Vehicle Routing: Implementierung und Analyse der „Parallel Savings Based Heuristic“.

eingereicht von

Alexander Landgraf
Matrikelnummer 8925890
Hoffmeistergasse 6/15
1120 Wien

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Magister rerum socialium oeconomicarumque
(Mag. rer. soc. oec.)

ausgeführt am Institut für Ökonometrie, Systemtheorie und Operations Research
der Technischen Universität Wien
unter Anleitung von
Univ.Doz. Ass.Prof. Dipl.-Ing. Dr.techn. Alexander Mehlmann

Sozial- und Wirtschaftswissenschaftliche Fakultät
Universität Wien

Studienrichtung: Wirtschaftsinformatik
Studienzweig: Betriebsinformatik

Wien, im Dezember 1995.

Vorwort

Die vorliegende Arbeit bildet den Abschluß meines Studiums der Betriebsinformatik an der Technischen Universität Wien. Betriebsinformatik ist ein interdisziplinäres Fach, in gleichem Maße zuständig für sozialwissenschaftliche, wirtschaftswissenschaftliche und technisch/naturwissenschaftliche Themen. Diese thematische Vielfalt vermittelt im Lauf der Ausbildung die Fähigkeit, verschiedenartige Problemstellungen aus unterschiedlichen Aufgabenbereichen zu lösen. Damit wird es möglich, Problemstellungen und Lösungsverfahren verschiedener Forschungszweige zu verbinden und dadurch neue Problemlösungen zu schaffen. Doch gerade bei einem breiten Themenspektrum muß man sich gegen Ende des Studiums auf gewisse Fächer spezialisieren. Der Schwerpunkt meines Studiums läßt sich durch folgende Aufgabe beschreiben: „Technische und wirtschaftliche Probleme mit Computerunterstützung lösen." Vor allem Lehrveranstaltungen aus Betriebswirtschaftslehre, Operations Research, Softwaredesign und Programmiersprachen vermittelten mir das dafür nötige Handwerkszeug.

Die Aufgabenstellung der Diplomarbeit entspricht dem oben beschriebenen Studienschwerpunkt: Ein wirtschaftlich relevantes, sehr schwieriges Problem wird mit Methoden der Informatik gelöst. Das Thema wird nicht nur theoretisch betrachtet, vielmehr ist das Endprodukt ein lauffähiges Computerprogramm, das auf jedem Personal Computer ausgeführt werden kann. Beim Erstellen eines Computerprogramms muß jedes noch so kleine Detail exakt gelöst werden, andernfalls arbeitet das gesamte Programm nicht korrekt. Der arbeitsintensivste Teil der Diplomarbeit war demzufolge das Erzeugen des auf der beiliegenden Diskette gespeicherten Programmcodes. Aber auch die umfangreiche Literatursuche und das Studium dieser Literatur fließen nur indirekt (und in Form des Literaturverzeichnisses...) in die Arbeit ein. Die schriftliche Diplomarbeit hingegen ist lediglich eine Dokumentation und Zusammenfassung der zuvor genannten Punkte.

An dieser Stelle möchte ich mich bei all jenen bedanken, die zum Entstehen dieser Arbeit beigetragen haben. Besonderer Dank gilt den stets freundlichen und bemühten Mitarbeitern diverser Bibliotheken, Dr. Gerhard Lehner, der die mühevolle Arbeit des Korrekturlesens auf sich genommen hat, meiner Familie und – last but not least – meiner geduldigen Ingrun.

Inhaltsverzeichnis

1 Einleitung

1.1 Vehicle Routing Problem: Definition, Ursprung und Entwicklung

Der Begriff „Vehicle Routing Problem“ bezeichnet folgende Problemstellung:

Gegeben sei eine Menge von Kunden und deren Nachfrage nach einem bestimmten Produkt.

Ein vorhandener Fuhrpark soll von einem zentralen Depot aus die Kunden beliefern.
Die Transportkapazität eines jeden Fahrzeugs ist begrenzt.
Alle Fahrzeuge beginnen und beenden ihre Liefertouren im Depot.
Die Transportkosten für alle existierenden Teilstrecken sind bekannt.
Jeder Kunde wird genau einmal von genau einem Fahrzeug beliefert.

Wie verlaufen die kostenminimalen Liefertouren?

Das Vehicle Routing Problem (VRP) – in der Literatur auch als Truck Dispatching, Vehicle Scheduling oder Delivery Problem bezeichnet – wurde erstmals im Jahr 1959 von Dantzig und Ramser formuliert (vgl. Dantzig, G. B., Ramser, J. H. (1959)). In ihrer Arbeit beschrieben sie ein heuristisches Lösungsverfahren mit dessen Hilfe die Wegstrecken, die Tanklastwagen bei der Versorgung von Tankstellen mit Treibstoff zurücklegen, minimiert werden sollten. Alle Tanklastwagen wurden in einem zentralen Depot mit Treibstoff beladen, verfügten über die selbe maximale Kapazität und mußten am Ende ihrer Liefertour wieder im Depot sein. Die von den Tankstellen nachgefragten Mengen waren bekannt. Jede Tankstelle wurde von genau einem Tanklastwagen beliefert. Es stand immer eine ausreichenden Menge von Tanklastwagen zur Verfügung, und es wurde nur eine Treibstoffsorte berücksichtigt.

Clarke und Wright verbesserten das von Dantzig und Ramser vorgeschlagene Lösungsverfahren (vgl. Clarke, G., Wright, J. W. (1964)). Ihr Artikel enthielt als erster die genauen Angaben einer größeren, realen Problemstellung, sowie die von ihrem Algorithmus erzeugte Lösung. Diese Vorgangsweise wurde in der Folge von vielen Forschern übernommen und lieferte damit eine wesentliche Grundlage für den Vergleich verschiedener Lösungsverfahren.

Ab Mitte der 1960er Jahre wurde VRP in zahlreichen wissenschaftlichen Arbeiten behandelt. Vor allem den heuristischen Lösungsverfahren wurde dabei viel Aufmerksamkeit zuteil. Exakte Lösungsverfahren wurden – wohl aufgrund der hohen Komplexität von VRP – eher

stiefmütterlich behandelt. Erst in letzter Zeit wurden auf diesem Gebiet größere Fortschritte erzielt.

Heute wird Vehicle Routing als Sammelbegriff für eine umfangreiche Klasse von Problemen gebraucht, deren kleinster gemeinsamer Nenner die obige verbale Problemformulierung ist. Einige interessante Beispiele der vielen in der Praxis auftauchenden und untersuchten Problemeinschränkungen und Problemerweiterungen werden im Abschnitt 1.3 angesprochen.

1.2 Praxisrelevanz

Wo liegt die praktische Bedeutung des Vehicle Routing Problem? Die offensichtlichste Anwendungsmöglichkeit ergibt sich aus der ursprünglichen Problemstellung und liegt in der Verteilung von physischen Gütern. Sei es beim Ausliefern von Tageszeitungen oder rasch verderblichen Nahrungsmitteln, beim Abholen von Schülern mit Schulbussen oder beim Versorgen einer großen Kaufhauskette mit Waren: Das „optimale“ Gestalten der zurückzulegenden Touren erscheint erstrebenswert.

Löst man sich etwas von den unter 1.1 verwendeten Begriffen, so wird VRP auf eine Reihe weiterer Problemstellungen anwendbar. Die Fahrzeuge müssen nicht unbedingt das sein, was wir im allgemeinen Sprachgebrauch unter Fahrzeug verstehen. In konkreten Modellen können Personen, beliebige körperliche Sachen (z.B. Werkzeuge, Maschinen) aber auch immaterielle Güter (z.B. Dienstleistungen) an die Stelle des Fahrzeugs treten. Das Ausliefern einer Ware kann man besser mit „Erbringen einer Leistung“ bezeichnen. Diese Leistung kann aus dem Übergeben bzw. Übernehmen einer Sache bestehen, sie kann jedoch ebenso das Erbringen einer Dienstleistung sein. Die Begriffe Depot, Kunde und Wegstrecke lassen sich natürlich auf die gleiche Weise verallgemeinern. Damit sind Lösungsalgorithmen für VRP beispielsweise auf das Routing von Daten in einem Netzverbund (Telephon, Computernetzwerke), auf das Planen von Überwachungsfahrten bei privaten Objektsicherungsfirmen oder die Einsatzplanung beliebiger Ressourcen anwendbar.

Aus wirtschaftswissenschaftlicher Sicht versucht man mit dem Einsatz von Optimierungsmethoden auf den zuvor beschriebenen Gebieten immer eine Kostenminimierung zu erreichen. Diese Kostenminimierung ergibt sich aus möglichst effizientem Einsatz bzw. möglichst effizienter Nutzung vorhandener Ressourcen. Rückt man

vom Postulat der Kostenminimierung ab, so eröffnen sich weitere Anwendungsgebiete: Mit geeigneten Modifizierungen der Kosten- und Zielfunktion könnte etwa eine Minimierung des Schadstoffausstoßes oder des Ressourceneinsatzes erreicht werden.

1.3 Varianten von VRP

Die Vielzahl der (potentiellen) Anwendungsmöglichkeiten ist eine Erklärung für die intensive Behandlung von VRP in der Literatur einerseits, sowie die große Anzahl von untersuchten Varianten und Nebenbedingungen andererseits. In diesem Abschnitt werden einige interessante Einschränkungen und Erweiterungen, die bereits untersucht wurden und werden, kurz angesprochen:

1. Die zur Verfügung stehenden Fahrzeuge haben unterschiedliche Transportkapazitäten und/oder verursachen unterschiedliche Kosten. Man spricht in diesem Zusammenhang von einer heterogenen Transportflotte.

2. Es sind nur Wahrscheinlichkeitsverteilungen der nachgefragten Mengen bekannt.

3. Es werden verschiedene Produkte gleichzeitig ausgeliefert. Bei dieser Problemstellung müssen bestimmte Produkt- oder Fahrzeugeigenschaften berücksichtigt werden: Unterschiedliche Treibstoffe müssen in getrennten Tanks transportiert werden, Verpackungsmaße mit dem Frachtraumvolumen übereinstimmen, etc.

4. Die Touren bestehen sowohl aus Auslieferungen als auch aus Rückholungen (z.B. saubere Wäsche und Schmutzwäsche). Diese unterschiedlichen Aktivitäten können entweder von einem anderen Fahrzeug übernommen werden, oder sie werden innerhalb einer Tour gemischt bzw. nacheinander ausgeführt.

5. Anstelle eines zentrales Depots existieren mehrere, räumlich getrennte Depots. Start- und Enddepot der Fahrzeuge könnten in diesem Fall verschieden sein. Es steht nicht mehr fest, von welchem Depot aus welcher Kunde versorgt wird. Ein weiteres Problem in diesem Zusammenhang ist das Festlegen von Standorten für Depots („depot location“), das die im laufenden Betrieb entstehenden Kosten stark beeinflussen kann.

6. Bestimmte Ressourcen haben Restriktionen in zeitlicher Hinsicht („time windows“): Jeder Kunde nimmt die Lieferung nur innerhalb bestimmter Zeitperioden entgegen (Terminvereinbarungen mit Kunden, vertraglich fixierte Lieferzeitpunkte), die Fahrzeuge bzw. Fahrer sind nur in gewissen Perioden einsatzbereit (Mittagspausen, gesetzlich festgelegte Arbeitszeitbestimmungen) oder aber die Verbindungen zwischen manchen Kunden existieren nur innerhalb bestimmter Zeitperioden (Schiffsverbindungen, Flugverbindungen oder Computernetzwerke, die nicht rund um die Uhr in Betrieb sind).

7. Es werden zeitliche Aspekte explizit berücksichtigt: Wartezeiten beim Beladen im Depot, beim Entladen, unterschiedliche Fahrtdauern in Abhängigkeit der Tageszeit (Stoßzeiten), etc.

Die hier genannten Aspekte illustrieren wie weit gestreut die einzelnen Aufgaben unter dem Sammelbegriff VRP sind; und doch bilden sie nur einen kleinen Ausschnitt der tatsächlich in der Literatur behandelten Problemstellungen.

2 Modellformulierungen

In diesem Kapitel werden – stellvertretend für die zahlreichen existierenden – zwei verschiedene mathematische Modellformulierungen von VRP zitiert.

2.1 Diskrete Programmierung

Die folgende Formulierung von VRP als Problem der diskreten Programmierung wurde zu Beginn der 1980er Jahre von Fisher und Jaikumar erstellt (vgl. Fisher, M. L., Jaikumar, R. (1981)). Aufbauend auf dieser und etlicher leicht abgewandelter Formulierungen wurden viele Lösungsverfahren entwickelt.

Konstanten

K Anzahl der zur Verfügung stehenden Fahrzeuge.

n Anzahl der Kunden, die beliefert werden müssen. Die Kunden sind von 1 bis n indiziert, Index 0 ist das Depot.

b_k Maximale Kapazität von Fahrzeug k.

a_i Von Kunde i nachgefragte Menge.

c_{ij} Transportkosten für die Strecke zwischen Kunde i und Kunde j.

Variablen

y_{ik} Nimmt den Wert 1 an, falls Kunde i von Fahrzeug k beliefert wird, andernfalls 0.

x_{ijk} Nimmt den Wert 1 an, falls das Fahrzeug k direkt von Kunde i zu Kunde j fährt, andernfalls 0.

(1) $\text{Minimiere } \sum_{ijk} c_{ij} x_{ijk}$

(2) $\sum_{i} a_i y_{ik} \le b_k, \qquad k = 1,\ldots,K$

(3) $\sum_{k} y_{ik} = \begin{cases} K & i = 0 \\ 1, & i = 1,\ldots,n \end{cases}$

(4) $y_{ik} \in \{0,1\}, \qquad (i = 0,\ldots,n) \wedge (k = 1,\ldots,K)$

(5) $\sum_{i} x_{ijk} = y_{jk}, \qquad (j = 0,\ldots,n) \wedge (k = 1,\ldots,K)$

(6) $\sum_{j} x_{ijk} = y_{ik}$, $(i = 0,...,n) \wedge (k = 1,...,K)$

(7) $\sum_{ij \in S \times S} x_{ijk} \leq |S| - 1,$ $(S \subseteq \{1,...,n\}) \wedge (2 \leq |S| \leq n-1) \wedge (k = 1,...,K)$

(8) $x_{ijk} \in \{0,1\},$ $(i = 0,...,n) \wedge (j = 0,...,n) \wedge (k = 1,...,K)$

Durch die Zielfunktion (1) soll die Summe der Transportkosten, welche durch das Befahren der Routen aller einzusetzender Fahrzeuge entstehen, minimiert werden. Die Kapazitätsrestriktionen (2) verhindern, daß ein Fahrzeug auf seiner Route mehr ausliefern muß, als es befördern kann. Die Gleichungen (3) legen fest, daß alle Kunden von genau einem, das Depot jedoch von allen einzusetzenden Fahrzeugen besucht werden; Gleichungen (4) definieren y_{ik} als binäre Variablen. (5) und (6) sind Bedingungen für zusammenhängende Routen. Die Restriktionen (7) sind die bekannten „subtour elimination inequalities“, und durch die Bedingungen (8) werden auch die x_{ijk} zu binären Variablen.

Man beachte: Während (2) bis (4) die Restriktionen eines Generalized Assignment Problems sind, definieren (5) bis (8) für eine beliebige Wertzuweisung der y_{ik}, die (2) - (4) erfüllt, ein Traveling Salesman Problem (TSP).

2.2 Dynamische Programmierung

Von Christofides, Mingozzi und Toth stammt die, ebenfalls zu Beginn der 1980er Jahre veröffentlichte, Formulierung von VRP als Modell der dynamischen Programmierung (vgl. Christofides, N., Mingozzi, A., Toth, P. (1981a)). Die Autoren sehen ihr Modell allerdings nicht als taugliche Grundlage für ein allgemeines Lösungsverfahren. Vielmehr stellen sie in ihrer Arbeit eine Methode vor, welche das Berechnen von (unteren) Schranken für die Kosten von VRP Lösungen gestattet. Diese Methode kann beispielsweise in Branch & Bound Verfahren eingebettet werden.

Der Graph $G = (X, A)$ ist definiert durch die Menge seiner Knoten X und die Menge seiner ungerichteten Kanten A. Die Teilmenge $X' = \{x_i | i = 2,...,n\}$ von X beinhaltet die zu beliefernden Kunden, x_1 ist das zentrale Depot. $X = X' \cup \{x_1\}$. Jeder Kunde fragt die Menge q_i nach. Es stehen M gleichartige Fahrzeuge mit jeweils maximaler Kapazität Q zur Verfügung.

Die Transportkosten von Kunde i zu Kunde j sind in c_{ij} gegeben. v(S) gibt die Transportkosten der optimalen Lösung des TSP an, das durch die Kunden in S und das Depot gebildet wird. $f(m,S)$ sind die geringsten möglichen Kosten, um die (Teil-)Menge S von Kunden mit m Fahrzeugen zu beliefern.

