Traveling Salesman Problem mit Iteriertem Matching

Frank Lauxtermann

UNIVERSITÄT OSNABRÜCK

Diplomarbeit im Studiengang Diplom Mathematik
an der Universität Osnabrück
im Fachbereich Mathematik/Informatik

vorgelegt bei
Professor Dr. Oliver Vornberger
am 12. März 1996

von
Frank Lauxtermann
Bonhoefferstraße 8
49078 Osnabrück

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Wohl jeder Mensch hat sich schon einmal — bewußt oder unbewußt — mit dem *Traveling Salesman Problem* (TSP) beschäftigt. Es ist gleichzeitig ein sehr alltägliches wie mathematisch schwer lösbares Problem. Es geht darum, eine Rundreise durch eine gegebene Menge von Orten zu finden, bei der jeder Ort genau einmal besucht wird und die Streckenlänge minimal ist. Anwendungen finden sich nicht nur für den dem Problem den Namen gebenden Handlungsreisenden, sondern auch für Platinenhersteller, in der Textilindustrie und in der Kristallographie, um nur eine kleine Auswahl zu nennen.

Das TSP gehört zu der Klasse der Optimierungsprobleme. Ein Optimierungsproblem ist charakterisiert durch eine Menge von Variablen, einigen Bedingungen und einer Zielfunktion. Letztere gilt es zu minimieren, wobei die Bedingungen, die den möglichen Wertebereich der Variablen einschränken, einzuhalten sind. Das TSP ist wohl der bekannteste Vertreter der Optimierungsprobleme.

Wie bereits angedeutet, gehört das Traveling Salesman Problem zu den mathematisch schwierigsten Problemen, es ist $\mathcal{NP}$-schwierig. Das bedeutet, daß im allgemeinen eine exakte Lösung nicht oder nur mit nicht vertretbarem Aufwand zu finden ist. Daher versucht man, sich mit Heuristiken zu helfen, die in deutlich geringerer Laufzeit recht gute Näherungslösungen berechnen. Die Lösungsqualität solcher Verfahren ist meistens umgekehrt proportional zur Laufzeit: In kurzer Rechenzeit sind meist nur relativ schlechte Näherungslösungen zu finden.

Obwohl viel Energie in die Suche nach effizienten Heuristiken für das TSP gesteckt wurde, werden immer noch neue Erfolge auf diesem Gebiet erzielt. Zum einen erlaubt die wachsende Leistung moderner Computer die Implementierung von Algorithmen, deren Umsetzung bisher an praktischen Grenzen (zu wenig Speicherplatz, mangelnde Rechenleistung) scheiterten. Es werden aber auch neue Verfahren entwickelt oder aus anderen Einsatzgebieten auf das TSP übertragen.

Auch diese Arbeit stellt eine neue Heuristik vor. Sie basiert auf *Iteriertem Matching*, einem Verfahren, welches bereits bei der Anwendung auf zweidimensionale Packprobleme gute Ergebnisse lieferte. Die Idee besteht darin, die Menge von Orten durch Zusammenfassen sukzessive zu verkleinern. Dazu werden geeignete Ortspaare gebildet. Allgemein bedeutet

„geeignet“ so viel wie „nah beieinander“. Diese Ortspaare werden dann wie ein einzelner neuer Ort aufgefaßt.

Durch diese Maßnahme ist das Problem auf die Suche einer Tour durch eine etwa halb so große Anzahl von Orten reduziert. Zu jedem dieser Orte werden Wege durch die Teilorte gespeichert, aus denen sie bestehen. Wird das Verfahren nun wiederholt auf die immer größer werdenden Orte angewandt, so müssen am Ende nur noch die Wege der letzten beiden Orte verbunden werden. Es entsteht eine Tour durch alle Orte des ursprünglichen Problems.

In Kapitel 1 werden die wichtigsten Grundlagen der Graphentheorie und zur Komplexität von Problemen und Algorithmen vorgestellt. Außerdem werden Testinstanzen vorgestellt, anhand derer das hier erläuterte Verfahren getestet werden soll. In Kapitel 2 werden Probleme aus der Praxis beschrieben, die mit Hilfe von TSP-Heuristiken zu lösen sind. Kapitel 3 faßt eine Reihe von Verfahren für das TSP zusammen. Sie dienen als Grundlage für einen Vergleich mit der neuen Heuristik. Diese wird in den Kapiteln 4 und 5 ausführlich beschrieben. Im ersten Teil werden die einzelnen Schritte des Verfahrens erläutert, im zweiten dann die Ergebnisse aufgezeigt.

Um die Suche nach einer Lösung verfolgen zu können, wurde neben einer rein textorientierten Fassung des Programms auch eine grafische Benutzeroberfläche implementiert. Ihre Möglichkeiten werden in Kapitel 6 dargelegt. Eine zusammenfassende Betrachtung bildet neben einem Ausblick den Abschluß dieser Arbeit.

Kapitel 1

Definition

Dieses Kapitel soll die wichtigsten Begriffe aus der Graphentheorie und der Optimierung einführen, sowie eine formale Definition des *Traveling Salesman Problems* vorstellen. Wichtige Grundlagen für die Analyse der Komplexität von Algorithmen bilden einen weiteren Abschnitt. Abschließend werden Probleminstanzen vorgestellt, anhand derer das in dieser Arbeit erläuterte Verfahren getestet werden soll.

1.1 Das Traveling Salesman Problem

Das Traveling Salesman Problem findet man sowohl in der Graphentheorie als auch in der kombinatorischen Optimierung. Dementsprechend gibt es viele, selbstverständlich äquivalente Definitionen. Hier soll eine graphentheoretische Formulierung vorgestellt werden. Dazu bedarf es jedoch einiger anderer Definitionen vorweg.

Zuerst sollen die wichtigsten Begriffe aus der Graphentheorie eingeführt werden. Ein *(ungerichteter* oder *symmetrischer) Graph* $G = (V, E)$ besteht aus einer endlichen Menge von *Knoten* V und *Kanten* E. Eine Kante $e \in E$ hat zwei *Endknoten* $u, v \in V$. Sie wird daher auch mit $e = uv$ oder $e = (u, v)$ bezeichnet. In diesem Fall heißt es auch, e *inzidiert* mit u und v. Die Anzahl aller mit u inzidierenden Kanten heißt der *Grad* von u. Ein Graph heißt *gerichtet oder asymmetrisch*, wenn die Kanten (u, v) und (v, u) nicht identisch sind.

In einem *gewichteten* Graphen sind den Kanten *Gewichte* oder *Kosten* zugeordnet. Die Kosten einer Kante uv werden mit c_{uv}, die Kosten einer Kantenmenge $F \subset E$ mit $c(F) = \sum_{(u,v)\in F} c_{uv}$ bezeichnet. Sind den Kanten keine Kosten zugeordnet, heißt der Graph *ungewichtet*.

Ein Graph heißt *vollständig*, wenn er für jedes Knotenpaar $u, v \in V$ die Kante $(u, v) \in E$ enthält. Ein vollständiger Graph auf n Knoten wird auch durch K_n bezeichnet. Ein Graph

$G' = (V', E')$ heißt *Teilgraph* von G, wenn $V' \subset V$ und $E' \subset E$. Ein *Weg* ist eine Menge von Kanten $\{v_1 v_2, v_2 v_3, \ldots, v_k\}$ mit $v_i \neq v_j$ wenn $i \neq j$ und $(v_i, v_{i+1}) \in E$ für alle $i = 1, \ldots, k-1$ gilt. Ein Graph heißt *zusammenhängend*, wenn es einen Weg zwischen jedem u und v aus V gibt. Eine *Zusammenhangskomponente* ist ein zusammenhängender Teilgraph von G.

Die folgenden Definitionen beziehen sich auf ungerichtete, gewichtete Graphen.

Definition 1.1 (Zyklus). Ein Menge von Kanten $C = \{v_1 v_2, v_2 v_3, \ldots, v_k v_1\}$ heißt *Zyklus der Länge* k oder k*-Zyklus*, wenn $v_i \neq v_j$ für $i \neq j$. Abgekürzt schreibt man auch $C = (v_1, v_2, \ldots, v_k)$. Ein Graph heißt *nicht-zyklisch*, wenn er keinen Zyklus enthält. □

Definition 1.2 (Hamilton-Tour). Ein Zyklus der Länge n in einem Graphen mit n Knoten heißt *Hamilton-Tour*. Die *Kosten* eine Hamilton-Tour T sind definiert durch $c(T) = \sum_{(u,v) \in T} c_{uv}$. □

Selbstverständlich gibt es Graphen, auf denen es keine Hamilton-Touren gibt. Diese sind aber in der Praxis ohne große Bedeutung. Besonders interessant dagegen sind vollständige Graphen, deren Knoten Punkte in einer Ebene repräsentieren und deren Kantengewichte sich durch metrische Funktionen (Abstände) berechnen lassen. In diesen existieren immer Hamilton-Touren. Sie seien im folgenden *geometrische* Graphen genannt.

Ein spezieller Zyklus auf einem vollständigen Graphen ist die *konvexe Hülle*.

Definition 1.3 (Konvexe Hülle). Eine Menge $W \subset \mathbb{R}^2$ heißt *konvex*, wenn für zwei Punkte $x, y \in W$ auch $[x, y] \in W$ gilt. Die *konvexe Hülle* einer endlichen Teilmenge $S = \{x_1, \ldots, x_n\} \subset \mathbb{R}^2$ ist die kleinste konvexe Menge $H_S \subset \mathbb{R}^2$, für die gilt $x_i \in H_S$ für alle $x_i \in S$.

Definition 1.4 (Symmetrisches Traveling Salesman Problem). Gegeben sei der vollständige Graph K_n. Das *(Symmetrische) Traveling Salesman Problem* besteht darin, auf diesem Graphen eine Hamilton-Tour zu finden, deren Kosten minimal sind, also eine Hamilton-Tour T_{min}, für die gilt $c(T_{min}) = \min_{T \text{ ist Hamilton-Tour}} c(T)$. □

Man beachte, daß es in der Regel keine eindeutige Lösung dieses Problems gibt. Abbildung 1.1 zeigt ein Beispiel.

Die Lösungssuche für ein TSP auf einem nicht vollständigen, aber symmetrischen Graphen $G = (V, E)$ kann auf einem vollständigen Graphen simuliert werden. Dazu werden einfach alle fehlenden Kanten eingefügt, wobei ihre Gewichte sehr hoch gesetzt werden, beispielsweise auf $\sum_{(i,j) \in E} c_{ij}$. Wird nun eine optimale Lösung dieses Problems gefunden, in der eine dieser eingefügten Kanten auftritt, so gibt es keinen Hamilton-Zyklus im originalen Problem. Ansonsten ist die optimale Lösung auch für das originale Problem gültig.

Ist der Graph $G = (V, E)$ mit den Kantengewichten c_{ij} asymmetrisch, gibt es also Kanten (i, j) und (j, i) mit $c_{ij} \neq c_{ji}$, hilft man sich mit einem ähnlichen Trick. Man erstellt einen

Abbildung 1.1: *Zwei verschiedene optimale Lösungen für ein Problem*

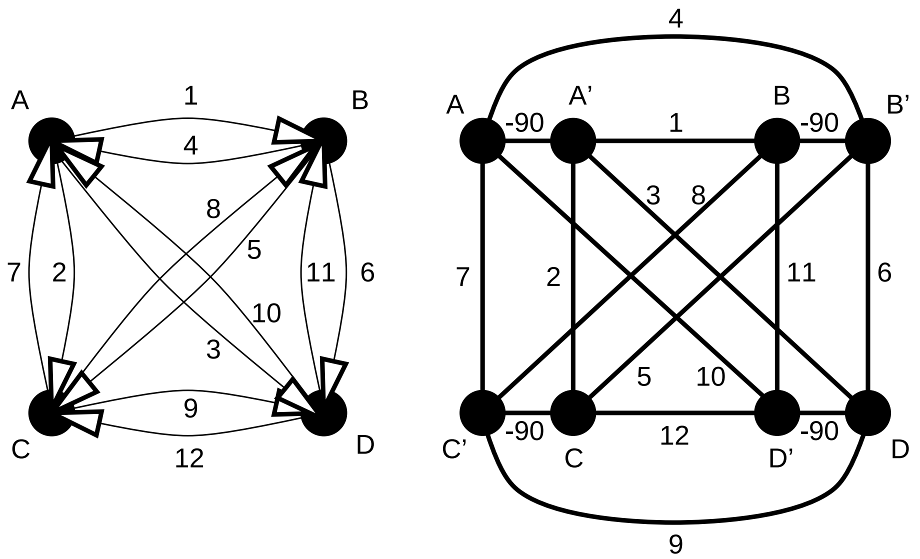

Abbildung 1.2: *Ein asymmetrischer Graph (links) und seine Transformation in einen symmetrischen Graphen (rechts).*

neuen Graphen $\tilde{G}$ dessen Knotenmenge $\tilde{V}$ zusätzlich zu den Knoten aus $V = \{1, \ldots, n\}$ $n = |V|$ weitere Knoten $\tilde{1}, \ldots, \tilde{n}$ enthält. Ein Knoten $\tilde{\imath}$ heiße *Spiegelknoten* zu i. Als Kantenmenge nimmt man $\tilde{E} = \{(i, \tilde{\imath}) : i = 1, \ldots, n\} \cup \{(\tilde{\imath}, j) : (i, j) \in E\}$. Man kommt also von einem Originalknoten immer nur zu einem Spiegelknoten und umgekehrt. Die Kantengewichte werden mit $\tilde{c}_{i,\tilde{\imath}} = -M$ für $i = 1, \ldots, n$ und $\tilde{c}_{\tilde{\imath},j} = c_{ij}$ für $(i, j) \in E$ gewählt. Dabei ist M ausreichend groß zu wählen, beispielsweise $M = \sum_{(i,j)\in E} c_{ij}$. Es ist offensichtlich, daß für jeden gerichteten Hamilton-Zyklus in G mit den Kosten c ein ungerichteter Hamilton-Zyklus mit den Kosten $\tilde{c} = c - nM$ existiert. Dazu geht man einfach für jede Kante (i, j) im Hamilton-Zyklus auf G den Weg $(i\tilde{\imath}j\tilde{\jmath})$ auf $\tilde{G}$. Außerdem wird jede optimale Tour in $\tilde{G}$ alle Kanten mit dem Gewicht $-M$ enthalten. Solch eine Tour kann man wieder zu einem gerichteten Hamilton-Zyklus in G wandeln, indem man einfach den oben genannten Schritt umgekehrt durchführt.

Das asymmetrische Traveling Salesman Problem und das TSP auf allgemeinen (nichtvollständigen) Graphen lassen sich also durch einen Algorithmus für das symmetrische TSP lösen. Die folgenden Betrachtungen beschränken sich daher auf das symmetrische Traveling Salesman Problem, es sei denn, es wird ausdrücklich auf einen anderen Fall hingewiesen.

Nun gehört das TSP zu der Klasse der $\mathcal{NP}$-schwierigen Optimierungsprobleme. Für diese lassen sich im allgemeinen keine exakten Lösungen finden, beziehungsweise würde die Suche danach zu großen Aufwand bedeuten. Natürlich werden auch mehr und mehr Verfahren entwickelt, um optimale Lösungen für das TSP zu finden, doch ist es in der Praxis von größerem Interesse, möglichst schnell möglichst gute Näherungen zu finden. Wird im folgenden von Lösungen gesprochen, so sind damit Näherungslösungen gemeint, eine wirkliche Lösung des Problems 1.4 wird dagegen von nun an „optimale Lösung“ genannt.

1.2 Komplexität von Algorithmen

Neben der Qualität einer Näherungslösung spielt auch die benötigte Zeit eine erhebliche Rolle bei der Beurteilung eines Algorithmus. So kann es beispielsweise sinnvoller sein, eine schlechtere Lösung in Kauf zu nehmen, als viele Stunden Rechenzeit zu investieren. Bei Algorithmen, die in jedem Fall exakte Lösungen liefern sollen, ist die Zeit das Hauptkriterium für ihre Beurteilung.

1.2.1 Die O-Notation

Selbstverständlich kann man keine Aussagen über die wirkliche Laufzeit eines Algorithmus machen, da diese von der tatsächlichen Implementierung und der Plattform, auf der diese läuft, abhängt. Im allgemeinen ist man aber auch mehr daran interessiert, Aussagen über das Verhalten bei wachsender Größe n der zu behandelnden Probleminstanzen zu bekommen.

Die *O-Notation* gibt eine obere Schranke für den Aufwand im ungünstigsten Fall — *worst case* genannt — an.

Definition 1.5. Gegeben seien $f : \mathbb{N} \to \mathbb{N}$ und $g : \mathbb{N} \to \mathbb{N}$. Die Funktion f heißt $O(g(n))$ in ihrer *Input-Länge* n (*oder:* hat den Aufwand $O(g(n))$), wenn positive Konstanten c und n_0 existieren, so daß $0 \leq f(n) \leq c \cdot g(n)\ \forall n \geq n_0$. □

Beispielsweise gibt es für die Suche eines Maximums in einer unsortierten Menge von n Zahlen Algorithmen, die dieses mit linearem Aufwand lösen. Sie sind $O(n)$.

Die Input-Länge ist ein Maß für die Menge der Eingabedaten für eine zu lösende Probleminstanz. Das Auffinden des Maximums von n Zahlen benötigt nur diese Zahlen als Eingabedaten. Die Input-Länge beträgt also n. Sucht man dagegen die Kante eines Graphen $G = (V, E)$ mit den größten Kosten, so ist die Input-Länge die Anzahl der Kanten $|E|$.

Die O-Notation kann sowohl für die Laufzeit als auch für den Speicherplatzbedarf eines Algorithmus benutzt werden. Wird im folgenden gesagt, ein Algorithmus habe einen Aufwand $O(g(n)))$, so ist grundsätzlich der zeitliche Aufwand gemeint. Wird die Notation für den Speicherplatz gebraucht, wird dies ausdrücklich notiert. Außerdem beziehen sich alle Aussagen auf sequentielle Algorithmen.

Man ist natürlich daran interessiert, für gegebene Probleme möglichst schnelle Algorithmen zu finden. Daher gibt man nicht eine beliebige obere Schranke für einen Algorithmus an, sondern (von Konstanten abgesehen) die kleinste, also die asymptotische Laufzeit des Algorithmus. Beispielsweise ist der *Quicksort-Algorithmus* für das Sortieren von n Zahlen nicht nur $O(n^2)$, wie auch der *Bubblesort*, sondern $O(n \log n)$.

Bei der Suche nach dem schnellsten Algorithmus gibt es natürlich Grenzen. So gibt es beispielsweise keinen Algorithmus, der das Sortieren mit geringerem Aufwand als $O(n \log n)$ löst. Es ist sinnvoll, die O-Notation auch auf Probleme zu übertragen.

Definition 1.6. Gegeben sei $g : \mathbb{N} \to \mathbb{N}$. Ein Problem läßt sich in $O(g)$ lösen, wenn es einen Algorithmus gibt, der beliebige Probleminstanzen in $O(g)$ löst. □

Von einem Algorithmus A, für den ein Polynom p existiert, so daß A den Aufwand $O(p(n))$ hat, sagt man, er habe polynomialen Aufwand beziehungsweise er ist polynomiell. Alle anderen Algorithmen haben exponentiellen Aufwand (sind exponentiell).

1.2.2 Die Klassen $\mathcal{P}$ und $\mathcal{NP}$

Bei der Untersuchung der Probleme auf ihre Komplexität werden zwei Klassen unterschieden. Zum einen gibt es die *Entscheidungsprobleme*, deren Lösung „Ja“ oder „Nein“ lautet, zum anderen die *Optimierungsprobleme*. Ein Optimierungsproblem ist charakterisiert durch eine Menge von Variablen, durch Bedingungen, die den Wertebereich dieser Variablen einschränken und durch eine *Zielfunktion*, die unter Beachtung der Bedingungen minimiert werden soll.

Definition 1.7. Die Klasse $\mathcal{P}$ bezeichne die Menge aller Entscheidungsprobleme, die mit polynomialem Aufwand lösbar sind. □

Die meisten interessanten Probleme aus der kombinatorischen Optimierung gehören zur Klasse $\mathcal{NP} \supset \mathcal{P}$. Sie lassen sich bisher nicht mit polynomialem Aufwand lösen. Es ist sehr wahrscheinlich, daß dieses auch gar nicht möglich ist.

Die Definition der Klasse $\mathcal{NP}$ kann mit Hilfe der *nicht-deterministischen Turing Maschine* geschehen. An dieser Stelle soll keine vollständige Definition der Turing Maschine gegeben werden. Für das Verständnis der Klasse $\mathcal{NP}$ genügt es, sich die nicht-deterministische Turing Maschine als ein theoretisches Werkzeug vorzustellen, das die üblichen Algorithmen verarbeiten kann, aber noch einen weiteren Befehl erlaubt. Dieser Befehl ermöglicht es, zwei verschiedene Abschnitte eines Algorithmus gleichzeitig auszuführen. Liefert ein Abschnitt des Algorithmus für eine Instanz eines gegebenen Entscheidungsproblems die Antwort „Ja“, dann wird die Instanz mit „Ja“ beantwortet und die Abarbeitung des Algorithmus beendet. Nur wenn kein Abschnitt „Ja“ liefert, lautet die Antwort für die Instanz „Nein“.

Definition 1.8. Für ein Entscheidungsproblem P gilt genau dann $P \in \mathcal{NP}$, wenn die nicht-deterministische Turing Maschine für jede beliebige Instanz des Problems die richtige Antwort liefert und sie für jede „Ja“-Antwort polynomialen Aufwand benötigt. □

Da die nicht-deterministische Turing Maschine auch deterministische Algorithmen verarbeiten kann, gilt $\mathcal{P} \subseteq \mathcal{NP}$. Bisher konnte aber weder $\mathcal{P} \neq \mathcal{NP}$ noch $\mathcal{P} = \mathcal{NP}$ bewiesen werden. Es gibt gute Gründe dafür, daß $\mathcal{P}$ eine echte Teilmenge ist. Schließlich hat man für eine Vielzahl von Problemen nur Algorithmen exponentiellen Aufwands gefunden.

Definition 1.9 (Polynomielle Reduzierbarkeit). Ein Problem A heißt *polynomiell reduzierbar* auf ein Problem B, wenn es einen polynomiellen Algorithmus für A gibt, der die Lösung von B als Teilschritt benutzt. Jeder Aufruf des Algorithmus für B zählt dabei als ein Schritt, unabhängig von dem tatsächlich notwendigen Aufwand. □

Definition 1.10 ($\mathcal{NP}$-Vollständigkeit). Ein Entscheidungsproblem P heißt *$\mathcal{NP}$-vollständig*, wenn einerseits $P \in \mathcal{NP}$ und andererseits jedes Problem aus $\mathcal{NP}$ polynomiell reduzierbar auf P ist. □

Mit anderen Worten: Wird für ein $\mathcal{NP}$-vollständiges Problem ein polynomieller Algorithmus gefunden, so sind automatisch alle $\mathcal{NP}$-vollständigen Probleme mit polynomialem Aufwand lösbar und es würde $\mathcal{P} = \mathcal{NP}$ gelten. Da viele Forscher für die unterschiedlichsten dieser Probleme vergeblich nach polynomiellen Algorithmen gesucht haben, wird angenommen, daß dem nicht so ist. Der Beweis ist aber — wie schon erwähnt — noch nicht erbracht.

Bisher wurden nur Theorien für Entscheidungsprobleme vorgestellt. Um Aussagen über das TSP machen zu können, benötigt man analoge Aussagen für Optimierungsprobleme. Diese gewinnt man, indem man jedem Optimierungsproblem ein Entscheidungsproblem zuordnet. Das Verfahren soll anhand des TSPs erläutert werden.

Definition 1.11 (Traveling Salesman Entscheidungsproblem). Gegeben sei der Graph $G = (V, E)$, und eine Zahl b. Das *Traveling Salesman Entscheidungsproblem* ist dann die Frage, ob es einen Hamilton-Zyklus gibt, dessen Kosten maximal b sind. □

Dieses Problem ist $\mathcal{NP}$-vollständig [Joh 1985]. Ein Optimierungsproblem, dessen zugehöriges Entscheidungsproblem $\mathcal{NP}$-vollständig ist, nennt man *$\mathcal{NP}$-schwierig*. Das heißt in diesem Fall

Satz 1.12. Das Traveling Salesman Problem ist $\mathcal{NP}$-schwierig. □

Mit Hilfe des Entscheidungsproblems kann man auch das Optimierungsproblem lösen. Dazu überlegt man sich, daß eine Hamilton-Tour auf einem Graph $G = (V, E)$ niemals teurer als $U = c_{max} \cdot n$ mit $c_{max} = \max_{(i,j)\in E} |c_{ij}|$ und $|V| = n$ sein kann. Ebenso wenig kann sie billiger als $L = -c_{max} \cdot n$ sein. Mit diesen beiden Werten als oberer und unterer Grenze beginnt man eine *binäre Suche*. Man löst also das Entscheidungsproblem für $b = \frac{L+U}{2}$, verkleinert das Suchintervall je nach Ergebnis auf $[L, b]$ oder $]b, U]$ und wiederholt diese Schritte, bis obere und untere Grenze des Suchintervalls zusammenfallen. Diese Grenze ist dann der Wert b_{opt} einer optimalen Lösung.

Mit Hilfe dieses Wertes kann man die optimale Lösung selbst bestimmen. Dazu erhöht man nach und nach die Kosten jeder Kante des Graphen auf $U + 1$, überprüft nach dieser Änderung, ob es immer noch einen Hamilton-Zyklus mit den Kosten b_{opt} gibt und macht die Änderung im negativen Fall rückgängig. Alle nicht geänderten Kanten stellen dann eine optimale Lösung dar.

Ist also ein Entscheidungsproblem in $\mathcal{P}$, so ist auch das zugehörige Optimierungsproblem polynomiell lösbar, denn das Verfahren zur Suche des Wertes der optimalen Lösung fragt $O(\log n)$ mal nach einer Antwort auf das Entscheidungsproblem, der Algorithmus zur Bestimmung der Lösung selbst im Falle des TSP $O(n^2)$ mal.

Nun wäre man vielleicht zufrieden, gäbe es wenigstens einen Algorithmus, der in polynomialer Zeit eine Lösung mit den Kosten c_H liefert, für die gilt $c_H \leq r \cdot c_{opt}, r \geq 1$. Dann könnte man Verfahren implementieren, die mit polynomialem Aufwand gute Näherungslösungen liefern. Leider ist auch das nicht möglich, wie in [Sah 1976] bewiesen wird.

