Technische Universität Berlin

Institut für Geodäsie und Geoinformationstechnik

Fachgebiet Photogrammetrie und Kartographie

Diplomarbeit

Methoden zur Routenoptimierung in Geo-Informationssystemen

am Beispiel einer Sozialstation mit Hauspflegediensten

vorgelegt von Jörn Hatzky

April 2001

Technische Universität Berlin

Institut für Geodäsie und Geoinformationstechnik

Fachgebiet Photogrammetrie und Kartographie

Diplomarbeit

Methoden zur Routenoptimierung in Geo-Informationssystemen

am Beispiel einer Sozialstation mit Hauspflegediensten

eingereicht bei:

Prof. Dr.-Ing. Jörg Albertz
Fachgebiet Photogrammetrie und Kartographie

vorgelegt von:

Jörn Hatzky
Matrikelnummer: 153699

April 2001

Berlin, den 15. April 2001

Hiermit erkläre ich an Eides statt, daß ich die Diplomarbeit selbständig angefertigt und keine anderen als die angegebenen Quellen benutzt habe.

_____

Jörn Hatzky

Inhaltsverzeichnis

1. Einführung

1.1 Motivation

Die Caritas-Station in Berlin-Tempelhof ist eine soziale Einrichtung zur Pflege hilfebedürftiger und vorwiegend älterer Menschen. Die Kunden der Station sind durchschnittlich 76 Jahre alt und werden im Rahmen des Hauspflegedienstes von Krankenschwestern, Pflegerinnen und Zivildienstleistenden medizinisch betreut. Der Kundenstamm umfaßt etwa 200 Personen, die Fluktuation ist mit 60 wegfallenden und neu hinzukommenden Klienten pro Monat sehr hoch.

Da die Struktur des Caritas-Verbandes für jeden Berliner Stadtbezirk nur eine Sozialstation vorsieht, umfaßt das Einzugsgebiet den gesamten Bezirk Tempelhof. Wegen der günstigen geographischen Lage wird der zum Nachbarbezirk Steglitz gehörende Ortsteil Lankwitz ebenfalls mitversorgt (vgl. Abb. 7.5, S. 80).

Betrachtet man die Größe und Infrastruktur dieses Areals, wird schnell deutlich, daß bei Kundenbesuchen lange Wege zurückzulegen sind und mit erheblichen Fahrtzeiten zu rechnen ist, d.h. die meisten Straßen führen durch eng besiedeltes Gebiet. Auf insgesamt 48 Quadratkilometern leben 230.000 Einwohner und durch das hohe Verkehrsaufkommen müssen zusätzliche Wartezeiten in Kauf genommen werden. So führt eine Mitarbeiterin der Station an einem Arbeitstag durchschnittlich etwa sechs bis acht Hausbesuche durch, wobei die Behandlungszeit pro Pflegeperson nur ca. 20 Minuten beträgt.

1.2 Warum Routenoptimierung?

Anläßlich des Mißverhältnisses von Fahrtzeit und eigentlicher Behandlungszeit entstand die Idee, die Routenplanung der Caritas-Station mit Hilfe eines Computerprogramms zu verbessern. Durch die Anwendung sollen

- Touren mit *möglichst kurzer Wegstrecke* und / oder
- Touren mit *möglichst geringer Fahrtzeit*

gefunden werden.

Um der geographisch-räumlichen Komponente der Problemstellung gerecht zu werden, soll die Umsetzung in einem Geo-Informationssystem (GIS) erfolgen. Sogenannte „Objekte“ wie Straßen, Wohnblöcke, Grundstücksgrenzen oder Adressen können in einem GIS mit Hilfe geeigneter Daten modelliert und abgebildet werden. Durch Einführung eines Koordinatensystems sind diese Objekte im Computer eindeutig lokalisierbar und erhalten einen direkten örtlichen Bezug zu ihrer Umgebung (Weiteres dazu in Kapitel 7).

2. Graphentheoretische Grundlagen

Wie läßt sich das in der Einführung beschriebene Wegeproblem der Caritas-Station mathematisch behandeln? Um Berechnungen im Tempelhofer Straßennetz durchführen zu können, ist eine Abstraktion notwendig, für die einige grundlegende Begriffe aus der Graphentheorie benötigt werden (nach DOMSCHKE, DREXL 1991, S.54 ff.).

2.1 Graph

Ein *Graph G* besteht aus einer nichtleeren Menge *V* von *Knoten*, welche durch natürliche Zahlen *i = 1, 2, ..., n* gekennzeichnet werden. Die Knoten sind durch eine Menge *E* von *Kanten* (bei ungerichteten Graphen) oder *Pfeilen* (bei gerichteten Graphen) miteinander verbunden. Es wird eine Abbildung definiert, welche jeder Kante oder jedem Pfeil aus *E* genau ein Knotenpaar *i, j* aus *V* zuordnet. Man schreibt bei ungerichteten Graphen *G = [V, E]* und bei gerichteten Graphen *G = (V, E)*.

2.2 Ungerichteter Graph

Lassen sich den Verbindungen zwischen den Knotenpaaren keine Richtungen zuweisen, spricht man von einem *ungerichteten* Graph *G*. Die Elemente der Menge *E* heißen *Kanten*. Zwei Knoten *i* und *j*, welche durch eine Kante *[i, j]* miteinander verbunden werden, bezeichnet man als Nachbarn. Man sagt, diese Knoten sind mit der Kante *inzident* (und umgekehrt). Der *Grad g* eines Knotens ergibt sich aus der Anzahl der mit ihm inzidenten Kanten.

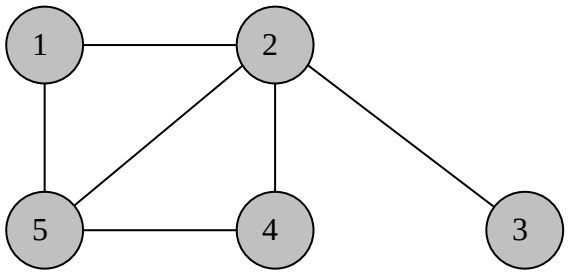


Abb. 2.1: Ungerichteter Graph *G* = [V, E] mit der Knotenmenge *V* = {1, 2, 3, 4, 5} und der Kantenmenge *E* = {[1, 2], [1, 5], [2, 3], [2, 4], [2, 5], [4, 5]}. Knoten 2 ist inzident mit den Kanten [2, 1], [2, 5], [2, 4] und [2, 3], besitzt also den Grad *g* = 4.

2.3 Gerichteter Graph

Lassen sich den Verbindungen zwischen den Knotenpaaren Richtungen zuweisen, spricht man von einem *gerichteten* Graph. Die Elemente der Menge *E* heißen *Pfeile*.

Ein Pfeil, der von Anfangsknoten *i* zu Endknoten *j* führt, wird mit *(i, j)* gekennzeichnet. Dabei nennt sich Knoten *j Nachfolger* von Knoten *i* und Knoten *i Vorgänger* von Knoten *j*.

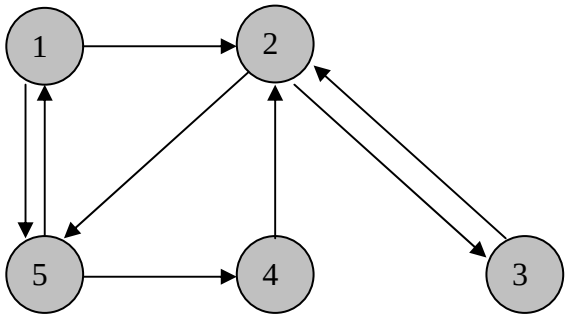


Abb. 2.2: Gerichteter Graph *G = (V, E)* mit Knotenmenge *V* = {1, 2, 3, 4, 5} und Pfeilmenge *E* = {(1, 2), (1, 5), (2, 3), (2, 5), (3, 2), (4, 2), (5, 1), (5, 4)}. Knoten 2 ist Nachfolger von Knoten 1 und Knoten 5 besitzt den Grad 4 (Anzahl der ein- und ausgehenden Pfeile).

2.4 Schlichter Graph

Ein Graph ohne parallele Kanten oder Pfeile heißt *schlichter* Graph. Als *parallel* werden Pfeile oder Kanten mit identischen Anfangs- und Endknoten bezeichnet. Eine *Schlinge* ist eine Kante vom Typ *[i, i]* oder ein Pfeil vom Typ *(i, i).* Sie führt zu dem Knoten zurück, in dem sie ihren Ursprung hat.

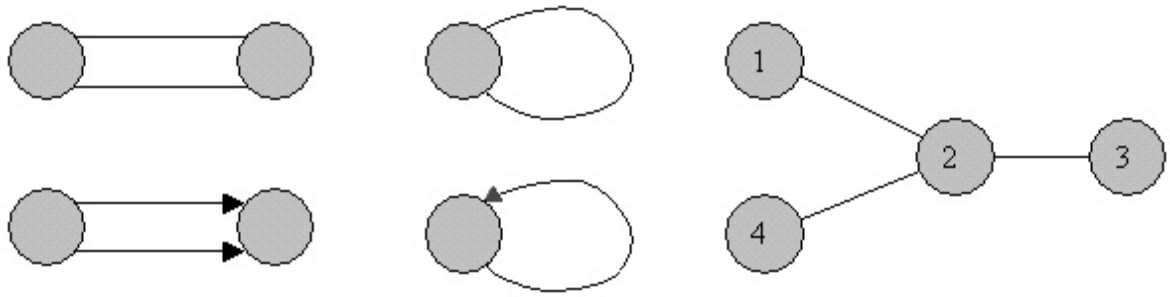


Abb. 2.3: Parallele Kanten und Pfeile (links), Schlingen (mitte) und ein ungerichteter schlichter Graph (rechts).

2.5 Digraph

Ein schlichter gerichteter Graph *G = (V, E)* mit endlicher Knotenmenge heißt *Digraph*.

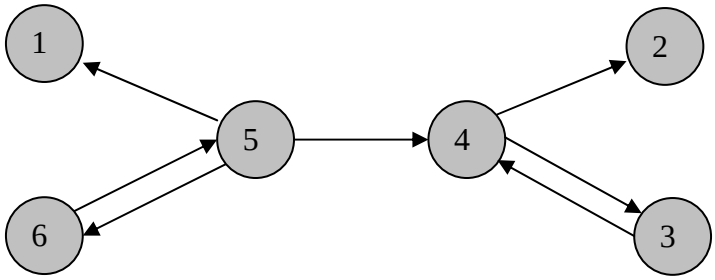


Abb. 2.4: Beispiel für einen Digraph

2.6 Vollständiger Graph

Einen schlichten ungerichteten Graph nennt man *vollständig*, wenn zu jedem Knotenpaar *i, j* eine Kante *[i, j]* existiert. Bei *n* vorhandenen Knoten besitzt ein vollständiger Graph also $n \cdot (n-1) / 2$ Kanten.

Ein Digraph heißt dementsprechend vollständig, wenn für jedes Knotenpaar *i, j* ein Pfeil *(i, j)* und ein Pfeil *(j, i)* existiert. Zu *n* Knoten gibt es also $n \cdot (n-1)$ Pfeile.

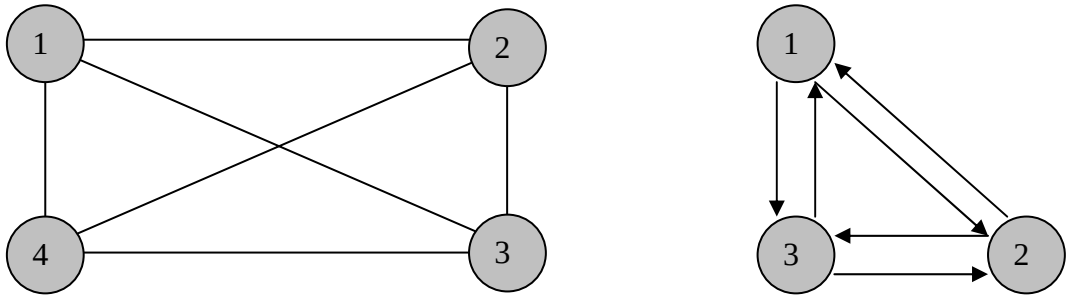


Abb. 2.5: Ungerichteter vollständiger Graph (links) und vollständiger Digraph (rechts)

2.7 Bewerteter Graph

Ein ungerichteter oder gerichteter Graph *G*, dessen sämtliche Kanten bzw. Pfeile eine Bewertung *c[i, j]* bzw. *c(i, j)* aufweisen, heißt *bewerteter* Graph. Der Wert *c* repräsentiert die *Kosten*, die beim Transport von Knoten *i* nach Knoten *j* entstehen. Dies kann z.B. die Länge der Verbindung sein oder die Zeit, welche zum Durchlaufen der Kante benötigt wird. Ein bewerteter Graph wird als *G = [V, E, c]* bzw. *G = (V, E, c)* gekennzeichnet, für die Bewertung *c[i, j]* einer Kante bzw. *c(i, j)* eines Pfeils verwendet man die Indexschreibweise c_{ij} .

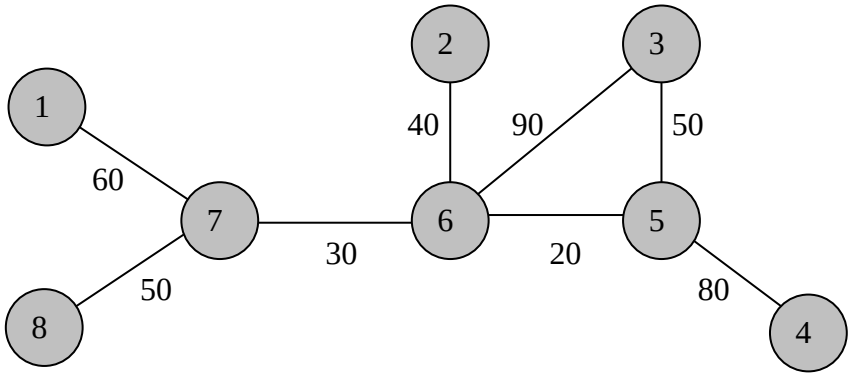


Abb. 2.6: Bewerteter Graph mit den Kosten $c_{17} = 60$, $c_{26} = 40$, $c_{35} = 50$, $c_{36} = 90$, $c_{45} = 80$, $c_{56} = 20$, $c_{67} = 30$, $c_{78} = 50$.

2.8 Weg in einem Graphen

Eine Folge $p_1,...,p_t$ von Pfeilen in einem gerichteten Graphen heißt *Weg* von *G*, wenn eine Folge $j_0,...,j_t$ von Knoten mit $p_h = (j_{h-1}, j_h)$ für alle *h = 1,...,t* existiert. Einen Weg symbolisiert man durch alle ihn ihm enthaltenden Knoten, z. B. $w = (j_0,...,j_t)$.

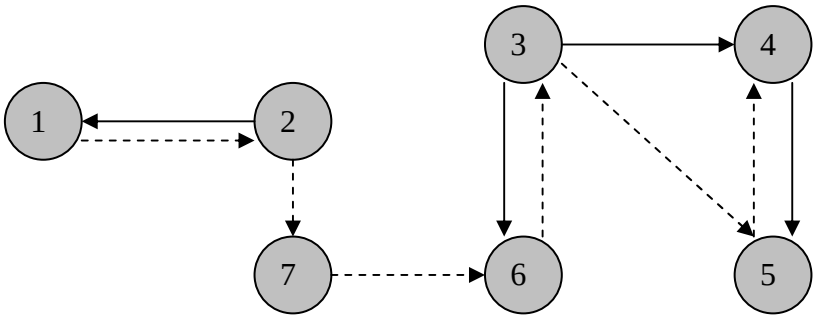


Abb. 2.7: Der Graph enthält den gestrichelten Weg *w* = (1, 2, 7, 6, 3, 5, 4).

2.9 Kette in einem Graphen

Eine Folge $p_1,...,p_t$ von Pfeilen in einem gerichteten Graphen heißt *Kette* von *G*, wenn eine Folge $j_0,...,j_t$ von Knoten mit $p_h = (j_{h-1}, j_h)$ oder $p_h = (j_h, j_{h-1})$ für alle *h = 1,...,t* existiert. Ein Kette kennzeichnet man durch die Schreibweise $k = (j_0,...,j_t)$.

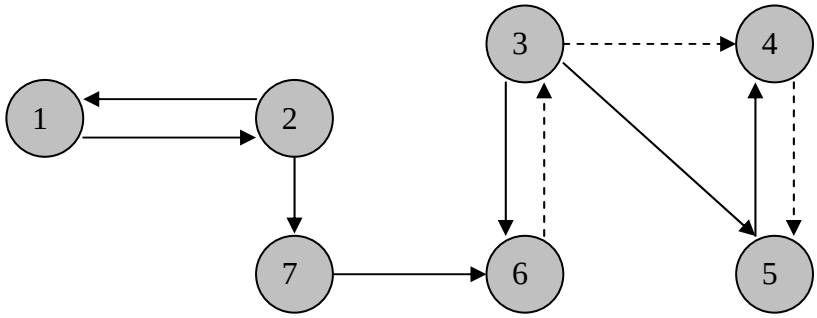


Abb. 2.8: Der Graph enthält die gestrichelte Kette *k = (5, 4, 3, 6)*. Die Begriffe Weg und Kette lassen sich analog auch auf ungerichtete Graphen übertragen.

2.10 Zyklus in einem Graphen

Ein Weg mit identischem Anfangs- und Endknoten ($j_0 = j_t$) heißt geschlossener Weg oder *Zyklus*. In Zusammenhang mit dem Traveling Salesman Problem (TSP), welches in den Kapitel 3.2.6 und 6 ausführlich behandelt wird, spricht man auch von einer Rundreise oder Tour.

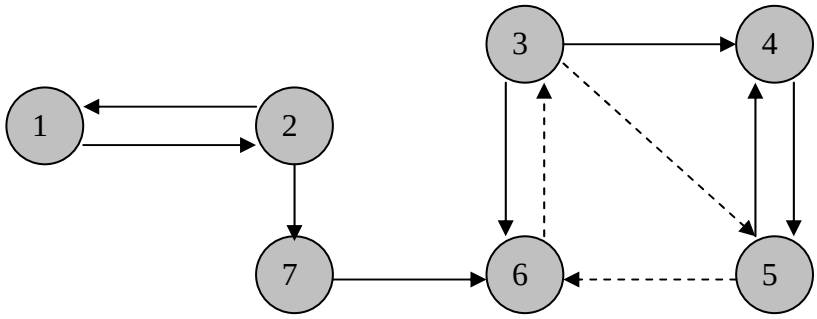


Abb 2.9: Der Graph enthält den gestrichelten Zyklus (3, 5, 6, 3).

2.11 Kreis in einem Graphen

Eine Kette mit identischem Anfangs- und Endknoten ($j_0 = j_t$) nennt man geschlossene Kette oder *Kreis*. Der Unterschied zu einem Zyklus besteht in der Knotenreihenfolge: bei einem Zyklus ergibt sich die Reihenfolge aus der Pfeilrichtung (vgl. Kap. 2.8), bei einem Kreis ist auch die entgegengesetzte Richtung erlaubt (vgl. Kap. 2.9).

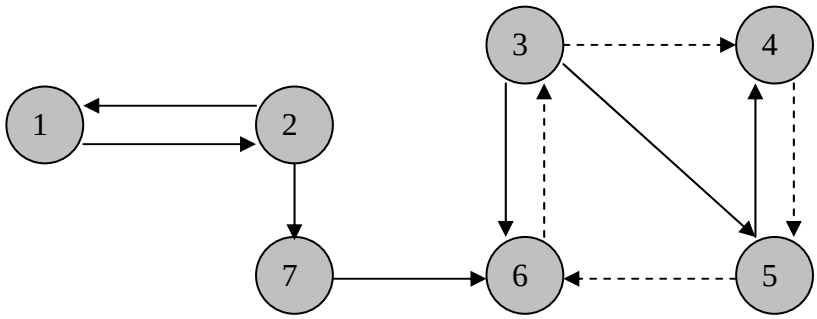


Abb. 2.10: Der Graph enthält den gestrichelten Kreis (6, 5, 4, 3, 6).

2.12 Zusammenhängender Graph

Ein ungerichteter oder gerichteter Graph heißt *zusammenhängend*, wenn jedes Knotenpaar von *G* durch mindestens eine Kette verbunden ist.

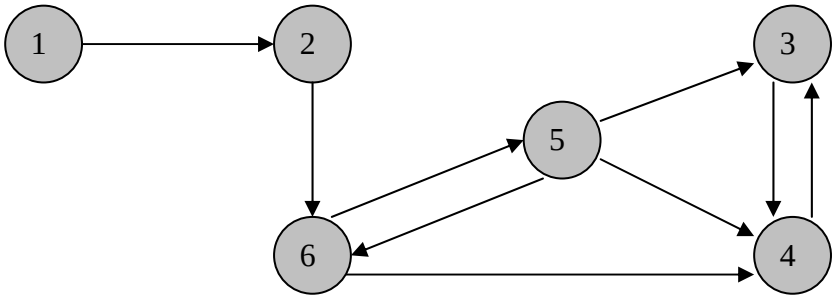


Abb. 2.11: Zusammenhängender gerichteter Graph

2.13 Baum

Ein ungerichteter zusammenhängender Graph, der keinen Kreis enthält, heißt *Baum*. Ein Baum mit *n* Knoten besitzt demzufolge *n-1* Kanten. Innerhalb eines Baumes unterscheidet man zwischen Quelle (Wurzel), Ästen (Zweigen) und Senken (Blättern).

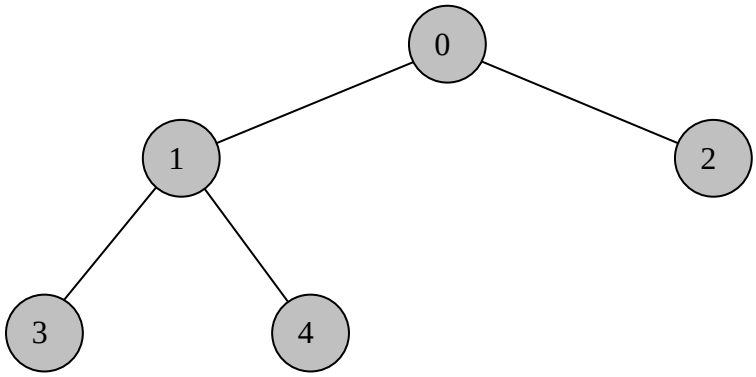


Abb. 2.12: Knoten 0 stellt die Quelle des Baumes dar, die Nachfolger der Quelle (1 und 2) nennt man Äste und die Knoten ohne Nachfolger (2, 3 und 4) Senken.

2.14 Kürzester Weg in einem Graphen

Sei $G = (V, E, c)$ ein gerichteter bewerteter Graph und $w = (j_0,..., j_t)$ ein Weg von G. Dann ist die Summe aller Pfeilbewertungen $c(w) = \sum_{h=1}^{t} c_{j_{h-1}j_h}$ die Länge des Weges w. Einen Weg w_{ij}^* von G bezeichnet man als *kürzesten Weg* von Knoten i nach Knoten j, falls in G kein anderer Weg von i nach j mit $c(w_{ij}) < c(w_{ij}^*)$ existiert. $c(w_{ij}^*)$ nennt man in diesem Fall die *kürzeste Entfernung* von i nach j in G. Analog dazu lassen sich kürzeste Ketten definieren. Die Begriffe sind ebenfalls auf ungerichtete Graphen übertragbar.

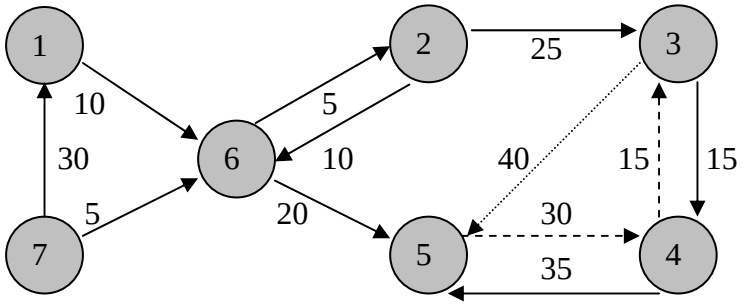


Abb. 2.13: Der gestrichelte Weg w_1 = (5, 4, 3) ist der kürzeste Weg von Knoten 5 nach Knoten 3. Die kürzeste Entfernung ergibt sich zu $c(w_1) = 30 + 15 = 45$. Für die Rückfahrt von Knoten 3 nach Knoten 5 findet sich hingegen der noch kürzere gepunktete Weg w_2 = (3, 5) mit einer Entfernung von $c(w_2) = 40$.

2.15 Übertragung

Wie man anhand des vorangegangenen Abschnittes sieht, finden sich in der Graphentheorie wichtige Analogien zu dem Wegeproblem der Caritas-Station.

Das Tempelhofer Straßennetz kann in den Formalismen der Graphentheorie als zusammenhängender gerichteter bewerteter Graph betrachtet werden. Straßenkreuzungen bzw. Punkte, an dem zwei oder mehrere Straßensegmente aufeinander treffen, heißen Knoten. Die Verbindungsstücke zwischen den Knoten nennen sich gerichtete Kanten oder Pfeile.

Diese Nomenklatur der ’berechenbaren’ Begriffe gestattet die mathematische Behandlung der Problemstellung. Ein abstrakter Graph mit Knoten und Kanten stellt den adäquaten Ereignisraum dar, in dem nach einem kürzesten oder schnellsten Weg gesucht werden kann.

Hinweis zu den Abbildungen in dieser Arbeit

Die in den folgenden Kapiteln dargestellten Abbildungen stammen von verschiedenen Quellen. Der größte Teil von ihnen wurde (wie die Abbildungen dieses Kapitels) von mir erstellt, in solchen Fällen erfolgen keine Quellenangaben.

Für die Abbildungen, die ich von anderen Autoren übernommen bzw. modifiziert habe, erscheint die Quellenangabe aus Platzgründen nicht direkt unter der Abbildung, sondern im Abbildungsverzeichnis (vgl. Seite A).

3. Optimierungsprobleme

3.1 Die Klassen P, NP und NP-vollständig

Bei der Lösung von Optimierungsproblemen stellt sich die Frage nach der Größe des Rechenaufwandes und der Höhe des Speicherplatzbedarfes. Mit Hilfe der Komplexitätstheorie können Aussagen darüber getroffen werden, wie diese beiden Faktoren im ungünstigsten Fall zu veranschlagen sind. Daraus ergibt sich eine folgende Klassifizierung, die alle Optimierungsprobleme im wesentlichen in drei Gruppen unterteilt (nach ZELEWSKI 1989, S. 51 ff.):

- Als *P* bezeichnet man Probleme, die von einem Algorithmus mit polynomialem Aufwand gelöst werden können. Diese Probleme werden als „leicht“ eingestuft, die dazugehörigen Algorithmen nennt man effizient.
- Als *NP* bezeichnet man Probleme, die von einem nicht-deterministischen Algorithmus mit polynomialem Aufwand gelöst werden können. (Für deterministische Verfahren ist der Aufwand hingegen nicht polynomial, sondern exponentiell beschränkt.)
- Als *NP-vollständig* bezeichnet man Probleme, für die bisher kein Algorithmus bekannt ist, der auch das schwierigste Problem gleichen Typs mit polynomialem Aufwand lösen könnte. Sie sind nur mit exponentiellem Aufwand lösbar und werden als „schwer“ eingestuft. Die dazugehörigen Algorithmen nennt man ineffizient.

Doch was bedeutet eigentlich mit „polynomialen Aufwand“ lösbar?

Angenommen, die Größe eines Optimierungsproblems ist von nur einem Parameter n abhängig (z.B. beim TSP von der Stadtanzahl n, vgl. Kap. 3.2.6) und ein Algorithmus benötigt zur Lösung des Problems genau $5n^2 + 7n + 3$ Rechenschritte. Mit hinreichend großem n ist der Rechenaufwand proportional zum Polynom $f(n) = n^2$, man sagt er hat die Ordnung (oder Komplexität) $O(n^2)$. Ist die Funktion *f* ein Polynom von n, spricht man von polynomialem Berechnungsaufwand. In allen anderen Fällen, z. B. $O(2^n)$, nennt man den Aufwand exponentiell (DOMSCHKE, DREXL 1991, S.111 f.).

Beispiele aus der Klasse *P* sind Kürzeste-Wege-Probleme und lineare Zuordnungsprobleme (vgl. Kap. 3.2).

Die meisten Problemtypen der kombinatorischen Optimierung hingegen gehören zur Klasse der *NP-vollständigen* Probleme. Beispiele sind das Rucksackproblem (vgl. Kap. 3.2.3) oder das Traveling Salesman Problem (vgl. Kap. 3.2.6). Für größere Probleminstanzen können wegen des exponentiell wachsenden Rechenaufwandes nur noch heuristische Lösungsverfahren eingesetzt werden (Weiteres dazu in Kapitel 4.2).

3.2 Kombinatorische Optimierung

Probleme der kombinatorischen Optimierung treten im alltäglichen Leben in den unterschiedlichsten Variationen auf. Im Bereich von Wirtschaft, Planung und Logistik lassen sich im wesentlichen vier Hauptgruppen ausmachen, wobei stellvertretend jeweils zwei oder drei Beispiele genannt seien (nach DOMSCHKE, DREXL 1991, S. 107 f.):

- *Zuordnungsprobleme*
  - Lineares Zuordnungsproblem: *n* Arbeitern sollen *n* Tätigkeiten bei bekannten Ausführungskosten so zugeordnet werden, daß jeder Arbeiter genau eine Arbeit ausführt und der Arbeitsplan kostenminimal wird.
  - Quadratisches Zuordnungsproblem: *n* gleich große Maschinen sollen auf *n* gleich großen Plätzen so angeordnet werden, daß die Summe der Transportkosten zwischen den Maschinen minimal wird.
  - Stundenplanproblem: welcher Lehrer soll zu welcher Zeit in welchem Raum welche Klasse unterrichten?
- *Reihenfolgeprobleme*
  - Problem des Handlungsreisenden (Traveling Salesman Problem, vgl. Kap. 3.2.6)
  - Tourenplanungsprobleme: Belieferung von Kunden eines Möbelhauses durch mehrere Fahrzeuge, so daß die zurückgelegte Gesamtstrecke minimal wird.
  - Maschinenbelegungsprobleme: In welcher Reihenfolge sollen Aufträge auf einer Maschine ausgeführt werden, so daß die Summe zeitlicher Überschreitungen von zugesagten Fertigstellungsterminen minimal wird?
- *Gruppierungsprobleme*
  - Fließbandabstimmung: Zusammenfassung und Zuordnung von Arbeitsgängen zu Fließbandstationen
  - Clusteranalyse: Bildung von möglichst ähnlichen Kundengruppen hinsichtlich bestimmter Kriterien (z.B. Alter, Beruf, Einkommen, Wohnlage etc.)
- *Auswahlprobleme*
  - Rucksackproblem (vgl. Kap. 3.2.3)
  - Set Partioning- und Set Covering - Probleme: Auswahl einer kostenminimalen Menge von Auslieferungstouren unter einer großen Anzahl möglicher Touren

In den folgenden Kapiteln sollen sechs kombinatorische Optimierungsprobleme (u.a. das Rucksack Problem und das Traveling Salesman Problem) ausführlicher vorgestellt werden.

3.2.1 Map-Coloring-Problem

Eine leicht verständliche und sehr anschauliche kombinatorische Aufgabe wird dem Betrachter beim Durchblättern eines Atlanten vor Augen geführt. In einer Landkarte sind alle Länder so einzufärben, daß zwei benachbarte Länder nie die gleiche Farbe besitzen. Dieses Karten-Färbungsproblem (engl. Map-Coloring-Problem) gehört zur Gruppe der ’beschränkten Erfüllbarkeitsprobleme’ (Constraint Satisfaction Problems) und soll anhand einer kleines Beispiels mit sechs Ländern illustriert werden (vgl. Abb. 3.1).

Angenommen, es stehen nur die drei Grundfarben rot, grün und blau zur Verfügung und Land *A* und *B* sind bereits grün und rot eingefärbt. Ohne viel zu überlegen, kommt man zu dem Schluß, daß Land *E* blau gefärbt werden muß. Rot für Land *C* und grün für Land F ergibt sich gezwungenermaßen und schließlich ist nur noch Land *D* ein weißer Fleck auf der Landkarte. Man hat die Qual der Wahl: blau oder rot. Damit ist das Problem ohne eine aufwendige Suche gelöst. Ein derartiges Vorgehen wird als ’Most Constrainting Variable Heuristic’ bezeichnet (JAKIMOVSKI 1999).

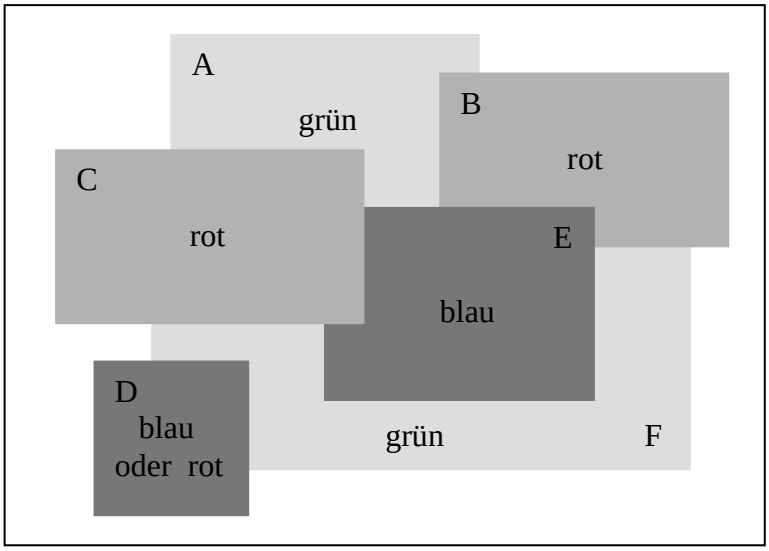


Abb. 3.1: Map-Coloring-Problem

Bei einer anderen Variante sucht man sich ein Land heraus und gibt diesem eine intuitiv gewählte Farbe, welche das Problem vielleicht besser löst. Solche Methoden sind unter dem Namen ’Least Constrainting Value Heuristic’ bekannt. (Der Begriff ’Heuristik’ und die damit verbundene Theorie werden in Kapitel 4.2 eingehend erläutert).

Nach dem außerhalb der Mathematik vielleicht bekanntesten Satz der Graphentheorie, dem Vier-Farben-Satz, reichen vier Farben aus, um jede beliebige Landkarte der Welt nach der obigen Forderung einzufärben (DIESTEL 1996, S.97 ff. u. S.122 f.).

3.2.2 Acht-Damen-Problem

Ein weiteres, einfach zu erläuterndes Beispiel ist das Acht-Damen-Problem: auf einem Schachbrett sollen acht Damen so platziert werden, daß sie sich nicht gegenseitig bedrohen. Schon der berühmte Mathematiker, Astronom und Geodät Carl Friedrich Gauss beschäftigte sich mit diesem Problem um 1850, konnte jedoch keine vollständige Lösung dafür finden (WIRTH 1986, S.156 ff.).

Für einen Computer und die erwähnte ’Least Constrainting Value Heuristic’ ist das Acht-Damen-Problem heutzutage *kein* Problem mehr; in kürzester Zeit werden alle möglichen Lösungen generiert. Die Anzahl der Damen kann mit entsprechender Vergrößerung des Schachbrettes sogar auf bis zu tausend erhöht werden (*n*-Damen-Problem).

Bei Verwendung einer sogenannten ’Minimum Conflict Heuristic’ erscheint allerdings auch diese Anzahl eher marginal: das Verfahren ist in der Lage, ein 1-Millionen-Damen-Problem in weniger als 50 Schritten zu lösen. Solche Methoden finden aufgrund der extrem hohen Leistungsfähigkeit verstärkt in Wissenschaft und Logistik ihren Einsatz.
So gelang es beispielsweise Mitarbeitern der amerikanischen Weltraumbehörde NASA mit Hilfe einer Minimum-Conflict-Strategie die Planungsphase für Beobachtungen des Weltraumteleskopes „Hubble“ extrem zu verkürzen. Der Zeitraum von drei Wochen, welcher früher notwendig war, um ein Beobachtungsintervall von einer Woche zu planen, konnte auf zehn Minuten reduziert werden (JAKIMOVSKI 1999).

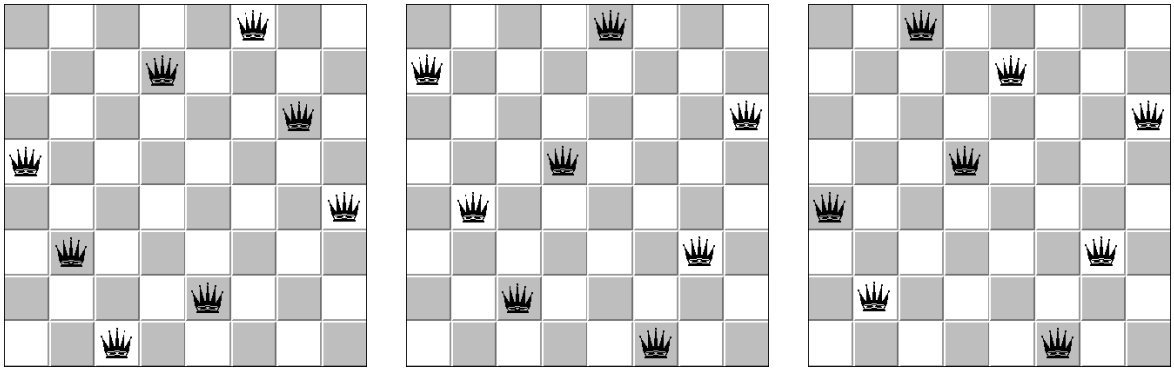


Abb. 3.2: Drei der insgesamt 92 möglichen Lösungen des Acht-Damen-Problems

3.2.3 Rucksackproblem

Das Rucksackproblem gehört zur Gruppe der binären Optimierungsprobleme (0/1-Probleme) und ist NP-vollständig (vgl. Kap. 3.1). Wie der Name bereits andeutet, existieren für die Variablen, welche die Problemstruktur beschreiben, nur zwei Zustände: Eins oder Null. Im Falle des Rucksacks (engl. knapsack) bedeutet dies: ein (unzerteilbarer) Gegenstand *muß* eingepackt werden *(1)* oder *darf nicht* eingepackt werden *(0)*.
Mathematisch läßt sich das Rucksack-Problem folgendermaßen formulieren:

$$\text{Maximiere die Zielfunktion } F = \sum_{i=1}^{n} c_i x_i \text{ unter der Nebenbedingung}$$

$$\sum_{i=1}^{n} g_i x_i \leq G \quad \text{(Gewichtsrestriktion)} \quad \text{mit } x_i \in \{0, 1\} \text{ für alle } i \,.$$

In einen Rucksack sollen also i Gegenstände mit dem Nutzen c_i so gepackt werden, daß der Gesamtnutzen der transportierten Waren maximal wird. Die Summe der einzelnen Gewichte g_i darf dabei das Fassungsvermögen G des Rucksackes nicht überschreiten. Im Gegensatz zu allgemeinen ganzzahligen linearen Optimierungsproblemen (vgl. Kap. 4.1.1.2) existiert bei Rucksackproblemen nur diese eine Nebenbedingung.

Die Problematik des Kofferpackens soll anhand eines praktisches Beispiels verdeutlicht werden. Der Leistungssportler Peter B. will die Gelegenheit nutzen und nach Ende seiner Wettkämpfe drei Wochen Urlaub in Australien machen. Peter B. hat sich durch seine langjährige Karriere im Hochleistungssport hervorragende Kenntnisse über den unerlaubten Handel mit Dopingmitteln angeeignet.
Auf dem Flughafen trifft er zufällig einen alten Freund, den Manager Paul C. Dieser fliegt während seiner Tätigkeit für eine große Firma mehrmals im Jahr nach Hause, um vom Aussterben bedrohte Tierarten ins Land zu schmuggeln.
Die beiden beschließen einen Plan, in dem sie die 20 kg Freigepäck nicht für Peters Gewichte, sondern für illegale Waren verwenden wollen. Zu Hause soll das wertvolle Gut möglichst gewinnbringend an einen Händler verkauft werden. Nach einem kurzen Streit einigen sie sich darauf, daß jeder drei Artikel seiner Wahl bestimmen darf (vgl. Tabelle 3.1 auf der nächsten Seite).

Gegenstand	Black-Swan-Federn	Krokodilzähne	Nandrolon	Anabolika	Emu-Eier	Amphetamine
Gewicht in kg	3	4	6	6	5	7
Gewinn in $	8	10	14	12	9	10
Rel.Gew.($/kg)	2,67	2,5	2,33	2	1,8	1,43
Rangfolge	1	2	3	4	5	6

Tab. 3.1: Die erste Schmuggelliste

Die beiden haben Spaß an der Sache gefunden und überlegen sogleich, was sie auf ihrer nächsten Reise mitnehmen könnten:

Gegenstand	Perftoran	Kukaburra-Füße	Testosteron	Oxygent	Wombat-Fell	Haifischflossen
Gewicht in kg	5	3	5	6	7	8
Gewinn in $	14	8	8	9	10	11
Rel. Gew. ($/kg)	2,8	2,67	1,6	1,5	1,43	1,38
Rangfolge	1	2	3	4	5	6

Tab 3.2: Die zweite Schmuggelliste (Zahlenwerte aus SCHOLL et al. 1997)

Den ersten wichtigen Schritt haben Peter und Paul schon getan. Sie haben für jeden Artikel den relativen Gewinn ausgerechnet und wissen nun, welche Ware im Verhältnis zum Gewicht den höchsten Gewinn abwirft. Doch die entscheidende Frage steht noch aus: welche Produkte sollen die Schmuggler in ihren Koffer packen, damit der Gewinn maximal wird?

Das Rucksackproblem soll an späterer Stelle weiter verfolgt werden:

- In Kapitel 4.1.1 wird das exakte Lösungsverfahren Branch & Bound beschrieben.
- In Kapitel 4.2.6 wird das heuristische Lösungsverfahren Tabu Search vorgestellt. Damit versuchen Peter und Paul eine möglichst gute Lösung für Liste 2 zu finden.
- In Kapitel 5 wird den beiden dann mit Branch & Bound endgültig auf die Sprünge geholfen. Die optimalen Kofferinhalte für beide Listen werden berechnet.

3.2.4 Königsberger Brückenproblem

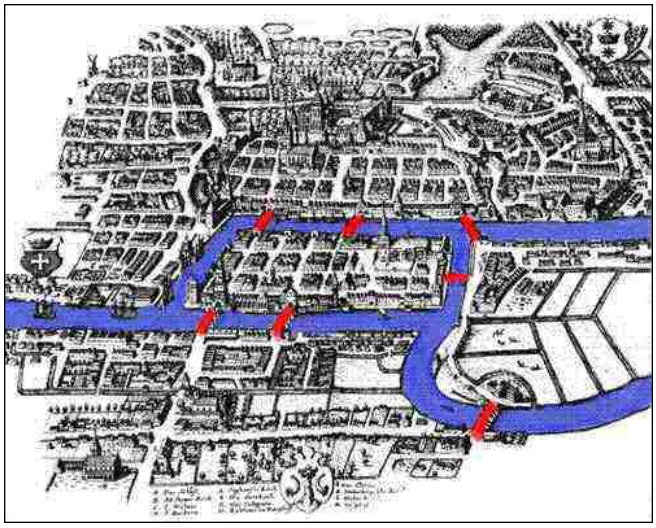


Abb. 3.3: Königsberg um 1730

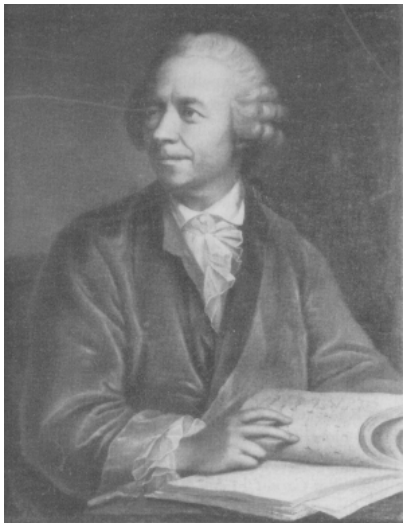


Abb. 3.4: Leonhard Euler

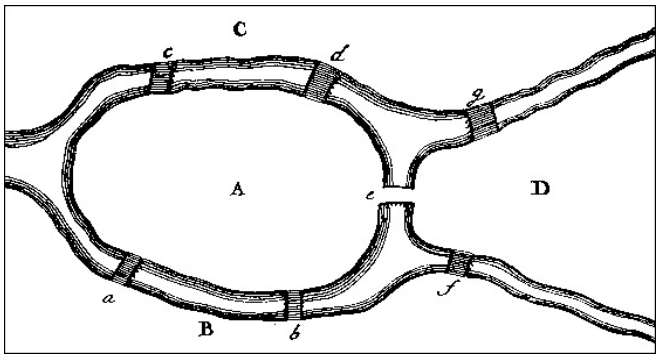


Abb. 3.5: Originalskizze Eulers

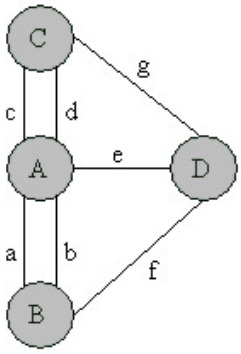


Abb. 3.6: Ein Graph ist einfacher

Die historische Ouvertüre zu allen kombinatorischen *Wege*problemen führt nach Königsberg, wo im Jahre 1736 der berühmte Mathematiker, Physiker und Astronom Leonhard Euler (1707-1783) eine geheimnisvolle Skizze (vgl. Abb. 3.5) studierte:

„Das Problem, das ziemlich bekannt sein soll, war folgendes: Zu Königsberg in Preußen ist eine Insel A, genannt ‚der Kneiphof', und der Fluß, der sie umfließt, teilt sich in zwei Arme, wie dies aus Figur 3.5 ersichtlich ist. Über die Arme dieses Flusses führen sieben Brücken *a, b, c, d, e, f* und *g*. Nun wurde gefragt, ob jemand seinen Spaziergang so einrichten könne, daß er jede dieser Brücken einmal und nicht mehr als einmal überschreite. Es wurde mir gesagt, daß einige diese Möglichkeit verneinen, andere daran zweifeln, daß aber niemand sie erhärte. Hieraus bildete ich mir folgendes höchst allgemeine Problem: Wie auch die Gestalt des Flusses und seine Zerteilung in Arme, sowie die Anzahl der Brücken ist, zu finden, ob es möglich sei, jede Brücke genau einmal zu überschreiten oder nicht" (EULER 1736, s. BORNDÖRFER et al. 1999).

Das sogenannte Königsberger Brückenproblem war also mit Sicherheit ein mathematisches Problem, jedoch auch für Euler von höchst ungewöhnlicher Natur, da „es weder die Bestimmung einer Größe erforderte, noch eine Lösung mit Hilfe des Größenkalküls gestattete“ (EULER 1736). In einem Briefwechsel mit Leibniz nennt er diese Mathematik ohne Zahlen *Geometria situs*, Geometrie der Lage. Form und Größe eines Objekts, sagt Euler, treten in den Hintergrund, wichtig ist allein die Struktur der Anordnung.

Wie hat nun Euler das Brückenproblem gelöst? Der erste Schritt war eine Abstraktion: Euler verwandelte die Stadtskizze Königsbergs (vgl. Abb. 3.3) in einen Graphen mit Knoten und Kanten (vgl. Abb. 3.5 u. 3.6). Dabei bemerkte er schnell, daß die Anzahl der Kanten, mit denen ein Knoten verbunden ist, der sogenannte Grad, eine wichtige Rolle spielt. Als Ergebnis formulierte er zwei schlichte Sätze (EULER 1736):

- Theorem I : *Die Anzahl der Knoten mit ungeradem Grad ist gerade.*
- Theorem II : *Einen Spaziergang gibt es genau dann, wenn höchstens zwei Knoten ungeraden Grad haben.*

Damit war das Problem auch schon gelöst. Die Königsberger konnten jetzt also guten Gewissens einen sonntäglichen Spaziergang durch ihre Altstadt wagen. Die beiden Eulerschen Theoreme waren übrigens die ersten Beiträge zur topologischen Graphentheorie und ebneten den Weg für eine ganz neue Disziplin innerhalb der Mathematik

3.2.5 Hamiltonsches Kreisproblem

Der irische Mathematiker Sir William Rowan Hamilton (1805-1865) erfand 120 Jahre später das Ikosaederspiel, welches eine gewisse Ähnlichkeit mit dem Brückenproblem aufweist. Der zu dem Spiel gehörige Graph (vgl. Abb. 3.7) besitzt 20 Knoten, deren Buchstaben Orte wie Brüssel, Canton, Delhi, ..., und Zanzibar darstellen.