(1) $$f(m,S) = \min_{L \subset S}\left[f(m-1,S-L) + v(L)\right] \qquad (m = 2,\ldots,M) \wedge (S \subseteq X')$$

(2) $$f(1,S) = v(S) \qquad S \subseteq X'$$

(3) $$\sum_{x_i \in S} q_i - (m-1)\cdot Q \le \sum_{x_i \in L} q_i \le Q \qquad (m = 2,\ldots,M) \wedge (S \subseteq X')$$

(4) Für die $S \subseteq X'$ muß gelten: $$\left(\sum_{x_i \in X'} q_i\right) - (M-m)\cdot Q \le \sum_{x_i \in S} q_i \le m \cdot Q$$

Gleichungen (1) und (2) definieren die rekursive Zielfunktion. Die Anzahl der verwendeten Fahrzeuge beschreibt darin als Zustandsvariable die einzelnen Stufen des dynamischen Systems. Als Entscheidungsvariable dient die Menge der Kunden, die in der jeweils betrachteten Stufe einem bestimmten Fahrzeug zugeordnet wird. Die Nebenbedingung (3) stellt die Kapazitätsrestriktion sowohl für jene $(m-1)$ Fahrzeuge, welche die Kunden $(S-L)$ bedienen, als auch für jenes Fahrzeug, dessen Kundenkreis L ist, dar. Ungleichung (4) rechts fordert, daß m Fahrzeuge die Nachfrage aller Kunden aus S transportieren können. (4) links achtet darauf, daß die Nachfrage aller Kunden, die nicht in S sind, von den verbleibenden $(M - m)$ Fahrzeugen befördert werden kann.

Obwohl dieses Modell zweifellos sehr elegant ist, konnte es keine praktische Bedeutung erlangen. Der Grund dafür liegt im riesigen Lösungsraum („curse of dimensionality“, vgl. dazu Stepan, A., Fischer, E. O. (1992), S. 199), den der zugehörige Zustandsgraph schon bei relativ kleinen Problemen bildet. Es gibt bis heute kein allgemeines Lösungsverfahren, das den abzusuchenden Zustandsgraphen auf eine Dimension einschränkt, welche „bewältigbar“ ist.

3 Lösungsverfahren

Die Mathematik ist das Alphabet,
mit dem Gott die Welt geschrieben hat.
(Galileo Galilei)

Dieses Kapitel erläutert im Abschnitt 3.1 die Rahmenbedingungen für Verfahren, um Lösungen des VRP zu berechnen. Unter 3.2 und 3.3 werden zwei verschiedene, grundlegende Möglichkeiten das VRP zu lösen gegenübergestellt. Der umfangreichste Teilabschnitt 3.4 beschreibt letztendlich jenen Algorithmus, der im Rahmen dieser Diplomarbeit implementiert wurde.

3.1 Komplexität des VRP

Die Komplexitätstheorie analysiert Probleme und Algorithmen bezüglich der von ihnen benötigten Ressourcen. (Die Bedeutung der Komplexitätstheorie für Operations Research wird in Stockmeyer, L. J. (1992) bzw. Bomze, I. M., Grossmann, W. (1993) ausführlich dargelegt; vgl. zu den folgenden Ausführungen auch Leitsch, A., Salzer, G. (1993).) Unter einem Problem versteht man dabei eine allgemeine Frage, die beantwortet werden soll; ein Algorithmus ist ein Verfahren zum Beantworten einer bestimmten Frage. Üblicherweise werden die beiden Ressourcen Laufzeit und Speicherbedarf betrachtet. Die Komplexität des (hinsichtlich einer bestimmten Ressource) effizientesten Algorithmus zur Lösung eines gewissen Problems bestimmt die Komplexität dieses Problems.

Die sog. Zeitkomplexitätsfunktion eines Algorithmus ordnet jeder möglichen Größe[1] von Probleminstanzen den maximalen Zeitaufwand zu, den der Algorithmus benötigt, um Probleminstanzen der jeweiligen Größe zu lösen. Eine (Zeitkomplexitäts-) Funktion $f(n)$ ist von Ordnung $g(n)$ – kurz: $f(n)$ ist $O(g(n))$ – wenn es eine Konstante c gibt, sodaß $|f(n)| \leq c \cdot |g(n)|$ für fast alle $n \geq 0$. Ein polynomieller Algorithmus hat eine Zeitkomplexitätsfunktion $O(p(n))$ wobei $p(n)$ ein Polynom in n ist. Die Zeitkomplexitätsfunktion eines exponentiellen Algorithmus kann hingegen nicht durch ein

[1] Die Größe einer Probleminstanz ist definiert durch die Länge jener Zeichenkette, die ensprechend den Regeln eines bestimmten Kodierungsschemas gebildet wurde und die Probleminstanz eindeutig beschreibt.

derartiges Polynom beschränkt werden. Tabelle 3.1 (vereinfacht, aus: Leitsch, A., Salzer, G. (1993)) stellt die Zeitkomplexität mehrerer polynomieller und exponentieller Algorithmen bei verschiedenen Instanzengrößen gegenüber. Aufgrund ihres extremen Wachstums und der damit verbundenen hohen Laufzeit sind exponentielle Algorithmen beim heutigen Stand der Technik zum Lösen großer Probleminstanzen nicht geeignet.

Komplexität	Instanzengröße n							
	10		30		40		60	
n	.00001	Sek.	.00003	Sek.	.00004	Sek.	.00006	Sek.
n^2	.0001	Sek.	.0009	Sek.	.0016	Sek.	.0036	Sek.
n^3	.001	Sek.	.027	Sek.	.064	Sek.	.216	Sek.
n^5	.1	Sek.	24.3	Sek.	1.7	Min.	13.0	Min.
2^n	.001	Sek.	17.9	Min.	12.7	Tage	366	Jhdte.
3^n	.059	Sek.	6.5	Jahre	3855	Jhdte.	1.3e13	Jhdte.

Tabelle 3.1 - Vergleich einiger polynomieller und exponentieller Komplexitätsfunktionen.

Probleme, für die man nachweisen kann, daß es keinen polynomiellen Algorithmus zu ihrer Lösung gibt, heißen polynomiell unlösbar. Polynomielle Unlösbarkeit entsteht entweder durch eine Lösungsmenge mit exponentieller Mächtigkeit oder dadurch, daß das Finden einer Lösung nur mit exponentiellem Zeitaufwand möglich ist. Manche Probleme sind nicht nur polynomiell unlösbar, sondern sogar unentscheidbar: Es gibt keinen einzigen Algorithmus zu ihrer Lösung. Polynomiell lösbare Probleme gehören einer von zwei Gruppen an: Die eine Gruppe beinhaltet Probleme, die mit einem deterministischen Algorithmus in polynomieller Zeit lösbar sind; bei Mitgliedern der zweiten Gruppe läßt sich diese Lösbarkeit nur durch einen indeterministischen Algorithmus erreichen. Abbildung 3.1 verdeutlicht die soeben angesprochenen Einteilungskriterien.

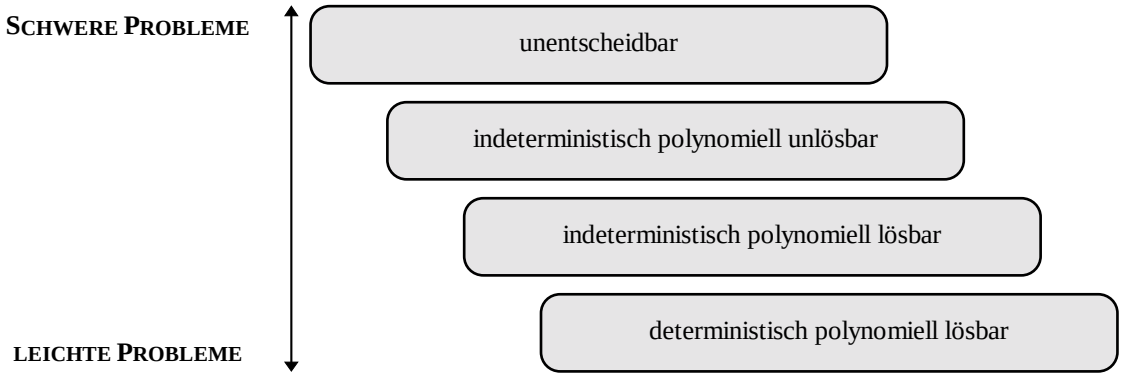


Abbildung 3.1 - Klassifizierung von Problemen.

Die Klasse P (P ← „polynomial“) beinhaltet alle Entscheidungsprobleme für die es einen polynomiellen Algorithmus gibt, der alle Instanzen des jew. Entscheidungsproblems löst. Ein Entscheidungsproblem gehört der Klasse NP (NP ← „nondeterministic polynomial“) an wenn ein polynomieller Algorithmus existiert, der feststellen kann, ob eine zu überprüfende Lösung des jew. Entscheidungsproblems tatsächlich eine Lösung ist. Ein Problem P wird als „NP-vollständig“ bezeichnet, wenn jedes andere NP-vollständige Problem Q durch eine polynomielle Transformation auf P zurückgeführt werden kann. NP-vollständige Probleme gelten als besonders schwierig zu lösen. Die genaue Beziehung der Klassen P, NP und NP-vollständig konnte noch nicht nachgewiesen werden; man nimmt jedoch an, daß P ≠ NP gilt. In Abbildung 3.2 wird diese vermutete Struktur dargestellt. Bislang konnte für kein NP-vollständiges Problem ein polynomieller Algorithmus gefunden werden. Falls dies für ein einziges derartiges Problem gelingen würde, wäre (entgegen der herrschenden Meinung) die Gleichung P = NP bewiesen.

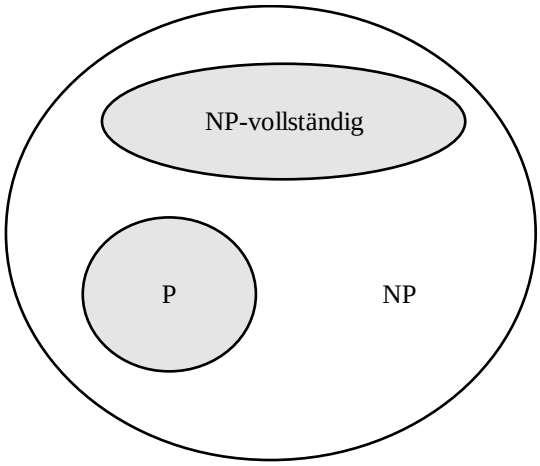


Abbildung 3.2 - Vermutete Beziehung der Problemklassen P, NP und NP-vollständig.

Die obigen Ausführungen stecken in Verbindung mit der Tatsache, daß das Vehicle Routing Problem NP-vollständig ist (vgl. Lenstra, J. K., Rinnoy Kan, A. H. G. (1981)), das Terrain ab, auf dem man sich beim Entwickeln und Implementieren von Lösungsverfahren für VRP bewegt. Es existiert vermutlich kein Algorithmus, der jede Instanz von VRP in polynomiell beschränkbarer Zeit exakt löst. Die Überlegungen der Komplexitätstheorie gehen jedoch vom schlimmsten möglichen Fall aus. Es ist daher im allgemeinen möglich, Lösungsalgorithmen zu konstruieren, die im Schnitt ein sehr viel besseres als das erwartete exponentielle Laufzeitverhalten zeigen. Viele Anwendungen benötigen darüber hinaus nicht unbedingt eine

optimale Lösung sondern begnügen sich mit Näherungen an das globale Optimum. Diese eingeschränkten Anforderungen können oft von schnellen, heuristischen Lösungsverfahren ausreichend erfüllt werden.

3.2 Exakte Lösungsverfahren

Im Abschnitt 3.1 wurde aufgezeigt, daß das Entwickeln von exakten Optimierungsverfahren für das VRP zu den schwierigsten Aufgaben im Bereich des Operations Research gehört. Nicht zuletzt aus diesem Grund existiert vergleichsweise wenig Literatur auf diesem Gebiet. Für Beispiele vgl. Christofides, N., Mingozzi, A., Toth, P. (1981, 1981b), Kolen, A. W. J., Rinnoy Kan, A. H. G., Trienekens, H. W. J. M. (1987) oder Desrochers, M., Desrosiers, J., Solomon, M. (1992).

Alle genannten Lösungsverfahren basieren auf sog. Branch & Bound Verfahren. Mit Branch & Bound Methoden löst man diskrete Optimierungsprobleme indem man die Menge der zulässigen Lösungen systematisch in immer kleiner werdende Teilmengen aufteilt. Für jede dieser Teilmengen wird eine Schranke für den Zielfunktionswert berechnet. Mithilfe der dadurch gewonnenen Information versucht man, möglichst große Lösungsteilmengen zu möglichst frühen Zeitpunkten in der Berechnung von weiterer Betrachtung auszuschließen. Mit anderen Worten: Man trachtet danach, schnell eine „gute“ bzw. optimale Lösung zu finden. Alle Lösungsteilmengen, deren Zielfunktionsschranke schlechter als der beste bisher gefundene Lösungswert ist, müssen nicht weiter untersucht werden. Im Endeffekt muß nicht der gesamte, i.d.R. sehr große Lösungsraum durchsucht werden.

Abbildung 3.3 beschreibt die allgemeine Funktionsweise von Branch & Bound Verfahren. Folgende Elemente müssen vorhanden sein, um ein Problem $P = \min[f(x)|x \in S]$ lösen zu können:

1. Ein abgeschwächtes Problem („relaxation“) $Q = \min[g(x)|x \in T]$ zu P, sodaß $S \subseteq T$ und für alle $x, y \in S$ gilt $f(x) > f(y) \Rightarrow g(x) > g(y)$. Die verwendete Problemabschwächung ist der wichtigste Bestandteil jedes Branch & Bound Verfahrens. Sie soll einerseits leicht zu berechnen sein, andererseits eine gute Schranke liefern.

2. Eine Verzweigungsregel („branching rule“), die den Lösungsraum S_i jedes Subproblems P_i in k disjunkte Teilmengen S_{ij} aufteilt, sodaß $\bigcup_{j=1}^{k} S_{ij} = S_i$.
3. Eine Methode, um die optimale Lösung $g^*(x)$ zu jedem abgeschwächten Problem Q_i zu berechnen bzw. um das Optimum zu approximieren.
4. Eine Regel, die das nächste zu untersuchende Subproblem auswählt („subproblem selection rule“).

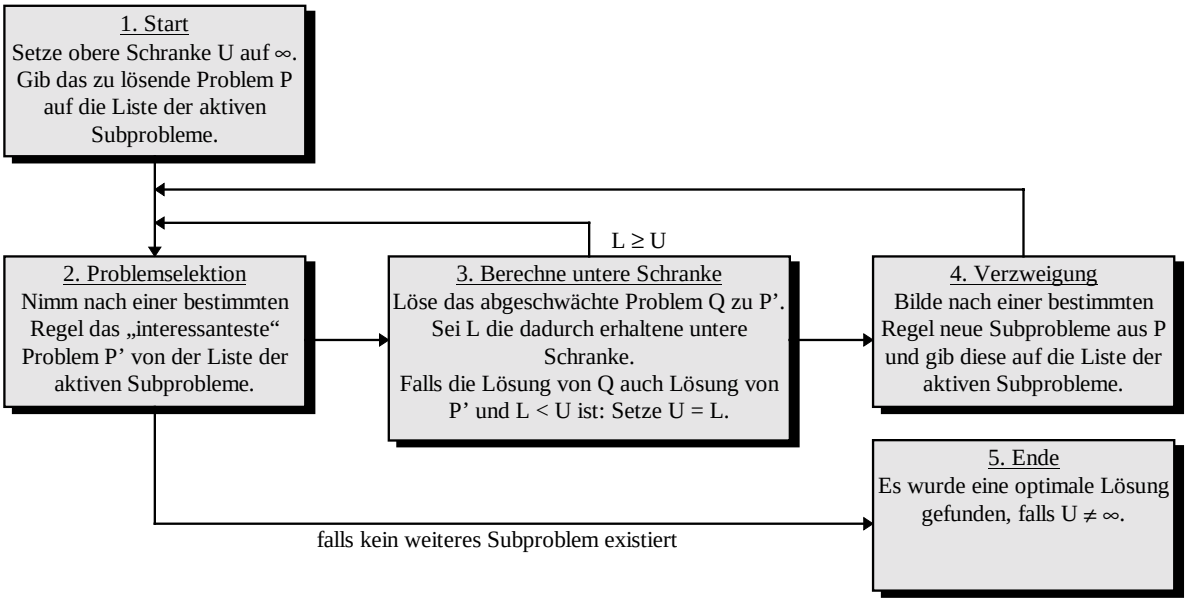


Abbildung 3.3 - Funktionsweise von Branch & Bound Verfahren.

