Das Travelling-Salesman-Problem
oder
das Problem des Handlungsreisenden

Wissenschaftlich-Praktische-Arbeit

eingereicht von: Thomas Blaßkiewitz

bearbeitet im Zeitraum 2000 – 2001

am mathematischen Institut in der Abteilung für Optimierung
der Universität Leipzig

Betreuerin: Dr. rer. nat. Anita Kripfganz

Bibliographische Beschreibung der Arbeit und Referat

Thomas Blaßkiewitz

Es ist die Aufgabe, ein Programm zu erstellen, welches das Problem des Handlungsreisenden in einem vertretbaren Rechenaufwand exakt löst.

Umfang: 23 Seiten; Anlagen I-III,

Diskette 3,5″ (TRAVSAL3.EXE; DEUTLAND.MTX)

Referat

Im heutigen Zeitalter ist das Reisen unverzichtbar. Dabei kommt es nicht mehr darauf an, von einem Ort zum anderen zu gelangen, sondern diesen Weg zu optimieren.

Immer wieder trifft man auf das Problem des Handlungsreisenden, der sich am Ort 1 befindet und die Absicht hat, n-1 Orte zu besuchen. Dabei will er jede Ortschaft genau einmal durchreisen und zum Schluss wieder in den Ort 1 zurückkehren. Daraus ergibt sich, dass Ort 1 gleichzeitig der Start- und Zielort ist und die Orte 2,...,n-1 Durchgangsorte sind. Seinen Weg will er dabei nach den Gesichtspunkten Zeit, Entfernung oder Kosten optimieren.

Nun soll ein Programm erstellt werden, welches die günstigste Reihenfolge der Orte, die durchreist werden sollen, ermittelt.

Abstract

Travelling is indispensable in modern society. Nowadays the question is not to get from one place to another but how to optimize the way.

The problem is that of a commercial traveller who is at place 1 and intends to visit n-1 places. He wants to travel through each of these places exactly once and to go back to place 1 in the end. So place 1 is starting point as well as destination while places 2, ... , n-1 are transit places. The traveller wants to optimize the way according to time, distance or costs.

Now a programme is to be designed which determines the most effective sequence of all places through which the traveller passes.

Inhaltsverzeichnis

1. Verzeichnis der verwendeten Abkürzungen.....5

2. Einführung und Zielstellung.....6

3. Grundbegriffe aus der Graphentheorie.....7

3.1 Der Graph.....7

3.2 Knoten und Kanten.....7

3.3 Der parallele Graph.....8

3.4 Der schlichte Graph.....8

3.5 Der vollständige Graph.....8

3.6 Wege und Zyklen.....8
3.6.1 Kantenfolgen.....9
3.6.2 Die triviale Kantenfolge.....9
3.6.3 Der Kantenzug.....9
3.6.4 Der Weg.....9
3.6.5 Zusammenhängende Knoten.....9
3.6.6 Zusammenhängende Graphen.....9
3.6.7 Der Zyklus.....9

3.7 Der gerichtete Graph.....10

3.8 Der bewertete Graph.....11

4. Kombinatorik.....12

4.1 Einführung.....12

4.2 Die Permutation.....12

4.3 Die Kombination.....13

5. Ein dynamisches Lösungsverfahren.....14

5.1 Einführung.....14

5.2 Das Prinzip des dynamischen Verfahrens.....14

5.3 Ein Beispiel für dieses dynamisches Verfahren.....15

6. Bedienungsanleitung für das Programm anhand eines Beispiels.....17
6.1 Starten des Programms und Eingabe einer Matrix.....17
6.2 Verlassen des Programms mit dem Befehl „exit“.....18
6.3 Korrekturen in der Matrix.....18
6.4 Speichern der Matrix.....18
6.5 Die Ausgabe.....18
6.6 Beenden und Neustarten des Programms.....19
6.7 Laden der Matrix.....19

