



TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Informatik

CSR-06-04

Probleme der Tourenbildung

Michael Kämpf

Mai 2006

Chemnitzer Informatik-Berichte

Inhaltsverzeichnis

1	Einleitung	1
2	Grundbegriffe	5
2.1	Graphentheorie	5
2.2	Transportoptimierung	7
3	Probleme bei der Tourenbildung	11
3.1	Das Traveling Salesman Problem (TSP)	11
3.2	Das klassische Vehicle Routing Problem (VRP)	12
3.3	Erweiterungen des klassischen VRP	14
3.3.1	Das Multiple Traveling Salesman Problem (m-TSP)	14
3.3.2	Berücksichtigung von Zeitfenstern	15
3.3.3	Mehrere Lager	16
3.3.4	Einsatz einer heterogenen Fahrzeugflotte	17
3.3.5	Kombination von Be- und Entladung	18
3.3.6	Einbeziehung von Rückgabemöglichkeiten	19
4	Lösungsverfahren für das TSP und VRP	23
4.1	Exakte Verfahren	23
4.1.1	Suchbaumstrategien	25
4.1.2	Probleme und Lösungsansätze	28
4.2	Heuristische Verfahren (Heuristiken)	35
4.2.1	Eröffnungsverfahren	35
4.2.2	Verbesserungsverfahren	38
4.2.3	Kombination heuristischer Verfahren	41
4.2.4	Moderne heuristische Verfahren	41
5	Zusammenfassung und Ausblick	45
A	Algorithmen	47
A.1	Exakte Verfahren	47
A.1.1	Branch-and-Bound-Verfahren	49
A.1.2	Branch-and-Cut-Verfahren	50
A.2	Heuristische Verfahren	52
A.2.1	Eröffnungsverfahren für das TSP und VRP	52
A.2.2	Eröffnungsverfahren für das VRP	58

A.2.3	Verbesserungsverfahren für das TSP und VRP	61
A.3	Moderne heuristische Verfahren	64
B	Übersetzungen	67
	Sachregister	71
	Literaturverzeichnis	75

Abbildungsverzeichnis

2.1	Kantentypen	6
2.2	Wege und Kreise	6
2.3	Hamiltonsche Kreise	6
2.4	Fluss zwischen Knoten	7
2.5	Minimalgerüst, Eulergraph und Matching	7
2.6	Transportwege beim Routing Problem	8
3.1	Ein Traveling Salesman Problem	11
3.2	Ein Vehicle Routing Problem	13
3.3	Ein Multiple Traveling Salesman Problem	15
3.4	Ein VRP ohne und mit Zeitfenstern	16
3.5	Ein Multidepot Vehicle Routing Problem	16
3.6	Ein Pickup and Delivery Problem	18
3.7	Ein VRP mit Rückgabemöglichkeiten	20
4.1	Kombinatorische Explosion eines Suchbaums	26
4.2	Exakte Suchverfahren	28
4.3	Christofides-Heuristik	35
4.4	Auswahl heuristischer Tourenbildungsverfahren	37
4.5	Die Verfahren 2-opt und 3-opt	39
4.6	Einige Schritte des Lin-Kernighan-Algorithmuses	40
4.7	Or-opt bei $s = 3$	41
5.1	Überblick über die Probleme bei der Tourenbildung	45
5.2	Überblick über die Lösungsverfahren beim TSP und VRP	46

Tabellenverzeichnis

B.1	deutschsprachige Übersetzungen für englische Fachbegriffe (Teil 1)	67
B.2	deutschsprachige Übersetzungen für englische Fachbegriffe (Teil 2)	68
B.3	deutschsprachige Übersetzungen für englische Fachbegriffe (Teil 3)	69

Algorithmenverzeichnis

A.1	Enumeration (TSP)	48
A.2	Branch-and-Bound (TSP)	49
A.3	Branch-and-Cut (TSP)	51
A.4	Nearest Neighbour (TSP)	53
A.5	Nearest Neighbour (VRP)	53
A.6	Multiple Fragment (TSP)	54
A.7	General Insertion (TSP)	54
A.8	Nearest/Farthest Insertion (TSP)	55
A.9	Cheapest Insertion (TSP)	55
A.10	Random Insertion (TSP)	56
A.11	Smallest/Largest Summation Insertion (TSP)	56
A.12	Double Spanning Tree (TSP)	57
A.13	Christofides-Heuristik (TSP)	57
A.14	Simultaner Saving-Algorithmus (nach Clarke und Wright)	58
A.15	Sequentieller Saving-Algorithmus (nach Clarke und Wright)	59
A.16	Sweep-Algorithmus (nach Gillett und Miller)	60
A.17	Clustering-Algorithmus	60
A.18	r-opt-Verfahren (TSP)	61
A.19	2-opt-Verfahren (TSP)	62
A.20	3-opt-Verfahren (TSP)	62
A.21	variables r-opt-Verfahren (nach Lin und Kernighan) (TSP)	63
A.22	Or-opt-Verfahren (nach Or) (TSP)	64
A.23	Simulated-Annealing-Verfahren	65
A.24	Tabu-Search-Verfahren	65
A.25	Grundprinzip evolutionärer Algorithmen	66

Symbolverzeichnis

0	Zentrales Lager eines Transportgraphen
$\{0_1, 0_2, \dots, 0_k\}$	Menge von Lagern eines Transportgraphen
a_i	Gewicht des Verbraucherkonus i
A	Restriktionsmatrix
A_i	Ankunftszeitpunkt beim Verbraucher i
b	Restiktionsvektor
b	maximales Gewicht eines Verbraucherkonus bei NGAP
bs_{ij}	Backhaul Saving der Kante zwischen Knoten i und j , $s_{ij} \in \mathbb{R}$
c_{ij}	Kosten einer Kante von Knoten i nach Knoten j , $c_{ij} = c(i, j)$
c^T	transponierter Kostenvektor
$c(S_i)$	Kosten einer TSP-Route durch die Verbraucher der Menge S_i
C_{max}	maximale Kapazität der einzelnen Transportfahrzeuge, $C_{max} > 0$
C_{max}^k	maximale Kapazität eines Transportfahrzeuges k , $C_{max}^k > 0$
$C_{max}(v_i)$	maximale Kapazität eines Transportfahrzeuges des Lagers v_i , $C_{max}(v_i) > 0$
d_i	benötigte Menge eines Verbrauchers, $d_i > 0$
d^T	transponierter Normalenvektor einer Hyperebene
D_i	Abfahrtszeitpunkt vom Verbraucher i
D_P	Menge von Probleminstanzen
D_V	Menge aller Verbraucher des RP, $D_V = \{j_1, j_2, \dots, j_n\}$
e_i	frühestmöglicher Belieferungszeitpunkt von Verbraucher i
e_i^+	frühestmöglicher Belieferungszeitpunkt von Verbraucher i beim PDP
e_i^-	frühestmöglicher Belieferungszeitpunkt von Verbraucher i beim PDP
E_G	Kantenmenge eines Graphen, $E_G = \{(i, j)\} = \subseteq V_G \times V_G$
E	Kantenmenge eines Transportgraphen, $E \subseteq V \times V$
E^m	m -dimensionale Kantenmenge, $E^m = (E \times E)^{m-1} \times E$, $m \geq 1$
E_R	Kantenmenge des RP, $E_R = V_R \times V_R$

f	nichtganzzahliger optimaler Lösungswert
$f(x)$	Zielfunktion
$f(y_k)$	Kosten einer optimalen TSP-Route durch die Verbraucher
$f_{LA}(y_k)$	Kosten einer Linearen Approximation
F	Anzahl der noch folgenden Entladepunkte, $F \subseteq V$
$F(i, q)$	untere Schranke für die Lieferung von q Einheiten zum Verbraucher i
$F(S, i, t)$	niedrigste Kosten eines Pfades
G	Graph, $G = (V, E)$
G_R	Graph des RP, $G_R = (V_R, E_R)$
G^k	einzelner Graph aus einer Menge von Graphen, $G^k \in G = \cup_i G^i$
i	Index eines Verbrauchers
i^-	Index eines Verbrauchers mit Beladeforderung
i^+	Index eines Verbrauchers mit Entladeforderung
I	Kantenmenge zwischen den Verbrauchern, $I \subset N \times N$
I	Probleminstanz
K_i	Produzentenmenge des RP
K_P	Menge aller Produzenten des RP, $K_P = \{i_1, i_2, \dots, i_m\}$
l_i	spätmöglicher Belieferungszeitpunkt von Verbraucher i
l_i^+	spätmöglicher Belieferungszeitpunkt von Verbraucher i beim PDP
l_i^-	frühestmöglicher Belieferungszeitpunkt von Verbraucher i beim PDP
l_j	Kundenanzahl für Transportfahrzeug j
$l(i, j)$	Länge eines kürzesten Weges von i nach j
m_P	Abbildungsfunktion einer Lösung $S_P(I)$ der Probleminstanz I in den Bereich der reellen Zahlen, $m_P : S_P \rightarrow \mathbb{R}$
M	Matching
n	Anzahl der Orte (TSP) oder Verbraucher (VRP)
N	Menge der Verbraucher eines Transportgraphen, $N = \{1, 2, \dots, n\}$
N^+	Menge der Beladepunkte des PDP, $N^+ \subset N$
N^-	Menge der Entladepunkte des PDP, $N^- \subset N$
\mathcal{NP} -hart	Klasse von Optimierungsproblemen deren zugehörige Entscheidungsprobleme nur mit einem nichtdeterministischen polynomiellen Algorithmus gelöst werden können

$N(S)$	Nachbarschaft der Lösung S
p_i	abzugebende Menge eines Produzenten des RP, $p_i > 0$
$P(i)$	Menge der Vorgänger eines Knotens i
$r(S)$	untere Schranke für Anzahl der Transportfahrzeuge zur Bedarfsdeckung der Menge S
$\dot{R}(i)$	Menge der von i aus erreichbaren Knoten, $\dot{R}(i) = R(i) \setminus \{i\}$
s_{ij}	Saving der Kante zwischen Knoten i und j , $s_{ij} \in \mathbb{R}$
s	Anzahl benachbarter Knoten bei Or-opt
S	Knotenteilmenge des TSP, $S \subseteq V$
\hat{S}	Schätzwert des maximalen Savings
$S(i)$	Menge der Nachfolger eines Knotens i
$S_P(I)$	Menge von möglichen Lösungen einer Probleminstanz
T	Abkühltemperatur
v	Anzahl der Transportfahrzeuge
v^k	Anzahl der Transportfahrzeuge des Lagers k
$v(i, q, k)$	minimale Kosten von k Touren mit Fracht q zu verschiedenen letzten Verbrauchern aus der Menge $\{1, 2, \dots, i\}$
V	Knotenmenge eines Transportgraphen (Verbraucher inkl. Lager), $V = \{0, 1, \dots, n\} = N \cup 0$
V^k	Knotenmenge eines einzelnen Graphen aus einer Menge von Graphen, $V^k = \{0_k\} \times N$
V_R	Knotenmenge des RP, $V_R = K_P \cup D_V$
V_G	Knotenmenge eines Graphen, $V = \{1, 2, \dots, n\}$
W	Menge der v-tree-Relaxation
W_i	Wartezeitpunkt im Verbraucher i
x	Entscheidungsvektor
x^*	kostenminimale Lösung
x_{ij}	Entscheidungsvariable für die Nutzung der Kante vom Knoten i zum Knoten j , $x_{ij} \in \{0, 1\}$
x_{ij}^k	Entscheidungsvariable für die Nutzung der Kante vom Knoten i zum Knoten j durch Transportfahrzeug des Lagers k , $x_{ij}^k \in \{0, 1\}$
x_{ijk}	Entscheidungsvariable für die Nutzung der Kanten von Knoten i nach Knoten j durch Transportfahrzeug k
X	Menge von Lösungsvektoren, die einen v-tree bilden, $x = \{x \mid x \in \{0, 1\}\}$
y_i	Beladung eines in i ankommenden Transportfahrzeuges

y_{ik}	Entscheidungsvariable, ob ein Fahrzeug k einen Verbraucher i besucht, $y_{ik} \in \{0, 1\}$
\mathbb{R}	Wertebereich der reellen Zahlen
\mathbb{R}^n	Wertebereich der reellen Zahlen mit n Dimensionen
\mathbb{Z}^n	Wertebereich der ganzen Zahlen mit n Dimensionen
\mathbb{Z}_+^n	Wertebereich der positiven ganzen Zahlen mit n Dimensionen
α	Straffaktor, $\alpha \geq 0$
$\delta(i)$	Grad eines Knoten i
Δ_c	Kostendifferenz zweier Lösungen
θ	Zufallszahl, $\theta \in [0, 1) \subset \mathbb{R}$
π	Knotenreihenfolge
π_j	Knotenreihenfolge für Transportfahrzeug j
ϕ_{ij}	Fluss, transportierte Menge entlang der Kante (i, j)

Abkürzungsverzeichnis

Abb.	Abbildung
Abk.	Abkürzung
Alg.	Algorithmus
AP	Assignment Problem
BnB	Branch and Bound
BnC	Branch and Cut
bzw.	beziehungsweise
COP	Combinatorial Optimization Problem
DVRP	Deadline Vehicle Routing Problem
engl.	englisch
et al.	et alii (lateinisch: und andere)
f.	folgende (Seite)
ff.	folgende (Seiten)
GA	Genetischer Algorithmus
gdw.	genau dann, wenn
i. A.	im Allgemeinen
i. d. R.	in der Regel
inkl.	inklusive
ILP	Linear Integer Programming
IP	Integer Programming
LP	Linear Programming
m-TSP	Multiple Traveling Salesman Problem
MDVRP	Multi Depot Vehicle Routing Problem
NGAP	Nonlinear General Assignment Problem
\mathcal{NP}	nichtpolynomial
PDP	Pickup and Delivery Problem
RP	Routing Problem
SDVRP	Single Depot Vehicle Routing Problem
SPP	Shortest Path Problem
Tab.	Tabelle

TSP	Traveling Salesman Problem
TW	Time Window
u. B. d. N.	unter Beachtung der Nebenbedingungen
u. a.	unter anderem
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPTW	Vehicle Routing Problem with Time Windows
z. B.	zum Beispiel

1 Einleitung

Die Tourenbildung beschäftigt sich mit der Konstruktion kostengünstiger Transportrouten zur Belieferung von Verbrauchern. Sie ist eine der weitreichendsten Erfolgsgeschichten des Operations Research. Das starke Interesse an diesen Problemen durch Industrie und Forschung liegt zum einen am wirtschaftlichen Potenzial der Tourenbildung und -optimierung, zum anderen macht ihr Reichtum an Struktur sie zu einem faszinierenden Forschungsgebiet.

In der vorliegenden Arbeit soll ein Überblick über einige, u. a. auch neuere mathematische Modell- und Lösungsansätze gegeben werden. Auf Grund der hohen Anzahl der Veröffentlichungen auf diesem Gebiet wird nicht zwingend ein Anspruch auf die vollständige Darlegung aller möglichen Problemstellungen im Zusammenhang mit dem TSP sowie dem VRP und deren Lösungsansätze erhoben. An den gegebenen Stellen wird statt dessen auf weiterführende Literatur verwiesen.

Eine der ersten Problemstellungen im Zusammenhang mit der Tourenbildung stellt das *Traveling Salesman Problem (TSP)* dar. Es beschäftigt sich mit der Frage, wie der Verlauf der kostengünstigsten Tour durch eine fest vorgegebene Anzahl von Orten zu erfolgen hat, welche jeweils nur einmal besucht werden dürfen. Aufbauend auf dieser grundlegenden Formulierung des TSP wurden Erweiterungen in verschiedene Richtungen vorgenommen, um die Betrachtung noch komplexerer Probleme ermöglichen zu können. Beim *Multiple Traveling Salesman Problem (m-TSP)* werden beispielsweise nicht nur eine, sondern mehrere kostenminimale TSP-Touren gesucht, deren Bildung gleichberechtigt nebeneinander erfolgt. Ein wesentliches Merkmal des TSP, dass jeder Ort nur einmal aufgesucht werden darf, bleibt jedoch auch beim m-TSP erhalten. Eine ähnliche Aufgabe ergibt sich beim *Vehicle Routing Problem (VRP)*, welches sich vom m-TSP darin unterscheidet, dass für einzelne Transportfahrzeuge Touren zur Bedarfsdeckung bestimmter Verbraucher¹ gebildet werden müssen. Erste Formulierungen auf diesem Gebiet stammen von Dantzig und Ramser². Als Ausgangspunkt der Lieferungen ist beim ursprünglichen, klassischen VRP ein zentrales Lager vorgegeben, in welchem die einzelnen Transportfahrzeuge stationiert sind. Als zusätzliche Schwierigkeiten kommen hier noch

- *Kapazitätsrestriktionen*, jedes Transportfahrzeug besitzt eine einzuhal-

¹ Verbraucher stellen in diesem Zusammenhang entweder Orte mit Forderungen nach Aufnahme benötigter oder Abgabe überschüssiger Güter dar.

² siehe [DR59]

tende, maximale Ladekapazität, und

- *Zeitrestriktionen*, sowohl das zentrale Lager als auch einzelne Verbraucher sind innerhalb festgelegter Zeitintervalle (*Time Windows (TW)*) zu erreichen und zu verlassen (In diesem Zusammenhang wird auch von *Deadline Vehicle Routing Problem* gesprochen.),

der einzelnen Transportfahrzeuge hinzu. Es bestehen folglich zwei Probleme, deren Lösungen voneinander abhängig sind und zur Bestimmung der Gesamtlösung dienen:

- das *Zuordnungsproblem* einzelner Verbraucher zu Transportfahrzeugen und
- das *Tourenbildungsproblem* einer TSP-Route durch alle zu einem Transporter zugeordneten Verbraucher.

Weiterreichende Betrachtungen des VRPs führen letztendlich zu Forschungsaktivitäten, welche sich im Wesentlichen einer der drei folgenden Kategorien zuordnen lassen:

- (i) nur Be- oder Entladeorte sind Gegenstand der Betrachtung
- (ii) Option der Mitnahme von Waren während eines Transportes
- (iii) Kombination von Be- und Entladung innerhalb einer Tour

Durch diese Problemvielfalt bedingt, ergeben sich auch verschiedene mathematische Formulierungen und Lösungsansätze mit unterschiedlicher Komplexität. Der Punkt (i) stellt dabei zwar den einfachsten Fall des VRP dar, ist aber dennoch ein sehr komplexes kombinatorisches Problem, dessen Suche nach einer exakten Lösung sich als \mathcal{NP} -hart erweist³. Die Möglichkeit der Rücknahme von Gütern zum zentralen Lager während einer Auslieferung, wie sie unter (ii) formuliert ist, tritt in der Literatur oftmals als VRPB (*Vehicle Routing Problem with Backhauls*) auf⁴. Das unter (iii) beschriebene Problem wird letztendlich als *Pickup and Delivery Problem* (PDP) bezeichnet.

Das Zusammenspiel von theoretischen und praktischen Untersuchungen ist für die Lösung des VRP von besonderem Interesse. Einerseits stellt die Theorie mathematische Modelle und Algorithmen bereit und andererseits liefert die Praxis die notwendigen hard- und softwaretechnischen Voraussetzungen zur reellen Anwendbarkeit. Somit ist ein Kompromiss zwischen der Implementierung der mitunter sehr komplexen Algorithmen in einem Rechnersystem und dem Herausfiltern von problemspezifischen Charakteristiken der Modelle zu finden. Es ist darauf zu achten, dass die wirtschaftliche Verwertbarkeit der entstandenen Lösungen bei der Modellabstraktion stets erhalten bleibt.

Im Abschnitt 2 werden zunächst einige wenige graphentheoretische Grundlagen vermittelt, welche im weiteren Verlauf der Arbeit von Interesse sind. Nach

³ Beweis siehe [LR81]

⁴ siehe beispielsweise [GBAS85]

einleitenden Bemerkungen zum TSP soll im Abschnitt 3 das VRP etwas genauer betrachtet werden. Dabei erfolgt auch eine Betrachtung der vielfältigen Variationsmöglichkeiten und Erweiterungen des klassischen VRP, wie z. B. das *Multi Depot Vehicle Routing Problem (MDVRP)*. Die exakten Lösungsverfahren sowie Heuristiken, um lokaloptimale Lösungen zu erzielen, stehen im Mittelpunkt der Betrachtungen des Abschnittes 4. Die zugehörigen Algorithmen befinden sich im Anhang A.

Wie bereits an den einleitenden Worten erkennbar ist, wird auch im weiteren Verlauf der Arbeit auf englischsprachige Fachbegriffe zurückgegriffen. Grund dafür sind die zumeist in englischer Sprache verfassten Titel, auf welche in dieser Arbeit verwiesen wird. Eine sinngemäße deutschsprachige Übersetzung dieser, im Englischen üblichen Formulierungen ist in den Tabellen B.1 und B.2 des Anhanges B zu finden.

2 Grundbegriffe

Zunächst sollen einige wenige grundlegende Begriffe der Graphentheorie und der Transportoptimierung vorgestellt werden, die im Weiteren Verwendung finden.

2.1 Graphentheorie

Ein *Graph* $G = (V_G, E_G)$ besteht aus einer endlichen, nichtleeren *Menge von Knoten* $V_G = \{1, 2, \dots, n\}$ und einer endlichen *Menge von Kanten* $E_G = \{(i, j) \mid \exists \text{ direkte Verbindung vom Knoten } i \in V \text{ zum Knoten } j \in V\}$. Werden für die einzelnen Kanten darüber hinaus noch *Bewertungen* $c_{ij} = c(i, j) \in \mathbb{R}_+$ angegeben, so entsteht ein *bewerteter Graph*. Bewertungen stehen bei Kanten z. B. für Entfernungen, Reisezeiten oder Reisekosten zwischen zwei Knoten i und j . Bewertungen werden bei Kanten auch als *Gewichte* bezeichnet. Für gewöhnlich erfolgt die Zusammenfassung der individuellen Bewertungen der Kanten in der *Kostenmatrix*

$$C = (c_{ij}) \quad c_{ij} = \begin{cases} c_{ij}, & \text{falls } (i, j) \in E_G, i \neq j \\ \infty, & \text{sonst} \end{cases}$$

Graphen können nach gerichteten und ungerichteten Graphen unterschieden werden. Bei einem *gerichteten Graphen*, auch als *Digraph*¹ bezeichnet, hat jede Kante einen Anfangs- und einen Endknoten, so dass zwischen der Kante (i, j) und der entgegengesetzten Kante (j, i) unterschieden wird. Für die Kante (i, j) eines Digraphen ist i *Vorgänger* von j und j *Nachfolger* von i , wobei $P(i)$ die *Menge der Vorgänger* von i und $S(i)$ die *Menge der Nachfolger* von i bezeichnet. Die Knoten i und j sind in diesem Falle *Nachbarn*. Der *Grad* $\delta(i)$ gibt die Anzahl der Nachbarn eines Knoten i an. Ein Digraph gilt als *symmetrisch*, wenn er beide entgegengesetzten Kanten enthält. In diesem Falle wird auch von einem *ungerichteten Graphen* gesprochen. Sind Anfangs- und Endknoten identisch, entsteht eine *Schlinge*. Durch die Verbindung von zwei Knoten durch mehrere Kanten kommen *Mehrfachkanten* zu Stande (siehe Abbildung 2.1). Es entsteht ein *Multigraph*.

