bzw. möglichst alle interessanten Teile des Lösungsraums abgesucht werden (Diversifikation) und
2. die Suche in interessant erscheinenden Bereichen besonders gründlich durchgeführt wird (Intensivierung).

Der Begriff der Meta-Heuristik wird in der Literatur unterschiedlich gefaßt. Wir wollen unter einer Meta-Heuristik ein allgemeingültiges heuristisches Verfahren verstehen, das eine andere (problemspezifische) Heuristik steuert.²¹

Eine ausführliche Darstellung der in den letzten Jahren entwickelten Meta-Heuristiken liegt jenseits der Intention dieser Arbeit. Einen guten Überblick kann man sich z.B. anhand von [Ree93], [OL96], [TGGP01], [VMOR99], [YI] und [RH01] verschaffen.

Da die Entwicklung der verschiedenen Meta-Heuristiken auf unterschiedlichen Sichtweisen, Vorstellungen und Analogien basiert, werden oft verschiedene Begriffe für sehr ähnliche oder gleiche Konzepte benutzt. Bei genauer Untersuchung unterscheiden sich die Meta-Heuristiken deshalb oft weniger voneinander, als dies zunächst den Anschein hat. Eine gewisse Systematisierung scheint deshalb sinnvoll.

Die wesentlichen Ideen der verschiedenen Meta-Heuristiken beinhalten

- die Benutzung mehrerer unterschiedlicher Nachbarschaften,
- den Wechsel zu Ersatz-Zielfunktionen,
- die Verwendung mehrerer Startlösungen,
- den Einsatz von Zufallselementen,
- die Kombination mehrerer (Teil-)Lösungen,
- Gedächtnis und Lernen.

Eine eindeutige Zuordnung der einzelnen Meta-Heuristiken zu bestimmten Kategorien muß daran scheitern, daß bei allen erfolgreichen Meta-Heuristiken mehrere Ideen miteinander kombiniert und auf verschiedene Aspekte der Suche angewendet werden. Eine große Zahl von Veröffentlichungen

²¹siehe auch [GL97, S. 17]

beschäftigt sich mit Erweiterungen von Meta-Heuristiken, um diese für bestimmte Anwendungen noch besser geeignet zu machen bzw. bestimmte Schwächen der Verfahren abzubauen. Dabei werden oft Ideen benutzt, die von anderen Meta-Heuristiken bereits bekannt sind. Dementsprechend besteht auch durchaus nicht immer Einigkeit darüber, wodurch eine bestimmte Meta-Heuristik wesentlich gekennzeichnet ist. Glover und Laguna schreiben dazu in [GL97, S. 17]:

„[However,] a rigorous classification of different meta-heuristics is a difficult and risky enterprise, because the leading advocates of alternative methods often differ among themselves about the essential nature of the methods they espouse.“²²

Einige Veröffentlichungen versuchen eine integrierende Sicht auf mehrere Meta-Heuristiken zu entwickeln. Dabei wird meistens gezeigt, daß sich eine Reihe anderer Meta-Heuristiken auch als Spezialfälle einer bestimmten Meta-Heuristik darstellen lassen.²³

Ein anderer Ansatz wird von Yagiura und Ibaraki [YI] verfolgt. Die Autoren versuchen, das Vorgehen von Meta-Heuristiken allgemein wie folgt zu fassen:

Algorithmus 2.2. Meta-Heuristik nach Yagiura und Ibaraki

1. Erzeuge eine [Menge von] Startlösung[en] X .
2. Verbessere X mit einer (verallgemeinerten) Lokalen Suche:
 - 2.1 Setze $x^{akt} := x \in X$
 - 2.2 (Schleife) Solange die *Abbruchbedingung* nicht erfüllt ist, wiederhole:
 - 2.2.1 Bestimme mit einer spezifizierten *Auswahl-Strategie* eine Lösung x' aus der *Nachbarschaft* $\mathcal{N}(x^{akt})$.
 - 2.2.2 Falls für die *Bewertungsfunktion* $\tilde{f}$ gilt: $\tilde{f}(x^{akt}) > \tilde{f}(x')$, setze $x^{akt} := x'$.

Durch die Angabe der Nachbarschaft, der Auswahl-Strategie, der Bewertungsfunktion und einer Abbruchbedingung sowie durch die Spezifikation von Schritt 1 können verschiedene Meta-Heuristiken beschrieben werden.

²²[Wie auch immer,] eine genaue Klassifikation verschiedener Meta-Heuristiken ist ein schwieriges und riskantes Unterfangen, weil die führenden Verfechter alternativer Methoden oft untereinander verschiedene Ansichten über den wesentlichen Charakter der von ihnen vertretenen Methoden haben.

²³z.B. [TGGP01], [CH01b], [LMS]

Einen ähnlichen Ansatz verfolgen Vaessens et. al. [VAL98]. Sie präsentieren eine stark verallgemeinerte Ablaufstruktur für eine Lokale Suche (Local Search Template) und zeigen, daß dieses Template zu verschiedenen Meta-Heuristiken spezialisiert werden kann (z.B. Variable Depth Search, Tabu Search, Genetische Algorithmen).

In diesem Abschnitt kann nur eine knappe Übersicht über die wichtigsten Ideen der Meta-Heuristiken gegeben werden. Wir werden im Abschnitt 5.2 darauf eingehen, daß sich viele dieser Ideen mit den in dieser Arbeit vorgestellten Methoden gut verbinden lassen.

2.3.1 Variation der Nachbarschaft

Variable-Neighborhood-Descent-Heuristik (VND) Eine einfache Idee, einen frühzeitigen Abbruch der Lokalen Suche zu vermeiden, ist die wechselnde Verwendung von zwei oder mehr Nachbarschaften. Verfahren, die solche Wechsel zwischen verschiedenen Nachbarschaften systematisch durchführen, bezeichnen Hansen und Mladenović in [HM01b] als „Variable Neighborhood Descent“.

Angenommen, es stehen die Nachbarschaften $(\mathcal{N}_1, \dots, \mathcal{N}_k)$ zur Verfügung. Man beginnt mit der Lokalen Suche unter Verwendung von $\mathcal{N}_1$, was eine bezüglich der Nachbarschaft $\mathcal{N}_1$ lokal optimale Lösung liefert. Diese ist aber i.a. nicht lokal optimal bezüglich der Nachbarschaft $\mathcal{N}_2$, weshalb die Lokale Suche mit $\mathcal{N}_2$ gestartet werden kann. Anschließend wechselt man die Nachbarschaften immer weiter, bis eine Lösung vorliegt, die bezüglich aller Nachbarschaften $\mathcal{N}_1, \dots, \mathcal{N}_k$ lokal optimal ist.

In [PR95] wenden Potvin und Rousseau das beschriebene Vorgehen auf die 2-opt-Nachbarschaft und 2-opt*-Nachbarschaft²⁴ für das m-TSPTW an. Sie stellen fest, daß die Kombination dieser zwei Nachbarschaften (Komplexität ist $\mathcal{O}(n^2)$) oft bessere Lösungen liefert als die Lokale Suche mit der 3-opt-Nachbarschaft (Komplexität $\mathcal{O}(n^3)$).

Das beschriebene Vorgehen ist besonders effektiv, wenn sich die verwendeten Nachbarschaften gut ergänzen. Daher sollte die Schnittmenge der Nachbarschaften paarweise gering sein, was z.B. bei 2-opt-Nachbarschaft und 2-opt*-Nachbarschaft der Fall ist.

²⁴siehe k -opt*-Nachbarschaft auf Seite 48

Variable Neighborhood Search Die von Hansen und Mladenović in [HM97] und [HM01b] formulierte „Variable Neighborhood Search“ (VNS), kann als eine Erweiterung der VND angesehen werden.

Das Vorgehen der VNS kann in seiner Basisform durch folgenden Algorithmus beschrieben werden:

Algorithmus 2.3. Variable Neighborhood Search²⁵

1. (*Initialisierung*) Wähle eine Menge von Nachbarschaften $\mathcal{N}_1, \dots, \mathcal{N}_{k_{max}}$, eine Startlösung y und eine *Abbruchbedingung*.
2. (*Schleife*) Solange die *Abbruchbedingung* nicht erfüllt ist, wiederhole:
 - 2.1 Setze $k := 1$
 - 2.2 Solange $k \leq k_{max}$ ist, wiederhole:
 - 2.2.1 (*Schütteln*) Generiere zufällig eine Lösung $y' \in \mathcal{N}_k(y)$.
 - 2.2.2 (*Lokale Suche*) Wende eine Lokale Suche auf y' an. Dadurch erhält man die lokal optimale Lösung y'' .
 - 2.2.3 (*Durchführung ja/nein*)
Falls y'' besser ist als y , setze $y := y''$ und $k := 1$,
sonst setze $k := k + 1$.

Bei der VNS wird also (im Gegensatz zur VND) nicht direkt in der Nachbarschaft der besten bekannten Lösung y gesucht, sondern ausgehend von einer zufällig gewählten Lösung y' der Nachbarschaft $\mathcal{N}_k(y)$.

Hansen und Mladenović weisen in [HM01a] darauf hin, daß die Lokale Suche in Schritt 2(.2).2 auch einen Wechsel zwischen verschiedenen Nachbarschaften, also eine VND, beinhalten kann.

Variable Depth Search (VDS) Die wesentliche Idee des Verfahrens besteht in einer dynamischen Anpassung der Größe der Nachbarschaft. Bei einer Austausch-Nachbarschaft kann z.B. die Zahl der gleichzeitig durchgeführten Austausche variiert werden.

Glover und Laguna betrachten die VDS als spezielle Form einer Ejection Chain.²⁶ Nachbarschaftsschritte werden dabei als zusammengesetzte Schritte („compound moves“) betrachtet, die aus einer Kette von Teilschritten („simple moves“) aufgebaut werden. Der Aufbau dieser Kette von Teilschritten wird nach bestimmten Kriterien abgebrochen, z.B. wenn die Qualität des

²⁵vgl. [HM01b, S. 451]

²⁶siehe [GL97, S. 135]

so generierten zusammengesetzten Nachbarschaftsschritts ein Qualitätslimit unterschreitet. Anschließend wird die beste Teilkette, ausgehend vom Start bis zu einem frei wählbaren letzten Teilschritt, bestimmt und der zugehörige zusammengesetzte Schritt durchgeführt.

Die Idee der VDS wurde von Lin und Kernighan sehr erfolgreich auf das Graph Partitioning Problem [KL70] und auf das Traveling Salesman Problem [LK73] angewendet. Verbesserte Varianten des Lin-Kernighan-Algorithmus stellen auch heute die leistungsfähigsten Heuristiken für das TSP dar. Das Vorgehen des Lin-Kernighan-Algorithmus wurde bereits auf Seite 45f. beschrieben.

Wechsel der Zielfunktion Die Eigenschaft einer Lösung, lokal optimal zu sein, kann nicht nur durch einen Wechsel der Nachbarschaft aufgehoben werden. Ebenso kann auch ein Wechsel zwischen (evtl. mehreren) Ersatz-Zielfunktionen und der Zielfunktion vorgenommen werden, um einen zeitigen Abbruch der Suche zu vermeiden.

Bei vielen Problemen lassen sich geeignete Ersatz-Zielfunktionen ableiten, die der eigentlichen Zielfunktion insofern nahekommen, daß Nachbarschaftsschritte, die eine Verbesserung der Ersatz-Zielfunktion bewirken, häufig (aber nicht immer) auch eine Verbesserung der Zielfunktion zur Folge haben. Der Wechsel zu einer solchen Ersatz-Zielfunktion führt deshalb i.d.R. zu keiner bzw. nur zu einer moderaten Verschlechterung der Lösung bezüglich der ursprünglichen Zielfunktion.

Der Wechsel zwischen verschiedenen Zielfunktionen wird in [Cal00] beim TSPTW, in [CW98] bzw. [CW01] und [Fun98] beim VRPTW erfolgreich eingesetzt. In allen genannten Fällen wird das Vorgehen mit dem zuvor beschriebenen Wechsel zwischen verschiedenen Nachbarschaften kombiniert.

Modifikation von Zielfunktion oder Daten Eine Reihe von Verfahren basieren auf einer gezielten Veränderung der Zielfunktion oder der dem Problem zugrundeliegenden Daten. Durch diese Modifikation wird es ggf. ermöglicht, zuvor lokal optimale Lösungen wieder zu verlassen.

Noising Methods Bei den von Charon und Hudry beschriebenen „Noising Methods“ (siehe [CH01a] und [CH01b]) werden entweder die Zielfunktion oder die dem Problem zugrundeliegenden Daten durch ein künstlich aufgebrachtes Rauschen r gestört. Das Rauschen wird gemäß einer Zufalls-

verteilung erzeugt und so gesteuert, daß dessen Mittelwert und Standardabweichung während des Optimierungsprozesses gegen Null gehen. Das aufgebrauchte Rauschen dient bei diesen Verfahren dazu, lokal optimale Lösungen wieder verlassen und die Lokale Suche fortsetzen zu können. Durch das sukzessive Absenken des Rauschens nähert sich der Suchprozeß einer Lösung des Originalproblems.

Charon und Hudry unterscheiden drei Formen von Noising Methods, die dadurch gekennzeichnet sind, welche Größen durch das Rauschen gestört werden.

1. Die Störung wird auf die Daten aufgebracht. Beim TSPTW können z.B. die zeitlichen und/oder räumlichen Abstände zwischen den Knoten gestört werden.
2. Die Berechnung der Veränderung der Kosten beim Durchführen eines Nachbarschaftsschritts von y zu y' wird gestört.
Anstelle von $\Delta c(y, y') = c(y) - c(y')$ wird der Lokalen Suche ein gestörter Gewinn $\Delta c^{noised}(y, y') = \Delta c(y, y') + r$ zugrunde gelegt.
3. Die Kostenfunktion c selbst wird gestört, indem ein Teil der Daten vernachlässigt wird.

Simulated Annealing (SA) Das Verfahren zählt zu den ersten Meta-Heuristiken, die auf einer Analogie zu natürlichen Prozessen beruhen. 1953 beschrieben Metropolis et. al. [MRR⁺53] ein Modell zur Simulation des physikalischen Prozesses beim Abkühlen (tempern) von Kristallen auf einem Computer. Erst 30 Jahre später wurde von Kirkpatrick et. al. in [KGV83] vorgeschlagen, die Methodologie zur Lösung kombinatorischer Optimierungsprobleme einzusetzen.

Die wesentliche Idee des Verfahrens besteht in einer Erweiterung der Lokalen Suche, die es erlaubt, daß auch verschlechternde Nachbarschaftsschritte mit einer gewissen Wahrscheinlichkeit zugelassen werden. Dadurch ist es prinzipiell möglich, ein lokales Optimum wieder zu verlassen. Die Zufallsentscheidung über die Annahme einer verschlechternden Lösung wird so durchgeführt, daß die Wahrscheinlichkeit der Akzeptanz von einem „Temperatur“-Parameter T abhängig ist. Mittels einer Abkühlungsstrategie wird T langsam an null angenähert. Dadurch reduziert sich im Lauf der Zeit die Wahrscheinlichkeit, daß verschlechternde Nachbarschaftsschritte akzeptiert werden, ebenfalls auf null.

Algorithmus 2.4. Simulated Annealing

1. (*Initialisierung*) Generiere eine Startlösung y^0 und initialisiere die aktuelle Lösung $y^{akt} := y^0$ und den Temperatur-Parameter $T := T^0 \gg 0$.
2. Wiederhole, bis eine Abbruchbedingung erfüllt ist:
 - 2.1 Bestimme eine zufällig eine Nachbarlösung $y' \in \mathcal{N}(y^{akt})$ mit einer gleichmäßigen Verteilung über $\mathcal{N}(y^{akt})$.
 - 2.2 Setze $p := \begin{cases} 1, & \text{falls } z(y') \leq z(y^{akt}) \\ e^{-\frac{z(y') - z(y^{akt})}{T}}, & \text{sonst.} \end{cases}$
 - 2.3 Setze $y^{akt} := y'$ mit Wahrscheinlichkeit p .
 - 2.4 (*Abkühlung*) Reduziere T gemäß einer Abkühl-Strategie.

SA wurde für die Lösung sehr vieler verschiedener kombinatorischer Optimierungsprobleme erfolgreich eingesetzt.

Charon und Hudry verweisen darauf, daß SA als spezielle Variante der Noising Methods aufgefaßt werden kann, bei dem der Gewinn eines Nachbarschaftsschritts durch eine logarithmische Rauschfunktion gestört wird, wobei die (sinkende) Temperatur die Rausch-Rate steuert.²⁷ Diese Sichtweise motiviert auch die hier vorgenommene Einordnung von SA als Modifikation der Zielfunktion.

Glätten von Daten (Data Smoothing) Eine wiederholte Modifikation der Daten soll auch beim Data Smoothing (DS) das Ausbrechen aus lokalen Optima ermöglichen. Im Gegensatz zu den zuletzt beschriebenen Verfahren benutzt man aber eine systematische und keine zufallsgesteuerte Modifikation.

Mit einer parametrierbaren Glättungsfunktion werden die Daten zuerst einander angeglichen, bevor die Lokale Suche durchgeführt wird. Danach wird die Glättung durch Anpassung des Glättungsparameters schrittweise reduziert und die Lokale Suche jeweils ausgehend von der letzten gefundenen Lösung erneut gestartet. So nähert man sich iterativ den ursprünglichen Daten. In der letzten Iteration wird die Glättung ganz beseitigt und mit den Originaldaten gerechnet.

In [GH94] und [SDFM97] wird Data Smoothing auf das TSP angewendet. Die Glättung bewirkt das Nivellieren der räumlichen Abstände zwischen den

²⁷siehe [CH01a, S. 91]

Knoten, d.h. geringe Abstände werden künstlich verlängert und/oder große Abstände werden künstlich verkürzt. Die Autoren untersuchen verschiedene Glättungsfunktionen (sigmoidal, exponentiell, logarithmisch, hyperbolisch).

In [CGW99] wird die abwechselnde Wiederholung des Verfahrens zwischen einer konvexen und einer konkaven Glättungsfunktion vorgeschlagen. Dieses Vorgehen, das als Sequential Smoothing Algorithmus bezeichnet wird, dominiert die oben genannten Glättungsfunktionen.

Guided Local Search (GLS) GLS wurde erstmalig 1995 von Voudouris und Tsang [VT95] beschrieben. Bei diesem Verfahren wird das Verlassen lokaler Optima durch eine gezielte Modifikation der Zielfunktion erzwungen. Dazu werden Eigenschaften, durch die die bereits gefundenen lokal optimalen Lösungen gekennzeichnet sind, durch einen zusätzlichen Term in der Zielfunktion bestraft. Der Strafterm wird jeweils beim Erreichen eines lokalen Optimums aktualisiert.

Gegeben sei eine Menge von Eigenschaften $F = \{f_1, \dots, f_M\}$. Jeder Eigenschaft $f_i \in F$ können Kosten $c(f_i)$ zugeordnet werden. Die Indikatorfunktion $I_i(y)$ gibt an, ob die Lösung $y \in Y$ die Eigenschaft f_i aufweist oder nicht:

$$I_i(y) = \begin{cases} 1, & \text{falls } y \text{ das Merkmal } f_i \text{ besitzt} \\ 0, & \text{sonst.} \end{cases}$$

Die Festlegung der berücksichtigten Eigenschaften und deren Kosten ist abhängig vom jeweiligen Problem. Für Routing Probleme können z.B. die Zugehörigkeit von Bögen zu einer Lösung als deren Eigenschaften angesehen werden. Die Kosten dieser Eigenschaften können dann durch die Länge des Bogens angegeben werden.

GLS wechselt ständig zwischen einer Lokalen Suche und der Anpassung des Strafvektors $p = (p_1, \dots, p_M)$. Der Lokalen Suche wird die erweiterte Kostenfunktion

$$z'(y) = z(y) + \lambda \cdot \sum_{i=1}^M p_i \cdot I_i(y)$$

zugrunde gelegt. Dabei sind $z(y)$ die ursprüngliche Kostenfunktion des Optimierungsproblems und λ ein Regulierungsparameter, mit dessen Festlegung der Einfluß des Strafterms auf die Bewertung einer Lösung im Verhältnis zu deren ursprünglichen Kosten problemabhängig kontrolliert werden kann.

Zu Beginn des Verfahrens werden alle Komponenten des Strafvektors p auf null gesetzt. Bricht die Lokale Suche in einem lokalen Optimum y' ab, so werden die Komponenten des Strafvektors um eins erhöht, für die der Ausdruck („utility expression“)

$$util(y', f_i) = I_i(y') \cdot \frac{c(f_i)}{1 + p_i}$$

maximal ist. Durch diese Festlegung wird erreicht, daß teure Eigenschaften zuerst bestraft und somit tendenziell vermieden werden. Auf längere Sicht werden die Eigenschaften (bei wiederholter Bestrafung) proportional zu ihren Kosten bestraft.

In [Vou97] wird der Einsatz von GLS zur Lösung des TSP, des Quadratischen Zuordnungsproblems (QAP), des Radio Link Frequency Assignment Problems und für ein Workforce Scheduling Problem untersucht. In [VT98] und [LT98] wird die Idee von GLS auch auf Genetische Algorithmen übertragen. Das resultierende Verfahren nennen die Autoren „Guided Genetic Algorithm“.

2.3.2 Multi-Start Verfahren

Die Erkenntnis, daß bei einer Lokalen Suche oft nur Lösungen in der „Nähe“ der Startlösung untersucht werden, führt zu der einfachen Idee, ausgehend von mehreren möglichst unterschiedlichen Startlösungen jeweils eine neue Suche zu initiieren. Die Möglichkeit, Startlösungen vollkommen zufällig zu wählen („random restart“), wird bereits in [Lin65] und [LK73] betrachtet.

Bei vielen kombinatorischen Problemen erweist sich aber die zufällige Wahl von Startlösungen als nicht sehr erfolgreich, weil zu häufig lokal optimale Lösungen mit relativ schlechten Zielfunktionswerten gefunden werden. Bessere Ergebnisse lassen sich erreichen, wenn man verschiedene aussichtsreiche Startlösungen zur Verfügung hat.

GRASP – Greedy Randomized Adaptive Search Procedure GRASP ist ein iteratives Verfahren, bei dem jede Iteration aus dem Generieren einer Startlösung und einer anschließenden Lokalen Suche besteht. Bei GRASP wird eine sogenannte „adaptive zufallsgesteuerte Greedy-Heuristik“ benutzt, um eine relativ gute Startlösung zu generieren.

Unter einer Greedy-Heuristik versteht man ein konstruktives Verfahren, das eine Lösung schrittweise konstruiert. In jedem Schritt wird je ein Element

(z.B. ein Knoten beim TSP) zur aktuellen Teillösung hinzugefügt bzw. fixiert. Dieses Element wird immer so gewählt, daß der Zuwachs der Zielfunktion minimal ist.²⁸

Bei GRASP wird das nächste zu fixierende Element im Gegensatz zur Greedy-Heuristik aus einer Kandidatenliste (restricted candidate list) *zufällig* ausgewählt. In der Kandidatenliste werden die Festlegungen der vorangegangenen Schritte berücksichtigt. So enthält die Kandidatenliste nur zulässige Elemente, die ggf. gemäß des Greedy-Kriteriums geordnet sind.

Eine sehr erfolgreiche Anwendung von GRASP auf VRPTW wird in [KB95] beschrieben. Eine umfassende Bibliographie der Veröffentlichungen zu Methodik und Anwendungen von GRASP bietet [FR01].

Iterated Local Search (ILS) Beim ILS wird eine neue Startlösung für die Lokale Suche jeweils ausgehend von der aktuellen (lokal optimalen) Lösung bestimmt. Der prinzipielle Ablauf einer ILS wird in [LMS] wie folgt dargestellt:

Algorithmus 2.5. Iterated Local Search

1. $y^0 = \text{GenerateInitialSolution}()$
2. $y^* = \text{LocalSearch}(y^0)$
3. Wiederhole, bis eine Abbruchbedingung erfüllt ist:
 - 3.1 $y' = \text{Perturbation}(y^*, \text{history})$
 - 3.2 $y'^* = \text{LocalSearch}(y')$
 - 3.3 $y^* = \text{AcceptanceCriterion}(y^*, y'^*, \text{history})$

Dabei bezeichnen *GenerateInitialSolution* eine Eröffnungsheuristik und *LocalSearch* eine Verbesserungsheuristik. Letztere ist meistens (aber nicht notwendig) eine Lokale Suche.

Mit der Methode *Perturbation* wird zu der jeweils aktuellen (lokal optimalen) Lösung y^* eine neue Startlösung y' erzeugt. Dazu bedient man sich häufig eines Nachbarschaftsschritts („kick-move“) einer anderen Nachbarschaft. Die erzeugte neue Startlösung y' soll sich von der Lösung y^* so unterscheiden, daß das nachfolgende Verbesserungsverfahren *LocalSearch* i.d.R. nicht dazu in der Lage ist, die Veränderung von y^* zu y' wieder rückgängig zu machen. Dennoch soll die Lösungsqualität von y' nicht zu schlecht

²⁸Eine Greedy-Heuristik für das TSP ist z.B. die Nächste-Nachbar-Heuristik.

sein. Das kann z.B. dadurch erreicht werden, daß *Perturbation* nur wenige Teile (z.B. nur wenige Bögen) einer Lösung ändert.²⁹ Alternativ zum Einsatz von kick-Schritten kann z.B. auch eine Lokale Suche mit einer modifizierten Zielfunktion oder gestörten Daten (vgl. „Noising Methods“ auf Seite 57) etc. eingesetzt werden. Der Parameter *history* steht für Informationen, die den bisherigen Verlauf der Suche betreffen. Diese können Einfluß auf das Verhalten von *Perturbation* haben.

Bei geeigneter Wahl von *Perturbation* wird das Verfahren *LocalSearch* die Lösung y' mit hoher Wahrscheinlichkeit mit wenigen Verbesserungen in eine lokal optimale Lösung y^* überführen, die eine hohe Lösungsqualität aufweist. Da y^* aber schlechter sein kann als y' , wird mit der Methode *AcceptanceCriterion* explizit darüber entschieden, ob diese Lösung Ausgangspunkt für die weitere Suche sein soll oder nicht. Auch diese Entscheidung kann (aber muß nicht) vom bisherigen Verlauf der Suche abhängig gemacht werden.

Wird hier (wie beim Simulated Annealing) das Metropolis-Akzeptanzkriterium verwendet, so erhält man ein spezielles Simulated Annealing Verfahren. Dieses ist durch sehr große Schritte gekennzeichnet, welche (aufgrund der in jedem Schritt durchgeführten lokalen Optimierung) stets zu einem lokalen Optimum führen. Dieses Vorgehen wird auch als „Large-step Markov Chain“ [MOF91] oder als „Chained Local Optimization“ [MO96] bezeichnet.

ILS wurde mit beeindruckendem Erfolg auf das TSP angewandt, wobei der sehr leistungstarke Lin-Kernighan-Algorithmus als lokales Suchverfahren dient („Iterated Lin-Kernighan“).³⁰ Nachfolgend wurde ILS auch zur Lösung vieler weiterer kombinatorischer Optimierungsprobleme eingesetzt. Eine ausführliche Beschreibung von ILS und einen Überblick über eine Reihe von Anwendungen bietet [LMS].

2.3.3 Populationsbasierte bzw. evolutionäre Verfahren

Genetische Algorithmen und Evolutionäre Strategien Genetische Algorithmen (GA) und Evolutionäre Strategien (ES) orientieren sich an der

²⁹Für das (symmetrische) TSP wird z.B. häufig der „double bridge move“ als Kick-Schritt verwendet, da er durch sequentielle k -opt-Schritte nicht rückgängig gemacht werden kann. Andererseits werden durch diesen speziellen 4-opt-Schritt lediglich vier Kanten ersetzt, so daß die erhaltene Lösung viele gute Eigenschaften der Lösung y^* behält. Siehe z.B. [MOF91], [MO96] oder [HRM97].

³⁰ siehe [Joh90], [MOF91], [ABCC98] und [ACR00]

Anpassung einer Population von Individuen über mehrere Generationen an die äußeren Bedingungen (natürliche Auswahl). Sie sind dadurch gekennzeichnet, daß zu jedem Zeitpunkt mehrere Lösungen verwaltet und (quasi) gleichzeitig modifiziert werden können. Eine solche Menge von Lösungen wird in Anlehnung an das natürliche Vorbild als Population bezeichnet. Beide Verfahren basieren auf der wiederholten Anwendung der drei Operatoren Selektion, Rekombination³¹ und Mutation auf der jeweils aktuellen Population.³²

Die grundlegenden Ideen der GA gehen auf die Arbeit von Holland [Hol75] zurück. De Jong [De 75] und Goldberg [Gol89] zeigten, daß es möglich ist, komplexe Probleme mit GA zu lösen. Die ES wurden von bereits 1973 von Rechenberg [Rec73] vorgeschlagen und von Schwefel [Sch81] aufgegriffen.

Im Gegensatz zur englischsprachigen Literatur, wo ES lange Zeit relativ wenig Beachtung fanden, ist es in der deutschsprachigen Literatur üblich, eine begriffliche Trennung von GA und ES vorzunehmen. GA (im engeren Sinne) sind dann dadurch gekennzeichnet, daß eine Beschreibung der Lösungen anhand von Chromosomen (Bitvektoren) erfolgt und die Rekombination und Mutation auf diesen Chromosomen operieren. Bei den ES werden i.d.R. keine Bitvektoren für die Darstellung einer Lösung verwendet. ES variieren außerdem gezielt die Größe der Population. Dadurch können u.a. auch eine Intensivierung bzw. Diversifizierung der Suche erreicht werden. Aktuelle Veröffentlichungen sprechen häufig von Genetischen Algorithmen, auch wenn keine Chromosomen gebildet werden.

Die Fortschreibung der Populationen geschieht bei GA und ES nach dem gleichen Schema:

³¹Die Rekombination wird häufig auch als „Kreuzung“ bzw. als „Crossover“ bezeichnet.

³²Strenggenommen fallen GA und ES nicht unter den Begriff einer Meta-Heuristik, denn sie dienen ihrer originären Idee zufolge nicht dazu, eine untergeordnete Suche zu steuern. Insbesondere beim Einsatz von GA zur Lösung von kombinatorischen Optimierungsproblemen werden sie jedoch oft mit einer Lokalen Suche oder anderen Suchverfahren kombiniert, um die Effektivität und Effizienz des Verfahrens zu erhöhen. So gibt es z.B. Realisierungen, bei denen der Kreuzungsoperator oder der Mutationsoperator eine Lokale Suche durchführen. In der Literatur werden die GA deshalb fast immer den Meta-Heuristiken zugeordnet.

Algorithmus 2.6. Genetischer Algorithmus/Evolutionäre Strategie

1. (*Startlösung*) Generiere eine Startpopulation P .
2. Wiederhole, bis eine Abbruchbedingung erfüllt ist:
 - 2.1 Bestimme die „Fitness“ jeder Lösung $y^i \in P$.
 - 2.2 (*Selektion*) Erzeuge durch Auswahl von Lösungen aus P eine Eltern-Population P' .
 - 2.3 (*Rekombination*) Erzeuge eine neue Population P'' von Lösungen („Nachkommen“), die jeweils durch eine Rekombination von je zwei (oder mehr) Lösungen aus P' („Eltern“) gebildet werden.
 - 2.4 (*Mutation*) Modifiziere zufällig einige Lösungen von P'' .
 - 2.5 Erzeuge eine neue Population P^{akt} durch Auswahl von Elementen aus P^{akt} und P'' .

Die Selektion erfolgt stets zufallsgesteuert, wobei Lösungen mit einer höheren Fitness eine höhere Auswahl-Wahrscheinlichkeit besitzen. Normalerweise wird das sogenannte Roulette-Verfahren angewendet, d.h. die Wahrscheinlichkeit p_i für die Auswahl einer Lösung $y^i \in P$ ist

$$p_i = \frac{z(y^i)}{\sum_{y \in P} z(y)}.$$

Von dieser Vorgabe abgesehen, können Selektion, Rekombination und Mutation sehr unterschiedlich aufgefaßt und implementiert werden. Dies hat zu einer großen Vielfalt unterschiedlicher Realisierung geführt. Auch die Anwendungen von GA (und ES) sind sehr zahlreich und vielfältig.

Scatter Search und Path Relinking Scatter Search und Path Relinking verwalten ebenso wie GA und ES mehrere Lösungen gleichzeitig (Referenzmenge) und erzeugen neue Lösungen anhand der Kombination von Eigenschaften mehrerer bekannter Lösungen. Beide Verfahren können nach folgendem Muster ablaufen:

Algorithmus 2.7. Template für Scatter Search/Path Relinking³³

1. (*Ausgangslösungen*) Generiere eine oder mehrere zufällig gewählte Ausgangslösungen.
2. Wiederhole für eine bestimmte Anzahl von Iterationen:
 - 2.1 (*Diversifikation*) Generiere zu jeder Ausgangslösungen eine Menge divers verteilter Testlösungen.
 - 2.2 (*Verbesserung und Aktualisierung der Referenzmenge*) Wende auf alle im Schritt 2.1 erzeugten Testlösungen ein Verbesserungsverfahren an, um eine oder mehrere verbesserte Lösungen zu erzeugen. Aktualisiere die Referenzmenge, die aus den b besten gefundenen Lösungen besteht.
3. (*Scatter Search/Path Relinking Phase*) Wiederhole für eine bestimmte Anzahl von Iterationen:
 - 3.1 (*Teilmengen generieren*) Generiere Teilmengen der Referenzmenge als Basis für die Erzeugung kombinierter Lösungen.
 - 3.2 (*Kombiniere Lösungen*) Erzeuge zu jeder im Schritt 3.1 erzeugten Teilmenge X mit einer „Kombinationsmethode“ eine Menge $C(X)$ von kombinierten Lösungen.
 - 3.3 (*Verbesserung und Aktualisierung der Referenzmenge*) Wende ein Verbesserungsverfahren auf alle im Schritt 3.2 erzeugten Lösungen an, um eine oder mehrere verbesserte Lösungen zu erzeugen. Aktualisiere die Referenzmenge.

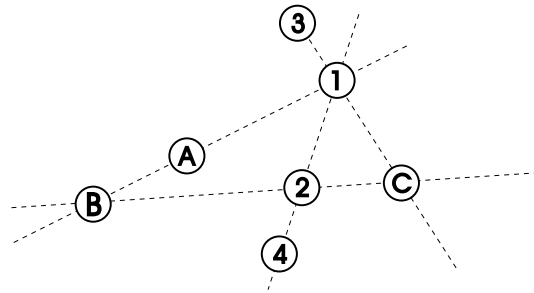
Scatter Search und Path Relinking unterscheiden sich nur darin, wie der Schritt 3.2 ausgeführt wird. Beim Scatter Search werden „lineare Kombinationen“ erzeugt, indem Eigenschaften von mehreren Referenzlösungen (nicht-konvex) miteinander kombiniert (und, sofern notwendig, auf ganzzahlige Werte gerundet) werden.

Beim Path Relinking werden Kombinationen auf einem Pfad in einem „Nachbarschaftsraum“ von einer Referenzlösung zu einer anderen (und darüber hinaus) erzeugt. Die „linearen Kombinationen“ des Scatter Search können als spezielle Pfade in einem euklidischen Raum aufgefaßt werden. Daher bezeichnet Glover das Path Relinking als eine verallgemeinerte Form des Scatter Search.

Eine detaillierte Beschreibung und Verweise auf Anwendungen von SS und PR finden sich in [Glo98] und [Glo99].

³³siehe [Glo98, S. 10]

³⁵Die Abb. 2.14 wurde aus [Glo99] übernommen.



A, B, C sind Lösungen der Referenzmenge. Es wird eine Menge von Lösungen auf dem durch A und B definierten Linien-Segment erzeugt. Aus dieser Menge wird die Lösung 1, die aus einer nicht-konvexen Kombination von A und B resultiert, in die Referenzmenge aufgenommen. Analog werden die Lösungen 2 bis 4 erzeugt.

Abbildung 2.14: Beispiel für „lineare Kombinationen“ beim Scatter Search im zweidimensionalen Raum.³⁵

Ameisen-Algorithmen Ameisen-Algorithmen³⁶ basieren auf der Simulation des Verhaltens einer Menge von Ameisen, die ein (Optimierungs-)Problem mittels einer sehr einfachen Kommunikation kooperativ lösen. Die Idee wurde zuerst von Colorni et. al. [CDM91] veröffentlicht.

Bei der algorithmischen Umsetzung wird für jede Ameise ein einfacher Prozeß (Agent) verwendet, der schrittweise eine neue Lösung des Problems aus einzelnen Komponenten k_i aufbaut. In jedem Schritt wird die bereits erzeugte partielle Lösung a durch Hinzufügen einer weiteren Komponente erweitert. Die Auswahl der nächsten Komponente basiert auf einer Bewertung der noch verwendbaren Komponenten bezüglich der Teillösung a .

Die wesentliche Idee der Ameisen-Algorithmen besteht darin, daß die Bewertung nicht ausschließlich anhand von (a priori vorhandenen) Informationen über das Problem vorgenommen wird. Zusätzlich werden auch Informationen bezüglich bereits erzielter Resultate während des Ablaufs des Verfahrens verwendet. Außerdem agieren immer mehrere Ameisen parallel.

Der folgende Algorithmus gibt die grobe Struktur eines Ameisen-Algorithmus wieder (vgl. [MC01, S. 475]). Dabei bezeichnet $\eta_i(a)$ eine a priori

³⁶In der englischsprachigen Literatur werden die Bezeichnungen „Ant-System“ (AS) bzw. „Ant Colony Optimization“ (ACO) verwendet.

feststellbare Bewertung der Einfügung von Komponente k_i in die Teillösung a und $\tau_i(a)$ ein Maß für die Qualität der Lösungen, die mit Hilfe der Komponente k_i bereits erzeugt wurden. m ist die Zahl der Ameisen und n gibt an, wieviele Komponenten für die Erzeugung einer Lösung ausgewählt werden müssen.

Algorithmus 2.8. Grundstruktur eines Ameisen-Algorithmus

1. Wiederhole, bis eine Abbruchbedingung erfüllt ist:
 - 1.1 Initialisiere einen Vektor $A = \{a_1, \dots, a_m\}$ partieller Lösungen:
 $a_i := \emptyset, i = 1, \dots, m$.
 - 1.2 Wähle für jede Ameise $i = 1, \dots, m$ zufällig eine zulässig einfügbare Komponente k_i zur Erweiterung der partiellen Lösung a_i aus. Dabei sind die Wahrscheinlichkeiten gemäß einer Funktion abhängig von a_i , $\eta_i(a_i)$ und $\tau_i(a_i)$ verteilt.
 - 1.3 Wenn es partielle Lösungen a_i in A gibt, die noch vervollständigt werden müssen (und zulässig vervollständigt werden können), gehe zu 1.2.
 - 1.4 Bestimme für jede Ameise $i = 1, \dots, m$ den Zielfunktionswert $z(a_i)$ und aktualisiere die Werte $\tau_j, j = 1, \dots, n$ entsprechend.

In den letzten Jahren wurden eine Reihe unterschiedlicher Ameisen-Algorithmen entwickelt. Diese unterscheiden sich in vielfältiger Weise vor allem in der Definition der Wahrscheinlichkeitsverteilung im Schritt 1.2 und in der Aktualisierung der Werte von $\tau_j, j = 1, \dots, n$ im Schritt 1.4. Einen umfassenden Überblick über verschiedene Erweiterungen und Anwendungen der Ameisen-Algorithmen geben [Tai99a] [MC01].

2.3.4 Verfahren mit Gedächtnis

Tabu Search (TS) Eine Sonderstellung unter den Meta-Heuristiken nimmt das von Glover 1986 mit [Glo86] und [Glo89] eingeführte Tabu Search (TS) ein. TS faßt eine Reihe von Prinzipien zusammen, die für die intelligente Steuerung der Suche eingesetzt werden können. Eine zentrale Rolle nimmt dabei die sinnvolle Speicherung und Nutzung eines Teils der im Lauf der Suche generierten Informationen in Form eines „Gedächtnisses“ ein.³⁷

³⁷Die in der Literatur häufig zu findende Gleichsetzung von Tabu-Search mit der (alleinigen) Verwendung einer Tabu-Liste für die Steuerung der Suche entspricht nicht der eigentlichen Idee von Tabu-Search. So schreiben Glover und Laguna in [GL97, S. 1]: „*The philosophy of tabu search (TS) is to derive and exploit a collection of principles of intelligent problem solving.*“

Die Ausgestaltung der Prinzipien muß bei der Anwendung auf ein konkretes Problem oder eine Problemklasse vorgenommen werden. Dementsprechend beschäftigen sich zahlreiche Publikationen der letzten Jahre mit der möglichen Ausgestaltung dieser Prinzipien für den Einsatz innerhalb von Heuristiken für Vehicle Routing und Scheduling Problem und viele andere kombinatorische Probleme.

Eine angemessene Darstellung aller im Rahmen von Tabu-Search propagierten Ideen und Prinzipien kann im Rahmen dieser Arbeit nicht erfolgen. U.a. werden auch viele Ideen der bereits beschriebenen Meta-Heuristiken im Rahmen von Tabu Search aufgegriffen. An dieser Stelle sei nur eine Auswahl der wichtigsten Prinzipien genannt:

- Verwendung eines Kurzzeitgedächtnisses (z.B. in Form einer Tabu-Liste, was den Namen Tabu Search motiviert), um lokale Optima verlassen zu können
- Verwendung eines Langzeitgedächtnisses für die Umsetzung übergeordneter Suchziele, wie
 - Intensivierung und Diversifikation
 - strategische Oszillation
 - „Path Relinking“
- Adaptivität des Gedächtnisses auf der Basis von Aktualität, Häufigkeit, Qualität und Einfluß eines Nachbarschaftsschritts
- Suche auf Basis von Ejection Chains
- Generieren neuer Lösungen oder Attribute aus bereits gefundenen („Vocabulary Building“)
- Verwendung von Kandidatenlisten

Eine umfassende Darstellung von Tabu Search bietet das von Glover und Laguna veröffentlichte Buch [GL97].

Reactive Tabu Search In den letzten Jahren erschienen auch eine Reihe von Veröffentlichungen, die den Einsatz von Reactive Tabu Search beschreiben. Dabei handelt es sich um eine in [BT94] eingeführte Erweiterung des Tabu Search, die eine automatische Anpassung der Tabu-Listen im Verlauf der Optimierung vornimmt. Dadurch soll einerseits das Verharren der

Suche in einem kleinen Teilbereich des Lösungsraums wirksam verhindert werden (was sonst nur mit langen Tabu-Listen möglich wäre). Andererseits sollen lange Tabu-Listen vermieden werden, da diese den betrachteten Lösungsraum einschränken und dadurch das Auffinden attraktiver neuer Lösungen erschweren können.

Die Kernidee besteht darin, die Anzahl und Frequenz des Auftretens gleicher Lösungen innerhalb des Suchprozesses zu protokollieren und ständig zu überwachen. Tritt eine mehrfache Wiederholung gleicher Zustände auf, wird die Tabu-Liste verlängert. Treten längere Zeit keine Wiederholungen auf, wird sie verkürzt. Überschreiten ein oder mehrere Zustände mehrfach eine vorgegebene Maximalzahl an Wiederholungen, so wird ein Verharren der Suche in der Nähe eines chaotischen Attraktors vermutet und eine Folge von Zufallsschritten ausgeführt, um diese Region verlassen zu können.

Anwendungen von Reactive Tabu Search beschreiben z.B. [BP97], [CR97], [BB99], [NB00] und [BR01].

Kapitel 3

Lokale Suche in der k -opt-Nachbarschaft

Gegenstand dieses Kapitels ist die Beschreibung der k -opt-Nachbarschaft für Vehicle Routing und Scheduling Probleme. Dazu wird zunächst eine formale Beschreibung einer Lösung des TSP und der k -opt-Nachbarschaft beim TSP entwickelt (Abschnitt 3.1). Diese Darstellungen werden dann schrittweise auf das TSPR und die k -opt-Nachbarschaft für das TSPR (Abschnitt 3.2) sowie auf das m-TSPR und die zugehörige k -opt-Nachbarschaft erweitert (Abschnitt 3.3). Im Abschnitt 3.4 wird darauf eingegangen, wie der Aufwand der Lokalen Suche bei großen Nachbarschaften sinnvoll eingeschränkt werden kann.

3.1 Lokale Suche für das TSP

Das Traveling Salesman Problem (TSP) ist eines der bekanntesten und meist untersuchten Probleme der kombinatorischen Optimierung. Die Entwicklung von Lösungsverfahren für das TSP ist Gegenstand intensiver Forschung seit mehr als vierzig Jahren. Dies liegt offenbar vor allem daran, daß das TSP einerseits leicht zu formulieren und zu verstehen ist, dessen optimale Lösung aber andererseits ein $\mathcal{NP}$ -hartes Problem darstellt. Außerdem tritt das TSP bei vielen Lösungsansätzen für komplexere kombinatorische Optimierungsprobleme als Teilproblem auf und ist häufig Teil von angewandten Problemstellungen.

In diesem Abschnitt werden k -opt-Schritte und die k -opt-Nachbarschaft

ten für das TSP beschrieben. Dabei geht es nicht darum, die Lokale Suche in k -opt-Nachbarschaften tatsächlich zur Lösung des TSP einzusetzen. Vielmehr sollen die Methoden auf die Erweiterungen des TSP bis zum m-TSPR angewandt werden können. Das TSP dient daher als Basis-Modell, anhand dessen eine Reihe wichtiger Begriffe eingeführt und erklärt werden.

Zunächst wird eine Modellierung des (asymmetrischen) TSP anhand graphentheoretischer Begriffe vorgenommen. Mit deren Hilfe wird die k -opt-Nachbarschaft formal beschrieben.

3.1.1 Einige Begriffe der Graphentheorie

Zuerst werden einige Grundbegriffe der Graphentheorie definiert. Dies dient im wesentlichen der Festlegung der im weiteren benutzten Schreibweisen. Darauf aufbauend, wird dann der Verkettungsoperator $\oplus$ definiert, der es erlaubt, das Zusammenfügen von Pfaden aus Teilpfaden auf einfache Weise zu beschreiben.

Definition 3.1. (gerichteter Graph)

Ein gerichteter Graph $G = (V, A)$ besteht aus einer nicht-leeren Knotenmenge $V = \{0, \dots, n\}$ und einer nicht-leeren Bogenmenge $A \subseteq \{(i, j) : i, j \in V\}$.

Bögen der Form (i, i) , die einen Knoten $i \in V$ mit sich selbst verbinden, werden als Schlinge bzw. Loop bezeichnet. Wir interessieren uns für gerichtete Graphen, die keine Schlingen und keine parallelen Bögen aufweisen. Diese heißen *schlichte* Graphen. Wenn es in einem schlichten gerichteten Graphen zu jedem Paar verschiedener Knoten einen Bogen gibt, spricht man von einem vollständigen Graphen.

Definition 3.2. (bewerteter Graph)

Der gerichtete Graph $G = (V, A)$ kann mit der Abbildung $c : A \rightarrow \mathbb{R}$ zu einem bewerteten Graphen $G = (V, A, c)$ erweitert werden. Die Bewertung $c_{ij} = c(i, j)$ eines Bogens $(i, j) \in A$ wird auch als Kosten des Bogens bezeichnet.

Es werden nun einige Begriffe und Schreibweisen vereinbart, die sich für die nachfolgenden Definitionen und schließlich für eine kompakte Beschreibung der k -opt-Nachbarschaften als nützlich erweisen werden.

Definition 3.3. (Bogenfolge, Menge der Bogenfolgen)

Es sei $G = (V, A)$ ein schlichter gerichteter Graph. Dann heißt $b = (v_0, \dots, v_m) \in V^{(m+1)}$ eine Bogenfolge der Länge m ($m \geq 0$) auf G , wenn

für alle $j \in \{1, \dots, m\}$ gilt: $a_j = (v_{j-1}, v_j) \in A$.

Bogenfolgen b der Länge $m \geq 1$ können alternativ auch durch Angabe der m Bögen dargestellt werden, welche die Knoten $v_0, \dots, v_m$ miteinander verbinden: $b = (a_1, \dots, a_m)$.

Die Länge einer Bogenfolge b bezeichnet die Zahl der in b enthaltenen Bögen und wird mit $|b|$ angegeben. B^m bezeichnet die Menge aller Bogenfolgen der Länge m und B die Menge aller Bogenfolgen $B = \bigcup_{k \geq 0} B^k$.

Bemerkung 3.4. Eine Bogenfolge der Länge 0 enthält genau einen Knoten, eine Bogenfolge der Länge 1 enthält genau einen Bogen. Zur Vereinfachung der Darstellung identifizieren wir nachfolgend den Knoten $v_i \in V$ und die Bogenfolge $(v_i) \in B^0$ sowie den Bogen $(i, j) \in A$ und die Bogenfolge $((i, j)) \in B^1$ miteinander. Demzufolge gilt auch $V = B^0$ und $A = B^1$.

Bemerkung 3.5. In einem vollständigen gerichteten Graphen $G=(V, A)$ definiert jedes Tupel $(v_0, \dots, v_m) \in V^{(m+1)}$ eine Bogenfolge der Länge m , d.h., es gilt $(v_0, \dots, v_m) \in B^m$.

Definition 3.6. ($V(b^m)$)

Es sei b^m eine Bogenfolge der Länge m . Dann bezeichnet $V(b^m) \subseteq V$ die Menge aller Knoten, die mit der Bogenfolge inzidieren, d.h. für $b^m = (v_0, \dots, v_m)$ gilt $V(b^m) = \{v_0, \dots, v_m\}$.

Definition 3.7. ($A(b^m)$)

Es sei b^m eine Bogenfolge der Länge m . Dann bezeichnet $A(b^m) \subseteq A$ die Menge aller Bögen, die in der Bogenfolge b^m benutzt werden, d.h. $A(b^0) = \emptyset$ und für $b^m = (a_1, \dots, a_m)$ gilt $A(b^m) = \{a_1, \dots, a_m\}$.

Definition 3.8. (Kosten einer Bogenfolge)

Es sei $G = (V, A, c)$ ein gerichteter bewerteter Graph und $b^m \in B^m$ eine Bogenfolge auf G . Dann bezeichnet

$$c(b^m) = \sum_{a \in A(b^m)} c(a)$$

die Kosten der Bogenfolge b^m .

Aufbauend auf dem Begriff der Bogenfolge können wir nun die Begriffe *Pfad* und *Kreis* einführen. Diese werden es uns später gestatten, Lösungen des TSP und Nachbarschaftsrelationen zwischen diesen Lösungen zu beschreiben.

Definition 3.9. (Pfad, Menge aller Pfade)

Eine Bogenfolge $p^m = (v_0, \dots, v_m)$ der Länge m auf G heißt ein Pfad, wenn alle Knoten von p^m paarweise verschieden sind, d.h. es gilt

$$v_i \neq v_j \quad \text{für alle } i, j \in \{0, \dots, m\}, i \neq j.$$

P^m bezeichnet die Menge aller Pfade der Länge m von G und P die Menge aller Pfade: $P = \bigcup_{i=0}^n P^i$.

Definition 3.10. (Hamiltonpfad, Menge aller Hamiltonpfade)

Ein Hamiltonpfad p^H des Graphen G ist ein Pfad, der jeden Knoten von G enthält, d.h., es gilt $V(p^H) = V$. P^H bezeichnet die Menge aller Hamiltonpfade von G .

Definition 3.11. (Kreis, Menge aller Kreise)

Eine Bogenfolge $k^m = (v_0, \dots, v_m)$ der Länge m auf G heißt ein Kreis, wenn die ersten m Knoten paarweise verschieden sind und der letzte Knoten gleich dem ersten Knoten ist, d.h. es gilt

$$\begin{aligned} v_i &\neq v_j && \text{für alle } i, j \in \{0, \dots, m-1\}, i \neq j, \\ v_m &= v_0. \end{aligned}$$

K^m bezeichnet die Menge aller Kreise der Länge m von G und K die Menge aller Kreise, also $K = \bigcup_{i=0}^n K^i$.

Bemerkung 3.12. Ein Kreis der Länge 1 ist eine Schlinge. In einem schlichten Graphen kann es daher per Definition keinen Kreis der Länge 1 geben.

Definition 3.13. (Hamiltonkreis, Menge aller Hamiltonkreise)

Ein Hamiltonkreis k^H des Graphen G ist ein Kreis, der jeden Knoten von G enthält, d.h. es gilt $V(k^H) = V$. K^H bezeichnet die Menge aller Hamiltonkreise von G .

Nun kann der Verkettungsoperator $\oplus$ definiert werden. Dieser erlaubt es, mehrere Bogenfolgen miteinander zu verknüpfen.

Definition 3.14. (Verkettung von Bogenfolgen: $r \oplus s$)

Sei $r = (v_0, \dots, v_k) \in B^k$, $s = (v'_0, \dots, v'_l) \in B^l$ und $(v_k, v'_0) \in A$. Dann bewirkt der Verkettungsoperator $\oplus : B \times B \rightarrow B$ die Abbildung:

$$r \oplus s = (v_0, \dots, v_k, v'_0, \dots, v'_l) \in B^{(k+s+1)}.$$

Bemerkung 3.15. Der Verkettungsoperator $\oplus$ erfüllt das Assoziativgesetz, d.h. für $b_1, b_2, b_3 \in B$ gilt

$$(b_1 \oplus b_2) \oplus b_3 = b_1 \oplus (b_2 \oplus b_3) = b_1 \oplus b_2 \oplus b_3.$$

Aus der Bemerkung 3.4 und Definition 3.14 folgt sofort, daß man jede Bogenfolge auch als eine Verkettung ihrer Knoten darstellen kann:

Bemerkung 3.16. Jede Bogenfolge $b^m = (v_0, \dots, v_m)$ kann als Verkettung ihrer Knoten aufgefaßt werden:

$$b^m = v_0 \oplus v_1 \oplus \dots \oplus v_m = \bigoplus_{i=0}^m v_i.$$

Bemerkung 3.17. Seien $r, s \in P$ zwei Pfade von G . Dann ist $r \oplus s$ genau dann ein Pfad von G , wenn r und s keinen gemeinsamen Knoten besitzen:

$$r \oplus s \in P \iff V(r) \cap V(s) = \emptyset.$$

Nachfolgend werden ausschließlich vollständige gerichtete Graphen betrachtet. Zur Vereinfachung der Schreibweise wird dies aber nicht jedesmal angegeben.

3.1.2 Ein Modell für das TSP

Das Traveling Salesman Problem (TSP) kann anschaulich wie folgt formuliert werden: Ein Handelsreisender hat n Kunden genau einmal zu besuchen und anschließend zu seinem Ausgangspunkt (Depot¹) zurückzukehren. Gesucht ist eine Reihenfolge der Kunden, bei der die insgesamt zurückgelegte Strecke minimal ist.

Wir betrachten im folgenden grundsätzlich asymmetrische Traveling Salesman Probleme.² Ein TSP mit n Städten kann basierend auf dem Begriff

¹Beim TSP ist der Standort des Handelsreisenden gegenüber den Kundenstandorten nicht ausgezeichnet. Deshalb ist es nicht üblich, diesen Standort als Depot zu bezeichnen. Der Begriff des Depots wird aber einer einheitlichen Darstellung und Sprechweise halber bereits hier eingeführt, da er für Modelle mit mehreren Touren (m-TSP) immer benötigt wird.

²In der Literatur wird teilweise auch das symmetrische TSP betrachtet, bei dem vorausgesetzt wird, daß die Kosten eines Bogens unabhängig von seiner Orientierung sind, d.h. es gilt stets $c_{ij} = c_{ji}$. Das symmetrische TSP ist insbesondere deshalb von Bedeutung, weil es eine Reihe von Algorithmen gibt, die eine symmetrische Kostenmatrix voraussetzen. Man kann sich ggf. dadurch helfen, daß sich ein asymmetrisches TSP mit n Kunden in ein äquivalentes symmetrisches TSP mit $2n$ Kunden überführen läßt [KL].

des Hamiltonkreises eines Graphen mit $n + 1$ Knoten definiert werden:

Definition 3.18. (TSP als Hamiltonkreis (Kreismodell))

Sei $G' = (V', A', c')$ ein vollständiger gerichteter und bewerteter Graph mit

$$\begin{aligned} V' &= \{0, \dots, n\}, \\ A' &= \{(i, j) : i, j \in V, i \neq j\}. \end{aligned}$$

Jedem Bogen (i, j) sind die Kosten $c'_{ij} = c'(i, j)$ zugeordnet.

Dann ist das TSP die Frage nach einem Hamiltonkreis $k \in K^H(G')$ mit minimalen Kosten:

$$TSP : \quad \min \quad c'(k) \quad (3.1a)$$

so daß

$$k \in K^H(G') \quad (3.1b)$$

Wenn man den Knoten 0 als Ausgangsstandort des Handelsreisenden und die Knoten $1, \dots, n$ als die Standorte der Kunden auffaßt und die Kosten c als Entfernungen zwischen den Standorten interpretiert, entspricht das Modell 3.18 genau der zuvor angegebenen anschaulichen Formulierung des TSP. Im folgenden werden wir das Modell 3.18 zur Vereinfachung der Schreibweise als „Kreismodell des TSP“ bezeichnen.

Alternativ kann man das TSP auf der Basis eines Hamiltonpfades für einen Graphen mit $n + 2$ Knoten definieren. Dabei repräsentiert der Knoten 0 den Abfahrtsstandort (Depotausgang) und der Knoten $n + 1$ den Zielstandort (Depoteingang) des Handelsreisenden. Diese Definition, die wir im weiteren als „Pfadmodell des TSP“ bezeichnen werden, ist für viele der nachfolgenden Betrachtungen besser geeignet und wird daher eine wichtige Rolle spielen.

Definition 3.19. (TSP als Hamiltonpfad (Pfadmodell))

Sei $G = (V, A, c)$ ein schlichter gerichteter und bewerteter Graph mit

$$\begin{aligned} V &= \{0, \dots, n + 1\}, \\ A &= \{(i, j) : i \in V \setminus \{n + 1\}, j \in V \setminus \{0\}, i \neq j\}. \end{aligned}$$

Jedem Bogen (i, j) sind die Kosten $c_{ij} = c(i, j)$ zugeordnet.

Dann ist das TSP die Frage nach einem Hamiltonpfad $p \in P^H$ mit minimalen

Kosten, der ausgehend vom Knoten 0 zum Knoten $n+1$ führt:

$$TSP : \quad \min \quad c(p) \quad (3.2a)$$

so daß

$$p = 0 \oplus p' \oplus n + 1, \quad p' \in P^n, \quad p \in P^H. \quad (3.2b)$$

Bemerkung 3.20. Das Kreismodell und das Pfadmodell können wie folgt äquivalent ineinander transformiert werden:³ Ein äquivalentes Kreismodell geht aus dem Pfadmodell hervor, wenn der Graph G' gebildet wird, indem die Knoten 0 und $n + 1$ des Graphen G zum Knoten 0 des Graphen G' vereinigt werden. Die Kosten der Bögen können einfach übertragen werden:

$$\begin{aligned} c'_{ij} &:= c_{ij} & \forall i, j \in \{1, \dots, n\}, \\ c'_{i0} &:= c_{i(n+1)} & \forall i \in \{1, \dots, n\} \quad \text{und} \\ c'_{0j} &:= c_{0j} & \forall j \in \{1, \dots, n\} \end{aligned}$$

Jeder Hamiltonpfad $p^H \in P^H(G)$ wird dann auf einen Hamiltonkreis $k^H \in K^H(G')$ mit gleichen Kosten abgebildet.