7. Literaturverzeichnis.....20

8. Danksagungen.....21

9. Selbständigkeitserklärung.....22

10. Thesen.....23

11. Anlagen.....I
11.1 Eine Beispielmatrix für die Bedienungsanleitung.....I
11.2 Deutschlandkarte mit eingezeichneter Route.....II
11.3 Das Programm (TRAVSAL3.EXE) auf einer Diskette 3,5″ inkl. der fertigen Beispielmatrix (DEUTLAND.MTX).....III

1. Verzeichnis der verwendeten Abkürzungen

Abkürzung	**Bezeichnung**
∅	leere Menge
C	Kombination
C_n	n-Zyklus
E(G)	Kantenmenge von G
e	Kanten
G	Graph
P	Permutation
TSP	Travelling-Salesman-Problem
u; v	Knoten
V(G)	Knotenmenge von G
W	Kantenfolge

2. Einführung und Zielstellung

In der vorliegenden Arbeit wird folgendes Problem behandelt:
Ein Handlungsreisender, der sich am Ort 1 befindet, hat die Absicht, n-1 Orte 2,...,n zu besuchen. Dabei will er jede Ortschaft genau einmal durchreisen und zum Schluss wieder in den Ort 1 zurückkehren. Daraus ergibt sich, dass Ort 1 gleichzeitig der Start- und Zielort also Ort n ist und die Orte 2,...,n Durchgangsorte sind. Seinen Weg will er dabei nach den Gesichtspunkten Zeit, Entfernung oder Kosten optimieren.
Dazu muss er sich moderner Computertechnik bedienen, da der Rechenaufwand mit steigender Anzahl der Orte enorm hoch wird. Für die Lösung seines Problems findet er verschiedene Typen numerischer Verfahren. Ein Teil ist so konzipiert, dass die Aufgabe exakt gelöst wird. Allerdings scheitern diese in der Regel bei einer größeren Anzahl von Orten wegen des mit der Problemgröße stark progressiv steigenden Rechenaufwandes. Ein anderer Teil benutzt heuristische Ansätze und sucht nach guten Näherungslösungen, die auch bei wesentlich größeren Problemen den Handlungsreisenden zufrieden stellen.
Ziel dieser Arbeit ist es nun,

die Aufgabe als Problem der Graphentheorie zu diskutieren und

einen exakten Algorithmus mit einer dynamischen Struktur vorzustellen.

Dazu werden entsprechende Begriffe eingeführt und Beispiele vorgestellt, an denen dann auch der Algorithmus erprobt wird.

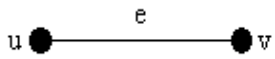
3. Grundbegriffe aus der Graphentheorie[1]

3.1 Graph

Ein **Graph** G = (V(G), E(G)) besteht aus zwei endlichen Mengen:
V(G), der **Knotenmenge** des Graphen, oft nur mit V bezeichnet, die eine nichtleere endliche Menge von Elementen ist, die **Knoten** genannt werden, und
E(G), der **Kantenmenge** des Graphen, häufig nur mit E bezeichnet, die eine Menge von Knotenpaaren ist, die **Kanten** genannt werden,
sodass jede Kante $e \in E(G)$ einem untergeordneten Paar von Knoten (u, v) zugeordnet ist, die als **Endknoten** von e bezeichnet werden.

3.2 Knoten und Kanten

Knoten werden manchmal auch als **Punkte**, **Knotenpunkte** oder **Ecken** bezeichnet.
Wenn e eine Kante mit den Endknoten u und v ist, dann sagt man, dass sie u und v verbindet. Man bezeichnet dann e auch durch e = (u,v)
Beispiel:



Man beachte, dass die Definition eines Graphen die Möglichkeit zulässt, dass eine Kante identische Endknoten hat, d.h., es ist möglich, einen Knoten u mit sich selbst durch eine Kante zu verbinden – eine derartige Kante wird als **Schlinge** bezeichnet.
Beispiel:

[1] betreffend die Punkte 3.1 bis 3.6: Clark/Holton 1994, S. 2-33

3.3 Der parallele Graph

Es sei G ein Graph. Wenn zwei (oder mehrere) Kanten von G dieselben Endknoten haben, dann werden diese Kanten **parallel** genannt.