¹ Abkürzung für engl. *directed graph*

(b)). Dazu muss der Grad eines jeden Knotens gerade sein. Ein *Minimalgerüst* eines bewerteten Graphen entsteht, wenn alle Knoten durch die Kanten mit den minimalen Bewertungen verbunden werden (siehe Abbildung 2.5 (a)). Die Summe der Bewertungen der Kanten des Gerüsts wird dadurch minimal. Ein *Matching* M eines Graphen ist eine Teilmenge dessen Kantenmenge. Diese Teilmenge besitzt die Eigenschaft, dass keine zwei voneinander verschiedenen Kanten aus M mit ein und demselben Knoten des Graphen verbunden sind (siehe Abbildung 2.5 (c)). Ein Matching M heißt *perfekt*, wenn M alle Knoten des Graphen abdeckt.

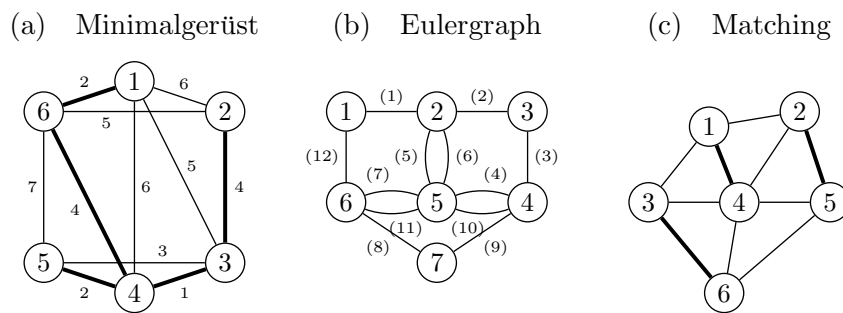


Abb. 2. 4

Abbildung 2.4: Fluss zwischen Knoten

Ein *Fluss* ϕ_{ij} kennzeichne die transportierte Menge eines Gutes entlang einer Kante (i, j) , $j \in \dot{R}(i)$ innerhalb eines Netzwerkes (siehe Abbildung 2.4).

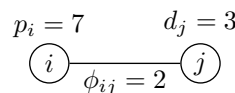


Abb. 2. 5

Abbildung 2.5: Minimalgerüst, Eulergraph und Matching

Die im Folgenden betrachteten Graphen sollen stets *zusammenhängende, symmetrische und bewertete* Graphen sein, welche weder Schlingen noch Mehrfachkanten enthalten.

2.2 Transportoptimierung

Beim *Routing Problem (RP)* ist der kostengünstigste Transport eines Gutes von gewissen *Produzenten* $K_P = \{i_1, i_2, \dots, i_m\}$ zu *Verbrauchern* $D_V = \{j_1, j_2, \dots, j_n\}$ eines Gutes gesucht (siehe Abbildung 2.6). Produzenten und Verbraucher bilden dabei die Knotenmenge $V_R = K_P \cup D_V$ eines Graphen $G_R = (V_R, E_R)$. Die Kantenmenge $E_R = K_P \times D_V$ gibt alle möglichen Transportwege innerhalb des Graphen an. Sei die von einem Produzenten $i \in K_P$

abzugebende Menge mit $p_i > 0$ und die von einem Verbraucher $j \in D_V$ benötigte Menge mit $d_j > 0$ sowie die Kosten für den Transport eines Gutes von i nach j entlang der Kante $(i, j) \in E_R$ mit c_{ij} bezeichnet, so lässt sich das entstehende Routing Problem mathematisch wie folgt formulieren:

$$(RP) \quad \min \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \tag{2.1}$$

u. B. d. N.

$$\sum_{j=1}^n x_{ij} = p_i \quad \text{für } i = 1, 2, \dots, m \tag{2.2}$$

$$\sum_{i=1}^m x_{ij} = d_j \quad \text{für } j = 1, 2, \dots, n \tag{2.3}$$

$$x_{ij} \geq 0 \quad \text{für } i = 1, 2, \dots, m, j = 1, 2, \dots, n \tag{2.4}$$

mit

x_{ij} - von i nach j transportierte Menge

Die auf einer Kante $(i, j) \in E_R$ transportierte Menge ist beim *unkapazitierten Transportproblem*, wie es hier betrachtet werden soll, nach oben unbeschränkt. Die Kosten bezeichnen zumeist die Fahrzeit, die Distanz oder anderweitige Kosten einer Strecke.

Abb. 2.6

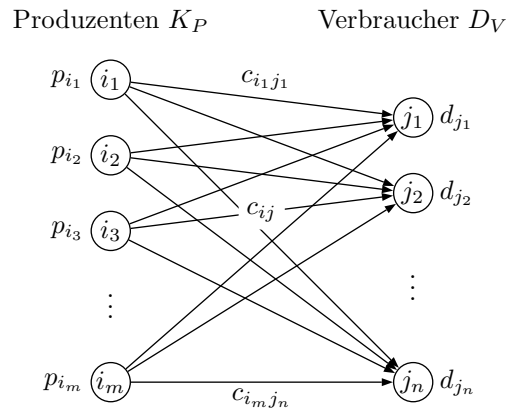


Abbildung 2.6: Transportwege beim Routing Problem

Übersteigt die produzierte Menge den Bedarf, d. h. $\sum_{i=1}^n p_i > \sum_{j=1}^m d_j$, so ist ein weiterer *fiktiver Verbraucher* $n + 1$ mit $d_{n+1} = \sum_{i=1}^n p_i - \sum_{j=1}^m d_j$ einzuführen. Ähnliches gilt, wenn der Bedarf die produzierte Menge übersteigt. In diesem Falle muss ein weiterer *fiktiver Produzent* eingeführt werden, so dass stets $\sum_{i=1}^n p_i = \sum_{j=1}^m d_j$ gilt. Des Weiteren sei noch angemerkt, dass der zulässige Bereich des RP auf Grund der Bedingung

$$0 \leq x_{ij} \leq \min(p_i, d_j) \quad \text{für } i = 1, 2, \dots, m, j = 1, 2, \dots, n$$

beschränkt ist. Er stellt ein nichtleeres konvexes Polytop dar, innerhalb dessen das Routing Problem stets eine optimale Lösung besitzt.

An dieser Stelle sei noch ein Spezialfall des RP, das *Assignment Problem (AP)*, betrachtet. Bei diesem ist $m = n$ zugewiesen und es gilt für die Abgabe- und Bedarfsmengen $p_i = d_j = 1 \forall i, j = 1, 2, \dots, n$. Weitere Einschränkungen dieses Problems führen letztendlich zum TSP, welches im Kapitel 3.1 betrachtet werden soll. Die Verringerung der Kantenmenge E_R auf $E_R \subset K_P \times D_V$, die kapazitätsmäßige Beschränkung der von einem Produzenten an mehrere Verbraucher abzugebenden Mengeneinheiten sowie eine weitere Spezialisierung des Zuordnungsproblems führen schließlich zu den unter 3.2 ff. betrachteten Problemen des Vehicle Routing.

3 Probleme bei der Tourenbildung

Das folgende Kapitel befasst sich mit einem speziellen Typ von *Combinatorial Optimization Problems (COP)*¹, dem Vehicle Routing Problem (VRP). Bei diesem müssen Zuordnungen von Verbrauchern zu Fahrzeugen getroffen und danach die Bildung von Transportrouten durchgeführt werden. Das Problem eine kostenminimale Reiseroute zu finden, besteht auch beim Traveling Salesman Problem (TSP). Aus diesem Grunde wird zu Beginn des Kapitels noch einmal kurz auf das TSP eingegangen.

3.1 Das Traveling Salesman Problem (TSP)

Im Sinne der Transportoptimierung bzw. Güterumverteilung besteht das *Problem des Traveling Salesman* darin, eine kostenminimale Reiseroute zur Belieferung von n Orten durch ein Transportfahrzeug mit unendlicher Kapazität zu finden, ohne einen dieser Orte mehr als einmal zu besuchen².

Gegeben sei ein schlingenfreier Graph $G = (V, E)$ wie er im Abschnitt 2.1 beschrieben ist (siehe Abbildung 3.1).

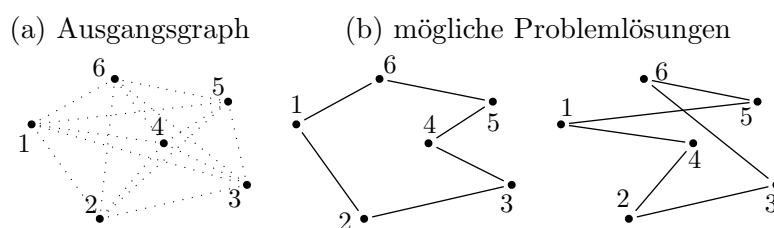


Abb. 3. 1

Abbildung 3.1: Ein Traveling Salesman Problem

Gesucht ist der Hamiltonsche Kreis minimaler Länge, welcher durch alle n Knoten geht und jeden dieser genau einmal enthält.

¹ Ein COP besteht aus einer Grundmenge E und einer Menge $F \subseteq 2^E$ möglicher Problemlösungen.

² siehe [LLRS85]

Es ergibt sich somit folgendes Optimierungsproblem:

$$(TSP) \quad \min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \quad (3.1)$$

u. B. d. N.

$$\sum_{j=1}^n x_{ij} = 1 \quad \text{für } i = 1, 2, \dots, n \quad (3.2)$$

$$\sum_{i=1}^n x_{ij} = 1 \quad \text{für } j = 1, 2, \dots, n \quad (3.3)$$

$$x_{ij} \in \{0, 1\} \quad (3.4)$$

mit

$$x_{ij} = \begin{cases} 1, & \text{falls } \exists \text{ direkte Verbindung von } i \text{ nach } j \\ 0, & \text{sonst} \end{cases}$$

Bei der exakten Suche nach der optimalen Lösung sollten sinnvollerweise zusätzliche Bedingungen zur Entfernung von Kurzzyklen hinzugefügt werden, wie z. B.:

$$\sum_{\{i,j\} \subseteq S} x_{ij} \leq |S| - 1 \quad \forall S \subset V, |S| \geq 2 \quad (3.5)$$

und

$$|\text{Schnittkanten von } S \text{ mit } \{x_{ij}\}| \geq 2 \quad \forall (S, V - S), \emptyset \subset S \subset V \quad (3.6)$$

3.2 Das klassische Vehicle Routing Problem (VRP)

Das *klassische Vehicle Routing Problem* formulierten Dantzig und Ramser³ erstmals 1959. Zu Grunde liegt ein COP, bei dem alle Verbraucher $i \in N = \{1, 2, \dots, n\}$ mit den Bedarfsmengen d_i an einem homogenen Gut, die ausgehend von einem zentralen Lager 0, beliefert werden müssen. Die Belieferung erfolgt mittels v identischen Transportfahrzeugen, welche eine maximale Transportkapazität von C_{max} besitzen (*Kapazitätsrestriktion*). Das Ziel ist die Minimierung der Transportkosten, wobei $c_{ij} \geq 0$ die Kosten, unabhängig von der transportierten Menge, für den Transport eines Gutes vom Verbraucher i zum Verbraucher j mit $0 \leq i, j \leq n$ bezeichnet.

Den Ausgangspunkt bildet demnach ein bewerteter Graph $G = (V, E)$ mit der Knotenmenge $V = N \cup \{0\}$ sowie der Kantenmenge $E = (\{0\} \times N) \cup I \cup (N \times \{0\})$, wobei $I \subset N \times N$ die Menge der verbindenden Kanten bezeichnet (siehe Abbildung 3.2).

³ siehe [DR59]

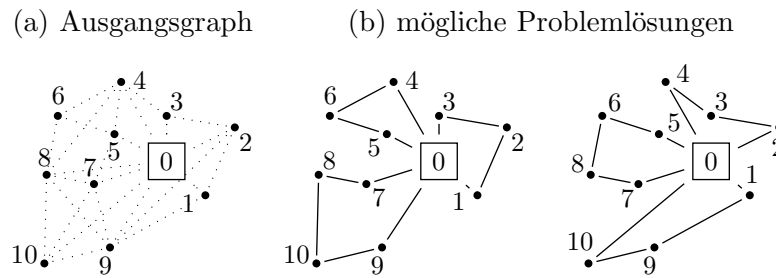


Abb. 3.2

Abbildung 3.2: Ein Vehicle Routing Problem

Die mathematische Formulierung des VRP als COP sei wie folgt⁴:

$$\text{(VRP)} \quad \min \sum_{(i,j) \in E} c_{ij} x_{ij} (+ \sum_{j \in N} c x_{0j}) \quad (3.7)$$

u. B. d. N.

$$\sum_{j \in N} x_{ij} = 1 \quad \text{für } i \in N \quad (3.8)$$

$$\sum_{j \in N} x_{j0} + \sum_{j \in N} x_{0j} = 2v \quad (3.9)$$

$$\sum_{j \in N} x_{ij} + \sum_{j \in N} x_{ji} = 2 \quad \text{für } i \in N \quad (3.10)$$

$$x_{ij}(y_i + d_i - y_j) \leq 0 \quad \text{für } (i, j) \in I \quad (3.11)$$

$$0 \leq y_i \leq C_{max} \quad \text{für } i \in N \quad (3.12)$$

$$x_{ij} \in \{0, 1\} \quad \text{für } (i, j) \in E \quad (3.13)$$

$$y_i \text{ ganzzahlig} \quad \text{für } i \in N \quad (3.14)$$

mit

$$x_{ij} = \begin{cases} 1, & \text{falls } (i, j) \in E \text{ von einem der Fahrzeuge genutzt wird} \\ 0, & \text{sonst} \end{cases}$$

y_i – Beladung eines in i ankommenden Transporters

Die in Klammer der Gleichung (3.7) angegebenen Kosten können entfallen, wenn die fixen Kosten für jedes das zentrale Lager verlassende Fahrzeug ohne Beachtung bleiben sollen.

Als untere Schranke für die zur Belieferung der Verbraucher benötigte Fahrzeugmenge kann $v_{min} = \left\lceil \frac{\sum_{i \in N} d_i}{C_{max}} \right\rceil$ definiert werden. Dadurch ist zugleich die minimale Anzahl der zu bildenden Fahrzeugrouten festgelegt.

Treten Verbraucher mit $n = \left\lceil \frac{d_i}{C_{max}} \right\rceil - 1 > 0$ auf, so werden diesen im Vorfeld n einzelne Touren fest zugeordnet und der Bedarf des Verbrauchers i auf $d_{i_{neu}} = d_i - nC_{max}$ herabgesetzt. Ist der Bedarf des Verbrauchers i nach der Zuordnung bereits gedeckt, d. h. $d_{i_{neu}} = 0$, kann der Verbraucher aus dem Graphen entfernt werden.

⁴ nach [AG88, S. 67]

3.3 Erweiterungen des klassischen VRP

Die Existenz von Nebenbedingungen, wie

- unendliche Transportkapazitäten,
- Einbeziehung von Zeitfenstern,
- Start von mehreren Lagern,
- Einbeziehung von Rückgabemöglichkeiten,
- Kombination von Be- und Entladung und
- variable Beladepkapazitäten

erfordert zusätzliche Überprüfungen während der Suche nach einem lokalen oder globalen Optimum unter Anwendung eines vorhandenen Algorithmuses.

Im Allgemeinen spricht nichts gegen eine beliebige *Kombination der* oben genannten *Teilprobleme*. Zu beachten ist jedoch, dass die zusätzlichen Nebenbedingungen einerseits zu weiteren Dimensionen des Suchraumes und andererseits zu einer Einschränkung des entstandenen Suchraumes führen können. In beiden Fällen bleiben die Lösungsalgorithmen zum Auffinden einer globaloptimalen Lösung aber nach wie vor \mathcal{NP} -hart, weil sowohl die Menge der möglichen als auch die Menge der unmöglichen Problemlösungen erst einmal bestimmt werden muss⁵.

3.3.1 Das Multiple Traveling Salesman Problem (m-TSP)

Durch die Aufhebung der Kapazitätsbeschränkungen aller v Transportfahrzeuge, d. h. $C_{max} = \infty \forall v$, entsteht das so genannte *Multiple Traveling Salesman Problem (m-TSP)*. Bei diesem sind n Orte und m Traveling Salesmen gegeben. Ziel ist das Auffinden einer kostenminimalen Lösung, welche aus m Touren besteht, wobei alle Orte genau einmal durch einen Beliebigen der Traveling Salesmen besucht werden müssen.

Christofides und Eilon⁶ strukturieren das Problem dazu so um, dass ein TSP mit $n + m$ Orten entsteht (siehe Abbildung 3.3). Das zentrale Lager, der Knoten $\{0\}$, wird hierfür m mal vervielfältigt, wodurch die Menge $\{0_1, 0_2, \dots, 0_m\}$ gebildet wird. Jede dieser Kopien erhält eine Verbindung mit den Knoten $i, \forall \{0, i\} \in E$ und wird mit den originalen Reisekosten c_{0i} versehen. Verbindungskanten zwischen den einzelnen Lagern $(0_i, 0_j), i \neq j, 1 \leq i, j \leq m$ bekommen den Wert unendlich zugewiesen, d. h. $c_{0_i 0_j} = \infty$.

Ist eine mögliche heuristische Anfangslösung gefunden worden, können unter

⁵ Lenstra und Rinnooy zeigen dies in [LR81]

⁶ siehe [CE69]

Abb. 3.3

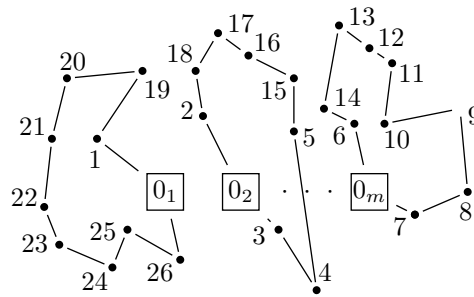


Abbildung 3.3: Ein Multiple Traveling Salesman Problem

Zuhilfenahme des Verfahrens 3-opt (siehe Abschnitt 4.2) und Beachtung der Kapazitätsrestriktion der Transportfahrzeuge schnell gute heuristische Ergebnisse erzielt werden⁷. Der Austausch von Knoten/Verbrauchersorten innerhalb einer Tour ist dabei genauso zulässig wie der Austausch zwischen mehreren Touren.

3.3.2 Berücksichtigung von Zeitfenstern

In der Praxis fordern einige oder alle Verbraucher oftmals Zeitintervalle, in denen ihre Belieferung erfolgen muss (*Zeitrestriktion*). Für das TSP und VRP bedeutet dies eine Vorgabe von entsprechenden *Zeitfenstern* $[e_i, l_i]$ für alle Knoten $i \in N$, wobei e_i für den frühestmöglichen und l_i für den spätmöglichen Belieferungszeitpunkt steht⁸. Zusätzlich kann noch die Festlegung einer für alle Touren einzuhaltenden Maximaldauer sowie die Angabe eines Zeitfensters für den möglichen Startpunkt eines Transportes vom Lager erfolgen. Es entsteht ein gerichteter Graph (siehe Abbildung 3.4).

Im Folgenden sei festgelegt, dass die Fahrzeit von einem Verbraucher $i \in N$ zu einem Verbraucher $j \in N, j \neq i$ dem Wert t_{ij} entspreche und die Ankunftszeit beim Verbraucher i mit A_i sowie die Abfahrtszeit von i mit D_i bezeichnet sei. Darüber hinaus gelte für alle $i \in N$ mit $A_i < e_i$, dass $A_i = e_i$, wodurch eine Wartezeit $W_i = e_i - A_i$ entsteht. Deren Minimierung könnte beispielsweise ebenfalls Ziel einer Optimierung sein.

Der Einbezug von Zeitfenstern in die Tourenbildung erfordert somit die Erweiterung der Formeln (3.7) - (3.14) um folgende zusätzliche Nebenbedingungen:

$$x_{ij}(D_i + t_{ij}D_j) \leq 0 \quad \text{für } (i, j) \in A \quad (3.15)$$

$$e_i \leq D_i \leq l_i \quad \text{für } i \in N \quad (3.16)$$

⁷ siehe [Lin65]

⁸ siehe [Chr85]

Abb. 3.4

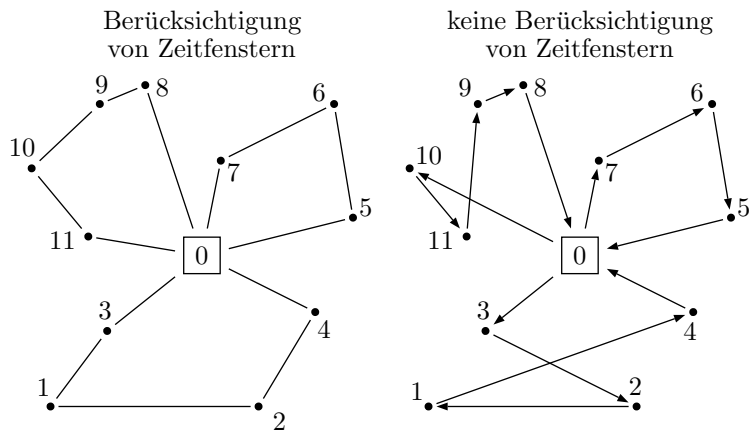


Abbildung 3.4: Ein VRP ohne und mit Zeitfenstern

3.3.3 Mehrere Lager

Die Existenz mehrerer Lager bedingt die Ersetzung des zentralen Lagers $\{0\}$ durch eine Menge aus Lagern ($K = \{0_1, 0_2, \dots, 0_m\}$). Als Ausgangspunkt dient in diesem Falle eine Menge von m Graphen, zusammengesetzt aus den einzelnen Graphen $G^k = (V^k, E^k), k \in K$, wobei $V^k = \{0_k\} \cup N$ dessen Knotenmenge und $E^k = (\{0_k\} \times N) \cup I \cup (N \times \{0_k\})$ dessen Kantenmenge angeben. Die Menge N beschreibt wie beim klassischen VRP die Menge der Verbraucher und $I \subset N \times N$ die möglichen Transportwege zwischen den Verbrauchern (siehe Abbildung 3.5).

Abb. 3.5

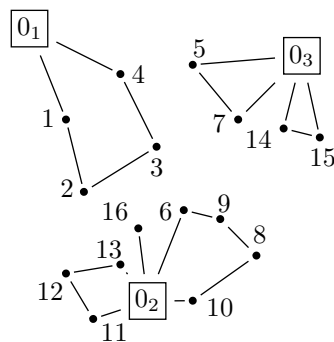


Abbildung 3.5: Ein Multidepot Vehicle Routing Problem

Die Lösung des zu Grunde liegenden Problemes ist an sich in zwei verschiedenen Varianten denkbar. Bei der ersten Variante müssen alle Transportfahrzeuge stets zu ihrem Ausgangslager zurückkehren. Dies erfordert eine Neudefinition des klassischen VRP als ein Flussproblem mit mehreren Gütern.

Die Variablen x_{ij} und v des klassischen VRP von Seite 13 f. werden ersetzt

durch die Variablen

$$x_{ij}^k = \begin{cases} 1, & \text{falls } (i, j) \in E \text{ von einem Fahrzeug} \\ & \text{des Lagers } k \in K \text{ genutzt wird} \\ 0, & \text{sonst} \end{cases}$$

und

$$v^k \quad \text{Anzahl der Transportfahrzeuge im Lager } k \in K$$

Die mathematische Formulierung als COP lautet dann wie folgt⁹

$$\text{(MDVRP)} \quad \min \sum_{k \in K} \sum_{(i,j) \in E^k} c_{ij} x_{ij}^k \quad (3.17)$$

u. B. d. N.

$$\sum_{k \in K} \sum_{j \in V^k} x_{ij}^k = 1 \quad \forall i \in N \quad (3.18)$$

$$\sum_{j \in N} x_{j0_k}^k + \sum_{j \in N} x_{0_k j}^k = 2v^k \quad \forall k \in K \quad (3.19)$$

$$\sum_{j \in V^k} x_{ij}^k + \sum_{j \in V^k} x_{ji}^k = 2 \quad \forall i \in V^k, k \in K \quad (3.20)$$

$$x_{ij}^k (y_i + d_i - y_j) \leq 0 \quad \forall (i, j) \in I, k \in K \quad (3.21)$$

$$0 \leq y_i \leq C \quad \forall i \in N \quad (3.22)$$

$$x_{ij}^k \in \{0, 1\} \quad \forall (i, j) \in E^k, k \in K \quad (3.23)$$

$$y_i \quad \text{ganzzahlig} \quad \forall i \in N \quad (3.24)$$

Die zweite Variante ermöglicht die *Rückkehr* eines jeden Transportfahrzeuges zu einem beliebigen Lager $k \in K$. Hierbei müssen zusätzliche Nebenbedingungen aufgenommen werden, welche die Einhaltung der Flussbedingungen aus Gleichung (3.10) für jedes der Lager sicherstellen.