Durch die Umkehr der obigen Abbildung kann ausgehend vom Graphen G' der Graph G erhalten werden. Dadurch wird aus einem Hamiltonkreis in G' ein Hamiltonpfad in G mit identischen Kosten (siehe Abb. 3.1).

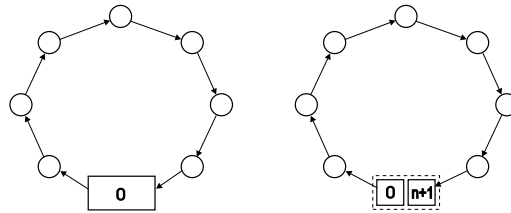


Abbildung 3.1: Überführen eines Hamiltonkreises in einen Hamiltonpfad

Unabhängig davon, ob das Kreismodell oder das Pfadmodell zugrunde liegt, werden die Knoten $N = \{1, \dots, n\}$ im weiteren als Kundenknoten

³Unter der Äquivalenz zweier Modelle wird verstanden, daß es eine eindeutige Abbildung gibt, die jede optimale Lösung des einen Modells auf eine optimale Lösung des anderen Modells abbildet. Auf den Beweis der Äquivalenz von Kreismodell und Pfadmodell, der rein technischen Charakter hätte, wird hier verzichtet.

bezeichnet. Die übrigen Knoten heißen Depotknoten. Um eine einheitliche Schreibweise zu erleichtern, bezeichnen wir einen Depotausgang auch mit o und einen Depoteingang mit d . Beim Kreismodell des TSP bezeichnen o und d demzufolge denselben Knoten. Jeder Kreis, der einen Depotknoten beinhaltet, bzw. jeder Pfad, der in einem Depotknoten beginnt und endet, wird Tour genannt. Insbesondere ist jede Lösung des TSP eine Tour, die alle Kundenknoten enthält.

3.1.3 Ein Modell für die k -opt-Nachbarschaft

Die k -opt-Nachbarschaft wurde bereits im Abschnitt 2.2.2 auf Seite 40 erläutert. Hauptgegenstand dieses Unterabschnitts ist die formale Beschreibung der k -opt-Nachbarschaften.

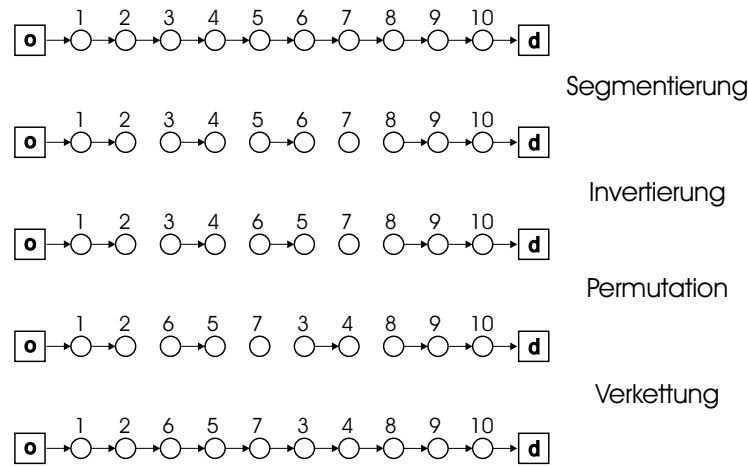
In der Literatur ist es üblich, die k -opt-Nachbarschaft entsprechend ihrer anschaulichen Definition zu modellieren. Der Fokus der Betrachtung liegt dabei auf den k gelöschten und den k eingefügten Kanten oder Bögen. Eine solche Beschreibung ist jedoch nicht immer gut geeignet, um die wesentlichen Unterschiede der so erzeugten Lösung und eine Systematik bzw. Struktur der k -opt-Nachbarschaft deutlich zu machen. Deshalb wird in dieser Arbeit eine Modellierung der k -opt-Nachbarschaft anhand der im wesentlichen unveränderten Bestandteile der Lösung vorgenommen. Im Zentrum der Betrachtungen stehen die erhalten gebliebenen Teilpfade (Segmente) der Ausgangslösung, die nur ihre Orientierung und/oder ihre Reihenfolge untereinander ändern können.

k -opt-Nachbarschaft

Es sei eine zulässige Ausgangslösung gegeben, die durch einen Hamiltonpfad repräsentiert wird. Dieser Pfad geht vom Depotknoten o aus und endet im Depotknoten d . Der Übergang zu einer Nachbarlösung kann durch vier Teilschritte beschrieben werden: Es werden k Bögen aus dem Pfad gelöscht, so daß $k + 1$ Segmente entstehen. Einige der Segmente werden invertiert, und die Segmente werden untereinander vertauscht. Schließlich werden die Segmente (in ihrer neuen Reihenfolge) wieder durch k neue Bögen miteinander verbunden (siehe Abb. 3.2).

Die beschriebene Konstruktion eines k -opt-Schritts kann formal als die Nacheinanderausführung der vier Operatoren

- (1) k -Segmentierung,

Abbildung 3.2: Vier Teilschritte eines k -opt-Schritts

- (2) k -Invertierung,
- (3) k -Permutation und
- (4) Verkettung

aufgefaßt werden. Diese vier Operatoren (bzw. Klassen von Operatoren) werden zunächst einzeln definiert, bevor die Definition des k -opt-Schritts angegeben werden kann.

Bis auf die Definition des Verkettungsoperators haben alle folgenden Definitionen die gleiche Struktur und umfassen jeweils

- die Darstellung eines Operators,
- die Menge aller Operatoren,
- die Beschreibung des Abbilds eines Elements unter dem Operator und
- die Menge aller Bilder.

(1) k -Segmentierung von Pfaden Eine k -Segmentierung ist eine Abbildung, die eine Bogenfolge b der Länge m ($m \geq k$) in ein Tupel von $k + 1$ Teilfolgen bzw. Segmenten überführt. Diese Segmente entstehen dadurch, daß insgesamt k Bögen aus b entfernt werden.

Unabhängig von der konkreten Gestalt von $b \in B^m$ existieren jeweils $\binom{m}{k}$ verschiedene k -Segmentierungen von b .⁴ Diese können durch die Angabe der Längen der einzelnen Segmente σ_i eindeutig spezifiziert werden.

Definition 3.21. (k -Segmentierung: $\mathbf{s}_{(\ell_0, \dots, \ell_k)}^{k,m}$, $\mathbf{S}^{k,m}$)

Sei $\ell = (\ell_0, \dots, \ell_k) \in \mathbb{N}^{k+1}$ mit $\sum_{i=0}^k \ell_i = m - k$. Dann heißt die Abbildung

$$\mathbf{s}_{(\ell_0, \dots, \ell_k)}^{k,m} : B^m \rightarrow B^{\ell_0} \times \dots \times B^{\ell_k}$$

ein Segmentierungsoperator der Stufe k , falls für jede beliebige Bogenfolge $b \in B^m$ mit $b = \sigma_0 \oplus \sigma_1 \oplus \dots \oplus \sigma_k$ und $\ell_i = |\sigma_i|$, $i = 0, \dots, k$

$$\mathbf{s}_{(\ell_0, \dots, \ell_k)}^{k,m}(b) = (\sigma_0, \dots, \sigma_k) \quad \text{gilt.}$$

Die Menge aller Segmentierungsoperatoren der Stufe k auf einer Bogenfolge der Länge m sei bezeichnet mit $\mathbf{S}^{k,m}$. Eine beliebige k -Segmentierung der Bogenfolge b wird geschrieben als $\mathbf{s}^k(b)$. $\mathbf{S}^k(b)$ ist die Menge aller k -Segmentierungen der Bogenfolge b .

Bemerkung 3.22. Bei der Bezeichnung $\mathbf{s}_{(\ell_0, \dots, \ell_k)}^{k,m}$ kann auf die Angabe von k und m prinzipiell verzichtet werden, denn der Vektor $\ell = (\ell_0, \dots, \ell_k)$ kodiert die Längen der zu bildenden Segmente und besitzt deshalb die Länge $k + 1$. Außerdem ergibt sich m laut Definition zu:

$$m = k + \sum_{i=0}^k \ell_i.$$

Da es für allgemeine Betrachtungen oft unerheblich ist, welche Länge die einzelnen Segmente tatsächlich besitzen, wird es meistens genügen, lediglich k und m bzw. nur k anzugeben. Dann kann die verkürzte Schreibweise $\mathbf{s}^{k,m}$ bzw. $\mathbf{s}^k$ benutzt werden, die für einen beliebigen Segmentierungsoperator der Stufe k steht.

(2) k -Invertierung Um einen bestimmten k -opt-Schritt erzeugen zu können, müssen gewisse der $k + 1$ Segmente invertiert werden. Dazu dient der Invertierungsoperator $\mathbf{i}^k$, der mehrere Segmente gleichzeitig invertieren kann.

⁴ Dies ist Anzahl der Möglichkeiten, genau k der insgesamt m Bögen aus der Bogenfolge b zu eliminieren. Dabei zerfällt die Bogenfolge in $k + 1$ Segmente.

Definition 3.23. (Inverse Bogenfolge)

Sei $b = (v_0, \dots, v_m) \in B^m$ eine Bogenfolge der Länge m und für alle $i \in \{1, \dots, m\}$ seien die Bögen (v_i, v_{i-1}) in A enthalten. Dann heißt

$$b^{-1} = (v_m, v_{m-1}, \dots, v_0) \in B^m$$

die zu b inverse Bogenfolge.

Bemerkung 3.24. Sei $p \in P^m$ ein Pfad der Länge m , dessen inverse Bogenfolge p^{-1} existiert. Dann ist p^{-1} ebenfalls ein Pfad der Länge m .

Definition 3.25. (k -Invertierung: $\mathbf{i}_{(\delta_0, \dots, \delta_k)}^k, \mathbf{I}^k$)

Es sei $\delta = (\delta_0, \dots, \delta_k) \in \{-1, +1\}^{k+1}$ ein $k+1$ -Tupel mit $\delta_0 = \delta_k = +1$. Dann heißt die Abbildung $\mathbf{i}_{(\delta_0, \dots, \delta_k)}^k : \mathbf{S}^k \rightarrow \mathbf{S}^k$ ein Invertierungsoperator der Stufe k , falls für jede k -Segmentierung $\mathbf{s}^k = (\sigma_0, \dots, \sigma_k) \in \mathbf{S}^k$ gilt:

$$\mathbf{i}_{(\delta_0, \dots, \delta_k)}^k(\mathbf{s}^k) = (\sigma_0^{\delta_0}, \dots, \sigma_k^{\delta_k}).$$

Die Menge aller Invertierungsoperatoren einer k -Segmentierung sei bezeichnet mit $\mathbf{I}^k$. Eine (beliebige) k -Invertierung von $\mathbf{s}^k$ wird geschrieben als $\mathbf{i}^k(\mathbf{s}^k)$. $\mathbf{I}^k(\mathbf{s}^k)$ ist die Menge aller k -Invertierungen von $\mathbf{s}^k$.

Wie man anhand der Definition 3.25 erkennt, können die Teilpfade σ_0 und σ_k nicht invertiert werden. Dies erklärt sich wie folgt: Bei einer Invertierung der Segmente σ_0 und σ_k würden die Depotknoten o und d nicht mehr am Touranfang bzw. am Tourende auftauchen. Damit wäre die Zulässigkeit der erzeugten Lösung nicht mehr garantiert.⁵

(3) k -Permutation Um einen bestimmten k -opt-Schritt zu erhalten, muß ggf. auch die Reihenfolge der Segmente verändert werden. Dies wird mit dem Permutationsoperator realisiert. Allerdings muß der Permutationsoperator nicht alle Permutationen zulassen, um die komplette k -opt-Nachbarschaft erzeugen zu können. Damit dieselben Nachbarlösungen möglichst nicht mehrfach erhalten werden, muß stattdessen die Position eines Segments festgehalten werden. Wir fixieren deshalb die Segmente σ_0 und σ_k , die stets wie ein gemeinsames Segment behandelt werden müssen.⁶

⁵Prinzipiell wäre es möglich, eine gleichzeitige Invertierung, verbunden mit einem Austausch der Segmente σ_0 und σ_k , zuzulassen. Die so gebildeten k -opt-Schritte erzeugen Touren, die aus den übrigen k -opt-Schritten durch eine komplette Umkehrung der Gesamttour hervorgehen.

Definition 3.26. (k -Permutation: $\mathbf{p}_{(\pi_0, \dots, \pi_k)}^k, \mathbf{P}^k$)

Sei $(\pi_0, \dots, \pi_k)$ eine Permutation von $(0, \dots, k)$ mit $\pi_0 = 0, \pi_k = k$. Dann heißt die Abbildung $\mathbf{p}_{(\pi_0, \dots, \pi_k)}^k : \mathbf{S}^k \rightarrow \mathbf{S}^k$ ein Permutationsoperator der Stufe k , falls für jede k -Segmentierung $\mathbf{s}^k = (\sigma_0, \dots, \sigma_k) \in \mathbf{S}^k$ gilt:

$$\mathbf{p}_{(\pi_0, \dots, \pi_k)}^k(\mathbf{s}^k) = (\sigma_{\pi_0}, \dots, \sigma_{\pi_k}).$$

Die Menge aller Permutationsoperatoren einer k -Segmentierung sei bezeichnet mit $\mathbf{P}^k$. Eine (beliebige) k -Permutation von $\mathbf{s}^k$ wird geschrieben als $\mathbf{p}^k(\mathbf{s}^k)$. $\mathbf{P}^k(\mathbf{s}^k)$ ist die Menge aller k -Permutationen von $\mathbf{s}^k$.

(4) Verkettung Die Verkettung $\oplus$ ist eine Abbildung $\oplus : \mathbf{S}^k \rightarrow P$, die es erlaubt, die Teilpfade einer k -Segmentierung wieder zu einer Bogenfolge zu verbinden:

Definition 3.27. (Verkettung: $\oplus$)

Sei $b \in B^m$ und $\mathbf{s}^k(b) = (\sigma_0, \dots, \sigma_k) \in \mathbf{S}^k(b)$. Dann ist

$$\oplus(\mathbf{s}^k(b)) = \bigoplus_{i=0}^k \sigma_i = b.$$

k -opt-Schritt und k -opt-Nachbarschaft Die Nacheinanderausführung der Abbildungen k -Segmentierung, k -Invertierung, k -Permutation und Verkettung bildet eine Bogenfolge b auf eine ihrer Nachbarlösungen der k -opt-Nachbarschaft ab. Diese Abbildung wird im weiteren als k -opt-Schritt $\mathbf{m}^k$ bezeichnet.

Definition 3.28. (k -opt-Schritt: $\mathbf{m}^{k,m}, \mathbf{M}^{k,m}$)

Sei $b \in B^m$,

$$\begin{aligned} \ell &= (\ell_0, \dots, \ell_k) \text{ mit } \sum_{i=0}^k \ell_i + k = m, \\ \delta &= (\delta_0, \dots, \delta_k) \text{ mit } \delta_i \in -1, +1 \text{ für alle } 1 \leq i \leq k-1, \delta_0 = \delta_k = +1, \\ \pi &= (\pi_0, \dots, \pi_k) \text{ eine Permutation von } (0, \dots, k), \\ \mathbf{s}_\ell^{k,m} &\in \mathbf{S}^k, \mathbf{i}_\delta^k \in \mathbf{I}^k, \mathbf{p}_\pi^k \in \mathbf{P}^k. \end{aligned}$$

⁶Dies wird unmittelbar deutlich, wenn man zum Kreismodell 3.18 zurückkehrt. Dabei werden die Segmente σ_0 und σ_k am Depotknoten zu einem gemeinsamen Segment verbunden.

Dann ist der Operator $\mathbf{m}_{\ell,\delta,\pi}^{k,m} : B \rightarrow B$ die Nacheinanderausführung von $\mathbf{s}_{\ell}^{k,m}$, $\mathbf{i}_{\delta}^k$, $\mathbf{p}_{\pi}^k$ und $\oplus$:

$$\mathbf{m}_{\ell,\delta,\pi}^{k,m} = \oplus \circ \mathbf{p}_{\pi}^k \circ \mathbf{i}_{\delta}^k \circ \mathbf{s}_{\ell}^{k,m}.$$

Der Operator $\mathbf{m}_{\ell,\delta,\pi}^{k,m}$ heißt k -opt-Schritt.

Die Menge aller k -opt-Schritte einer Bogenfolge der Länge m sei bezeichnet mit $\mathbf{M}^{k,m}$. Ein (beliebiger) k -opt-Nachbar der Bogenfolge $b \in B^m$ wird geschrieben als $\mathbf{m}^k(b)$:

$$\mathbf{m}^k(b) = \oplus(\mathbf{p}^k(\mathbf{i}^k(\mathbf{s}^k(b)))).$$

$\mathbf{M}^k(b)$ ist die Menge aller k -opt-Nachbarn von b , d.h. die zulässigen Lösungen, die in $\mathbf{M}^k(b)$ enthalten sind, bilden die k -opt-Nachbarschaft von b .

Definition 3.29. (k -opt-Nachbarschaft, k -Optimalität)

Es sei $\mathcal{P}$ ein VRSP mit der Lösungsmenge X und den zulässigen Lösungen $Y \subseteq X$ (vgl. Abschnitt 2.1). $G = (V, A, c)$ sei der zu $\mathcal{P}$ gehörende gerichtete Graph mit $X = B^m$. Dann ist die k -opt-Nachbarschaft $\mathcal{N}_{k-opt}(b)$ einer Lösung $b \in B^m$ gegeben durch

$$\mathcal{N}_{k-opt}(b) = \mathbf{M}^k(b) \cap Y$$

und b heißt k -optimal, wenn gilt:

$$c(b) \leq c(b') \quad \text{für alle } b' \in \mathcal{N}_{k-opt}(b).$$

Oft werden Nachbarschaften betrachtet, die nur bestimmte Klassen von k -opt-Schritten zulassen. Diese können dadurch beschrieben werden, daß die Teil-Operatoren aus denen die k -opt-Schritte aufgebaut sind, eingeschränkt oder ganz fixiert werden. So könnte z.B. die Klasse der „nicht-invertierenden“ k -opt-Schritte betrachtet werden. Diese umfaßt alle Schritte, die die k -Invertierung $\mathbf{i}_{(1,1,\dots,1)}^k$ besitzen.

Definition 3.30. (Klasse von k -opt-Schritten, k -opt-Typ)

Eine Menge von k -opt-Schritten, deren Teil-Operatoren (k -Segmentierung, k -Invertierung oder k -Permutation) gleiche Eigenschaften haben, wird als *Klasse von k -opt-Schritten* bezeichnet.

Eine Klasse von k -opt-Schritten, deren k -Invertierungen und k -Permutationen identisch sind, heißt *k -opt-Typ*.

Beispiel 3.1. Die 3-opt-Nachbarschaft umfaßt die acht k -opt-Typen:

$$\begin{array}{ll}
 \mathbf{m}_{*,(1,1,1,1),(0,1,2,3)}^3 & \mathbf{m}_{*,(1,-1,1,1),(0,1,2,3)}^3 \\
 \mathbf{m}_{*,(1,1,-1,1),(0,1,2,3)}^3 & \mathbf{m}_{*,(1,-1,-1,1),(0,1,2,3)}^3 \diamond \\
 \mathbf{m}_{*,(1,1,1,1),(0,2,1,3)}^3 \diamond & \mathbf{m}_{*,(1,-1,1,1),(0,2,1,3)}^3 \diamond \\
 \mathbf{m}_{*,(1,1,-1,1),(0,2,1,3)}^3 \diamond & \mathbf{m}_{*,(1,-1,-1,1),(0,2,1,3)}^3
 \end{array}$$

Der ‘*’ steht hier für eine beliebige Segmentierung. Nur die mit „ $\diamond$ “ gekennzeichneten Schritte sind echte 3-opt-Schritte.⁷

Die obige Darstellung der k -opt-Typen ergibt sich aus der Schreibweise für einzelne k -opt-Schritte. Wie das Beispiel 3.1 zeigt, ist diese Notierung aber recht schwerfällig. Deshalb führen wir die „abc-Notation“ ein, die eine kompaktere Darstellung der k -opt-Typen erlaubt. Bei der „abc-Notation“ werden die aufeinanderfolgenden Segmente der Ausgangslösung mit aufsteigenden Kleinbuchstaben $a, b, c, \dots$ bezeichnet. Da man sich das letzte Segment mit dem ersten Segment (über das Depot) verbunden vorstellen muß, erhalten beide den gleichen Namen, d.h. das letzte Segment wird ebenfalls als a bezeichnet.⁸ Die Ausgangslösung kann dann als Folge der einzelnen Segmente dargestellt werden: $[abc \dots a]$.

Ein k -opt-Typ wird durch seine k -Invertierung und k -Permutation vollständig beschrieben. Die Invertierung eines Segments wird durch die Verwendung des zugehörigen Großbuchstabens dargestellt. Die Permutation wird durch die neue Reihenfolge der Segmente angegeben.

Beispiel 3.2. (Fortsetzung von Beispiel 3.1)

Die acht k -opt-Typen der 3-opt-Nachbarschaft können in der abc-Notation wie folgt geschrieben werden:

$$\begin{array}{ll}
 [abca] = \mathbf{m}_{*,(1,1,1,1),(0,1,2,3)}^3 & [aBca] = \mathbf{m}_{*,(1,-1,1,1),(0,1,2,3)}^3 \\
 [abCa] = \mathbf{m}_{*,(1,1,-1,1),(0,1,2,3)}^3 & [aBCa] = \mathbf{m}_{*,(1,-1,-1,1),(0,1,2,3)}^3 \diamond \\
 [acba] = \mathbf{m}_{*,(1,1,1,1),(0,2,1,3)}^3 \diamond & [aCba] = \mathbf{m}_{*,(1,-1,1,1),(0,2,1,3)}^3 \diamond \\
 [acBa] = \mathbf{m}_{*,(1,1,-1,1),(0,2,1,3)}^3 \diamond & [aCBa] = \mathbf{m}_{*,(1,-1,-1,1),(0,2,1,3)}^3
 \end{array}$$

⁷vgl. Abb. 2.7 auf Seite 44

⁸Wenn diese beiden Segmente unterschieden werden sollen, bezeichnen wir das letzte Segment mit a_1 und das erste Segment mit a_2 , so daß gilt $a = a_1 \oplus a_2$.

3.1.4 Ein Algorithmus für die Lexikographische Suche

Die prinzipielle Vorgehensweise der Lexikographischen Suche wurde bereits im Abschnitt 2.2.2 beschrieben.⁹ Wir verwenden nun die eingeführte Syntax, um einen Algorithmus anzugeben, der eine Lexikographische Suche in der k -opt-Nachbarschaft für ein beliebiges aber festes k durchführt.

Gegeben sei eine Startlösung y_0 eines TSP mit n Knoten. Algorithmisch läßt sich eine Lexikographische Suche für die k -opt-Nachbarschaft mit k ineinander verschachtelten Schleifen wie folgt realisieren:

Algorithmus 3.1. Lexikographische Suche (LEX^k)

1. (*Startlösung*) $y^{akt} := y_0$.
2. (*Initialisierung*) Setze $(\ell_0, \dots, \ell_k) := (0, \dots, 0)$ und $\lambda = 0$.
3. (*Schleife über k Schleifen*) Solange $\lambda \geq 0$ wiederhole:
 - 3.1 (ℓ_k berechnen) Setze $\ell_k := n - k - \sum_{i=0}^{k-1} \ell_i$.
 - 3.2 (*Rückkehr zur vorherigen Schleife*)
Wenn $\ell_k < 1$, dann
setze $\ell_\lambda := 0$ und gehe zu 3.5.
 - 3.3 (*Übergang zur nächsten Schleife*)
Wenn $\lambda < k$, dann
setze $\lambda := \lambda + 1$ und gehe zu 3.
 - 3.4 (*Nachbarschaftsschritte durchführen*)
 - 3.4.1 Erzeuge die k -Segmentierung $s := s_{(\ell_0, \dots, \ell_k)}^k(y^{akt})$.
 - 3.4.2 (*alle k -opt-Typen testen*)
Erzeuge alle Lösungen der Form $y' = \oplus(\mathbf{p}^k(\mathbf{i}^k(s)))$ und
wenn y' zulässig ist und $c(y') < c(y^{akt})$, dann
 - 3.4.2.1 setze $y^{akt} := y'$,
 - 3.4.2.2 gehe zu 2.
 - 3.5 Setze $\lambda := \lambda - 1$.
 - 3.6 Setze $\ell_\lambda := \ell_\lambda + 1$.
4. Stopp.

Der Schleifenzähler λ erlaubt es, zwischen den k Schleifen zu wechseln. Jede Schleife i legt die Länge ℓ_i des i -ten Segments fest. Die Länge ℓ_k des letzten Segments ergibt sich zwingend und wird im Schritt 3.1 berechnet. Ist diese Länge kleiner als eins, kann keine entsprechende Segmentierung mehr

⁹siehe Seite 42f.

vorgenommen werden. Dann wird die Länge ℓ_λ erneut mit null initialisiert und der Schleifenzähler λ um eins zurückgesetzt. So werden mit Hilfe der k Schleifen systematisch alle $\binom{n}{k}$ möglichen k -Segmentierungen erzeugt.

Immer wenn λ den Wert k erreicht hat, liegt ein neuer Vektor $(\ell_0, \dots, \ell_k)$ und somit eine andere k -Segmentierung $\mathbf{s} = \mathbf{s}_{(\ell_0, \dots, \ell_k)}^k(y^{akt})$ vor. Dann werden im Schritt 3.4 alle k -opt-Typen gebildet. Die Nachbarlösungen $y' \in \mathcal{N}_k(y^{akt})$ erhält man durch das Verketteten der invertierten und permutierten Segmente. Eine Überprüfung der Zulässigkeit der Lösung ist beim TSP nicht notwendig.

Sobald eine verbessernde Lösung y' gefunden wurde, erfolgt sofort ein Rücksprung zu Schritt 2. Dadurch wird eine Tiefensuche-Strategie realisiert. Eine Abänderung des Algorithmus zur Bestensuche ist leicht möglich.

3.1.5 Effizienzbetrachtungen für die Lexikographische Suche

Mit dem Algorithmus 3.1 verfügen wir über ein einfaches Vorgehen, mit dem die Lexikographische Suche auf einer k -opt-Nachbarschaft ausgeführt werden kann. Doch dieser Algorithmus ist nicht sehr effizient. Die folgenden Modifikationen können teilweise zur erheblichen Beschleunigung des Algorithmus beitragen:

1. Die k -Invertierungen und k -Permutationen sind im wesentlichen unabhängig von der Länge und konkreten Beschaffenheit der einzelnen Segmente $\sigma_0, \dots, \sigma_k$. Deshalb kann vorab einmalig bestimmt werden, welche Verknüpfungen von k -Invertierungen $\mathbf{i}_\delta^k$ und k -Permutationen $\mathbf{p}_\pi^k$, d.h. welche k -opt-Typen bei jeder Ausführung von Schritt 3(4).2 des Algorithmus 3.1 berücksichtigt werden müssen. Dazu genügt es, vor dem Schritt 2 einmalig alle relevanten Kombinationen in einem Vektor $((\delta_1, \pi_1), \dots, (\delta_w, \pi_w))$ zu speichern. Bei jedem Durchlaufen von Schritt 3(4).2 wird nur noch dieser Vektor abgearbeitet.
2. Falls eine Verbesserung gefunden wurde, wird im Schritt 3(4).2 ein Rücksprung zu Schritt 2 veranlaßt. Damit werden die Längen aller Segmente wieder auf null gesetzt, und die Suche beginnt von vorne.

Dies ist einerseits inhaltlich richtig, da durch den Übergang zur Nachbarlösung y' eine neue Startlösung vorliegt und deshalb die gesamte Nachbarschaft dieser Lösung abgesucht werden muß. Andererseits ist dieses Vorgehen aber nicht effizient.

Da sich die neue Lösung oft nur wenig von der vorherigen Lösung unterscheidet, ist es recht unwahrscheinlich, daß Nachbarschaftsschritte, die zuvor keine Verbesserung bewirken konnten, nun verbessernd sind. Die Suche von vorn zu beginnen, bedeutet aber, daß genau diejenigen Nachbarschaftsschritte zuerst untersucht werden, die zuvor nicht erfolgreich waren.

Daher ist es meistens wesentlich günstiger, die Suche ohne Reinitialisierung der Segmentlängen, d.h. mit Nachbarschaftsschritten fortzuführen, die (zuletzt) noch nicht untersucht wurden. Der Abbruch der Suche muß dann aber so angepaßt werden, daß trotzdem die gesamte Nachbarschaft der neuen Lösung y' komplett abgesucht wird, sofern nicht vorher ein weiterer verbessernder Nachbarschaftsschritt gefunden wurde.

Dies kann z.B. erreicht werden, indem die Suche zum Schluß, d.h. wenn im ersten (unvollständigen) Durchgang keine Verbesserung gefunden wurde, noch einmal von vorn gestartet wird. Erst wenn auch dabei keine Verbesserung gefunden wird, bricht die Suche endgültig ab.

3. Für die Untersuchung der k -opt-Typen im Schritt 3(.4).2 müssen ggf. ein oder mehrere Segmente invertiert werden. Dies würde beim asymmetrischen TSP einen Aufwand der Größe $\mathcal{O}(n)$ verursachen. Die lexikographische Suchreihenfolge bietet aber gerade den Vorteil, daß sich die Länge der Segmente der direkt nacheinander zu untersuchenden Lösungen um eins voneinander unterscheidet (bzw. es erfolgt ein Rücksetzen von Segmenten auf die Länge Null). Daher können die interessierenden Informationen von einem Schritt des Algorithmus zum nächsten weitergereicht werden. Das Anpassen dieser Informationen kann dann mit konstantem Aufwand, d.h. $\mathcal{O}(1)$ realisiert werden.¹⁰

Dies kann auch für den sukzessiven Aufbau der benötigten Informationen zu invertierten Segmenten genutzt werden. Beim asymmetrischen TSP genügt es, die Länge jedes invertierten Segments mitzuführen. Diese Informationen können beim Rücksetzen von Segmenten (Schritte 2 und 3.2) problemlos initialisiert und bei der Erweiterung eines Segments um einen Knoten (Schritt 3.3) mit konstantem Aufwand angepaßt werden.

¹⁰Eine ausführliche Beschreibung erfolgt im Abschnitt 3.2.1 auf Seite 93

4. Wie bereits festgestellt wurde, enthält die k -opt-Nachbarschaft auch alle kleineren ℓ -opt-Nachbarschaften zu $\ell = 2, 3, \dots, k - 1$.¹¹ Werden alle k -opt-Schritte berücksichtigt, sind die Nachbarschaftsschritte der kleineren Nachbarschaften darin jedoch mehrfach enthalten. Daher ist es besser, statt der kompletten k -opt-Nachbarschaft explizit alle echten Nachbarschaften für 2 bis k Kantenaustausche zu untersuchen. So kann das mehrfache Überprüfen der gleichen Nachbarschaftsschritte weitgehend ausgeschlossen werden.¹²
5. Eine weitere Verbesserung der Effizienz des Algorithmus kann dadurch erreicht werden, daß Schleifen vorzeitig abgebrochen werden, sofern man sicher sein kann, daß dadurch keine Verbesserungen übersehen werden.

Wir betrachten deshalb die Kostenänderung

$$\Delta c = c(y') - c(y^{akt}).$$

beim Übergang von y^{akt} zu einer Nachbarlösung $y' \in \mathcal{N}(y^{akt})$:

Unter der Voraussetzung, daß sich die Kosten einer Lösung als Summe der Kosten der in der Lösung enthaltenen Bögen darstellen läßt, kann eine Kostenänderung beim asymmetrischen TSP nur durch

- (a) die Differenz der Kosten von gelöschten und eingefügten Bögen (Δc_1) und
- (b) die Invertierung eines oder mehrerer Segmente (Δc_2)

verursacht werden. Eine Verbesserung der Lösung y^{akt} liegt nur dann vor, wenn $\Delta c = \Delta c_1 + \Delta c_2$ negativ ist.

Wenn es also gelingt, für alle y' aus einer Teilmenge von $M \subseteq \mathcal{N}(y^{akt})$ untere Schranken für Δc_1 und für Δc_2 anzugeben, deren Summe nicht negativ ist, so braucht M nicht mehr abgesucht zu werden.

¹¹siehe Seite 41

¹²Der Wechsel zwischen den verschiedenen echten k -opt-Nachbarschaften kann z.B. im Sinne einer Variable Neighborhood Descent Heuristik (vgl. Abschnitt 2.3) erfolgen. Die Reihenfolge, in der die Nachbarschaften abgesucht werden, kann auf unterschiedliche Weise festgelegt werden. Aus Geschwindigkeitsgründen ist es meist besser, die „kleinen“ Nachbarschaften zuerst abzusuchen.

Δc_1 Die Abschätzung von Δc_1 basiert darauf, daß (bei echten k -opt-Nachbarschaften) genau k Kanten ausgetauscht werden. Die Kostenänderung ist gleich der Differenz der Kosten von eingefügten Bögen c^{ins} und eliminierten Bögen c^{rem} :

$$\Delta c_1 = \sum_{i=1}^k c_i^{ins} - \sum_{i=1}^k c_i^{rem} \quad (3.3)$$

Eine grobe untere Schranke für Δc_1 erhält man, wenn man die Differenz der k in der Lösung y^{akt} enthaltenen Bögen mit den höchsten Kosten und den k in y^{akt} nicht enthaltenen Bögen mit den geringsten Kosten bildet.

Diese Abschätzung kann jedoch in jeder weiteren Schleife verbessert werden, da in der i -ten Schleife des Algorithmus das i -te Segment festgelegt wird. Damit stehen bereits i Bögen fest, die aus der Lösung y^{akt} eliminiert werden müssen. Umgekehrt müssen innerhalb der weiteren Schleifen Bögen eingefügt werden, die mit den Randknoten der bereits fixierten Segmente inzidieren. Daher können i.d.R. auch nicht mehr die k billigsten Bögen des Graphen zum Einsatz kommen, so daß die untere Schranke weiter angehoben werden kann.

Δc_2 Abhängig davon, wie stark die Kostenmatrix des Problems asymmetrisch ist, kann eine Abschätzung von Δc_2 auf unterschiedliche Arten gewonnen werden:

Im einfachsten Fall ist die Kostenmatrix symmetrisch und Δc_2 ist gleich null.

Für Matrizen die nur „wenig asymmetrisch“ sind, kann der maximale Gewinn, der mit der Invertierung eines oder mehrerer Segmente erzielt werden kann, zu Beginn der Suche einmalig (relativ grob) abgeschätzt werden. Dazu wird die maximale Differenz der Kosten von eingehenden Bögen und ausgehenden Bögen für alle Knoten $v_i \in V$ bestimmt und summiert:

$$\Delta c_2 \geq \sum_{i \in V} \left(\min_{j \in V, (i,j) \in A, (j,i) \in A} [c_{ji} - c_{ij}, 0] \right). \quad (3.4)$$

Diese Abschätzung ist allerdings auch dann relativ grob, wenn nur einzelne stark asymmetrische Bögen in der Kostenmatrix enthalten sind.

Wesentlich bessere Abschätzungen können erhalten werden, wenn man berücksichtigt, welche Bögen die aktuelle Lösung y^{akt} tatsächlich besitzt. Durch das Umkehren von Segmenten können nur diese Bögen invertiert werden.

$$\Delta c_2 \geq \sum_{i \in V} \left(\min_{j \in V, (i,j) \in A(y^{akt}), (j,i) \in A} [c_{ji} - c_{ij}, 0] \right). \quad (3.5)$$

Diese untere Schranke muß bei jedem Wechsel zu einer neuen Nachbarlösung (Schritt 3.(4).2.1) aktualisiert werden. Der Aufwand für die Bestimmung der Schranke beträgt zu Beginn des Algorithmus einmalig $\mathcal{O}(n^2)$. Die Aktualisierung nach dem Übergang zu einer Nachbarlösung kann mit Aufwand $\mathcal{O}(k * n)$ durchgeführt werden, weil sich in jedem Schritt nur k Bögen ändern.

Es sei bereits an dieser Stelle darauf hingewiesen, daß sich die Kostenabschätzungen auch auf Probleme mit zusätzlichen Nebenbedingungen übertragen, da die Menge der zulässigen Lösungen dann nur noch stärker eingeschränkt wird.

Eine wesentliche Verbesserung der Abschätzungen für Δc_1 und Δc_2 läßt sich erzielen, wenn man die Lexikographische Suche in der k -opt-Nachbarschaft nicht nur nach (echten) k -opt-Nachbarschaften, sondern darüber hinaus nach k -opt-Typen aufteilt und nacheinander für die einzelnen k -opt-Typen durchführt. Dann stehen in der i -ten Schleife nicht nur i der zu eliminierenden Bögen fest. Auch ein Teil der Bögen, die später eingefügt werden müssen, ist bereits bekannt und können bei der Bestimmung von Δc_1 einkalkuliert werden. Bei der Abschätzung von Δc_2 können die bereits gebildeten Segmente berücksichtigt werden, weil bekannt ist, ob diese invertiert werden oder nicht.

Die so verbesserten Abschätzungen können helfen, den Gesamtaufwand für die Durchführung der Lexikographischen Suche zu reduzieren. Diesen Vorteilen steht allerdings ein erhöhter Aufwand für das mehrfache Durchführen der Lexikographischen Suche für jeden zu betrachtenden k -opt-Typ gegenüber. Weitere Einsparungen sind bei Problemen mit zusätzlichen Restriktionen möglich, weil evtl. eher festgestellt werden kann, wenn es mit den bereits feststehenden Segmenten nicht mehr möglich ist, eine zulässige Lösung zu bilden oder die Kostenabschätzung schärfer gemacht werden kann.

Die Berechnung der Abschätzungen für Δc ist relativ aufwendig. Darüber hinaus können Abbrüche oft nur in den inneren Schleifen des Algorithmus generiert werden, wo sie nur eine geringe Ersparnis an Rechenzeit bewirken können. Ob es insgesamt sinnvoll ist, Abbrüche mit Hilfe von Kostenabschätzungen durchzuführen, hängt vom gegebenen Problem ab. Eine Analyse anhand von Probleminstanzen des TSPTW und VRP wird im Abschnitt 6.3 vorgenommen.

Über die genannten Modifikationen der Lexikographischen Suche hinaus ist es manchmal sinnvoll, nur einen Teil der k -opt-Nachbarschaft abzusuchen. Dazu ist es ggf. erwünscht, nur einen Teil der existierenden k -opt-Typen zu untersuchen, wie es z.B. bei der Or -opt-Nachbarschaft der Fall ist.¹³

Dies kann innerhalb des Algorithmus 3.1 einfach realisiert werden, wenn die zu untersuchenden k -opt-Typen, wie bereits unter Punkt 1 beschrieben, vorab in einem Vektor abgelegt werden. Die Durchführung der Lexikographischen Suche beschränkt sich dann auf die k -opt-Typen, die in diesem Vektor spezifiziert wurden. Es genügt daher, nur die erwünschten Schritte in den Vektor aufzunehmen.

3.2 Probleme mit beschränkten Ressourcen

Wir betrachten nun Probleme, bei denen beschränkte Ressourcen zu berücksichtigen sind, und interessieren uns dafür, wie eine Lokale Suche effizient durchgeführt werden kann.

Angenommen, es liegt eine zulässige Lösung $y^{akt} \in Y$ vor und im Rahmen einer Lokale Suche sollen alle Nachbarlösungen $y' \in \mathcal{N}(y^{akt})$ untersucht werden. Dann müssen hinsichtlich jeder Nachbarlösung *zwei* Fragen beantwortet werden:

1. Ist die Nachbarlösung zulässig?
2. Ist die Nachbarlösung besser als die aktuelle Lösung?

Entscheidend für die Effizienz der Lokalen Suche ist u.a., ob die Beantwortung dieser Fragen stets mit geringem zusätzlichem Aufwand möglich ist.¹⁴

¹³siehe dazu Abschnitt 3.4 auf Seite 119

¹⁴Würde z.B. für jede Nachbarlösung an jedem der n einzelnen Knoten die Einhaltung der Ressourcenbeschränkungen explizit geprüft, so entstünde für die Zulässigkeitsprüfung einer Nachbarlösung ein Aufwand von mindestens $\mathcal{O}(n)$. Die Komplexität der Lokalen Suche in der k -opt-Nachbarschaft würde dann von $\mathcal{O}(n^k)$ auf $\mathcal{O}(n \cdot n^k)$ steigen.

Beim TSP (ohne Ressourcenbeschränkungen) ist dies sehr einfach. Die Zulässigkeit der Nachbarlösung kann bereits durch die Wahl geeigneter Nachbarschaftsschritte gewährleistet werden. Eine Überprüfung braucht dann überhaupt nicht stattzufinden. Ob der Nachbarschaftsschritt eine Verbesserung bewirkt, läßt sich beim symmetrischen TSP feststellen, indem die Kosten der gelöschten und hinzugefügten Bögen miteinander verglichen werden.¹⁵

Werden den Lösungen jedoch zusätzliche Beschränkungen, z.B. in Form von Kundenzeitfenstern, Kapazitäten oder Reihenfolgebeschränkungen, auferlegt, so muß die Zulässigkeit jeder betrachteten Nachbarlösung explizit geprüft werden. Auch durch eine kompliziertere Zielfunktion kann das Verfahren drastisch an Effizienz einbüßen, wenn die Berechnung der Zielfunktion aufwendig ist.

Die Überprüfung der Zulässigkeit einer Nachbarlösung im Rahmen einer Sequentiellen Suche verursacht bei einem TSP mit Zeitfenstern i.a. einen Aufwand der Größe $\mathcal{O}(n)$, was für Probleme mit zunehmender Kundenzahl n immer ungünstiger wird. Dagegen hat Savelsbergh 1985 gezeigt [Sav85], daß sich die Zulässigkeitsprüfung bei Problemen mit Zeitfenstern und Reihenfolgebeschränkungen mit konstantem Aufwand durchführen läßt, wenn eine lexikographische Suchreihenfolge eingehalten wird.

Gegenstand dieses Abschnitts ist eine Verallgemeinerung des Vorgehens von Savelsbergh. Dadurch kann es auf alle Probleme angewandt werden, die sich mit dem Modell 1.1 beschreiben lassen. Darüber hinaus können aber auch die in Abschnitt 1.2.3 beschriebenen Erweiterungen berücksichtigt werden. Dies ermöglicht insbesondere eine weitgehende Verallgemeinerung der Zielfunktion. Sofern sich die Zielfunktion ebenfalls abhängig von den benutzten Bögen einer Lösung darstellen läßt, kann sie mit Hilfe einer eigenen Ressource modelliert werden. Die Bestimmung des Zielfunktionswertes der Lösung fällt dann mit der Bestimmung des Wertes dieser Ressource am letzten Knoten der Tour zusammen und verursacht keinen zusätzlichen Aufwand.

Wir betrachten zunächst das TSP mit Zeitfenstern, das nur die beschränkte Ressource „Zeit“ besitzt. Anschließend verallgemeinern wir die Vorgehensweise für endliche Mengen beschränkter Ressourcen.

¹⁵Für das symmetrische TSP ist die Durchführung einer Lexikographischen Suche nicht sinnvoll, da dafür eine Sequentielle Suche bzw. die darauf aufbauende Lin-Kernighan-Suche eingesetzt werden kann, die u.a. durch die Nutzung des Gewinnkriteriums wesentlich effizienter ist.

3.2.1 Berücksichtigung von Zeitfenstern (TSPTW)

Gegeben sei ein TSP mit dem zugehörigen Graphen $G = (V, A, c)$. Jedem Bogen $(i, j) \in A$ ist zusätzlich zu den Kosten c_{ij} eine Zeit d_{ij} (Fahrtdauer) zugeordnet. Zu jedem Knoten $i \in V$ gibt es ein Zeitfenster $[a_i, b_i]$ und eine Servicezeit s_i . Bei einer zulässigen Lösung muß die Ankunftszeit am Knoten i innerhalb des Zeitfensters $[a_i, b_i]$ liegen. Es ist jedoch jederzeit möglich, eine beliebig lange Wartezeit vor dem Knoten i einzuschieben, um eine zu frühe Ankunft zu vermeiden.

Das TSPTW ist, wie in Abschnitt 1.2.2 beschrieben, ein Spezialfall von Modell 1.1 auf Seite 13 und nimmt (etwas vereinfacht formuliert) die folgende Gestalt an:

$$\min \sum_{(i,j) \in A} c_{ij} x_{ij} \quad (3.6a)$$

so daß

$$\sum_{j: (i,j) \in A} x_{ij} = 1, \quad \forall i \in V \quad (3.6b)$$

$$\sum_{i: (i,j) \in A} x_{ij} = 1, \quad \forall j \in V \quad (3.6c)$$

$$x_{ij}(T_i + s_i + d_{ij} - T_j) \leq 0, \quad \forall (i, j) \in A \quad (3.6d)$$

$$a_i \leq T_i \leq b_i, \quad \forall i \in V \quad (3.6e)$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A \quad (3.6f)$$

Die Entscheidungsvariablen x_{ij} nehmen den Wert eins an, wenn der Bogen (i, j) in der Lösung enthalten ist. Die Variable T_i gibt die Ankunftszeit am Knoten $i \in V$ an. Da beim TSP(TW) nur eine Tour vorliegt, können die Restriktionen (1.1b)–(1.1d) von Modell 1.1 durch die einfacheren Formulierungen (3.6b) und (3.6c) äquivalent ersetzt werden. Die Tatsache, daß vor der Ankunft an einem Knoten beliebig lange gewartet werden darf, wird durch die $\leq$ -Relation in (3.6d) ausgedrückt. Damit das Problem (3.6) lösbar ist, muß i.a. mindestens ein Bogen existieren (vom Depoteingang zum Depotausgang), der eine negative Dauer aufweist.

Eine der Servicezeit s_i entsprechende Größe, tritt im Modell 1.1 nicht auf. Die explizite Berücksichtigung der Servicezeit am Knoten i ist aber auch nicht notwendig, wenn diese Zeit auf die Fahrzeiten d_{ij} aller Bögen aufgeschlagen

wird, die den Knoten i verlassen. Unter Verwendung derart modifizierter Zeiten $d_{ij}' = d_{ij} + s_i$ könnte (3.6d) auch durch

$$x_{ij}(T_i + d_{ij}' - T_j) \leq 0, \quad \forall (i, j) \in A$$

ersetzt werden. Im Gegensatz zu den meisten Veröffentlichungen wird hier aber auf diese Vereinfachung verzichtet. Dies geschieht vor allem deshalb, weil das Modell damit allgemein genug bleibt, um den Unterschied zwischen längeren Segmenten und einzelnen Knoten einer Lösung verschwimmen zu lassen. Das heißt, daß man sich anstelle eines Knotens in diesem Modell immer auch eine fixe Folge von Knoten (also ein Segment) vorstellen kann. Die Servicezeit eines Segments wird dann mit der Zeit gleichgesetzt, die zwischen der Ankunft am ersten Knoten des Segments und der Abfahrt am letzten Knoten des Segments vergeht. Auf diesen Aspekt wird in Unterabschnitt 5.2.3 näher eingegangen.

Zulässigkeitstests in konstanter Zeit Zeitliche Zulässigkeit eines Segments σ heißt, daß es eine Belegung für die Abfahrtszeiten T_i aller Knoten $i \in V(\sigma)$ gibt, so daß die Restriktionen (3.6d) und (3.6e) eingehalten werden.

In [KS92] bzw. [KS97] beschreiben Kindervater und Savelsbergh am Beispiel der 2-opt-Nachbarschaft und der Or -opt-Nachbarschaft, wie ein Test auf Zulässigkeit unter Verwendung von regelmäßig zu aktualisierenden „globalen Variablen“ mit konstantem Zeitaufwand vorgenommen werden kann. Die Idee beruht im wesentlichen darauf, daß sich die $k + 1$ Segmente $(\sigma_0, \dots, \sigma_k)$, in die die aktuelle Lösung y^{akt} bei einem k -opt-Schritt zerlegt wird, bei den aufeinanderfolgenden Iterationen der Lexikographischen Suche i.d.R. nur durch die Verschiebung eines Knotens voneinander unterscheiden. Deshalb können aufwendig zu bestimmende Informationen, die die einzelnen Segmente betreffen, von Iteration zu Iteration weitergegeben und entsprechend angepaßt werden (siehe Abb. 3.3 auf Seite 96).

Für die schnelle Überprüfung der Zeitfenster bei der Lexikographischen Suche genügt es, für jedes Segment σ drei Informationen zu kennen:

- D_σ – die Gesamtfahrzeit einschließlich der Servicezeiten (Wartezeiten zählen nicht dazu),
- LAT_σ – die späteste Ankunftszeit am ersten Knoten des Segments und
- EDT_σ – die früheste Abfahrtszeit am letzten Knoten des Segments.

Zulässigkeit von Nachbarlösungen Angenommen, für zwei Segmente σ_1 und σ_2 sind die Werte LAT_{σ_1} , EDT_{σ_1} , LAT_{σ_2} und EDT_{σ_2} bekannt, dann kann in konstanter Zeit überprüft werden, ob die Verkettung $\sigma = \sigma_1 \oplus \sigma_2$ zeitlich zulässig ist. Dies ist genau dann der Fall, wenn σ_1 und σ_2 zulässig sind und

$$EDT_{\sigma_1} + d_{\sigma_1\sigma_2} \leq LAT_{\sigma_2}. \quad (3.7)$$

gilt. Dabei bezeichnet $d_{\sigma_1\sigma_2}$ die Fahrzeit vom letzten Knoten von σ_1 zum ersten Knoten von σ_2 .

Sind LAT , EDT und D für alle Segmente $\sigma_0, \dots, \sigma_k$ bekannt, so kann die Zulässigkeit jeder Nachbarlösung, die sich durch eine Verkettung dieser Segmente darstellen läßt, in konstanter Zeit geprüft werden. Dazu wird die neue Tour Schritt für Schritt durch Verkettung mit je einem weiteren Segment in der gewünschten Reihenfolge zusammengesetzt. In jedem der insgesamt k Schritte müssen die Zulässigkeit der Verkettung geprüft und außerdem die Werte LAT , EDT und D für den neuen „Tour-Rumpf“ berechnet werden.

Bestimmung von D , LAT und EDT Die Werte von D , LAT und EDT können für Segmente, die nur aus einem Knoten v bestehen, bzw. für Segmente, die durch die Verkettung zweier Segmente σ_1 und σ_2 entstehen, mit konstantem Aufwand bestimmt werden.

Es gilt für $\sigma = (v)$, $v \in V$:

$$D_\sigma = s_v \quad (3.8a)$$

$$LAT_\sigma = b_v \quad (3.8b)$$

$$EDT_\sigma = a_v \quad (3.8c)$$

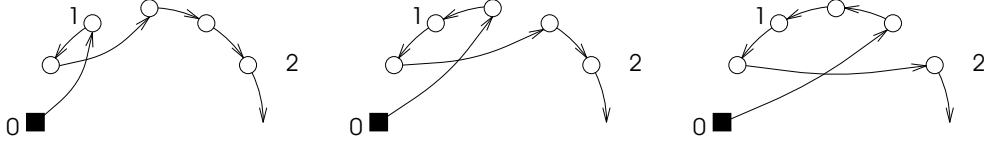
und für $\sigma = \sigma_1 \oplus \sigma_2$:¹⁶

$$D_\sigma = D_{\sigma_1} + d_{\sigma_1\sigma_2} + D_{\sigma_2} \quad (3.8d)$$

$$LAT_\sigma = \min\{LAT_{\sigma_1}, LAT_{\sigma_2} - d_{\sigma_1\sigma_2} - D_{\sigma_1}\} \quad (3.8e)$$

$$EDT_\sigma = \max\{EDT_{\sigma_1} + d_{\sigma_1\sigma_2} + D_{\sigma_2}, EDT_{\sigma_2}\} \quad (3.8f)$$

¹⁶Eine ausführliche Herleitung der Verbindungstheoreme erfolgt im Anhang A.



Bei der 2-opt-Nachbarschaft wird in den aufeinanderfolgenden Iterationen je ein Knoten vom letzten Segment σ_2 zum vorletzten Segment σ_1 verschoben.

Abbildung 3.3: Lexikographische Suche in der 2-opt-Nachbarschaft (die Abb. wurde aus [KS97] übernommen)

Da jedes Segment σ einer Tour als Verkettung seiner Knoten aufgefaßt werden kann, können die Werte von D_σ , LAT_σ und EDT_σ iterativ bestimmt werden, indem sukzessive alle Knoten des Segments aneinandergehängt werden. Falls benötigt, können genauso auch invertierte Segmente Knoten für Knoten aufgebaut werden. Dabei kann die Zulässigkeit der Verkettung bei jedem Schritt überprüft werden, so daß sofort festgestellt wird, wenn das konstruierte Segment unzulässig wird. Der Aufwand für die Berechnung von D_σ , LAT_σ und EDT_σ ist dabei proportional zur Länge des Segments σ .

Bei einer Lexikographischen Suche (Algorithmus 3.1) kann die Bestimmung dieser Informationen aber so in den Ablauf der Suche integriert werden, daß die notwendigen Aktualisierungen in jedem Schritt des Algorithmus nur konstanten Aufwand erfordern.¹⁷ Dazu nutzt man aus, daß sich die im Schritt 3(4).1 von Algorithmus 3.1 konstruierten k -Segmentierungen bei aufeinanderfolgenden Durchläufen der innersten Schleife nur in den beiden letzten Segmenten unterscheiden. In jedem Schritt wird ein Knoten vom Anfang des letzten Segments σ_k abgetrennt. Dieser Knoten wird dann am Ende oder am Anfang des vorletzten Segments angehängt, abhängig davon, ob Segment σ_{k-1} oder das invertierte Segment $(\sigma_{k-1})^{-1}$ aufgebaut werden soll. Ein Beispiel für den Aufbau eines invertierten Segments bei der 2-opt-Nachbarschaft zeigt die Abb. 3.3.

Das Anhängen eines Knotens an das Segment σ_{k-1} kann, wie gerade be-

¹⁷Wird eine verbessernde Lösung gefunden und zu dieser Lösung übergegangen, so ist der Aufwand höher. Dies spielt aber für die hier angestellten Betrachtungen keine Rolle, da dies (insbesondere in der Nachbarschaft einer guten Ausgangslösung) „selten“ passiert.

schrieben, mit konstantem Aufwand durchgeführt werden. Beim Abtrennen des jeweils ersten Knotens von σ_k lassen sich die neuen Werte von D_{σ_k} , LAT_{σ_k} und EDT_{σ_k} dagegen nicht mit konstantem Aufwand aus den bisherigen Werten bestimmen. Stattdessen können aber zu Beginn der Lexikographischen Suche *einmalig* die Werte von D , LAT und EDT für alle „End-Segmente“ der Lösung y^{akt} , d.h. für alle Segmente, die den letzten Knoten der aktuellen Lösung y^{akt} beinhalten, bestimmt werden. Der Aufwand für die Bereitstellung dieser Werte beträgt nur einmalig $\mathcal{O}(n)$ und fällt daher nicht ins Gewicht.

3.2.2 Mengen beschränkter Ressourcen

Das zuvor betrachtete TSPTW ist ein Spezialfall eines TSPR mit genau einer beschränkten Ressource. Bezugnehmend auf das Modell 1.1 untersuchen wir nun Probleme mit einer Menge von Ressourcen R . Die Idee besteht darin, die effiziente Überprüfung der Zeitfenster beim TSPTW auf die Überprüfung anderer Ressourcen zu übertragen.

Ein Zeitfenster legt eine untere und eine obere Schranke für den Wert der Ressource Zeit bei Ankunft am jeweiligen Knoten fest. Alle Ressourcen $r \in R$ unterliegen Restriktionen, die den Zeitfenstern beim TSPTW strukturell gleichen. Obwohl diese Ressourcen und ihre Beschränkungen inhaltlich vollkommen andere Sachverhalte modellieren, können sie formal genauso behandelt werden. Das bedeutet insbesondere, daß die Ressourcenbeschränkungen für jede Ressource mit Hilfe von je drei globalen Variablen pro Segment mit konstantem Aufwand überprüft werden können. Der Test (3.7) auf Zulässigkeit der Verkettung $\sigma = \sigma_1 \oplus \sigma_2$ wird entsprechend auf alle Ressourcen von R erweitert:

$$EDT_{\sigma_1}^r + d_{\sigma_1\sigma_2}^r \leq LAT_{\sigma_2}^r, \quad \forall r \in R. \quad (3.9)$$

Für die Zeitfenster wurde die Festlegung getroffen, daß eine vorzeitige Ankunft des Fahrzeugs am Knoten prinzipiell zulässig ist. Die Einhaltung der unteren Schranke des Zeitfensters wird dadurch hergestellt, daß ggf. bis zum Beginn des Zeitfensters vor dem Knoten gewartet werden muß.¹⁸ Diese Festlegung ist für andere Ressourcen jedoch oft nicht sinnvoll, weil sich der Wert einer Ressource¹⁹ meistens *nicht* durch „Warten“ ändern läßt.

¹⁸Die Möglichkeit zu „warten“, wird im Modell 3.6 durch die $\leq$ -Relation in (3.6e) abgebildet. Diese entspricht der Zeile (1.1e) von Modell 1.1.

¹⁹Mit dem Wert der Ressource r ist hier der tatsächlich aufgetretene Verbrauch der

Aus zwei Gründen stellt dies kein unüberwindliches Problem dar. Erstens genügt es für viele Ressourcen ohnehin, nur eine obere Schranke zu berücksichtigen, weil der Zuwachs des Ressourcenwerts auf allen Bögen nicht negativ ist. Die untere Schranke ist dann eigentlich irrelevant und kann z.B. auf null gesetzt werden (z.B. Fahrzeugauslastung beim CVRP). Zweitens kann bei Bedarf auch eine untere Schranke modelliert werden, die strikt eingehalten wird. Dazu betrachtet man statt einer Ressource r_i , die Ressource $\bar{r}_i$, die das Komplement von r_i modelliert. Die obere Schranke für Ressource $\bar{r}_i$ erfüllt die Funktion einer unteren Schranke von r_i . Müssen sowohl untere also auch obere Schranken strikt eingehalten werden, kann man dies absichern, indem beide Ressourcen (r_i und ihr Komplement $\bar{r}_i$) verwendet werden.

Obwohl das Problem damit prinzipiell gelöst ist, erscheint das beschriebene Vorgehen doch etwas umständlich. Es ist besser, einen zweiten Typ von Ressourcen einzuführen, für die das „Warten“ nicht erlaubt ist. Wir unterteilen den Ressourcenvektor R deshalb in die Ressourcen R^1 , für die das „Warten“ erlaubt ist und Ressourcen R^2 , deren untere Schranken an jedem Knoten strikt einzuhalten sind. Das Modell 1.1 kann entsprechend angepaßt werden, indem die Restriktionen (1.1e) durch

$$x_{ij}(T_i^r + d_{ij}^r - T_j^r) \leq 0, \quad \forall r \in R^1, \quad \forall (i, j) \in A \quad (3.10a)$$

$$x_{ij}(T_i^r + d_{ij}^r - T_j^r) = 0, \quad \forall r \in R^2, \quad \forall (i, j) \in A \quad (3.10b)$$

ersetzt werden.