1.3 Testinstanzen

Um einen Vergleich mit anderen Verfahren zu ermöglichen, gibt es eine Reihe von Beispielinstanzen, die größtenteils praktische Probleme repräsentieren. In der TSPLIB [Rei 1995] von Gerhard Reinelt sind viele dieser Beispielinstanzen in einer einheitlichen Beschreibung verfügbar. Eine kleine Auswahl dieser Beispiele schlägt Reinelt in [Rei 1994] vor. Sowohl das Beschreibungsformat als auch die Beispielinstanzen (Tabelle 1.1) wurden hier übernommen. In der Tabelle sind die Namen der Beispielinstanzen, ihre Größe (die Anzahl der Knoten) und die Kosten einer optimalen Lösung gegeben. Ist bisher noch keine optimale Lösung gefunden worden, so sind die Grenzen angegeben, innerhalb derer sich die Kosten einer solchen befinden werden.

Problem	Größe	Grenzkosten
d198	198	15780
lin318	318	42090
fl417	417	11841
pcb442	442	50778
u574	574	36905
p654	654	34643
rat783	783	8806
pr1002	1002	259045
u1060	1060	224094
pcb1173	1173	56892
d1291	1291	50801
rl1323	1323	270199
fl1400	1400	20127
u1432	1432	152970
fl1577	1577	22137–22249
d1655	1655	62128
vm1748	1748	336556
rl1889	1889	316536
u2152	2152	64163–64294
pr2392	2392	378032
pcb3038	3038	137694
fl3795	3795	28594–28772
fnl4461	4461	182566
rl5934	5934	554070–556146

Tabelle 1.1: *Beispielinstanzen für das TSP mit den Kosten bzw. den Grenzen der optimalen Lösungen*

Bei den Beispielinstanzen handelt es sich ausnahmslos um geometrische Probleme. Dies zeigt die außergewöhnliche Bedeutung dieser Gruppe. Auf eine Auswahl praxisnaher Probleme wurde Rücksicht genommen, da zufällige Instanzen kaum die Besonderheiten praktischer Anwendungen modellieren können.

Die Qualität eines Verfahrens wird bestimmt durch die Lösungen, die es liefert. Die Lösungen werden daher nicht in absoluten Zahlen, sondern mit Hilfe der Formel

$$q = 100 \cdot \left(\frac{c_H}{c_L} - 1 \right)$$

angegeben. Dabei sind c_H die Kosten der durch die Heuristik gefundenen Lösung und c_L die Kosten der optimalen Lösung beziehungsweise die untere Grenze dafür.

Kapitel 2

Das TSP in der Praxis

Die praktische Relevanz des TSP ist schon durch die Definition einsichtig. Es gibt unzählige Varianten des Problems, eine kürzeste Tour durch eine vorgegebene Anzahl an Orten zu finden. Doch es gibt auch noch andere Anwendungen, bei denen Heuristiken zum TSP sinnvoll und effektiv eingesetzt werden können. Einige Beispiele sollen an dieser Stelle aufgeführt werden.

2.1 Platinenproduktion

Eine Standardanwendung für TSP-Heuristiken ist das Bohren von Platinen. Um einen Leiter von einem Layer der Platine auf ein anderes zu führen, oder um Bausteine auf der Platine befestigen zu können, muß in die Platine ein Loch gebohrt werden. Dabei können unterschiedliche Durchmesser erforderlich sein. Es ist klar, daß das Wechseln des Bohrers erheblich mehr Zeit erfordert, als das Bewegen des Bohrers und das Bohren an sich. Daher wird man alle Löcher gleichen Durchmessers in einem Durchgang bohren und erst dann den Bohrer wechseln.

Nimmt man also die Ausgangsposition der Bohrmaschine sowie die Positionen aller Löcher eines Durchmessers als Knoten und die Zeit, die beim Bewegen des Bohrers von einer Position zur nächsten benötigt wird, als Kantengewichte, so hat man ein Traveling Salesman Problem, von dem in diesem Fall mehrere nacheinander zu lösen sind. Je eine Lösung stellt den kürzesten Weg für den Bohrer über alle Löcher mit gleichem Durchmesser dar.

Doch schon vor der eigentlichen Produktion der Platine ist das TSP anwendbar. Für jedes Layer der Platine muß eine Maske produziert werden. Diese wird im allgemeinen mit einem mechanischen Plotter erstellt. Auch die Steuerung dieses Gerätes kann durch eines der im folgenden vorgestellten Verfahren optimiert werden.

2.2 Auftrags-Abarbeitung in einem Lagerhaus

Aus einem Lager sei eine Menge unterschiedlicher Gegenstände zu besorgen. Dazu wird ein Fahrzeug durch die Gänge zu den einzelnen Regalen gefahren, das Material gesammelt und am Ende abgeliefert. Betrachtet man die Orte, an denen die Gegenstände gelagert sind, als Knoten eines Graphen und die Zeit, die von einem Ort zum nächsten benötigt wird als Kantengewichte, so hat man ein Traveling Salesman Problem. Eine Lösung des Problems stellt die zeitlich kürzeste Tour für das Fahrzeug dar, mit der alle Gegenstände besorgt werden können.

2.3 Kristall-Analyse

Die Analyse von Kristallen kann durch die Messung unterschiedlicher Intensitäten von reflektierten Röntgenstrahlen geschehen. Dazu muß eine entsprechende Apparatur zu mehreren zehntausend Positionen bewegt werden, an der sie die Messungen durchführen kann. Auch hier hat man mit den verschiedenen Positionen als Knoten und den Zeiten, die benötigt werden, um die Positionen anzusteuern, als Kantengewichte ein Traveling Salesman Problem in drei Dimensionen vorliegen. Seine Lösung — also die zeitlich kürzeste Tour über alle Positionen — kann aufgrund der großen Zahl an Knoten erhebliche Zeitgewinne gegenüber einer zufällig gewählten Tour bedeuten.

2.4 Zuschneiden von Tapeten

Bisher sind nur Anwendungen aufgezählt worden, bei denen die Analogie zum Traveling Salesman Problem offensichtlich war. Das Zuschneiden von Tapeten scheint dagegen keinerlei Ähnlichkeit aufzuweisen.

Es seien n Bahnen einer Tapete von einer Rolle zuzuschneiden. Diese Tapete habe ein Muster, dessen Länge 1 sein soll. Das läßt sich mit geeigneter Skalierung immer erreichen. Die Bahn i soll nun an Position a_i im Muster beginnen und an Position b_i enden, $0 \leq a_i, b_i \leq 1$ (siehe Abb. 2.1). Wird nun die Bahn j direkt nach Bahn i geschnitten, so entsteht ein Verschnitt von

$$c_{ij} = \begin{cases} a_j - b_i & \text{wenn } b_i \leq a_j \\ 1 + a_j - b_i & \text{sonst} \end{cases}$$

Legt man nun noch den Anfang der Rolle mit a_0 fest und verlangt man, daß abschließend noch ein Schnitt gemacht wird, so daß die Rolle auch wieder bei $b_0 = a_0$ endet, so wird die Aufgabe, den Gesamtverschnitt zu minimieren, zu einem TSP mit $n + 1$ Knoten und

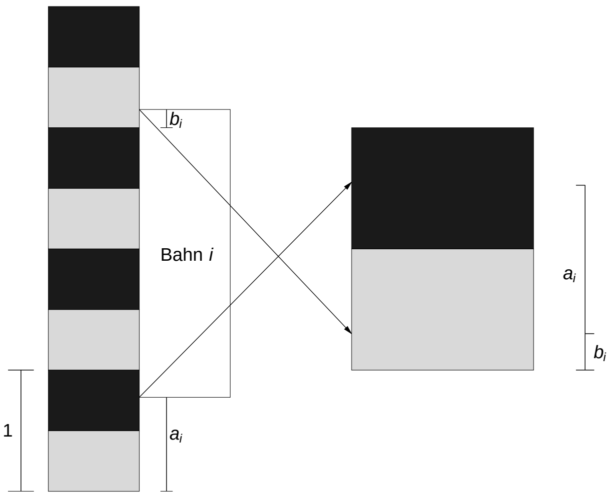

Abbildung 2.1: *Die Werte a_i und b_i bezeichnen den Anfang und das Ende einer Tapetenbahn bezüglich eines Musters*

der Kostenmatrix (c_{ij}). Die Lösung dieses Problems beschreibt die Reihenfolge, in der die Bahnen von der Rolle abzuschneiden sind.

2.5 Gruppierung von Datenfeldern

In vielen Bereichen gibt es Fragen nach dem gegenseitigen Einfluß unterschiedlicher Faktoren. Dazu kann man eine Matrix $A = (a_{ij})$ aufstellen, in der der Wert von a_{ij} ein Maß für den Einfluß eines Faktors i auf einen Faktor j ist. Beispielsweise könnten die Zeilen Marketing Strategien darstellen, während die Spalten Situationen kennzeichnen, in denen diese Strategien erfolgreich eingesetzt wurden.

Das Ziel ist es, die Zeilen und Spalten der Matrix so zu vertauschen, daß Abhängigkeiten von Teilmengen der Zeilen und Spalten klar sichtbar werden. In Abbildung 2.2 wird nach den Permutationen klar, daß zwischen den Zeilen $1, 3, 5$ und den Spalten $1, 3$ starke Abhängigkeiten existieren. Das gleiche gilt für die Zeilen $2, 4$ und die Spalten $2, 4$. Um dieses Ziel zu erreichen, wird ein *Effektivitätsmaß* eines Permutationspaares $\mathrm{Eff}(\sigma, \rho)$ eingeführt. Seien σ und ρ Permutationen der Zeilen beziehungsweise der Spalten. Dann erhält man die Summe der Produkte von benachbarten Elementen in einer $m \times n$-Matrix durch

$$\mathrm{Eff}(\sigma, \rho) = \sum_{i=0}^{m} \sum_{j=1}^{n} a_{\sigma(i)j} a_{\sigma(i+1)j} + \sum_{j=0}^{n} \sum_{i=1}^{m} a_{i\rho(j)} a_{i\rho(j+1)},$$

$$\begin{array}{cc} \begin{array}{c}1\\2\\3\\4\\5\end{array} & \begin{pmatrix}1&0&1&0\\0&1&0&1\\1&0&1&0\\0&1&0&1\\1&0&1&0\end{pmatrix} \\ & \begin{array}{cccc}1&2&3&4\end{array} \\ & \text{(a)} \end{array} \qquad \begin{array}{cc} \begin{array}{c}1\\3\\5\\2\\4\end{array} & \begin{pmatrix}1&1&0&0\\1&1&0&0\\1&1&0&0\\0&0&1&1\\0&0&1&1\end{pmatrix} \\ & \begin{array}{cccc}1&3&2&4\end{array} \\ & \text{(b)} \end{array}$$

Abbildung 2.2: *Gruppierung von Datenfeldern*

wobei Füllzeilen und -spalten zugefügt wurden, deren Elemente alle 0 sind. Für die Permutationen gilt $\sigma(m+1) = \rho(n+1) = \sigma(0) = \rho(0) = 0$. Beide Summen können unabhängig voneinander optimiert werden. Für diese Optimierung werden zwei Traveling Salesman Probleme gelöst, eines mit der Kostenmatrix

$$c_{ij} = -\sum_{k=1}^{n} a_{ik}a_{jk}$$

für das Zeilenproblem und entsprechend eines mit der Kostenmatrix

$$\tilde{c}_{ij} = -\sum_{k=1}^{m} a_{ki}a_{kj}$$

für das Spaltenproblem. Die negativen Vorzeichen sind nötig, um das urspüngliche Maximierungsproblem auf ein Minimierungsproblem zu überführen. Die Lösung der Probleme beschreibt die Reihenfolge, in der die Zeilen beziehungsweise die Spalten anzuordnen sind. Eine Lösung für das in Abb. 2.2a) gezeigte Problem ist (wie in (b) zu sehen) $1, 3, 5, 2, 4$ für das Zeilenproblem und $1, 3, 2, 4$ für das Spaltenproblem.

2.6 Verwaltung eines Fuhrparks

Bei der Verwaltung eines Fuhrparks hat man neben der Tatsache, daß man für mehrere Fahrzeuge optimale Routen finden will, auch andere Ergänzungen und Einschränkungen des TSP zu beachten. So hat beispielsweise jedes Fahrzeug nur eine bestimmte Kapazität, bei der Auslieferung müssen eventuell Termine eingehalten werden, Tank- und Raststops müssen eingeplant werden. Desweiteren müssen manche Orte gegebenenfalls vor anderen angefahren werden. Solche Orte wären Depots, an denen die Ladung erst einmal aufgenommen werden muß.

Das *Vehicle Routing Problem* umfaßt die meisten dieser Punkte, die anderen können durch geschickte Modellierung aufgenommen werden. Gibt es keine zeitlichen Einschränkungen und ist die Zahl m der Fahrzeuge fest, so läßt sich das Vehicle Routing Problem als TSP lösen. Dazu steht zunächst jedes Fahrzeug an einem festgelegten Knoten $n + 1$. Jeder Knoten des Graphen muß von genau einem Fahrzeug genau einmal besucht werden. Man verbindet mit dem Einsatz eines Fahrzeuges i noch zusätzliche Kosten w_i. Dadurch werden Betriebskosten simuliert, die unabhängig von der Streckenlänge beim Einsatz eines Fahrzeuges entstehen. Die Aufgabe, unter allen m Fahrzeugen eine Teilmenge auszuwählen, die alle Städte unter den gegebenen Nebenbedingungen besucht, nennt man auch das *m-Salesman Problem* oder *m-TSP*. In [Bel 1974] wird eine Transformation des m-TSP in ein asymmetrisches TSP vorgestellt.

Kapitel 3

Bekannte Lösungsstrategien

Die zur Lösung des TSP eingesetzten Heuristiken lassen sich zwei Klassen zuordnen. Zum einen gibt es die „konstruktiven Verfahren“, die mit unterschiedlichen Strategien die Knoten des Graphen nach und nach zu einer Tour verbinden. Sie werden vornehmlich eingesetzt, um möglichst schnell grobe Näherungslösungen zu ermitteln. Die „iterativen Verfahren“, die eine gegebene Lösung mit geeigneten Veränderungen verbessern, liefern dagegen bessere Lösungen, benötigen dazu allerdings auch mehr Zeit. Diese Gruppe von Heuristiken läßt sich noch einmal dadurch unterteilen, daß manche terminieren, wenn sie keine Verbesserungsmöglichkeiten mehr finden, während andere beliebig lange laufen können und die gefundenden Näherungslösungen sich im allgemeinen immer mehr dem Optimum nähern. Wichtige Repräsentanten dieser Gruppen sollen hier vorgestellt werden. Vorher sollen aber noch Maßnahmen besprochen werden, die der Vorbereitung der Lösungssuche dienen.

3.1 Kandidatenmengen

Prinzipiell kann jedes Knotenpaar eines geometrischen Graphen in einer Hamilton-Tour benachbart sein. Mit bloßem Auge erkennt man jedoch, daß es in vielen Fällen unvernünftig wäre, zwei Knoten unmittelbar nacheinander zu besuchen. Man wüßte von vornherein, daß es, sucht man einen Rundweg durch Deutschland, keinen Sinn macht, von Hamburg direkt nach München zu reisen. Man versucht daher die Menge der zu betrachtenden Nachfolgeknoten einzuschränken. Bei geeigneter Wahl wird die Lösungsqualität nicht oder nur unwesentlich schlechter, die Laufzeit dafür bei einigen Verfahren erheblich geringer.

3.1.1 Nächste Nachbarn

In vielen Problemen genügt es, als Kandidaten für Folgeknoten nur eine bestimmte Anzahl an nächsten Nachbarn zu wählen.

Definition 3.1 (k-nearest-neighbour Teilgraph). Der *k-nearest-neighbour Teilgraph* N eines Graphen $G = (V, E)$ auf n Knoten ist definiert durch $N = (V, E_N)$ mit

$$E_N = \{uv \in E : c_{uv} \leq c_{uw_k}, \text{ wobei } c_{uw_1} \leq \ldots \leq c_{uw_n} \text{ für alle } uw_i \in E\}$$

□

Um diese Kandidatenmenge zu berechnen, ist im allgemeinen ein Aufwand von $O(n^2)$ zu leisten. Speziell für Graphen, die die Dreiecksungleichung erfüllen ($c_{uv}+c_{vw} \geq c_{uw} \; \forall u, v, w \in V$), kann jedoch mit Hilfe des *Delaunay Graphen* eine schnellere Variante implementiert werden (siehe unten). In Abbildung 3.1 sieht man eine solche Kandidatenmenge. Leider gibt es Fälle, in denen der Graph nicht mehr zusammenhängend ist. Dieser Fall tritt besonders dann auf, wenn der Graph aus mehreren Gruppen besteht, in denen die Knoten sehr dicht beieinander liegen, die aber selbst deutlich voneinander getrennt sind. Dann finden sich nämlich die nächsten Nachbarn eventuell ausschließlich unter den Knoten der gleichen Gruppe.

Für gleichmäßiger verteilte Knoten ist diese Wahl, die einen sehr dünn besetzten Graphen erzeugt und daher für gute Laufzeiten sorgt, jedoch sehr hilfreich. Auf der Probleminstanz *pr2392* findet sich eine optimale Lösung im 8-Nearest-Neighbour Teilgraph, auf der Instanz *pcb442* sogar im 6-Nearest-Neighbour Teilgraph [Rei 1994].

3.1.2 Delaunay Graph

Der Delaunay Graph basiert auf der Delaunay Triangulation, dem dualen Diagramm zum Voronoi-Diagramm. Für einen zweidimensionalen, geometrischen Graphen stellt das Voronoi-Diagramm eine Partitionierung der Ebene $\mathbb{R}^2$ in Voronoi-Regionen

$$\mathrm{VR}(v) := \{x \in \mathbb{R}^2 : d(x, v) \leq d(x, u) \; \forall u \in V, u \neq v\}$$

dar, wobei $d(x, y)$ eine metrische Funktion ist. In der Voronoi-Region eines Knotens $v \in V$ sind also alle Punkte der Ebene enthalten, die näher an v als an irgendeinem anderen Knoten des Graphen liegen. Allgemein lautet die Definition

Definition 3.2 (Voronoi-Diagramm). Sei $S = \{P_1, \ldots, P_N\}$ eine endliche Teilmenge aus $\mathbb{R}^m$ und $d : \mathbb{R}^m \times \mathbb{R}^m \longrightarrow \mathbb{R}$ eine Metrik. Eine *Voronoi Region* $\mathrm{VR}(P_i)$ eines Punktes $P_i \in S$ ist dann die Menge

$$\mathrm{VR}(P_i) = \{P \in \mathbb{R}^m : d(P, P_i) \leq d(P, P_j) \; \forall j = 1, \ldots, N, j \neq i\}.$$

Die Menge aller N Voronoi Regionen heißt *Voronoi Diagramm.* □

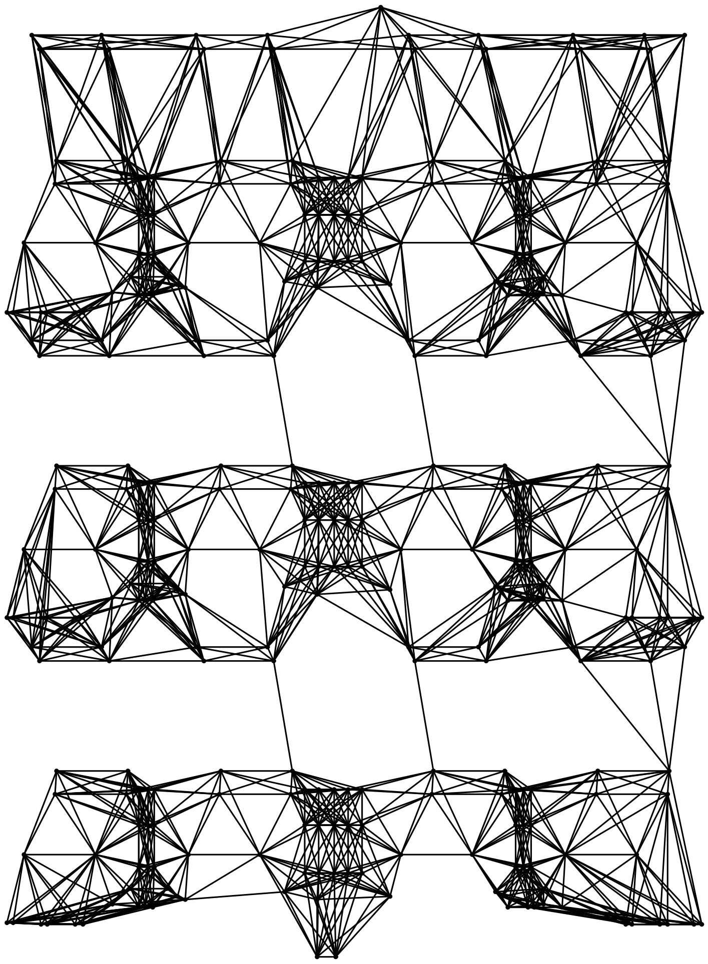

Abbildung 3.1: *10-Nearest-Neighbour Teilgraph auf Probleminstanz lin318*

Die Delaunay-Triangulation ergibt sich aus dem Voronoi-Diagramm, indem man alle Knoten miteinander verbindet, deren Voronoi-Regionen aneinander grenzen.

Definition 3.3 (Delaunay Triangulation). Gegeben sei das Voronoi Diagramm einer endlichen Teilmenge $S = \{P_1, \dots P_n\} \subset \mathbb{R}^m$. Dann heißt der ungerichtete Graph $G = (S, D)$ mit

$$D = \{\{P_1, P_2\} : \mathrm{VR}(P_1) \cap \mathrm{VR}(P_2) \neq \emptyset\}$$

Delaunay Triangulation. □

Abbildung 3.2 zeigt für eine Menge von acht Punkten in $\mathbb{R}^2$ das Voronoi-Diagramm und

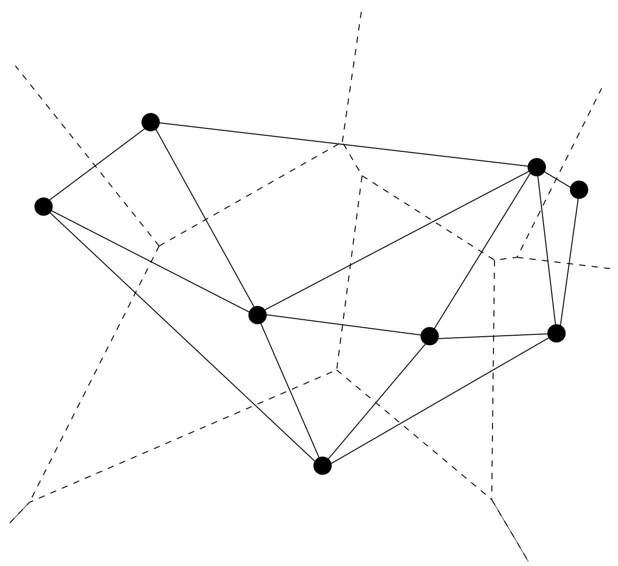

Abbildung 3.2: *Das Voronoi-Diagramm für eine Punktmenge aus acht Elementen in $\mathbb{R}^2$ (gestrichelte Linien) und die entsprechende Delaunay Triangulation.*

die Delaunay Triangulation.

Mit Hilfe der Delaunay Triangulation erhält man eine gute Kandidatenmenge, indem man außer den Kanten in D noch alle Kanten uw hinzufügt, für die es ein $v \in S$ gibt, so daß $uv \in D, vw \in D$. Diese Kandidatenmenge — sie soll *Delaunay Graph* heißen — bietet sich besonders für stark geclustertete Graphen an, also Graphen bei denen an verschiedenen Stellen Häufungen von Knoten auftreten. Bei einer Nearest-Neighbour Kandidatenmenge entsteht dann das Problem, daß Kanten nur innerhalb dieser Cluster auftreten und wichtige Verbindungen zwischen den Clustern fehlen. In Abbildung 3.3 sieht man eine solche Kandidatenmenge. Der Graph ist sehr dicht und — was noch wichtiger ist — er ist zusammenhängend. Es ist sehr wahrscheinlich, daß eine optimale Lösung für das ursprüngliche Problem in diesem Teilgraphen zu finden ist. Gute Näherungslösungen finden sich in jedem Fall. Um noch sicherer zu gehen, kann man die noch fehlenden Kanten aus einem Nearest-Neighbour Teilgraph hinzufügen. Eine Kombination aus Delaunay Graph und 10-Nearest-Neighbour Teilgraph zeigt Abbildung 3.4.

Bereits angesprochen worden ist die Möglichkeit, mit Hilfe der Delaunay Triangulation die Berechnung der k-Nearest-Neighbour Kandidatenmenge für geometrische Graphen zu beschleunigen. Dazu wird auf der Triangulation von jedem Knoten aus eine Breitensuche gestartet, die sich maximal k Schritte vom Startknoten entfernt. Es ist offensichtlich, daß sich alle k nächsten Knoten unter den dabei besuchten befinden. Zwar ist die worst-case Laufzeit immer noch $O(n^2)$, jedoch sind in der Praxis lineare Abhängigkeiten zu beobachten

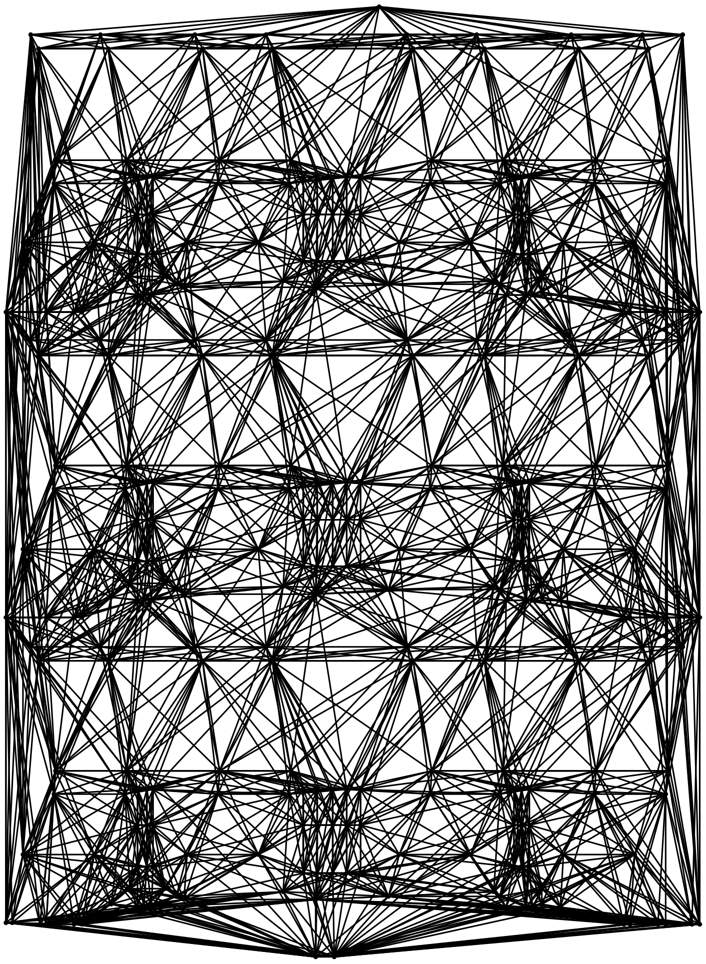

Abbildung 3.3: *Delaunay Graph auf der Probleminstanz lin318*

(Abb. 3.5). Dabei gibt es allerdings gewisse Abhängigkeiten von der Verteilung der Knoten in der Probleminstanz.