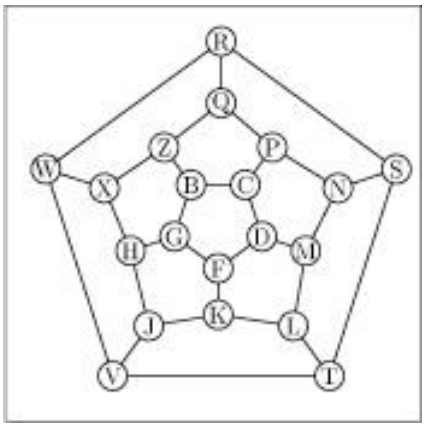


Abb. 3.7 Ikosaederspiel

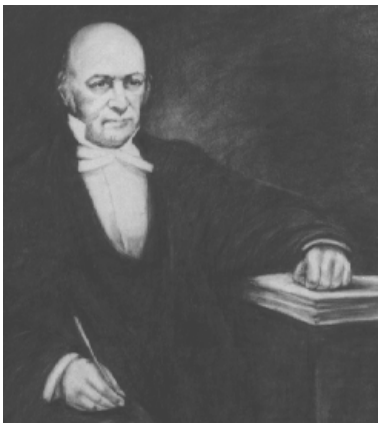


Abb. 3.8 W. R. Hamilton

Die Aufgabe des Spielers ist es, ausgehend von einem frei wählbaren Startplatz alle Städte in beliebiger Reihenfolge zu besuchen und am Ende in den Anfangsort zurückzukehren. Einzige Bedingung dabei ist, daß jede Stadt nur genau einmal durchfahren werden darf.
Die Frage, ob eine solche zyklische Rundreise innerhalb eines Graphen möglich ist, wird als Hamiltonsches Kreisproblem bezeichnet (engl. Hamilton Cicuit Problem, kurz HCP). Ist dies der Fall, nennt man die Tour einen Hamiltonkreis. Das HCP gehört ebenfalls zur Gruppe der NP-vollständigen Probleme, ist also nur schwer und aufwendig zu lösen. BORNDÖRFER et. al. (1999) schreiben dazu : „Um zu einem Ergebnis zu gelangen, gibt es keine bessere Lösungsmethode, als einfach alle Möglichkeiten durch Enumeration auszuprobieren. Höchstwahrscheinlich geht es nicht besser, ein endgültiger Beweis ist allerdings trotz größter Anstrengungen nicht erbracht.“
Hamilton selbst hatte für sein Spiel ein Lösungsverfahren erarbeitet, für das er sogar die Anfangsorte vorschreiben konnte. Allerdings ließen sich auch durch diese Variante die Verkaufszahlen nicht in die Höhe treiben. Eher das Gegenteil war der Fall: die Spielfreude wurde den Lesern schon durch Lektüre des ersten Kapitels der Spielanleitung verdorben, in dem Hamilton höchst kompliziert erklärt, warum sein *Ikosaeder*spiel auf einem *Dodekaeder* (vgl. Abb. 3.9) stattfindet.

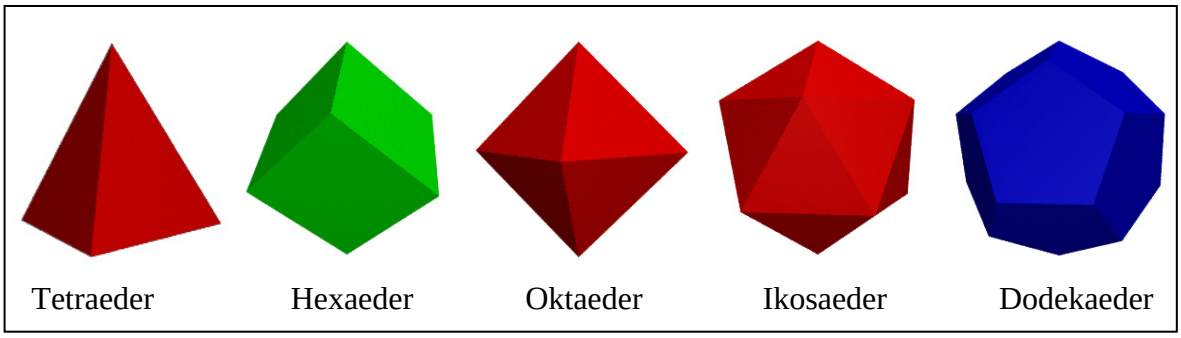


Abb. 3.9: Die fünf platonischen Körper (Seitenansicht)

3.2.6 Traveling Salesman Problem (TSP)

Abb. 3.10 TSP-Knotenpunkt

Quo vadis?

Der Handlungsreisende Herr K. hat von seiner Firma den Auftrag bekommen, in einigen deutschen Städten ein neues Produkt zu präsentieren. Er soll jeden Ort genau einmal aufsuchen, das Produkt vorstellen und am Ende in seine Heimatstadt, in der er die Reise begonnen hat, zurückkehren. Wie soll Herr K. fahren, um den kürzesten Weg für seine Rundreise zu finden oder mit anderen Worten: in welcher Reihenfolge muß er die Städte besuchen, damit die Länge seiner Tour minimal wird?

Dies ist das berühmte Traveling Salesman Problem oder kurz TSP. Das TSP ist eines der meistdiskutierten Optimierungsprobleme und kann als Prototyp einer ganzen Klasse von Problemen gelten. Da sich das eingangs geschilderte Wegeproblem der Berliner Caritas-Station (vgl. Kap. 1.1) weitgehend als ein solches TSP darstellen läßt, kann also ein direkter Bogen von der Theorie zur Praxis (und umgekehrt) gespannt werden.

Überall dort, wo es gilt, unter Kombinationen von sehr vielen Möglichkeiten die günstigste herauszusuchen, lassen sich Analogien zum TSP finden. Für die unzähligen Anwendungen in Industrie, Wirtschaft, Logistik und Wissenschaft seien hier einige Beispiele genannt (nach GRÖTSCHEL, PADBERG 1999):

- Zuweisung von Ladung und Personal an Lastwagen oder Frachtflugzeuge
- Maschinenbelegung
- Fahrplangestaltung
- Entwurf von Mikrochips
- Kapitalanlageplanung
- Gestaltung von Kommunikationsnetzen

3.2.6.1 Anwendungsbeispiel

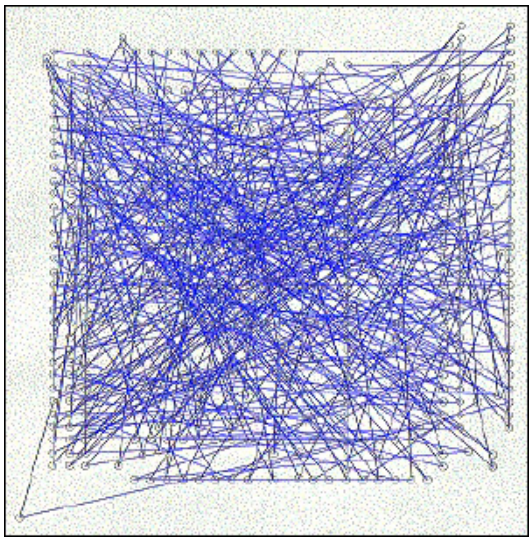


Abb. 3.11 Willkürliche Reihenfolge

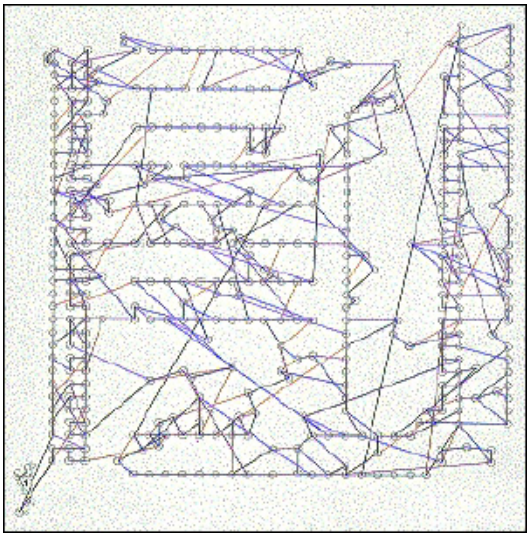


Abb. 3.12 Zwischenergebnis

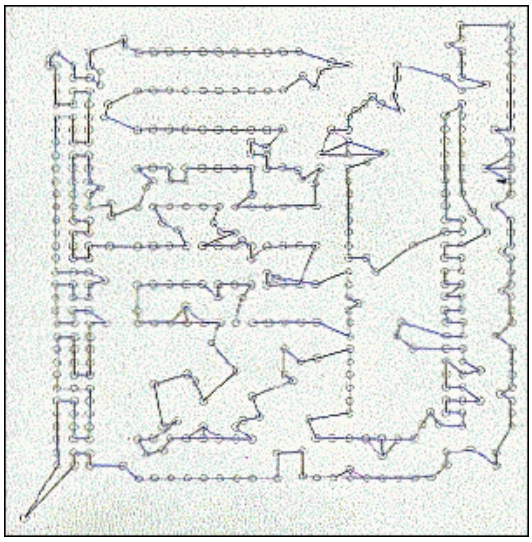


Abb 3.13 Fast schon das Optimum

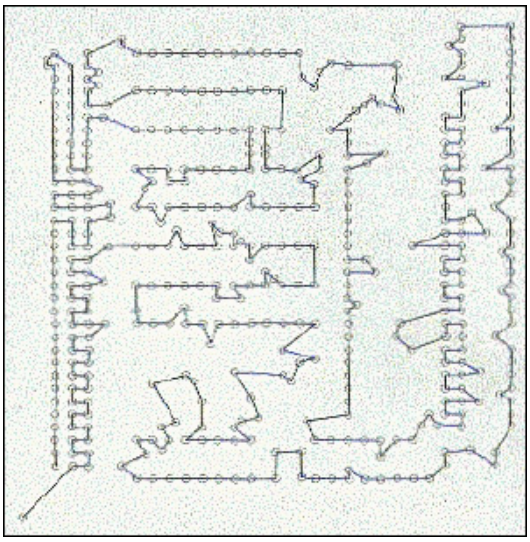


Abb. 3.14 Exakte kürzeste Rundreise

Das Problem des Handlungsreisenden in einer industriellen Anwendung, vorgestellt von MARTIN GRÖTSCHEL. In möglichst kurzer Zeit sind 442 Löcher in eine Platine zu bohren Der Gesamtweg, den der Bohrkopf zwischen den vorgegebenen Lochkoordinaten zurückzulegen hat, soll also minimiert werden. Von links nach rechts: ein Weg der jede Station in willkürlicher Folge genau einmal berührt (Abb 3.11), ein Zwischenstadium der Optimierung (Abb. 3.12), ein mit dem Sintflut-Algorithmus in ca. 450.000 Austauschschritten optimierter Weg mit einer Gesamtlänge von 130,05 cm (Abb. 3.13) sowie die von OLAF HOLLAND gefundene mathematisch exakte Lösung, die mit einer Gesamtlänge von 128,75 cm nur unwesentlich kürzer ist (Abb. 3.14). Die Ruheposition des Bohrkopfes (Start und Ziel) ist jeweils in der linken unteren Ecke (DIEDRICH et al. 1996).

3.2.6.2 Graphentheoretische Beschreibung

Die Anordnung der zurückzulegenden Wegstrecken und aufzusucheneden Orte kann durch einen bewerteten vollständigen Digraph *G = (V, E, c)* mit *n* Knoten repräsentiert werden. Dabei stellt die Menge der Knoten *(V)* die zu besuchenden Orte und die Menge der Kanten *(E)* zwischen zwei Knoten die zu befahrenen Wegstrecken dar. Die Bewertung der Strecken kann durch einen Kostenfaktor erfolgen, z.B. die Länge des Weges in Kilometern oder die Fahrtzeit in Minuten. Gesucht ist schließlich ein kürzester geschlossener Weg (oder Zyklus), in dem jeder Knoten genau *einmal* enthalten ist.

3.2.6.3 Mathematische Formulierung

Zur mathematischen Formulierung werden die Variablen $x_{ij}\,(i, j = 1,...,n)$ verwendet:

$x_{ij} = 1$ falls nach Knoten i unmittelbar Knoten j aufgesucht wird

$x_{ij} = 0$ in allen anderen Fällen

Damit läßt sich das TSP für einen bewerteten Digraphen *G = (V, E, c)* mit *n* Knoten und der Kostenmatrix $C = (c_{ij})$ (vgl. Kap. 3.2.6.5) wie folgt beschreiben:

$$\text{Minimiere } F(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \text{ unter den Nebenbedingungen}$$

$\sum_{j=1}^{n} x_{ij} = 1$ *(für i = 1,...,n)* und $\sum_{i=1}^{n} x_{ij} = 1$ (für *j = 1,...,n*) mit $x_{ij} \in \{0,1\}$ für alle *i, j = 1,...,n.*

Zusätzlich zu den obigen Forderungen müssen Nebenbedingungen zur Vermeidung von Kurzzyklen eingehalten werden. Kurzzyklen sind geschlossene Wege, die nur einen Teil der Knoten des Graphen enthalten. In einem Graph mit *n* = 9 Knoten können z.B. die folgenden Kurzzyklen auftreten:

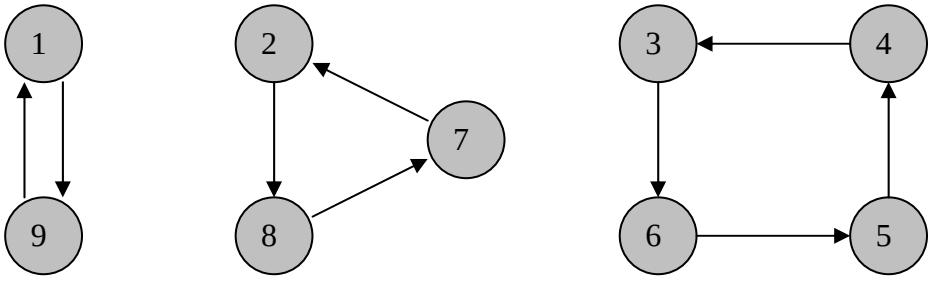


Abb. 3.16: Die Wege w_1 = (1, 9, 1), w_2 = (2, 8, 7, 2) und w_3 = (3, 6, 5, 4, 3) sind Kurzzyklen.

Um solche Kurzzyklen zu verhindern, muß für jede Permutation $(i_1, i_2, ..., i_k)$ von k der n Knoten [mit k = 2, 3, ..., n / 2 , falls n gerade und k = 2, 3, ..., (n-1) / 2 , falls n ungerade] folgende Bedingung gelten: $x_{i_1 i_2} + x_{i_2 i_3} + ... + x_{i_k i_1} \leq k - 1$.

Bereits ein Graph mit n = 14 Knoten besitzt etwa drei Millionen solcher Restriktionen (DOMSCHKE, DREXL 1991, S.110). Die Zyklusbedingungen sind *ein* Grund dafür, daß das TSP für große Städtezahlen so schwer zu lösen ist.

3.2.6.4 Rundreise

Sei $G = (V, E, c)$ ein bewerteter Digraph mit n Knoten. Einen Zyklus p, der jeden Knoten von G genau einmal enthält, bezeichnet man als *Rundreise* oder Hamiltonschen Zyklus von G. In einem vollständigen Graph mit n Knoten existieren (n-1)! / 2 mögliche Rundreisen. Die Bestimmung einer *kürzesten* Rundreise von G führt zum TSP.

3.2.6.5 Kostenmatrix

Die Kantenbewertungen eines Graphen können in einer Kostenmatrix abgelegt werden. Sie läßt sich folgendermaßen definieren:

Sei $G = (V, E, c)$ ein bewerteter Digraph mit n Knoten, dann bezeichnet man die n x n – Matrix $C = (c_{ij})$ [mit $c_{ij} = c(i, j)$, falls $(i, j) \in E$ und $c_{ij} = \infty$ sonst] als *Kostenmatrix* von G.

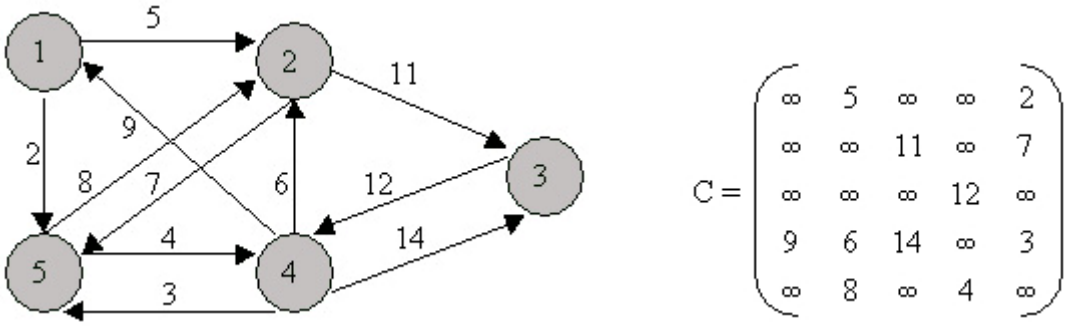


Abb. 3.17 Bewerteter Digraph und die dazugehörige (asymmetrische) Kostenmatrix

3.2.6.6 Symmetrisches / Asymmetrisches TSP

Ist die Kostenmatrix eines vollständigen Digraphen symmetrisch (es gilt $c(i, j) = c(j, i)$ für jedes Pfeilpaar), spricht man von einem symmetrischen TSP. Ansonsten liegt ein asymmetrisches TSP vor. Vollständige ungerichtete Graphen liefern in jedem Fall ein symmetrisches TSP.

4. Optimierungsverfahren und Algorithmen

Was ist ein Algorithmus? „Im anschaulichen Sprachgebrauch versteht man unter einem *Algorithmus* eine Vorschrift, nach der gegebene Größen (*Eingangswerte*) durch ein endliches System von Regeln (*Operationen*) in eindeutig bestimmter Reihenfolge in andere Größen (*Ergebnisse*) umgeformt werden.

Bei näherer Analyse des Begriffes sind drei wesentliche Bestimmungsstücke zu erkennen:

a) Ein System von Regeln.
b) Ein diskreter (algorithmischer) Prozeß, der zu vorgegebenen Eingangswerten Folgen von Zwischenergebnissen erzeugt.
c) Eine Abbildung (Verhaltensfunktion), die eine Zuordnung zwischen Eingangswerten und Ergebnissen (ohne Berücksichtigung der Zwischenergebnisse) herstellt“ (nach BRONSTEIN, SEMENDJAJEW 1979).

Etwas kompakter formuliert stellt ein Algorithmus „die endliche Folge von eindeutig bestimmten Elementaranweisungen dar, die den Lösungsweg eines Problems exakt und vollständig beschreiben“ (LACKA, PENIUTOWSKA 1996).

Eine Zusammenschau der verschiedenen Klassen von Optimierungsverfahren und der dazugehörigen Algorithmen bietet das Modell von GOLDBERG (nach REICH, GLEIßENBERGER 1996):

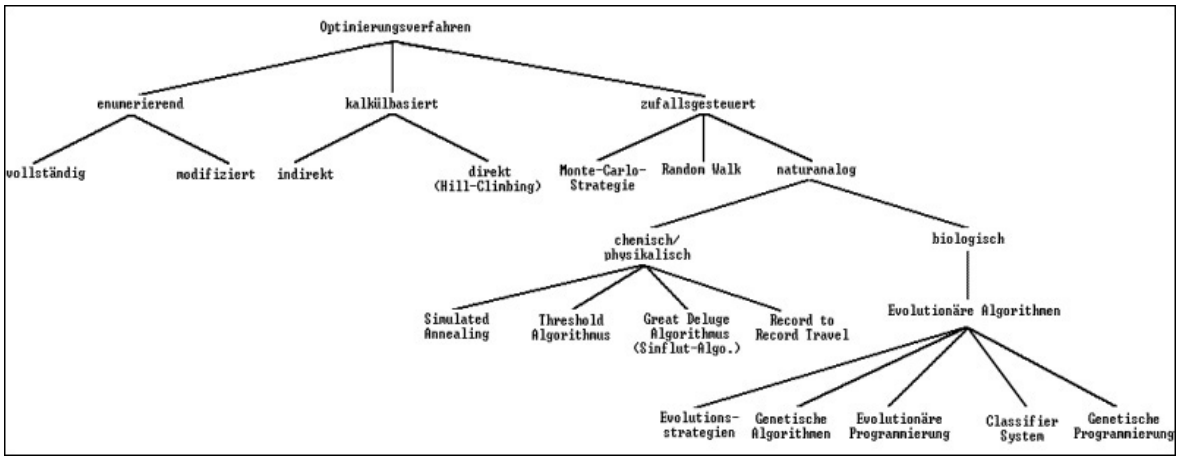


Abb. 4.1: Hierarchie der Optimierungsverfahren

4.1 Exakte Verfahren

Exakte Verfahren zur Lösung von Optimierungsproblemen liefern in endlich vielen Schritten ein *globales Optimum*. Man bekommt also (bis hin zu gewissen Problemgrößen) in jedem Fall das bestmögliche Ergebnis. Dies ist der entscheidende Vorteil gegenüber anderen Methoden. Bei Anwendung von z. B. heuristischen Verfahren (mehr dazu in Kapitel 4.2) besteht die Gefahr, daß der Algorithmus in einem lokalen Optimum stecken bleibt und nicht mehr herausfindet. Zudem kann dort keine Aussage darüber getroffen werden, wie weit das gefundene Ergebnis von der optimalen Lösung entfernt ist.
Die exakten Verfahren lassen sich in zwei größere Gruppen unterteilen (nach DOMSCHKE, DREXL 1991, S. 112):

1. Schnittebenenverfahren
   - Das ursprüngliche Optimierungsproblem wird zuerst zu einem größeren Problem erweitert und vereinfacht, z.B. durch Lockern oder Weglassen von restriktiven Nebenbedingungen. Für das vergrößerte Problem, welches alle Lösungsmöglichkeiten des Ausgangsproblems enthält, versucht man mit Hilfe geeigneter Schnitttechniken ein optimales Ergebnis zu finden. Dieses Optimum löst dann auch das eingebettete, kleinere Problem.
2. Entscheidungsbaumverfahren
   - Vollständige Enumeration

     Alle in Frage kommenden Lösungen werden systematisch und nacheinander untersucht. Diese Methode benötigt ein Maximum an Rechen- und Zeitaufwand und kann im Falle von NP-vollständigen Problemen nur für kleinere Probleminstanzen angewendet werden.
   - Begrenzte Enumeration

     Das Ausgangsproblem wird schrittweise in kleinere, vereinfachte Teilprobleme zerlegt. Innerhalb der einzelnen Teilprobleme muß dann nur noch eine beschränkte Anzahl von möglichen Lösungen untersucht werden.

Ein bekanntes und sehr leistungsfähiges Schnittebenenverfahren ist das Branch & Cut - Verfahren, mit dem zur Zeit die besten Ergebnisse für große TSPe erzielt werden (vgl. Kap. 6.4.5). Unter den Methoden der begrenzten Enumeration ist vor allem das Branch & Bound-Verfahren zu nennen, welches eine weite Verbreitung auf allen Gebieten der Optimierung gefunden hat. Diese beiden Verfahren werden in den folgenden Kapiteln ausführlicher behandelt.

4.1.1 Branch and Bound

4.1.1.1 Verfahrensablauf

Abb. 4.2: Optimieren auf Bäumen Je beschränkter, desto besser

Das in den Sechziger Jahren entwickelte Branch & Bound-Verfahren benutzt zwei Lösungsprinzipien, aus denen sich auch der Name ableitet: *branching* und *bounding*.

Im ersten Schritt wird ein zu lösendes Ausgangsproblem P_0 so in k Teilprobleme verzweigt (*branching*), daß die zwei folgenden Bedingungen gelten:

$$(1)\ X(P_0) = \bigcup_{i=1}^{k} X(P_i) \quad \text{und} \quad (2)\ X(P_i) \cap X(P_j) = \varnothing \text{ für alle } i \neq j\,.$$

$X(P_i)$ stellt dabei die Menge der zulässigen Lösungen eines Problem P_i dar.

Die Gleichung (1) besagt, daß die Unterteilung von P_0 in k Teilprobleme so zu erfolgen hat, daß die Vereinigung der k Lösungsmengen diejenige von P_0 ergibt. Gleichung (2) fordert für alle paarweisen Durchschnitte der k Probleme die leere Menge.

Die Probleme $P_1, P_2, ..., P_k$ können dann ihrerseits wieder in kleinere Teilprobleme verzweigt werden. Diese Vorgehensweise läßt sich gut mit Hilfe eines Lösungsbaumes illustrieren:

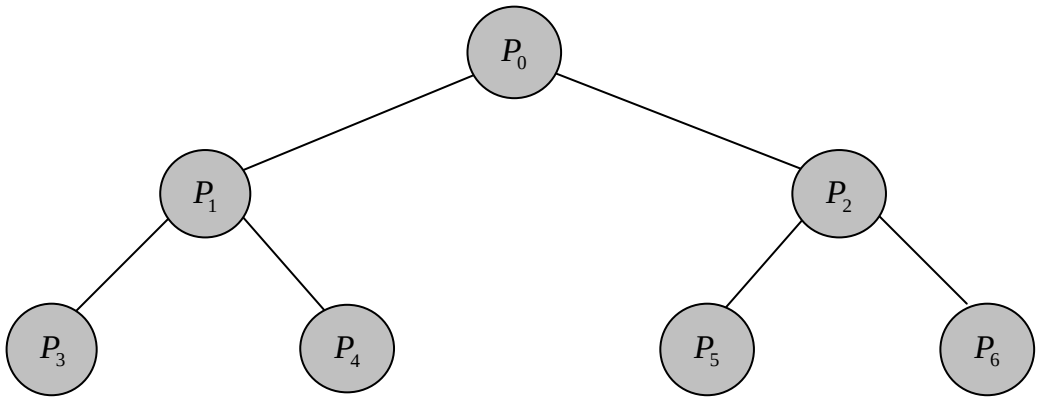


Abb. 4.3: Binärer Lösungsbaum

P_0, P_1 und P_2 sind in jeweils *k* = 2 Teilprobleme verzweigt. Eine solche Struktur wird als Binärbaum bezeichnet. Das Ausgangsproblem P_0 heißt Wurzel oder Quelle des Baumes. Dementsprechend ist P_1 die Quelle des linken Teilbaumes mit den dazugehörigen Knoten P_3 und P_4. Den Nachfolger eines Knotens nennt man Ast. Hat ein Knoten keine Nachfolger, spricht man von einem Blatt.

Der zweite Schritt des Verfahrens dient zur Beschränkung des Verzweigungsprozesses. Man berechnet Schranken für die Zielfunktionswerte (*bounding*) und entscheidet mit Hilfe dieser, ob Teilprobleme weiter verzweigt werden müssen oder nicht. Dabei kann stets eine untere Schranke $\underline{F}$ für den Zielfunktionswert einer optimalen Lösung angegeben werden. Zu Beginn des Algorithmus setzt man entweder $\underline{F} = \infty$ oder versucht, ein besseres $\underline{F}$ mittels einer Heuristik zu bestimmen. Im Laufe des Verfahrens ergibt sich die aktuelle untere Schranke dann immer aus der bisher besten bekannten Lösung.

Zusätzlich kann für jedes Problem P_i (*i = 0, 1, ..., k*) eine obere Schranke $\overline{F}_i$ für den Zielfunktionswert einer optimalen Lösung von P_i ermittelt werden. Dazu vereinfacht man das Problem P_i, indem man Nebenbedingungen lockert oder wegläßt (Relaxation). Das relaxierte (gelockerte) Problem heißt P_i' von P_i und hat die Eigenschaft $X(P_i) \subseteq X(P_i')$. Es gibt mehrere Möglichkeiten, solche Lockerungen durchzuführen: in der LP-Relaxation wird z.B. die Ganzzahligkeitsbedingung für Variablen weggelassen (diese Methode verwendet das Programm TENOR BABE, weiteres dazu in Kapitel 6). Eine andere Variante ist die Lagrange-Relaxation: man verzichtet auf Nebenbedingungen, die das Problem besonders erschweren und fügt diese gewichtet in die Zielfunktion ein.

Wann nun müssen Teilprobleme zerlegt werden und wann nicht? Wenn eine weitere Betrachtung eines bestimmtes Astes des Lösungsbaums nicht mehr notwendig ist, spricht man von einem *ausgeloteten* Problem (DOMSCHKE, DREXL 1991, S. 115).

Dabei lassen sich drei Fälle unterscheiden. Ein Problem heißt ausgelotet, wenn

1. : $\overline{F}_i \leq \underline{F}$ ist. Die optimale Lösung des Teilproblems P_i kann nicht besser sein, als die beste zulässige Lösung des Ausgangsproblem P_0.
2. : $\overline{F}_i > \underline{F}$ ist. Die optimale Lösung von P_i ist gleichzeitig die neue beste zulässige Lösung für P_0. Sie wird gespeichert und die neue untere Schranke auf $\underline{F} = \overline{F}_i$ gesetzt.
3. : P_i' keine Lösung besitzt. Damit ist auch die Menge der zulässigen Lösungen von P_i leer, es gilt $X(P_i) = \varnothing$.

In welcher Reihenfolge soll man die zu verzweigenden Probleme P_i auswählen? Es gibt mehrere Möglichkeiten, zwei wichtige sind LIFO und MUB / MLB.
Die Methode LIFO (Last-In-First-Out) wählt zum Verzweigen das zuletzt in die Kandidatenliste eingefügte Problem aus. Probleme, welche zu einem bestimmten Zeitpunkt des Verfahrens noch nicht ausgelotet sind, befinden sich auf einem Weg im Lösungsbaum, der von P_0 zum aktuellen Problem P_i weist. Die Suche ist in die *Tiefe* gerichtet (Depth First Search). Ein Vorteil dieser Methode ist, daß sich stets nur wenige Probleme in der Kandidatenliste befinden und man schnell zu einer zulässigen (jedoch relativ schlechten) Anfangslösung gelangt.
Beim Verfahren MUB / MLB (Maximal-Upper-Bound / Minimal-Lower-Bound) wird der Kandidat mit der größten (Minimierungsprobleme: kleinsten) Schranke ausgewählt. Man hofft, daß sich unter den zulässigen Lösungen von P_i auch die optimale Lösung für P_0 befindet. Die Suche richtet sich in die *Breite* (Breadth First Search). Nachteilig bei dieser Methode ist, daß viele Probleme in der Kandidatenliste abgelegt werden müssen, was mitunter zu Speicherplatzproblemen führen kann. Dafür ist die erste zulässige Lösung, die man erhält, meist schon sehr gut. In TENOR BABE sind zusätzlich zwei weitere Auswahlmöglichkeiten implementiert. Die FIFO-Regel (First-In-First-Out) benutzt zum Verzweigen jeweils das Problem, welches der Kandidatenliste als erstes zugefügt wurde. Bei der vierten Methode wird der Kandidat nach dem Zufallsprinzip bestimmt.

Schließlich muß man sich Gedanken darüber machen, nach welchen Regeln die Teilprobleme zerlegt werden sollen, was wiederum stark abhängig ist vom Aufgabentyp. Für das Rucksackproblem bildet z.B. die Sortierung nach größtem relativem Nutzen eine wirkungsvolle Methode. Dazu dividiert man die Koeffizienten aus Zielfunktion und Nebenbedingung und verzweigt dann nach der Variable mit dem größten Quotienten.
Dazu ein kleines Beispiel, gegeben sei folgendes Rucksackproblem:
Maximiere die Funktion $2x_1 + 3x_2$ unter der Nebenbedingung $4x_1 + 2x_2 \leq 10$.
Der relative Nutzen der Variablen ergibt sich zu $x_1 = 2/4 = 0{,}5$ und $x_2 = 3/2 = 1{,}5$.
Man verzweigt das Problem also zuerst nach x_2, wenn dies nicht bereits vorher geschehen ist. (Eine Binärvariable darf, entsprechend einem Knoten im Binärbaum, nur *einmal* verzweigt werden.)

4.1.1.2 Ganzzahliges lineares Optimierungsproblem

Die Theorie des Branch & Bound – Verfahrens soll anhand einer konkreten Aufgabe verdeutlicht werden. Dazu wird das folgende lineare Optimierungsproblem (LOP) mit ganzzahligen Variablen betrachtet (Zahlenwerte nach DOMSCHKE, DREXL 1991, S.115):

Aufgabe ist es, die Zielfunktion $F(x_1, x_2)$ unter Beachtung von gewissen Bedingungen so zu optimieren, daß sich für x_1, x_2 maximale Werte ergeben:

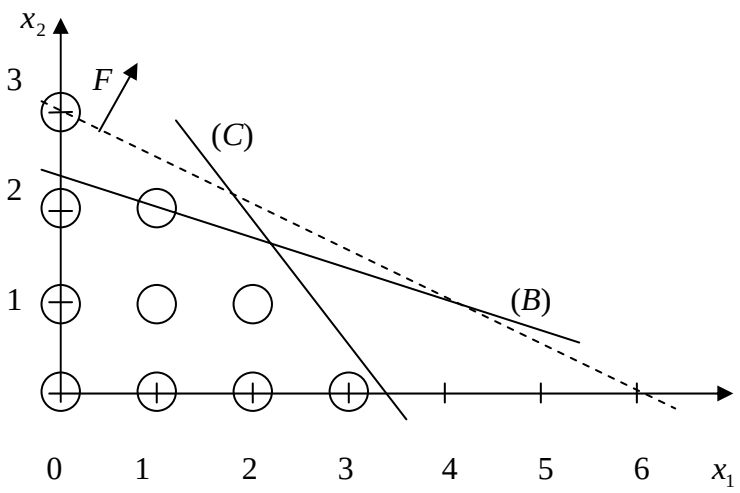
Maximiere $F(x_1, x_2) = x_1 + 2x_2$ (*A*)

unter den Nebenbedingungen

$x_1 + 3x_2 \leq 7$ (*B*)

$3x_1 + 2x_2 \leq 10$ (*C*)

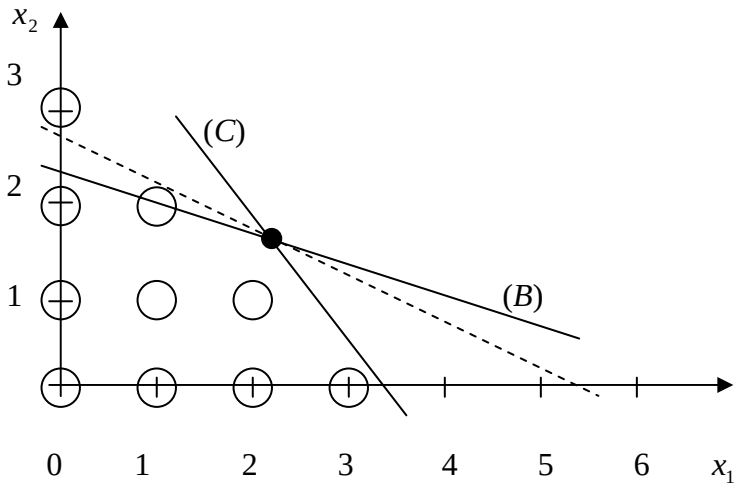
mit $x_1, x_2 \geq 0$ und $x_1, x_2 \in Z$ (*D*)



Die eingekreisten Punkte bilden die zulässige Lösungsmenge des Ausgangsproblems P_0.

Der Schnittpunkt der durchgezogenen Geraden (*B*) und (*C*) ist keine Lösung, weil dort die Bedingung der Ganzzahligkeit (*D*) nicht erfüllt wird.

Abb. 4.4: Zulässige Lösungsmenge für P_0 im Funktionsschaubild



Das Ausgangsproblem P_0 wird im ersten Schritt zu P_0' vereinfacht, indem man die Ganzzahligkeitsbedingung wegläßt (Relaxation). Der Schnittpunkt von (*B*) und (*C*) zeigt die optimale Lösung für P_0'.

Abb. 4.5: Gelockertes Problem: optimale Lösung für P_0'

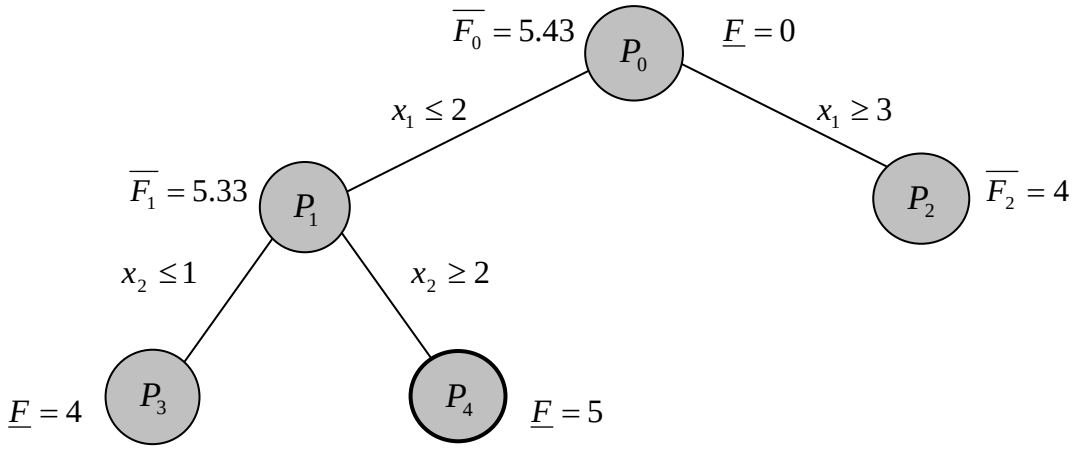


Abb. 4.6: Optimales Ergebnis für P_0 im Lösungsbaum

Wie wird die optimale Lösung für P_0 mit Hilfe von Branch & Bound gefunden? Anhand des obigen Baumdiagramms soll der Lösungsweg schrittweise beschrieben werden (nach DOMSCHKE, DREXL 1991, S. 116):

Der Ursprung des Graphen $x_1 = x_2 = 0$ (vgl. Abb. 4.4) ist eine zulässige Lösung für P_0, als Startwert wird also die untere Schranke $\underline{F} = 0$ angesetzt.

Problem P_0: Man führt eine Relaxation des Ausgangsproblems P_0 durch, indem man die Nebenbedingung der Ganzzahligkeit (*D*) wegläßt. Als optimale Lösung für das gelockerte Problem P_0' ergeben sich $x_1 = 2{,}29$ und $x_2 = 1{,}57$ mit dem Zielfunktionswert $F = 5{,}43$. (Die Werte können z.B. graphisch ermittelt werden, vgl. Abb. 4.5). Das Resultat ist zwar für P_0 unzulässig, liefert aber die obere Schranke für das Originalproblem: $\overline{F_0} = 5{,}43$. Da $\underline{F}$ kleiner ist als $\overline{F_0}$, muß P_0 verzweigt, d.h. in Teilprobleme zerlegt werden. Ausgehend von der Lösung für P_0' werden die Teilprobleme P_1 und P_2 gebildet. Für P_1 kommt neben den Bedingungen (*A*) bis (*D*) die Forderung $x_1 \le 2$ hinzu, für P_2 wird zusätzlich die Beschränkung $x_1 \ge 3$ eingeführt.

Problem P_1: P_1' entsteht wiederum durch Weglassen der Ganzzahligkeitsbedingung. Die optimale Lösung für P_1' ist $x_1 = 2$ und $x_2 = 1{,}67$ mit dem Zielfunktionswert $\overline{F_1} = 5{,}33$, welcher zugleich die neue obere Schranke ist. Da $\underline{F}$ kleiner als $\overline{F_1}$ ist, muß P_1 weiter zerlegt werden. Es entstehen die Teilprobleme P_3 und P_4 mit den zusätzlichen Nebenbedingungen $x_2 \leq 1$ bzw. $x_2 \geq 2$.

Problem P_2: Das gelockerte Problem P_2' hat die optimale Lösung $x_1 = 3$ und $x_2 = 0{,}5$ mit $\overline{F_2} = 4$. Wegen $\underline{F} < \overline{F_2}$ muß auch hier weiter verzweigt werden. Vor Durchführung dieser Maßnahne sollen jedoch zuerst die Teilprobleme von P_1 untersucht werden.

Problem P_3: Neben der Zielfunktion (*A*) und den Bedingungen (*B*) bis (*D*) müssen für dieses Problem auch die durch den Verzweigungsprozeß entstandenen Restriktionen $x_1 \leq 2$ und $x_2 \leq 1$ erfüllt werden. Der optimale Zielfunktionswert für P_3' ergibt sich zu $F = 4$ mit $x_1 = 2$ und $x_2 = 1$. Da x_1, x_2 ganze Zahlen sind, bilden die Ergebnisse gleichzeitig eine verbesserte Lösung für P_0 mit der neuen unteren Schranke $\underline{F} = 4$. P_3 ist damit ausgelotet (Fall 2, vgl. S. 25 unten).

Problem P_4: Zusätzlich zu den Forderungen (*A*) bis (*D*) sind die Beschränkungen $x_1 \leq 2$ und $x_2 \geq 2$ einzuhalten. Die optimale Lösung für P_4' und P_0 ist $x_1 = 1$, $x_2 = 2$ und $F = 5$. Damit hat man eine weitere Verbesserung für das Optimum des Ausgangsproblems gefunden. Die neue untere Schranke heißt $\underline{F} = 5$ und P_4 ist ebenfalls ausgelotet (Fall 2).

Zum Schluß das offen gebliebene **Problem** P_2: man sieht, daß die obere Schranke $\overline{F_2} = 4$ kleiner ist, als die aktuelle untere Schranke $\underline{F} = 5$ und weiß somit, daß ein optimales Ergebnis für P_2 nicht besser sein kann, als das *aktuelle* Optimum. P_2 ist damit auch ausgelotet (Fall 1, vgl. S. 25 unten).
Da sich im Lösungsbaum jetzt keine unausgeloteten Knoten mehr befinden, kann man sicher sein, mit $x_1 = 1$, $x_2 = 2$ und $F = 5$ das *globale* Optimum für das Originalproblem gefunden zu haben.

Abschließend wird das prinzipielle Vorgehen des Branch & Bound – Verfahrens in einem Flußdiagramm zusammengefaßt (nach DOMSCHKE, DREXL 1991, S.117).

(1) = Start der Prozedur, gegebenenfalls durch Anwendung einer Heuristik

(2) = Regeln zur Bildung von Relaxationen P_i

(3) = Regeln zum Ausloten von Problemen P_i'

(4) = Regeln zur Auswahlreihenfolge der zu verzweigenden Probleme P_i

(5) = Regeln zur Bildung von Teilproblemen eines Problems P_i

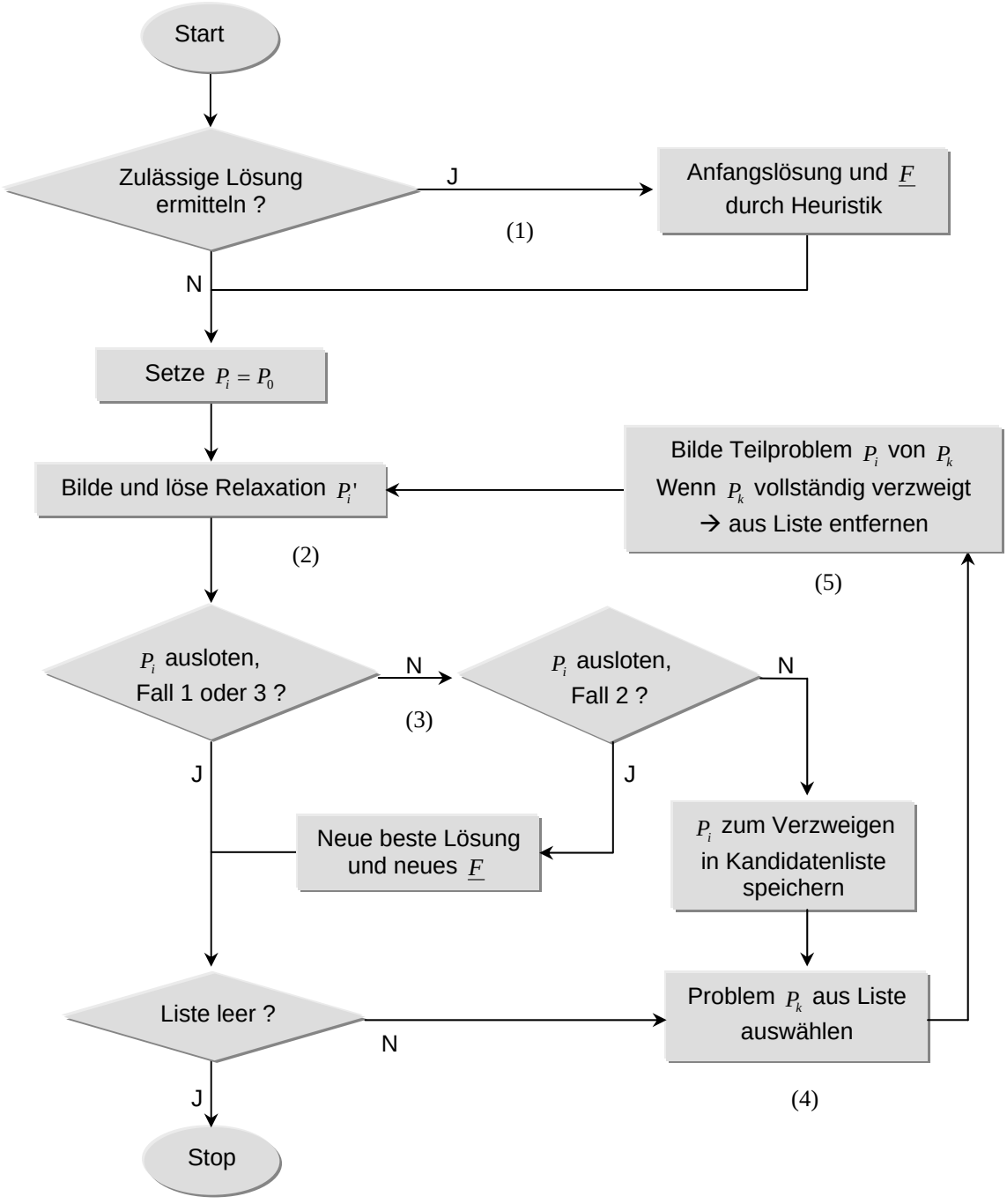
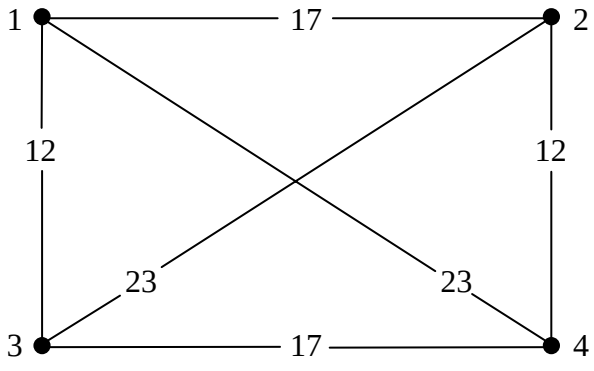


Abb. 4.7: Flußdiagramm Branch & Bound

4.1.2 Branch and Cut

Wie bereits in Kapitel 4.1 angedeutet, erweitert und vereinfacht man das ursprüngliche Optimierungsproblem zu Beginn eines Schnittebenenverfahrens so, daß das Auffinden einer unteren Schranke (minimalen Lösung) erleichtert wird. Das Ausgangsproblem wird in das vergrößerte Problem eingebettet, wobei die untere Schranke des neuen Problems auch die optimale Lösung für das alte liefert.

Unter den verschiedenen Einbettungstechniken haben sich besonders Methoden der 'Linearen Programmierung' bewährt. Eine dieser Methoden soll am Beispiel des Handelsreisendenproblems skizziert werden (nach GRÖTSCHEL, PADBERG 1999, S. 14).



Kante (1,2) : $c_{12} = 17$
Kante (1,3) : $c_{13} = 12$
Kante (1,4) : $c_{14} = 23$
Kante (2,3) : $c_{23} = 23$
Kante (2,4) : $c_{24} = 12$
Kante (3,4) : $c_{34} = 17$

Abb. 4.8: Vollständiger bewerteter Graph mit 4 Knoten und 6 Kanten

Im vorliegenden TSP mit n = 4 Knoten und 6 Kanten (vgl. Abb. 4.8) existieren genau (n-1)! / 2 = 3 mögliche Rundreisen. Jede dieser Rundreisen kann eindeutig charakterisiert werden, indem man den Kanten, die zu einer Tour gehören, den Wert Eins gibt und den restlichen Kanten den Wert Null (vgl. Abb. 4.9).

