4.1 Bekannte Verfahren

4.1.1 Roulette

Die einfachste 'Methode', die man vermutlich auch bei manueller Herangehensweise zuerst anwenden würde, ist, einige zufällig ausgesuchte Touren zu betrachten. Dieses Verfahren eignet sich natürlich allenfalls dazu, einen ersten Eindruck über einen günstigen Verlauf bzw. eine Ausgangstour für optimierende Heuristiken zu erhalten.

4.1.2 Nearest Neighbour

Eine ebenfalls naheliegende Vorgehensweise besteht darin, von der jeweiligen Stadt immer zur nächstliegenden noch nicht besuchten zu gehen. Das Problem bei diesem *Nearest-Neighbour-Verfahren* ist, daß durch die lokale Betrachtungsweise, bei der der globale Aufbau der Tour vernachlässigt wird, am Schluß nur noch lange 'Rückwege' übrigbleiben können. Dafür ist der Rechenaufwand vergleichbar gering, es müssen lediglich die Entfernungen zu den noch nicht besuchten Städten untersucht werden.

Die Ergebnisse des Algorithmus hängen vom Ausgangspunkt ab, bei dem man startet. Man sollte daher die Berechnung mit jeder Stadt als Startort durchführen und die beste Tour auswählen.

4.1.3 Nearest Addition

Beim *Nearest-Addition-Algorithmus* wird in eine bestehende Teiltour die Stadt J nach einem Ort M aus der Tour eingefügt, wobei im Gegensatz zur rein lokalen Ausrichtung des Nearest-Neighbour-Verfahrens J und M unter allen in Frage kommenden so gewählt werden, daß die Entfernung von M nach J minimal ist. Gestartet wird dabei mit einer aus einer Stadt bestehenden 'Rundreise'.

4.1.4 Insertion-Verfahren

Auch bei den *Insertion-Verfahren* wird die lokale Sichtweise des Nearest-Neighbour-Algorithmus zugunsten einer globaleren Vorgehensweise gelockert. Das Prinzip ist, eine Stadt in eine bestehende Tour so einzubauen, daß sich diese möglichst wenig verlängert. Die verschiedenen Varianten unterscheiden sich dabei darin, wie die Stadt ausgewählt wird.

Insertion

Im einfachsten Fall werden die hinzuzufügenden Orte in der Reihenfolge der Eingabe (also nach Nummern) und damit quasi zufällig ausgesucht. Da hierfür keine aufwendigen Routinen notwendig sind, ist das grundlegende Insertion-Prinzip im Programm gut zu erkennen.

Cheapest Insertion

Bei *Cheapest Insertion* wird dem Namen entsprechend die billigste Einfügung gewählt, d. h. es wird die Stadt ausgesucht, deren Einfügung jeweils die geringste Verlängerung der Tour bewirkt.

Nearest Insertion

Nearest Insertion sucht wie Nearest Addition die Stadt J heraus, deren Entfernung zu einem beliebigen Ort M der schon bestehenden Tour minimal ist, fügt diese dann aber nach dem Insertion-Prinzip ein.

Farthest Insertion

Farthest Insertion macht das genaue Gegenteil von Nearest Insertion: Als einzufügende Stadt wird diejenige gewählt, die die größtmögliche Entfernung zu allen Tourorten hat. Obwohl dies zuerst vielleicht unlogisch klingt, liefert der Algorithmus gute Resultate und zählt zumindest bei

metrischen Problemen zu den Standardverfahren, weswegen seine Ergebnisse auch als Bezugsgrößen in den Tabellen verwendet werden. Der Grund liegt darin, daß auch die weiter entfernten Orte irgendwann erfaßt werden müssen. Bei Nearest Insertion führt dies dazu, daß möglicherweise am Ende lange Strecken hinzugefügt werden. Die Auswahl der weitentfernten Orte direkt zu Beginn erzeugt hingegen erst ein grobe Rundreise über den ganzen abzudeckenden Bereich, die dann nachträglich verfeinert wird.