3.2 Konstruktive Heuristiken

Die Vorstellung der nun folgenden Verfahren dient nicht nur dazu, einen Überblick über die Forschung zu bekommen. Die Verfahren sollen später zum Vergleich mit der in dieser Arbeit vorgestellten Heuristik herangezogen werden. Damit dieser Vergleich möglichst genau

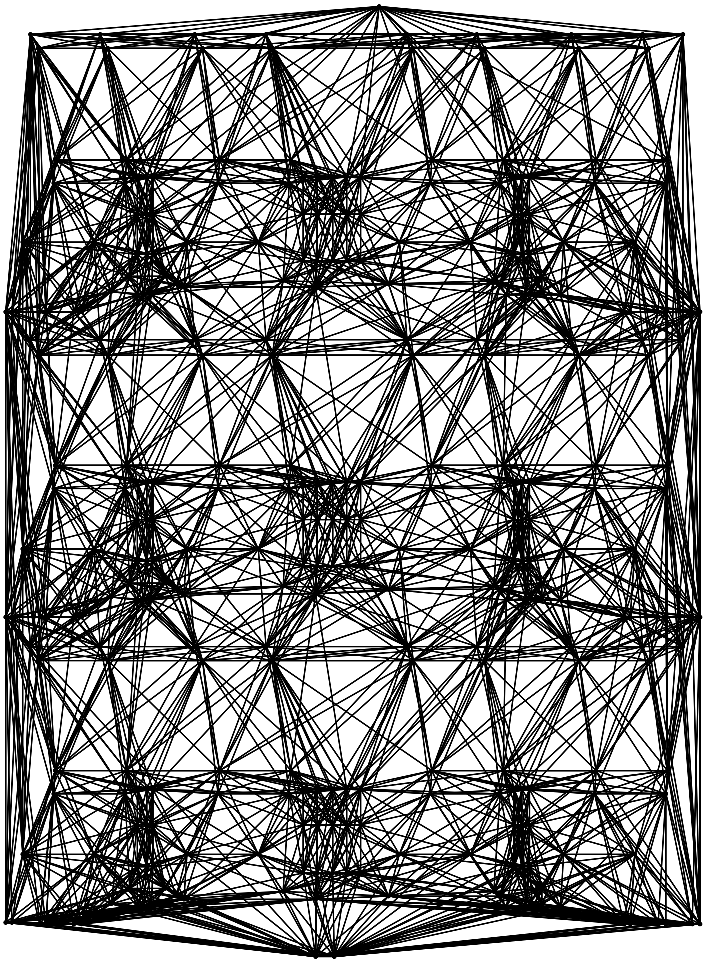

Abbildung 3.4: *Vereinigung des Delaunay Graphs und des 10-Nearest-Neighbour Teilgraphen auf lin318*

ist, wurden alle Verfahren aus [Rei 1994] entnommen. Alle Ergebnisse, insbesondere die Qualitäten der einzelnen Heuristiken, die in den folgenden Abschnitten angegeben werden, stammen aus dieser Quelle. Sie wurden unter identischen Bedingungen erzielt und können somit als Referenz herangezogen werden.

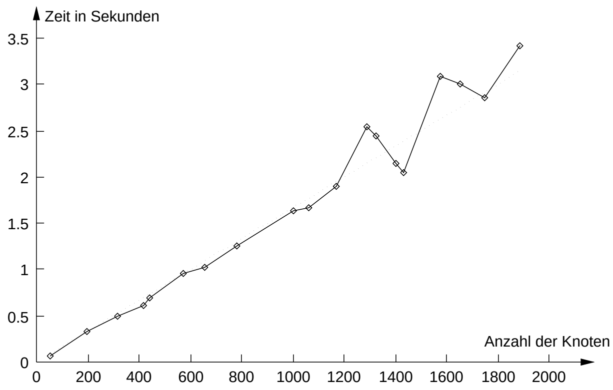

Abbildung 3.5: *Laufzeit für die Berechnung eines 10-Nearest-Neighbour Teilgraphen, wenn die Delaunay-Triangulation bekannt ist.*

3.2.1 Nearest-Neighbour Heuristik

Das wohl einfachste Verfahren zur Lösung des Traveling Salesman Problems ist die Nearest-Neighbour Heuristik. Hierbei wird mit einem beliebigen Knoten beginnend immer der jeweils nächste, noch nicht in der Tour befindliche Knoten an ein Ende der Tour angehängt.

Algorithmus: Nearest-Neighbour Heuristik

1. Wähle einen beliebigen Knoten j, setze $l = j$ und $T = \{1, \ldots, n\} \setminus \{j\}$.
2. Solange $T \neq \emptyset$:
   i. Bestimme den Knoten $j \in T$ mit $c_{lj} = \min\{c_{li} : i \in T\}$.
   ii. Hänge j an die Tour an und entferne j aus T.
   iii. Setze $l = j$.
3. Verbinde l mit dem Startknoten.

Der Aufwand für dieses Verfahren beträgt $O(n^2)$, da in jedem Schritt für den Endknoten des bisherigen Pfades die Distanz zu allen anderen Knoten berechnet werden muß. Nimmt man jedoch eine geeignete Kandidatenmenge zu Hilfe, so sind in der Praxis lineare Rechenzeiten zu beobachten.

Theoretisch kann bei diesem Verfahren ein beliebig schlechtes Ergebnis erzielt werden. Man kann nämlich beweisen, daß für jedes $r > 1$ eine TSP-Instanz existiert, für die die Nearest-Neighbour Heuristik eine Tour liefert, die mindestens r mal teurer als eine optimale Tour ist [Ros 1977]. Die schlechte Qualität der Touren ist leicht erklärbar, denn schon sehr früh werden die kurzen Kanten in die Tour eingefügt, ohne Rücksicht auf andere Knoten in der Nähe zu nehmen. Diese müssen dann später mit langen Kanten nachträglich eingefügt werden. Auf den Testinstanzen aus Tabelle 1.1 erzielt die beste Variante des Verfahrens eine durchschnittliche Qualität von etwas über 20. Dennoch wird die Nearest-Neighbour Methode häufig eingesetzt, besonders um gute Anfangslösungen für iterative Verfahren zu finden.

3.2.2 Insertion Heuristiken

Eine Insertion Heuristik beginnt mit einem Zyklus auf einem kleinen Teilgraphen, um dort nach und nach die anderen Knoten entsprechend festgelegter Kriterien einzufügen. Beispielsweise wähle man den Knoten

- dessen Abstand zu einem Tourknoten am geringsten ist.
- dessen minimaler Abstand zu einem Tourknoten maximal ist.
- dessen Abstand zu einem Tourknoten maximal ist.
- dessen maximaler Abstand zu einem Tourknoten minimal ist.
- dessen Aufnahme in die Tour die Kosten am wenigsten erhöht.
- zufällig.
- dessen Summe aller Abstände zu den Tourknoten maximal ist.
- dessen Summe aller Abstände zu den Tourknoten minimal ist.

Auch von diesen Kriterien gibt es noch Varianten, beispielsweise die unterschiedliche Wahl des Einfügepunktes. Laufzeit und Qualität der einzelnen Varianten sind sehr unterschiedlich. Bestenfalls erhält man einen Aufwand von $O(n^2 \log n)$ bei einer Qualität von etwa 15 oder Laufzeiten in der Größenordnung $O(n^2)$ mit einer Qualität von etwa 20.

Algorithmus: Insertion Heuristik

1. Wähle eine Anfangstour durch $k \geq 1$ Knoten $v_1, \dots, v_k$.
2. Setze $W = V \setminus \{v_1, \dots, v_k\}$.
3. Solange $W \neq \emptyset$:

i. Wähle einen Knoten j entsprechend einem bestimmten Kriterium.
ii. Füge diesen Knoten an eine Position in der Tour ein.
iii. Reduziere W um j.

Ähnlich wie bei der Nearest-Neighbour Heuristik lassen sich Kandidatenmengen heranziehen, um die Laufzeiten in der Praxis zu reduzieren. Anders als dort verschlechtert sich die Lösungsqualität dadurch aber erheblich. Der Einsatz einer konvexen Hülle als Anfangstour bei geometrischen Problemen ist erfolgreicher, da die Knoten der konvexen Hülle auch in einer optimalen Tour in der durch sie vorgegebenen Reihenfolge auftreten müssen (sonst gäbe es Kreuzungen in der Tour). Weil sich eine konvexe Hülle in $O(n \cdot \log n)$ berechnen läßt [Rei 1994], erhöht sich die Gesamtlaufzeit nur unwesentlich, der worst-case Aufwand gar nicht.

3.2.3 Doubletree-Heuristik

Erheblich schneller als die beiden bisher aufgeführten Methoden, sind Verfahren, die auf *Minimalen Spannbäumen* aufsetzen.

Definition 3.4 (Minimaler Spannbaum). Gegeben sei ein Graph $G = (V, E)$. Ein *Spannbaum* ist ein nicht-zyklischer, zusammenhängender Teilgraph $ST = (V, F)$ von G, $F \subset E$. Ein *Minimaler Spannbaum* ist ein Spannbaum $MST = (V, T)$ mit $T \subset E$ und $c(T)$ minimal unter allen Spannbäumen auf G. □

Ein solcher Spannbaum läßt sich in $O(n \cdot \log n)$ (geometrische Graphen) beziehungsweise $O(n^2)$ (allgemeine Graphen) berechnen [Rei 1994]. Man verdoppelt nun alle Kanten des Minimalen Spannbaums, damit jeder Knoten geraden Grad besitzt. Dann kann man sicher eine *Euler-Tour* finden, also eine Tour, bei der jede Kante des Graphen genau einmal besucht wird. Unter der Voraussetzung, daß der Graph die Dreiecksungleichung erfüllt, findet man eine Hamilton-Tour, die keinesfalls teurer ist, als die Eulersche Tour selbst. Dies geschieht, indem man an einem beliebigen Knoten startend die Eulersche Tour traversiert und jeden Knoten, der dabei noch nicht besucht wurde, an den so entstehenden Hamilton-Weg anhängt. Abbildung 3.6 illustriert das Verfahren.

Algorithmus: Doubletree Heuristik

1. Berechne einen Minimalen Spannbaum.
2. Verdopple alle Kanten dieses Spannbaumes.
3. Berechne eine Eulersche Tour auf diesem Graphen.

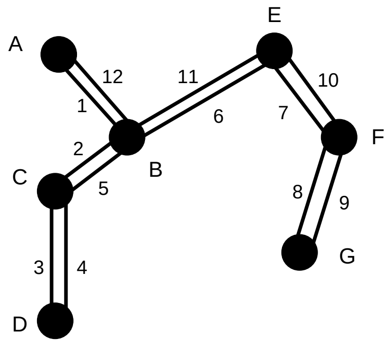

Abbildung 3.6: *Ein Minimaler Spannbaum, bei dem alle Kanten verdoppelt worden sind. Durchläuft man nun die Eulersche-Tour, die durch die Numerierung der Kanten gegeben ist, erhält man eine Hamilton-Tour A, B, C, D, E, F, G, A auf dem Graphen. Sie ist billiger als die Euler-Tour, da von D nach E und von G nach A „Abkürzungen“ genommen werden.*

4. Ermittle aus dieser Tour eine Hamilton-Tour.

Die Laufzeit des Algorithmus wird bestimmt durch die Berechnung des minimalen Spannbaumes, alle anderen Schritte benötigen nur linearen Aufwand. Daher beträgt der Gesamtaufwand $O(n^2)$ für allgemeine TSP-Instanzen und $O(n \cdot \log n)$ für euklidische Instanzen.

Die Laufzeiten für dieses Verfahren sind sehr gering, die Lösungsqualität jedoch auch. Sie liegt bei beinahe 40. Dennoch bietet sich das Verfahren an, um innerhalb kurzer Zeit eine Anfangslösung für iterative Verfahren zu finden. Da sich für Graphen, die die Dreiecksungleichung erfüllen, zudem beweisen läßt, daß mit diesem Verfahren ermittelte Touren maximal zweimal so lang sind, wie eine optimale Tour, bewegt man sich mit diesem Verfahren auch auf der sicheren Seite [Rei 1994].

3.2.4 Verfahren von Christofides

Ein weiteres Verfahren basierend auf Minimalen Spannbäumen wurde 1976 von N. Christofides vorgeschlagen [Chr 1976]. Dabei werden nicht alle Baumkanten verdoppelt, sondern auf den Knoten mit ungeradem Grad ein *Minimum Weight Perfect Matching* ermittelt. Auch dadurch haben alle Knoten geraden Grad, und man kann eine Eulersche Tour bestimmen.

Definition 3.5 (Perfect Matching). Ein *Perfect Matching* auf einem Graphen $G = (V, E)$ mit $|V| = 2n$ ist eine Menge $M \subset E$ von n Kanten, so daß jeder Knoten genau mit einer Kante dieser Menge inzidiert. Ein *Minimum Weight Perfect Matching* ist ein Perfect Matching mit $c(M)$ minimal. □

Die Berechnung des Matchings kann in $O(n^3)$ geschehen [Edm 1965]. Das Verfahren von Christofides hat damit kubischen Aufwand, da in einem Spannbaum alle Knoten ungeraden Grad haben können. Daher sucht man nicht immer das Minimum aller Perfect Matchings, sondern gibt sich mit einer guten Näherung zufrieden. Der Aufwand zur Suche des Matchings und damit auch der Gesamtaufwand reduziert sich dadurch auf $O(n^2)$. In der Praxis ist die Laufzeit abhängig vom konkreten Problem, da die Anzahl der ungeraden Knoten stark variiert. Christofides konnte zwar beweisen, daß bei Graphen, die die Dreiecksungleichung erfüllen, die Kosten einer durch dieses Verfahren gewonnenen Tour niemals größer als das eineinhalbfache einer optimalen Tour sind, doch ist die durchschnittlich erzielte Qualität auf den Testinstanzen mit knapp 20 noch recht niedrig.

3.2.5 Savings Heuristik

Eine weitere Möglichkeit, gute Näherungslösungen für das TSP zu finden, ist das Verbinden mehrerer Touren zu einer einzigen Tour. Dieses Vorgehen findet man bei der Savings Heuristik.

Algorithmus: Savings Heuristik

1. Wähle einen Basisknoten $b \in V$ und lege $n-1$ Touren $(b, v)\ \forall v \in V \setminus \{b\}$ fest.
2. Solange noch mehr als eine Tour existiert:
   i. Für jedes Tourenpaar T_1, T_2 berechne die Reduktion der Gesamtkosten, falls die Touren verbunden werden, indem je eine Kante zum Basisknoten entfernt wird und die so entstandenen offenen Enden der Touren verbunden werden (siehe Abb. 3.7).
   ii. Verbinde die beiden Touren, bei denen die Kostenreduktion maximal ist.

Ursprünglich wurde dieses Verfahren für das *Vehicle Routing Problem* entwickelt. Wie aber bereits in Abschnitt 1.1 angesprochen, ist das TSP nur ein Spezialfall. Es kann daher mit dem Verfahren gelöst werden.

Der Aufwand für das Verfahren beträgt im worst case $O(n^2 \log n)$. Damit ist es den anderen Verfahren in der Geschwindigkeit unterlegen. Geeignete Datenstrukturen, die es erlauben, nur die Information neu zu berechnen, die sich auch tatsächlich nach der Verbindung zweier

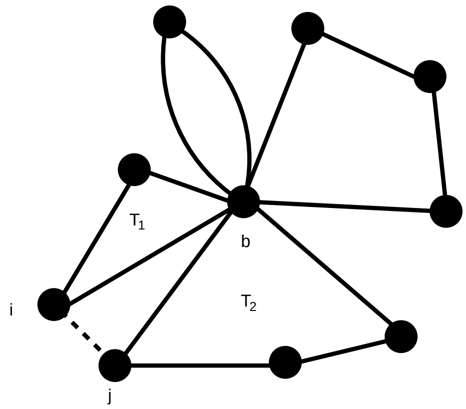

Abbildung 3.7: *Savings Heuristik: Touren T_1 und T_2 werden an den Knoten i, j verbunden, die Kostenersparnis beträgt $c_{ib} + c_{jb} - c_{ij}$.*

Touren geändert hat, halten die durch die O-Notation versteckten Konstanten jedoch so klein, daß dennoch gute Laufzeiten erzielt werden.

Die gelieferten Lösungen sind von guter Qualität. Im Durchschnitt liegen sie nur etwa 11,5% über den Kosten einer optimalen Lösung.

3.2.6 Greedy Algorithmus

Der Greedy Algorithmus stellt eine allgemeinere Form der Nearest-Neighbour Heuristik dar. Während bei letzterer immer nur der nächste Knoten *eines* bestimmten Knotens betrachtet wurde, wird nun der nächste Knoten *jedes* anderen Knoten gesucht.

Algorithmus: Greedy

1. Sortiere $E_n = \{e_1, \ldots, e_m\}$, so daß $c_{e_1} \leq c_{e_2} \leq \ldots \leq c_{e_m}$.
2. Setze $T = \emptyset$.
3. Für alle Kanten $e \in E$:
   i. Wenn $T \cup \{e\}$ zu einer Hamilton-Tour ausgebaut werden kann (oder bereits eine Hamilton-Tour ist), dann setze $T = T \cup \{e\}$.

Der Test, ob sich $T' = T \cup \{e\}$ zu einer Hamilton-Tour ausbauen läßt, ist bei geeignet gewählten Datenstrukturen in konstanter Zeit machbar. Damit wird die Laufzeit des Verfahrens durch das Sortieren der Kanten bestimmt, beträgt also $O(m \cdot \log m)$. Die Qualität der Lösungen dieses Verfahrens ist nur wenig schlechter als die der Savings Heuristik.

3.3 Iterative Verfahren

Neben den konstruktiven Verfahren, deren Ziel es ist, eine gute Lösung zu erzeugen, gibt es iterative Verfahren, die eine bereits vorhandene Lösung verbessern. Die wichtigsten und bekanntesten Vertreter sollen hier vorgestellt werden.

3.3.1 Node und Edge Insertion

Bei der Node beziehungsweise Edge Insertion Heuristik werden immer wieder Knoten (Kanten) aus einer bestehenden Tour entfernt und an einer günstigeren Stelle eingefügt. Dies geschieht, bis dadurch keine Verbesserung der Lösung mehr gefunden werden kann.

Algorithmus: Node (Edge) Insertion

1. Gegeben sei eine Tour T.
2. Wiederhole:
   i. Prüfe für jeden Knoten v (die Kante von Knoten v zu seinem Nachfolger) ob das Einfügen dieses Knotens (dieser Kante) an einer anderen Stelle in der Tour die Kosten reduziert.
   ii. Gibt es mindestens einen solchen Knoten (eine solche Kante), so nehme den besten (die beste) von diesen und füge ihn (sie) an der entsprechenden Stelle ein.

   bis keine Kostenreduktion mehr möglich ist.

Abbildung 3.8 illustriert die Vorgehensweise. Das Verfahren ist im worst-case nicht polynomiell, denn in diesem Fall würde jeder einzelne Schritt die Kosten der Tour nur um eine Einheit verringern, weshalb die Laufzeit abhängig von den Tourkosten ist.

Ein Nachteil dieses Verfahrens ist, daß die Lösungsqualität in erheblichen Maße davon abhängt, wie gut die Anfangslösung ist. Um gute Ergebnisse zu erzielen, muß man also

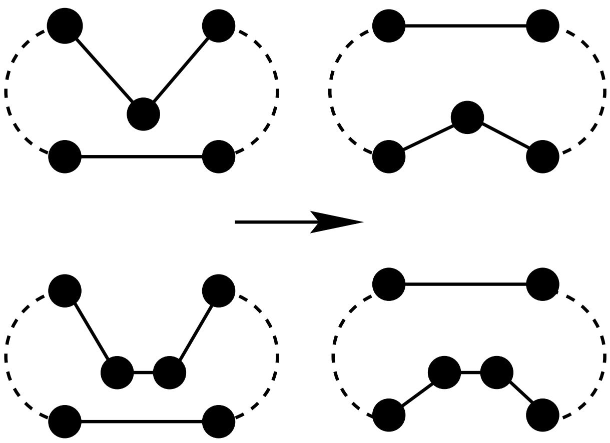

Abbildung 3.8: *Ein Schritt beim Node (oben) und Edge (unten) Insertion Verfahren.*

schon viel Aufwand in die Bestimmung der initialen Tour legen. Bei einer zufälligen Anfangslösung werden beispielsweise Qualitäten von etwa 90 erzielt. Eine Lösung, die von einer Nearest-Neighbour Heuristik geliefert wurde, wird auf etwa 17 verbessert. Lösungen des Savings-Verfahrens werden auf eine Qualität von etwa 9 gesteigert.

3.3.2 2-Opt Verfahren

Man kann sich für geometrische Graphen leicht überlegen, daß eine Tour, die Kreuzungen von Kanten enthält, nicht optimal sein kann. Kreuzen sich beispielsweise wie in Abbildung 3.9 (links) zwei Kanten ab und cd, so ist es besser, statt dessen die Kanten ac und bd in die Tour aufzunehmen. Auf der rechten Seite der Abbildung sieht man zwei Kanten, die sich nicht kreuzen, bei denen aber der gleiche Schritt auch zu einer Verbesserung der Lösung führt. Ein *2-Opt* Schritt besteht daher aus dem Entfernen zweier Kanten und dem erneuten Einfügen dieser Kanten auf die einzig andere Weise.

Algorithmus: 2-Opt Verfahren

1. Gegeben sei eine Tour T.
2. Wiederhole:
   i. Prüfe für jeden Knoten v alle möglichen 2-Opt Schritte, an denen die Kante zwischen v und seinem Nachfolger beteiligt ist.

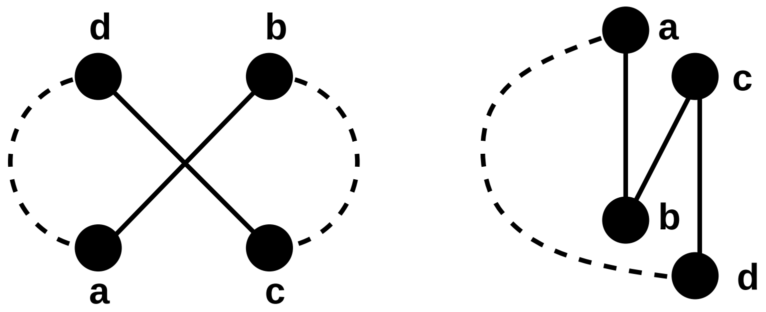

Abbildung 3.9: *Zwei Situationen für die Anwendung eines 2-Opt Schrittes.*

ii. Gibt es mindestens einen 2-Opt Schritt, der die Tourkosten reduziert, so führe den besten unter ihnen aus.

bis kein verbessernder 2-Opt Schritt gefunden wird.

Auch dieses Verfahren ist aus den gleichen Gründen wie oben nicht polynomiell. Die Lösungsqualität dagegen ist kaum noch von der Qualität der initialen Lösung abhängig, so daß ein einfaches Verfahren zur Erzeugung der Startlösung genügt. Durchschnittliche Werte liegen bei etwa $8,5\%$ über dem Optimum.

Das Verfahren kann durch Einsatz einer Kandidatenmenge beschleunigt werden, doch sind dabei deutliche Qualitätseinbußen zu registrieren. Eine Kombination mit dem Node Insertion Verfahren dagegen kann die Lösungsqualität noch einmal steigern.

Weitere Verbesserungen der Lösungsqualität erhält man durch k-Opt Schritte. Bei diesen wird die Tour nicht in zwei, sondern in k Teile aufgesplittet und anschließend neu zusammengesetzt. Der zeitliche Aufwand für dieses Verfahren ist erheblich, da es alleine einen Aufwand von $O(n^k)$ erfordert, den besten k-Opt Schritt zu finden. Zudem gibt es nicht — wie im 2-Opt Verfahren — nur eine Möglichkeit, die einzelnen Teile des Graphen neu zusammenzufügen.

3.3.3 Lin-Kernighan-Heuristiken

Wie im letzten Abschnitt angedeutet, ermöglichen komplexere Schritte im allgemeinen bessere Lösungen. Außerdem bleiben die Verfahren mit einfachen Schritten schnell in lokalen Optima des Lösungsraumes stecken.

Lin und Kernighan [Lin 1973] schlugen 1973 vor, jeden Verbesserungsschritt aus kleinen Teilschritten zusammenzusetzen, die jeder für sich nicht unbedingt die Tour verkürzen müssen. Seitdem sind viele Varianten dieser Heuristik veröffentlicht worden [Mak 1993].

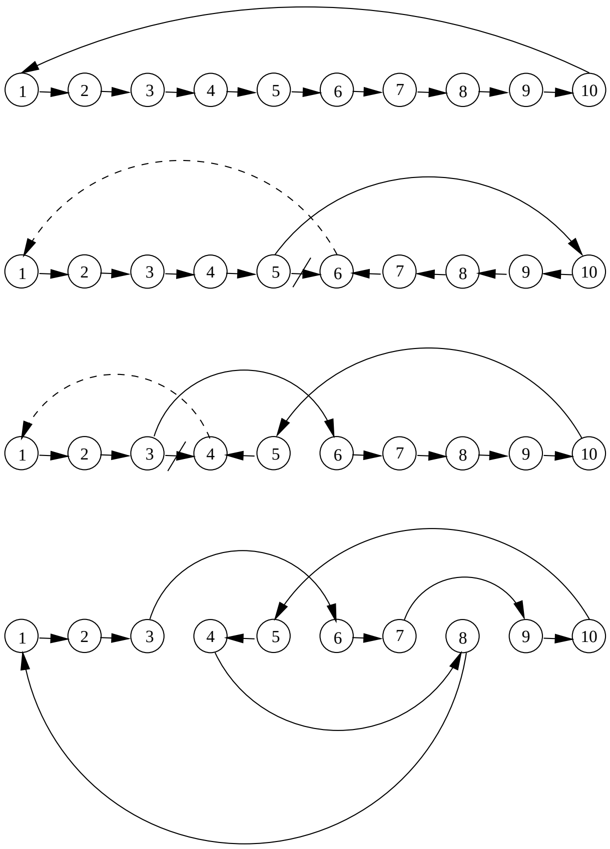

Abbildung 3.10: *Illustration eines Lin-Kernighan Schrittes.*

Auch hier soll jedoch die Variante aus [Rei 1994] vorgestellt werden.