Beispiel:

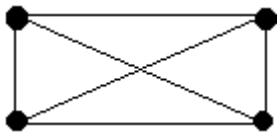
3.4 Der schlichte Graph

Ein Graph heißt schlicht, wenn er keine Schlingen und keine parallelen Kanten enthält.

3.5 Der vollständige Graph

Ein Graph heißt **vollständig**, wenn jeder Knoten mit jedem anderen verbunden ist.

Beispiel:



3.6 Wege und Zyklen

3.6.1 Eine **Kantenfolge** in einem Graphen ist eine endliche Folge der Gestalt

$$W = v_0e_1v_1e_2v_2...v_{k-1}e_kv_k,$$

deren Terme abwechselnd Knoten und Kanten sind, sodass, für $1 \leq i \leq k$, die Kante e_i die Endknoten v_{i-1} und v_i hat.

Somit befindet sich jede Kante e_i unmittelbar zwischen zwei Knoten, mit denen sie **inzident** ist.

Wir nennen die obige Kantenfolge W eine v_0-v_k-**Kantenfolge** oder eine **Kantenfolge von** v_0 **nach** v_k. Der Knoten v_0 heißt **Anfangsknoten** der Kantenfolge W, während v_k als **Endknoten** von W bezeichnet wird. Man beachte, dass die Knoten v_i, i=0,..., k, nicht unterschiedlich sein müssen.

Die Knoten $v_1,...,v_{k-1}$ der obigen Kantenfolge W werden als **innere Knoten** bezeichnet. Die ganze Zahl k, die durch die Anzahl der Kanten in der Kantenfolge gegeben ist, heißt die **Länge** der Kantenfolge W.

3.6.2 Eine **triviale Kantenfolge** beinhaltet keine Kanten.

3.6.3 Wenn die Kanten e_1, e_2,..., e_k der Kantenfolge $W = v_0e_1v_1e_2v_2...e_kv_k$ alle unterschiedlich sind, dann heißt W ein **Kantenzug.**
Mit $e_1 = (v_0,v_1)$,..., $e_k = (v_{k-1}, v_k)$ ergibt sich für W auch eine kürzere Beschreibung: $W = v_0v_1...v_k$.

3.6.4 Wenn die Knoten $v_0,v_1,...,v_k$ der Kantenfolge $W = v_0e_1v_1e_2v_2...e_kv_k$, unterschiedlich sind, dann heißt W ein **Weg**.
Ein Weg mit $v_0 = v_k$ heißt **geschlossen**.

3.6.5 Der Knoten u wird als **zusammenhängend** mit einem Knoten v in einem Graphen G bezeichnet, wenn es einen Weg von u nach v gibt.

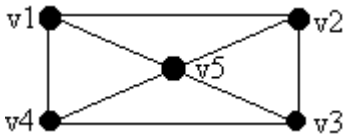
3.6.6 Ein Graph G heißt **zusammenhängend**, wenn je zwei seiner Knoten zusammenhängend sind.

3.6.7 Ein nichttrivialer geschlossener Kantenzug in einem Graphen G heißt ein **Zyklus** (geschlossener Weg in G), wenn sein Anfangs- bzw. Endknoten ungleich den Durchgangsknoten ist und wenn die Durchgangsknoten untereinander unterschiedlich sind. Anders gesagt ist der geschlossene Kantenzug $C = v_1e_1v_2...v_ne_nv_1$ dann ein Zyklus, wenn

a) C mindestens eine Kante hat und
b) $v_1, v_2,...,v_n$ alles unterschiedliche Knoten sind.

Ein Zyklus der Länge k, d.h. mit k Kanten, heißt ein **k-Zyklus**. Ein k-Zyklus heißt gerade, wenn k gerade ist, oder **ungerade**, wenn k ungerade ist.
Ein 3-Zyklus wird oft als **Dreikreis** bezeichnet.