4.1.5 Nearest Merger

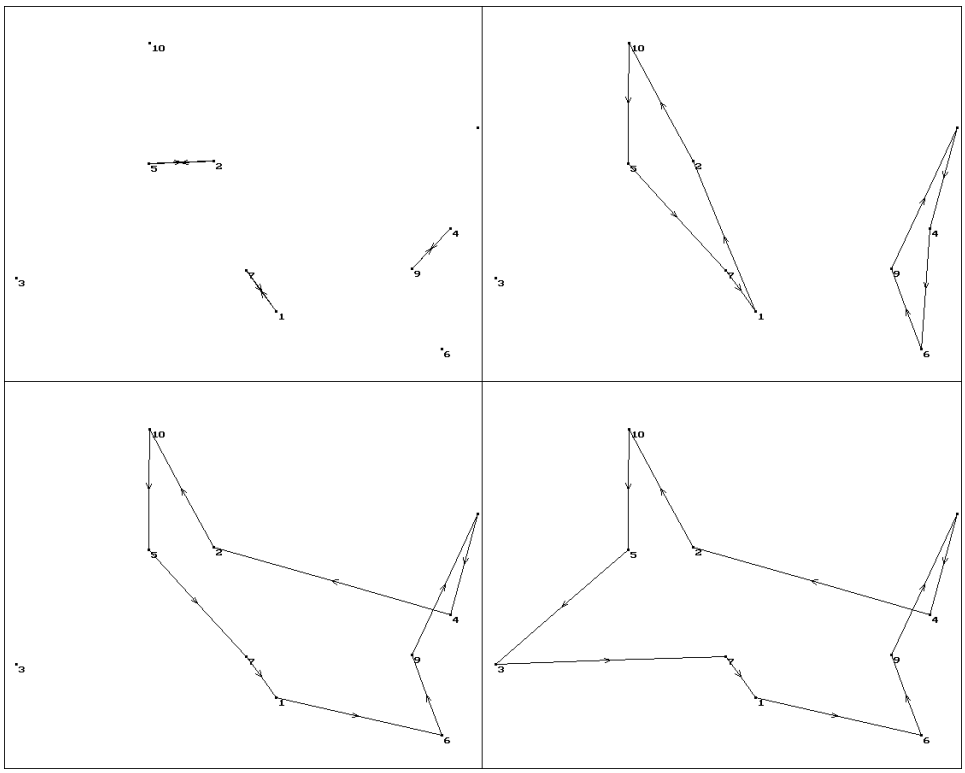

Abbildung 4.1: Touraufbau beim Nearest-Merger-Verfahren

Das *Nearest-Merger-Verfahren* beruht auf dem Verschmelzen von Teiltouren zu größeren Rundreisen, bis schließlich alle n Städte erfaßt werden, wobei man mit n aus je einem Ort bestehenden Wegen beginnt. In jedem Schritt werden die zwei Teiltouren I1 und J1 zusammengelegt, für die der Abstand zwischen einer Stadt aus der ersten und einer aus der zweiten minimal ist. Besteht einer der Rundwege aus nur einem Punkt, so wird dieser so in den anderen Weg eingebaut, daß sich dieser geringstmöglich verlängert. Umfassen beide jedoch mindestens zwei Knoten, so bestimmt man aus I1 zwei Städte I2 und J2 und aus Tour J1 K2 und L2 derart, daß

$$\text{ENTFTAB[I2,L2]+ENTFTAB[K2,J2]-ENTFTAB[I2,J2]-ENTFTAB[K2,L2]} \tag{4.1}$$

minimal ist. Das bedeutet, daß man beim darauffolgenden Zusammenfügen der Teilwege durch Entfernen der Kanten (I,J) und (K,L) sowie Hinzufügen von (I,K) und (J,L) eine möglichst geringe Verlängerung bzw. eventuell sogar eine (möglichst große) Verkürzung gegenüber der Summe der Länge der Einzelwege erhält. Abbildung 4.1 zeigt anhand von vier Zwischenschritten einen solchen Touraufbau.

Im Programm 4.8 werden die Teiltouren im Feld SUBWEG gespeichert, das in Zeile 11 mit 'Ein-Städte-Touren' initialisiert wird. Die Variable N enthält die Anzahl der noch bestehenden Teil-

touren, solange sie größer als eins ist, werden in den Zeilen 13-23 die zu verschmelzenden Teilwege I1, J1 bestimmt und in 25-32 die günstigste Verschmelzung ermittelt, die in 34/35 durchgeführt wird. Eine Sonderbehandlung von Ein-Städte-Reisen ist dabei nicht notwendig. Der neue Weg wird in SUBWEG[I1] gespeichert, während SUBWEG[J1] gelöscht und N um eins verringert wird. Ist N=1, dann ist die letzte erzeugte 'Teiltour' SUBWEG[I1] die gesuchte (Gesamt-)Tour.

Da im Nearest-Merger-Algorithmus kein Startpunkt angegeben werden muß, ist auch keine automatische Startpunktwahl notwendig, was zu geringen Rechenzeiten führt.