In Abbildung 3.10 ist ein Lin-Kernighan Schritt dieser Variante dargestellt. Die initiale Tour sei $(1, 2, \ldots, 10, 1)$. Nun wird durch einen 2-Opt Schritt die Tour in die beiden Wege $(1, \ldots, 5)$ und $(6, \ldots, 10)$ aufgetrennt, die wiederum durch die Kante $(5, 10)$ verbunden werden. Eine Kante $(6, 1)$ würde den entstehenden Weg wieder zu einer Tour werden lassen, vorausgesetzt, die Richtungen in allen Teilpfaden sind richtig gesetzt. Statt dessen wird diese fiktive Tour jedoch noch einmal durch einen 2-Opt Schritt modifiziert. Die Kante $(3, 4)$ wird entfernt und dafür die Kante $(3, 6)$ eingefügt. Die Kante $(4, 1)$ hätte wieder eine zulässige Lösung zur Folge. Doch auch diese Tour bleibt fiktiv; der Knoten 8 wird durch einen Node Insertion Schritt zwischen die Knoten 1 und 4 eingesetzt.

In der vorgestellten Variante sind alle Teilschritte entweder 2-Opt Schritte oder Node Insertions. Sie werden nach einem bestimmten Kriterium ausgewählt, das die größte Verbesserung verspricht (beziehungsweise die geringste Verschlechterung bedeutet). Außerdem wurde die Zahl der Teilschritte in einem Lin-Kernighan Schritt begrenzt und eine Kandidatenmenge eingesetzt.

Offensichtlich gibt es eine Menge von Möglichkeiten, Parameter zu wählen. Die Qualität der Ergebnisse liegt je nach Wahl der Parameter zwischen 2 und 3. Die besten Ergebnisse wurden mit Anfangstouren erzielt, die durch die Heuristik von Christofides errechnet wurden. Die Abweichungen bei anderen Anfangstouren sind jedoch gering, sowohl die Laufzeit als auch die Qualität der Lösungen betreffend.

3.3.4 Simulated Annealing, Genetische Algorithmen

Neben den vorgestellten, „klassischen“ Verfahren existieren seit geraumer Zeit neuere Ansätze zur Bewältigung des TSP. Diese entstanden zum Teil aus der Beobachtung der Natur.

Eines dieser Verfahren ist das *Simulated Annealing*. Bei diesem Verfahren wird der physikalische Vorgang bei der Gerinnung von Flüssigkeiten simuliert. Man kann dort beobachten, daß eine Substanz besonders gleichmäßig geordnete Kristalle bildet, wenn ihre Temperatur sehr langsam gesenkt wird, also die Schwingungsfrequenz der Moleküle langsam reduziert wird. Beim Traveling Salesman Problem wird analog zunächst eine relativ schlechte Ausgangstour gewählt. Diese wird zufällig geändert. Eine solche Änderung wird akzeptiert, wenn sich entweder die Tour dadurch verbessert oder nur um einen bestimmten Prozentsatz verschlechtert. Dieser Prozentsatz entspricht quasi der Temperatur beim physikalischen Versuch. Während also zu Beginn des Verfahrens auch deutliche Verschlechterungen akzeptiert werden, werden am Ende beinahe nur noch Verbesserungen angenommen. Diese Schritte werden wiederholt, bis ein festgelegtes Stop-Kriterium erfüllt ist.

Die Ergebnisse dieses Verfahrens sind umso besser, je länger das Verfahren läuft. Besonders entscheidend dabei ist die Änderung der „Temperatur“. Ein zu schnelles Abkühlen beschränkt die Suche der Lösung auf eine kleine Teilmenge des Lösungsraumes, wodurch die Gefahr wächst, daß nur ein relativ schlechtes lokales Optimum gefunden wird.

Ebenfalls der Natur abgeschaut sind *Genetische Algorithmen*. Hier erzeugt man zufällig eine Menge von Lösungen („Population“), aus deren Elementen („Individuen“) durch Modifikation („Mutation“) und Kombination („Kreuzung“) neue Individuen geschaffen werden. Aus der alten Population und deren Nachfahren werden entsprechend bestimmter Kriterien Individuen ausgewählt, die die Population im nächsten Iterationsschritt („Generation“) darstellen.

Die Qualität der Lösungen dieses Verfahrens wird von der Leistungsfähigkeit der Mutations- und Kreuzungs-Operatoren, sowie der Art der Selektion bestimmt. Es ist bei geeigneter Wahl meistens möglich, optimale Lösungen zu finden. Dafür werden allerdings dann auch sehr große Laufzeiten benötigt. Ein Einsatz an Stelle der in den Abschnitten 3.2 und 3.3 vorgestellten Methoden bietet sich nicht an, da für die Suche vergleichbar guter Näherungslösungen mehr Zeit nötig ist.

Kapitel 4

Eine Heuristik mit iteriertem Matching

In dieser Arbeit wird eine neue Heuristik für das TSP besprochen. Sie basiert auf *Iteriertem Matching*, einem Verfahren, das bereits in der Verschnittoptimierung [Fri 1994] gute Ergebnisse geliefert hat.

Als erstes soll eine grobe Übersicht über den gewählten Algorithmus gegeben werden. Die einzelnen Schritte dieses Algorithmus werden dann in den folgenden Abschnitten detaillierter besprochen. Zunächst müssen allerdings wichtige Begriffe eingeführt und erläutert werden.

4.1 Begriffsdefinitionen

Ein *Matching* auf einem Graphen ist eine Verbindung je zweier seiner Knoten zu einem Paar. Welche Knoten dabei gepaart werden, hängt von verschiedenen Kriterien ab. Während das *maximale Matching* das Ziel hat, möglichst viele Knoten zu paaren, wird bei einem *Maximum Weight Matching* die Summe aller Kantengewichte der Knotenpaare maximiert. Eine weitere Variante ist das *Minimum Weight Perfect Matching*, das bereits in 3.2.4 vorgestellt wurde.

Definition 4.1 (Matching). Ein *Matching* auf einem Graphen $G = (V, E)$ ist eine Menge von Kanten M, so daß jeder Knoten mit maximal einer Kante aus M inzidiert. $\square$

Definition 4.2 (Maximales Matching). Ein *Maximales Matching* ist ein Matching M auf einem Graphen $G = (V, E)$ mit $|M|$ maximal. $\square$

Definition 4.3 (Maximum Weight Matching). Ein *Maximum Weight Matching* ist ein Matching M mit $c(M)$ maximal. $\square$

Abbildung 4.1: *Maximales (links) und Maximum Weight Matching (rechts).*

Einen Vergleich zwischen einem maximalen Matching und einem Maximum Weight Matching zeigt Abbildung 4.1.

Zur Berechnung des Maximum Weight Matchings wurde in dieser Arbeit ein Programm von Ed Rothberg verwandt. Es basiert auf dem 1973 von H. Gabow vorgestelltem „N-cubed weighted matching“ [Gab 1973]. Die Laufzeit des Programms ist im worst case kubisch abhängig von der Anzahl der Knoten.

Eine Verbindung aus zwei Knoten wird von nun an *Cluster* genannt. Beim *iterierten Matching* wird auf dem zugrundeliegenden Graphen ein Matching durchgeführt, mit den entstandenen Clustern ein neuer *Clustergraph* aufgebaut und das Verfahren wiederholt. Es werden so viele Iterationen durchgeführt, bis die Anzahl an Clustern einen vorher festgelegten Wert erreicht.

Definition 4.4 (Cluster). Gegeben sei ein Graph $G = (V, E)$, $G' = (V', E')$ sei ein Teilgraph von G. Außerdem sei $H_k = \{T_1, \ldots, T_k\}$ eine Menge verschiedener Hamilton-Touren auf dem Graphen G'. Dann heißt das Tupel $C = (V', H_k)$ *Cluster.* □

Definition 4.5 (Clustergraph). Gegeben sei ein Graph $G = (V, E)$. Sei $\Gamma = \{C_1, \ldots, C_n\}, C_i = (V_i, H^i_{k_i})$ eine Menge von Clustern aus G und

$$\Psi = \{(C_i, C_j), i \neq j : \exists (v_1, v_2) \in H^i_\alpha \text{ und } (u_1, u_2) \in H^j_\beta \text{ mit } (v_1, u_1) \in E \text{ und } (u_2, v_2) \in E\}$$

eine Menge von Kanten zwischen den Clustern. Dann heißt der Graph (Γ, Ψ) *Clustergraph.* □

Die Begriffe „Cluster“ und „Knoten“ sollen im folgenden synonym verwandt werden. Dieses macht Sinn, da ein Cluster nur ein verallgemeinerter Knoten ist (das Tupel $(v, (v))$ mit einem Knoten und der einzigen Hamilton-Tour darauf ist nach Definition 4.4 ein Cluster) und ein Clustergraph dementsprechend ein Graph auf verallgemeinerten Knoten. Ein Knoten in der ursprünglichen Definition wird von nun an als *Elementarknoten* bezeichnet. Bezieht sich eine Aussage ausschließlich auf ein Cluster mit mindestens zwei Elementarknoten, wird von einem echten Cluster gesprochen.

In der Definiton 4.5 des Clustergraphen wurde nicht darauf eingegangen, wie die Kanten zwischen zwei Clustern gewichtet werden. Dort wurde nur gesagt, daß eine Kante zwei Cluster verbindet, wenn es zwischen je einem Elementarknoten der beiden Cluster eine Kante in der Menge E des Graphen G gibt. Die Kantengewichte sollen dagegen die spezielle Struktur der echten Cluster mit ihren Hamilton-Touren berücksichtigen. Sie werden daher durch eine *Benefitfunktion* bestimmt, die genau das leistet. Die Kantengewichte zwischen zwei Clustern werden daher auch *Benefit* genannt.

Definition 4.6 (Benefit, Benefitfunktion). Seien C_1, C_2 zwei Cluster eines Graphen $G = (V, E)$ und $e = (C_1, C_2)$ eine Kante zwischen diesen Clustern. Das Gewicht dieser Kante wird bestimmt durch eine *Benefitfunktion* $f(C_1, C_2, E)$, die abhängig von den Clustern und der zugrundeliegenden Kantenmenge ist. Man nennt das Gewicht dieser Kante auch *Benefit*.

4.2 Skizze des Lösungsalgorithmus

In dieser Arbeit wird das iterierte Matching auf das TSP angewandt. Im ersten Schritt werden dazu Elementarknoten mit Hilfe eines Maximum Weight Matchings gepaart und miteinander verbunden. Die entstandenen Cluster stellen gemeinsam mit den zwischen den Clustern existierenden Kanten ein reduziertes Problem dar, auf das das gleiche Verfahren angewandt werden kann. Sind nur noch zwei Cluster übrig, stellen die auf diese Weise ausgewählten Kanten im einfachsten Fall zwei Hamilton-Wege auf den Elementarknoten der Cluster dar. Werden diese beiden optimal verbunden, hat man eine Hamilton-Tour im Ursprungsgraphen.

Algorithmus: Iteriertes Matching

1. Wiederhole:

   i. Finde ein Maximum Weight Matching auf Graphen.

   ii. Verknüpfe gepaarte Knoten zu Clustern.

   bis nur noch zwei Cluster übrig sind.

2. Verbinde die letzten beiden Cluster optimal.

Schon angesprochen wurde, daß die Kantengewichte zwischen den Clustern durch eine Benefitfunktion bestimmt werden. Die Wahl dieser Funktion hat entscheidenden Einfluß auf das Gesamtergebnis. Sie muß außerdem für die Umwandlung billiger Kanten in große Gewichte sorgen (da das Matching ja Kanten mit besonders großen Gewichten bevorzugt).

Ist an einer Paarung zweier Cluster mindestens ein echtes beteiligt, gibt es mehrere Hamilton-Touren auf der Vereinigung der beiden Elementarknotenmengen. Nur die beste zu speichern, ist nicht sinnvoll, denn dadurch werden vorzeitig gute Lösungsalternativen ausgeschlossen, wie in Abbildung 4.2 zu sehen ist. Das Problem besteht darin, geeignete Touren zu konstruieren und auszuwählen.

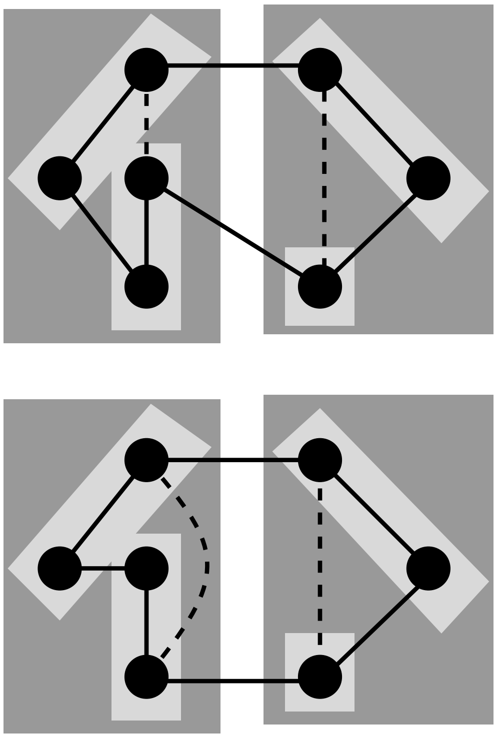

Abbildung 4.2: *Paarung zweier Cluster. Oben wurden nur die lokal besten Touren gespeichert, unten auch teurere.*

Ein weiteres Problem beim Einsatz des iterierten Matchings ist die Laufzeit des Matchings. Ein Maximum Weight Matching ist bestenfalls in $O(n^3)$ machbar [Edm 1965]. Eine Reduktion der zu untersuchenden Kanten sorgt jedoch für eine merkliche Senkung der durch die O-Notation verborgenen Konstanten. Hier macht sich der Einsatz einer Kandidatenmenge bezahlt.

In dünn besetzten Graphen besteht allerdings ein besonderes Problem. Hier existiert nicht immer eine Hamilton-Tour. Selbst wenn man voraussetzt, daß eine solche existiert, so kann es immer noch Kanten geben, die in keiner Hamilton-Tour vorkommen. Enthält ein Matching eine solche Kante, so werden Zyklen konstruiert, die nicht zu einer zulässigen Lösung führen. Diese Kanten müssen also von vornherein ausgeschlossen werden. Die gestrichelte Kante in Abbildung 4.3 stellt eine solche Kante dar.

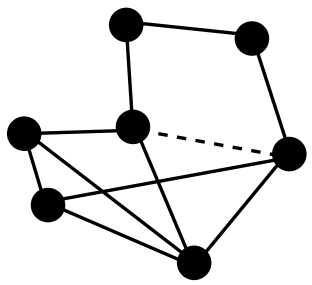

Abbildung 4.3: *„Verbotene“ Kante bei der Suche nach einer Hamilton-Tour.*

4.3 Darstellung einer Lösung

Der wichtigste Punkt ist aber wohl die Darstellung einer Lösung an sich. Das Verfahren drängt die (für das TSP untypische) Darstellung als Baum auf, denn in jedem Schritt werden je zwei „kleine“ Cluster zu einem „größeren“ zusammengefaßt. In jedem Cluster

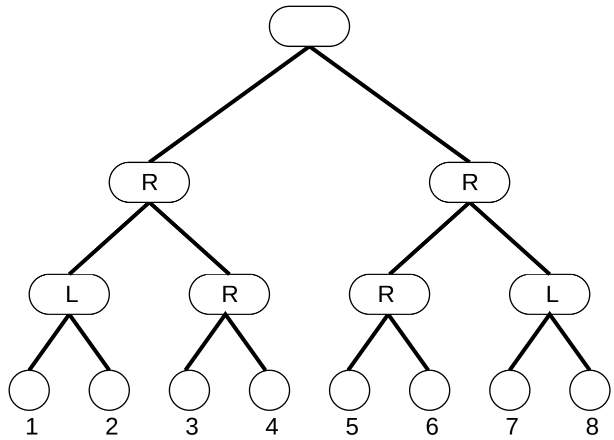

Abbildung 4.4: *Darstellung der Lösung als Baum. Die abgebildete Lösung lautet* $4, 3, 1, 2, 7, 8, 6, 5$.

muß also gespeichert werden, aus welchen Knoten es entstanden ist. Außerdem müssen

Informationen darüber vorhanden sein, wie die Elementarknoten der Cluster durchlaufen werden. Im einfachsten Fall wird in jedem Cluster nur eine Tour gespeichert. Eine neue Tour entsteht dann dadurch, daß beide Teiltouren aus den Söhnen dieses Clusters im Lösungsbaum nacheinander durchlaufen werden, wobei festgelegt werden kann, welcher von beiden als erstes genommen wird (siehe Abb. 4.4).

4.4 Speichern mehrerer Wege

Durch Abbildung 4.2 wurde klar, daß es nicht sinnvoll ist, bei der Verbindung zweier Cluster nur die lokal beste Hamilton-Tour zu speichern. Dieses würde die Qualität der Näherungslösung am Ende erheblich einschränken.

Es ist aber nicht möglich, alle Touren zu berechnen, die auf den Elementarknoten eines Clusters entstehen können. Das hieße, alle Permutationen auf den Elementarknoten zu bestimmen, und wäre mit der Suche nach der optimalen Lösung identisch. Ziel ist es aber, mit vergleichbar geringem Aufwand eine Menge guter (*suboptimaler*) Touren zu finden. Das kann geschehen, indem bei der Paarung zweier Cluster neue Touren dadurch konstruiert werden, daß aus jedem Cluster eine Tour hergenommen wird, an einer Stelle aufgebrochen und mit der anderen Tour verbunden wird. Dabei besteht in symmetrischen Graphen die Möglichkeit, eine Tour sowohl vorwärts als auch rückwärts zu durchlaufen. Eine Tour aufzubrechen bedeutet, daß eine Kante aus der Tour entfernt wird. Man erhält dadurch einen Hamilton-Weg. Im folgenden soll kein Unterschied zwischen den Begriffen „Weg“ und „Tour“ gemacht werden.

Speicherplatzbedarf und Laufzeitverhalten sprechen dagegen, die Touren als Elementarknotenfolge zu speichern. Eine Möglichkeit, die neu entstandenen Zyklen eindeutig zu beschreiben, ist das Zurückgreifen auf bereits vorhandene Informationen in den Söhnen eines Clusters. Seien also C_1 und C_2 zu paarende Cluster, $v_1, \dots, v_p$ die Elementarknoten in C_1 und $u_1, \dots, u_q$ die Elementarknoten in C_2. Weiter bezeichne $v_{i,j}$ die Folge $(v_i v_{i+1} \dots v_j)$ und $\tilde{v}_{i,j}$ die Folge $(v_i v_{i-1} \dots v_j)$ (entsprechend für u), wobei $v_{i,j} = ()$, wenn $i > j$ und $\tilde{v}_{i,j} = ()$, wenn $i < j$. Die Menge der möglichen Touren für die Kombination der Wege $v_{1,p}$ aus C_1 und $u_{1,q}$ aus C_2 lautet dann

$$W = \{(v_{1,i} u_{j,q} u_{1,j-1} v_{i+1,p}), (v_{1,i} \tilde{u}_{j,1} \tilde{u}_{q,j+1} v_{i+1,p}) \ \forall i = 1, \dots, p, j = 1, \dots, q\}.$$

Tabelle 4.1 zeigt ein erläuterndes Beispiel.

Jede Tour in einem Cluster muß eindeutig beschrieben werden. Sei nun ein Cluster $C = (V, H_k)$ aus den beiden Clustern $C_1 = (V_1, H^1_{k_1})$ und $C_2 = (V_2, H^2_{k_2})$ entstanden. Die Tour $T \in H_k$ bestehe dabei aus den Touren $T_\alpha = v_{1,p} \in H^1_{k_1}$ und $T_\beta = u_{1,q} \in H^2_{k_2}$. Die Menge W sei die Menge der möglichen Tourkombinationen aus T_α und T_β. Die Tour T ist dann eindeutig beschrieben durch die Indizes α und β der Touren aus $H^1_{k_1}$ und $H^2_{k_2}$,

i	j	neue Touren	
1	1	(1,2,9,7,3 5,4,8)	(1,2,7,9,3,5,4,8)
1	2	(1,9,7,2,3,5,4,8)	(1,9,2,7,3,5,4,8)
3	1	(1,3,5,2,9,7,4,8)	(1,3,5,2,7,9,4,8)
3	3	(1,3,5,7,2,9,4,8)	(1,3,5,7,9,2,4,8)
4	2	(1,3,5,4,9,7,2,8)	(1,3,5,4,9,2,7,8)
5	1	(1,3,5,4,8,2,9,7)	(1,3,5,4,8,2,9,7)

Tabelle 4.1: *Die Tabelle zeigt verschiedene Möglichkeiten, zwei Touren $v_{1,5} = (1, 3, 5, 4, 8)$ und $u_{1,4} = (2, 9, 7)$ zu kombinieren. In der dritten Spalte steht jeweils die Tour, die entsteht, wenn man die Tour $u_{1,4}$ vorwärts durchläuft, in der vierten Spalten wird $u_{1,4}$ rückwärts durchlaufen.*

den Werten $i = 1, \ldots, p$ und $j = 1, \ldots, q$, die das Element aus W beschreiben, sowie — in symmetrischen Graphen — die Richtung der Tour T_β. Der Elementarknoten v_i heißt dann *Austrittspunkt* aus der Tour T_α, der Elementarknoten u_j heißt *Eintrittspunkt* in die Tour T_β. Da es sich bei T um einen Zyklus handelt, ist die Festlegung von v_1 als „Anfangsknoten“ keine Einschränkung. Schließlich ist der Zyklus $(v_{a,p}, v_{1,a-1}, v_a), a > 1$ mit dem Zyklus $(v_{1,p}, v_1)$ identisch. Auch die Tatsache, daß nur bei dem rechten Sohn eine Richtung vorgegeben werden kann ist in symmetrischen Graphen keine Einschränkung, denn für jeden Zyklus $(v_{1,p}, v_1)$ existiert eine Darstellung $(\tilde{v}_{p,1}, v_p)$, die den gleichen Zyklus beschreibt.

Bei der Berechnung der Menge W ist zu beachten, daß es Sonderfälle gibt, in denen zwei verschiedene Beschreibungen der vorgestellten Form das gleiche Element darstellen. Dieser Fall tritt dann auf, wenn das Cluster im rechten Sohn höchstens zwei Elementarknoten enthält. Sind es genau zwei Elementarknoten gilt nämlich $(u_{2,q}\, u_{1,1}) = (\tilde{u}_{q,2}\, \tilde{u}_{1,1})$, im Falle eines einzelnen Elementarknotens gilt $u_{1,1} = \tilde{u}_{1,1}$. In diesem Fall darf der rechte Sohn nur in einer Richtung durchlaufen werden. Ein Lösungsbaum auf acht Elementarknoten ist in Abbildung 4.5 skizziert.

4.5 Wahl der Benefitfunktion

Eine zentrale Rolle des gesamten Verfahrens spielt die Wahl der Benefitfunktion, also die Art und Weise, in der einem potentiellen Paar von Clustern eine Qualität zugeordnet wird. Diese Qualität entspricht dem Gewicht der Kanten im Clustergraphen bei der Berechnung des Matchings. Bei der Bestimmung der Qualität einer Kante kann ist es sinnvoll, nicht nur ihr Gewicht zu betrachten, sondern auch die Kosten der Wege innerhalb der Cluster. Der

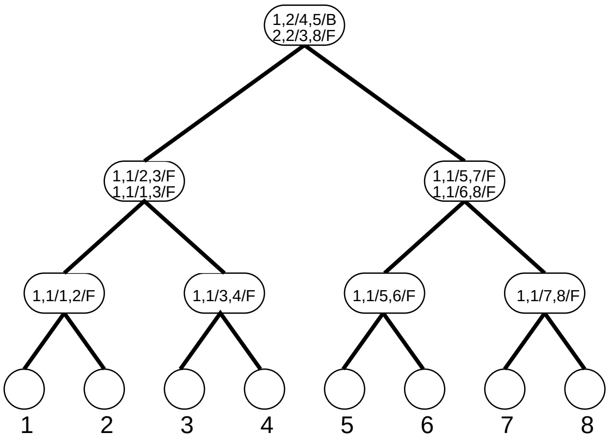

Abbildung 4.5: *Ein Lösungsbaum mit maximal zwei gespeichertern Wegen pro Cluster. Die ersten Informationen in einem Cluster stellen die Indizes der Wege, die aus den Söhnen gewählt wurden, dar. Darauf folgt der Aus- und Eintrittspunkt beider gewählter Wege. Schließlich folgt die Festlegung der Richtung des rechten Sohnes. In der Wurzel sind die Wege* $1, 2, 3, 4, 5, 7, 6, 8$ *und* $1, 3, 8, 7, 5, 6, 4, 2$ *gespeichert.*

Einfachheit halber bietet es sich an, als Bewertungsgrundlage die Kosten der entstehenden Touren in den neuen Knoten heranzuziehen. Durch diese Maßnahme wird verhindert, daß zwei Cluster miteinander gepaart werden, die zwar nahe beieinander liegen, deren Verbindung aber schlechte Teiltouren hervorrufen würde (siehe Abb. 4.6).

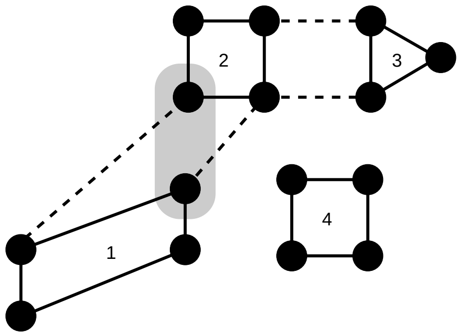

Abbildung 4.6: *Obwohl Cluster 1 und 2 Knoten haben, die sehr dicht beieinanderliegen (grau unterlegt), ist die Verbindung von 2 und 3 günstiger.*

Da ein Maximum Weight Matching angewandt wird, um günstige Verbindungen zu finden, müssen gute Kanten im Clustergraphen — solche, die Cluster so miteinander verbinden, daß gute Teiltouren entstehen — mit hohen Gewichten versehen werden. Hier bietet sich die Skalierung der Benefits auf ganzzahlige Werte zwischen 0 und 100 an. Wie diese Skalierung vor sich geht, wird im folgenden aufgezeigt.