Die bei VRP Branch & Bound Algorithmen verwendeten Problemabschwächungen sind recht unterschiedlich: Sie reichen von Spanning Tree über Shortest Path bis hin zu State Space Relaxations. Obwohl die zitierten Lösungsverfahren bereits sehr komplex sind, können damit nur relativ kleine Probleme – in der Größenordnung von unter 100 Kunden – in annehmbarer Zeit gelöst werden.

3.3 Näherungsweise Lösungsverfahren

Im Gegensatz zu exakten Methoden kann beim Einsetzen heuristischer Lösungsverfahren nicht garantiert werden, daß eine optimale Lösung gefunden wird. Meist ist es jedoch möglich, Zeitschranken bezüglich der benötigten Rechenzeit und/oder die maximale Abweichung vom optimalen Zielfunktionswert anzugeben. Durch diese erstrebenswerten

Eigenschaften können Heuristiken überall dort eingesetzt werden, wo eine „beinahe“ optimale Lösung zum Erfüllen einer Aufgabe ausreicht oder wo das Lösen einer Optimierungsaufgabe einen bestimmten zeitlichen Rahmen nicht überschreiten darf.

Es existieren zahlreiche Möglichkeiten, heuristische Algorithmen für das VRP zu konstruieren. Stellvertretend für die vielen existierenden Lösungsansätze seien die folgenden Arbeiten genannt: Von Clarke und Wright (vgl. Clarke, G., Wright, J. W. (1964)) stammt der klassische Savings Algorithmus; Fisher und Jaikumar (vgl. Fisher, M. L., Jaikumar, R. (1981)) ordnen im ersten Schritt ihres Verfahrens den Fahrzeugen Kunden zu und bilden in einem zweiten Schritt optimale Fahrtrouten; Osman (vgl. Osman, I. H. (1993)) vergleicht Algorithmen mit den relativ neuen Lösungsansätzen „Simulated Annealing“ und „Tabu Search“.

Altinkemer und Gavish klassifizieren in ihrer Arbeit (Altinkemer, K., Gavish, B. (1991)) bestehende Heuristiken nach der Methode mit der Lösungen aufgebaut werden:

1. Cluster first-route second: Im ersten Schritt werden Zuordnungen zwischen Fahrzeugen und Kunden getroffen (Es werden sog. „clusters“ gebildet.), im zweiten Schritt optimiert man die Rundreise innerhalb jedes Clusters.
2. Route first-cluster second: Hier werden im ersten Schritt möglichst gute Rundreisen gebildet, die aber i.d.R. bestimmte Restriktionen nicht erfüllen. Im zweiten Schritt entfernt man solange Kunden aus den Rundreisen und bildet aus den entfernten Kunden neue Rundreisen, bis eine zulässige Lösung erreicht wird.
3. Savings method: Verfahren aus dieser Gruppe generieren zunächst eine Startlösung (z.B. jedes Fahrzeug bedient einen einzigen Kunden) und bilden danach iterativ verbesserte Lösungen. Als Maß für mögliche weitere Verbesserungen dienen „savings“ (z.B. die Ersparnis die dadurch entsteht, daß ein Fahrzeug auf einer Tour zwei Kunden bedient).
4. Improvement method: Man erreicht – ausgehend von einer beliebigen Startlösung – ein lokales Optimum indem man Teile der einzelnen Touren untereinander austauscht. Oft führt man das Verbesserungsverfahren für mehrere Startlösungen durch und wählt die beste der gefundenen Lösungen aus.
5. Mathematical programming.

6. Interactive optimization: Der Dialog zwischen dem menschlichen Optimierungsexperten und der Maschine charakterisiert interaktive Verfahren. Der Anwender kann an verschiedenen Punkten in der Berechnung Parameter setzen, welche die Richtung der weiteren Lösungssuche bestimmen. Mit menschlicher Hilfe kann das Verfahren somit flexibel auf nicht modellierte Umstände Rücksicht nehmen. Da der Mensch auch weiterhin eine Steuerungsfunktion inne hat, stoßen interaktive Optimierungssysteme oft auf wesentlich höhere Akzeptanz.

3.4 Die Heuristik von Altinkemer und Gavish

Die folgenden Unterabschnitte enthalten die mathematischen Grundlagen jenes Optimierungsverfahrens, das im Rahmen dieser Arbeit implementiert und analysiert wurde. Es handelt sich dabei um ein heuristisches Lösungsverfahren, das von Altinkemer und Gavish vorgestellt wurde („parallel savings based heuristic“; vgl. Altinkemer, K., Gavish, B. (1991)). Die Funktionsweise der Heuristik wird unter 3.4.1 beschrieben. Zwei bekannte Subprobleme, die im Zug der Berechnung entstehen, sowie deren Lösungswege werden in den Abschnitten 3.4.2 und 3.4.3 diskutiert.

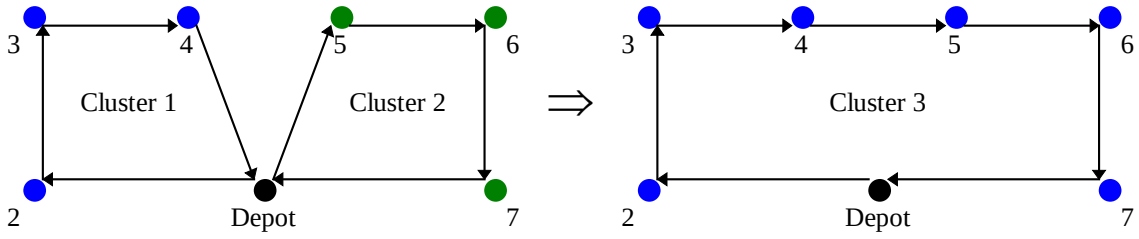
3.4.1 Parallel Savings Algorithm

Gegeben sei der n-dimensionale vollständige, gewichtete, ungerichtete Graph $G = \langle V, E, C \rangle$, der n-dimensionale Nachfragevektor $\vec{d} = (d_1, d_2, ..., d_n)$, die maximale Transportkapazität Q der $(n-1)$ gleichartigen verfügbaren Fahrzeuge, sowie eine Steuervariable $T \in \{1, ..., n\}$. Die Menge der Knoten $V = V_D \cup V_C$ setzt sich zusammen aus dem zentralen Depot $V_D = \{1\}$ und der Menge der Kunden $V_C = \{2, ..., n\}$; der Knoten 1 wird also immer als Depot interpretiert. E steht für die Menge der Kanten aus G. Die (mengenunabhängigen) „Transportkosten“, die beim Transit von Knoten i zu Knoten j entstehen, gehen aus der n-dimensionalen symmetrischen Kostenmatrix C hervor: $c_{ij} \geq 0$. Der Nachfragevektor enthält jene Mengen $d_i \geq 0$, die vom Depot an die jeweiligen Knoten geliefert werden sollen. Das Depot hat immer eine Nachfrage von 0.

Unter einem **Cluster** versteht man einen Teilgraphen $G' = \langle V', E', C' \rangle$ von G, der einem einzelnen Fahrzeug zugeordnet ist. Die Knotenmenge $V' = V_D \cup V_C'$ jedes Clusters besteht aus

dem Depot und einer nichtleeren Teilmenge der Kunden $V'_C \subseteq V_C$, $|V'_C| > 0$. Jene Kanten aus E, welche die Knoten aus V′ verbinden, bilden die Kantenmenge $E' = \{e_{ij} \in E | i, j \in V'\}$. Analog dazu enthält C′ die Gewichte der Kanten aus E′. Der kleinstmögliche Cluster enthält demnach das Depot und einen Kunden. Der größtmögliche Cluster ist der ursprüngliche Graph G; er enthält das Depot und alle Kunden. Die **Kosten** $TSP(i)$ **eines Clusters** i werden durch eine kostenminimale Liefertour bestimmt, auf der, ausgehend vom Depot, alle Kunden genau einmal besucht werden. Das Bestimmen dieser Kosten ist ein nichttriviales Problem und wird in der Literatur als **Traveling Salesman Problem** (TSP) intensiv behandelt. Ein Lösungsalgorithmus wird im Abschnitt 3.4.2 behandelt.

Jeder Cluster bestimmt also die Liefertour eines Fahrzeuges. Vereinigt man zwei verschiedene Liefertouren i, j zu einer einzigen, größeren Liefertour $k = i \cup j$, so kann man eine Kostenersparnis erzielen (vgl. dazu Abbildung 3.4; alle Kantengewichte sind 1). Diese Kostenersparnis wird als **Saving** bezeichnet und ist definiert als $S(i, j) = TSP(i) + TSP(j) - TSP(i \cup j)$. Voraussetzung für das Zusammenlegen der Liefertouren ist, daß die Kapazitätsrestriktion nicht verletzt wird: Die Gesamtnachfrage der Kunden im neuen Cluster darf die Transportkapazität des Lieferfahrzeuges nicht übersteigen.



Transportkosten Cluster 1: 4
Transportkosten Cluster 2: 4

Transportkosten Cluster 3: 7

Abbildung 3.4 - Kostenersparnis durch das Zusammenfassen zweier Cluster.

Der Parallel Savings Algorithm (PSA) berechnet – ausgehend von einer teuren, zulässigen Startlösung – in einem iterativen Verfahren solange billigere Lösungen, bis keine Verbesserung mehr zulässig ist. Als Startlösung dienen $(n-1)$ Cluster, die aus dem Depot und jeweils einem Kunden bestehen. Abbildung 3.5 zeigt ein Beispiel für eine derartige Startlösung, bei der jeder Kunde von einem eigenen Fahrzeug beliefert wird. Verbesserungen werden durch das Zusammenlegen zweier Cluster zu einem größeren Cluster erzielt. Seien in

einem allgemeinen Iterationsschritt insgesamt m Cluster vorhanden. Es gibt dann genau $(m\cdot(m-1))/2$ verschiedene Möglichkeiten, Paare zu bilden (Cluster zu vereinigen). Maximal $\lfloor m/2 \rfloor$ Paare können gleichzeitig gebildet werden. Als Maßzahl für die Verbesserung, die durch eine Paarbildung erzielt werden kann, werden die oben definierten Savings herangezogen; unzulässige Paarbildungen bringen keine Lösungsverbesserung und werden daher mit 0 bewertet. Der Steuerparameter T bestimmt wie viele Cluster in jedem Iterationsschritt vereinigt werden. Natürlich muß $T \leq \lfloor m/2 \rfloor$ gelten. Es stellt daher folgendes Problem: Bilde aus m vorhandenen Cluster T disjunkte Paare, sodaß die gesamte durch das Zusammenlegen der Cluster erzielbare Kostenersparnis maximal ist.

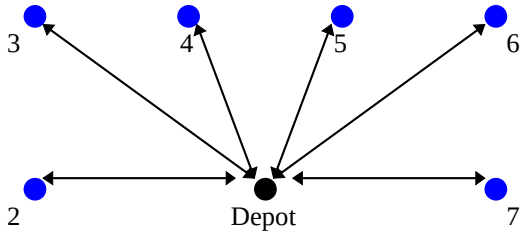


Abbildung 3.5 - Startlösung für PSA.

Diese Problemstellung heißt **Maximum Weight Matching Problem** (MWMP). Ein Lösungsalgorithmus wird im Abschnitt 3.4.3 besprochen. Lösungen des MWMP enthalten jedoch i.d.R. $\lfloor m/2 \rfloor$ Paare, während PSA nur T Paare sucht. Um dieses Hindernis zu umgehen, schlagen Altinkemer und Gavish vor, für das Lösen des MWMP $(m-2\cdot T)$ künstliche Cluster hinzuzufügen. Die Savings bei künstlichen Cluster müssen so beschaffen sein, daß Paarbildungen zwischen einem künstlichen und einem ursprünglichen Cluster Vorrang vor Paarbildungen zwischen zwei ursprünglichen Cluster haben. Als Lösung des MWMP ergeben sich dann $(m-2\cdot T)$ Paare aus einem künstlichen und einem ursprünglichen Cluster sowie T Paare aus zwei ursprünglichen Cluster. Nur die letztgenannten T Paare werden tatsächlich vereinigt. Für $S(i,j)$ ergibt sich daraus:

$$S(i,j) = \begin{cases} 0 \ldots\ldots \text{Cluster i und Cluster j sind künstlich} \\ 0 \ldots\ldots \text{Kapazitaetsrestriktion verletzt} \\ \infty \ldots\ldots \text{Cluster i oder Cluster j ist künstlich} \\ TSP(i) + TSP(j) - TSP(i \cup j) \ldots \text{weder i noch j sind künstlich} \end{cases}$$

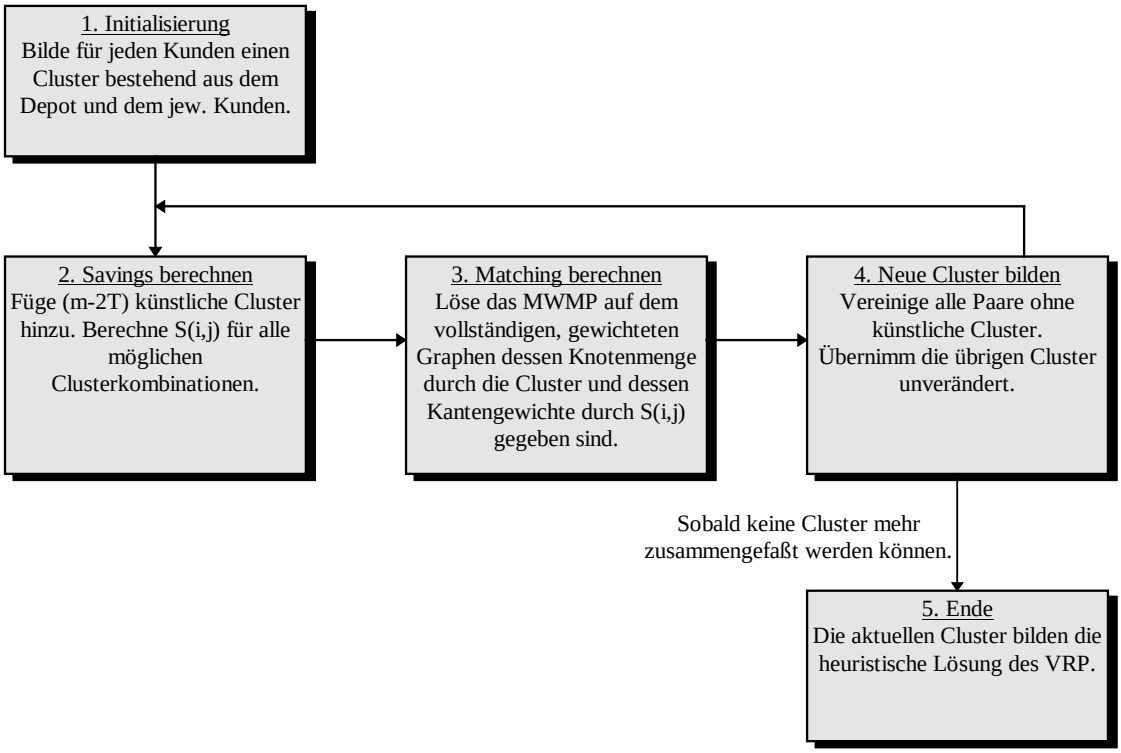


Abbildung 3.6 - Funktionsweise des VRP Lösungsalgorithmus.

Die Ablaufstruktur des gesamten Lösungsalgorithmus ist in Abbildung 3.6 zusammengefaßt. Abbildung 3.7 verdeutlicht den Zusammenhang zwischen PSA und den beiden Subproblemen, die in jeder Iteration gelöst werden müssen:

1. Um die Savings zu berechnen, muß für jede mögliche Clusterkombination mindestens ein Traveling Salesman Problem gelöst werden.
2. Die Lösung des TSP liefert für jeden (möglichen) Cluster eine optimale Fahrtroute und die Gesamtkosten dieser Fahrtroute.
3. PSA verwaltet die ursprünglichen und künstlichen Cluster und bestimmt in jedem Iterationsschritt die symmetrische Matrix der Savings.
4. Die Lösung des MWMP bestimmt, welche Cluster im aktuellen Iterationsschritt zusammengefaßt werden.