3.3.4 Einsatz einer heterogenen Fahrzeugflotte

Eine *heterogene Fahrzeugflotte* zur Bedarfsdeckung entsteht dadurch, dass ein Lager über m verschiedenartige Transportfahrzeuge $v = \{v_1, v_2, \dots, v_m\}$ mit unterschiedlicher Transportkapazität $C_{max}(v_i)$, $i = 1, 2, \dots, m$ verfügt¹⁰. Die Lösung dieses Problem es erfordert die Einführung von fiktiven Lagern $k \in K = \{0_1, 0_2, \dots, 0_m\}$ für jeden der m Fahrzeugtypen. Diese fiktiven Lager bekommen Kanten zu den Verbrauchern $i, \forall i \in \{0, i\}$, mit denen auch das ursprüngliche Lager $\{0\}$ verbunden war, hinzugefügt. Außerdem wird ein Transport zwischen den fiktiven Lagern verboten, d. h. $c_{ij} = \infty, i, j \in K, i \neq j$. Danach ähnelt das Problem dem aus Abschnitt 3.3.3, bei welchem mehrere

⁹ nach [Fis95, S. 48]

¹⁰ siehe [RS94]

Lager gegeben und die Rückkehr der Transportfahrzeuge zu ihrem Ausgangslager gefordert waren. Zur Verwendung des dort dargelegten Lösungsansatzes müssen lediglich die Bedingungen (3.21), (3.22) und (3.24) ersetzt werden durch

$$x_{ij}^k (y_i^k + d_i - y_j^k) \leq 0 \quad \forall (i, j) \in I, k \in K \quad (3.25)$$

$$0 \leq y_i^k \leq C_{max}^k \quad \forall i \in N, k \in K \quad (3.26)$$

$$y_i^k \quad \text{ganzzahlig} \quad \forall i \in N, k \in K \quad (3.27)$$

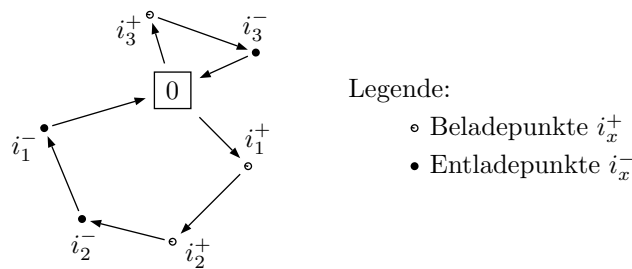
mit

- y_i^k – Beladung eines in i ankommenden Transporters $k \in K$
- C_{max}^k – maximale Transportkapazität eines Transporters $k \in K$

3.3.5 Kombination von Be- und Entladung

Einen weiteren Spezialfall des VRP stellt das *Pickup and Delivery Problem* (*PDP*) dar. Gegenstand des PDP ist, dass jeder Verbraucher $i \in N$ den Gütertransport von einem Beladepunkt i^+ zu einem Entladepunkt i^- anfordert. Vorgegeben sei wiederum ein bewerteter Graph $G = (V, E)$ mit der Knotenmenge $V = \{0\} \cup N$, $N = N^+ \cup N^-$, wobei $\{0\}$ das zentrale Lager, $N^+ = \{i^+ \mid i \in N\}$ die Menge der Beladepunkte und $N^- = \{i^- \mid i \in N\}$ die Menge der Entladepunkte beschreibt, sowie der Kantenmenge $E = (\{0\} \times N^+) \cup I \cup (\{0\} \times N^-)$. Die Menge $I \subset (N^+ \times N^-) \times (N^+ \times N^-)$ gibt die möglichen Verbindungen zwischen den Be- und Entladepunkten an (siehe Abbildung 3.6).

Abb. 3.6



Legende:

- Beladepunkte i_x^+
- Entladepunkte i_x^-

Abbildung 3.6: Ein Pickup and Delivery Problem

Bei Verwendung von zeitlichen Restriktionen müssen *Beladezeiträume* $[e_{i^+}, l_{i^+}]$ und *Entladezeiträume* $[e_{i^-}, l_{i^-}]$ beachtet werden. Zudem seien jeder Kante $(i, j) \in E$ die Transportkosten c_{ij} sowie die Fahrzeiten t_{ij} zugeordnet. Zur Bedarfsdeckung steht eine Menge K von homogenen Transportfahrzeugen mit einer jeweiligen Maximalkapazität von C_{max} Transporteinheiten zum Gütertransport bereit. Ziel ist die *Minimierung der anfallenden Reisekosten* unter

Einbeziehung der *Zeit- und Kapazitätsrestriktionen*¹¹:

$$(PDP) \quad \min \sum_{(i,j) \in E, k \in K} c_{ij} x_{ij}^k \quad (+ \sum_{j \in N, k \in K} c x_{0j}^k) \quad (3.28)$$

u. B. d. N.

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \quad \text{für } i \in N^+ \quad (3.29)$$

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in N} x_{ji}^k = 2 \quad \text{für } i \in N^+ \cup N^-, k \in K \quad (3.30)$$

$$\sum_{j \in V} x_{i+j}^k + \sum_{j \in N} x_{ji}^k = 2 \quad \text{für } i \in N, k \in K \quad (3.31)$$

$$D_{i^+} + t_{i+i^-} \leq D_{i^-} \quad \text{für } i^+, i^- \in N \quad (3.32)$$

$$x_{ij}^k (D_i + t_{ij} - D_j) \leq 0 \quad \text{für } (i, j) \in I, k \in K \quad (3.33)$$

$$e_i \leq D_i \leq l_i \quad \text{für } i \in N^+ \cup N^- \quad (3.34)$$

$$x_{ij}^k (y_i + d_i - y_j) \leq 0 \quad i^+, i^- \in N, k \in K \quad (3.35)$$

$$0 \leq y_i \leq C_{max} \quad \text{für } i \in N^+ \cup N^- \quad (3.36)$$

$$x_{ij}^k \in \{0, 1\} \quad \text{für } (i, j) \in E, k \in K \quad (3.37)$$

$$y_i \text{ ganzzahlig} \quad \text{für } i \in N \quad (3.38)$$

mit

$$x_{ij}^k = \begin{cases} 1, & \text{falls } (i, j) \in E \text{ von einem Fahrzeug } k \in K \text{ genutzt wird} \\ 0, & \text{sonst} \end{cases}$$

y_i – Beladung eines in i ankommenden Transporters

3.3.6 Einbeziehung von Rückgabemöglichkeiten

Es sei die Annahme getroffen, dass Transportfahrzeuge sowohl Gütertransporte von einem Lager als auch zu einem Lager durchführen müssen. Anstatt separate Touren zur Auslieferung und zum Abtransport von Gütern zu konstruieren, könnte ein zur Güterauslieferung gesandter Transporter gleichzeitig auch Güter mit zu seinem Ausgangs- bzw. Ziellager zurücktransportieren (siehe Abbildung 3.7). Dies setzt allerdings stets voraus, dass genügend freier Transportraum zum Abtransport bereitsteht.

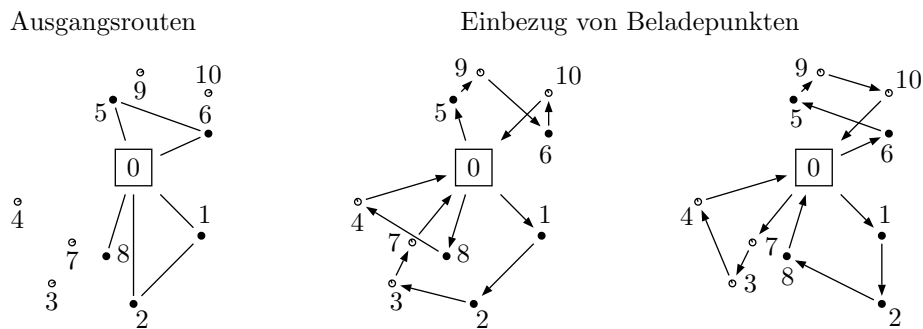
Neben exakten Verfahren zur Lösung dieses Problemes, dargelegt von Yano et al.¹² und Gelinas¹³, wurden auch zahlreiche Heuristiken entwickelt. Einige dieser sollen hier nur kurz vorgestellt werden, weil das Problem im Abschnitt 4.2.1 noch einmal aufgegriffen wird.

¹¹ nach [AG88, S. 69]

¹² siehe [YCR⁺87]

¹³ [Gel91]

Abb. 3.7



Legende siehe Abbildung 3.6 auf Seite 18

Abbildung 3.7: Ein VRP mit Rückgabemöglichkeiten

Die meisten der heuristischen Verfahren basieren auf einer Modifikation eines der folgenden Verfahren¹⁴:

- Clarke-und-Wright-Savings¹⁵,
- Cheapest-Insertions¹⁶ oder
- Spacefilling-Curves¹⁷

Deif und Bodin¹⁸ beschreiben zwei Varianten des Savingverfahrens von Clarke und Wright unter Einbezug von Rücktransportpunkten. In der ersten Variante wird die Bedingung gestellt, dass Beladepunkte erst dann zu einer Transportroute hinzugefügt werden dürfen, wenn alle Auslieferungen erfolgt sind. Diese Ausgangsrouten werden nur aus den Beladeorten unter Beachtung des Clarke-und-Wright-Savings¹⁹ gebildet. Die Hinzunahme von Rücktransportpunkten zu den gebildeten Abladerouten erfolgt dann ebenfalls unter Zuhilfenahme des Clarke-und-Wright-Savings, welches hierbei folgende Bedeutung hat, wobei i Auflade- und j Entladepunkt oder umgekehrt:

$$s_{ij} = c_{0i} + c_{0j} - c_{ij} \tag{3.39}$$

Das Verfahren besitzt allerdings den Nachteil, dass eine hohe Anzahl relativ kurzer Touren entsteht. Daraufhin haben Deif und Bodin als Erweiterung des Verfahrens das obige Saving s_{ij} in das *Backhaul Saving*

¹⁴ Die drei hier angeführten Verfahren werden im Abschnitt 4.2 auf Seite 35 genauer charakterisiert.

¹⁵ siehe [CW64]

¹⁶ siehe [RSL77]

¹⁷ siehe [BP82]

¹⁸ Nachzulesen in [DB84]

¹⁹ Saving kann mit dem deutschen Wort *Ersparnis* übersetzt werden.

$$bs_{ij} = c_{0i} + c_{0j} - c_{ij} - \alpha \hat{S} \quad (3.40)$$

wobei

\hat{S} - Schätzwert des maximalen Savings
 α - Straffaktor

abgeändert. Eine weitere Verbesserung erzielen sie, indem sie das Einfügen von Entladepunkten nach Aufladepunkten bereits dann erlauben, wenn das Backhaul Saving einen positiven Wert annimmt.

Als zweite Heuristik für die Einbeziehung von Rückgabemöglichkeiten seien zwei *Modifikationen des Insertion-Verfahrens* betrachtet. Diese verwenden folgende Strategie des *Stop-Based-Insertion*²⁰: Bilde die für die nachfolgenden Verbesserungen zu Grunde liegenden Touren für alle Entladepunkte nach einer beliebigen Heuristik. Füge anschließend die Aufladepunkte k zwischen zwei Punkte i und j unter Verwendung der Kostengleichung

$$c_k = c_{ik} + c_{kj} - c_{ij} - \alpha F \quad (3.41)$$

wobei

F - Anzahl Entladepunkte, die nach k folgen
 α - Straffaktor

zu den Ausgangsrouten hinzu, d. h. das Einfügen von Aufladepunkten erfolgt sobald ausreichend freie Kapazität auf einem Transporter vorhanden ist. Allerdings besteht eine hohe Abhängigkeit von der Anzahl der Entladepunkte, die einem Aufladepunkt folgen. Sinnvoller ist daher, die auf dem Transporter verbleibende Transportmenge zu beachten (*Load-Based-Insertion*). Diesen Gesichtspunkt berücksichtigen Casco et al.²¹. Darüber hinaus schlagen sie vor, die Bildung der zu Grunde liegenden Abladerouten unter Verwendung von Transportfahrzeugen mit einer leicht reduzierten Transportkapazität vorzunehmen. Bei der Einbeziehung der Aufladeorte ist diese dann wieder auf den ursprünglichen Wert heraufzusetzen. Des Weiteren legen Casco et al. noch Regeln fest, welche während allen Etappen der Tourenbildung zu beachten sind.

Die Verwendung von Heuristiken, welche auf *Spacefilling-Curves-Verfahren* beruhen, setzen genaueres Wissen über die räumlichen Positionen der einzelnen Punkte innerhalb eines festgelegten Koordinatensystems voraus. Dieses Wissen ist aber nicht in allen Fällen gegeben, so dass deren Anwendung nicht immer möglich ist, weshalb es innerhalb dieser Arbeit nicht weiter betrachtet werden soll. Auch das im vorangehenden Absatz erwähnte Verfahren von Casco et al. bezieht die Koordinaten der Auf- und Entladepunkte bei der Suche nach einer dem Optimum möglichst naheliegenden Problemlösung mit ein.

²⁰ nach [GBAS85]

²¹ siehe [AG88, S. 127-147]

4 Lösungsverfahren für das TSP und VRP

Tourenbildungsprobleme, wie das TSP und das VRP, sind aus kombinatorischer Sicht \mathcal{NP} -harte Probleme¹. Das bedeutet zugleich, dass es derzeit nicht möglich ist, einen Algorithmus zu finden, der solche Probleme in jedem Fall in polynomialer Zeit löst. Daher wurden eine Vielzahl von Heuristiken entwickelt, also Algorithmen, die auf den jeweiligen Anwendungsfall abgestimmt sind und hier eine vernünftige, wenn auch mathematisch zumeist nicht beweisbare, optimale Lösung liefern. Dennoch ist die Optimierung von Problemen der Tourenbildung schon seit mehreren Jahrzehnten eine faszinierende Forschungsaufgabe von großer ökonomischer Bedeutung.

Um die Schwierigkeiten zu verstehen, die das VRP beinhaltet, sollte sich vor Augen geführt werden, dass heute Instanzen des TSP mit mehreren tausend Orten mit Branch-and-Cut-Methoden exakt gelöst werden können, während beim VRP schon Instanzen mit über 70 Verbrauchern eine große Herausforderung darstellen.

4.1 Exakte Verfahren

Bevor die Lösungsverfahren zur Bestimmung einer *globaloptimalen Lösung* vorgestellt werden, sind noch einige Begriffe zu erläutern, welche im Zusammenhang mit Optimierungsproblemen auftreten.

Mit Hilfe von Methoden des *Linear Programming (LP)* lassen sich Optimierungsprobleme lösen, welche

- eine *lineare Zielfunktion* und
- *lineare Nebenbedingungen*

besitzen.

¹ Lenstra und Rinnooy zeigen dies in [LR81].

Aus mathematischer Sicht ergibt sich folgendes Problem:

$$(LP) \quad \min f(x) = c^T x \quad (4.1)$$

u. B. d. N.

$$Ax \geq b \quad (4.2)$$

$$x \geq 0 \quad (4.3)$$

$$x \in \mathbb{R}^n \quad (4.4)$$

Unter Verwendung der *Simplexmethode* oder *Interior-Point-Methoden* sind diese Problemstellungen effizient lösbar.

Eine Spezialisierung des LP-Problems entsteht im Falle einer Ersetzung der Nebenbedingung (4.4) durch

$$x \in \mathbb{Z}^n \quad (4.5)$$

Durch die Methoden des so genannten *Integer Programming (IP)* werden Möglichkeiten zur Verfügung gestellt, mit denen für dieses Problem eine exakte Lösung gefunden werden kann. Ist die Zielfunktion hierbei linear, so tragen Methoden des *Linear Integer Programming (ILP)* zur Problemlösung bei. Solche Optimierungsprobleme des IP und ILP sind im mathematischen Sinne gleichbedeutend mit COPs, welche sich i. A. mit dem Auffinden von Lösungen für *Permutations-, Reihenfolge- und Zuordnungsproblemen* befassen. Auch die *Dekomposition* in einzelne Problemtelmengen fällt in diesen Aufgabenbereich.

Ein COP soll im Folgenden ein Minimierungsproblem sein, welches aus

- einer Menge D_P von Instanzen I ,
- einer endlichen Menge $S_P(I)$ von (möglichen) Lösungen für jede Instanz $I \in D_P$ und
- einer Funktion m_P , die jeder Lösung $x \in S_P(I)$ zu jeder Instanz $I \in D_P$ einen positiven reellwertigen Lösungswert $m_P(x, I)$ zuordnet,

besteht. Ziel ist das Auffinden einer kostenminimalen Lösung x^* des COP:

$$x^* \in S_P(I) \text{ mit } m_P(x^*, I) \leq m_P(x, I) \quad \forall x \in S_P(I)$$

Sowohl für das TSP als auch für das VRP bedeutet dies eine Minimierung der zurückgelegten Wegstrecke:

$$L(\pi) = \sum_{i=1}^{n-1} d_{\pi(i), \pi(i+1)} + d_{\pi(n), \pi(1)} \quad (4.6)$$

$$L(\pi) = \sum_{j=1}^m \sum_{i=1}^{l_j-1} d_{\pi_j(i), \pi_j(i+1)} + d_{\pi_j(l_j), 0} + d_{0, \pi_j(l_j)} \quad (4.7)$$

wobei

π - Reihenfolge der Knoten (Permutation von $\{1, 2, \dots, n\}$)

π_j - Reihenfolge für Transporter j

l_j - Anzahl der Kunden für Transporter j

Gesucht ist jeweils die kostengünstigste Besuchsreihenfolge der n Knoten durch einen Traveling Salesman oder einen von v Transportern². Die Lösungsmenge des VRP wird für gewöhnlich noch durch explizite Nebenbedingungen, wie Ressourcen- und Zeitrestriktionen, weiter eingeschränkt. Auf Grund der endlichen Knotenmenge ergibt sich ebenfalls eine *endliche Anzahl von möglichen Lösungen* für das zugehörige COP, welche i. d. R. *exponentiell* mit der Problemgröße wächst.

Um eine exakte Lösung für ein COP zu erhalten, gibt es zwei prinzipielle Möglichkeiten:

- *Vollständige Enumeration*: Alle möglichen Permutationen der Problemlösung werden untersucht und bewertet.
- *Beschränkte Enumeration*: Nur eine eingeschränkte Anzahl von Lösungspermutationen wird zur Untersuchung herangezogen.

Wie bereits einleitend erwähnt wurde, gelten beide Problemlösungsklassen als \mathcal{NP} -hart, so dass die verwendeten Suchalgorithmen eine exponentielle Zeitkomplexität besitzen. Diese *untere zeitliche Schranke* versuchen die Lösungsverfahren der beschränkten Enumeration weiter zu verringern. Sie lassen sich im Wesentlichen in zwei Kategorien einteilen:

- Branch-and-Bound-Verfahren (BnB) und
- Branch-and-Cut-Verfahren (BnC).

Beiden Verfahren können verschiedene mathematische Zerlegungsmodelle mit unterschiedlichen Relaxationen zur Bestimmung einer unteren Grenze der Problemlösung zu Grunde liegen. Einige dieser für die exakte Lösung des TSP und des VRP interessanten Modelle sollen nach der Einführung von Suchbäumen etwas genauer betrachtet werden.

4.1.1 Suchbaumstrategien

Branch-and-Bound (BnB)

Die Branch-and-Bound-Methode beruht auf einer *Divide-and-Conquer-Strategie*, bei der die zu untersuchenden Teilprobleme unabhängig voneinander

² In diesem Falle ergeben sich v einzelne TSP-Touren.

sind³. Beim BnB wird durch Benutzung von *Primal- und Dualheuristiken* versucht, möglichst große Gruppen von Lösungen als nicht optimal zu erkennen, um sie von der vollständigen Enumeration ausschließen zu können. Die Aufgabe des *Branching* ist es, eine *Partitionierung der Lösungsmenge* vorzunehmen. Auf diese Partitionen werden dann bestimmte Heuristiken angewandt (*Bounding*). Die Hauptschwierigkeiten beim BnB bestehen demnach darin,

- „gute“ Zerlegungstechniken,
- einfache Datenstrukturen für die Abarbeitung der Teilprobleme und
- Heuristiken für gute Schranken zum Ausschluss einer möglichst großen Menge von Teilproblemen

zu finden. Das Grundprinzip ist der sukzessive Aufbau eines Suchbaumes, welcher aus einer Wurzel und den Wegen zu den aktiven, zu untersuchenden Knoten besteht (siehe Abbildung 4.1).

Abb. 4.1

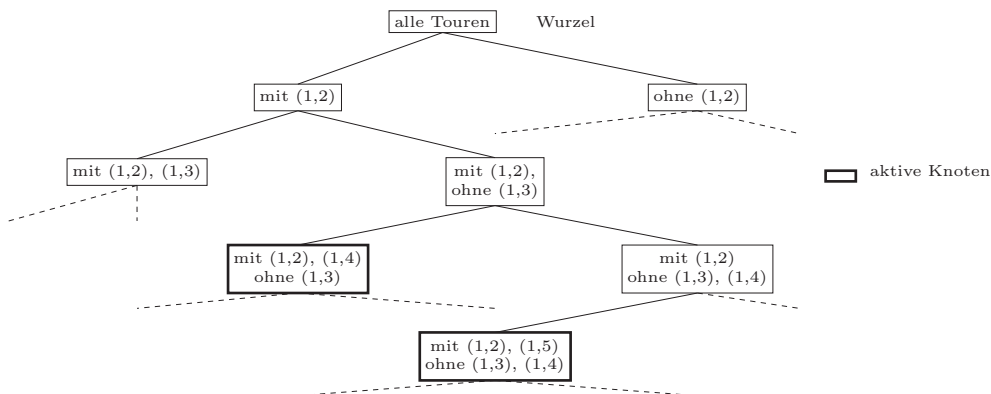


Abbildung 4.1: Kombinatorische Explosion eines Suchbaums

Genau genommen hat die *Bounding-Operation* zwei Aufgaben: zum einen entfernt sie solche Knoten (Teilmengen), welche auf Grund ihrer hohen Kosten nicht als Lösung des Problem in Frage kommen, zum anderen findet sie die kostengünstigsten Knoten. Um dies zu entscheiden wird mit Hilfe einer *Schrankenfunktion* eine untere Schranke für jede Teilmenge⁴ bestimmt, welche die günstigste Vervollständigung der partiellen Lösung des Problem angibt. Falls demnach ein Teilbaum noch nicht komplett existiert, kann so festgestellt werden, wie gut die Lösung des Teiles des Suchbaumes maximal werden kann.