Eine Kante soll einen großen Benefit haben, wenn ihre Aufnahme in eine Tour die Kosten möglichst wenig erhöht. Beispielsweise könnte man der Kante den Benefit 100 zuordnen, deren Gewicht das kleinste im gesamten Graphen ist. Ist das Gewicht einer anderen Kante dann x-mal so groß erhält sie den Benefit $\frac{1}{x} \cdot 100$. Eine andere Möglichkeit besteht darin, einer Kante (i, j) den Benefit 100 zuzuordnen, wenn diese sowohl von i als auch von j aus diejenige mit den geringsten Kosten ist, wenn also gilt

$$c_{ij} \leq c_{iv} \text{ und } c_{ij} \leq c_{vj} \ \forall v.$$

Auch hier erhalten die anderen Kanten dann einen Benefit relativ zum bestmöglichen Wert. Schließlich kann auch die Kombination dieser beiden Varianten eine sinnvolle Gewichtung ergeben.

Der Benefit einer Kante ist in den drei Fällen nicht allein von ihr selbst abhängig. Außerdem müssen bei der Gewichtung von Kanten zwischen zwei echten Clustern die in diesen gespeicherten Touren berücksichtigt werden. Daher macht es Sinn, zuerst für jedes Knotenpaar einen *Pre-Benefit* zu berechnen. Bei diesem handelt es sich um einen Wert, der umso kleiner ist, je besser ein Knotenpaar zunächst scheint. Es gibt mehrere Möglichkeiten, diesen Pre-Benefit zu errechnen:

- Es werden die Kosten aller Touren berechnet, die aus der Verbindung aller Teiltouren in den beiden Knoten entstehen können und
  - die Kosten der besten
  - die Kosten der schlechtesten
  - der Durchschnitt der Kosten

  dieser Touren als Pre-Benefit genommen.

- Es werden die Kosten aller Touren berechnet, die aus der Verbindung der besten k Teiltouren in den beiden Knoten entstehen können und
  - die Kosten der besten
  - die Kosten der schlechtesten
  - der Durchschnitt der Kosten

  dieser Touren als Pre-Benefit genommen.

- In geometrischen Problemen wird der Abstand der Schwerpunkte voneinander als Pre-Benefit genommen.

Für jeden Knoten i wird der beste Pre-Benefit b_i gespeichert, an dem er beteiligt ist. Außerdem wird der global beste Pre-Benefit $b_{min} = \min_{i=1\ldots,n} b_i$ gespeichert. Der Benefit berechnet sich jetzt aus der Formel

$$\text{benefit}(i,j) = \begin{cases} 100 \cdot \left\lfloor \frac{l_{max} - \text{prebenefit}(i,j)}{l_{max} - b_{min}} \right\rfloor & \text{wenn } l_{max} \geq \text{prebenefit}(i,j) \\ 0 & \text{sonst} \end{cases}$$

Dabei stellt l_{max} den maximal zulässigen Pre-Benefit dar. Er wird auch *Referenzwert* genannt. Jede Verbindung mit größerem Pre-Benefit erhält automatisch das Kantengewicht 0.

Der Referenzwert l_{max} selbst kann indirekt manipuliert werden. Er ergibt sich aus der Formel

$$l_{max} = \begin{cases} \frac{f_{ref} - q_{min}}{1 - q_{min}} \cdot b_{min} & \text{für } q_{min} < 1 \\ f_{ref} \cdot b_{min} & \text{für } q_{min} = 1 \end{cases}.$$

Dabei ist q_{min} die Mindestqualität, die von einer Kante gefordert wird. Mit dem *Referenzfaktor* f_{ref} erhält man den Pre-Benefit, der dieser Mindestqualiät entspricht, indem man $f_{ref} \cdot b_{min}$ berechnet. Eine Kante mit dem Pre-Benefit $f_{ref} \cdot b_{min}$ erhält dementsprechend die Qualität q_{min}. Es gilt $0 \leq q_{min} \leq 1$ und $f_{ref} \geq 1$.

Der Grund für den Einsatz einer Mindestqualität liegt in der „Gier“ des Matchings begründet. In Abbildung 4.7 würde sich ein gutes Matching beispielsweise aus der Verbindung je zweier Knoten auf dem Kreis ergeben. Die Kante zwischen A und B erhöht zwar die Summe der Kantengewichte, doch ihr Erscheinen im Matching macht wenig Sinn, da sie sicherlich nicht in der optimalen Lösung des Problems auftritt. Bewertet man die Kante mit 0, weil sie einer gewissen Mindestqualität nicht genügt, so erhöht sie die Summe der Kantengewichte nicht und wird dementsprechend vom Matchingalgorithmus nicht ausgewählt. Die Knoten A und B können dadurch später besser in den Kreis integriert werden.

Durch den Referenzfaktor f_{ref} kann beeinflußt werden, wie genau die Bewertung einer Kante ist. Zwei Kanten gleicher Qualität müssen nicht zwangsläufig den gleichen Pre-Benefit haben. Dieser muß nur im gleichen Intervall $[b_1, b_2]$ liegen. Die Grenzen dieses Intervalls sind gegeben durch den kleinsten und den größten Pre-Benefit, für den die verlangte Qualität erzielt wird. Die Länge des Intervalls wird daher durch f_{ref} festgelegt.

Bei der Bestimmung des Benefits wird nicht wirklich l_{max} verwendet, sondern ein Wert, der noch von der Kante (i,j) abhängt. Der Benefit ist dadurch nicht nur von globalen Gegebenheiten abhängig, sondern kann auch lokale Besonderheiten berücksichtigen. Abbildung 4.8 zeigt einen Graphen, bei dem eine Teilmenge der Knoten sehr dicht beieinander liegt

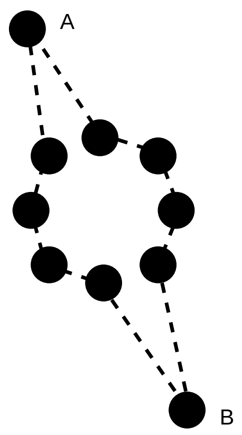

Abbildung 4.7: *Eine Kante zwischen A und B würde die Summe der Kantengewichte bei einem Maximum Weight Matching zwar erhöhen, jedoch ergäbe sich, wird sie mit 0 bewertet, auf den anderen Kanten die gleiche Auswahl.*

(rechts) und der Rest der Knoten durch größere Distanzen getrennt ist (links). Eine Bewertung der Kanten nur nach der kürzesten Kante im Graphen ist nicht sinnvoll, da der Teilgraph links praktisch unabhängig vom Rest behandelt werden kann. Die Formel für

Abbildung 4.8: *Bei geclusterten Graphen ist es sinnvoll, die Qualität einer Kante auch von lokalen Gesichtspunkten abhängig zu machen.*

den *Referenzwert der Benefitfunktion* $l_{max}(i,j)$ lautet daher

$$l_{max}(i,j) = \begin{cases} \frac{f_{ref}-q_{min}}{1-q_{min}} \cdot b_{base}(i,j) & \text{für } q_{min} < 1 \\ f_{ref} \cdot b_{base}(i,j) & \text{für } q_{min} = 1 \end{cases} .$$

mit $b_{base}(i,j) = (1-g)\frac{b_i+b_j}{2} + g \cdot b_{min}$, wobei $0 \leq g \leq 1$ den *globalen Einfluß* bei der Berechnung festlegt. Durch g kann man also bestimmen, welchen Einfluß der global beste Pre-Benefit auf die Bewertung einer Kante hat. Entsprechend lautet also der Benefit für eine Kante (i, j)

$$\text{benefit}(i,j) = \begin{cases} 100 \cdot \left\lfloor \frac{l_{max}(i,j) - \text{prebenefit}(i,j)}{l_{max}(i,j) - b_{base}(i,j)} \right\rfloor & \text{falls prebenefit}(i,j) \leq l_{max}(i,j) \\ 0 & \text{sonst} \end{cases}$$

Die Aussagen der letzten Absätze sollen anhand einer Abbildung verdeutlicht werden (Abb. 4.9). Oben links in der Abbildung ist der Graph gemeinsam mit den Pre-Benefits der Kanten zu sehen. Oben rechts werden die Benefits ganz ohne globalen Einfluß berechnet.

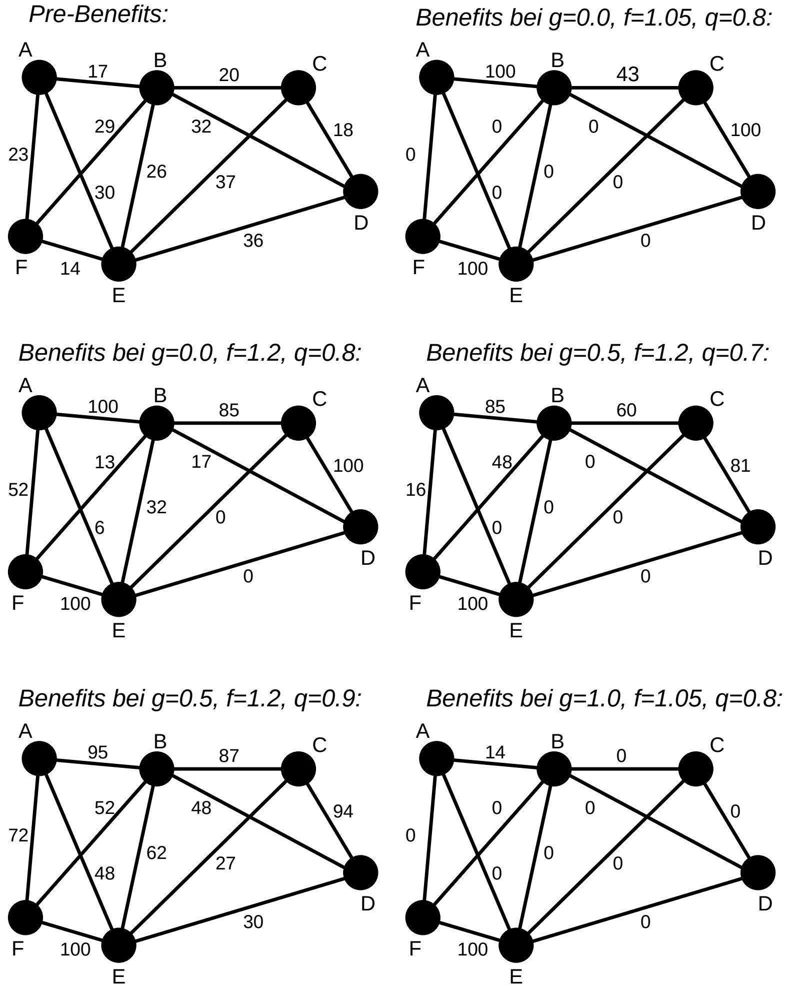

Abbildung 4.9: *Ein Graph mit den gegebenen Pre-Benefits und den daraus resultierenden Benefits für verschiedene Parameter.*

Da es an den Knoten A und B keine Kante mit besserem Pre-Benefit als 17 gibt, erhält die Kante zwischen ihnen den Benefit 100. Entsprechendes gilt für die Kanten (C, D) und (E, F). Es gilt nämlich in allen genannten Fällen $b_{base}(i, j) = \text{prebenefit}(i, j)$. Für die Kante (B, C) gilt dagegen beispielsweise $b_{base}(B, C) = \frac{b_B + b_C}{2} = \frac{17+18}{2} = 17,5$. Mit dem Referenzfaktor $f_{ref} = 1,05$ und der Mindestqualität $q_{min} = 0.8$ ergibt sich daher für $l_{max}(B, C) = \frac{0,25}{0,2} \cdot 17,5 = 21,87$ und schließlich $\text{benefit}(B, C) = 100 \cdot \frac{21,87-20}{21,87-17,5} = 43$. Für das Matching wird die Kante den Wert 0 erhalten, da sie nicht die Mindestqualität von $0,8 \cdot 100 = 80$ hat.

In der Mitte links ist der Referenzfaktor auf $f_{ref} = 1,2$ erhöht. Dadurch erhöht sich auch der Benefit der Kante (B, C) auf 83, wodurch sie für die Berechnung des Matchings relevant wird.

Rechts daneben in der Abbildung wurde ein 50 prozentiger globaler Einfluß eingestellt. Daher hat nur die Kante (F, E) den Benefit 100. Für (A, B) gilt nämlich $b_{base}(A, B) = 0,5 \cdot 17 + 0,5 \cdot 14 = 15,5$. Es folgt $l_{max}(A, B) = 1,67 \cdot 15,5 = 25,83$ und $\text{benefit}(A, B) = 85$.

Bei ausschließlich globalem Einfluß (unten rechts) ergibt sich für die Kante (A, B) mit $b_{base}(A, B) = 14$, $l_{max}(A, B) = 17,5$ sogar $\text{benefit}(A, B) = 14$. Es werden daher nur wenige Kanten in das Matching aufgenommen werden. Das führt dazu, daß sehr viele Matchings berechnet werden müssen, da sich die Anzahl von Clustern pro Iterationsschritt nur geringfügig reduziert.

Abschließend soll bemerkt werden, daß das Greedy-Verfahren simuliert werden kann, indem $g = q_{min} = f_{ref} = 1.0$ gewählt wird. Allerdings ist diese Simulation durch den Einsatz des Matchings deutlich langsamer. Positiv dagegen ist die Tatsache, daß im Gegensatz zum Original so viele gleichteure Kanten wie möglich auf einmal ausgewählt werden.

4.6 Einsatz von Kandidatenmengen

Bereits erwähnt wurde die Tatsache, daß das Matching auf n Knoten in $O(n^3)$ berechnet wird. Allerdings läßt sich die tatsächliche Laufzeit um in der O-Notation untergehende Faktoren beschleunigen. Dies erreicht man, indem man die Anzahl der zu untersuchenden Kanten im Graphen deutlich einschränkt. Diese Einschränkung führt nicht zwangsläufig zu einer schlechteren Lösung. Viele Kanten in vollständigen Graphen sind sicher nicht in einer optimalen Tour enthalten (siehe Abb. 4.10). Diese können von vornherein entfernt werden.

Bereits in Abschnitt 3.1 wurde auf Kandidatenmengen eingegangen. Diese lassen sich auch beim iterierten Matching sinnvoll einsetzen, wie ein Vergleich der Laufzeiten zeigt (Abb. 4.11). Auf einem vollständigen Graphen mit 300 Knoten dauert die Berechnung des Matching bereits deutlich länger, als bei einem dünn besetzten Graphen mit der gleichen Anzahl an Knoten. Daher wird das Matching nur auf einem dünn besetzten Graphen durchgeführt,

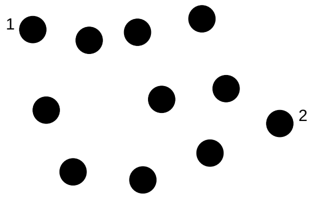

Abbildung 4.10: *Die Kante von 1 nach 2 wird sicher nicht in der optimalen Tour erscheinen.*

der aus allen Knoten und aus den durch die Kandidatenmengen ausgewählten Kanten besteht. Im geographischen Fall bietet es sich an, eine Kombination aus Nächsten Nachbarn und Delaunay Graph als Kandidatenmenge zu wählen. In den meisten Fällen werden optimale Touren ausschließlich über Kanten dieser Menge führen.

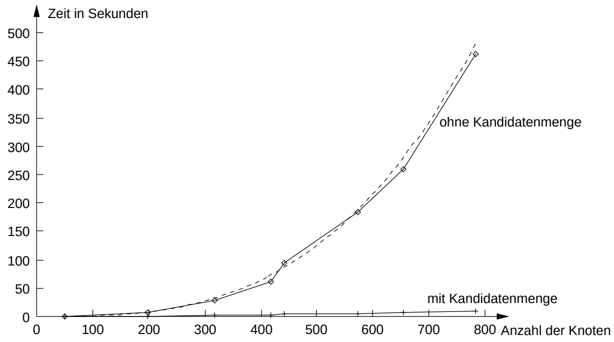

Abbildung 4.11: *Vergleich der Laufzeiten eines Matchings mit und ohne Einsatz einer Kandidatenmenge. Bei einem vollständigen Graphen nimmt die Laufzeit kubisch mit der Anzahl der Knoten zu (zum Vergleich ist die Funktion $c \cdot x^3$ abgetragen). Bei Einsatz der Kombination von Delaunay Graph und 10-Nearest-Neighbour Subgraph ist diese Zunahme nur noch quadratisch.*

Die Kandidatenmenge wird solange eingesetzt, bis die Anzahl der Cluster die Schwelle n_{cand} unterschreitet. Für kleinere Clustergraphen wird das Matching auf dem vollständigen Graphen durchgeführt. Durch diese Maßnahme wird erreicht, daß auch bei schlechter Wahl der Kandidatenmenge — beispielsweise, wenn diese nicht zusammenhängend ist — dennoch Lösungen akzeptabler Qualität gefunden werden.

Zur Berechnung der Delaunay Triangulation wird eine Implementation des *Sweepline* Algorithmus von Steve J. Fortune verwendet. Dieser wurde 1987 in [For 1987] vorgestellt. Leider benötigt dieser sehr viel Speicher, weshalb sich die Untersuchungen auf Probleminstanzen mit weniger als 2 000 Knoten beschränken mußten.

4.7 Zusammenfassen zweier Knoten

Nachdem durch das Matching festgelegt wurde, welche Knoten gut zueinander „passen", muß aus je zwei gepaarten Knoten ein neuer erzeugt werden, nämlich deren *Vater* im Lösungsbaum. Die bereits in den Söhnen gespeicherten Informationen sollten nach Möglichkeit nur vom Vater referenziert und nicht erneut abgespeichert werden.

Beispielsweise sollen die Cluster $C_1 = (V_1, H^1_{h_1})$ und $C_2 = (V_2, H^2_{h_2})$ zusammengefaßt werden. Es seien $|V_1| = p$ und $|V_2| = q$. Aus der Kombination zweier einzelner Touren ergeben sich $2pq$ neue mögliche Touren (siehe Abschnitt 4.4). Insgesamt können also $2h_1h_2pq$ neue Wege entstehen. Eine solche Menge ist bereits für kleine Cluster nicht mehr zu verwalten. Es muß also eine *Selektion* unter allen möglichen Wegen durchgeführt werden.

Diese Selektion ist auf zweierlei Art denkbar. Einerseits kann man eine feste Zahl t vorgeben, die die maximale Anzahl zu speichernder Wege darstellt (*Absolute Selektion*). Andererseits kann man fordern, daß kein gespeicherter Weg mehr als p Prozent teurer als der beste Weg ist (*Relative Selektion*). Um sich in kleinen Clustern möglichst viele Optionen offenzuhalten, kann es sinnvoll sein, zu Beginn des Verfahrens sehr viele Wege auszuwählen, jedoch zum Ende hin nur noch wenige der besten. Dazu kann man einen *Abnahmefaktor* $0 < d \leq 1$ einstellen, mit der die *Selektionsrate* t beziehungsweise p nach jedem Iterationsschritt multipliziert wird. Das ist auch insofern sinnvoll, als daß bei kurzen Touren ein großer Kostenunterschied eine sehr kleine Rolle in der endgültigen Lösung spielt, er jedoch die Möglichkeit neuer, besserer Kombinationen in den nächstgrößeren Clustern bietet.

Auch die Anzahl der Touren, die aus der Kombination zweier Zyklen der zugrunde liegenden Cluster entstehen können, sollte eingeschränkt werden. Es macht sicherlich Sinn, hier nur eine bestimmte Anzahl guter Wege zuzulassen. Allerdings werden alleine für die Berechnung aller Tourkosten $p \cdot q$ Schritte benötigt. Daher soll die Menge der Elementarknotenpaare, die überhaupt als Verbindungsknoten in Frage kommen, reduziert werden. In geometrischen Graphen bietet sich folgende Vorgehensweise an (Abb. 4.12). Von beiden Clustern werden die nächsten Elementarknoten zum Schwerpunkt des jeweils anderen Clusters bestimmt. Mit diesem werden dann alle die Touren berechnet, die mindestens einen dieser Elementarknoten enthalten. Die besten k so entstehenden Touren werden dann für die weitere Selektion herangezogen.

Leider läßt sich die Menge der zu untersuchenden Kombinationsmöglichkeiten nicht einschränken. Bei einem geometrischen TSP seien bereits k Wegekombinationen in einem Cluster gespeichert, wobei k gleichzeitig die maximale Anzahl zu speichernder Wege darstelle (die Selektionsrate bei absoluter Selektion). Die Kosten des k-ten Weges seien c.

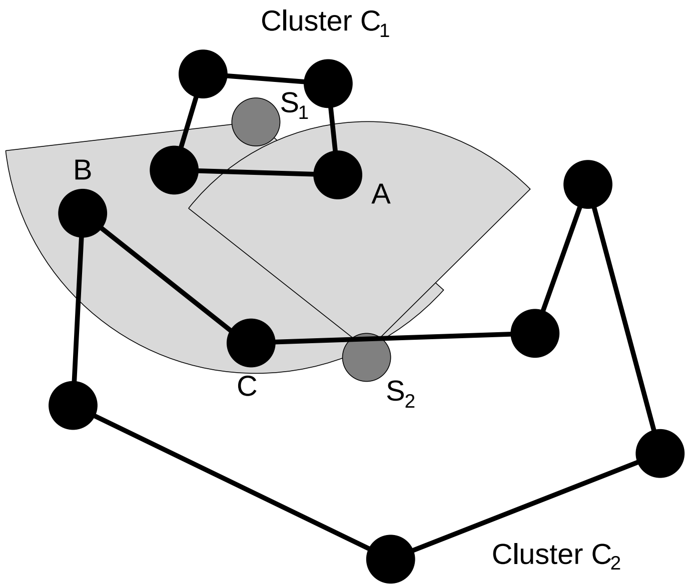

Abbildung 4.12: *Sollen zwei Cluster C_1 und C_2 eines geometrischen Graphen verbunden werden, so werden über ihre Schwerpunkte (S_1 und S_2) gute Verbindungsknoten bestimmt. Aus der Sicht von Cluster C_2 liegt der Elementarknoten A aus C_1 besonders günstig, für C_1 liegen B und C sehr gut. Für jeden so bestimmten Elementarknoten eines Clusters werden alle möglichen Verbindungen mit den Elementarknoten des anderen Clusters berechnet.*

Man könnte nun annehmen, daß bei der Kombination zweier Wege aus den Söhnen mit den Kosten c_1 und c_2, mit $c_1 + c_2 \geq c$ keinesfalls ein Weg entstehen könnte, dessen Kosten kleiner als c ist. Sind die Wege in aufsteigenden Reihenfolge in den Knoten gespeichert, so müßte dann keine der Folgekombinationen mehr berechnet werden. Die Aussage trifft aber nicht zu, wie Abbildung 4.13 zeigt. Daher müssen für jede mögliche Kombination die Kosten der entstehenden Touren ausgerechnet werden.

Trotz allem bietet es sich an, die Wege in sortierter Folge zu speichern, nicht zuletzt, weil einige Benefitfunktionen nur einige der besten gespeicherten Wege als Berechnungsgrundlage heranziehen. Außerdem müssen dann nicht alle möglichen Kombinationen gespeichert werden, bevor die Selektion einige von ihnen wieder verwirft. Stellt sich bei der Kombination zweier Zyklen heraus, daß die Kosten der daraus entstehenden Tour nicht den Selektionskriterien genügt, braucht sie gar nicht erst gespeichert zu werden.

Zur Sortierung der Wege wird eine Variante des binären Suchbaumes eingesetzt. Neben den üblichen Informationen wird in jedem Teilbaum die Anzahl der Knoten festgehalten, die in diesem enthalten sind. Auf diese Weise kann, ohne den Baum traversie-

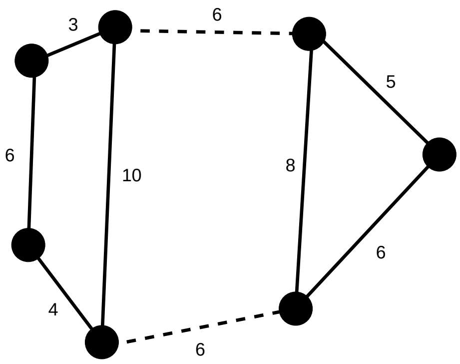

Abbildung 4.13: *Die Kosten der bei der Kombination der beiden Wege entstehenden Tour sind mit 36 Einheiten kleiner als die Summe der Kosten beider Teilwege (42 Einheiten). Die an den Kanten angegebenen Kosten entsprechen bis auf Runden den Abständen der Knoten voneinander.*

ren zu müssen, jeder beliebigen Knoten anhand seiner Position in der sortierten Folge in $O(\log n)$ angesteuert werden. In Abbildung 4.14 ist beispielsweise in einem üblichen

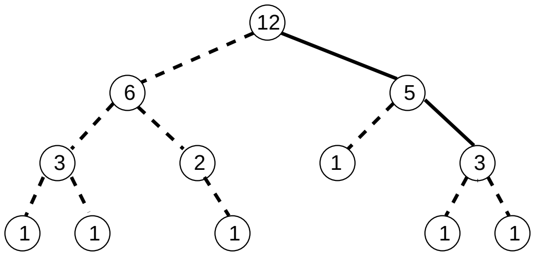

Abbildung 4.14: *Erweiterter binärer Suchbaum. Die in den Knoten geschriebenen Zahlen stellen die Anzahl der Elemente in dem entsprechenden (Teil-) Baum dar. Die durchgezogene Linie führt zum elften Element in der sortierten Folge der Elemente. Die sortierten Elemente sind selbst nicht dargestellt.*

Binärbaum — der linke Sohn enthält Elemente, die kleiner sind als das der Wurzel, der rechte Sohn größere — der Weg zum elften Knoten als durchgezogene Linie dargestellt. Von der Wurzel ausgehend findet man den Weg mit folgendem rekursiven Algorithmus:

Algorithmus: Suche i-ten Knoten im binären Suchbaum

1. Sei n_l die Anzahl der Elemente im linken Teilbaum.
2. Wenn $i \leq n_l$: Suche den i-ten Knoten im linken Teilbaum.
3. Wenn $i = n_l + 1$: Wurzel ist der gesuchte Knoten.
4. Wenn $i > n_l + 1$: Suche $(i - n_l - 1)$-ten Knoten im rechten Teilbaum.