Beispiel:



In der Abbildung ist

$C = v_1v_2v_3v_4v_1$ ein 4-Zyklus,

$T = v_1v_2v_5v_3v_4v_5v_1$ ein nichttrivialer geschlossener Kantenzug, der keinen Zyklus darstellt (da v_5 zweimal als innerer Knoten vorkommt) und

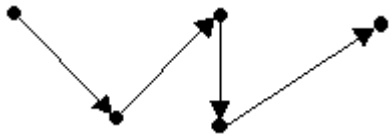
$C' = v_1v_2v_5v_1$ ein Dreikreis.

3.7 Der gerichtete Graph[2]

Bei einem gerichteten Graph wird den Kanten mit Hilfe eines Pfeils eine bestimmte Richtung zugewiesen.

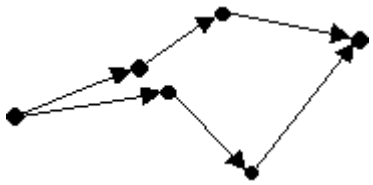
Ein gerichteter Graph heißt **Pfad** oder **Schlange**, wenn er durch einen offenen gleichgerichteten Kantenzug darstellbar ist (d.h. in den Knoten treffen die Spitze des einen Pfeils und der Schaft des anderen Pfeils zusammen).

Beispiel:



Ein gerichteter Graph heißt **Masche**, wenn sich in zwei gleichgerichtete Pfade mit gemeinsamen Anfangs- und Endpunkt zerlegen lässt.

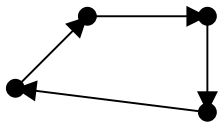
Beispiel:



[2] Athen/Bruhn 1994, Bd. 2, S. 356

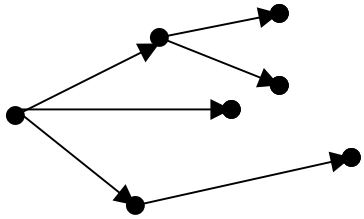
Ein gerichteter Graph heißt **Schleife**, wenn er durch einen geschlossenen gleichgerichteten Kantenzug darstellbar ist.

Beispiel:



Ein gerichteter Graph heißt **Baum**, wenn er keine Maschen und Schleifen enthält.

Beispiel:



3.8 Der bewertete Graph

Ein Graph heißt dann bewertet, wenn die **Kanten** eine entsprechende **Bewertung** (Zeit, Entfernung oder Kosten) erhalten.

4. Kombinatorik[3]

4.1 Einführung

Kombinatorik ist die Theorie der endlichen Mengen, insbesondere die Untersuchung gewisser zahlentheoretischer Auswahlfunktionen, z.B. die Anzahl der Primzahlen. Kombinatorische Schlussweisen beherrschen viele Gebiete der Mathematik: Zahlentheorie, endliche Gruppentheorie, Wahrscheinlichkeitsrechnung und Statistik. In der **elementaren** Kombinatorik geht es insbesondere um Anzahlprobleme von Anordnungen und Auswahlen aus endlichen Mengen (Permutation, Variation, Kombination). Dabei werden laufend die folgenden Abzählungsregeln verwandt:

a) **Summenregel:** Hat man für einen Entscheidungsvorgang die Wahl zwischen **k** Arten, die sich gegenseitig ausschließen und für die $\mathbf{m_1, m_2, ..., m_k}$ Möglichkeiten bestehen, so gibt es insgesamt $\mathbf{m_1 + m_2 + ... + m_k}$ Entscheidungsmöglichkeiten.

b) **Produktregel:** Sind nacheinander **r** Entscheidungen zu treffen, wobei die **k**-te Entscheidung – unabhängig von den vorhergegangenen Entscheidungen – $\mathbf{n_k}$ Möglichkeiten zulässt, so gibt es insgesamt $\mathbf{n_1 \bullet n_2 \bullet n_3 \bullet ... \bullet n_T}$ Entscheidungsmöglichkeiten.

4.2 Die Permutation[4]

Ist eine endliche Anzahl von **n** Elementen gegeben, so bezeichnet man die mögliche Anordnung dieser Elemente als **Permutation** der gegebenen Elemente.