Während der Abarbeitung des Suchbaumes erfolgt eine Speicherung der Kosten der bisher günstigsten Lösung, um eine obere Schranke für die restlichen verbleibenden, noch zu untersuchenden Teilprobleme des Suchbaumes (*aktive Knoten*) zu haben. Alle aktiven Knoten, bei welchen die untere Kostenschranke

³ siehe [NM93, S. 392-402]

⁴ Bei einem Maximierungsproblem entsprechend der obere Schrankenwert oder Vorzeichenumkehr bestimmter Teile der Problemfunktionen.

oberhalb des aktuellen oberen Schrankenwertes liegt, können aus dem Suchbaum entfernt und der Zweig abgeschnitten werden, weil sich in ihm keine bessere Lösung befinden kann. Sind die Kosten eines aktiven Knotens unterhalb des Schrankenwertes, entsteht eine neue obere Schranke. Um zu entscheiden, welcher der aktiven Knoten als nächster untersucht werden soll, existieren mehrere Auswahlstrategien:

- *Depth First*: Es wird mit dem Teilproblem weitergemacht, welches als letztes eingefügt wurde (Stack, Tiefensuche).
- *Breadth First*: Es wird das Teilproblem behandelt, welches als erstes eingefügt wurde (Schlange, Breitensuche).
- *Best First*: Es wird das Teilproblem ausgewählt, das die beste untere Schranke hat (Heap/Priority Queue, Bestensuche).
- *Diving*: Solange keine zulässige Lösung existiert, wird das Depth-First-Verfahren angewandt und danach zum Best-First-Verfahren gewechselt.

Ist ein kostengünstiger Knoten an einer Stelle des Suchbaumes gefunden worden, der bisher aber nur eine Teillösung des Problems darstellt, muss er mittels der *Branch-Operation* entsprechend verzweigt, seine neuen Knoten zum Suchbaum hinzugefügt und die neuen Knoten aktiv gesetzt werden. Für eine solche Expandierung eines Teilproblems stehen verschiedene Branching-Strategien bereit:

- *Maximal Fractional*: Branching auf der Variablen, deren Wert möglichst nahe 0.5 ist.
- *Maximal Objective*: Branching auf der Variablen mit dem höchsten Zielfunktionswert.
- *Maximal Fractional Coefficient*: Kombination der ersten beiden Verfahren, d. h. Branching auf der Variablen, die möglichst nahe 0.5 liegt und den höchsten Zielfunktionswert besitzt.
- *Maximal Non Zero*: Branching auf der Variablen, die nicht null ist und den höchsten Zielfunktionswert besitzt.
- *Maximal Binary*: Branching auf der Variablen, welche möglichst in der Nähe von 1.0 ist (nur für binäre Entscheidungsprobleme).

Der Wert der oberen Schranke reduziert sich solange, bis der kostengünstigste Wert erreicht ist (siehe Abbildung 4.2 (a)). Durch die permanente Verringerung der oberen Schranke besteht der Suchbaum am Ende, wenn keine Knoten mehr aktiv sind, nur noch aus denjenigen Knoten, welche den optimalen Lösungsweg und deren Reihenfolge beschreiben.

Abb. 4. 2

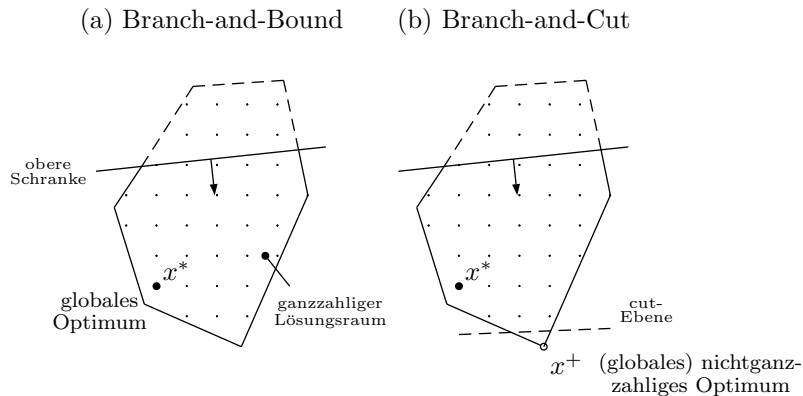


Abbildung 4.2: Exakte Suchverfahren

Branch-and-Cut (BnC)

Die Methode Branch-and-Cut⁵ ist eine Erweiterung des ursprünglichen Branch-and-Bound-Verfahrens. Beim Branch-and-Cut-Verfahren wird das Kriterium, dass eine gefundene Lösung ganzzahlig sein muss, unterdrückt, indem die Nebenbedingung (4.5) von Seite 24 in die Nebenbedingung (4.4) verändert wird (*LP-Relaxation*⁶). Aus dem ILP entsteht ein LP mit der optimalen Lösung x^* . Ist x^* ganzzahlig, so ist x^* auch die optimale Lösung für das ILP und die Suche nach dem globalen Optimum gilt als abgeschlossen. Andernfalls kann x^* durch eine Hyperebene (*cut-Ebene*) von den ganzzahligen Lösungen des ILPs getrennt werden (siehe Abbildung 4.2 (b)), d. h. es gibt einen Vektor d und eine Zahl f , so dass

$$d^T x^* < f \quad \text{und} \quad d^T x \geq f \quad \text{für alle ganzzahligen Lösungen } x.$$

Diese neue Ungleichung kann nun zur LP-Relaxierung hinzugefügt werden, so dass eine Beschneidung des zulässigen Lösungspolytops erfolgt. Danach wird die Suche nach einer neuen Lösung mit Hilfe des Suchbaumes und Anwendung des BnB-Verfahrens fortgesetzt. Es kann gezeigt werden, dass auf Grund der Endlichkeit der Knotenmenge und damit auch des Suchraumes dieses Verfahren terminiert. Das Problem des Verfahrens ist, dass es i. A. schwierig ist, eine trennende Hyperebene zu finden.

4.1.2 Probleme und Lösungsansätze

Die im vorangehenden Abschnitt 4.1.1 beschriebenen Methoden Branch-and-Bound und Branch-and-Cut können zur Bestimmung einer exakten Lösung

⁵ vorgestellt in [RLPT01]

⁶ Relaxation bedeutet beim COP die Entfernung von Nebenbedingungen des COPs, so dass es einfacher lösbar wird.

der nachfolgenden Probleme genutzt werden. Möglich wird dies durch die spezielle Struktur der mathematischen Formulierungen, welche zumeist auf sich gegenseitig ausschließenden Mengen beruhen. Im weiteren Verlauf werden nun verschiedene Beschreibungen für das TSP und das VRP angegeben, welche sowohl Ansätze zur Erzeugung von Branching- als auch Bounding-Operatoren bieten⁷.

Nonlinear Generalized Assignment Problem (NGAP)

Durch eine, im Gegensatz zu den Gleichungen (3.7) - (3.14) etwas andere mathematische Formulierung lässt sich das VRP als Nonlinear Generalized Assignment Problem formulieren. Dabei werden die einzelnen Verbraucher jeweils einem Fahrzeug zugewiesen:

$$(NGAP) \quad \min \sum_k f(y_k) \tag{4.8}$$

u. B. d. N.

$$\sum_i a_i y_{ik} \leq b \quad \forall k = 1, 2, \dots, v \tag{4.9}$$

$$\sum_k y_{0k} = v \tag{4.10}$$

$$\sum_k y_{ik} = 1 \quad \forall i = 1, 2, \dots, n \tag{4.11}$$

$$y_{ik} \in \{0, 1\} \quad \forall i = 1, 2, \dots, n, k = 1, 2, \dots, v \tag{4.12}$$

$$a_i \in \mathbb{R} \quad \forall i = 1, 2, \dots, n \tag{4.13}$$

$$b \in \mathbb{R} \tag{4.14}$$

mit

$$y_{ik} = \begin{cases} 1, & \text{falls Fahrzeug } k \text{ besucht Verbraucher } i \\ 0, & \text{sonst} \end{cases}$$

$$y_k = (y_{0k}, \dots, y_{nk})$$

a_i – Gewicht des Verbraucherkonus i

b – maximales Gewicht eines Verbraucherkonus

$f(y_k)$ – Kosten einer optimalen TSP-Route durch die Verbraucher

$$N(y_k) = \{i \mid y_{ik} = 1\}$$

Als Darstellung der Funktion $f(y_k)$ ergibt sich

$$f(y_k) = \min \sum_{ij} c_{ij} x_{ijk} \tag{4.15}$$

u. B. d. N.

$$\sum_i x_{ijk} = y_{jk} \quad \forall j = 0, \dots, n \tag{4.16}$$

$$\sum_j x_{ijk} = y_{ik} \quad \forall i = 0, \dots, n \tag{4.17}$$

$$\sum_{ij \in S \times S} x_{ijk} \leq |S| - 1 \quad \forall S \subseteq N(y_k), 2 \leq |S| \leq n \tag{4.18}$$

$$x_{ijk} \in \{0, 1\} \quad \forall i = 0, \dots, n, j = 0, 1, \dots, n \tag{4.19}$$

⁷ Genauere Beschreibungen dazu sind in [Fis95, 10 ff.] zu finden.

mit

$$x_{ijk} = \begin{cases} 1, & \text{falls Fahrzeug } k \text{ direkt von } i \text{ nach } j \text{ fährt} \\ 0, & \text{sonst} \end{cases}$$

Diese Formulierung des Problem (NGAP) ist zwar mathematisch exakt, allerdings besteht ein sehr hoher Berechnungsaufwand für die Funktion $f(y_k)$. Eine Ersetzung dieser durch die *Lineare Approximation*

$$f_{LA}(y_k) = \sum_i d_i y_{ik}$$

ist daher ratsam. Dadurch entsteht ein *Linear Generalized Assignment Problem (LGAP)*, welches z. B. mittels Methoden der *Lagrange Relaxation* gelöst werden kann.

Set Partitioning Problem

Eine weitere Möglichkeit, eine optimale Lösung für das VRP zu bestimmen, stellt die Anwendung der *Set-Partitioning-Methode* dar. Dieser liegt eine Menge S aus n Verbrauchern und eine Ansammlung C von Teilmengen S_1, S_2, \dots, S_m dieser Menge S mit $S_i \subseteq S, S = \bigcup_{i=1}^m S_i$ zu Grunde. Beim VRP bilden diese Teilmengen die Menge aller möglichen Fahrzeugrouten, wobei jeder Teilmenge die Kosten $c_{S_i} = c(S_i)$ für die TSP-Tour der Menge S_i zugeordnet werden:

$$c_{S_i} = \begin{cases} c(S_i), & \text{falls } \sum_{j \in S_i} d_j \leq C_{max} \\ \infty, & \text{sonst} \end{cases}$$

Gesucht sind die v Teilmengen S_{i_1}, \dots, S_{i_v} von S mit den minimalen Kosten und $\bigcup_i S_i = S, S_{i_m} \cap S_{i_n} = \emptyset, \forall m, n = 1, 2, \dots, v, m \neq n$. Es ergibt sich das folgende COP:

$$(SPP) \quad \min \sum_{i=1}^m c_{S_i} y_i \tag{4.20}$$

u. B. d. N.

$$\sum_{i=1}^m y_i = v \tag{4.21}$$

$$\sum_{j=1}^m \sum_{k=1}^v x_{ij} y_j = 1 \quad \text{für } i = 1, 2, \dots, n \tag{4.22}$$

$$y_j \in \{0, 1\} \quad \text{für } j = 1, 2, \dots, n \tag{4.23}$$

mit

$$y_i = \begin{cases} 1, & \text{falls die Tour } i \text{ gewählt wurde} \\ 0, & \text{sonst} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{falls Verbraucher } i \text{ in Tour } j \text{ enthalten} \\ 0, & \text{sonst} \end{cases}$$

Das Set-Partitioning ist insbesondere für kleinere Probleminstanzen, bei denen sich nur wenige mögliche Fahrzeugrouten bilden lassen, effektiv einsetzbar.

Polyhedral Combinatoric

Ursprünglich wurde der Ansatz, welcher auf *Polyhedral-Combinatoric-Verfahren* beruht, nur zur Lösung des TSP verwendet. Durch einfache Erweiterungen kann es allerdings auch für das VRP angewandt werden. In Anlehnung an ein *Flussproblem* lässt sich das VRP wie folgt beschreiben:

$$(PC) \quad \min \sum_{i < j} c_{ij} x_{ij} \tag{4.24}$$

u. B. d. N.

$$\sum_{j < i} x_{ij} + \sum_{j > i} x_{ij} = 2 \quad \text{für } i = 1, 2, \dots, n \tag{4.25}$$

$$\sum_{i,j \in S \times S} x_{ij} \leq |S| - r(S) \quad \forall S \subset \{1, 2, \dots, n\},$$

$$2 \leq |S| \leq n - 1 \tag{4.26}$$

$$x_{ij} \in \{0, 1\} \quad \text{für } 1 \leq i < j \leq n \tag{4.27}$$

$$\sum_j x_{0j} = 2v \tag{4.28}$$

$$x_{0j} \in \{0, 1, 2\} \quad \text{für } j = 1, 2, \dots, n \tag{4.29}$$

mit

$$r(S) = \left\lceil \frac{\sum_{i \in S} d_i}{C_{max}} \right\rceil \tag{4.30}$$

Für das TSP entfallen die Gleichungen (4.28) sowie (4.29) und die Menge $r(S)$ wird stets auf 1 gesetzt.

v-Matching-Relaxation

Durch Hinzunahme weiterer Ungleichungen zum Polyhedral-Combinatoric-Ansatz und Dualisierung entsteht das *v-Matching-Problem*. Die Gleichungen (4.24) - (4.26) werden um folgende Beziehungen erweitert:

$$\sum_{l=0}^k \sum_{i,j \in W_l} x_{ij} \leq \sum_{l=0}^k |W_l| - \left[\frac{1}{2} \sum_{l=1}^k V(W_l) + V(W_l - W_0) + V(W_l \cap W_0) \right] \tag{4.31}$$

mit

$$W_l \subseteq \{1, 2, \dots, n\}, \quad l = 0, 1, \dots, k$$

$$|W_l - W_0| \geq 1 \quad \text{für } l = 1, 2, \dots, k$$

$$|W_l \cap W_0| \geq 1 \quad \text{für } l = 1, 2, \dots, k$$

$$|W_l \cap W_{l'}| = 1 \quad \text{für } 1 \leq l \leq l' \leq k$$

Die Spezialisierung $d_i = 1 \forall i$ ermöglicht die Formulierung eines TSP.

Shortest-Path-Relaxation

Die *Shortest-Path-Relaxation* kann ebenfalls zur Bestimmung von Schranken für das TSP und das VRP genutzt werden. Hierfür wird ein neuer Graph G_2 mit den Knoten (i, q) für $i = 1, 2, \dots, n$ und $q = 0, 1, \dots, v$ definiert. Die Verbindung der Kanten (i, q) und $(j, q + d_j)$ wird mit den Kosten c_{ij} versehen. Des Weiteren sei festgelegt, dass

- $l(i, q)$ – Länge des kürzesten Weges von $(0, 0)$ nach (i, q)
- $F(i, q) = l(i, q) + c_{i0}$ – untere Schranke für die Kosten einer Lieferung von q Einheiten zum Verbraucher i
- $v(i, q, k)$ – minimale Kosten von k Touren mit Fracht q zu verschiedenen letzten Verbrauchern aus $\{1, 2, \dots, i\}$

Der Wert für $v(i, q, k)$ kann durch die Rekursion

$$v(i, q, k) = \min\{v(i-1, q, k); \min_{q'}[v(i-1, q-q', k-1) + F(i, q')]\} \quad (4.32)$$

berechnet werden. $v(n, \sum_i d_i, v)$ ergibt sich dann als untere Schranke für das VRP. Diese stellt eine zulässige Lösung dar. Allerdings ist in Gleichung (4.32) der Fall von Mehrfachlieferungen an einen Kunden erlaubt. Um dies zu verbieten, können *Lagrange Penalties* für die Kunden mit verletzenden Lieferbedingungen eingeführt werden.

v-tree-Relaxation

Eines der im nachfolgenden Kapitel 4.2.1 etwas genauer beschriebenen Verfahren, die Heuristik von Christofides, beruht auf einem einzelnen Spannbaum. Die Verallgemeinerung dessen führt zu so genannten v -Bäumen (*v-trees*), welche zur Bestimmung einer unteren Schranke für das VRP genutzt werden können⁸. Angenommen es seien ein Graph mit $n + 1$ Knoten und v Transportfahrzeuge gegeben, dann lässt sich ein v -tree mit genau $n + v$ Kanten so konstruieren, dass er den Graphen aufspannt. Das VRP kann dann als ein kostenminimaler, knotengradbeschränkter v -tree mit Nebenbedingungen modelliert werden. Gebe $x_{ij} \in \{0, 1\}$ an, ob die Kante (i, j) in der Lösung vorkommt, so entstehen zulässige Lösungsvektoren

$$x = (x_{01}, x_{02}, \dots, x_{0n}, x_{12}, \dots, x_{n-2, n-1}, x_{n-1, n}),$$

welche die Menge

$$X = \{x \mid x \in \{0, 1\} \text{ und bildet einen } v\text{-tree mit } \sum_{i=1}^n x_{0i} = 2v\}$$

⁸ siehe [CMT81]

erzeugen. Die Formulierung des VRP lautet dann wie folgt:

$$(\text{vTree}) \quad \min \sum_{e \in E} c_e x_e \quad (4.33)$$

u. B. d. N.

$$\sum_{e \in E \setminus I} x_e = 2v \quad (4.34)$$

$$\sum_{e \in E} x_e = 2 \quad \forall i \in N \quad (4.35)$$

$$\sum_{\substack{e \in E \\ i \in S, j \notin S}} x_e \geq 2r(S) \quad \forall S \subset N, |S| > 1 \quad (4.36)$$

$$x_e \in \{0, 1, 2\} \quad \forall e \in E \quad (4.37)$$

$$0 \leq x_e \leq 1 \quad \forall e = \{i, j\} \in E, i, j \neq 0 \quad (4.38)$$

$$0 \leq x_e \leq 2 \quad \forall e = \{0, i\} \in E \quad (4.39)$$

mit

$$r(S) = \left\lceil \frac{\sum_{i \in S} d_i}{C_{\max}} \right\rceil \quad (4.40)$$

Fisher⁹ gibt des Weiteren eine Strategie an, wie eine Lagrange-Relaxation mit zusätzlichen Nebenbedingungen durchgeführt werden kann. Er beschreibt, wie durch so genannte *Lagrange Multipliers* die Kosten im Falle einer Verletzung zu erhöhen sind.

Dekompositionsverfahren

Eine optimale Lösung für das MDVRP kann beispielsweise durch ein *Dekompositionsverfahren*, etwa dem von *Dantzig und Wolfe*¹⁰, berechnet werden. Dabei wird das zu Grunde liegende Hauptproblem in mehrere Teilprobleme zerlegt. Möglich wird dies durch die spezielle Struktur der meisten LP, die oftmals aus mehreren eigenständigen Planungsaufgaben zusammengesetzt sind. Ein Teilproblem selbst besteht aus wenigen Gleichungen oder Ungleichungen. Untereinander sind die Teilprobleme meist nicht miteinander verknüpft.

Beim MDVRP könnte demnach erst einmal eine Zuordnung der einzelnen Verbraucher zu den verschiedenen Lagern stattfinden, um eine Zerlegung in zu lösende Teilprobleme zu erreichen. Die Teilprobleme bestehen dann jeweils aus einem Lager und einer bestimmten Anzahl an Verbrauchern. Die anschließende Lösung der Teilprobleme besteht dann in der Suche nach einer optimalen Zuordnung der Verbraucher zu den Transportfahrzeugen des zugehörigen Lagers.

⁹ siehe [Fis95]

¹⁰ siehe [DDSS95]

Dynamic Programming

Die Methoden des Dynamic Programming, auch *Dynamic Optimization* genannt, kommen meist bei der Lösung von TSP und VRP mit Zeitfenstern zum Einsatz¹¹. Die Grundidee beruht ebenfalls auf der Theorie der Dekompositionsverfahren. Ein Problem wird, wie bei einer Divide-and-Conquer-Strategie, in mehrere Teilprobleme P_i zerlegt. Allerdings sind beim Dynamic Programming die Teilprobleme voneinander abhängig. Diese einzelnen Teilprobleme werden dann jeweils gelöst und deren Ergebnis E_i gespeichert, so dass E_i zur Lösung größerer Probleme verwendet werden kann. Dem Verfahren liegt das sogenannte Optimalitätsprinzip zu Grunde. Dieses besagt, dass in einer optimalen Folge von Entscheidungen auch jede Teilfolge in demselben Sinne optimal ist. Nur wenn dieses Prinzip für ein Problem gilt, kann Dynamic Programming angewandt und eine exakte Lösung gefunden werden.

Für die Einbeziehung von Zeitfenstern ist folgende Formulierung nützlich: die Funktion $F(S, i, t)$ gebe die niedrigsten Kosten eines Pfades an, welcher im Knoten 0 beginnt, alle Verbraucher in $S \subseteq N$ besucht, im Verbraucher $i \in S$ endet und zum Zeitpunkt t bereit ist, den Verbraucher i zu beliefern. Daraus ergibt sich ein zweidimensionales Label $(t, F(S, i, t)) \in \mathbb{R}^2$, $e_i \leq t \leq l_i$, bestehend aus einem Zeitpunkt und den zugehörigen Kosten, welche sich durch folgende Rekursion bestimmen lassen:

$$\begin{aligned}
 F(S, j, t) = \min_{(i,j) \in E} \{ & F(S - \{j\}, i, t') + c_{ij} \mid i \in S - \{j\}, \\
 & t' \leq t - t_{ij}, a_i \leq t' \leq b_i \} \\
 \forall S \subseteq N, j \in S \text{ und } & a_j \leq t \leq b_j
 \end{aligned} \tag{4.41}$$

wobei

$$F(S, j, t) = \begin{cases} F(S, j, a_j), & \text{falls } t \leq a_j \\ \infty, & \text{falls } b_j \leq t \end{cases} \tag{4.42}$$

Rekursionsbeginn:

$$F(\{j\}, j, \max\{a_j, a_0 + t_{0,j}\}) = \begin{cases} c_{0,j} & \text{falls } (0, j) \in E \\ \infty, & \text{sonst} \end{cases} \tag{4.43}$$

Für das TSP mit Zeitfenstern (TSPTW) ergibt sich dann folgende optimale Lösung:

$$\min_{(i,0) \in E} \max_{a_i \leq t \leq b_i} \{ F(N, i, t) + c_{i,0} \mid i \in N, t \leq b_0 - t_{i,0} \}$$

¹¹ siehe [Chr85]

4.2 Heuristische Verfahren (Heuristiken)

Im Gegensatz zu den exakten Verfahren besitzen heuristische Verfahren eine polynominale Zeitkomplexität $O(n^r)$. Dafür muss allerdings in Kauf genommen werden, dass nur die Bestimmung einer *suboptimalen Lösung* möglich ist. Häufig wird bei den Heuristiken eine Unterscheidung zwischen *Eröffnungsverfahren* zur Konstruktion einer zulässigen Anfangslösung und *Verbesserungsverfahren* zur schrittweisen Annäherung der Anfangslösung an das lokale Optimum getroffen. Im Gegenzug existieren aber auch *Gesamtschrittverfahren*, welche von vornherein eine sehr gute zulässige (sub-)optimale Lösung liefern. Ein Beispiel dafür ist der *Patching-Algorithmus*¹².

4.2.1 Eröffnungsverfahren

Für die Bestimmung einer einzelnen TSP-Tour kann folgendes Verfahren verwendet werden (siehe Abbildung 4.3):

- *Heuristik von Christofides*: Bestimmung des Minimalgerüsts G' eines Graphen G . Erzeugung eines minimalen Matchings M für die Knoten mit ungeraden Grad von G' . Konstruktion einer Eulertour¹³ aus $G' \cup M$ und daraus Bildung einer TSP-Tour, z. B. durch Tiefensuche.