Im konkreten Beispiel erkennt man also an der Wurzel, daß der elfte Knoten sich im rechten Teilbaum befinden muß und dort an vierter Stelle steht. Dort befindet er sich im rechten Teilbaum, da sich im linken Teilbaum nur ein Element befindet. Der gesuchte Knoten ist hier an zweiter Position zu finden. Dies ist aber die Wurzel des Teilbaumes selbst, da sich im linken Teilbaum wiederum nur ein Element befindet.

Zunächst soll das Speichern eines Weges bei absoluter Selektion beschrieben werden. Dabei werden die soeben berechneten Kosten c einer Wegekombination mit den Kosten c_r des r-ten gespeicherten Weg im Baum verglichen, wobei r die Selektionsrate ist. Ist $c \leq c_r$ muß die neue Kombination in den Baum aufgenommen werden, ansonsten kann sie verworfen werden.

Bei relativer Selektion werden die Kosten c der neuen Kombination mit den Kosten c_1 der in der Wurzel gespeicherten Kombination verglichen. Gilt $c \leq (1 + \frac{r}{100}) \cdot c$ mit der Selektionsrate r, so wird die neue Kombination aufgenommen.

Nachdem alle Kombinationen durchgerechnet wurden, wird eine Inorder-Traversierung des Baumes durchgeführt. Bei absoluter Selektion werden die ersten r Kombinationen im neuen Cluster gespeichert, bei relativer Selektion alle Wege, die nicht mehr als r Prozent teurer sind als der billigste. Dieser Schritt ist notwendig, da bei der oben beschriebenen Vorauswahl ja nur mit den zum aktuellen Zeitpunkt besten Wegen verglichen wurde. Da neue Kombinationen aber bessere Wege ergeben können, sind unter Umständen Wege im Baum gespeichert, die die Selektionskriterien nicht mehr erfüllen. Die Zahl dieser Wege wird aber nicht allzu groß sein, weil bei der Vorauswahl mit Kombinationen guter Wege begonnen wird, die voraussichtlich auch gute neue Kombinationen liefern.

4.8 Vor- und Nachbereitung des Matchings

Damit das Matching effektiv durchgeführt werden kann, werden einige Vorbereitungen getroffen. Bei der Besprechung der Benefitfunktionen in Abschnitt 4.5 wurde bereits erwähnt, daß ein Benefit aus zuvor bestimmten Pre-Benefits berechnet wird. Diese werden mit Hilfe der Gewichte der Kanten zwischen den Elementarknoten ermittelt.

Wurde eine Kandidatenmenge eingesetzt, so muß diese natürlich auch auf jeden Clustergraphen übertragen werden. Die Kanten des ursprünglichen Graphen müssen also auf Kanten jedes neuen Graphen übertragen werden. Auch hierfür wird auf die Elementarknoten der Cluster zurückgegriffen.

4.8.1 Berechnung des Pre-Benefits

Beim Pre-Benefit einer Kante zwischen zwei Elementarknoten handelt es sich immer um das Gewicht dieser Kante. Inzidiert dagegen mindestens ein echtes Cluster mit der Kante, so gibt es mehrere Möglichkeiten, den Pre-Benefit zu berechnen. Auch dieses wurde bereits in Abschnitt 4.5 angesprochen. Dort wurde aber noch nicht konkret ausgeführt, wie dieser Pre-Benefit ermittelt wird. Das soll an dieser Stelle nachgeholt werden.

Im letzten Abschnitt wurde schon das Problem angesprochen, daß es beim Verbinden zweier Zyklen $(v_1, \ldots, v_p)$ und $(u_1, \ldots, u_q)$ $p \cdot q$ verschiedene Kombinationsmöglichkeiten gibt. Auch bei der Berechnung des Pre-Benefits ist das Berechnen all dieser Kombinationen nicht sinnvoll. Daher wird die gleiche Auswahl an Wegkombinationen getroffen, die auch dort besprochen wurde, unabhängig davon, welche Benefitfunktion ausgewählt wurde.

Beim Zusammenfassen zweier Cluster wurde eine Selektion über alle dann noch möglichen Wegkombination durchgeführt, da nicht alle gespeichert werden konnten. Das muß bei der Berechnung des Pre-Benefits nicht unbedingt geschehen, da hier nur die Kosten der Wege eine Rolle spielen. Diese werden aber ohnehin berechnet.

Bestmögliche Kombination

Bei der Auswahl der bestmöglichen Kombination als Benefit werden alle Wege der beiden beteiligten Cluster miteinander verbunden und die Kosten der besten dabei entstehenden Tour als Pre-Benefit zurückgeliefert.

Alternativ kann die Anzahl der Wege, die aus den beiden Clustern herangezogen werden auf eine kleine Zahl eingeschränkt werden. Dann werden nur die ersten k Wege beider Cluster kombiniert. Der dabei ermittelte Wert kann als Näherung für die tatsächlich bestmögliche Kombination aufgefaßt werden.

Durchschnitt aller Kombinationen

Wird die bestmögliche Kombination als Benefitfunktion gewählt, kann es passieren, daß die Verbindung zweier Cluster als gut bewertet wird, obwohl es nur eine wirklich gute

Verbindungsmöglichkeit zwischen ihnen gibt. Die Verbindung eines dieser beiden Cluster mit einem anderen hätte zwar einen schlechteren besten Weg zur Folge, würde jedoch noch eine Reihe weiterer guter Wege liefern.

Statt der bestmöglichen Kombination könnte also der Durchschnitt aller Kombinationen bessere Resultate liefern. Daher steht auch eine Benefitfunktion zur Verfügung, die unter allen möglichen Kombinationen von Touren den Durchschnitt der Kosten bildet und als Pre-Benefit liefert.

Auch hier kann die Anzahl der Wege, die für die Berechnung herangezogen werden, auf eine kleine Zahl k gesetzt werden. Dann werden nur die besten k Wege jedes Clusters für den Pre-Benefit ausgewertet.

Schlechteste Kombinationsmöglichkeit

Eine weitere Überlegung führt dazu, bei der Kombination der Cluster die Kosten der teuersten entstehenden Tour als Pre-Benefit zu nehmen. Schließlich dürfte ein Clusterpaar, dessen Tourkombination selbst im schlechtesten Fall noch relativ gute Ergebnisse liefert, besonders gut zueinander passen.

Wie in den beiden zuvor geschilderten Fällen, ist auch hier eine Näherung über die Auswahl von k besten Wegen in beiden Clustern eines möglichen Paares vorgesehen.

Abstand der Schwerpunkte

In geometrischen Graphen gibt es eine besondere Möglichkeit, einen Pre-Benefit zu erhalten. Man führt einfach für jedes Cluster den Schwerpunkt aller Elementarknoten mit und betrachtet dann den Abstand zweier Schwerpunkte als Pre-Benefit. Das hat den Vorteil, daß unabhängig von den gespeicherten Wegen ein Wert ermittelt werden kann, wofür selbstverständlich erheblich weniger Zeit benötigt wird als bei den oben genannten Benefitfunktionen.

Beim Berechnen des Pre-Benefits wird für jedes Cluster sein nächster Nachbar (bezüglich des Pre-Benefits) und der global beste Pre-Benefit gespeichert. Damit kann dann der Benefit bei der Berechnung des Matchings in konstanter Zeit ermittelt werden.

4.8.2 Kandidatenmengen auf echten Clustergraphen

Nachdem nach einem Matching die Cluster verbunden worden sind, müssen, sofern eine Kandidatenmenge eingesetzt wird, auch wieder Kanten zwischen den neuen Clustern eingefügt werden. Dazu läuft das Programm über alle Kanten der ursprünglichen Kandidatenmenge und prüft, ob sich die beiden Elementarknoten am Ende dieser Kante in unterschiedlichen Clustern befinden. In diesem Fall werden die beiden Cluster durch eine neue Kante verbunden. Der Wert dieser Kante ist der Pre-Benefit, wie er sich durch die oben erläuterte Vorgehensweise ergibt.

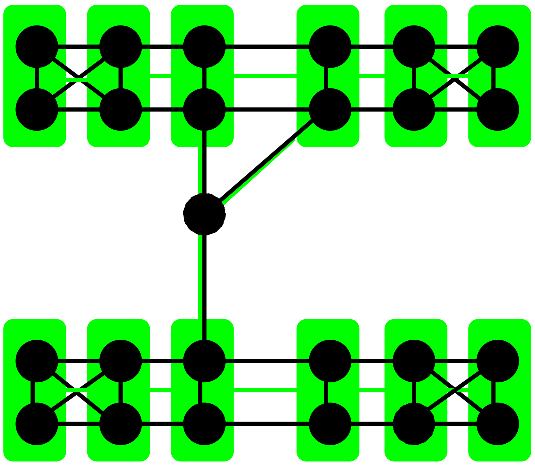

Abbildung 4.15: *Nachdem beim Matching die grau unterlegten Elementarknoten gematcht wurden, wird die Kandidatenmenge (3-Nearest-Neighbour-Subgraph, schwarze Linien) auf die neuen Cluster übertragen (graue Linien).*

4.9 Artikulationskanten

In dünn besetzten Graphen entsteht das Problem, daß bestimmte Kanten nicht in einer Hamilton-Tour vorkommen. In Abschnitt 1.1 wurde zwar gezeigt, wie ein dünn besetzter Graph G auf einen vollständigen Graphen K_n abgebildet werden kann, so daß eine optimale Lösung auf K_n auch gleichzeitig eine optimale Lösung in G darstellt. Heuristiken liefern aber nur Näherungslösungen, die unter Umständen auf dem ursprünglichen Graphen keinen Hamilton-Zyklus darstellen.

Enthält das Matching eine Kante, die in keiner Hamilton-Tour vorkommt, besteht die Gefahr, daß keine zulässige Lösung gefunden wird. Man benötigt also ein Verfahren, mit dem die Aufnahme solcher Kanten in das Matching verhindert werden kann. Diese Kanten sind durch folgende Definition charakterisiert.

Definition 4.7 (Artikulationskante). Gegeben sei ein Graph $G = (V, E)$. Eine Kante $uv \in E$ heißt *Artikulationskante*, wenn sich durch Entfernen der Knoten u und v inklusive aller inzidierender Kanten die Anzahl der Zusammenhangskomponenten in G erhöht. □

Die Definition ist analog zur bekannten Definition eines *Artikulationspunktes*.

Artikulationskanten müssen von vornherein aus dem Graphen entfernt werden. Der Algorithmus zur Suche aller Artikulationskanten entspricht in etwa dem für die Artikulationspunkte. Zunächst wird von einem ausgesuchten Knoten aus eine Tiefensuche auf dem Graphen durchgeführt. In dem entstehenden Baum werden auch sogenannte Rückwärtskanten eingefügt. Dieses sind Kanten, die zurück in Richtung Wurzel führen. Auf diesem Baum werden gemäß Satz 4.9 alle Artikulationskanten bestimmt.

Bezeichnung 4.8. Sei $G = (V, E)$ ein Graph, $i, j \in V$, $A, B \subset V$.

T_G bezeichne einen durch Tiefensuche auf G enstandenen Baum inklusive der Rückwärtskanten.

$i \to j$ bedeute, daß es einen Weg in T_G von i nach j gibt, der ausschließlich Vorwärtskanten enthält.

$N(i)$ bezeichne die Menge aller Knoten in T_G, die Nachfolger von i sind, inklusive i selbst; $N(i) = \{j \in V : i \to j\}$

$N_j(i)$ sei die Menge $N(i) \setminus N(j)$

$V(i)$ bezeichne die Menge aller Knoten in T_G, die über Vorgänger von i erreichbar sind, ohne daß Rückwärtskanten benutzt werden; $V(i) = \{j \in V, j \neq i : \exists k \in V \text{ mit } k \to i \text{ und } k \to j\}$.

$V_j(i)$ sei die Menge $V(i) \setminus V(j)$.

$A \not\to B$ bedeute, daß es keinen Weg aus der Knotenmenge A in die Knotenmenge B gibt.

□

Für diese Arbeit sind nur zusammenhängende Graphen interessant. Dort gilt für Artikulationskanten folgender Satz.

Satz 4.9. Sei $G = (V, E)$ ein zusammenhängender Graph und T_G ein dazugehöriger, durch Tiefensuche entstandener Baum. Eine Kante $e = (i, j) \in E$ ist genau dann eine Artikulationskante, wenn eine der folgenden Bedingungen zutrifft:

1. e ist Vorwärtskante in T_G *und*

i. i ist Wurzel *und*

a) j hat mehr als einen direkten Sohn *oder*

b) i hat mehr als zwei direkte Söhne *oder*

c) i hat genau zwei direkte Söhne und j hat mindestens einen direkten Sohn.

ii. i ist nicht Wurzel *und*

a) j besitzt mindestens einen direkten Sohn k, so daß gilt $N(k) \not\to V(i)$ *oder*

b) i besitzt mindestens einen direkten Sohn $k \neq j$, so daß gilt $N(k) \not\to V(i)$.

2. e ist Rückwärtskante *und*

i. j ist Wurzel *und*

a) j hat mehr als einen direkten Sohn *oder*

b) es gibt mindestens einen direkten Sohn k von i, so daß gilt $N(k) \not\to V_j(i)$.

ii. j ist nicht Wurzel *und*

a) $V_j(i) \not\to V(j)$ und $N(i) = \{i\}$ *oder*

b) für mindestens einen direkten Sohn k von i sind mindestens zwei der Aussagen $V_j(i) \not\to V(j)$, $N(k) \not\to V_j(i)$, $N(k) \not\to V(j)$ wahr.

□

Der Beweis dieser Aussage ist offensichtlich. Bei dem Baum T_G handelt es sich nur um eine andere Darstellungsweise des Graphen. Eine Kante e ist daher Artikulationskante, wenn der Baum in zwei Zusammenhangskomponenten getrennt wird, sobald diese Kante entfernt wird. In den im Satz angeführten Fällen werden entweder Äste „auseinandergebrochen“ oder voneinander getrennt. Die Kante e ist also in jedem aufgezählten Fall eine Artikulationskante. In jedem anderen Fall gilt dies nicht. Trifft eine der Aussagen der Form $A \not\to B$ nicht zu, heißt das, daß ein Weg aus der einen in die andere Knotenmenge existiert. Die Knotenmengen bleiben also verbunden. Treffen Aussagen der Form „... hat direkten Sohn ...“ nicht zu, werden die entsprechenden Äste nicht voneinander getrennt. In jedem Fall bleibt der Graph zusammenhängend.

Abbildung 4.16 soll den Sachverhalt an einem konkreten Beispiel verdeutlichen. Die Tiefensuche starte bei Knoten 1 und werde immer mit dem Knoten fortgesetzt, dessen Index am kleinsten ist. Dann erhält man den rechts dargestellten Baum. Die Kante $(2,3)$ ist eine Artikulationskante, weil Knoten 3 als direkten Sohn Knoten 6 hat, von dem aus kein Knoten aus $V(2) = \{1\}$ erreichbar ist. Eine weitere Artikulationskante ist die Kante $(8,5)$, denn es gilt $N(8) = \{8\}$ und es gibt keine Kante von $V_5(8) = \{7\}$ nach $V(5) = \{1,2,3,4,6\}$.

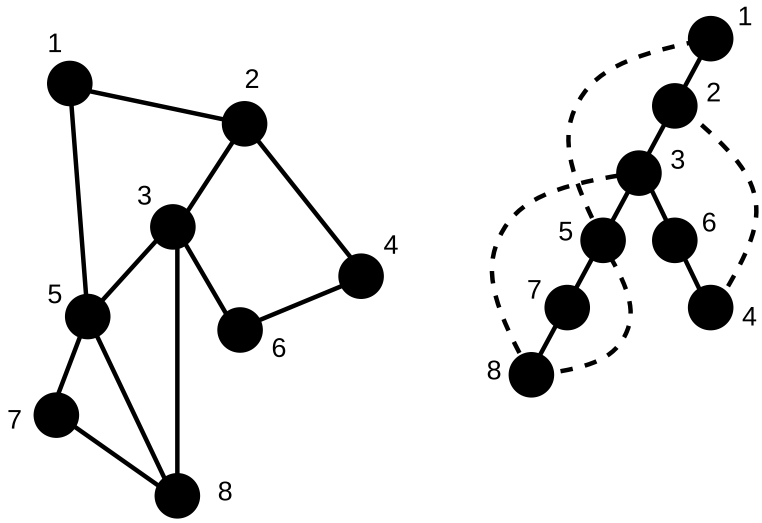

Abbildung 4.16: *Ein dünn besetzter Graph und seine Darstellung als Baum mit Rückwärtskanten (gestrichelte Linien).*

4.10 Zusammenfassende Beschreibung des Algorithmus

Dieses Kapitel soll durch eine ausführliche Beschreibung des Algorithmus zusammengefaßt werden. Die einzelnen Schritte, die in den vorigen Abschnitten detailliert beschrieben wurden, werden zu einem vollständigen Lösungsverfahren zusammengestellt.

TSP mit Iteriertem Matching

1. Initialisiere globale Variablen und Datenstrukturen.
2. Berechne gegebenenfalls Kandidatenmenge.
3. Berechne Pre-Benefits aller Kanten, speichere den Pre-Benefit der besten inzidierenden Kante für jeden Elementarknoten und den global besten Pre-Benefit.
4. Wiederhole
   i. Entferne Artikulationskanten.
   ii. Bestimme Kantengewichte der Clusterkanten ($c_{ij} = \text{benefit}(i, j)$) für Kante (i, j). Ist das Gewicht kleiner als q_{min}, setze $c_{ij} = 0$.
   iii. Berechne Maximum Weight Matching auf Clustergraphen.
   iv. Erzeuge aus gepaarten Clustern neue, größere Cluster.
   v. Verbinde neue Cluster, Kantengewichte sind zunächst die Pre-Benefits.
   vi. Vermindere die Selektionsrate um d_{sel}, $0 < d_{sel} \leq 1$

   bis nur noch zwei Cluster vorhanden.
5. Suche bestmögliche Verbindung der letzten beiden Cluster.

Das Initialisieren der globalen Variablen umfaßt in erster Linie das Setzen der Parameter für die Vorbereitung des Programmablaufs sowie den Ablauf selbst. Hier werden festgelegt:

- Einsatz von Kandidatenmengen
- Mindestgröße des Clustergraphen für Einsatz einer Kandidatenmenge
- Selektion und Selektionsrate
- Abnahmefaktor für die Selektionsrate
- Mindestqualität der Kanten
- Benefitfunktion
- Referenzfaktor der Benefitfunktion
- Globaler Einfluß bei der Berechnung des Benefits

Bei der grafischen Version des Programms werden außerdem die Parameter für die Darstellung gesetzt (siehe Kapitel 6).

Nachfolgend müssen alle Kanten mit dem Pre-Benefit versehen werden. Waren Kantengewichte explizit bei der Beschreibung der Probleminstanz gegeben, werden diese verwandt. Ansonsten muß für jede Kante eine Distanzfunktion ausgewertet werden. Diese ist durch die Beschreibung der Instanz festgelegt. Zur schnellen Berechnung der Benefits wird zum einen der beste aller Pre-Benefits gespeichert, zum anderen notiert man zu jedem Knoten den besten Pre-Benefit aller inzidierenden Kanten.

Für das Matching werden dann die Pre-Benefits der Kanten in die tatsächlichen Kantengewichte, die Qualität beziehungsweise der Benefit der Kante, transformiert. Dieses läßt sich mit der in Abschnitt 4.5 aufgestellten Formel für jede Kante in konstanter Zeit bewältigen. Auf dem aktuellen Clustergraphen wird dann ein Maximum Weight Matching bestimmt. Alle Knoten, die nun durch eine Kante des Matchings verbunden sind, werden zu einem neuen, größeren Cluster zusammengefaßt. Dabei werden unter allen möglichen Tourkombinationen diejenigen ausgewählt, die die Selektionskriterien erfüllen und mit dem Cluster abgespeichert.

Ist der Clustergraph noch so groß daß weiterhin die Kandidatenmenge eingesetzt werden soll, so werden die Kanten des Clustergraphen aus dem vorigen Iterationsschritt hergenommen und auf den neuen Graphen übertragen. Die Kanten werden zunächst mit den Pre-Benefits gewichtet, die sich aus den möglichen Kombinationen der neuen Cluster ergeben.

Ist eine Abnahme der Selektionsrate gewünscht — beispielsweise um zu Beginn des Verfahrens auch schlechtere Touren zu speichern, am Ende aber nur noch die guten — so geschieht dieses jetzt, bevor ein neuer Iterationsschritt mit der Berechnung der Benefits eingeleitet wird.

Sind nach einer gewissen Anzahl Iterationen nur noch zwei Cluster übrig, werden die Touren in diesen beiden hergenommen und alle möglichen Kombinationen durchgespielt. Die bestmögliche Kombination wird dann als Näherungslösung geliefert.

Zur Aufwandsanalyse sollen zunächst die einzelnen Schritte unabhängig voneinander betrachtet werden. Das Berechnen der Kandidatenmenge benötigt einen Aufwand von schlimmstenfalls $O(n^2)$. Die Ermittlung der ersten Pre-Benefits geschieht ebenfalls in $O(n^2)$, da jedes Knotenpaar genau einmal untersucht wird.

Die folgenden Schritte werden in einer Schleife durchgeführt. Innerhalb dieser Schleife verringert sich die für den Aufwand maßgebliche Zahl k an Clustern. Die Berechnung der Benefits geschieht für jede Kante in konstanter Zeit. Alle Benefits in einer Iteration werden also in $O(k^2)$ bestimmt. Das Matching-Verfahren hat einen Aufwand von $O(k^3)$.

Der Aufwand für das Erzeugen der neuen Cluster aus den Paarungen ist abhängig von der gewählten Selektion. Bei relativer Selektion wird der Aufwand im schlimmsten Fall exponentiell, da je nach Einstellung der Parameter alle Tourkombinationen gespeichert werden. Diese müssen im nächsten Schritt wieder miteinander kombiniert werden. Bei absoluter Selektion dagegen ist die Anzahl an Touren pro Cluster auf einen konstanten Wert c beschränkt. Daher müssen in einem Iterationsschritt für ein Clusterpaar maximal $2c^2\lfloor\frac{n}{k}\rfloor^2$ Tourkombinationen berechnet werden. Daraus ergibt sich ein Gesamtaufwand für diesen Schritt von $O(\frac{n^2}{k})$.

Die Verbindung der Cluster geschieht durch Analyse aller Kanten auf dem urspünglichen Graphen, benötigt also im schlimmsten Fall $O(n^2)$. Die Bewertung der Kanten durch die Pre-Benefits hängt allerdings von der Wahl der Benefitfunktion ab. Die Funktion Abstand der Schwerpunkte ist unabhängig von der Anzahl der gespeicherten Wege und Elementarknoten in einem Cluster. Daher kann die Bewertung für jedes Paar in konstanter Zeit geschehen, was einen Gesamtaufwand von $O(k^2)$ ergibt. Alle anderen Benefitfunktionen greifen auf die gespeicherten Wege eines Clusters zurück. Für die Bewertung einer Kante müssen also alle gespeicherten Touren der inzidierenden Cluster miteinander kombiniert werden. Wie bei der Konstruktion der Cluster bedeutet dieses also einen exponentiellen Aufwand bei relativer Selektion und einen Aufwand von $O(n^2)$ bei der absoluten Selektion.

Die Schleife wird im schlimmsten Fall n mal durchlaufen, nämlich dann, wenn bei jedem Matching nur zwei Knoten gepaart werden. Für die gesamte Schleife ist daher bei relativer Selektion exponentieller Aufwand nötig, bei absoluter Selektion ein Aufwand von

$$
\begin{array}{lll}
O\left(\sum_{k=1}^{n}(k^2+k^3+\frac{n^2}{k}+n^2+k^2)\right) = O(n^4) & \text{bei Benefitfunktion} \\
& \text{„Abstand der Schwerpunkte“} \\
O\left(\sum_{k=1}^{n}(k^2+k^3+\frac{n^2}{k}+n^2+n^2)\right) = O(n^4) & \text{sonst.}
\end{array}
$$

Der Aufwand für den letzten Schritt, die Verbindung der beiden letzten Cluster ist nur abhängig von der Anzahl der in den beiden Clustern gepeicherten Wege und Elementarknoten. Diese Verbindung läuft genauso ab, wie die Verbindung zweier Cluster in den einzelnen Iterationen. Daher gilt auch für den Aufwand die gleiche Aussage.

Zusammenfassend hat das besprochene Verfahren also bei relativer Selektion im worst case einen exponentiellen Aufwand, bei absoluter Selektion einen Aufwand von $O(n^4)$. Im wesentlichen trägt die Berechnung des Matchings zu diesem Aufwand bei. Diese ist aber bei Einsatz einer Kandidatenmenge sehr viel schneller, als es die aymptotische Laufzeit von $O(n^3)$ erwarten läßt.

Kapitel 5

Ergebnisse und Vergleich

Wie man im letzten Kapitel sehen konnte, sind eine Reihe von unterschiedlichen Parametereinstellungen für die vorgestellte Heuristik möglich. Durch unterschiedliche Einstellungen werden sehr verschiedene Ergebnisse erzielt. Welchen Einfluß die Parameter auf die Qualität und die Laufzeit des Verfahrens haben, soll nachfolgend untersucht werden.

Anschließend soll eine Einordnung des Verfahrens in die Reihe der in Kapitel 3 vorgestellten, bisher bekannten Heuristiken vorgenommen werden.

Alle gemessenen Zeiten sind auf einer Sparc 10 Workstation mit Hilfe der Funktion `getrusage()` ermittelt worden. Es wurden zum Teil Messungen wiederholt, um Meßfehler gering zu halten. Dabei ergaben sich Abweichungen der Ergebnisse um etwa 5%.

5.1 Einfluß der Parameter auf die Ergebnisse

Die vorgestellte Heuristik wird durch eine Reihe von Parametern beeinflußt. Viele dieser Parameter haben ein großes Spektrum an möglichen Werten. Es ist daher nicht möglich, eine Einstellung zu wählen, mit der für jede Instanz die bestmögliche Näherungslösung gefunden wird.