Wenn für eine Ressource r des neuen Ressourcentyps R^2 geprüft werden soll, ob die Verkettung $\sigma = \sigma_1 \oplus \sigma_2$ zulässig ist, sind die Einhaltung der oberen und unteren Schranken von r zu testen. Zusätzlich zu (3.9) ist für diese Ressourcen deshalb die Bedingung

$$LDT_{\sigma_1}^r + d_{\sigma_1\sigma_2}^r \geq EAT_{\sigma_2}^r \quad (3.11a)$$

einzuhalten. Dabei bestimmt sich die „späteste Abfahrtszeit“ $LDT_{\sigma_1}^r$ und die „früheste Ankunftszeit“ $EAT_{\sigma_1}^r$ bezüglich Ressource r mittels

$$LDT_{\sigma_1}^r = EDT_{\sigma_1}^r + D_{\sigma_1}^r \quad \text{und} \quad (3.11b)$$

$$EAT_{\sigma_2}^r = LAT_{\sigma_2}^r - D_{\sigma_2}^r \quad (3.11c)$$

(vgl. Anhang A).

Ressource (in der Realität) gemeint. Dieser Wert soll im Modell für alle Knoten i mit der Variablen T_i^r repräsentiert werden.

3.2.3 Abhängige Ressourcen

Im Abschnitt 1.2.3 werden voneinander abhängige Ressourcen durch die Einführung der Resource Extension Function (REF) f_{ij}^r modelliert.

Diese Erweiterung kann analog für die Ressourcentypen R^1 und R^2 formuliert werden (vgl. Gleichung (1.2)). Dabei bezeichnet T_i^R den Vektor aller Ressourcenvariablen am Knoten i :

$$x_{ij}(f_{ij}^r(T_i^R, T_j^R) - T_j^r) \leq 0, \quad \forall r \in R^1, \forall (i, j) \in A \quad (3.12a)$$

$$x_{ij}(f_{ij}^r(T_i^R, T_j^R) - T_j^r) = 0, \quad \forall r \in R^2, \forall (i, j) \in A \quad (3.12b)$$

Wir betrachten wieder die Verkettung $\sigma = \sigma_1 \oplus \sigma_2$. Dabei sei i der letzte Knoten von σ_1 und j der erste Knoten von σ_2 . Um die Zulässigkeit der Verkettung $\sigma_1 \oplus \sigma_2$ zu überprüfen, muß festgestellt werden, ob das Restriktionensystem (3.12) für $x_{ij} = 1$ eine Lösung besitzt. Dabei ist zu beachten, daß der zulässige Bereich für die Werte von T_i^R und T_j^R durch die Ressourcenfenster (1.1f) und aufgrund der Zugehörigkeit von i zu σ_1 und von j zu σ_2 bereits eingeschränkt ist.

Für die Durchführung der Lexikographischen Suche ist entscheidend, ob die Überprüfung der Zulässigkeit einer Verkettung $\sigma = \sigma_1 \oplus \sigma_2$ in konstanter Zeit durchgeführt werden kann. Da dies nicht allgemein für jede REF gesichert werden kann, fordern wir folgende zusätzliche Eigenschaften:

- (i) Für jede Ressource $r \in R$ ist die REF f_{ij}^r bezüglich jedes ihrer Argumente $T_i^{r'}$, $T_j^{r'}$, $r' \in R$ stetig und monoton.
- (ii) Die Bestimmung von $f_{ij}^r(T_i^R, T_j^R)$ ist bei gegebenen Vektoren T_i^R und T_j^R für alle $(i, j) \in A$ und $r \in R$ in konstanter Zeit durchführbar.
- (iii) Der Ressourcenvektor $R = (r_1, \dots, r_{|R|})$ sei so geordnet, daß die REF $f_{ij}^{r_\ell}(T_i^R, T_j^R)$ jeder Ressource $r_\ell \in R$ ausschließlich von den Werten solcher Ressourcen abhängig ist, deren Index kleiner oder gleich ℓ ist. Außerdem sei $f_{ij}^{r_\ell}$ unabhängig vom Wert von $T_j^{r_\ell}$. Sei also $1 \leq \ell \leq |R|$ und bezeichne $T_i^{R_\ell} = (T_i^{r_1}, \dots, T_i^{r_\ell})$, so soll gelten: $f_{ij}^{r_\ell} = f_{ij}^{r_\ell}(T_i^{R_\ell}, T_j^{R_{\ell-1}})$.

Satz 3.31. *Die Eigenschaften (i), (ii) und (iii) sind hinreichend, um die Überprüfung der Zulässigkeit einer Verkettung $\sigma = \sigma_1 \oplus \sigma_2$ in konstanter Zeit durchführen zu können.*

Beweis: Wir betrachten die Verkettung von $\sigma = \sigma_1 \oplus \sigma_2$. Die Verkettung wird durch die Aufnahme des Bogens (i, j) in die aktuelle Teillösung, d.h. durch die Festlegung $x_{ij} := 1$, hergestellt. Wegen (3.12) kann dadurch der Zulässigkeitsbereich der Ressourcenvariable T_j^r weiter eingeschränkt werden. Es wird gezeigt, daß der (nicht leere) Zulässigkeitsbereich von T_j^r für alle $r \in R$ ein geschlossenes Intervall ist, dessen untere und obere Grenze in konstanter Zeit bestimmt werden können. Dazu wird die Beweismethode der vollständigen Induktion benutzt. Zuerst untersuchen wir deshalb die Verknüpfung von zwei Segmenten, die jeweils aus genau einem Knoten bestehen, und anschließend zwei beliebige Segmente, die zu einem großen Segment zusammengesetzt werden. Wir verwenden im folgenden die Bezeichnungen L_i^r und U_i^r bzw. L_j^r und U_j^r für die untere und obere Grenze des Zulässigkeitsbereichs von T_i^r bzw. T_j^r , $r \in R$.

- (1) Angenommen, die Segmente σ_1 und σ_2 bestehen nur aus je einem Knoten, d.h. $V(\sigma_1) = \{i\}$ und $V(\sigma_2) = \{j\}$. Dann ist $x_{ik} = 0$ und $x_{kj} = 0$ für alle $k \in V$. Daher bedeuten die Restriktionen (3.12) keine Einschränkungen für die Ressourcenvariablen T_i^r und T_j^r . Die Zulässigkeitsbereiche von T_i^r und T_j^r sind deshalb für alle $r \in R$ geschlossene Intervalle, ausschließlich derer, die durch die Restriktionen (1.1f) vorgegeben sind, d.h. $T_i^r \in [a_i^r, b_i^r]$ und $T_j^r \in [a_j^r, b_j^r]$, $\forall r \in R$.

Die Ressourcen lassen sich entsprechend ihrer Ordnung im Ressourcenvektor R nacheinander in konstanter Zeit prüfen. Die Überprüfung der Ressource r_1 geschieht wie folgt:

Wegen (3.12) und $x_{ij} = 1$ gilt:

$$\begin{aligned} f_{ij}^{r_1}(T_i^R, T_j^R) &\stackrel{(iii)}{=} f_{ij}^{r_1}(T_i^{r_1}) \stackrel{(3.12a)}{\leq} T_j^{r_1}, \text{ falls } r_1 \in R^1 \text{ bzw.} \\ f_{ij}^{r_1}(T_i^R, T_j^R) &\stackrel{(iii)}{=} f_{ij}^{r_1}(T_i^{r_1}) \stackrel{(3.12b)}{=} T_j^{r_1}, \text{ falls } r_1 \in R^2. \end{aligned}$$

Außerdem ist $f_{ij}^{r_1}$ wegen (i) stetig und monoton in $T_i^{r_1}$. Ist $f_{ij}^{r_1}$ monoton wachsend, so folgt daraus

$$\begin{aligned} L_j^{r_1} &= \max[f_{ij}^{r_1}(a_i^{r_1}), a_j^{r_1}], \quad U_j^{r_1} = b_j^{r_1}, & \text{wenn } r_1 \in R^1 \text{ bzw.} \\ L_j^{r_1} &= \max[f_{ij}^{r_1}(a_i^{r_1}), a_j^{r_1}], \quad U_j^{r_1} = \min[f_{ij}^{r_1}(b_i^{r_1}), b_j^{r_1}], & \text{wenn } r_1 \in R^2. \end{aligned}$$

Ist $f_{ij}^{r_1}$ monoton fallend, dann gilt

$$\begin{aligned} L_j^{r_1} &= \max[f_{ij}^{r_1}(b_i^{r_1}), a_j^{r_1}], \quad U_j^{r_1} = b_j^{r_1}, & \text{wenn } r_1 \in R^1 \text{ bzw.} \\ L_j^{r_1} &= \max[f_{ij}^{r_1}(b_i^{r_1}), a_j^{r_1}], \quad U_j^{r_1} = \min[f_{ij}^{r_1}(a_i^{r_1}), b_j^{r_1}], & \text{wenn } r_1 \in R^2. \end{aligned}$$

Wegen der Stetigkeit von $f_{ij}^{r_1}$ ist der Zulässigkeitsbereich von $T_j^{r_1}$ nach der Verkettung von σ_1 und σ_2 entweder leer (falls $L_j^{r_1} > U_j^{r_1}$) oder ein geschlossenes Intervall. Die Intervallgrenzen können aufgrund der Voraussetzung (ii) in konstanter Zeit bestimmt werden.

Die obige Vorgehensweise und Argumentation kann analog für die Zulässigkeitsbereiche aller weiteren Ressourcenvariablen $T_j^{r_2}, \dots, T_j^{r_{|R|}}$ in dieser Reihenfolge fortgesetzt werden. Für die Bestimmung der Intervallgrenzen von $T_j^{r_k}, 2 \leq k \leq |R|$ werden wegen (ii) nur (die bereits ermittelten) Grenzen $L_i^{r_v}, U_i^{r_v}$ und $L_j^{r_v}, U_j^{r_v}$ der Zulässigkeitsbereiche der Ressourcenvariablen $T_i^{r_v}$ und $T_j^{r_v}, 1 \leq i, j < k$ benötigt.

Für jede Ressource $r_m \in R$ bezeichne

$$\alpha_i^{r_m} = \begin{cases} L_i^{r_m}, & \text{falls } f_{ij}^{r_m} \text{ monoton wachsend ist,} \\ U_i^{r_m}, & \text{falls } f_{ij}^{r_m} \text{ monoton fallend ist} \end{cases}$$

und

$$\beta_i^{r_m} = \begin{cases} U_i^{r_m}, & \text{falls } f_{ij}^{r_m} \text{ monoton wachsend ist,} \\ L_i^{r_m}, & \text{falls } f_{ij}^{r_m} \text{ monoton fallend ist.} \end{cases}$$

Dann gilt für $r_k \in R^2$:

$$L_j^{r_k} = \max [f_{ij}^{r_k} (\alpha_i^{r_1}, \dots, \alpha_i^{r_k}, \alpha_j^{r_1}, \dots, \alpha_j^{r_{k-1}}), a_j^{r_k}] \quad (3.13a)$$

$$U_j^{r_k} = \min [f_{ij}^{r_k} (\beta_i^{r_1}, \dots, \beta_i^{r_k}, \beta_j^{r_1}, \dots, \beta_j^{r_{k-1}}), b_j^{r_k}]. \quad (3.13b)$$

Die Berechnungen für $r_k \in R^2$ übertragen sich analog.

- (2) Seien σ_1 und σ_2 Segmente, die aus einem oder mehreren Knoten bestehen, und die Zulässigkeitsbereiche aller Ressourcenvariablen T_i^r und $T_j^r, r \in R$ seien geschlossene Intervalle mit

$$L_i^r \leq T_i^r \leq U_i^r \quad \text{bzw.} \quad L_j^r \leq T_j^r \leq U_j^r.$$

Dann kann die Feststellung der Zulässigkeitsbereiche wie unter (1) durchgeführt werden, wobei in (3.13a) und (3.13b) jeweils $a_j^{r_k}$ durch $L_j^{r_k}$ und $b_j^{r_k}$ durch $U_j^{r_k}$ zu ersetzen sind. Der Zulässigkeitsbereich der Ressourcenvariable $r_1 \in R^2$ ergibt sich z.B. bei monoton wachsender Funktion $f_{ij}^{r_1}(T_i^{r_1})$ durch

$$\max[f_{ij}^{r_1}(L_i^{r_1}), L_j^{r_1}] \leq T_j^{r_1} \leq \min[f_{ij}^{r_1}(U_i^{r_1}), U_j^{r_1}].$$

Demzufolge ist der Zulässigkeitsbereich jeder Ressourcenvariablen T_j^r , $r \in R$ nach der Verkettung von σ_1 mit σ_2 entweder leer oder ein geschlossenes Intervall, dessen Grenzen in konstanter Zeit bestimmt werden können.

□

3.3 Multiple TSP (m-TSP)

Als eine einfache Erweiterung des TSP wird in der Literatur das m-TSP betrachtet. Dieses unterscheidet sich vom TSP darin, daß statt einer Tour nun m Touren zu bilden sind, die an einem gemeinsamen Depot d starten und enden und gemeinsam jeden Kunden genau einmal besuchen.²⁰

3.3.1 Das Giant-Route-Modell für das m-TSP

Bereits 1969 haben Christofides und Eilon in [CE69] eine Graph-Transformation benutzt, mit der man ein m-TSP mit n Kundenknoten in ein äquivalentes TSP mit $n + m$ Knoten übertragen kann. Dazu werden $m - 1$ Kopien des Depotknotens d angelegt. Alle Kopien erhalten die gleichen eingehenden und ausgehenden Bögen wie das Original.

Die Menge der m Depotknoten sei mit $D = \{d_1, \dots, d_m\}$ bezeichnet. Die Depotknoten dienen jeweils gleichzeitig als Endknoten einer Tour und Anfangsknoten der nächsten Tour. Eine Lösung des resultierenden TSP wird in der Literatur häufig als Giant-Route bezeichnet (siehe Abb. 3.4). Wie man leicht sieht, repräsentiert die Giant-Route auch eine Lösung des zugrundeliegenden m-TSP.

Ausgehend vom Giant-Route-Modell mit m Depotknoten $d_i \in D$ wird beim Übergang zu dessen Pfaddarstellung ein Depotknoten aufgeteilt. So entsteht ein Pfad mit $m + 1$ Depotknoten, wobei der erste Depotknoten keine eingehenden Bögen und der letzte Depotknoten keine ausgehenden Bögen besitzt (siehe Abb. 3.5). Allerdings ist es hilfreich, eine einheitliche Darstellung aller Depots zu besitzen. Dazu werden alle Depotknoten $d_i \in D$ in je zwei Knoten aufgeteilt, die wir als Depotausgang o_i und Depoteingang d_i

²⁰Hier wird zunächst wieder das Kreismodell des TSP zugrundegelegt. Bei Anwendung auf das Pfadmodell des TSP werden m Pfade vom Depotausgang o zum Depoteingang d betrachtet.

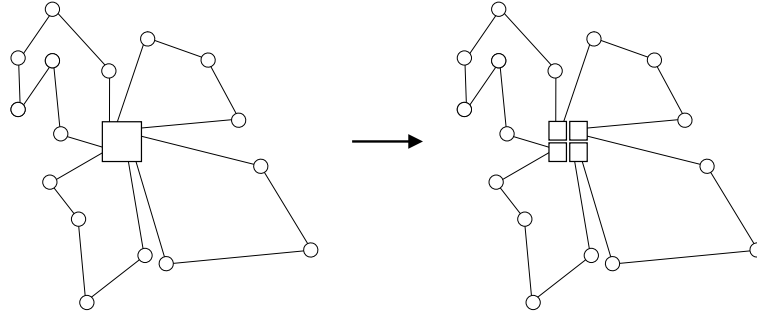


Abbildung 3.4: Giant-Route für ein m -TSP mit vier Touren: die Lösung des m -TSP mit vier einzelnen Touren (links) wird zu einer großen Tour – der Giant-Route (rechts).

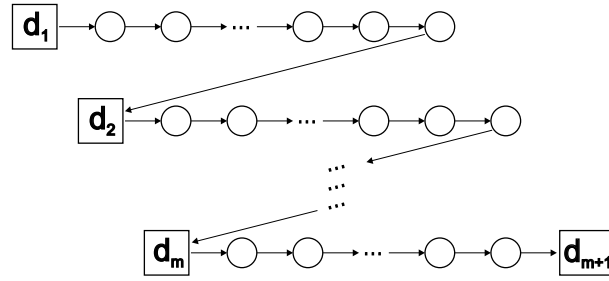


Abbildung 3.5: Pfaddarstellung einer Giant-Route ($m + 1$ Depotknoten)

bezeichnen. Die Zahl der Depotknoten erhöht sich dadurch auf $2m$ (siehe Abb. 3.6).

Nun kann analog zum TSP auch ein Pfadmodell für das m -TSP angegeben werden. Dieses unterscheidet sich vom Pfadmodell des TSP nur in der Definition des zugehörigen Graphen.

Definition 3.32. (Pfadmodell des m -TSP)

Das Pfadmodell gemäß Definition 3.19 auf einem schlichten gerichteten und bewerteten Graph $G = (V, A, c)$ heißt Pfadmodell des m -TSP, wenn für G

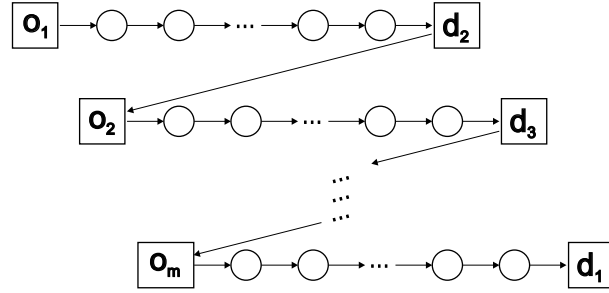


Abbildung 3.6: Pfaddarstellung einer Giant-Route mit getrennten Depotein- und Depotausgängen

gilt:

$$V = \{1, \dots, n\} \cup D, \quad (3.14a)$$

$$D = D^o \cup D^d, \quad D^o = \{o_1, \dots, o_m\}, \quad D^d = \{d_1, \dots, d_m\}, \quad (3.14b)$$

$$A = \{(i, j) : i, j \in V, i \neq j, \quad (3.14c)$$

$$i \in D^d \implies j \in D^o, \quad (3.14d)$$

$$i \in D^o \implies j \notin D^d, \quad (3.14e)$$

$$i \neq d_1, j \neq o_1\}. \quad (3.14f)$$

Bezeichne o den Depotausgang und d den Depoteingang, so werden die Kosten der Bögen von und zu den Kopien der Depotknoten entsprechend übertragen:

$$c(o_i, j) = c(o, j) \quad \forall i \in \{1, \dots, m\}, j \in \{1, \dots, n\} \quad (3.14g)$$

$$c(j, d_i) = c(j, d) \quad \forall i \in \{1, \dots, m\}, j \in \{1, \dots, n\} \quad (3.14h)$$

$$c(d_i, o_j) = 0 \quad \forall i \in \{1, \dots, m-1\}, j \in \{2, \dots, m\} \quad (3.14i)$$

(3.14a) und (3.14b) beschreiben die Zusammensetzung der Knotenmenge V aus n Kundenknoten sowie je m Depoteingangs- und Depotausgangsknoten. (3.14c)–(3.14f) definieren die Bogenmenge A . (3.14d) legt fest, daß ein Depoteingangsknoten ausschließlich mit einem Depotausgangsknoten verbunden werden kann. Durch (3.14e) wird sichergestellt, daß ein Depotausgang nicht direkt mit einem Depoteingang verbunden werden kann. Wenn auf diese Forderung verzichtet wird, können auch Lösungen entstehen, die ein oder

mehrere „leere Touren“ enthalten. (3.14f) sorgt dafür, daß jeder Hamiltonpfad in G im Depot d_1 beginnt und endet. Mit (3.14g) und (3.14h) werden die Kosten der Bögen von und zum Depot auf die Bögen von und zu den Kopien der Depotausgänge und Depoteingänge übertragen. (3.14i) legt fest, daß auf den Bögen von einem Depoteingang zu einem Depotausgang keine Kosten entstehen.

Analog zum Pfadmodell läßt sich auch das Kreismodell des TSP auf das m-TSP übertragen. Im Unterschied zum Pfadmodell müssen lediglich die durch 3.14f verbotenen Bögen zugelassen werden, damit es möglich ist, einen Hamiltonkreis im Graphen G zu bilden.

Um zu verhindern, daß Touren gebildet werden können, die keine Kundenknoten bedienen, kann man direkte Bögen zwischen den Depotausgängen und Depoteingängen verbieten bzw. deren Kosten so groß setzen, daß diese nicht unmittelbar nacheinander besucht werden. Das Giant-Route-Modell kann auch verwendet werden, wenn (anstelle von Kopien eines einzigen Depots) tatsächlich mehrere Depots existieren. Daher ermöglicht das Giant-Route-Modell eine einheitliche Sicht auf das Traveling Salesman Problem, das Multiple TSP, Multi-Depot Vehicle Routing Probleme und Location Routing Probleme.²¹

3.3.2 k -opt-Schritte auf der Giant-Route

Bei der Konstruktion von Heuristiken für m-TSP, VRP oder VRPTW werden häufig k -opt-Nachbarschaften für die Verbesserung von Lösungen benutzt. Viele Autoren wenden diese Nachbarschaften aber nur auf je eine Tour an (siehe z.B. [HG98], [CW01], [LS99]). In einigen Veröffentlichungen werden auch einzelne tourübergreifende Nachbarschaftsschritte explizit berücksichtigt. Meistens werden dann ein oder zwei der folgenden Schritte verwendet:

- 2-opt*-Schritte²²: [PR95],[dFS⁺00]
- Anwendung der Or -opt-Nachbarschaft auf zwei Touren, teilweise auch nur Relocation-Schritte: [dFS⁺00],[CL99]
- Verwendung eines „Cross-Exchange-Schritts“²³: [BGG⁺95], [PG99], [GTA99], [IMKY02]

²¹siehe z.B. [LNT88]

²²Der 2-opt*-Schritt wird oft auch als „Cross-Move“ bezeichnet. Die k -opt*-Nachbarschaft wurde in Abschnitt 41 auf Seite 48 beschrieben.

Die betrachteten Nachbarschaften können jedoch noch weiter verallgemeinert werden, wenn man die Giant-Route-Darstellung des m-TSP zugrundelegt. Da eine Giant-Route strukturell einer Lösung eines TSP entspricht, können aus methodischer Sicht genausogut beliebige k -opt-Schritte auf der Giant-Route untersucht werden, die z.B. einzelne Segmente invertieren oder mehr als zwei Touren gleichzeitig modifizieren.

Die „Giant-Route-Sicht“ führt damit zu einer Verallgemeinerung der Inter-Tour-Nachbarschaften, die alle oben angeführten Nachbarschaften als Spezialfälle umfaßt. Darüber hinaus können auch viele weitere Schritte dargestellt werden. Dazu gehört eine Reihe sehr interessanter Nachbarschaftsschritte, was durch die Beispiele in den Abbildungen 3.7, 3.8 und 3.9 illustriert werden kann.

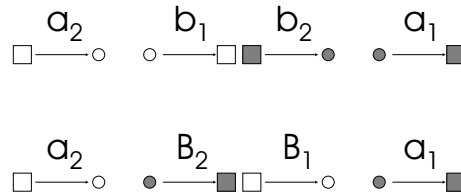


Abbildung 3.7: 2-opt-Schritt vom Typ $[aBa]$: Zwei Touren tauschen ein Anfangs- bzw. ein Endsegment miteinander.

Die im Zusammenhang mit der Definition der k -opt-Nachbarschaft im Abschnitt 3.1 eingeführten Begriffe k -Segmentierung, k -Permutation, k -Invertierung, Verkettung und k -opt-Schritt sind allgemein für Bogenfolgen definiert und können deshalb direkt auf eine Giant-Route bzw. auf das Pfadmodell 3.32 übertragen werden. Dementsprechend kann auch die Lexikographische Suche auf ein m-TSP angewendet werden.

²³Der Cross-Exchange-Schritt wird in [TBG⁺97] beschrieben. Dieser Schritt tauscht je ein Segment zwischen zwei verschiedenen Touren, ohne die Orientierung der Segmente zu ändern. Er kann als Verallgemeinerung von 2-opt*-Schritten und Or -opt-Schritten aufgefaßt werden. Auf der Giant-Route lassen sich alle diese Schritte als 4-opt-Schritte vom Typ $[adcba]$ auffassen. In der Literatur zum TSP werden Schritte der Form $[adcba]$ auch als „double bridge moves“ bezeichnet und insbesondere häufig als Kick-Schritte im Rahmen einer Iterated Local Search beim TSP eingesetzt.

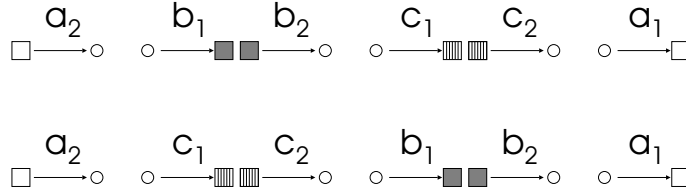


Abbildung 3.8: 3-opt-Schritt vom Typ $[acba]$: Analog zum 2-opt*-Schritt werden die Anfangs- und Endsegmente der Touren untereinander getauscht.

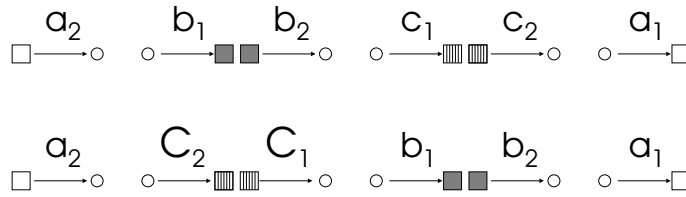


Abbildung 3.9: 3-opt-Schritt vom Typ $[aCba]$, der gleichzeitig Segmente zwischen drei Touren austauscht. Dabei wird das Endsegment c_1 invertiert und zum Anfangssegment C_1 und das Anfangssegment c_2 zum Endsegment C_2 .

3.3.3 Berücksichtigung beschränkter Ressourcen beim m-TSP

Sollen im Rahmen einer Lexikographischen Suche beschränkte Ressourcen berücksichtigt werden, so kommt es beim m-TSPR ebenso wie beim TSPR darauf an, daß die Überprüfung der Zulässigkeit und die Bestimmung des Zielfunktionswertes für jeden Nachbarschaftsschritt in konstanter Zeit durchgeführt werden kann. Dabei können die Überlegungen aus Abschnitt 3.2 auch auf das m-TSPR übertragen werden. Eine Sonderrolle spielen jedoch die Inter-Depot-Bögen, die einen Depoteingang mit einem Depotausgang verbinden. Diese Bögen repräsentieren den Übergang zwischen zwei Touren und sind beim Pfadmodell des TSP nicht zugelassen.

Bestimmung von LAT und EDT Um die Lexikographische Suche ohne größere Modifikationen auf die Lösung eines m-TSPR übertragen zu können, müssen alle Berechnungen unabhängig davon sein, ob ein Segment einer einzelnen Tour oder ein Segment innerhalb einer Giant-Route betrachtet wird.

Verkettungen über einen Inter-Depot-Bogen verknüpfen zwei unabhängige Touren miteinander.

Sei σ_1 ein zulässiges Segment, dessen letzter Knoten i ein Depoteingangsknoten ist, und σ_2 ein zulässiges Segment, dessen erster Knoten j ein Depotausgangsknoten ist. Dann muß die Verkettung $\sigma = \sigma_1 \oplus \sigma_2$ stets zulässig sein, und das entstehende Segment σ muß sich linksseitig wie σ_1 und rechtsseitig wie σ_2 verhalten, d.h. es muß für alle Ressourcen $r \in R$ gelten:

$$LAT_\sigma^r = LAT_{\sigma_1}^r \text{ und } EDT_\sigma^r = EDT_{\sigma_2}^r.$$

Die Bestimmung der Werte LAT^r und EDT^r kann jedoch weiterhin für alle Ressourcen r gemäß (3.8e) und (3.8f) erfolgen, wenn die Ressourcenverbräuche $d_{ij}^r = d_{\sigma_1\sigma_2}^r$ auf den Inter-Depot-Bögen hinreichend klein gewählt werden, d.h. $d_{ij}^r < -M$, wobei $M > 0$ eine große Zahl ist. Dann gilt sowohl

$$\begin{aligned} LAT_\sigma^r &= \min\{LAT_{\sigma_1}^r, LAT_{\sigma_2}^r - d_{\sigma_1\sigma_2}^r - D_{\sigma_1}^r\} \\ &= LAT_{\sigma_1}^r, \end{aligned} \tag{3.15a}$$

$$\begin{aligned} EDT_\sigma^r &= \max\{EDT_{\sigma_1}^r + d_{\sigma_1\sigma_2}^r + D_{\sigma_2}^r, EDT_{\sigma_2}^r\} \\ &= EDT_{\sigma_2}^r \end{aligned} \tag{3.15b}$$

als auch

$$EDT_{\sigma_1}^r + d_{\sigma_1\sigma_2}^r \leq LAT_{\sigma_2}^r, \tag{3.15c}$$

und die Zulässigkeit der Verkettung ist somit stets gesichert.

Bestimmung von D Der Wert D_σ^r wird benötigt, um die Werte von LAT^r und EDT^r für Verkettungen von σ mit weiteren Segmenten bestimmen zu können.²⁴ Er soll angeben, um wieviel sich die Ressource r beim Durchlaufen des Segments σ insgesamt verändert hat. Mit der Rückkehr zu einem Depot beginnt jedoch eine neue Tour, deren Ressourcenbilanz normalerweise unabhängig von der vorherigen Tour ist.

Der „richtige“ Wert von D_σ^r ist deshalb abhängig davon, ob das Segment linksseitig oder rechtsseitig verkettet wird. Es ist also notwendig, zwei verschiedene Werte für den Ressourcenverbrauch im Segment σ zu verwalten. Wir bezeichnen diese Werte mit $D_\sigma^{\text{left},r}$ für den linksseitigen Wert von D_σ^r

²⁴vgl. (3.8e) und (3.8f) auf Seite 95

und $D_{\sigma}^{\text{right},r}$ für den rechtsseitigen Wert. Um außerdem in der Lage zu sein, den gesamten Ressourcenverbrauch einer Lösung zu bestimmen (z.B. für die effiziente Berechnung der Kostenfunktion), wird außerdem der Wert $D_{\sigma}^{\text{sum},r}$ verwaltet, der alle Ressourcenverbräuche von r der in σ vollständig enthaltenen Touren angibt.

Für jeden einzelnen Knoten v gilt $D_{\sigma}^{\text{left},r} = D_{\sigma}^{\text{right},r} = s_v^r$ und $D_{\sigma}^{\text{sum},r} = 0$. Die Berechnung von $D_{\sigma}^{\text{left},r}, D_{\sigma}^{\text{right},r}$ und $D_{\sigma}^{\text{sum},r}$ kann für die Verkettung $\sigma = \sigma_1 \oplus \sigma_2$ ebenfalls nach wie vor mit konstantem Aufwand durchgeführt werden, erfordert aber eine Fallunterscheidung danach, ob die Segmente σ_1 und σ_2 (eine oder mehrere) vollständige Touren umfassen, und danach, ob der hinzugefügte Bogen (i, j) ein Inter-Depot-Bogen ist. Die insgesamt acht unterschiedlichen Fälle sind in Abb. 3.10 auf der nächsten Seite graphisch dargestellt.

Für die in Abb. 3.10 unterschiedenen vier Hauptfälle (a) bis (d) bestimmen sich die Werte von $D_{\sigma}^{\text{left},r}$, $D_{\sigma}^{\text{right},r}$ und $D_{\sigma}^{\text{sum},r}$ durch:

$$\begin{aligned}
 & D_{\sigma}^{\text{left},r} = D_{\sigma_1}^{\text{left},r} + \phi_{ij}d_{ij}^r + D_{\sigma_2}^{\text{left},r} \\
 \text{(a)} \quad & D_{\sigma}^{\text{right},r} = D_{\sigma}^{\text{left},r} \\
 & D_{\sigma}^{\text{sum},r} = 0 \\
 & D_{\sigma}^{\text{left},r} = D_{\sigma_1}^{\text{left},r} + \phi_{ij}d_{ij}^r + \phi_{ij}D_{\sigma_2}^{\text{left},r} \\
 \text{(b)} \quad & D_{\sigma}^{\text{right},r} = D_{\sigma_2}^{\text{right},r} \\
 & D_{\sigma}^{\text{sum},r} = (1 - \phi_{ij})D_{\sigma_2}^{\text{left},r} + D_{\sigma_2}^{\text{sum},r} \\
 & D_{\sigma}^{\text{left},r} = D_{\sigma_1}^{\text{left},r} \\
 \text{(c)} \quad & D_{\sigma}^{\text{right},r} = \phi_{ij}D_{\sigma_1}^{\text{right},r} + \phi_{ij}d_{ij}^r + D_{\sigma_2}^{\text{right},r} \\
 & D_{\sigma}^{\text{sum},r} = D_{\sigma_1}^{\text{sum},r} + (1 - \phi_{ij})D_{\sigma_1}^{\text{right},r} \\
 & D_{\sigma}^{\text{left},r} = D_{\sigma_1}^{\text{left},r} \\
 \text{(d)} \quad & D_{\sigma}^{\text{right},r} = D_{\sigma_2}^{\text{right},r} \\
 & D_{\sigma}^{\text{sum},r} = D_{\sigma_1}^{\text{right},r} + D_{\sigma_1}^{\text{sum},r} + \phi_{ij}d_{ij}^r + D_{\sigma_2}^{\text{left},r} + D_{\sigma_2}^{\text{sum},r}
 \end{aligned}$$

Dabei ist

$$\phi_{ij} = \begin{cases} 0, & \text{falls der Bogen } (i, j) \text{ ein Inter-Depot-Bogen ist,} \\ 1, & \text{sonst.} \end{cases}$$

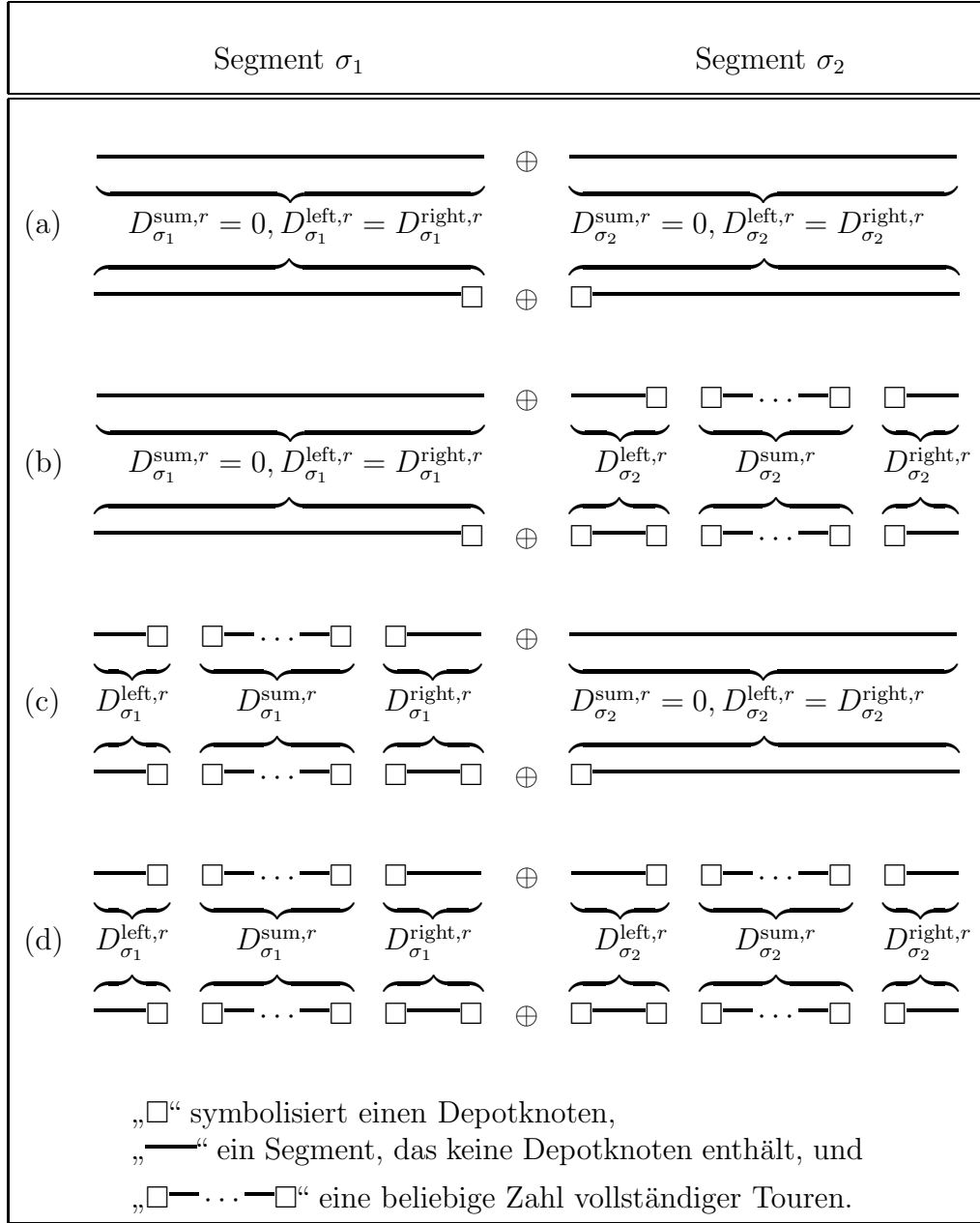


Abbildung 3.10: Fallunterscheidung bei der Bestimmung des Ressourcenverbrauchs für Segmente der Giant-Route

3.3.4 k -opt*-Nachbarschaft

Für die Suche nach verbessernden k -opt-Schritten kommen bei einer Lösung eines m-TSP, die mehrere Touren umfaßt, auch k -opt*-Schritte in Frage. Die k -opt*-Nachbarschaft ist eine Erweiterung der k -opt-Nachbarschaft. Sie enthält auch solche Lösungen, die – bezogen auf die Giant-Route – Kurzzyklen umfassen. Sofern jeder Kurzzyklus mindestens ein Depot enthält, repräsentieren diese Nachbarlösungen zulässige Lösungen des m-TSP.²⁵

Erweiterung der Beschreibung eines k -opt-Schritts

Zur Beschreibung von k -opt*-Schritten kann die prinzipielle Struktur des k -opt-Schritts als Nacheinanderausführung von k -Segmentierung, k -Invertierung, k -Permutation und Verkettung (Definition 3.28) beibehalten und erweitert werden. Die Erweiterung bezieht sich lediglich auf die abschließende Verkettung der Segmente. Während bei einem k -opt-Schritt nur eine einzige Verkettung erlaubt ist – jedes Segment wird mit dem (nach der k -Permutation feststehenden) Nachfolgersegment verkettet – kommen bei einem k -opt*-Schritt mehrere Möglichkeiten in Betracht. Demzufolge können maximal k

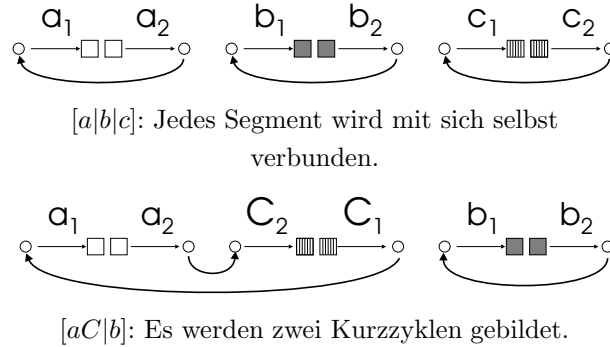


Abbildung 3.11: 3-opt*-Schritte vom Typ $[a|b|c]$ und $[aC|b]$ ²⁶

²⁵Die k -opt*-Nachbarschaft wurde bereits im Abschnitt 2.2.2 beschrieben (siehe S. 48).

²⁶In der Pfaddarstellung der Giant-Route erfordern das letzte und das erste Segment eine gesonderte Betrachtung, da diese eigentlich miteinander verbunden sein müßten. Zur Vereinfachung der Beschreibung gehen wir davon aus, daß diese Segmente bereits miteinander verbunden wurden und somit genau k Segmente vorliegen. Wir benutzen auch für k -opt*-Schritte die bereits für k -opt-Schritte eingeführte Kurzschreibweise (siehe S. 84) und kennzeichnen getrennte Kurzyklen durch einen senkrechten Strich.

Kurzzyklen entstehen.

Angenommen, nach der k -Segmentierung enthält jedes Segment ein Depot. Dann könnte aus jedem einzelnen Segment ein zulässiger Kurzzzyklus gebildet werden, indem das Ende des Segments mit seinem eigenen Anfang verbunden wird (siehe Abb. 3.11). Da für die ersten $(k - 1)$ Segmente jeweils entschieden werden kann, ob das Segment mit seinem Nachfolgersegment verbunden wird oder nicht, existieren zu jedem k -opt-Typ insgesamt $2^{(k-1)}$ verschiedene k -opt*-Typen. Unterscheiden sich zwei k -opt-Typen ausschließlich bezüglich ihrer k -Permutation, dann sind die zugehörigen k -opt*-Typen nicht alle voneinander verschieden. So ist die k -Permutation z.B. vollkommen bedeutungslos, wenn insgesamt k Kurzzyklen gebildet werden.

Beispiel 3.3. Wir betrachten die beiden 3-opt-Typen $[a_2Cba_1]$ und $[a_2bCa_1]$. Wir gehen davon aus, daß die Segmente a_1 und a_2 stets miteinander verkettet werden ($a = a_1 \oplus a_2$). Dann lassen sich die k -opt-Typen schreiben als $[aCb]$ und $[abC]$.

Zu jedem der beiden k -opt-Typen gibt es $2^{(k-1)} = 4$ zugehörige k -opt*-Typen:

$$\begin{aligned} [aCb] &\Rightarrow [aCb], [a|Cb], [aC|b] \text{ und } [a|C|b] \\ [abC] &\Rightarrow [abC], [a|bC], [ab|C] \text{ und } [a|b|C] \end{aligned}$$

Dabei repräsentieren die Kurzzyklen umfassenden k -opt*-Typen $[a|Cb]$ und $[a|bC]$ bzw. $[a|C|b]$ und $[a|b|C]$ dieselben Nachbarlösungen.

Um die Anzahl der möglichen k -opt*-Typen festzustellen, betrachten wir eine alternative Darstellung der k -opt*-Schritte. Statt die k -Permutation und die Art der anschließenden Verkettung getrennt zu modellieren, lassen sich deren unterschiedliche Kombinationen integriert darstellen. Dazu beschreiben wir die k -Permutation nicht mehr, indem wir die neue Reihenfolge der Segmente angeben (vgl. Definition 3.26). Stattdessen notieren wir für jedes Segment den Index desjenigen Segments, welches bei der Durchführung des k -opt-Schritts das Nachfolgersegment werden soll. Diese Darstellung erlaubt es, beliebige Kurzzyklen zu bilden. Insgesamt gibt es $k!$ verschiedene Möglichkeiten, die Nachfolgerindizes zu wählen, und folglich $k! \cdot 2^{(k-1)}$ verschiedene k -opt*-Typen. Man beachte, daß in dieser Zahl nicht nur echte k -opt*-Typen enthalten sind.

Beispiel 3.4. Wir betrachten als Beispiel k -opt*-Schritte mit $k = 4$. Dann wird die Ausgangslösung in vier Segmente $\sigma_1, \dots, \sigma_4$ (bzw. a, b, c und d) zerlegt. Ohne eine Umordnung der Segmente erhält man den k -opt-Typ $[abcd]$, der die Lösung unverändert läßt („Identität“). Die Reihenfolge der Segmentindizes kann durch den 4-Tupel $(1, 2, 3, 4)$ angegeben werden. Notieren wir stattdessen an der i -ten Position, welches das Nachfolgersegment des i -ten Segments ist, dann wird die Ausgangslösung durch den Tupel $(2, 3, 4, 1)$ beschrieben.

Nun betrachten wir als Beispiele die 4-opt*-Typen $[adcb]$, $[ad|cb]$ und $[a|dc|b]$. Diese werden durch die folgenden 4-Tupel beschrieben:

$$\begin{aligned} [adcb] &\rightarrow (4, 1, 2, 3) \\ [ad|cb] &\rightarrow (4, 3, 2, 1) \\ [a|dc|b] &\rightarrow (1, 2, 4, 3) \end{aligned}$$

Die Interpretation des ersten 4-Tupels lautet: Das Nachfolgersegment von Segment σ_1 (a) ist das Segment σ_4 (d), Nachfolger von Segment σ_2 (b) ist σ_1 (a), Nachfolger von Segment σ_3 (c) ist σ_2 (b) und Nachfolger von Segment σ_4 (d) ist σ_3 (c).

Überprüfung der Zulässigkeit

Im Abschnitt 3.2 wurde beschrieben, wie die Einhaltung von Ressourcenbeschränkungen jeweils bei der Verkettung von Segmenten effizient überprüft werden kann. Diese Überprüfung kann analog bei der Suche in einer k -opt*-Nachbarschaft eingesetzt werden. Die Tatsache, daß bei k -opt*-Schritten Kurzzyklen entstehen können, beeinflusst die Vorgehensweise nicht, weil die zu verbindenden Segmente nicht disjunkt sein müssen.

Bei der Betrachtung von k -opt*-Schritten muß zusätzlich gesichert werden, daß jeder gebildete Kurzzyklus ein Depot enthält. Dazu kann die Zahl der in einem Segment enthaltenen Depotknoten in einer zusätzlichen Ressource abgelegt werden. Eine Verkettung eines Segments mit sich selbst ist nur dann zulässig, wenn die jeweilige Ressource einen Wert größer null aufweist.

Im Unterabschnitt 3.3.3 mußte der Wert D_σ^r , der im Zusammenhang mit der effizienten Überprüfung der Ressource r bei Verkettungen benötigt wird, beim Übergang zum m-TSP auf die drei Werte $D_\sigma^{\text{left},r}$, $D_\sigma^{\text{right},r}$ und $D_\sigma^{\text{sum},r}$ erweitert werden. Sofern ein Segment mit sich selbst verbunden wird, werden die Werte $D_\sigma^{\text{left},r}$ und $D_\sigma^{\text{right},r}$ nicht weiter benötigt, da der entstehende Kurzzyklus ohnehin nicht mit weiteren Segmenten verkettet werden darf. Der Wert von $D_\sigma^{\text{sum},r}$ wurde dagegen bestimmt, um den gesamten Ressourcenverbrauch der Ressource r einer Lösung stets in konstanter Zeit berechnen zu können. Dabei müssen die angegebenen Aktualisierungen (vgl. Seite 109) beim Entstehen von Kurzzyklen angepaßt werden:

1. Beim Verbinden eines Segments mit sich selbst können nur die Fälle (a) oder (d) eintreten. Dabei würden die Ressourcenverbräuche des Segments (mit Ausnahme des Verbrauchs auf dem gerade hinzugefügten Bogen) doppelt gezählt werden.
2. Wenn die Lösung aus mehreren Kurzzyklen besteht, werden die Ressourcenverbräuche nur jeweils innerhalb eines Kurzzyklus aggregiert.

Da die Kurzzyklen nicht durch Verkettungen miteinander verbunden werden, müssen die Ressourcenverbräuche der einzelnen Zyklen explizit summiert werden, um den gesamten Verbrauch der Lösung zu erhalten.

3.4 Suche in großen k -opt-Nachbarschaften

Aufgrund der vorangegangenen Betrachtungen ist es prinzipiell leicht möglich, auch k -opt-Nachbarschaften mit sehr großem k ($k \geq 4$) zu betrachten. Große Nachbarschaften sind aus zweierlei Sicht interessant. Rousseau und Gendreau [RGP02a] schreiben dazu:

*„The advantages of using a large neighborhood are not only the increased probability of finding a better solution at each iteration but also the reduction of the need to invoke specially-designed methods to avoid local minima.“*²⁷

Allerdings wächst die Größe der k -opt-Nachbarschaften exponentiell mit k , so daß die bislang betrachtete vollständige Enumeration der Nachbarschaft für $k \geq 4$ nicht in sinnvoller Zeit durchführbar ist. Darüber hinaus muß die Frage gestellt werden, ob der durch eine Erweiterung implizierte höhere Rechenaufwand auch eine angemessene Verbesserung der Lösungsqualität herbeiführen kann bzw. ob man durch eine alternative Vorgehensweise (z.B. in Form eines Multi-Start Verfahrens) bei gleichem Aufwand bessere Ergebnisse erreichen könnte.

Zweifelsohne weisen sehr viele Lösungen einer k -opt-Nachbarschaft relativ hohe Kosten auf und sind deshalb eigentlich uninteressant. Im Abschnitt 3.1 wurde ab Seite 86 bereits beschrieben, wie man Lösungen, die keine Verbesserung herbeiführen können, innerhalb der Lexikographischen Suche frühzeitig erkennen und bei der weiteren Suche ausklammern kann. Um die Effizienz der Suche weiter zu verbessern, kann man darüber hinaus möglichst große Teilbereiche der Nachbarschaften von vornherein heuristisch ausschließen, die voraussichtlich keine oder nur wenige „gute“ Lösungen enthalten. Der Übergang zu solchermaßen reduzierten k -opt-Nachbarschaften erlaubt es, die Nachbarschaften mit kleinem k wesentlich schneller und zusätzliche Nachbarschaften mit größerem k (etwa bis $k = 8$) zu untersuchen.

²⁷Die Vorteile der Verwendung einer großen Nachbarschaft liegen nicht nur in der erhöhten Wahrscheinlichkeit, bei jeder Iteration eine bessere Lösung zu finden, sondern auch in der Verringerung der Notwendigkeit, speziell entwickelte Methoden zur Vermeidung von lokalen Minima aufzurufen.

In diesem Abschnitt wird zunächst darauf eingegangen, wie die k -opt-Nachbarschaften sinnvoll reduziert werden können. Anschließend werden alternative Suchstrategien für die k -opt-Nachbarschaften angesprochen, die nicht auf einer vollständigen Enumeration der Nachbarschaft beruhen und somit die Untersuchung größerer Nachbarschaften erlauben.

3.4.1 Reduktion der k -opt-Nachbarschaft

Die Idee bei der Reduktion von Nachbarschaften besteht darin, (große) Teile der Nachbarschaft (heuristisch) auszuschließen, weil diese wahrscheinlich keine oder nur wenige verbessernde Lösungen beinhalten. Dadurch kann der Aufwand für die Suche in der Nachbarschaft ggf. deutlich reduziert werden, ohne daß sich die Qualität der gefundenen Lösungen wesentlich verschlechtert.

Verwendung von Nächste-Nachbarn-Listen Der Einsatz von Nächste-Nachbarn-Listen (NN-Liste) basiert auf der Annahme, daß eine gute oder optimale Lösung eines VRSP mit hoher Wahrscheinlichkeit *ausschließlich* Bögen (i, j) besitzt, die den Knoten i mit einem der ℓ ($\ell < n$) zu ihm nächstgelegenen Knoten j verbindet.

Man bestimmt zu Beginn für jeden Kundenknoten $i \in N$ eine NN-Liste. Diese umfaßt die ℓ nächsten Nachbarn des jeweiligen Knotens. Bei der Konstruktion einer Nachbarlösung darf ein Bogen (i, j) nur dann eingefügt werden, wenn der Knoten j in der NN-Liste von Knoten i enthalten ist. Anders formuliert, wird eine neue Nachbarschaft betrachtet, der ein reduzierter Graph $G' = (V, A', c)$ mit $A' \subseteq A$ zugrunde liegt. G' beinhaltet zu jedem Knoten $i \in V$ die ℓ kürzesten Bögen $(i, j) \in A$ und (ggf. zusätzlich) alle Bögen der aktuellen Lösung y . Der Einsatz von NN-Listen wird sinngemäß bereits in [LK73] für das TSP vorgeschlagen, wobei für ℓ der Wert 5 angegeben wird. Auch in Rego [Reg96] werden NN-Listen für das TSP eingesetzt.

Wie das folgende Beispiel deutlich macht, ist die obige Annahme bei einem TSP nicht immer erfüllt.

Beispiel 3.5. Man betrachte die Abb. 3.12a, die den reduzierten Graphen G' bei einem symmetrischen TSP zeigt. Die Orientierungen der Bögen spielen bei dem symmetrischen Problem keine Rolle. Deshalb wurden nur ungerichtete Kanten eingezeichnet. Die fetten Kanten geben die Ausgangslösung an, die durch eine Greedy-Heuristik entstanden sein könnte, bei der zuerst der Bogen $(1, 2)$ fixiert wurde. Der Parameter ℓ wurde in diesem Beispiel gleich fünf gesetzt.

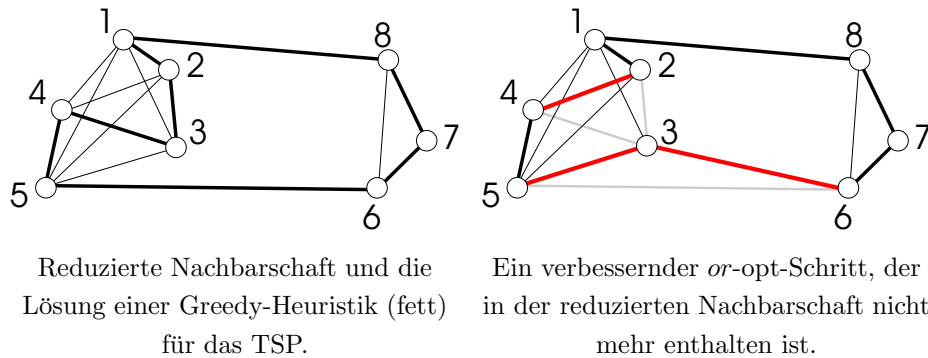


Abbildung 3.12: Durch NN-Listen reduzierte Nachbarschaft beim TSP.

Ein vollständiger Graph mit zehn Knoten würde 45 Kanten besitzen. Durch die NN-Listen wurde die Zahl der Kanten auf 22 reduziert. Dabei sind allerdings auch die Kanten (3, 6) und (2, 8) verlorengegangen. Deren Verfügbarkeit würde aber eine Verbesserung der Lösung erlauben. So könnte z.B. der *Or*-opt-Schritt durchgeführt werden, der die Bögen (2, 3), (3, 4) und (5, 6) gegen die (2, 4), (5, 3) und (3, 6) tauscht und damit den Knoten 3 an die Position zwischen 5 und 6 verschiebt (Abb. 3.12b).

Dennoch ist die Verwendung von NN-Listen beim TSP i.a. sehr erfolgreich. Oft wird die Forderung allerdings auch dadurch abgeschwächt, daß nicht alle neuen Bögen zu einem der jeweils nächsten ℓ Nachbarn führen müssen. Beim Lin-Kernighan-Algorithmus darf z.B. der letzte Bogen ohne Einschränkungen gewählt werden.

Werden Probleme mit Ressourcenbeschränkungen betrachtet, dann kann die Gültigkeit der obigen Annahme durch die Existenz der Restriktionen zusätzlich eingeschränkt werden. Dies gilt vor allem dann, wenn die Ressourcenbeschränkungen die Gestalt des zulässigen Lösungsraums Y wesentlich beeinflussen. Beispiele dafür wären TSPTW oder VRPTW mit (einzelnen) engen Zeitfenstern. Im Extremfall kann es aufgrund der Ressourcenbeschränkungen dazu kommen, daß einige Knoten mit keinem ihrer ℓ nächsten Nachbarknoten verbunden werden können. Die Berücksichtigung von (zu kurzen) NN-Listen führt dann dazu, daß kaum noch verbessernde Nachbarschaftsschritte gefunden werden. Wenn man die NN-Listen dagegen zu lang gewählt, ist der dadurch erzielbare Performance-Gewinn gering.

Eine Anpassung des Parameters ℓ an die konkrete Problemstellung erscheint daher insbesondere für Probleme mit Ressourcenbeschränkungen notwendig. Dennoch werden NN-Listen fixer Länge von einigen Autoren auch

für die Lösung von m-TSPR eingesetzt. Beispiele dafür sind der Flower-Algorithmus von Rego für das VRP [Reg98], GENIUS-TW für m-TSPTW [GHLS95] bzw. GENIUS-CP für m-TSPR [PGR97].

Auf einer ähnlichen Idee basieren die von Toth und Vigo in [TV02] beschriebenen „Granular Neighborhoods“. Toth und Vigo unterteilen die Bogenmenge ebenfalls in „kurze“ und „lange“ Bögen. Sie gehen bei der Konstruktion der „Granular Neighborhoods“ aber von einer anderen Annahme aus. Diese besagt, daß eine verbessernde Nachbarlösung *mindestens* einen neuen „kurzen“ Bogen beinhalten muß. Daher werden Schritte, die nur lange Bögen einfügen, nicht berücksichtigt.

In [TV02] werden die „Granular Neighborhoods“ im Rahmen eines Tabu Search beim VRP verwendet. Dabei wird die Unterteilung in kurze und lange Bögen anhand eines „granularity threshold“ ϑ vorgenommen:

$$\vartheta = \beta \cdot \frac{c(y')}{n + m}.$$

Dabei bezeichnet $y' \in Y$ eine bekannte zulässige Lösung, die durch eine schnelle Heuristik generiert wurde, und β einen Ausdünnungsparameter, dessen Wert zwischen 0.5 und 5 gewählt wurde. Bögen, deren Kosten nicht größer als der Schwellwert ϑ sind, zählen zu den kurzen Bögen. Alle übrigen Bögen sind lang. Das Verfahren konnte für eine Reihe von Benchmark-Problemen in sehr kurzer Zeit im Vergleich zu anderen erfolgreichen Meta-Heuristiken exzellente Lösungen bestimmen.²⁸

Beschränkung von Segmentlängen

Eine sehr erfolgreiche reduzierte k -opt-Nachbarschaft ist die von Or [Or76] vorgeschlagene und nach ihm benannte *Or*-opt-Nachbarschaft.²⁹ Or betrachtet 3-opt-Schritte vom Typ $[acba]$. Er reduziert die Nachbarschaft dadurch, daß er ausschließlich das Verschieben von maximal drei aufeinanderfolgenden Knoten an eine neue Position innerhalb der Tour erlaubt. Durch diese Einschränkung kann die Größe der Nachbarschaft durch $\mathcal{O}(n^2)$ (statt $\mathcal{O}(n^3)$ für die 3-opt-Nachbarschaft) abgeschätzt werden.

Bezogen auf den 3-opt-Typ $[acba]$ kann man die obige Forderung äquivalent formulieren als: Mindestens eins der beiden „inneren“ Segmente, d.h. „b“

²⁸siehe [TV02] und [LGPS00]

²⁹siehe Abschnitt 2.2.2 auf Seite 48

oder „c“, darf höchstens drei Knoten umfassen. Auf der Basis dieser Formulierung kann die Idee von Or verallgemeinert und auch auf größere k -opt-Nachbarschaften übertragen werden. Dazu wird eine Mindestzahl „kurzer innerer“ Segmente für die Nachbarschaftsschritte der reduzierten k -opt-Nachbarschaft gefordert. Für die Beschreibung dieser Nachbarschaft verwenden wir zwei Parameter. Der Parameter ℓ_s bezeichnet die maximale Länge eines kurzen Segments. Der Wert von k_s ($k_s < k$) gibt an, wieviele kurze „innere“ Segmente jeder Nachbarschaftsschritt mindestens aufweisen soll. Der Wert $k_\ell = k - k_s$ gibt die Zahl der übrigen Segmente an, deren Länge nicht begrenzt werden soll.