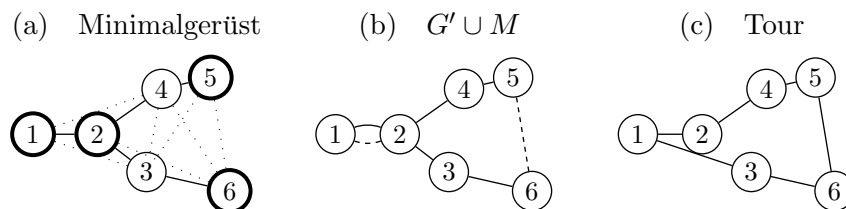


Abb. 4.3

Abbildung 4.3: Christofides-Heuristik

Bei den Eröffnungsverfahren, welche zur Lösung des VRP verwendet werden, findet für gewöhnlich eine Unterscheidung zwischen Sukzessiv- und Parallelverfahren statt.

Sukzessivverfahren

Sukzessivverfahren bestimmen durch Lösung des Zuordnungsproblems zunächst alle für die Tourenbildung in Frage kommenden Teilmengen und su-

¹² siehe [NM93, S. 456 ff.]

¹³ Eine Eulertour ist eine Rundreise, die, im Gegensatz zum TSP, jeden Knoten *mindestens einmal* enthalten muss. Sie existiert nur, wenn der Grad eines jeden Knoten des Graphen gerade ist.

chen danach eine optimale Reihenfolge innerhalb einer Teilmenge. Diese Methodik wird auch „cluster first – route second“ genannt. Die bekanntesten Vertreter dieser Verfahren sind

- der *Sweep-Algorithmus* von Gillett und Miller¹⁴ (siehe Abbildung 4.4 (b)),
- das *Spacefilling-Curves-Verfahren* von Bartholdi und Platzman¹⁵ sowie
- das *Clustering* (siehe Abbildung 4.4 (e)).

Allerdings beziehen diese Verfahren geographische Informationen bei der Gebietseinteilung (Clusterbildung) mit ein, wodurch sich ihre Anwendbarkeit auf einen speziellen Problemkreis einschränkt.

Die „route first – cluster second“-Verfahren sind ebenfalls Sukzessivverfahren, arbeiten aber nach der umgekehrten Strategie, d. h. sie bestimmen zuerst möglichst große Touren und suchen dann nach einer optimalen Aufteilung dieser Touren. Beim

- *Giant-Tour-Verfahren*

wird eine große Rundtour (*Giant-Tour*) gebildet, welche einmal durch alle Verbraucher führt (siehe Abbildung 4.4 (f)). Anschließend findet eine Zerlegung dieser Giant-Tour unter Beachtung von Kapazitäts- und Zeitschranken statt.

Parallelverfahren

Parallelverfahren versuchen das Zuordnungs- und das Reihenfolgeproblem annähernd gleichzeitig zu lösen. Sie können sowohl zur Bildung einzelner Touren (TSP) als auch mehrerer Touren (VRP) verwendet werden. Das wesentliche Kennzeichen der Parallelverfahren ist die *sukzessive Einbeziehung von Knoten oder Kanten* unter Nutzung eines *Greedy-Suchverfahrens*. Ziel ist die Bildung eines Hamiltonschen Kreises K_H , falls ein TSP zu lösen ist, oder mehrerer Hamiltonscher Kreise als mögliches Ergebnis eines VRP. Bei beiden Problem sind vorhandene Kapazitäts- und Zeitrestriktionen stets zu beachten. Tritt eine Verletzung dieser Restriktionen auf, muss im Falle eines VRP eine weitere Tour mit einem neuen Transportfahrzeug gestartet werden. Liegt hingegen ein TSP zu Grunde, bewirkt eine Restriktionsverletzung, dass eine ungültige Lösung entsteht und es ist eine neue TSP-Tour zu bestimmen.

Die meisten der Heuristiken besitzen eine Laufzeit von $O(n^2)$. Im Wesentlichen gibt es bei heuristischen Verfahren zwei grundlegende Verfahrenstypen:

¹⁴ siehe [GM74]

¹⁵ siehe [BP82]

Abb. 4. 4

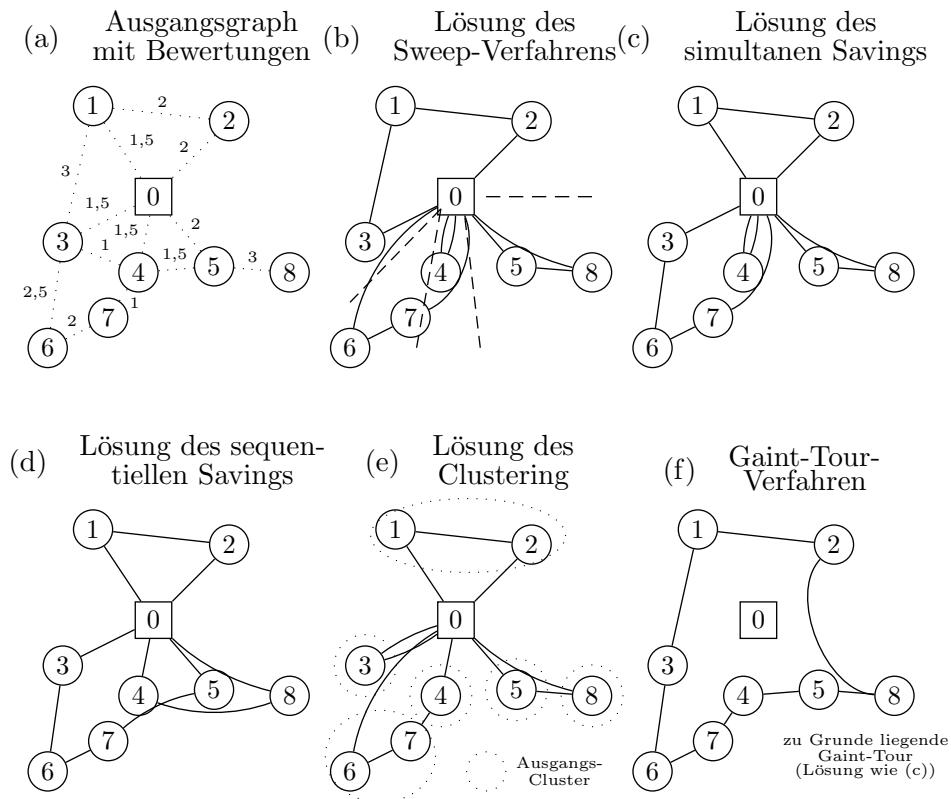


Abbildung 4.4: Auswahl heuristischer Tourenbildungsverfahren

- *Neighbour-Verfahren*: Iterative Hinzunahme eines weiteren Knoten bzw. einer weiteren Kante nach bestimmten Kriterien zu einem bestehenden Weg bis alle Knoten bzw. Kanten angefügt wurden. Dann Verbindung des zuletzt besuchten Knotens mit dem Startknoten der Tour.
- *Insertion-Verfahren*: Iterative Entfernung einer Kante eines bestehenden Kreises durch Hinzunahme eines weiteren Knoten bzw. einer weiteren Kante nach bestimmten Kriterien bis alle Knoten bzw. Kanten eingefügt wurden.

Die effektivsten Neighbour-Verfahren sind die Verfahren

- *Nearest-Neighbour*: Der hinzuzunehmende Knoten ist der noch nicht besuchte Knoten mit dem geringsten Kostenzuwachs.
- *Farthest-Neighbour*: Als nächster Knoten wird der noch nicht besuchte Knoten mit dem höchsten Kostenzuwachs hinzugenommen.

Zur Klasse der Insertion-Verfahren gehören u. a. folgende Verfahren¹⁶:

¹⁶ Für genauere Angaben zu den angegebenen Verfahren siehe [Rei94].

- *Nearest-Insertion*: Hinzugefügt wird der noch nicht besuchte Knoten, dessen Hinzunahme die geringste Erhöhung der bisherigen Kosten bewirkt¹⁷.
- *Furthest-Insertion*: wie Nearest-Insertion, nur dass bei Hinzunahme eines Knoten eine größtmögliche Erhöhung der Kosten erfolgen soll.
- *Sequentielles Clarke-und-Write-Saving*: Bildung aller Pendeltouren zwischen dem Lager und den Verbrauchern $T_i = \{(0, i), (i, 0)\}$, $\forall i \in V$. Beginn einer neuen Tour ausgehend von den beiden noch nicht betrachteten Knoten, die das meiste Ersparnis $s_{ij} = c_{0i} + c_{0j} - c_{ij}$ liefern. Hinzunahme eines weiteren Knoten l , der minimalen Kostenzuwachs verursacht $\min\{c_{il}, c_{lj}\}$ (siehe Abbildung 4.4 (d)).
- *Simultanes Clarke-und-Write-Saving*: ähnlich dem sequentiellen Clarke-und-Write-Saving, nur dass hier versucht wird den Anfangspunkt der einen Tour mit dem Endpunkt einer anderen Tour gewinnbringend zu verschmelzen (siehe Abbildung 4.4 (c)).
- *Farthest-Insertion*: Auswahl des noch nicht besuchten Knoten, dessen minimale Kosten bei einer Hinzunahme zu den bereits betrachteten Knoten maximal unter allen noch nicht betrachteten Knoten sind.
- *Farthest-Insertion (2)*: wie Farthest-Insertion, nur das zu den bereits betrachteten Knoten der noch nicht betrachtete Knoten hinzugefügt wird, dessen maximale Kosten bei einer Hinzunahme minimal sind.

Im Zusammenhang mit der Einbeziehung von Rückgabeorten beim VRP (siehe Abschnitt 3.3.6) seien hier noch zwei weitere Insertion-Verfahren von Casco et al. erwähnt, die auf dem Savingsverfahren von Clarke und Wright beruhen¹⁸:

- *Stop-Based-Insertion*: Hinzufügung von Rückgabeorten zu einer bestehenden Tour erst dann erlaubt, wenn nachfolgend nur noch wenige Abladeorte durch den Transporter zu besuchen sind.
- *Load-Based-Insertion*: wie Stop-Based-Insertion, allerdings erfolgt eine Betrachtung der auf dem Transporter noch verbleibenden Ablademenge.

4.2.2 Verbesserungsverfahren

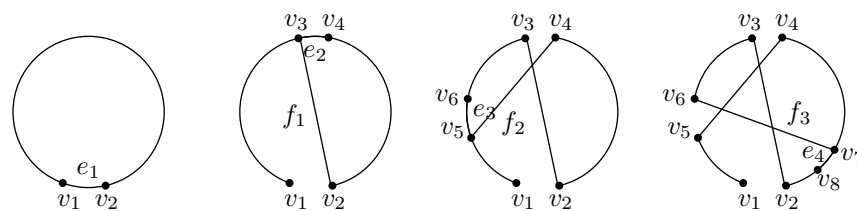
Verbesserungsverfahren basieren auf einer unvollständigen Abarbeitung des ursprünglichen Suchraumes. Sie versuchen in der Regel, durch *lokale Suche* eine bessere Lösung zu erzielen. Allerdings bedeutet dies auch, dass die gefundene Lösung nicht zwingend die optimale Tour sein muss. Es wird lediglich eine *suboptimale Lösung des Problemes* angestrebt. Die zu Grunde liegende Verbesserungsheuristik wird dabei so lange auf eine Tour angewandt, bis sie keine Verbesserung mehr erzielen kann. Ausgangspunkt für eine lokale

¹⁷ siehe [RSL77]

¹⁸ siehe [AG88, S.127-147]

r-opt-Verfahren, welches von Lin und Kernighan²⁰ entwickelt wurde. Es bestimmt in jedem Schritt die Menge der auszutauschenden Kanten des zu Grunde liegenden Hamiltonschen Kreises. Durch Entfernung einer Kante entsteht aus einer Tour ein Weg. Das eine Ende dieses Weges wird versucht so mit einem seiner inneren Knoten zu verbinden, dass eine andere Kante gelöscht werden kann. Dadurch entsteht wiederum ein Weg, der als Ausgangspunkt für eine weitere solche Untersuchung genutzt wird (siehe Abbildung 4.6). Als Grundidee dient die Überlegung, dass ein $(k+1)$ -opt-Tausch günstiger sein könnte als ein k -opt-Tausch. Ein solcher Schritt der Prozedur wird folglich so lange wiederholt, wie die Gewinnsumme (gelöschte minus eingefügte Kantengewichte) positiv ist und noch nicht behandelte Kanten existieren. Wurde während der Iteration eine bessere Tour gefunden, so ersetzt diese die aktuelle Tour und der nächste Knoten wird untersucht. Das Verfahren endet, wenn alle Knoten der Ausgangstour untersucht worden sind. Für gewöhnlich findet das Verfahren fast die globaloptimale Lösung²¹.

Abb. 4. 6



e_i - Kanten der Ausgangstour
 f_i - Kanten, die nicht zur Ausgangstour gehören

Abbildung 4.6: Einige Schritte des Lin-Kernighan-Algorithmuses

Or-opt Verfahren

Or²² schlägt eine Modifikation des Verfahrens 3-opt vor, wodurch das so genannte *Or-opt-Verfahren*²³ entsteht. Der Vorteil von Or-opt ist, dass die Anzahl der Kantenvertauschungen gegenüber 3-opt wesentlich verringert werden, bei fast der gleichen Güte der Näherungslösung. Das Verfahren hat eine Zeitkomplexität von $O(3n^2)$. Es prüft der Reihe nach für s benachbarte Knoten des aktuellen Hamiltonschen Kreises, ob diese s Knoten zwischen zwei anderen Knoten so eingefügt werden können, dass sich ein kürzerer, kostengünstiger Hamiltonscher Kreis ergibt (siehe Abbildung 4.7). Im ersten Durchlauf wird $s = 3$, dann $s = 2$ und schließlich $s = 1$ gesetzt.

²⁰ siehe [LK73]

²¹ siehe [LLRS85]

²² siehe [Or76]

²³ Verallgemeinert wird das Verfahren auch *Relokation* genannt.

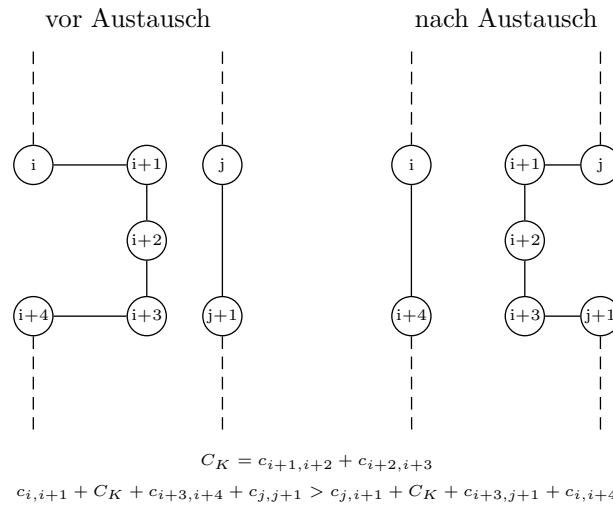


Abb. 4.7

Abbildung 4.7: Or-opt bei $s = 3$

4.2.3 Kombination heuristischer Verfahren

Numerische Tests ergaben, dass eine Verfahrenskombination aus Eröffnungsverfahren, gefolgt von dem Verbesserungsverfahren 2-opt und schließlich 3-opt oder Or-opt Hamiltonsche Kreise liefert, deren Kosten selten mehr als 2 bis 3% vom Optimum entfernt sind²⁴. Weitere Annäherungen an das Optimum sind häufig durch wiederholte Auswahl unterschiedlicher Startknoten und anschließender Anwendung eines der obigen Verfahren zu erzielen.

4.2.4 Moderne heuristische Verfahren

Suchtechniken

Die beiden nachfolgend vorgestellten Verfahren erlauben es Veränderungen vorzunehmen, auch wenn diese nicht zu Verbesserungen einer Tour führen. Dadurch soll erreicht werden, dass die lokalen Optima, welche die Leistung der im vorherigen Abschnitt beschriebenen Heuristiken begrenzen, verlassen werden können.

Ausgehend von einer zulässigen Startlösung S wird um diese eine gewisse Nachbarschaft $N(S)$ konstruiert, innerhalb welcher eine bessere Alternative S' gesucht wird. Das Verfahren wird dann mit S' iteriert.

Üblicherweise bedeutet Nachbarschaft das Austauschen von n_1 Verbrauchern einer gegebenen Tour mit n_2 Verbrauchern einer anderen Tour, wobei n_1 und n_2

²⁴ siehe [LLRS85]

nach oben beschränkt sind. Zumeist wird nur der Austausch einzelner Kunden erlaubt, also $n_1 \leq 1$ und $n_2 \leq 1$.

Als Abbruchkriterien wird meist eine maximale Iterationszahl angegeben. Allerdings kann die Iteration aber auch vorzeitig abgebrochen werden, wenn beispielsweise

- keine Verbesserungen mehr innerhalb der letzten k Iterationsschritte erfolgten,
- die zulässige Nachbarschaft leer ist oder
- die gefundene Lösung innerhalb eines angegebenen Toleranzbereiches liegt.

Tabu Search

Beim Tabu-Search-Verfahren wird innerhalb jeder Iteration die beste zulässige Alternative innerhalb der Nachbarschaft gesucht und diese ausgewählt²⁵. Innerhalb der nächsten t Iterationen darf diese Auswahl nicht rückgängig gemacht werden²⁶.

Simulated Annealing

Die Nachbarschaft $N(S)$ wird anhand einer vorgegebenen Ordnung durchsucht. Für jede Alternative $S' \in N(S)$ wird die Kostendifferenz zu S gebildet:

$$\Delta_c = c(S) - c(S').$$

Die Lösung S' wird sofort als neue Startlösung gewählt, wenn $\Delta_c \leq 0$ ist. Im Falle von $\Delta_c > 0$ wird überprüft, ob gilt

$$e^{-\Delta_c/T} \geq \theta$$

mit

$$\theta \in \mathbb{R}, 0 \leq \theta < 1 \text{ und zufällig gewählt}$$

$$T - \text{zeitlicher Fortschritt}$$

Ist diese Ungleichung erfüllt, so ersetzt S' ebenfalls S . Der Parameter T wird nach jeder Iteration verringert. Ist nach n Iterationen keine Verbesserung eingetreten, kann T wieder etwas erhöht werden, um ggf. ein lokales Minima zu verlassen²⁷.

²⁵ siehe [GHL93]

²⁶ Wechselt beispielsweise ein Verbraucher i von der Tour T_n zur Tour T_m , so ist es ihm für die nächsten t Iterationsschritte verboten, zur Tour T_n zurückzuwechseln.

²⁷ Das Simulated Annealing ist u. a. in [AK89] ausführlich beschrieben.

Evolutionäre Algorithmen

Unter dem Begriff evolutionäre Algorithmen sind eine Reihe von *Meta-Heuristiken*, wie z. B.

- *Genetische Algorithmen (GA)*,
- *Evolutionary Programming* und
- *Genetic Programming*

zusammengefasst, welche versuchen die Grundprinzipien der natürlichen Evolution in einfacher Weise nachzuahmen²⁸. Es kommen Mechanismen wie

- die *Selektion* (natürliche Auslese),
- die *Rekombination* (Kreuzung) und
- die *Mutation* (kleine zufällige Veränderungen)

zum Einsatz. Im Unterschied zu den Suchverfahren des Simulated Annealing und Tabu Search wird nicht von einer einzelnen aktuellen Lösung, sondern von einer ganzen Lösungsmenge (*Population*) ausgegangen. Bei der Suche nach einer optimalen Lösung ist somit eine größere Vielfalt vorhanden, wodurch die Suche robuster wird. Es ist nicht mehr so leicht, in einem lokalen Optimum hängen-zubleiben.

²⁸ siehe [Rec73]

5 Zusammenfassung und Ausblick

Eine exakte Lösung für Routing Probleme wie das TSP und das VRP ist i. A. nicht in annehmbarer Zeit zu finden. Ihre zugehörigen Algorithmen besitzen eine exponentielle Laufzeit, d. h. der aktuelle Stand der Technik beschränkt die Größe der lösbaren Probleminstanzen. Dennoch haben auch diese Lösungsansätze ihre Existenzberechtigung, weil sie auf jeden Fall eine optimale Lösung eines Routing Problemes bestimmen können. Diese kann beispielweise dem Nachweis dienen, wie stark die Lösung einer Heuristik vom Optimum abweicht. Allerdings ist keine Verallgemeinerung der dabei getroffenen Aussagen möglich, weil die Güte einer heuristischen Lösung sehr stark vom Aussehen des Graphen abhängt.

Moderne heuristische Verfahren umgehen die Misstände vorheriger Heuristiken. Sie versuchen durch Zulassung schlechterer Lösungen bei der Suche nach einem Optimum, lokalen Extrema zu entkommen.

Die beiden nachfolgenden Abbildungen sollen noch einmal die Ausführungen der vorangehenden Kapitel graphisch verdeutlichen. In Abbildung 5.1 ist eine mögliche Einteilung der Problemstellungen des TSP und des VRP sowie deren zugehörige Problemerkweiterungen dargestellt.

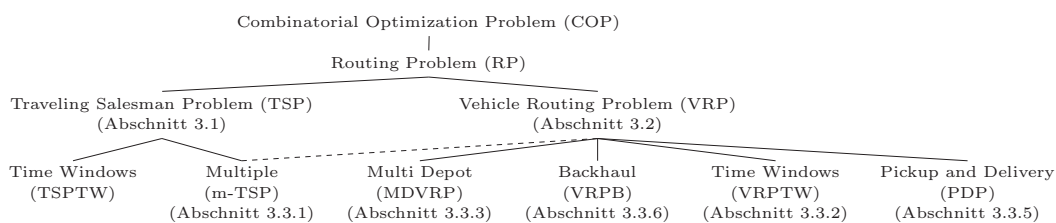


Abb. 5. 1

Abbildung 5.1: Überblick über die Probleme bei der Tourenbildung

Abbildung 5.2 zeigt Lösungsverfahren, welche vorrangig zur Lösung des VRP verwendet werden können. Durch entsprechende Anpassungen sind diese auch für die Lösung des TSP anwendbar.

Die Untersuchung aktuellster Forschungsergebnisse und Entwicklung neuer Implementierungsvarianten, wie beispielsweise dem Einsatz moderner und paralleler Rechentechnik, sollte ein grundlegendes Ziel weiterer Aktivitäten auf

Abb. 5. 2

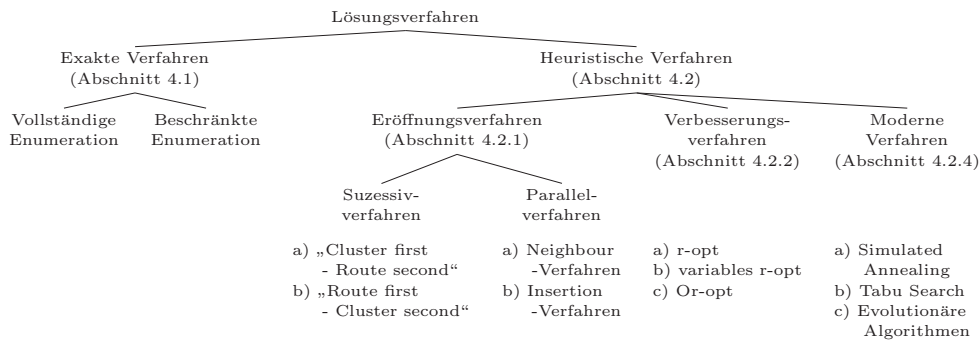


Abbildung 5.2: Überblick über die Lösungsverfahren beim TSP und VRP

diesem Gebiet sein. Erste Ansätze zur Parallelisierung der in dieser Arbeit dargelegten exakten Verfahren werden von Ralphs¹ vorgestellt.