5.1.1 Einsatz der Kandidatenmengen

Der Einsatz von Kandidatenmengen wurde in erster Linie mit einer zu erwartenden Verbesserung der Laufzeit begründet. Dieses spiegelt sich in Abbildung 5.1 wider. Während ohne Kandidatenmengen die Laufzeit kubisch mit der Anzahl der Knoten wächst, ist bei Einsatz einer Kandidatenmenge (hier Delaunay Graph und 10-Nearest-Neighbour Teilgraph) nur ein lineares Wachstum zu beobachten. Der Grund liegt in der Anzahl der zu bewertenden Kanten. Bei einem vollständigen Graphen K_n gibt es $n \cdot (n-1)$ Kanten. Dagegen hat jeder Knoten in der hier eingesetzten Kandidatenmenge durchschnittlichen Grad 15. Damit gibt es im gesamten Graphen nur rund $15n$ Kanten. Ein weiterer Grund für die

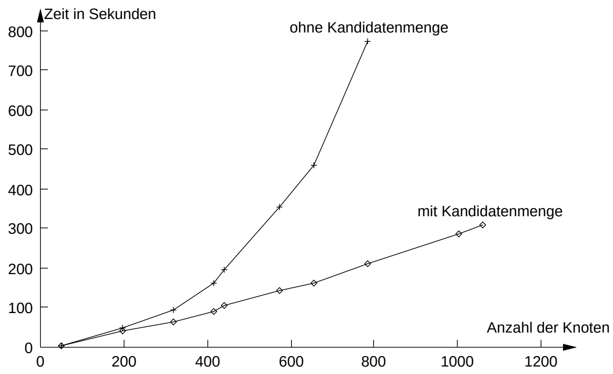

Abbildung 5.1: *Vergleich der Laufzeiten gleicher Instanzen mit und ohne Kandidatenmenge bei identischer Wahl aller anderen Parameter*

Kandidatenmengen ist die Tatsache, daß das eingesetzte Programm zur Berechnung des Matchings einen großen Speicherbedarf hat. Die Instanz *rat783* war die größte, die auf dem eingesetzten Rechner ohne Kandidatenmenge untersucht werden konnte.

Die Lösungsqualität sollte durch die Reduktion der Kantenzahl nicht zurückgehen. Auch dieses ist sichergestellt, wie Tabelle 5.1 zeigt. Wie in Abschnitt 4.6 erläutert, wird bei einer vorher festgelegten Größe des Clustergraphen die Kandidatenmenge „abgeschaltet". Dadurch ist es möglich, auch Instanzen zu lösen, bei denen nur eine k-Nearest-Neighbour Kandidatenmenge eingesetzt wurde, die einen nicht-zusammenhängenden Graphen darstellt. Ein Test mit dem Delaunay Graph, einem 10-Nearest-Neighbour Subgraph und der Kombination beider lieferte für unterschiedliche Probleminstanzen identische Lösungen.

Die Berechnung der Kandidatenmenge selbst, ist mit keinem großen Rechenaufwand verbunden, wie in Abbildung 5.2 zu sehen ist.

5.1.2 Wahl der Benefitfunktion

Die Wahl der Benefitfunktion beeinflußt nicht nur die Qualität der Lösung, sondern auch die Laufzeit (siehe Abschnitt 4.5). In geometrischen Graphen ist im Hinblick auf die Laufzeit die Benefitfunktion „Abstand der Schwerpunkte" die beste Wahl, da dann der Benefit einer Kante in konstanter Zeit aus den Pre-Benefits ermittelt werden kann. Die Laufzeiten

Anzahl Knoten	mit	ohne
	Kandidatenmenge	
51	11.3	11.3
198	7.5	7.5
318	16.6	14.2
417	13.1	13.1
442	13.5	13.5
574	17.0	17.0
654	16.4	16.4
783	13.3	13.3

Tabelle 5.1: *Vergleich der Qualität der Lösungen bei den Probleminstanzen aus Abbildung 5.1. Der Einsatz der Kandidatenmenge erhöht die Kosten der Tour nur in einem Fall.*

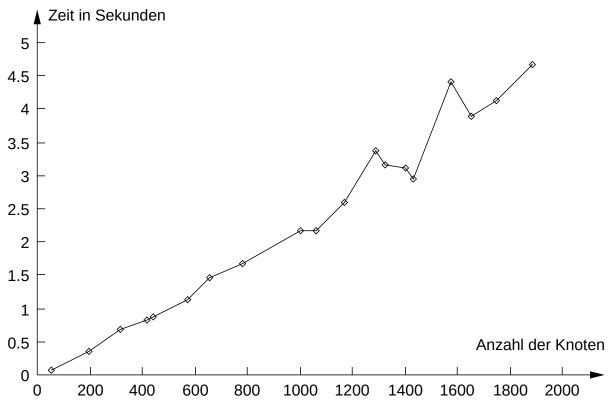

Abbildung 5.2: *Laufzeit für die Berechnung des Delaunay Graphen und des 10-Nearest-Neighbour Teilgraphen*

der anderen Benefitfunktionen sind abhängig von der Anzahl der gespeicherten Touren und/oder der Anzahl der Elementarknoten der beteiligten Cluster.

In Abbildung 5.3 sieht man recht deutlich, daß die Berechnung des Benefits aufgrund der Abstände der Schwerpunkte zweier Cluster ein gute Wahl ist. Die Lösungsqualität ist bei dieser Wahl grundsätzlich hoch. Sie wird nur in Einzelfällen geringfügig übertroffen. Die anderen Benefitfunktionen existieren in je zwei Varianten. Bei der einen Variante werden alle Touren der beiden Endcluster einer Kante bei der Berechnung des Benefits berücksichtigt, bei der anderen nur eine feste Zahl k. Diese wurde in diesem Fall auf $k = 3$

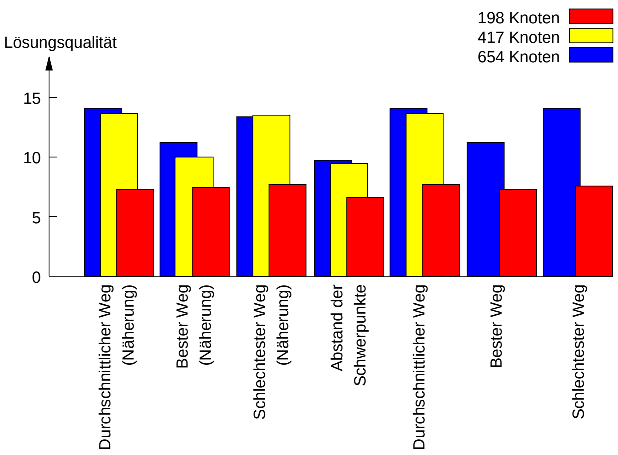

Abbildung 5.3: *Abhängigkeit der Lösungsqualität von der Wahl der Benefitfunktion*

gesetzt. Die Näherungsvarianten liefern dennoch etwas bessere Lösungen. In allgemeinen Graphen — in denen keine Schwerpunkte existieren und somit die Funktion „Abstand der Schwerpunkte„ nicht eingesetzt werden kann — sollte also die Näherungsvariante der Benefitfunktion „Bestmögliche Kombination“ gewählt werden.

Abbildung 5.4 zeigt die Laufzeiten für die verschiedenen Benefitfunktionen. Diese wurden ermittelt für die Probleminstanzen *lin318*, *pcb442* und *p654*. Es ist zu erkennen, daß die Laufzeit nicht nur von der Anzahl der Knoten in einem Graphen abhängig ist. So wird die Instanz *pcb442* beispielsweise in drei von vier Fällen schneller abgearbeitet, als die Instanz *lin318*.

Die schnellere Bearbeitung einer Instanz mit der Benefitfunktion „Abstand der Schwerpunkte“ wird auch hier ersichtlich. Während für die Instanz *p654* bei Einsatz dieser Benefitfunktion eine Gesamt-Laufzeit von etwa 150 s gemessen wurde, ergaben sich bei den anderen Funktionen Laufzeiten von etwa 400 s. Hierbei wurden nur die Varianten der Funktionen eingesetzt, die bei der Berechnung des Benefits auf die ersten drei Touren beider Cluster zurückgreifen. Werden alle Touren berücksichtigt, sind noch erheblich höhere Laufzeiten zu beobachten.

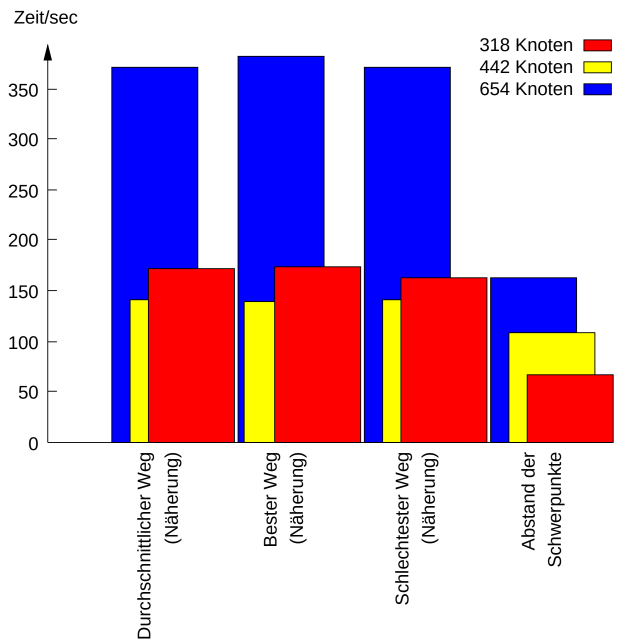

Abbildung 5.4: *Laufzeiten bei verschiedenen Benefitfunktionen*

5.1.3 Einfluß der Benefit-Optionen

Großen Einfluß auf die Bestimmung eines Matchings haben die Parameter, mit denen die gewählte Benefitfunktion eingestellt wird. Dieses sind im einzelnen die Mindestqualität der Kanten q_{min}, der Referenzfaktor f_{ref} und der globale Einfluß g der Pre-Benefits bei der Berechnung der Benefits.

Der globale Einfluß sollte möglichst niedrig eingestellt sein, wie die Diagramme in Abbildung 5.5 zeigen. Im Durchschnitt über alle Instanzen erhält man bei $g = 0,05$ das beste Ergebnis. Zwar ist in den Abbildungen zu erkennen, daß sich hier nicht immer ein absolutes Minimum einer Messung befindet. Es ist aber auch ersichtlich, daß an keiner anderen Stelle für alle Instanzen gleichzeitig ein besseres Ergebnis gefunden wird.

Ähnliche Ergebnisse wie in diesen Abbildungen wurden auch für andere Einstellungen der Mindestqualität und des Referenzfaktors sowie für andere Instanzen erzielt. Leider sind keine Gemeinsamkeiten in den einzelnen Kurvenverläufen zu erkennen, so daß in Einzelfällen eine sonst gute Wahl des Wertes g zu schlechten Ergebnissen führen kann.

Die Diagramme in Abbildung 5.6 zeigen die Abhängigkeit der Lösungsqualität vom Referenzfaktor. In beiden Abbildungen ist zu sehen, daß gute Ergebnisse bei einem Referenzfaktor von $f_{ref} = 1,75$ erzielt werden. In nur einem Fall gibt es eine bessere Lösung, diese

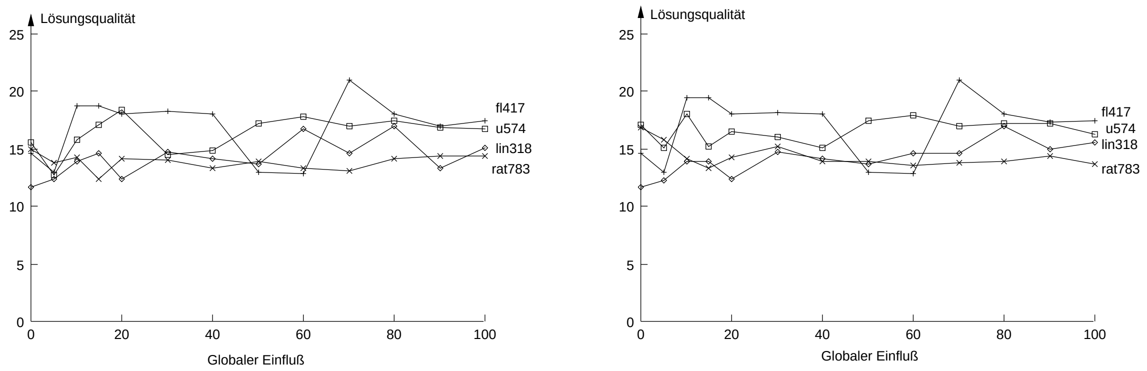

Abbildung 5.5: *Lösungsqualität in Abhängigkeit vom globalen Einfluß des besten Pre-Benefits bei* $q_{min} = 0,5$ *und* $f_{ref} = 1,5$. *(links) sowie* $q_{min} = 0,75$ *und* $f_{ref} = 1,5$ *(rechts).*

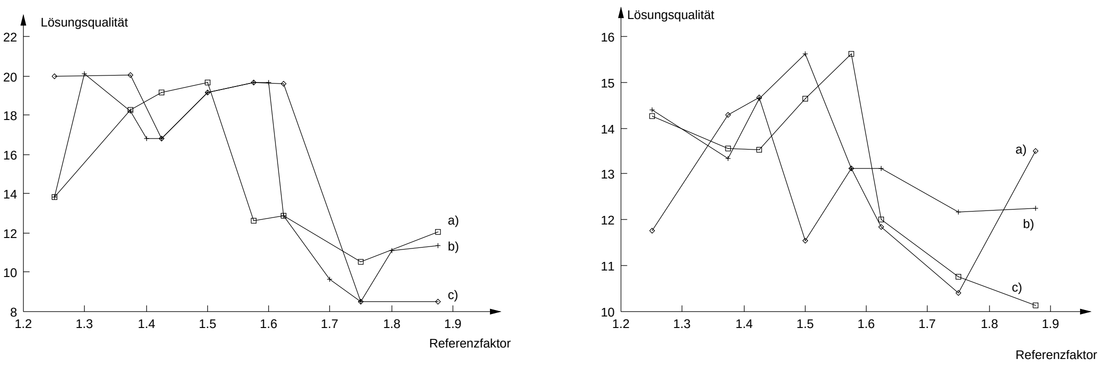

Abbildung 5.6: *Abhängigkeit der Lösungsqualität vom Referenzfaktor bei der Probleminstanz* fl417 *und a)* $q_{min} = 0,15$, $g = 0,05$, *b)* $q_{min} = 0,25$, $g = 0,1$, *c)* $q_{min} = 0,1$, $g = 0,1$. *(links) und der Probleminstanz* lin318 *bei einer Mindestqualität von* $q_{min} = 0,1$ *und einem globalen Einfluß von a)* $g = 0,05$, *b)* $g = 0,1$, *c)* $g = 0,15$. *(rechts)*

wird allerdings mit globalem Einfluß $g = 0,15$ erzielt. Dieses ist aber, wie oben erläutert keine besonders günstige Wahl.

Der Verlauf der einzelnen Kurven ist hier sehr viel ähnlicher als dies beim globalen Einfluß der Fall war. Es ist daher davon auszugehen, daß mit $f_{ref} \in [1,6;1,8]$ im allgemeinen für alle Instanzen gute Ergebnisse erzielt werden.

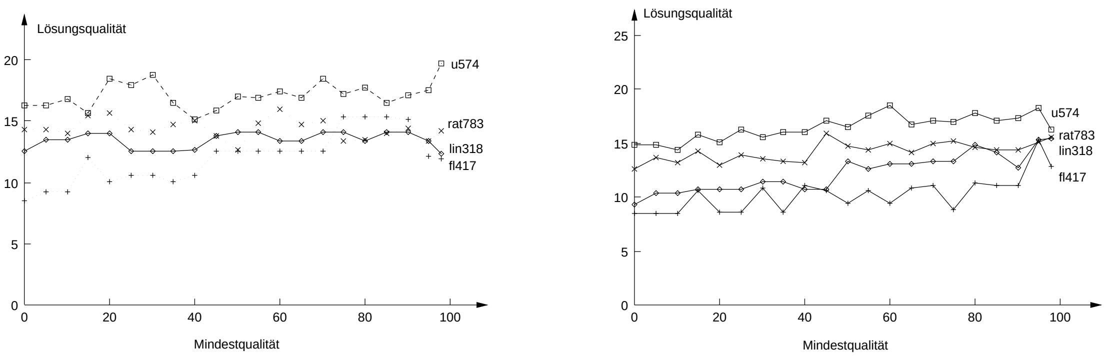

Abbildung 5.7: *Abhängigkeit der Lösungsqualität von der Mindestqualität bei $f_{ref} = 1,875$ und $g = 0,05$ (links) sowie bei $f_{ref} = 1,75$ und $g = 0,05$ (rechts).*

Bei der Mindestqualität lassen sich weniger gut Aussagen über eine geeignete Wahl machen. Die beiden Diagramme in Abbildung 5.7 zeigen für die dargestellten Instanzen einen sehr flachen Kurvenverlauf mit leicht ansteigender Tendenz bei zunehmender Mindestqualität. Daher erscheint die Wahl von $q_{min} \in [0,05; 0,2]$ gute Resultate zu liefern. Wie beim globalen Einfluß kann es aber auch hier in Einzelfällen zu schlechten Lösungen kommen.

5.1.4 Einfluß der Selektion

Ein ganz wesentlicher Punkt ist neben der Wahl der Benefitfunktion auch die Art der Selektion und die Selektionsrate. Bei den Messungen stellte sich heraus, daß die relative Selektion keine gute Wahl darstellt. Als Nachteile waren hier sehr viel größere Laufzeiten und erheblicher Speicherbedarf festzustellen. Beispielsweise wurde bei der Probleminstanz *d198* bei einer initialen Selektionsrate von 20% und einem Abnahmefaktor von 0.5 eine Laufzeit von 1 500 s gemessen. Bei absoluter Selektion (Selektionsrate 15) wurden dagegen nur knapp 40 s benötigt. Die Testinstanz *lin318* mußte abgebrochen werden, da nicht ausreichend Speicherplatz zur Verfügung stand.

Die Messungen wurden daher durchweg bei absoluter Selektion durchgeführt. Den Einfluß der Selektionsrate auf die Lösungsqualität zeigt Abbildung 5.8. Die Lösungsqualität nimmt für alle Instanzen deutlich zu, wenn die Selektionsrate — also die Anzahl maximal gespeicherter Wege in einem Cluster — von 1 auf 3 erhöht wird. Eine weitere, allerdings geringere Verbesserung ergibt sich, werden bis zu sieben Touren pro Cluster abgelegt. Weitere Steigerungen der Selektionsrate bringen im allgemeinen nur wenig, können in Einzelfällen allerdings für drastische Verbesserungen sorgen, wie am Beispiel der Instanz *p654* zu sehen ist. Interessant ist die Beobachtung, daß sich die Qualität einer Lösung verschlechtern kann, wenn sich die Selektionsrate erhöht.

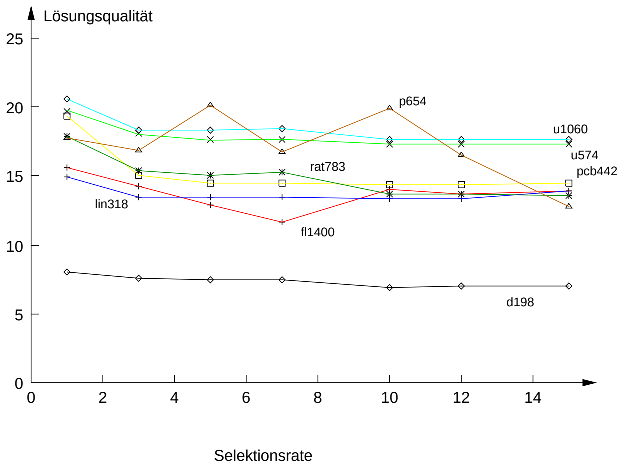

Abbildung 5.8: *Lösungsqualität in Abhängigkeit von der Selektionsrate*

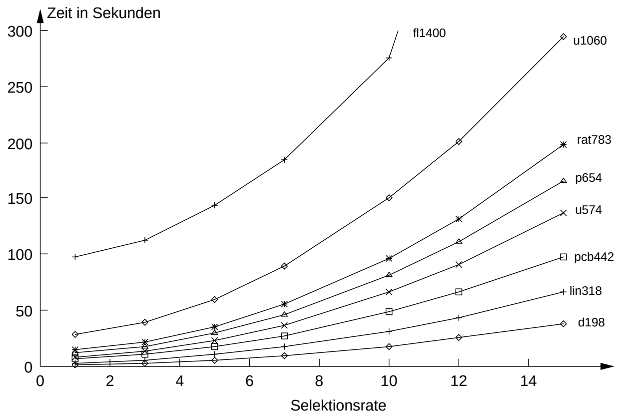

Abbildung 5.9: *Laufzeit in Abhängigkeit von der Selektionsrate*

Abbildung 5.9 zeigt, wie sich mit der Anzahl der gespeicherten Wege die Laufzeit erhöht. Es ist zu sehen, daß die Selektionsrate die Laufzeit mehr beeinflußt, als die Größe der Probleminstanz (siehe auch Abbildung 5.1). Sollen Lösungen also möglichst schnell geliefert werden, sollte die Selektionsrate auf Werte zwischen 2 und 5 eingestellt werden. Für gute Näherungslösungen sollten Werte zwischen 10 und 15, gegebenfalls auch höher eingestellt werden.

5.2 Ergebnisse im Vergleich zu anderen Heuristiken

Um das vorgestellte Verfahren objektiv mit anderen Heuristiken vergleichen zu können, wurde die Savings-Heuristik implementiert. Anhand dieser Implementierung konnten die Laufzeitmessungen aus [Rei 1994] überprüft werden. Sie wurde gewählt, weil ihr Verfahren Ähnlichkeiten zum iterierten Matching aufweist. Beide Verfahren gehören zu der Klasse der konstruktiven Heuristiken, beide setzen aus Teiltouren immer größere Touren zusammen und bei beiden bewertet eine Funktion die Qualität einer potentiellen Kombination.

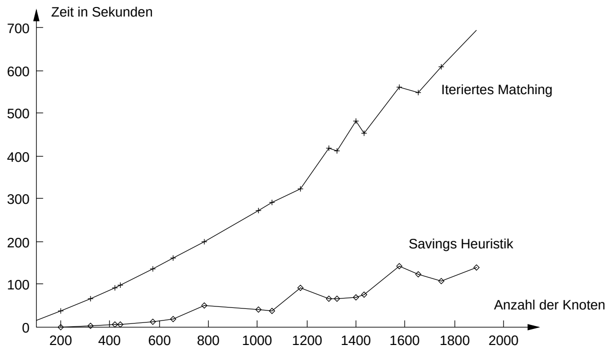

Abbildung 5.10: *Vergleich der Laufzeiten der Savings-Heuristik und des Iterierten Matchings*

Bei der Savings-Heuristik wurde auf den Einsatz einer Kandidatenmenge verzichtet, obwohl diese bei erheblich besserer Laufzeit keine Minderung der Lösungsqualität bedeutet hätte [Rei 1994]. Abbildung 5.10 zeigt die Laufzeiten für die Savings-Heuristik und das Iterierte Matching. Die Savings-Heuristik ist hier überlegen, obwohl sie zu den langsamsten der in Kapitel 3 vorgestellten Verfahren gehört.

Die Tabelle 5.2 gibt einen Überblick über die Lösungsqualität einzelner Probleminstanzen bei ausgewählten Verfahren. Waren bei einer Heuristik mehrere Varianten untersucht worden, wurde die jeweils beste in die Tabelle aufgenommen. Es ist zu erkennen, daß das Verfahren mit iteriertem Matching keine mit den Ergebnissen der Savings-Heuristik oder des Greedy Algorithmus vergleichbaren Resultate liefert. Da die Laufzeit des Verfahrens höher ist als die der anderen Verfahren, ist es keine Alternative zu den bekannten Heuristiken.

Instanz	N.-Neighbour	Insertion	Christofides	Savings	Greedy	It. Matching
d198	20.88	8.17	15.67	10.69	6.69	6.47
lin318	15.90	9.18	18.42	7.76	14.13	11.83
fl417	25.36	3.29	24.52	12.82	9.48	8.50
pcb442	13.51	12.23	18.59	8.61	14.62	15.00
u574	18.67	11.64	20.08	9.92	10.20	14.82
p654	25.98	9.87	21.73	14.35	18.82	12.87
rat783	18.86	13.37	21.34	9.15	10.04	13.24
pr1002	18.16	12.50	20.67	12.27	12.96	14.11
u1060	24.14	11.43	18.97	11.00	12.16	16.13
pcb1173	18.09	16.58	18.77	10.94	13.66	17.65
d1291	17.33	22.13	24.31	12.43	9.89	20.37
rl1323	16.81	20.64	14.05	10.73	8.13	28.75
fl1400	30.23	8.47	22.10	11.52	16.30	16.46
u1432	19.21	12.63	24.05	8.32	15.04	11.33
fl1577	23.30	18.70	13.27	14.17	8.63	36.84
d1655	19.81	17.69	18.29	12.34	10.49	17.36
vm1748	19.11	13.87	21.73	11.98	10.49	18.59
rl1889	25.62	17.30	14.00	11.59	10.82	31.03
Durchschnitt	20.61	13.32	19.52	11.14	11.81	17.30

Tabelle 5.2: *Vergleich der Qualität des Verfahrens mit iteriertem Matching mit verschiedenen anderen konstruktiven Heuristiken. Die Ergebnisse wurden abgesehen von denen der Savings-Heuristik [Rei 1994] entnommen.*

Kapitel 6

Beschreibung der Benutzeroberfläche

Wie in den vorangegangenen Kapiteln deutlich wurde, läßt sich das in dieser Arbeit vorgestellte Verfahren durch eine Vielzahl von Parametern steuern. Eine übersichtliche Art, diese Parameter einzustellen, bietet eine graphische Benutzeroberfläche. Sie ermöglicht außerdem, das Fortschreiten des Algorithmus zu beobachten, also eine Veranschaulichung des Verfahrens.

Neben einer rein textorientierten Fassung des Programms wurde daher eine Version für X-Windows implementiert. Beide Versionen nutzen den gleichen Kern, so daß nur die Kommunikation mit dem Benutzer getrennt programmiert werden mußte. Auf diese Weise ist die Anpassung an andere Umgebungen möglich, indem nur die wenigen Routinen der Benutzerschnittstelle angepaßt werden.

Für X-Windows wurde die Benutzeroberfläche mit Hilfe von Tcl/Tk gestaltet. Um möglichst systemunabhängig zu bleiben, wurde auf die Nutzung besonderer Fähigkeiten dieses Toolkits verzichtet. Im Endeffekt löst jede Auswahl eines Menüpunktes nur den Aufruf einer C-Funktion aus. Umgekehrt werden aus C-Funktionen heraus einfach Tcl/Tk-Kommandos aufgerufen, die beispielsweise das Zeichnen einer Linie ermöglichen. Diese C-Funktionen sind die einzigen, die bei einer Anpassung an andere Systeme geändert werden müßten.