Die Komplexität von k -opt-Nachbarschaften wird durch die Zahl der Möglichkeiten determiniert, die k Segmente auszuwählen.³⁰ Durch die Beschränkung von k_s Segmenten auf eine Länge von höchstens ℓ_s kann anstelle der Zahl $\binom{n}{k}$ nun eine obere Schranke von

$$\underbrace{\binom{n}{k - k_\ell}}_{\substack{\text{Position der langen} \\ \text{Segmente wählen}}} \cdot \underbrace{\binom{k - 1}{k_s}}_{\substack{\text{Position der kurzen} \\ \text{Segmente wählen}}} \cdot \underbrace{\ell_s^{k_s}}_{\substack{\text{Länge der kurzen} \\ \text{Segmente wählen}}}$$

angegeben werden. Daher kann die Größe dieser reduzierten Nachbarschaft durch $\mathcal{O}(n^{k_\ell}) = \mathcal{O}(n^{k-k_s})$ abgeschätzt werden.

Die Or -opt-Nachbarschaft ist dann ein Spezialfall einer 3-opt-Nachbarschaft mit beschränkten Segmentlängen, bei dem $\ell_s = 2$ und $k_s = 1$ gewählt wird.³¹ Dementsprechend ist die Komplexität einer Suche in der Or -opt-Nachbarschaft $\mathcal{O}(n^2)$.

Es sei darauf hingewiesen, daß die hier beschriebene Beschränkung von Segmentlängen ohne nennenswerten Aufwand in die Lexikographische Suche (siehe Algorithmus 3.1 auf Seite 85) integriert werden kann. Es genügt, in einer Variablen die aktuelle Anzahl der bereits gebildeten „langen“ Segmente zu verwalten. So kann bei der Erhöhung der Länge eines Segments im Schritt 3.3 sofort festgestellt werden, wenn die geforderte Mindestzahl kurzer

³⁰vgl. Seite 41

³¹Die Länge eines Segments gibt die Anzahl der Bögen an. Ein Segment der Länge zwei besteht deshalb aus zwei Bögen, die insgesamt drei Knoten miteinander verbinden. Auf die zusätzliche Beschränkung der Or -opt-Nachbarschaft auf 3-opt-Schritte vom Typ $[acba]$ wird im weiteren noch explizit eingegangen werden.

Segmente nicht mehr eingehalten werden kann. Dann wird die entsprechende Schleife analog zum Schritt 3.2 sofort abgebrochen.

Berücksichtigung ausgewählter k -opt-Typen

Eine weitere Möglichkeit, die k -opt-Nachbarschaft zu reduzieren, besteht darin, nur bestimmte k -opt-Typen zu benutzen. Auch diese Option wird z.B. bei der Or -opt-Nachbarschaft wahrgenommen, bei der nur 3-opt-Schritte vom Typ $[acba]$ berücksichtigt werden.

Die wesentlichen Argumente dafür sind:

1. Ein Teil der Invertierungen können auch durch die Untersuchung der 2-opt-Nachbarschaft gefunden werden.
2. Bei Problemen mit Ressourcenbeschränkungen (z.B. bei Existenz von engen Zeitfenstern) sind die Aussichten gering, zulässige Nachbarschaftsschritte zu finden, die ganze Segmente invertieren.

Beide Argumente müssen aber sehr in Frage gestellt werden: Auch wenn die 2-opt-Nachbarschaft untersucht wird, müssen dadurch nicht alle Verbesserungen gefunden werden, die mit Hilfe der (echten) 3-opt-Schritte der Typen $[aCba]$, $[acBa]$ und $[aBCa]$ erzeugt werden können. Daher wird durch die Einschränkung der Nachbarschaft auch Verbesserungspotential verschenkt.

Die Frage ist also, ob dies durch eine entsprechende Einsparung an Rechenzeit gerechtfertigt werden kann. Dem steht jedoch entgegen, daß die Komplexität der Suche durch die eingeführte Beschränkung nicht wirklich reduziert wird. Sie bleibt im Falle der 3-opt-Nachbarschaft bei $\mathcal{O}(n^3)$ bzw. bei $\mathcal{O}(n^2)$ für die Or -opt-Nachbarschaft mit beschränkten Segmentlängen. Allein die Erhöhung der Kundenzahl um eins hat also (für $k \ll n$) einen wesentlich größeren Einfluß auf den Rechenaufwand als die betrachtete Einschränkung.

Darüber hinaus können Nachbarschaftsschritte, die aufgrund der Ressourcenbeschränkungen unzulässig sind, bei der Verwendung der Lexikographischen Suche effizient ausgeschlossen werden. Sobald ein invertiertes Segment einmal unzulässig geworden ist, kann es durch das Hinzufügen weiterer Knoten nicht wieder zulässig werden. Da die inneren Segmente (und nur diese werden invertiert) Knoten für Knoten erweitert werden, kann die Verlängerung dieses Segmentes abgebrochen werden, wenn es unzulässig ist. Sind also tatsächlich sehr viele Invertierungen von Segmenten wegen der Ressourcenbeschränkungen nicht zulässig, so sinkt bei geeigneter Vorgehensweise auch der Aufwand für die Untersuchung der invertierten k -opt-Typen entsprechend.

Auf Basis dieser Argumentation erscheint die Beschränkung auf einzelne k -opt-Typen für kleine k ($k \leq 4$) wenig sinnvoll. Wird jedoch für k ein größerer Wert gewählt, so entsteht eine neue Situation, weil die Zahl der k -opt-Typen sowohl aufgrund der Zahl der möglichen Permutationen, als auch durch die Invertierungen der Segmente exponentiell mit k wächst.³² Die Beschränkung auf nicht invertierende k -opt-Typen kann dabei helfen, die Zahl der k -opt-Typen etwas länger in einem beherrschbaren Rahmen zu halten.

³²vgl. die Abschätzung auf Seite 41

Kapitel 4

Cyclic-Transfer-Nachbarschaften

Im Kapitel 3 wurde mit den k -opt-Nachbarschaften eine große Klasse kantenorientierter Nachbarschaften behandelt und gezeigt, wie die Lokale Suche in dieser Nachbarschaft effizient auf ein VRSP mit Ressourcenbeschränkungen angewendet werden kann. Um die Qualität der gefundenen Lösungen zu verbessern, wurde versucht, möglichst große k -opt-Nachbarschaften zu untersuchen. Alternativ dazu können zusätzlich andere Nachbarschaften betrachtet werden, die die k -opt-Nachbarschaften gut ergänzen. Besonders interessant sind solche Nachbarschaften, bei denen Optimierungsverfahren wie z.B. Dynamische Programmierung oder Branch-And-Bound für die Suche nach verbessernden Schritten eingesetzt werden können.

Ein Beispiel dafür ist das von Balas und Simonetti in [BS01] beschriebene Verfahren zur Lösung von TSP mit Reihenfolgebeschränkungen. Das auf der Dynamischen Programmierung basierende sehr effiziente Verfahren dient zur (exakten) Optimierung von TSP mit Reihenfolgebeschränkungen. Es kann auch als Heuristik für TSP ohne Beschränkungen eingesetzt werden, indem künstlich Reihenfolgebeschränkungen hinzugefügt werden. Dadurch wird eine sehr große Nachbarschaft für das TSP erhalten, die aber durch das Optimierungsverfahren sehr effizient untersucht werden kann.

Die Autoren betonen darüber hinaus, daß ihre Nachbarschaft für das TSP auch sehr gut mit einer 3-opt-Nachbarschaft kombinierbar ist. Damit verschiedene Nachbarschaften gut zusammenpassen, fordern sie, daß die Zahl der in den verschiedenen Nachbarschaften gleichzeitig enthaltenen Lösungen möglichst klein sein soll. Balas und Simonetti schreiben dazu:

„The promise of combining [in this way] local search procedures based on different neighborhood definitions can be expected to depend on how different – or, mathematically speaking, how close to orthogonal – those neighborhoods are.“¹

In dieser Arbeit bleibt daneben ein wichtiger Aspekt, daß sich eine neue Nachbarschaft auch bei Problemen mit Ressourcenbeschränkungen effizient untersuchen lassen muß.

Gegenstand dieses Kapitels sind deshalb die Cyclic-Transfer-Nachbarschaften, mit denen alle zuvor genannten Ansprüche erfüllt werden können. Die Cyclic-Transfer-Nachbarschaften sind eine Klasse knotenorientierter Nachbarschaften, bei denen mehrere Knoten gleichzeitig zwischen verschiedenen Touren einer Lösung verschoben werden.

Das Verschieben eines einzelnen Knotens (oder äquivalent dazu das Verschieben eines einzelnen Segments) kann i.a. durch einen 3-opt-Schritt der Form „acba“ auf der Giant-Route dargestellt werden. Werden in einem Nachbarschaftsschritt gleichzeitig b einzelne Knoten² verschoben, so ist dieser Schritt i.a. nur in einer k -opt-Nachbarschaft mit $k \geq 3 \cdot b$ enthalten. Dementsprechend sind die meisten Lösungen, die durch das Verschieben von zwei oder mehr Knoten erhalten werden, in keiner k -opt-Nachbarschaft mit $k \leq 5$ enthalten.

Im Unterschied zur k -opt-Nachbarschaft kann die Suche in der CT-Nachbarschaft unter bestimmten Voraussetzungen auf der Basis der Dynamischen Programmierung durchgeführt werden. Dadurch ist es möglich, auch sehr große Nachbarschaften effizient zu untersuchen. Wir werden zeigen, daß die Berücksichtigung von Ressourcenbeschränkungen unabhängig von der eigentlichen Suche nach verbessernden Cyclic-Transfer-Schritten sichergestellt werden kann. Dabei wird auch auf die bereits im Abschnitt 3.2 eingeführte Überprüfung der Ressourcenbeschränkungen für Toursegmente zurückgegriffen.

Das Kapitel 4 ist wie folgt aufgebaut: Zu Beginn wird im Abschnitt 4.1 eine formale Beschreibung der Cyclic Transfers entwickelt. Anschließend wird im Abschnitt 4.2 dargelegt, wie die CT-Nachbarschaft mit Hilfe eines Transfergraphen repräsentiert werden kann. Verbessernde Cyclic Transfers bilden

¹[BS01, S. 27]: Es kann erwartet werden, daß die Erfolgsaussichten des Kombinierens Lokaler Suchprozeduren auf der Basis unterschiedlicher Nachbarschaftsdefinitionen [auf diese Weise], abhängig davon sind, wie verschieden – oder, mathematisch gesprochen, wie orthogonal – diese Nachbarschaften sind.

²Damit sind Knoten gemeint, die in der Ausgangslösung nicht unmittelbar aufeinander folgen und deshalb nicht zu einem Segment zusammengefaßt werden können.

im zugehörigen Transfergraphen stets einen Kreis mit negativer Länge. Im Abschnitt 4.3 werden zwei Ansätze für die Suche nach verbessernden Cyclic-Transfer-Schritten im Transfergraphen beschrieben. Ein großer Teil des Aufwands bei der Suche in einer CT-Nachbarschaft betrifft den Aufbau des Transfergraphen. Im Abschnitt 4.4 wird erklärt, wie durch die Wiederverwendung bereits berechneter Informationen und geeignete Reduzierungen der Nachbarschaften die Effizienz verbessert werden kann. Der abschließende Abschnitt 4.5 beschäftigt sich mit der Idee, Cyclic-Transfer-Schritte für die gezielte Auflösung von Touren und somit zur Reduktion der Tourenzahl eines m-TSPR zu benutzen.

4.1 Formale Beschreibung der Cyclic Transfers

Der Begriff der Cyclic Transfers wurde in [TP93] für Vehicle Routing und Scheduling Probleme eingeführt.³ Bei einem Cyclic Transfer wird eine Teilmenge der jeweiligen Kundenknoten zyklisch zwischen mehreren Touren der Ausgangslösung verschoben. Die Menge aller Lösungen, die sich mittels Cyclic Transfers aus einer zulässigen Lösung y generieren lassen, bildet die CT-Nachbarschaft von y .⁴

Bevor eine formale Beschreibung angegeben wird, soll das abstrakte Konzept der Cyclic Transfers durch das folgende Beispiel veranschaulicht werden.

Beispiel 4.1. Die Abb. 4.1 auf der nächsten Seite zeigt ein Beispiel für einen Cyclic Transfer der Stufe drei. Dabei werden Teilmengen der Kundenknoten zyklisch unter drei verschiedenen Touren weitergegeben.

Jede Tour gibt einen Teil ihrer bisherigen Kunden ab und erhält dafür neue Kunden aus *genau einer* anderen Tour. Im Gegensatz zu der Darstellung in [TP93] wird nicht gefordert, daß die Zahl der abgegeben und erhaltenen Kunden gleich sein muß.

Bei dem Beispiel der Abb. 4.1 gibt die Tour $(o, 1, 2, 3, d)$ die beiden Kunden 2 und 3 ab und erhält dafür den Kunden 9. Die neue Tour ergibt sich durch eine Optimierung der Reihenfolge der jetzt zugehörigen Kunden 9 und 1. In unserem Beispiel entsteht dadurch die Tour $(o, 9, 1, d)$. Die Kunden 2 und 3 werden in die Tour $(o, 4, 5, 6, 7, d)$ eingefügt,

³Die wesentlichen Ideen dieses Artikels wurden bereits im Abschnitt 2.2.1 auf Seite 37 dargestellt.

⁴Cyclic Transfers können auch für andere kombinatorische Probleme formuliert werden, die (analog zum m-TSP) ein Clustering erfordern, z.B. für das Capacitated Minimum Spanning Tree Problem oder das Arc Routing Problem. Dies wird in dieser Arbeit jedoch nicht weiter verfolgt.

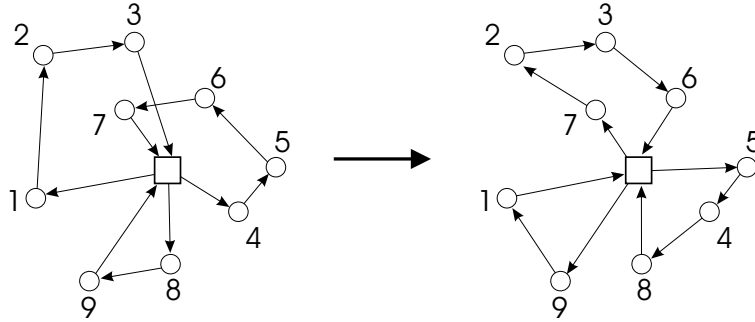


Abbildung 4.1: Beispiel für einen Cyclic Transfer der Stufe drei, der die Knoten $\{2, 3\}$, $\{4, 5\}$ und $\{9\}$ zyklisch zwischen den drei Touren verschiebt.

wobei gleichzeitig die Kunden 4 und 5 weitergeschoben werden. Es entsteht die Tour $(o, 7, 2, 3, 6, d)$. Im dritten Teil des Cyclic Transfers werden die Kunden 4 und 5 in die ursprüngliche Tour $(o, 8, 9, d)$ eingefügt, die den Kunden 9 ihrerseits bereits abgegeben hat. Es entsteht die Tour $(o, 5, 4, 8, d)$.

Um die Beschreibung der Cyclic Transfers zu vereinfachen, vereinbaren wir die folgenden Bezeichnungen und Schreibweisen:

Das zugrundeliegende Problem $\mathcal{P}$ sei ein m-TSP oder ein m-TSPR.

$G = (V, A, c)$ sei der zugehörige gerichtete Graph von $\mathcal{P}$ mit $V = N \cup D$. Dabei ist $N = \{1, \dots, n\}$ die Menge der Kundenknoten und $D = \{o, d\}$ die Menge der Depotknoten.

Eine Tour t ist ein Pfad in G , der am Depotausgang o beginnt und am Depoteingang d endet. Eine Menge x von m Touren heißt Lösung von $\mathcal{P}$, wenn gilt: $\bigcup_{t \in x} V(t) = V$ und $V(t_i) \cap V(t_j) = D, \forall t_i, t_j \in x$. Die Menge aller Lösungen von $\mathcal{P}$ wird mit X bezeichnet. Eine Lösung $x \in X$ erfüllt i.a. nicht die Ressourcenbeschränkungen.

Die Teilmenge $Y \subseteq X$ bezeichnet die Menge aller zulässigen Lösungen von $\mathcal{P}$. Sei $\tilde{X} \subseteq X$, so bezeichnet $T(\tilde{X})$ die Menge aller Touren von $\tilde{X}$: $T(\tilde{X}) = \bigcup_{x \in \tilde{X}} x$.

Die Durchführung eines Cyclic-Transfer-Schritts soll analog zur Definition der k -opt-Schritte durch die Verknüpfung mehrerer einfacher Operatoren beschrieben werden. Da ein Cyclic Transfer das gleichzeitige Verschieben von Kundenmengen zwischen verschiedenen Touren beinhaltet, benötigen wir eine Beschreibung für die „Kundenteilmengen einer Tour“, einen Operator zum

Entfernen einer Kundenteilmenge aus einer Tour und einen Operator zum Einfügen zusätzlicher Kunden in eine Tour.

Definition 4.1. (Transfermenge $\mathcal{S}$)

Es seien $x = \{t_1, \dots, t_m\} \in X$, $\tilde{X} \subseteq X$ und $i \in \{1, \dots, m\}$. Dann heißt jedes Element S der Menge

$$\mathcal{S}(t_i) = \text{Pot}(V(t_i) \setminus D)$$

eine Transfermenge von t_i .

Die Elemente der Mengen $\mathcal{S}(x) = \bigcup_{i=1}^m \mathcal{S}(t_i)$ bzw. $\mathcal{S}(\tilde{X}) = \bigcup_{x \in \tilde{X}} \mathcal{S}(x)$ heißen auch Transfermengen von x bzw. Transfermengen von $\tilde{X}$.

Definition 4.2. (Enthaltende Tour $t(S)$)

Sei $t \in T(X)$ eine zulässige Tour von $\mathcal{P}$ und $S \in \mathcal{S}(t)$ eine Transfermenge von t , so ist $t(S) = t$ diejenige Tour, aus der die Teilmenge S gebildet wurde.

Bemerkung 4.3. Man stelle sich vor, daß die Transfermengen $S \in \mathcal{S}(t_i)$ einer Tour t_i jeweils mit der Tour t_i gekennzeichnet sind. Dadurch ist es z.B. auch möglich, die zugehörige Tour einer leeren Transfermenge zu bestimmen. Es gibt also zu jeder Tour eine zugehörige leere Transfermenge, und die leeren Transfermengen von unterschiedlichen Touren sind voneinander unterscheidbar. Die leere Transfermenge einer Tour $t_i \in x$ sei als $S_0(t_i) = S_{0i}$ bezeichnet. Seien $t_i, t_j \in x$ und $t_i \neq t_j$, so sind die zugehörigen leeren Transfermengen S_{0i} und S_{0j} voneinander unterscheidbar, weil $t(S_{0i}) \neq t(S_{0j})$. $\mathcal{S}_0(x) = \{S_0(t) : t \in x\}$ bezeichnet die Menge aller leeren Transfermengen.

Definition 4.4. (Minus-Operator $\ominus$)

Eine Abbildung $\ominus : T(X) \times \mathcal{S}(X) \rightarrow T(X)$, $(t, S) \mapsto t'$ mit der Eigenschaft

$$V(t') = V(t) \setminus S \tag{4.1}$$

heißt Minus-Operator.

Definition 4.5. (Plus-Operator $\boxplus$)

Eine Abbildung $\boxplus : \mathcal{S}(X) \times T(X) \rightarrow T(X)$, $(S, t) \mapsto t'$ mit der Eigenschaft

$$V(t') = V(t) \cup S \tag{4.2}$$

heißt Plus-Operator.

Bemerkung 4.6. Die Operatoren $\boxminus$ und $\boxplus$ sind durch ihre Definition nicht eindeutig festgelegt, denn beide Operatoren können die Reihenfolge der Kundenknoten in der Tour t' beliebig festlegen. Dies ist auch beabsichtigt, denn $\boxminus$ und $\boxplus$ ermöglichen so eine generische Beschreibung einer Tour-Transformation. Für die Festlegung eines konkreten Cyclic-Transfer-Schritts müssen die beiden Operatoren noch genauer spezifiziert werden.

Durch das Verknüpfen der Operatoren $\boxminus$ und $\boxplus$ erhält man einen neuen Operator. Mit dessen Hilfe läßt sich die Transformation einer Tour, die diese im Rahmen eines Cyclic Transfers erfährt, vollständig repräsentieren. Da dieser Operator eine Kundenteilmenge einer Tour gegen eine Kundenteilmenge einer anderen Tour austauscht, bezeichnen wir ihn als Austauschoperator.

Definition 4.7. (Austauschoperator $\boxminus\boxplus$)

Es seien $\boxminus$ ein Minus-Operator und $\boxplus$ ein Plus-Operator. Dann heißt eine Abbildung $\boxminus\boxplus : \mathcal{S}(X) \times T(X) \times \mathcal{S}(X) \rightarrow T(X)$, $(S, t, S') \mapsto t' = \boxminus\boxplus S, t, S'$ mit der Eigenschaft

$$t' = \boxplus(S, \boxminus(t, S')) \quad (4.3)$$

Austauschoperator auf der Basis von $\boxminus$ und $\boxplus$.

Gegebenenfalls ist die explizite Angabe der Operatoren $\boxminus$ und $\boxplus$ nicht notwendig. Dann kann an Stelle von $\boxminus\boxplus$ auch nur $\curvearrowright$ geschrieben werden.

Ist $t = t(S')$, so kann auf die Angabe von t verzichtet und die Kurzschreibweise

$$\curvearrowright (S, S') = \curvearrowright (S, t(S'), S') \quad \text{bzw.} \quad S \curvearrowright S' = \curvearrowright (S, t(S'), S')$$

verwendet werden.

Bemerkung 4.8. Es sei explizit darauf hingewiesen, daß der Ausdruck $S \curvearrowright S'$ eine Tour beschreibt. Da die durch den Operator modifizierte Tour $t(S')$ bei der Kurzschreibweise nicht mehr als Argument auftaucht, ist dies nicht mehr so offensichtlich. Gerade diese Darstellung entspricht jedoch der gewählten Repräsentation von Austauschoperatoren als Bögen eines Transfergraphen im nächsten Abschnitt und ist daher für die weiteren Ausführungen gut geeignet.

Jetzt sind alle Vorbereitungen getroffen, um die Definition eines Cyclic Transfers der Stufe b anzugeben. Dieser verschiebt b Transformengen zyklisch zwischen den zugehörigen (paarweise verschiedenen) Touren einer Lösung.

Definition 4.9. (Cyclic Transfer der Stufe b)

Es seien $x \in X$ eine Lösung mit m Touren, $b \in \{2, \dots, m\}$ und $S_1, \dots, S_b \in \mathcal{S}(x)$.

Eine partielle Abbildung $ct^{b, \boxplus} : X \times \mathcal{S}(X)^b \dashrightarrow X$, $(x, (S_1, \dots, S_b)) \mapsto x' = ct^{b, \boxplus}(x, (S_1, \dots, S_b))$ heißt ein Cyclic Transfer der Stufe b mit den Operatoren $\boxminus$ und $\boxplus$, falls es eine b -elementige Tourmenge $\{t_1, \dots, t_b\} \subseteq x$ gibt, so daß gilt:

$$\begin{aligned} S_i &\in \mathcal{S}(t_i), \forall i \in \{1, \dots, b\} \\ t'_i &= \boxminus_{\curvearrowright} (S_{i-1}, t_i, S_i), \forall i \in \{2, \dots, b\} \text{ und } t'_1 = \boxminus_{\curvearrowright} (S_b, t_1, S_1), \\ x' &= (x \setminus \{t_1, \dots, t_b\}) \cup \{t'_1, \dots, t'_b\}, \\ x' &\in X. \end{aligned}$$

Bemerkung 4.10. Um einen CT zu spezifizieren, genügt es entsprechend der obigen Definition, eine Ausgangslösung x und ein Tupel der Transfermengen $(S_1, \dots, S_b)$ anzugeben, die getauscht werden sollen. Die zugehörigen Touren $t_1, \dots, t_b$ ergeben sich dann zwingend. Allerdings müssen die Transfermengen in paarweise verschiedenen Touren der Lösung x enthalten sein, damit der zugehörige Cyclic Transfer existiert.

Beispiel 4.2. (Fortsetzung von Beispiel 4.1)

Als Beispiel betrachten wir noch einmal die Abb. 4.1 auf Seite 124, die einen Cyclic Transfer der Stufe drei zeigt. Die Ausgangslösung (links) besteht aus den Touren $t_1 = (o, 1, 2, 3, d)$, $t_2 = (o, 4, 5, 6, 7, d)$ und $t_3 = (o, 8, 9, d)$.

Durch den Cyclic Transfer $ct^3(\{t_1, t_2, t_3\}, (\{2, 3\}, \{4, 5\}, \{9\}))$ entsteht die Lösung $\{t'_1, t'_2, t'_3\}$ (rechts) mit $t'_1 = (o, 9, 1, d)$, $t'_2 = (o, 7, 2, 3, 6, d)$ und $t'_3 = (o, 5, 4, 8, d)$.

Definition 4.11. (Cyclic-Transfer-Nachbarschaft $\mathcal{N}_{ct}$)

Es sei $x \in X$ eine Lösung, die aus m Touren besteht. Dann heißt die Menge aller zulässigen Lösungen, die durch einen Cyclic Transfer ct aus x hervorgehen, Cyclic-Transfer-Nachbarschaft von x :

$$\mathcal{N}_{ct}(x) = Y \cap \{x' : \exists S_1, \dots, S_b, 2 \leq b \leq m \text{ mit } x' = ct^b(x, (S_1, \dots, S_b))\}.$$

4.2 Aufbau eines Transfergraphen

In [AOS01] betrachten Ahuja et. al. die Suche nach verbessernden „multi-exchange“-Schritten beim Capacitated Minimum Spanning Tree Problem. Sie

zeigen, wie diese Suche auf ein Kürzeste-Wege-Problem mit Nebenbedingungen in einem „improvement graph“ zurückgeführt werden kann. Die „multi-exchange“-Schritte korrespondieren mit den Cyclic-Transfer-Schritten beim m-TSP. In diesem Abschnitt wird beschrieben, wie die Idee von Ahuja et. al. auf die hier formulierten wesentlich allgemeineren Cyclic Transfers übertragen werden kann.

4.2.1 Repräsentation von Cyclic Transfers im Transfergraphen

Die Repräsentation von Cyclic Transfers in einem Graphen kann dadurch motiviert werden, daß alle Cyclic Transfers aus einzelnen Austauschoperationen zusammengesetzt werden können. Dabei müssen unmittelbar aufeinanderfolgende Austauschoperationen zueinander passen, d.h. die Kundenmenge, die bei dem vorangehenden Austausch aus einer Tour herausgelöst wird, muß durch den nachfolgenden Austausch in die nächste Tour eingebaut werden. Diese Abhängigkeit kann in einem gerichteten Graphen abgebildet werden, den wir Transfergraph nennen.

Darüber hinaus soll es keine weiteren Abhängigkeiten zwischen den verschiedenen an einem Cyclic Transfer beteiligten Touren geben. Das heißt insbesondere, daß die Kosten und die Zulässigkeit einer Tour von $x \in X$ nur von der Tour selbst, nicht aber von der Beschaffenheit der übrigen Touren von x abhängig sein sollen. Wir bezeichnen diese Eigenschaft als „Tour-Separabilität“ und setzen sie im weiteren als grundsätzlich gegeben voraus.

Definition 4.12. (Tour-Separabilität)

Das Problem $\mathcal{P}$ heißt *tour-separable*, wenn für jede Lösung $x \in X$ von $\mathcal{P}$ mit $t_i, t_j \in T(x)$, $V(t_i) \cap V(t_j) = \emptyset$ gilt:

1. die Zulässigkeit von Tour t_i ist unabhängig von t_j
2. die Kosten $c(t_i)$ der Tour t_i sind unabhängig von t_j
3. die Kosten von x lassen sich darstellen als

$$c(x) = \sum_{t \in x} c(t). \quad (4.5)$$

Definition 4.13. (Transfergraph G^T)

Es seien $y \in Y$ gegeben und die Operatoren $\boxminus$ und $\boxplus$ spezifiziert. Dann heißt

der gerichtete bewertete Graph $G^T(y) = (V^T, A^T, c^T)$ Transfergraph von y , falls gilt:

$$V^T = \mathcal{S}(y) \quad (4.6a)$$

$$A^T = \{(S_i, S_j) : S_i, S_j \in \mathcal{S}(y), t(S_i) \neq t(S_j) \text{ mit} \\ S_i \stackrel{\boxplus}{\curvearrowright} S_j \in T(Y)\} \quad (4.6b)$$

$$c^T((S_i, S_j)) = c(S_i \stackrel{\boxplus}{\curvearrowright} S_j) - c(t_j) \quad (4.6c)$$

Jeder Bogen (S_i, S_j) des Transfergraphen $G^T(y)$ kann als Repräsentant der Austauschoperation $S_i \curvearrowright S_j$ betrachtet werden. Im Ergebnis des Austausches muß eine zulässige Tour $t' = S_i \curvearrowright S_j \in T(Y)$ entstehen. Die Kosten des Bogens (S_i, S_j) geben die Änderung der Kosten von Tour t_j an, wenn der Austauschoperator $S_i \curvearrowright S_j$ angewendet wird.

Ein Cyclic Transfer der Stufe b wird im Transfergraphen $G^T(y)$ durch einen Kreis $(S_1, \dots, S_b, S_1)$ der Länge b dargestellt. Die von dem Kreis berührten Knoten von G^T geben dabei die transferierten Kundenteilmengen der am Cyclic Transfer beteiligten Touren von y an.

Man beachte, daß nicht jeder Kreis in G^T einen Cyclic Transfer repräsentiert. Für einen Cyclic Transfer muß zusätzlich gelten, daß die von dem Kreis berührten Knoten $S_1, \dots, S_b$ nur solche Kundenteilmengen repräsentieren, deren zugehörige Touren $t(S_i)$ paarweise verschieden sind. Zur Vereinfachung der Sprechweise wird diese zusätzliche Forderung nachfolgend als CT-Konsistenz bezeichnet.

Beispiel 4.3. Als Beispiel betrachten wir den (nicht vollständigen) Transfergraphen aus Abb. 4.2. Darin beschreiben die Kreise (S_1, S_3, S_1) , $(S_1, S_3, S_6, S_7, S_1)$ oder (S_2, S_4, S_6, S_2) zulässige Cyclic-Transfer-Schritte, d.h. diese Kreise erfüllen die CT-Konsistenz.

Dagegen repräsentieren die Kreise $(S_3, S_6, S_7, S_5, S_3)$ oder $(S_2, S_4, S_6, S_3, S_2)$ keine Cyclic-Transfer-Schritte, da die Tour t_3 bzw. die Tour t_2 durch diese Kreise zweifach berührt werden.

4.2.2 Behandlung von leeren Kundenteilmengen

Entsprechend der Definition 4.9 können bei einem Cyclic Transfer auch leere Teilmengen zwischen den beteiligten Touren verschoben werden. Nur dadurch wird es möglich, im Rahmen eines Cyclic-Transfer-Schritts alle Kundenknoten aus einer Tour zu entfernen und damit diese Tour faktisch aufzulösen.

Leere Kundenteilmengen können im Transfergraphen prinzipiell genauso behandelt werden wie alle übrigen (nicht-leeren) Teilmengen $S \in \mathcal{S}(y)$ auch.

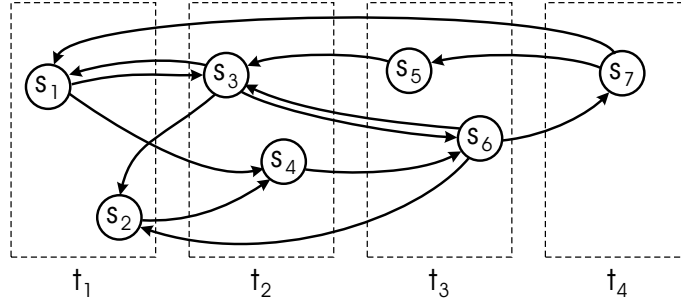


Abbildung 4.2: Beispiel für einen nicht vollständigen Transfergraphen

Durch das Einfügen eines zusätzlichen Knotens S_{0*} kann jedoch die Zahl der notwendigen Bögen im Transfergraphen G^T reduziert werden. Dazu werden zwei spezielle Eigenschaften der leeren Transfermengen genutzt.

Für alle Touren $t_i \in T(y)$ bezeichne $S_{0i} \in \mathcal{S}(t_i)$ die zugehörige leere Transfermenge. Sind t_i, t_j und t_k paarweise verschiedene Touren von y und $S_k \in \mathcal{S}(t_k)$, dann gilt:

1. Bögen zwischen leeren Transfermengen, d.h. Bögen der Form (S_{0i}, S_{0j}) können ausgeschlossen werden, da der zugehörige Austausch $S_{0i} \curvearrowright S_{0j}$ weder Kunden aus der Tour t_j entfernt noch neue Kunden einfügt. Die Tour t_j bleibt deshalb unverändert.⁵
2. Es gilt stets

$$S_{0i} \curvearrowright S_k = S_{0j} \curvearrowright S_k,$$

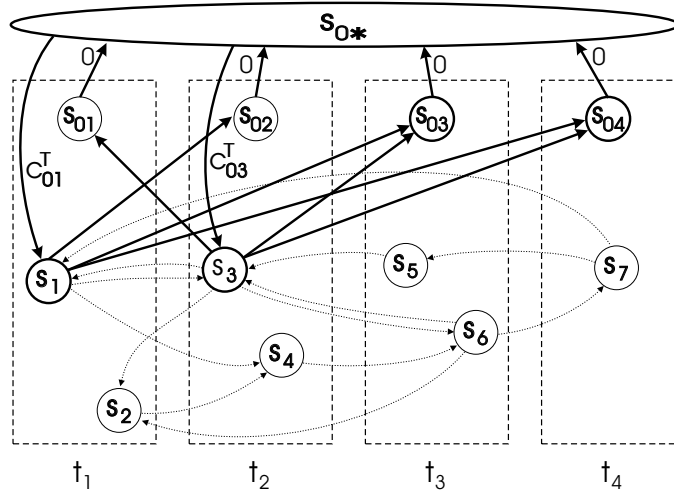
d.h. es ist für die Modifikation von Tour t_k irrelevant, welche leere Transfermenge in diese Tour eingefügt wird.

Der Transfergraph G^T kann deshalb wie folgt äquivalent transformiert werden (vgl. Abb. 4.3):

- füge einen zusätzlichen Knoten S_{0*} ein,

⁵Dies stimmt genau genommen nur dann, wenn die bei den Cyclic Transfers verwendeten Operatoren $\boxminus$ und $\boxplus$ die betroffene Tour tatsächlich nicht ändern, sofern die einzufügende bzw. die zu entfernende Transfermenge leer sind. Dies wäre z.B. nicht unbedingt der Fall, wenn diese Operatoren die betroffene Tour reoptimieren.

⁶Zur besseren Übersichtlichkeit wurden nur Bögen von und zu den Knoten S_1 und S_3 bzw. die kostenfreien Bögen von den Knoten $S_{01}, \dots, S_{04}$ eingezeichnet.


 Abbildung 4.3: Transfergraph⁶ mit zusätzlichem Knoten S_{0*}

- entferne alle Bögen der Form (S_{0i}, S_k) ,
- füge alle Bögen der Form (S_{0i}, S_{0*}) mit Kosten c^T gleich null ein,
- füge alle Bögen der Form (S_{0*}, S_k) mit den Kosten

$$c_{0k}^T = c(S_{0i} \curvearrowright S_k) - c(t_k)$$

ein, dabei ist S_{0i} eine beliebige leere Transfermenge.

Dieser Graph enthält $|\mathcal{S}(y)|$ Bögen von und zum Knoten S_{0*} . Dafür entfallen alle Bögen der Form (S_{0i}, S_j) mit $t(S_{0i}) \neq t(S_j)$ und $S_j \notin \mathcal{S}_0(y)$.

4.2.3 Auswahl der Transfermengen

Alle bisherigen Überlegungen betrafen die komplette CT-Nachbarschaft. Diese Nachbarschaft ist aber sehr groß. Der zugehörige Transfergraph besitzt $|V^T| = |\mathcal{S}(y)| = \sum_{t \in T(y)} |\mathcal{S}(t)|$ Knoten. Eine Tour mit l Kundenknoten besitzt insgesamt 2^l Transfermengen. Angenommen, die Lösung y besteht aus den m Touren $t_1, \dots, t_m$ und jede Tour t_i besitzt ℓ_i Kundenknoten ($i = 1, \dots, m$), dann ist die Zahl der Knoten von G^T

$$|V^T| = \sum_{i=1}^m 2^{\ell_i}. \quad (4.7)$$

Wegen (4.7) kann insbesondere die Existenz einzelner langer Touren zu einer sehr großen Knotenzahl im Transfergraphen führen. Der Aufbau eines so großen Graphen einschließlich der Bestimmung der Bögen und deren Kosten würde bereits bei Problemen mit moderater Kundenzahl einen sehr hohen Aufwand bedeuten.

Es ist deshalb sinnvoll, sich beim Aufbau des Transfergraphen auf bestimmte Transfermengen zu beschränken, aus denen die Knotenmenge V^T des Transfergraphen zusammengesetzt werden soll. Sobald die Knotenmenge V^T festgelegt ist, ergeben sich die Bögen des Transfergraphen und deren Kosten durch die Anwendung der entsprechenden Austauschoperatoren. Mit einer Vorschrift, die festlegt, welche Kundenteilmengen in die Knotenmenge V^T eingehen, wird daher implizit ein reduzierter Transfergraph und somit eine zugehörige reduzierte CT-Nachbarschaft $\mathcal{N}_{ct}^{\text{red}}(V^T) \subseteq \mathcal{N}_{ct}$ spezifiziert.

Prinzipiell lassen sich sehr viele solche Vorschriften modellieren. Wir betrachten Transfergraphen, deren Knoten die Transfermengen folgender Typen repräsentieren:

- einzelne Kundenknoten,
- Toursegmente fester Länge,
- Toursegmente, die mit dem ersten Kundenknoten der Tour beginnen,
- Toursegmente, die mit dem letzten Kundenknoten der Tour aufhören, und
- Kombinationen der vorgenannten Typen.

Einzelne Knoten und Segmente fester Länge

Eine naheliegende Idee ist, nur einelementige Transfermengen im Transfergraphen zu berücksichtigen. Sei also $t \in T(y)$ und bezeichne

$$\mathcal{S}_1(t) = \{S : S \in \mathcal{S}(t) \text{ und } |S| = 1\}$$

die Menge aller einelementigen Transfermengen von t . Dann ist

$$\mathcal{S}_1(y) = \bigcup_{t \in T(y)} \mathcal{S}_1(t)$$

die Menge aller einelementigen Transfermengen von y .

Wird für die Bildung des Transfergraphen G^T anstelle der Menge $\mathcal{S}(y)$ die Menge $\mathcal{S}_1(y)$ zugrunde gelegt, so reduziert sich die Zahl der Knoten von G^T auf $|\mathcal{S}_1(y)| = n$. Die zugehörige CT-Nachbarschaft umfaßt die Menge aller 1-Transfers⁷ und somit u.a. alle Exchange-Schritte, die als 1-Transfers der Stufe 2 aufgefaßt werden können.

Werden zusätzlich zu $\mathcal{S}_1(y)$ auch die m leeren Teilmengen $\mathcal{S}_{0i} \in \mathcal{S}_0(y)$ in den Transfergraphen aufgenommen, so können auch Relocation-Schritte dargestellt werden. Die CT-Nachbarschaft umfaßt dann u.a. alle λ -Interchange-Schritte für $\lambda = 1$ für alle Tourpaare aus $T(y)$. Darüber hinaus können auch einige Node Ejection Chains⁸ gefunden werden, die maximal einen Knoten je Tour verschieben.⁹

Da man einen einzelnen Kunden immer auch als ein Toursegment mit einem Kunden auffassen kann, repräsentiert die Menge $\mathcal{S}_1(y)$ alle Toursegmente, die genau einen Kundenknoten beinhalten. Analog dazu können allgemein Toursegmente mit k aufeinanderfolgenden Kundenknoten berücksichtigt werden. Die entsprechenden Teilmengen von $\mathcal{S}(y)$ nennen wir $\mathcal{S}_1(y)$, $\mathcal{S}_2(y), \dots, \mathcal{S}_{l_{max}}(y)$. Dabei gibt l_{max} die maximale Kundenzahl einer Tour von y an. Jede dieser Mengen besitzt höchstens $n - k + 1$ Elemente.

Anfangs- und Endsegmente von Touren

Anstelle von Segmenten fester Länge können mit gleichem Aufwand auch Toursegmente mit festem Anfang oder festem Ende als Kundenteilmengen herangezogen werden.

Sei $t \in T(y)$ eine Tour der zulässigen Lösung y . Dann bezeichnet $\mathcal{S}_{start}(t)$ die Menge aller „Startsegmente“ von t , d.h. die Menge aller Segmente, die mit dem ersten Kundenknoten von t beginnen. Analog bezeichne $\mathcal{S}_{end}(t)$ die Menge aller „Endsegmente“ von t . Die Mengen der Startsegmente bzw. End-

⁷Thompson und Psaraftis nennen Cyclic Transfers, die jeweils m Knoten zwischen b Touren weiterschieben, einen „ b -cyclic m -transfer“ (vgl. Abschnitt 2.2.1, Seite 37f.). Die Menge aller 1-Transfers umfaßt dementsprechend alle Cyclic Transfers beliebiger Stufe b , die jeweils genau einen Knoten zwischen den Touren verschieben.

⁸Node Ejection Chains wurden im Abschnitt 2.2.1 auf 36f. beschrieben.

⁹Welche Nachbarschaftsschritte tatsächlich im Transfergraphen repräsentiert werden, ist genaugenommen davon abhängig, wie die Austauschoperatoren $\boxplus$ gewählt werden. Darauf wird im Unterabschnitt 4.2.4 ausführlich eingegangen.

segmente von y sind

$$\mathcal{S}_{start}(y) = \bigcup_{t \in T(y)} \mathcal{S}_{start}(t) \quad \text{und} \quad \mathcal{S}_{end}(y) = \bigcup_{t \in T(y)} \mathcal{S}_{end}(t)$$

Da es in jeder Tour genauso viele Startsegmente bzw. Endsegmente gibt, wie die Tour Kundenknoten enthält, gilt:

$$|\mathcal{S}_{start}(y)| = \sum_{t \in T(y)} |\mathcal{S}_{start}(t)| = n = \sum_{t \in T(y)} |\mathcal{S}_{end}(t)| = |\mathcal{S}_{end}(y)| \quad (4.8)$$

Durch die Mengen $\mathcal{S}_{start}(y)$ (bzw. $\mathcal{S}_{end}(y)$) wird eine CT-Nachbarschaft definiert, die fast die gesamte 2-opt*-Nachbarschaft beinhaltet.¹⁰ Dies wird sofort plausibel, wenn man sich bewußt macht, daß durch einen 2-opt*-Schritt jeweils die Anfangsstücke (bzw. äquivalent dazu die Endstücke) zweier verschiedener Touren gegeneinander getauscht werden (siehe Abb. 2.11 auf Seite 49). Darüber hinaus sind jedoch auch komplexere Nachbarschaftsschritte möglich. So können z.B. drei oder mehr Anfangsstücke zyklisch miteinander getauscht werden (Abb. 4.4).

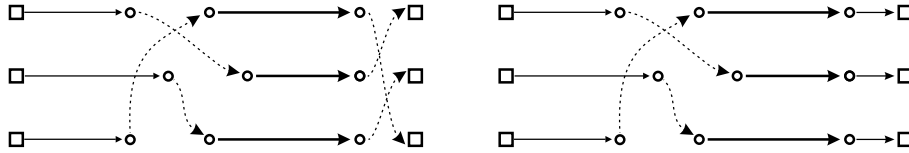


Abbildung 4.4: Zyklisches Vertauschen von drei Tourendstücken (links). Die Endstücke sind durch eine dicke Linie dargestellt. Dieser Schritt ist zu dem 3-opt-Schritt auf der rechten Seite äquivalent.

Da die 2-opt*-Nachbarschaft besonders für Probleme mit Zeitfenstern u.ä. Restriktionen sehr interessant ist, trifft dies auch für CT-Nachbarschaften zu, deren Transfergraph Knoten aus $\mathcal{S}_{start}$ oder $\mathcal{S}_{end}$ beinhaltet.

Zu jeder Tour gibt es ein Startsegment, das alle Kundenknoten der Tour umfaßt. Deshalb kann auch eine komplette Tour in eine andere Tour eingebaut werden. Wird der Transfergraph über der Knotenmenge $V^T = \mathcal{S}_{start}(y) \cup \mathcal{S}_0(y)$ aufgebaut, so lassen sich durch die Cyclic-Transfer-Schritte auch Touren auflösen.

¹⁰Um die komplette 2-opt*-Nachbarschaft abzudecken, müssen die Knoten von $\mathcal{S}_0$ zu V^T hinzugenommen werden.

Kombinationen

Prinzipiell besteht die Möglichkeit, für die Knoten V^T des Transfergraphen eine beliebige Teilmenge von $\mathcal{S}(y)$ zu wählen. So kann problemlos eine große Zahl unterschiedlicher CT-Nachbarschaften erzeugt werden. Verschiedene solcher Nachbarschaften können miteinander kombiniert werden, indem im Sinne einer Variable-Neighborhood-Descent-Heuristik zwischen den verschiedenen Nachbarschaften gewechselt wird. Wenn die einzelnen Nachbarschaften schnell und effizient abgesucht werden können, bleibt der Rechenaufwand dabei relativ gering.

Eine einfache Möglichkeit, kompliziertere Knotenmengen zu beschreiben, besteht darin, mehrere kleine Knotenmengen miteinander zu vereinigen. Diese Möglichkeit wurde bereits für die Menge der $\mathcal{S}_0(y)$ der leeren Kundenteilmengen angedeutet. An dieser Stelle soll nur auf einige Beispiele eingegangen werden:

1. Vereinigung aller Toursegmente bis zu einer maximalen Länge von l Kundenknoten: $V^T = \bigcup_{i=0}^l \mathcal{S}_i(y)$.
Wird z.B. $l = 3$ gewählt, beinhaltet die reduzierte CT-Nachbarschaft $\mathcal{N}_{ct}^{\text{red}}(V^T)$ u.a. alle *Or-opt*-Schritte zwischen mehreren Touren.
2. Vereinigung aller Start- und Endsegmente: $V^T = \mathcal{S}_{\text{start}}(y) \cup \mathcal{S}_{\text{end}}(y)$.
Die zugehörige Nachbarschaft erlaubt auch Schritte, die ein Startsegment gegen ein Endsegment tauscht und umgekehrt. Diese Nachbarschaft ist z.B. für das VRP interessant, besonders wenn zusätzlich auch die inversen Start- und Endstücke berücksichtigt werden.
3. Vereinigungen mit der Menge $\mathcal{S}_0$:
Allein auf der Basis von $\mathcal{S}_0(y)$ läßt sich zwar keine sinnvolle CT-Nachbarschaft aufbauen, in Vereinigung mit anderen Knotenmengen schafft die Menge $\mathcal{S}_0(y)$ jedoch die Möglichkeit, in eine Tour ausschließlich Kunden einzufügen bzw. aus einer Tour ausschließlich Kunden entfernen zu können. Dadurch wird es u.a. möglich, eine leere Tour zu erzeugen und damit faktisch aufzulösen.

Durch die Vereinigung verschiedener Teilmengen $\mathcal{S}_i(y) \subset \mathcal{S}(y)$ können in der zugehörigen CT-Nachbarschaft auch Verbesserungen gefunden werden, die durch die sequentielle Betrachtung der einzelnen Nachbarschaften $\mathcal{N}_{ct}^{\text{red}}(\mathcal{S}_i(y))$ nicht gefunden werden können. Allerdings wächst sowohl der Aufwand für die Konstruktion des Transfergraphen G^T als auch der Aufwand

für die Suche nach verbessernden Cyclic Transfers in G^T relativ schnell. Dies hängt vor allem daran, daß die Zahl der Bögen im Graphen superlinear mit der Zahl der Knoten wächst (im ungünstigsten Fall ist der Transfergraph ein vollständiger Graph, dann ist $|A^T| = |V^T|^2$). Die Wahl geeigneter Knotenmengen V^T sollte man deshalb von der verfügbaren Rechenzeit, der Größe und weiteren Eigenschaften des Problems (z.B. Existenz enger Zeitfenster) abhängig machen.

4.2.4 Wahl des Austauschoperators

Ein Cyclic Transfers der Stufe b bewirkt die b -fache Ausführung des Austauschoperators $\boxplus$, der seinerseits aus den Operatoren $\boxminus$ und $\boxplus$ aufgebaut ist. Solange die Wirkungsweise der Operatoren $\boxminus$ und $\boxplus$ nicht eindeutig festgelegt ist, ist die Definition der Cyclic Transfers eine generische Definition (vgl. Bemerkung 4.6 und Definition 4.9).

Gegenstand dieses Unterabschnitts ist die Bestimmung geeigneter Operatoren $\boxminus$ und $\boxplus$, die den Austauschoperator $\boxplus$ und die darauf aufbauenden Cyclic-Transfer-Schritte festlegen.

Allgemeine Überlegungen

Die Aufgabe des Austauschoperators $S_i \boxplus S_j$ ist es, aus der bestehenden zulässigen Tour t_j eine neue zulässige Tour zu konstruieren. Dabei muß die vorgegebene Transfermenge S_j entfernt und die Transfermenge S_i eingefügt werden. Die Durchführung der Austauschoperationen beim Aufbau des Transfergraphen ist die einzige Stelle, bei der neue Touren konstruiert werden.

Bisher war es deshalb auch nicht notwendig, auf die Berücksichtigung von beschränkten Ressourcen einzugehen. Bei der Realisierung des Austauschoperators muß jedoch berücksichtigt werden, ob das vorliegende Problem $\mathcal{P}$ ein m-TSP oder ein m-TSPR ist. Der Möglichkeit, die gegebenen Ressourcenbeschränkungen effizient berücksichtigen zu können, muß und kann ausschließlich bei der Wahl des Austauschoperators Rechnung getragen werden.

Die Eigenschaften des Austauschoperators haben wesentlichen Einfluß auf die Qualität der Lösungen innerhalb der CT-Nachbarschaft und die Geschwindigkeit, mit der der zugehörige Transfergraph gebildet werden kann. Einerseits müssen die Austauschoperatoren beim Aufbau des Transfergraphen sehr oft ausgeführt werden. Soll mit Hilfe der Cyclic Transfers eine effiziente Heuristik entwickelt werden, darf dies jedoch nicht zu viel Zeit ko-

sten. Andererseits wird die Qualität der im Transfergraphen repräsentierten Nachbarlösungen i.d.R. wesentlich davon abhängen, wie gut die durch den Austauschoperator erzeugten Touren sind. Bei der Wahl des Austauschoperators ist deshalb zwischen den beiden Zielen Qualität und Geschwindigkeit abzuwägen.

Bei einem m-TSPR ist ggf. auch die Anzahl der gebildeten zulässigen Touren davon abhängig, wie gut der Austauschoperator in der Lage ist, aus der neu zusammengesetzten Kundenmenge eine zulässige (und kostengünstige) Tour zu bilden. Wenn dies nur selten gelingt, ist der Transfergraph nur dünn besetzt, und es bestehen geringe Aussichten, einen verbessernden Cyclic Transfer darin zu finden.

Prinzipiell ist es denkbar, daß innerhalb eines einzelnen Cyclic-Transfer-Schritts verschiedene Austauschoperatoren eingesetzt werden.¹¹ Dies wird im weiteren jedoch nicht berücksichtigt. Alle folgenden Betrachtungen beschränken sich auf CT-Nachbarschaften, bei denen alle Austauschoperatoren gleich sind. Jeder Cyclic Transfer der Stufe b wendet ein und denselben Austauschoperator b -fach an. Im Transfergraphen repräsentieren dementsprechend alle Bögen den gleichen Austauschoperator.

Schema eines heuristischen Austauschoperators

Für eine genauere Beschreibung seien folgende Bezeichnungen vereinbart: $t_i, t_j \in T(Y)$, $S_i \in \mathcal{S}(t_i)$, $S_j \in \mathcal{S}(t_j)$ und $t'_j = S_i \boxminus \boxplus S_j$.

Der Austauschoperator $S_i \boxminus \boxplus S_j$ muß aus den Knoten $(V(t_j) \setminus S_j) \cup S_i$ eine zulässige Tour t'_j erzeugen. Die Suche nach einer kostenminimalen Tour führt i.a. auf ein TSPR, das als $\mathcal{P}_{ij}$ bezeichnet werden soll. Der qualitativ beste Austauschoperator ist zweifelsohne ein Optimierungsalgorithmus, der das Problem $\mathcal{P}_{ij}$ lösen kann. Da das TSPR aber ein $\mathcal{NP}$ -vollständiges Problem ist, ist ein solcher Operator praktisch jedoch nur dann einsetzbar, wenn die Zahl der Knoten $|V(t'_j)| = |V(t_j)| - |S_j| + |S_i|$ klein ist.

Für die Konstruktion längerer Touren t'_j ist es deshalb notwendig, eine Heuristik zur Lösung von $\mathcal{P}_{ij}$ einzusetzen. Prinzipiell kann dafür aus der

¹¹Die Darstellung im Transfergraphen würde es erlauben, gleichzeitig mehrere verschiedene Austauschoperatoren zu repräsentieren. Ein Knoten des Transfergraphen wäre dann zusätzlich zu der ihm zugeordneten Transfermenge $S \in \mathcal{S}(y)$ auch durch die ihm zugeordneten Operatoren $\boxminus$ und $\boxplus$ gekennzeichnet. $\boxminus$ und $\boxplus$ legen dann fest, auf welche Weise die Transfermenge S aus der Tour $t(S)$ zu entfernen bzw. in eine andere Tour einzufügen ist.

gesamten Palette der Eröffnungs- und Verbesserungsheuristiken für TSPR ausgewählt werden. Unter Berücksichtigung der vorangestellten allgemeinen Überlegungen soll folgendes Schema für die Struktur des Austauschoperators vorgeschlagen werden:

Der Austauschoperator $\boxminus\boxplus$ führt entsprechend der Definition 4.7 nacheinander die Operatoren $\boxminus$ und $\boxplus$ durch. Deren Struktur ist gegeben durch

Operator	Aktion	Ergebnis
$\boxminus$	(1) Entferne die Kundenknoten S_j aus der Tour t_j	$t_j^{(1)}$
	(2) Wende die Verbesserungsheuristik $\mathcal{H}_1$ auf $t_j^{(1)}$ an	$t_j^{(2)}$
$\boxplus$	(3) Füge die Kundenknoten S_i sequentiell in die Tour $t_j^{(2)}$ ein	$t_j^{(3)}$
	(4) Wende die Verbesserungsheuristik $\mathcal{H}_2$ auf $t_j^{(3)}$ an	t'_j

Der zweifache Aufruf einer Verbesserungsheuristik ist sinnvoll, weil die gefundenen Lösungen bei einer Verbesserungsheuristik von der jeweiligen Startlösung abhängig sind. Der wiederholte Einsatz von Verbesserungsverfahren während der Konstruktion einer neuen Lösung führt i.d.R. zu besseren Lösungen. Dies wird z.B. auch bei Einfügeheuristiken für das VRPTW genutzt.¹² Analog dazu könnte das Verfahren $\mathcal{H}_2$ sogar jedesmal nach dem Einfügen eines weiteren Knotens im Schritt (3) erneut aufgerufen werden. Allerdings würde dies den Rechenaufwand bei der Konstruktion des Transfergraphen deutlich erhöhen, weil die Operation $\boxplus$ für jeden Bogen des Graphen durchgeführt werden muß.

Der Operator $\boxminus$ erzeugt für alle aus einem Knoten S_i ausgehenden Bögen des Transfergraphen das gleiche Ergebnis. Deshalb muß auch die Verbesserungsheuristik $\mathcal{H}_1$ nur für jeden Knoten einmal durchgeführt werden. Da der Transfergraph i.d.R. wesentlich weniger Knoten als Bögen besitzt, kann die Heuristik $\mathcal{H}_1$ etwas aufwendiger gewählt werden.

Durch die Wahl der Verbesserungsheuristiken $\mathcal{H}_1$ und $\mathcal{H}_2$ kann der Rechenaufwand beim Aufbau des Transfergraphen gesteuert werden. Im Extremfall können diese auch „leere Operationen“ (dummies) repräsentieren,

¹²vgl. [KB95], [Rus95a], [CL99]

um den Aufwand auf ein Minimum zu reduzieren.

Erweiterung der Cyclic Transfers

Ist $\mathcal{P}$ ein m-TSPR, so ist es aufgrund der Ressourcenbeschränkungen nicht immer möglich, eine zulässige Tour $t'_j = S_i \boxminus S_j$ aufzubauen. Der zugehörige Bogen (S_i, S_j) des Transfergraphen G^T muß dann weggelassen werden. Trifft dies auf sehr viele Bögen von G^T zu und ist der Transfergraph deshalb nur dünn besetzt, so kann zwar einerseits die Suche nach Kreisen negativer Länge erheblich beschleunigt werden, andererseits bestehen jedoch auch nur geringe Chancen, einen verbessernden Cyclic-Transfer-Schritt finden zu können.

Um einen dichter besetzten Transfergraphen zu erhalten, kann man dem Austauschoperator $\boxminus$ erlauben, statt einer Tour auch mehrere Touren zu erzeugen. Damit wird das Problem $\mathcal{P}_{ij}$ von einem TSPR in ein m-TSPR überführt. Weil sich dadurch der Lösungsraum für den Austauschoperator erweitert, können so i.d.R. mehr und ggf. auch bessere zulässige Lösungen gefunden werden.

Die durch den Transfergraphen repräsentierte CT-Nachbarschaft kann infolge dieser Erweiterung des Austauschoperators auch Lösungen enthalten, die eine größere Tourenzahl aufweisen als die Startlösung y . Falls die Reduzierung der Tourenzahl das primäre Optimierungsziel des Problems $\mathcal{P}$ ist, muß dies auch auf die Probleme $\mathcal{P}_{ij}$ übertragen werden, so daß deren Lösungen nur dann mehrere Touren aufweisen, wenn dies erforderlich ist.

Die Realisierung eines solchen Austauschoperators kann z.B. so aussehen: Der Operator $\boxminus$ wird gemäß des obigen Schemas gewählt. Der Operator $\boxplus$ konstruiert eine Startlösung von $\mathcal{P}_{ij}$, die aus der Tour $t_j^{(2)}$ und einer zweiten Tour besteht. Diese zweite Tour enthält alle Kundenknoten von S_i . Die Reihenfolge der Kunden kann im einfachsten Fall aus der Tour t_i übernommen werden. Anschließend wird die Verbesserungsheuristik $\mathcal{H}_2$ gestartet. Diese kann die beiden übergebenen Touren wieder zu einer Tour zusammenfügen, sofern dies zulässig und verbessernd ist.

4.3 Suche nach verbessernden Cyclic-Transfer-Schritten

In den vorangehenden Abschnitten 4.1 und 4.2 dieses Kapitels ging es darum, wie Cyclic-Transfer-Schritte in einem Transfergraphen dargestellt werden

können und wie man einen solchen Transfergraphen aufbaut. Nun setzen wir voraus, daß bereits ein Transfergraph vorliegt, und gehen der Frage nach, auf welche Weise verbessernde Cyclic-Transfer-Schritte im Transfergraphen identifiziert werden können.

Bei vielen Vehicle Routing und Scheduling Problemen lassen sich die Kosten einer Lösung y gemäß der Gleichung (4.5) als Summe der Kosten ihrer Touren darstellen. Sei diese Eigenschaft der Kostenfunktion also gegeben und y' eine zulässige Lösung, die durch den Cyclic Transfer $ct^b(y, S^b)$ aus y hervorgeht, dann geben die Kosten des zugehörigen Kreises in $G^T(y)$ die Kostenänderung beim Übergang von y zu y' an:

$$c(y') - c(y) = \left[\sum_{i=1}^{b-1} c^T(i, i+1) \right] + c^T(b, 1) \quad (4.9)$$

Die Suche nach einem verbessernden Cyclic-Transfer-Schritt kann deshalb zunächst auf die Bestimmung eines Kreises $(S_1, \dots, S_b, S_1)$ in G^T mit negativen Kosten zurückgeführt werden. Allerdings müssen die Knoten des Kreises zusätzlich die CT-Konsistenz einhalten, damit tatsächlich nur Cyclic Transfers bestimmt werden.