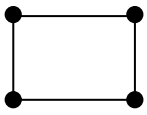
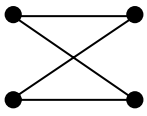
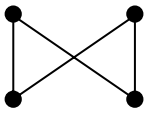
Rundreise	(1,2)	(1,3)	(1,4)	(2,3)	(2,4)	(3,4)
1: 	(1	1	0	0	1	1)
2: 	(1	0	1	1	0	1)
3: 	(0	1	1	1	1	0)

Abb. 4.9: Mögliche Rundreisen mit den Kantenattributen 1 und 0

Jede Rundreise wird durch diese Notation zu einem (Inzidenz-) Vektor, der genau 4 Einsen und ansonsten Nullen enthält. Mathematisch läßt sich damit jede Tour als ein Punkt im 6-dimensionalen Raum darstellen. Dieses Prinzip kann auf Probleme beliebiger Größe angewendet werden: ein 10-Städte-Problem wird z.B. durch einen vollständigen Graph mit $(n\text{-}1) + (n\text{-}2) + ... + (n\text{-}9) = 45$ Kanten charakterisiert. Jeder Punkt im 45-dimensionalen Raum, der eine Rundreise darstellen soll, muß also 10 Einsen und 35 Nullen enthalten.
Unter den Touren soll eine mit möglichst geringen Kosten (hier: Wegstrecke) gefunden werden. Dazu multipliziert man die Komponenten der Rundreise-Vektoren mit den entsprechenden Längeneinheiten der einzelnen Kanten (vgl. Abb. 4.8 u. 4.9) und addiert die Produkte anschließend. Als Ergebnis erhält man die Längen der Rundreisen:

$$\text{Tour 1: } \begin{pmatrix}1\\1\\0\\0\\1\\1\end{pmatrix} \cdot \begin{pmatrix}15\\11\\23\\26\\12\\17\end{pmatrix} = 55, \qquad \text{Tour 2: } \begin{pmatrix}1\\0\\1\\1\\0\\1\end{pmatrix} \cdot \begin{pmatrix}15\\11\\23\\26\\12\\17\end{pmatrix} = 81, \qquad \text{Tour 3: } \begin{pmatrix}0\\1\\1\\1\\1\\0\end{pmatrix} \cdot \begin{pmatrix}15\\11\\23\\26\\12\\17\end{pmatrix} = 72\,.$$

Der entscheidende Trick des Schnittebenenverfahrens liegt darin, daß für die Vektorkomponenten nicht nur die Werte 1 oder 0 zugelassen werden, sondern auch alle Zahlen dazwischen. Das mag auf den ersten Blick unsinnig erscheinen - wie soll man 0,38 mal von einer Stadt zur nächsten fahren? Die Menge der Lösungen wird mit dieser Maßnahme jedoch erheblich erweitert und man erreicht so die gewollte Einbettung des ursprünglichen Problems in ein größeres (GRÖTSCHEL, PADBERG 1999).
Die Zielfunktion (die Gesamtlänge der Tour) ist eine lineare Funktion der 6 Variablen. Jede Änderung einer Vektorkomponente wirkt sich proportional auf die Länge der dazugehörigen Teilstrecke und damit auch auf die Zielfunktion aus.
Aber nicht jeder Inzidenzvektor mit 4 Einsen und 2 Nullen stellt eine *zulässige* Tour dar, es sind gewisse Nebenbedingungen zu beachten:

- Alle Knoten einer zulässigen Rundreise müssen den Grad 2 besitzen, nämlich eine eingehende und eine ausgehende Kante. Diese Restriktion läßt sich in einer *linearen* Gleichung beschreiben: die Summe aller Vektorkomponenten, welche die mit einem Knoten inzidenten Kanten repräsentieren, muß 2 sein.
- Es dürfen keine Kurzzyklen entstehen. Diese Forderung kann ebenfalls als *lineare* Ungleichung formuliert werden (vgl. Kap. 3.2.6.3).

Das Problem kann also als ein lineares Optimierungsproblem (LOP) mit linearen Nebenbedingungen betrachtet werden. Für die Suche nach einem Minimum eines solchen LOPs gibt es verschiedene Verfahren, eines der erfolgreichsten ist der Simplex-Algorithmus (DOMSCHKE, DREXL 1991, S. 18 ff.) .

Die Arbeitsweise dieses Verfahrens läßt sich am besten geometrisch erläutern, wobei der Einfachheit halber eine Betrachtung im dreidimensionalen Raum genügen soll. Alle Punkte (bzw. Vektoren) dieses Raumes mit Koordinaten zwischen 0 und 1 bilden einen Einheitswürfel der Kantenlänge 1, dessen linke untere Ecke im Koordinatenursprung liegt. Die Ecken des Würfels werden durch Vektoren dargestellt, deren Komponenten sämtlich 0 oder 1 sind. Dies gilt im 6- oder 45-dimensionalen Raum genauso, bloß hat man es dort mit wesentlich mehr Ecken zu tun.

Eine lineare Ungleichungs-Nebenbedingung in einem *n*-dimensionalen Körper bewirkt, daß nicht mehr alle, sondern nur noch ausgewählte Kanten betrachtet werden. Das Gebilde wird zu einem von ebenen Flächen begrenzten Körper zurechtgeschnitten. Nimmt man weitere Nebenbedingungen hinzu und betrachtet alle zulässigen Punkte, die diese Bedingungen erfüllen, so läßt sich der Körper zu einem sog. konvexen Polytop verallgemeinern. Das Minimum einer linearen Zielfunktion wird auf einem Polytop stets in einer Ecke vermutet. Der Simplex-Algorithmus sucht nach dem Greedy-Prinzip (vgl. Kap. 4.2.1) die Ecken des Polytops ab, bis er keine bessere Lösung mehr findet und die zuletzt besuchte Ecke ist dann das gewünschte Minimum.

Allerdings handelt es sich bei dem gefundenen Ergebnis nur um das Minimum des erweiteren Problems, der vom Simplex-Algorithmus gelieferte Lösungsvektor enthält in der Regel keine ganzzahligen Koordinaten. Das Minimum des ursprünglichen Problems ist noch immer unbekannt. Man muß den Körper also weiter beschneiden, und zwar solange, bis das kleinste Polytop erreicht ist, das sämtliche Inzidenzvektoren von Touren enthält.

Das Problem bei dieser Schnitttechnik ist, daß häufig *ein* Schnitt nicht ausreicht und man in den meisten Fällen nicht weiß, *wie* man schneiden soll. Lineare Programme sind bisher für höchstens 9 Städte bekannt. Das zum TSP 9 gehörige Polytop besitzt 9 Gleichungen und 42.104.442 Ungleichungen mit 36 Variablen (entsprechend den 36 möglichen Direktverbindungen zwischen 9 Städten). Für 10 Städte sind es wahrscheinlich 51.043.900.866 Ungleichungen (mit Sicherheit nicht weniger) und für größere Probleme kennt man noch nicht einmal theoretisch ein vollständiges Sortiment von Ungleichungen (GRÖTSCHEL, PADBERG 1999, S. 7).

Trotzdem können Probleminstanzen, die weit über 10 Städte hinaus gehen, mit Methoden der linearen Programmierung gelöst werden. Das erweiterte Problem wird abermals durch ein noch größeres und einfacheres ersetzt. Von den astronomisch vielen Nebenbedingungen benutzt man jetzt nur eine sehr kleine Teilmenge, d.h. der Körper wird nur sehr grob beschnitten. Nachdem ein Lösungsvektor (eine optimale Ecke des zu großen Polytops) gefunden ist, wird eine Nebenbedingung eingeführt, die zusammen mit der gefundenen Ecke gleich ein möglichst großes Stück des Körpers abschneidet.
Im nächsten Schritt aktiviert man eine weitere Teilmenge der bisher ignorierten Nebenbedingungen und erhält als Lösung z.B. eine Rundreise, welche aus Teiltouren (Kurzzyklen) besteht. Um diese unzulässige Lösung auszuschließen, wird wiederum eine zusätzliche Nebenbedingung eingeführt, die den entsprechenden Punkt vom Polytop abschneidet.
Diesen Prozeß wiederholt man solange, bis alle Nebenbedingungen berücksichtigt sind und der Lösungsvektor nur noch aus ganzen Zahlen besteht. Ein iteratives Verfahren, welches sich mit Hilfe dieser Technik der optimalen Lösung immer weiter annähert, bezeichnet man als Schnittebenen-Algorithmus (cutting plane algorithm).
Ist die Anzahl aller Ungleichungen, welche das Originalpolytop beschreiben, bekannt, liefert der Schnittebenen-Algorithmus nach einer endlichen Anzahl von Schritten mit Gewissheit das Optimum. Hat man es jedoch mit Problemen zu tun, welche die Dimension von 9 Städten übersteigen, kann der Algorithmus beendet sein, bevor eine minimale Lösung gefunden wurde.
In solchen Fällen schafft die Anwendung eines Enumerationsverfahrens Abhilfe. Wenn der Schnittebenen-Algorithmus unschlüssig, d.h. ohne eine Lösung für das ursprüngliche Problem endet, gibt es im Lösungsvektor Kanten, die weder mit 0 noch mit 1 belegt sind. Man wählt dann eine solche Kante aus und setzt ihren Wert willkürlich auf Eins, was bedeutet, daß die Tour diesen Weg nehmen muß. Im entgegengesetzten Fall weist man der Kante eine Null zu, was bedeutet, daß die Tour diesen Weg nicht nehmen darf. Von den beiden Lösungen, welche sich für die eingeschränkten und somit vereinfachten Teilprobleme ergeben, verwendet man die bessere.
Sind die Lösungen jedoch wiederum unschlüssig (nicht ganzzahlig), verzweigt sich das Problem erneut in zwei Alternativen. Dieser Ablauf entspricht dem in Kapitel 4.1.1 geschilderten Branching-Prozeß. Ein ganzer Baum von Lösungsmöglichkeiten entsteht, welcher mit geeigneten Mitteln abgesucht werden muß. Das dafür entworfene Suchverfahren stellt eine Weiterentwicklung des Branch and Bound - Verfahrens dar

und trägt den Namen ’Branch and Cut’. Mit dieser Verzweigungs- und Schnitttechnik werden zur Zeit die besten Ergebnisse für kombinatorische Optimierungsprobleme erzielt (vgl. z.B. Kap. 6.4.5, TSP-Weltrekord).

Die Leistungsfähigkeit der geschilderten Algorithmen läßt sich gut anhand der sagenhaften Rundreise des Odysseus dokumentieren, welche MARTIN GRÖTSCHEL und MANFRED PADBERG einmal unter dem Aspekt der Kostenminimierung betrachtet haben.

Gesucht war die kürzeste Tour durch insgesamt 16 antike Orte des Mittelmeerraumes, wobei die griechische Insel Ithaka den Start- und Zielpunkt bildete. Als optimale Lösung des Schnittebenenverfahrens ergab sich eine Strecke von 6859 km Luftlinie (3054 km weniger, als Odysseus mit dem Flugzeug in der von ihm gewählten Reihenfolge gebraucht hätte). Das Ergebnis wurde auf einem normalen Arbeitsplatzrechner mit nur vier Iterationsschritten in wenigen Millisekunden berechnet. Für die stumpfsinnige Enumeration aller 653.837.184.000 Möglichkeiten brauchte derselbe Rechner immerhin 92 Stunden (GRÖTSCHEL, PADBERG 1999, S. 1).

4.1.2.1 Zurechtschneiden von Polytopen

Der entscheidene Punkt des Schnittebenenverfahrens, das Zurechtschneiden der Polytope, soll abschließend an einem Beispiel skizziert werden (nach GRÖTSCHEL, PADBERG 1999, S.16). Um die Sache etwas anschaulicher zu gestalten, werden nur die beiden Dimensionen *x* und *y* betrachtet. Eine lineare Ungleichungs-Nebenbedingung wird im Funktionsgraphen durch eine Gerade dargestellt. Alle Punkte, die auf einer bestimmten Seite der Geraden liegen, sind unzulässig. Ein Vieleck (Polytop), welches durch mehrere Geraden begrenzt wird, umschließt demnach alle zulässigen Lösungen des Problems (vgl. Abb. 4.10).

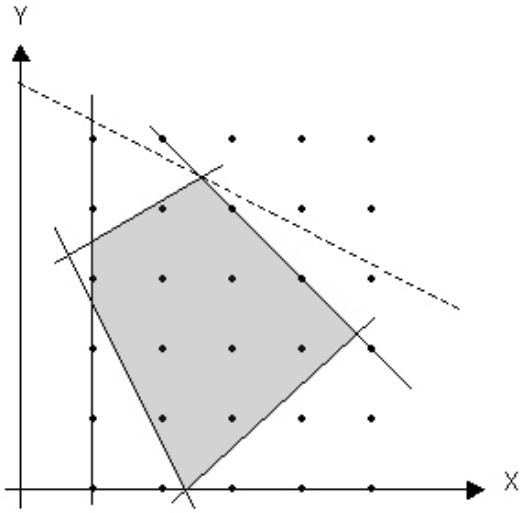


Abb. 4.10: Zulässige Lösungen des linearen Problems

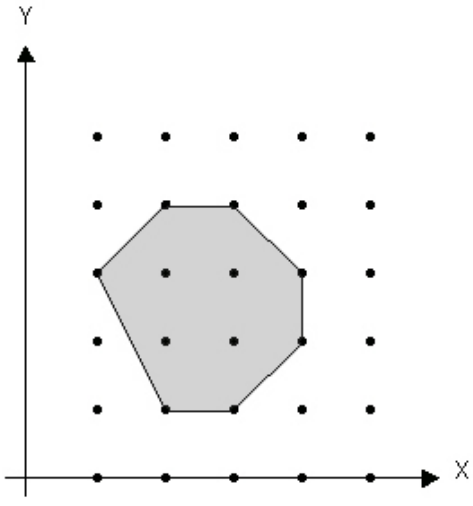


Abb. 4.11: Zulässige Lösungen des Originalproblems

Die Kostenfunktion des Problems ist ebenfalls linear. Aus diesem Grunde gibt es Geraden (für *n*-dimensionale Körper: Hyperebenen), auf denen die Kostenfunktion konstant ist. In Abbildung 4.10 symbolisiert die gestrichelte Gerade solch gleichbleibende Kosten: die Punkte rechts oberhalb sind teurer, die links unterhalb sind billiger als die Punkte auf der Geraden selbst. Die optimale Lösung (der billigste Punkt) muß eine Ecke des Polytops sein. Und zwar genau diejenige, welche die Gerade der günstigsten erreichbaren und konstanten Kosten berührt.
Das lineare Problem ist eine Erweiterung des Ausgangsproblems. Die zulässigen Lösungen des Ausgangsproblems müssen Vektoren mit ganzzahligen Koordinaten sein, was im Funktionsschaubild durch die schwarzen Punkte symbolisiert wird. Die optimale Lösung wird also unter den Gitterpunkten innerhalb des Vielecks gesucht. Aus diesem Grunde versucht man, daß Polytop so zurechtzuschneiden, daß sich nur noch zulässige Punkte mit ganzzahligen Koordinaten (und alles was dazwischen liegt) darin befinden (vgl. Abb. 4.11).

Den Vorgang des ’Zurechtschneidens’ erreicht man durch die Einführung von zusätzlichen Nebenbedingungen. Der Lösung des Problems nähert man sich schrittweise, indem in jeder Iteration eine neue Bedingung hinzugenommen und eine neues Optimum berechnet wird.

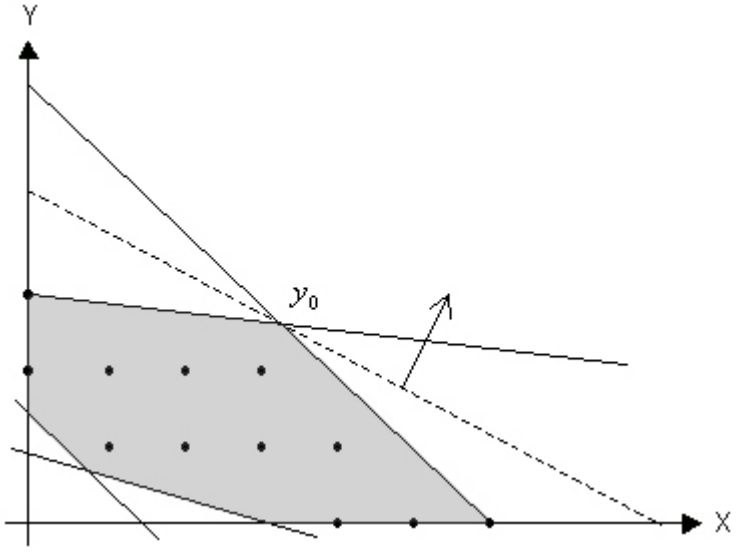


Abb. 4.12: Erster Schritt: Optimum y_0 des linearen Problems

Zu einem (nicht ganzzahligen) Optimum y_0 des linearen Problems soll eine Gerade gefunden werden, die y_0 zwar abschneidet, aber alle zulässigen Gitterpunkte auf dem Polytop beläßt (vgl. Abb. 4.12).

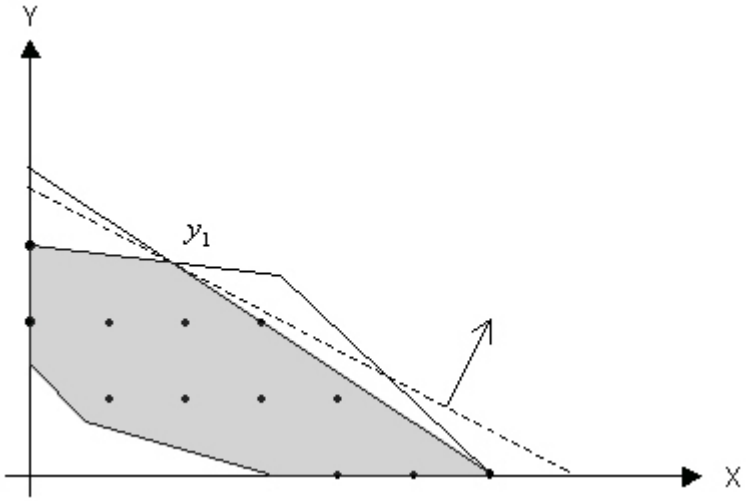


Abb. 4.13: 2. Schritt: Optimum y_1 des linearen Problems

Unter den vielen Geraden, die diese Forderung erfüllen, ist zusätzlich diejenige zu suchen, welche ein maximal großes Stück vom Vieleck abschneidet. Man löst das Problem also nochmals mit dieser weiteren Nebenbedingung und findet ein Optimum y_1, welches wieder keine (ganzzahlige) Lösung des ursprünglichen Problems darstellt (vgl. Abb. 4.13).

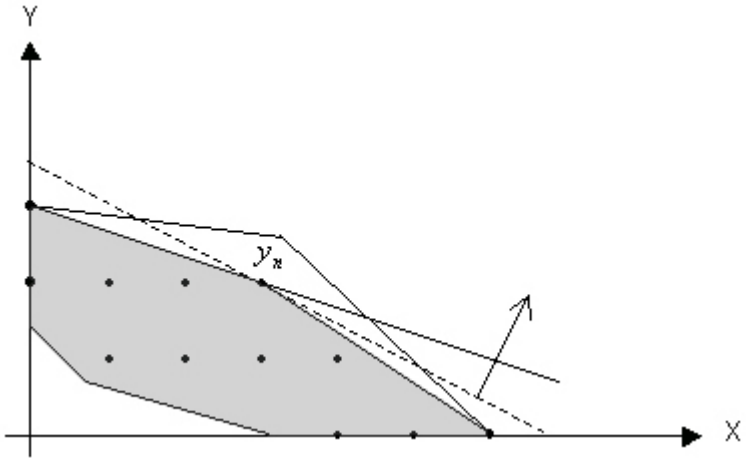
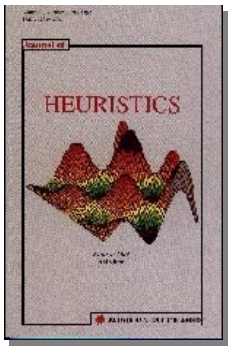


Abb. 4.14: *n*'ter Schritt: Optimum y_n des Originalproblems

Der Prozeß wird durch den Schnittebenen-Algorithmus solange wiederholt, bis das Optimum y_n mit einem Gitterpunkt zusammen fällt (vgl. Abb. 4.14). Das Ergebnis ist jetzt ganzzahlig und damit eine zulässige Lösung des Ausgangsproblems.

4.2 Heuristische Verfahren



Was sind Heuristiken? Es wird erzählt, daß Archimedes “Heureka” (Ich habe es gefunden) ausrief, als er entdeckte, daß der Schmied den König mit der Krone aus Gold betrogen hatte. Neben dem archimedischen Auftriebsprinzip haben wir dieser Geschichte vielleicht auch den Ursprung der Heuristik zu verdanken.

Abb. 4.15: Journal of Heuristics (University of Colorado, USA)

Das Wort Heuristik ist aus dem griechischen Verb *heuriskein* abgeleitet und bedeutet soviel wie *finden* oder *entdecken*. Meyer’s Lexikon definiert Heuristik als “die Kunst, wahre Aussagen zu finden, im Unterschied zur Logik, die lehrt, wahre Aussagen zu begründen.” BILL bezeichnet Heuristik als „Sammelbegriff für Faustregeln oder Erfahrungswerte, die von Menschen oder Computern eingesetzt werden, um bei einer Problemlösung die Suche einzugrenzen“ (BILL 1999 A, S. 391). Etwas freier interpretiert könnte man heuristische Verfahren auch als “intelligentes Herumprobieren” bezeichnen (OTTO, 1994). Entscheidend dabei ist, daß das Auffinden einer Lösung *nicht* garantiert wird und man keine Informationen über die Güte der Lösung bekommt.

Ein Beispiel: Herr K. möchte seinen Koffer möglichst effizient für eine lange Reise packen. Dazu wendet er folgende Heuristik an: er packt zuerst große schwere Gegenstände ein, es folgen kleinere leichte Waren und zum Schluß füllte er die verbliebenen Lücken mit formbaren Artikeln wie z.B. Socken auf. Solche einfachen, häufig empirisch gefundenen Methoden können sehr nützlich sein für Problemstellungen, die mit Hilfe von Algorithmen und Computern gelöst werden sollen.

Heuristische Verfahren lassen sich in folgende Gruppen unterteilen (nach DOMSCHKE, DREXL 1991, S. 112):

1. Eröffnungsverfahren: Man bestimmt eine zulässige Anfangslösung, welche mit anderen Methoden verbessert werden kann (mehr dazu in Kapitel 6.1).
2. Verbesserungsverfahren: Eine gegebene Anfangslösung wird schrittweise (iterativ) verbessert (Weiteres dazu in Kapitel 6.2).
3. Unvollständige exakte Verfahren (z.B. abgebrochene Branch & Bound - Verfahren)
4. Kombinationen aus 1. bis 3.

In den folgenden Abschnitten werden einige der bekannteren Verfahren vorgestellt, wobei jeweils ein Bezug zum TSP hergestellt wird. Auf diese Weise können die verschiedenen Optimierungsmethoden direkt miteinander verglichen werden.

4.2.1 Greedy-Algorithmus

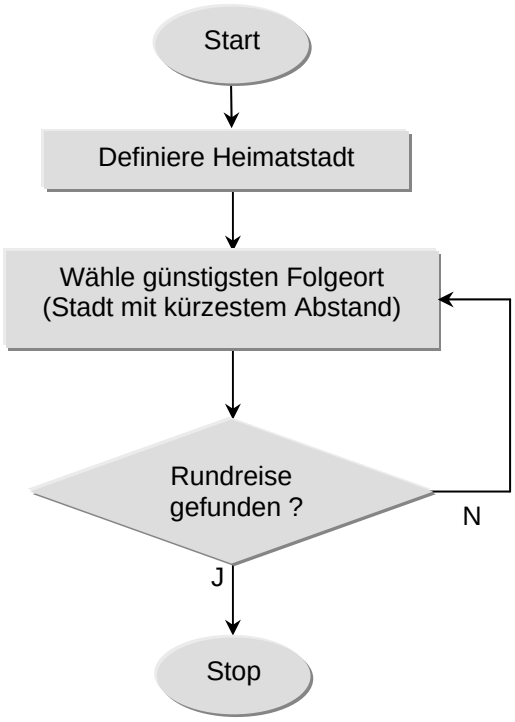


Abb. 4.16: Flußdiagramm Greedy

Das Prinzip des Verfahrens lautet: „Nimm den besten Happen zuerst“ (engl. greedy = gierig). Übertragen auf das TSP bedeutet dies: der Algorithmus wählt ausgehend von der Heimatstadt den jeweils besten Folgeort, d.h. die Stadt mit dem kürzesten Abstand (vgl. Methode des besten Nachfolgers, Kap. 6.1.1). Ob durch dieses kurzsichtige Vorgehen eine gute Gesamtlösung verbaut wird, spielt keine Rolle - wichtig ist nur die momentan günstigste Lösung. Die erzielten Ergebnisse sind deswegen nur selten optimal und können bei unvollständigen Graphen sogar in Sackgassen führen.

Anders sieht es aus, wenn man das Greedy-Prinzip zwar beibehält, aber neue Regeln zur Stadt- bzw. Kantenauswahl einführt (nach MADEJ, SIEKIERKA 1996):

1. Die Kanten bekommen Gewichte, die ihren Längen entsprechen, und werden in einer Liste nach Größe aufsteigend sortiert (zuerst die kürzeste, dann die zweitkürzeste usw.)
2. Von jeder Stadt müssen zwei Kanten ausgehen (jeder Knoten muß den Grad 2 besitzen).
3. Ein Kreis ist nur dann gestattet, wenn er alle Knoten des Problems enthält (es dürfen keine Kurzzyklen entstehen, vgl. Kap. 3.2.6.3).

Das Verfahren arbeitet mit zwei Listen: Liste I enthält die nach Gewichten sortierten Kanten, in Liste II werden die Kandidaten (Kanten) für die zu suchende Rundreise aufgenommen. Sie ist zu Beginn der Prozedur leer. In jedem Schritt wird aus Liste I der beste Kandidat (die Kante mit der geringsten Länge) herausgenommen und in Liste II gespeichert. Dabei wird stets überprüft, ob die Kantenmenge in Liste II „ausführbar“ ist, d.h. nicht gegen Regel 2 oder 3 verstößt. Kommt man an einen Punkt, wo die Menge nicht mehr ausführbar ist, wird der zuletzt hinzugefügte Kandidat wieder herausgenommen und für den Rest des Verfahrens nicht mehr verwendet. Die Prozedur endet, sobald die Kantenmenge in Liste II einen Kreis darstellt, der alle Knoten des Problems enthält.

Mit einem so arbeitenden Greedy-Algorithmus können sehr gute Ergebnisse erzielt werden, in diesem Fall macht sich blinde Gier also durchaus bezahlt.

Ein Beispiel für das obige Verfahren (nach MADEJ, SIEKIERKA 1996): In einem ebenen Koordinatensystem sind sechs Städte *A*, *B*, *C*, *D*, *E* und *F* gegeben. Die Entfernung zwischen zwei Städten A und B läßt sich über $s_{AB} = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$ ermitteln.

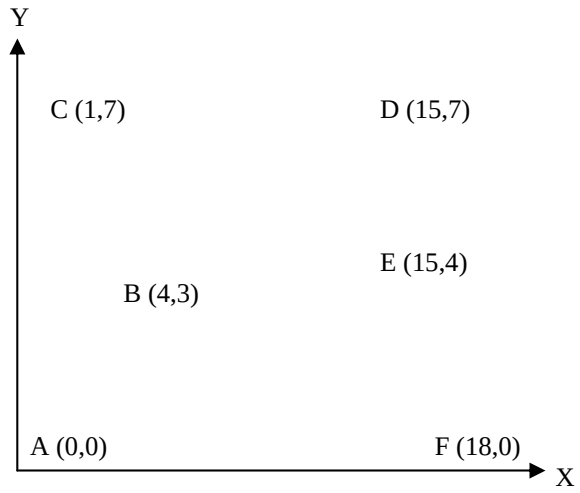


Abb. 4.17: Ausgangskonfiguration

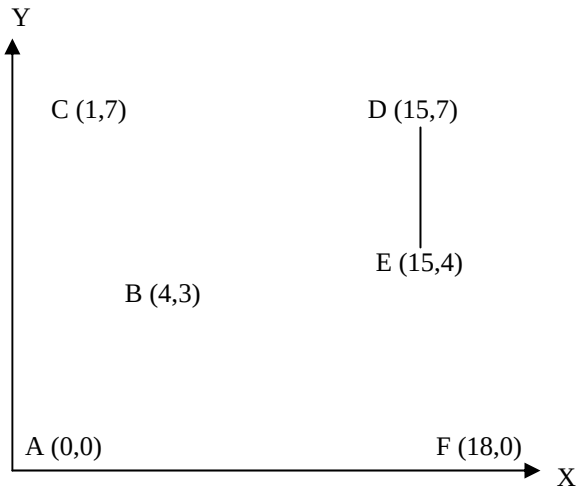


Abb. 4.18: Zwischenstadium 1

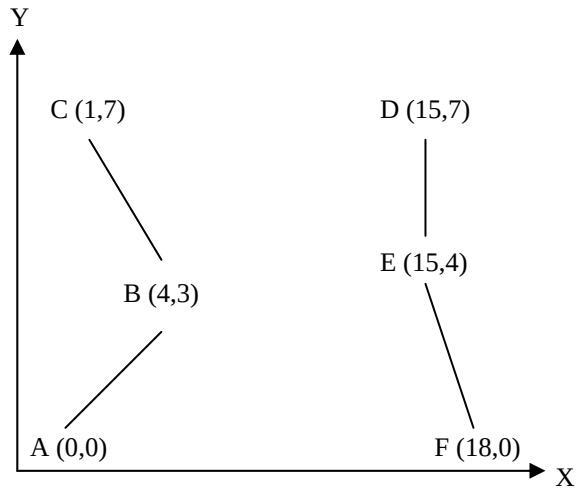


Abb. 4.19: Zwischenstadium 2

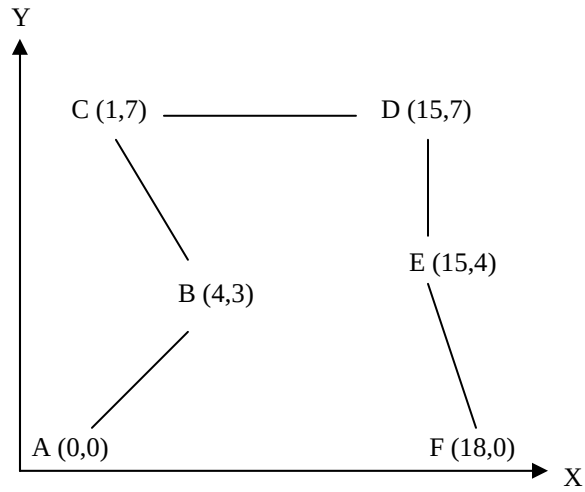


Abb. 4.20: Zwischenstadium 3

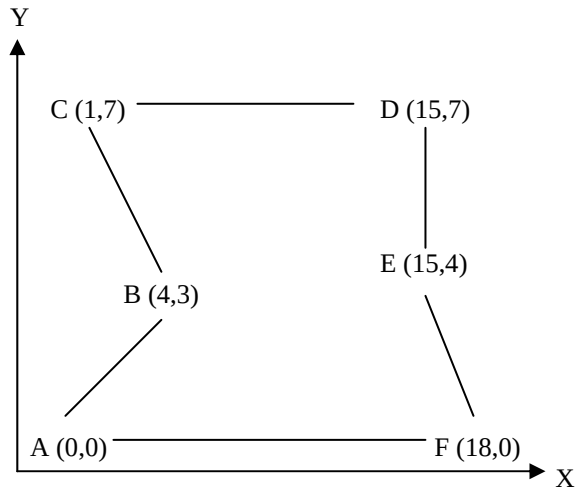


Abb. 4.21: Gefundene Rundreise

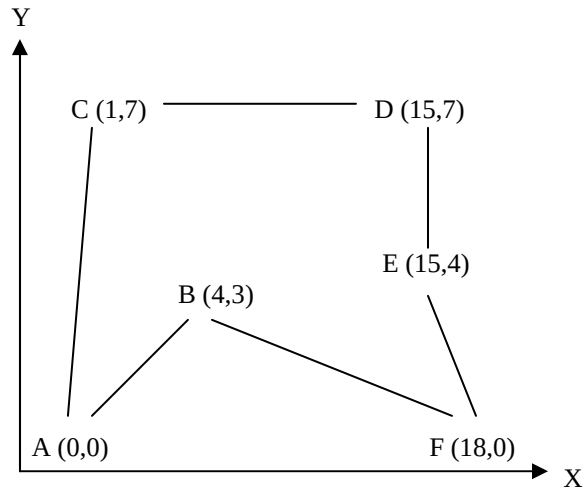


Abb. 4.22: Optimale Lösung

Erläuterungen zu den Abbildungen:

Abbildung 4.18:

- Die kürzeste Kante in Liste I ist Verbindung *DE*. Sie wird zuerst als Kandidat in Liste II aufgenommen.

Abbildung 4.19:

- Die Strecken *AB, BC* und *EF* haben alle die Länge 5. Ihre Auswahl widerspricht weder Regel 2 noch 3, sie werden ebenfalls in die Kandidatenliste aufgenommen.

Abbildungen 4.20:

- Die nächst kürzeste Strecke ist *AC*. Die Hinzunahme dieser Kante ergibt den Kurzzyklus *AB-BC-CA*, was gegen Regel 2 verstößt. *AC* wird aus Liste II wieder entfernt und für die restliche Prozedur nicht mehr verwendet.
- Der nächste Kandidat ist Kante *DF*. Die Aufnahme ergibt ebenfalls einen Kurzzyklus (*DE-EF-FD*). *DF* ereilt das gleiche Schicksal wie Kante *AC*.
- Als nächster Kandidat wird Kante *BE* von Liste I nach Liste II umgespeichert. Die Knoten *B* und *E* erhalten dadurch den Grad 3, was gegen Regel 3 verstößt. Auch *BE* spielt für die weitere Problemlösung keine Rolle mehr.
- Es folgt die Aufnahme von Kante *CD*. Keine Regel ist verletzt, *CD* wird akzeptiert.

Abbildung 4.21:

- Es existiert bereits der Weg (oder die Kette) *A, B, C, D, E, F*. Um einen geschlossenen Weg (oder Kreis) zu erhalten, fehlt noch die Verbindung von *F* nach *A*. Die Aufnahme von Kante *AF* ist regelkonform und bringt gleichzeitig die Lösung des Problems. Eine Rundreise durch die sechs Städte ist gefunden.

Abbildung 4.22:

- Die optimale Rundreise ist mit einer um vier Prozent kürzeren Streckenlänge nur unwesentlich besser als die gefundene Tour aus Abbildung 4.22.

4.2.2 Hill-Climbing

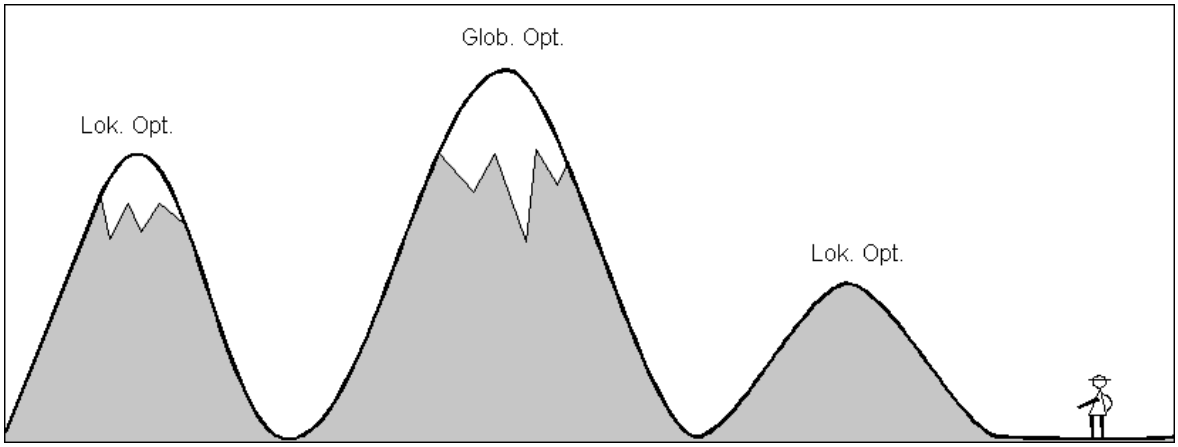


Abb 4.23: Aufgabe für den Wanderer: immer nach oben klettern

Der Hill-Climbing Algorithmus gestattet stets nur bessere Folgelösungen. Ein Wanderer, welcher den höchsten Gipfel in einem Gebirge erklimmen will, darf sich demzufolge nur nach oben bewegen. Für das TSP bedeutet dies, dass nur kürzere Rundreisen als neue Lösung akzeptiert werden. Die Gefahr, daß das Verfahren in lokalen Minima (Maxima) steckenbleibt ist offensichtlich - das globale Optimum wird nur selten erreicht.

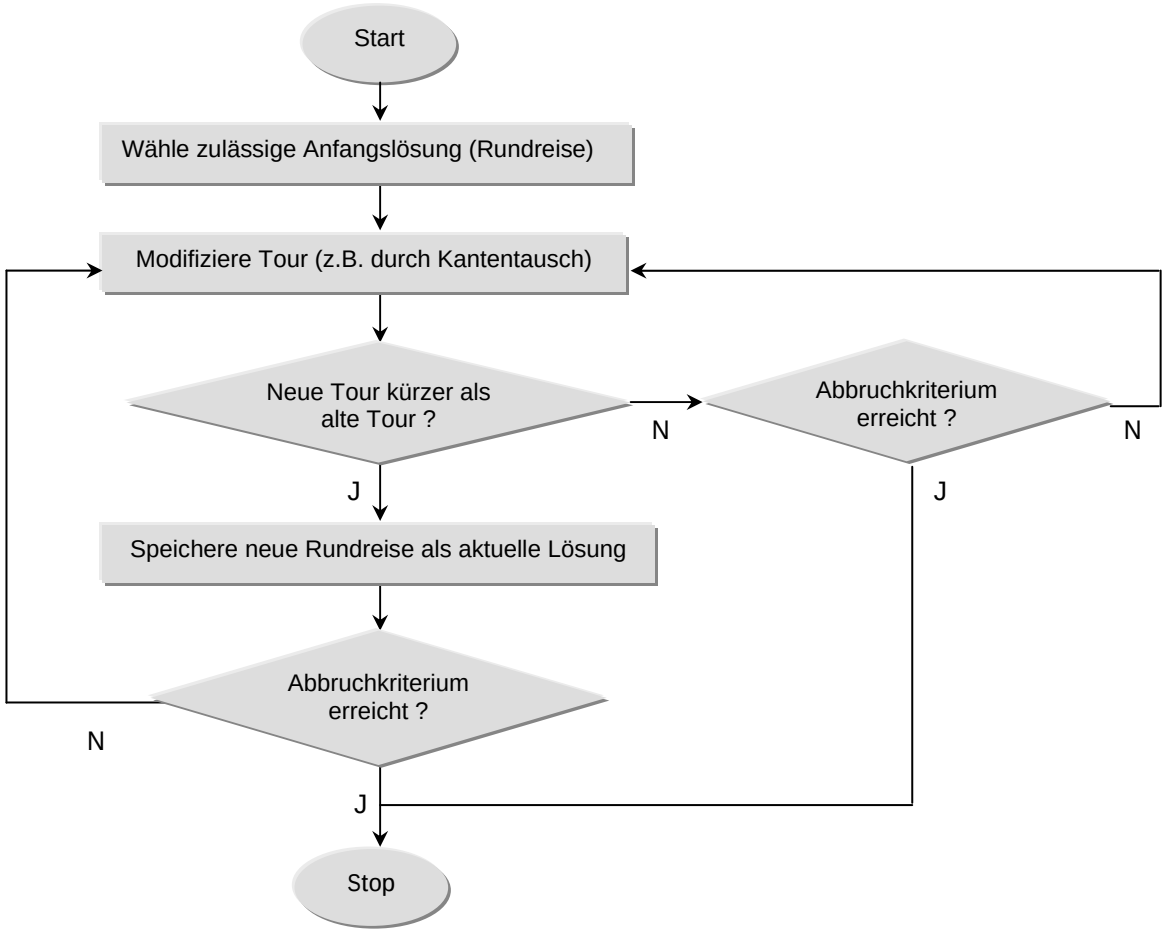


Abb. 4.24: Flußdiagramm Hill-Climbing

4.2.3 Simulated Annealing (SA)

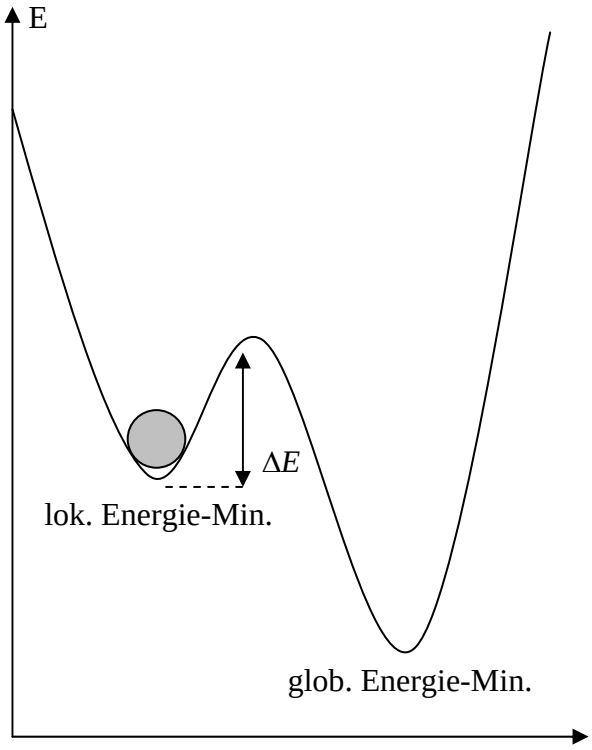


Abb. 4.25: Atom in lokaler Energiemulde

Ein interessante Analogie zum TSP findet sich in der Werkstoffphysik (OTTO 1994). Bei der Herstellung von perfekten Metall-Kristallen versucht man völlig regelmäßige atomare Gitterstrukturen zu erzeugen. Ein Kristall mit optimaler Symmetrie besitzt die günstigste Energiebilanz von allen möglichen Anordnungen. Er hat den Zustand der geringsten Gesamtenergie erreicht, da sich alle Atome auf ihren idealen Plätzen in globalen Energieminima befinden (vgl. Abb. 4.25).

Die Produktion solcher „Einkristalle“, die man z. B. zu Forschungszwecken benötigt, ist jedoch mit einigen Schwierigkeiten verbunden. Wird eine Metallschmelze zu schnell abgekühlt, verlieren die Atome rasch die nötige Bewegungsenergie und frieren in ungünstigen Gitterpositionen fest. Es bilden sich sogenannte Fehlstellen im Kristallgitter. Die Atome bleiben in einer lokalen Energiemulde hängen und schaffen es nicht, die Energiebarriere ΔE zu überwinden, um einen energetisch günstigeren Ort (einen Ort geringerer Energie) einzunehmen (vgl. Abb. 4.25).

Metalle, die bereits erstarrt sind und nicht die gewünschte optimale Gitterstruktur aufweisen, können durch nochmalige Erwärmung bis unter ihren Schmelzpunkt „getempert“ oder „ausgeglüht“ werden. Das Verfahren, im Englischen „Annealing“ genannt, ermöglicht ein Ausheilen der Fehlstellen. Die Atome in den Kristallen können einen günstigeren Gitterplatz einnehmen, da die benötigte kinetische Energie zum Überspringen der Barriere in Form von Wärme geliefert wird. Die Kristalle werden sehr langsam und unter ständiger Zuführung von Wärme aus der Schmelze abgekühlt, jegliche Art von Erschütterung muß vermieden werden. Entscheidend dabei ist die richtige Dosierung der Wärmezufuhr und die behutsame Steuerung des Abkühlungsprozesses. Einerseits sollen möglichst viele Atome ihren idealen Platz einnehmen, andererseits darf das Erstarren der Schmelze auch nicht verhindert werden.

Was hat der Ausflug in die Metallurgie mit dem TSP zu tun? Viele iterative Lösungsverfahren für das TSP entsprechen dem schnellen Abkühlen einer Metallschmelze. Nur Konfigurationen mit einer kürzeren Weglänge werden als neue Lösung akzeptiert und die Verbesserung der aktuellen Route ist wichtiger als das Erreichen eines optimalen Resultates. Das Verfahren bleibt, genau wie das Atom in der Energiemulde, in einem lokalen Minimum stecken.

Die Schlußfolgerung zur Beseitigung dieses Dilemmas liegt auf der Hand: man muß auch vorübergehende Verschlechterungen zulassen, wenn man das globale Minimum erreichen will. Genau dies wird beim Simulated Annealing - Algorithmus getan.

Mit der sog. Metropolis-Wahrscheinlichkeit $P(\Delta E) = \exp(-\Delta E / (k \times T))$ werden auch energetisch schlechtere Konfigurationen für das atomare Gitter angenommen, übertragen auf das TSP also Touren mit längeren Wegstrecken. Dabei ist ΔE die Energiedifferenz, k die Boltzmann-Konstante und T die Temperatur. (Auf theoretische Hintergründe soll hier verzichtet werden, Näheres findet sich z.B. bei BRAUSE 1991, S. 199 ff.). Um eine praktische Entscheidung herbeizuführen, läßt man den Computer eine Zufallszahl zwischen 0 und 1 berechnen. Ist sie kleiner als $P(\Delta E)$, wird die schlechtere Lösung akzeptiert (vgl. Abb. 4.26).

Wie kann der Steuerfaktor 'Temperatur', welcher das Abkühlen der Metallschmelze regelt, nun im Handelsreisenden-Problem untergebracht werden? Das Kreuzprodukt aus k und T in der obigen Formel ist genauso wie ΔE eine Energiegröße. In dem Quotienten $\frac{\Delta E}{k \times T}$ werden also zwei Energiewerte miteinander verglichen. An deren Stelle treten beim TSP Entfernungen. Der Einfachheit halber läßt man die Boltzmann-Konstante k weg und drückt die Temperatur T direkt in Längeneinheiten aus. Wählt man z.B. als Anfangstemperatur eine Entfernung, die länger ist als jede Kante des Problems, wird nahezu jede Modifizierung akzeptiert, da $P(\Delta E)$ in der Nähe von 1 liegt (exp(0) = 1).

Durch diesen Kunstgriff können anstelle von idealen Einkristallen optimale Rundreisen gezüchtet werden. THOMAS OTTO (1994) schreibt dazu: „Die Stationen der Reise verhalten sich wie Atome in der Schmelze. Mit sinkender Temperatur wird die Schmelze zähflüssiger, nur noch kleine Variationen sind zugelassen. Bei T = 0 erstarrt die Lösung, es sollte eine nahezu optimale Lösung erreicht sein".

Wenn man dem SA-Verfahren zu akzeptablen Ergebnissen gelangen und Aussagen über die Güte der Lösungen machen will, bleibt einem das Experimentieren mit unterschiedlichen Abkühlgeschwindigkeiten und Temperatur-Verweildauern nicht erspart.

Ist die Abkühlung vorsichtig genug vorgenommen worden, kann man immerhin mit „guten“ Ergebnissen rechnen. Wie gut die erzielten Resultate wirklich sind, also wie weit entfernt vom globalen Optimum, läßt sich mit dieser Methode allerdings nicht ermitteln – es handelt sich um eine Heuristik.