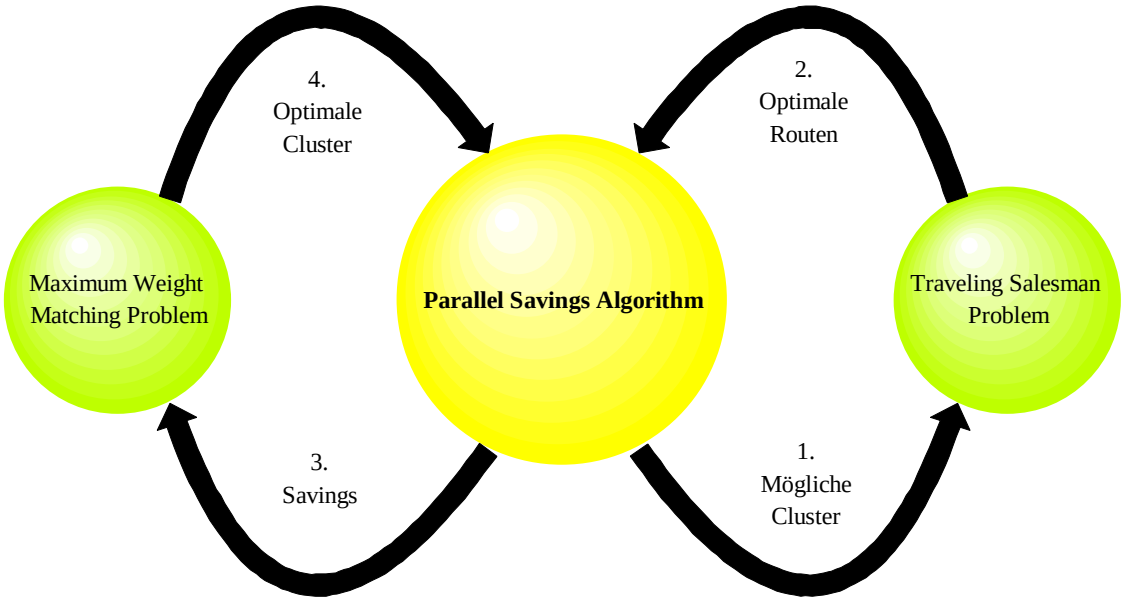


Abbildung 3.7 - Zusammenhang der Subprobleme.

3.4.2 Subproblem #1: Traveling Salesman Problem

Das erste Subproblem bei der Berechnung der Savings besteht darin, optimale Fahrtrouten für bestimmte Teilmengen der VRP Kostenmatrix zu berechnen. Diese Aufgabenstellung ist als „Traveling Salesman Problem“ (Handelsreisendenproblem, TSP) bekannt. Das TSP ist ebenso wie das VRP NP-vollständig; es wurde und wird in der Literatur sehr ausführlich behandelt (Für einen Überblick vgl. Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B. (Hrsg.) (1985).) und kann getrost als das Paradeproblem der kombinatorischen Optimierung bezeichnet werden. Sei $G=(V,A)$ ein vollständiger, ungerichteter Graph mit Knoten $V=\{1,\ldots,n\}$ und Kanten $A=\{(ij)|i,j=1,\ldots,n \wedge i<j\}$. Jeder Kante seien nichtnegative Kosten c_{ij} zugeordnet. Dann läßt sich das (symmetrische) TSP formal wie folgt definieren:

(1) $\text{Minimiere} \sum_{i\in V}\sum_{j>i} c_{ij}x_{ij}$

(2) $\sum_{j<i} x_{ji} + \sum_{j>i} x_{ij} = 2, \qquad i,j\in V$

(3) $\sum_{i\in S}\sum_{j\in S, j>i} x_{ij} \le |S|-1, \qquad \forall S\subset V, S\neq\varnothing$

(4) $x_{ij}\in\{0,1\}, \qquad i,j\in V, j>i$

Die Entscheidungsvariablen x_{ij} nehmen den Wert 1 an wenn die Kante (ij) in der optimalen Rundreise liegt und bleiben andernfalls 0. Gleichungen (2) sorgen dafür, daß jeder Knoten den Grad 2 hat. Ungleichungen (3) halten die Menge der in der optimalen Lösung enthaltenen Kanten minimal („subtour elimination inequalities“).

Optimale Lösungen des TSP werden im Programm PSA mithilfe eines Branch & Bound Algorithmus berechnet. Dieses Lösungsverfahren baut auf der von Held und Karp (vgl. Held, M., Karp, R. M. (1970, 1971)) entwickelten 1-Tree Relaxation auf. Einige Ideen von Helbig Hansen und Krarup (vgl. Helbig Hansen, K., Krarup, J. (1974)), Smith und Thompson (vgl. Smith, T. H. C., Thompson, G. L. (1977)) sowie Volgenant und Jonker (vgl. Volgenant, T., Jonker, R. (1982)) wurden übernommen. Im folgenden Abschnitt werden die Grundlagen für den TSP Lösungsalgorithmus umrissen.

Unter einem **Baum** („tree“) versteht man in der Graphentheorie eine kreisfreie, zusammenhängende Teilmenge von Kanten eines ungerichteten Graphen. Ein **spannender Baum** („spanning tree“) verbindet alle Knoten des Graphen. Ein **1-Tree** ist ein spannender Baum auf der Knotenmenge $\{2,\ldots,n\}$ vereinigt mit 2 verschiedenen Kanten, die mit Knoten 1 verbunden sind. Jeder 1-Tree enthält genau einen Zyklus. Knoten 1 liegt immer auf diesem Zyklus. Gibt es keinen 1-Tree mit geringerem Gewicht, so heißt der 1-Tree **minimal**.

Held und Karp haben einige Zusammenhänge zwischen 1-Trees und Touren entdeckt und deren Bedeutung erkannt[2]:

1. Eine Tour ist ein 1-Tree in dem jeder Knoten den Grad 2 hat.
2. Falls ein 1-Tree minimal ist und eine Tour ist, so ist er eine minimale Tour.
3. Jede lineare Transformation der Kantengewichte $c'_{ij} = c_{ij} + \pi_i + \pi_j$ mit reellen Summanden π_k erhält zwar die relative Ordnung der TSP Lösungen, führt jedoch i.d.R. zu unterschiedlichen minimalen 1-Trees.
4. Minimale 1-Trees können mit vergleichsweise geringem Aufwand berechnet werden.

---

[2] Etwa zur selben Zeit hat Christofides unabhängig von Held und Karp ein sehr ähnliches Lösungsverfahren vorgeschlagen (vgl. Christofides, N. (1970)).

Sei C^* das Gewicht einer optimalen Lösung des TSP auf dem zuvor beschriebenen Graphen G mit der Kostenmatrix C. Mit einer gemäß 3. transformierten Kostenmatrix C′ verändert sich der Wert der optimalen Lösung auf $C^* + 2 \cdot \sum_{i=1}^{n} \pi_i$. Bezeichnet man mit c_k das Gewicht des k-ten 1-Trees in bezug auf C und mit $d_{i,k}$ den Grad von Knoten i in eben diesem 1-Tree, so ergibt $\min_k \left[c_k + \sum_{i=1}^{n} \pi_i \cdot d_{i,k} \right]$ das Gewicht eines minimalen 1-Trees in bezug auf C′. Natürlich gilt $C^* + 2 \cdot \sum_{i=1}^{n} \pi_i \geq \min_k \left[c_k + \sum_{i=1}^{n} \pi_i \cdot d_{i,k} \right]$. Durch einfache Umformungen erhält man daraus $C^* \geq \min_k \left[c_k + \sum_{i=1}^{n} \pi_i \cdot \left(d_{i,k} - 2 \right) \right]$. Diese Ungleichung liefert eine unendliche Menge von unteren Schranken für optimale Lösungen des TSP: $w(\pi) = \min_k \left[c_k + \sum_{i=1}^{n} \pi_i \cdot \left(d_{i,k} - 2 \right) \right]$. Die schärfste Schranke dieser Art ist $w^*(\pi) = \max_\pi [w(\pi)]$.

Im TSP Lösungsalgorithmus wird eine Subgradientenoptimierung zum Bestimmen des optimalen Multiplikators π mit einem Branch & Bound Verfahren kombiniert. Der Zustand des Branch & Bound Verfahrens wird zu jeder Zeit durch die Liste der aktiven Subprobleme beschrieben. Ein Eintrag in diese Liste hat die Form $\{I, E, \bar{\pi}, L, \#_1, \#_2\}$. Die Menge I (inkludiert) enthält alle Kanten, die in jeder Lösungen des Subproblems enthalten sein muß; E (exkludiert) hingegen ist die Menge aller Kanten, die in keiner Lösung des Subproblems enthalten sein dürfen. Der Vektor $\bar{\pi}$ beschreibt den Zustand der Subgradientenoptimierung. L ist die untere Schranke für alle zulässigen TSP Lösungen. $\#_1$ ist die Anzahl der Subgradientenoptimierungsschritte, die insgesamt auf das Subproblem angewendet wurden, und $\#_2$ ist die Nummer jenes Iterationsschrittes, in dem die untere Schranke L gefunden wurde.

Sei $\bar{\pi}_i$ der n-dimensionale Strafkostenvektor, der im Iterationsschritt i bestimmte wurde. Die Subgradientenoptimierung startet mit dem Ausgangswert $\bar{\pi}_0 = \varnothing$. Ein allgemeiner Iterationsschritt läuft folgendermaßen ab:

1. Bilde die transformierte Kostenmatrix C′.
2. Berechne einen minimalen 1-Tree bezüglich C′ unter Berücksichtigung von I und E.

3. Berechne die untere Schranke $L = c + \sum_{i=1}^{n} \pi_i \cdot (d_i - 2)$.

4. Bestimme den verbesserten Strafkostenvektor $\pi_i \leftarrow \pi_i + 2 \cdot (d_i - 2)$.

Die Subgradientenoptimierung terminiert in jedem der folgenden Fälle: Der minimale 1-Tree ist eine Tour; der Wert der unteren Schranke ist höher als die niedrigste bisher bekannte Lösung; p aufeinanderfolgende Iterationsschritte haben keine Verbesserung der unteren Schranke ergeben; q Iterationsschritte wurden durchgeführt. In den beiden letztgenannten Fällen darf das betrachtete Subproblem natürlich nicht verworfen werden; vielmehr setzt an dieser Stelle der Verzweigungsschritt des Branch & Bound Verfahrens ein.

Aus jedem 1-Tree, der während der Subgradientenoptimierung berechnet wird, versucht ein von Volgenant und Jonker vorgeschlagenes Verfahren eine zulässige Tour zu formen. Immer wenn im Zuge der Berechnung eine günstigere als die beste bisher bekannte Tour gefunden wird, versucht ein lokales Suchverfahren (vgl. Lin, S. (1965)), diese noch weiter zu verbessern. Abbildung 3.8 beschreibt die Struktur des gesamten Algorithmus.

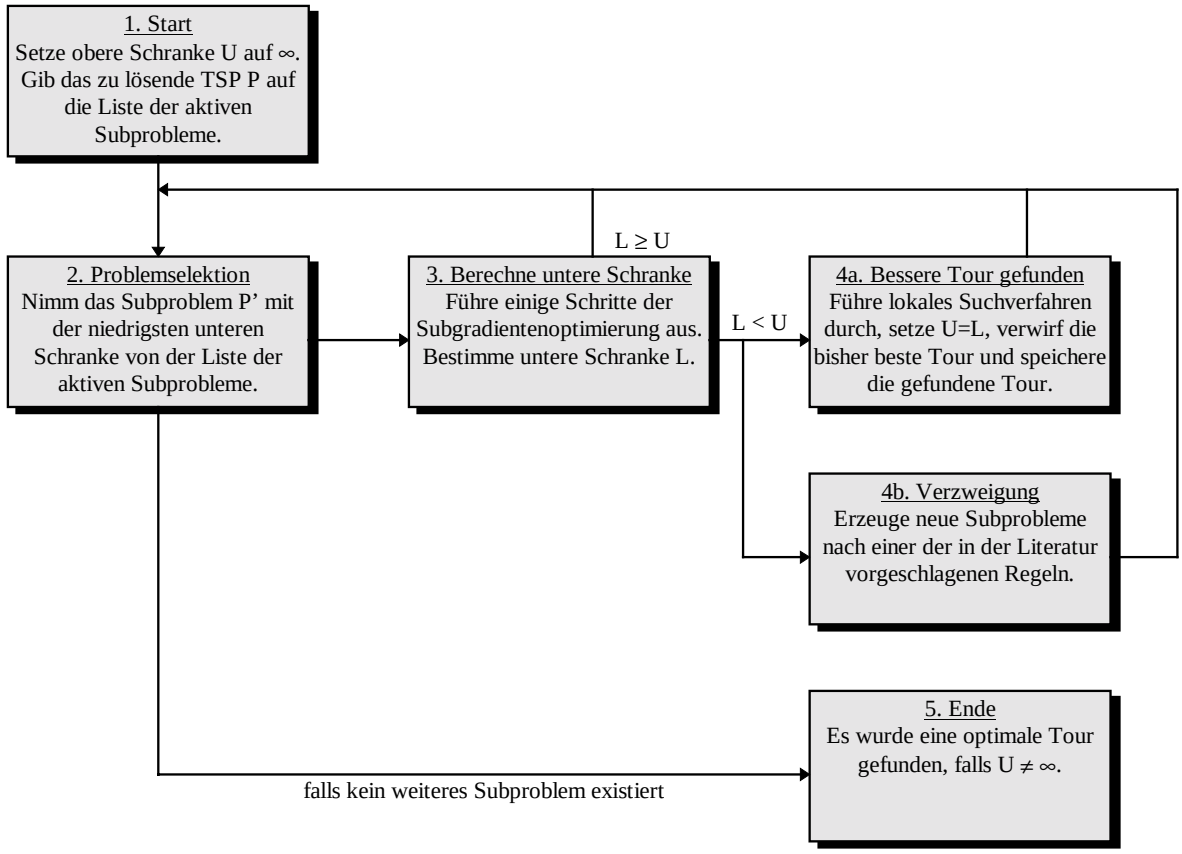


Abbildung 3.8 - Funktionsweise des TSP Lösungsalgorithmus.

3.4.3 Subproblem #2: Maximum Weight Matching Problem

Ausgangspunkt für Subproblem #2 ist die Überlegung, daß das Zusammenlegen zweier Liefertouren zu einer einzigen, größeren Liefertour eine Kostenersparnis bringen kann. Die Aufgabe des zweiten Subproblems lautet jene Paare von Fahrtrouten zu finden, durch deren Zusammenlegen die maximale Kostenersparnis erzielt werden kann. In der Literatur wird diese Problemstellung als Maximum Weight Matching Problem (MWMP) diskutiert.

Das MWMP ist nicht NP-vollständig. Edmonds erarbeitete die mathematischen Grundlagen und stellte Mitte der 1960er Jahre den ersten polynomialen Algorithmus zum Finden optimaler MWMP Lösungen vor (vgl. Edmonds, J. (1965a, 1965b)). Gondran und Minoux[3] (Gondran, M., Minoux, M. (1984)) geben in ihrer Arbeit einen Überblick über verschiedene Matchingprobleme und gehen ausführlich auf einen Lösungsalgorithmus ein. Lovász und

[3] An dieser Stelle ist eine Warnung angebracht: Ausgerechnet jenes Kapitel, das den Lösungsalgorithmus für das MWMP beschreibt, ist an mehreren Stellen fehlerhaft und unvollständig.

Plummer (Lovász, L., Plummer, M. D. (1986)) analysieren Matchingprobleme aus mathematisch-theoretischer Sicht. Derigs (Derigs, U. (1988)) analysiert Zusammenhänge zwischen Flußproblemen und Matchingproblemen und beschreibt eine Vielzahl von Lösungsverfahren.

Gegeben sei ein gewichteter ungerichteter Graph $G = \langle V, E, C \rangle$. V steht für die Menge der Knoten, E für die Menge der Kanten und C für die Matrix der Kantengewichte. Nun läßt sich das MWMP folgendermaßen formal beschreiben (Für eine allgemeinere Modellformulierung vgl. Edmonds, J., Johnson, E. L. (1969).):

(1) Maximiere $z = \sum_{(i,j)\in E} c_{ij} \cdot x_{ij}$

(2) $0 \le \sum_{i\in V} x_{ij} \le 1 \qquad j \in V$

(3) $x_{ij} \in \{0,1\} \qquad i, j \in V$

(4) $x_{ii} = 0 \qquad i \in V$

(5) $x_{ij} = x_{ji} \qquad i, j \in V$

Ein **Matching** M ist eine Teilmenge der Kanten eines ungerichteten Graphen $G = \langle V, E \rangle$, sodaß die Kanten in M keinen einzigen gemeinsamen Endpunkt haben. Ein Knoten $i \in V$ heißt **zugeordnet** in M („saturated“), falls i Endpunkt einer Kante in M ist. Andernfalls nennt man den Knoten i **isoliert** („non-saturated“, „isolated“). Das **Gewicht eines Matchings** M ist die Summe der Gewichte jener Kanten, die in M enthalten sind: $G(M) = \sum_{e\in M} c_e$. Als **alternierenden Weg** in bezug auf M („alternating chain“) bezeichnet man einen Weg in G, dessen Kanten abwechselnd in M und $\overline{M} = E - M$ liegen. Ein alternierender Weg dessen Anfangs- und der Endknoten ident sind, heißt **alternierender Kreis** („alternating cycle“). Als **Blüte** („blossom“) wird ein alternierender Kreis mit einer ungeraden Anzahl von Knoten bezeichnet. Falls ein alternierender Weg zwei isolierte Knoten verbindet handelt es sich um einen **augmentierenden alternierenden Weg** („augmenting alternating chain“). Durch einen **Transfer** entlang eines alternierenden Weges W bildet man aus einem Matching M ein Matching M′: Man entfernt alle Kanten $K_- = W \cap M$ aus M und fügt alle Kanten $K_+ = W - (W \cap M)$ zu M hinzu. Nach einem Transfer entlang eines augmentierenden alternierenden Weges enthält M′ um eine Kante mehr als M.