Um den kürzesten Kreis im Graphen G^T zu bestimmen, kann man ausgehend von jedem Knoten $S_i \in V^T$ ein Kürzeste-Wege-Problem lösen, bei dem der Weg zum Ausgangsknoten S_i zurückführen muß.¹³ Jeder dabei gefundene Weg ist ein Kreis in G^T , und der kürzeste dieser Kreise ist gleichzeitig der kürzeste Kreis in G^T .¹⁴ Die Berücksichtigung der CT-Konsistenz muß in die Suche nach einem kürzesten Weg integriert werden.

Wie im folgenden gezeigt wird, läßt sich die CT-Konsistenz mit Hilfe einer Menge beschränkter Ressourcen modellieren. Das dadurch entstehende Kürzeste-Wege-Problem mit Ressourcenbeschränkungen (SPR) kann prinzipiell mit dem „Permanent Labelling Algorithmus“ von Desrochers und Soumis gelöst werden. Da das SPR aber $\mathcal{NP}$ -schwer ist,¹⁵ kann dieser Ansatz nur für relativ kleine Transfergraphen umgesetzt werden. Eine Alternative dazu besteht im Einsatz einer „Label Correcting Heuristik“, die von Ahuja et. al. in

¹³Schlingen, d.h. Bögen, die einen Knoten mit sich selbst verbinden, sind im G^T nicht enthalten. Daher berührt jeder Weg von S_i nach S_i mindestens einen weiteren Knoten von G^T .

¹⁴Dies folgt unmittelbar, da auch der kürzeste Kreis in G^T (sofern existierend) mindestens zwei Knoten von G^T enthält.

¹⁵vgl. [DDI⁺98, S. 63] oder [AMO93, S. 141]

[AOS01] für das Capacitated Minimum Spanning Tree Problem vorgeschlagen wurde.

Modellierung der CT-Konsistenz

Es sei $y = \{t_1, \dots, t_m\}$ eine Lösung mit m Touren. Dann kann die Einhaltung der CT-Konsistenz mit Hilfe einer Menge $R^T = \{r_1^T, \dots, r_m^T\}$ von m beschränkten Ressourcen wie folgt erzwungen werden:

- jeder Tour t_i wird eine Ressource r_i^T zugeordnet,
- jeder Bogen (S_j, S_k) erhöht den Wert der Ressource r_i um $\begin{cases} 1, & \text{falls } t(S_k) = t_i, \\ 0, & \text{sonst,} \end{cases}$
- für jeden Knoten S_j des Transfergraphen G^T wird gefordert: $r_j^T \in [0, 1]$.

Um alle Kreise mit negativen Kosten finden zu können, startet man die Suche ausgehend von jedem Knoten S_i von G^T mit $S_i \neq S_0$ und setzt dabei jeweils alle Ressourcen des Start-Labels auf null. Der Wert der Ressource r_i^T erhöht sich immer dann um eins, wenn ein Bogen des Transfergraphen benutzt wird, der zu einem Knoten führt, dessen Transfermenge der Tour t_i zugeordnet ist. Durch die Festsetzung der oberen Schranke aller Ressourcen auf eins wird die Einhaltung der CT-Konsistenz sichergestellt.

Die Kosten eines Kreises können als zusätzliche Ressource r_c^T modelliert werden. Da nur Kreise mit negativen Kosten interessieren, kann für diese Ressource eine obere Schranke von 0 angegeben werden, um die Effizienz der Suche zu erhöhen. Wenn die Suche bei jedem Knoten einmal gestartet wird, werden dadurch keine verbessernden Cyclic Transfers ausgeschlossen. Dies begründet sich in folgendem einfachen Zusammenhang:

Satz 4.14. Sei $Z = (z_1, \dots, z_n)$ eine Folge von n Zahlen mit $z_1 + \dots + z_n < 0$. Dann existiert eine zyklische Permutation π von Z , so daß jede Partialsumme $z_{\pi(1)} + \dots + z_{\pi(k)}$ mit $1 \leq k \leq n$ negativ ist.

Beweis: Sei $l \in \{1, \dots, n\}$ der größte Index, für den die Summe $z_1 + \dots + z_{l-1}$ maximal wird. Dann gilt für die Partialsumme, die am Index l beginnt und

die Zahlen bis zum Index j summiert:¹⁶

$$\begin{aligned} \text{falls } l \leq j \leq n : \quad & \sum_{i=l}^j z_i = \sum_{i=1}^j z_i - \sum_{i=1}^{l-1} z_i < 0 \\ \text{falls } 1 \leq j < l : \quad & \sum_{i=l}^n z_i + \sum_{i=1}^j z_i \leq \sum_{i=l}^n z_i + \sum_{i=1}^{l-1} z_i = \sum_{i=1}^n z_i < 0 \end{aligned}$$

□

Daher gibt es in jedem Kreis $(S_1, \dots, S_b, S_1)$ mit negativen Kosten einen Knoten $S_i \in \{S_1, \dots, S_b\}$, so daß die Kosten aller Pfade, die im Knoten S_i beginnen und entlang des Kreises verlaufen, negativ sind.

Sobald ein Cyclic Transfer $ct^b(y, (S_1, \dots, S_b))$ mit negativen Kosten gefunden wurde, interessieren nur noch Cyclic Transfers, die zu noch besseren Lösungen führen. Dementsprechend kann die obere Schranke für die Kosten eines Kreises verschärft werden. Ist m die maximale Länge eines Kreises in G^T , so gilt:

$$r_c^T \leq \frac{c^T(S_1, \dots, S_b, S_1)}{m}. \quad (4.10)$$

Die Begründung dafür liefert der folgende Satz.

Satz 4.15. Sei $Z = (z_1, \dots, z_n)$ eine Folge von n Zahlen mit $z_1 + \dots + z_n < c < 0$. Dann existiert eine zyklische Permutation von Z , so daß jede partielle Summe kleiner als $\bar{c} = c n^{-1}$ ist.

Beweis: Man betrachte die Zahlenfolge $Z' = (z'_1, \dots, z'_n)$ mit $z'_i := z_i - \bar{c}$ für alle $1 \leq i \leq n$. Da die Summe der Zahlen von Z' negativ ist, existiert wegen Satz 4.14 eine zyklische Permutation π , so daß alle Partialsummen von Z' negativ sind. Wendet man dieselbe Permutation π auf Z an, so gilt für die Partialsummen:

$$\sum_{i=1}^k z_{\pi(i)} = \sum_{i=1}^k [z'_{\pi(i)}] + k \bar{c} < 0 + k \bar{c} = c, \quad \forall 1 \leq k \leq n.$$

□

¹⁶Die Beweisidee wurde aus [LK73, S. 502] übernommen.

Exakter Labelling Algorithmus für das SPR

Zur Lösung des Kürzeste-Wege-Problems mit Zeitfenstern (SPTW) wurde von Desrochers und Soumis 1988 ein exakter Labelling Algorithmus mit pseudo-polynomialer Laufzeit vorgeschlagen, den sie „Permanent Labelling Algorithmus“ nennen [DS88a]. Diese Bezeichnung weist darauf hin, daß es sich um einen Label Setting Algorithmus¹⁷ handelt. Der Permanent Labelling Algorithmus wird in [DS88a] für SPTW beschrieben. Eine Verallgemeinerung des Vorgehens auf Vektoren beschränkter Ressourcen ist leicht möglich, so daß das dort beschriebene Vorgehen auch auf Kürzeste-Wege-Probleme mit mehreren Ressourcen (SPR) angewendet werden kann.

Label Correcting Heuristik für das SPR

In [AOS01] wird eine Heuristik zur Bestimmung von Kreisen negativer Länge beschrieben. Diese Heuristik entsteht durch eine Modifikation eines Label Correcting Algorithmus. Die Autoren verwenden dafür den Label Correcting Algorithmus von Pape [Pap80].

Durch die angesprochene Modifikation wird erreicht, daß nur „*subset disjoint cycles*“ Berücksichtigung finden. Der Begriff der *subset disjoint cycles* entspricht in der hier gewählten Sprechweise einem Kreis, der die CT-Konsistenz einhält. Daher kann die Heuristik auch für die Suche nach einem Kreis negativer Länge unter Berücksichtigung der CT-Konsistenz angewandt werden.

Zunächst betrachten wir einen Label Correcting Algorithmus ohne Berücksichtigung zusätzlicher Forderungen, d.h. gesucht sei ein kürzester Weg in einem gerichteten Graphen $G(V, A, c)$ von einem Startknoten $S_{\text{start}} \in V$ zu einem Zielknoten $S_{\text{dest}} \in V$.

Label Correcting Algorithmus Der Algorithmus verwaltet zu jedem Knoten S_i des Graphen zwei Informationen (Labels).

1. Distanz-Label:

Das Distanz-Label (bzw. Kosten-Label) $d(S_i)$ ist entweder ∞ , um anzuzeigen, daß noch kein Pfad vom Startknoten S_{start} zum Knoten S_i gefunden wurde, oder es gibt die Länge des aktuellen Pfades von S_{start} zu S_i an.

¹⁷vgl. [AMO93, Kapitel 4]

2. Vorgänger-Label:

Das Vorgänger-Label $pred(S_i)$ gibt den Vorgängerknoten von S_i innerhalb des aktuellen Pfades von S_{start} zu S_i mit der Länge (bzw. den Kosten) $d(S_i)$ an.

$P(S_i)$ bezeichne den Pfad vom Startknoten S_{start} zum Knoten S_i , der als (umgekehrte) Abfolge der Vorgänger-Label erhalten wird:

$$P(S_i) = (S_{\text{start}}, \dots, pred(pred(S_i)), pred(S_i), S_i).$$

Der Ablauf eines (generischen) Label Correcting Algorithmus läßt sich wie folgt beschreiben:¹⁸ Zu Beginn werden alle Distanz-Labels auf ∞ gesetzt. Nur das Distanz-Label des Knotens S_{start} erhält den Wert null. Der Algorithmus prüft nun, ob die Optimalitätsbedingung $d(S_j) \leq d(S_i) + c_{ij}$ für alle Bögen (S_i, S_j) erfüllt ist. Wird ein Bogen gefunden, der diese Bedingung verletzt, muß für diesen Bogen eine Aktualisierung des Distanz-Labels („*distance update*“) vorgenommen werden. Die Aktualisierung des Distanz-Labels wird solange fortgesetzt, bis die Optimalitätsbedingungen für alle Bögen aus A erfüllt sind.

Der angegebene Algorithmus ist generisch, weil bislang nicht angegeben wurde, wie die Suche nach verletzten Optimalitätsbedingungen zu organisieren ist. Man verwendet dafür eine Struktur „*LIST*“, die folgende Eigenschaft besitzt: *LIST* enthält alle Knoten $S_i \in V$, die (möglicherweise) einen ausgehenden Bogen (S_i, S_j) besitzen, der die Optimalitätsbedingung verletzt.

Die Suche nach Bögen, die die Optimalitätsbedingung verletzen, wird nun durchgeführt, indem die Knoten von *LIST* sukzessive untersucht werden:

Algorithmus 4.1. Label Correcting Algorithmus¹⁸

1. Setze $d(S_i) := \infty$ für alle $S_i \in A \setminus \{S_{\text{start}}\}$.
2. Setze $d(S_{\text{start}}) := 0$, $pred(S_{\text{start}}) := 0$ und $LIST := \{S_{\text{start}}\}$.
3. Solange $LIST \neq \emptyset$:
 - 3.1 Entferne das erste Element S_i von *LIST*.
 - 3.2 (*distance update*)
 Prüfe für jeden Bogen (S_i, S_j) die Optimalitätsbedingung.
 Falls $d(S_j) > d(S_i) + c_{ij}$:

¹⁸vgl. [AMO93, Kapitel 5]

3.2.1 Setze $d(S_j) = d(S_i) + c_{ij}$.

3.2.2 Setze $pred(S_j) = S_i$.

3.2.3 Füge S_j zu $LIST$ hinzu.

Für den Schritt 3(.2).3 können verschiedene Realisierungen betrachtet werden. Bei der FIFO-Strategie wird ein hinzuzufügender Knoten immer am Ende von $LIST$ angefügt. Für diese Strategie kann der Aufwand des Algorithmus mit $\mathcal{O}(|V||A|)$ abgeschätzt werden. Für dünn besetzte Graphen mit wenigen Bögen hat sich die „Dequeue-Implementierung“ in der Praxis als besonders günstig erwiesen.

Berücksichtigung der CT-Konsistenz Die Label Correcting Heuristik von Ahuja et. al. [AOS01] versucht, Pfade zu identifizieren, die die CT-Konsistenz einhalten. Dazu wird das *distance update* des Label Correcting Algorithmus 4.1 wie folgt modifiziert:

1. Beim Entfernen eines Knotens S_i von $LIST$ wird geprüft, ob der aktuelle Pfad von S_{start} zu S_i die CT-Konsistenz einhält. Falls dies nicht der Fall ist, wird der Knoten S_i nicht untersucht.
2. Wird im Schritt 3.2 des Algorithmus ein Bogen gefunden, der die Optimalitätsbedingung verletzt, so werden beim *distance update* folgende drei Fälle unterschieden:
 - (a) Der Knoten S_j liegt auf dem Pfad $P(S_i)$, d.h. $S_j \in V(P(S_i))$. Dann wird der Teilpfad $(S_j, \dots, S_i)$ durch das Anhängen des Bogens (S_i, S_j) zu einem Kreis in G erweitert, der wegen $d(S_j) > d(S_i) + c_{ij}$ eine negative Länge aufweist.
 - (b) $S_j \notin V(P(S_i))$, und es gibt ein $S_k \in P(S_i)$ mit $t(S_j) = t(S_k)$. Dann verletzt der Pfad $(S_{\text{start}}, \dots, S_i, S_j)$ die CT-Konsistenz. Die Aktualisierung der Labels von Knoten S_j wird in diesem Fall nicht durchgeführt und S_j auch nicht zu $LIST$ hinzugefügt.
 - (c) In allen anderen Fällen wird das *distance update* so durchgeführt, wie im Algorithmus 4.1 angegeben.

Ahuja et. al. berichten, daß diese Heuristik sehr schnell terminiert und bei einer Testreihe sehr häufig alle verbessernden Kreise finden konnte, wenn die Suche von jedem Knoten $S_i \in G^T$, $S_i \neq S_{0*}$ einmal gestartet wird.

¹⁸vgl. [AMO93, S. 141]

4.4 Effiziente Lokale Suche in der Cyclic-Transfer-Nachbarschaft

Es wurde bereits dargelegt, wie die CT-Nachbarschaft zu einer gegebenen Lösung $y \in Y$ mit Hilfe eines Transfergraphen repräsentiert und in diesem Graphen nach einem verbessernden Cyclic Transfer gesucht werden kann. Jetzt geht es darum, auf der Basis der CT-Nachbarschaften eine effiziente Lokale Suche durchzuführen. Insbesondere soll der Aufwand für die Bestimmung des Transfergraphen möglichst gering gehalten werden. Erhebliche Einsparungen können dabei ggf. erzielt werden, wenn man ausnutzt, daß die Transfergraphen der innerhalb der Lokalen Suche aufeinanderfolgenden CT-Nachbarschaften i.a. identische Teilgraphen besitzen.

Es sei $y \in Y$ eine zulässige Lösung mit m Touren und y' ein Element der CT-Nachbarschaft von y . $ct^b(y, (S_1, \dots, S_b))$ sei der Cyclic-Transfer-Schritt, der y in y' überführt. $G^T(y) = (V^T, A^T, c^T)$ ist der Transfergraph von y . Der Transfergraph von y' heißt $G^T(y')$.

Angenommen, die Tourenzahl von y' ist ebenfalls m ¹⁹ und $t_1, \dots, t_b$ sind die durch den Cyclic Transfer modifizierten Touren von y . Die übrigen Touren $t_{b+1}, \dots, t_m$ von y sind unverändert in y' enthalten, d.h.

$$\begin{aligned} t_i &= t(S_i), & \forall i \in \{1, \dots, b\} \text{ und} \\ t'_1 &= S_b \boxminus S_1, \\ t'_i &= S_{i-1} \boxminus S_i, & \forall i \in \{1, \dots, b\} \text{ und} \\ t'_i &= t_i, & \forall i \in \{b+1, \dots, m\}. \end{aligned}$$

Dann gilt für die Transfergraphen $G^T(y)$ und $G^T(y')$:

1. Der Teilgraph von $G^T(y)$, der durch die Knotenmenge

$$\left\{ S : S \in \bigcup_{i=b+1}^m \mathcal{S}(t_i) \right\} \subset V^T$$

induziert wird, ist auch in $G^T(y')$ enthalten.

2. die Bögen (S_i, S_j) von $G^T(y)$ sind im Graphen $G^T(y')$ mit unveränderten Kosten wiederzufinden, falls gilt:

$$S_i \in \mathcal{S}(y') \text{ und } j \in \{b+1, \dots, m\}.$$

¹⁹Diese Annahme dient nur der Vereinfachung der Darstellung. Die Überlegungen lassen sich auch auf Fälle mit steigender oder fallender Tourenzahl übertragen.

Diese Bögen müssen demzufolge nach dem Übergang zu einer verbessernden Lösung y' im Rahmen der Lokalen Suche nicht neu berechnet werden. Wie groß die tatsächliche Einsparung gegenüber einer vollständigen Neuberechnung des Transfergraphen $G^T(y')$ ist, hängt davon ab, wieviele Touren durch den Cyclic Transfer unverändert geblieben sind und wie lang diese Touren sind.

Wechsel zwischen reduzierten Nachbarschaften

Durch das Berücksichtigen eingeschränkter Transfermengen erhält man eine reduzierte CT-Nachbarschaft (vgl. Abschnitt 4.2.3). Unterschiedliche reduzierte CT-Nachbarschaften können im Sinne einer Variable-Neighborhood-Descent-Heuristik (VND) miteinander kombiniert werden. Bei der VND wird der Wechsel zu einer alternativen Nachbarschaft erst vollzogen, wenn keine Verbesserung in der augenblicklich untersuchten Nachbarschaft mehr möglich ist. Dies ist auch sinnvoll, um die zuvor angesprochenen Einsparungspotentiale bei der Bestimmung der Transfergraphen vollständig zu nutzen.

Es ist darüber hinaus denkbar, auch beim Wechsel zwischen verschiedenen reduzierten CT-Nachbarschaften den zuvor bestimmten Transfergraphen ganz oder teilweise weiter zu nutzen. Dies ist z.B. dann der Fall, wenn die neue Nachbarschaft die vorhergehende komplett umfaßt. So kann man mit einer kleinen reduzierten CT-Nachbarschaft beginnen und diese Schritt für Schritt durch die Berücksichtigung zusätzlicher Transfermengen (Knoten in G^T) systematisch erweitern.

Auch bei der Wahl der Operatoren $\boxminus$ und $\boxplus$ kann der investierte Rechenaufwand für die Bestimmung guter Touren zunächst klein gehalten werden. Erst wenn keine Verbesserung in der aktuellen CT-Nachbarschaft gefunden werden kann, versucht man durch eine Reoptimierung der Touren die Kosten der Bögen im Transfergraphen zu reduzieren und so weiteres Verbesserungspotential aufzudecken.

Eine weitere Möglichkeit besteht darin, nach jedem verbessernden Cyclic-Transfer-Schritt die dadurch betroffenen (und tatsächlich veränderten) Touren einer zusätzlichen Reoptimierung (z.B. mit k -opt-Nachbarschaften auf einer aus diesen Touren gebildeten Giant-Route) zu unterziehen. Dadurch werden die Möglichkeiten, Teile des Transfergraphen wiederzuverwenden, nicht zusätzlich eingeschränkt.

4.5 Auflösen von Touren

Beim m-TSPR ist die Reduzierung der Tourenzahl ein häufig formuliertes und sehr schwer zu verfolgendes Ziel. Es ist leicht nachzuvollziehen, daß die Reduktion der Tourenzahl einer gegebenen Lösung y mit Hilfe von k -opt-Nachbarschaften aber auch mit den knotenorientierten Nachbarschaften wie den Cyclic Transfers oft nicht erfolgreich ist.

Die Schwierigkeit liegt vor allem darin begründet, daß es beim Absuchen des Lösungsraums keine eindeutigen Merkmale dafür gibt, ob man sich einer Lösung mit weniger Touren nähert oder nicht. Auch die Berücksichtigung nachrangiger Ziele, wie z.B. die Minimierung der gesamten Weglänge oder der gesamten Fahrtdauer, muß nicht zielführend sein. Außerdem wird die Auflösung einer Tour durch Umverteilen der Kunden auf andere Touren durch die Restriktionen erschwert.

Die wesentlichen Ideen zum zielgerichteten Erzeugen von Lösungen mit geringen Tourenzahlen lassen sich wie folgt zusammenfassen:

Vorgehen	Beispiele für das VRPTW
Einsatz konstruktiver Verfahren, die von vornherein versuchen, mit möglichst wenig Touren eine zulässige Lösung aufzubauen	[Sol87] (I1-Heuristik), [KB95] (GRASP), [CR97], [LS99] (RNETS), [GTA99] (MACS)
Einsatz spezieller Verfahren, die gezielt versuchen, (kurze) Touren dadurch aufzulösen, daß sequentiell alle Kunden der Tour in andere Touren verschoben werden	[KB95] (GRASP), [LS99] (RNETS), [SF99], [CW01] (AKRed)
Modifikation der Zielfunktion, um Bildung von Touren mit wenigen Kunden zu begünstigen	[HG98]
Heuristische Column Generation Verfahren	[RT95] (AMP), [Tai99b]

Mit den in diesem Kapitel beschriebenen Cyclic Transfers ist es kaum möglich, Touren gezielt aufzulösen. In diesem Abschnitt werden Ansätze dis-

kutiert, wie dennoch grundlegende Ideen der Cyclic Transfers für die gezielte Auflösung von Touren genutzt werden können.

Modifikation der Zielfunktion

Um Nachbarschaftsschritte zu finden, die die Tourenzahl reduzieren, besteht eine erste Maßnahme darin, hohe Fixkosten für jede (nicht-leere) Tour in die Kostenfunktion aufzunehmen.²⁰ Werden die Fixkosten ausreichend hoch gewählt, so ist jeder Nachbarschaftsschritt, der die Zahl der Touren reduziert, auch ein verbessernder Schritt.

Durch diese Modifikation der Zielfunktion wird sichergestellt, daß keine mögliche Reduktion der Tourenzahl „übersehen“ wird. Vorausgesetzt, daß die leeren Transfermengen in der Knotenmenge V^T enthalten sind, kann die Suche in der CT-Nachbarschaft den Abbau einer oder mehrerer Touren bewirken. Meistens führt diese einzelne Maßnahme jedoch nicht zum Erfolg, denn eine Tour kann nur dann aufgelöst werden, wenn alle Kundenknoten dieser Tour in *eine einzige andere Tour* transferiert werden können. Die Möglichkeit, die Kunden einer Tour auf mehrere verschiedene Touren zu verteilen, wird durch die CT-Konsistenz explizit ausgeschlossen.

Um die Kunden einer Tour (falls möglich) dennoch auf andere Touren aufteilen zu können, können zwei verschiedene Ansätze verfolgt werden. Eine Möglichkeit besteht darin, die CT-Konsistenz für eine einzelne Tour aufzuheben. Dabei muß durch andere Beschränkungen abgesichert werden, daß trotzdem nur solche Wege im Transfergraphen gefunden werden können, die real durchführbare Nachbarschaftsschritte repräsentieren. Alternativ kann man auch versuchen, die Kunden einer Tour in mehreren Schritten auf die übrigen Touren zu verteilen. Diese beiden Ansätze sollen im folgenden etwas ausführlicher dargestellt werden.

Verteilen der Kunden einer Tour in einem Schritt

Im weiteren bezeichne t_{rem} diejenige Tour von y , deren Knoten auf die übrigen Touren von y aufgeteilt werden sollen. Alle geeigneten Transfer-Schritte

²⁰Die Fixkosten können allen Bögen zugeschlagen werden, die von einem Depotausgangsknoten zu einem Kundenknoten führen. Dagegen werden solchen Bögen, die einen Depotausgang direkt mit einem Depoteingang verbinden, keine Fixkosten zugerechnet. Dadurch bleiben ggf. entstehende „Leertouren“ unbestraft.

werden im Transfergraphen durch einen Weg P_{rem} mit folgenden Eigenschaften repräsentiert:

1. Die CT-Konsistenz bleibt für alle übrigen Touren $t \in T$, $t \neq t_{rem}$ erhalten, d.h. es wird höchstens eine Transfermenge nach t transferiert.
2. Es werden nur leere Transfermengen in die Tour t_{rem} transferiert.
3. Ist $\mathcal{S}_{rem}$ die Menge aller Knoten von P_{rem} , die Transfermengen von t_{rem} repräsentieren, so sind diese Transfermengen von $\mathcal{S}_{rem}$ durchschnittsfremd und beinhalten gemeinsam alle Kunden von t_{rem} :

$$S_i, S_j \in \mathcal{S}_{rem} \text{ und } S_i \neq S_j : S_i \cap S_j = \emptyset \quad (4.11a)$$

$$\bigcup_{S \in \mathcal{S}_{rem}} S = V(t_{rem}) \setminus D \quad (4.11b)$$

Der Transfergraph kann also im wesentlichen genauso wie zuvor aufgebaut werden. Die Knoten des Transfergraphen repräsentieren unterschiedliche Transfermengen. Die Bögen stellen Austauschoperationen dar. Lediglich Bögen, die nicht-leere Kundenmengen in die Tour t_{rem} verschieben würden, können ausgeschlossen werden.

Anstelle der CT-Konsistenz müssen für die Tour t_{rem} die Restriktionen (4.11a) und (4.11b) bei der Suche nach kürzesten Wegen im Transfergraphen eingehalten werden. Dies erfordert eine entsprechende Anpassung des Algorithmus für die Suche nach verbessernden Cyclic-Transfer-Schritten im Transfergraphen.

Verteilen der Kunden einer Tour in mehreren Schritten

Geht man davon aus, daß es nicht möglich oder zu schwierig ist, die Kunden einer Tour in einem Schritt zu verteilen, kann man auch versuchen, dies in mehreren aufeinanderfolgenden Schritten zu erreichen. Dabei soll die CT-Konsistenz aber in jedem Schritt eingehalten werden.

Die Tour von y , deren Knoten verteilt werden sollen, sei weiterhin mit t_{rem} bezeichnet. Unser erklärtes Ziel ist es, die Kunden aus der Tour t_{rem} *vollständig* auf andere Touren umzuverteilen. Um dies zu erreichen werden zwei Modifikationen vorgenommen:

1. Die Zielfunktion wird durch einen zusätzlichen Term so erweitert, daß die Verkürzung der Tour t_{rem} proportional zur Zahl der entfernten Knoten belohnt wird.

2. Bögen des Transfergraphen, die das Einfügen von Kundenknoten aus anderen Touren in die Tour t_{rem} implizieren, werden verboten.

Die Modifikation der Zielfunktion macht es attraktiv, Kundenknoten aus der Tour t_{rem} auch dann herauszuschieben, wenn dies mit einer Erhöhung der übrigen Kosten verbunden ist. Je mehr Kundenknoten in einem Schritt entfernt werden können, desto schneller kommen wir dem Ziel einer leeren Tour näher. Deshalb erfolgt die Belohnung proportional zur Zahl der entfernten Knoten. Die bisher betrachteten Kosten sollen durch die „Belohnungen“ deutlich überlagert werden. Sie werden jedoch nicht aus der Betrachtung ausgeschlossen, weil sie für eine sinnvolle Auswahl zwischen verschiedenen Cyclic-Transfer-Schritte sorgen, die gleich viele Kunden aus t_{rem} eliminieren.

Da wir nicht an einer Tour mit wenigen Kunden, sondern ausschließlich an einer leeren Tour interessiert sind, verbieten wir zusätzlich das Einfügen anderer Kundenknoten. Dies dient in erster Linie dazu, mit dem Vorgehen nicht nur die aktuell kürzeste Tour reduzieren zu können. Läßt man diese Forderung weg und ist t_{rem} nicht die kürzeste Tour der Lösung y , so werden die Kunden von t_{rem} (schrittweise oder falls möglich mit einem Mal) zuerst gegen die Kunden der kürzesten Tour ausgetauscht, wodurch sich die Kundenzahl von t_{rem} reduziert und keine zusätzlichen Kosten entstehen.

Die praktische Realisierung dieses Ansatzes ist sehr einfach, weil nur der Transfergraph etwas modifiziert aufgebaut werden muß.

Kapitel 5

Erweiterungen der Konzepte

Der mit dem Ressourcenmodell und den in den Kapiteln 3 und 4 beschriebenen Konzepten vorgegebene Rahmen bildet eine gute Basis für die Entwicklung leistungstarker Heuristiken durch die Integration dieser Ansätze in eine Meta-Strategie. Das große Potential, das mit dem Einsatz dieser Methoden verbunden ist, ist vor allem in der Anwendbarkeit auf nahezu alle Vehicle Routing und Scheduling Probleme zu sehen.

Auf die vielfältigen Erweiterungsmöglichkeiten der Konzepte kann im Rahmen dieser Arbeit aufgrund der gebotenen Kürze nur hingewiesen werden. Sie beziehen sich einerseits auf die Modellierung von Problemen, deren Darstellung mit Hilfe des Ressourcenmodells möglich, aber nicht sofort offenbar ist. Andererseits bieten sich eine Reihe algorithmischer Erweiterungen an.

Im Abschnitt 5.1 wird darauf eingegangen, wie Probleme mit heterogenem Fuhrpark bzw. mehreren Depots und wie VRSP mit weichen Restriktionen mit dem Ressourcenmodell dargestellt werden können. Diese Problemklassen besitzen für praktische Planungsaufgaben oft eine hohe Bedeutung. Anschließend werden im Abschnitt 5.2 mehrere Möglichkeiten angesprochen, die Effizienz der beschriebenen Heuristiken weiter zu verbessern. In diesem Zusammenhang werden vor allem weitere heuristische Beschränkungen der Nachbarschaften vorgeschlagen, die zur Beschleunigung der Suche beitragen können. Darüber hinaus werden Ansätze zur Integration mit verschiedenen Meta-Heuristiken und einem Constraint-Programming-System aufgezeigt.

5.1 Erweiterte Modelle

Im Kapitel 1 wurde bereits dargelegt, daß sich viele VRSP mit Hilfe beschränkter Ressourcen modellieren lassen. Wir gehen im folgenden darauf ein, daß dies auch auf Probleme mit heterogenem Fuhrpark bzw. mehreren Depots und Probleme mit weichen Restriktionen zutrifft. Man beachte, daß daraus zwar geschlossen werden kann, daß die zuvor beschriebenen Heuristiken darauf anwendbar sind. Dabei bleibt jedoch offen, ob sie auch in der Lage sind, für diese Probleme gute Lösungen zu bestimmen. In jedem Fall können die Methoden genutzt werden, um andere, für diese Probleme gut geeignete Heuristiken, zu ergänzen.

5.1.1 Heterogener Fuhrpark und mehrere Depots

Das Modell 1.1 kann prinzipiell auch zur Beschreibung eines m-TSPR mit heterogenem Fuhrpark bzw. eines Multiple-Depot m-TSPR verwendet werden. In beiden Fällen ist es lediglich erforderlich, mehrere Depots zu unterscheiden. Beim m-TSPR mit heterogenem Fuhrpark, wird für jeden Fahrzeugtyp ein „eigenes Depot“ eingerichtet.

Durch die Wahl der Ressourcenfenster am Depoteingang bzw. am Depotausgang können spezifische Startbedingungen für die Ressourcen für eine Tour festgelegt werden. Dadurch ist es auch leicht möglich, vom gewählten Fahrzeugtyp abhängige Fixkosten oder Kapazitäten zu modellieren. Mit Hilfe abhängiger Ressourcen ist es in gewissen Grenzen auch möglich, unterschiedliche, vom verwendeten Depot bzw. Fahrzeugtyp abhängige Kostenfunktionen zu formulieren. Mit zusätzlichen Ressourcen können auch Inkompatibilitäten zwischen bestimmten Fahrzeugtypen und einzelnen Kunden sowie vielfältige Beschränkungen für die Verfügbarkeit bestimmter Fahrzeuge angegeben werden.

Solche Modelle aufzustellen ist leicht möglich. Die beschriebenen Heuristiken lassen sich dann auch direkt auf m-TSPR mit heterogenem Fuhrpark anwenden. Dabei ist jedoch zu prüfen, ob die vorgeschlagenen Nachbarschaften ausreichen, um stets gute Lösungen zu finden. Denkbar ist, daß wesentlich bessere Lösungen gefunden werden, wenn weitere, dem Problemtyp angepaßte Nachbarschaften hinzugenommen werden. Die Nachbarschaftsschritte könnten z.B. einen gezielten Wechsel der Touren zwischen mehreren Fahrzeugen bzw. Depots zum Gegenstand haben. Bei einer solchen Nachbarschaft ist es naheliegend, diese als spezielle Cyclic-Transfer-Nachbarschaft aufzufassen.

5.1.2 Modelle mit Strafkosten und weichen Restriktionen

In einigen Veröffentlichungen werden auch VRSP mit weichen Zeitfenstern („soft time windows“) betrachtet.¹ Darunter versteht man Probleme, bei denen die Zeitfenster nicht strikt eingehalten werden müssen. Durch die Einführungen von zusätzlichen Termen in die Zielfunktion werden Lösungen bestraft, die die Zeitfenster verletzen. Die Berücksichtigung von weichen Zeitfenstern führt dazu, daß weniger (falls noch andere harte Restriktionen existieren) oder gar keine Lösungen mehr unzulässig sind.

Das Ressourcenmodell läßt sich auch für die Darstellung weicher Restriktionen benutzen. Anstelle der in dieser Arbeit angegebenen Aktualisierung der Zulässigkeitsbereiche für die Ressourcenvariablen, brauchen bei weichen Restriktionen für diese Variablen nur ihre tatsächlichen Werte bestimmt werden. Erst für die Bestimmung der Kosten wird die Unter- bzw. Überschreitung von Ressourcenbeschränkungen berücksichtigt. Die effiziente Berechnung der Strafterme wird mit einer Kostenressource realisiert, die von den zu bestrafenden Ressourcenvariablen abhängig ist. Diese Behandlung weicher Ressourcen erlaubt es, mehrere weiche und harte Restriktionen in einem Modell zu integrieren.

Dieser Ansatz zur Darstellung weicher Restriktionen läßt sich auch nutzen, um Meta-Strategien zu realisieren, bei denen die Zulässigkeit der Lösungen nicht von Beginn an erzwungen wird und die Suche langfristig durch die Modifikation der Strafterme in zulässige Bereiche des Lösungsraums gelenkt werden soll.

5.2 Effizienzverbesserungen und Integration mit Meta-Strategien und CP

In diesem Abschnitt werden zunächst weitere Ideen zu Verbesserung der Effizienz der bereits beschriebenen Heuristiken beschrieben. Anschließend werden Ansätze zur Integration mit verschiedenen Meta-Strategien und die Möglichkeit der Verbindung mit einem Constraint Programming System beschrieben.

¹z.B. [BGG⁺95], [KPS92], [TBG⁺97]

5.2.1 Verbesserungen der Effizienz

Obwohl bereits eine Vielzahl unterschiedlicher Ideen zur Reduktion des mit der Lokalen Suche verbundenen Aufwands berücksichtigt wurden, ist es möglich, daß bei einer weitergehenden Analyse eine Reihe weiterer Maßnahmen zur Verbesserung der Effizienz aufgedeckt werden können. In diesem Zusammenhang soll auf die folgenden Ansatzpunkte verwiesen werden:

Gemeinsame Behandlung unterschiedlicher k -opt-Typen In den Abschnitten 3.1.5 und 6.3 wurde darauf verwiesen, daß es Vorteile bietet, die k -opt-Nachbarschaften nach unterschiedlichen k -opt-Typen getrennt zu untersuchen. Dadurch können relativ einfach Abbruchkriterien innerhalb der Lexikographischen Suche überprüft und genutzt werden, die die Effizienz deutlich steigern. Allerdings wird dieser Vorteil spätestens dann wieder aufgehoben, wenn dieselben Toursegmente für die Untersuchung eines anderen k -opt-Typs später erneut aufgebaut werden müssen.

Eine bessere Alternative besteht deshalb vermutlich darin, die k -opt-Typen gemeinsam zu untersuchen und Abbrüche immer dann zu generieren, wenn kein k -opt-Typ mehr zu einer zulässigen oder verbessernden Lösung führen kann. Die Realisierung ist jedoch recht kompliziert und erfordert die Lösung einer Reihe von Detailproblemen. Zu berücksichtigen ist in jedem Fall, daß die Verwaltung der notwendigen Informationen, um über einen möglichen Abbruch entscheiden zu können, deutlich weniger aufwendig sein muß, als die erneute Konstruktion der Toursegmente.

Ausnutzen der Tourunabhängigkeiten Bei der Untersuchung von k -opt-Nachbarschaften kommt es immer wieder dazu, daß bereits eine Reihe von unterschiedlichen k -opt-Typen ergebnislos untersucht wurden. Wird später ein verbessernder Schritt gefunden und durchgeführt, müssen diese k -opt-Typen erneut untersucht werden. Dabei kann jedoch ausgenutzt werden, daß ein verbessernder k -opt-Schritt mindestens einen Bogen der modifizierten Touren ändern muß (weil die Verbesserung sonst schon vorher gefunden worden wäre).

Außerdem kann es häufig vorkommen, daß einzelne Touren schon zeitig ihre endgültige Form angenommen haben und nicht mehr geändert werden, während andere Touren noch häufig verbessernd modifiziert werden können.

Eine einfache Möglichkeit, diese Zusammenhänge auszunutzen, besteht darin, die Touren innerhalb der Giant-Route nach dem Zeitpunkt ihrer letz-

ten Änderung zu ordnen. Dabei sollten die zuletzt geänderten Touren am weitesten vorn stehen. Beginnt man nun die Lexikographische Suche vom Anfang, werden immer zuerst k -opt-Schritte untersucht, an denen eine bzw. mehrere Touren beteiligt sind, die zuletzt geändert wurden.

Sobald das erste Segment alle zuletzt geänderten Touren umfaßt und demzufolge alle zu verändernden Bögen ausschließlich in den zuletzt ungeänderten Touren enthalten sein müssen, kann die Suche abgebrochen werden. Voraussetzung dafür ist allerdings, daß alle bereits gefunden Verbesserungen zwischengespeichert wurden. Ein solches Speichern bereits gefundener Verbesserungen wird in [CW01] für k -opt-Nachbarschaften mit $k = 2$ bzw. 3 beschrieben.

Sinnvolle Einschränkungen der Cyclic-Transfer-Nachbarschaften

Die Untersuchung der Cyclic-Transfer-Nachbarschaften ist trotz der bereits angestellten Überlegungen zu deren effizienter Gestaltung² relativ aufwendig. Um den erforderlichen Rechenaufwand für die Konstruktion des Transfergraphen einerseits und für die Suche nach Verbesserungen in diesem Graphen andererseits weiter zu reduzieren, kann versucht werden, die Zahl der Bögen im Transfergraphen einzuschränken.

Dazu können ähnliche Überlegungen wie bei den k -opt-Nachbarschaften angestellt werden. Zum Beispiel ist es sehr selten gewinnbringend, eine Transfermenge S_i aus der Tour $t(S_i)$ in eine andere Tour t_j zu verschieben, wenn alle Knoten von S_i „sehr weit“ von allen Knoten der Tour t_j entfernt sind. Die Frage, ob Knoten weit voneinander entfernt sind, könnte z.B. anhand von Nächste-Nachbarn-Listen entschieden werden.

Gegebenenfalls können derartige Einschränkungen auch weiter verschärft werden, ohne die Qualität der gefundenen Lösungen wesentlich zu beeinflussen. In jedem Fall kann man zuerst nur den reduzierten Transfergraphen aufbauen und darin nach Verbesserungen suchen. Erst wenn dies nicht gelingt, werden weitere Bögen hinzugenommen.

5.2.2 Meta-Heuristiken für die Diversifikation der Suche

In dieser Arbeit wurde fast ausschließlich die Lokale Suche im Sinne eines reinen Abstiegsverfahrens untersucht. Obwohl die erzielten Ergebnisse häufig

²vgl. Abschnitt 4.2.3 S. 131ff.

bereits mit denen erfolgreicher Meta-Heuristiken vergleichbar sind, steht die Notwendigkeit einer gezielten Diversifikation der Suche außer Frage. Durch eine Diversifikation kann das stets verbleibende Risiko, bei einzelnen Instanzen ein relativ schlechtes lokales Optimum zu erhalten, wesentlich reduziert werden. Zahlreiche Publikationen der letzten Jahre belegen eindrucksvoll, daß der Einsatz von Meta-Heuristiken für die Diversifikation sehr erfolgreich sein kann.

Insofern wurde in dieser Arbeit nur der erste Schritt zur Entwicklung einer sehr guten Verbesserungsheuristik getan. Einer Einbindung der beschriebenen Verfahren in verschiedene Meta-Heuristiken steht aber nichts im Wege, so daß eine hervorragende Ausgangssituation für viele weitere Entwicklungen geschaffen wurde.

Für eine Reihe von Meta-Heuristiken erscheint die Integration mit der Lexikographischen Suche in k -opt-Nachbarschaften bzw. mit Cyclic-Transfer-Nachbarschaften problemlos und natürlich. Dazu zählen z.B. Variable Neighborhood Search, Iterated-Local-Search, Guided-Local-Search und Dynamic-Contraction (siehe unten). Darüber hinaus werden auch in vielen anderen Meta-Heuristiken lokale Verbesserungsverfahren integriert (z.B. ES, Ameisensysteme [GTA99], [DS]) verwendet werden, wobei die Rolle der Lokalen Suche innerhalb der Meta-Heuristik unterschiedlich festgelegt werden kann.

Bei fast allen kombinatorischen Optimierungsproblemen zählen Tabu-Search-Verfahren zu den erfolgreichsten eingesetzten Meta-Heuristiken. Um beim Einsatz der Lexikographischen Suche eine Tabu-Liste zu berücksichtigen, besteht u.a. die Möglichkeit eine spezielle „Tabu-Ressource“ zu verwenden. Die Ressourcenverbräuche dieser neuen Ressource können nach jedem Nachbarschaftsschritt aktualisiert werden. So ist es möglich, ohne hohen Aufwand konkrete Nachbarschaftsschritte zu verbieten. Ebenso können auch, wie es häufig praktiziert wird, alle Lösungen tabu gesetzt werden, die bestimmte Bögen aus der Lösung eliminieren würden. Unter Verwendung abhängiger Ressourcen kann auch ein „threshold accepting“ in die Überprüfung integriert werden. Eine solche Modellierung hat den Vorteil, daß die Überprüfung der Tabu-Listen keinen signifikanten Mehraufwand impliziert und die Tabu-Restriktionen automatisch zur Effizienzsteigerung der Suche benutzt werden. Darüber hinaus überträgt sich die Überprüfung der Tabu-Listen dann ohne weitere Änderungen der Implementierung auch auf die Cyclic-Transfer-Nachbarschaften.

5.2.3 Fixierung von Bögen und Einsatz von Makro-Knoten für k -opt-Nachbarschaften

Es wurde bereits darauf hingewiesen, daß ein Vorteil der im Rahmen dieser Arbeit verwirklichten Konzepte darin besteht, daß Toursegmente genauso behandelt werden können, wie einzelne Knoten. Aus dieser generalisierenden Sichtweise lassen sich zwei Ansätze entwickeln.

Der erste Ansatz geht auf einen Vorschlag von Cordone und Wolfer Calvo in [CW97] zurück. Sie beschreiben, wie bei einem Routing Problem mit Zeitfensterrestriktionen die Zeitfenster für beliebige Toursegmente effizient überprüft werden können. Sie bezeichnen die Toursegmente als Makro-Knoten (Macronodes) und benutzen analog zu dem Herangehen von Savelsbergh „globale Variablen“, um die Zulässigkeit einer Verkettung von Makro-Knoten zu überprüfen. Sie schlagen vor, einen Teil oder alle der insgesamt n^2 in einer aktuell gegebenen Tour enthaltenen Makro-Knoten einschließlich der zugehörigen globalen Variablen einmalig zu bestimmen und zu speichern.

Der Vorteil besteht darin, daß anschließend jeder beliebige k -opt-Schritt aus diesen Makro-Knoten zusammengesetzt werden kann. Daher ist man nicht mehr an eine lexikographische Suchstrategie gebunden, um k -opt-Schritte mit konstantem Aufwand untersuchen zu können. Wenn die Konzepte aus Kapitel 3 verwendet werden, ist dieses Vorgehen auch direkt auf VRSP mit beliebigen beschränkten Ressourcen übertragbar. Auf diesem Ansatz aufbauende Suchstrategien werden in [CW97] nicht behandelt. Wenn die Bindung an die Lexikographische Suche wegfällt, kann man sich alternativ an einer Sequentiellen Suche oder dem Lin-Kernighan-Algorithmus orientieren. Gegebenenfalls können dadurch weitere Effizienzvorteile erzielt werden.

In einem zweiten Ansatz kann ausgenutzt werden, daß Toursegmente ein ideales Werkzeug zur Modellierung fixierter Bögen darstellen. Sind z.B. bereits mehrere Lösungen bekannt, kann man nach Bögen suchen, die in einigen oder allen guten Lösungen enthalten sind und diese zeitweise fixieren. Dadurch wird die Problemgröße reduziert, was es bei einem relativ kleinen resultierenden Problem ermöglicht, Nachbarschaften mit größerem k zu untersuchen.

Die Möglichkeit, die Darstellung eines Problems durch Fixierung bestimmter Lösungseigenschaften zu kontrahieren und damit zu verkleinern und ggf. später durch Aufhebung der Fixierungen wieder zu expandieren, ist eine in der Literatur immer wieder anzutreffende grundlegende Idee. Ansätze, die im Sinne einer Meta-Heuristik, Lösungen abwechselnd kontrahieren und spä-

ter wieder expandieren, tauchen unter Verwendung unterschiedlicher Begriffe mehrfach auf. So nennt Saab den in [Saa97] beschriebenen Ansatz „Dynamic Contraction“. Walshaw [Wal00], [Wal01] verfolgt einen ähnlichen Ansatz, den er als „Multilevel Refinement“ bezeichnet. Einer entsprechenden Idee folgt auch der in [RGP02b] entwickelte Operator „SMART“ für das VRPTW.

5.2.4 Integration mit Constraint Programming

Die Modellierung von Restriktionen ist in dieser Arbeit auf das Ressourcenmodell beschränkt. Noch flexibler sind Systeme, die auf dem Constraint Programming (CP) basieren und eine deklarative Beschreibung sehr unterschiedlicher Restriktionen erlauben. Damit ist es prinzipiell möglich auch Restriktionen zu formulieren, die mit Hilfe von Ressourcenbeschränkungen nicht oder nur sehr schwer darzustellen sind. Als Beispiel dafür seien multiple Zeitfenster oder andere multiple Ressourcenfenster angeführt.

Wir betrachten deshalb den in Kapitel 3 beschriebenen Ansatz zur Berücksichtigung beschränkter Ressourcen aus der Sicht eines CP-Systems. Die Prüfung der Zulässigkeit für die Ressourcenfenster kann als eine spezielle Propagation der entsprechenden Restriktionen aufgefaßt werden. Die Zulässigkeitsbereiche $[EAT_i^r, LAT_i^r]$ bzw. $[EDT_i^r, LDT_i^r]$ können als Domänen von Variablen betrachtet werden, die angeben, welche Werte für die Ressource r bei der Ankunft bzw. bei der Abfahrt am Knoten i möglich sind. Die Propagation der zugehörigen Constraints geschieht unter Ausnutzung der speziellen Struktur der Restriktionen und in enger Abstimmung mit der Suchstrategie. Daher kann die Propagation wesentlich effizienter (nämlich mit konstantem Aufwand) erfolgen.

Eine solche Sicht auf die in dieser Arbeit behandelten Ressourcenbeschränkungen impliziert einen bislang nicht behandelten Ansatz zur Integration einer Lokalen Suche mit einem CP-System, bei der die Effizienz der Suche trotz des sehr allgemeinen Ansatzes weitgehend erhalten bleiben kann.

Kapitel 6

Ergebnisse

In diesem Kapitel soll die Leistungsfähigkeit der in den Kapiteln 3 und 4 beschriebenen Konzepte überprüft werden. Durch zahlreiche Vergleichsrechnungen werden die einzelnen Vorschläge explizit getestet. Dabei wird analysiert, in wie weit die beabsichtigten Wirkungen erzielt werden können. Ein Anliegen dieses Kapitels ist außerdem, deutlich zu machen, welche (ggf. reduzierten) Nachbarschaften für bestimmte Problemtypen besonders geeignet sind.

Über den reinen Nachweis der Anwendbarkeit der Heuristiken hinausgehend, wird die Konkurrenzfähigkeit zu anderen Algorithmen durch einen Vergleich der Ergebnisse bei der Lösung von Benchmarkproblemen getestet. In einigen Abschnitten gehen wir außerdem darauf ein, wie einzelne Aspekte der bisher recht formal beschriebenen Konzepte praktisch umgesetzt werden.

Die Analysen beschränken sich auf Benchmarkprobleme zum TSPTW, VRP und zum VRPTW. Für diese Problemklassen existieren sehr viele Benchmarkprobleme unterschiedlicher Größe und Charakteristik. Die meisten dieser Probleme wurden mit zahlreichen z.T. sehr unterschiedlichen Algorithmen gelöst, so daß eine sehr gute Vergleichsbasis gegeben ist. Für einige Probleme sind sogar optimale Lösungen oder sehr gute untere Schranken bekannt. Diese erlauben es, die Qualität heuristisch gefundener Lösungen besser abzuschätzen.

Um den Umfang der Arbeit zu beschränken, werden die zahlreichen Möglichkeiten, durch die Hinzunahme beschränkter Ressourcen weitere VRSP modellieren und lösen zu können, nicht untersucht. Bei der im Rahmen dieser Arbeit vorgenommenen Implementierung wurden alle Kapazitäts- und Zeitfensterrestriktionen ausschließlich mit Hilfe des allgemeinen Ressourcen-

modells realisiert. Deshalb dokumentieren die durchgeführten Rechnungen die Realisierbarkeit und Anwendbarkeit der beschriebenen Konzepte.

Das Kapitel 6 ist wie folgt aufgebaut. Bevor mit der eigentlichen Analyse begonnen wird, gehen wir im Abschnitt 6.1 auf die generelle Problematik einer sinnvollen und fairen Bewertung heuristischer Verfahren ein. Anschließend werden in Abschnitt 6.2 die Probleminstanzen zum TSPTW, VRP und VRPTW beschrieben, die bei den Untersuchungen der folgenden Abschnitte als Benchmarkprobleme eingesetzt werden. Der Abschnitt 6.3 beschäftigt sich ausschließlich mit Untersuchungen zur Lokalen Suche mit k -opt-Nachbarschaften. Anhand einiger Instanzen des TSPTW und des VRP wird überprüft, welche Vorteile der Einsatz von k -opt-Nachbarschaften mit $k > 3$ bzw. k -opt*-Nachbarschaften mit $k > 2$ hat und welchen Einfluß die im Abschnitt 3.4 diskutierten Reduktionen der k -opt-Nachbarschaft auf Lösungsqualität und Ablaufgeschwindigkeit haben. Im Abschnitt 6.4 werden der kombinierte Einsatz von k -opt-Nachbarschaften und Cyclic-Transfer-Nachbarschaften beim VRP und beim VRPTW sowie die Einsetzbarkeit der CT-Nachbarschaften für das gezielte Auflösen von Touren untersucht. Abschließend wird im Abschnitt 6.5 anhand einer einfachen Meta-Strategie mit Kick-Schritte beispielhaft gezeigt, wie unter Verwendung der vorgestellten Konzepte, leistungsfähige hybride Heuristiken entwickelt werden können. Wir vergleichen die damit erzielten Ergebnisse mit den aktuell besten Meta-Heuristiken für TSPTW, VRP bzw. VRPTW.

6.1 Qualität von heuristisch bestimmten Lösungen

Heuristiken sind Verfahren, die die Optimalität ihrer Lösungen nicht gewährleisten können. In die Bewertung von Heuristiken gehen vor allem der Rechenaufwand bzw. die daraus resultierende Rechenzeit des Verfahrens sowie eine Beurteilung der Qualität der gefundenen Lösungen ein.

Für die Aufwandsabschätzung kann man z.B. eine *worst-case*- oder *average-case*-Analyse durchführen. Da dies für VRSP meistens sehr schwierig ist, bestimmt man stattdessen oft auch die Zahl der ausgeführten Basisoperationen oder die Dauer der Berechnungen auf einem bestimmten Computer, die bei der Anwendung der Algorithmen auf ein oder mehrere Benchmarkprobleme notwendig sind.¹

Wesentlich schwieriger ist es, die Qualität einer Lösung für eine Problem-
instanz oder darüber hinaus die Qualität der durch eine gegebene Heuristik
generierten Lösungen für eine Problemklasse zu beurteilen. Viele Heuristiken
besitzen eine stochastische Komponente und erzeugen deshalb bei wiederhol-
tem Aufruf für ein und dieselbe Probleminstanz unterschiedliche Lösungen.
Für die Bewertung der Lösungsqualität könnte man den Erwartungswert der
Zielfunktion verwenden. Dies muß jedoch nicht das einzige Kriterium sein,
denn abhängig vom Einsatzzweck einer Heuristik kann es z.B. durchaus sinn-
voll sein, die Streuung der Zielfunktionswerte ebenfalls zu berücksichtigen.
Die Lösungsqualität einer solchen Heuristik bezogen auf eine einzige Pro-
bleminstanz zu bewerten, stellt dann genau genommen ein multikriterielles
Problem dar.²

Sieht man von diesen Überlegungen zunächst ab, erscheint es plausibel,
die Qualität einer Lösung anhand der Abweichung des Zielfunktionswertes c
vom optimalen Zielfunktionswert c^* zu messen:

$$\Delta c = c - c^*. \quad (6.1)$$

Damit die Bewertung einer Lösung invariant gegenüber Skalenwechseln ist,
wird normalerweise die relative Abweichung vom Optimum verwendet:

$$\Delta c^{rel} = \frac{\Delta c}{c^*} = \frac{c - c^*}{c^*} \quad (6.2)$$

Gerade für große (und damit interessantere) Instanzen kombinatorischer
Probleme ist der optimale Zielfunktionswert c^* oft nicht bekannt. Da die rela-
tive Abweichung zum Optimum dann nicht bestimmt werden kann, verwendet
man stattdessen häufig die relative Abweichung vom Zielfunktionswert der
besten bislang bekannten Lösung oder von einer unteren Schranke.

¹In den meisten Publikationen werden die Rechenzeiten angegeben, die von den Au-
toren zur Lösung von Benchmarkproblemen benötigt wurden. Um eine Vergleichbarkeit
von verschiedenen Ergebnissen zu erreichen, ist es dann jedoch notwendig, die Rechen-
zeiten anhand der Leistungsparameter der verwendeten Computer zu skalieren. Praktisch
ist dies jedoch extrem schwierig, denn die Performance eines Rechners wird von sehr vie-
len z.T. schwer vergleichbaren Faktoren beeinflusst. Außerdem bleibt immer offen welche
zusätzlichen Aspekte die Rechenzeiten beeinflussen haben können (z.B. verwendete Compi-
lertypen und deren Parametereinstellungen, Effizienz der Programmierung).

²Bei deterministischen Heuristiken kommt man sofort zu derselben Problematik, sobald
die Lösungen mehrerer Probleminstanzen gleichzeitig bewertet sollen.

Die Qualität der Lösungen einer Heuristik allgemein zu bewerten, ist damit immer noch nicht möglich, weil sich die Größe Δc^{rel} nur auf eine einzelne Problemistanz bezieht. In Ermangelung besserer Alternativen wird deshalb üblicherweise die relative Abweichung der Zielfunktionswerte über eine Menge von Testproblemen betrachtet und ggf. gemittelt. Für die Bildung der Testmengen werden i.d.R. Benchmark-Probleme herangezogen, die in der Literatur veröffentlicht wurden und ggf. bereits von mehreren Autoren untersucht wurden. Dies erleichtert oft die Frage nach optimalen bzw. sehr guten bekannten Lösungen.

Bei dieser Form der Bewertung der Lösungen bleiben folgende Einschränkungen zu beachten:

1. Sind einige der optimalen Zielfunktionswerte c^* einer Testmenge nicht bekannt, so ist die Bewertung abhängig von der Kenntnis guter Lösungen bzw. Schranken.
2. Die Wahl der Testmenge hat ggf. entscheidenden Einfluß auf die Bewertung einer Heuristik.
3. Abhängig vom Einsatzzweck der Heuristik kann es wichtig sein, daß
 - die Abweichungen vom Optimum nicht zu stark schwanken oder
 - der Algorithmus auch bei frühzeitigem Abbruch der Rechnungen eine gute Lösung liefert.

In diesem Kapitel kann es deshalb nicht darum gehen, allgemein „die Qualität“ der untersuchten Heuristiken oder „die beste Heuristik“ zu bestimmen. Es soll stattdessen überprüft werden, ob sie zu anderen erfolgreichen Verfahren vergleichbare Ergebnisse erzielen können.

Bei der Entwicklung der Heuristiken wurde u.a. großer Wert auf die Anwendbarkeit auf m-TSPR mit relativ frei modellierbaren Ressourcenbeschränkungen gelegt. Da die Verfahren in diesem Sinne also wesentlich allgemeiner konzipiert sind, als dies für alle zum Vergleich herangezogenen Verfahren der Fall ist, *dürfen* sie die Spezifika eines bestimmten Modells nicht in gleicher Weise ausnutzen. Daher ist auch nicht zu erwarten, daß Verfahren, die für ein spezielleres Modell entwickelt wurden, dominiert werden. Man kann den Ansatz dagegen als sehr erfolgreich bezeichnen, wenn dessen hohe Flexibilität keine gravierenden Nachteile beim Lösungsverhalten nach sich zieht.

6.2 Auswahl der Probleminstanzen

Da die verwendeten Heuristiken für sehr unterschiedliche VRSP sinnvoll einsetzbar sein sollen, ist es wichtig, auch die Tests an mehreren verschiedenen Problemen durchzuführen. Es werden deshalb Probleminstanzen des TSPTW, VRP und des VRPTW verwendet.

Die Beschränkung auf diese drei Probleme begründet sich einerseits in der notwendigen Begrenzung des Testumfangs und andererseits in der Absicht, möglichst gut interpretierbare Ergebnisse zu erhalten. Instanzen des TSPTW und des VRP sind dafür besonders gut geeignet, weil jeweils nur eine beschränkte Ressource vorliegt. Mit Hilfe des VRPTW kann darüber hinaus getestet werden, ob auch eine Methodik auch auf mehrere beschränkte Ressourcen übertragen werden kann.

Für alle drei Probleme liegen Testinstanzen vor, die bereits von Autoren vieler unterschiedlicher Verfahren als Benchmark-Probleme benutzt wurden und zu denen deshalb sehr gute und teilweise auch bewiesene optimale Lösungen bekannt sind. Da eine große Zahl unterschiedlicher Tests durchgeführt werden soll, beschränken wir uns auf eine gut überschaubare Zahl von Probleminstanzen, für die bekannt ist, daß sie schwierig genug sind, um die Qualität einer Heuristik sinnvoll zu beurteilen.