Das Simulated Annealing – Verfahren zusammengefaßt in einem Flußdiagramm:

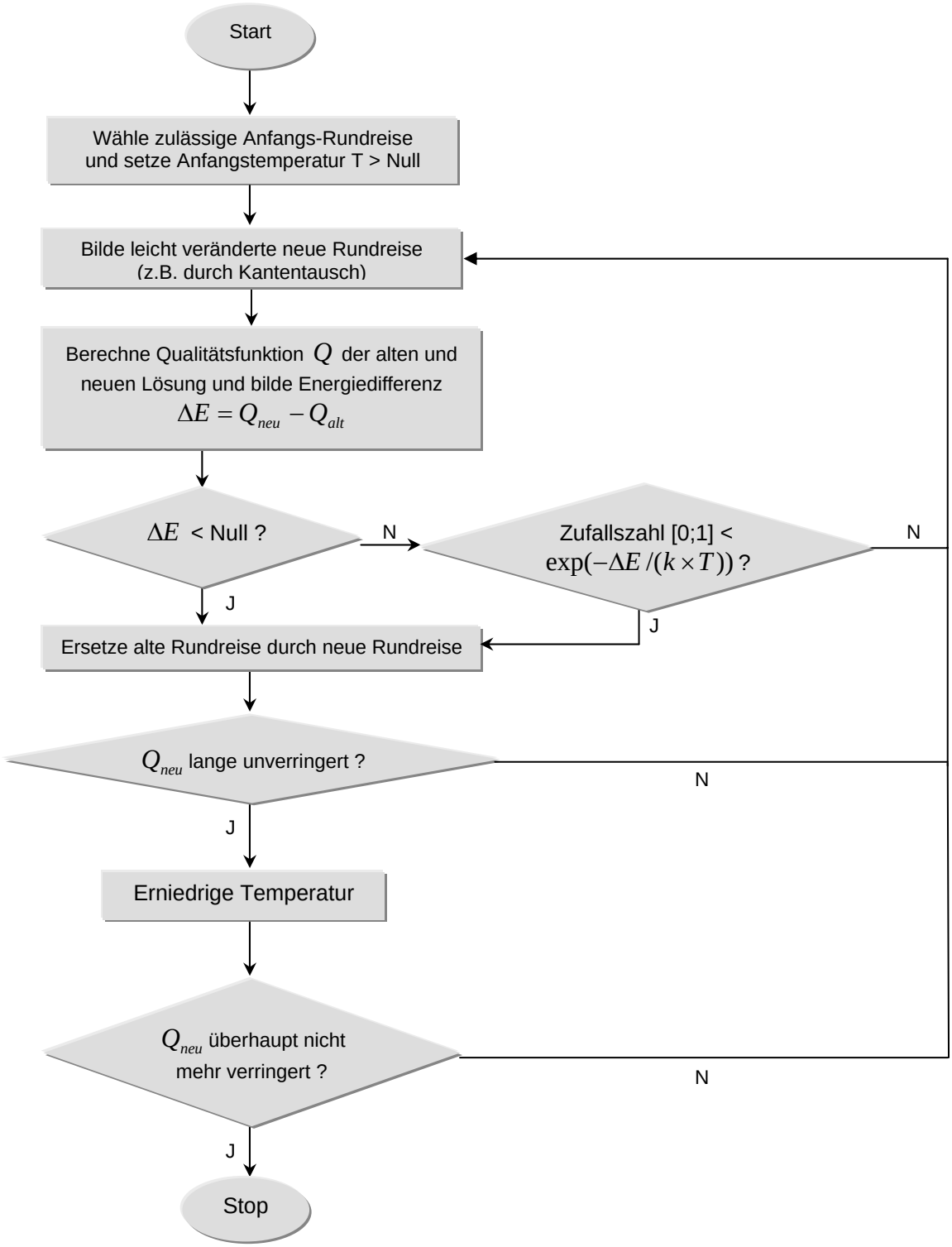


Abb. 4.26: Flußdiagramm Simulated Annealing

4.2.4 Threshold Accepting (TA)

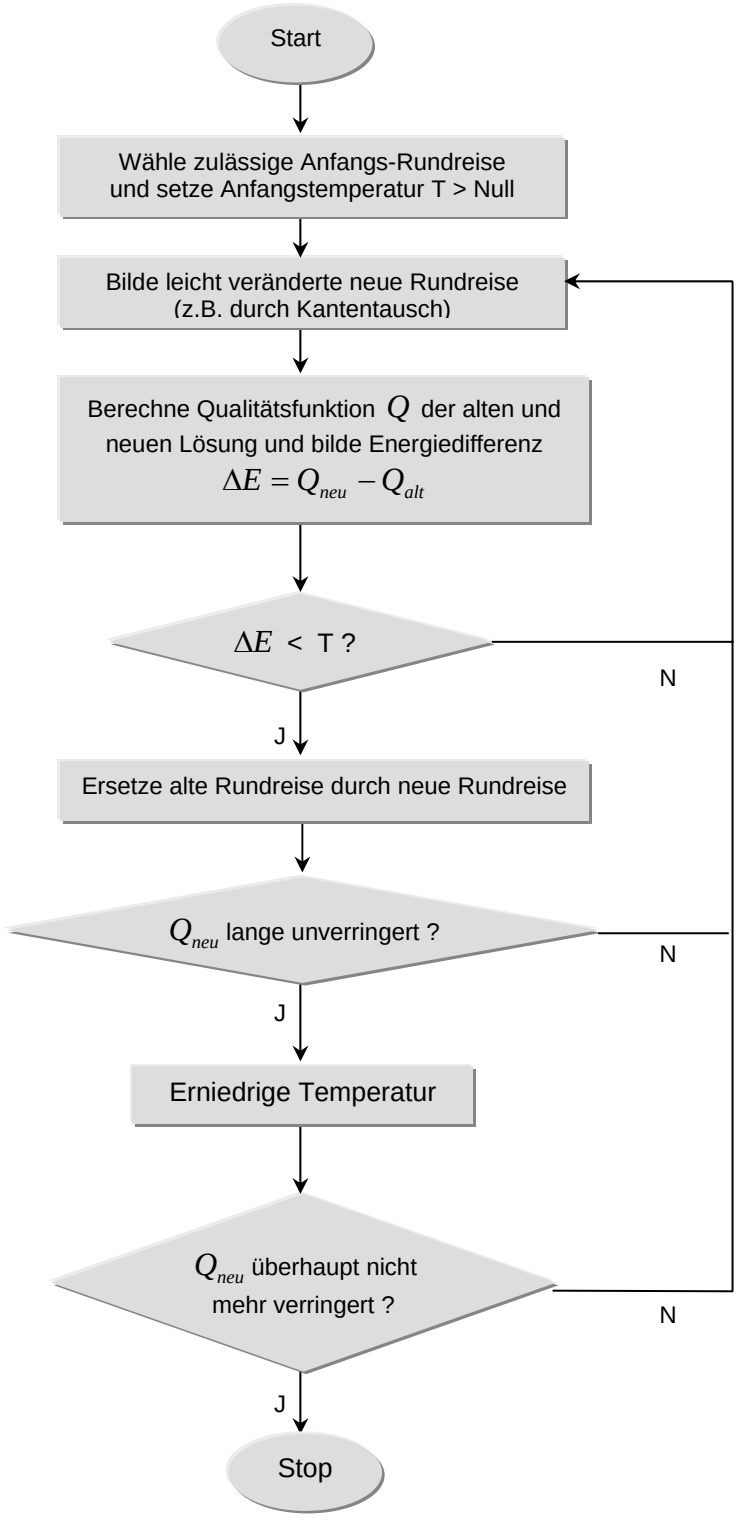


Abb. 4.27: Flußdiagramm Threshold Accepting

Das 1990 von den IBM-Mitarbeitern G. Dueck und T. Scheuer veröffentlichte TA-Verfahren weist gegenüber SA eine kleine, aber sehr effiziente Veränderung auf.

Der SA-Algorithmus nimmt kürzere Rundreisen (energetisch ungünstigere Konfigurationen) sofort und längere Rundreisen mit der Wahrscheinlichkeit $P(\Delta E)$ an.

Das TA-Verfahren hingegen akzeptiert alle neuen Touren, bei denen die Längenänderung ΔE unterhalb eines positiven Schwellenwertes bleibt (welcher aus praktischen Gründen einfach wieder als Temperatur bezeichnet wird).

Die Abfrage, ob die neue Lösung besser oder schlechter ist, als die alte, fällt weg – ebenso die Berechnung von Exponentialfunktion und Zufallszahl. Übrig bleibt nur der Vergleich von Längenunterschied und Temperaturschwelle (vgl. Abb. 4.26 und 4.27).

Der TA-Algorithmus spart dadurch nicht nur einen erheblichen Teil an Rechenzeit ein, er liefert häufig sogar bessere Ergebnisse (kürzere Rundreisen) als das SA-Verfahren, wie Dueck und Scheuer in Testreihen feststellten (OTTO, 1994).

4.2.5 Sintflut-Algorithmus

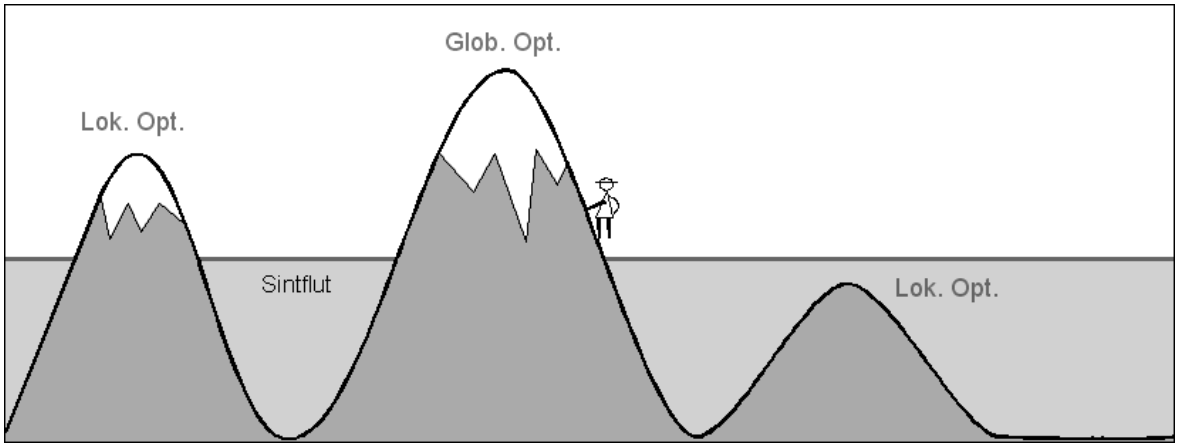


Abb. 4.28: Aufgabe für den Wanderer: keine nassen Füße bekommen

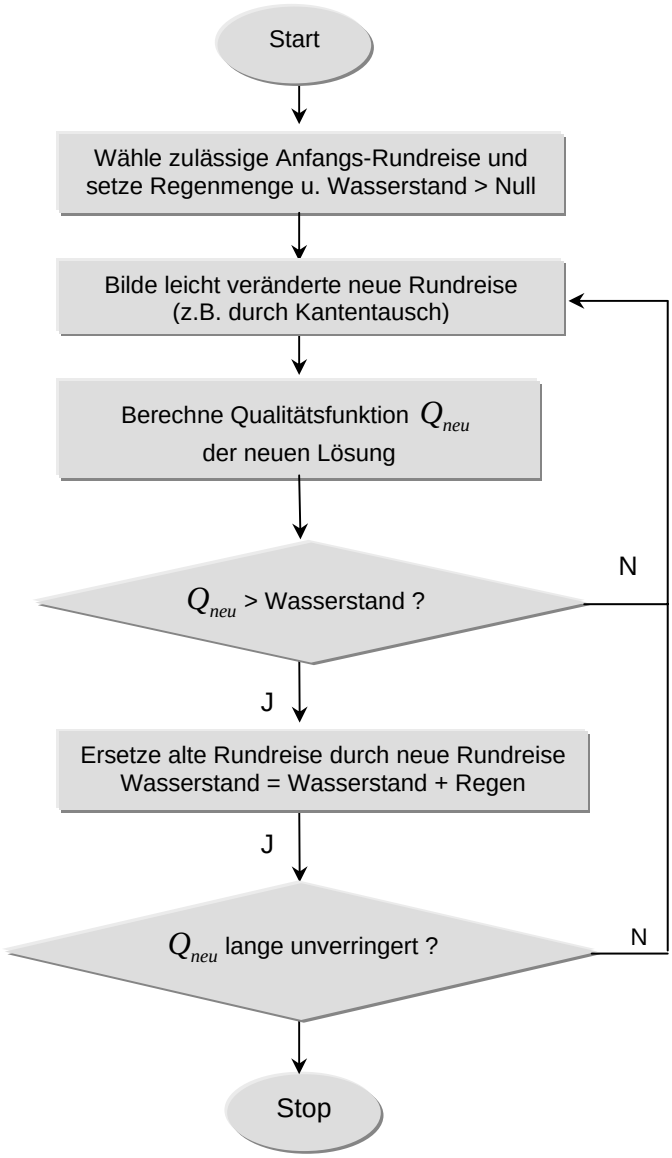


Abb. 4.29: Flußdiagramm Sintflut

Wie schon in Kap. 4.2.4 erwähnt, braucht man einen wohl dosierten Plan zur Temperatur-Erniedrigung, um zu guten Ergebnissen, in unserem Fall also kurzen Rundreisen, zu gelangen.

Um diese Tüftelei zu umgehen stellte Dueck 1993 einen weiteren Abkömmling des SA-Verfahrens vor: den Great Deluge - Algorithm (GDA), auch Sintflut - Algorithmus genannt. Das Steuern der Prozedur wird hier nur noch von einem einzigen Parameter, der Regenmenge, erledigt (vgl. Abb. 4.29).

Das Verfahren ist einfacher, aber nicht unbedingt besser als seine Vorfahren. Dueck ermittelte in zahlreichen Tests, dass GDA im Vergleich zu TA etwa die doppelte Lösungszeit benötigt (OTTO, 1994).

4.2.6 Tabu Search

Abb. 4.30: Rückweg verboten

Tabu Search hat sich innerhalb der letzten zehn Jahre zu einem der leistungsfähigsten und meist benutzten Optimierungsverfahren entwickelt und findet auch in der ArcView-Extension „Network-Analyst, Version 1.0“ seine Anwendung (mehr dazu in Kapitel 7.2.1).

Die Vorgehensweise dieser Methode basiert auf der Idee, die mehrfache Berechnung von identischen Lösungsvarianten zu vermeiden. Mit Hilfe sogenannter Tabu-Listen wird die Rückkehr zu bereits untersuchten Konfigurationen für eine gewisse Anzahl von Zügen (Tabudauer *TD*) verhindert. Als Züge werden dabei die Prozesse bezeichnet, die von einer zulässigen Lösung zur nächsten führen, beispielsweise das Ein- und Auspacken von Gegenständen beim Rucksackproblem (oder das Austauschen von Kanten beim Handelsreisendenproblem). Jeder durchgeführte Zug wird in der Tabu-Liste gespeichert und das Komplement dazu für die Dauer von *TD* Schritten verboten (vgl. Tab. 4.2, nächste Seite). *TD* = 1 würde beispielsweise bedeuten, daß ein ausgepackter Gegenstand im nächsten Zug nicht sofort wieder eingepackt werden darf (bzw. eine Kante nicht umgehend an ihre alte Stelle zurückverschoben werden kann). *TD* = 2 würde dieses Verbot auch für den übernächsten Zug aufrecht erhalten, usw.

Der Vorteil der Tabu-Strategie ist offensichtlich: die Berechnungszeit zur Lösungssuche wird erheblich reduziert. Außerdem erhält man ein geeignetes Werkzeug zum Verlassen lokaler Optima an die Hand (auch schlechtere Folgelösungen sind erlaubt). Der Wert für *TD* ist der ausschlaggebende Faktor in diesem Verfahren und entscheidet darüber, wie schnell und mit welcher Güte man brauchbare Ergebnisse bekommt. Wenn man zu guten Resultaten gelangen will, empfiehlt sich auch hier das Experimentieren mit verschiedenen Lösungsvarianten.

Für die Wahl von *TD* sind folgende Möglichkeiten denkbar (nach DOMSCHKE et al. 1996):

- Statisch : *TD* wird mit einem festen Wert belegt.
- Variabel : Für *TD* werden im Laufe der Prozedur zufällige Werte eingesetzt, die innerhalb eines vorgegebenen Intervalls erzeugt werden.
- Größenorientiert : *TD* wird der Problemgröße *n* angepaßt (beim Rucksackproblem der Anzahl der Gegenstände, beim TSP der Anzahl der Städte).

Der Verfahrensablauf soll durch ein konkretes Beispiel verdeutlicht werden. Die beiden Schmuggler Peter B. und Paul C. aus Kapitel 3.2.3 (S. 13) versuchen mit einer Tabu Search - Strategie den Gewinn ihres zweiten Koffers zu optimieren:

Gegenstand	Perftoran	Kukaburra-Füße	Testosteron	Oxygent	Wombat-Fell	Haifischflossen
Gewicht in kg	5	3	5	6	7	8
Gewinn in $	14	8	8	9	10	11
Rel. Gew. ($/kg)	2,8	2,67	1,6	1,5	1,43	1,38
Rangfolge	1	2	3	4	5	6

Tab 4.1: Die zweite Schmugglerliste (Zahlenwerte aus SCHOLL et al. 1997)

Der Lösungsgang läßt sich wie folgt skizzieren (nach DOMSCHKE et al. 1996):
Das Gewichtslimit für den Koffer beträgt 20 kg. Als zulässige Anfangslösung wählen Peter und Paul die drei Gegenstände mit dem größten Gewinn (5, 1 und 6) und speichern sie als bislang bestes Ergebnis. Ausgehend von dieser Startkonfiguration wird in jeder Iteration ein Zug ausgeführt, welcher zu einer neuen aktuellen Lösung führt. Der gegenteilige Zug wird für *TD* = 3 Schritte (statisch) verboten. Unter allen nicht tabu gesetzten Zügen wählen die beiden jeweils denjenigen, der zur besten Folgelösung (dem größten Gewinnzuwachs bzw. dem kleinsten Gewinnverlust) führt. Ist die Folgelösung besser als das bislang beste Ergebnis, ersetzt sie dieses. Gibt es mehrere beste Züge, entscheidet eine sogenannte 'Tie Break' - Regel: die beiden packen in diesem Fall den leichtesten Gegenstand ein bzw. den schwersten aus. Als Abbruchkriterium werden 11 Iterationsschritte festgelegt.

Iterat.	akt. Lsg.	$	kg	best. Erg.	Gew. ($)	Gew. (kg)	Tabuliste	Zug	Tabu-Zug
1	5 , 1 , 6	35	20	5 , 1 , 6	35	20	[ ]	- 5	+ 5
2	1 , 6	25	13	5 , 1 , 6	35	20	[+ 5]	+ 4	- 4
3	4 , 1 , 6	34	19	5 , 1 , 6	35	20	[+ 5 , - 4]	- 6	+ 6
4	4 , 1	23	11	5 , 1 , 6	35	20	[+ 5 , - 4 , + 6]	+ 2	- 2
5	4 , 1 , 2	31	14	5 , 1 , 6	35	20	[- 4 , + 6 , - 2]	+ 3	- 3
6	4 , 1 , 3 , 2	39	19	4 , 1 , 3 , 2	39	19	[+ 6 , - 2 , - 3]	- 4	+ 4
7	1 , 3 , 2	30	13	4 , 1 , 3 , 2	39	19	[- 2 , - 3 , + 4]	+ 5	- 5
8	**5 , 1 , 3 , 2**	**40**	**20**	**5 , 1 , 3 , 2**	40	20	[- 3 , + 4 , - 5]	- 2	+ 2
9	5 , 1 , 3	32	17	5 , 1 , 3 , 2	40	20	[+ 4 , - 5 , + 2]	- 3	+ 3
10	5 , 1	24	12	5 , 1 , 3 , 2	40	20	[- 5 , + 2 , + 3]	+ 6	- 6
11	5 , 1 , 6	35	20	5 , 1 , 3 , 2	40	20	[+ 2 , + 3 , - 6]	- 5	+ 5

Tab. 4.2: Tabu Search für Peter und Pauls 2. Koffer

Das Zeichen '+' bedeutet Einpacken und das Zeichen '-' Auspacken, der zuletzt verbotene Zug wird am Ende der Tabuliste gespeichert, rückt im Laufe des Verfahrens nach vorn und fällt nach *TD* Schritten heraus.

Das beste Ergebnis findet sich in Iterationsschritt 8: der mit den Gegenständen 1, 2, 3 und 5 gefüllte Koffer wiegt 20 kg und ergibt einen Gewinn von 40 Dollar. Um es vorweg zu nehmen: es ist auch die optimale Lösung, wie in Kapitel 5.1 gezeigt wird. Da Peter und Paul dies nicht wissen können - es handelt sich um eine heuristische Methode - lassen sie das Verfahren bis zum Abbruchkriterium weiterlaufen. In Iteration 11 wird wieder die Anfangslösung 5, 1, 6 erreicht. Setzt man die Prozedur an dieser Stelle fort, wird ein Kreisen (das wiederholende Aufsuchen der immer gleichen Lösungen) trotzdem vermieden. Die Tabuliste ist jetzt nämlich anders besetzt (vgl. Tab. 4.3).

Iterat.	Akt. Lsg.	$	kg	best. Erg.	Gew. ($)	Gew. (kg)	Tabuliste	Zug	Tabu-Zug
11	5 , 1 , 6	35	20	5 , 1 , 3 , 2	40	20	[+ 2 , + 3 , - 6]	- 5	+ 5
12	1 , 6	25	13	5 , 1 , 3 , 2	40	20	[+ 3 , - 6 , + 5]	+ 4	- 4
13	4 , 1 , 6	34	19	5 , 1 , 3 , 2	40	20	[- 6 , + 5 , - 4]	- 1	+ 1
14	4 , 6	20	14	5 , 1 , 3 , 2	40	20	[+ 5 , - 4 , + 1]	...	...

Tab. 4.3: Kreisen wird vermieden

Nach abermaligem Ausführen der Züge - 5 und + 4 ist man in Iteration 13 gezwungen, Gegenstand 1 heraus zu nehmen und gelangt so zu der neuen Konfiguration 4 , 6.

Wie wichtig der Steuerparameter *TD* ist, läßt sich an dem Beispiel ebenfalls gut erkennen:

- Peter und Paul probieren *TD* = 2. Ein Kreisen ist jetzt unvermeidlich, sie landen immer wieder bei ihrer Anfangslösung 5 , 1 , 6 – einem lokalen Optimum.
- *TD* = 4 ergibt in Iteration 6 die Kombination 4, 1, 3, 2 mit der Tabuliste - 4, + 6, - 2, - 3. Da Gegenstand 4 nicht entfernt werden darf, läuft das Verfahren am Optimum vorbei.

Tabelle 4.4 zeigt die Ergebnisse einer Untersuchung von DOMSCHKE, KLEIN und SCHOLL (DOMSCHKE et al. 1996). Es wurden 100 Probleminstanzen (je 20 Rucksäcke mit n = 25, 50, 100, 200, 500 Gegenständen) mit unterschiedlichen Werten für *TD* (*TD* = 1, 5, 10, n/10, variabel) getestet. Abgebildet sind die durchschnittlichen Resultate für 1000 Iterationen.

	TD = 1	TD = 5	TD = 10	TD = n / 10	TD = variabel
Verbesserung gegenüber der Startlösung	19%	40%	64%	105%	109%
Abweichung von der optimalen Lösung	38%	29%	20%	6%	4%
Anzahl der gefunden optimalen Lösungen	1	7	8	17	34

Als Bezugsgrößen für die prozentualen Angaben wurden für jede Instanz der Gesamtgewinn der Anfangslösung sowie der Gesamtgewinn der optimalen Lösung (ermittelt durch ein Branch & Bound - Verfahren) zu Grunde gelegt.

Um einen direkten Vergleich mit den an vorderer Stelle behandelten heuristischen Methoden zu ermöglichen, soll das Prinzip der Tabu-Suche abschließend für das TSP skizziert werden. Abbildung 4.31 zeigt den Verfahrensablauf in einem Flußdiagramm:

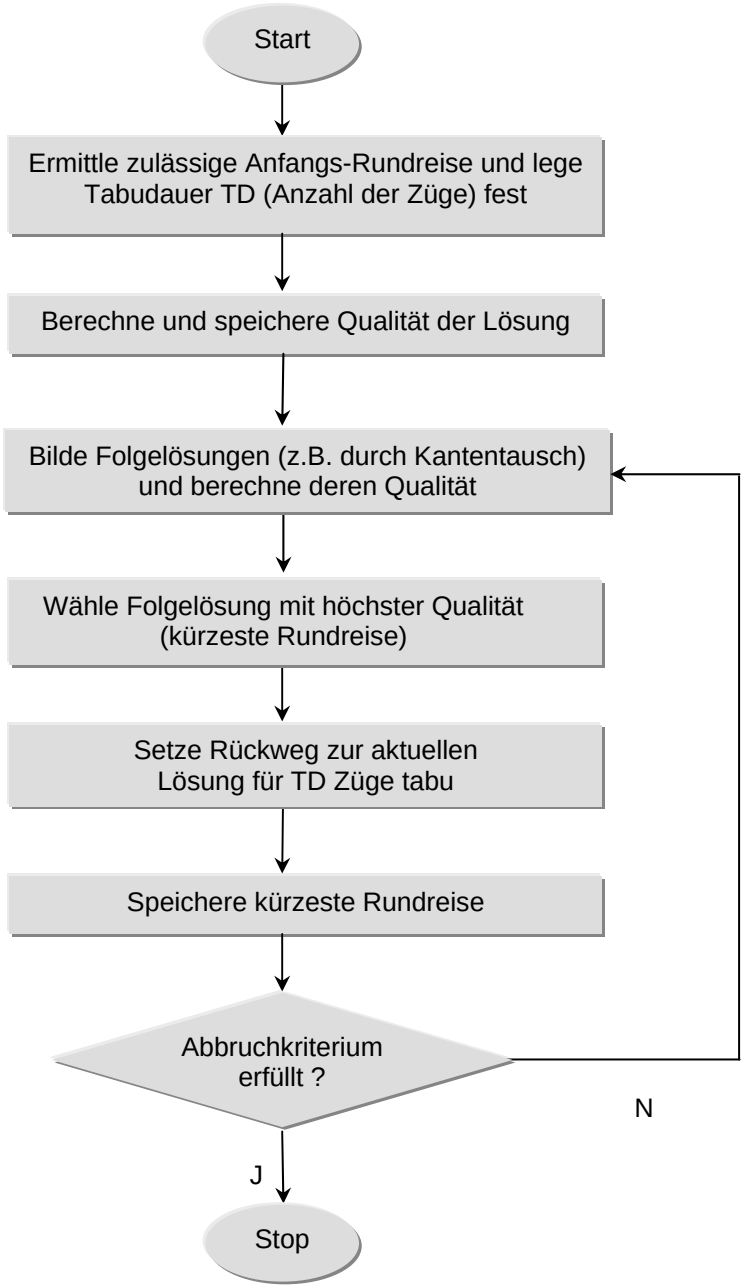


Abb. 4.31: Flußdiagramm Tabu Search

4.2.7 Genetische Algorithmen

Die Idee von Genetischen Algorithmen (GA) basiert auf dem darwinistischen Fortpflanzungsprinzip 'Survival of the Fittest'. Man wählt aus einer Population die beiden besten Individuen aus und erzeugt mittels geeigneter Operatoren Nachkommen in der Hoffnung, daß sich „gute Anlagen" weitervererben. Durch Mutationsverfahren besteht die Möglichkeit, völlig unabhängige und neue Lösungen zu generieren (was sehr nützlich sein kann, wenn der Algorithmus in einem lokalen Optimum festsitzt). Für die Benennung der einzelne Prozesse wird das Vokabular der Biologie benutzt.

In Bezug auf das TSP könnte ein Genetischer Algorithmus z.B. folgendermaßen ablaufen:

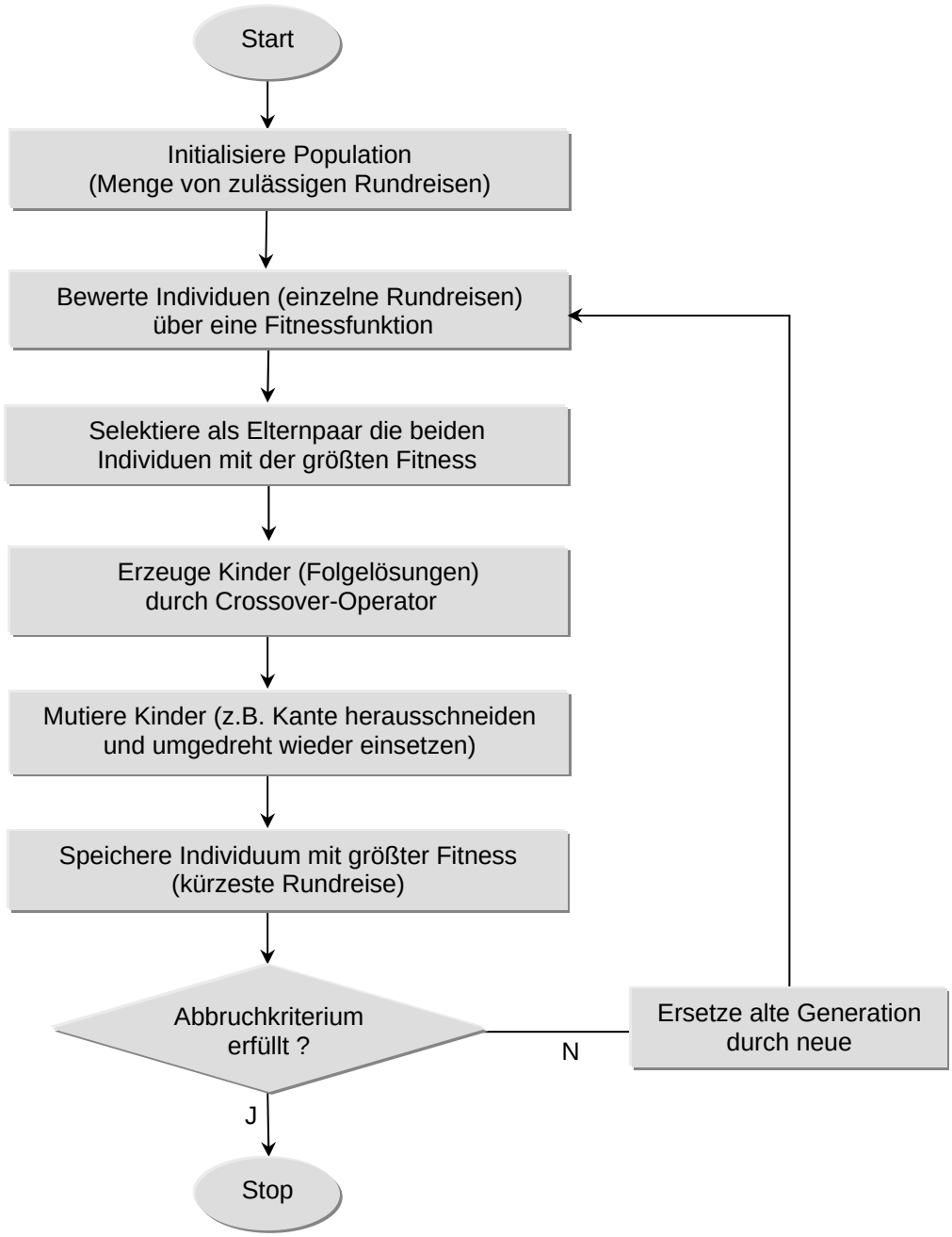


Abb. 4.32: Flußdiagramm eines Genetischen Algorithmus

Der *Crossover*-Operator funktioniert nach folgendem Prinzip:
Gegeben sei ein TSP mit den 10 Städten A bis K. Es existieren die beiden zulässigen Rundreisen

(1) A B C D E F G H I J K und
(2) A C B D F E G J H I K .

Durch Kreuzen der Lösungen (1) und (2) .....

(1) A B C D E F G H I J K
(2) A C B D F E G J H I K

….. können die Nachkommen (3) und (4) erzeugt werden.

(3) A B C D F E G J H I K
(4) A C B D E F G H I J K

Bei Anwendung des Crossover-Operators ist zu beachten, daß nicht jede Rundreise mit jeder anderen auf beliebige Weise gekreuzt werden kann. Auch in den Nachfolge-Touren darf gemäß der Definition in Kapitel 3.2.6 (S. 18) jede Stadt nur einmal besucht werden.
So ist z. B. die folgende Kreuzung der Lösungen (1) und (2) nicht gestattet:

(1) A B C D E F G H I J K
(2) A C B D F E G J H I K

Mutationen können bewerkstelligt werden, indem man Kanten (oder auch Segmente, die aus mehr als zwei Städten bestehen) herausschneidet und um 180° gedreht wieder einsetzt. Im folgenden Fall mutiert Rundreise (5) zu Rundreise (6):

(5) A B C D E F G H I J K
C D G H I J herausschneiden
D C J I H G um 180° drehen
(6) A B D C E F J I H G K und wieder einsetzen

Ähnliche Ziele wie die genetischen Algorithmen verfolgt die Evolutionsstrategie. Auch hier versucht man Optimierungsprozesse, welche in der Natur über mehrere Millionen Jahre hinweg angedauert haben, in Computermodellen zu simulieren. Eine ausführliche Darstellung findet sich z.B. bei RECHENBERG (1994).

4.2.8 Cooperative Simulated Annealing

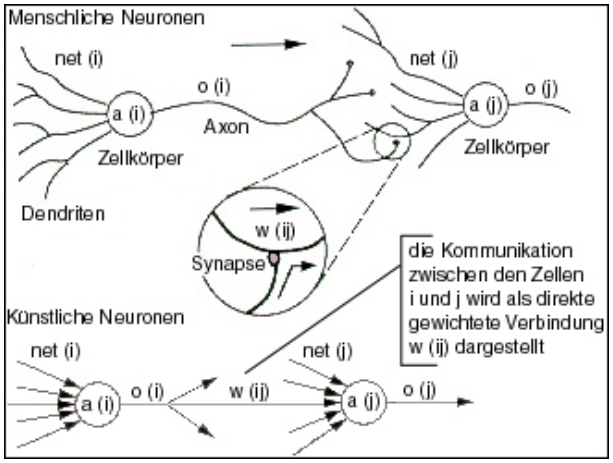
Eine interessante Alternative zum herkömmlichen Verfahren des Simulated Annealing hat OLIVER WENDT im Rahmen seiner Dissertation am Fachbereich Wirtschaftswissenschaften der Johann Wolfgang Goethe-Universität in Frankfurt a. M. entwickelt (WENDT 1995 B).
Das neue Verfahren nennt sich COSA (COoperative Simulated Annealing) und paart die Strategie des langsamen Abkühlens von Kristallen mit den Eigenschaften von Genetischen Algorithmen.
WENDT hat die wesentlichen Ergebnisse seiner Arbeit in einem Institutsbericht zusammengefaßt und definiert dort COSA als „hybrides Verfahren aus SA und GA, welches die Populationsidee und die Idee des Informationsaustausches zwischen Individuen von Genetischen Algorithmen übernimmt, aber auf die dort zum Informationsaustausch angewandten Crossover-Operatoren zugunsten hierfür geschaffener kooperativer Transitionen / Mutationen verzichtet. Die Überlebenswahrscheinlichkeit der Mutanten wird nicht, wie bei GA üblich, durch das 'Survival of the Fittest'-Prinzip bestimmt, sondern vielmehr wie beim SA anhand der Metropolis-Funktion, also in Abhängigkeit der Auswirkung auf den Zielfunktionswert und einer virtuellen Kontrollvariable, genannt Temperatur" (WENDT 1995 A, S. 11).

WENDT hat mit dieser neuen Methode diverse Optimierungsprobleme gelöst und in umfangreichen Tests Vergleiche mit anderen Verfahren durchgeführt. Dabei konnten erstaunliche Ergebnisse erzielt werden (nach WENDT 1995 A, S. 14 f.):

- Beim 51-Städte-Problem von CHRISTOFIDES (1969) wurde das globale Optimum von 428.87 in 25 Versuchen zu jeweils 500.000 Transitionen kein einziges Mal verfehlt.
- Mit COSA gelang es (nach Wissen von WENDT) erstmalig einem heuristischen Verfahren, die optimale Lösung des TSP 318 von LIN (1973) zu reproduzieren, welche mit Schnittebenenverfahren als optimal nachgewiesen wurde.
- Für das TSP 442 von GRÖTSCHEL (1991) wurde der Wert von 50881 gefunden, ein Ergebnis, welches bislang von keinem anderen heuristischen Verfahren erzielt werden konnte.

Zusammenfassend läßt sich feststellen, daß COSA sowohl von der Qualität (Längen der Rundreisen) als auch der Quantität (Anzahl der gefundenen guten Lösungen) deutlich bessere Resultate liefert als SA, TA, GDA und GA.

4.2.9 Neuronale Netze



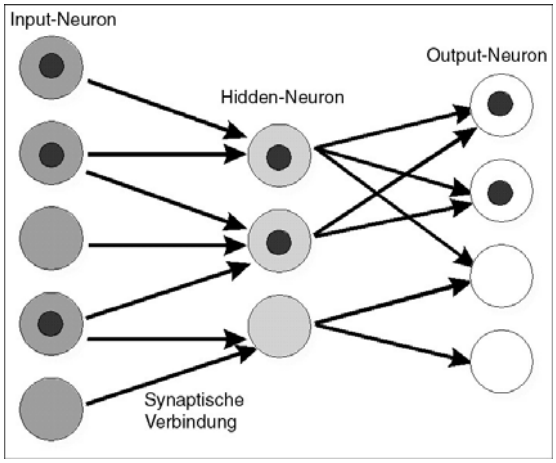


Abb. 4.33: Simulation menschlicher Neuronen **Abb. 4.34:** Feed-Forward-Netz

Mit Hilfe von Neuronalen Netzen versucht man die Funktionsweise des menschlichen Gehirnes in Computern nachzuahmen. Die Nervenzellen (Neuronen) sind durch Dendriten, Axone und synaptische Leitungen in einem dichten Netzwerk miteinander verbunden (vgl. Abb. 4.33). Über diese Verbindungen können Informationen ausgetauscht und weiterverarbeitet werden.

Es gibt mehrere Möglichkeiten zur Modellierung eines solchen Netzes. Eine davon ist das sog. Feed-Forward-Modell, in dem mehrere Schichten (Layer) unterschiedlicher Neuronen angeordnet sind. Man unterscheidet dabei zwischen folgenden Zelltypen (vgl. Abb. 4.34):

1. Input-Neuronen: Informationen aus der Netzumgebung (oder von anderen Neuronen) werden aufgenommen und über einen Reiz an die Hidden-Neurone weitergeleitet.
2. Hidden-Neuronen: die Informationen werden gefiltert, mit anderen Informationen überlagert, gewichtet und schließlich an die Output-Neuronen weitergegeben.
3. Output-Neuronen: die Informationen werden interpretiert und in ein ausgabefähiges Ergebnis umgewandelt (z.B. eine Aussage der Form: ja oder nein, kleiner oder größer).

Der Informationsfluß läuft also nach dem Schema Eingabe - Verarbeitung - Ausgabe ab. Charakteristisch für Feed-Forward-Netze ist, daß die Informationen jeweils nur in einer Richtung weitergegeben werden. Im Gegensatz dazu lassen sich auch sog. Feed-Back-Netze modellieren, in denen alle Neuronen rückgekoppelt sind, also sowohl ’Output’ abgeben als auch ’Input’ aufnehmen können (ZELL 1996 und HEINE, VOGT 1996).

Alexander Budnik und Tanya Filipova haben die Prinzipien eines Neuronalen Netzwerkes auf das TSP angewendet und in einem Java Applet programmtechnisch umgesetzt. Die sogenannte „Elastic Net-Method“ kann im Internet auf der Homepage der Autoren getestet werden (BUDNIK, FILIPOVA 1995).

Der Wirkungsweise des elastischen Netzes läßt sich in etwa folgendermaßen skizzieren:
Am Anfang werden zehn Städte (symbolisiert durch kleine Rechtecke) zufällig auf einem quadratischen Feld verteilt. In der Mitte dieses Feldes liegt ein elastisches kreisförmiges Band, welches die Rundreise durch die Städte repräsentiert. Es enthält 30 in gleichen Abständen angeordnete Neuronen (veranschaulicht durch kleine Punkte), die durch die Kreisstruktur miteinander verbunden sind (Abb. 4.35a, 4.35b, S. 57). Das „Gummiband" wird mit dem Start der Berechnung systematisch verformt und in jeder Iteration wird versucht, seine Geometrie etwas besser der 10-Städte-Konfiguration anzupassen (Abb. 4.36a, 4.36b). Ziel des Algorithmus ist es, das Band so durch die zehn Städte zu legen, daß seine Länge (und damit die der Rundreise) minimal wird. Das Verfahren terminiert, wenn keine nennenswerte Verbesserung mehr erzielt werden kann, die Tourlänge sich von einer Iteration zur nächsten also nicht mehr signifikant ändert (Abb. 4.37a, 4.37b).
In jedem Berechnungsschritt kommen zwei grundlegende Regeln zum Tragen:

1. Jedes Neuron bewegt sich auf die nächstliegende Stadt zu (oder anders ausgedrückt: die Stadt mit dem kürzesten Abstand zum Neuron bekommt den Reiz zugesprochen).
2. Die Neuronen werden dabei so auf dem Band verschoben, daß die Gesamt-Pfadlänge minimiert wird.

Mit diesem Vorgehen wird jeder Stadt ein bestimmter Bereich des Bandes zugeordnet. Dabei tritt eine gewisse Art der Überbestimmung auf: auf 10 Städte kommen 30 Neuronen. Wie viele Neuronen wandern also auf die gleiche Stadt zu bzw. wie viele Reize bekommt eine Stadt zugesprochen? Der Verhältnis 'Stadt zu Neuronen' ist entfernungsabhängig und ändert sich stetig im Laufe des Algorithmus (vgl. S. 57). Zu Beginn hat jede Stadt etwa den gleichen Einfluß auf den Gesamtkreis. Je besser sich das Band an eine Stadt annähert, desto größer wird deren Einfluß, so daß im Laufe des Prozesses jede Stadt ihre eigene Gewichtung bekommt (BUDNIK, FILIPOVA, 1995).
Für diese Applikation wird eine spezielle Ausprägung von Neuronalen Netzwerken, ein sogenanntes Kohonen-Netz, verwendet. Der Vorteil eines solches Netzes ist, daß man es nicht zu „trainieren" braucht, es organisiert sich von selbst. Bei allgemeinen neuronalen Netzen hingegen muß ein „Training" durchgeführt werden. Im Laufe des Optimierungsprozesses werden die Steuerparameter so verändert, daß das Netz lernt, sich auf die neuen Gegebenheiten einzustellen. Verläuft der Lernprozeß erfolgreich, kann der Algorithmus mit jedem Iterationsschritt die gestellte Aufgabe besser abschätzen – die Ergebnisse werden immer genauer und zuverlässiger. Eine Darstellung zu Kohonen-Netzen findet sich z.B. bei BRAUSE (1991) und ZELL (1996).

Beispiel A

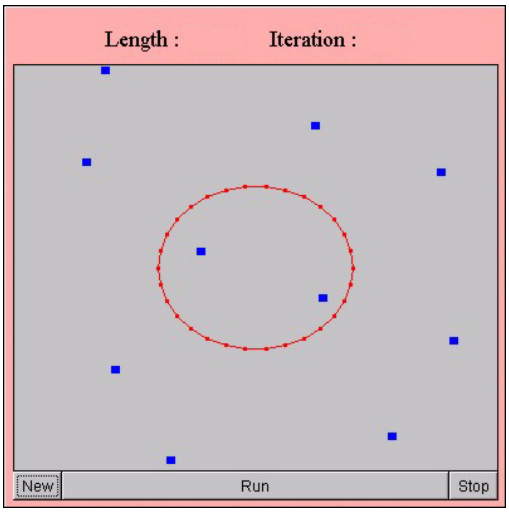


Abb. 4.35a: Startkonfiguration

Beispiel B

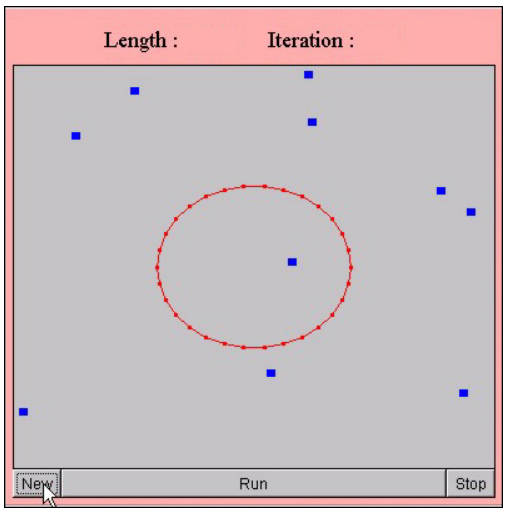


Abb. 4.35b: Startkonfiguration

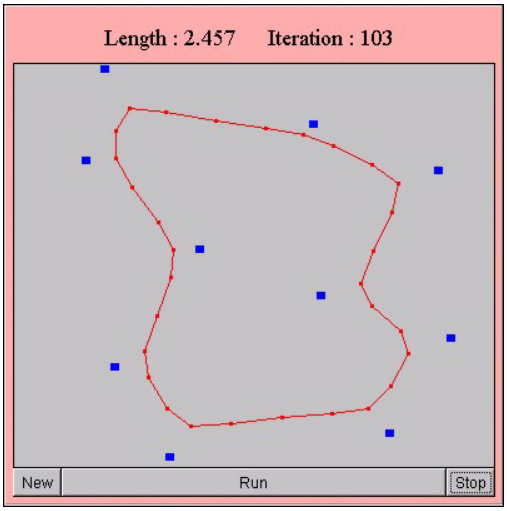


Abb. 4.36a: Zwischenergebnis

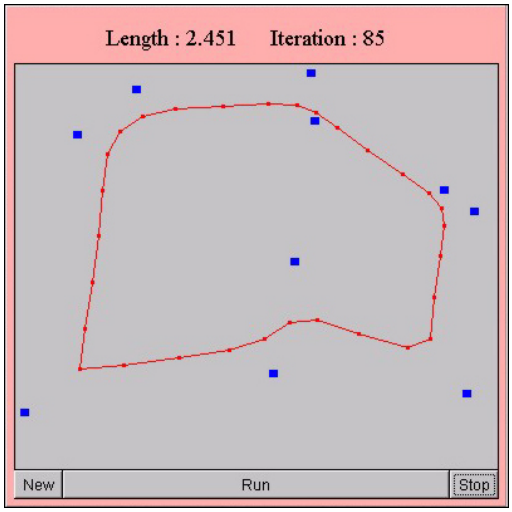


Abb. 4.36b: Zwischenergebnis

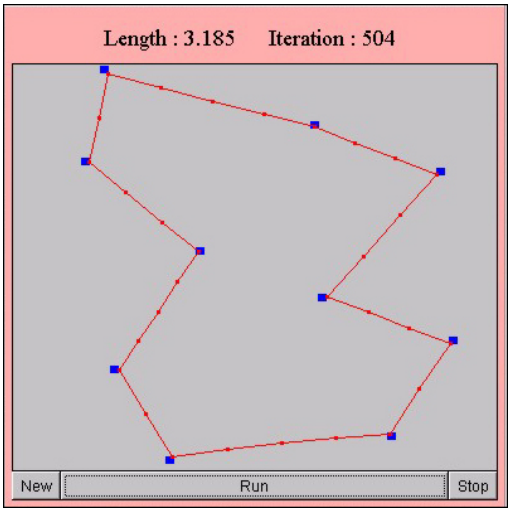


Abb. 4.37a: Kürzeste Rundreise

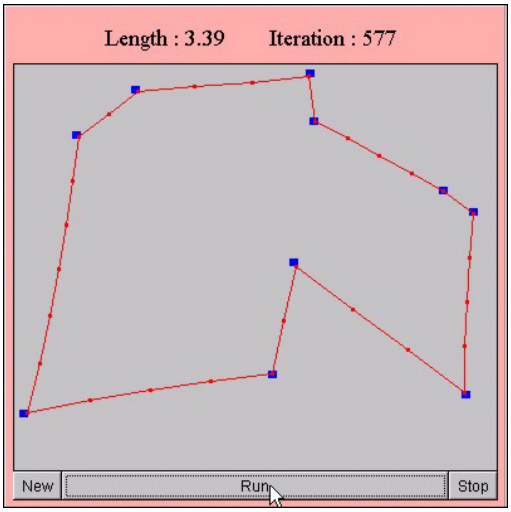


Abb. 4.37b: Kürzeste Rundreise

5. Lösungen für das Rucksackproblem

Wie in Kapitel 3.2.3 (S. 14) angedeutet, soll den Schmugglern Peter B. und Paul C. in diesem Kapitel geholfen werden. Dazu wird noch einmal die Tabelle mit den Waren betrachtet, welche die beiden in Peters Koffer nach Deutschland schicken wollen:

5.1 Der erste Koffer

Gegenstand	Black-Swan-Federn	Krokodilzähne	Nandrolon	Anabolika	Emu-Eier	Amphetamine
Gewicht in kg	3	4	6	6	5	7
Gewinn in $	8	10	14	12	9	10
Rel. Gew. ($/kg)	2,67	2,5	2,33	2	1,8	1,43
Rangfolge	1	2	3	4	5	6

Tab 5.1: Die erste Schmugglerliste

Die Frage, welche Gegenstände die beiden einpacken sollen, um einen optimalen Gewinn zu erzielen, war offen geblieben. Die Antwort soll nun mit Hilfe des Programmes TENOR BABE 1.0 gegeben werden. BABE (Branch And Bound Evaluator) ist ein Modul der Optimierungssoftware TENOR, welche in Kapitel 6 etwas ausführlicher beschrieben wird. Mit Hilfe des folgenden Auszugs aus dem Berechnungsprotokoll kann die Lösungsfindung nachvollzogen werden:

Problemgröße : 6 Struktur-Variablen, 1 Nebenbedingung
Originalproblem
- Maximiere : 8,0 X1 + 10,0 X2 + 14,0 X3 + 12,0 X4 + 9,0 X5 + 10,0 X6
- Wertebereiche : X1 , X2 , X3 , X4 , X5 , X6 $\in$ {0,1}
- Nebenbedingung : 3,0 X1 + 4,0 X1 + 6,0 X1 + 6,0 X1 + 5,0 X1 + 7,0 X1 $\leq$ 20,0

Standardeinstellungen
- Auswahlregel : LIFO (Tiefensuche)
- Verzweigungsregel : kleinster Abstand zur nächsten ganzen Zahl
- Bearbeitungsreihenfolge : aufsteigend

Lösungsgang :
1. P0 ist das Ausgangsproblem. Relaxation: X = (1, 1, 1, 1, 0.2, 0); F = 45.8
   $\rightarrow$ Knoten nicht auslotbar: Problem wird in die Kandidatenliste gesetzt
   Auswahlvorschlag aus Kandidatenliste: P0 Ausgewählt: P0
2. P1 entsteht aus P0 und X5 = 0 Relaxation: X = (1, 1, 1, 1, 0, 0.143); F = 45.43
   $\rightarrow$ Knoten nicht auslotbar: Problem wird in die Kandidatenliste gesetzt
   Auswahlvorschlag aus Kandidatenliste: P1 Ausgewählt: P1
3. P2 entsteht aus P1 und X6 = 0 Relaxation: X = (1, 1, 1, 1, 0, 0); F = 44
   $\rightarrow$ Knoten ausgelotet, Fall b
   Auswahlvorschlag aus Kandidatenliste: P1 Ausgewählt: P1
4. P3 entsteht aus P1 und X6 = 1 Relaxation: X = (1, 1, 1, 0, 0, 1); F = 42
   $\rightarrow$ Knoten ausgelotet, Fall a
   Auswahlvorschlag aus Kandidatenliste: P0 Ausgewählt: P0
5. P4 entsteht aus P0 und X5 = 1 Relaxation: X = (1, 1, 1, 0.333, 1, 0); F = 45
   $\rightarrow$ Knoten nicht auslotbar: Problem wird in die Kandidatenliste gesetzt

Auswahlvorschlag aus Kandidatenliste: P4 Ausgewählt: P4
6. P5 entsteht aus P4 und X4 = 0 Relaxation: X = (1, 1, 1, 0, 1, 0.286); F = 43.86
→ Knoten ausgelotet, Fall a
Auswahlvorschlag aus Kandidatenliste: P4 Ausgewählt: P4
7. P6 entsteht aus P4 und X4 = 1 Relaxation: X = (1, 1, 0.333, 1, 1, 0); F = 43.67
→ Knoten ausgelotet, Fall a
Ausgangsproblem gelöst !!