P_n – Anzahl der Permutationen, wenn alle Elemente verschieden sind

Beispiel: a, b, c, d

$$P_4 = 4! = 1 \bullet 2 \bullet 3 \bullet 4 = 24$$

$\Rightarrow P_n = n!$

[3] Athen/Bruhn 1994, Bd. 2, S. 497

[4] Grimm 1994, S. 30

4.3 Die Kombination[5]

Ist eine endliche Anzahl von **n** Elementen gegeben, so bezeichnet man jede Auswahl von **k** Elementen ohne Berücksichtigung ihrer Reihenfolge als **Kombination**.

Kombination ohne Wiederholung

C_n^k - Anzahl der Kombination, wenn jedes Element in einer Kombination nur einmal vorkommt.

Beispiel: 4 Elemente, 2. Klasse

$$C_4^2 = \binom{4}{2} = \frac{4 \cdot 3}{1 \cdot 2} = 6$$

$$\Rightarrow C_n^k = \binom{n}{k}$$

[5] Grimm 1994, S. 31

5. Ein dynamisches Lösungsverfahren

5.1 Einführung

Das dynamische Verfahren zur Lösung des TSP, welches nun vorgestellt werden soll, wird von Uwe Schöning im Buch „Algorithmen – kurz gefasst“ beschrieben. Die **Grundlage** für den resultierenden Algorithmus bildet die **Bestimmung kürzester** Wege in einem **Graphen zwischen** einem **Knoten A** und einem **Knoten E**.

Eine **Rundreise** ist dadurch charakterisiert, dass **im vollständigen Graphen** mit den **Knoten** $v_1,...,v_{n-1}$ und **nichtnegativen Kantenbewertungen** ein **kürzester Weg** von v_1 bis $v_n = v_1$ über die Knoten $v_2,...,v_{n-1}$ gesucht wird. Dieser Weg ist dann ein **n-Zyklu**s in G kürzester Gesamtlänge. Die **Knoten entsprechen** den **Orten**.

Hierfür wird nun ein **dynamisches Verfahren** vorgeschlagen. Dieses beruht darauf, dass ein **kürzester Weg von** einem **Zwischenknoten** v_j, $j \in \{2,...,n-1\}$ **über** einen **Nachfolger** v_k, $k \in \{2,...,n-1\}$, $k \neq j$, bis **zum Zielort** $v_n = v_1$ **zerlegt** werden kann in den **Schritt über die Kante von v_j nach v_k** und in den **kürzesten Weg von v_k nach v_n**. Dabei darf dann aber der **Knoten v_j nicht wieder durchlaufen** werden. Durch diese Herangehensweise versucht man, von vornherein **ungünstige Zwischenwege** zu **vermeiden**.

5.2 Das Prinzip des dynamischen Verfahrens

Im folgenden benutzen wir für die **Orte** nur die **Bezeichnung** durch ihre **Nummern** 1,...,n .

Wenn eine optimale Rundreise (ohne Beschränkung der Allgemeinheit) bei Stadt 1 beginnt und dann die Stadt k besucht, so muss der Weg von k aus durch die Städte $\{2,...,n\}-\{k\}$ zurück zu 1 ebenfalls optimal sein (unter allen solchen Wegen). Hier deutet sich ein **Optimalitätsprinzip** an, das man wie folgt umsetzen kann.

Sei g(i,S) die Länge des kürzesten Wegs, der bei Stadt i beginnt, dann durch jede Stadt der Menge S mit $S \subseteq \{1,...,n\} - \{i\}$ genau einmal geht, um bei Stadt 1 zu enden.