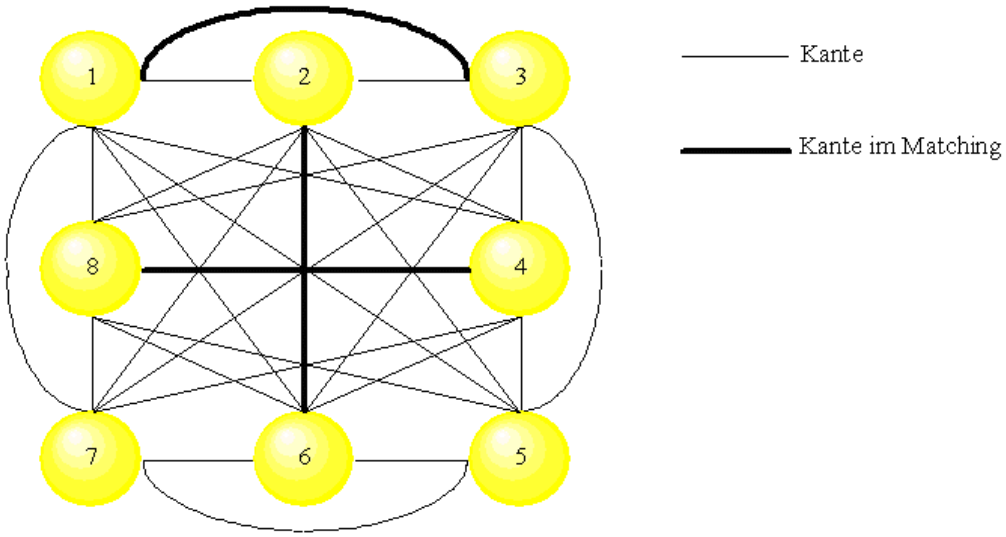


Abbildung 3.9 - Beispiel für ein Matching.

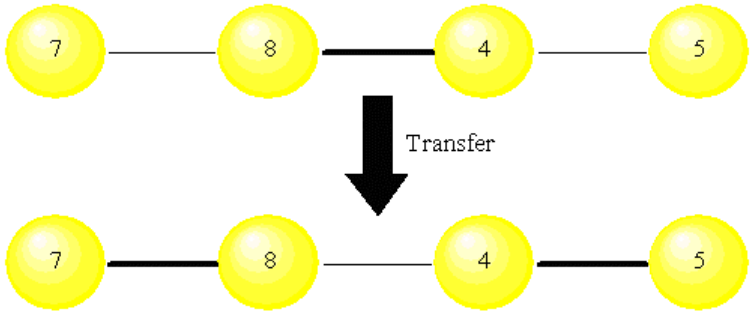


Abbildung 3.10 - Transfer entlang eines augmentierenden alternierenden Weges.

Unter einem **alternierenden Baum** („alternating tree“) in bezug auf ein Matching M versteht man einen zusammenhängenden, kreisfreien Subgraphen von G mit folgenden Eigenschaften:

1. Die Wurzel ist der einzige isolierte Knoten im Baum.
2. Der Weg zwischen der Wurzel und jedem anderen Knoten des Baums ist ein alternierender Weg in bezug auf M.
3. Der Weg zwischen der Wurzel und jedem Blatt des Baumes beinhaltet eine gerade Anzahl von Kanten.

Jene Knoten in einem alternierenden Baum, die von der Wurzel über einen Weg mit gerader (ungerader) Anzahl von Kanten erreichbar sind, heißen **gerade (ungerade) Knoten**. Die Wurzel eines alterniernden Baumes ist immer gerade.

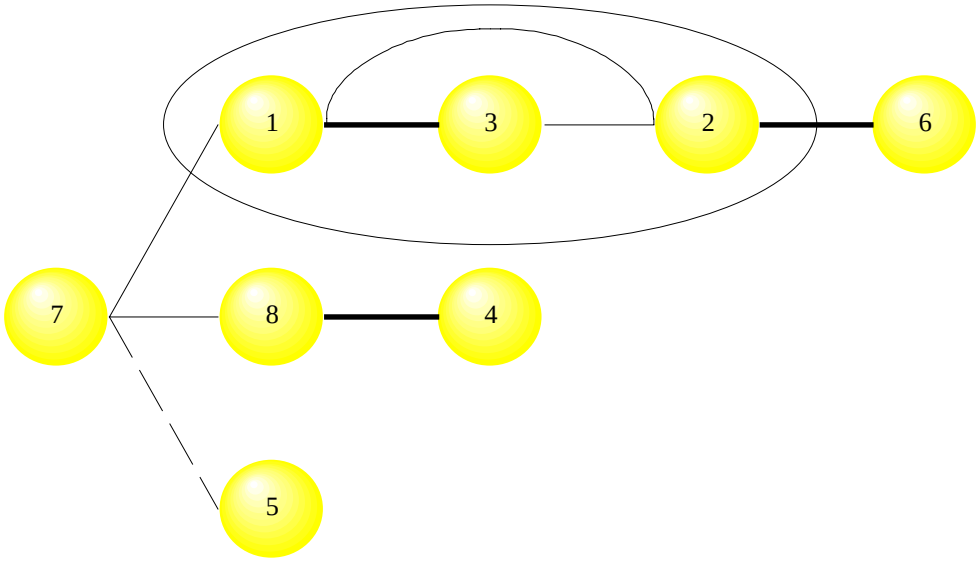


Abbildung 3.11 - Alternierender Baum mit Blüte.

Einen **geschrumpften Graph** $G' = \langle V', E' \rangle$ bildet man aus $G = \langle V, E \rangle$ indem man ein Teilmenge ΔV der Knoten aus V durch einen neuen Knoten x ersetzt, alle Kanten mit beiden Endpunkten in ΔV aus E entfernt und alle Kanten der Form $(i \in \Delta V, j \notin \Delta V)$ durch (i, x) ersetzt. Die inverse Operation zum Schrumpfen eines Graphen nennt man **expandieren**.

Mit dem Begriff **Grenzerfolg eines alternierenden Weges** W bezeichnet man jenen Betrag um den sich das Gewicht von M bei einem Transfer entlang W ändert: $\delta(W) = G(W \cap M) - G\left(W \cap \overline{M}\right) = G(M') - G(M)$. Der Grenzerfolg eines alternierenden Kreises ist analog dazu definiert. M ist genau dann ein Matching mit maximalem Gewicht, wenn es in bezug auf M keinen alternierenden Weg und keinen alternierenden Kreis mit streng positivem Grenzerfolg gibt.

Für die vorliegende Arbeit wurde der von Gondran und Minoux (vgl. Gondran, M., Minoux, M. (1984), S. 279-336) beschriebene Algorithmus zum Lösen des MWMP implementiert. Es handelt sich dabei um ein primal-dual Verfahren, das mit Aufwand $O\left(|V|^2 \cdot |E|\right)$ optimale Lösungen des MWMP auf dem Graph $G = \langle V, E, C \rangle$ bestimmt. Abbildung 3.12 beschreibt die Ablaufstruktur.

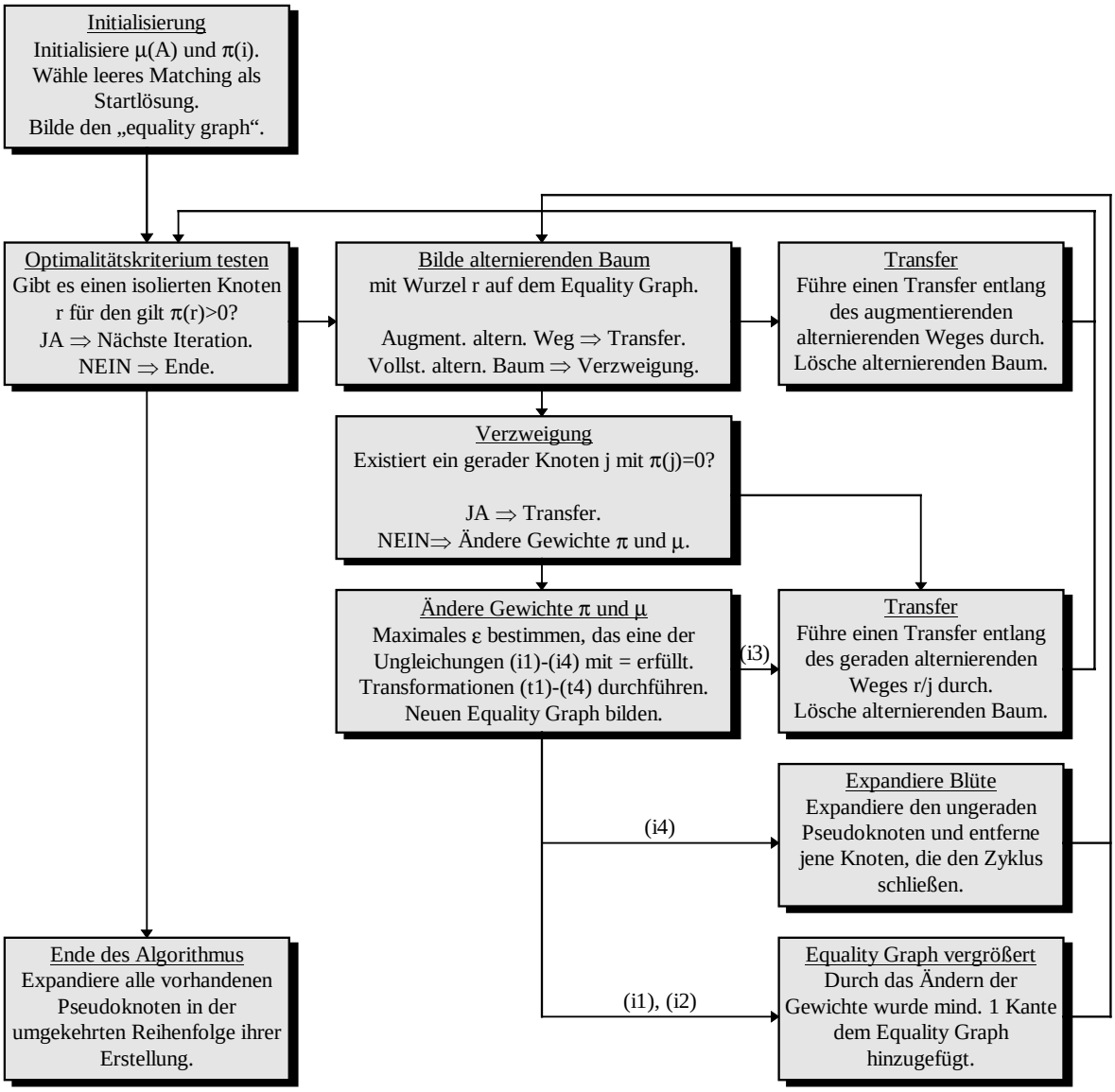


Abbildung 3.12 - Funktionsweise des MWMP Lösungsalgorithmus.

Jedem Knoten $v \in V$ wird ein Gewicht $\pi(v)$, jeder Blüte w, die während der Berechnung gebildet wird, ein Gewicht $\mu(w)$ zugeordnet. Der **Grenzerfolg einer Kante** $e = (i, j)$ ist definiert als $\bar{c}_e = c_e - \pi(i) - \pi(j) - \sum_{A/e \in A} \mu(A)$. Die Menge A besteht dabei aus allen Blüten, die im aktuellen Zustand des Verfahrens existieren. Man wählt als zulässige Startlösung ein leeres Matching M, initialisiert $\pi(v)$ mit dem maximalen Kantengewicht aus G und $\mu(w)$ mit 0. Der sog. **Equality Graph** $\bar{G} = \langle V, \bar{E} \rangle$ ist jener Subgraph von G für den gilt: $\bar{E} = \{e \in E | \bar{c}(e) = 0\}$. Solange es bezüglich M einen isolierten Knoten r mit $\pi(r) > 0$ gibt, konstruiert man auf dem Equality Graph $\bar{G}$ einen alternierenden Baum mit r als Wurzel.

Während man den alternierenden Baum aufbaut, kann es vorkommen, daß man eine Kante, die zwei gerade Knoten verbindet, in den Baum aufnehmen müßte. Dadurch würde jedoch eine Blüte entstehen; ein Kreis also, mit einer ungeraden Anzahl von Knoten. Da ein Baum kreisfrei sein muß, schrumpft man in diesem Fall den ursprünglichen Graphen: Man entfernt alle Knoten die auf dem entsprechenden Kreis liegen und ersetzt sie durch einen sog. **Pseudoknoten** („pseudo-vertex“). In bestimmten Situationen (spätestens nach dem Zutreffen des Optimalitätskriteriums) werden die Pseudoknoten wieder expandiert. Dieses Schrumpfen von Blüten zu Pseudoknoten und das spätere Expandieren der Pseudoknoten ist der wesentliche Punkt in dem beschriebenen Verfahren. Die Idee geht auf die frühe Arbeit von Edmonds zurück und wird in verschiedenen Variationen in sämtlichen Matching Algorithmen verwendet.

Beim Konstruieren des alternierenden Baumes findet man entweder einen augmentierenden alternierenden Weg zwischen r und einem anderen isolierten Knoten oder man bildet einen vollständigen alternierenden Baum. Im ersten Fall kann man die Anzahl der Kanten in M durch einen augmentierenden Transfer erhöhen, im zweiten Fall muß man die Gewichte π und μ ändern. Das Verfahren terminiert nachdem maximal $|V|$ alternierende Bäume aufgebaut wurden und liefert als Ergebnis ein Matching mit maximalem Gewicht.

Eine vollständige Beschreibung des Algorithmus würde den Rahmen dieser Arbeit sprengen. Für das Konstruieren alternierender Bäume, die Beschreibung geeigneter Datenstrukturen, das Ungleichungssystem (i1)-(i4) und die Transformationen (t1)-(t4) sei daher auf das oben genannte Buch von Gondran und Minoux sowie den entsprechenden Programmcode auf der beiliegenden Diskette verwiesen.

4 Implementierung

Vollkommenheit entsteht offensichtlich nicht dann,
wenn man nichts mehr hinzuzufügen hat,
sondern wenn man nichts mehr wegnehmen kann.
(Antoine de Saint-Exupéry)

Kapitel 4 soll das informationstechnische Umfeld, die grundlegenden Ideen bei der Entwicklung sowie die Funktionsweise des Softwaresystems dokumentieren, das den zuvor beschriebenen PSA realisiert.

4.1 Schnittstellen

4.1.1 Position im Schichtmodell

Die grundlegende Idee beim Erstellen des Programms PSA ist es, ein abgegrenztes „Service“ zu bieten, das von möglichst vielen „Kunden“ in Anspruch genommen werden kann. Das Service, das PSA anbietet, ist das schnelle Berechnen von guten Näherungslösungen für Vehicle Routing Probleme. Potentielle Kunden sind alle Anwendungsprogramme, die in irgendeinem Zusammenhang eine gute Lösung für ein VRP benötigen.

Um einerseits dieser funktionellen Anforderung gerecht zu werden und darüber hinaus auch möglichst unabhängig von einer konkreten Computersystemarchitektur zu sein, ist PSA als eigenständig ausführbares Programm konzipiert. Ein beliebiges Anwendungsprogramm übernimmt den Dialog mit einem Anwender und ruft wenn nötig PSA auf. Die eindeutige Beschreibung des zu lösenden Problems („input“) sowie die berechnete Lösung („output“) werden nach festen Codierungsschemata über Dateien ausgetauscht. Das Betriebssystem bietet sowohl dem Anwendungsprogramm als auch PSA seine Serviceleistungen an und übernimmt die Kommunikation mit der Hardware. Dieser Sachverhalt wird in Abbildung 4.1 verdeutlicht.

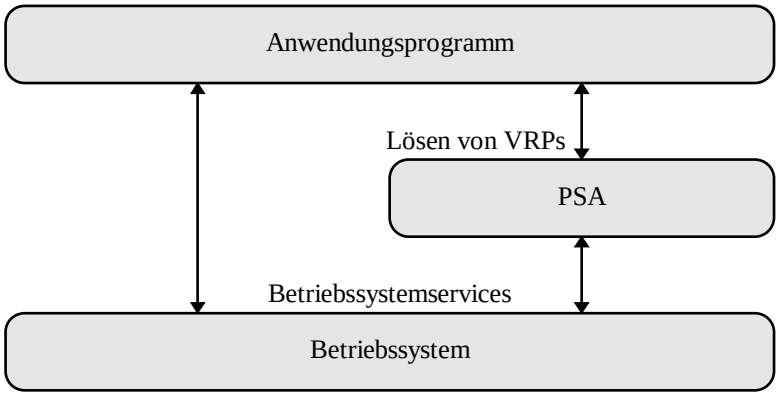


Abbildung 4.1 - Position des Programms PSA im Schichtmodell.