Lösung : eindeutige Lösung
Zielfunktionswert : 44,0
Variablen : X1 = 1,0 X2 = 1,0 X3 = 1,0 X4 = 1,0 X5 = 0,0 X6 = 0,0

In diesem Fall bildet die Kombination aus den Artikeln mit den höchsten relativen Gewinnen (1, 2, 3 und 4) die optimale Lösung. Es entsteht ein von Gewinn von 44 Dollar. Der dazugehörige Lösungsbaum:

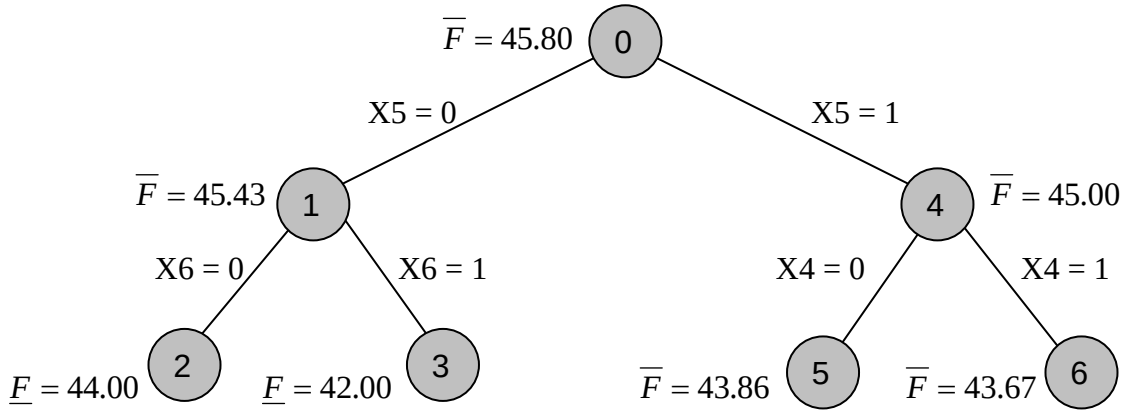


Abb. 5.1: Binärer Lösungsbaum für das 1. Schmugglerproblem

Eine Breitensuche bewirkt für dieses Beispiel keine Änderung, der aufspannende Lösungsbaum ist mit dem obigen identisch. Peter und Paul sind mit dem Ergebnis zufrieden und wollen Branch & Bound jetzt auch für ihren zweiten Koffer ausprobieren.

5.2 Der zweite Koffer

Gegenstand	Perftoran	Kukaburra-Füße	Testosteron	Oxygent	Wombat-Fell	Haifischflossen
Gewicht in kg	5	3	5	6	7	8
Gewinn in $	14	8	8	9	10	11
Rel. Gew. ($/kg)	2,8	2,67	1,6	1,5	1,43	1,38
Rangfolge	1	2	3	4	5	6

Tab 5.2: Die zweite Schmugglerliste (Zahlenwerte aus SCHOLL et al. 1997)

Die optimale Kombination der Gegenstände für einen maximalen Gewinn wird wieder mit Hilfe von TENOR BABE ermittelt.

Auszug aus den Berechnungsprotokoll:

Problemgröße : 6 Struktur-Variablen, 1 Nebenbedingung
Originalproblem
- Maximiere : 14,0 X1 + 8,0 X2 + 8,0 X3 + 9,0 X4 + 10,0 X5 + 11,0 X6
- Wertebereiche : X1 , X2 , X3 , X4 , X5 , X6 $\in$ {0,1}
- Nebenbedingung : 5,0 X1 + 3,0 X1 + 5,0 X1 + 6,0 X1 + 7,0 X1 + 8,0 X1 $\leq$ 20,0

Standardeinstellungen
- Auswahlregel : LIFO (Tiefensuche)
- Verzweigungsregel : kleinster Abstand zur nächsten ganzen Zahl
- Bearbeitungsreihenfolge : aufsteigend

Lösungsgang :
1. P0 ist das Ausgangsproblem. Relaxation: X = (1, 1, 1, 1, 0.143, 0); F = 40.43
   $\rightarrow$ Knoten nicht auslotbar: Problem wird in die Kandidatenliste gesetzt

   Auswahlvorschlag aus Kandidatenliste: P0 Ausgewählt: P0
2. P1 entsteht aus P0 und X5 = 0 Relaxation: X = (1, 1, 1, 1, 0, 0.125); F = 40.38
   $\rightarrow$ Knoten nicht auslotbar: Problem wird in die Kandidatenliste gesetzt
   Auswahlvorschlag aus Kandidatenliste: P1 Ausgewählt: P1
3. P2 entsteht aus P1 und X6 = 0 Relaxation: X = (1, 1, 1, 1, 0, 0); F = 39
   $\rightarrow$ Knoten ausgelotet, Fall b
   Auswahlvorschlag aus Kandidatenliste: P1 Ausgewählt: P1
4. P3 entsteht aus P1 und X6 = 1 Relaxation: X = (1, 1, 0.8, 0, 0, 1); F = 39.4
   $\rightarrow$ Knoten nicht auslotbar: Problem wird in die Kandidatenliste gesetzt
   Auswahlvorschlag aus Kandidatenliste: P3 Ausgewählt: P3
5. P4 entsteht aus P3 und X3 = 0 Relaxation: X = (1, 1, 0, 0.667, 0, 1); F = 39
   $\rightarrow$ Knoten ausgelotet, Fall a
   Auswahlvorschlag aus Kandidatenliste: P3 Ausgewählt: P3
6. P5 entsteht aus P3 und X3 = 1 Relaxation: X = (1, 0.667, 1, 0, 0, 1); F = 38.33
   $\rightarrow$ Knoten ausgelotet, Fall a
   Auswahlvorschlag aus Kandidatenliste: P0 Ausgewählt: P0
7. P6 entsteht aus P0 und X5 = 1 Relaxation: X = (1, 1, 1, 0, 1, 0); F = 40
   $\rightarrow$ Knoten ausgelotet, Fall b

Ausgangsproblem gelöst !!

Lösung : eindeutige Lösung
Zielfunktionswert : 40,0
Variablen : X1 = 1,0 X2 = 1,0 X3 = 1,0 X4 = 0,0 X5 = 1,0 X6 = 0,0

Hier ergibt die Kombination der Waren 1, 2 , 3 und 5 den maximalen Gewinn von 40 Dollar. Peter und Paul wissen jetzt also mit Sicherheit, daß schon das Ergebnis, welches sie in Kapitel 4.2.6 mit Hilfe der Tabu-Suche gefunden haben, optimal war.

Der dazugehörige Lösungsbaum:

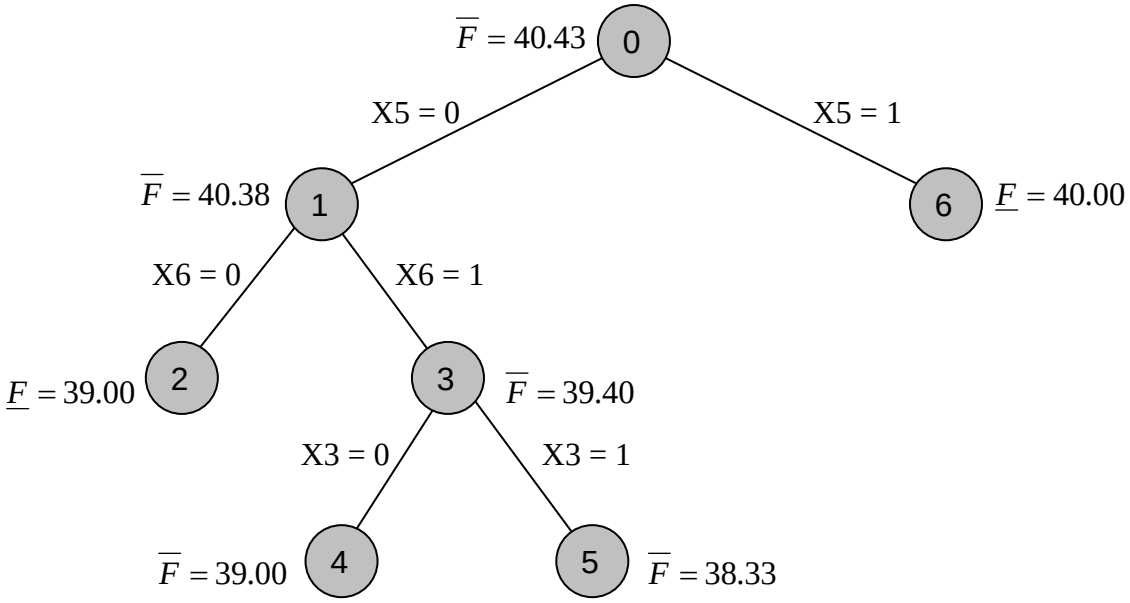


Abb. 5.2: Binärer Lösungsbaum für das 2. Schmugglerproblem (Tiefensuche)

Die optimale Lösung steckt bereits im rechten Folgeknoten der Quelle. Mit Hilfe einer Breitensuche wären Peter und Paul hier also schneller ans Ziel gekommen. Um dies nachzuprüfen, ersetzen sie in der Auswahlregel LIFO durch MUB / MLB und berechnen ihren zweiten Koffer noch einmal:

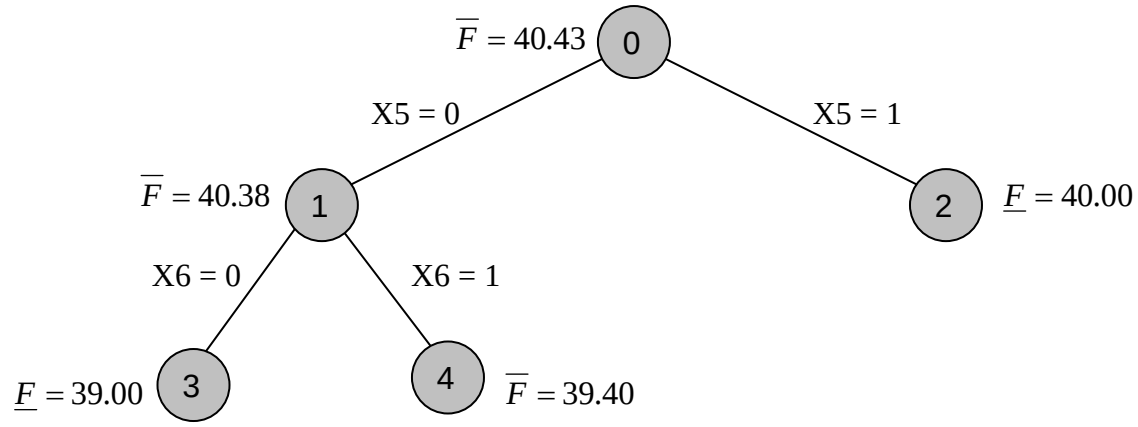


Abb. 5.3: Binärer Lösungsbaum für das 2. Schmugglerproblem (Breitensuche)

Wie erwartet repräsentiert Knoten Nr. 2 den maximalen Gewinn von 40 Dollar. Nachdem Knoten 3 und 4 ausgelotet sind, ist eine weitere Suche im linken Teilbaum nicht mehr nötig.

6. Lösungen für das Traveling Salesman Problem

Für ein TSP mit 10 und ein weiteres mit 25 Städten soll im nachfolgenden Abschnitt die Lösungsfindung beschrieben werden. Die Resultate wurden mit dem Programm TENOR TSP, Version 1.0, berechnet. Das Softwarepaket TENOR (Tutorial ENvironment for Operations Research) wurde 1996 an der Technischen Universität Darmstadt am Fachgebiet Operations Research von der Gruppe um Prof. Dr. WOLFGANG DOMSCHKE entwickelt. Es ist frei über das Internet erhältlich (s. Literaturverzeichnis), enthält sechs verschiedene Module zur Lösung von Optimierungsproblemen und ist insbesondere als Lernprogramm für Studenten gedacht.

Zur Lösung der Beispielaufgaben habe ich mir zwei symmetrische TSPe auf ungerichteten Graphen ausgedacht, da sich die benutzten Heuristiken für derartige Problemstellungen besonders einfach gestalten. Generell kann die Vorgehensweise auch auf gerichtete Graphen (asymmetrische TSPe) übertragen werden.

Bei der mathematischen Formulierung des Problems werden folgende Vereinbarungen getroffen: die *kürzeste* Verbindung zwischen zwei Knoten (Städten) in einem vollständigen ungerichteten Graphen $G = [V, E, c]$ wird durch eine Kante (geradlinige Straße) repräsentiert (vgl. Abb. 6.1).

Damit dies gewährleistet ist, muß die Dreiecksungleichung gelten: $C_{ik} \leq C_{ij} + C_{jk}$.

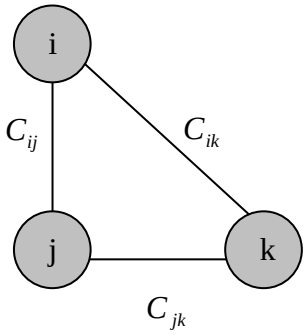


Abb. 6.1: Straßen in einem Graphen

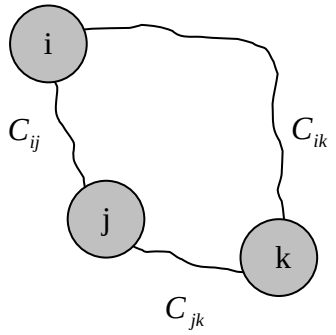


Abb. 6.2: Straßen in der Realität

Mit den Kosten C_{ij} ist in diesem Fall die Länge der Verbindung zwischen Stadt *i* und Stadt *j* gemeint. In einem wirklichen Straßennetz kann es durchaus vorkommen, daß die direkte Verbindung zwischen zwei Städten länger ist, als der „Umweg“ über eine dritte Station (vgl. Abb. 6.2): in solchen Fällen gilt $C_{ik} > C_{ij} + C_{jk}$.

In den Kostenmatrizen für das TSP 10 und 25 mußte diesem Umstand Rechnung getragen werden. Erstaunlicherweise war jedoch nur beim TSP 25 eine Korrektur weniger Werte um höchstens zwei Kilometer notwendig, um die Dreiecksungleichung überall zu erfüllen. Die beiden Beispiele können also durchaus als realistisch bezeichnet werden.

Will man die nachfolgenden Verfahren auf nicht - vollständigen Graphen (mit unsymmetrischen Kostenmatrizen) einsetzen, kann man sich mit einem einfachen Trick behelfen: man ersetzt die Kosten $c_{ij} = \infty$ (für $i \neq j$) der nicht vorhandenen Kanten einfach durch eine sehr große Zahl $c_{ij} = M$. Ein Algorithmus kann mit einer solchen Matrix dann rechnen, und die künstlich hinzugefügten Kanten spielen bei der Lösungsfindung wegen ihrer enormen Länge keine Rolle.

6.1 Eröffnungsverfahren

Mit Hilfe eines Eröffnungsverfahrens läßt sich für ein Optimierungsproblem eine zulässige Anfangslösung generieren, welche im allgemeinen noch keine optimale Lösung darstellt. Zwei wichtige Eröffnungsverfahren sind die Methode des besten Nachfolgers sowie die Methode der sukzessiven Einbeziehung, im folgenden kurz BN und SE genannt.

6.1.1 Methode des besten Nachfolgers (BN)

Es wird ein beliebiger Anfangsknoten ausgewählt und zu allen anderen Knoten die Entfernung ermittelt. Derjenige mit der geringsten Entfernung bildet den 2. Knoten. Anschließend wird aus den noch verbliebenen Knoten wieder derjenige mit dem kürzesten Abstand als Folgeknoten hinzugefügt. Nach n Iterationen hat man eine Rundreise gefunden, welche eine zulässige Anfangslösung bildet, vom globalen Optimum aber in den meisten Fällen stark abweicht. Dieses Verfahren gehört zu den in Kap. 4.2.1 beschriebenen Greedy-Algorithmen (DOMSCHKE 1982, S. 96 f. und DOMSCHKE, DREXL 1991, S. 120).

6.1.2 Methode der sukzessiven Einbeziehung (SE)

Zu Beginn wählt man zwei beliebige Knoten aus. In jeder Iteration wird ein Knoten hinzugefügt, welcher noch nicht in der Tour enthalten ist. Der neue Knoten soll dabei so eingepaßt werden, daß die Länge der Rundreise sich möglichst wenig vergrößert.

Eine Variante besteht darin, mit zwei möglichst weit voneinander entfernten Knoten anzufangen. Man bestimmt von allen Knoten, die noch nicht zur Rundreise gehören, jeweils den kleinsten Abstand zu einem Tour-Knoten und fügt denjenigen mit dem größten Wert ein. Durch diese Auswahl wird die Struktur der Rundreise schon zu einem frühen Zeitpunkt der Entwicklung beeinflußt und die so erhaltenen Lösungen sind meist besser als die Ergebnisse zufälliger Selektionsmethoden. Im Vergleich zum Verfahren des besten Nachfolgers liefert die sukzessive Einbeziehung der Knoten deutlich bessere Anfangslösungen (DOMSCHKE 1982, S. 97 und DOMSCHKE, DREXL 1991, S. 120 f.).

6.2 Verbesserungsverfahren

Ist für ein Problem eine gültige Anfangslösung gegeben oder hat man eine solche ermittelt, kann sie mit einem Verbesserungsverfahren Schritt für Schritt optimiert werden. Die Verfahren lassen sich in deterministische und stochastische Methoden unterteilen. Deterministische Ansätze liefern bei Wiederholung einer Aufgabe mit gleichen Anfangsbedingungen jeweils ein identisches Ergebnis. Bei stochastischen Ansätzen hingegen ist eine Zufallskomponente enthalten, welche unterschiedliche Resultate erzeugt (vgl. z.B. Simulated Annealing, Kap. 4.2.3).

Unter den deterministischen Methoden sind vor allem die *r-optimalen* Verfahren zu nennen. Hier wird ausgehend von der Startlösung in jeder Iteration überprüft, ob sich die Länge der aktuellen Tour durch einen Austausch von r Kanten mit r anderen Kanten verringern läßt. Ist dies der Fall, wird der Tausch vorgenommen, andernfalls wird die nächste Option überprüft. Das Verfahren terminiert, wenn alle Möglichkeiten untersucht worden sind und durch einen r-Kantentausch keine Verbesserung mehr erzielt werden kann. Man spricht dann von einer r-optimalen Rundreise.

Der Berechnungsaufwand wächst dabei exponentiell mit der Kantenzahl r, so daß vor allem 2- und 3-optimale Methoden (im folgenden 2-Opt und 3-Opt genannt) zum Einsatz kommen (DOMSCHKE 1982, S. 98 ff. und DOMSCHKE, DREXL 1991, S. 121 ff.).

6.2.1 2-Opt-Verfahren

Der Austausch von jeweils 2 Kanten wird untersucht. Die Rundreise heißt 2-optimal, wenn alle denkbaren Varianten überprüft wurden und eine Verkürzung der Tourlänge durch einen 2-Tausch nicht mehr möglich ist. Pro Iteration entsteht ein zeitlicher Rechenbedarf der Ordnung n^2 (n = Anzahl der Städte).

6.2.2 3-Opt-Verfahren

Der Austausch von jeweils 3 Kanten wird untersucht. Die Rundreise heißt 3-optimal, wenn alle denkbaren Konfigurationen getestet wurden und eine verbesserte Lösung durch einen 3-Tausch nicht mehr erreichbar ist. Pro Iteration wird ein zeitlicher Rechenaufwand der Ordnung n^3 benötigt.

6.3 TSP 10 – kleine Rundreise durch Deutschland

		Bre	Düs	Ess	Ful	Hal	Kob	Mar	Mün	Pot	Sal
1	Bremerhaven	x	319	286	387	371	423	347	210	381	224
2	Düsseldorf		x	33	263	410	125	181	115	505	301
3	Essen			x	261	380	152	179	83	475	271
4	Fulda				x	219	176	84	254	351	214
5	Halle					x	376	272	350	137	156
6	Koblenz						x	115	218	508	331
7	Marburg							x	179	396	218
8	Münster								x	421	214
9	Potsdam									x	207
10	Salzgitter										x

Tab. 6.1: Kostenmatrix für das TSP 10

Mit "Kosten" sind hier die kürzesten Verbindungen zwischen jeweils zwei Städten auf dem Straßenverkehrsnetz (in Kilometern) gemeint. Die Entfernungen wurden mit Hilfe des Internet-Routenplaners unter http://www.reiseplanung.de berechnet.

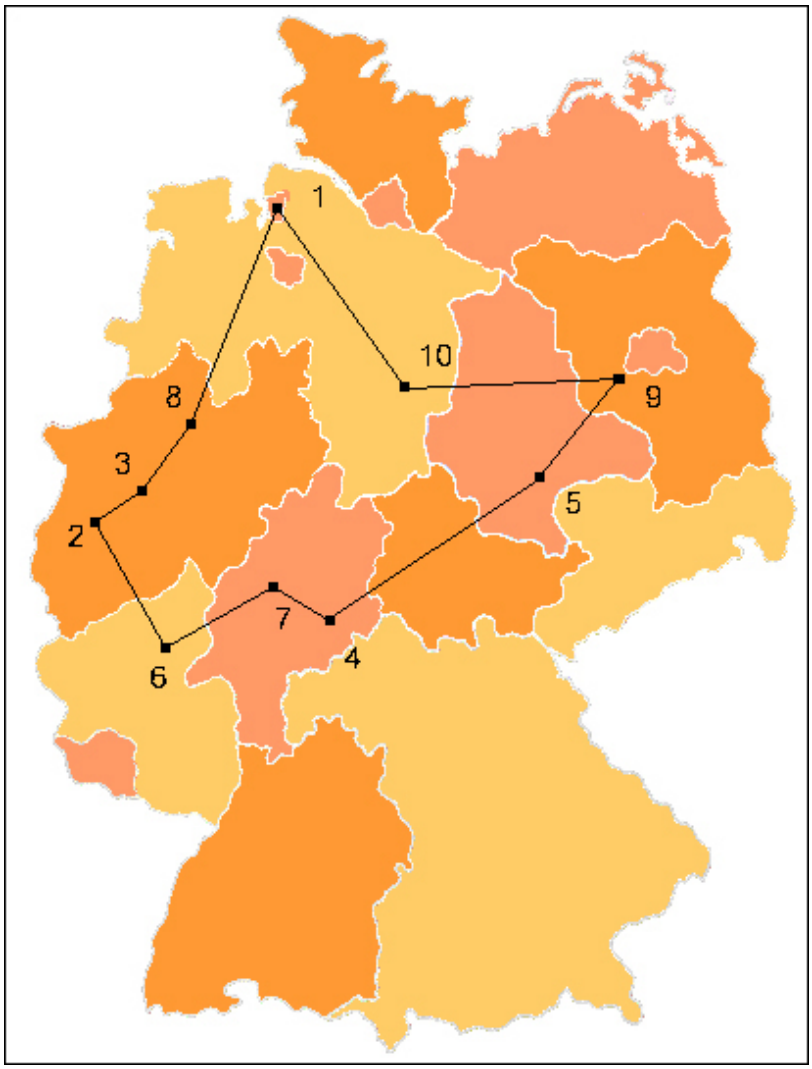


Abb. 6.3:
Eine 3-optimale Rundreise durch 10 deutsche Städte, Gesamtlänge: 1437 km. Insgesamt stehen (10-1)! / 2 = 181.440 mögliche Touren zur Auswahl.

6.3.1 Lösung BN

Auszug aus dem Berechnungsprotokoll

Initialisierung: Gewählter Startknoten: 9 , r = [9] , c(r) = 0

1. Iteration: Knoten mit geringster Entfernung zu 9 hinzugefügt: 5 , r = [9 5], c(r) = 137

2. Iteration: Knoten mit geringster Entf. zu 5 hinzugefügt: 10 , r = [9 5 10], c(r) = 293

...

9. Iterat.: Kn. mit geringst. Entf. zu 8 hinzugef.: 1 , r = [9 5 10 4 7 6 2 3 8 1], c(r) = 1157

10. Iteration: Abbruch: Alle Knoten aufgenommen. Rundreise wird geschlossen.

Ermittelte Rundreise : r = [9 1 8 3 2 6 7 4 10 5 9], c(r) = **1538**

6.3.2 Lösung SE

Auszug aus dem Berechnungsprotokoll

Initialisierung: Gewählte Subtour: r = [9 1 9] , c(r) = 762

1. Iteration, 1. Schritt: Knoten mit maximaler kürzester Entfernung zu r : 6

1. Iteration, 2. Schritt: Knoten 6 in die Rundreise eingefügt r = [9 1 6 9], c(r) = 1312

2. Iteration, 1. Schritt: Knoten mit maximaler kürzester Entfernung zu r : 8

2. Iteration, 2. Schritt: Untersuchung der Einfügeposition von Knoten 8 zwischen 9 und 1
r = [9 6 1 8 9], c(r) = 1562

...

8. Iteration, 11. Schritt: Knoten an Position, an der er geringste Kosten verursacht, eingefügt. r = [9 1 8 3 2 6 7 4 10 5 9], c(r) = 1538

9. Iteration, 1. Schritt: Abbruch: Alle Knoten in die Rundreise aufgenommen.

Ermittelte Rundreise : r = [9 1 8 3 2 6 7 4 10 5 9], c(r) = **1538**

6.3.3 Lösung 2-Opt

Auszug aus dem Berechnungsprotokoll

Initialisierung: Start mit Rundreise: r = [9 1 8 3 2 6 7 4 10 5 9], c(r) = 1538

1. Iteration, Untersuchung aller 2-Tausche:

Tausch von [9,1] ,[8,3] gegen [9,8] ,[1,3] , Kostendifferenz = 243

...

Tausch von [8,1] ,[10,9] gegen [8,10] ,[1,9] , Kostendifferenz = 178

Abbruch: Keine Verbesserung durch 2-Tausch mehr möglich.

Rundreise ist 2-optimal: r = [9 5 4 7 6 2 3 8 1 10 9], c(r) = **1437**

6.3.4 Lösung 3-Opt

Auszug aus dem Berechnungsprotokoll

Initialisierung: Start mit Rundreise: r = [9 5 4 7 6 2 3 8 1 10 9], c(r) = 1437

1. Iteration, Untersuchung aller 3-Tausche:

Tausch von [9,5] ,[5,4] ,[4,7] gegen [5,7] ,[5,4] ,[4,9] , Kostendifferenz = 402

...

Tausch von [10,9] ,[3,8] ,[1,10] gegen [9,10] ,[3,1] ,[8,10] , Kostendifferenz = 193

Abbruch: Keine Verbesserung durch 3-Tausch mehr möglich.

Rundreise ist 3-optimal: r = [9 5 4 7 6 2 3 8 1 10 9], c(r) = **1437**

6.3.5 Strategisches Vorgehen

Bei dem obigen Beispiel wurde der Startpunkt im BN-Verfahren willkürlich gewählt. Die ersten beiden Knoten der gefundenen Lösung wurden dann als Anfangs-Subroute für das SE-Verfahren verwendet. Da es so reiner Zufall ist, ob man eine *gute* Anfangslösung für ein Verbesserungsverfahren erhält, soll die Sache einmal systematisch betrachtet werden.

Als erstes wird jede Stadt als Startpunkt ausprobiert. Diejenige, welche die kürzeste Route liefert, wird als Anfangsknoten für das SE-Verfahren verwendet. Hier bestimmt man unter Hinzunahme aller möglichen Folgeknoten wieder ein Optimum. Die gefundene Rundreise stellt dann für das Lösungsverfahren 2-Opt eine bereits gute Anfangslösung dar. Schließlich kann mit der 3-Opt-Methode überprüft werden, ob weitere Verbesserungen der erzielten Resultate möglich sind. Eine andere Strategie zur Bildung einer guten Anfangslösung ist die in Kapitel 6.1.2 geschilderte Methode.

BN		**SE**		**SE**		**2-Opt**	**3-Opt**
St.kn.	km	Anf.kn.	km	Anf.kn.	km	km	km
1	1538	4, 1	1437	10, 1	1538		
2	1879	4, 2	1437	10, 2	1437	1437	1437
3	1878	4, 3	1437	10, 3	1437		
4	1518	---	---	10, 4	1437	Rund-	Rund-
5	1579	4, 5	1437	10, 5	1437	reise	Reise
6	1538	4, 6	1437	10, 6	1538	ist 2-	ist 3-
7	1624	4, 7	1437	10, 7	1538	optimal	optimal
8	1538	4, 8	1437	10, 8	1538		
9	1538	4, 9	1437	10, 9	1538		
10	1518	4, 10	1437	---	---		
Zeit:	0,4 s		1,2 s		1,2 s	0,8 s	5,6 s

Für das TSP 10 wird eine gute Lösung schon im BN-Verfahren, eine 3-optimale Rundreise bereits im SE-Verfahren gefunden.

Die Berechnungen wurden mit einem Intel Pentium III Prozessor (450 MHz) durchgeführt.

Tab. 6.2: Vergleich der Optimierungsergebnisse für das TSP 10

6.4 TSP 25 – große Rundreise durch Deutschland

		Aac	Bay	Ber	Bra	Bre	Dor	Dre	Erf	Fle	Fra	Fre	Ham	Han	Kar	Kas	Kie	Kon	Lei	Mar	Mün	Nür	Pas	Ros	Saa	Wür
1	Aachen	x	481	609	397	353	147	617	406	480	236	450	464	348	323	285	518	520	509	471	594	448	667	629	225	345
2	Bayreuth		x	353	315	460	409	227	145	595	246	416	483	348	299	257	570	360	191	276	233	78	238	515	399	138
3	Berlin			x	222	359	470	193	265	403	481	733	277	276	604	346	327	709	166	143	582	430	568	215	653	437
4	Braunschweig				x	166	258	291	170	298	313	576	173	67	445	149	259	618	189	87	533	368	546	278	486	329
5	Bremen					x	222	450	318	143	378	639	115	114	508	237	186	701	345	240	672	507	697	280	531	433
6	Dortmund						x	485	290	349	218	464	333	208	338	156	386	531	379	331	565	417	636	484	317	315
7	Dresden							x	213	575	428	640	428	351	512	332	499	583	108	212	448	305	429	393	600	345
8	Erfurt								x	453	228	472	339	206	344	134	428	467	116	155	365	199	382	403	400	177
9	Flensburg									x	520	781	156	251	650	379	77	842	487	375	807	641	833	255	658	575
10	Frankfurt										x	264	456	308	133	168	543	326	341	356	358	213	432	586	174	110
11	Freiburg											x	719	570	132	431	806	127	576	611	321	341	487	849	253	296
12	Hamburg												x	150	588	294	94	773	339	226	700	535	712	172	617	484
13	Hannover													x	439	145	237	623	247	141	559	394	585	284	480	334
14	Karlsruhe														x	300	675	200	448	481	287	224	431	718	126	167
15	Kassel															x	381	487	225	204	448	283	493	422	341	205
16	Kiel																x	860	411	299	792	627	792	179	703	571
17	Konstanz																	x	548	622	203	282	369	870	324	291
18	Leipzig																		x	113	421	269	402	342	513	281
19	Magdeburg																			x	506	354	493	249	530	331
20	München																				x	166	166	745	412	258
21	Nürnberg																					x	219	592	338	103
22	Passau																						x	731	544	322
23	Rostock																							x	759	579
24	Saarbrücken																								x	263
25	Würzburg																									x

Tab. 6.3: Die Kostenmatrix für das TSP 25 ist per Definition (vgl. Kap. 3.2.6.6) symmetrisch (Entfernung AB entspricht Entfernung BA) und braucht für weitere Berechnungen nur an ihrer Diagonale gespiegelt zu werden. Die Kilometerangaben wurden mit Hilfe des Routenplaners unter http://www.reiseplanung.de berechnet.

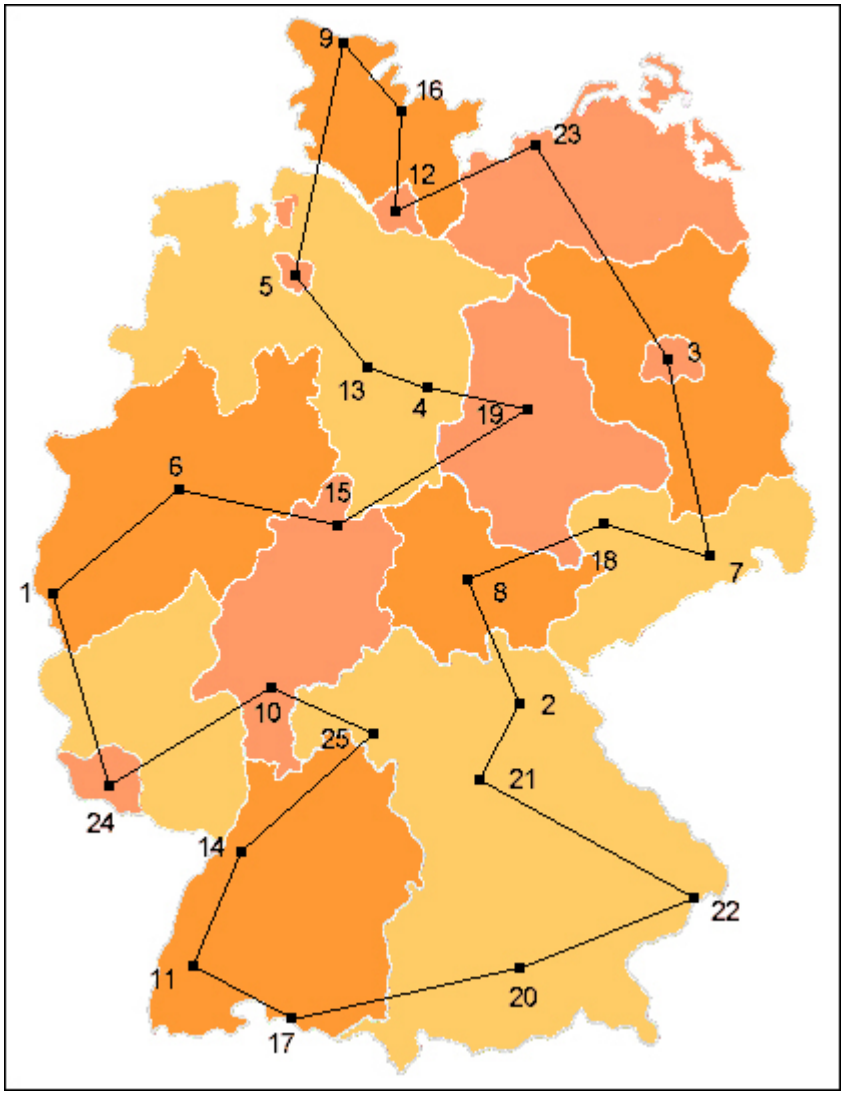


Abb. 6.4: Eine 3-optimale Rundreise durch 25 deutsche Städte, Gesamtlänge: 3639 km.

Insgesamt existieren (25-1)! / 2 = 310.224.200.866.619.719.680.000 mögliche Touren, die Nummerierung der Städte korrespondiert mit Tabelle 6.3.

Analog zum TSP 10 werden auch hier die Optimierungsmethoden Bester Nachfolger (BN) und Sukzessive Einbeziehung (SE) sowie das 2-optimale (2-Opt) und 3-optimale Verfahren (3-Opt) getestet.

6.4.1 Lösung BN

Initialisierung: Gewählter Startknoten: 3 , r = [3], c(r) = 0

1. Iteration: Knoten mit geringster Entf. zu 3 hinzugefügt: 19 , r = [3 19], c(r) = 143

...

25. Iteration, 1. Schritt: Abbruch: Alle Knoten aufgenommen. Rundreise wird geschlossen.

Ermittelte Rundreise :

r = [3 19 4 13 5 12 16 9 23 18 7 8 15 6 1 24 14 11 17 20 21 2 25 10 22 3], c(r) = **4357**

6.4.2 Lösung SE

Initialisierung: Gewählte Subtour: r = [3 19 3] , c(r) = 286

1. Iteration, 1. Schritt: Knoten mit maximaler kürzester Entfernung zu r : 17

1. Iteration, 2. Schritt: Knoten 17 in die Rundreise eingefügt r = [3 17 19 3], c(r) = 1474

...

24. Iteration, 1. Schritt: Abbruch: Alle Knoten in die Rundreise aufgenommen.

Ermittelte Rundreise :

r = [3 7 18 8 2 21 22 20 17 11 14 25 10 24 1 6 15 4 13 5 12 9 16 23 19 3], c(r) = **3715**

6.4.3 Lösung 2-Opt

Initialisierung, Start mit Rundreise:

r = [3 7 18 8 2 21 22 20 17 11 14 25 10 24 1 6 15 4 13 5 12 9 16 23 19 3], c(r) = 3715

1. Iteration, Untersuchung aller 2-Tausche:

Tausch von [3,19] ,[23,16] gegen [3,23] ,[19,16] , Kostendifferenz = 192

...

Tausch von [9,16] ,[23,3] gegen [9,23] ,[16,3] , Kostendifferenz = 290

Abbruch: Keine Verbesserung durch 2-Tausch mehr möglich. Rundreise ist 2-optimal:

r = [3 7 18 19 8 2 21 22 20 17 11 14 25 10 24 1 6 15 4 13 5 12 9 16 23 3], c(r) = **3690**

6.4.4 Lösung 3-Opt

Initialisierung, Start mit Rundreise:

r = [3 7 18 19 8 2 21 22 20 17 11 14 25 10 24 1 6 15 4 13 5 12 9 16 23 3], c(r) = 3690

1. Iteration, Untersuchung aller 3-Tausche:

Tausch von [3,7] ,[7,18] ,[18,19] gegen [7,19] ,[7,18] ,[18,3] , Kostendifferenz = 72

...

Tausch von [23,3] ,[9,16] ,[12,23] gegen [3,23] ,[9,12] ,[16,23] , Kostendifferenz = 86

Abbruch: Keine Verbesserung durch 3-Tausch mehr möglich. Rundreise ist 3-optimal:

r = [3 7 18 8 2 21 22 20 17 11 14 25 10 24 1 6 15 19 4 13 5 9 16 12 23 3], c(r) = **3639**

6.4.5 Strategisches Vorgehen

Analog zu Kapitel 6.3.5 soll auch hier ein Blick auf die systematische Vorgehensweise geworfen werden:

BN		**SE**		**SE**		**2-Opt**	**3-Opt**
St.kn.	km	Anf.kn.	km	Anf.kn.	km	km	km
1	4287	10, 1	3690	25, 1	3808		
2	4224	10, 2	3690	25, 2	3808		
3	4357	10, 3	3690	25, 3	3808		
4	4234	10, 4	3699	25, 4	3817		
5	4422	10, 5	3698	25, 5	3798		
6	4490	10, 6	3734	25, 6	3838		
7	4113	10, 7	3699	25, 7	3817		
8	4213	10, 8	3700	25, 8	3808		
9	4593	10, 9	3699	25, 9	3808		
10	*3941*	---	---	25, 10	3690	*3680*	*3639*
11	4404	10, 11	3704	25, 11	3808		
12	4500	10, 12	3698	25, 12	3816	Rundreisen	Rundreise
13	4212	10, 13	3690	25, 13	3808	10, 6 ... und	ist
14	4207	10, 14	3690	25, 14	3808	25,17... sind	3-optimal
15	4364	10, 15	3698	25, 15	3698	2-optimal	
16	4536	10, 16	*3680*	25, 16	3878	(aber nicht	
17	4218	10, 17	3690	25, 17	*3680*	identisch!)	
18	4232	10, 18	3708	25, 18	3816		
19	4178	10, 19	3698	25, 19	3816		
20	4330	10, 20	3704	25, 20	3690		
21	4184	10, 21	3700	25, 21	3808		
22	4293	10, 22	3690	25, 22	3808		
23	4338	10, 23	3690	25, 23	3808		
24	4593	10, 24	3704	25, 24	3720		
25	*3941*	10, 25	3690	---	---		
Zeit:	2 s		11 s		11 s	8 s	9 m 14 s

Tab. 6.4: Vergleich der Optimierungsergebnisse für das TSP 25

Beim TSP 25 wird eine 2-optimale Lösung bereits mit dem SE-Verfahren gefunden. Für die Ermittlung der 3-optimalen Rundreise hingegen ist ein erheblicher Mehraufwand an Rechenschritten notwendig ($O(n^3)$ beim 3-Opt im Gegensatz zu $O(n^2)$ beim 2-Opt).
Vergleicht man die Berechnungszeiten der besten Lösungen von TSP 10 und TSP 25 (5,6 sec zu 9 min 14 sec) wird deutlich, wie schnell (nämlich exponentiell) der Lösungsaufwand mit der Anzahl *n* der Städte wächst.

Im Vergleich zum derzeitigen Weltrekord sind die beiden angeführten Beispiele eher winzig. Im Sommer 1998 gelang den amerikanischen Wissenschaftlern Applegate, Bixby und Cook von der Rice University sowie Chvatal von der Rutgers University mit einem Branch and Cut - Verfahren die nachweisbar kürzeste Rundreise durch 13509 Orte in den USA (vgl. Abb. 6.5).
Die Tour geht durch alle Städte der Vereinigten Staaten mit mehr als 500 Einwohnern und läßt den bisherigen Rekord von 7397 Städten weit hinter sich. Eine beachtliche Leistung, wenn man bedenkt, daß (13509-1)! / 2 ≅ $5{,}462941 \cdot 10^{49931}$ verschiedene Möglichkeiten zur Auswahl stehen (eine Zahl die wohl selbst das Vorstellungsvermögen von Astronomen übersteigen dürfte). Das Ende der Fahnenstange ist allerdings auch mit dieser Marke sicherlich noch nicht erreicht.

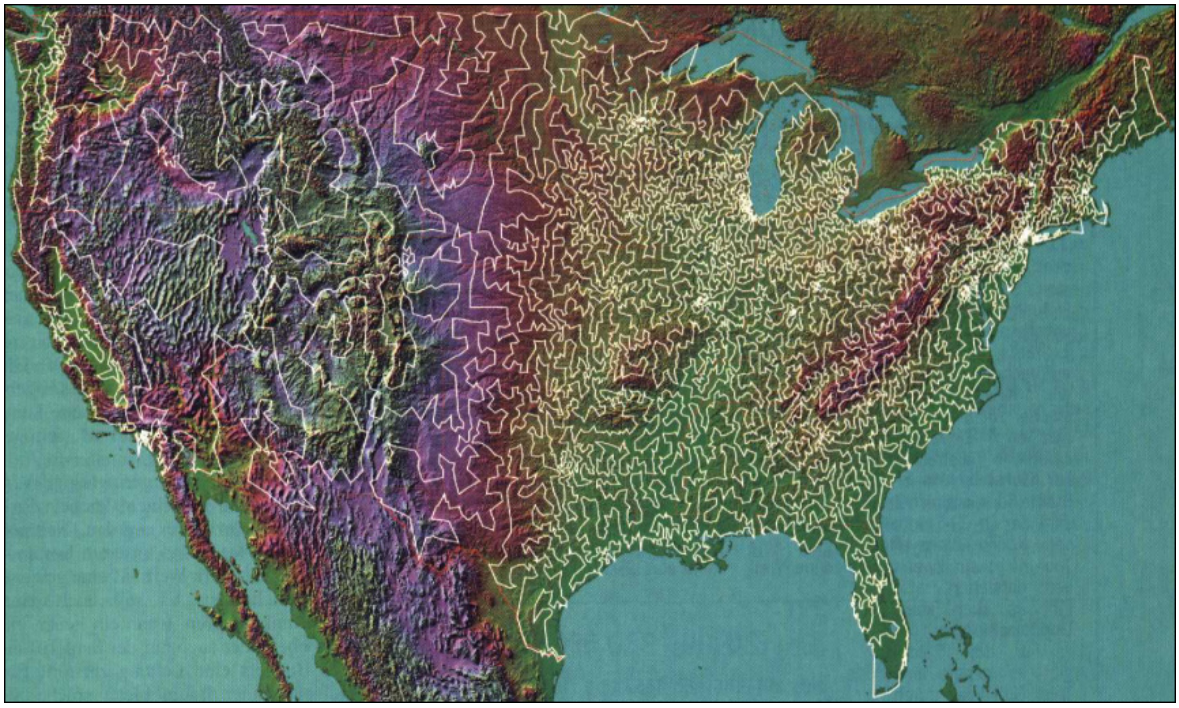


Abb. 6.5: TSP-Weltrekord: die kürzeste Rundreise durch 13509 amerikanische Städte

Das Programmpaket CONCORDE (Combinatorial Optimization and Networked Combinatorial Optimization Research and Development Environment), mit dem Applegate, Bixby, Cook und Chvatal die obige Rundreise berechnet haben, kann zu wissenschaftlichen Zwecken über die URL http://www.keck.caam.rice.edu/concorde.html kostenlos heruntergeladen werden. Es enthält neben dem Modul zur Lösung großer TSP noch etliche andere Optimierungsfunktionen und läßt sich auf einem UNIX oder LINUX-Rechner installieren. Der Quellcode aller Programmteile liegt in ANSI C vor.

7. Realisierung der Caritas-Routenplanung in einem GIS

Nach den verschiedenen Optimierungsmethoden soll nun die Umsetzung der Routenplanung für die Caritas-Sozialstation innerhalb eines Geo-Informationssystems (GIS) beschrieben werden.

7.1 Was ist ein Geo-Informationssystem?

Nach BARTELME dient ein Geo-Informationssystem der „Erfassung, Speicherung, Verarbeitung und Darstellung aller Daten, die einen Teil der Erdoberfläche und die darauf befindlichen technischen und administrativen Einrichtungen sowie ökonomische und ökologische Gegebenheiten beschreiben“ (BARTELME 1988, S. 5).

BILL definiert ein GIS als „ rechnergestütztes System, das aus Hardware, Software, Daten und Anwendungen besteht. Mit ihm können raumbezogene Daten digital erfaßt und redigiert, gespeichert und reorganisiert, modelliert und analysiert sowie alphanumerisch und graphisch präsentiert werden“ (BILL 1999 A, S. 4).

Unter *raumbezogenen* Daten versteht man Daten, die in irgendeiner Form mit einer bestimmten Position auf der Erde verknüpft sind. Häufig liefern solche Daten direkte Informationen zu realen (flächenhaften oder räumlichen) Objekten, in unserem Fall z.B. zu Straßenabschnitten, Wohnblöcken, Grundstücken oder Häusern. Man spricht dann von einer primären Metrik (BILL 1999 A, S. 5). Es kann sich aber auch um ’immaterielle’ Daten wie Postleitzahlen, Hausnummern oder Einwohnerzahlen handeln, die den Bezug zu einem Objekt nur indirekt herstellen (sekundäre Metrik). Im Vermessungswesen wird die räumliche Verknüpfung erreicht, in dem man allen Objekten 2- oder 3-dimensionale Koordinaten zuweist. Auf diese Weise läßt sich z.B. ein Wohnhaus eindeutig in einem Stadtplan oder einer Landkarte lokalisieren.

7.2 ArcView GIS

Für die Umsetzung der Routenplanung in ein rechnergestütztes GIS-Projekt habe ich das Programmsystem „ArcView“ der Firma ESRI (Environmental Systems Research Institute) gewählt, welches in der neuesten Version 3.2 am Fachgebiet Photogrammetrie und Kartographie der TU Berlin vorhanden ist. ArcView bietet geeignete Module und Funktionen zur Bearbeitung der Aufgabenstellung und ist „mit über 500.000 verkauften Lizenzen das weltweit meist genutzte Desktop-GIS“ (Quelle: ESRI). Ein entscheidender Vorteil von ArcView, welcher einhergeht mit dem großen Kundenkreis, sind die zahlreichen Zusatzanwendungen, die von kommerziellen und privaten Nutzern programmiert wurden. Eine große Anzahl dieser Skripte werden zum kostenlosen

Download auf der Homepage von ESRI angeboten (http://gis.esri.com/arcscripts/scripts.cfm). Zudem ist mit der systemeigenen Programmiersprache „Avenue“ die Möglichkeit gegeben, eigene Applikationen zu entwickeln.