Man beachte, dass die Lösung des TSP darin besteht, g(1,{2,...,n}) zu berechnen. Die Funktion kann wie folgt rekursiv geschrieben werden:

$$\mathbf{g(i,S) = m_{i1}, \quad für\ S=\varnothing}$$

$$\mathbf{g(i,S) = \min_{j\in S}\,(m_{ij} + g(j,S - \{j\})), \quad für\ S\neq\varnothing}$$

5.3 Ein Beispiel für dieses dynamische Verfahren[6]

Gegeben sei folgende **Entfernungsmatrix** zum Graphen G mit 4 Knoten:

M =

Ort	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

Der Algorithmus berechnet die folgenden Werte:

$g(2,\varnothing) = m_{21} = 5, \qquad g(3,\varnothing) = m_{31} = 6, \qquad g(4,\varnothing) = m_{41} = 8$

$$g(2,\{3\}) = m_{23} + g(3,\varnothing) = 15$$
$$g(2,\{4\}) = m_{24} + g(4,\varnothing) = 18$$
$$g(3,\{2\}) = m_{32} + g(2,\varnothing) = 18$$
$$g(3,\{4\}) = m_{34} + g(4,\varnothing) = 20$$
$$g(4,\{2\}) = m_{42} + g(2,\varnothing) = 13$$
$$g(4,\{3\}) = m_{43} + g(3,\varnothing) = 15$$

$$g(2,\{3,4\}) = \min\,(m_{23} + g(3,\{4\}), m_{24} + g(4,\{3\}) = 25$$
$$g(3,\{2,4\}) = \min\,(m_{32} + g(2,\{4\}), m_{34} + g(4,\{2\}) = 25$$
$$g(4,\{2,3\}) = \min\,(m_{42} + g(2,\{3\}), m_{43} + g(3,\{2\}) = 23$$

[6] Schöning 1997, S. 98 f.

$$
\begin{aligned}
&g(1,\{2,3,4\}) \\
&\quad = \min\left(m_{12} + g(2,\{3,4\}),\ m_{13} + g(3,\{2,4\}),\ m_{14} + g(4,\{2,3\})\right) \\
&\quad = \min(35, 40, 43) \\
&\quad = 35
\end{aligned}
$$

Indem man nachvollzieht, über welche **Minimumsbildung** der gesuchte Wert zustande kommt, erhält man auch die zugehörige Lösung. Gegeben **durch den 4-Zyklus** ist die **optimale Rundreise** 1-2-4-3-1.

6. Bedienungsanleitung für das Programm anhand eines Beispiels

Der Umgang mit dem Programm erfordert grundsätzlich keine besonderen Computerkenntnisse, wenn man es beherrscht, Werte einzugeben und diese auch mit „Enter“ zu bestätigen. Beim Programmieren stand die **Funktionalität** und nicht die Optik **im Vordergrund**. D.h., dass sowohl auf die Eingabe-, als auf auch die Verarbeitungs- und die Ausgabegaberoutine sehr viel Wert gelegt wurde. Hiermit rückte allerdings die Optik in den Hintergrund, sodass es keine Animationen der optimalen Wege geben wird.
Ich möchte nun anhand eines Beispiels die **Bedienung** und die **Funktionen** des **Programms** erläutern. Die für das Beispiel entsprechende Matrix (Siehe Anlage I) und Deutschlandkarte (Siehe Anlage II) finden Sie in den Anlagen.

6.1 Starten des Programms und Eingabe einer Matrix

Zu Beginn **startet** man das **Programm mit** einem **Doppelklick** auf die Anwendungsdatei **„TRAVSAL3.EXE“**. Sie werden eine kurze Begrüßung lesen können. Des weiteren haben Sie die Möglichkeit eine bereits abgespeicherte Matrix zu laden, doch dazu später mehr.
Sie wollen nun eine komplett **neue Rundreise** berechnen lassen und drücken deshalb einmal die **„Enter“-Taste**. Jetzt haben Sie die **Wahl** zwischen **zwei bis zehn Orten**, die Sie besuchen möchten. Da das Beispiel zehn Orten beinhaltet, geben Sie „10“ ein und bestätigen dies mit „Enter“.
Sie werden noch darauf hingewiesen, dass **Ort 1** Ihr **Startort** sein wird und dass man das Programm jederzeit durch die Eingabe von „exit“ abbrechen kann → *6.2 Verlassen des Programms mit dem Befehl „exit*“.
Jetzt werden Sie aufgefordert, die **Entfernung** von **Ort 1**, Leipzig, **zu Ort 2**, Berlin, **einzugeben**. Also geben Sie nun entsprechend der Matrix im Anhang „146.25“ ein. Beachten Sie bitte, dass Sie **statt** des gewöhnlichen **Komma**s einen **Punkt** setzten. Diese Eingabe **bestätigen** Sie nun **mit „Enter“**. Jetzt werden die **restlichen Entfernungen** von Ort 1 zu den Orten 3 bis 10 abgefragt. Danach folgen die Entfernungen von Ort 2 zu den Orten 1 und 3 bis 10. Die Entfernung zu