Aus der obigen Aufgabenstellung lassen sich die Funktionen ableiten die PSA beherrschen muß, um sein Service anbieten zu können. Es sind dies:

- Einlesen der zu lösenden Probleminstanz aus einer Datei.
- Interpretieren von Aufrufparametern (Einlesen von Steuerungsinformationen).
- Berechnen einer Näherungslösung nach dem unter 3.4 beschriebenen Verfahren.
- Ausgeben der gefundenen Lösung in eine Datei.
- Ausgeben von Fehlermeldungen falls Fehler auftreten.
- Setzen eines Rückgabewertes, der über Erfolg oder Mißerfolg der Berechnung informiert.

4.1.2 Datenfluß

Abbildung 4.2 veranschaulicht die von PSA benötigten Eingabedaten sowie die erzeugten Ausgabedaten. Die beiden Dateien „Kostenmatrix“ und „Nachfragevektor“ beschreiben das zu lösende VRP. Die symmetrische Kostenmatrix enthält die Transportkosten für alle Kanten des zugrundeliegenden Netzwerkes. Die in den einzelnen Knoten nachgefragten Mengen werden im Nachfragevektor übergeben. Der Knoten mit der Nummer 1 stellt immer das Depot dar. In der „Lösungsdatei“ wird die beste Lösung, die PSA ermitteln konnte, an das aufrufende Programm zurückgeliefert. Alle Fehlermeldungen werden auf den Standard Fehlerausgabekanal geschrieben. Im Erfolgsfall liefert das Programm den Wert 0 zurück; ein Rückgabewert, der größer als 0 ist, zeigt aufgetretene Fehler an. Als Steuerinformation werden die Dateinamen der Kostenmatrix, des Nachfragevektors sowie der zu erstellenden Lösungsdatei übergeben (vgl. dazu Abschnitt 4.1.4).

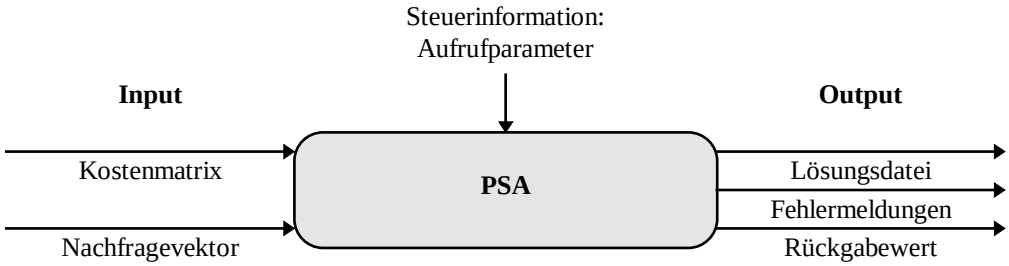


Abbildung 4.2 - Datenfluß bei der Verwendung von PSA.

Alle drei Dateien enthalten reinen ASCII Text. Diese Darstellungsform benötigt zwar relativ viel Speicherplatz, kann aber mit jedem gängigen Texteditor und jeder gängigen Tabellenkalkulation erzeugt werden. Auch das Portieren zwischen unterschiedlichen Betriebssystemen ist unproblematisch.

4.1.3 Dateiformate

Das Dateiformat für Kostenmatrizen geht aus der unten stehenden Definition hervor. Während die Kennung die Konsistenzprüfung erleichtert, unterstützt die (redundante) Angabe der Dimension das Einlesen der Matrix in ein dynamisch erzeugtes Objekt. Da die Matrix symmetrisch ist werden nur die Einträge von der Hauptdiagonalen abwärts gespeichert. Für die Anwendung PSA sind die Elemente der Hauptdiagonalen nicht relevant. Sie werden ignoriert und können deshalb beliebige Werte aus dem zulässigen Wertebereich annehmen. Der Knoten mit der Nummer 1 wird immer als das Depot interpretiert.

```
Matrix ← Matrix_Kopf Matrix_Daten

Matrix_Kopf ← Matrix_Kennung Matrix_Dimension

Matrix_Daten ←
      Matrix_Zeile(1)
      Matrix_Zeile(2)
      ...
      Matrix_Zeile(n)

Matrix_Kennung ←
      [Kommentar]
      „Symmetrische Matrix“ EOL
      [Kommentar]

Matrix_Dimension ←
      [Kommentar]
      „Dimension“ TAB Ganze_Zahl EOL
      [Kommentar]
```

```
Matrix_Zeile(n) ←
      [Kommentar]
      Matrix_Datenzeile(n)
      [Kommentar]

Matrix_Datenzeile(1) ← Reelle_Zahl EOL
Matrix_Datenzeile(n) ← Reelle_Zahl TAB Matrix_Datenzeile(n-1)

Kommentar ← Kommentarzeile

Kommentarzeile ←
      Kommentarzeile
      Kommentarzeile
Kommentarzeile ← EOL
Kommentarzeile ← „;“ Zeichenkette EOL

Ganze_Zahl ← Eine ganze Zahl aus [-32767/+32767].
Reelle_Zahl ← Eine reelle Zahl aus [-3.4e38/+3.4e38].
Zeichenkette ← Eine beliebige Folge von druckbaren Zeichen.
TAB ← Das Zeichen „Tabulator“.
EOL ← Das Zeichen „End of Line“.
```

Das Dateiformat für den Nachfragevektor, der an PSA übergeben wird, hat einen ähnlichen Aufbau wie jenes der Kostenmatrix. Der Dateiinhalt kann als n-dimensionaler Zeilenvektor interpretiert werden. Die Nachfrage des Knoten 1 (des Depots) ist immer 0. Alle anderen Knoten müssen über eine streng positive Nachfrage verfügen. Die Dimension des Nachfragevektors muß mit der Dimension des Kostenmatrix übereinstimmen.

```
Vektor ← Vektor_Kopf Vektor_Daten

Vektor_Kopf ← Vektor_Kennung Vektor_Dimension

Vektor_Daten ←
      [Kommentar]
      Vektor_Datenzeile(n)
      [Kommentar]

Vektor_Kennung ←
      [Kommentar]
      „Vektor“ EOL
      [Kommentar]

Vektor_Dimension ←
      [Kommentar]
      „Dimension“ TAB Ganze_Zahl EOL
      [Kommentar]

Vektor_Datenzeile(1) ← Reelle_Zahl EOL
Vektor_Datenzeile(n) ← Reelle_Zahl TAB Vektor_Datenzeile(n-1)
```

Falls PSA eine Lösung für das VRP berechnen kann, wird diese im unten angeführten Format in einer Datei gespeichert. Die Kopfzeile hilft dem (menschlichen) Anwender beim Interpretieren des Dateiinhalts. Jede Datenzeile beschreibt die Route, die ein bestimmtes Fahrzeug zurückzulegen hat und enthält die auszuliefernde Menge des betrachteten Produkts sowie die durch diese Route entstehenden Kosten. Als letzte Zeile der Lösungsdatei werden Summen über die ausgelieferte Menge und die entstehenden Kosten gebildet.

```
Lösung ← Lösung_Kopf Lösung_Daten Lösung_Gesamt

Lösung_Kopf ← „Vehikel“ TAB „Menge“ TAB „Kosten“ TAB „Route“ EOL

Lösung_Daten ← Ganze_Zahl TAB Reelle_Zahl TAB Reelle_Zahl TAB Route(n) EOL

Lösung_Gesamt ← „Total“ TAB Relle_Zahl TAB Reelle_Zahl EOL

Route(1) ← Ganze_Zahl EOL
Route(n) ← Ganze_Zahl TAB Route(n-1)
```

4.1.4 Aufrufparameter

Die Kommandozeilenparameter, mit denen PSA aufgerufen werden muß, sind in Tabelle 4.1 enthalten. Bei fehlerhaftem Programmaufruf wird eine Fehlermeldung und ein Hilfetext ausgegeben. Die Eingabedateien müssen den Dateiformaten aus Abschnitt 4.1.3 entsprechen.

Aufrufparameter	**Optional**	**Beschreibung**
-c <cost_matrix>	Nein	<cost_matrix> ist der Dateiname der zu verwendenden symmetrischen Kostenmatrix. Die Kostenmatrix darf keine negativen Elemente enthalten.
-d <demand_vector>	Nein	<demand_vector> ist der Dateiname des zu verwendenden Nachfragevektors. Die Nachfrage des Depots muß 0 sein. Alle anderen Nachfragemengen müssen größer als 0 sein.
-q <vehicle_quantity>	Nein	<vehicle_quantity> gibt die maximale Transportkapazität der Fahrzeuge an.
-s <solution_file>	Nein	<solution_file> ist der Dateiname unter dem die berechnete Lösung gespeichert wird.
-m <mergers>	Ja	<mergers> gibt die Anzahl der Cluster an, die in jedem Iterationsschritt zusammengefaßt werden. Erlaubter Wertebereich ist [1...20]. Dieser Parameter entspricht dem Steuerparameter T.
-o	Ja	Falls diese Option angegeben wird überschreibt PSA die Datei <solution_file>, falls diese bereits existiert.

Tabelle 4.1 - Aufrufparameter für das Programm PSA.

4.2 Systemplattform

4.2.1 Hardware

Das Programm PSA wurde auf einem handelsüblichen Personal Computer entwickelt. Die genauen Konfigurationsdaten sind Tabelle 4.2 zu entnehmen. Alle Testfälle wurden auf diesem Computer berechnet (vgl. dazu Kapitel 5), und auch die entsprechenden Laufzeitmessungen wurden auf diesem System vorgenommen.

Komponente	Bezeichnung
Prozessor	INTEL 80486dx-2/66
Interner Speicher	16KB interner Cache, 256KB externer Cache, 8MB Hauptspeicher
Externe Speichermedien	1GB Magnetplatte, 3.5“ 1.44MB Diskettenlaufwerk, 5.25“ 1.2MB Diskettenlaufwerk
Ein-/Ausgabegeräte	VGA Grafikkarte, Monitor, Tastatur, Maus, Tintenstrahldrucker

Tabelle 4.2 - Hardwarekonfiguration des Entwicklungsrechners.

4.2.2 Software

Tabelle 4.3 enthält die Softwarekonfiguration des Entwicklungsrechners. Das erstellte Softwaresystem wurde vollständig in C++ geschrieben. Betriebssystemspezifische Funktionen wurden nicht verwendet. Da es für fast jedes Betriebssystem einen C++ Compiler gibt, kann der Programmcode sehr einfach auf andere (leistungsfähigere) Maschinen portiert werden. Lediglich einige hardwarespezifische Makrodeklarationen müssen angepaßt werden.

Komponente	Bezeichnung	Bemerkung
Betriebssystem	MS-DOS 6.2	-
Compiler	Borland C++ 3.1	Die verwendeten Compilereinstellungen sind auf der beiliegenden Diskette in den beiden Projektdateien (*.PRJ) im Verzeichnis \SOURCE gespeichert.
Textverarbeitung	MS Word 6.0	Die vollständige Diplomarbeit befindet sich auf der beiliegenden Diskette im Verzeichnis \DOC unter dem Namen DIPLOM.DOC.
Tabellenkalkulation	MS Excel 5.0	Wurde zum Erstellen der Tabellen und Diagramme in dieser Arbeit sowie der Vehicle Routing Eingabedaten für die Testläufe verwendet.

Tabelle 4.3 - Softwarekonfiguration des Entwicklungsrechners.

4.3 Organisation des Programmcodes

PSA wurde objektorientiert entworfen und implementiert. Jede Klasse im Programmsystem verfügt über eine einfache Schnittstelle nach außen und übernimmt das Lösen einer genau

definierten Aufgabe. Der Algorithmus, mit dem die entsprechenden Aufgaben gelöst werden, und die dafür benötigten Datenstrukturen sind i.d.R. nur innerhalb der jeweiligen Klasse sichtbar; andere Klassen und Programmteile werden von diesen „Klassenspezifika“ abgekapselt. Tabelle 4.4 gibt einen Überblick über die implementierten Klassen, deren Aufgaben und die Organisation des Sourcecodes. Die gleiche Information enthält Tabelle 4.5 für mehrere Klassen, die an vielen Stellen im Programm benötigt werden.

Sourcecode	**Klasse**	**Beschreibung**
PSA_MAIN.CPP	-	Enthält das Hauptprogramm und den new-Handler. Besorgt das Interpretieren der Kommandozeile, das Einlesen der Eingabedaten und das Speichern der Lösung.
VRP_PSA .CPP	VRP_PSA	Implementiert den Algorithmus PSA.
CLSTR.CPP	VRP_Cluster	Klasse zum Speichern und Manipulieren eines Clusters.
CLSTR_CA.CPP	VRP_Cluster_Cache	Verwaltet die aktiven Cluster, speichert alle bereits berechneten Cluster, verhindert Mehrfachberechnungen.
CLSTR_HT.CPP	VRP_Cluster_Container	Hash Tabelle für schnellen Zugriff auf die bereits berechneten, gespeicherten Cluster.
CLSTR_ID.CPP	VRP_Cluster_ID	Bildet eindeutige Schlüssel für Cluster. Ein eindeutiger Schlüssel ist Voraussetzung dafür, daß Cluster in einer Hash Tabelle gespeichert werden können.
MWMP.CPP	MWMP	Löst das Maximum Weight Matching Problem nach dem Algorithmus von Gondran und Minoux..
CALC.CPP	BranchAndBound	Abstrakte Basisklasse. Implementiert ein Branch & Bound Verfahren zum Lösen des Traveling Salesman Problem (1-Tree Relaxation). Enthält jedoch keine Verzweigungsregel.
	Gavish_Srikanth	Löst das TSP mit der Verzeigungsregel von Gavish und Srikanth.
	Volgenant_Jonker	Löst das TSP mit der Verzeigungsregel von Volgenant und Jonker.
	Smith_Thompson	Löst das TSP mit der Verzeigungsregel von Smith und Thompson.
PROBLEMS.CPP	Subproblem	Speichert ein aktives Subproblem des TSP Lösungsverfahrens..
	SubproblemStack	Implementiert die „Liste der aktiven Subprobleme“ für das TSP Lösungsverfahren.
MOT.CPP	MinimumOneTree	Abstrakte Basisklasse zum Berechnen minimaler 1-Trees.
	PRIM_MinimumOneTree	Berechnet minimale 1-Trees nach dem Algorithmus von Prim.
	KRUSKAL_MinimumOneTree	Berechnet minimale 1-Trees nach dem Algorithmus von Kruskal.

Sourcecode	Klasse	Beschreibung
MOT_HEUR.CPP	AdjacentNode	Speichert einen Eintrag in eine Adjazenzliste.
	AdjacentList	Implementiert eine Adjazenzliste.
	ChangeMOTtoTour	Versucht nach einer von Volgenant und Jonker vorgeschlagenen Heuristik aus minimalen 1-Trees Touren zu erzeugen.
2OPT.CPP	TwoOptimality	Lokales Suchverfahren. Versucht während des TSP Lösungsverfahrens gefundene Touren weiter zu verbessern.

Tabelle 4.4 - Organisation des Sourcecodes (PSA).

Sourcecode	Klasse	Beschreibung
GRAPH.CPP	Matrix	Abstrakte Basisklasse für Matrizen.
	FullMatrix	Reelle Matrix.
	HalfMatrix	Reelle symmetrische Matrix.
	Vector	Reeller Vektor.
	Edge	Kante eines ungerichteten, gewichteten Graphen.
	EdgeBit	Symmetrische Matrix für binäre Werte.
BIT.CPP	BitMatrix	Matrix für binäre Werte.
	BitVector	Vektor für binäre Werte.
MACHINE.H	-	Maschinenabhängige Makrodefinitionen.
MIN.H	-	Template Funktionen: Minimum und Maximum.
RCA.H	RCA<class T>	Template Klasse: Array mit Bereichsprüfung.
STACK.H	STACK<class T>	Template Klasse: Stack.

Tabelle 4.5 - Organisation des Sourcecodes (Hilfsklassen).

Abbildung 4.3 illustriert den Zusammenhang der oben beschriebenen Klassen. Vom Hauptprogramm aus wird, nachdem alle Eingabedaten korrekt eingelesen wurden, die Methode VRP_PSA::ComputeVRPSolution aufgerufen, um eine Lösung des VRP zu berechnen. Die Klasse VRP_PSA nimmt ihrerseits die Dienste der Klassen VRP_Cluster_Cache (zum Verwalten der Cluster), MWMP (zum Lösen des MWMP) und Gavish_Srikanth (zum Lösen des TSP) in Anspruch, etc. Für Details der Implementierung sei auf den eigentlichen Programmcode verwiesen. Alle Funktionen und Methoden sind mit ausführlichen Kommentaren versehen. Die Header Dateien (*.h) sind nicht dokumentiert, da für die Implementierung die Dokumentation des eigentlichen Quellcodes vorzuziehen ist. Abschließend sei noch die Bemerkung angebracht, daß das gesamte Programm auf

bestmögliche Verständlichkeit und Stabilität, nicht jedoch auf bestmögliche Geschwindigkeit ausgelegt ist.