4.1.6 Minimal-Spanning-Tree-Rundreise

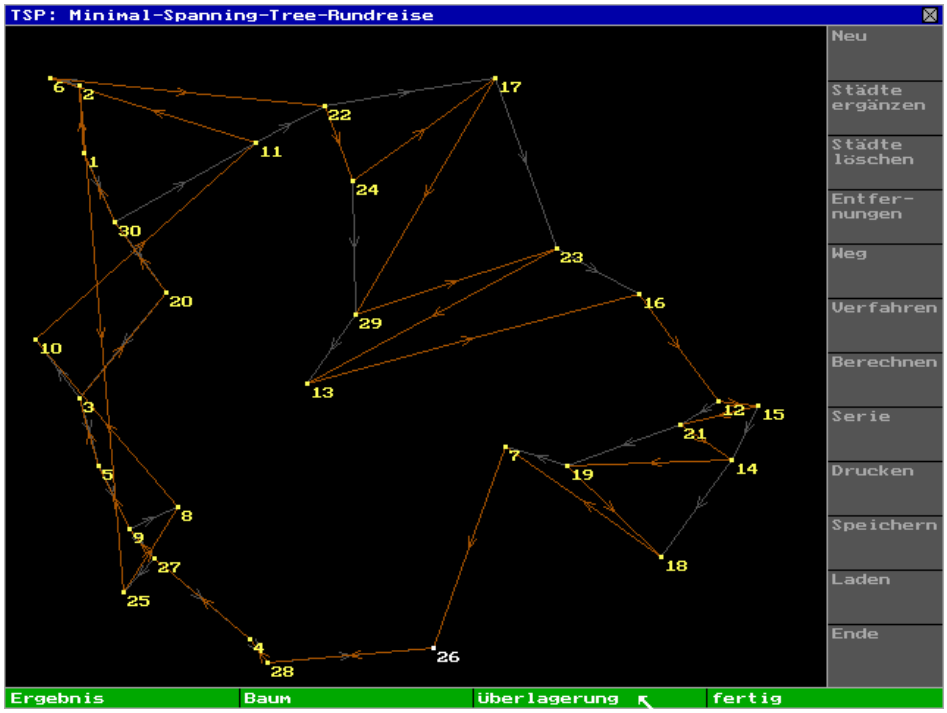

Abbildung 4.2: Überlagerung von MST und Rundreise

Bei den bisherigen Verfahren war es nicht möglich, eine Aussage über die Qualität der gelieferten Tour im Vergleich zur optimalen Lösung zu machen. Dies ist beim *Minimal-Spanning-Tree-Rundreise-Algorithmus* zumindest im metrischen Fall anders. Wie der Name schon andeutet, geht er vom Minimal Spanning Tree des Graphen aus: Verdoppelt man alle Kanten des MST, so erhält man eine Rundreise, die allerdings jede Stadt mindestens zweimal enthält. Um daraus eine Tour zu machen, kürzt man ab, d. h. man durchläuft den Rundweg von einer Anfangsstadt ausgehend, und jedes Mal, wenn man einen Ort zum zweitenmal besuchen würde, überspringt man ihn und geht direkt zum nächsten weiter.

Wie schon in Abschnitt 3.2 erläutert, stellt die Länge des MST eine untere Schranke für die Längen der Touren des zugehörigen TSP dar. Andererseits verringert sich, falls die Dreiecksungleichung gilt, die Länge der aus dem verdoppelten Minimal Spanning Tree entstandenen Rundreise durch das Abkürzen bzw. sie bleibt maximal gleich, d. h. die Länge des Endergebnisses beträgt höchstens das zweifache der Kosten des MST. Daraus folgt:

$$\frac{\text{Länge der Tour}}{\text{Länge der optimalen Tour}} \leq 2 \tag{4.2}$$

Das Minimal-Spanning-Tree-Rundreise-Verfahren liefert im metrischen Fall also eine Tour, die maximal doppelt so lang wie die optimalen Tour ist. Im nichtmetrischen Fall ist dies hingegen nicht

gewährleistet, da sich ohne die Dreiecksungleichung beim 'Abkürzen' die Rundreise auch verlängern kann.

Für die Erzeugung des Minimal Spanning Tree wird eine modifizierte Version der PRIM-Prozedur aus Programm 3.2 verwendet. PRIM_R verzichtet auf die Optimierung, durch die bei zwei aufeinanderfolgenden Kanten, bei denen der Endknoten der ersten mit dem Startknoten der zweiten übereinstimmt, die trennende Null und die doppelte Stadtnummer weggelassen wird. Der Eintrag einer Kante besteht also immer aus drei Zeichen, nämlich aus einer Null zur Trennung, dem Startort und der Nummer des Endortes, was die Berechnung der Rundreise in der Prozedur MSTR vereinfacht. Dort wird die Tour in der Variablen W vom Typ WEG aufgebaut, indem sie mit ANFANG initialisiert und durch wiederholten Aufruf von NAECHSTE erweitert wird, bis die Stringlänge der Stadtanzahl entspricht.

NAECHSTE übernimmt die zentrale Aufgabe des Algorithmus, ausgehend von der übergegebenen vorherigen Stadt N die nächste aus dem verdoppelten MST zu ermitteln und, falls notwendig, dabei Orte zu überspringen, also abzukürzen. Die Verdopplung der Kanten wird dabei nicht tatsächlich durchgeführt, sondern statt dessen jede in Hin- und Rückrichtung betrachtet: In den Zeilen 51-55 wird nach einer Kante gesucht, die N entweder als Start- oder als Endpunkt enthält und der jeweils andere Punkt als neues N verwendet. Dies wird solange wiederholt, bis man zu einem N kommt, das noch nicht in W enthalten ist. Aufgrund der Reihenfolge der Bearbeitung ist gewährleistet, daß im nächsten Durchlauf nicht die gleiche Kante in umgekehrter Richtung betrachtet wird, was eine Endlosschleife zur Folge hätte. Hat man schließlich einen noch nicht in W enthaltenen Wert gefunden, so wird dieser als nächste anzufügende Stadt an MSTR zurückgegeben.

Das Programm erlaubt es, neben der berechneten Rundreise auch den Ausgangs-MST sowie eine Überlagerung beider anzuzeigen, wie es in Abbildung 4.2 zu sehen ist.