6.1 Menüführung

Nach Aufruf des Programms erscheint eine Oberfläche wie in Abbildung 6.1 dargestellt. Oben befindet sich die Hauptmenüleiste, darunter die Darstellungsfläche und ganz unten die Zeile mit den Statusinformationen. Rechts neben der Darstellungsfläche sind Einstelloptionen zu finden, mit denen die Darstellung des Graphen beeinflußt werden kann.

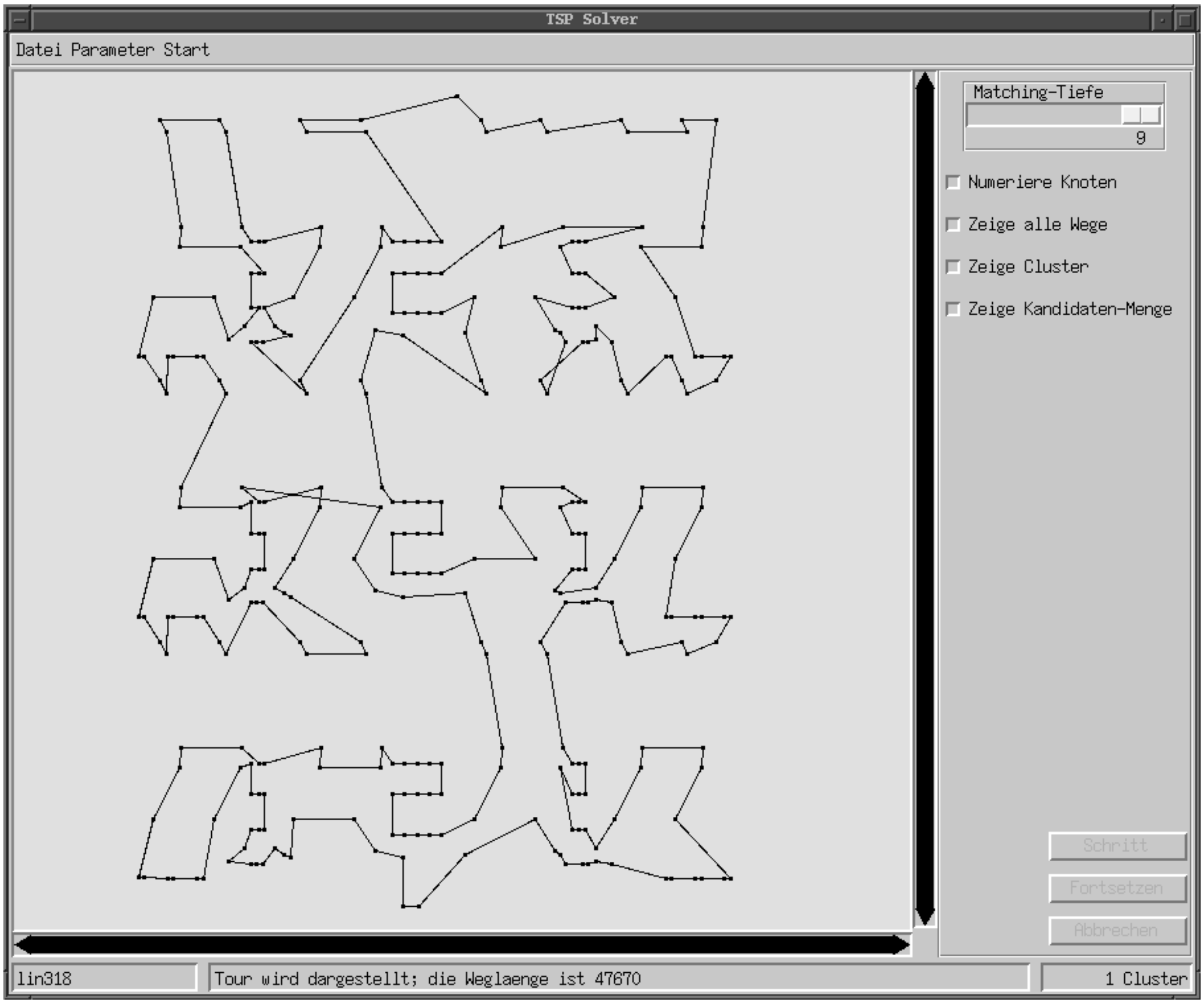

Abbildung 6.1: *Die Benutzeroberfläche*

In der Menüleiste erkennt man die Punkte *Datei*, *Parameter* und *Start*. Diese enthalten die folgenden Unterpunkte:

Menüpunkt Datei:

Öffnen: zum Einlesen einer Problembeschreibung

Speichern: zum Speichern eines Ergebnisses

Screen Dump: speichert den Inhalt der Darstellungsfläche als Postscript Datei

Beenden: zum Beenden des Programms.

Menüpunkt Parameter:

- Vorbereitung: Auswahl von Kandidatenmengen und der zugehörigen Parameter
- Programmablauf: Einstellen der den Programmablauf beeinflussenden Faktoren (Benefit-Funktion, Selektion, Selektionsrate etc.)
- Statistik: Abfrage statistischer Informationen nach Ablauf des Programms.

Menüpunkt Start:

- Berechne Kandidatenmenge: berechnet nur die Kandidatenmenge und wartet anschließend auf weitere Instruktionen vom Benutzer
- Starte Programm: startet die Lösungssuche auf dem geladenen Graphen via Iteriertem Matching
- Schrittweise: wie *Starte Programm*, jedoch wird nach jedem Matching die aktuelle Clustermenge angezeigt und auf weitere Instruktionen des Benutzers gewartet
- Vergleichsverfahren starten: berechnet eine Lösung mit Hilfe der Savings Heuristik
- Verifiziere: berechnet die Kosten der kanonischen Tour $(1, 2, 3, 4, \ldots, n, 1)$ durch einen Graphen. Dient zur Verifikation der Distanzfunktionen.

Mit den Optionen am rechten Rand kann die Darstellung des Graphen und der Lösung beeinflußt werden. Mit dem *Slider* „Matching-Tiefe“ ganz oben kann — nachdem bereits eine Lösung gefunden und dargestellt wurde — jede einzelne Ebene des Lösungsbaumes dargestellt werden. Dabei erhalten entgegen der üblichen Konvention die Blätter des Baumes die Tiefe 0, die Wurzel entsprechend die Tiefe n. Dementsprechend wird bei der Einstellung k des Sliders die Menge an Clustern dargestellt, die sich nach dem k-ten Matching ergab.

Wird die Option „Numeriere Knoten“ eingeschaltet, wird zu jedem dargestellten Knoten seine Nummer angezeigt. Dieses macht nur für kleine Graphen oder große Vergrößerungsstufen (siehe unten) Sinn.

Ist die Option „Zeige alle Wege“ aktiviert, wird in einem Cluster nicht nur der beste gespeicherte Weg, sondern jeder in ihm notierte Weg angezeigt. Auch diese Option macht nur im Einzelfall Sinn, da die einzelnen Wege nicht unterschieden werden können.

Durch Anwählen der Option „Zeige Cluster“ kann jedes Cluster nach einem Matchingschritt durch ein farbiges Polygon unterlegt werden. Das Polygon umschließt dazu die Voronoi-Regionen aller im Cluster enthaltenen Elementarknoten. Im Regelfall stellen gleiche Farben auch gleiche Cluster dar. Stehen jedoch nicht ausreichend Farben zur Verfügung, werden auch unterschiedliche Cluster gleich gefärbt. Diese sind aber immer noch durch die angezeigten Wege voneinander unterscheidbar.

Möchte man sich die Kandidatenmenge ansehen, auf der die Lösung bestimmt wurde, so muß der Punkt „Zeige Kandidatenmenge“ gewählt werden.

Um einzelne Bereiche eines Graphen genauer betrachten zu können, steht eine Vergrößerungsfunktion zur Verfügung. Um diese zu aktivieren, muß mit der Maus ein Rechteck über den Bereich gezogen werden, der vergrößert dargestellt werden soll. Dieser Bereich wird dann auf die Größe der gesamten Darstellungsfläche ausgedehnt. Verzerrungen sind dabei zulässig. Um den Bereich zu markieren, ist die Maus zum oberen linken Punkt des gewünschten Bereiches zu bewegen, die linke Maustaste zu drücken und bei gedrückter Maustaste der untere rechte Punkt des Rechtecks anzusteuern. Das Loslassen der Maustaste aktiviert die Vergrößerung. In jeder Vergrößerungsstufe führt ein Drücken der rechten Maustaste zur Darstellung des Graphen in Originalgröße.

Wurde eine schrittweise Ausführung des Verfahrens gewählt, so erscheinen nach jedem Matching in der unteren rechten Ecke des Fensters drei Schaltflächen mit den Titeln *Schritt*, *Fortsetzen* und *Abbrechen*. Der Benutzer muß einen dieser *Buttons* anklicken, um entweder ein weiteres Matching durchzuführen, die Ausführung ohne weitere Unterbrechung fortzusetzen oder gänzlich abzubrechen.

In der untersten Zeile, der Statuszeile des Fensters finden sich einige Informationen zum Ablauf des Programms. Ganz links erscheint der Name der geladenen Probleminstanz, in der Mitte erscheinen Meldungen, welche Aktionen das Programm gerade ausführt und rechts steht die Anzahl der Cluster zum aktuellen Zeitpunkt.

6.2 Darstellung des Graphen und der Lösung

Ist der geladene Graph darstellbar, so wird er entsprechend der Größe des Fensters skaliert und angezeigt. Die einzelnen Knoten erscheinen als kleine schwarze Kreise, Kanten als durchgezogene Linien. Es werden nur die Kanten angezeigt, die zur Lösung des Problems oder zu einer Kandidatenmenge — falls deren Ansicht eingeschaltet ist — gehören.

Jede Zusammenhangskomponente stellt ein Cluster dar. Diese können zusätzlich farbig unterlegt werden, indem die Option „Cluster anzeigen“ eingeschaltet wird. Wird die engültige Lösung angezeigt, so werden die beiden letzten Cluster, aus denen diese Tour entstanden ist, farbig unterlegt.

Mit dem Slider „Matching-Tiefe“ kann der Lösungsbaum nachträglich betrachtet werden. Wird der Slider ganz nach rechts bewegt, so wird die vollständige Lösung angezeigt. Ist der Slider ganz links, werden die Elementarknoten gezeigt. Mit jeder Bewegung nach rechts erscheint eine weitere Ebene des Lösungsbaumes, also die Cluster, die in den entsprechenden Iterationen entstanden sind.

Mit den Darstellungsoptionen können, wie oben beschrieben, zusätzlich Darstellungsoptionen ein oder ausgeschaltet werden. Jede Auswahl wirkt sich unmittelbar auf die Darstellung aus.

6.3 Einstellung der Parameter

Abbildung 6.2: *Parametereinstellung bei der Grafischen Benutzeroberfläche*

Die verschiedenen Parameter werden bei der grafischen Version des Programms über einen eigenen Dialog eingestellt (Abb. 6.2). Im oberen Abschnitt finden sich sieben sogenannte *Radiobuttons*, die für die entsprechende Anzahl an Benefitfunktionen stehen. Darunter befindet sich links ein Abschnitt, in dem die Parameter für die Selektion eingestellt werden können. Zwischen absoluter und relativer Selektion wird durch Radiobuttons entschieden, die Selektionsrate nebst ihres Abnahmefaktors wird in zwei Editierfeldern festgelegt.

Rechts daneben finden sich weitere Editierfelder, in denen Angaben über die Kantenbewertungen und die Mindestgröße des Clustergraphen für die Kandidatenmenge gemacht werden können. Im obersten Feld wird die Mindestqualität der Kanten eingegeben, darunter der entsprechende Referenzfaktor, also der Faktor, mit dem der kürzeste Weg in einem Cluster multipliziert werden muß um genau die angegebene Mindestqualität zu erreichen.

Der Anteil des global besten Pre-Benefits eines Clustergraphen bei der Berechnung der Kantenqualitäten gibt man im dritten Feld ein. Der Wert im vierten Feld, der Anteil der lokalen Pre-Benefits, ergibt sich daraufhin automatisch. Die letzte Angabe legt fest, bei welcher Größe der Einsatz der Kandidatenmenge beendet ist.

Änderungen in diesem Dialog werden unmittelbar nach dem Bestätigen durch Klicken des *OK-Buttons* wirksam. Dadurch können bei schrittweiser Ausführung des Programms noch während des Programmablaufes Parametereinstellungen modifiziert werden.

Die Art der Kandidatenmenge wird mit einem eigenen Dialog bestimmt (Abb. 6.3). Ganz

Abbildung 6.3: *Dialog zur Einstellung einer Kandidatenmenge*

oben befindet sich ein Schalter, mit dem ausgewählt werden kann, ob überhaupt eine Kandidatenmenge erstellt werden soll. Wenn ja, kann mit den darunter stehenden Schaltern der Delaunay Graph sowie ein Nearest-Neighbour Subgraph eingeschaltet werden. Die Anzahl der nächsten Nachbarn wird im Editierfeld darunter eingegeben. Eingaben sind hier nur möglich, wenn der Nearest-Neighbour Subgraph eingeschaltet ist.

Wird in den Dialogen mit der linken Maustaste auf den *OK-Button* unten links geklickt, werden die Einstellungen an das Programm übergeben. Das Klicken des *Abbrechen-Buttons* unten rechts nimmt alle zuletzt vorgenommenen Änderungen zurück und beendet den Dialog. Wird in einem der Editierfelder die *Enter-Taste* gedrückt, entspricht dies einem Klick auf *OK*, das Betätigen der *Escape-Taste* bedeutet Abbrechen. Mit der *Tabulator-Taste* gelangt man durch die einzelnen Eingabefelder.

Die in der Abbildung angegebenen Werte sind die Voreinstellungen. Sie entsprechen in etwa den Werten, die nach den Ergebnissen, wie sie im letzten Abschnitt dargestellt wurden, sinnvoll erscheinen.

6.4 Abfrage statistischer Daten

Wurde Lösung ermittelt, können über den Menüpunkt *Parameter/Statistik...* Informationen zum Programmablauf erfragt werden. Es erscheint ein Fenster (Abb. 6.4), das neben den Kosten der ermittelten Lösung auch die Laufzeit über für einzelne Schritte der Heuristik enthält.

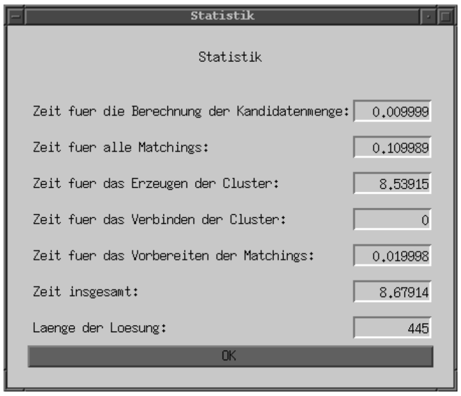

Abbildung 6.4: *Ausgabefenster für statistische Daten*

Die Zeitangaben beziehen sich grundsätzlich auf alle Berechnungen, die während des Programmablaufes gemacht wurden. Zeiten, die für Ausgaben auf den Bildschirm benötigt wurden sind nicht mit erfaßt.

Kapitel 7

Zusammenfassung und Ausblick

Mit dieser Arbeit wurde eine neue Heuristik für das TSP vorgestellt. Bevor die Ideen dieser Arbeit aufgezeigt wurden, sind zunächst andere, bereits bekannte Verfahren dargestellt worden. Es folgte eine Skizze der Anforderungen an die Verfahren in der Praxis. Die hier eingeführte Heuristik basiert auf iteriertem Matching. Dieses sorgt durch Paarung von Knoten zu Clustern dafür, daß sich die Größe eines zu lösenden Problems immer weiter reduziert, bis nur noch zwei Cluster miteinander verbunden werden müssen. Sie gehört daher zu der Klasse der konstruktiven Heuristiken.

Eine wesentliche Idee bei der Konstruktion der Cluster ist das Speichern mehrerer Hamilton-Touren auf deren Elementarknoten. Es stellte sich heraus, daß die Ergebnisse der Heuristik sich mit zunehmender Zahl an gespeicherten Wegen zunächst deutlich verbessern. Es konnte auch eine Obergrenze von 10 für diese Zahl angegeben werden. Wird diese überschritten, ist das Verhältnis von Qualitätsverbesserung zur Erhöhung der Laufzeit ungünstig.

Wichtige Steuermechanismen für die Berechnung eines Matchings stellen die Benefitfunktionen dar. Sie bestimmen das Gewicht beziehungsweise den Benefit einer Kante im Clustergraphen. Dieses ist abhängig von den in den Endknoten gespeicherten Elementarknoten und Hamilton-Touren, von einem sogenannten Pre-Benefit aller mit den Endknoten inzidierenden Kanten und gegebenenfalls von den Pre-Benefits anderer Kanten des Graphen. Der Pre-Benefit einer Kante ist dabei ein Wert, der — abhängig von der Wahl der Benefitfunktion — in etwa den entstehenden Tourkosten bei Verbindung ihrer Endknoten entspricht. Die besten Ergebnisse auf geometrischen Probleminstanzen ergaben sich mit einer Funktion, die im wesentlichen vom Abstand der Schwerpunkte zweier Cluster abhängt. Da sie unabhängig von den gespeicherten Wegen ist, ergeben sich bei ihr die mit Abstand besten Laufzeiten. Die Lösungsqualität ist dabei nicht nennenswert niedriger als bei Benefitfunktionen, die auch die gespeicherten Wege zur Berechnung der Kantengewichte heranziehen.

Verschiedene Parameter haben Einfluß auf die Ergebnisse einer Benefitfunktion. Der Benefit einer Kante (u, v) kann abhängig sein von den Pre-Benefits der Kanten, die mit u oder v inzidieren einerseits, von den Pre-Benefits aller anderen Kanten des Graphen andererseits. Im ersten Fall wird von lokalem Einfluß, im zweiten Fall von globalem Einfluß der Pre-Benefits untereinander gesprochen. Die besten Ergebnisse wurden bei geringem bis keinem globalen Einfluß erzielt. Ein weiterer Parameter für die Benefitfunktion ergibt sich aus einer geforderten Mindestqualität. Eine Kante, die nicht mindestens diese Qualität aufweist, wird mit 0 gewichtet. Sie wird dadurch keinesfalls in ein Matching aufgenommen. Hier ergaben sich gute Resultate bei der Forderung nach einer Mindestqualität von etwa 15%. Der letzte Parameter ist der sogenannte Referenzfaktor. Bei der Berechnung des Benefits einer Kante (u, v) ergibt dieser Faktor multipliziert mit dem bestmöglichen Pre-Benefit aller mit u oder v inzidierenden Kanten den Pre-Benefit, der der Mindestqualität entspricht. Eine gute Wahl dieses Wertes liegt bei etwa 1,75.

Abschließend folgte ein Vergleich mit anderen Heuristiken, speziell der Savings-Heuristik. Es ist festzustellen, daß die Savings-Heuristik in kürzerer Laufzeit bessere Ergebnisse als das iterierte Matching erzielt. Das gleiche gilt für den Greedy-Algorithmus. Das iterierte Matching ist daher in dieser Form keine Alternative zu den bekannten Verfahren. Ein Einsatz empfiehlt sich höchstens dann, wenn die Darstellung der Lösung als Baum für eine Nachoptimierung sinnvoll erscheint.

Zur Visualisierung der Arbeitsweise des Verfahrens sowie der gelieferten Lösungen des Verfahrens wurde eine grafische Oberfläche mit Hilfe von Tcl/Tk implementiert. Unter dieser sind alle beschriebenen Parameter einstellbar und die Suche nach einer Lösung kann Schritt für Schritt verfolgt werden. Während der Implementation des Verfahrens diente sie unter anderem dazu, Schwächen erster Ansätze aufzudecken.

Diese Arbeit kann Ausgangspunkt sein für weitere Arbeiten auf dem Gebiet des *Traveling Salesman Problem*. Beispielsweise könnte das Verfahren in einen *Backtracking*-Rahmen integriert werden. Dabei würden berechnete Paarungen schrittweise rückgängig gemacht werden, wenn sie sich im Laufe weiterer Iterationen als nicht geeignet erweisen. Dazu müßten Teiltouren während des Verfahrens beurteilt werden. Dies kann durch die Berechnung von unteren Schranken für eine Lösung auf gegebenen Clustergraphen geschehen.

Weiterhin bietet es sich an, das Verfahren zu parallelisieren. Dazu könnte der Graph einer Probleminstanz in verschiedene Teilgraphen unterteilt werden, die dann auf die zur Verfügung stehenden Prozessoren verteilt werden. Dort werden auf den Teilgraphen Näherungslösungen berechnet, die schließlich miteinander verbunden werden. Denkbar wäre auch, die Berechnung des Matchings und das Erzeugen der Cluster als zeitlich aufwendigste Aktionen des Verfahrens zu parallelisieren. Die Vorgehensweise hier ist aber nicht so offensichtlich, wie im ersten Fall.

Eine andere Möglichkeit, die Arbeit fortzusetzen, wäre eine Modifikation für das Vehicle Routing Problem, beziehungsweise Spezialfällen davon. Man würde dann nicht mehr fordern, genau eine Hamilton-Tour zu finden. Gesucht wäre dann eine Menge von maximal k

Touren, wobei k die Anzahl der zur Verfügung stehenden Fahrzeuge ist. Das vorgestellte Verfahren müßte dazu also nur früher abbrechen, wobei sicherzustellen ist, daß jede berechnete Tour auch eines einer gegebenen Menge an Depots enthält, von denen aus Ware verteilt wird. Mit Hilfe von Ergänzungen bei den Benefitfunktionen ließen sich weitere Kostenfaktoren aus der Praxis modellieren. Ein praktischer Einsatz des Verfahrens beispielsweise bei der Verwaltung eines Fuhrparks scheint daher möglich.

Literaturverzeichnis

[Bel 1974] M. Bellmore, S. Hong, „Transformation of Multisalesmen Problem to the Standard Traveling Salesman Problem“, Journal of the ACM 21, 1974 S. 400–504,

[Chr 1976] N. Christofides, „Worst Case Analysis of a New Heuristic for the Travelling Salesman Problem“, Research Report, Carnegie-Mellon University, Pittsburgh, 1976 in: G. Reinelt, „The Traveling Salesman“, Springer Verlag, Berlin Heidelberg 1994

[Cod 1992] B. Codenotti, L. Margara, „Efficient Clustering Techniques for the Geometric Traveling Salesman Problem“, International Computer Science Institute, Berkeley and Consiglio Nazionale dell Richerche, Pisa, Juni 1992

[Ebe 1981] J. Ebert, „Effiziente Graphenalgorithmen“, Akademische Verlagsgesellschaft, Wiesbaden, 1981

[Edm 1965] J. Edmonds, „Maximum Matching and a Polyhedron with 0,1-Vertices“, Journal of Research of the National Bureau of Standards B 69, 165, S. 125–130,

[Fri 1994] Andreas Fritsch, „Verschnittoptimierung durch iteriertes Matching“, Diplomarbeit, Universität Osnabrück, Januar 1994

[For 1987] Steve J. Fortune, „A Sweepline Algorithm for Voronoi Diagrams“, Algorithmica 2, 1987, S. 153–174

[Gab 1973] H. Gabow, „Implementation of Algorithms for Maximum Matching on Nonbibartite Graphs“, Ph.D. Thesis, Stanford University, 1973

[Gar 1977] R.S. Garfinkel, „Minimizing wallpaper waste, part I: a class of traveling salesman problems.“, Operations Research 25, 1977, S. 741–751

[Joh 1985] D.S. Johnson, C.H. Papadimitriou, „Computational Complexity“, in: E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, „The Traveling Salesman Problem“, Wiley-Interscience, Chichester, 1985, S. 37–85

[Kop 1994] H. Kopka, „LaTeX — Band 1: Einführung“, Addison-Wesley, Bonn, 1994

[Kop 1995] H. Kopka, „LaTeX — Band 2: Ergänzungen“, Addison-Wesley, Bonn, 1995

[Law 1985] Eugene L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, „The Travling Salesman Problem“, Wiley-Interscience, Chichester, 1985

[Lin 1973] S. Lin, Brian W. Kernighan, „An Effective Heuristic Algorithm for the Travelling Salesman Problem“, Operations Research 21, 1973, S. 498–516

[Mak 1993] K.-T. Mak, A.J. Morton, „A Modified Lin-Kernighan Traveling Salesman Heuristic“, Operations Research Letters 13, 1993, S. 127–132

[Ous 1994] John K. Ousterhout, „Tcl and the Tk toolkit“, Addison Wesley, Reading, Massachusetts, 1994

[Pot 1993] Jean-Yves Potvin, „Genetic Algorithms for the Traveling Salesman Problem“, Université de Montréal, September 1993

[Rei 1991] Gerhard Reinelt, „TSPX — A Software Package for Traveling Salesman Problem Solving, User's Guide“, Universität Augsburg, Dezember 1991

[Rei 1994] Gerhard Reinelt, „The Traveling Salesman, Computational Solutions for TSP Applications“, Springer Verlag, Berlin Heidelberg 1994

[Rei 1995] Gerhard Reinelt, „TSPLIB 95“, Universität Heidelberg, World-Wide-Web, `http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html`, 1995

[Ros 1977] D.J. Rosenkrantz, R.E. Stearns, P.M. Lewis, „An Analysis of Several Heuristics for the Traveling Salesman Problem“, SIAM Journal on Computing 6, 1977, S. 563–581

[Sah 1976] S. Sahni, T. Gonzales, „$\mathcal{P}$-complete Approximation Problems“, Journal of the ACM, 1976, S. 555-565

[Sie 1988] Hans-Christian Siegert, „EDV-Unterstützung im Fuhrpark“, Verlag moderne industrie, Landsberg am Lech, 1988

[Zie 1988] Hans-Jörg Ziegler, „Computerunterstützte Transport- und Tourenplanung“, expert-Verlag, Ehningen bei Böblingen, 1988

Danksagung

An dieser Stelle möchte ich mich bei Herrn Professor Dr. Oliver Vornberger bedanken, der mich während dieser Diplomarbeit betreut hat. Außerdem danke ich Herrn Dipl.-Inf. Volker Schnecke, Herrn Dipl.-Math. Frank Thiesing und Herrn Dr. Frank Lohmeyer für wichtige Hinweise und Anregungen sowie für moralische Unterstützung.

Erklärung

Ich versichere, daß ich diese Arbeit selbständig angefertigt habe und keine außer den genannten Hilfsmitteln und Quellen verwendet habe.

Osnabrück, den 12. März 1996

Frank Lauxtermann