6.2.1 TSP mit Zeitfenstern

Ascheuer et. al. publizierten in [AFG97] insgesamt 52 Probleminstanzen des TSPTW, die zwischen 11 und 232 Knoten und eine asymmetrische Abstandsmatrix besitzen. Das Optimierungsziel besteht in der Minimierung der insgesamt zurückgelegten Strecke. Die Autoren haben mit Hilfe eines Branch-and-Cut-Ansatzes untere Schranken für alle Probleminstanzen und für einige kleinere Probleme auch optimale Lösungen bestimmt.

Dieselben Probleminstanzen wurden auch in [BS01] untersucht. Balas und Simonetti beschreiben dort ein Dynamisches Programm, das einen großen Teil der Probleme optimal lösen konnte. Für einige große Instanzen sind sie gezwungen, eine heuristische Relaxation zu benutzen. Sie können auch dann sehr gute Lösungen bestimmen.³ Somit sind für alle betrachteten Probleme

³Das Verfahren liefert für fast alle Instanzen die besten bekannten Lösungen. Balas und Simonetti schreiben dazu: „... We conclude that for time-window problems our approach clearly advances the state of the art.“ (... Wir schließen daraus, daß unser Ansatz den Stand der Wissenschaft für Zeitfenster-Probleme deutlich verbessert.)

optimale Lösungen bzw. sehr gute heuristische Lösungen und untere Schranken bekannt.

Wir beschränken uns auf eine Auswahl von 15 Probleminstanzen. Diese wurden anhand von zwei Kriterien festgelegt wurden:

1. Die verwendete Implementierung der I1-Heuristik⁴ findet direkt eine zulässige Lösung des TSPTW.
2. Die Probleminstanz besitzt mindestens 50 Kundenknoten.

Um in erster Linie die Leistung der Verbesserungsheuristik beurteilen zu können, ist es sinnvoll mit einer *einfachen* Eröffnungsheuristik zu starten, die aber in der Lage sein muß, eine zulässige Lösung zu generieren. Dieses kann durch die Einführung des ersten Kriteriums sehr einfach realisiert werden. Das zweite Kriterium wurde gewählt, weil die Aussagekraft der kleineren Instanzen für die Beurteilung der Leistungsfähigkeit einer Heuristik gering eingeschätzt wird. Einige der kleinen Probleminstanzen werden bereits durch sehr einfache Verbesserungsverfahren optimal gelöst. Außerdem terminiert die Lokale Suche oft bereits nach wenigen Iterationen, so daß die Qualität der Lösung sehr stark von der gewählten Startlösung abhängig ist.

Eine Übersicht über die 15 Probleminstanzen gibt die linke Seite der Tabelle 6.2 auf Seite 173. Im Problemnamen ist jeweils die Zahl der Kundenknoten enthalten.⁵ Die Spalte „BS“ enthält die Zielfunktionswerte der besten von Balas und Simonetti gefundenen Lösungen und die Spalte „LB“ gibt den optimalen Zielfunktionswert (mit * gekennzeichnet) bzw. die von Ascheuer et. al. berechnete untere Schranke an.

6.2.2 VRP

Für Untersuchungen zum VRP werden die 14 von Christofides, Mingozi und Toth in [CMT79] veröffentlichten Probleminstanzen verwendet. Diese umfassen zwischen 50 und 199 Kundenknoten. Sie besitzen keine Zeitfenster, und als Entfernungen dienen Euklidische Abstände in der Ebene. Alle Instanzen besitzen Kapazitätsrestriktionen und die Hälfte der Instanzen eine zusätzliche Beschränkung der maximalen Tourlänge.

⁴Die I1-Heuristik ist eine häufig als Eröffnungsverfahren verwendete sequentielle Einfügeheuristik, die von Solomon [Sol87] vorgeschlagen wurde.

⁵Die Probleme rgb132, rgb152, rgb193 und rgb233 besitzen je zwei Kundenknoten weniger.

Die Instanzen dienen seit mehreren Jahrzehnten als Benchmarkprobleme. Mit Hilfe von Tabu-Search-Verfahren wurden die besten bislang bekannten Lösungen bestimmt. Dabei wurden meistens sehr lange Rechenzeiten benötigt. Das Granular Tabu Search von Toth und Vigo [TV98] bzw. [TV02] setzt mit wesentlich geringeren Rechenzeiten neue Maßstäbe. Einen Überblick die Problemeigenschaften und Ergebnisse der erfolgreichsten Tabu-Search-Verfahren gibt Tabelle 6.1.

Nr.	Typ ^a	TA/RT ^b	TR ^c		XK ^d		RR ^e	TV ^f		best	
		z [*]	z [*]	CPU ¹	z [*]	CPU ²	z [*]	z [*]	CPU ³		
1	50 C	524,61	524,61	6,0	524,61	29,22	524,61	524,61	0,81	524,61	^H
2	75 C	835,26	835,77	53,8	835,26	48,80	835,32	838,60	2,21	835,26	
3	100 C	826,14	829,45	18,4	826,14	71,93	827,53	828,56	2,39	826,14	
4	150 C	1028,42	1036,16	58,8	1029,56	149,90	1044,35	1033,21	4,51	1028,42	
5	199 C	1291,45 ^{RT}	1322,65	90,9	1298,58	272,52	1334,55	1318,25	7,50	1291,45	
6	50 CD	555,43	555,43	13,5	555,43	30,67	555,43	555,43	0,86	555,43	
7	75 CD	909,68	913,23	54,6	965,62	102,13	909,68	920,72	2,75	909,68	
8	100 CD	865,94	865,94	25,6	881,38	98,15	866,75	869,48	2,90	865,94	
9	150 CD	1162,55	1177,76	71,0	—	168,08	1164,12	1173,12	5,67	1162,55	
10	199 CD	1395,85 ^{RT}	1418,51	99,8	1439,29	368,37	1420,84	1435,74	9,11	1395,85	
11	120 C	1042,11	1073,47	22,2	1042,11	91,23	1042,11	1042,87	3,18	1042,11	
12	100 C	819,56	819,56	16,0	819,56	56,61	819,56	819,56	1,10	819,56	^G
13	120 CD	1541,14	1573,81	59,2	1618,55	201,75	1550,17	1545,51	9,34	1541,14	
14	150 CD	866,37	866,37	65,7	915,24	152,98	866,37	866,37	1,41	866,37	

^a Anzahl der Kunden, C: mit Kapazitätsrestriktion, D: mit beschränkter Tourlänge

^b Taillard [Tai93] bzw. Rochat und Taillard [RT95].

^c „Taburoute“ von Gendreau et. al. [GHL94].

^d Xu and Kelly [XK96] bzw. Golden et. al. [GWKC98].

^e Rego and Roucairol [RR96], parallele Implementierung.

^f Toth and Vigo [TV98].

¹ Minuten auf einer Silicon Graphics Workstation (36 MHz, 5,7 Mflops).

² Minuten auf einer DEC ALPHA Workstation (DEC OSF/1 v 3.0).

³ Minuten auf einem Pentium PC mit 200 MHz (etwa 3-mal schneller als die SG Workstation von ¹ und halb so schnell, wie die DEC ALPHA Workstation von ²).

^G Lösung ist optimal [GWKC98].

^H Lösung ist optimal [HCM95].

Tabelle 6.1: Ergebnisse einiger Tabu-Search Verfahren für die VRP-Instanzen von Christofides et. al. (die Tabelle wurde etwas verkürzt aus [LGPS00] übernommen)

6.2.3 VRP mit Zeitfenstern

Als Benchmark-Probleme zum VRPTW werden die Instanzen von Solomon [Sol87] benutzt. Die insgesamt 56 Problem instanzen besitzen jeweils genau 100 Kundenknoten, deren zeitliche und räumliche Abstände aus den Koordinaten in der Euklidischen Ebene bestimmt werden. Die Fahrzeuge besitzen eine beschränkte Kapazität und allen Kundenknoten sind Bedarfe und Zeitfenster zugeordnet.

Die Instanzen werden in sechs Kategorien unterteilt, die mit C1, C2, R1, R2, RC1 und RC2 bezeichnet werden. Dabei steht das „C“ für Instanzen, deren Kundenknoten in räumlichen Clustern angeordnet sind. Bei den mit „R“ gekennzeichneten Instanzen wurden die Koordinaten zufällig bestimmt. Die Kunden der „RC“-Instanzen sind zum Teil zufällig und zum Teil in Clustern angeordnet. Bei allen Instanzen hat das Depot eine zentrale Position.

Bei den Problem instanzen der Kategorien C1, R1 und RC1 besitzen die Fahrzeuge eine relativ kleine Kapazität, wodurch eine hohe Tourenzahl erzwungen wird (zwischen 9 und 19 Touren). Die deutliche höhere Fahrzeugkapazität der übrigen Problem instanzen aus C2, R2 und RC2 erlaubt stets Lösungen mit nicht mehr als zwei bis vier Touren.

Als Optimierungsziel wird in der Regel eine Lösung mit minimaler Tourenzahl und einer möglichst geringen Gesamtstrecke angesehen. Dabei ist die geringe Anzahl der Touren das dominierende Ziel. Im Gegensatz zu einigen früheren Veröffentlichungen ist es außerdem mittlerweile üblich, keine Rundung der Entfernungen vorzunehmen, was einen Vergleich der Ergebnisse vieler Autoren möglich macht.

Da die Problem instanzen von Solomon seit vielen Jahren von fast allen Autoren als Benchmark-Probleme benutzt wurden, existiert eine sehr breite Vergleichsbasis. Es ist deshalb davon auszugehen, daß die besten bekannten Lösungen bereits optimal oder sehr nahe am Optimum sind. Für einzelne Instanzen konnte die Optimalität der heuristisch ermittelten Lösungen bereits nachgewiesen werden [Irn02]. Dies betrifft insbesondere viele Instanzen der Kategorien C1 und C2. Diese Problem instanzen sind allerdings so einfach, daß sie auch von sehr vielen Heuristiken bereits optimal gelöst werden. Wir nehmen die Kategorien C1 und C2 deshalb von den meisten Betrachtungen aus.

6.3 Validierung der k -opt-Nachbarschaften

In diesem Abschnitt konzentrieren wir uns auf die Analyse der im Kapitel 3 beschriebenen Konzepte zur Durchführung einer Lokalen Suche für k -opt-Nachbarschaften. Wir gehen dabei folgenden Fragen nach:

1. Können durch eine Lokale Suche mit k -opt-Nachbarschaften mit $k > 3$ wesentliche Vorteile erzielt werden?
2. Wie gut sind die Lösungen, die durch die Erweiterung der k -opt-Nachbarschaften auf die Giant-Route erzielt werden können?
3. Wie groß sind die Aufwandseinsparungen, die mit Hilfe von Kostenabschätzungen erzielt werden können?
4. Welche Vor- und Nachteile sind mit der Reduktion der k -opt-Nachbarschaften verbunden?
5. Welche Vorteile bietet der Einsatz von k -opt*-Nachbarschaften bei der Lösung von m-TSPR?

Für die Beantwortung dieser Fragen werden i.d.R. Vergleichsrechnungen mit einer Reihe von Testinstanzen durchgeführt werden. Um die Übersichtlichkeit zu wahren und den Fokus auf die wesentlichen Effekte zu konzentrieren, beschränken wir uns bei der Darstellung in dieser Arbeit teilweise auf die Angabe von Mittelwerten für die Zielfunktionswerte (z) und die Rechenzeiten in Sekunden. Um einen rechnerunabhängigen Vergleich der Ergebnisse zu ermöglichen, geben wir zusätzlich die durchschnittliche Zahl der berechneten Verkettungsoperationen (VO) an. Bevor die einzelnen Untersuchungen durchgeführt werden, gehen wir auf das zugrundeliegende Schema für die Lokale Suche in k -opt-Nachbarschaften ein.

6.3.1 Eine VND für die Lokale Suche

Für die praktische Durchführung der Lokalen Suche mit einer k -opt-Nachbarschaft wurde nicht direkt der Algorithmus 3.1 für die Lexikographische Suche verwendet. Stattdessen wird eine Erweiterung im Sinne einer Variable-Neighborhood-Descent-Heuristik (VND)⁶ vorgenommen und die Lexikographische

⁶die VND wurde im Abschnitt 2.3 beschrieben, siehe Seite 55.

Suche auf eine Folge von (reduzierten) k -opt-Nachbarschaften angewandt. Dieses Vorgehen wird im folgenden erläutert:

Zuerst werden alle zu betrachtenden k -opt-Typen in einer Liste hierarchisch geordnet. Dies geschieht so, daß k -opt-Typen mit kleinerem k stets vor solchen k -opt-Typen mit größerem k stehen. Bei k -opt-Typen mit gleichem k soll die Zahl der invertierten Segmente aufsteigend sein. Bei der Durchführung aller 2-opt- und 3-opt-Schritte, d.h. für die komplette 3-opt-Nachbarschaft sieht diese Liste z.B. so aus: $([aBa], [acba], [aCba], [acBa], [aBCa])$. Man beachte, daß nur die „echten“ k -opt-Typen berücksichtigt werden.

Anschließend wird schrittweise für jeden k -opt-Typ dieser Liste eine vollständige Lokale Suche durchgeführt. Dafür wird der Algorithmus 3.1 eingesetzt. Nach jedem dieser Schritte ist die aktuelle Lösung bezüglich des zuletzt betrachteten k -opt-Typs lokal optimal. Wurde in einem dieser Schritte wenigstens eine Verbesserung gefunden, so erfolgt sofort ein Rücksprung zum Anfang der Liste, d.h. in unserem Beispiel zum k -opt-Typ $[aBa]$. Die Prozedur wird solange fortgesetzt, bis keine Verbesserungen mehr möglich sind. Die dann vorliegende Lösung ist lokal optimal bezüglich aller Nachbarschaften, die durch die k -opt-Typen der Liste vorgegeben sind.

Diese spezielle VND kann in folgendem Algorithmus zusammengefaßt werden:

Algorithmus 6.1. VND für k -opt-Nachbarschaften

1. Erzeuge eine Liste l aller zu berücksichtigenden k -opt-Typen.
2. Sortiere die Liste nach den oben formulierten Kriterien, d.h. lexikographisch aufsteigend nach der Zahl k und der Anzahl der invertierten Segmente und bezeichne die Einträge von l mit $(l_1, \dots, l_m)$.
3. Setze $i := 1$.
4. (*Schleife*) Wenn $i \leq m$, wiederhole:
 - 4.1 Führe mit Algorithmus 3.1 eine Lexikographische Suche ausschließlich mit dem k -opt-Typ l_i durch.
 - 4.2 Falls eine Verbesserung gefunden wurde, dann gehe zu 3.
 - 4.3 Setze $i := i + 1$.
5. Stopp.

Durch den Einsatz des Algorithmus 6.1 wird versucht, eine Reihe plausibler Überlegungen und Annahmen zu berücksichtigen:

1. Bei der Lexikographischen Suche gemäß Algorithmus 3.1 kann der weitere Aufbau eines Segments (d.h. die Fortsetzung der entsprechenden Schleife) häufig zeitiger abgebrochen werden, wenn der zu untersuchende k -opt-Typ von vornherein feststeht. Durch die Aufteilung der Nachbarschaften nach verschiedenen k -opt-Typen kann deshalb die Effizienz der Lexikographischen Suche verbessert werden. Darüber hinaus wird die Berücksichtigung der nachfolgenden Überlegungen teilweise erst durch diese Aufteilung möglich.
2. Mit den kleineren Nachbarschaften zu beginnen, hat den Vorteil, daß oft eine große Zahl notwendiger und leicht zu findender Verbesserungen bereits zu Beginn und ohne hohen Aufwand bestimmt werden.
3. Nachbarschaften, die ein oder mehrere Segmente invertieren, haben (falls die Lösung bereits 2-optimal ist) oft eine geringe Aussicht auf Erfolg, weil die zulässigen verbessernden Invertierungen evtl. bereits alle durchgeführt sind. Sie werden daher in der Liste weiter nach hinten gestellt und somit tendenziell seltener untersucht.
4. Nachbarschaften mit kleinem k abzusuchen, verursacht einen verhältnismäßig geringen Aufwand und kann deshalb auch mehrfach wiederholt werden, ohne die gesamte Rechenzeit deutlich zu beeinflussen. Dies motiviert den Rücksprung zum Anfang der Liste im Schritt 4.2 von Algorithmus 6.1, sobald eine durchgeführte Lokale Suche eine verbesserte der Lösung hervorbringt.
5. Da ein Wechsel der Nachbarschaft jeweils mit dem erneuten sukzessiven Aufbau der Toursegmente beim Neustart der Lexikographischen Suche verbunden und damit potentiell rechenintensiv ist, wird in der einem k -opt-Typ zugehörigen Nachbarschaft im Schritt 4.1 immer bis zum Auffinden des lokalen Minimums gesucht, bevor man zu einer neuen Nachbarschaft übergeht.

Darüber hinaus deuten die bei zahlreichen Testrechnungen gesammelten Erfahrungen darauf hin, daß es oft vorteilhaft ist, bei der Lokalen Suche eine Bestensuche-Strategie zu verfolgen.⁷ Wir verwenden daher grundsätzlich die Bestensuche.

⁷Es sei betont, daß es sich hierbei um eine reine Vermutung handelt, die durch experimentelle Erfahrungen motiviert ist. Zusätzlich sei lediglich darauf hingewiesen, daß die „worst-case“-Komplexität, bei der Erstensuche und der Bestensuche gleich ist.

6.3.2 Leistungsfähigkeit der k -opt-Nachbarschaften

Es soll zunächst untersucht werden, welcher Qualitätsgewinn durch den Einsatz von kleinen und mittleren k -opt-Nachbarschaften überhaupt erzielt werden kann. Wir beschränken uns dabei zunächst auf Nachbarschaften mit $k \leq 4$, da der Rechenaufwand für die Untersuchung der vollständigen k -opt-Nachbarschaften für die größeren Probleme sonst zu stark wächst.

TSPTW Die Ergebnisse dieser Tests sind auf der rechten Seite von Tabelle 6.2 zu finden. Die Werte in der Spalte „I1“ geben die Zielfunktionswerte der Lösungen der I1-Heuristik an. Die Ergebnisse der folgenden Spalten wurden erhalten, indem zuerst mit der I1-Heuristik eine Startlösung bestimmt und diese durch eine Lokale Suche mit der vollständigen 2-opt-, 3-opt- bzw. 4-opt-Nachbarschaft verbessert wurde.

Die letzten drei Zeilen der Tabelle geben den Mittelwert der Zielfunktionswerte ($\bar{z}$), der Rechenzeiten (CPU) und der Zahl der benötigten Verkettungsoperationen (VO) in Millionen über alle 15 Probleminstanzen an.

Die Zielfunktionswerte der I1-Heuristik weichen im Mittel mehr als 45 Prozent von den von Balas und Simonetti publizierten Lösungen ab. Nach der Verbesserung mit der 2-opt- bzw. der 3-opt-Nachbarschaft sinkt die Abweichung auf 21 Prozent bzw. auf 4 Prozent. Durch den Einsatz der 4-opt-Nachbarschaft kann die Abweichung auf durchschnittlich 3 Prozent reduziert werden. Die Schwankungsbreite der Abweichungen nimmt mit zunehmender Problemgröße etwas zu. Bei der Verwendung der 4-opt-Nachbarschaft sind die Lösungen aber bei keiner Probleminstanz mehr als 4 Prozent von der unteren Schranke (Spalte LB) entfernt.

Erwartungsgemäß steigen die Rechenzeiten beim Übergang zu größeren Nachbarschaften deutlich. Dennoch fällt auf, daß der Zuwachs von der 2-opt- zur 3-opt-Nachbarschaft (mit einem Faktor von etwa 50) wesentlich drastischer ausfällt, als dies zwischen der 3-opt- und der 4-opt-Nachbarschaft der Fall ist (weniger als Faktor 2). Dies ist dadurch zu erklären, daß beim Absuchen der 3-opt-Nachbarschaft sehr viele Verbesserungen gefunden und durchgeführt werden. Dies ist in der Tabelle nur indirekt an der sehr deutlichen Verbesserung der Zielfunktionswerte abzulesen, die nur durch viele verbessernde Schritte erreicht wird.

Bei allen betrachteten Probleminstanzen wurden durch den Übergang zur 4-opt-Nachbarschaft Verbesserungen erzielt. Die Ergebnisse machen deutlich, daß durch eine Beschränkung der k -opt-Nachbarschaften auf $k = 2$ oder $k = 3$

	BS ^a	LB ^b	I1 ^c	2-opt ^d	3-opt ^d	4-opt ^d
rbg50a	431	414*	579 (0,2)	563 (0,2)	439 (2)	433 (9)
rbg55a	814	814*	846 (0,2)	838 (0,2)	838 (0)	826 (3)
rbg67a	1048	1048*	1235 (0,3)	1085 (0,3)	1052 (1)	1051 (10)
rbg88a	1153	1153*	1552 (0,7)	1231 (0,8)	1165 (11)	1164 (29)
rbg92a	1093	1093*	1682 (0,7)	1453 (0,8)	1118 (31)	1105 (43)
rbg125a	1409	1409*	2164 (1,8)	1544 (2,0)	1424 (49)	1420 (106)
rbg132	1360	1360*	1665 (2,1)	1609 (2,3)	1444 (45)	1443 (82)
rbg132.2	1083	1069	1357 (2,1)	1347 (2,1)	1175 (61)	1145 (141)
rbg152	1783	1783*	2608 (3,4)	2067 (3,8)	1815 (115)	1796 (217)
rbg152.2	1628	1598	1888 (3,6)	1870 (3,7)	1708 (84)	1685 (222)
rbg152.3	1542	1526	1879 (3,6)	1879 (3,7)	1643 (116)	1618 (201)
rbg172a	1799	1788	2896 (4,6)	2143 (4,9)	1854 (197)	1826 (377)
rbg193.2	2018	1969	3343 (7,1)	2603 (8,1)	2145 (401)	2143 (575)
rbg201a	2189	2158	3452 (7,5)	2638 (8,5)	2253 (383)	2225 (676)
rbg233.2	2190	2146	4122 (12,5)	3341 (12,9)	2313 (1254)	2257 (1670)
$\bar{z}$	1436	1421,9	2084,5	1747,4	1492,4	1475,8
CPU	—	—	3,36	3,61	183,8	291,2
VO [10 ⁶]	—	—	0,17	0,20	65,2	103,8

^a beste Lösung von Balas und Simonetti [BS01].

^b optimale Lösung (mit *) oder untere Schranke von Ascheuer et. al. [AFG97].

^c I1-Heuristik von Solomon [Sol87] (eigene Implementierung).

^d I1-Heuristik und Verbesserung mit der angegebenen k -opt-Nachbarschaft.

Rechenzeiten in Klammern sind Sekunden auf einem PC mit AMD-Duron-Prozessor (700MHz).

Tabelle 6.2: Lokale Suche mit vollständigen k -opt-Nachbarschaften beim asymmetrischen TSPTW, Probleminstanzen von Ascheuer et. al.

wesentliche Verbesserungspotentiale verloren gehen können.⁸ 4-opt-Schritte sind deshalb für TSPTW i.a. interessant.

Weil die Rechenzeiten für die größeren Probleminstanzen bereits ab $k = 5$ extrem wachsen, können die Tests nicht in gleicher Form fortgeführt werden. Durch verschiedene Maßnahmen zur Beschleunigung der Rechnungen und zur Einschränkung der Nachbarschaften, wie sie in Abschnitt 3.4 beschrieben wurden, läßt sich der Rechenaufwand aber reduzieren. Dann können k -opt-Nachbarschaften mit $k > 4$ untersucht werden, was zu weiteren Verbesserungen der Lösungsqualität führt. Wir verweisen auf die Ergebnisse

⁸Eine Verbesserung von ca. 1 Prozent wird hierbei als wesentlich bezeichnet, weil das gesamte Verbesserungspotential (Abstand der 3-optimalen Lösung zu einer unteren Schranke) nicht mehr als etwa 5 Prozent beträgt.

VRP Analog zum TSPTW wurden Vergleichsrechnungen für das VRP anhand der Instanzen von Christofides et. al. durchgeführt. Im Unterschied zum TSPTW sind jetzt zwei Möglichkeiten, für die Lokale Suche zu unterscheiden. Die k -opt-Schritte können entweder nur innerhalb einer einzelnen Tour untersucht (Intra-Tour-Nachbarschaften) oder auf die Giant-Route angewandt werden. Letzteres erlaubt auch den Tausch von Kundenknoten zwischen verschiedenen Touren (Inter-Tour-Schritte).

Nr.	best ^a	SW0 ^b	SW2 ^c	SW3 ^d	SW4 ^e
1	524,6	812,8 (0,2)	655,7 (0,3)	531,9 (0,4)	531,9 (1,0)
2	835,3	1255,5 (0,5)	950,8 (0,5)	904,3 (0,6)	904,3 (0,9)
3	826,1	1361,2 (1,0)	978,1 (1,1)	868,3 (1,5)	868,3 (4,7)
4	1028,4	2069,6 (2,1)	1408,1 (2,3)	1103,4 (2,9)	1103,4 (8,2)
5	1291,5	2598,7 (3,5)	1770,2 (3,7)	1452,7 (4,3)	1452,7 (9,0)
6	555,4	860,4 (0,2)	706,9 (0,3)	687,1 (0,3)	687,1 (0,5)
7	909,7	1442,2 (0,4)	1232,3 (0,5)	1230,6 (0,5)	1230,6 (0,6)
8	865,9	1503,5 (0,8)	1210,7 (0,9)	1077,2 (1,1)	1077,2 (1,8)
9	1162,6	2357,0 (1,7)	1754,8 (1,9)	1658,0 (2,0)	1658,0 (2,7)
10	1395,9	3067,1 (3,0)	2263,2 (3,1)	2148,9 (3,4)	2148,9 (4,4)
11	1042,1	3111,5 (1,5)	1459,2 (1,7)	1277,1 (2,6)	1275,7 (13,7)
12	819,6	1095,7 (0,9)	972,8 (1,0)	916,3 (1,3)	916,3 (3,1)
13	1541,1	3300,5 (1,2)	2265,7 (1,3)	2205,9 (1,5)	2205,9 (2,3)
14	866,4	1146,9 (0,9)	1005,2 (1,0)	947,5 (1,1)	947,5 (2,1)
$\bar{z}$	976,0	1855,9	1331,0	1215,0	1214,8
CPU	—	1,29	1,39	1,68	3,93
VO [10 ⁶]	—	0,23	0,24	0,31	1,02

^a beste bekannte Lösung (aus [LGPS00] übernommen).

^b Sweep-Verfahren ohne Reoptimierung der Touren.

^c Sweep-Verfahren, Reoptimierung der Touren mit der 2-opt-Nachbarschaft.

^d Sweep-Verfahren, Reoptimierung der Touren mit der 3-opt-Nachbarschaft.

^e Sweep-Verfahren, Reoptimierung der Touren mit der 4-opt-Nachbarschaft.

Rechenzeiten in Klammern sind Sekunden auf einem PC mit AMD-Duron-Prozessor (700MHz).

Tabelle 6.3: Lokale Suche mit Intra-Tour-Nachbarschaften beim VRP, Problem instanzen von Christofides et. al.

Wir vergleichen zunächst den Einsatz unterschiedlicher k -opt-Nachbarschaften ausschließlich in der Form von Intra-Tour-Nachbarschaften (siehe Tabelle 6.3). Die Lösungen der Sweep-Heuristik sind ohne die Reoptimierung der Touren bei einzelnen Instanzen bis zu 200 Prozent und im Durchschnitt mehr als 80 Prozent von der besten bekannten Lösung entfernt. Durch die

Reoptimierung mit der 2-opt-Nachbarschaft kann der Abstand sehr schnell (im Mittel 0,1 Sekunden für die Verbesserung) auf durchschnittliche 33 Prozent verkürzt werden. Wird die größere 3-opt-Nachbarschaft verwendet, dann reduziert sich der mittlere Abstand zu den besten bekannten Lösungen nochmals deutlich auf etwa 22 Prozent. Die für die Verbesserung aufgewandte Rechenzeit ist mit durchschnittlich 0,7 Sekunden wesentlich größer, aber insgesamt noch immer sehr gering. Der Übergang zur 4-opt-Nachbarschaft hat so gut wie keine Verbesserung mehr zur Folge. Scheinbar kann das Verbesserungspotential, das mit Intra-Tour-Schritten erschließbar ist, für die (nicht sehr langen) Touren bereits mit der 3-opt-Nachbarschaft weitgehend ausgeschöpft werden. Der erforderliche Aufwand für die Untersuchung der 4-opt-Nachbarschaft ist deutlich höher als bei der 3-opt-Nachbarschaft, erscheint aber noch akzeptabel. Dabei ist jedoch zu berücksichtigen, daß dies sehr stark von der Länge der einzelnen Touren abhängig ist.

Wir führen dieselbe Untersuchung nun unter Verwendung der k -opt-Nachbarschaften auf der Giant-Route durch. Dabei werden noch keine k -opt*-Schritte berücksichtigt. Die Ergebnisse sind in Tabelle 6.4 zusammengefaßt.

Als Startlösung wird die Lösung verwendet, die mit der Sweep-Heuristik und einer anschließenden 3-opt-Suche auf den einzelnen Touren erhalten wird. Diese Lösung wird mit einer k -opt-Suche auf der Giant-Route weiter verbessert.

Die Suche mit der 2-opt-Nachbarschaft auf der Giant-Route reduziert die Abweichungen zu den besten bekannten Lösungen von 22 Prozent auf durchschnittlich etwa 11 Prozent. Die dafür aufgewandte Rechenzeit beträgt bei allen Instanzen weniger als 30 Sekunden. Weitere Verbesserungen sind mit der 3-opt-Nachbarschaft erzielbar. Die Abweichungen liegen jetzt nur noch bei durchschnittlich etwa 8 Prozent, wobei die mittlere Rechenzeit auf 86 Sekunden anwächst. Bei der Lokalen Suche mit der 4-opt-Nachbarschaft führt das exponentielle Wachstum der Größe der Nachbarschaft bei den größeren Instanzen zu Rechenzeiten von mehreren Stunden, die als inakzeptabel angesehen werden können. Die Abweichung zu den besten bekannten Lösungen kann im Mittel auf etwa 4,5 Prozent gesenkt werden. Bei der Instanz 11 beträgt sie aber immer noch fast 13 Prozent.

► Die Ergebnisse zeigen, daß die Ausdehnung der k -opt-Nachbarschaften auf die Giant-Route durchaus wirkungsvoll ist. Die Tatsache, daß bei einigen Instanzen auch für $k = 4$ relativ schlechte Lösungen erhalten wer-

Nr.	best ^a	SW3 ^b	SW3+GR2 ^c	SW3+GR3 ^d	SW3+GR4 ^e
1	524,6	531,9 (0,4)	531,9 (0,4)	531,9 (1,0)	531,9 (22)
2	835,3	904,3 (0,6)	897,2 (1,8)	893,8 (11,1)	875,6 (327)
3	826,1	868,3 (1,5)	854,2 (2,5)	846,0 (15,7)	837,9 (519)
4	1028,4	1103,4 (2,9)	1087,6 (5,2)	1069,6 (48,5)	1069,5 (2122)
5	1291,5	1452,7 (4,3)	1450,5 (6,4)	1423,7 (198,7)	1362,7 (15619)
6	555,4	687,1 (0,3)	613,5 (1,0)	571,0 (3,0)	566,0 (38)
7	909,7	1230,6 (0,5)	1004,0 (2,6)	974,6 (7,8)	944,7 (340)
8	865,9	1077,2 (1,1)	967,0 (3,0)	895,8 (33,1)	873,9 (747)
9	1162,6	1658,0 (2,0)	1323,5 (9,6)	1273,1 (186,6)	1226,7 (4423)
10	1395,9	2148,9 (3,4)	1624,9 (23,1)	1538,4 (610,1)	1458,7 (24637)
11	1042,1	1277,1 (2,6)	1275,3 (3,6)	1274,3 (18,5)	1176,6 (900)
12	819,6	916,3 (1,3)	906,3 (1,7)	880,0 (32,2)	837,5 (1162)
13	1541,1	2205,9 (1,5)	1857,8 (5,7)	1852,0 (24,0)	1642,2 (4524)
14	866,4	947,5 (1,1)	947,5 (1,2)	938,2 (11,1)	924,6 (379)
$\bar{z}$	976,0	1215,0	1095,8	1068,7	1023,5
CPU	—	1,68	4,85	85,8	3983
VO [10 ⁶]	—	0,31	0,85	23,6	1245

^a beste bekannte Lösung (aus [LGPS00] übernommen).

^b Sweep-Heuristik, Reoptimierung der Touren mit der 3-opt-Nachbarschaft.

^c SW3 wie bei ^b, Reoptimierung der Giant-Route mit der 2-opt-Nachbarschaft.

^d SW3 wie bei ^b, Reoptimierung der Giant-Route mit der 3-opt-Nachbarschaft.

^e SW3 wie bei ^b, Reoptimierung der Giant-Route mit der 4-opt-Nachbarschaft.

Rechenzeiten in Klammern sind Sekunden auf einem PC mit AMD-Duron-Prozessor (700MHz).

Tabelle 6.4: Lokale Suche in vollständigen k -opt-Nachbarschaften beim VRP, Probleminstanzen von Christofides et. al.

den, deutet darauf hin, daß es auch mit dieser recht großen Nachbarschaft nicht immer gelingt, eine ungünstige Verteilung der Kunden auf die einzelnen Touren zu korrigieren.

6.3.3 Auswirkungen effizienzsteigernder Maßnahmen

Im folgenden sollen die Auswirkungen der in den Abschnitten 3.1.5 und 3.4 beschriebenen Maßnahmen zur Effizienzsteigerung einerseits auf die Laufzeit und andererseits auf die Lösungsqualität der Algorithmen untersucht werden.

Aufwandsreduktion bei der Lexikographischen Suche

Selbst eine erhebliche Reduktion der Größe einer k -opt-Nachbarschaft muß nicht zwangsläufig dazu führen, daß diese anschließend schneller abgesucht werden kann. Um tatsächlich Aufwand zu sparen, muß die Reduktion im Einklang mit der verwendeten Suchstrategie vorgenommen werden. Umgekehrt kann die Suche ggf. auch beschleunigt werden, ohne die Größe der Nachbarschaft überhaupt zu ändern.

Wir überlegen zunächst, wie eine Reduktion des Aufwands bei der Durchführung einer Lexikographischen Suche überhaupt erreicht werden kann. Dazu betrachten wir den Algorithmus 3.1 der die Lexikographische Suche realisiert. In diesem Algorithmus werden k ineinander verschachtelte Schleifen abgearbeitet. Jede dieser Schleifen dient dazu, unter Beibehalten der bereits gebildeten Segmente ein weiteres Toursegment Knoten für Knoten aufzubauen. Eine Einsparung läßt sich genau dann erzielen, wenn der Aufbau eines Toursegments nicht bis zu Ende fortgeführt und die entsprechende Schleife abgebrochen werden kann. Diese Einsparungen sind besonders groß, wenn solch ein Abbruch innerhalb einer der äußeren Schleifen vorgenommen wird und alle inneren Schleifen (an dieser Stelle) ebenfalls nicht durchlaufen werden.

Um festzustellen, ob eine Schleife abgebrochen werden kann, wird unmittelbar bei der Erweiterung des zugehörigen Segments überprüft, ob eine dieses Segment betreffende vorgegebene Forderung verletzt wird. Angenommen, innerhalb der i -ten Schleife wird festgestellt, daß das Segment σ_i eine bestimmte Forderung nicht erfüllt.⁹ Dann brauchen die nachfolgenden Segmente $\sigma_{i+1}, \dots, \sigma_{k-1}$ nicht erst konstruiert werden, weil die Verletzung dadurch nicht aufgehoben werden kann. Alle weiteren Schleifen werden deshalb übersprungen und Segment σ_i unmittelbar um einen weiteren Knoten verlängert.

Falls das aktuelle Segment σ_i nicht nur eine bestimmte Forderung verletzt, sondern außerdem feststeht, daß diese Verletzung durch das Verlängern des Segments σ_i nicht mehr aufgehoben werden kann, wird die aktuelle Schleife sogar direkt abgebrochen. Dann werden die Berechnungen mit der vorhergehenden Schleife, fortgesetzt und das Segment σ_{i-1} um einen Knoten

⁹Entsprechend der im Kapitel 3 eingeführten Schreibweise, werden die Segmente mit $\sigma_0, \dots, \sigma_k$ bezeichnet. Dementsprechend seien die k Schleifen ebenfalls mit $0, \dots, k-1$ numeriert. Das letzte Segment σ_k wird nicht knotenweise aufgebaut, sondern vorab für alle Endsegmente bestimmt. Deshalb werden tatsächlich nur k Schleifen benötigt.

erweitert.

Eine stets zu berücksichtigende Forderung ist die Zulässigkeit der Lösung (d.h. insbesondere jedes einzelnen Segments der Lösung). Es können aber auch zusätzliche Restriktionen für die zu bildenden Segmente formuliert und die k -opt-Nachbarschaften auf diese Weise gezielt eingeschränkt werden. Durch das Abbrechen der Schleifen wird gewährleistet, daß sich die Einschränkung der Nachbarschaft auch in einer entsprechenden Verkürzung der Rechenzeit niederschlägt. Aufgrund der Konstruktion der Lexikographischen Suche tragen Abbrüche innerhalb der äußeren Schleifen am wirksamsten zur Reduktion des Aufwands bei.

Man beachte, daß Restriktionen, die nicht an einem einzelnen Segment überprüft werden können, wesentlich schwieriger in die Lexikographische Suche zu integrieren sind. Dies betrifft z.B. die Forderung, nur Bögen zu den „nächsten Nachbarn“ eines Knotens zuzulassen, auf die wir noch eingehen werden. Weiterhin ist zu berücksichtigen, daß die zusätzlichen Restriktionen bei jeder Erweiterung eines Segments überprüft werden sollen. Dies muß deshalb sehr schnell (auf jeden Fall in konstanter Zeit) durchführbar sein.

Beschleunigung durch Kostenabschätzungen

Im Abschnitt 3.1.5 werden eine Reihe von Betrachtungen angestellt, die bei der Implementierung der Lexikographischen Suche berücksichtigt wurden. So wird die dort angesprochene Aufteilung der Lexikographischen Suche auf die einzelnen k -opt-Typen bereits mit dem Algorithmus 6.1 realisiert. In diesem Unterabschnitt wird untersucht, welche Auswirkungen die auf Seite 88 beschriebenen Abbrüche aufgrund von Kostenabschätzungen auf die Laufzeit der Lexikographischen Suche haben.

Innerhalb der Lexikographischen Suche kann in jeder Schleife versucht werden, die minimalen Kosten abzuschätzen, die jede Nachbarlösung aufweisen muß, die die bereits festgelegten Segmente enthält. Je eher abgeschätzt werden kann, daß keine Verbesserung mehr möglich ist, desto mehr Rechenaufwand kann dadurch gespart werden. Umgekehrt können um so bessere Abschätzungen gemacht werden, je mehr Segmente bereits feststehen. Dabei ist zu beachten, daß eine Kostenabschätzung nur dann in den Ablauf der Lexikographischen Suche integriert werden kann, wenn sie mit konstantem Aufwand durchgeführt werden kann.

Kostenabschätzung für nicht invertierte Segmente Wir gehen davon aus, daß die mit einem k -opt-Schritt verbundene Kostenänderung anhand einer Bilanz der Kosten der gelöschten und eingefügten Bögen bestimmt werden kann. Für die Abschätzung dieser Kosten können zunächst alle zum aktuellen Zeitpunkt bereits feststehenden Bögen mit ihren tatsächlichen Kosten eingerechnet werden. Dazu gehören alle gelöschten Bögen, mit denen die bereits feststehenden Toursegmente untereinander verbunden waren. Wenn der k -opt-Typ feststeht, können auch die Bögen einkalkuliert werden, mit denen bereits bekannte Segmente untereinander verbunden werden müssen.

Bögen, die ein bereits bekanntes Segment mit einem noch nicht feststehenden Segment verbinden sollen, können relativ einfach durch den billigsten in Frage kommenden Bogen abgeschätzt werden. Die eigentliche Schwierigkeit besteht darin, in konstanter Zeit abzuschätzen, welche zusätzlichen Kosten für die Einbindung der in den weiteren Schleifen der Lexikographischen Suche noch zu bildenden Segmente mindestens entstehen müssen. Dabei ist zu diesem Zeitpunkt noch nicht festgelegt, wie aus dem verbliebenen „Restsegment“ die weiteren Segmente aufgebaut werden. Es steht weder fest, welche Bögen dabei gelöscht werden, noch, welche Bögen schließlich einzufügen sind.

Die wesentliche Idee besteht darin, die maximale Einsparung zu bestimmen, die durch einen Austausch *eines* Bogens des noch nicht aufgeteilten Restsegments gegen einen beliebigen anderen Bogen (mit gleichem Startknoten) erzielt werden kann. Da jederzeit bekannt ist, wieviele Bögen noch getauscht werden müssen, kann daraus eine obere Schranke für die noch zu erzielenden Kosteneinsparungen bestimmt werden.

Bei einer direkten Umsetzung dieser Idee müßten für die Feststellung dieser maximalen Einsparung alle Bögen des Restsegments untersucht werden. Der Aufwand für die Abschätzung würde dann linear mit der Länge des Restsegments wachsen. Stattdessen wird für die Berechnung dieser Information eine zusätzliche (künstliche und unbeschränkte) Ressource eingeführt, die wie alle anderen Ressourcen während der Lexikographischen Suche mit konstantem Aufwand aktuell gehalten werden kann.¹⁰

¹⁰Die Ressource verwaltet die Information, welche Einsparung durch den Austausch eines Bogens des jeweiligen Segments maximal erzielt werden kann. Beim Verketteten zweier Segmente ergibt sich der Wert der Ressource für das neue Segment als Maximum der Werte der zu verkettenden Segmente und der maximalen Einsparung, die dadurch erzielt werden kann, daß der neue Bogen ausgetauscht wird, der die beiden Segmente verbindet.

Kostenabschätzung für invertierte Segmente Da längere invertierte Segmente beim Vorliegen enger Zeitfenster selten zulässig sind, können die Schleifen bei der Lexikographischen Suche oft schon deshalb zeitig abgebrochen werden, weil ein aufgebautes Segment unzulässig geworden ist. k -opt-Typen die ein oder mehrere Segmente invertieren tragen deshalb i.d.R. nur unwesentlich zum gesamten Rechenaufwand bei. Wir verwenden deshalb bei der Abschätzung der Kosten für invertierte Segmente die weniger aufwendig zu bestimmende Gleichung (3.4).¹¹

Ergebnisse Die Rechenzeiten, die in der obigen Tabellen 6.2 und 6.4 angegeben wurden, beziehen sich auf die Durchführung der Rechnungen unter Verwendung der Kostenabschätzungen. In den Tabelle 6.5 sind die mittleren Rechenzeiten für dieselben Probleminstanzen mit und ohne Verwendung der Kostenabschätzungen gegenübergestellt. Die zweite Zeile vergleicht die mittlere Zahl der durchgeführten Verkettungsoperationen.

Es ist zu beachten, daß sich die Angaben jeweils auf eine komplette Rechnung beziehen. Deshalb wird der größte Teil des Aufwands, der für die 2-opt-Nachbarschaft angegeben ist, durch die I1-Heuristik verursacht. Die tatsächlich für die Verbesserung aufgewandte Zeit liegt bei der 2-opt-Nachbarschaft im Mittel bei 0.3 Sekunden (bzw. 0.25 Sekunden mit Kostenabschätzungen). Bei den kleineren Problemen liegt die Rechenzeit für die Verbesserungen durch 2-opt-Schritte bereits in der Größenordnung der Meßgenauigkeit.

Kostenabschätzung	i1	2-opt		3-opt		4-opt	
		ohne	mit	ohne	mit	ohne	mit
CPU	3,36	3,66	3,61	199,2	183,8	347,5	291,2
VO [10 ⁶]	0,17	0,22	0,20	69,6	65,2	122,5	103,8

Tabelle 6.5: Aufwandsreduktion durch Kostenabschätzung beim TSPTW für eine Auswahl der Probleminstanzen von Ascheuer et. al. [AFG97]

Die mit Hilfe der Kostenabschätzungen erzielten Einsparungen bewegen sich etwa zwischen 5 und 15 Prozent des gesamten Aufwands und sind damit relativ gering. Allerdings ist dieses Ergebnis wesentlich von den gegebenen Problemstellungen abhängig. Für die gewählten Instanzen ist der Effekt aus zwei Gründen besonders klein:

¹¹vgl. Seite 89f.

1. Viele Probleminstanzen besitzen enge Zeitfenster, so daß es aufgrund der stets vorgenommenen Zulässigkeitsprüfung ohnehin häufig zum Abbruch von Schleifen kommt.
2. Die Abstandsmatrizen der Probleminstanzen weisen viele Nullen auf. Dies führt dazu, daß die Abschätzungen, die oft die Länge des kürzesten Bogens benutzen, sehr schwach werden.

Dieselbe Untersuchung wurde auch für die 14 VRP-Instanzen durchgeführt. Als Start-Heuristik wurde diesmal das Sweep-Verfahren (ohne Reoptimierung der einzelnen Touren) verwendet, das i.a. eine sehr schlechte Ausgangslösung liefert, so daß sehr viele Verbesserungsschritte gefunden werden. Die Ergebnisse dieses Vergleichs sind in Tabelle 6.6 zusammengefaßt. Die Einsparungen betragen für diese Instanzen bei der 3-opt- und der 4-opt-Nachbarschaft mehr als 35 bzw. 50 Prozent.

Kostenabschätzung	Sweep	2-opt		3-opt		4-opt	
		ohne	mit	ohne	mit	ohne	mit
$\bar{z}$	1855.9	1095.8		1068.7		1023.5	
CPU	1.29	5.73	4.85	137.7	85.8	8285	3983
VO [10 ⁶]	0.23	1.20	0.90	41.7	23.6	2585	1245

Tabelle 6.6: Aufwandsreduktion durch Kostenabschätzung beim VRP für die Probleminstanzen von Christofides et. al. [CMT79]

► Da die Kostenabschätzungen keinen Einfluß auf die Qualität der gefundenen Lösungen haben und ab $k = 3$ bei den untersuchten Probleminstanzen oft zu einer deutlichen Reduktion des Aufwands führen, werden diese bei allen weiteren Untersuchungen benutzt.

Einsatz reduzierter k -opt-Nachbarschaften

Im Abschnitt 3.4 werden Reduktionen der k -opt-Nachbarschaften vorgeschlagen, bei denen nicht mehr sichergestellt ist, daß alle verbessernden Nachbarschaftsschritte gefunden werden. Man hofft jedoch, daß die Qualität der Lösungen auch mit diesen Nachbarschaften hoch bleibt.

Beschränkungen von Segmentlängen Für die Beschreibung von k -opt-Nachbarschaften mit beschränkten Segmentlängen verwenden wir entsprechend der Darstellung in Abschnitt 3.4 die Bezeichnungen ℓ_s für die maximale Länge eines kurzen Segments, und k_ℓ für die maximale Zahl langer Segmente. Durch die Angabe von ℓ_s und k_ℓ kann eine beliebige k -opt-Nachbarschaft eingeschränkt werden. In der reduzierten k -opt-Nachbarschaft sind dann nur k -opt-Schritte enthalten, bei denen höchstens k_ℓ innere Segmente länger als ℓ_s sind.

Die Tabelle 6.7 zeigt, welche Auswirkungen eine Beschränkung auf *ein* langes Segment ($k_\ell = 1$) haben kann. Dabei wird der Parameter ℓ_s zwischen 2 und 20 variiert. $\ell_s = \infty$ gibt zum Vergleich den Fall ohne Beschränkung von Segmentlängen an. Dabei bezeichnet $\bar{z}$ den über alle betrachteten Instanzen gebildeten Mittelwert der Zielfunktionswerte, CPU den Mittelwert der Rechenzeit in Sekunden¹² und VO [10⁶] die mittlere Zahl der Verkettungsoperationen in Millionen. In den Zeilen $\Delta\bar{z}$, ΔCPU und ΔVO sind die prozentualen Abweichungen, gegenüber der Rechnung mit unbeschränkten Segmentlängen angegeben.

ℓ_s	3-opt					
	∞	20	10	5	3	2
$\bar{z}$	1492.4	1492.8	1492.8	1492.4	1505.9	1500.5
$\Delta\bar{z}$	—	+0.0%	+0.0%	+0.0%	+0.8%	+0.4%
CPU	183.8	78.6	47.2	27.4	18.6	16.2
ΔCPU	—	-57%	-74%	-85%	-90%	-91%
VO [10 ⁶]	65.2	27.7	16.0	8.7	5.5	4.6
ΔVO	—	-58%	-75%	-87%	-92%	-93%

ℓ_s	4-opt					
	∞	20	10	5	3	2
$\bar{z}$	1475.8	1477.0	1477.0	1475.8	1482.3	1481.8
$\Delta\bar{z}$	—	+0.1%	+0.1%	+0.0%	+0.2%	+0.2%
CPU	291.2	146.4	90.5	51.4	34.3	25.1
ΔCPU	—	-50%	-69%	-82%	-88%	-91%
VO [10 ⁶]	103.8	52.4	31.9	17.5	11.3	7.9
ΔVO	—	-50%	-69%	-83%	-89%	-92%

Tabelle 6.7: k -opt-Nachbarschaften mit höchstens einem „langen“ Segment beim TSPTW

¹²auf einem PC mit AMD-Duron-Prozessor (700MHz)

Die Ergebnisse zeigen, daß eine deutliche Einschränkung der Nachbarschaften und eine entsprechende Einsparung an Rechenzeit möglich ist, ohne wesentlich schlechtere Ergebnisse zu erhalten. Bei einzelnen Probleminstanzen können die Verschlechterungen auch etwas größer sein (bis ca. 2 Prozent), als es die Mittelwerte angeben. Dies wird teilweise dadurch kompensiert, daß es bei anderen Probleminstanzen zu Verbesserungen der gefundenen Lösung kommt.

Derselbe Vergleich wurde auch für die VRP-Instanzen von Christofides et. al. mit k -opt-Nachbarschaften auf der Giant-Route durchgeführt (siehe Tabelle 6.8). Hier können die Segmentlängen nicht ganz so drastisch einge-

ℓ_s	3-opt					
	∞	20	10	5	3	2
$\bar{z}$	1068.7	1068.7	1071.5	1072.4	1075.1	1072.1
$\Delta\bar{z}$	—	+0.0%	+0.2%	+0.3%	+0.7%	+0.5%
CPU	85.8	37.2	20.7	13.2	12.7	11.9
Δ CPU	—	-57%	-76%	-85%	-85%	-86%
VO [10^6]	23.6	10.3	5.6	3.3	3.1	2.9
Δ VO	—	-56%	-76%	-86%	-87%	-88%

ℓ_s	4-opt					
	∞	20	10	5	3	2
$\bar{z}$	1023.5	1031.3	1025.2	1036.4	1038.7	1059.9
$\Delta\bar{z}$	—	+0.5%	+0.1%	+0.9%	+1.4%	+2.9%
CPU	3983	2431	507.8	175.2	84.7	46.3
Δ CPU	—	-39%	-87%	-96%	-98%	-99%
VO [10^6]	1245	738.0	153.7	51.1	24.6	12.4
Δ VO	—	-41%	-88%	-96%	-98%	-99%

Tabelle 6.8: k -opt-Nachbarschaften mit höchstens einem „langen“ Segment beim VRP

schränkt werden, ohne deutliche Qualitätsverluste hinzunehmen. Bereits bei $\ell_s = 10$ steigen die mittleren Kosten aber nur noch um etwa 0.2 Prozent. Die Reduktion des Rechenaufwands liegt dann aber trotzdem bei mehr als 85 Prozent.

► Insgesamt erweisen sich die Einschränkungen der Segmentlängen als ein sehr wirkungsvolles Mittel zur Erhöhung der Effizienz bei einer Lexiko-

graphischen Suche. Die Qualitätseinbußen können (bei den betrachteten Probleminstanzen) fast vernachlässigt werden, wenn die maximale Segmentlänge der kurzen Segmente ℓ_s nicht zu klein gewählt wird. Die starke Reduktion des Rechenaufwands ermöglicht es, bei vergleichbarem Aufwand k -opt-Nachbarschaften mit größerem k zu untersuchen, was zu deutlich besseren Resultaten führt.

Beschränkung auf „Nächste-Nachbarn“ Die Idee besteht darin, nur solche Nachbarschaftsschritte zu betrachten, bei denen die neuen Bögen einen Kundenknoten stets mit einem seiner ℓ nächsten Nachbarn verbindet.¹³

Die Reduktion der Nachbarschaft wird in diesem Fall dadurch erreicht, daß Restriktionen für die neu aufzunehmenden Bögen (und nicht mehr für die konstruierten Segmente) formuliert werden. Die Berücksichtigung solcher Restriktionen läßt sich aber nicht in gleichem Maße effizienzsteigernd in die Lexikographische Suche integrieren, wie die Beschränkung von Segmentlängen.

Um Effizienzvorteile erzielen zu können, muß es gelingen, die reduzierte k -opt-Nachbarschaft schneller abzusuchen, als die vollständige Nachbarschaft. Der Abbruch einer Schleife innerhalb der Lexikographischen Suche ist jedoch nur möglich, wenn *alle* in dieser und den nachfolgenden Schleifen erzeugten Nachbarschaftsschritte die neue (oder evtl. auch eine andere) Restriktion verletzen. Dies steht allgemein nur dann im voraus fest, wenn es zwei Segmente σ_i und σ_j gibt, die miteinander verbunden werden müssen, aber keine Möglichkeit mehr besteht, diese Segmente so zu konstruieren, daß der erste Knoten von σ_j zu den ℓ nächsten Nachbarn des letzten Knotens von σ_i gehört. Man kann vier Situationen unterscheiden:

1. Beide Segmente σ_i und σ_j sind bereits konstruiert und die zusätzliche Restriktion kann direkt überprüft werden.
2. Segment σ_i liegt bereits fest und Segment σ_j muß noch konstruiert werden. Dann muß geprüft werden, ob für den Aufbau von σ_j noch Knoten verwendet werden können, die zu den ℓ nächsten Nachbarn des letzten Knotens von σ_i gehören.
3. Segment σ_j liegt bereits fest und Segment σ_i muß noch konstruiert werden. Dann muß geprüft werden, ob für den Aufbau von σ_i noch

¹³vgl. Seite 117

Knoten verwendet werden können, zu deren ℓ nächsten Nachbarn der erste Knoten von σ_j gehört.

4. Beide Segmente σ_i und σ_j müssen noch konstruiert werden. Dann muß geprüft werden, ob für den Aufbau von σ_i noch Knoten verwendet werden können, so daß ein möglicher erster Knoten von σ_j zu den ℓ nächsten Nachbarn eines möglichen letzten Knotens von σ_i gehört.

Soll eine entsprechende Überprüfung in die Lexikographische Suche integriert werden, so muß sie auf jeden Fall effizient (d.h. mit konstantem Aufwand) durchführbar sein, damit die Komplexität $\mathcal{O}(n^k)$ der Lexikographischen Suche in der k -opt-Nachbarschaft erhalten bleibt.

Die ersten drei Kriterien können effizient überprüft werden. Dazu werden vorab einmalig zu jedem Knoten die ℓ nächsten Nachbarn bestimmt, so daß während der Suche in konstanter Zeit festgestellt werden kann, ob ein Knoten zu den ℓ nächsten Nachbarn eines anderen Knotens zählt oder nicht. Für die Überprüfung des zweiten Kriteriums wird außerdem für jeden Knoten gespeichert, ab welcher Position der aktuellen Lösung der letzte Knoten steht, der zu den ℓ nächsten Nachbarn gehört. Analog kann für die Überprüfung des dritten Kriteriums zu jedem Knoten die Position gespeichert werden, ab der kein zulässiger Vorgängerknoten mehr gefunden werden kann.

Die beiden ersten Kriterien wurden in der oben angegebenen Form implementiert und in den Algorithmus 3.1 für die Lexikographische Suche integriert. Die Tabelle 6.9 zeigt die Auswirkungen einer Reduktion der 3-opt- bzw. 4-opt-Nachbarschaft durch die Wahl von $\ell = 30$ bzw. $\ell = 50$ für das TSPTW.

ℓ	3-opt mit			4-opt mit			LB
	∞	50	30	∞	50	30	
$\bar{z}$	1492.4	1650.2	1719.9	1475.8	1614.1	1702.9	1421.9
$\Delta\bar{z}$	—	+17%	+12%	—	+15%	+9.4%	—
CPU	183.8	98.1	61.6	291.2	204.3	107.0	—
Δ CPU	—	-47%	-66%	—	-30%	-63%	—
VO [10 ⁶]	65.2	34.7	21.2	103.8	72.3	37.5	—
Δ VO	—	-47%	-67%	—	-30%	-64%	—

Tabelle 6.9: Auswirkungen der Reduktion auf „nächste Nachbarn“ beim TSPTW (Probleminstanzen von Ascheuer et. al.)

Durch die fixe Beschränkung auf die ℓ nächsten Nachbarn wird einerseits nur eine moderate Beschleunigung der Rechnungen erreicht. Gleichzeitig ist

diese Reduktion selbst für $\ell = 50$ noch zu scharf, da durch diese Einschränkung nur Lösungen mit ca. 10 bis 20 Prozent höheren Kosten gefunden werden. Der geringe Gewinn an Performance läßt sich teilweise auch durch diese zu harte Einschränkung erklären. Bei einem detaillierten Vergleich für einzelne Probleminstanzen ist festzustellen, daß für das Erreichen gleichwertiger Lösungen in der reduzierten Nachbarschaft teilweise deutlich mehr Schritte durchgeführt werden müssen als zuvor. Dadurch können die Rechenzeiten bei einzelnen Instanzen ggf. sogar etwas ansteigen.

ℓ	3-opt				4-opt			
	∞	30	20	10	∞	30	20	10
$\bar{z}$	1068.7	1070.3	1070.2	1074.1	1023.5	1024.6	1039.6	1045.7
$\Delta\bar{z}$	—	+0.1%	+0.1%	+0.5%	—	+0,1%	+1,6%	+2,2%
CPU	85.8	51.8	50.3	28.0	3983	1037	582.4	572.1
Δ CPU	—	-40%	-41%	-67%	—	-74%	-85%	-86%
VO [10 ⁶]	23.6	14.5	13.8	7.6	1245	307.9	174.7	172.7
Δ VO	—	-39%	-42%	-68%	—	-75%	-86%	-86%

Tabelle 6.10: Auswirkungen der Reduktion auf „nächste Nachbarn“ beim VRP (Probleminstanzen von Christofides et. al.)

Dieselbe Analyse wurde ebenfalls für die 14 VRP-Instanzen von Christofides et. al. durchgeführt (Tabelle 6.10). Hier führt die Reduktion nicht zu so starken Qualitätsverlusten. Bei einer Einschränkung der Nachbarschaft auf die 20 nächsten Nachbarn ($\ell = 20$) werden die Lösungen im Mittel bei $k = 3$ nur um 0,1 Prozent schlechter. Bei $k = 4$ läßt sich eine höhere Sensibilität feststellen, so daß die mittleren Kosten bei $\ell = 20$ um ca. 1,6 Prozent ansteigen. Durch eine Vergrößerung der NN-Listen auf 30 Nachbarn kann die Erhöhung der Kosten jedoch wieder auf ca. 0,1 Prozent gesenkt werden. Dabei konnten immernoch mehr als 70 Prozent der Rechenzeit gespart werden.