Ort 2 wird hier logischerweise nicht abgefragt, da diese zwingend „0“ beträgt. Auf diese Weise geben Sie nun die komplette Matrix ein.
Wenn Sie sich die langwierige Eingabe ersparen wollen, laden Sie einfach die dem Programm beiliegende fertige Matrix „DEUTLAND.MTX“ (Siehe Anlage 3) → *6.7 Laden der Matrix*.

6.2 Verlassen des Programms mit dem Befehl „exit“

Während der **Eingabe** der Matrix können Sie das Programm jederzeit **abbrechen**, indem Sie als Wert einfach **„exit“ eingeben** und dies mit „Enter“ bestätigen.

6.3 Korrekturen in der Matrix

Wenn Sie die letzte Entfernung von Ort 10, Dresden, zu Ort 9, München, eingegeben und bestätigt haben, gibt es noch die Möglichkeit Wertänderungen in der Matrix vorzunehmen, falls Sie einmal eine falsche Eingabe getätigt haben. Hierzu geben Sie **„c“** ein und **bestätigen** dies. Sie werden nun nach dem **Ausgangsort**, dem **Zielort** und der **Entfernung** abgefragt, die Sie **ändern** möchten. Danach wird die korrigierte Matrix erneut angezeigt und Sie können durch die Eingabe von „c“ weitere Korrekturen vornehmen. Wenn Sie der Meinung sind, dass Sie alle Werte **korrekt** in die **Matrix** eingegeben haben, drücken Sie **„Enter“**.

6.4 Speichern der Matrix

Nach der **Korrekturabfrage**, haben Sie die **Möglichkeit** Ihre Matrix **abzuspeichern**, um einem nochmaligen langwierigen Eingeben zu entgehen. Dazu geben Sie **„s“** ein und **bestätigen** dies. Jetzt müssen Sie einen **Dateinamen**, welcher **nicht länger** als **neun Zeichen** sein darf, eingeben, z.B. „matrix1“, und mit **„Enter“** bestätigen. Die Datei wird im gleichen Verzeichnis abgespeichert, in dem sich auch das Programm befindet.
Falls Sie ihre Matrix **nicht abspeichern** wollen, drücken Sie einfach **„Enter**“.

6.5 Die Ausgabe

Nach dem **Speichern** gibt Ihnen nun das Programm die **optimale** (minimale) **Weglänge** und die **optimale Rundreise** aus. Bei unserem Beispiel beträgt die optimale Weglänge „1847.25“ Einheiten (in diesem Fall Kilometer). Die optimale Rundreise ist: 1 - 7 - 9 - 8 - 6 - 4 - 5 - 3 - 2 - 10 - 1 (Leipzig - Erfurt - München - Stuttgart - Frankfurt - Köln - Kassel - Hamburg - Berlin - Dresden – Leipzig).
Die Deutschlandkarte mit der eingezeichneten Route finden Sie in den Anlagen (Siehe Anlagen II).

6.6 Beenden und Neustarten des Programms

Nach der **Ausgabe** können Sie das Programm durch Drücken der **„Enter“**-Taste **beenden**. **Falls** Sie das Programm jedoch **noch einmal nutzen** möchten geben Sie „**r**“ ein und **bestätigen** dies mit „Enter“.