Der strukturelle Aufbau eines ArcView - Projektes kann folgendermaßen skizziert werden:

- Den Kern bildet die *Projektdatei* (*.apr). In ihr werden Verweise zu allen Datensätzen (z.B. Bilder, Tabellen oder Shape-Files) gespeichert, die für die Ausführung des Projektes notwendig sind. Solche Dateien sind in der Regel mit den folgenden Projektkomponenten verknüpft:
- *Views* stellen verschiedene Ansichtsfenster innerhalb eines Projektes dar.
- Ein *View* ist in mehrere *Themes* (Themen) untergliedert. Diese Themen sind vergleichbar mit Layern eines Bildverarbeitungsprogramms und ermöglichen die getrennte Darstellung von unterschiedlichen Objekten auf mehreren Ebenen.
- *Tables* sind Tabellen, aus denen Werte eingelesen und verarbeitet werden können.
- *Charts* sind Diagramme mit denen Daten graphisch präsentiert werden können.
- Durch *Layouts* können Formatierungen für den Ausdruck von Projektansichten vorgenommen werden.
- Mit Hilfe von *Scripts* (Skripten) können eigene Anwendungen erstellt werden.

7.2.1 Programmerweiterungen

Das Standardsoftwarepaket von ArcView enthält ein Basismodul, in welchem die Grundwerkzeuge (*Tools*) zur Bearbeitung eines GIS-Projektes implementiert sind. Für Spezialanwendungen sind Programmerweiterungen, sogenannte *Extensions*, erforderlich, die zusätzliche Funktionalität bieten. Größere Extensions müssen meist separat installiert werden und lassen sich dann in der Menüsteuerung des Basismoduls aktivieren.

Bei der Erstellung des Routenoptimierungsprojektes kamen mehrere Extensions zum Einsatz, von denen die folgenden auch für die Projektarbeit gebraucht werden:

- Network Analyst, Version 1.0, der Firma ESRI

  Diese Erweiterung bietet die Möglichkeit, kürzeste und schnellste Wege in Netzwerken zu finden und ist sozusagen das Herzstück des Projektes. Mich hat natürlich interessiert, welche Methode der Network Analyst zur Routenoptimierung benutzt. Von ESRI kam folgende Antwort per E-Mail: “The latest version of the Network Analyst extension uses Tabu Search heuristic with 2-opt as the only move set defining the neighborhood. The next release of Network Analyst for Arcview 8.x will contain a revised algorithm for the traveling salesman problem.”

Der gesamte E-Mail-Verkehr mit der ESRI-Zentrale in den USA läuft über die deutsche Hauptniederlassung, so daß es nicht möglich war, die zuständigen Leute direkt zu befragen. Weitere Informationen konnten leider nicht eingeholt werden.

- Spatial Analyst, Version 1.1, der Firma ESRI
  Mit dieser Extension können statistische Analysen, z.B. zur Verteilung und Altersstruktur der Bevölkerung, vorgenommen und graphisch dargestellt werden. Eine weitere Funktion dient der Durchführung von Abstandstransformationen. Fragestellungen vom Typ „wo ist der ideale Ort für eine weitere Sozialstation“ können mit Hilfe dieses Werkzeugs bearbeitet werden.
- Deutsche Adressen-Geocodierung, Version 1.2, der Firma ESRI
  Die Standardausführung von ArcView enthält amerikanische Regeln zur Adressen-Geokodierung. Wenn man deutsche Adressenformate benutzen möchte, sollte man auf diese Erweiterung zurückgreifen (Weiteres dazu in Kapitel 7.5.2).
- CSS_Temphof, Version 1.0 beta, selbst geschrieben
  Diese Extension paßt die Benutzeroberfläche von ArcView an das Routen-optimierungsproblem an. Nur Menüoptionen, Schaltflächen, Werkzeuge, Ansichtsfenster und Themen erscheinen, die für die Arbeit mit dem Projekt wirklich notwendig sind.

Zwei Erweiterungen haben die Erstellung des Projektes wesentlich erleichtert, werden aber für die Nutzung der Applikation nicht unbedingt benötigt:

- Table Edit Extension, Version 2.1, der Firma SWEGIS
  Mit dieser Extension können Tabellen (z.B. Adressenlisten oder Dateien der Straßendatenbank) komfortabel editiert werden.
- Analysis Extension, Version 1.3, der Firma SWEGIS
  Die Erweiterung bietet Möglichkeiten zur Analyse und Konvertierung von Objekten. Polylinien lassen sich beispielsweise unkompliziert in Polygone umwandeln.

Die letzte Extension ist zur Ausführung des Projektes ebenfalls nicht vonnöten, bietet aber interessante Nutzungsmöglichkeiten für die Zukunft:

- HTML Image Mapper, Version 3.0, der Firma Alta 4.
  Aus einem View und den dazugehörigen Themen können automatisch Dateien im HTML-Format (Hyper Text Markup Language) generiert werden. Dies gestattet z.B. die Erstellung von Internetkarten, in denen die Attribute verschiedener Themen gespeichert sind und per Mausklick online abgerufen werden können.

7.3 Daten

Als Grundlage für die rechnergestützte Umsetzung der Routenplanung mußten geeignete raumbezogene Daten für das Interessengebiet ’Bezirk Tempelhof und Ortseil Lankwitz’ beschafft werden. Ich habe folgendes kartographisches und statistisches ’Rohdaten-Material’ zur Weiterverarbeitung benutzt:

- Karte von Berlin (K10) im Rasterdatenformat, Maßstab 1:10 000. Insgesamt werden 15 Kartenblätter benötigt: 303-1, 303-2, 303-3, 303-4 / 304-2 / 403-1, 403-2, 403-3, 403-4 / 404-2, 404-4 / 413-1, 413-2, 413-3, 413-4. Der Datensatz befindet sich im Besitz des Fachgebietes Photogrammetrie und Kartographie der TU Berlin. Herausgeber des Kartenwerks ist die Senatsverwaltung für Stadtentwicklung, Abteilung III - Geoinformation und Vermessung.
- Digitale Grundkarte von Berlin (DIGK) im Vektordatenformat, Maßstab variabel. Die Daten liegen im Drawing-Exchange-Format (DXF) vor und enthalten das öffentliche Verkehrsstraßennetz, Wohnblöcke und statistische Gebiete. Herausgeber des Kartenwerks ist die Senatsverwaltung für Stadtentwicklung, Abteilung III - Geoinformation und Vermessung. Das Material wurde mit Genehmigung des Herausgebers vom Statistischen Landesamt Berlin im Rahmen einer wissenschaftlichen Arbeit kostenlos zur Verfügung gestellt.
- Einwohnerzahlen auf Wohnblockebene in acht verschiedenen Altersstufen: unter 6, 6-14, 15-17, 18-26, 27-44, 45-54, 55-64 und über 64 Jahre. Zusätzlich ist der Anteil der Ausländer enthalten. Die Daten liegen in Form einer Excel-Tabelle vor und wurden vom Statistischen Landesamt Berlin, Referat II A 16 - Einwohnerregister, käuflich erworben.
- Adressenlisten mit Kunden der Caritas-Sozialstation. Die Daten liegen als Excel-Tabellen vor und umfassen den aktuellen Kundenstamm der Monate Mai und September 2000. Aus Gründen des Datenschutzes sind keine Namen angegeben. Die Listen wurden im Rahmen einer wissenschaftlichen Arbeit von der Station kostenlos zur Verfügung gestellt.

Ich habe allen Institutionen versichert, daß das bereitgestellte Material ausschließlich zu wissenschaftlichen Zwecken genutzt wird und nicht vervielfältigt oder an Dritte weitergegeben wird.

7.3.1 Kartenhintergrund

Als Hintergrund für das Interessengebiet Tempelhof / Lankwitz wurden die oben erwähnten Blätter der K10 verwendet. Die Karte zeigt Verkehrswege, Straßennamen, Gebäude, Parkanlagen, Gewässer, Grundstücksgrenzen, Hausnummerbereiche etc. im Fünffarbendruck (vgl. Abb. 7.1) und dient als Orientierungshilfe für die Routenplanung. Außerdem bietet sie eine gute Möglichkeit, die von ArcView lokalisierten Kundenadressen (Straße, Hausnummer) zu überprüfen (Weiteres dazu in Kapitel 7.5.3).

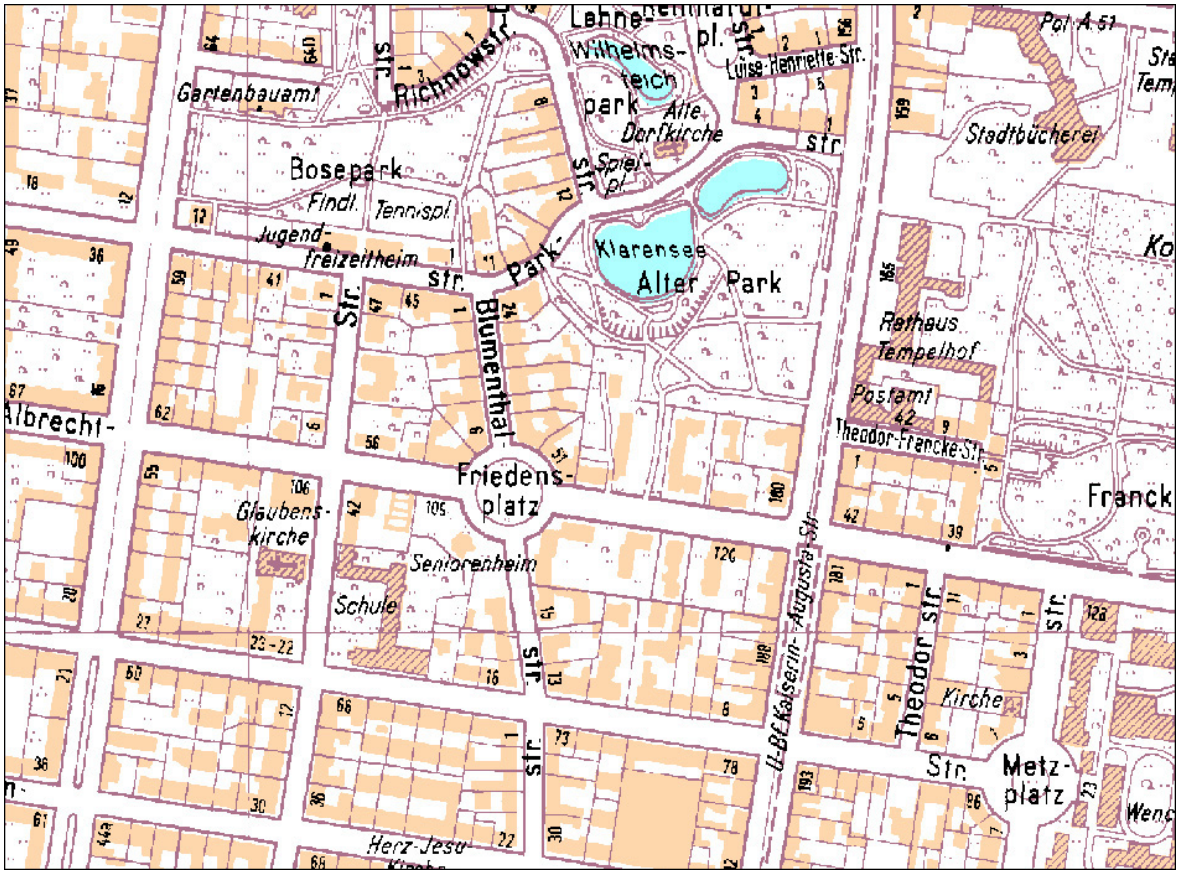


Abb. 7.1: Ausschnitt der K10 – Rasterdatenbild

Bei den Kartenblättern handelt es sich um sogenannte Rasterdaten. Das bedeutet, daß alle abgebildeten Details aus einzelnen Bildelementen (Picture Elements, kurz Pixeln) zusammengesetzt sind, die sich lediglich durch ihren Grauwert (bei Schwarzweiß-Bildern) oder ihre Farbe (bei Farbbildern) unterscheiden. Welche Pixel zusammengehören und welches Pixel zu welchem Objekt kann ein Computerprogramm ohne spezielle Bildverarbeitungsverfahren nicht feststellen - Straßen und Häuser werden als solche nicht erkannt. Aus diesem Grunde kommen Rasterdaten für die Routenoptimierung nicht in Frage und dienen lediglich als Hintergrundinformation.

7.3.2 Straßenverkehrsnetz

Der wichtigste Datensatz für die Routenoptimierung ist das Straßenverkehrsnetz. Ein Straßensegment im Straßennetz wird als einfache Linie idealisiert und stellt meist die Achse der Straßenbegrenzung dar (vgl. Abb. 7.2). Damit ein Computerprogramm in der Lage ist, diese Linien „zu durchlaufen“, muß es sich bei den Daten, die das Straßennetz beschreiben, um sogenannte Vektordaten handeln. Jedes Objekt in diesem Datentyp wird koordinatenmäßig erfaßt und gespeichert (vgl. Kapitel 7.3.3). Auf diese Weise lassen sich Aussagen über die Größe und Richtung eines Objektes treffen. Im Gegensatz zu Rasterdaten kann man bei Vektordaten also von „intelligenten Daten“ sprechen.

Für die Modellierung des Tempelhofer Straßennetzes habe ich die DXF-Dateien des Statistischen Landesamtes mit ArcView eingelesen und anschließend in das ArcView-typische Shape-Format (*.shp) konvertiert. Attribute wie Straßennamen oder Hausnummernbereiche blieben dabei erhalten.

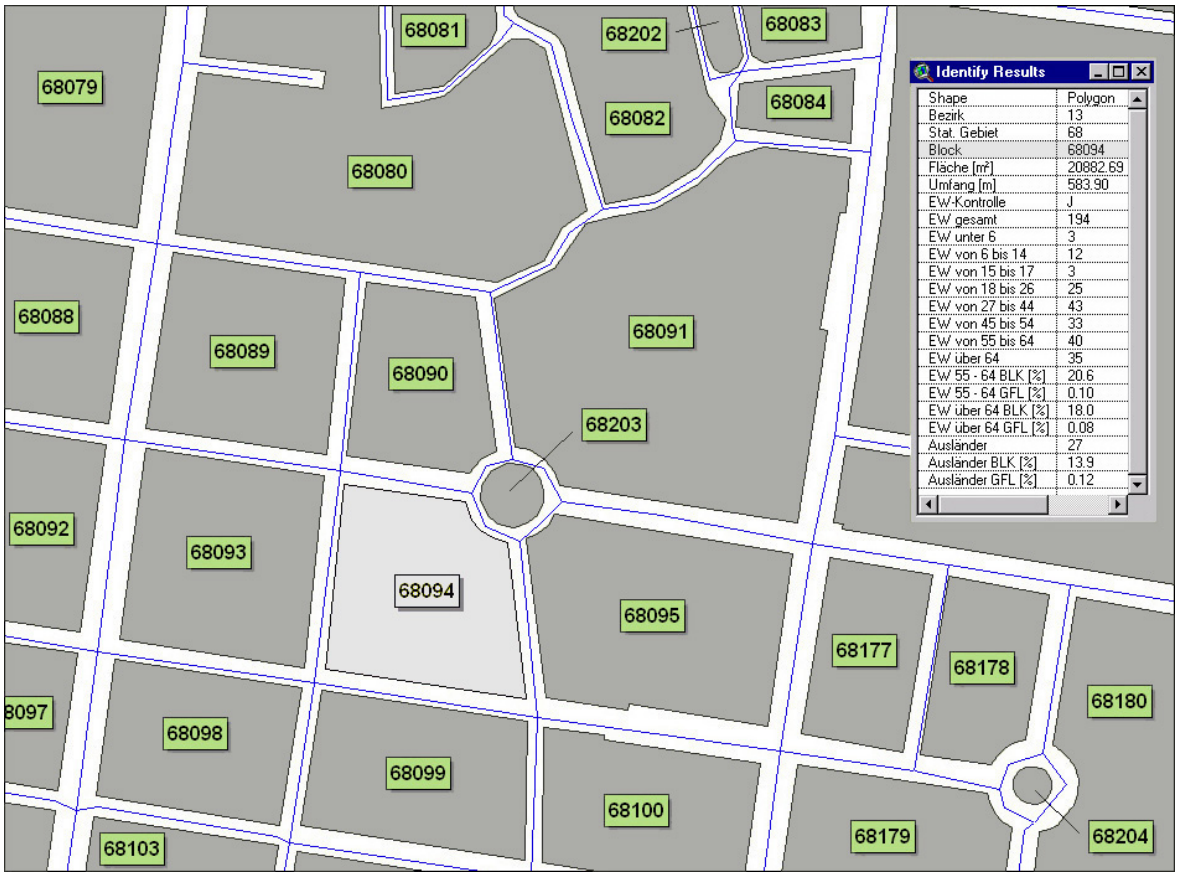
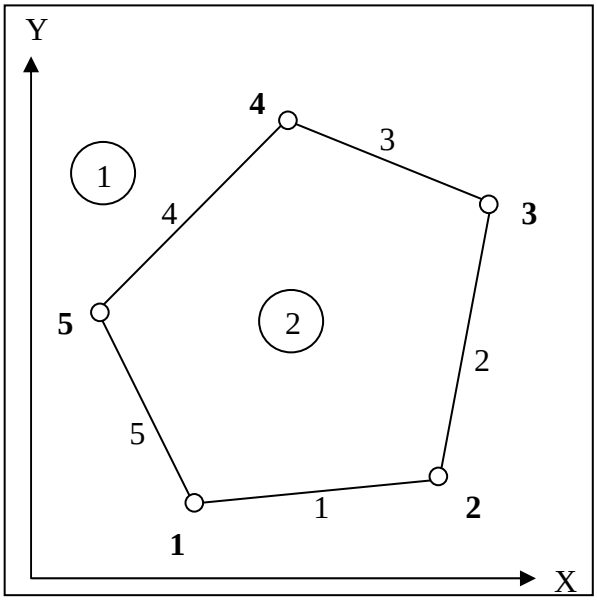


Abb. 7.2: Straßennetz und Wohnblöcke in vektorisierter Form (vgl. mit Abb. 7.1)
(EW steht für Einwohner, BLK ist die Einwohnerzahl in Prozent bezogen auf einen Wohnblock, GFL ist die Einwohnerzahl in Prozent bezogen auf die Gesamtfläche des Interessengebietes)

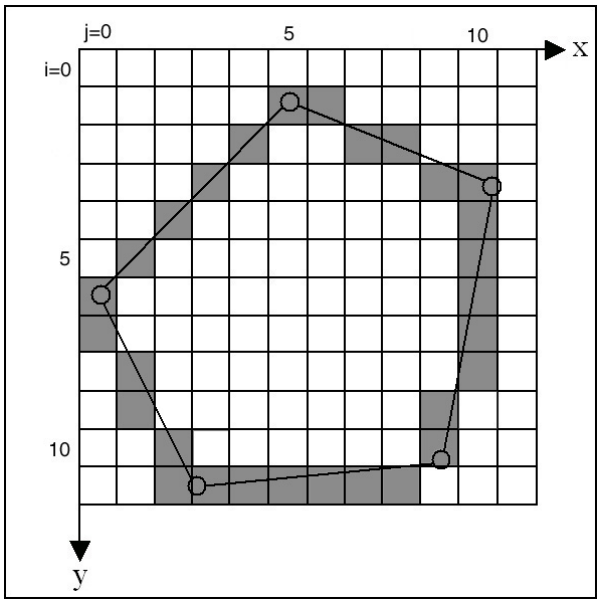
7.3.3 Speicherung von Raster- und Vektordaten



Kante	von	nach	li. Fläche	re. Fläche
1	1	2	2	1
2	2	3	2	1
3	3	4	2	1
4	4	5	2	1
5	1	5	1	2

Knoten	X	Y
1	x_1	y_1
2	x_2	y_2
3	x_3	y_3
4	x_4	y_4
5	x_5	y_5

Abb. 7.3: Speicherung einer Vektorgraphik als Kanten- und Knotentabelle



Grauwerte $g_{i,j}$ (i = Zeile, j = Spalte)

$g_{0,0}$ $g_{0,1}$ $g_{0,2}$ . . . . . $g_{0,11}$

$g_{1,0}$.

$g_{2,0}$.

. .

. $g_{4,2} = 0$ (schwarz) .

. $g_{5,2} = 255$ (weiß) .

. .

. (Bei einer Bildtiefe von 8 Bit .

. existieren $2^8 = 256$ Grauwerte) .

. .

. .

$g_{11,0}$ . . . . . $g_{11,11}$

Abb. 7.4: Speicherung einer Rastergraphik als Pixel-Grauwert-Matrix

Vektorgraphiken zeichnen sich durch logische Datenstrukturierung, einen guten Objektbezug und geringe Datenmengen aus. Punkte und Linien bilden die graphischen Grundstrukturen, Flächen erscheinen als geschlossene Linienzüge. Durch die Ordnung nach Objektlinien entsteht eine linienhafte Betrachtungsweise. Rasterdaten hingegen sind nur nach der Position der Pixel geordnet, was die logische Datenstrukturierung und den Objektbezug stark einschränkt. Eine flächenhafte Betrachtungsweise dominiert, die anfallenden Datenmengen sind in der Regel sehr groß (nach BILL 1999 A, S. 21 ff.).

7.3.4 Statistische Gebiete und Wohnblöcke

Ebenfalls ins Shape-Format umgewandelt wurden die DXF-Dateien mit den statistischen Gebieten und Wohnblöcken. Statistische Gebiete sind größere Flächeneinheiten, welche häufig den Grenzen von Ortsteilen entsprechen (vgl. Abb. 7.5). Wohnblöcke werden an ihren Rändern in den meisten Fällen von Straßenzügen begrenzt. Informationen wie Blocknummer, Fläche, Umfang, Einwohner- oder Ausländerzahlen können hier abgefragt werden (vgl. Abb. 7.2, S. 78).

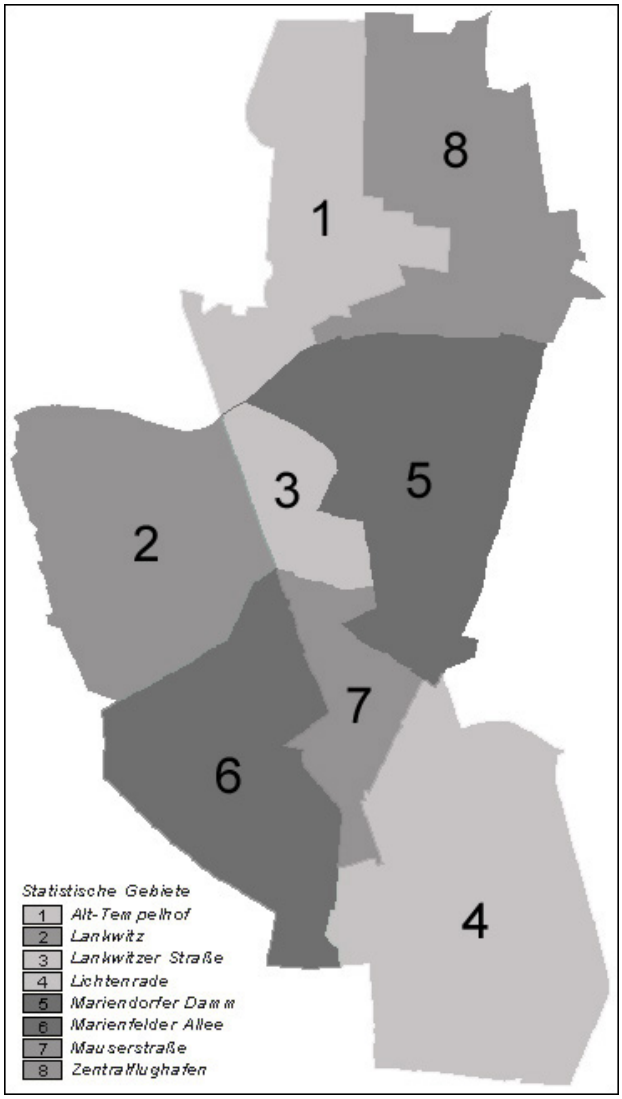


Abb. 7.5: Statist. Gebiete im Bezirk Tempelhof

Berlin vom Großen ins Kleine (vgl. Abb. 7.6): Es existieren 23 Bezirke, 79 Ortsteile, 190 Postleitzahlbereiche, 195 statistische Gebiete, 338 Verkehrszellen, 880 Teilverkehrszellen und ungefähr 12100 Wohnblöcke.

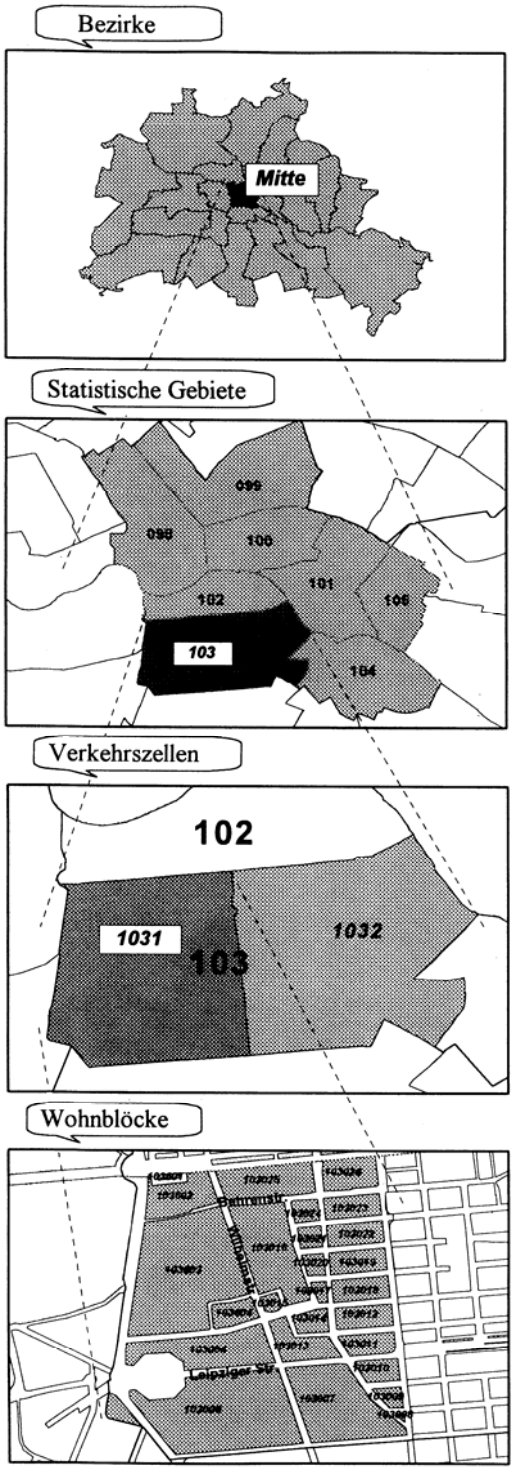


Abb. 7.6: Regionale Gliederung

7.3.5 Einwohnerzahlen

Die vom Statistischen Landesamt erworbene Excel-Datei mit den Einwohnerzahlen des Interessensgebietes habe ich in einen dBASE-File (*.dbf) konvertiert. Dieses Format wird von ArcView für die Arbeit mit Tabellen bevorzugt und kann direkt eingelesen werden.

Eine Editierung der Daten kann direkt (z.B. mit Excel), aber auch unter ArcView vorgenommen werden - das Programm verfügt über Lese- und Schreibrechte für alle externen Dateien. Als sehr nützlich bei der Speicherung und Verwaltung von Projektdaten, welche sich in Tabellen befinden, hat sich die Funktion 'Join Table' erwiesen. Diese ermöglicht das Zusammenfügen von Tabellen über gemeinsame Datenfelder. So konnte z.B. die dBASE-Tabelle mit den Einwohnerzahlen über das Feld 'Blocknr.' an die Attribut-Tabelle des Themas Wohnblöcke angehängt werden. Eine Modifizierung der Originaldaten durch aufwendiges Zusammenkopieren erübrigte sich somit.

Als Ergebnis lassen sich für jeden Wohnblock neben Attributen wie Fläche, Umfang und Blocknummer auch detaillierte Einwohnerzahlen und Ausländeranteile abfragen. Die Aufschlüsselung der Bevölkerung erfolgt dabei in acht verschiedenen Altersgruppen:

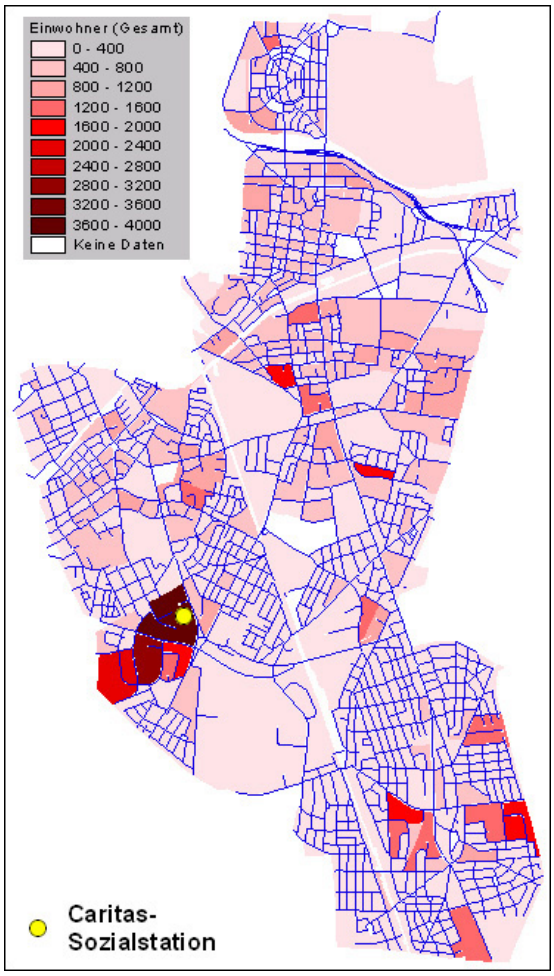


Abb. 7.7: Einwohneranzahl (insgesamt)

- Einwohner unter 6 Jahre
- Einwohner von 6 bis 14 Jahre
- Einwohner von 15 bis 17 Jahre
- Einwohner von 18 bis 26 Jahre
- Einwohner von 27 bis 44 Jahre
- Einwohner von 45 bis 54 Jahre
- Einwohner von 55 bis 64 Jahre
- Einwohner über 64 Jahre

Für eine graphische Präsentation der statistischen Daten innerhalb von ArcView müssen die Shape-Dateien, welche die relevanten Informationen enthalten, in sogenannte Grid-Files umgewandelt werden.

Die Abbildungen 7.7 (links) und 7.8 bis 7.13 (auf den nächsten beiden Seiten) zeigen die Verteilung der potentiellen Caritas-Kunden (ab 55 Jahre) und der Ausländer im Interessengebiet.

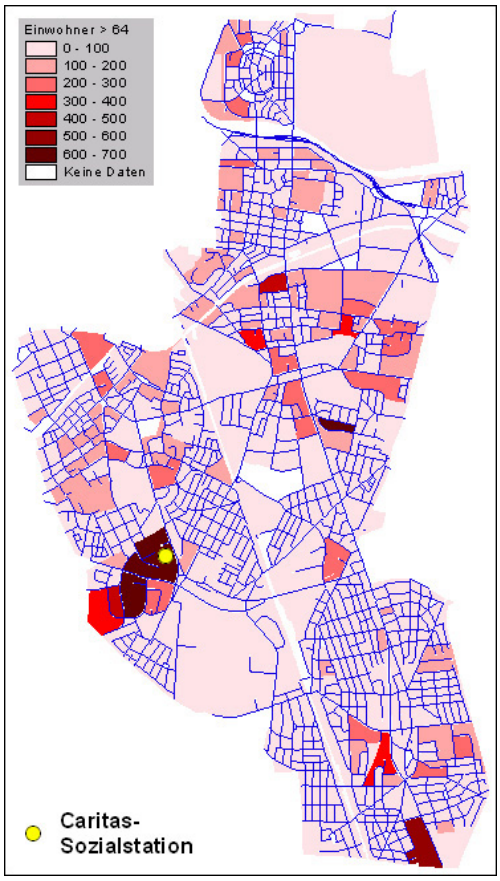


Abb. 7.8: Einwohner über 64 Jahre

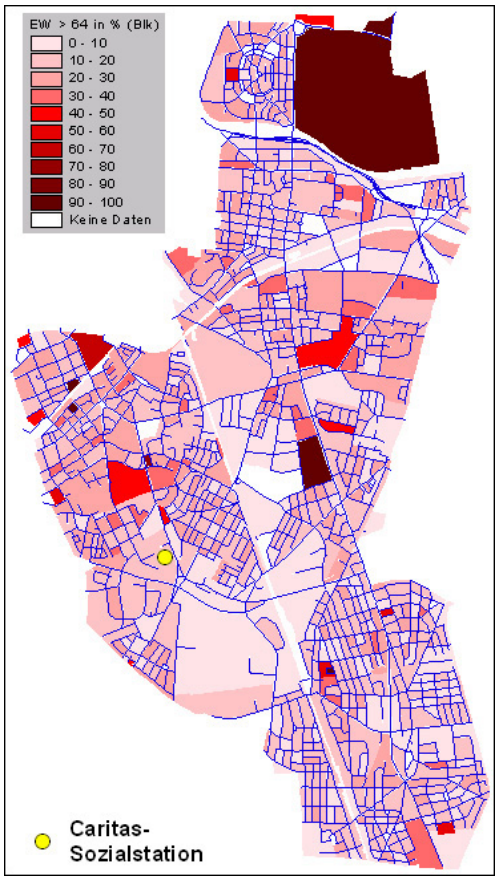


Abb. 7.9: Einwohner über 64 Jahre in %

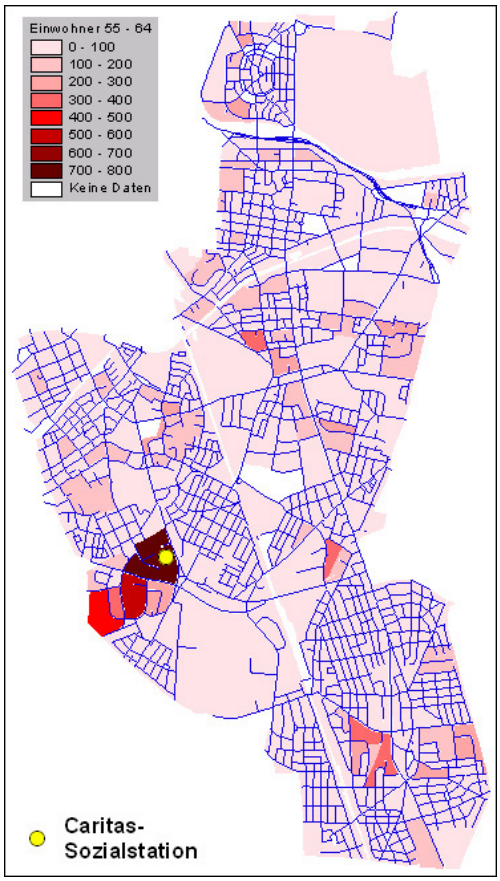


Abb. 7.10: EW von 55 bis 64 Jahre

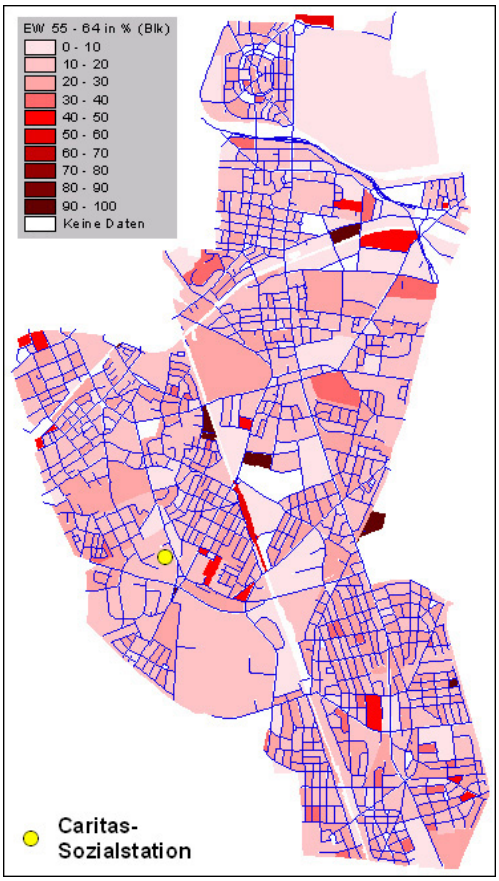


Abb. 7.11: EW von 55 bis 64 Jahre in %

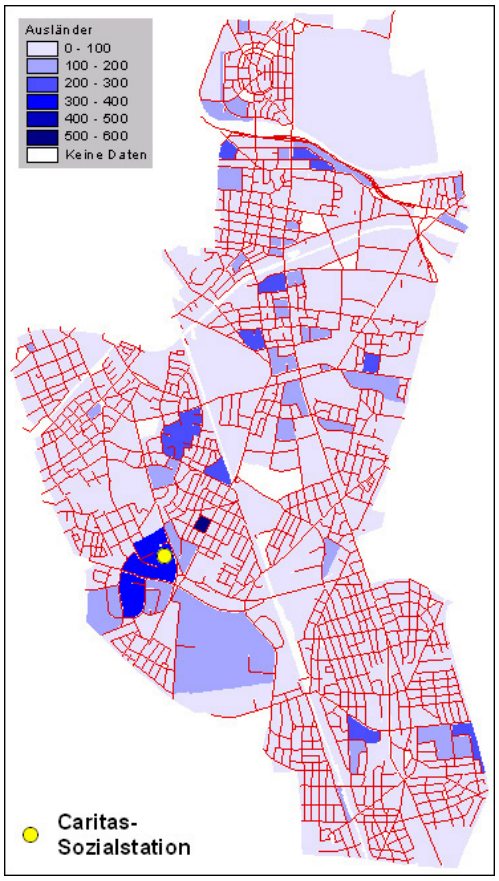


Abb. 7.12: Ausländeranzahl

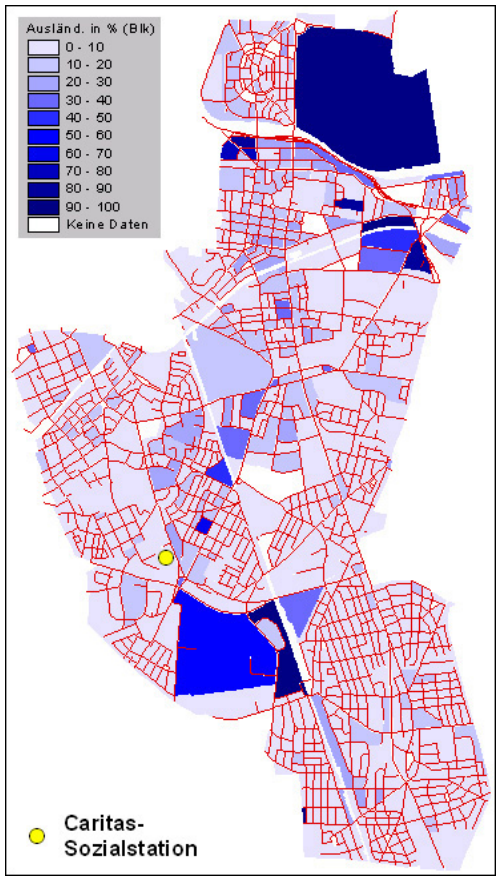


Abb. 7.13: Ausländeranzahl in %

Kurioses Ergebnis der Statistik: Auf dem Areal des Zentralflughafens Berlin-Tempelhof, welches einen statistischen Wohnblock bildet (vgl. Abb. 7.13 rechts oben), leben insgesamt drei ausländische Bürger, die zudem alle älter als 64 Jahre sind.

7.3.6 Kunden-Adressenlisten

Von der Caritas-Station in Tempelhof habe ich zwei Datensätze mit Adressen von aktuellen Kunden in Form von Excel-Tabellen bekommen. Die erste Liste vom Monat Mai 2000 enthält 205 Kunden mit den dazugehörigen Adressen (Straße, Hausnummer, Postleitzahl). Die Namen der Personen wurden in Kunde1, Kunde2 usw. geändert. Die Liste vom September 2000 enthält 149 Kunden. Zusätzlich zu den obigen Attributen ist hier noch das Alter der Personen angegeben. Die beiden Excel-Tabellen wurden ins dBASE-Format konvertiert und in ArcView eingelesen. ArcView besitzt Funktionalitäten, mit denen Adressen direkt in ein Thema importiert und graphisch dargestellt werden können. So soll z.B. ein Haus, in dem ein Kunde wohnt, an der entsprechenden Stelle des Straßennetzes durch ein ausgewähltes Symbol dargestellt werden. Eine ausführliche Beschreibung zum Thema ’Adressen’ erfolgt in Kapitel 7.5.

7.4 Koordinatensystem

Alle Objekte, die in den verschiedenen Themen eines Views abgebildet sind, sollen eindeutig lokalisiert werden können. Außerdem soll ihre Ausdehnung meßbar sein. Um dies zu gewährleisten, wird ein einheitliches Koordinatensystem benötigt. Für die drei Vektordatensätze des Statistischen Landesamtes (Straßennetz, Wohnblöcke und statistische Gebiete) war ein solches Referenzsystem a priori gegeben. Ihnen liegt das Regionale Bezugssystem (RBS) für Berlin in Soldner-Koordinaten zugrunde (vgl. Abb. 7.14). Dieses System ist bei der Konvertierung der DXF-Dateien ins Shape-Format erhalten geblieben.

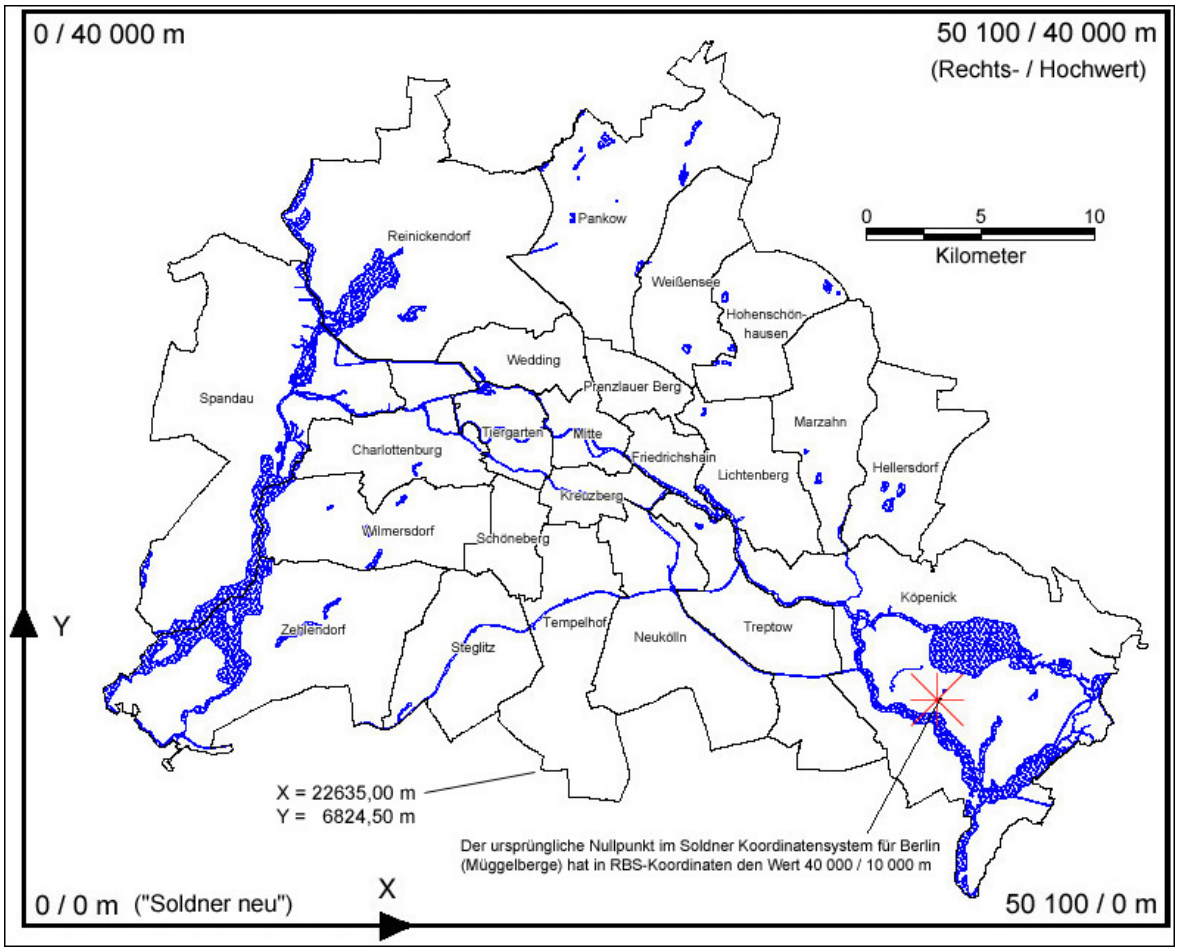


Abb. 7.14: Regionales Bezugssystem für Berlin

Die in den Shape-Files enthaltenen Vektordaten speichert ArcView in einem kartesischen X-Y-Koordinatensystem, welches allgemein als „Weltkoordinatensystem“ bezeichnet wird. Dieses Weltkoordinatensystem ist im vorliegenden Projekt gleichzusetzen mit dem Regionalen Bezugssystem (RBS). Anstelle von ’Weltkoordinatensystem’ und ’Weltkoordinaten’ wird im weiteren Text deswegen von RBS und Soldner-Koordinaten gesprochen.

7.4.1 Geokodierung von Rasterdaten

Damit die Vorgabe eines einheitlichen Referenzrahmens eingehalten wird, müssen die als Hintergrund verwendeten Kartenblätter der K10 in das Regionale Bezugssystem eingepaßt werden. Den Vorgang, der die Transformation der Rasterdaten in das Soldner-Koordinatensystem beschreibt, nennt man Geokodierung. (Im englischen Sprachgebrauch wird in diesem Zusammenhang auch häufig der Begriff 'Georeferencing' benutzt. Das deutsche Äquivalent 'Georeferenzierung' steht allerdings eher für den Prozeß der Zuweisung raumbezogener Referenzinformationen und Lageangaben und schließt den tatsächlichen Transformationsprozeß nicht mit ein (BILL 1999 A, S. 141)).

Im Gegensatz zu Vektordaten wird die Speicherung von Rasterdaten in ArcView über die Anzahl der Pixel in Zeilen- und Spaltenrichtung erreicht. Der Ursprung des Pixelkoordinatensystems befindet sich in der linken oberen Ecke des Rasterdatenbildes. Wird z.B. ein Kartenblatt der K10 in einen View geladen, so transformiert ArcView die Pixelkoordinaten in Soldner-Koordinaten. Um diese Transformation durchführen zu können, sucht das Programm beim Laden des Rasterdatensatzes nach entsprechenden Informationen. Einige Bildformate (z.B. ERDAS oder GeoTIFF) enthalten solche Informationen zur Geokodierung in ihrem Dateiheader. Liegen die benötigten Informationen nicht zusammen mit dem Rasterdatensatz vor, können sie auch in einer separaten Datei, dem sogenannten Wordfile, gespeichert werden.

ArcView liest die Geokodierungsinformationen eines Rasterdatenbildes nach der folgenden Prioritätenliste ein:

- Aus dem Worldfile (wenn es existiert)
- Aus dem Dateiheader (falls ein solcher vorhanden ist)
- Mit Hilfe der Zeilen- und Spalteninformation (Ähnlichkeitstransformation)

Mit einem Worldfile kann man das Programm also zwingen, die Informationen im Dateiheader zu übergehen.