Zu beachten ist, daß die relativ guten Ergebnisse beim VRP nur erhalten wurden, weil zu Beginn eine Reoptimierung der einzelnen Touren ohne die NN-Listen durchgeführt wurde. Die NN-Listen wurden erst für die Suche auf der Giant-Route eingesetzt. Bei dem Versuch die Nächste-Nachbarn-Listen von Beginn an zu verwenden, ohne die einzelnen Touren der Lösung des Sweep-Verfahrens vorab zu reoptimieren, verschlechterten sich die Lösungen i.d.R. um 10 bis 20 Prozent. Offenbar ist es bei einer relativ schlechten Startlösung notwendig, diese zunächst ohne NN-Listen zu verbessern. Später

können die NN-Listen benutzt werden, ohne daß die Lösungsqualität dadurch wesentlich beeinflußt wird.

► Eine Reduktion der k -opt-Nachbarschaften durch Beschränkung der zulässigen neuen Bögen auf die ℓ nächsten Nachbarn läßt sich im Rahmen einer Lexikographischen Suche für die Einsparung von Rechenaufwand nutzen. Die Reduktion der Nachbarschaften mit Hilfe von NN-Listen muß sehr vorsichtig erfolgen, um die Lösungsqualität nicht deutlich zu beeinträchtigen.

6.3.4 Einsatz von k -opt*-Nachbarschaften

Im Abschnitt 3.3.4 wurde beschrieben, wie durch relativ einfache Modifikationen des Vorgehens bei der Suche in k -opt-Nachbarschaften auch beliebige k -opt*-Nachbarschaften untersucht werden können. In der Literatur wird fast ausschließlich die 2-opt*-Nachbarschaft verwendet. Die entsprechenden Nachbarschaftsschritte werden teilweise auch als „Cross-Exchange“ bezeichnet und sind besonders beim VRPTW attraktiv, weil die 2-opt*-Schritte keine Segmente invertieren.

Da wir über eine generelle Beschreibung der k -opt*-Nachbarschaften verfügen, sind wir in der Lage zu untersuchen, welche Vorteile der Einsatz von k -opt*-Nachbarschaften für $k = 2$ und für größere Werte von k besitzt. Dazu wurden die 2-opt*- und 3-opt*-Nachbarschaft für die VRPTW-Instanzen von Solomon eingesetzt. In Tabelle 6.11 sind die Ergebnisse einander gegenübergestellt. Als Ausgangslösung dienen jeweils die Lösungen der I1-Heuristik einschließlich der Verbesserungen, die durch eine 3-opt-Suche für jede einzelne Tour gefunden werden.

Die Erweiterung der 2-opt-Nachbarschaft um die 2-opt*-Schritte führt zu einer deutlichen Verbesserung der Ergebnisse. Dabei ist der zusätzliche Rechenaufwand relativ gering. Die Ausdehnung der 3-opt-Nachbarschaft auf die vollständige 3-opt*-Nachbarschaft¹⁴ erhöht nicht nur die Lösungsqualität, sondern führt gleichzeitig zu einer Beschleunigung. Dies erklärt sich dadurch, daß eine Reihe von Verbesserungen bereits bei der Untersuchung der 2-opt*-Typen gefunden wird. Dadurch muß die größere 3-opt-Nachbarschaft seltener untersucht werden.

¹⁴Die vollständige 3-opt*-Nachbarschaft umfaßt die echten 2-opt-, 3-opt-, 2-opt*- und 3-opt*-Schritte, d.h. alle folgenden k -opt-Typen: $[aBa]$, $[aa|b]$, $[aa|B]$, $[acba]$, $[aCba]$, $[acBa]$, $[aBCa]$, $[aa|bC]$, $[aCa|b]$, $[aBa|c]$, $[aCa|B]$, $[aBa|C]$, $[aa|b|c]$, $[aa|B|c]$, $[aa|b|C]$ und $[aa|B|C]$. Die Invertierung des Start- und Endsegments „a“ wird nicht betrachtet.

	I1+S3	2-opt	2-opt, 2-opt*	2-opt, 3-opt	2-opt, 2-opt* 3-opt, 3-opt*	2-opt, 2-opt* 3-opt
R1	13.5	13.5	13.5	13.3	13.3	13.3
	1367.4	1378.2	1290.4	1274.3	1253.6	1253.6
	0.1	0.1	0.4	7.2	6.2	5.5
	0.3	0.6	1.6	19.3	16.1	14.4
RC1	13.5	13.4	3.4	13.4	13.4	13.4
	1566.8	1568.6	1459.3	1428.4	1425.2	1433.8
	0.1	0.1	0.5	8.7	5.4	4.1
	0.4	0.3	1.7	23.6	14.1	10.9
R2	3.3	3.3	3.3	3.3	3.3	3.3
	1205.8	1283.9	1081.9	1018.4	1011.1	1012.1
	0.4	0.4	0.8	11.9	10.8	9.9
	1.7	1.8	2.9	32.2	28.1	26.1
RC2	3.9	3.9	3.9	3.9	3.9	3.9
	1462.9	1592.2	1326.5	1194.1	1184.8	1191.1
	0.2	0.2	0.6	11.2	9.1	8.3
	1.1	1.1	2.1	30.2	23.6	21.7
Alle	8.6	8.6	8.6	8.5	8.5	8.5
	1382.3	1434.5	1273.6	1217.3	1206.3	1209.6
	0.2	0.2	0.6	9.6	7.9	7.0
	0.9	1.0	2.1	26.1	20.6	18.5

Zu jeder Rechnung sind die Anzahl der Touren, die Gesamtstrecke, die Zahl der Verkettungsoperatoren in Millionen und die Rechenzeit in Sekunden in dieser Reihenfolge angegeben. Alle Werte wurden über die Instanzen der jeweiligen Problemklasse gemittelt.

Tabelle 6.11: k -opt*-Nachbarschaften für das VRPTW

Da nicht ausgeschlossen werden kann, daß die Verbesserung beim Übergang zur 3-opt*-Nachbarschaft ausschließlich den 2-opt*-Schritten zu verdanken ist, wurde die 3-opt-Nachbarschaft in der letzten Spalte von Tabelle 6.11 nur durch die 2-opt*-Nachbarschaft ergänzt. Dabei zeigt sich, daß teilweise gleichwertige Lösungen auch nur mit der Erweiterung mit der 2-opt*-Nachbarschaft erhalten werden. Gleichzeitig reduziert sich der Rechenaufwand um mehr als 10 Prozent.

Über die hier dargestellten Ergebnisse hinaus wurden weitere zahlreiche weitere Untersuchungen u.a. auch am VRP vorgenommen. Dabei führte die

Verwendung der 2-opt*-Nachbarschaft ebenfalls zu deutlichen Verbesserungen. k -opt*-Nachbarschaften mit $k > 2$ erhöhen dagegen fast immer nur den Rechenaufwand. Nur in seltenen Fällen ist es möglich, durch diese k -opt*-Schritte Verbesserungen zu erzielen.

► Die Untersuchung der 2-opt*-Schritte führt sowohl beim VRP als auch beim VRPTW i.d.R. zu deutlichen Verbesserungen der gefundenen Lösungen. Der Rechenaufwand kann sich teilweise sogar gleichzeitig reduzieren. Beim VRPTW können auch mit Hilfe von 3-opt*-Schritten zusätzliche Verbesserungen erreicht werden. Dagegen erschließen k -opt*-Nachbarschaften mit $k > 2$ bei den VRP-Instanzen kein oder nur sehr geringes Verbesserungspotential.

6.3.5 k -opt-Nachbarschaften für das VRPTW

Abschließend sollen die k -opt-Nachbarschaften nun auch auf die VRPTW-Instanzen von Solomon angewandt werden. Dabei lassen wir die Instanzen der Kategorien C1 und C2 weg, weil diese, wie bereits argumentiert wurde, kaum interessant sind.

Um den Umfang der Untersuchungen zu beschränken, werden nur die Ergebnisse einiger ausgewählte Vergleichsrechnungen angegeben. Die Rechnungen wurden diesmal auf mehreren PC durchgeführt, die mit einem AMD Athlon Prozessor mit 1400 MHz ausgestattet sind. Anhand der je Zeiteinheit durchgeführten Verkettungsoperationen läßt sich feststellen, daß die Rechner etwa 1,9-mal schneller sind, als der AMD Duron mit 700 MHz. Allerdings wurden auch unter den weitgehend baugleichen Rechnern Rechenzeitunterschiede bis etwa 10 Prozent festgestellt.

Tabelle 6.12 zeigt die Ergebnisse bei einer Durchführung der Lokalen Suche mit der 4-opt-Nachbarschaft auf der Giant-Route. Als Startlösung wurden die Lösungen einer eigenen Implementierung der I1-Heuristik von Solomon verwendet (Spalte I1). Die weiteren Spalten zeigen, wie sich die Einschränkung der Längen der Toursegmente (bis auf ein inneres Segment) auswirkt. Dabei wurde stets mit der Reoptimierung der Einzeltouren mit der 4-opt-Nachbarschaft begonnen.

Solange alle Segmente bis zu einer Länge von drei Knoten zugelassen werden, wird die Qualität der Lösungen auch bei den VRPTW-Instanzen durch die Beschränkung der Längen der Toursegmente kaum beeinflusst.

		I1		4-opt		
		ℓ_s	∞	3	2	1
R1	Strecke	1388,3	1228,0	1228,0	1232,2	1251,2
	Touren	13,5	13,2	13,2	13,2	13,3
RC1	Strecke	1583,7	1419,8	1419,8	1419,8	1419,8
	Touren	13,5	13,3	13,3	13,3	13,3
R2	Strecke	1301,0	1012,7	1012,7	1011,6	1052,6
	Touren	3,3	3,2	3,2	3,2	3,2
RC2	Strecke	1592,2	1154,9	1154,9	1160,9	1223,1
	Touren	3,9	3,9	3,9	3,9	3,9
Alle	Strecke	1445,6	1191,6	1191,6	1195,0	1224,9
	Touren	8,6	8,5	8,5	8,5	8,5
	CPU	0,2	58,2	58,2	33,6	6,2
	VO [10 ⁶]	0,1	41,4	41,4	22,9	4,0

Tabelle 6.12: k -opt-Nachbarschaften mit höchstens einem „langen“ Segment beim VRPTW

Bei den Berechnungen wurden allen Touren hohe Fixkosten (1000 pro Tour) zugerechnet, um die Tourenzahl ggf. reduzieren zu können. Die Ergebnisse zeigen, daß es dennoch kaum möglich ist, mit Hilfe der k -opt-Nachbarschaften Touren aufzulösen. Bei den Untersuchungen zum VRP wurde bereits festgestellt, daß k -opt-Nachbarschaften nur bedingt dazu in der Lage sind, eine vorgegebene Clusterung zu verändern. Daher ist es leicht nachzuvollziehen, daß die Anzahl der Touren kaum verändert wurde.

Um die Qualität der Lösungen einzuschätzen, kann ein Vergleich mit den Ergebnissen anderer Autoren vorgenommen werden. Dies ist nur für die Instanzen sinnvoll, bei denen die gleiche Tourenzahl bestimmt wurde, denn die Tourlänge ist oft erheblich von der Tourenzahl abhängig. Die Tabellen 6.14 und 6.13 geben einen Vergleich der mit der 4-opt-Nachbarschaft (jetzt einschließlich der 2-opt*-Nachbarschaft) berechneten Lösungen mit den Ergebnissen der folgenden fünf Meta-Heuristiken für VRPTW:¹⁵

GENEROUS:	Potvin und Bengio, [PB96]
CR:	Chiang und Russel, [CR96]
GenSAT:	Thangiah et. al., [TOS95]
RS:	Rochat und Semet, [RS94]
AKred:	Cordone und Wolfer Calvo, [CW98]

¹⁵Alle Ergebnisse wurden aus [CW98] übernommen.

Die Tabellen enthalten nur dann Einträge, wenn die Verbesserung mit der 4-opt-Nachbarschaft und das zum Vergleich herangezogene Verfahren Lösungen mit gleicher Tourenzahl bestimmt haben. Die Zahlen geben die Weg-Differenz zwischen beiden Lösungen an. Negative Zahlen bedeuten, daß die 4-opt-Nachbarschaft eine Lösung mit einem kürzeren Weg bestimmt hat. In den letzten beiden Zeilen sind die Differenzen summiert und zur gesamten berechneten Weglänge (über alle Instanzen mit gleicher Tourenzahl) ins Verhältnis gesetzt.

Problem	Generous	CR	RS	Akred	Gensat
RC102		-53.2			
RC104				-75.4	
RC106		-54.2			26.8
RC107		-104.1		6.0	
RC108	-32.9	-37.9	4.4	-3.1	
RC201	-262.2	-197.2	67.1	81.4	242.8
RC202	-332.6	-293.6	-173.2	104.4	-20.6
RC204	-83.4	-93.4	84.5	96.8	21.6
RC208	-152.7	-156.7	-23.5	10.4	-22.7
Alle	-863.8	-990.2	-40.7	220.4	247.9
%	-7,5%	-7,9%	-0,7%	+2,6%	+2,7%

Tabelle 6.13: Vergleich der 4-opt-Suche mit Meta-Heuristiken für die RC1- und RC2-Instanzen von Solomon

Aus beiden Tabellen ergibt sich ein weitgehend einheitliches Bild. GENEROUS und CR lassen deutliche Nachteile gegenüber der 4-opt-Suche erkennen. Sie weisen durchgängig bei allen vergleichbaren Instanzen schlechtere Lösungen auf. Die Lösungsqualität von RS ist etwa vergleichbar mit der 4-opt-Suche. GenSAT bzw. AKred erweisen sich in diesem Vergleich als überlegen, finden aber nicht grundsätzlich die besseren Lösungen.

Bei der Bewertung muß berücksichtigt werden, daß hier Meta-Heuristiken mit einer reinen Lokalen Suche verglichen werden. Wenn die Suchstrategien der Meta-Heuristiken gut funktionieren, sollte man erwarten können, daß die Lokale Suche dominiert wird. Weil alle zum Vergleich herangezogenen Meta-Heuristiken in der aktuellen Literatur häufig zitierte Verfahren sind, die als leistungsstark gelten, ist das relativ gute Abschneiden der reinen Lokalen Suche in diesem Vergleich bemerkenswert. Zumindest GENEROUS und CR könnten jedoch durch eine Lokale Suche mit k -opt-Nachbarschaften ergänzt und dadurch mit hoher Wahrscheinlichkeit deutlich verbessert werden.

Problem	Generous	CR	RS	Akred	Gensat
R103	-199.5		-189.4	-14.5	
R104	-39.4	-53.4	12.4	-27.0	-22.4
R105		-63.2			
R107	-74.0		31.2		-23.0
R108	5.4	-4.6	33.1	18.5	100.4
R111	-36.0		77.1	19.4	
R112	-30.8	-13.8	23.6	-39.6	-2.8
R201	-265.8	-205.8	-197.2	14.7	-65.8
R202			83.3		8.8
R203	-108.0	-80.0		99.5	-33.0
R204	-125.0		-45.6		-24.0
R205	-42.9	-95.9	-89.5	17.1	-11.9
R206	-42.4	-50.4	67.6	86.9	190.7
R207	-58.4	-103.4	80.7	31.2	-8.4
R208			43.9		
R209	-63.9	-87.9		46.5	189.1
R210	-53.9	-152.9	4.7	54.3	40.1
R211	-42.8	-106.8	75.8	51.4	54.2
Alle	-1177.3	-1018.0	11.8	358.3	391.9
%	-15,0%	-9,8%	-0,7%	+2,7%	+4,1%

Tabelle 6.14: Vergleich der 4-opt-Suche mit Meta-Heuristiken für die R1- und R2-Instanzen von Solomon

Abschließend sei darauf hingewiesen, daß der vorgenommene Vergleich nur als fair anzusehen ist, wenn er in engen Grenzen interpretiert wird. Bei den Rechnungen der Meta-Heuristiken war die Reduzierung der Anzahl der Touren das oberste Ziel. In dieser Hinsicht haben alle Meta-Heuristiken deutlich bessere Ergebnisse erzielt. Dies ist auch nicht anders zu erwarten, weil eine Auflösung von Touren durch die Suche in einer k -opt-Nachbarschaft kaum zu finden ist. Alle erfolgreichen Verfahren benutzen dafür spezielle Strategien.

6.4 Cyclic-Transfer-Nachbarschaften für das VRP und das VRPTW

Die vorangegangenen Untersuchungen haben gezeigt, daß es auch mit der Ausdehnung der k -opt-Nachbarschaften auf die Giant-Route nicht immer möglich ist, eine ungünstige Aufteilung der Kunden auf die einzelnen Touren

zu korrigieren. Da die im Kapitel 4 beschriebenen Cyclic Transfers grundsätzlich eine Umverteilung der Kunden zwischen verschiedenen Touren vornehmen, sollten sie für diesen Zweck besser geeignet sein. Die Vermutung, daß durch die Kombination von k -opt-Nachbarschaften und CT-Nachbarschaften eine besonders leistungsstarke Verbesserungsheuristik gebildet werden kann, liegt deshalb nahe. Die Kombination der verschiedenen Nachbarschaften erfolgt im Rahmen einer VND, wie sie bereits zur Kombination unterschiedlicher k -opt-Nachbarschaften verwendet wurde.¹⁶

In diesem Abschnitt soll anhand des VRP und des VRPTW untersucht werden, welche Vorteile durch die Suche in CT-Nachbarschaften erzielt werden können. Dabei wird die Suche in den CT-Nachbarschaften jeweils mit einer k -opt-Suche kombiniert, die entweder nur auf den einzelnen Touren oder auf der Giant-Route ausgeführt wird. Anhand der VRPTW-Instanzen von Solomon gehen wir anschließend auf die Auflösung von Touren mit Hilfe der CT-Nachbarschaften ein.

6.4.1 Durchführung der Suche in den Cyclic-Transfer-Nachbarschaften

Die im Kapitel 4 angegebene generische Beschreibung der CT-Nachbarschaften beinhaltet eine Reihe von Wahlmöglichkeiten bei der konkreten Ausgestaltung dieser Nachbarschaften. Um den Umfang dieses Abschnitts zu beschränken, soll nur *eine* konkrete Ausprägung der CT-Nachbarschaften untersucht werden. Die dafür getroffenen Festlegungen werden nachfolgend angegeben. Sie basieren auf zahlreichen im Vorfeld durchgeführten Rechenexperimenten, die die getroffenen Entscheidungen als geeigneten Kompromiß zwischen Rechenaufwand und Lösungsqualität erscheinen lassen.

Konstruktion des Transfergraphen

Für die Konstruktion des Transfergraphen G^T muß ein konkreter Austauschoperator $\boxplus$ angegeben werden, mit dessen Hilfe die Bögen des Transfergraphen bestimmt werden sollen. In Abschnitt 4.2.4 wurde ein allgemeines Schema für einen heuristischen Austauschoperator vorgeschlagen,¹⁷ das uns

¹⁶Das Prinzip der Variable-Neighborhood-Descent-Heuristik wird im Abschnitt 2.3 (Seite 55) vorgestellt. Auf Seite 169f. ist die Durchführung einer VND für die kombinierte Untersuchung mehrerer k -opt-Nachbarschaften beschrieben.

¹⁷siehe S. 136

als Vorlage für die Beschreibung des verwendeten Austauschoperators dient.

Es seien t_i und t_j zwei verschiedene Touren der aktuellen Lösung und S_i und S_j Kundenmengen dieser Touren. Wir betrachten die Bestimmung des Bogens (S_i, S_j) des Transfergraphen G^T . Dieser Bogen repräsentiert die modifizierte Tour t_j , die durch den Austausch $S_i \boxplus S_j$ erzeugt wird. Für diesen Austausch müssen die Knoten S_j aus der Tour t_j entfernt und die Knoten S_i in die verbleibende Tour eingefügt werden. Dies geschieht wie folgt:

$t_j \rightarrow t_j^{(1)}$	Die Kundenmenge S_j wird aus der Tour t_j entfernt. Dabei wird die Reihenfolge der verbleibenden Kunden innerhalb der Tour nicht geändert.
$t_j^{(1)} \rightarrow t_j^{(2)}$	Die neue Tour $t_j^{(1)}$ wird mit einer Heuristik $\mathcal{H}_1$ verbessert. ¹⁷ Dafür setzen wir eine k -opt-Suche in einer reduzierten 3-opt-Nachbarschaft ein. Die Reduktion der 3-opt-Nachbarschaft wird durch eine Beschränkung der Segmentlängen erreicht, ¹⁸ wobei nur ein langes inneres Segment und ein kurzes Segment mit höchstens vier Knoten erlaubt werden.
$t_j^{(2)} \rightarrow t_j^{(3)}$	Die Knoten der Transfermenge S_i werden nacheinander in die Tour $t_j^{(2)}$ eingefügt. Dabei wird jeweils die erste kostenminimale Einfügeposition innerhalb der Tour gewählt. Kann ein Knoten nicht eingefügt werden, wird die Durchführung des Austauschoperators abgebrochen und der entsprechende Bogen im Transfergraphen nicht erzeugt. Werden alle Knoten von S_i erfolgreich in $t_j^{(2)}$ eingefügt, erhält man die Tour $t_j^{(3)}$.
$t_j^{(3)} \rightarrow t_j^{(4)}$	Auf die in dem allgemeinen Schema vorgesehene Reoptimierung der Tour $t_j^{(3)}$ wird zugunsten der Performance beim Aufbau des Transfergraphen verzichtet.

Verbesserung der Performance

Die Bestimmung der Bögen des Transfergraphen kann wesentlich beschleunigt werden. Dazu nutzt man aus, daß sich die einzelnen Bögen teilweise nur in ein oder mehreren Kundenknoten unterscheiden, die zusätzlich in die zu bildende Tour eingefügt werden müssen. Falls der Bogen $S_i \curvearrowright S_j$ bereits

¹⁸Beschränkungen von Segmentlängen bei der k -opt-Suche werden im Abschnitt 3.4 erklärt.

berechnet wurde und der Bogen $S'_i \curvearrowright S_j$ mit $S'_i \supset S_i$ zu bestimmen ist, so muß die neue Tour nicht von Beginn an konstruiert werden. Stattdessen kann man direkt mit der Tour $S_i \curvearrowright S_j$ beginnen, die nur um die fehlenden Kunden $S'_i \setminus S_i$ erweitert wird.

Um die Durchführbarkeit des Austauschs $S_i \curvearrowright S_j$ schnell prüfen zu können, werden alle gebildeten Touren in einer Hash-Table verwaltet. Touren werden darin anhand eines Schlüssels gesucht, der die enthaltenen Kundenknoten kodiert.

Wenn Transfermengen betrachtet werden, die aus unterschiedlich langen Toursegmenten bestehen, kann durch die Einhaltung einer geeigneten Reihenfolge bei der Bestimmung der Bögen des Transfergraphen gesichert werden, daß die Einfügung der kürzeren Toursegmente stets zuerst getestet wird. Soll später ein Bogen $S'_i \curvearrowright S_j$ mit $S'_i \supset S_i$ bestimmt werden, so muß für den Bogen $S_i \curvearrowright S_j$ eine entsprechende Eintragung in der Hash-Table gefunden werden. Existiert diese nicht, kann auch der aktuell zu bestimmende Bogen $S'_i \curvearrowright S_j$ nicht existieren und die Konstruktion der zugehörigen Tour entfällt.

Suche im Transfergraphen

Für die Suche nach einem kürzesten Kreis, der die Bedingung der CT-Konsistenz erfüllt, wurden in Abschnitt 4.3 sowohl ein exaktes Verfahren als auch eine Heuristik angegeben. Anhand zahlreicher Experimente hat sich gezeigt, daß es vorteilhaft ist, in größeren (reduzierten) CT-Nachbarschaften zu suchen. Da der Aufwand für die exakte Suche sehr schnell wächst, kann die exakte Suche dann i.d.R. nicht mehr durchgeführt werden. Deshalb wird die Label Correcting Heuristik von Ahuja et. al. [AOS01] eingesetzt.

Der Nachteil, mit der Label Correcting Heuristik nicht mehr mit Sicherheit bestimmen zu können, ob eine Verbesserung in der betrachteten CT-Nachbarschaft zu finden ist, wird durch die Möglichkeit, in größeren CT-Nachbarschaften suchen zu können, überkompensiert. Wir setzen deshalb grundsätzlich die Label Correcting Heuristik von Ahuja et. al. für die Suche im Transfergraphen ein.

Kombination reduzierter CT-Nachbarschaften

Die vollständige Cyclic-Transfer-Nachbarschaft gemäß Definition 4.11 ist bereits für Probleme mit relativ wenigen Kunden sehr groß. Deshalb würde der Aufbau des zugehörigen Transfergraphen für die zu untersuchenden Instan-

zen des VRP bzw. VRPTW praktisch kaum durchführbar sein. Stattdessen werden die in Unterabschnitt 4.2.3 beschriebenen reduzierten Cyclic-Transfer-Nachbarschaften verwendet. Diese werden dadurch gebildet, daß nur bestimmte Transfermengen als Knoten des Transfergraphen berücksichtigt werden.

Wir betrachten ausschließlich reduzierte CT-Nachbarschaften, bei denen die Knotenmenge des Transfergraphen aus folgenden Transfermengen zusammengesetzt werden kann:

Bezeichnung	berücksichtigte Transfermengen S_i, S_j
E	alle leeren Kundenteilmengen
S	einzelne Kunden
D	Toursegmente mit genau zwei Kunden
H	Touranfangsstücke ¹⁹
T	Tourendstücke ¹⁹

Durch die Vereinigung mehrerer dieser Knotenmengen werden größere Transfergraphen impliziert. Wir kombinieren stets mindestens drei der genannten Mengen und verwenden für die Untersuchung in diesem Abschnitt CT-Nachbarschaften, deren Transfergraph aus folgenden Knoten aufgebaut werden:

Knotenmenge des Transfergraphen	Bezeichnung der Nachbarschaft
$E \cup H \cup T$	EHT
$E \cup S \cup H$	ESH
$E \cup S \cup T$	EST
$E \cup S \cup D$	ESD
$E \cup S \cup H \cup T$	ESHT
$E \cup S \cup D \cup H \cup T$	ESDHT

Die obigen reduzierten Cyclic-Transfer-Nachbarschaften können im Sinne einer VND miteinander kombiniert werden. Dabei wird wie bei der k -opt-Suche aus Gründen der Performance stets mit den kleineren Nachbarschaften begonnen, die wesentlich schneller untersucht werden können.

¹⁹Als Touranfangsstück bzw. Tourendstück einer Tour t bezeichnen wir alle Toursegmente, die den ersten Kunden von t bzw. den letzten Kunden von Tour t und keine Depotknoten enthalten.

In der obigen Tabelle sind die Nachbarschaften so geordnet, daß die Größe der Transfergraphen in dieser Reihenfolge wächst. Die Transfergraphen der ersten vier Nachbarschaften besitzen zwar etwa gleichviele Knoten, die Zahl der zulässigen Bögen, ist aber meistens geringer, wenn Touranfangs- oder Tourendstücke eingefügt werden müssen, weil das Einfügen der Knoten eines langen Toursegments häufiger nicht zulässig möglich ist. In den Nachbarschaften ESH und EST sind vier bzw. fünf Knotenmengen zusammengefaßt. Deshalb sind die zugehörigen Transfergraphen deutlich größer als die übrigen. Aufgrund dieser Argumentation gehen wir im folgenden davon aus, daß auch die Größen der CT-Nachbarschaften in der Reihenfolge EHT, ESH, EST, ESD, ESH, ESDHT monoton wachsend sind, wobei nur Nachbarschaften ESH und EST als gleich groß angesehen werden.

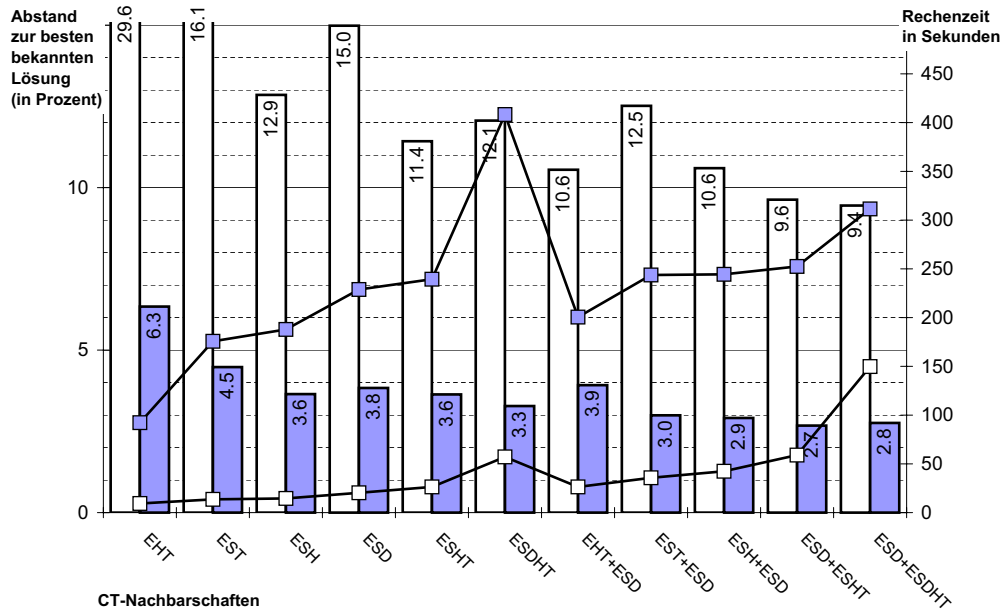
Im Rahmen der Variable-Neighborhood-Descent-Heuristik wird die Suche in den CT-Nachbarschaften stets mit einer k -opt-Suche verbunden. Diese kann entweder nur auf den einzelnen Touren oder darüber hinaus auch auf der Giant-Route durchgeführt werden. Die k -opt-Suche wird immer vor der Suche in der CT-Nachbarschaft aufgerufen. Anderenfalls kann es passieren, daß die Verbesserungsheuristik $\mathcal{H}_1$, die als Teil des Austauschoperators ausgeführt wird, sehr viele Verbesserungen findet. Da der Austauschoperator beim Aufbau des Transfergraph sehr häufig aufgerufen wird, würde dies den Aufwand für die Konstruktion des Transfergraphen deutlich erhöhen.²⁰

6.4.2 Ergebnisse für das VRP

In diesem Abschnitt wird ein Vergleich unterschiedlicher CT-Nachbarschaften bei der Anwendung auf die Probleminstanzen von Christofides et. al. vorgenommen. Das Diagramm 6.1 stellt die Ergebnisse von elf verschiedenen Experimenten dar, bei denen die bereits beschriebenen CT-Nachbarschaften bzw. Kombinationen von je zwei dieser Nachbarschaften eingesetzt wurden.

Die Balken geben jeweils die mittlere Abweichung der gefundenen Lösungen zu den besten bekannten Lösungen an. Die zugehörigen Werte sind an der linken Achse abzulesen. Auch die eingezeichneten Hilfslinien beziehen sich auf diese Achse. Die aufgewandte Rechenzeit wurde ebenfalls über alle Probleminstanzen gemittelt. Die Zeiten sind in der Abbildung durch kleine Quadrate markiert, deren Werte an der rechten Größenachse abzulesen sind.

²⁰Wie bereits im Abschnitt 4.2.4 erläutert wurde, muß die Heuristik $\mathcal{H}_1$ für jeden Knoten des Transfergraphen einmal aufgerufen werden.



helle Füllung: k -opt-Suche nur für einzelne Touren

dunkle Füllung: k -opt-Suche auf der Giant-Route

Balken/linke Größenachse: mittlerer Abstand zur besten bekannten Lösung

Linien/rechte Größenachse: mittlere Rechenzeit in Sekunden (AMD Athlon 1400MHz)

Abbildung 6.1: Vergleich verschiedener Cyclic-Transfer-Nachbarschaften beim VRP

Für jede betrachtete CT-Nachbarschaft wurden zwei Rechnungen durchgeführt. Bei der ersten Rechnung wurde im Rahmen der VND zuerst eine Reoptimierung der *einzelnen Touren* mit Hilfe einer 4-opt-Suche vorgenommen. Danach wird zur jeweiligen CT-Nachbarschaft gewechselt. Werden zwei CT-Nachbarschaften untersucht, so wird stets mit der kleineren Nachbarschaft begonnen. Wenn die aktuelle Lösung bezüglich aller Nachbarschaften lokal optimal ist, bricht die Suche ab. Die Ergebnisse dieser Rechnungen sind in Abb. 6.1 mit weißen Balken und die Rechenzeiten mit weißen Quadraten dargestellt. Die dunkel gefüllten Balken und Quadrate geben die Ergebnisse der zweiten Rechnung an. Dabei wird zusätzlich eine k -opt-Suche mit einer

reduzierten 4-opt-Nachbarschaft auf der *Giant-Route* durchgeführt. Insgesamt wird also im Rahmen der VND zwischen der k -opt-Suche für einzelne Touren, der k -opt-Suche auf der Giant-Route und der Suche in der CT-Nachbarschaft bzw. den angegebenen CT-Nachbarschaften in dieser Reihenfolge gewechselt.

Die ersten sechs Nachbarschaften sind nach wachsender Größe geordnet. Je größer die betrachtete CT-Nachbarschaft ist, desto besser sind im Mittel auch die erhaltenen Lösungen. Lediglich die CT-Nachbarschaft ESD liefert etwas schlechtere Lösungen als die geringfügig kleinere Nachbarschaft ESH. Gleichzeitig wächst erwartungsgemäß auch der Rechenaufwand deutlich an. Dies ist einerseits darauf zurückzuführen, daß wesentlich größere Transfergraphen aufgebaut werden müssen. Andererseits bricht die Suche auch später ab, weil mehr Verbesserungen gefunden werden.

Die Ergebnisse in den vier letzten Spalten sind besser als bei allen übrigen Rechnungen, insbesondere auch besser als die Ergebnisse in der Spalte ESDHT. Dies ist erstaunlich, weil die Nachbarschaft ESDHT alle übrigen Nachbarschaften umfaßt. Eine mögliche Ursache für dieses Ergebnis kann darin bestehen, daß für die Suche nach verbessernden Nachbarschaftsschritten im Transfergraphen eine Heuristik verwendet wird. Weil der Transfergraph der Nachbarschaft ESDHT sehr groß ist, könnten verbessernde Cyclic Transfers durch die Heuristik leicht „übersehen“ werden.

Auffallend sind die großen Unterschiede der Ergebnisse, abhängig davon, ob mit oder ohne die k -opt-Suche auf der Giant-Route gerechnet wird. Die wesentlich besseren Ergebnisse beim Einsatz der k -opt-Suche auf der Giant-Route können nicht mit vergleichbarem Rechenaufwand erreicht werden, wenn entweder nur CT-Nachbarschaften oder nur k -opt-Nachbarschaften untersucht werden.²¹ Offenbar ergänzen sich die beiden Nachbarschaften beim VRP sehr gut. Dies läßt sich auch beobachten, wenn man den Ablauf einer solchen Optimierung verfolgt. Durch den Wechsel zwischen den unterschiedlichen Nachbarschaften können häufig neue Verbesserungen gefunden werden.

Der erhöhte Rechenaufwand kann durch die Verbesserung der Lösungsqualität gerechtfertigt werden, die alleine durch das Verwenden größerer CT-Nachbarschaften mit vergleichbarem Aufwand nicht erreicht wurde. Dabei ist anzumerken, daß der Rechenaufwand sehr stark von der Problemgröße ab-

²¹Wenn ausschließlich die reduzierte 4-opt-Suche auf der Giant-Route, die bei den übrigen Rechnungen verwendet wurde, und keine CT-Nachbarschaft in die VND aufgenommen wird, liegt die Abweichung von der besten bekannten Lösungen im Mittel bei 6,8 Prozent.

hängig ist. Dies ist der Darstellung in Abb. 6.1 nicht zu entnehmen, weil dort der Mittelwert der Rechenzeiten über alle 14 Instanzen abgebildet ist. Für die erste Instanz mit nur 50 Kunden (vgl. Tabelle 6.1 S. 167) werden z.B. höchstens 10,1 Sekunden und im Mittel über alle betrachteten Nachbarschaften nur 5,2 Sekunden benötigt. Dabei wird fast immer das Optimum gefunden. Am längsten dauert das Lösen der Probleminstanz 5 mit 199 Kunden, wenn die 4-opt-Suche auf der Giant-Route und die Nachbarschaft ESDHT eingesetzt wird. Dabei beträgt die Rechenzeit fast 2000 Sekunden.

Durch das Vorschalten kleinerer CT-Nachbarschaften, wie es in den fünf rechten Spalten der Abb. 6.1 dargestellt ist, können solche extrem hohen Rechenzeiten vermieden werden, ohne die Lösungsgüte zu beeinträchtigen. Die Rechenzeit für die Probleminstanz 5 reduziert sich z.B. auf etwa 1100 Sekunden, wenn wie in der letzten Spalte angegeben zusätzlich die Nachbarschaft ESD untersucht wird.

► Unter Berücksichtigung von CT-Nachbarschaften können beim VRP oft deutlich bessere Lösungen gefunden werden, als bei einer reinen k -opt-Suche. Besonders die Kombination einer k -opt-Suche auf der Giant-Route mit der Suche in CT-Nachbarschaften ist sehr wirkungsvoll. Die Untersuchung großer CT-Nachbarschaften verursacht einen relativ hohen Rechenaufwand. Dieser kann dadurch reduziert werden, daß zuerst kleine CT-Nachbarschaften untersucht und größere CT-Nachbarschaften erst danach betrachtet werden.

6.4.3 Ergebnisse für das VRPTW

Dieselbe Untersuchung, die wir zuvor für das VRP vorgenommen haben, führen wir nun auch für die VRPTW-Instanzen von Solomon durch. Dazu soll soweit wie möglich die gleiche Versuchsanordnung benutzt werden. Bei den folgenden Betrachtungen wird die Qualität der Lösungen wie zuvor ausschließlich an der insgesamt zurückzulegenden Strecke einer Lösung gemessen. Die Minimierung der Touren- bzw. Fahrzeugzahl soll zunächst unberücksichtigt bleiben, weil die Reduktion der Tourenzahl ein eigenes schwieriges Problem darstellt, das nur mit speziellen Methoden gut gelöst werden kann.

Da die Zahl der Touren einen wesentlichen Einfluß auf die minimal zurückzulegende Strecke haben kann, ist es nur sinnvoll Lösungen mit derselben Tourenzahl zu vergleichen.²² Wir legen die Tourenzahl mit der Bestimmung

der Startlösung fest und vergleichen anschließend nur die Verbesserungen hinsichtlich der Gesamtstrecke einer Lösung. Weil sich die besten publizierten Lösungen, teilweise auf eine andere Tourenzahl beziehen, können diese nicht zum Vergleich herangezogen werden. Stattdessen wird die Verbesserung der Lösungen gegenüber der Ausgangslösung zur Beurteilung der Heuristiken benutzt. Dies ist ausreichend, um einen Vergleich der verschiedenen CT-Nachbarschaften untereinander vorzunehmen.

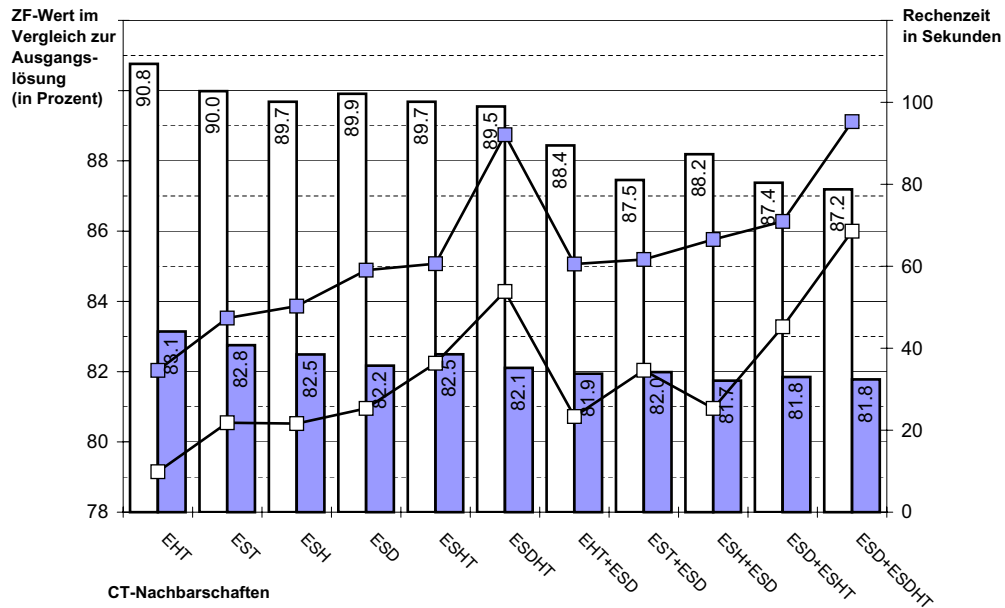
Für die Untersuchung werden erneut nur die Probleminstanzen der Problemklassen R1, R2, RC1 und RC2 herangezogen. Die Ausgangslösung wird mit der I1-Heuristik bestimmt. Anschließend werden alle Touren mit einer 4-opt-Suche verbessert. Die dann erhaltene Lösung betrachten wir im folgenden als Startlösung der zu vergleichenden Verbesserungsverfahren.

Das Diagramm in Abb. 6.2 stellt die Ergebnisse in gleicher Form wie zuvor beim VRP dar. Die linke Größenachse gibt jetzt an, wie groß die insgesamt zurückzulegende Strecke im Verhältnis zur Ausgangslösung ist.

Mit jeder der betrachteten CT-Nachbarschaften kann eine deutliche Verbesserung erzielt werden. Auffallend ist, daß die Nachbarschaft ESD, die beim VRP relativ schlecht abgeschnitten hat, jetzt im Vergleich zu den anderen Nachbarschaft zu etwas besseren Lösungen führt. Eine Erklärung dafür könnte sein, daß die Touranfangsstücke bzw. Tourendstücke, aus denen die Knotenmengen H und T aufgebaut werden, teilweise recht lang sind. Bei der Existenz von Zeitfenstern können die zugehörigen Kundenknoten häufig nicht vollständig in eine andere Tour eingebaut werden, so daß der Transfergraph relativ wenig Bögen enthält. Im Gegensatz dazu bestehen die in S bzw. D enthaltenen Knotenmengen stets nur aus ein bzw. zwei Kundenknoten und der zugehörige Transfergraph wird i.d.R. deutlich mehr Bögen besitzen.

Besonders auffallend ist auch bei den VRPTW-Instanzen der große Unterschied in der Lösungsqualität, abhängig davon, ob die k -opt-Suche auf der Giant-Route in die VND einbezogen wird oder nicht. Analog zur Argumentation beim VRP schließen wir daraus, daß sich die CT-Nachbarschaften und die k -opt-Suche auf der Giant-Route gegenseitig sehr gut ergänzen. Die Kombination mehrerer CT-Nachbarschaften führt auch bei den betrachteten Instanzen von Solomon sowohl mit als auch ohne k -opt-Suche auf der Giant-Route zu Verbesserungen der Lösungsqualität, die allein durch die Vergrößerung der CT-Nachbarschaften nicht erreicht werden.

²²siehe auch die Abschnitte 4.5 und 6.3.5



helle Füllung: k -opt-Suche nur für einzelne Touren

dunkle Füllung: k -opt-Suche auf der Giant-Route

Balken/linke Größenachse: Vergleich der Gesamtstrecke mit der Startlösung

Linien/rechte Größenachse: mittlere Rechenzeit in Sekunden (AMD Athlon 1400MHz)

Abbildung 6.2: Vergleich verschiedener Cyclic-Transfer-Nachbarschaften beim VRPTW

► Die CT-Nachbarschaften bilden auch beim VRPTW eine wertvolle Ergänzung zur k -opt-Suche auf der Giant-Route. Die Qualität der Lösungen verbessert sich mit der Untersuchung größerer CT-Nachbarschaften. Wie beim VRP ist es sinnvoll, mehrere CT-Nachbarschaften in einer VND zu integrieren und mit der Untersuchung der kleineren Nachbarschaften zu beginnen.

6.4.4 Auflösung von Touren beim VRPTW

Bei den VRPTW-Instanzen von Solomon werden i.d.R. die Minimierung der Tourenanzahl und die Minimierung der gesamten Strecke als Ziele betrachtet.

Bei den bisherigen Rechnungen wurde die Tourenanzahl, deren Reduktion i.a. als das dominierende Ziel angesehen wird, nicht explizit berücksichtigt.

Die in Bezug auf die Minimierung der Tourenanzahl erfolgreichsten heuristischen Verfahren, verwenden spezielle Methoden, um Lösungen mit geringer Tourenzahl zu finden. Eine häufig praktizierte Vorgehensweise besteht darin, die Tourenzahl einer gegebenen Lösung durch das Auflösen einer Tour zu reduzieren. Eine solche Tourauflösung wird dadurch erreicht, daß alle Kunden einer Tour t_{rem} auf die übrigen Touren der aktuellen Lösung y^{akt} verteilt werden. Wenn dies gelingt, kann t_{rem} anschließend aus der Lösung gestrichen werden.

In diesem Unterabschnitt führen wir ein Rechenexperiment durch. Damit soll der im Abschnitt 4.5 beschriebenen Ansatz erprobt werden, eine solche Tourauflösung unter Verwendung von Cyclic-Transfer-Nachbarschaften zu suchen. Wir testen die Tourauflösung in mehreren Schritten (vgl. Abschnitt 4.5) und benutzen dafür die CT-Nachbarschaft ESD. Für die Durchführung der Rechnungen werden folgende Festslegungen getroffen:

1. Bei der Kostenberechnung einer Tour werden ausreichend hohe Fixkosten für jede Tour eingerechnet. Dadurch wird erreicht, daß jeder Nachbarschaftsschritt, der die Tourenzahl einer Lösung reduziert, selbst dann ein verbessernder Schritt ist, wenn sich die gesamte Gesamtstrecke der Lösung erhöht. Für die Instanzen von Solomon werden Fixkosten in Höhe von 1000 je Tour verwendet, die für diesen Zweck ausreichend groß sind.
2. Besteht die aktuelle Lösung y^{akt} aus m Touren $t_1, \dots, t_m$, so wird grundsätzlich jede der m Touren darauf untersucht, ob sie aufgelöst werden kann. Wird eine solche Möglichkeit zur Tourauflösung gefunden, dann wird diese unmittelbar ausgeführt. Anschließend die Prozedur von neuem gestartet.
3. Während des Versuchs, die Kunden einer Tour t_{rem} von y^{akt} auf andere Touren zu verteilen, die Suche in der CT-Nachbarschaft modifiziert:
 - (a) Alle Bögen, die Kundenknoten aus anderen Touren in die Tour t_{rem} einfügen würden, werden verboten, d.h. aus dem Transfergraphen eliminiert.
 - (b) Nachbarschaftsschritte, die eine Verkürzung der Tour t_{rem} bewirken, werden proportional zur Zahl der entfernten Knoten belohnt.

Für die Instanzen von Solomon werden die Kosten eines solchen Nachbarschaftsschritts für jeden Kunden, der aus t_{rem} entfernt wurde, um 100 reduziert.

Die Suche nach Tourauflösungen mit Hilfe einer CT-Nachbarschaft kann selbst als eine spezielle Nachbarschaftssuche aufgefaßt werden. Wir setzen für die Suche die CT-Nachbarschaft ESD ein und bezeichnen die entstehende „Tourauflösungs-Nachbarschaft“ mit ESD_{rem} . Wir integrieren die Nachbarschaft ESD_{rem} in eine VND, wie sie den Rechnungen im Abschnitt 6.4.3 zugrundeliegt.²³

Ausgehend von der Lösung der I1-Heuristik wurde die VND auf alle 39 Instanzen der Problemklassen R1, R2, RC1 und RC2 angewendet. Um den Einfluß der einzelnen Nachbarschaften besser deutlich zu machen, wurde die VND schrittweise erweitert. Die Tabelle 6.15 faßt die Ergebnisse zusammen.

	best ^a	I1 ^b	4-opt ^c	4-opt+CT ^d	4opt+CT+ESD _{rem} ^e
R1	148	162	159	154	153
RC1	92	108	106	100	100
R2	30	36	36	34	34
RC2	26	31	31	29	29
Alle	291	338	332	317	316

^a beste bekannte Lösungen²⁴, die aus insgesamt 15 verschiedenen Veröffentlichungen zusammengestellt wurden

^b Lösungen der I1-Heuristik

^c wie bei ^b mit anschließender 4-opt-Suche auf der Giant-Route

^d wie bei ^c und zusätzlich Suche in den CT-Nachbarschaften ESH und ESD

^e wie bei ^d und zusätzlich Tourauflösung mit ESD_{rem}

Tabelle 6.15: Anzahl der Touren beim VRPTW

Die mit der I1-Heuristik berechneten Lösungen besitzen 47 Touren mehr als notwendig. Weil den einzelnen Touren von Beginn an hohe Fixkosten zugerechnet werden, findet die bereits die 4-opt-Suche auf der Giant-Route einige Tourauflösungen. Danach werden mit insgesamt 332 Touren noch immer 41 Touren mehr benötigt, als bei den besten bekannten Lösungen. Durch

²³Als Cyclic-Transfer-Nachbarschaften werden ESH und ESD eingesetzt. Die VND umfaßt also insgesamt die Nachbarschaften: 2-opt-, 3-opt- und 4-opt-Nachbarschaft für die einzelnen Touren, 2-opt-, 3-opt- und 4-opt-Nachbarschaft für die Giant-Route, ESH, ESD und die Nachbarschaft ESD_{rem} .

²⁴url: www.top.sintef.no/vrp/bknown.html, am 5.12.2002

die zusätzliche Untersuchung der CT-Nachbarschaften ESH und ESD werden weitere 15 Touren aufgelöst. Die Ergebnisse der letzten Spalte werden erhalten, wenn die zuvor beschriebene Tourauflösung in die VND einbezogen wird. Durch diese sehr aufwendigen Rechnungen kann insgesamt nur eine weitere Tour aufgelöst werden.

Dieses Ergebnis ist gemessen am zusätzlichen Aufwand und am möglichen Verbesserungspotential unbefriedigend. Folgende Ursachen können dafür in Betracht gezogen werden:

1. Bei dem Versuch, Touren aufzulösen, wird lediglich das Umverteilen einzelner Kundenknoten aus der Tour t_{rem} belohnt. Es wird nicht darauf geachtet, daß die übrigen Touren „eng gepackt“ werden müssen, damit noch viele weitere Kunden darin eingebaut werden können. Gegebenenfalls ist es besser eine geänderte Zielfunktion zu verwenden, z.B. die Minimierung der Gesamtfahrzeit anstelle der Minimierung der Gesamtstrecke. Dies würde dazu führen, daß Kunden „zeitlich eng gepackt“ werden und so größere zeitliche Freiräume erhalten bleiben. Dann passen ggf. noch weitere Kunden in die Tour.
2. Die vorgeschlagene Methode zur Auflösung der Touren ist verhältnismäßig aufwendig. Deshalb kann sie nur selten aufgerufen werden, um die Rechenzeit nicht inakzeptabel anwachsen zu lassen. Cordone und Wolfler Calvo [CW01], die mit einem wesentlich einfacheren Verfahren bessere Lösungen finden konnten, versuchen so häufig wie möglich, Touren aufzulösen. Um den Aufwand zu reduzieren, wäre es denkbar, zunächst mit einer wesentlich einfacheren Heuristik zu beginnen, um möglichst viele Kunden aus der Tour t_{rem} zu eliminieren. Erst wenn diese Heuristik nicht mehr weiterkommt und noch Kunden in t_{rem} verblieben sind, werden CT-Nachbarschaften betrachtet.

Da der Fokus dieser Arbeit nicht auf die Entwicklung einer speziellen Methode zur Reduktion der Tourenanzahl gelegt werden soll, wurde von einer umfassenden Analyse der Ursachen im Rahmen dieser Arbeit abgesehen. Aufgrund dessen wurde auch auf die Untersuchung des Ansatzes zur Tourauflösung in einem Schritt verzichtet.

6.5 Eine einfache Diversifikation mit Kick-Schritten

Bisher wurden ausschließlich Verfahren auf der Basis einer reinen Lokalen Suche betrachtet, d.h. es werden grundsätzlich keine Nachbarschaftsschritte ausgeführt, die nicht verbessernd sind. Mit der Einbindung der vorgeschlagenen Heuristiken in eine Meta-Heuristik bestehen zusätzliche Potentiale, die Qualität der gefundenen Lösungen zu erhöhen. Es soll im folgenden gezeigt werden, daß es bereits mit einfachen Meta-Strategien möglich ist, heuristische Verfahren zu konstruieren, deren Lösungsqualität mit den besten bekannten Meta-Heuristiken verglichen werden kann.

Wir beschreiben zunächst, wie eine Diversifikation der Suche mit Kick-Schritten erreicht werden kann und wie geeignete Kick-Schritte bestimmt werden. Anschließend wird gezeigt, daß es mit diesem Ansatz möglich ist, die Qualität der gefundenen Lösungen für TSPTW, VRP und VRPTW weiter zu verbessern.

6.5.1 Diversifikation mit Kicks

Wir verwenden als Meta-Strategie eine Diversifikation der Suche anhand von großen Schritten (Kick-Schritten), die nicht verbessernd sein müssen und immer dann ausgeführt werden, wenn die Lokale Suche in einem lokalen Optimum terminiert. Durch einen solchen Kick-Schritt soll die aktuelle Region des Lösungsraums verlassen und die Suche in einer neuen Region fortgesetzt werden. Die Suche mit Hilfe von Kick-Schritten kann in einem einfachen Algorithmus zusammengefaßt werden:

Algorithmus 6.2. Lokale Suche mit Kick-Schritten

Gegeben: $\mathcal{N}$ Nachbarschaft für die Lokale Suche und
 $\mathcal{N}_{kick}$ Nachbarschaft für die Kick-Schritte

1. Bestimme mit einer Eröffnungsheuristik eine Startlösung y' .
2. Setze die beste Lösung $y^{best} := y'$.
3. (*Schleife*) Solange eine *Abbruchbedingung* nicht erfüllt ist, wiederhole:

- 3.1** Bestimme ausgehend von y' mit einer Lokalen Suche mit $\mathcal{N}$ eine lokal optimale Lösung y'' .
- 3.2** Falls y'' besser ist als y^{best} , setze $y^{best} := y''$.
- 3.3** Wähle zufällig einen Nachbarschaftsschritt $\mathbf{m}^{kick} \in \mathcal{N}_{kick}$ mit $\mathbf{m}^{kick}(y^{best}) \in Y$ und setze $y' = \mathbf{m}^{kick}(y^{best})$.

Damit die Diversifikation gut funktioniert, sollte jeder Kick-Schritt eine neue zulässige Lösung y' erzeugen, die im Idealfall alle folgenden Eigenschaften aufweist:

Aufwand: Die Bestimmung des Kick-Schritts verursacht nur geringen Aufwand.

Qualität: Die durch den Kick-Schritt erzeugte neue Lösung y' ist besser als die Lösung eines schnellen Eröffnungsverfahrens.

Unterschiedlichkeit:

1. Die Lösung y' liegt in einem Teil des Lösungsraums, der noch nicht oder verhältnismäßig wenig untersucht wurde. Insbesondere wurde die Lösung y' bislang noch nicht untersucht.
2. Die Lokale Suche in $\mathcal{N}$ führt ausgehend von y' zu einem neuen lokalen Optimum $y'' \neq y^{best}$.

Insbesondere die Gewährleistung der Unterschiedlichkeit im Sinne der letzten Forderungen würde einen sehr hohen Aufwand für die Bestimmung eines Kick-Schritts nach sich ziehen und steht damit im Widerspruch zu der ersten Forderung. Da die gewünschten Eigenschaften also nicht garantiert werden können, versuchen wir stattdessen, Kick-Schritte zu erzeugen, für die diese Eigenschaften häufig zutreffen. Im Ergebnis zahlreicher unterschiedlicher Experimente wurde dafür folgende Vorgehensweise entwickelt.

Wir dekomponieren das Problem, einen guten Kick-Schritt zu bestimmen, indem die Kick-Schritte aus mehreren Teilschritten zusammengesetzt werden, die kleinen k -opt-Nachbarschaften entnommen werden. Innerhalb eines Kick-Schritts $P = (p_1, \dots, p_m)$ darf jeder Bogen höchstens einmal geändert werden, d.h. die Teilschritte $p_1, \dots, p_m$ müssen paarweise unterschiedliche Bögen der aktuellen Lösung verändern. Wir wählen die Teilschritte aus k -opt-Nachbarschaften mit $3 \leq k \leq 5$ aus und setzen einen Kick-Schritt abhängig von der Problemklasse aus 3 bis 10 Teilschritten zusammen. Ein Kick-Schritt P ist

demzufolge fast immer²⁵ ein Element einer echten k -opt-Nachbarschaft mit $k \geq 3 \cdot m$.

Die Teilschritte $p_1, \dots, p_m$ werden nacheinander bestimmt. Jeder Teilschritt p_i ist ein optimaler Schritt aus einer zufällig gewählten, stark reduzierten k -opt-Nachbarschaft. Diese reduzierten k -opt-Nachbarschaften werden dadurch gebildet, daß

- nur ein k -opt-Typ berücksichtigt wird, der zufällig ausgewählt wird,
- Bögen in der Nachbarschaft verboten werden, die bereits in einem anderen Teilschritt geändert wurden und
- zufällig bestimmte, enge Intervalle für die Segmentlängen der einzelnen Segmente vorgegeben werden.

Sind die Segmentlängen der k -opt-Schritte für die Lokale Suche beschränkt worden, so werden die Intervalle für die Segmentlängen eines Teilschrittes p_i für den Kick-Schritt P so gewählt, daß ein einzelner k -opt-Schritt innerhalb der Lokalen Suche nicht ausreicht, um den Teilschritt p_i rückgängig zu machen. Kann in einer der gebildeten Nachbarschaften kein zulässiger Teilschritt gefunden werden, so wird eine neue Nachbarschaft bestimmt und darin gesucht.

Durch die separate Bestimmung der Teilschritte können diese aus einer kleinen Nachbarschaft optimal ausgewählt werden, ohne den Rechenaufwand sehr groß werden zu lassen. Damit können vollkommen zufällig gewählte und ggf. extrem schlechte Lösungen vermieden werden. Die Auswahl aus einer kleinen Nachbarschaft erleichtert i.d.R. auch das Auffinden zulässiger Lösungen, weil nur wenige Bögen gleichzeitig verändert werden.

6.5.2 Kick-Schritte für das TSPTW

Im Unterschied zu den bisherigen analytischen Betrachtungen geht es jetzt darum, hochqualitative Lösungen in möglichst kurzer Zeit zu bestimmen. Dafür werden die Ergebnisse der bereits durchgeführten Untersuchungen für die Gestaltung einer effizienten Lokalen Suche exemplarisch umgesetzt.

Wir verwenden für die Lösung reduzierte k -opt-Nachbarschaften und diversifizieren die Suche mit mehreren Kick-Schritten. Die Versuchsanordnung wird durch folgende Festlegungen beschrieben:

²⁵Die Aussage trifft lediglich für einige Sonderfälle nicht zu, bei denen in verschiedenen Teilschritten direkt benachbarte Bögen getauscht werden.