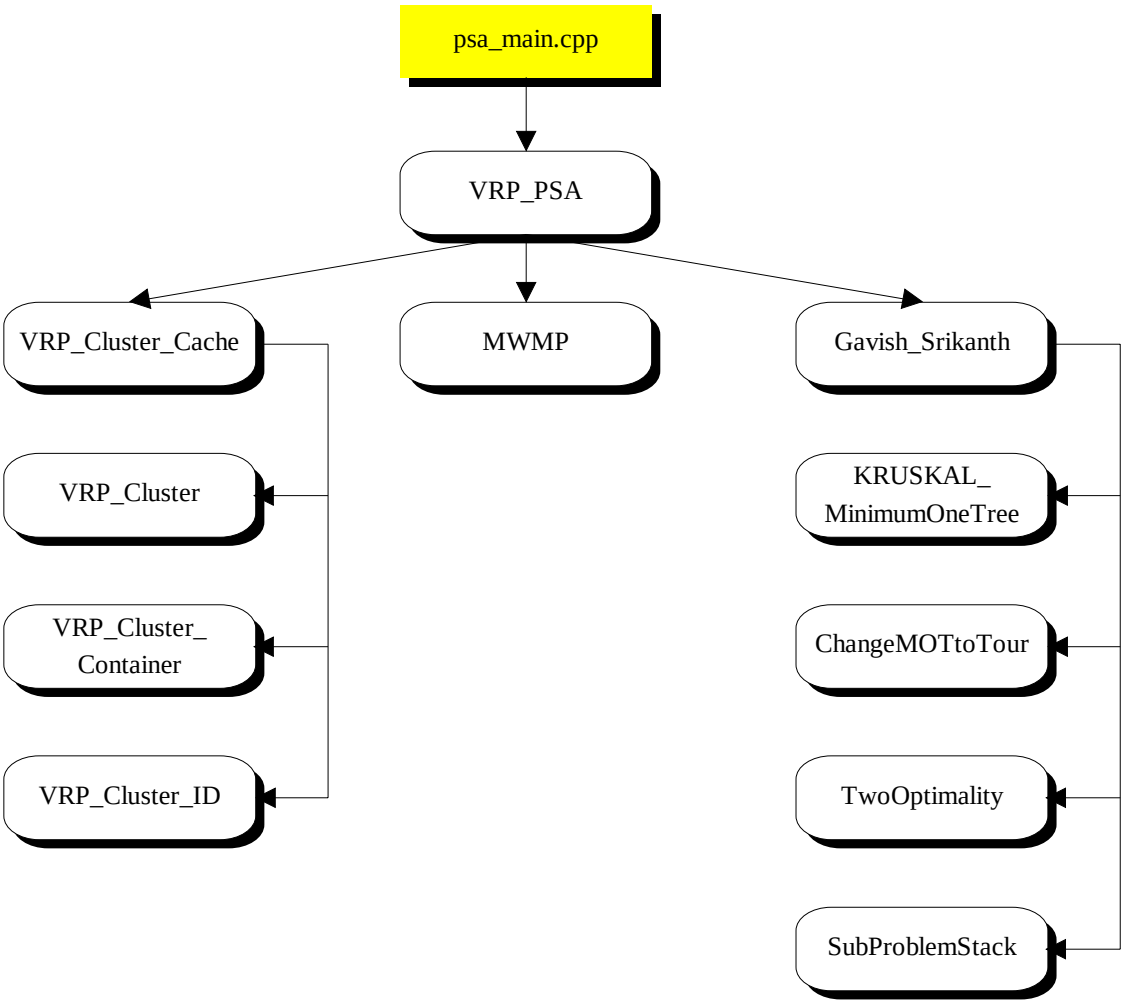


Abbildung 4.3 - Zusammenhang der wichtigsten Klassen.

5 Ergebnisse

Kapitel 5 enthält eine quantitative Analyse des implementierten Vehicle Routing Lösungsverfahrens. Abschnitt 5.1 beschreibt die Testprobleme, die für die Analyse ausgewählt wurden, und enthält die berechneten Lösungen. Über die Qualität der von PSA berechneten Lösungen wird unter 5.2 eine Aussage getroffen. Im abschließenden Abschnitt 5.3 wird die benötigte Rechenzeit betrachtet.

5.1 Testdaten

Als Testdaten für die Analyse von PSA wurden 7 Probleme aus bekannten Arbeiten ausgewählt. Die genauen Daten sind in Tabelle 5.1 zusammengefaßt. Alle Gewichtsangaben wurden in kg umgewandelt, Kostenrestriktionen wurden nicht berücksichtigt. Vorsicht ist beim Interpretieren der Ergebnisse von Problem 5 angebracht: Clarke und Wright betrachteten in ihrer Arbeit einen heterogenen Fuhrpark während für PSA alle Fahrzeuge die selbe Kapazität haben müssen. Gaskell hat bei seinen Berechnungen für jeden Kunden 10 EH zur Zielfunktion addiert („allowance“). Dieser lineare Faktor wurde aus seinen Ergebnissen herausgerechnet.

Nr.	Knoten	Kapazität	Quelle	Dateien
1	13	6000	Dantzig, G. B., Ramser, J. H., (1959)	miles_13.xls vr_c13.txt vr_d13.txt
2	22	6000	Gaskell, T. J. (1967)	miles_22.xls vr_c22.txt vr_d22.txt
3	23	4500	Gaskell, T. J. (1967)	miles_23.xls vr_c23.txt vr_d23.txt
4	30	4500	Gaskell, T. J. (1967)	miles_30.xls vr_c30.txt vr_d30.txt
5	31	5000	Clarke, G., Wright, J. W. (1964)	miles_31.xls vr_c31.txt vr_d31.txt
6	33	8000	Gaskell, T. J. (1967)	miles_33.xls vr_c33.txt vr_d33.txt
7	51	8000	Christofides, N., Eilon, S. (1969)	miles_51.xls vr_c51.txt vr_d51.txt

Tabelle 5.1 - Beschreibung und Quellenverzeichnis der Testdaten.

Jedes der oben genannten Probleme wurde mit dem Programm PSA für 9 verschiedene Werte des Steuerparameters T gelöst. Die Gesamtkosten der berechneten Liefertouren gehen aus der folgenden Tabelle hervor.

Nr.	T=1	T=2	T=3	T=4	T=6	T=8	T=10	T=15	T=20	Minimum
1	290	300	292	318	302	302	302	302	302	290
2	388	414	398	403	420	400	444	444	444	388
3	569	569	569	570	570	574	570	570	570	569
4	506	506	518	548	522	540	549	551	551	506
5	1717	1711	1710	1743	1844	1711	1731	1759	1759	1710
6	505	509	506	512	499	510	513	551	505	499
7	559	561	543	582	568	570	585	578	615	543

Tabelle 5.2 - Kosten der Liefertouren in Abhängigkeit von T.

5.2 Lösungsqualität

Um die Qualität der berechneten Lösungen beurteilen zu können, werden die besten Ergebnisse von PSA den besten Ergebnissen einer anderen Arbeit gegenübergestellt. Christofides und Eilon (vgl. Christofides, N., Eilon, S. (1969)) haben die Leistungsfähigkeit mehrerer VRP Lösungsverfahren verglichen und dazu (unter anderem) die hier verwendeten Testdaten benützt.

Nr.	PSA	Literatur	Relativ	Quelle
1	290	290	0,0%	Christofides, N., Eilon, S. (1969)
2	388	375	+3,5%	Christofides, N., Eilon, S. (1969)
3	569	729	-21,9%	Christofides, N., Eilon, S. (1969)
4	506	585	-13,5%	Christofides, N., Eilon, S. (1969)
5	1710	-	-	
6	499	490	+1,8%	Christofides, N., Eilon, S. (1969)
7	543	556	-2,3%	Christofides, N., Eilon, S. (1969)

Tabelle 5.3 - Vergleich der Lösungen.

Die Daten aus Tabelle 5.3 ergeben kein klares Bild bezüglich der Lösungsqualität von PSA. Die Lösungen der Probleme 2 und 6 sind um wenige Prozentpunkte schlechter als die Vergleichsdaten. Für Problem 1 sind die Resultate ident, hier handelt es sich vielleicht um ein globales Optimum. Bei den Problemen 3, 4 und 7 konnten bessere Lösungen gefunden werden.

PSA liefert also weder konstant bessere noch konstant schlechtere Lösungen als andere Verfahren; die berechneten Lösungen bewegen sich im Bereich anderer veröffentlichter

Lösungen. Der Steuerparameter T stellt – zumindest in der momentanen Implementierung – einen Nachteil von PSA dar: Aus Tabelle 5.2 kann man erkennen, daß sich die minimalen Kosten bei unterschiedlichen Werten von T ergeben. Man muß daher mehrere T-Werte „durchprobieren“, um ein gutes Ergebnis zu erhalten. Eine mögliche Verbesserung von PSA wäre es, T während der Berechnung nicht konstant zu halten, sondern von der aktuellen Anzahl an Clustern abhängig zu machen. Am Beginn der Berechnungen könnte T einen hohen Wert haben. Damit würden in jeder Iteration viele Cluster zusammengefaßt werden. Im weiteren Verlauf müßte der Steuerparameter sinken, damit das Verfahren nicht zu rasch zu einer schlechten Lösung konvergiert.

5.3 Laufzeitverhalten

Die Laufzeiten von PSA beim Berechnen jedes einzelnen Testfalles sind in Tabelle 5.4 zusammengefaßt. Abbildung stellt die Daten dieser Tabelle graphisch dar. Vor allem aus dem Säulendiagramm kann man zwei Tatsachen erkennen:

1. Die Laufzeit steigt bei konstantem Steuerparameter T mit zunehmender Problemgröße stark an. Ursache dafür ist vor allem der MWMP Lösungsalgorithmus mit Aufwand $O\left(|V|^2 \cdot |E|\right)$.
2. Die Laufzeit eines bestimmten Problems steigt mit wachsendem Steuerparameter T zunächst an und sinkt danach tendenziell. Ursache dafür ist, daß mit wachsendem T sowohl die Anzahl der auszuführenden Iterationen sinkt, als auch die Größe der zu lösenden MWMP abnimmt.

Nr.	T=1	T=2	T=3	T=4	T=6	T=8	T=10	T=15	T=20
1	0,99	1,10	0,93	0,88	1,04	1,04	1,04	1,04	1,04
2	8,46	11,87	11,48	10,77	9,34	7,91	7,03	6,15	6,15
3	15,93	18,30	17,97	16,32	13,85	13,19	9,29	8,13	8,08
4	42,42	44,12	42,25	33,63	31,54	30,55	27,25	18,57	18,57
5	48,52	47,64	42,47	29,56	22,75	21,76	17,91	10,11	10,16
6	79,07	89,12	78,79	73,85	55,82	44,18	39,34	21,81	21,81
7	209,45	313,96	337,14	312,42	316,92	290,00	244,01	178,79	117,03

Tabelle 5.4 - Laufzeit (Sekunden) in Abhängigkeit von T.

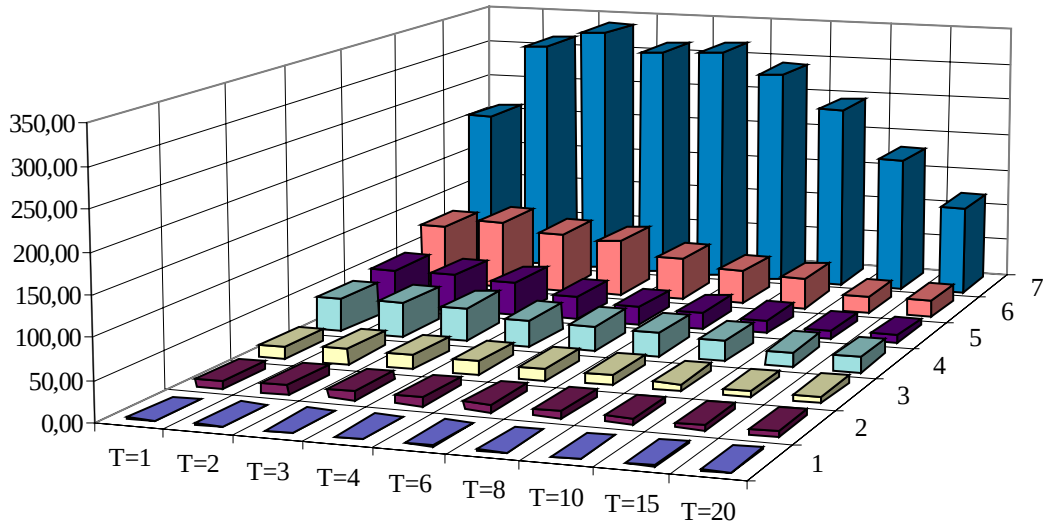


Abbildung 5.1 - Graphische Darstellung des Laufzeitverhaltens.

Der gesamte Algorithmus PSA besteht aus 2 unterschiedlichen Teilen: Teil 1 berechnet in jedem Iterationsschritt eine Savingsmatrix und löst dabei laufend TSPs, Teil 2 löst für jede Savingsmatrix das MWMP und faßt die zugeordneten Cluster zusammen. Um das Verhältnis der Laufzeiten dieser beiden Teile zur Gesamtlaufzeit zu ermitteln, wurden mit einer speziellen Programmversion die Rechendauern einzeln gemessen. Tabelle 5.5 enthält die zum Lösen aller MWMPs benötigte Rechenzeit im Verhältnis zur Gesamtlaufzeit. (Der jeweils verbleibende Zeitanteil wird zum Lösen aller TSPs benötigt.)

Nr.	**T=1**	**T=2**	**T=3**	**T=4**	**T=6**	**T=8**	**T=10**	**T=15**	**T=20**
1	49,49%	50,00%	47,31%	30,68%	15,38%	15,38%	15,38%	15,38%	15,38%
2	34,40%	51,39%	62,72%	61,19%	59,42%	47,28%	33,57%	25,04%	25,04%
3	22,79%	42,62%	39,40%	45,77%	45,99%	29,57%	26,59%	10,82%	9,53%
4	21,50%	43,59%	44,36%	50,64%	56,63%	54,30%	44,55%	18,36%	18,36%
5	20,51%	51,55%	62,35%	85,12%	83,56%	83,32%	80,40%	65,18%	65,45%
6	16,26%	35,76%	40,23%	43,82%	55,32%	57,33%	50,97%	30,95%	19,39%
7	30,59%	67,92%	74,48%	80,48%	79,61%	82,34%	83,63%	79,66%	69,62%

Tabelle 5.5 - Rechenzeit zum Lösen aller MWMPs relativ zur Gesamtlaufzeit.

Die beiden folgenden Abbildungen stellen die absoluten Rechenzeiten der beiden zuvor beschriebenen Programmteile graphisch dar. Vier Tatsachen sind auffällig:

1. Der Rechenaufwand zum Lösen der MWMPs steigt bei konstantem Steuerparameter T mit zunehmender Problemgröße stark an.

2. Der Rechenaufwand zum Lösen der MWMPs eines bestimmten Problems steigt mit wachsendem T zunächst an und sinkt danach tendenziell.
3. Der Rechenaufwand zum Lösen aller TSPs steigt mit der Problemgröße stark an. Ursache dafür ist der offensichtlich überlineare Aufwand des TSP Lösungsalgorithmus.
4. Der Rechenaufwand zum Lösen aller TSPs eines bestimmten Problems sinkt mit wachsendem Steuerparameter T. Ursache dafür ist die mit wachsendem T abnehmende Anzahl an durchzuführenden Iterationen des Algorithmus PSA.