Für die meisten Erweiterungen des klassischen VRP existieren zwar viele gute theoretische Ansätze, aber diese sind teilweise weit von ihrer exakten Berechenbarkeit entfernt. Zudem ist das VRP mit seinen vielfältigen Variationen noch nicht so weit untersucht, dass alle praktischen Anwendungsfälle genau formuliert worden sind. Als Beispiele seien hier

- Transporterverfügbarkeiten zu unterschiedlichen Zeiten,
- gemeinsame Belieferung eines Verbrauchers durch mehrere Transporter und
- Transport verschiedenartiger Güter

genannt.

Diese Ausführungen sollen deutlich machen, dass, bedingt durch die ständigen Erweiterungen der technischen Möglichkeiten und die wachsenden wirtschaftlichen Anforderungen, noch keine Stagnation auf dem Gebiet der Tourenbildung und -optimierung in Sicht ist.

¹ siehe [Ral02]

A Algorithmen

A.1 Exakte Verfahren

Um das Verständnis für die Lösungsalgorithmen des Branch-and-Bound (BnB) bzw. das Branch-and-Cut-Verfahren (BnC) zu erhöhen, sollen zunächst noch einmal einige grundlegende Gesichtspunkte der kombinatorischen Optimierung betrachtet werden. Für detailliertere Aussagen siehe Abschnitt 4.1.

Gegeben sei ein Combinatorial Optimization Problem

$$COP = (D_I, S_P(I)), S_P(I) \subseteq 2^{D_I}$$

bestehend aus:

- D_I - Menge der Instanzen I
- $S_P(I)$ - Menge der Lösungen jeder Instanz $I \in D_I$
- m_P - Bewertungsfunktion $m_P : S_P \rightarrow \mathbb{R}$

Die Lösung $x^* \in S_P(I)$ heißt im Falle einer Minimierung optimale Lösung gdw.

$$m_P(x^*, I) \leq m_P(x, I) \quad \forall x \in S_P(I)$$

Im Sinne des Integer Programming (IP) ausgedrückt, bedeutet dies, dass eine Lösung des folgenden Problemes gesucht ist:

$$\min m_P(x) = c^T x$$

u. B. d. N.

$$Ax \geq b$$

$$x \geq 0$$

$$x \in \mathbb{Z}_+^n$$

mit

$$A \in \mathbb{R}^{m \times n} \quad \text{- Restriktionsmatrix}$$

$$b \in \mathbb{R}^m \quad \text{- Restriktionsvektor}$$

$$c \in \mathbb{R}^n \quad \text{- Kostenvektor}$$

Für das TSP gelte im Speziellen: gesucht ist eine Rundreise durch n Orte, wobei jede Stadt nur einmal besucht werden darf. Es entsteht folgende IP-Formulierung:

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij}$$

u. B. d. N.

$$\begin{aligned} \sum_{i=1}^n x_{ij} &= 1 \quad \forall j \\ \sum_{j=1}^n x_{ij} &= 1 \quad \forall i \\ x_{ij} &\in \{0, 1\} \end{aligned}$$

Zur Elimination von Subtouren können die beiden folgenden Bedingungen genutzt werden:

$$\begin{aligned} \sum_{\{i,j\} \subseteq S} x_{ij} &\leq |S| - 1 \quad \forall S \subset V, |S| \geq 2 \\ |\text{Schnittkanten von } (S, \{x_{ij}\})| &\geq 2 \quad \forall (S, V - S), \emptyset \subset S \subset V \end{aligned}$$

Als optimale Lösung entsteht eine Permutation $\pi(1, 2, \dots, n)$ der Menge $\{1, 2, \dots, n\}$ mit den minimalen Kosten:

$$\sum_{i=1}^{n-1} c_{\pi(i), \pi(i+1)} + c_{\pi(n), \pi(1)}$$

Bei der vollständigen Enumeration müssten alle Permutationen untersucht werden. Dazu kann folgender Algorithmus mit exponentieller Laufzeit verwendet werden:

Alg. A. 1

Algorithmus A.1: Enumeration (TSP)

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: optimale Tour T

$\pi_{start} = (1, 2, \dots, n)$

$\pi_{akt} = \pi_{start}$

$c_{\pi_{best}} = \infty$

$i = 1$

repeat

$c_{\pi_{akt}} = \sum_{j=1}^{n-1} c_{\pi_{akt}(j), \pi_{akt}(j+1)} + c_{\pi_{akt}(n), \pi_{akt}(1)}$ (Bewertung von π_{akt})

if $c_{\pi_{akt}} < c_{\pi_{best}}$

$c_{\pi_{best}} = c_{\pi_{akt}}$

$\pi_{best} = \pi_{akt}$

Vertausche $\pi_{akt}(i)$ und $\pi_{akt}(i + 1)$

$i = (i \bmod (n - 1)) + 1$

until $\pi_{akt} = \pi_{start}$

$T = \pi_{best}$

A.1.1 Branch-and-Bound-Verfahren

Eine andere Möglichkeit eine optimale Lösung zu erhalten, wäre die implizite Enumeration, bei welcher der Suchraum durch einen Suchbaum eingeteilt wird. Aus diesem werden Teilbäume eliminiert, falls dessen Lösungskosten zu hoch sind. Die Basis dafür bildet der Branch-and-Bound-Algorithmus mit den beiden Operationen:

- Bounding: Verwendung oberer und unterer Schranken für Zielfunktionswert
 - Obere Schranke: beste gefundene Lösung
 - Untere Schranke: günstigste Vervollständigung einer partiellen Lösung oder (diskrete) Relaxation des Problemes
 - Falls $\text{untereSchranke} \geq \text{obereSchranke}$ kann Teilbaum eliminiert werden
- Branching: Verzweigung innerhalb des Suchraumes und Betrachtung der disjunkten Teilprobleme

Der nachfolgende Branch-and-Bound-Algorithmus (Algorithmus A.2) ist speziell zur Lösung von Problemen mit binären Zielfunktionsvariablen gedacht, d. h. x_{ij} ist entweder 0 oder 1. In diesem Falle kann für jede der Variablen x_{ij} eine Fixierung eingeführt werden, welche angibt, ob die Variable im betreffenden Zweig des Suchbaumes bereits untersucht wurde. Sind alle Variablen fixiert, kann dieser Ast nicht weiter verzweigt werden. Er enthält entweder die optimale Lösung oder nicht.

Algorithmus A.2: Branch-and-Bound (TSP)

Alg. A. 2

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: optimale Tour T

Schritt 1: Bestimme untere Schranke uS für das Problem

Schritt 2: Bilde erste Problemlösung T_1 mittels Heuristik **or** erzeuge einen zulässigen Zweig des Suchbaumes mittels Tiefensuche
 $T_1 = (\pi(1), \pi(2), \dots, \pi(n))$ (Permutation von V)
 $oS = c(T_1) = \sum_{i=1}^{n-1} c_{\pi(i), \pi(i+1)} + c_{\pi(n), \pi(1)}$ (obere Schranke)
if $oS = uS$
 stop optimale Lösung gefunden: $T = T_1$
else
 Bilde erste Probleminstanz T'_1
 $S = \{T'_1\}$ (Menge der zu untersuchenden Probleminstanzen)

Schritt 3: Berechne optimale Problemlösung (Breitensuche)

while $S \neq \emptyset$
 $T' = \text{Top}(S)$, wobei $T' = (\pi(1), \pi(2), \dots, \pi(n))$
 if \exists nichtfixierte Variable in T'
 Fixiere nichtfixierte Variable in T' beliebig
 Berechne untere Schranke $uS(T')$ (Bounding)
 $uS(T') = \sum_{i=1}^{n-1} c_{\pi(i), \pi(i+1)} + c_{\pi(n), \pi(1)}$

```

if  $uS(T') \leq oS$ 
  if alle Variable in  $T'$  fixiert
     $oS = uS(T')$ 
     $T = T'$ 
  else
    Branching:
    Erzeuge durch Fixierung einer nichtfixierten
    Variable  $i$  von  $T'$  neue Teilprobleme  $T'_1, T'_2, \dots, T'_n$ 
    Füge  $T'_1, T'_2, \dots, T'_n$  in  $S$  ein

```

A.1.2 Branch-and-Cut-Verfahren

Beim Branch-and-Cut-Algorithmus liegt ebenfalls das Prinzip Branch-and-Bound zu Grunde. Allerdings wird zur Berechnung der unteren Schranke die LP-Relaxation verwendet. Im Konkreten heißt dies, dass die Ganzzahligkeitsbedingung $x_{ij} \in \{0, 1\}$ durch die Bedingung $0 \leq x_{ij} \leq 1$ ersetzt wird. Dadurch kann an Stelle von Integer Programming eines der Verfahren des Linear Programming (LP), wie z. B. die Simplex Methode bzw. eine Modifikation dieser, zur Lösung verwendet werden. Dahinter verbirgt sich die Überlegung, dass die Lösung eines nichtganzzahligen Problemes mittels LP einfacher zu finden ist als die eines ganzzahligen Problemes mittels IP.

Grundprinzip der LP-Relaxation ist eine Vergrößerung des zulässigen Lösungsbereiches. Die konvexe Hülle des Polytops, welche durch das IP beschrieben wird,

$$P_{IP} = \text{conv}\{x \in \{0, 1\}^n \mid Ax \leq b\},$$

wird durch ein größeres Polytop,

$$P_{LP} = \text{conv}\{x \in \mathbb{R}_+^n \mid Ax \leq b\},$$

der LP-Relaxation ersetzt. Dieser enthält nach wie vor die optimale Lösung. Die dabei entstehenden nichtganzzahligen optimalen Lösungen, müssen durch so genannte Cut-Ebenen von den ganzzahligen Lösungen abgetrennt werden, wodurch der LP-Polyeder innerhalb eines Suchbaumzweiges immer mehr beschnitten wird. Im weiteren Verlauf des BnC-Verfahrens sind diese Cut-Ebenen und die ganzzahligen Variablen in den Suchbaum aufzunehmen, welche in der Nähe der Cut-Ebenen liegen. Das Problem ist, dass nicht genau gesagt werden kann, auf welcher Seite der Cut-Ebene sich die optimale ganzzahlige Lösung befindet. Das ursprüngliche Problem zerfällt in zwei Teilprobleme:

$$\begin{array}{ll}
 \min c^T x & \min c^T x \\
 \text{u. B. d. N.} & \text{u. B. d. N.} \\
 Ax \geq b & Ax \geq b \\
 x_i \leq \lfloor x_i^* \rfloor & x_i \leq \lfloor x_i^* \rfloor + 1 \\
 x \geq 0 & x \geq 0 \\
 x \in \mathbb{Z}_+^n & x \in \mathbb{Z}_+^n
 \end{array}$$

Ist die optimale Lösung des LP ganzzahlig, so ist auch eine Lösung für das IP gefunden.

Im Falle des TSP sind neben der Entfernung der Bedingung der Ganzzahligkeit auch andere Relaxationen denkbar, wie z. B.:

- Entfernung der Subtour-Elimination
- nur Ausschluss kurzer Subtouren ($|S| \leq 2, 3, \dots$)
- Verbot der berechneten Subtouren iterativ

Dabei ist zu berücksichtigen, dass sich durch diese Relaxationen untere Schranken ergeben können, deren Wert i. A. besser als der Wert der optimalen Lösung ist. Somit muss im Falle einer neuen optimalen Lösung erst deren IP-Gültigkeit geprüft werden, bevor sie als neue aktuell beste Lösung akzeptiert werden kann.

Ein allgemeiner Algorithmus für das Branch-and-Cut-Verfahren kann wie folgt formuliert werden:

Algorithmus A.3: Branch-and-Cut (TSP)

Alg. A. 3

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: optimale Tour T

Schritt 1: Bestimme untere Schranke uS für das Grundproblem

Schritt 2: Bilde erste Problemlösung T_1 mittels Heuristik **or** erzeuge einen zulässigen Zweig des Suchbaumes mittels Tiefensuche

$T_1 = \{\pi(1), \pi(2), \dots, \pi(n)\}$ (Permutation von V)

$oS = c(T_1) = \sum_{i=1}^{n-1} c_{\pi(i), \pi(i+1)} + c_{\pi(n), \pi(1)}$ (obere Schranke)

if $oS = uS$

stop optimale Lösung gefunden: $T = T_1$

else

Bilde erste Problem Instanz T'_1

$S = \{T'_1\}$ (Menge der zu untersuchenden Problem Instanzen)

Erzeuge Menge von Ungleichungen L' , welche für das Grundproblem

gültig

sein sollen (LP-Relaxation)

Schritt 3: Berechne optimale Problemlösung (Breitensuche)

while $S \neq \emptyset$

$T' = Top(S)$, wobei $T' = \{\pi_1, \pi_2, \dots, \pi_n\}$

Bounding:

```

end = false
solution = false
C = L' (Menge der gültigen Ungleichungen dieses Suchbaumzweig)
while end ≠ true
  if ∃  $\hat{T}$  = LP-Lösung von T' unter Beachtung von C
    if  $c(\hat{T}) \leq oS$ 
      if  $\hat{T}$  ist ganzzahlige Problemlösung
        oS =  $c(\hat{T})$ , T = T', end = true
        solution = true
      else
        Separation:
        C' = Menge neuer verletzender Ungleichungen
        if C' ≠ ∅
          C = C ∪ C'
        else
          oS =  $c(\hat{T})$ , T = T', end = true
    else
      end = true
  if solution = false
    Branching:
    Wähle Variable i nach beliebiger Branching-Heuristik
    Erzeuge von T' aus neue Teilprobleme T'_1, T'_2, ..., T'_n unter
    Beachtung von L'
    Füge T'_1, T'_2, ..., T'_n in S ein

```

A.2 Heuristische Verfahren

A.2.1 Eröffnungsverfahren für das TSP und VRP

Die nachfolgend beschriebenen heuristischen Verfahren beziehen sich zumeist auf das TSP. Dies hat den einfachen Grund, dass die Algorithmen kürzer und nach Ansicht des Autors somit verständlicher sind. Allerdings können sämtliche Algorithmen ebenfalls auf das VRP angewendet werden. Dazu müssen sie lediglich um Abfragen, welche sich auf die Kapazitäts- und Zeitbeschränkungen der Fahrzeuge und Verbraucher beziehen, erweitert werden. Im Falle des Eröffnungsverfahrens Nearest Neighbour ist ein Beispiel dafür vorgegeben (siehe Algorithmus A.5).

Nearest-Neighbour-Algorithmen

Der zum aktuell betrachteten Verbraucher am nächstgelegene und damit kostengünstigste Verbraucher wird als Nachfolger in die Tour aufgenommen und bildet den Ausgangspunkt für die weitere Untersuchung. Die Auswahl des ersten Startknotens kann zufällig erfolgen. Algorithmus A.4 soll dieses Prinzip verdeutlichen.

Algorithmus A.4: Nearest Neighbour (TSP)

Alg. A. 4

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij})$, $i, j \in V$ mit $c_{ij} \sim \text{distance}(i, j)$, $i, j = 1, 2, \dots, n$

Ausgabe: Tour T

Schritt 1: Initialisierung:

beliebigen Startknoten $s \in V$ wählen
 $p = s$
 $U = V \setminus \{s\}$ (Menge der unzugeordneten Verbraucher)
 $T = \emptyset$

Schritt 2: Tourenbildung:

$i = 1$
 $d_T = 0$
while $U \neq \emptyset$
 $f = \min\{c_{pf}\}$, $f \in U$
 $T = T \cup (p, f)$
 $U = U \setminus \{f\}$
 $p = f$
 $T = T \cup (f, s)$

Der nachfolgende Algorithmus A.5 ist eine Abwandlung des zuvor betrachteten Nearest-Neighbour-Algorithmuses, um zu zeigen wie dieser auch auf das VRP angewandt werden kann. An dieser Stelle werden allerdings nur die Kapazitätsbeschränkungen der homogenen Transportfahrzeuge berücksichtigt. Des Weiteren ist die im Schritt 3 durchgeführte Tourenprüfung nicht zwingend notwendig, weil ein Transportfahrzeug auch mehrere Touren durchführen könnte. Durch eine Abwandlung des Algorithmuses ist es demnach möglich, ihn so einzusetzen, dass er zur Ermittlung der maximalen Menge von benötigten Transportfahrzeugen dient.

Algorithmus A.5: Nearest Neighbour (VRP)

Alg. A. 5

Eingabe: Verbrauchermenge $V = \{0, 1, \dots, n\}$ (0 – Lager)
 Kostenmatrix $C = (c_{ij})$, $i, j \in V$
 Transportfahrzeugmenge $K = \{v_1, v_2, \dots, v_k\}$
 maximale Transportfahrzeugkapazität C_{max}
 Verbraucherbedarfsmengen $D = \{d_i \leq C_{max} \mid i \in V\}$

Ausgabe: (1) Touren T_i , $i = 1, 2, \dots, k$
 (2) $\#$ Lösung

Schritt 1: Initialisierung:

$U = V \setminus \{0\}$ (Menge der unzugeordneten Verbraucher)
 $T = \emptyset$

Schritt 2: Tourenbildung:

$i = 1$
 $d_{T_1} = 0$
 $p = 0$
while $U \neq \emptyset$
 $f = \min\{c_{pf}\}$, $f \in U$
 $T_i = T_i \cup (p, f)$
if $d_{T_i} + d_f > C_{max}$
 $T_i = T_i \cup (p, 0)$ (aktuelle Tour beenden)

```

     $i = i + 1$ 
     $T_i = (0, f)$  (neue Tour beginnen)
     $d_{T_i} = d_f$ 
     $U = U \setminus \{f\}$ 
     $p = f$ 
     $T_i = T_i \cup (f, 0)$ 

```

Schritt 3: Tourenprüfung

```

if  $|T| > |K|$ 
stop  $\nexists$  Lösung (zu viele Touren gebildet)

```

Dem Algorithmus Multiple Fragment (Algorithmus A.6) liegt eine Greedy-Heuristik zu Grunde. Es wird versucht einzelne Pfade aufzubauen und diese durch Hinzunahme weiterer, möglichst kurzer Kanten zu verbinden bis ein geschlossener Kreis entsteht.

Alg. A. 6

Algorithmus A.6: Multiple Fragment (TSP)

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Initialisierung:

Sortiere alle Kanten aufsteigend in einem Array $e_0, e_1, e_2, \dots, e_m$
 mit $e_k = (i, j), i \neq j, i, j \in V$, so dass $c_{e_0} \leq c_{e_1} \leq \dots \leq c_{e_m}$

Schritt 2: Tourenbildung:

```

 $T = \emptyset$ 
 $i = 0$ 
foreach  $j \in V$ 
   $Grad(j) = 0$ 
 $U = V$  (Menge der unzugeordneten Verbraucher)
while  $U \neq \emptyset$ 
   $e_i = (v_1, v_2)$ 
  if ( $T \cup e_i \neq$  Kreis mit  $|T \cup e_i| < n$ )
    and ( $(Grad(v_1) \leq 2)$  and ( $Grad(v_2) \leq 2$ ))
     $T = T \cup e_i$ 
     $U = U \setminus \{v_1, v_2\}$ 
     $Grad(v_1) = Grad(v_1) + 1$ 
     $Grad(v_2) = Grad(v_2) + 1$ 
   $i = i + 1$ 

```

Insertion Algorithmen

Die nachfolgenden Algorithmen werden zu einer Klasse zusammengefasst, weil sie auf dem gleichen Grundprinzip basieren. Es wird von einem Startkreis ausgegangen und dieser sukzessive, durch Einfügung weiterer Verbraucher, zu einer kompletten Tour bzw. einem Hamiltonschen Kreis vervollständigt:

Alg. A. 7

Algorithmus A.7: General Insertion (TSP)

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Initialisierung:

Wähle Startkreis mit der Knotenmenge $T \subseteq V$
 $U = V \setminus T$ (Restknoten)

Schritt 2: Tourenbildung:

while $U \neq \emptyset$
 Wähle Knoten $v \in U$
 Füge Knoten v in Kreis T ein
 $U = U \setminus \{v\}$

Der folgende Insertion-Algorithmus fügt den Verbraucher mit dem geringsten Abstand (Nearest Insertion (NI)) bzw. den mit dem größten Abstand (Farthest Insertion (FI)) zum bisherigen Kreis hinzu:

Algorithmus A.8: Nearest/Farthest Insertion (TSP)

Alg. A. 8

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Initialisierung:

$T = \{(p, q), (q, p)\}$ mit $\min\{c_{pq}\}$ (NI) **or**
 $T = \{(p, q), (q, p)\}$ mit $\max\{c_{pq}\}$ (FI)
 $U = V \setminus \{p, q\}$ (Restknoten)

Schritt 2: Tourenbildung:

while $U \neq \emptyset$
 $v \in U$ und $s_i \in V \setminus U$ mit $\min\{c_{v, s_i}\}$ (NI) **or**
 $v \in U$ und $s_i \in V \setminus U$ mit $\max\{c_{v, s_i}\}$ (FI)
 $e_1 = (r, s_i) \in T$ (r ist Vorgänger von s_i)
 $e_2 = (s_i, t) \in T$ (t ist Nachfolger von s_i)
if $c_{rv} + c_{vs_i} - c_{rs_i} < c_{s_i v} + c_{vt} - c_{s_i t}$
 $T = (T \setminus \{(r, s_i)\}) \cup \{(r, v)\} \cup \{(v, s_i)\}$ (vor s_i einfügen)
else
 $T = (T \setminus \{(s_i, t)\}) \cup \{(s_i, v)\} \cup \{(v, t)\}$ (nach s_i einfügen)
 $U = U \setminus \{v\}$

Beim Cheapest-Insertion-Algorithmus erfolgt die Einfügung des Verbrauchers, dessen Hinzunahme die geringsten zusätzlichen Kosten verursacht:

Algorithmus A.9: Cheapest Insertion (TSP)

Alg. A. 9

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Initialisierung:

$T = \{(p, q), (q, p)\}$ mit $\min\{c_{pq}\}$
 $U = V \setminus \{p, q\}$ (Restknoten)

Schritt 2: Tourenbildung:

```

while  $U \neq \emptyset$ 
   $k = (r, s) \in T$  und  $v \in U$  mit  $\min\{c_{rv} + c_{vs} - c_{rs}\}$ 
   $T = (T \setminus \{(r, s)\}) \cup \{(r, v)\} \cup \{(v, s)\}$ 
   $U = U \setminus \{v\}$ 

```

Die Hinzunahme eines zufällig ausgewählten Verbrauchers wird als Random Insertion bezeichnet. Der Algorithmus dafür lautet wie folgt:

Alg. A. 10

Algorithmus A.10: Random Insertion (TSP)

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Initialisierung:

```

 $T = \{(p, q), (q, p)\}$  mit  $\min\{c_{pq}\}$ 
 $U = V \setminus \{p, q\}$  (Restknoten)

```

Schritt 2: Tourenbildung:

```

while  $U \neq \emptyset$ 
   $v \in U$  zufällig gewählt
   $k = (r, s) \in T$  mit  $\min\{c_{rv} + c_{vs} - c_{rs}\}$ 
   $T = (T \setminus \{(r, s)\}) \cup \{(r, v)\} \cup \{(v, s)\}$ 
   $U = U \setminus \{v\}$ 

```

Die Verfahren Smallest-Summation-Insertion (SSI) und Largest-Summation-Insertion (LSI) nehmen den Verbraucher zu bisher eingefügten Verbrauchern hinzu, dessen Summe der Entfernung zu den Knoten im Kreis minimal bzw. maximal ist:

Alg. A. 11

Algorithmus A.11: Smallest/Largest Summation Insertion (TSP)

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Initialisierung:

```

 $T = \{(p, q), (q, p)\}$  mit  $\min\{c_{pq}\}$  (SSI) or
 $T = \{(p, q), (q, p)\}$  mit  $\max\{c_{pq}\}$  (LSI)
 $U = V \setminus \{p, q\}$  (Restknoten)

```

Schritt 2: Tourenbildung:

```

while  $U \neq \emptyset$ 
   $d_{best} = \infty$ 
  foreach  $w \in U$ 
     $d_w = \sum_{i \in V \setminus U} c_{iw}$ 
    if  $d_w < d_{best}$  (SSI) or
    if  $d_w > d_{best}$  (LSI)
       $d_{best} = d_w$ 
       $v = w$ 
   $k = (r, s) \in T$  mit  $\min\{c_{r,v} + c_{v,s} - c_{r,s}\}$ 
   $T = (T \setminus \{(r, s)\}) \cup \{(r, v)\} \cup \{(v, s)\}$ 
   $U = U \setminus \{v\}$ 

```

Spannbaum Algorithmen

Die nächsten beiden Algorithmen A.12 und A.13 basieren auf Spannbaum-Heuristiken und sind wiederum zur Lösung des TSP entwickelt worden. Sollen sie auf das VRP angewendet werden, so sind im Vorfeld einzelne Verbrauchermengen zu bilden, welche die Kapazitäts- und Zeitbedingungen der Transportfahrzeuge einhalten. Dafür kann eine beliebige Heuristik verwendet werden.