6.7 Laden der Matrix

Nach dem Programm**start oder** Programm**neustart** können Sie eine fertige Matrix **laden**. Geben Sie **„l“** ein und bestätigen Sie dies mit **„Enter“**. Jetzt werden Sie aufgefordert einen **Dateinamen einzugeben**, der **nicht länger** als **neun Zeichen** sein darf, z.B. „matrix1“. Wenn Sie das **mit „Enter“ bestätigt** haben **und** diese **Datei existiert**, sind Sie sofort an der **Programmstelle**, wo Sie **Korrekturen** in der Matrix vornehmen können → *6.3 Korrekturen in der Matrix*.

7. Literaturverzeichnis

Athen/Bruhn:	Lexikon der Schulmathematik; Bd. 1-4; Weltbild Verlag; Augsburg 1994
Baumann, R.:	Informatik für die Sekundarstufe II; Bd. 1; Ernst Klett Schulbuchverlag; Stuttgart 1992
Borland GmbH:	Borland Pascal mit Objekten; München 1992
Borland GmbH:	Turbo Assembler; München 1992
Clark/Holton	Graphentheorie - Grundlagen und Anwendungen; Spektrum Akademischer Verlag; Heidelberg - Berlin 1994
Dornbusch/Kämmer:	Heimat und Welt – Weltatlas (Ausgabe Sachsen); Westermann; Braunschweig 1994
Griesel/Postel:	Informatik Heute; Bd. 1; Schroedel Schulbuchverlag; Hannover 1987
Grimm, B.:	Das große Tafelwerk; Volk und Wissen; Berlin 1994
Müller-Mehrbach:	Optimale Reihenfolgen; Springer Verlag; Berlin-Heidelberg-New York 1970
Schöning, U.:	Algorithmen – kurz gefasst; Spektrum Akademischer Verlag; Heidelberg-Berlin 1997

Sonstige Hilfsmittel:

- Software zum Programmieren: Borland Pascal 7.0
- Deutschlandkarte in den Anlagen: http://ourworld.compuserve.com/homepages/Martin_Adler/Germany/Germany.htm

8. Danksagungen

Hiermit möchte ich mich recht herzlich bei all denen bedanken, die mich bei der Anfertigung dieser Arbeit unterstützt haben: Zum Einen bei Frau Fizia, meiner Deutschlehrerin, die mir beigebracht hat, eine solche Dokumentation anzufertigen. Des weiteren bei Herrn Scheuermann, der mir jede Menge Literatur bezüglich des Programmierens zur Verfügung stellte. Ein besonderer Dank gebührt natürlich sowohl auch Herr Dr. Graubner, meinem Betreuungslehrer, der mich beim Anfertigen der Präsentation unterstützte und auch sonst bei Fragen immer ein offenes Ohr hatte, als auch Frau Dr. Meiler, die mir in programmiertechnischer Hinsicht eine große Hilfe war.
Doch am allermeisten möchte ich mich bei meiner Betreuerin Frau Dr. Kripfganz bedanken, die mich ein ganzes Jahr unterstützte und ständig für mich zur Verfügung stand, wenn ich Fragen oder Probleme hatte.

9. Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

Leipzig, 19. Juni 2001

Thomas Blaßkiewitz

10. Thesen

1. Die Arbeit beschäftigt sich mit dem Problem des Handlungsreisenden (Travelling-Salesman-Problem).
2. Das Anliegen ist es, ein Programm zu erstellen, welches dieses Problem in einem vertretbaren Rechenaufwand exakt löst.
3. Dafür war es nötig, sich die Grundlagen der Graphentheorie und Kombinatorik anzueignen.
4. Außerdem musste man sich mit mehreren dynamischen Lösungsverfahren beschäftigen, um am Ende das Beste auswählen zu können.
5. Mit diesem Wissen war es nun möglich, ein entsprechendes Programm zu erstellen.
6. Dieses Programm kann in Form eines Routenplaners völlig unabhängig von den Orten genutzt werden, insofern man die Möglichkeit hat, sich eine dem Problem gemäße Matrix zu erstellen.