Die für das Interessengebiet benötigten 15 Kartenblätter der K10 liegen ursprünglich im TIFF-Format vor und sind außerdem LZW-komprimiert (Lempel-Ziv und Welch). Ein Kartenblatt hat so eine Dateigröße von ca. 1 Megabyte (MB). Da LZW-komprimierte TIFF-Bilder in ArcView nur mit spezieller Zusatzsoftware dargestellt werden können, habe ich die Kartenblätter als unkomprimierte TIFF-Bilder abgespeichert, wobei sich eine Dateigröße von 19 MB pro Kartenblatt ergab. Aufgrund dieser sehr großen Datenmenge entschied ich mich, die Bilder ins BMP-Format (Windows Bitmap) zu konvertieren. Dieses unkomprimierte Dateiformat wird von ArcView ebenfalls unterstützt und verkleinert die

Bilder immerhin auf 9,5 MB pro Blatt. (Versuche mit dem sehr speicherfreundlichen JPEG-Format ergaben hinsichtlich Bildqualität (Auflösung) und Ladezeit keine zufriedenstellenden Ergebnisse).
Um sicher zu gehen, daß eine korrekte Geokodierung durchgeführt wird, habe ich für die 15 Bitmap-Dateien Worldfiles erstellt. Ein Worldfile besteht aus ASCII-Text und ist folgendermaßen aufgebaut (exemplarisch für Kartenblatt 303-3):

A	0.634920634920634920	Größe eines Pixels im RBS in X-Richtung [m]
D	0.000000000000000000	Rotationsterm für die Zeilen
B	0.000000000000000000	Rotationsterm für die Spalten
E	-0.634920634920634920	Größe eines Pixels im RBS in Y-Richtung [m]
C	20800.00	X-Koordinate des linken oberen Pixels im RBS [m]
F	7600.00	Y-Koordinate des linken oberen Pixels im RBS [m]

Die Parameter *A* und *E* sind für alle Kartenblätter gleich. Ein (mit 400 dpi gescanntes) digitales Kartenblatt hat eine Breite von 32 cm (5040 Pixel) und eine Höhe von 24 cm (3780 Pixel). Bei einem Kartenmaßstab von 1 : 10 000 entsprechen 100 m in der Natur 1 cm auf der Karte, ein Blatt deckt also die natürliche Fläche von 3200 x 2400 m ab. Die Größe eines Pixels im RBS in X- und Y-Richtung ergibt sich demnach zu $\frac{3200m}{5040} = \frac{2400m}{3780} = 0{,}634920\ldots$m. Dabei ist zu beachten, daß die Pixelgröße in Y-Richtung ein negatives Vorzeichen bekommt. Der Grund dafür sind die unterschiedlichen Ursprünge der Koordinatensysteme. Im Falle der Vektordaten befindet sich der Nullpunkt unten links, im Falle der Rasterdaten oben links (vgl. Abb. 7.15)
Die Parameter *D* und *B* stellen die Rotationsterme für Zeilen und Spalten dar und sollten nach Möglichkeit für alle Bilder gleich Null sein. Die Parameter *C* und *F* sind die Soldner-Koordinaten des linken oberen Pixels des Kartenblattes 303-3. Diese Werte differieren selbstverständlich von Kartenblatt zu Kartenblatt.
Der Übergang von Pixelkoordinaten in Soldner-Koordinaten geschieht über eine einfache 6-Parameter Affintransformation:

$X = Ax + By + C$
$Y = Dx + Ey + F$

mit X = berechnete X-Koordinate des Pixels im RBS
Y = berechnete Y-Koordinate des Pixels im RBS
x = Spaltennummer des Pixels im Rasterbild
y = Zeilennummer des Pixels im Rasterbild

Jedes mal, wenn eine Projektansicht vergrößert / verkleinert (’Zoom’) oder verschoben wird (’Pan’), wird eine solche Transformation berechnet. Durch die Geokodierung ist somit ein nahtloses Zusammenfügen der verschiedenen Kartenblätter innerhalb eines Views sowie eine klaffenfreie Überlagerung von Raster- und Vektordaten möglich (vgl. Abb. 7.15)

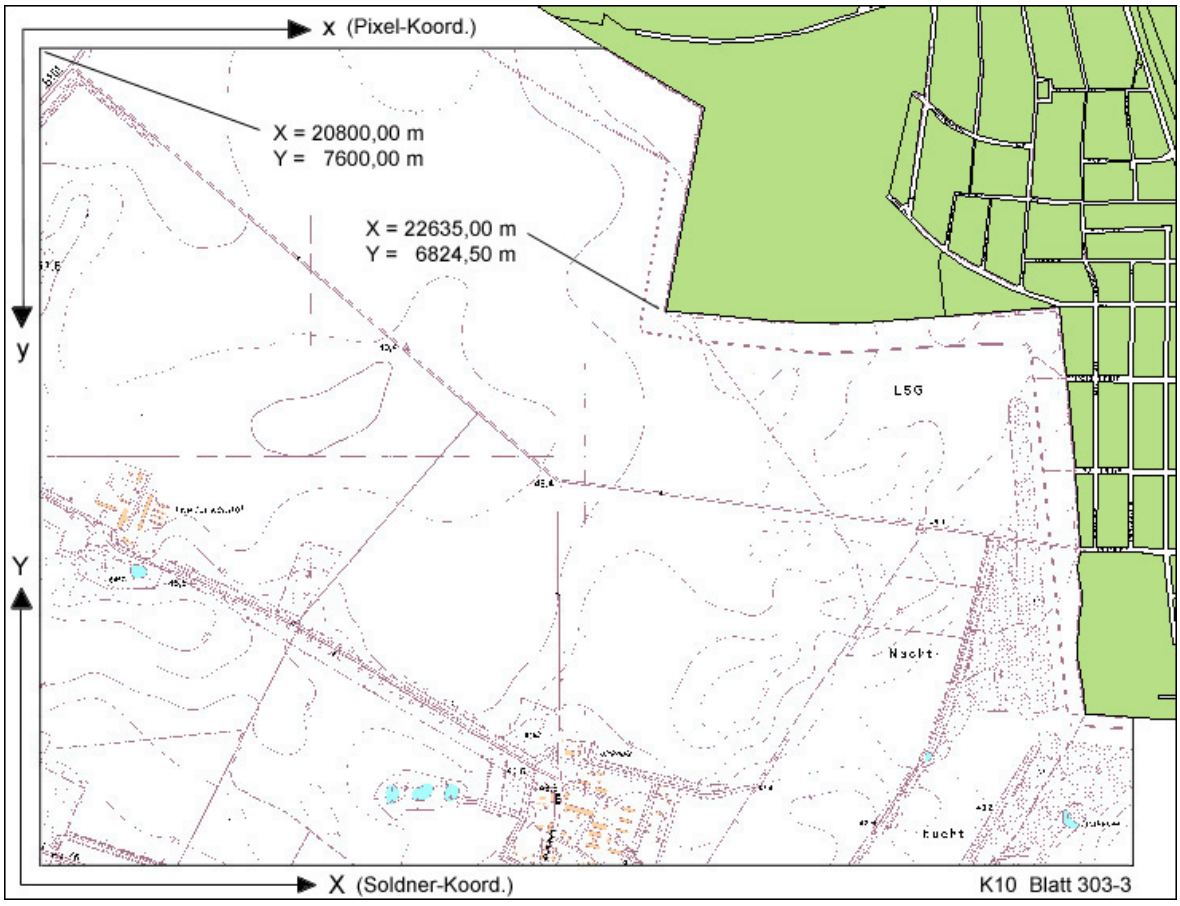


Abb. 7.15: Transformation von Pixel-Koordinaten in Soldner-Koordinaten

Damit ArcView einen Worldfile erkennt und richtig zuordnen kann, muß bei der Namensgebung folgende Konvention eingehalten werden: Der Name des Worldfiles muß identisch mit dem Namen des Rasterdatenbildes sein, ans Ende der Dateierweiterung wird lediglich ein ’w’ angehängt. Im Falle des Kartenblattes ’303-3.bmp’ erhält das dazugehörige Worldfile also den Namen ’303-3.bmpw’.

7.5 Adressen

Ein wesentliches Kriterium für eine effiziente Routenplanung ist das Adressen-Management. In diesem Zusammenhang ergeben sich eine Reihe von Fragen:

- Wie kann eine Liste mit Kundenadressen in ArcView eingelesen werden?
- Welche Konventionen bzgl. des Adressenformates müssen eingehalten werden?
- Wie können die Adressen graphisch dargestellt werden?
- Wie genau und wie zuverlässig ist die Adressenzuweisung?

In den folgenden Abschnitten soll diesen Problemen nachgegangen werden.

7.5.1 Adressenformat

ArcView bevorzugt für die Arbeit mit Tabellen das dBASE-Format (Attribute von Themen werden z.B. generell in dbf-Files gespeichert). Aus diesem Grunde habe ich die beiden Excel-Tabellen der Caritas-Station (Kundenlisten Mai und Juni) in dBASE-Dateien umgewandelt. Bei der Benennung und Reihenfolge der Datenfelder bestehen keine Beschränkungen, die von mir gewählte Anordnung sieht folgendermaßen aus:

Kunden Mai 2000

Kunde	Strasse	PLZ
CSS	Malteserstr. 171c	12277
Kunde1	Alt-Mariendorf 24	12107
Kunde2	Alt-Marienfelde 23	12277
Kunde3	Am Gemeindepark 40a	12249
Kunde4	Am Gemeindepark 42	12249
Kunde5	Am Horstenstein 28	12277
Kunde6	An den Klostergärten 31	12277
Kunde7	An den Klostergärten 37	12277
Kunde8	Bahnstr. 28	12277
Kunde9	Belßstr. 2	12277
Kunde10	Belßstr. 2	12277
Kunde11	Belßstr. 77	12249
Kunde12	Belßstr. 79a	12249
Kunde13	Belßstr. 79b	12249
Kunde14	Beselerstr. 14b	12249
Kunde15	Beyrodtstr. 32	12277
Kunde16	Beyrodtstr. 60	12277
Kunde17	Bleichertstr. 47	12277
Kunde18	Bornhagenweg 45	12309
Kunde19	Bruchwitzstr. 25	12247
Kunde20	Buckower Chaussee 21	12305

Kunden September 2000

Kunde	Alter	Strasse	PLZ
CSS	0	Malteserstr. 171c	12277
Kunde1	80	Alt-Lichtenrade 76	12309
Kunde2	92	Alt-Lichtenrade 100d	12309
Kunde3	78	Alt-Mariendorf 52	12107
Kunde4	65	Am Gemeindepark 26	12249
Kunde5	85	Am Gemeindepark 42	12249
Kunde6	94	An den Klostergärten 31	12277
Kunde7	87	An der Heilandsweide 5	12277
Kunde8	82	Bahnhofstr. 7a	12305
Kunde9	92	Bahnstr. 28	12277
Kunde10	78	Belßstr. 2	12277
Kunde11	77	Belßstr. 2	12277
Kunde12	91	Belßstr. 77	12249
Kunde13	78	Bernkastler Str. 26	12247
Kunde14	91	Beselerstr. 14b	12249
Kunde15	91	Beyrodtstr. 32	12277
Kunde16	72	Beyrodtstr. 60	12277
Kunde17	64	Bruchwitzstr. 13b	12247
Kunde18	86	Didostr. 2a	12109
Kunde19	84	Eisenacher Str. 71	12109
Kunde20	79	Emilienstr. 4	12277

Abb. 7.16: Kundenadressen der Caritas-Sozialstation (CSS) von Mai und September 2000

Die Sozialstation selbst erscheint in der Kundenliste, da ihre Adresse als Startpunkt für die Besuchstouren eingelesen und anschließend geokodiert werden soll (s. nächstes Kapitel).

7.5.2 Geokodierung von Adressen

Um einen räumlichen Bezug der Kundenadressen zum Thema Straßennetz herzustellen, steht in ArcView der sog. 'Geokodierungseditor' zur Verfügung. (Ähnlich wie im Falle der Rasterdaten (vgl. Kap. 7.4.1) spricht man auch bei der Adressen-Zuweisung von Geokodierung). Mit diesem Hilfsmittel sollen die Wohnorte der Caritas-Kunden möglichst zielsicher erkannt und durch ein Symbol im Straßennetz-Thema visualisiert werden.
Bevor die Arbeitsweise des Editors erläutert wird (s. nächste Seite), zunächst ein paar Worte zu ArcView GIS, Version 3.2.
In der Standardausführung des Programms sind leider nur amerikanische Adressformate implementiert, wodurch sich folgende Probleme ergeben:

- Die Hausnummern werden vor den Straßennamen gesetzt (z.B. 76 Franklin St.).
- Abkürzungen wie St., Rd., Drv., Pl., Blvd., Ave. etc. werden standardmäßig als Street, Road, Drive, Place, Boulevard und Avenue interpretiert.
- Für Postleitzahlen wird ein fünfstelliger ZIP Code (Postal Code) erwartet.
- Straßen, in denen die Hausnummern auf einer Seite 'hoch' laufen und auf der anderen Seite wieder 'herunter', scheint es in den USA nicht zu geben. In Berlin hingegen existieren eine Menge solcher Straßen. (Im weiteren Text wird in solchen Fällen von 'Hausnummern des Berliner Typs' gesprochen).

Ich habe eine E-Mail mit der Schilderung dieser Problematik an ESRI München geschickt und bekam auch umgehend Antwort. Die Geokodierung deutscher Adressformate ist möglich, wenn der Ordner 'geocode' im ArcView-Systemverzeichnis durch einen gleichnamigen Ordner ersetzt wird, der die deutschen Formate enthält. Die neue Adressen-Version benötigt 37 Dateien, welche mir zugeschickt wurden. Hier sind die obigen Probleme teilweise gelöst worden, allerdings kommen auch neue hinzu:

- Für die Angabe der Postleitzahl eines Straßenabschnittes kann im Format 'Straßen Adressbereiche links/rechts mit Zone' nur noch ein Feld gewählt werden (vgl. Abb. 7.17). Notwendig wären zwei Felder, da es durchaus vorkommen kann, daß der Wohnblock auf der linken Straßenseite eine andere Postleitzahl hat, als der Wohnblock auf der rechten. (Beim amerikanischen Äquivalent 'US Streets with Zone' sind zwei Felder für den ZIP Code vorhanden!)
- Für die Hausnummernbereiche 'links von / bis' und 'rechts von / bis' wird (wie in der amerikanischen Version) vorausgesetzt, daß die Straße auf einer Seite nur gerade und auf der anderen Seite nur ungerade Hausnummern hat. Adressen des Berliner Typs können nicht korrekt zugewiesen werden (mehr dazu in Kap. 7.5.3).

Nun jedoch zur Geokodierung von Adressen, welche in zwei Schritten vollzogen wird:

I a Auswahl des Referenzthemas, in welchem die Adressen abgebildet werden sollen. Im vorliegenden Fall ist dies das Thema Straßennetz.

I b Angabe der Attribute, die zur Geokodierung verwendet werden sollen. Dies sind der Straßenname, Hausnummernbereiche (von ... bis ...) auf linker und rechter Straßenseite sowie die Postleitzahl, wobei man sich für PLZ_links oder PLZ_rechts entscheiden muß (vgl. Abb. 7.17).

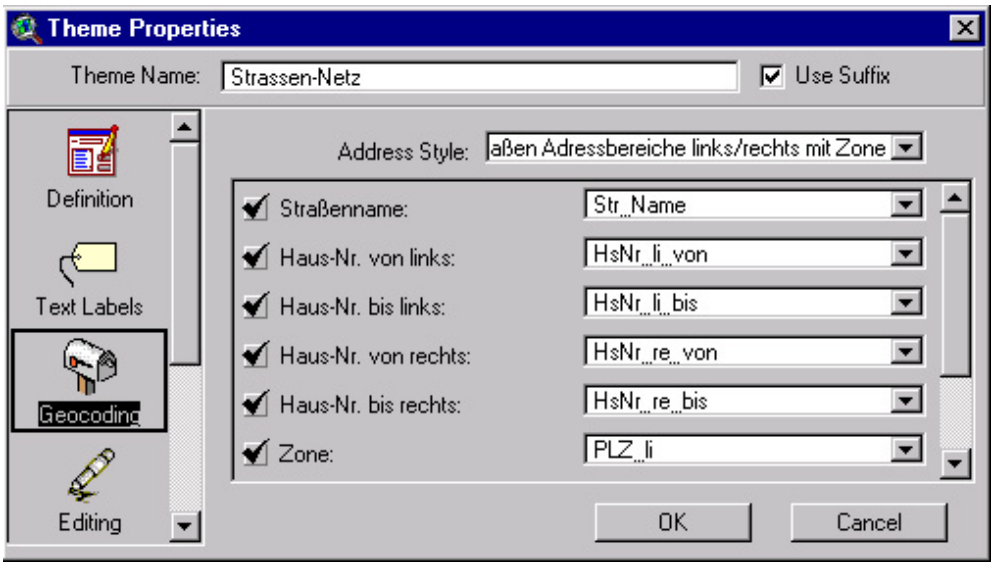


Abb. 7.17: Geokodierung des Themas Straßennetz

II a Einlesen der Tabelle, welche die Kundenadressen enthält. Im vorliegenden Fall ist das z.B. die Liste mit den Kunden von September 2000.

II b Angabe der Attribute, die zur Geokodierung verwendet werden sollen. Dies sind die Kundennamen, Straße (mit Hausnummer) und Postleitzahl (vgl. Abb. 7.18).

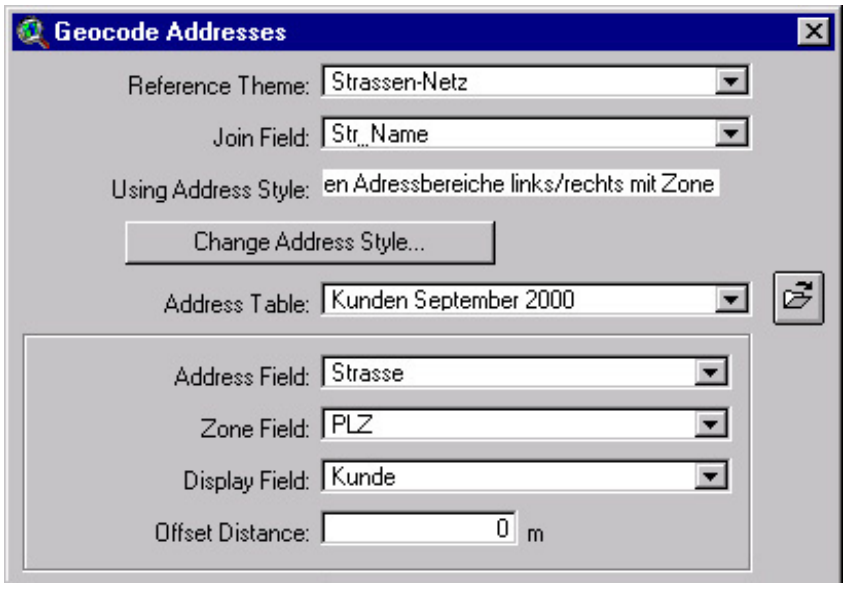


Abb. 7.18: Geokodierung von Kundenadressen

7.5.3 Adressen - Matching

Wenn die beiden obigen Schritte durchgeführt und zu den eingelesenen Adressen die passenden Orte gefunden wurden, werden im Straßennetz (genauer gesagt in einem neu erzeugten Thema) automatisch graphische Symbole eingefügt (vgl. Abb. 7.19).

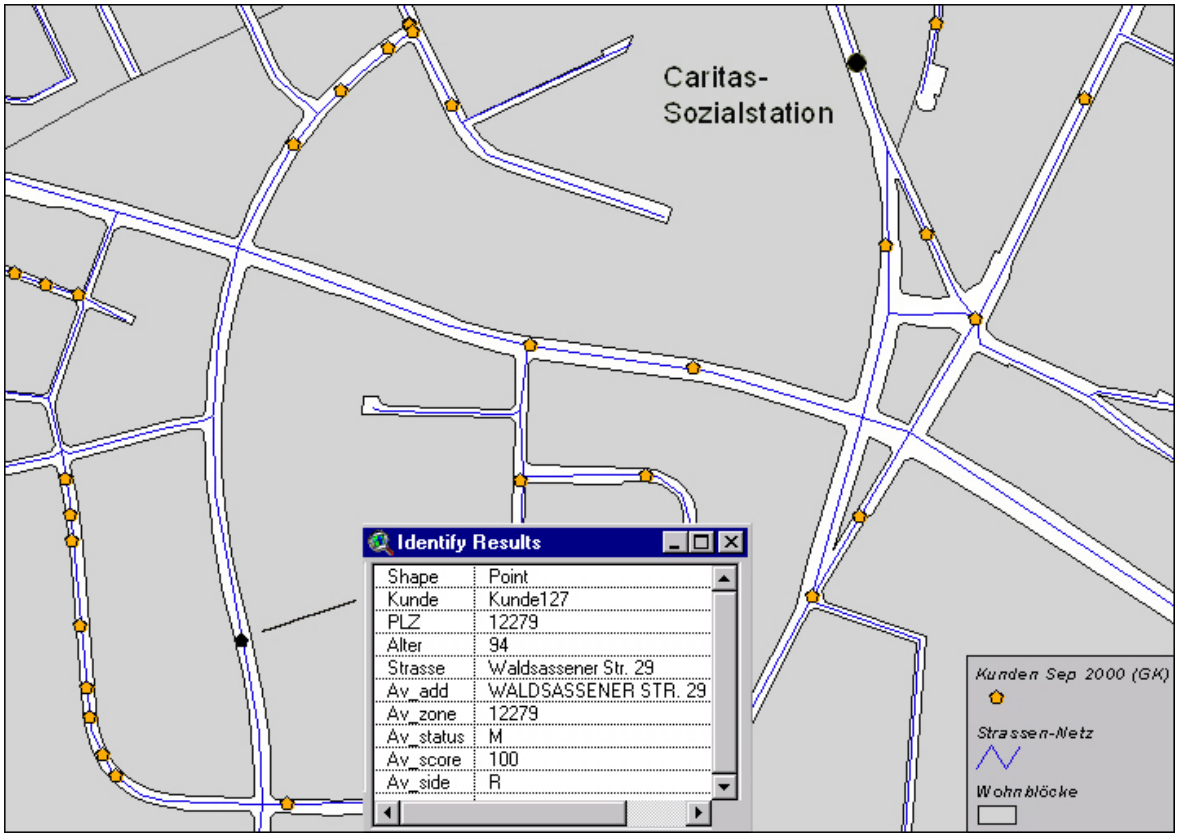


Abb. 7.19: Zugewiesene Kundenadressen im Thema Straßennetz
(Av = ArcView, M = Matched, Score = 100% Übereinstimmung, R = rechte Straßenseite)

Der nächste Schritt ist die Überprüfung der zugewiesenen Orte, sie kann mit Hilfe des Kartenhintergrundes der K10 leicht durchgeführt werden.

Nach der Geokodierung der Kunden-Adresslisten von Mai und September stellte ich fest, daß sich einige zugewiesene Symbole nicht an den korrekten Orten befanden. In den schlimmsten Fällen wurde ein Symbol, welches an den Anfang eines Straßensegmentes gehört hätte, genau an das andere Ende der Straße gesetzt. Bei langen Straßenzügen wurden Adressen so um mehrere hundert Meter falsch zugeordnet, was nicht mehr akzeptabel ist. Andere Adressen konnten nur grob falsch (in eine andere Straße) oder überhaupt nicht zugeordnet werden.

Bei dem Zuweisungsvorgang, dem sog. ’Matching’ (engl. abgleichen, übereinstimmen), werden die Attribute der Straßendatenbank mit den Attributen der Kundenliste verglichen. Die Ursache der Fehler liegt also wahrscheinlich im Datenmaterial der beiden Tabellen.

Ich habe einige Tests durchgeführt und konnte folgende Fehlerquellen (in der Straßendatenbank oder den Kundenlisten) lokalisieren:

1. Falsch geschriebene und / oder nicht existierende Straßennamen
2. Falsche Hausnummernbereiche und / oder Postleitzahlen
3. Fehlende Hausnummernbereiche und / oder Postleitzahlen
4. Die Richtung der Hausnummerfolge stimmt nicht mit der Digitalisierrichtung der Straßensegmente überein
5. Straßen mit Hausnummern des Berliner Typs

In den Fällen 1, 2 und 3 lassen sich die Fehler ohne größere Probleme beheben. Änderungen in der Straßendatenbank sind zwar zeitaufwendig, perfektionieren diese durch das Ausmerzen der Fehler jedoch für spätere Anwendungen.

Unangenehmer ist der 4. Fall: allen Liniensegmenten des Straßenetzes werden beim Vorgang der Digitalisierung Richtungen mitgegeben und jede Straße erhält so eine linke und eine rechte Straßenseite. Damit man weiß, wo links und rechts ist, muß man feststellen, ob die Linie (gerichtete Kante) von Knoten A nach Knoten B oder andersherum digitalisiert (gezeichnet) wurde. In ArcView gibt es einen einfachen Weg, dies herauszufinden: man vergibt für das entsprechende Thema (hier das Straßennetz) eine Liniensymbolik mit Pfeilen, welche die Digitalisierrichtung dann anzeigt (vgl. Abb. 7.20).

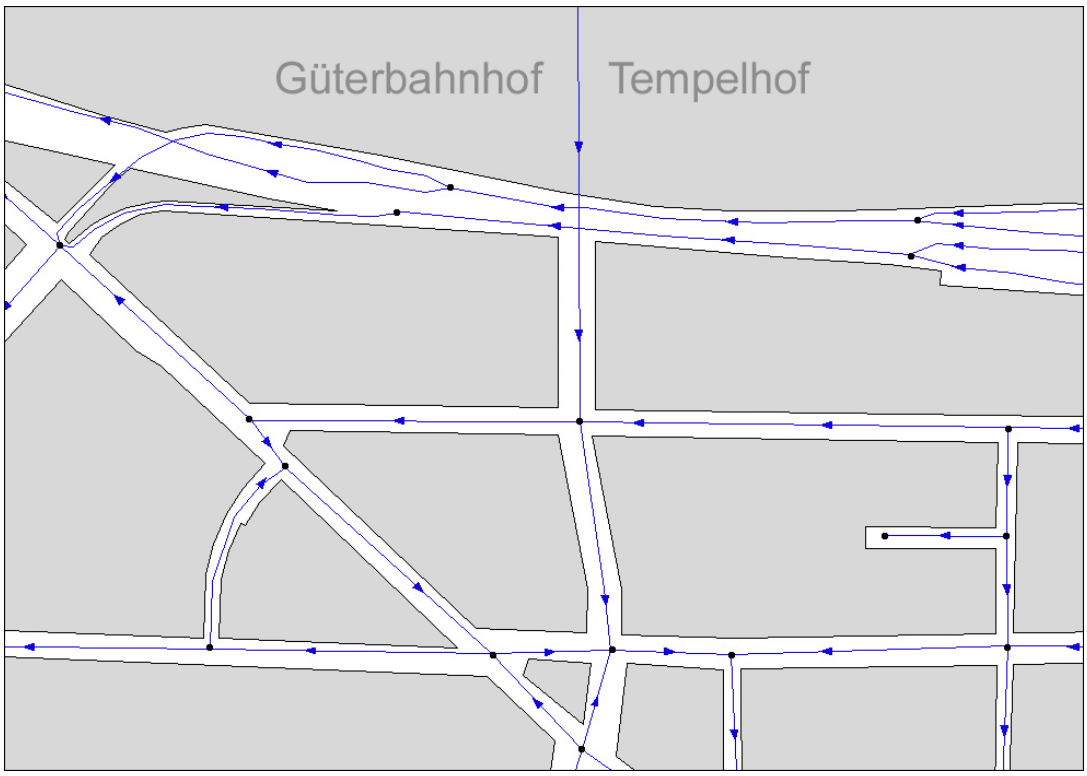


Abb. 7.20: Digitalisierrichtungen von Straßensegmenten

Wenn die Laufrichtung der Hausnummern in der Realität nicht mit der Digitalisierrichtung des Liniensegmentes übereinstimmt, weist der Geokodierungseditor die Kundenadressen genau in der falschen Reihenfolge zu (vgl. Abb. 7.21 und 7.22).

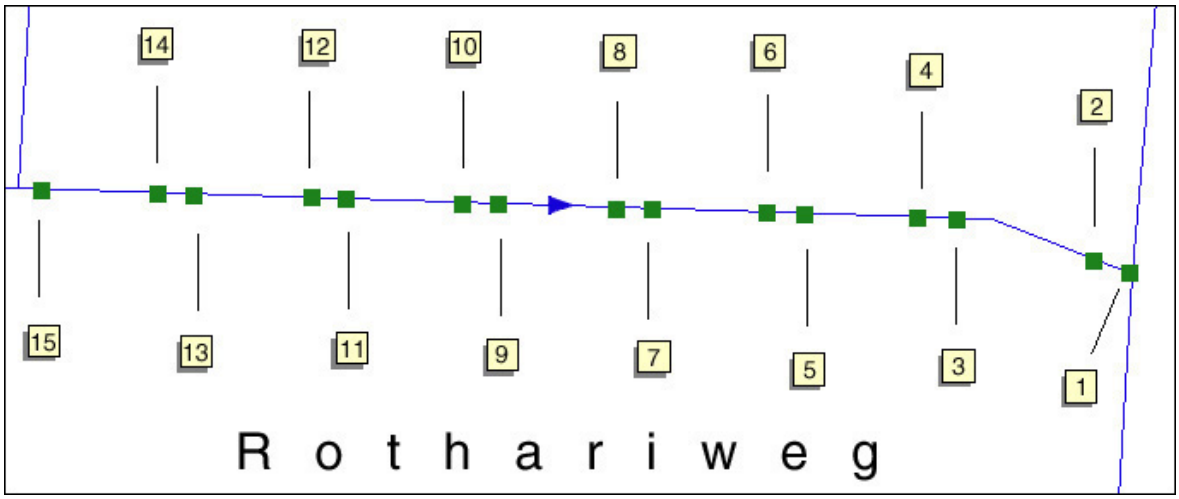


Abb. 7.21: Hausnummern auf linker und rechter Straßenseite

Attributes of Strassen-Netz

Shape	Str_Abschn	Str_Name	HsNr_li_von	HsNr_li_bis	HsNr_re_von	HsNr_re_bis	LR_li	LR_re	PLZ_li	PLZ_re
PolyLine	37960030	Rothariweg	001	015	002	014	2	1	12103	12103

Abb. 7.22: Falscher Eintrag in der Straßendatenbank

Um den Fehler zu beheben, müssen die Werte der Felder ’HsNr_li_von’ und ’HsNr_re_bis’ sowie die Werte der Felder ’HsNr_li_bis’ und ’HsNr_re_von’ getauscht werden (vgl. Abb. 7.23). (Die Felder ’LR_li’ und ’LR_re’ werden auf der nächsten Seite erläutert).

Attributes of Strassen-Netz

Shape	Str_Abschn	Str_Name	HsNr_li_von	HsNr_li_bis	HsNr_re_von	HsNr_re_bis	LR_li	LR_re	PLZ_li	PLZ_re
PolyLine	37960030	Rothariweg	014	002	015	001	1	2	12103	12103

Abb. 7.23: Berichtigter Eintrag in der Straßendatenbank

Nach der Vertauschung werden die Adressen in der richtigen Reihenfolge geokodiert. Die Zuweisung der Adressen geschieht dabei proportional zur Länge des Liniensegmentes. Im obigen Beispiel ist der Straßenabschnitt des Rothariweges 148 m lang und enthält 15 verschiedene Hausnummern. Es wird also ca. alle 20 m ein Symbol in das neu erzeugte Thema eingefügt, wobei Hausnummern der linken und rechten Seite leicht versetzt dargestellt werden (vgl. Abb. 7.21).

Der 5. Fall (Straßen des Berliner Typs) läßt sich am schwierigsten handhaben. Zur Übersicht seien hier noch einmal die drei verschiedenen Sorten von Hausnummern aufgelistet, die bei der Routenplanung in Betracht kommen:

(1) Es existieren nur gerade Hausnummern auf einer Straßenseite.

(2) Es existieren nur ungerade Hausnummern auf einer Straßenseite.

(3) Die Hausnummern sind auf beiden Seiten fortlaufend (Berliner Typ).

In der Straßendatenbank ist die Laufrichtung der Hausnummern mit LR_li und LR_re (linke und rechte Straßenseite) gekennzeichnet. Für alle Straßen (bis auf sehr wenige Ausnahmen) ergeben sich demnach drei mögliche Laufrichtungs-Kombinationen:

- LR_li = 1, LR_re = 2 : linke Seite gerade, rechte Seite ungerade Hausnummern
- LR_li = 2, LR_re = 1 : linke Seite ungerade, rechte Seite gerade Hausnummern
- LR_li = 3, LR_re = 3 : linke und rechte Seite fortlaufende (Berliner) Hausnummern

Anhand eines Beispiels soll die Problematik bei der Geokodierung von Hausnummern des Berliner Typs aufgezeigt werden (vgl. Abb. 7.24)

Attributes of Strassen-Netz

Shape	Str_Abschn	Str_Name	HsNr_li_von	HsNr_li_bis	HsNr_re_von	HsNr_re_bis	LR_li	LR_re	PLZ_li	PLZ_re
PolyLine	6830010	Bosestr.	022	023	027	024	3	3	12103	12103
PolyLine	6830020	Bosestr.	021	021	029	028	3	3	12103	12103
PolyLine	6830030	Bosestr.	019	020	031B	030	3	3	12103	12103
PolyLine	6830040	Bosestr.	032	036	018	013	3	3	12103	12103
PolyLine	6830050	Bosestr.	037	043	012	006	3	3	12103	12103
PolyLine	6830060	Bosestr.	044	047	005	001	3	3	12103	12103

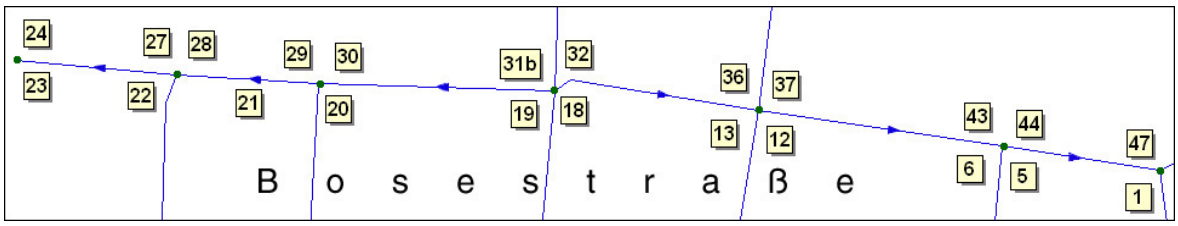


Abb. 7.24: Hausnummern des Berliner Typs

Das Wechseln der Digitalisierrichtung in der Mitte der Bosestraße läßt sich mit Hilfe einer Vertauschungsoperation (vgl. S. 93) noch kompensieren. Problematisch sind die fortlaufenden Hausnummern auf beiden Straßenseiten. So werden z.B. im östlichsten Segment die Nummern 1, 3 und 5 auf der linken und die Nummern 44 und 46 auf der rechten Straßenseite korrekt zugewiesen. Die Nummern 2 und 4 bzw. 45 und 47 werden überhaupt nicht gefunden, da der Geokodierungseditor rechts nur ungerade und links nur gerade Hausnummern erwartet. Man muß also mit einer Ausfallquote von 50 % bei Straßen des Berliner Typs rechnen. Die Tatsache, daß von den 2384 Straßensegmenten des Interessengebietes 556 (23 %) eine derartige Hausnummernverteilung besitzen, zeigt, daß

das Problem nicht vernachlässigt werden darf. (Ob solche Hausnummern nur typisch für Berlin sind oder auch in anderen Städten Verbreitung gefunden haben, kann ich nicht sagen).
Abhilfe könnte nur eine Neuprogrammierung bzw. Erweiterung des Editors bewirken. Ein möglicher Ansatz wäre die Nutzung der Laufrichtungs-Attribute, welche ursprünglich als zusätzliche Orientierungshilfe in die Straßendatenbank eingefügt wurden. Beim Einlesen des Wertes LR = 3 müßte der Editor auf die neu zu implementierende fortlaufende Nummerierung des Berliner Typs umschalten.

Mit Hilfe des Geokodierungseditors lassen sich übrigens nicht nur komplette Kundenlisten im Ganzen einlesen, sondern auch einzelne Adressen von Hand suchen (der Adressentext kann interaktiv eingegeben werden, vgl. Abb. 7.25 oben). Wichtig für beide Methoden sind die in den Einstellungen des Geokodierungseditors modifizierbaren Parameter ’Spelling Sensitivity’, ’Minimum Match Score’ und ’Minimum Score’ (vgl. Abb. 7.25 unten). Wird die Buchstabierungs-Sensibilität z.B. auf 100 gesetzt, werden nur vollkommen richtige Adressen (Straße, Haus Nr. und PLZ) zugewiesen. Setzt man den Wert weiter herunter und die beiden Score-Parameter auf Null, werden für das Matching sehr viele Kandidaten in Betracht gezogen. Abbildung 7.25 zeigt die Standardeinstellung, die in den meisten Fällen zu guten Ergebnissen führt. Bei Adressen, die entweder gar nicht oder nur mit einem niedrigen Score zugewiesen wurden, empfiehlt sich ein ’Rematching’ mit veränderten Parametern (mehr dazu auf der nächsten Seite).

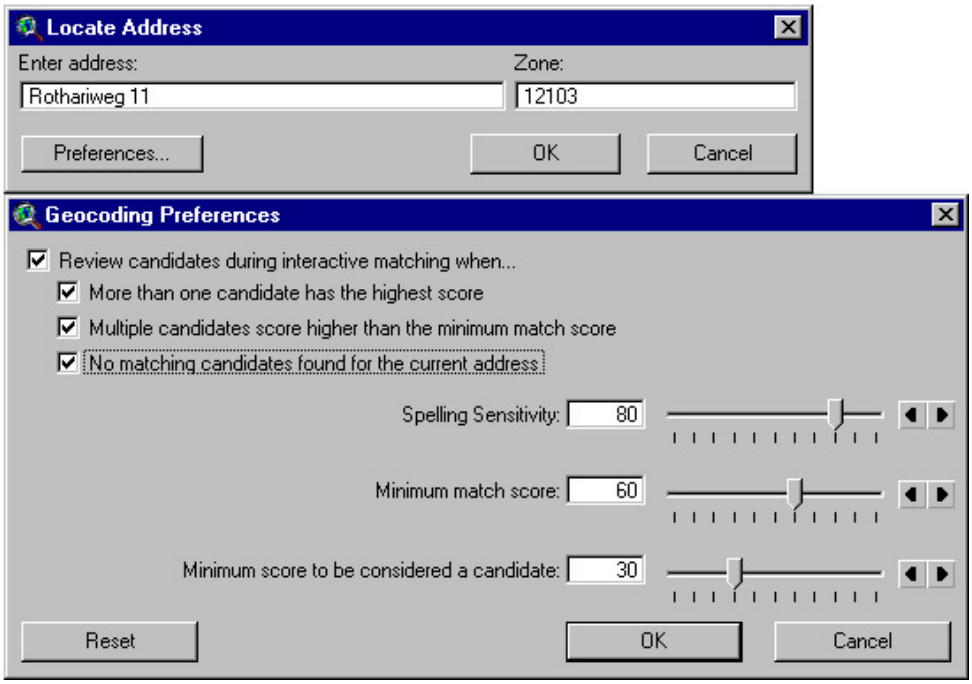


Abb. 7.25: Adressen-Eingabemaske (oben) mit Standardwerten zur Geokodierung (unten)

7.5.4 Adressen - Rematching

Abbildung 7.26 (oben) zeigt das Ergebnis des Rematchings für die September-Kundenliste der Caritas-Station. Bei einer Geokodierung mit den Werten 100, 60, 30 (vgl. S. 95) erhielten 97 % der Adressen einen guten Wert, eine Adresse konnte nur unbefriedigend und vier überhaupt nicht zugewiesen werden. Mit Hilfe des interaktiven Rematching-Prozesses können die Ursachen der Fehl- oder Nichtzuweisung aufgespürt werden. Dabei hat man die Möglichkeit entweder alle Adressen, nur Kandidaten der Gruppen 'Partial Match' oder 'No Match' oder auch von Hand selektierte Adressen nochmals neu zu geokodieren. Abbildung 7.26 (unten) zeigt die Überprüfung der Adresse, die beim ersten Durchgang nur als 'Partial' Match eingestuft wurde. Mit den Parametern 100, 60, 30 wurden nur Kandidaten mit einem 'Score' von 46 gefunden. Ändert man die Parameter auf 70, 60, 30 werden auch Kandidaten mit einem 'Score' von Hundert angezeigt, wobei der oberste die korrekte Adresse liefert. Der Grund für das Verfehlen dieses Kandidaten liegt in der Postleitzahl. Wie bereits angedeutet enthält das benutzte deutsche Adressenformat 'Straßen Adressbereiche links / rechts mit Zone' nur noch ein PLZ-Feld. Man muß sich bei der Geokodierung für 'PLZ_li' oder 'PLZ_re' als Referenzfeld der Straßendatenbank entscheiden (vgl. Abb. 7.17), wobei die Wahl wie im vorliegenden Fall der Bahnhofstraße dann genau die falsche sein kann.

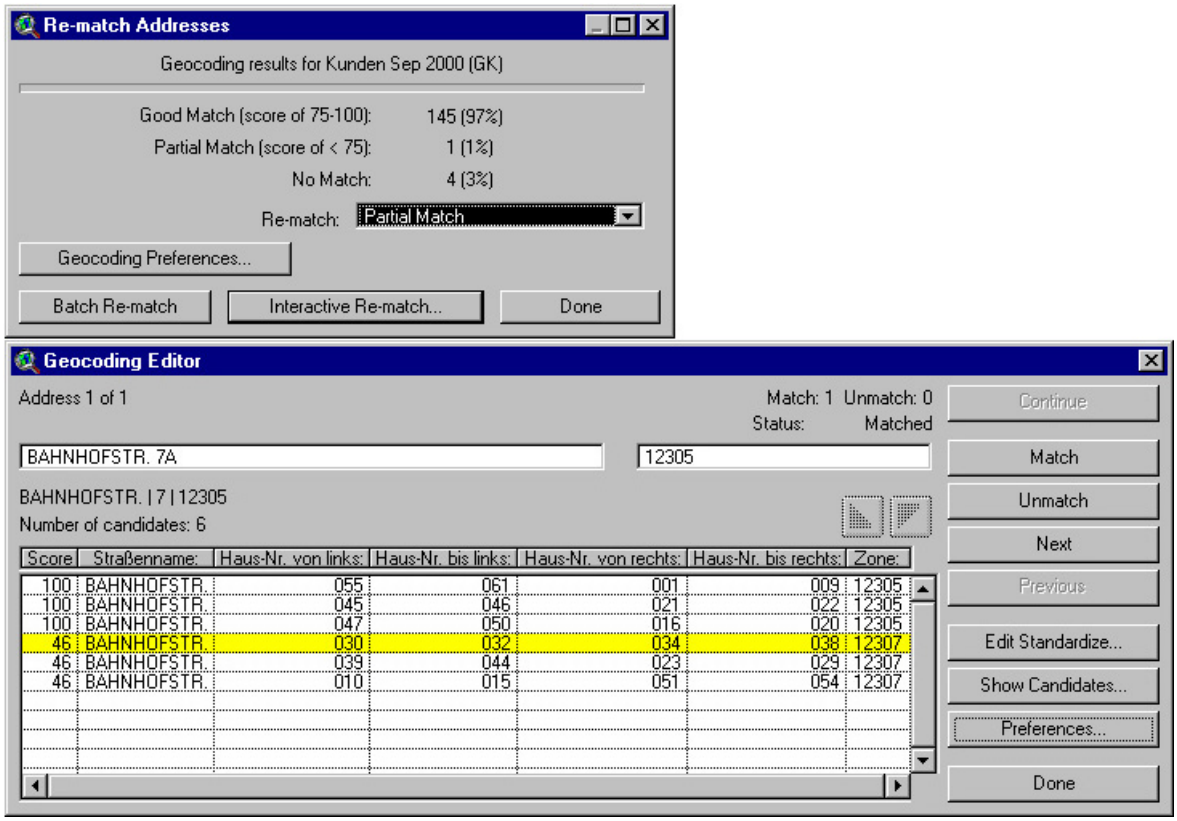


Abb. 7.26: Rematching von Adressen (oben), Ergebnis im Geokodierungseditor (unten)

Die linke Seite des Straßenabschnittes mit den Nummern 55 bis 61 gehört zum PLZ-Bereich 12307, die rechte Seite mit den Nummern 1 bis 9 dagegen zum PLZ-Bereich 12305. Als Referenzfeld war 'PLZ_re' gewählt, wodurch der korrekte Abgleich der Kundenadresse mit dem Eintrag in der Straßendatenbank verhindert wurde. Den Vorrang erhielt ein Kandidat mit „passender" Postleitzahl, welche durch die Einstellung 'Spelling Sensitivity = 100' höher eingestuft wurde, als das richtige Hausnummernintervall.
Solche Fälle ließen sich durch zwei Felder für die Postleitzahl vermeiden. Offenbar wurde bei der deutschen Adaption des Geokodierungseditors diese Tatsache einfach übersehen. (Das amerikanische Format 'US Streets with Zone' sieht eine solche Möglichkeit vor!)

Mit Hilfe des interaktiven Adressen-Rematching können Fehlzuweisungen schließlich von Hand berichtigt werden. Man wählt aus der Liste den passenden Kandidaten aus und gibt ihn zur Geokodierung frei. Auch Adressen des Berliner Typs, die im automatischen Batch-Lauf nicht gefunden wurden, können auf diese Weise zugewiesen werden.
So lassen sich nach und nach alle Fehler beseitigen. Am Ende des Optimierungsprozesses ist es nicht verwunderlich, wenn alle Adressen der Kundenliste mit einem Score von 100 unter den strengsten Bedingungen 'gematcht' werden.

7.5.5 Zwei verschiedene Gebiete

Die in den Abbildungen 7.22, 7.23 und 7.24 gezeigten Ausschnitte der Straßendatenbank gehören zum Thema „Testgebiet Tempelhof-Nord" (mehr dazu in Kapitel 8.2). Alle Straßensegmente dieses Ausschnittes werden zu beiden Seiten jeweils von nur einem statistischen Block umsäumt. Das bedeutet, daß es jeweils nur zwei Hausnummernabschnitte und zwei Postleitzahlen (links und rechts) gibt.
Dies gilt nicht für das gesamte Interessengebiet „Bezirk Tempelhof und Ortsteil Lankwitz". In diesem Thema existieren Straßensegmente, die von bis zu drei Blöcken auf linker und rechter Seite umgeben werden. Hier können also maximal sechs Hausnummernabschnitte und Postleitzahlen vorkommen, was sich in der Datenbank durch zusätzliche Felder bemerkbar macht (vgl. Abb. 7.27).

Attributes of Strassen-Netz

Shape	Str_Abschn	Str_Name	HsNr1von	HsNr1bis	LR1	PLZ1	HsNr2von	HsNr2bis	LR2	PLZ2	HsNr3von	HsNr3bis
PolyLine	7830030	Buckower Chaussee	025	058	3	12277	104	134	3	12277	135	135

LR3	PLZ3	HsNr4von	HsNr4bis	LR4	PLZ4	HsNr5von	HsNr5bis	LR5	PLZ5	HsNr6von	HsNr6bis	LR6	PLZ6
3	12277	060	069	3	12277	99	100	3	12277	102A	102B	3	12277

Abb. 7.27: Straßennetz-Attribute des Views „Bezirk Tempelhof und Ortsteil Lankwitz"

Die Geokodierung der Adressen erfolgt analog zu den obigen Ausführungen über die Felder ’HsNr1von’, ’HsNr1bis’, ’HsNr2von’, ’HsNr2bis’ und wahlweise ’PLZ1’ oder ’PLZ2’.

Dies stellt natürlich keine ideale Lösung dar, weil im Falle von mehreren Abschnitten nur die ersten beiden bei der Adresszuweisung berücksichtigt werden. Allerdings ist diese Tatsache im vorliegenden Projekt zu verschmerzen. Von den 2384 Straßensegmenten des gesamten Interessengebietes weisen nur 91 (3,8 %) eine Umgebung von mehr als zwei Blöcken auf, in 96, 2 % aller Fälle führt eine Geokodierung nach obigem Muster zu vollständigen Ergebnissen. Zudem kommt eine Umgebung mit mehr als zwei Blöcken in den meisten Fällen durch kleine ungenutzte Flächen zustande, die keine Wohngebiete darstellen, aber trotzdem als statistische Blöcke geführt werden.

Um dieses Problem endgültig zu umgehen, müßten die 91 betroffenen Segmente neu digitalisiert und in kleinere Stücke aufgeteilt werden, so daß auf linker und rechter Straßenseite jeweils nur ein Block angrenzt.

8. Routenoptimierung

In diesem Kapitel werden alle technischen Details erläutert, die für die Routenoptimierung in ArcView Bedeutung haben. Um realistische Touren erzeugen zu können, müssen in erster Linie Verkehrsregeln und verkehrstechnische Besonderheiten des Tempelhofer Straßennetzes berücksichtigt werden. Dies können zum Beispiel sein:

- Einbahnstraßen
- Kreisverkehr
- Für PKW gesperrte Straßen
- Abbiegeverbote
- Sackgassen
- Autobahnen (Ein- und Ausfahrten, Über- und Unterführungen)

Die Routenoptimierungsapplikation betrachtet die Topologie des Straßennetzes a priori als einen zusammenhängenden, ungerichteten Graphen mit gleichberechtigten, unbewerteten Kanten. Damit die oben aufgeführten Punkte erkannt und in der Routenplanung berücksichtigt werden, müssen Veränderungen in der Straßendatenbank (*) vorgenommen werden. Neben einer Optimierung der Wegstrecke soll auch eine Optimierung nach Fahrtzeit möglich sein. Hierfür sind ebenfalls entsprechende Vorbereitungen zu treffen.

Als Nachschlagewerk für die Verkehrsregeln und verkehrstechnischen Besonderheiten des Testgebietes habe ich den Autofahreratlas des STADTINFO-VERLAGES (1998) benutzt. In dem Atlas sind Standorte von Ampeln, die wichtigsten Verkehrsschilder (Vorfahrtsregeln, Abbiegeverbote etc.) und alle weiteren oben aufgeführten Punkte enthalten.

(*) **Anmerkung:** Der allgemeine Begriff ’Straßendatenbank’ charakterisiert eigentlich die Gesamtheit des Datenbestandes bezüglich des Straßennetzes. Die Informationen, die die Topologie und Struktur des Straßennetzes beschreiben, sind allerdings auf etliche verschiedene Ordner, Dateien und Dateitypen verteilt, welche miteinander korrespondieren. (Es existieren allein neun verschiedene Dateien mit dem Namen ’strassen.xxx’, die beiden wichtigsten sind ’strassen.shp’ und ’strassen.dbf ’. Der Zugriff auf die einzelnen Informationsbestandteile wird in der Projektdatei (*.apr) verwaltet).
Wenn im nachfolgenden Text von der ’Straßendatenbank’ geprochen wird, ist in der Regel die Attribut-Tabelle des Themas Straßennetz gemeint. Diese bezieht ihre Informationen direkt aus der Datei ’strassen.dbf ’.