Algorithmus A.12: Double Spanning Tree (TSP)

Alg. A. 12

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Bestimme minimalen Spannbaum MST nach Prim und Kruskal

Schritt 2: Graph $G_2 = (V, E_2)$ entsteht aus der Verdoppelung aller Kanten aus MST

Schritt 3: Bestimme Eulertour ET in G_2
 (ET existiert, falls jede Kante zweimal, Knotengrade gerade)

Schritt 4: Gib ET eine Orientierung

Schritt 5: Bestimme TSP-Tour T aus ET mittels Depth First Search

Wähle beliebigen Knoten $v_{start} \in V$

$p = v_{start}$

$T = \emptyset$

$U = V \setminus v_{start}$ (unmarkierte Knoten)

while $U \neq \emptyset$

Gehe von p entlang der Orientierung bis $q \in U$ erreicht

$T = T \cup \{p, q\}$

$U = U \setminus \{q\}$

$p = q$

$T = T \cup \{(p, v_{start})\}$

Algorithmus A.13: Christofides-Heuristik (TSP)

Alg. A. 13

Eingabe: Verbrauchermenge $V = \{1, 2, \dots, n\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Tour T

Schritt 1: Bestimme minimalen Spannbaum $MST = (V, E_{MST})$ nach Prim und Kruskal

$S = \{v \in MST \mid \text{Grad}(v) = \text{ungerade}\}$

Konstruiere Distanzmatrix D für S

Schritt 2: Bestimme perfektes kostenminimales Matching $M = (V, E_M)$ für S
 $G_2 = (V, E_{MST} \cup E_M)$

Schritt 3: Bestimme Eulertour ET in G_2

Schritt 4: Gib ET eine Orientierung

Schritt 5: Bestimme TSP-Tour T aus ET nach Depth First Search
(siehe Algorithmus A.12 (Double Spanning Tree))

A.2.2 Eröffnungsverfahren für das VRP

Im nun folgenden Abschnitt werden Lösungsalgorithmen vorgestellt, welche speziell für das VRP entwickelt wurden. Die Tourenprüfung im Schritt 4 kann dabei wieder equivalent ersetzt werden (siehe Algorithmen aus Abschnitt A.2.1).

Savings-Algorithmen

Die Touren der Savings-Verfahren werden so initialisiert, dass zunächst unzulässige Touren gebildet werden. Hierzu bekommt jeder einzelne Verbraucher ein eigenes Transportfahrzeug zugewiesen, so dass eine individuelle Belieferung der einzelnen Verbraucher erfolgt. Diese einzelnen Touren werden dann so verbunden, dass die Strecke vom Ende einer Tour zum Anfang einer anderen Tour möglichst kostengünstig zurückgelegt werden kann. Dieser Vorgang wird dann solange iteriert, bis auf Grund von Kapazitätsverletzungen keine weiteren Verschmelzungen vorgenommen werden können. Zwei Varianten der Savings-Verfahren seien hier vorgestellt.

Beim simultanen Saving werden alle Touren gleichzeitig erzeugt:

Alg. A. 14

Algorithmus A.14: Simultaner Saving-Algorithmus (nach Clarke und Wright)

Eingabe: Verbrauchermenge $V = \{0, 1, \dots, n\}$ (0 – Lager)
Verbraucherbedarfsmengen $D = \{d_i \mid i \in V\}$
Kostenmatrix $C = (c_{ij}), i, j \in V$
Transportfahrzeugmenge $K = \{v_1, v_2, \dots, v_k\}$
maximale Transportfahrzeugkapazität C_{max}

Ausgabe: (1) Touren $T_i, i = 1, 2, \dots, k$
(2) $\#$ Lösung

Schritt 1: Berechne Savings-Matrix $S = (s_{ij}), i, j = 1, 2, \dots, n$
for $i = 1$ **to** n
 $s_{ii} = 0$
for $j = i + 1$ **to** n
 $s_{ij} = s_{ji} = c_{i0} + c_{0j} - c_{ij}$

Schritt 2: Bilde alle Pendeltouren
foreach Verbraucher $i \in V \setminus \{0\}$
 $T_i = \{(0, i, 0)\}$
 $d(T_i) = d_i$

Schritt 3: Bilde Touren
while $S \neq (0)$

$$s_{ij} = \max\{s_{ij} \mid i \in T_{k_1} \text{ und } i \text{ letzter Verbraucher von } T_{k_1}, \\ j \in T_{k_2} \text{ und } j \text{ erster Verbraucher von } T_{k_2}\}$$

$$s_{jj} = s_{ji} = 0 \text{ (aus Savings-Matrix streichen)}$$

if ($T_{k_1} \neq T_{k_2}$)

if ($d(T_{k_1}) + d(T_{k_2}) \leq C_{max}$)

$T_{k_1} = T_{k_1} \setminus (i, 0) \cup T_{k_2} \setminus (0, j) \cup (i, j)$ (Verschmelze Touren)

$d(T_{k_1}) = d(T_{k_1}) + d(T_{k_2})$

Schritt 4: Tourenprüfung

if $|T| > |K|$

stop \nexists Lösung (zu viele Touren gebildet)

Im Falle des sequentiellen Savings werden die Transporttours nacheinander aufgebaut und immer dann mit einer neuen Tour begonnen, wenn die Kapazitätsrestriktionen dies erforderlich machen:

Algorithmus A.15: Sequentieller Saving-Algorithmus (nach Clarke und Wright)

Alg. A. 15

Eingabe: Verbrauchermenge $V = \{0, 1, \dots, n\}$ (0 – Lager)
 Verbraucherbedarfsmengen $D = \{d_i \mid i \in V\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$
 Transportfahrzeugmenge $K = \{v_1, v_2, \dots, v_k\}$
 maximale Transportfahrzeugkapazität C_{max}

Ausgabe: (1) Touren $T_i, i = 1, 2, \dots, k$
 (2) \nexists Lösung

Schritt 1: Berechne Savings-Matrix $S = s_{ij}, i, j = 1, 2, \dots, n$

for $i = 1$ **to** n

$s_{ii} = 0$

for $j = i + 1$ **to** n

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

Schritt 2: Bilde Touren

$i = 1$

$U = V \setminus \{0\}$ (Menge der unbesuchten Verbraucher)

while $\exists \max\{s_{jk} \mid j, k \in U, |U| > 1, d_j + d_k \leq C_{max}\}$

$T_i = \{(0, j), (j, k), (k, 0)\}$

$s_{jk} = s_{kj} = 0$ (aus Savings-Matrix streichen)

$U = U \setminus \{j, k\}$

$d(T_i) = d_j + d_k$

$M = U$

while $M \neq \emptyset$

$l = \max\{s_{li}, s_{lj}\}, l \in M$

$s_{jl} = s_{lj} = 0$

$s_{kl} = s_{lk} = 0$

$M = M \setminus \{l\}$

if $d(T_i) + d_l < C_{max}$

if $c_{jl} < c_{kl}$

$T_i = (T_i \setminus \{(j, 0)\}) \cup \{(j, l)\}$

else

$T_i = (T_i \setminus \{(k, 0)\}) \cup \{(k, l)\}$

$T_i = T_i \cup (l, 0)$

$U = M$

$i = i + 1$

if $U \neq \emptyset$

foreach $v \in U$

$T_i = \{(0, v), (v, 0)\}$

$i = i + 1$

Schritt 4: Tourenprüfung

if $|T| > |K|$
stop \nexists Lösung (zu viele Touren gebildet)

Geographische Algorithmen

Abschließend seien noch zwei Eröffnungsverfahren für das VRP vorgestellt, welche die geographischen Koordinaten der Verbraucher mit einbeziehen und dadurch nicht allgemein auf jedes Tourenbildungsproblem anwendbar sind.

Alg. A. 16

Algorithmus A.16: Sweep-Algorithmus (nach Gillett und Miller)

Eingabe: Verbrauchermenge $V = \{0, 1, \dots, n\}$ (0 – Lager)
 Verbraucherkoordinaten $XY = \{(x_i, y_i) \mid i \in V\}$
 Verbraucherbedarfsmengen $D = \{d_i \mid i \in V\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$
 Transportfahrzeugmenge $K = \{v_1, v_2, \dots, v_k\}$
 maximale Transportfahrzeugkapazität C_{max}

Ausgabe: (1) Touren $T_i, i = 1, 2, \dots, k$
 (2) \nexists Lösung

Schritt 1: Initialisierung:

foreach Verbraucher $i \in V \setminus \{0\}$
 Berechne Polarwinkel $\phi(i)$ mit Lager als Mittelpunkt

Schritt 2: Sortierung:

Sortiere nach aufsteigenden Polarwinkeln:
 $S = \{v_1, v_2, \dots, v_n \mid \phi(v_i) \leq \phi(v_j), i < j \forall i, j \in V \setminus \{0\}\}$

Schritt 3: Clusterbildung:

$i = 1$ (Nummer des Transportfahrzeuges)
 $U = V \setminus \{0\}$ (Menge der unbesuchten Verbraucher)
 $p = 0$ (Gesamtanzahl der Verbrauchermengen)

repeat

$T_i = \{v_p, \dots, v_u\}$, **where** $u \leq n$ **and** $\sum_{j=p}^u d_{v_j} \leq C_{max}$
 (Verbraucher dem Transporter i zuweisen)

$U = U \setminus T_i$

$p = p + 1$

$i = i + 1$

if $i > k$

stop \nexists Lösung

until $U = \emptyset$

Schritt 4: Bilde TSP-Routen $\forall T_i, i = 0, 1, \dots, p$ nach beliebiger Heuristik

Alg. A. 17

Algorithmus A.17: Clustering-Algorithmus

Eingabe: Verbrauchermenge $V = \{0, 1, \dots, n\}$ (0 – Lager)
 Verbraucherkoordinaten $XY = \{(x_i, y_i) \mid i \in V\}$
 Verbraucherbedarfsmengen $D = \{d_i \mid i \in V\}$
 Clusterradius r

Ausgabe: Cluster $C_i, i = 1, 2, \dots, m$

```

 $U = V \setminus \{0\}$  (Menge der unzugeordneten Verbraucher)
 $i = 1$ 
while  $U \neq \emptyset$ 
  Wähle Verbraucher  $j$  nach beliebiger Heuristik
   $C_i = \{j\}$ 
   $U = U \setminus \{j\}$ 
   $M = U$ 
  while  $M \neq \emptyset$ 
    Wähle nach beliebiger Heuristik Verbraucher  $k$ , der Cluster  $C_i$  am nächsten liegt
     $M = M \setminus \{k\}$ 
    if  $(\sum_{l \in C_i \cup \{k\}} d_l \leq C_{max})$  and  $(radius(C_i \cup \{k\}) \leq r)$ 
       $C_i = C_i \cup \{k\}$ 
       $U = U \setminus \{k\}$ 

```

A.2.3 Verbesserungsverfahren für das TSP und VRP

Dieser Abschnitt zeigt, wie mit einer beliebigen Eröffnungsheuristik erzeugte Touren durch lokale Suche verbessert werden können. Die Algorithmen werden hier zunächst nur für das TSP vorgestellt. Bei Anwendung auf das VRP erreichen sie nur eine Verbesserung innerhalb einzelner Touren (Intrachange). Um auch einen Austausch von Verbrauchern zwischen zwei oder mehreren Touren zu ermöglichen (Interchange), müssen sie so modifiziert werden, dass eine Prüfung auf Verletzung der Zeit- und Kapazitätsbedingungen erfolgt.

r-opt

Beim *r-opt*-Verfahren werden r Kanten eines Hamiltonschen Kreises gegen r andere Kanten dann ausgetauscht, wenn dieser Austausch eine kostengünstigere Variante darstellt:

Algorithmus A.18: r-opt-Verfahren (TSP)

Eingabe: Ausgangstour $T = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\}$
 Anzahl auszutauschender Kanten r
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Endtour T

```

repeat
   $c(T) = \sum_{i=1}^{n-1} c_{v_i, v_{i+1}} + c_{v_n, v_1}$ 
   $c_{best} = c(T)$ 
   $Z =$  Menge aller  $r$ -elementigen Teilmengen von  $T$ 
  foreach  $R \in Z$ 
     $S = T \setminus R$ 
    foreach Tour  $T_{neu}$ , die  $S$  enthält
      if  $(c(T_{neu}) < c_{best})$  then
         $c_{best} = c(T_{neu})$ 
         $T_{best} = T_{neu}$ 
  if  $(c_{best} < c(T))$  then

```

Alg. A. 18

$T = T_{neu}$
until $c_{best} = c(T)$

2-opt

Das Verfahren 2-opt betrachtet im Gegensatz zum r-opt-Verfahren nur den möglichen Austausch von genau zwei unterschiedlichen Kanten:

Alg. A. 19

Algorithmus A.19: 2-opt-Verfahren (TSP)

Eingabe: Ausgangstour $T = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Endtour T

repeat

$$c(T) = \sum_{i=1}^{n-1} c_{v_i, v_{i+1}} + c_{v_n, v_1}$$

$$c_{best} = c(T)$$

foreach Kante $e_i = (v_i, v_{i+1}) \in T$

foreach Kante $f_j = (v_j, v_{j+1}) \in T \setminus \{e_i\}$

$$c(T)_{neu} = c(T) - c_{v_i, v_{i+1}} - c_{v_j, v_{j+1}} + c_{v_i, v_{j+1}} + c_{v_j, v_{i+1}}$$

if $(c(T)_{neu} < c_{best})$ **then**

$$c_{best} = c(T)_{neu}$$

$$e_{best} = e_i, f_{best} = f_j$$

$$w_{best} = \{(v_i, v_j), (v_{i+1}, v_{j+1})\}$$

if $(c_{best} < c(T))$ **then**

$$T = (T \setminus \{e_{best} \cup f_{best}\}) \cup w_{best}$$

until $c_{best} = c(T)$

3-opt

Nachfolgender Algorithmus sucht nach einer besseren Lösung als die Ausgangslösung, indem er die Kosten eines Austausches von drei verschiedenen Kanten eines Hamiltonschen Kreises betrachtet:

Alg. A. 20

Algorithmus A.20: 3-opt-Verfahren (TSP)

Eingabe: Ausgangstour $T = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Endtour T

repeat

$$c(T) = \sum_{i=1}^{n-1} c_{v_i, v_{i+1}} + c_{v_n, v_1}$$

$$c_{best} = c(T)$$

foreach Kante $e_i = (v_i, v_{i+1}) \in T$

foreach Kante $f_j = (v_j, v_{j+1}) \in T \setminus \{e_i\}$

foreach Kante $g_k = (v_k, v_{k+1}) \in T \setminus \{e_i \cup f_j\}$

$$\Delta c(T) = c(T) - c_{v_i, v_{i+1}} - c_{v_j, v_{j+1}} - c_{v_k, v_{k+1}}$$

$$c_1(T) = \Delta c(T) + c_{v_{i+1}, v_{j+1}} + c_{v_j, v_{k+1}} + c_{v_k, v_i}$$

$$c_2(T) = \Delta c(T) + c_{v_i, v_{j+1}} + c_{v_j, v_k} + c_{v_{k+1}, v_{i+1}}$$

$$c_3(T) = \Delta c(T) + c_{v_i, v_j} + c_{v_{j+1}, v_{k+1}} + c_{v_k, v_{i+1}}$$

$$c_4(T) = \Delta c(T) + c_{v_i, v_{j+1}} + c_{v_j, v_{k+1}} + c_{v_k, v_{i+1}}$$

```

 $w_1 = \{(v_i, v_k), (v_{j+1}, v_{i+1}), (v_j, v_{k+1})\}$ 
 $w_2 = \{(v_i, v_{j+1}), (v_k, v_j), (v_{i+1}, v_{k+1})\}$ 
 $w_3 = \{(v_i, v_j), (v_{i+1}, v_k), (v_{j+1}, v_{i+1})\}$ 
 $w_4 = \{(v_i, v_{j+1}), (v_k, v_{i+1}), (v_j, v_{k+1})\}$ 
for  $typ = 1$  to 4
  if  $(c_{typ}(T) < c_{best})$  then
     $c_{best} = c_{typ}(T)$ 
     $e_{best} = e_i, f_{best} = f_j, g_{best} = g_k$ 
     $w_{best} = w_n$ 
  if  $(c_{best} < c(T))$  then
     $T = (T \setminus \{e_{best} \cup f_{best} \cup g_{best}\}) \cup w_{best}$ 
until  $c_{best} = c(T)$ 

```

variables r-opt

Bei dem von Lin und Kernighan entwickelten Verfahren¹ werden sich die Eigenschaften der r-opt-Verfahren zu Nutze gemacht und entsprechend erweitert. Es wird untersucht, ob ein r-opt-Austausch nicht noch durch einen (r+1)-Austausch verbessert werden könnte:

Algorithmus A.21: variables r-opt-Verfahren (nach Lin und Kernighan) (TSP)

Alg. A. 21

Eingabe: Ausgangstour $T = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Endtour T

foreach $(v_i, v_j) \in T$: Lösche Markierung von v_i

repeat

$gainsum_{best} = 0$

$i = 1$

Schritt 1: Wähle beliebigen, unmarkierten Knoten v_{k_1} mit Kante $e_i = (v_{k_1}, v_{k_2}) \in T$

Schritt 2: Wähle Kante $f_1 = (v_{k_2}, v_{k_3})$ mit $g_1 = c_{v_{k_1}, v_{k_2}} - c_{v_{k_2}, v_{k_3}} > 0$

if \nexists Kante f_1

Wähle neuen unmarkierten Knoten v_{k_1}

goto Schritt 1

repeat

$i = i + 1$

Wähle e_i und f_i , so dass gilt:

(a) $e_i = (v_{k_{2i-1}}, v_{k_{2i}})$, wobei

(1) $T' = \{T \setminus \{e_n, \forall 1 \leq n \leq i\}\} \cup \{f_n, \forall 1 \leq n \leq i-1\} \cup (v_{k_{2i}}, v_{k_1})$
 mit T' gültige Tour

(2) $gainsum_i = gainsum_{i-1} - c_{v_{k_{2i}}, v_{k_1}} - c_{v_{k_{2i-1}}, v_{k_{2i}}}$

if $(gainsum_i > gainsum_{best})$

$gainsum_{best} = gainsum_i$

Merke diese Lösung T'

(b) f_i mit Ausgangspunkt $v_{k_{2i}}$, wobei

(1) $gainsum_i = \sum_{j=1} g_i$ mit $g_i = c_{v_{k_{2i-1}}, v_{k_{2i}}} - c_{v_{k_{2i}}, v_{k_{2i+1}}}$

(2) nach Entfernung von $f_i \exists$ entfernbare Kante $e_{i+1} \in T$

until (keine weiteren Kanten e_i und f_i gefunden) **or** $(gainsum_i \leq gainsum^*)$

Markiere v_{k_1}

if $(gainsum_{best} > 0)$

$c(T') = C(T) - gainsum_{best}$

$T = T'$

¹ siehe [LK73]

until $\forall (v_i, v_j) \in T: v_i$ markiert)

Or-opt

Der von Or entwickelte Algorithmus² betrachtet 1, 2 oder 3 benachbarte Knoten und versucht diese kostengünstiger zwischen zwei andere Knoten des Kreises einzufügen:

Alg. A. 22

Algorithmus A.22: Or-opt-Verfahren (nach Or) (TSP)

Eingabe: Ausgangstour $T = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Endtour T

Bilde Hamiltonschen Kreis K_H aus T :

$$K_H = \bigcup_{(i, j) \in T} K_H(i) = i$$

repeat

$$c(K_H) = \sum_{i=1}^n c_{K_H(i), K_H(i+1)} + c_{K_H(n), K_H(1)}$$

$$c_{best} = c(K_H)$$

for $s = 3$ **downto** 1

for $i = 1$ **to** $n - s - 1$

$$\Delta c(K_H) = c(K_H) - c_{K_H(i), K_H(i+1)} - c_{K_H(i+s), K_H(i+s+1)} + c_{K_H(i), K_H(i+s+1)}$$

for $j = i + s + 1$ **to** n

$$c(K_H)_{neu} = \Delta c(K_H) + c_{K_H(j), K_H(i+1)} + c_{K_H(i+s), K_H(j+1)} - c_{K_H(j), K_H(j+1)}$$

if $(c(K_H)_{neu} < c_{best})$ **then**

$$c_{best} = c(K_H)_{neu}$$

$$i_{best} = i, j_{best} = j, s_{best} = s$$

if $(c_{best} < c(K_H))$ **then**

Einfügen der Kette $\{K_H(i_{best} + 1), \dots, K_H(i_{best} + s_{best} + 1)\}$ nach $K_H(j_{best})$ in K_H

until $c_{best} = c(K_H)$

Bilde Tour T aus K_H :

$$T = \{\bigcup_{i=1}^{n-1} (K_H(i), K_H(i+1))\} \cup (K_H(n), K_H(1))$$

A.3 Moderne heuristische Verfahren

Dieser Abschnitt enthält Verfahren, die den so genannten Meta-Heuristiken zugeordnet werden. Meta-Heuristiken sind allgemein anwendbare Verfahren um zu Grunde liegende, problemspezifische Heuristiken, wie beispielsweise die Verbesserungsverfahren des vorigen Unterkapitels, in erfolgsversprechende Regionen des Suchraumes zu leiten und so lokale Extrempunkte zu verlassen.

² siehe auch [Or76]

Simulated Annealing

Das Verfahren des Simulated Annealing akzeptiert mit einer bestimmten Wahrscheinlichkeit auch schlechtere Lösungen bei der lokalen Suche. Diese Wahrscheinlichkeit nimmt für gewöhnlich im Laufe der Zeit ab:

Algorithmus A.23: Simulated-Annealing-Verfahren

Alg. A. 23

Eingabe: Ausgangstour $T = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$
 Zeitvariable $\theta > 0$
 Wiederholungsfaktor $r > 0$
 Toleranzbereich $\epsilon > 0$

Ausgabe: Endtour T

repeat
for $i = 0$ **to** r
 T' = gültige Veränderung von T (Nachbarlösung)
if $(c(T') < c(T))$
 $T = T'$
else
 $x = \text{Zufallszahl} \in \mathbb{R}, 0 \leq x \leq 1$
 $\Delta_c = |c(T') - c(T)|$
if $(x < e^{-\frac{\Delta_c}{\theta}})$
 $T = T'$
 $\theta = \gamma\theta$ **mit** $\gamma = 1 - \epsilon$
 $r = \alpha r$ **mit** $1 < \alpha < 2$
until Abbruchkriterium erfüllt

Tabu Search

Eine weitere Meta-Heuristik ist das Tabu-Search-Verfahren (siehe Algorithmus A.24), welches sich eine bestimmte Anzahl von Lösungen des bisherigen Suchprozesses merkt. Die Suche wird dabei wiederum deterministisch geleitet. Es wird in jedem Schritt die beste benachbarte Lösung angenommen, auch wenn diese schlechter als die aktuelle ist. Um Zyklen zu vermeiden wird das wiederholte Besuchen von Lösungen innerhalb der nächsten t Iterationen verboten.