Startlösung: Als Eröffnungsverfahren wird die I1-Heuristik von Solomon verwendet.

k -opt-Nachbarschaften: Wir berücksichtigen k -opt-Nachbarschaften bis $k = 4$ mit folgenden Parametern:

k	max. Anzahl invertierter Segmente	max. Anzahl langer Segmente(k_ℓ)	max. Länge der kurzen Segmente (ℓ_s)
2	1	–	–
3	2	1	10
4	2	1	5

Kick-Schritte: In zwei Versuchen werden einmal insgesamt 10 und einmal 20 Kick-Schritte durchgeführt. Diese werden jeweils aus 8 Teilschritten zusammengesetzt. Die reduzierten Nachbarschaften, aus denen die Teilschritte bestimmt werden, sind zufällig gewählte k -opt-Typen mit $3 \leq k \leq 5$. Die Segmentlängen der inneren Segmente werden nur in einem Intervall der Länge 5 variiert, das für jedes Segment einzeln zufällig bestimmt wird.

Wurde nach einem Kick-Schritt eine neue beste Lösung gefunden, wird die Anzahl der Kick-Schritte mindestens auf die Hälfte der ursprünglichen Anzahl von Kicks (d.h. 10 bzw. 5) erhöht.

Die Ergebnisse dieser Rechnung sind in Tabelle 6.16 zusammengefaßt und werden mit den Ergebnissen von Balas und Simonetti verglichen. Balas und Simonetti können für viele ihrer Lösungen die Optimalität beweisen.²⁶ Bei den übrigen Instanzen sind ihre Lösungen (mit einer Ausnahme) höchstens 2,5% vom Optimum entfernt.

Wie die Ergebnisse in Tabelle 6.16 belegen, können die hervorragenden Ergebnisse von Balas und Simonetti bei den kleineren Instanzen ebenfalls erreicht werden. Auch bei den übrigen Problem-Instanzen sind die Abweichungen gering. In den Spalten Δz ist jeweils die relative Abweichung zur besten von Balas und Simonetti bestimmten Lösung angegeben. Diese Abweichung beträgt bei 10 Kick-Schritten im Mittel 0,7 Prozent, bei 20 Kick-Schritten sinkt sie im Mittel auf 0,4 Prozent. Für die Instanz rgb50a konnte bereits nach einem Kick-Schritt eine Lösung gefunden werden, die mehr als zwei Prozent besser als die beste Lösung von Balas und Simonetti ist.

²⁶vgl. Tabelle 6.2 auf Seite 173

²⁷vgl. [BS01, S. 20]

Problem- Instanz	Balas und Simonetti				kopt+10 Kicks			kopt+20 Kicks		
	z^a	CPU ^a	z^b	CPU ^b	z	CPU ^c	Δz	z	CPU ^c	Δz
rbg50a	441	107	431	1123	422	44	-2,1%	422	84	-2,1%
rbg55a	814	26			814	19	0,0%	814	41	0,0%
rbg67a	1048	29			1048	34	0,0%	1048	65	0,0%
rbg88a	1153	22			1153	82	0,0%	1153	113	0,0%
rbg92a	1093	48			1103	61	0,9%	1098	148	0,5%
rbg125a	1409	32			1412	92	0,2%	1412	195	0,2%
rbg132	1360	61			1370	99	0,7%	1370	209	0,7%
rbg132.2	1085	138	1083	1135	1085	168	0,2%	1085	226	0,2%
rbg152	1783	38			1788	140	0,3%	1786	261	0,2%
rbg152.2	1628	216			1662	135	2,1%	1655	267	1,7%
rbg152.3	1556	316	1542	2765	1549	167	0,5%	1542	322	0,0%
rbg172a	1799	111			1826	183	1,5%	1815	355	0,9%
rbg193.2	2018	408			2059	168	2,0%	2059	357	2,0%
rbg201a	2189	127			2214	237	1,1%	2214	463	1,1%
rbg233.2	2190	447			2254	260	2,9%	2205	519	0,7%
Alle	21566	2126			21759	1889	0,7%	21678	3625	0,4%

^{a, b} Die Rechnungen von Balas und Simonetti wurden auf einer Sun Ultra I Workstation mit verschiedenen Einstellungen durchgeführt, deren Rechenzeiten sich deutlich unterscheiden. Die Spalte *a* bezieht sich auf eine Standardeinstellung des Verfahrens.²⁷ In Spalte *b* sind bessere Lösungen zusammengefaßt, die mit anderen Einstellungen und erheblich höheren Rechenzeiten bestimmt wurden.

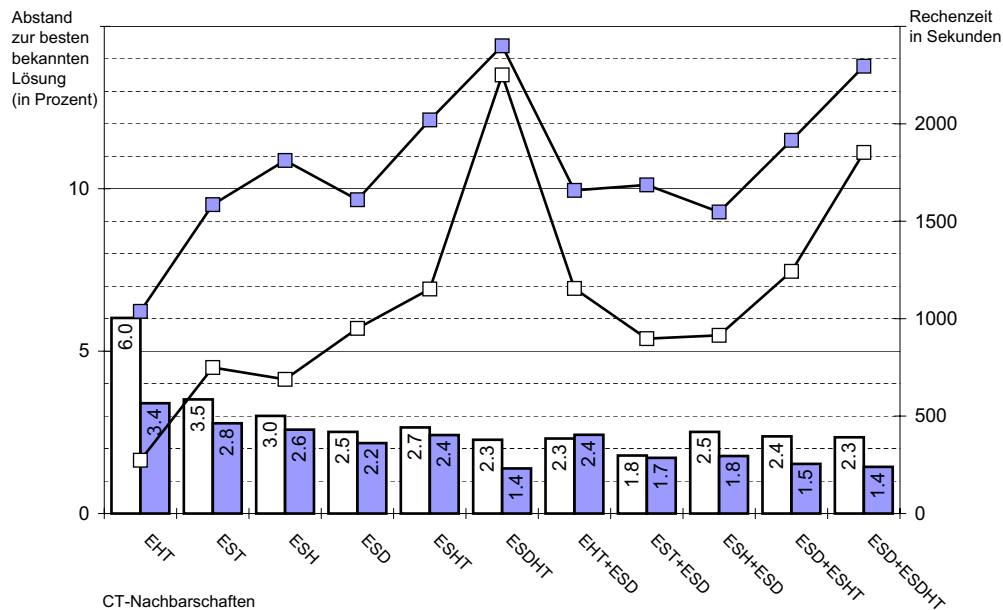
^c Rechenzeiten auf einem AMD Athlon Prozessor (1400MHz).

Tabelle 6.16: Vergleich der *k*-opt-Suche mit Kick-Schritten mit den Ergebnissen von Balas und Simonetti

Die Rechenzeiten auf verschiedenen Rechnerarchitekturen sind nur schwer miteinander vergleichbar. Der von Balas und Simonetti verwendete Computer ist jedoch etwa um den Faktor zehn langsamer, wenn man den Performance-Test von Dongarra [Don02] zugrundelegt. Die Rechenzeit für die Kombination von *k*-opt-Suche und Kick-Schritten ist sehr stark von der Zahl der durchgeführten Kick-Schritte abhängig. Bei den kleineren Probleminstanzen wurde die optimale Lösung oft bereits ohne bzw. mit wenigen Kick-Schritten erreicht. Berücksichtigt man nur die Zeit bis zum Auffinden der besten Lösung, so ist der Rechenaufwand dafür bei beiden Lösungsansätzen etwa gleich.

6.5.3 Kick-Schritte für VRP und VRPTW

Analog zum TSPTW prüfen wir nun, welche Auswirkungen Kick-Schritte beim VRP bzw. beim VRPTW haben. Dazu wiederholen wir die Rechnungen aus den Abschnitten 6.4.2 und 6.4.3 bei denen wir die Kombination der k -opt-Suche mit verschiedenen CT-Nachbarschaften untersucht haben. Diesmal führen wir zusätzlich immer dann einen Kick-Schritt durch, wenn die VND aus k -opt-Nachbarschaften und CT-Nachbarschaften terminiert. Es werden insgesamt zehn Kick-Schritte ausgeführt, die diesmal aus je drei Teilschritten zusammengesetzt werden. Die Teilschritte werden aus denselben reduzierten k -opt-Nachbarschaften mit $3 \leq k \leq 5$ ausgewählt, wie sie bereits für das TSPTW beschrieben wurden.



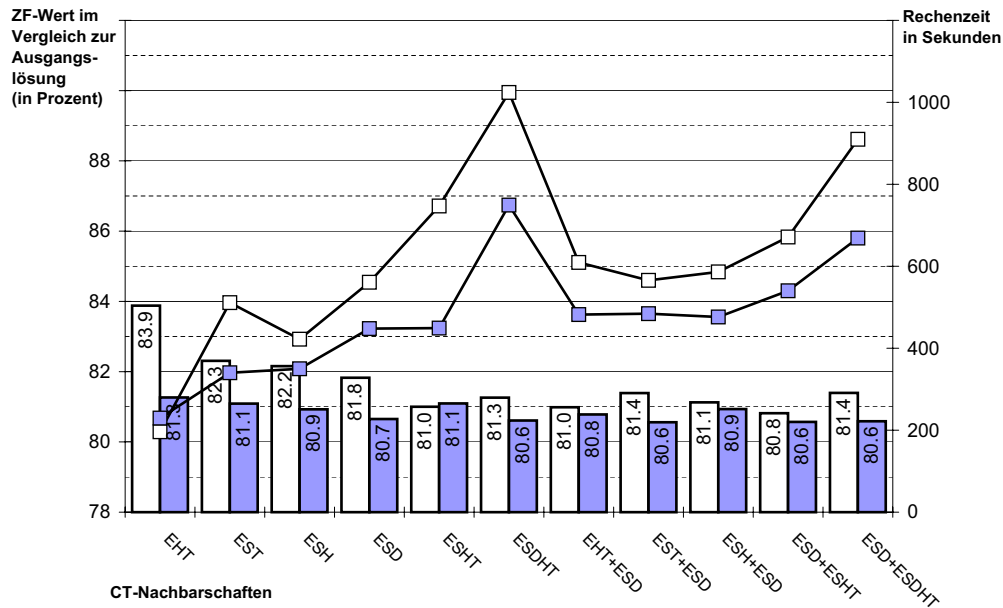
helle Füllung: k -opt-Suche nur für einzelne Touren

dunkle Füllung: k -opt-Suche auf der Giant-Route

Balken/linke Größenachse: mittlerer Abstand zur besten bekannten Lösung

Linien/rechte Größenachse: mittlere Rechenzeit in Sekunden (AMD Athlon 1400 MHz)

Abbildung 6.3: Ergebnisse beim Einsatz von Cyclic-Transfer-Nachbarschaften und zehn Kick-Schritten beim VRP



helle Füllung: k -opt-Suche nur für einzelne Touren

dunkle Füllung: k -opt-Suche auf der Giant-Route

Balken/linke Größenachse: Vergleich der Gesamtstrecke mit der Startlösung

Linien/rechte Größenachse: mittlere Rechenzeit in Sekunden (AMD Athlon 1400MHz)

Abbildung 6.4: Ergebnisse beim Einsatz von Cyclic-Transfer-Nachbarschaften und zehn Kick-Schritten beim VRPTW

Die Abbildungen 6.3 und 6.4 zeigen die Ergebnisse dieser Rechnungen. Ein Vergleich mit Abb. 6.1 auf Seite 198 macht deutlich, daß die gefundenen Lösungen beim VRP durch die Diversifikation mit Hilfe der Kick-Schritte erheblich verbessert werden konnten. Die mittlere Abweichung von den besten bekannten Lösungen reduziert sich bei der Kombination der CT-Nachbarschaften mit einzelnen Touren (helle Balken) von etwa zehn Prozent ohne Kick-Schritte bei fast allen betrachteten CT-Nachbarschaften auf weniger als drei Prozent. Der Rechenaufwand erhöht sich allerdings durch die insgesamt zehn Kick-Schritte auch etwa um den Faktor zehn.

Bei der Kombination von CT-Nachbarschaften mit der k -opt-Suche auf der Giant-Route (dunkle Balken) konnten auch ohne Kick-Schritte Lösungen

gefunden werden, die (abhängig von den verwendeten CT-Nachbarschaften) im Mittel nur etwa drei bis vier Prozent von den besten bekannten Lösungen abweichen. Auch diese Ergebnisse können durch zusätzliche Kick-Schritte noch einmal verbessert werden, so daß die mittleren Abweichungen meistens um etwa die Hälfte reduziert werden. Der Aufwand wächst mit den zehn Kick-Schritten etwa mit Faktor fünf bis sechs.

Beim Einsatz von Kick-Schritten für die Lösung der VRPTW-Instanzen von Solomon sind die zu beobachtenden Effekte ähnlich (siehe die Abbildungen 6.2 auf Seite 202 und 6.4 auf der vorherigen Seite). Bei der Kombination der CT-Nachbarschaften mit der k -opt-Suche auf einzelnen Touren reduziert sich die Gesamtstrecke von 87 bis 90 Prozent der Strecke der Ausgangslösung auf etwa 82 Prozent. Werden CT-Nachbarschaften und die k -opt-Suche auf der Giant-Route miteinander verbunden, kann die mittlere Gesamtstrecke der Lösungen durch die Kick-Schritte um etwa ein Prozent verbessert werden.

Der Aufwand bei der Kombination der CT-Nachbarschaften mit der k -opt-Suche auf einzelnen Touren ist diesmal fast immer höher als bei der Kombination mit der k -opt-Suche auf der Giant-Route. Dies wird plausibel, wenn man berücksichtigt, daß nach einem Kick-Schritt kaum Verbesserungen durch k -opt-Schritte auf einzelnen erzielt werden. Lösungen hoher Qualität werden deshalb durch eine längere Folge von CT-Schritten erhalten. Dabei muß der relativ hohe Aufwand zum Aufbau und Absuchen der CT-Nachbarschaft relativ häufig.

► Durch eine Diversifikation der Suche mit Hilfe einer Meta-Strategie können die Ergebnisse, die mit der k -opt-Suche, bzw. mit einer Suche in k -opt- und Cyclic-Transfer-Nachbarschaften bestimmt werden, bei allen untersuchten Problemklassen verbessert werden. Auch die Suche in mehreren sehr großen Nachbarschaften enthebt demzufolge nicht von der Notwendigkeit, eine zusätzliche Diversifikation der Suche vorzunehmen, um sehr gute Lösungen erhalten zu können.

Zusammenfassung

Die Entwicklung leistungsfähiger Heuristiken für Vehicle Routing und Scheduling Probleme ist seit mehreren Jahrzehnten Gegenstand intensiver weltweiter Forschung. Die meisten entwickelten Algorithmen können jedoch nur auf Instanzen eines bestimmten Problems angewendet werden, dessen spezifische Eigenschaften für die effiziente Suche nach guten Lösungen ausgenutzt werden.

Für praktische Anwendungen stellt die fehlende Robustheit dieser Heuristiken gegenüber Änderungen der Modellstruktur ein ernsthaftes Problem dar. Es kommt häufig vor, daß erst während der ersten praktischen Tests oder sogar noch später neue Restriktionen „entdeckt“ oder durch geänderte Rahmenbedingungen tatsächlich erst später relevant werden. Muß infolgedessen eine andere Heuristik eingesetzt werden, so ist damit ggf. ein Redesign der verwendeten Software verbunden und hohe Folgekosten sind unvermeidbar.

Der Beitrag dieser Dissertation liegt in der Entwicklung effizienter Verbesserungsheuristiken auf der Basis eines allgemeinen Ressourcenmodells. Durch Beschränkungen von Ressourcen können mit diesem Modell sehr unterschiedliche Restriktionen, wie Beschränkungen der Tourdauer, der Tourlänge, der Anzahl der Kunden einer Tour, Zeitfenster, Reihenfolgeabhängigkeiten oder Inkompatibilitäten von Aufträgen usw., formuliert werden. Heuristiken, die auf diesem allgemeinen Modell basieren, lösen einen großen Teil aller Vehicle Routing und Scheduling Probleme.

Mit den k -opt- und den Cyclic-Transfer-Nachbarschaften werden zwei in der Literatur bekannte Klassen von Austausch-Nachbarschaften aufgegriffen. Die vorliegende Arbeit zeigt, daß diese Nachbarschaften unter Berücksichtigung der Ressourcenbeschränkungen effizient abgesucht werden können. Die Einbindung der Nachbarschaften in eine Meta-Heuristik ist die zugrundeliegende Intention der Untersuchungen, steht aber nicht selbst im Zentrum der Betrachtungen.

Über die eigentliche Entwicklung der Verbesserungsverfahren hinaus, besteht ein wichtiger Beitrag der Arbeit in der Verallgemeinerung, Weiterentwicklung und der teilweise erstmalig vorgenommenen formalen Beschreibung grundlegender Konzepte dieser Nachbarschaften. Erst die Einführung geeigneter Begriffe und Definitionen macht es möglich, bereits bekannte Zusammenhänge einfacher darzustellen und neue Erkenntnisse abzuleiten. Die vorgenommenen Generalisierungen bekannter Methoden decken zahlreiche Alternativen bei deren konkreter Ausgestaltung auf. Die Wirksamkeit einiger daraus abgeleiteter Vorschläge wird durch ausführliche Vergleichsrechnungen mit Benchmark-Probleminstanzen nachgewiesen.

Methodisch baut die Arbeit dem Artikel [Sav85] von Savelsbergh auf. Darin wird u.a. beschrieben, wie Zeitfensterrestriktionen bei der Suche in 2-opt- und 3-opt-Nachbarschaften mit Hilfe einer Lexikographischen Suche effizient überprüft werden können. Die Ideen von Savelsbergh werden in dieser Arbeit erweitert und auf abstrakte beschränkte Ressourcen und beliebige k -opt-Schritte übertragen. Darauf aufbauend kann die effiziente Überprüfung unterschiedlicher Restriktionen nach einem allgemeinen Schema durchgeführt werden. Dabei können auch mehrere, teilweise voneinander abhängige Restriktionen berücksichtigt werden.

k -opt-Nachbarschaften k -opt-Schritte werden in der Literatur fast ausschließlich durch die Angabe der zu löschenden und neu einzufügenden Kanten bzw. Bögen beschrieben. Die in dieser Arbeit gegebene Definition eines k -opt-Schritts orientiert sich, anders als sonst üblich, an der Betrachtung von Tour-Segmenten und verwendet vier verschiedene Operatoren, die diese Tour-Segmente modifizieren. Diese neuartige, stärker strukturierte Darstellung führt zu einer anderen Sicht auf k -opt-Schritte. Sie impliziert die Klassifizierung unterschiedlicher k -opt-Schritte und führt so zur Definition der „ k -opt-Typen“. Mit Hilfe der in dieser Arbeit entwickelten „abc-Notation“ können k -opt-Typen kompakt und anschaulich dargestellt werden.

Ebenfalls neu ist die Ausdehnung der k -opt-Nachbarschaften auf mehrere Touren beim m-TSPR anhand einer Giant-Route-Darstellung. Sie erlaubt die konsistente Beschreibung und die effiziente Suche nach k -opt-Schritten über die Tourgrenzen hinweg. Eine Verallgemeinerung des Verkettungsoperators, der für die Beschreibung der k -opt-Schritte eingeführt wurde, macht es darüber hinaus möglich, beliebige k -opt*-Schritte analog darzustellen und k -opt*-Nachbarschaften ebenso effizient zu durchsuchen.

Die Komplexität der Suche in einer k -opt-Nachbarschaft bleibt durch die eingesetzten Techniken im worst case auf $\mathcal{O}(n^k)$ beschränkt. Das Absuchen vollständiger k -opt-Nachbarschaften impliziert für $k > 3$ dennoch einen in der Regel inakzeptabel großen Aufwand. Zwar kann dieser durch den Einsatz von Kostenabschätzungen um bis zu 50 Prozent gesenkt werden, was aber letztlich wegen des mit der Größe des Problems schnell wachsenden Aufwands kaum eine Rolle spielt. Eine Alternative wird in einer gezielten Reduktion der k -opt-Nachbarschaften gesehen. Sie soll dazu dienen, Teile der Nachbarschaften von der Suche auszuschließen, die nur selten verbessernde Lösungen enthalten. Die Größe der untersuchten Nachbarschaften kann dadurch nachhaltig verringert werden, ohne wesentlich schlechtere Lösungen zu erhalten. Mit der teilweisen Beschränkungen von Segmentlängen und der Eingrenzung auf Bögen zu den ℓ nächsten Nachbarn eines Kundenknotens konnten erfolgreiche Ansätze identifiziert werden, um eine solche Reduktion der k -opt-Nachbarschaften zu erreichen.

Cyclic-Transfer-Nachbarschaften Die CT-Nachbarschaften bilden eine große Klasse knotenorientierter Austausch-Nachbarschaften. Der Begriff der Cyclic Transfers geht auf den Artikel [TP93] von Thompson und Psaraftis aus dem Jahr 1993 zurück. In dieser Arbeit wird das Konzept der Cyclic Transfers aufgegriffen und auf beliebige, zwischen verschiedenen Touren zyklisch zu tauschende Knotenmengen erweitert. Für solche Cyclic-Transfer-Schritte wird eine formale generische Beschreibung entwickelt, deren wichtigster Bestandteil einzelne Austauschoperationen (Transfers) sind, aus denen Cyclic-Transfer-Schritte zusammengesetzt werden. Auf der Basis dieser Darstellung ist es möglich, einzelne Transfers mit den Bögen eines Transfergraphen zu identifizieren, der die gesamte Cyclic-Transfer-Nachbarschaft einer gegebenen Lösung repräsentieren kann. Diese Idee wurde von Ahuja et. al. [AOS01], die dasselbe Vorgehen zur Lösung des Capacitated Minimum Spanning Tree Problem verwenden, übernommen und für VRSP adaptiert. Die Suche nach einem verbessernden Cyclic-Transfer-Schritt kann auf die Suche nach einem Kreis negativer Länge im Transfergraphen zurückgeführt werden, wobei zusätzliche Restriktionen zu berücksichtigen sind.

Durch die implizite Repräsentation der Cyclic-Transfer-Schritte und die Suche nach verbessernden Schritten auf der Basis der Dynamischen Programmierung ist es möglich, sehr große Nachbarschaften zu untersuchen. Bei der Überprüfung von Ressourcenbeschränkungen können die bereits für die

k -opt-Nachbarschaften eingeführten Konzepte erneut verwendet werden. Der Ansatz ist für VRSP mit beschränkten Ressourcen besonders attraktiv, weil die Einhaltung der Restriktionen nur bei dem Aufbau des Transfergraphen überprüft werden muß. Bei der anschließenden Suche nach verbessernden Nachbarschaftsschritten im Transfergraphen spielen die Ressourcenbeschränkungen keine Rolle mehr.

Rechenergebnisse Die beschriebenen Konzepte wurden zur Lösung von Benchmark-Probleminstanzen zum TSPTW, VRP und VRPTW eingesetzt. Anhand zahlreicher Vergleichsrechnungen wurden eine Reihe grundlegender Fragestellungen untersucht. Damit konnte u.a. gezeigt werden, daß die Verwendung von k -opt-Nachbarschaften mit $k > 3$ oft zu einer deutlichen Verbesserung der gefundenen Lösungen beitragen kann. Solche Nachbarschaften wurden bislang kaum untersucht, weil die Größe der Nachbarschaften exponentiell mit k wächst. Durch geeignete Einschränkungen der Nachbarschaften kann der für das Absuchen erforderliche Aufwand sehr stark verringert werden, ohne die Lösungsqualität zu beeinträchtigen. Entsprechend reduzierte k -opt-Nachbarschaften mit $k = 4$ bzw. $k = 5$ können häufig gewinnbringend eingesetzt werden. So konnten allein beim Übergang von der 3-opt- zur 4-opt-Nachbarschaft bei den untersuchten VRP-Instanzen im Mittel Verbesserungen von mehr als vier Prozent und bei den TSPTW- bzw. VRPTW-Instanzen von gut einem Prozent beobachtet werden.

Mit den k -opt-Nachbarschaften auf der Giant-Route und den CT-Nachbarschaften können beim m-TSPR zwei sehr unterschiedliche Nachbarschaftstypen miteinander kombiniert werden, die sich gegenseitig sehr gut ergänzen. Dadurch können beim VRP und beim VRPTW für die untersuchten Instanzen im Mittel um sechs bis sieben Prozent bessere Lösungen gefunden werden. Die Leistungsfähigkeit solcher hybriden Heuristiken kann durch die Einbindung in eine Meta-Heuristik, die eine gezielte Diversifikation der Suche herbeiführt, weiter erhöht werden. So verbesserten sich die gefundenen Lösungen durch geeignete Kick-Schritte um weitere ein bis zwei Prozent.

Viele moderne Meta-Heuristiken, wie Tabu-Search, Variable Neighborhood Search, Iterated Local Search, Guided Local Search usw., realisieren die Steuerung einer untergeordneten Lokalen Suche in einer oder mehreren Nachbarschaften. In der vorliegenden Arbeit wird der Fokus bewußt auf die Ge-

staltung einer solchen Lokalen Suche gelegt, die als Kernelement dieser Meta-Heuristiken angesehen werden kann. Mit den entwickelten Modellen und Methoden ist es erstmalig möglich, zwei große, sehr unterschiedliche Klassen von Nachbarschaften auf der Basis eines allgemeinen Ressourcenmodells effizient zu durchsuchen. Deren Einbindung in dafür geeignete Meta-Heuristiken, bleibt als eine erfolgversprechende Aufgabe künftigen Arbeiten vorbehalten. Mit den bereits erreichten Ergebnissen der reinen Lokalen Suche, die einige Meta-Heuristiken übertreffen können, ist dafür eine hervorragende Ausgangssituation geschaffen.

Literaturverzeichnis

- [ABCC98] D. Applegate, R. Bixby, V. Chvátal und W. Cook. On the solution of traveling salesman problems. *Documenta Mathematica*, Extra Volume ICM 1998(III):645–656, 1998. 48, 63
- [ACR00] D. Applegate, W. Cook und A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. Technical report, Rice University Houston, 2000. 3, 63
- [AFG97] N. Ascheuer, M. Fischetti und M. Grötschel. Solving ATSP with time windows by branch-and-cut. Technical report, ZIB Berlin, 1997. 165, 173, 180
- [AL97] E. Aarts und J.K. Lenstra, Hrsg. *Local search in combinatorial optimization*. Wiley, Chichester, 1997. 26, 27, 30, 34, 35, 228
- [AM94] S. Anily und G. Mosheiov. The traveling salesman problem with delivery and backhauls. *Operations Research Letters*, 16:11–18, 1994. 21
- [AMO93] R.K. Ahuja, T.L. Magnanti und J.B. Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, N.J., 1993. 140, 143, 144, 145
- [AOS01] R.K. Ahuja, J.B. Orlin und D. Sharma. Multi-Exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91:71–97, 2001. 5, 127, 141, 143, 145, 195, 217
- [BB99] R. Battiti und A. Bertossi. Greedy, prohibition, and reactive heuristics for graph-partitioning. *IEEE Transactions on Computers*, 48(4):361–385, 1999. 70

- [BCK01] E.K. Burke, P.I. Cowling und R. Keuthen. Effective Local and Guided Variable Neighbourhood Search Methods for the Asymmetric Traveling Salesman Problem. In E.J.W. Boers et al., Hrsg., *EvoWorkshop 2001*, Seiten 203–212. Springer, 2001. 38
- [BG81] L. Bodin und B. Golden. Classification in Vehicle Routing and Scheduling. *Networks*, 11:97–108, 1981. 7
- [BGAB83] L. Bodin, B. Golden, A. Assad und M. Ball. Routing and Scheduling of Vehicles and Crews: The State of the Art. *Computers and Operations Research*, 10:163–211, 1983. 1
- [BGG⁺95] P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin und É.D. Taillard. A parallel tabu search heuristic for the vehicle routing problem with time windows. Technical Report CRT-95-84, Centre de recherche sur les transports, Université de Montréal, 1995. 105, 155
- [BM97] J. Brandão und A. Mercer. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research*, 100:180–191, 1997. 21
- [BP97] R. Battiti und M. Protasi. Reactive search, a history-based heuristic for MAX-SAT. *ACM Journal of Experimental Algorithmics*, 2, 1997. 70
- [BR01] M.P. Bastos und C.C. Ribeiro. Reactive tabu search with path-relinking for the Steiner Problem in graphs. In Ribeiro und Hansen [RH01], Seiten 39–58. 70
- [BS01] E. Balas und N. Simonetti. Linear Time Dynamic Programming Algorithms for new Classes of Restricted TSPs: A Computational Study. *Inform's Journal on Computing*, 13(1):56–75, 2001. 121, 122, 165, 173, 209
- [BT94] R. Battiti und G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994. 69
- [Cal00] R. Wolfler Calvo. A New Heuristic for the Traveling Salesman Problem with Time Windows. *Transportation Science*, 34(1):113–124, Februar 2000. 57

- [CDM91] A. Colorni, M. Dorigo und V. Maniezzo. Distributed Optimization by Ant Colonies. In *Proceedings of the 1991 European Conference on Artificial Life*, Seiten 134–142. Elsevier, 1991. 67
- [CE69] N. Christofides und S. Eilon. An algorithm for the vehicle-dispatching problem. *Operational Research Quarterly*, 20(3):309–318, 1969. 102
- [CGW99] S.P. Coy, B.L. Golden und E.A. Wasil. A computational study of smoothing heuristics for the traveling salesman problem. *European Journal of Operational Research*, 124:15–27, 1999. 60
- [CH01a] I. Charon und O. Hudry. The Noising Methods: A generalization of some metaheuristics. *European Journal of Operational Research*, 135:86–101, 2001. 57, 59
- [CH01b] I. Charon und O. Hudry. The Noising Methods: A Survey. In Ribeiro und Hansen [RH01], Seiten 245–262. 54, 57
- [CL99] Y. Caseau und F. Laburthe. Heuristics for large constrained vehicle routing problems. *Journal of Heuristics*, 5:281–303, 1999. 105, 138
- [CMT79] N. Christofides, A. Mingozzi und P. Toth. The Vehicle Routing Problem. In N. Christofides, A. Mingozzi, P. Toth und C. Sandi, Hrsg., *Combinatorial Optimization*, Kapitel 11, Seiten 315–338. Wiley, Chichester, 1979. 166, 181
- [CR96] W.-C. Chiang und R.A. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Transportation Science*, 25:19–45, 1996. 35, 190
- [CR97] W.-C. Chiang und R.A. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9(4):417–430, 1997. 70, 148
- [CW97] R. Cordone und R. Wolfer Calvo. Note on time window constraints in routing problems. Technical report, Dipartimento di Elettronica e Informazione, Politecnico de Milano, 1997. 159

- [CW98] R. Cordone und R. Wolfer Calvo. A heuristic for the vehicle routing problem with time windows. Technical report, Dipartimento di Elettronica e Informazione, Politecnico de Milano, 1998. 57, 190
- [CW01] R. Cordone und R. Wolfer Calvo. A Heuristic for the Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, 7:107–129, 2001. 57, 105, 148, 157, 205
- [DDI⁺98] G. Desaulniers, J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis und D. Villeneuve. A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems. In T.G. Crainic und G. Laporte, Hrsg., *Fleetmanagement and Logistics*, Kapitel 3, Seiten 57–91. Kluwer, Dordrecht, 1998. 7, 8, 10, 11, 12, 15, 21, 22, 140
- [DDSS95] J. Desrosiers, Y. Dumas, M.M. Solomon und F. Soumis. Time Constrained Routing and Scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma und G.L. Nemhauser, Hrsg., *Handbooks in Operations Research and Management Science, Vol. 8, Network Routing*, Kapitel 2, Seiten 35–139. Elsevier, Amsterdam, 1995. 7
- [De 75] K.A. De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. Dissertation, University of Michigan, USA, Michigan, 1975. 64
- [Det01] J. Dethloff. Vehicle routing and reverse logistics: The vehicle routing problem with simultaneous delivery and pick-up. *OR Spektrum*, 23:79–96, 2001. 21, 22
- [dFS⁺00] B. de Backer, V. Furnon, P. Shaw, P. Kilby und P. Prosser. Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics. *Journal of Heuristics*, 6:501–523, 2000. 4, 105
- [DIZ01] *Deutschland in Zahlen 2001*. Institut der deutschen Wirtschaft Köln, 2001. 1
- [DLS90] M. Desrochers, J.K. Lenstra und M.W. Savelsbergh. A classification scheme for vehicle routing and scheduling problems.

- European Journal of Operational Research*, 46:322–332, 1990. 5, 7, 8
- [Don02] J.J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Technical Report CS-89-85, Computer Science Departement, University of Tennessee., 2002. 210
- [DS] Marco Dorigo und Thomas Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. 158
- [DS88a] M. Desrochers und F. Soumis. A generalized permanent labeling algorithm for the shortest path problem with time windows. *INFOR*, 26(3):191–212, 1988. 11, 143
- [DS88b] M. Desrochers und F. Soumis. A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, 35:242–254, 1988. 11
- [FR01] P. Festa und M.G.C. Resende. GRASP: An Annotated Bibliography. In Ribeiro und Hansen [RH01], Seiten 325–367. 62
- [Ful72] D.R. Fulkerson. Flow Networks and Combinatorial Operations Research. In A.M. Geoffrion, Hrsg., *Perspectives on Optimization*, Seiten 139–171. Addison-Wesley, 1972. 10
- [Fun98] B. Funke. Verbesserungsheuristiken für das Vehicle Routing Problem mit Zeitfenstern und heterogenem Fuhrpark. Diplomarbeit, RWTH Aachen, Aachen, Sep 1998. 57
- [GDDS95] S. Gelinas, M. Desrochers, J. Desrosiers und M.M. Solomon. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61:91–109, 1995. 21
- [GH94] J. Gu und X. Huang. Efficient local search with search space smoothing: A case study of the traveling salesman problem. *IE-EE Transactions on Systems, Man, and Cybernetics*, 24(5):728–735, 1994. 59

- [GHL92] M. Gendreau, A. Hertz und G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992. 50
- [GHL94] M. Gendreau, A. Hertz und G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290, 1994. 167
- [GHL96] M. Gendreau, A. Hertz und G. Laporte. The traveling salesman problem with backhauls. *Computers Ops Res.*, 23(5):501–508, 1996. 21
- [GHLS95] M. Gendreau, A. Hertz, G. Laporte und M. Stan. A generalized insertion heuristics for the traveling salesman problem with time windows. Technical report, Centre de recherche sur les transports, Université de Montréal, Canada, 1995. 51, 117
- [Gie94] M. Gietz. *Computergestützte Tourenplanung mit zeitkritischen Restriktionen*. Physica-Verlag, 1994. 32
- [GJ79] M.R. Garey und D.S. Johnson. *Computers and intractability*. Freeman, New York, 1979. 25
- [GJ89] M. Goetschalckx und C. Jacobs-Blecha. The vehicle routing problem with backhauls. *European Journal of Operational Research*, 42:39–51, 1989. 21
- [GL97] F. Glover und M. Laguna. *Tabu search*. Kluwer, Dordrecht, 1997. 53, 54, 56, 68, 69
- [GLH97] M. Gendreau, G. Laporte und A. Hertz. An approximation algorithm for the traveling salesman problem with backhauls. *Operations Research*, 45(4):639–641, 1997. 21
- [GLMT99] M. Gendreau, G. Laporte, C. Musaraganyi und É.D. Taillard. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 26:1153–1173, 1999. 52
- [Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5:533–549, 1986. 68

- [Glo89] F. Glover. Tabu Search – Part I. *Operations Research Society of America*, 1:190–206, 1989. 68
- [Glo92a] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. Technical report, University of Boulder, School of Business, Boulder, Apr 1992. 36, 45
- [Glo92b] F. Glover. New ejection chain and alternating path methods for traveling salesman problems. *Computer Science and Operations Research*, Seiten 449–509, 1992. 36
- [Glo98] F. Glover. A Template for Scatter Search and Path Relinking. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer und D. Snyers, Hrsg., *Artificial Evolution, Lecture Notes in Computer Science 1363*, Seiten 13–54. Springer, 1998. 66
- [Glo99] F. Glover. Scatter search and path relinking. In D. Corne, M. Dorigo und F. Glover, Hrsg., *New Ideas in Optimization*, Seiten 297–316. McGraw Hill, 1999. 66
- [Gol89] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989. 64
- [GP02] Gregory Gutin und Abraham P. Punnnen, Hrsg. *The Travelling Salesman Problem and Its Variations*. Kluwer, 2002. 1
- [GTA99] L.M. Gambardella, É.D. Taillard und G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. Technical report, IDSIA, Lugano, Switzerland, 1999. 105, 148, 158
- [GWKC98] B.L. Golden, E.A. Wasil, J.P. Kelly und I-M. Chao. Metaheuristics in vehicle routing. In T.G. Crainic und G. Laporte, Hrsg., *Fleet Management and Logistics*. Kluwer, Boston, 1998. 167
- [HCM95] E. Hadjiconstantinou, N. Christophides und A. Mingozzi. A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest path relaxations. *Annals of Operations Research*, 61:21–43, 1995. 167

- [HG98] J. Homberger und H. Gehring. Ein Simulated-Annealing-Verfahren für das Standardproblem der Tourenplanung mit Zeitfensterrestriktionen. Technical report, FernUniversität Hagen, 1998. 105, 148
- [HM97] P. Hansen und N. Mladenović. Variable neighborhood search. *Computers & Operations Research*, 24:1097–1100, 1997. 56
- [HM01a] P. Hansen und N. Mladenović. Developements of Variable Neighborhood Search. In Ribeiro und Hansen [RH01], Seiten 415–440. 56
- [HM01b] P. Hansen und N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001. 55, 56
- [Hol75] J.H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975. 64
- [HRM97] I. Hong, A. B. Kahng R. und B. Moon. Improved Large-Step Markov Chain Variants for the Symmetric TSP. *Journal of Heuristics*, 3:63–81, 1997. 48, 63
- [IMKY02] T. Ibaraki, T. Uno M. Kubo, T. Masuda und M. Yagiura. Effective Local Search Algorithms for the Vehicle Routing Problem with General Time Window Constraints. url, 2002. 105
- [Irn02] S. Irnich. Iterated Local Search, 2002. 168
- [JM97] D.S. Johnson und L.A. McGeoch. The travelling salesman problem: A case study in local optimization. In Aarts und Lenstra [AL97], Seiten 215–310. 48
- [Joh90] D.S. Johnson. Local optimization and the traveling salesman problem. In *Proceedings of the 17th colloquium on Automata, Languages and Programming*, 1990. 63
- [KB95] G. Kontoravdis und J.F. Bard. A GRASP for the vehicle routing problem with time windows. *Operations Research Society of America*, 7(1):10–23, 1995. 62, 138, 148

- [KGV83] S. Kirkpatrick, C.D. Gelatt, Jr. und M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 58
- [KL] R. Kumar und H. Li. On Asymmetric TSP: Transformation to Symmetric TSP and Performance Bound. <http://www.engr.uky.edu/~kumar/jpub.html> (2.8.02). 75
- [KL70] B.W. Kernighan und S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49:291–308, 1970. 57
- [KPS92] Y.A. Koskosidis, W.B. Powell und M.M. Solomon. An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation Science*, 26(2):69–85, 1992. 155
- [KS92] G.A.P. Kindervater und M.W.P. Savelsbergh. Local Search in Physical Distribution Management. Technical Report EUR-CS-92-05, Erasmus University, Rotterdam, 1992. 23, 94
- [KS97] G.A.P. Kindervater und M.W.P. Savelsbergh. Vehicle routing: handling edge exchanges. In E. Aarts und J.K. Lenstra, Hrsg., *Local Search in Combinatorial Optimization*, Kapitel 10, Seiten 337–360. John Wiley & Sons Ltd., Chichester, 1997. 32, 94, 96
- [LGPS00] G. Laporte, M. Gendreau, J.-Y. Potvin und F. Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 7:285–300, 2000. 117, 167, 174, 176
- [Lin65] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965. 39, 61
- [LK73] S. Lin und B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973. 45, 48, 57, 61, 115, 142
- [LMS] H.R. Lorenço, O. Martin und T. Stützle. Iterated Local Search. [url: citeseer.nj.nec.com/420596.html](http://citeseer.nj.nec.com/420596.html) (1.9.2002). 54, 62, 63
- [LNT88] G. Laporte, Y. Nobert und S. Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, 22(3):161–172, 1988. 105

- [LS99] F.-H.F. Liu und S.-Y. Shen. A route-neighborhood-based metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 118:485–504, 1999. 105, 148
- [LT98] T.L. Lau und E.P.K. Tsang. The guided genetic algorithm and its application to the general assignment problem. In *IEEE, 10th International Conference on Tools with Artificial Intelligence (ICTAI'98), Taiwan*, 1998. 61
- [MC01] V. Maniezzo und A. Carbonaro. Ant Colony Optimization: A Overview. In Ribeiro und Hansen [RH01], Seiten 469–492. 67, 68
- [MCS92] H. Min, J. Current und D. Schilling. The multiple depot vehicle routing problem with backhauling. *Journal of Business Logistics*, 13(1):259–288, 1992. 21
- [MGB99] A. Mingozzi, S. Giorgi und R. Baldacci. An exact method for the vehicle routing problem with backhauls. *Transportation Science*, 33(3):315–329, 1999. 21
- [Min89] H. Min. The Multiple Vehicle Routing Problem with Simultaneous Delivery and Pickup Points. *Transportation Research*, 23A:377–386, 1989. 21
- [MO96] O. Martin und S.W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996. 63
- [MOF91] O. Martin, S.W. Otto und E.W. Felten. Large-step Markov chains for the Traveling Salesman Problem. *Complex Systems*, 5(3):299–326, 1991. 63
- [MOF92] O. Martin, S.W. Otto und E.W. Felten. Large-step Markov chains for the TSP incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224, 1992. 48
- [MRR⁺53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A Teller und M. Teller. Equation of state calculations for fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953. 58

- [NB00] W.P. Nanry und J.W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B*, 34:107–121, 2000. 70
- [Net99] D.M. Neto. *Efficient cluster compensation for Lin-Kernighan Heuristics*. Dissertation, Department fo Computer Science, University of Toronto, 1999. 48
- [OL96] I.H. Osman und G. Laporte. Metaheuristics: A bibliography. *Annals of Operational Research*, 63:513–628, 1996. 53
- [Or76] I. Or. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking*. Dissertation, Northwestern University, Evanston, Ill., 1976. 48, 117
- [OS96] I.H. Osman und S. Salhi. Local search strategies for the vehicle fleet mix problem. In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves und G.D. Smith, Hrsg., *Modern Heuristic Search Methods*, Kapitel 8, Seiten 131–153. Wiley, Chichester, 1996. 35
- [Osm93] I.H. Osman. Metastrategy Simulated Annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993. 35
- [OVD98] L.S. Ochi, D.S. Vianna und L.M.A. Drummond. A Parallel Genetic Algorithm for the Vehicle Routing Problem with Heterogeneous Fleet. In *Lecture Notes in Computer Science*, Seiten 187–195. Springer, 1998. 52
- [Pap80] U. Pape. Algorithm 562: Shortest Path Length. *ACM Transactions on Mathematical Software*, 6:450–455, 1980. 143
- [PB96] J.Y. Potvin und S. Bengio. The Vehicle Routing Problem with Time Windows–Part II: Genetic Search. *INFORMS Journal on Computing*, 8:165–172, 1996. 190
- [PG99] G. Pesant und M. Gendreau. A Constraint Programming Framework for Local Search Methods. *Journal of Heuristics*, 5:255–279, 1999. 4, 52, 105

- [PGR97] G. Pesant, M. Gendreau und J.-M. Rousseau. GENIUS-CP: A Generic Vehicle Routing Algorithm. In *Principles and Practice of Constraint Programming–CP97: Proceedings of the Third International Conference, LNCS 1330*, Seiten 420–434. Springer, 1997. 51, 117
- [PLR89] J.-Y. Potvin, G. Lapalme und J.-M. Rousseau. A generalized k-opt exchange procedure for the MTSP. *INFOR*, 27(4):474–481, 1989. 49
- [PR95] J.Y. Potvin und J.M. Rousseau. An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446, 1995. 55, 105
- [Rec73] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973. 64
- [Ree93] C.R. Reeves. *Modern heuristic technique for combinatorial problems*. Blackwell, Oxford, 1993. 53
- [Reg96] C. Rego. Relaxed tours and path ejections for the traveling salesman problem. Technical report, Universidade Portucalense, Departamento de Informática, Porto, 1996. 37, 115
- [Reg98] C. Rego. A subpath ejection method for the vehicle routing problem. *Management Science*, 44(10):1447–1459, 1998. 37, 117
- [Reg01] César Rego. Node-ejection chains for the vehicle routing problem: Sequential and parallel algorithms. *Parallel Computing*, 27(3):201–222, 2001. 36
- [RGP02a] L.-M. Rousseau, M. Gendreau und G. Pesant. Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, 8:43–58, 2002. 52, 114
- [RGP02b] L.-M. Rousseau, M. Gendreau und G. Pesant. Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows. *Journal of Heuristics*, 8:43–58, 2002. 160

- [RH01] C.C. Ribeiro und P. Hansen, Hrsg. *Essays and surveys in meta-heuristics*. Kluwer Academic Publishers, Dordrecht, 2001. 53, 222, 223, 225, 228, 230
- [RR96] C. Rego und C. Roucairol. A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In I.H. Osman und J.P. Kelly, Hrsg., *Meta-Heuristics: Theory & Application*, Kapitel 40, Seiten 661–675. Kluwer Academic, Boston, 1996. 167
- [RS94] Y. Rochat und F. Semet. A Tabu Search Approach for Delivering Pet Food and Flour in Switzerland. *Journal of Operational Research Society*, 45:1233–1246, 1994. 190
- [RT95] Y. Rochat und É.D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1:147–167, 1995. 52, 148, 167
- [Rus95a] R. Russel. Hybrid Heuristics for the Vehicle Routing Problem with Time Windows. *Transportations Science*, 29(2), 1995. 138
- [Rus95b] R.A. Russell. Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science*, 29(2):156–166, 1995. 36
- [Saa97] Y. Saab. Combinatorial Optimization by Dynamic Contraction. *Journal of Heuristics*, 3:207–224, 1997. 160
- [Sav85] M.W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4(6):285–305, 1985. 5, 92, 216
- [Sav90] M.W.P. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operational Research*, 47:75–85, 1990. 43
- [Sch81] H.P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, NewYork, 1981. 64
- [SDFM97] J. Schneider, M. Dankesreiter, W. Fettes und I. Morgenstern. Search-space smoothing for combinatorial optimization problems. *Physica A, Statistical and Theoretical Physics*, 243(1/2):77–112, 1997. 59

- [SF99] J. Schulze und T. Fahle. A parallel algorithm for the vehicle routing problem with time window constraints. *Annals of Operations Research*, 86:585–607, 1999. 148
- [Sha98] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. Technical report, Department of Computer Science, University of Strathclyde, Glasgow, 1998. 4, 52
- [Sol87] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987. 148, 166, 168, 173
- [Tai93] É.D. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–676, 1993. 33, 167
- [Tai99a] É.D. Taillard. Ant systems. Technical report, IDSIA, Lugano, Switzerland, 1999. 68
- [Tai99b] É.D. Taillard. A heuristic column generation method for the heterogeneous fleet vrp. *Recherche operationnelle/Operations Research*, 33(1):1–14, 1999. 148
- [TBG⁺97] É.D. Taillard, P. Badeau, M. Gendreau, F. Guertin und J.-Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31:170–186, 1997. 106, 155
- [TGGP01] É.D. Taillard, L.M. Gambardella, M. Gendreau und J.-Y. Potvin. Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135:1–16, 2001. 53, 54
- [Tha93] S.R. Thangiah. Vehicle routing with time windows using genetic algorithms. Technical Report SRU-CpSc-TR-93-23, Slippery Rock University, 1993. 35
- [TOS95] S.R. Thangiah, I.H. Osman und T. Sun. Metaheuristics for vehicle routing problems with time windows. Technical report, Institute of Mathematics and Statistics, University of Kent, Canterbury, 1995. 190

- [TOVS93] S.R. Thangiah, I.H. Osman, R. Vinayagamoorthy und T. Sun. Algorithms for the vehicle routing problems with time deadlines. *American Journal of Mathematical and Management Science*, 13(3& 4):323–355, 1993. 35
- [TP93] P.M. Thompson und H.N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993. 5, 37, 123, 217
- [TV98] P. Toth und D. Vigo. Granular tabu search. Technical report, DEIS, University of Bologna, 1998. 167
- [TV99] P. Toth und D. Vigo. A heuristic algorithm for the symmetric and asymmetric vehicle routing problems with backhauls. *European Journal of Operational Research*, 113:528–543, 1999. 21, 22
- [TV02] P. Toth und D. Vigo. The Granular Tabu Search and its Application to the Vehicle-Routing Problem. *erscheint in INFORMS Journal on Computing*, 2002. 117, 167
- [VAL98] R.J.M. Vaessens, E.H.L. Aarts und J.K. Lenstra. A local search template. *Computers and Operations Research*, 25(11):969–979, 1998. 55
- [vB01] A. van Breedam. Comparing descent heuristics and metaheuristics for the vehicle routing problem. *Computers & Operations Research*, 28:289–315, 2001. 34, 35
- [VMOR99] S. Voss, S. Martello, I.H. Osman und C. Roucairol. *Metaheuristics: Advances and trends in local search paradigms for optimization*. Kluwer Academic, Boston, 1999. 53
- [Vou97] C. Voudouris. *Guided Local Search for Combinatorial Optimization Problems*. Dissertation, Department of Computer Science, University of Essex, 1997. 61
- [VT95] C. Voudouris und E. Tsang. Guided Local Search. Technical report, Department of Computer Science, University of Essex, 1995. 60

- [VT98] C. Voudouris und E. Tsang. Guided Local Search Joins the Elite in Discrete Optimization. In *DIMACS Workshop on Constraint Programming and Large Scale Discrete Optimisation*. Rutgers, 1998. 61
- [Wal00] C. Walshaw. A Multilevel Approach to the Travelling Salesman Problem. Technical Report 00/IM/63, Computing and Mathematical Sciences, University of Greenwich, London, 2000. 160
- [Wal01] C. Walshaw. Multilevel Refinement for Combinatorial Optimisation Problems. Technical Report 01/IM/73, Computing and Mathematical Sciences, University of Greenwich, London, 2001. 160
- [XK96] J. Xu und J.P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30:379–393, 1996. 167
- [YI] M. Yagiura und T. Ibaraki. On Metaheuristic Algorithms for Combinatorial Optimization Problems. url: citeseer.nj.nec.com/412345.html (1.9.2002). 53, 54

Anhang A

Herleitung der Verbindungstheoreme

Wir geben eine Herleitung der Verbindungstheoreme von Savelsbergh (siehe Abschnitt 3.2.1 auf Seite 93) für das TSPTW an. Diese Verbindungstheoreme beschreiben, wie die Werte von D , LAT und EDT für ein Segment σ bestimmt werden können, welches durch die Verkettung zweier Segmente entstanden ist. Dabei sind

- D_σ – die Gesamtfahrzeit einschließlich der Servicezeiten (ohne Wartezeiten),
- LAT_σ – die späteste Ankunftszeit am ersten Knoten des Segments und
- EDT_σ – die früheste Abfahrtszeit am letzten Knoten des Segments.

Anschließend wird untersucht, wie sich die Berechnungen ändern, wenn das „Warten“ vor einem Knoten nicht erlaubt sein sollte.

„Warten“ ist erlaubt Es seien σ_1 und σ_2 zwei verschiedene Segmente einer zulässigen Lösung y eines TSPTW, deren Werte D , LAT und EDT bereits bekannt sind. Das Segment σ entsteht durch die Verkettung von σ_1 und σ_2 : $\sigma = \sigma_1 \oplus \sigma_2$. Die Knoten a_1, e_1 bzw. a_2, e_2 seien die Anfangs- und Endknoten der Segmente σ_1 bzw. σ_2 , und für die Fahrzeit zwischen den Segmenten gilt: $d_{\sigma_1\sigma_2} = d_{e_1a_2}$.

Nachfolgend wird die Bezeichnung s_σ für die Verweildauer innerhalb eines Segments σ verwendet. s_σ ist also die Zeit, die für das Durchlaufen von σ aufgewendet werden muß. Da hierbei (im Gegensatz zu D_σ) auch die Wartezeiten berücksichtigt werden, ist s_σ von der Ankunftszeit T am Beginn des

Segments abhängig:

$$s_\sigma = s_\sigma(T) = \max [D_\sigma, EDT_\sigma - T] .$$

Der Wert von D_σ ergibt sich aus der Definition, so daß sich eine Herleitung erübrigt. Beim Verbinden der Segmente σ_1 und σ_2 muß die Fahrtdauer für den zusätzlichen Bogen (e_1, a_2) berücksichtigt werden:

$$\begin{aligned} D_\sigma &= D_{\sigma_1} + d_{\sigma_1\sigma_2} + D_{\sigma_2} \\ &= D_{\sigma_1} + d_{e_1a_2} + D_{\sigma_2} \end{aligned}$$

Ausgangspunkt für die Bestimmung der spätesten Ankunftszeit LAT_σ am Segment σ ist die Feststellung, daß die Ankunft an σ_1 mit der Ankunft an σ zusammenfällt. Nach dem Durchlaufen von σ_1 muß das Segment σ_2 aber spätestens zur Zeit LAT_{σ_2} erreicht werden, damit die Zulässigkeit erhalten bleibt:

$$\begin{aligned} LAT_\sigma &= \min \left[LAT_{\sigma_1}, \max_{EAT_{\sigma_1} \leq T_{\sigma_1} \leq LAT_{\sigma_1}} \{LAT_{\sigma_2} - d_{\sigma_1\sigma_2} - s_{\sigma_1}(T_{\sigma_1})\} \right] \\ &= \min \left[LAT_{\sigma_1}, LAT_{\sigma_2} - d_{\sigma_1\sigma_2} \right. \\ &\quad \left. - \max \left[D_{\sigma_1}, \max_{EAT_{\sigma_1} \leq T_{\sigma_1} \leq LAT_{\sigma_1}} \{EDT_{\sigma_1} - s_{\sigma_1}(T_{\sigma_1})\} \right] \right] \\ &= \min \left[LAT_{\sigma_1}, LAT_{\sigma_2} - d_{\sigma_1\sigma_2} - \underbrace{\max(D_{\sigma_1}, EDT_{\sigma_1} - LAT_{\sigma_1})}_{s_{\sigma_1}^{\min}} \right] \end{aligned}$$

$s_{\sigma_1}^{\min}$ ist die minimale Zeit, die für das Durchlaufen des Segments σ_1 aufgewendet werden muß. Diese Zeit kann auch als die Servicezeit des Segments σ_1 aufgefaßt werden. Damit das Segment σ zulässig bleiben kann, muß außerdem $EDT_{\sigma_1} + d_{\sigma_1\sigma_2} \leq LAT_{\sigma_2}$ gelten. Daraus folgt schließlich:

$$LAT_\sigma = \min [LAT_{\sigma_1}, LAT_{\sigma_2} - d_{\sigma_1\sigma_2} - D_{\sigma_1}] .$$

Für die Bestimmung des Wertes von EDT_σ kann genutzt werden, daß die früheste Abfahrt von Segment σ mit der frühesten Abfahrt von Segment σ_2 übereinstimmen muß. Nach dem Durchlaufen von σ_1 kann jedoch frühestens

zum Zeitpunkt EDT_{σ_1} zum Segment σ_2 abgefahren werden:

$$\begin{aligned}
EDT_{\sigma} &= \max [EDT_{\sigma_1} + d_{\sigma_1\sigma_2} + s_{\sigma_2}(EDT_{\sigma_1} + d_{\sigma_1\sigma_2}), EDT_{\sigma_2}] \\
&= \max [EDT_{\sigma_1} + d_{\sigma_1\sigma_2} \\
&\quad + \max[D_{\sigma_2}, EDT_{\sigma_2} - (EDT_{\sigma_1} + d_{\sigma_1\sigma_2})], EDT_{\sigma_2}] \\
&= \max [EDT_{\sigma_1} + \max[d_{\sigma_1\sigma_2} + D_{\sigma_2}, EDT_{\sigma_2} - EDT_{\sigma_1}], EDT_{\sigma_2}] \\
&= \max [EDT_{\sigma_1} + d_{\sigma_1\sigma_2} + D_{\sigma_2}, EDT_{\sigma_2}].
\end{aligned}$$

„Warten“ ist nicht erlaubt Wenn das Warten vor einem Knoten nicht erlaubt ist, fällt der Unterschied zwischen s_{σ} und D_{σ} weg:

$$s_{\sigma} = D_{\sigma}.$$

Dies bedeutet insbesondere, daß die Verweilzeit in einem Segment auch nicht mehr von der Ankunftszeit abhängig ist. Dementsprechend ist die Differenz zwischen Ankunfts- und Abfahrtszeit an einem Segment fix, was sich auch auf die Zulässigkeitsintervalle überträgt:

$$\begin{aligned}
EAT_{\sigma} + D_{\sigma} &= EDT_{\sigma}, \\
LAT_{\sigma} + D_{\sigma} &= LDT_{\sigma}
\end{aligned}$$

Daher genügt es, D_{σ} , LAT_{σ} und EDT_{σ} zu kennen, um die übrigen Werte zu bestimmen.

In die Bestimmung dieser drei Werte ging die tatsächliche Verweilzeit s_{σ} im Segment σ aber ohnehin nicht ein, so daß sich daran nichts ändert. Für die Feststellung der Zulässigkeit einer Verkettung ist also lediglich die Einhaltung der beiden Restriktionen

$$EDT_{\sigma_1} + d_{\sigma_1\sigma_2} \leq LAT_{\sigma_2}$$

und

$$\begin{aligned}
LDT_{\sigma_1} + d_{\sigma_1\sigma_2} &\geq EAT_{\sigma_2} \\
LAT_{\sigma_1} + D_{\sigma_1} + d_{\sigma_1\sigma_2} &\geq EDT_{\sigma_2} - D_{\sigma_2} \\
LAT_{\sigma_1} + \underbrace{D_{\sigma_1} + d_{\sigma_1\sigma_2} + D_{\sigma_2}}_{D_{\sigma}} &\geq EDT_{\sigma_2} \\
LAT_{\sigma_1} + D_{\sigma} &\geq EDT_{\sigma_2}
\end{aligned}$$

zu überprüfen.

Lebenslauf

Persönliche Daten

Birger Funke
geboren am 18. Juli 1966 in Bad Dübén bei Leipzig

Ausbildung

- | | |
|-------------|--|
| 1973 – 1983 | 63. Oberschule in Leipzig
Abschluß: mittlere Reife |
| 1983 – 1984 | „Vorkurs“ an der Universität Leipzig
Abschluß: Hochschulreife |
| 1984 – 1989 | Studium an der Universität Leipzig,
Abschluß: Diplomlehrer für Mathematik und Physik |
| 1991 – 1992 | Forschungsstudium an der TH Leipzig
Angewandte Informatik |
| 1995 – 1998 | Zusatzstudium Operations Research an der RWTH Aachen
Abschluß: Magister des Operations Research |

Beschäftigungen

- | | |
|-------------|--|
| 1989 – 1990 | 71. Oberschule in Leipzig
Lehrer für Mathematik und Physik |
| 1992 – 1995 | TH Leipzig / ELITE-Stiftung Aachen
wissenschaftl. Mitarbeiter im BMFT-Forschungsprojekt
PROKON |
| 1996 – 1999 | ELITE-Stiftung in Aachen
Mitarbeit bei der Entwicklung eines Planungssystems für
die Brieftransporte der Deutschen Post AG |
| seit 1999 | RWTH Aachen, Lehr- und Forschungsgebiet
Operations Research und Logistik Management
wissenschaftl. Mitarbeiter |