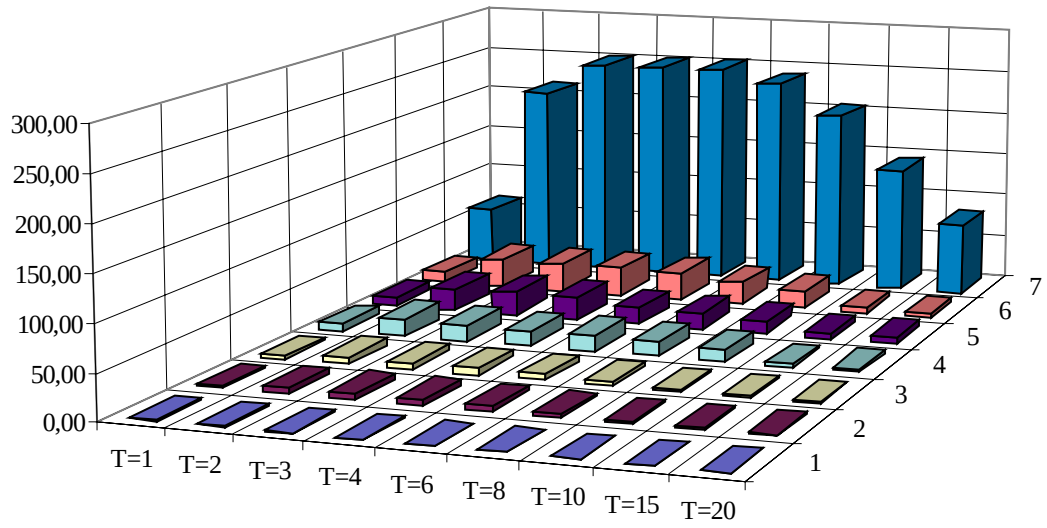


Abbildung 5.2 - Rechenaufwand (Sekunden) zum Lösen aller MWMPs.

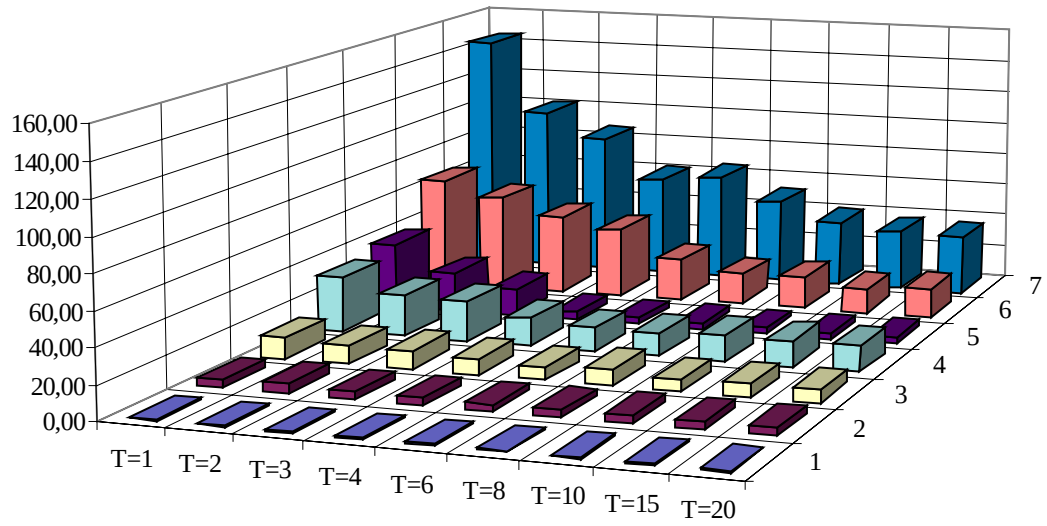


Abbildung 5.3 - Rechenaufwand (Sekunden) zum Lösen aller TSPs.

6 Zusammenfassung

Aufgabenstellung dieser Arbeit war das Implementieren eines heuristischen Lösungsverfahrens (PSA, parallel savings algorithm) für das Vehicle Routing Problem und die Analyse der Leistungsfähigkeit dieses Verfahrens. Die wesentlichen Erkenntnisse werden an dieser Stelle zusammengefaßt.

Der Begriff Vehicle Routing Problem (VRP) wird als Sammelbezeichnung für eine umfangreiche Klasse von Problemen gebraucht. Allen Problemen gemeinsam ist, daß der kostengünstigste Weg, um ein Produkt an mehrere Kunden zu liefern, gesucht wird. Die Vielzahl an untersuchten Problemvarianten und die große Anzahl wissenschaftlicher Publikationen zu diesem Thema demonstrieren die Praxisrelevanz von VRP.

VRP ist ein NP-vollständiges Problem; d.h. nach heutigem Stand der Wissenschaft gibt es kein Lösungsverfahren mit polynomiell begrenzbarer Zeitkomplexitätsfunktion, welches für alle Probleminstanzen eine optimale Lösung findet. Existierende exakte Lösungsverfahren können Probleminstanzen mit bis zu 100 Kunden mit vertretbarem Rechenaufwand lösen. Heuristische Lösungsverfahren liefern mit weitaus geringerem Rechenaufwand „gute“, jedoch i.d.R. nicht optimale, Lösungen des VRP.

Der implementierte Algorithmus PSA ist ein näherungsweises Lösungsverfahren, das unter Berücksichtigung sog. Savings in einem iterativen Verfahren möglichst kostengünstige Liefertouren bildet. Als Teilprobleme sind in jedem Iterationsschritt mehrere Traveling Salesman Problems sowie ein Maximum Weight Matching Problem zu lösen.

Auf der beiliegenden Diskette befindet sich das unter dem Betriebssystem MS-DOS lauffähige Programm PSA, sowie der gesamte Programmcode. Anhand von sieben Testfällen wurde die Leistungsfähigkeit des Lösungsverfahrens hinsichtlich Lösungsqualität und Laufzeit beurteilt:

- PSA liefert Ergebnisse, die im Bereich anderer veröffentlichter Ergebnisse liegen.
- PSA löst auf einem handelsüblichen Personal Computer das VRP mit 30 Kunden in ca. 50 Sekunden.

Literaturverzeichnis

Die alten Wörter sind die besten
und die kurzen die allerbesten.
(Winston Churchill)

ALTINKEMER, K., GAVISH, B. (1991).
Parallel savings based heuristic for the delivery problem. Operations Research Vol. 39 (1991), No. 3, May-Jun., S. 456-469.

AVIS, D. (1983).
A Survey of Heuristics for the Weighted Matching Problem. Networks Vol. 13 (1983), S. 475-493.

BALAS, E., TOTH, P. (1985).
Branch and bound methods. In: Lawler, E. L. et al. (1985), S. 361- 401.

BALL, M. O., DERIGS, U. (1983).
An Analysis of Alternative Strategies for Implementing Matching Algorithms. Networks Vol. 13 (1983), S. 517-549.

BECKMANN, M., KRELLE, W. (HRSG.) (1988).
Lecture Notes in Economics and Mathematical Systems. Vol. 300. Berlin, Heidelberg, New York, London, Paris, Tokyo: Springer-Verlag.

BECKMANN, M., KÜNZI, H. P. (HRSG.) (1980).
Lecture Notes in Economics and Mathematical Systems. Vol. 184. Berlin, Heidelberg, New York: Springer-Verlag.

BOMZE, I. M., GROSSMANN, W. (1993).
Optimierung - Theorie und Algorithmen. Eine Einführung in Operations-Research für Wirtschaftsinformatiker. Mannheim, Leipzig, Wien, Zürich: BI Wissenschaftsverlag.

BRANCO, I. M., COELHO, J. D. (1990).
The hamiltonian p-median problem. European Journal of Operational Research 47 (1990), S. 86-95.

BROWN, G. G., GRAVES, G. W. (1981).
Real-Time Dispatch of Petroleum Tank Trucks. Management Science Vol. 27, No. 1, Jan. 1981, S. 19-32.

BURKARD, R. E., DERIGS, U. (1980).
Assignment and Matching Problems: Solution Methods with FORTRAN-Programs. In: Beckmann, M., Künzi, H. P. (Hrsg.) (1980).

CHRISTOFIDES, N. (1970).
The shortest hamiltonian chain of a graph. SIAM J. Appl. Math. Vol. 19, No. 4, Dec. 1970, S. 689-696.

CHRISTOFIDES, N. (1985).
Vehicle Routing. In: Lawler, E. L. et al. (1985), S. 431- 448.

CHRISTOFIDES, N., EILON, S. (1969).
An Algorithm for the Vehicle-dispatching Problem. Operational Research Quarterly Vol. 20, No. 3, S. 309-318.

CHRISTOFIDES, N., MINGOZZI, A., TOTH, P. (1981A).
State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems. Networks Vol. 11 (1981), S. 145-164.

CHRISTOFIDES, N., MINGOZZI, A., TOTH, P. (1981B).
Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxations. Mathematical Programming 20 (1981), S. 255-282.

CLARKE, G., WRIGHT, J. W. (1964).
Scheduling of vehicles from a central depot to a number of delivery points. Operations Research Vol. 12 (1964), Nr. 4, Jul.-Aug., S. 568-581.

COFFMAN, E. G., LENSTRA, J. K., RINNOOY KAN, A. H. G. (HRSG.) (1992).
Computing. Handbooks in Operations Research and Management Science. Amsterdam, London, New York, Tokyo: North Holland.

DANTZIG, G. B., RAMSER, J. H. (1959).
The Truck Dispatching Problem. Management Science Vol. 6 (1960), No. 1, Oct. 1959, S. 80-91.

DERIGS, U. (1988).
Programming in Networks and Graphs. On the Combinatorial Background and Near-Equivalence of Network Flow and Matching Algorithms. In: Beckmann, M., Krelle, W. (Hrsg.) (1988).

DESROCHERS, M., DESROSIERS, J., SOLOMON, M. (1992).
A new optimization algorithm for the vehicle routing problem with time windows. Operations Research Vol. 40 (1992), No. 2, Mar.-Apr., S. 342-354.

DORNINGER, D., KARIGL, G. (1988).
Mathematik für Wirtschaftsinformatiker. Grundlagen, Modelle, Programme. Band 1. Wien, New York: Springer-Verlag.

DUMAS, Y., DESROSIERS, J., SOUMIS, F. (1991).
The pickup and delivery problem with time windows. European Journal of Operational Research 54 (1991), S. 7-22.

EDMONDS, J. (1965A).
Paths, Trees and Flowers. Canadian Journal of Mathematics Vol. 17 (1965), S. 449-467.

EDMONDS, J. (1965B).
Maximum Matching and a Polyhedron With 0,1-Vertices. Journal of Research of the National Bureau of Standards Vol. 69B (1965), No. 1+2, Jan.-Jun., S. 125-130.

EDMONDS, J., JOHNSON, E. L. (1969).
Matching: a well-solved class of integer linear programs. In: Guy, R. et al. (Hrsg.) (1969), S. 89-92.

FISHER, M. L. (1981).
The Lagrangian Relaxation Method for Solving Integer Programming Problems. Management Science Vol. 27, No. 1, Jan. 1981, S. 1-18.

FISHER, M. L., JAIKUMAR, R. (1981).
A Generalized Assignment Heuristic for Vehicle Routing. Networks Vol. 11 (1981), S. 109-124.

GABOW, H. N. (1976).
An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs. Journal of the Association for Computing Machinery Vol. 23 (1976), No. 2, Apr., S. 221-234.

GASKELL, T. J. (1967).
Bases for Vehicle Fleet Scheduling. Operational Research Quarterly Vol. 18, No. 3, S. 281-295.

GAVISH, B., SRIKANTH, K. (1986).
An Optimal Solution Method for Large-Scale Multiple Traveling Salesmen Problems. Operations Research Vol. 34, No. 5, Sep.-Oct. 1986, S. 698-717.

GOLDBERG, D. E. (1989).
Genetic algorithms in search, optimization, and machine learning. Reading, Menlo Park, Sydney, Don Mills, Madrid, San Juan, New York, Singapore, Amsterdam, Wokingham, Tokyo, Bonn: Addison-Wesley Publishing Company.

GONDRAN, M., MINOUX, M. (1984).
Graphs and Algorithms. Chichester, New York, Brisbane, Toronto, Singapore: John Wiley & Sons.

GUY, R., HANANI, H., SAUER, N., SCHÖNHEIM, J. (HRSG.) (1969).
Combinatorial Structures and Their Applications. Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications held at the University of Calgary, Calgary, Alberta, Canada, June 1969. New York, London, Paris: Gordon and Breach.

HELBIG HANSEN, K., KRARUP, J. (1974).
Improvements of the Held-Karp algorithm for the symmetric traveling-salesman problem. Mathematical Programming 7 (1974), S. 87-96.

HELD, M., KARP, R. M. (1970).
The Traveling-Salesman Problem and Minimum Spanning Trees. Operations Research Vol. 18 (1970), S. 1138-1162.

HELD, M., KARP, R. M. (1971).
The Traveling-Salesman Problem and Minimum Spanning Trees: Part II. Mathematical Programming Vol. 1 (1971), S. 6-25.

KOLEN, A. W. J., RINNOY KAN, A. H. G., TRIENEKENS, H. W. J. M. (1987).
Vehicle Routing with Time Windows. Operations Research Vol. 35, Mar.-Apr. 1987, S. 266-273.

KRUSKAL, J. B. (1956).
On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society, Vol. 7, Feb. 1956, S. 48-50.

LAPORTE, G., NOBERT, Y. (1981).
An exact algorithm for minimizing routing and operating costs in depot location. European Journal of Operational Research 6 (1981), S. 224-226.

LAWLER, E. L., LENSTRA, J. K., RINNOOY KAN, A. H. G., SHMOYS, D. B. (EDS.) (1985).
The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization. London, New York: Wiley & Sons Ltd.

LEITSCH, A., SALZER, G. (1993).
Einführung in die Theorie der Informatik. Skriptum zur Vorlesung. Technische Universität Wien: Institut für Computersprechen, Abteilung für Anwendungen der formalen Logik.

LENSTRA, J. K., RINNOY KAN, A. H. G. (1981).
Complexity of Vehicle Routing and Scheduling Problems. Networks Vol. 11 (1981), S. 221-227.

LIN, S. (1965).
Computer Solutions of the Traveling Salesman Problem. The Bell System Technical Journal Vol. 44, No. 10, Dec. 1965, S. 2245-2269.

LIN, S., KERNIGHAN, B. W. (1973).
An Effective Heuristic Algorithm for the Travelling-Salesman Problem. Operations Research Vol. 21 (1973), No. 2, Mar.-Apr. 1973, S. 498-516.

LIPPMAN, S. B. (1991).
C++ Primer. 2nd Edition. Reading, Menlo Park, New York, Don Mills, Wokingham, Amsterdam, Bonn, Sydney, Singapore, Tokyo, Madrid, San Juan, Milan, Paris: Addison-Wesley Publishing.

LITTLE, J. D. C., MURTY, K. G., SWEENEY, D. W., KAREL, C. (1963).
An Algorithm for the Traveling Salesman Problem. Operations Research Vol. 11, No. 6 (1963), Nov.-Dec., S. 972-989.

LOVÁSZ, L., PLUMMER, M. D. (1986).
Matching Algorithms. Annals of Discrete Mathematics 29 (1986), S. 357-382.

MALEK, M., GURUSWAMY, M., PANDYA, M., OWENS, H. (1989).
Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Annals of Operations Research 21 (1989), No. 1-4, S. 59-84.

MICHALEWICZ, Z. (1994).
Genetic Algorithms + Data Structures = Evolution Programs. Second, Extended Edition. Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona, Budapest: Springer-Verlag.

MINOUX, M. (1976).
Une caractérisation des couplages de poids maximal et de cardinalité fixée dans les graphes pondérés finis. Comptes rendus de l'Académie des Sciences (Paris) A 282 (1976), S. 353-354.

MINOUX, M. (1980).
Solutions entières et solutions continues des problèmes de couplages dans les graphes pondérés finis. Comptes rendus de l'Académie des Sciences (Paris) A 291 (1980), S. 363-364.

OSMAN, I. H. (1993).
Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 41 (1993), No. 1-4, S. 421-451.

PAPADIMITRIOU, C. H., STEIGLITZ, K. (1982).
Combinatorial Optimization. Algorithms and Complexity. London, Sydney, Toronto, New Delhi, Tokyo, Singapore: Prentice-Hall.

ROSS, G. T., SOLAND, R. M. (1975).
A branch and bound algorithm for the generalized assignment problem. Mathematical Programming 8 (1975), S. 91-103.

SEMET, F., TAILLARD, E. (1993).
Solving real-life vehicle routing problems efficiently using tabu search. Annals of Operations Research 41 (1993), No. 1-4, S. 469-488.

SMITH, T. H. C., THOMPSON, G. L. (1977).
A LIFO implicit enumeration search algorithm for the symmetric traveling salesman problem using Held and Karp's 1-tree relaxation. Annals of Discrete Mathematics 1 (1977), S. 479-493.

SOLOMON, M. M. (1987).
Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. Operations Research Vol. 35, No. 2, Mar.-Apr. 1987, S. 254-265.

STEPAN, A., FISCHER, E. O. (1992).
Betriebswirtschaftliche Optimierung. Einführung in die quantitative Betriebswirtschaftslehre. 3., erweiterte und überarbeitete Auflage. München, Wien: R. Oldenbourg, S. 190-266.

STOCKMEYER, L. J. (1992).
Computational Complexity. In: Coffman, E. G. et al. (1992), S. 455-517.

VOLGENANT, T., JONKER, R. (1982),
A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. European Journal of Operational Research 9 (1982), S. 83-89.

WEBER, G. M. (1981).
Sensitivity Analysis of Optimal Matchings. Networks Vol. 11 (1981), S. 41-56.