Algorithmus A.24: Tabu-Search-Verfahren

Alg. A. 24

Eingabe: Ausgangstour $T = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Endtour T_{best}

$L = \emptyset$ (Tabu-Liste)
 $T_{best} = T$

repeat
 $ende = false$
 $i = 1$
 $T_M = \emptyset$ (Menge der betrachteten Nachbarlösungen)

```

repeat
   $T'_i =$  gültige Veränderung von  $T$ ,  $T'_i \neq T$  (Nachbarlösung)
  if  $c_{T'_i} < c_{T_{best}}$ 
     $ende = true$ 
     $T^* = T'_i$ 
  else
     $T_M = T_M \cup \{T'_i\}$ 
     $i = i + 1$ 
     $T = T'_i$ 
    if  $i > r_l$ 
       $ende = true$ 
       $T^* =$  bestbewertete Tour aus  $T_M$ 
  until  $ende = true$ 
   $T = T^*$ 
   $L = L \cup \{T\}$ 
  if  $c(T) < c(T_{best})$ 
     $T_{best} = T$ 
until Abbruchkriterium erfüllt

```

Evolutionäre Algorithmen

Evolutionäre Algorithmen arbeiten mit einer Menge von Problemlösungen, welche individuell zu verbessern sind, sich aber voneinander unterscheiden. Die beste Lösung aus dieser Menge wird als Ergebnis des nachfolgenden Algorithmus zurückgegeben:

Alg. A. 25

Algorithmus A.25: Grundprinzip evolutionärer Algorithmen

Eingabe: Ausgangstouren $T = \{T_i = \{(v_1, v_2), (v_2, v_3), \dots, (v_n, v_1)\},$
 $i = 1, 2, \dots, k, T_i \cap T_j = \emptyset, i, j = 1, 2, \dots, k\}$
 Kostenmatrix $C = (c_{ij}), i, j \in V$

Ausgabe: Endtour T_{best}

$P = T$ (Population)
 $c =$ Bewertung(P, C)

```

repeat
   $Q_s =$  Selektion( $P$ )
   $Q_r =$  Rekombination( $Q_s$ )
   $P =$  Mutation( $Q_r$ )
   $c(T_P) =$  Bewertung( $P, C$ )
until Abbruchkriterium erfüllt

```

$T_{best} =$ bestbewertete Tour aus P

B Übersetzungen

Die nachfolgenden Übersetzungen der englischsprachigen Fachbegriffe in die deutsche Sprache sollen nur zum Verständnis beitragen und nicht als genaues Übersetzungsschema dienen:

englischer Fachbegriff und gebräuchliche Abkürzung	sinngemässe deutsche Bedeutung und Abkürzung
Traveling Salesman Problem (TSP)	Handelsreisendenproblem (HRP)
Multiple Traveling Salesman Problem (m-TSP)	Problem des mehrfachen Handelsreisenden
Vehicle Routing Problem (VRP)	Tourenplanungsproblem (TPP)
Deadline Vehicle Routing Problem (DVRP)	Tourenplanungsproblem mit Stichtermin
Time Window	Zeitfenster
Vehicle Routing Problem with Time Windows (VRPTW)	Tourenplanungsproblem mit Zeitfenstern
Single Depot Vehicle Routing Problem (SDVRP)	Tourenplanungsproblem mit einem einzelnen zentralen Lager (entspricht VRP)
Multi Depot Vehicle Routing Problem (MDVRP)	Tourenplanungsproblem mit mehreren Lagern
Vehicle Routing Problem with Backhauls (VRPB)	Tourenplanungsproblem mit Aufladepunkten
Improvement Heuristic	Verbesserungsverfahren
Combinatorial Optimization Problem (COP)	Kombinatorisches Optimierungsproblem (KOP)
Shortest Path Problem (SPP)	Problem des kürzesten Weges

Tab. B. 1

Tabelle B.1: deutschsprachige Übersetzungen für englische Fachbegriffe (Teil 1)

Tab. B. 2

englischer Fachbegriff und gebräuchliche Abkürzung	sinngemässe deutsche Bedeutung und Abkürzung
Linear Programming (LP)	Lineare Programmierung
Integer Programming (IP)	Ganzzahlige Programmierung
Integer Linear Programming (ILP)	Ganzzahlige lineare Programmierung
Dynamic Programming	Dynamische Programmierung
Dynamic Optimization	Dynamische Optimierung
Dekomposition	Zerlegung
Branch and Bound (BnB)	Verzweigen und Begrenzen
Branch and Cut (BnC)	Verzweigen und Abschneiden
Divide-and-Conquer	Teile und Herrsche
Set Partitioning Problem	Mengenbildungsproblem
Matching	Anpassung
Polyhedral Combinatorics	Vielflächige Kombinatorik
Lagrange Penalties	Lagrange-Strafen
Relaxation	Lockerung, Aufweichung der Grenzen
Saving	Ersparnis
Backhaul Saving	Ersparnis für Rückgabe
Load-Based-Insertion	Einfügung in Abhängigkeit der verbleibenden Transportmenge
Backhaul Saving	Ersparnis für Rückgabe
Load-Based-Insertion	Einfügung in Abhängigkeit der verbleibenden Transportmenge
Stop-Based-Insertion	Einfügung in Abhängigkeit der verbleibenden Verbraucher
Spacefilling-Curves	Raumfüllende Kurven
Sweep	Auskehren

Tabelle B.2: deutschsprachige Übersetzungen für englische Fachbegriffe (Teil 2)

englischer Fachbegriff und gebräuchliche Abkürzung	sinngemässe deutsche Bedeutung und Abkürzung
Tabu Search	Tabu-Suche
Simulated Annealing	Simuliertes Ausglühen

Tab. B. 3

Tabelle B.3: deutschsprachige Übersetzungen für englische Fachbegriffe (Teil 3)

Sachregister

B

Branch-and-Bound	25, 49
Auswahlstrategien	
Best First	27
Breadth First	27
Depth First	27
Diving	27
Bounding	26
Branch-Operation	27
Branching	26
Branching-Strategien	
Maximal Binary	27
Maximal Fractional	27
Maximal Fractional Coefficient	27
Maximal Non Zero	27
Maximal Objective	27
Schrankenfunktion	26
Branch-and-Cut	28, 51
cut-Ebene	28
LP-Relaxation	28

C

Combinatorial Optimization Problem	11
--	----

D

Deadline Vehicle Routing Problem ..	2
-------------------------------------	---

E

Eröffnungsverfahren	35, 52
Geographische Algorithmen ...	60
Clustering	36, 60
Sweep-Algorithmus	36, 60
Insertion Algorithmen	54

Cheapest Insertion	55
Farthest Insertion	55
Farthest-Insertion	38
Furthest-Insertion	38
General Insertion	54
Largest Summation Insertion	56
Load-Based Insertion	38
Nearest Insertion	38, 55
Random Insertion	56
Smallest Summation Insertion	56
Stop-Based Insertion	38
Neighbour Algorithmen	52
Farthest-Neighbour	37
Multiple Fragment	54
Nearest Neighbour	37, 52 f.
Parallelverfahren	36
Insertion-Verfahren	<i>siehe</i>
Eröffnungsverfahren \ Insertion Algorithmen	
Neighbour-Verfahren	<i>siehe</i>
Eröffnungsverfahren \ Neighbour Algorithmen	
Savings-Verfahren	<i>siehe</i>
Eröffnungsverfahren \ Savings-Algorithmen	
Savings-Algorithmen	58
Sequentielles Clarke-und-Write-Saving	38, 59
Simultanes Clarke-und-Write-Saving	38, 58
Spannbaum Algorithmen	57
Christofides-Heuristik	57
Double Spanning Tree	57
Sukzessivverfahren	35
Giant-Tour-Verfahren	36
Spacefilling-Curves	36

Exakte Verfahren

- beschränkte Enumeration 25
- untere zeitliche Schranke 25
- Branch-and-Bound *siehe*
 - Branch-and-Bound
- Branch-and-Cut *siehe*
 - Branch-and-Cut
- Dekomposition 24
- Enumeration 48
- Integer Programming 24
- Interior-Point-Methoden 24
- Linear Integer Programming 24
- Linear Programming 23
- Simplexmethode 24

G

-
- Gesamtschrittverfahren 35
 - Patching-Algorithmus 35
 - Graphentheorie 5
 - Bewerteter Graph 5
 - Digraph 5
 - Grad 5
 - Nachbar 5
 - Nachfolger 5
 - Vorgänger 5
 - Eulerscher Graph 6
 - Fluss 7
 - Geschlossene Eulersche Linie 6
 - Gewichteter Graph 5
 - Hamiltonscher Kreis 6
 - Kantenmenge 5
 - Knotenmenge 5
 - Matching 7
 - Perfektes Matching 7
 - Mehrfachkante 5
 - Minimalgerüst 7
 - Multigraph 5
 - Netzwerk 6
 - Schlinge 5
 - Ungerichteter Graph 5
 - Weg 6
 - Zusammenhängender Graph 6

H

Heuristische Verfahren

- Eröffnungsverfahren *siehe*
 - Eröffnungsverfahren
- Gesamtschrittverfahren *siehe*
 - Gesamtschrittverfahren
- Moderne heuristische Verfahren 64
 - Abbruchkriterien 42
 - Evolutionäre Algorithmen 43, 66
 - Simulated Annealing 42, 65
 - Tabu Search 42, 65
- Verbesserungsverfahren *siehe*
 - Verbesserungsverfahren

L

Lösungsansatz

- Dekompositionsverfahren 33
- Dynamic Optimization 34
- Dynamic Programming 34
- Polyhedral-Combinatoric-
Verfahren 31
- Set-Partitioning 30
- Shortest-Path-Relaxation 32
- v-Matching-Relaxation 31
- v-tree-Relaxation 32
- Linear Generalized Assignment Pro-
blem 30
- Lagrange Relaxation 30

M

-
- Multi Depot Vehicle Routing Pro-
blem 3, 16
 - Multiple Traveling Salesman Pro-
blem 1, 14

N

-
- Nonlinear Generalized Assignment Pro-
blem 29

P

-
- Pickup and Delivery Problem 2, 18
 - Beladepunkt 18
 - Beladezeitraum 18

Entladepunkt	18
Entladezeitraum	18

T

Tourenbildung	1
Transportoptimierung	5
Assignment Problem	9
Routing Problem	7
Abzugebende Menge	8
Benötigte Menge	8
Produzent	7
Verbraucher	7
Unkapazitiertes Transportpro- blem	8
Fiktiver Produzent	8
Fiktiver Verbraucher	8
Traveling Salesman Problem ...	1, 11
Kurzyklus	12

V

v-Matching-Problem	31
Vehicle Routing Problem	1, 12
Flussproblem	31
Zeitrestriktion	15
Vehicle Routing Problem with Back- hauls	2, 19
Backhaul Saving	20
Cheapest-Insertion	20
Clarke-und-Wright-Saving	20
Load-Based-Insertion	21
Spacefilling-Curves	20 f.
Stop-Based-Insertion	21
Verbesserungsverfahren ...	35, 38, 61
r-optimale Verfahren	39, 61
2-opt-Verfahren	39, 62
3-opt-Verfahren	39, 62
Lin-Kernighan-Algorithmus	40, 63
Or-opt-Verfahren	40, 64
variables r-opt-Verfahren	40, 63

Literaturverzeichnis

- [AG88] ASSAD, A. A. ; GOLDEN, B. L.: *Studies in Management Science and Systems*. Bd. 16: *Vehicle Routing: Methods and Studies*. North-Holland, 1988
- [AK89] AARTS, E. ; KORST, J.: *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons, 1989
- [BP82] BARTHOLDI, J. ; PLATZMANN, L.: An $O(N \log N)$ Planar Traveling Salesman Heuristic Based on Spacefilling Curves. In: *Operations Research Letters* 1 (1982), Nr. 4, S. 121–125
- [CE69] CHRISTOFIDES, N. ; EILON, S.: An Algorithm for the Vehicle-dispatching Problem. In: *Operational Research Quarterly* 20 (1969), S. 309–318
- [Chr85] CHRISTOFIDES, N.: Vehicle routing. In: LAWLER, E. L. (Hrsg.) ; LENSTRA, J. K. (Hrsg.) ; KAN, A. H. G. R. (Hrsg.) ; SHMOYS, D. B. (Hrsg.): *The Traveling Salesman Problem*. Chichester : John Wiley, 1985, S. 431–448
- [CMT81] CHRISTOFIDES, N. ; MINGOZZI, A. ; TOTH, P.: Exact algorithms for solving the vehicle routing problem based on spanning trees and shortest path relaxations. In: *Mathematical Programming* 20 (1981), S. 255–282
- [CW64] CLARKE, G. ; WRIGHT, J.: Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. In: *Operations Research* Bd. 12, INFORMS, 1964, S. 568–581
- [DB84] DEIF, I. ; BODIN, L.: Extension of the Clarke and Wright Algorithm for Solving the Vehicle Routing Problem with Backhauling. In: *Proceedings of the Babson Conference on Software Uses in Transportation and Logistics Management*. MA : Babson Park, 1984, S. 75–96
- [DDSS95] DESROSIER, J. ; DUMAS, Y. ; SOLOMON, M. M. ; SOUMIS, F.: Time Constrained Routing and Scheduling. In: *Handbooks in Operations Research and Management Science* Bd. 8. North Holland : Elsevier Science, 1995, S. 35–139
- [DR59] DANTZIG, G. B. ; RAMSER, R. H.: The Truck Dispatching Problem. In: *Management Science* Bd. 6, INFORMS, 1959, S. 80

- [Fis95] FISHER, M.: Vehicle Routing. In: *Network Routing* Bd. 8. North Holland : Elsevier Science, 1995, S. 1–34
- [GBAS85] GOLDEN, B. ; BAKER, E. ; ALFARO, J. ; SCHAFFER, J.: The Vehicle Routing Problem with Backhauling: Two Approaches. In: HAMMESFAHR, R. D. (Hrsg.): *Proceedings of the Twenty-First Annual Meeting of S. E. TIMS*. SC : Myrtle Beach, 1985, S. 90–92
- [Gel91] GELINAS, S.: *Fabrication de Tournees avec Rechargement*. Montreal, Ecole Polytechnique, Unveröffentlichte Master's Thesis, 1991
- [GHL93] GENDREAU, M. ; HERTZ, A. ; LAPORTE, G.: A tabu search heuristic for the vehicle routing problem / Centre de Recherche sur les Transports. Universit'e de Montreal, 1993 (777). – Publication
- [GM74] GILLETT, M. ; MILLER, L.: An heuristic algorithm for the vehicle dispatching problem. In: *Operations Research* Bd. 22, INFORMS, 1974, S. 240–349
- [Lin65] LIN, S.: Computer Solutions to the Traveling Salesman Problem / Bell System. 1965 (44). – Technical Journal. – 2245–2269 S.
- [LK73] LIN, S. ; KERNIGHAN, B.: An effective heuristic algorithm for the traveling salesman problem. In: *Operations Research* Bd. 21, INFORMS, 1973, S. 498–516
- [LLRS85] *Kapitel 7.3*. In: LAWLER, E. L. ; LENSTRA, J. K. ; RINNOOY KAN, A. H. G. ; SHMOYS, D. B.: *The Traveling Salesman Problem*. New York : John Wiley & Sons, 1985
- [LR81] LENSTRA, J. K. ; RINNOOY KAN, A. H. G.: Complexity of vehicle routing and scheduling problems. In: *Networks* Bd. 11. John Wiley & Sons, 1981, S. 221–227
- [NM93] NEUMANN, K. ; MORLOCK, M.: *Operations Research*. München : Hanser Verlag, 1993
- [Or76] OR, I.: *Travelling Salesman Type Combinatorial Problems and Their Relation to the Logistics of Blood Banking*. Northwestern University, Department of Industrial Engineering and Management Science, Ph. D. thesis, 1976
- [Ral02] RALPHS, T. K.: *Parallel Branch and Cut for Capacitated Vehicle Routing*. Internet Dokument. www.lehigh.edu/~tkr2/research/papers/PVRP.pdf. Version: Mai 2002
- [Rec73] RECHENBERG, I.: *Evolutionsstrategien*. Berlin : Friedrich Frommann Verlag, 1973
- [Rei94] REINELT, G.: *The Traveling Salesman - Computational Solutions for TSP Applications*. Heidelberg : Springer Verlag, 1994

- [RLPT01] RALPHS, T. K. ; L.KOPMAN ; PULLERBLANK, W. R. ; TROTTER, L. E.: *On the Capacitated Vehicle Routing Problem*. Internet Dokument. www.lehigh.edu/~tkr2/research/papers/vrp.pdf. Version: 2001
- [RS94] RIBEIRO, C. ; SOUMIS, F.: A column generation approach to the multiple depot vehicle scheduling problem. In: *Operations Research* Bd. 42, INFORMS, 1994, S. 41–52
- [RSL77] ROSENKRANTZ, D. ; STERNS, R. ; LEWIS, P.: An Analysis of Several Heuristics for the Traveling Salesman Problem. In: *SIAM Journal on Computing* Bd. 6, 1977, S. 563–581
- [YCR⁺87] YANO, C. A. ; CHAN, T. J. ; RICHTER, L. ; CULTER, T. ; MURTY, K. G. ; MCGETTIGAN, D.: Vehicle Routing at Quality Stores. In: *Interfaces* Bd. 17, 1987, S. 52–63

Chemnitzer Informatik-Berichte

In der Reihe der Chemnitzer Informatik-Berichte sind folgende Berichte erschienen:

- CSR-00-01** J.A. Makowsky und K. Meer, Polynomials of bounded tree-width, Januar 2000
- CSR-00-02** Andreas Goerdt, Efficient interpolation for the intuitionistic sequent calculus, Januar 2000, Chemnitz
- CSR-01-01** Werner Dilger, Evelyne Keitel, Kultur und Stil in der Informatik?, Januar 2001, Chemnitz
- CSR-01-02** Guido Brunnett, Thomas Schaedlich, Marek Vanco, Extending Laszlo's Algorithm to 3D, März 2001, Chemnitz
- CSR-01-03** P. Köchel, U. Nieländer, M. Sturm, KASIMIR - object-oriented KANban SIMulation Imaging Reality, März 2001, Chemnitz
- CSR-02-01** Andrea Sieber, Werner Dilger, Theorie und Praxis des Software Engineering, Oktober 2001, Chemnitz
- CSR-02-02** Tomasz Jurdzinski, Mirosław Kutylowski, Jan Zatoptionski, Energy-Efficient Size Approximation of Radio Networks with no Collision Detection, Februar 2002, Chemnitz
- CSR-02-03** Tomasz Jurdzinski, Mirosław Kutylowski, Jan Zatoptionski, Efficient Algorithms for Leader Election in Radio Networks, Februar 2002, Chemnitz
- CSR-02-04** Tomasz Jurdzinski, Mirosław Kutylowski, Jan Zatoptionski, Weak Communication in Radio Networks, Februar 2002, Chemnitz
- CSR-03-01** Amin Coja-Oghlan, Andreas Goerdt, André Lanka, Frank Schädlich, Certifying Unsatisfiability of Random $2k$ -SAT Formulas using Approximation Techniques, Februar 2003, Chemnitz
- CSR-03-02** M. Randrianarivony, G. Brunnett, Well behaved mesh generation for parameterized surfaces from IGES files, März 2003, Chemnitz
- CSR-03-03** Optimizing MPI Collective Communication by Orthogonal Structures, Matthias Kühnemann, Thomas Rauber, Gudula Rünger, September 2003, Chemnitz
- CSR-03-04** Daniel Balkanski, Mario Trams, Wolfgang Rehm, Heterogeneous Computing With MPICH/Madeleine and PACX MPI: a Critical Comparison, Dezember 2003, Chemnitz
- CSR-03-05** Frank Mietke, Rene Grabner, Torsten Mehlan, Optimization of Message Passing Libraries - Two Examples, Dezember 2003, Chemnitz

Chemnitzer Informatik-Berichte

- CSR-04-01** Karsten Hilbert, Guido Brunnett, A Hybrid LOD Based Rendering Approach for Dynamic Scenes, Januar 2004, Chemnitz
- CSR-04-02** Petr Kroha, Ricardo Baeza-Yates, Classification of Stock Exchange News, November 2004, Chemnitz
- CSR-04-03** Torsten Hoefler, Torsten Mehlan, Frank Mietke, Wolfgang Rehm, A Survey of Barrier Algorithms for Coarse Grained Supercomputers, Dezember 2004, Chemnitz
- CSR-04-04** Torsten Hoefler, Wolfgang Rehm, A Meta Analysis of Gigabit Ethernet over Copper Solutions for Cluster-Networking, Dezember 2004, Chemnitz
- CSR-04-05** Christian Siebert, Wolfgang Rehm, One-sided Mutual Exclusion A new Approach to Mutual Exclusion Primitives, Dezember 2004, Chemnitz
- CSR-05-01** Daniel Beer, Steffen Höhne, Gudula Rünger, Michael Voigt, Software- und Kriterienkatalog zu RAfEG - Referenzarchitektur für E-Government, Januar 2005, Chemnitz
- CSR-05-02** David Brunner, Guido Brunnett, An Extended Concept of Voxel Neighborhoods for Correct Thinning in Mesh Segmentation, März 2005, Chemnitz
- CSR-05-03** H. Eichner, C. Trinitis, T. Klug, J. Tao, S. Schuch, C. Clauß, T. Arens, S. Lankes, T. Bemmerl, W. Karl, W. Rehm, T. Mehlan, T. Hoefler, F. Mietke, C. Siebert, M. Schwind, Kommunikation in Clusterrechnern und Clusterverbundsystemen, Tagungsband zum 1. Workshop, Dezember 2005, Chemnitz
- CSR-05-04** Andreas Goerdts, Higher type recursive program schemes and the nested pushdown automaton, Dezember 2005, Chemnitz
- CSR-05-05** Amin Coja-Oghlan, Andreas Goerdts, André Lanka, Spectral Partitioning of Random Graphs with Given Expected Degrees, Dezember 2005, Chemnitz
- CSR-06-01** Wassil Dimitrow, Mathias Sporer, Wolfram Hardt, UML basierte Zeitmodellierung für eingebettete Echtzeitsysteme, Februar 2006, Chemnitz
- CSR-06-02** Mario Lorenz, Guido Brunnett, Optimized Visualization for Tiled Displays, März 2006, Chemnitz
- CSR-06-03** D. Beer, S. Höhne, R. Kunis, G. Rünger, M. Voigt, RAfEG - Eine Open Source basierte Architektur für die Abarbeitung von Verwaltungsprozessen im E-Government, April 2006, Chemnitz
- CSR-06-04** Michael Kämpf, Probleme der Tourenbildung, Mai 2006, Chemnitz

Chemnitzer Informatik-Berichte

ISSN 0947-5125

Herausgeber: Fakultät für Informatik, TU Chemnitz
Straße der Nationen 62, D-09111 Chemnitz