8.1 Graphische Benutzeroberfläche

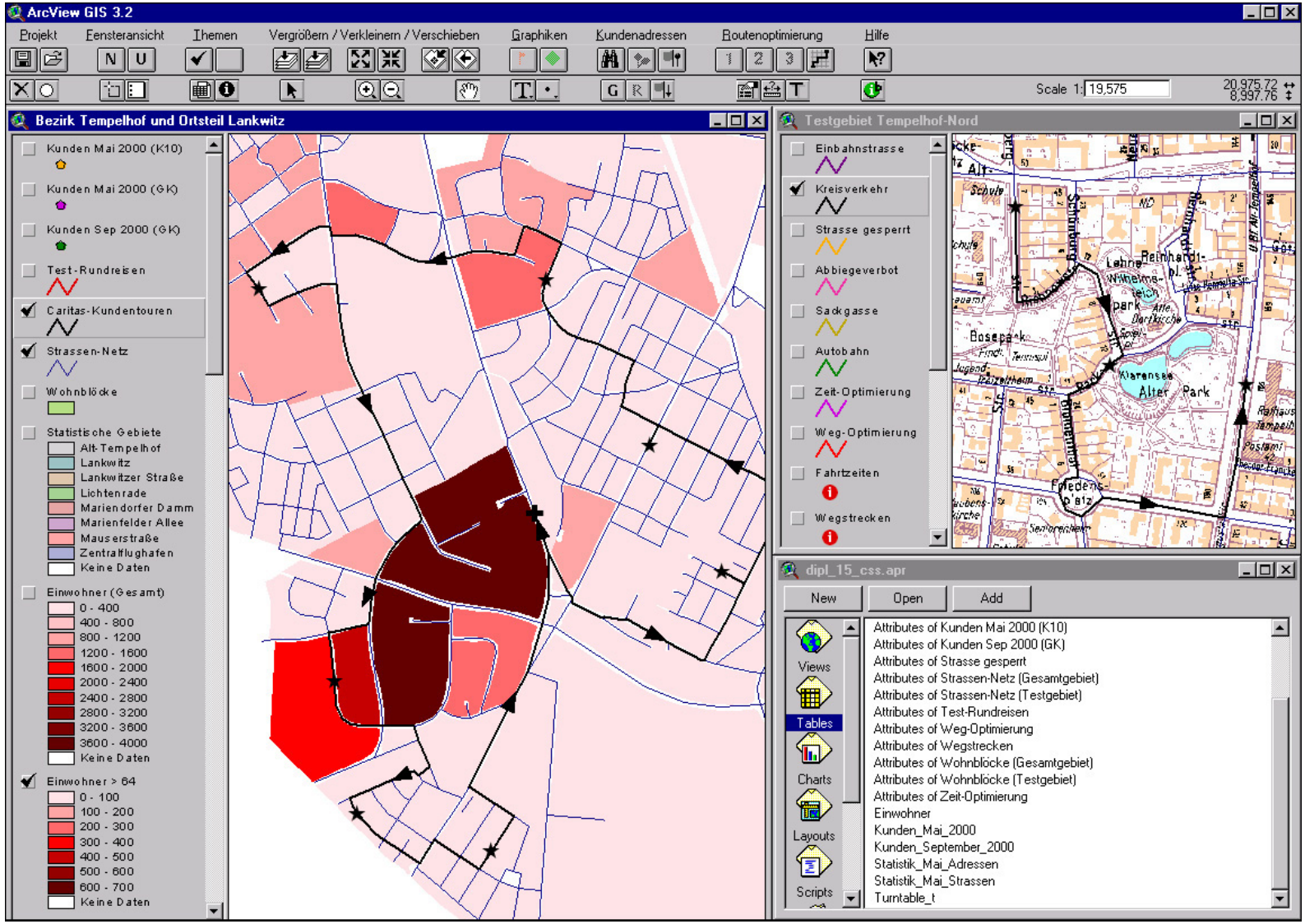


Abbildung 8.1 zeigt die für die Routenoptimierung zurechtgeschnittene Benutzeroberfläche von ArcView GIS 3.2.
Im Sinne einer möglichst einfachen und anwender-freundlichen Bedienung habe ich versucht, nur wirklich notwendige Menüpunkte, Schalt-flächen und Werkzeuge zu implementieren.

8.2 Testgebiet Tempelhof-Nord

Das Interessengebiet der Caritas-Station (Bezirk Tempelhof und Ortsteil Lankwitz) umfaßt 2384 Straßensegmente und 1046 statistische Blöcke. Eine Korrektur des gesamten Datenbestandes bezüglich Hausnummernabschnitten (vgl. Kap. 7.5.3) und Verkehrsregeln hätte eine wochenlange Sisyphusarbeit bedeutet. Aus diesem Grunde habe ich aus der Gesamtfläche ein kleineres Stück, das sogenannte „Testgebiet Tempelhof-Nord“, ausgeschnitten und als separates, neues Thema in das Projekt eingefügt. Alle zu diesem Thema gehörenden Dateien erhalten den Zusatz ’_t’ im Dateinamen (strassen_t.shp, strassen_t.dbf usw.). Dadurch werden Gesamt- und Testgebiet klar voneinander getrennt und Dateikonflikte vermieden.

Das neue Areal umfaßt 178 Straßensegmente und 80 statistische Blöcke (vgl. Abb. 8.2). In diesem Gebiet sollen alle notwendigen Anpassungen durchgeführt werden, um der tatsächlichen Wohn- und Verkehrssituation vor Ort möglichst nahe zu kommen.

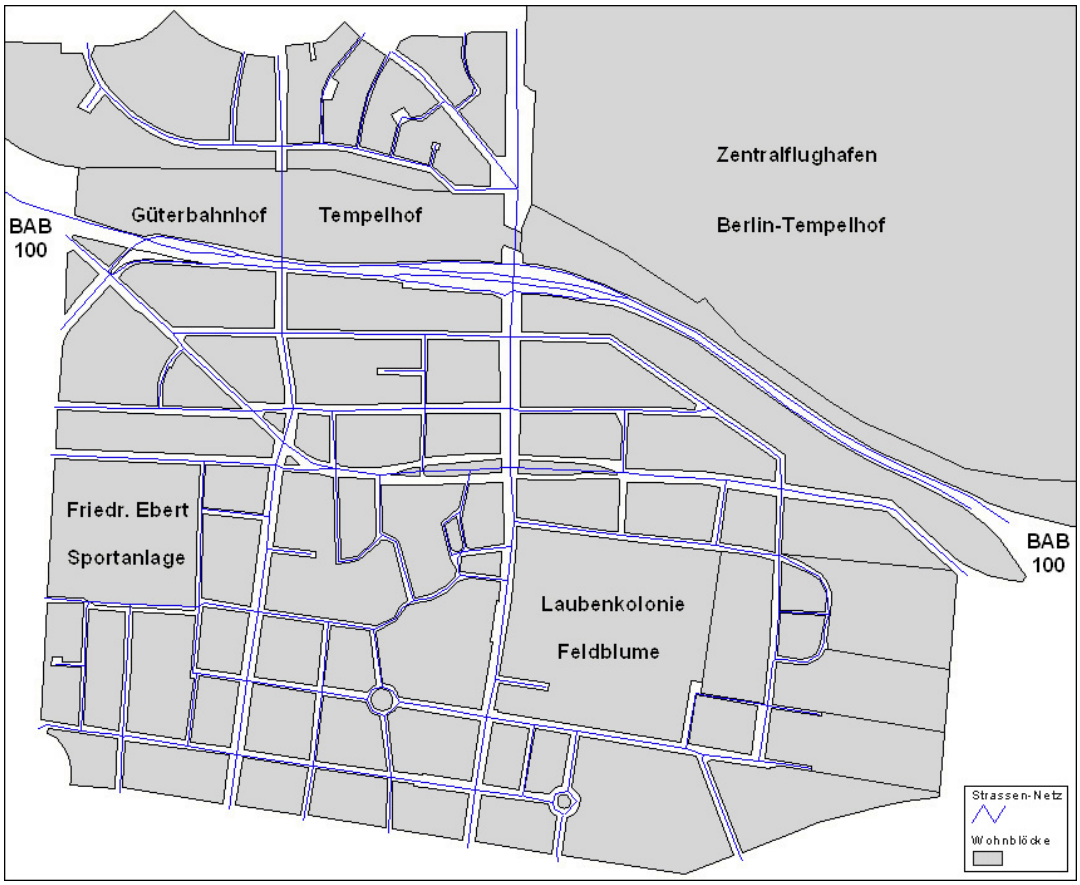


Abb. 8.2: Testgebiet Tempelhof-Nord

Um eine korrekte flächendeckende Adressenzuweisung für das Testgebiet zu gewährleisten, habe ich alle Hausnummernabschnitte überprüft und notfalls die in Kapitel 7.5.3 beschriebenen Vertauschungsoperationen vorgenommen.

8.2.1 Knoten und Kanten

Für die Berücksichtigung der verkehrstechnischen Besonderheiten müssen einzelne Straßenabschnitte und Kreuzungen mit bestimmten Attributen versehen werden (mehr dazu in Kapitel 8.3). Für die entsprechende Datenmodellierung ist eine vollständige Erfassung aller Knoten und Kanten des Testgebietes erforderlich (vgl. Abb. 8.3).

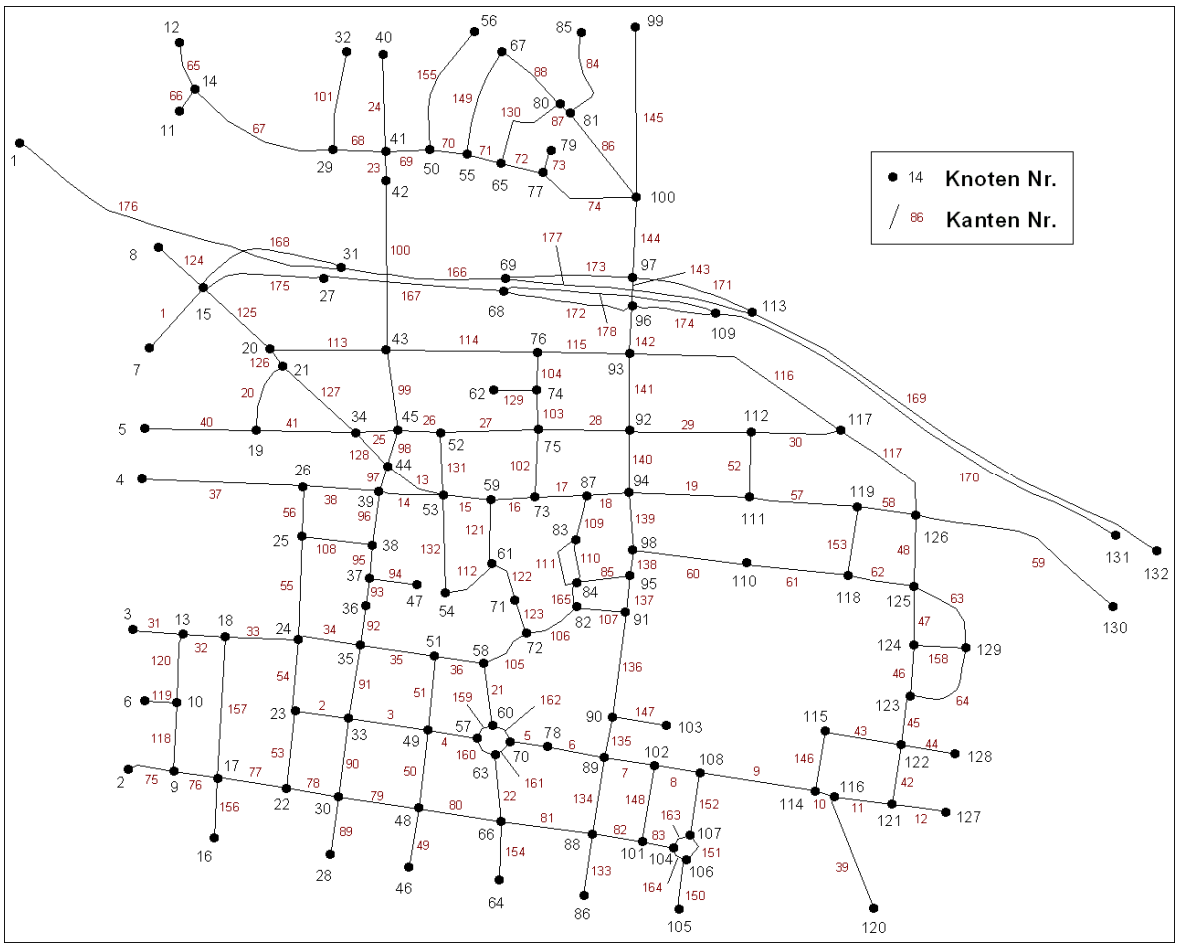
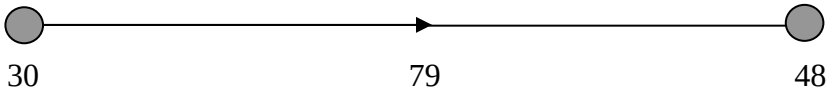


Abb. 8.3: Knoten- und Kantennummern des Testgebietes

Alle Kanten bekommen dem Straßenverlauf folgende Nummern und werden in der Datenbank im Feld ’Edge#’ gespeichert (# = Number). Zu jeder Kante gehört entsprechend der Digitalisierrichtung ein Start- und ein Endknoten, welche in den Feldern ’Fjunction’ und ’Tjunction’ gespeichert werden (engl. junction = Kreuzung, F = From, T = To).



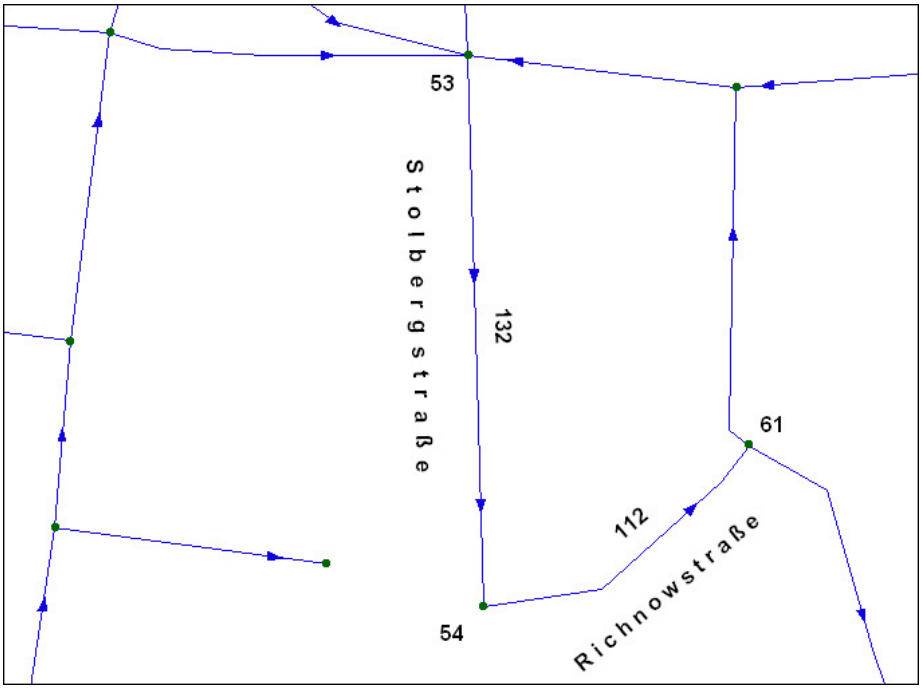
Kante 79 von Startknoten 30 nach Endknoten 48 hat also den folgenden Eintrag in der Straßendatenbank: Edge# = 79, Fjunction = 30, Tjunction = 48. Die Nummerierung der Knoten erfolgt straßenunabhängig mit fortlaufenden Nummern von West nach Ost.

8.3 Verkehrsregeln und verkehrstechnische Besonderheiten

In diesem Kapitel wird die datentechnische Realisierung der zu Beginn aufgeführten sechs Punkte innerhalb des ArcView - Projektes beschrieben.

8.3.1 Einbahnstraßen

Eine Einbahnstraße ist vergleichbar mit einem Pfeil in einem Graphen. Die Verbindung von Knoten A nach Knoten B darf nur in einer Richtung durchfahren werden. In ArcView werden Einbahnstraßen mit Hilfe des Feldes ’Oneway’ modelliert. Im folgenden Beispiel stellen die Stolbergstraße und die Richnowstraße Einbahnstraßen dar (vgl. Abb. 8.4).



Attributes of Strassen-Netz

Str_Name	Str_Abschn	Edge#	Fjunction	Tjunction	Oneway
Richnowstr.	36980010	112	54	61	FT
Stolbergstr.	43420010	131	52	53	
Stolbergstr.	43420020	132	53	54	FT

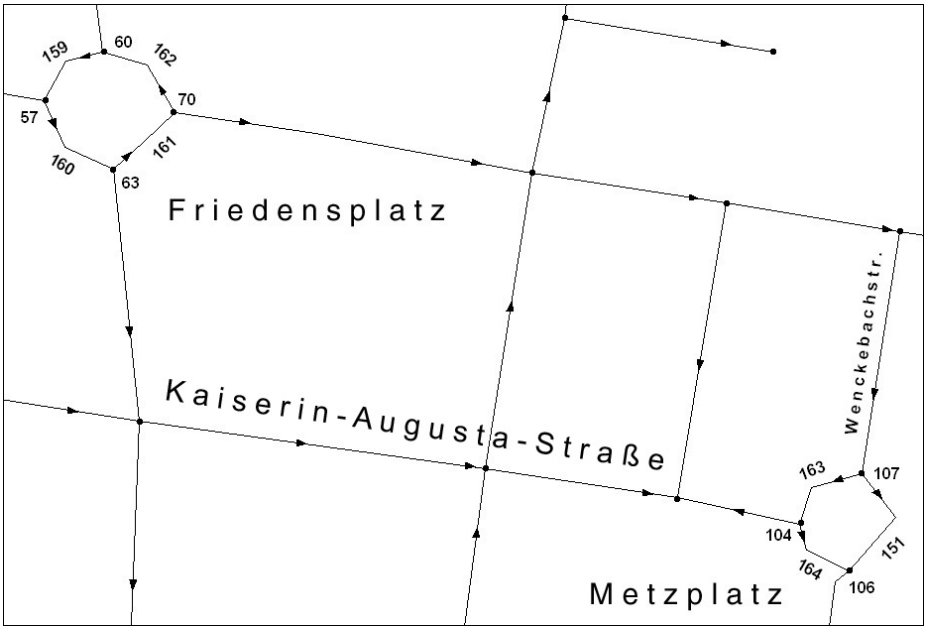
Abb. 8.4: Einbahnstraßen und die dazugehörigen Attribute

Kante 132 darf nur von Startknoten 53 nach Endknoten 54 und Kante 112 nur von Startknoten 54 nach Endknoten 61 (in Digitalisierrichtung) durchlaufen werden. Dies wird durch Setzen des Attributes ’FT’ im Feld ’Oneway’ erreicht. Will man die Durchfahrt nur entgegen der Digitalisierrichtung (vom Endknoten zum Startknoten) erlauben, setzt man anstelle von ’FT’ das inverse Attribut ’TF’. Die Kürzel *F* und *T* stehen hier wiederum für *From* und *To* (engl. von und nach) und werden auch im weiteren Kontext so benutzt.

8.3.2 Kreisverkehr

In einen Kreisverkehr dürfen sich Fahrzeuge nur durch Rechtsabbiegen einfädeln. Die einzig erlaubte Fahrtrichtung ist stets diejenige gegen den Uhrzeigersinn. Technisch gesehen läuft die Realisierung eines Kreisverkehrs also auf eine Einbahnstraße hinaus.

Das folgende Beispiel zeigt den Kreisverkehr am Friedensplatz und am Metzplatz (vgl. Abb. 8.5). Der östliche Bogen des Metzplatzes gehört zur Wenckebachstraße und stellt die Zufahrt zum Wenckebach-Krankenhaus dar (nicht abgebildet). Die Digitalisierrichtung ist hier entgegengesetzt zu der des restlichen Kreisels, was sich in der Turntable-Datei durch den Eintrag ’TF’ im Feld ’Oneway’ bemerkbar macht.



Attributes of Strassen-Netz

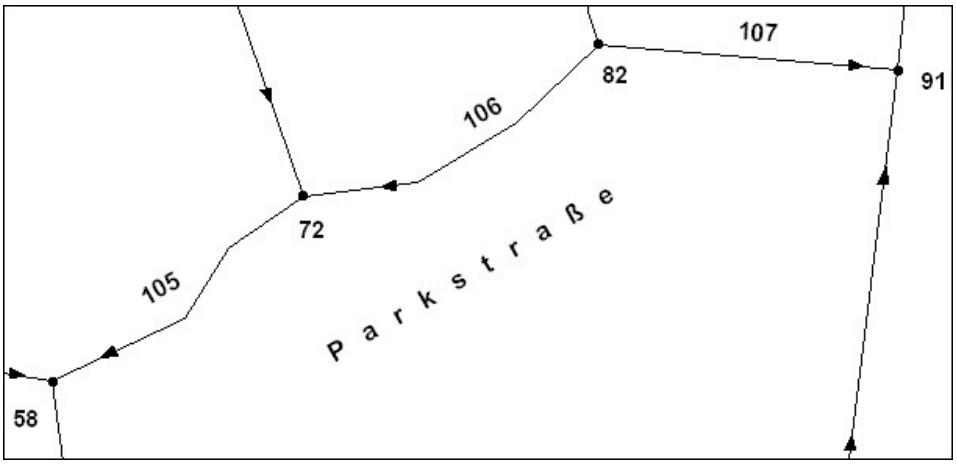
Str_Name	Str_Abschn	Edge#	Fjunction	Tjunction	Oneway
Friedensplatz	69740010	159	60	57	FT
Friedensplatz	69740020	160	57	63	FT
Friedensplatz	69740030	161	63	70	FT
Friedensplatz	69740040	162	70	60	FT
Metzplatz	70780010	163	107	104	FT
Metzplatz	70780020	164	104	106	FT
Wenckebachstr.	47770030	150	105	106	
Wenckebachstr.	47770040	151	107	106	TF
Wenckebachstr.	47770050	152	108	107	

Abb. 8.5: Kreisverkehr und die dazugehörigen Attribute in der Datenbank

Um Mißverständnissen entgegenzuwirken: die Pfeile in Abb. 8.5 symbolisieren - wie in den vorangegangenen und nachfolgenden Beispielen auch - die Digitalisierrichtungen der Liniensegmente (und nicht etwa die erlaubten Fahrtrichtungen).

8.3.3 Für PKW gesperrte Straßen

Straßen, die für die Autos komplett gesperrt, also in beiden Richtungen nicht befahrbar sind, bekommen in der Straßendatenbank im Feld ’Oneway’ den Eintrag ’N’. Im folgenden Beispiel ist das Mittelstück der Parkstraße, welches durch den Lehne-Park am Klarensee entlang führt, für den motorisierten Verkehr nicht zugelassen. Kante 106 von Knoten 82 nach Knoten 72 wird demnach das Attribut ’N’ zugewiesen (vgl. Abb. 8.6).



Attributes of Strassen-Netz

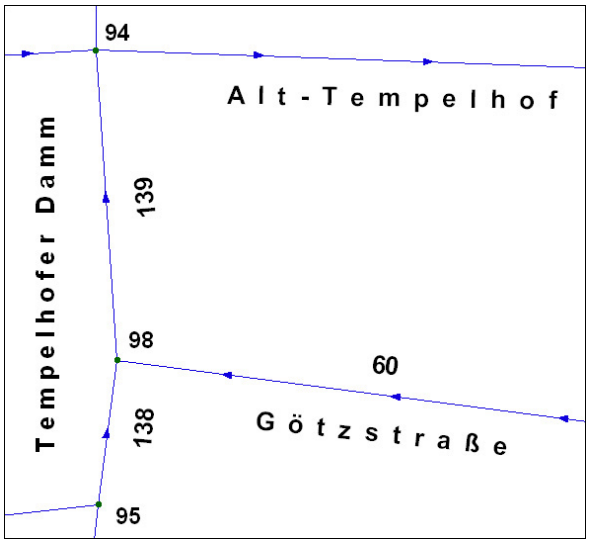
Str_Name	Str_Abschn	Edge#	Fjunction	Tjunction	Oneway
Parkstr.	34010010	105	72	58	
Parkstr.	34010020	106	82	72	N
Parkstr.	34010030	107	82	91	

Abb. 8.6: Gesperrte Straße mit den dazugehörigen Attributen

8.3.4 Abbiegeverbot

An einigen Stellen des Straßennetzes, wie z.B. großen Hauptverkehrsstraßen mit Mittelstreifen, ist das Abbiegen nur in einer Richtung erlaubt. Solche Verbote werden mit Hilfe einer separaten Datei realisiert, welche den Namen Turntable_t trägt. (Table bedeutet im Englischen Tabelle, der Zusatz ’_t’ kennzeichnet das Testgebiet).

Im folgenden Beispiel ist das Linksabbiegen von der Götzstraße in den Tempelhofer Damm nicht gestattet. Dies wird durch folgende Einträge im Turntable erreicht: der Abbiegevorgang von Kante 60 (F_Edge) über Knoten 98 (Node#) nach Kante 138 (T_Edge) bekommt in den Kostenfeldern ’Seconds’ und ’Meters’ den Wert -1 zugewiesen (vgl. Abb. 8.7). Alle anderen Abbiegemöglichkeiten am Knotenpunkt 98 sind erlaubt. Die für einen Abbiegevorgang benötigte Zeit wird an dieser Kreuzung mit 5 Sekunden veranschlagt (Feld ’Seconds’). Die entsprechenden Werte im Feld ’Meters’ werden zu Null gesetzt, da sich die Wegstrecke durch einen Abbiegevorgang nicht verlängert.



Turntable_t

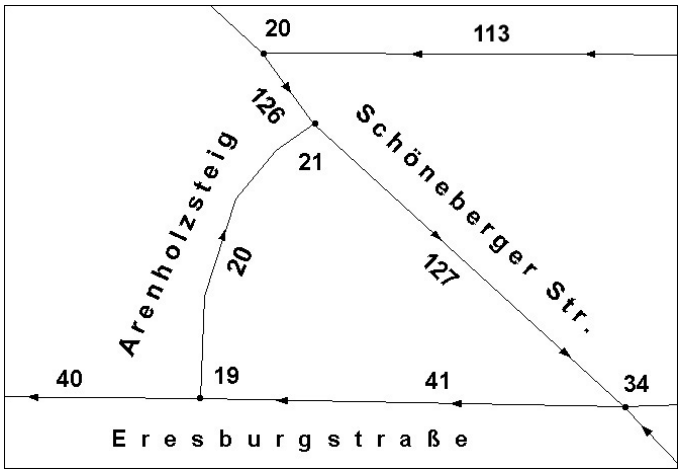
F_Edge	T_Edge	Node#	Seconds	Meters
172	174	96	-1	-1
172	142	96	10	0
172	143	96	10	0
143	174	96	8	0
142	174	96	8	0
171	173	97	-1	-1
171	144	97	10	0
171	143	97	10	0
144	173	97	8	0
143	173	97	8	0
60	138	98	-1	-1
138	60	98	5	0
60	139	98	5	0
139	60	98	5	0

Abb. 8.7: Abbiegeverbot und die entsprechenden Einträge in der Turntable-Datei

8.3.5 Sackgassen

Sackgassen, die mit einem Knoten ohne Folgeanschluß enden (vgl z.B. Abb. 8.4, S. 103), brauchen nicht durch zusätzliche Informationen ergänzt zu werden. Der einzige Weg, die Straße wieder zu verlassen (im Modell wie in der Realität), ist eine 180 Grad-Wende. Sackgassen, die an ihren Endknoten mit weiteren Kanten inzidieren, müssen jedoch durch Restriktionen „verkehrstauglich“ gemacht werden.

Im Beispiel unten ist das Abbiegen in die Sackgasse Arenholzsteig nur von der Eresburgstraße möglich, die Einfahrt von der Schöneberger Str. ist hingegen gesperrt. Die vier möglichen Abbiegevorgänge an Knoten 21 werden also durch den Eintrag -1 in den Kostenfeldern ’Seconds’ (Zeitoptimierung) und ’Meters’ (Wegoptimierung) unterbunden (vgl. Abb. 8.8). Weiteres zu den Themen „Weg, Zeit und Kosten“ folgt in Kapitel 10.



Turntable_t

F_Edge	T_Edge	Node#	Seconds	Meters
20	40	19	5	0
40	20	19	5	0
20	41	19	5	0
41	20	19	5	0
113	126	20	-1	-1
126	113	20	7	0
113	125	20	3	0
125	113	20	3	0
126	20	21	-1	-1
20	126	21	-1	-1
127	20	21	-1	-1
20	127	21	-1	-1

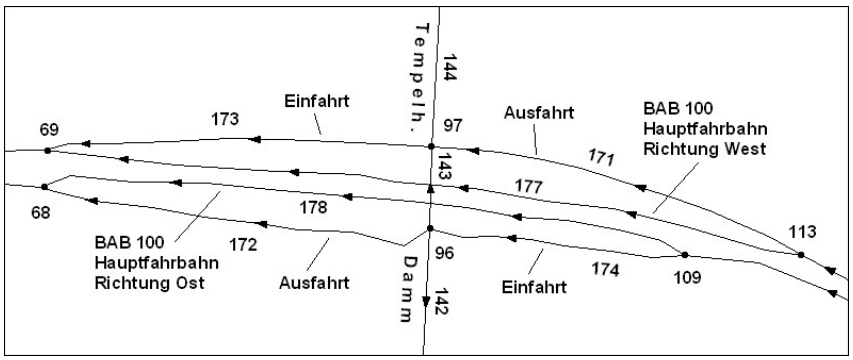
Abb. 8.8: Sackgasse und die dazugehörigen Attribute im Turntable

8.3.6 Autobahnen

Bei Autobahnen werden in der Regel die beiden Hauptfahrbahnen sowie Ein- und Ausfahrten als separate Liniensegmente modelliert. Alle Straßenabschnitte dürfen nur in einer Richtung durchfahren werden, das Prinzip ist also wieder das der Einbahnstraße.

8.3.6.1 Ein- und Ausfahrten

Am Ende von Ein- und Ausfahrten muß auf verbotene Abbiegemöglichkeiten geachtet werden. Die Modellierung der Hauptfahrbahn als Einbahnstraße gewährleistet, daß am Ende der Einfahrt kein „U-Turn“ vollzogen werden kann (Eintrag von ’FT’ oder ’TF’ im Feld ’Oneway’ der Straßendatenbank). Am Ende einer Ausfahrt kann es durch die jeweilige Verkehrsregelung mehrere Abbiegemöglichkeiten geben. Das folgende Beispiel zeigt die Bundesautobahn 100 am Kreuzungspunkt Tempelhofer Damm (vgl. Abb. 8.9).
Beide Ausfahrten in Ost- und Westrichtung gestatten sowohl ein Links- als auch ein Rechtsabbiegen in den Tempelhofer Damm. Verboten hingegen ist in beiden Fällen der Weg geradeaus - die sofortige Wiederauffahrt auf die Autobahn. Die Abbiegeregelungen werden, wie in den vorangegangenen Beispielen beschrieben, durch Einträge in den Turntable-Feldern ’Seconds’ und ’Meters’ erreicht. Dabei kann für jede Kreuzung ein eigener Zeitplan entworfen werden.
An den Knotenpunkten 96 und 97 wird die Wartezeit z.B. durch Ampeln verlängert. Die Zeitspanne für einen Abbiegevorgang am Ende der Ausfahrt wird mit 10 Sekunden etwas größer angesetzt, als das entsprechende Intervall zu Beginn der Einfahrt (8 Sekunden).



Turntable_t

F_Edge	T_Edge	Node#	Seconds
172	174	96	-1
172	142	96	10
172	143	96	10
143	174	96	8
142	174	96	8
171	173	97	-1
171	144	97	10
171	143	97	10
144	173	97	8
143	173	97	8
60	138	98	-1
138	60	98	5
60	139	98	5
139	60	98	5

Attributes of Strassen-Netz

Str_Name	Str_Abschn	Edge#	Fjunction	Tjunction	Oneway
BAB 100 Ausfahrt nach Tempelhofer Damm Nord ---> W	990420010	171	113	97	FT
BAB 100 Ausfahrt nach Tempelhofer Damm Süd ---> O	990430010	172	96	68	TF
BAB 100 Einfahrt von Tempelhofer Damm Nord ---> W	990440010	173	97	69	FT
BAB 100 Einfahrt von Tempelhofer Damm Süd ---> O	990450010	174	109	96	TF
BAB 100 Hauptfahrbahn 2 ---> W	993730010	177	113	69	FT
BAB 100 Hauptfahrbahn 2 ---> O	993740010	178	109	68	TF

Abb. 8.9: Autobahn Ein- und Ausfahrten und die dazugehörigen Attribute

8.3.6.2 Über- und Unterführungen

An großen Autobahnkreuzen werden häufig Brückenkonstruktionen gebraucht, die ein reibungsloses Abfließen des Verkehrs gewährleisten. Auch an Orten, wo Autobahntrassen auf andere Verkehrswege (wie z.B. Eisenbahngleise oder Bundesstraßen) treffen, sind Über- und Unterführungen notwendig. In ArcView gibt es zwei Möglichkeiten, solche 'Overpasses' und 'Underpasses' zu modellieren:

- Man digitalisiert beide Verkehrswege als ungebrochene und voneinander unabhängige Linien, so daß kein gemeinsamer Knotenpunt entsteht. Eine Linie führt automatisch über die andere hinweg, ein Abbiegen ist nicht möglich.
- Man digitalisiert beide Verkehrswege als vier gebrochene Linien, welche an einem gemeinsamen Knotenpunkt aufeinandertreffen. Der Attributtabelle des Straßennetzes fügt man zwei Felder 'F_Elev' and 'T_Elev' hinzu (engl. elevation = Höhe). Wenn z.B. Straße A am Knotenpunkt über Straße B hinweg führt, gibt man den beiden Segmenten von A den Wert Eins und den anderen beiden den Wert Null. Will man zu einem späteren Zeitpunkt aus der Überführung eine Kreuzung machen, weist man allen vier Liniensegmenten einfach denselben Wert zu.

Die erste Methode ist sicherlich die elegantere und wurde auch im vorliegenden Projekt verwirklicht. Das folgende Beispiel zeigt die Unterführung der Manteuffelstraße unterhalb der Bundesautobahn (BAB) 100 sowie die Überführung der Ausfahrt Alboinstraße überhalb der Hauptfahrbahn der BAB 100 (vgl. Abb. 8.10).

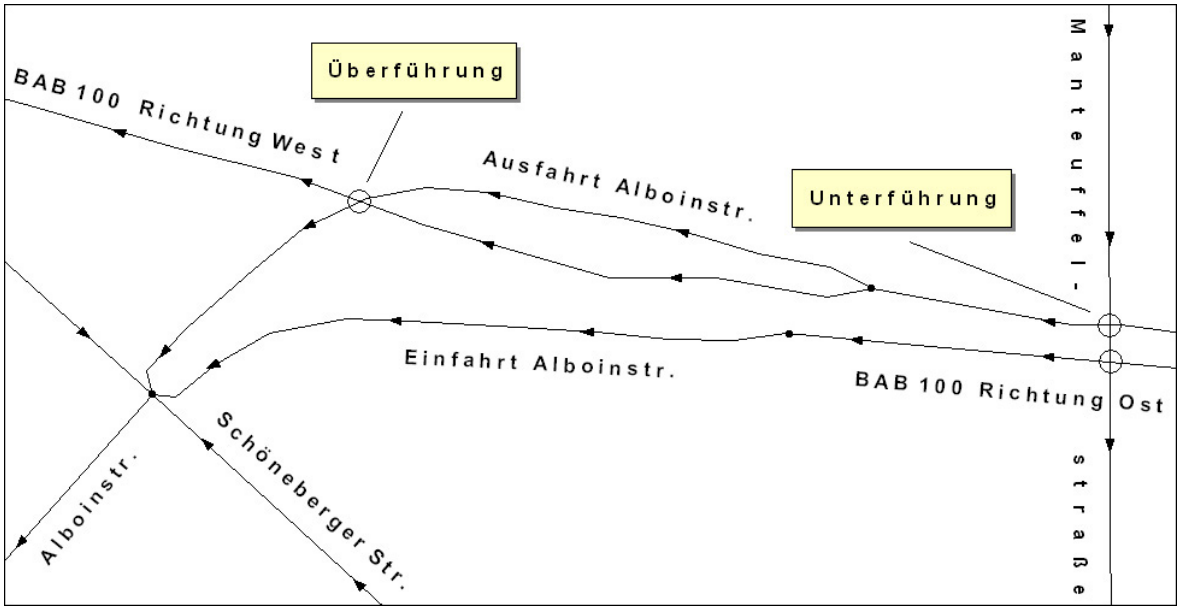


Abb. 8.10: Autobahn Über- und Unterführung

9. Optimierung nach Wegstrecke oder Fahrtzeit

In diesem Kapitel werden die technischen Voraussetzungen für eine Routenoptimierung im Testgebiet vorgestellt. Zuvor erfolgt eine kurze Beschreibung der ArcView - internen Skriptsprache „Avenue“.

9.1 Programmiersprache „Avenue“

Avenue gehört zur Gruppe der objektorientierten Programmiersprachen. Analog zu anderen Sprachen dieser Art (z.B. C++ oder Java) ist auch der Sprachumfang von Avenue durch vordefinierte Elemente strukturiert. Man unterscheidet unter anderem zwischen

- Klassen (Classes),
- Objekten (Objects) und
- Anfragen (Requests).

Objekte mit gleichen oder ähnlichen Eigenschaften werden in der Regel derselben Klasse zugeordnet. Die Ansicht des Testgebietes und die Ansicht des Gesamtgebietes im vorliegenden Projekt sind z. B. Objekte der Klasse ’View’. Will man über ein Objekt etwas bestimmtes wissen, so richtet man eine Anfrage an das Objekt. Dies geschieht nach dem Schema *Object.Request*, das Objekt wird immer durch einen Punkt von der Anfrage getrennt. Mit der Befehlsfolge ’theView.Print’ wird beispielsweise an das Objekt ’theView’ die Anfrage ’Print’ gestellt. Als Antwort würde der Ausdruck des Views erfolgen. Auf manche Anfragen erhält man als Rückgabewert ein neues Objekt. Die Befehlsfolge ’theName = theView.GetName’ z. B. liefert auf die Anfrage ’GetName’ den Namen des Objektes ’theView’ zurück und weist ihn sogleich der Variablen ’theName’ zu. Auch mehrere Anfragen hintereinander in der Notierung *Object.Request.Request.Request* sind möglich. Der Anfang eines Avenue-Skriptes könnte z.B. folgendermaßen aussehen:

```
theProject = av.GetProject
theView = theProject.FindDoc(“Testgebiet Tempelhof-Nord”)
theTheme = theView.FindTheme(“Strassen-Netz“)
```

In kompakterer Form ließe sich dafür schreiben:

```
theTheme = av.GetProject.FindDoc(“Testgebiet Tempelhof-Nord”).FindTheme(“Strassen-Netz“)
```

Jedes *Object.Request*-Paar wird einzeln von links nach rechts abgearbeitet. Auf jede Anfrage wird ein Objekt zurückgegeben und an dieses im nächsten Schritt eine weitere Anfrage gestellt. Das Endergebnis ist die Zuweisung des Objektes ’Strassen-Netz’ an die Variable ’theTheme’.

Avenue-Skripte können effektiv für verschiedene Aspekte der Routenoptimierung eingesetzt werden, wie in den nächsten Kapiteln gezeigt werden soll.

9.2 Optimierung nach Wegstrecke

Um Berechnungen bezüglich des Fahrweges anstellen zu können, ist die Erfassung aller Straßensegmentlängen des Testgebietes erforderlich. Der Attributtabelle des Themas Straßennetz fügt man zuerst ein Feld 'Meters' hinzu. Mit Hilfe des sog. 'Field Calculators' führt man dann das Avenue - Statement [Meters] = [Shape].ReturnLength aus (vgl. Abb. 9.1). Durch die Anfrage wird jedem Straßensegment der Tabelle seine Länge in Metern zugewiesen.

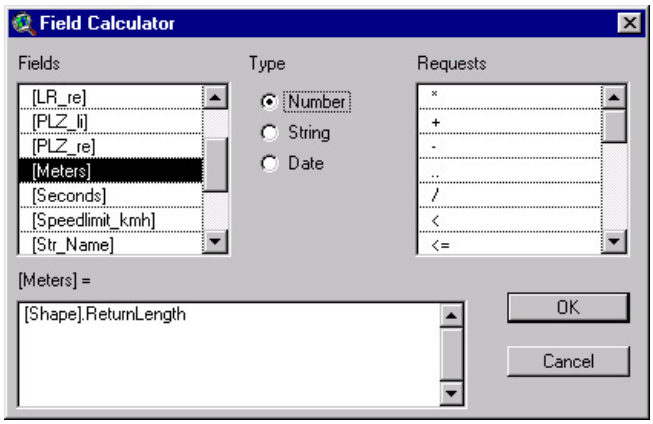


Abb. 9.1: Anfrage 'Kantenlängen'

Alle Liniensegmente des Testgebietes Tempelhof-Nord werden durch diesen Vorgang in bewertete Kanten umgewandelt. Abbildung 9.2 zeigt eine Übersicht zu den Reisekosten 'Wegstrecke'.

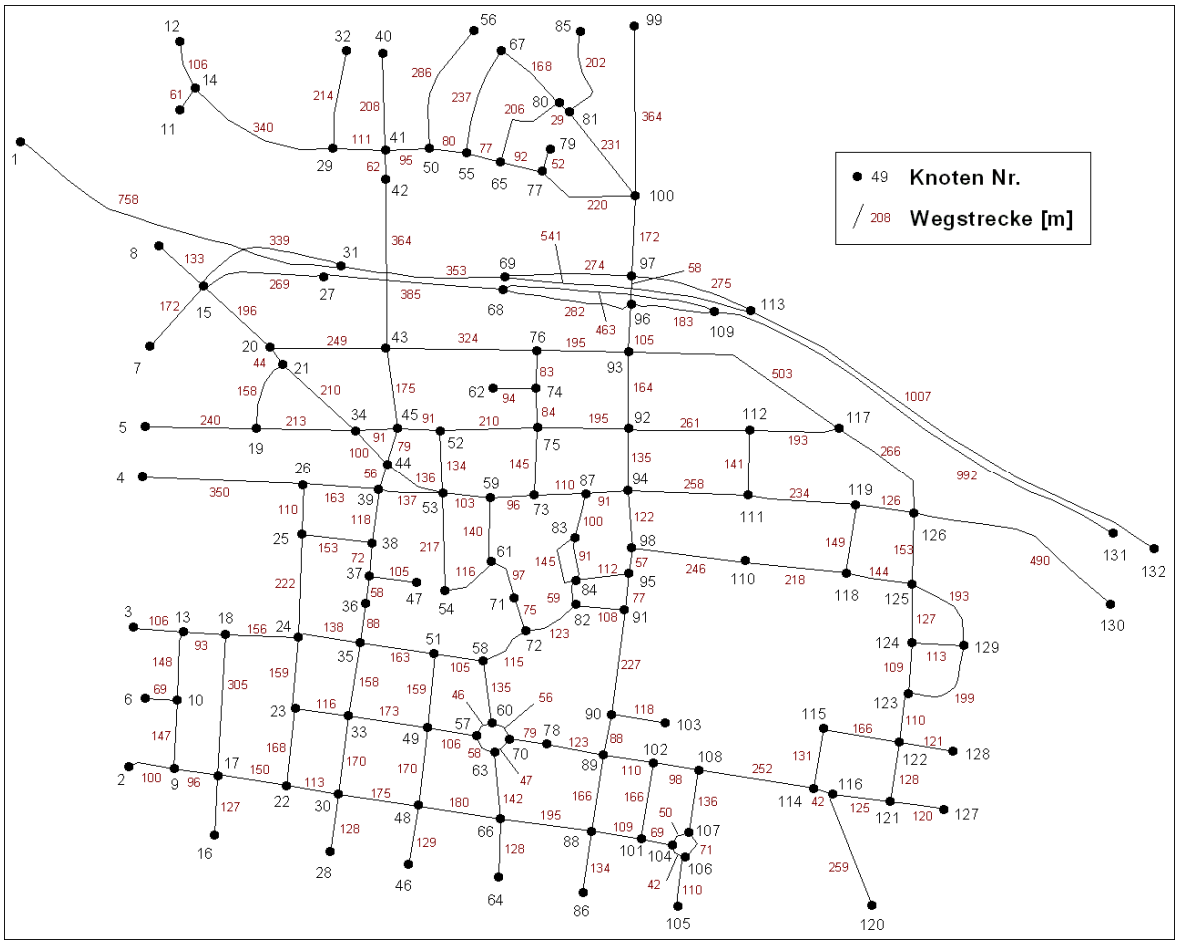


Abb. 9.2: Länge der Kanten im Testgebiet (auf ganze Zahlen gerundet)

9.3 Optimierung nach Fahrtzeit

Analog zu Kapitel 9.2 wird für jedes Straßensegment des Testgebietes die Zeit benötigt, die zum Durchlaufen einer Kante notwendig ist. Dazu muß man natürlich festlegen, wie schnell auf den einzelnen Straßenabschnitten gefahren werden darf. Ich habe für Autobahnen ein durchschnittliche Geschwindigkeit von 130 km/h und für alle restlichen Straßen ein Tempolimit von 50 km/h zu Grunde gelegt. Mit Hilfe dieser beiden Größen und den im vorigen Kapitel ermittelten Wegstrecken kann für jede Kante die Durchlaufzeit ermittelt werden.

Man fügt der Attributtabelle des Themas Straßennetz ein Feld 'Speedlimit_kmh' hinzu und belegt alle Kanten mit den Werten 50 oder 130. Anschließend erzeugt man ein neues Feld 'Seconds' und führt im 'Field Calculator' das in Abb. 9.3 gezeigte Avenue-Statement aus. Durch die Anfrage wird jedem Segment der Tabelle die Durchlaufzeit in Sekunden zugewiesen.

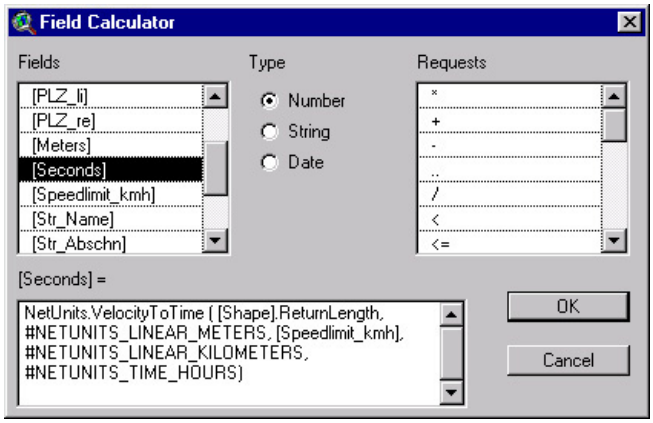


Abb. 9.3: Anfrage 'Durchlaufzeiten'

Die Einheiten von Geschwindigkeit, Zeit und Wegstrecke können über den Befehl 'NETUNITS' definiert werden. Für die Berechnung der Durchlaufzeiten ist es nicht unbedingt erforderlich, daß sie von der Größenordnung her zueinander passen. Solange die gewählten Einheiten den Werten in der Tabelle entsprechen, führt ArcView die Berechnung korrekt durch. Eventuell vorzunehmende Konvertierungen (z.B. Meter in Kilometer) werden dann automatisch berücksichtigt.

Die Ausgabe des 'ReturnLength-Requests' erfolgt in der Einheit des gewählten Koordinatensystems (in unserem Fall also in Metern). Hätte man für das Projekt ein Koordinatensystem mit dezimalen Gradangaben (z.B. Geograhische Koordinaten) gewählt und wären die Tempolimits in Meilen pro Stunde gegeben, so würde die Berechnung der Durchlaufzeiten genauso funktionieren. #NETUNITS_LINEAR_METERS müßte nur durch #NETUNITS_LINEAR_DEGREES und #NETUNITS_LINEAR_KILOMETERS nur durch #NETUNITS_LINEAR_MILES ersetzt werden.

Die Ausgabe des 'VelocityToTime-Requests' erfolgt standardmäßig in Sekunden (was im vorliegenden Fall genau das Richtige ist). Durch einen Zusatz (z.B. '/60') können diese aber auch in andere Einheiten (Minuten, etc.) umgerechnet werden.

Mit Hilfe des zweiten Avenue Statements bekommen alle Kanten des Testgebietes Tempelhof-Nord eine zusätzliche Bewertung. Abbildung 9.4 zeigt eine Übersicht zu den alternativen Reisekosten ’Fahrtzeit’.

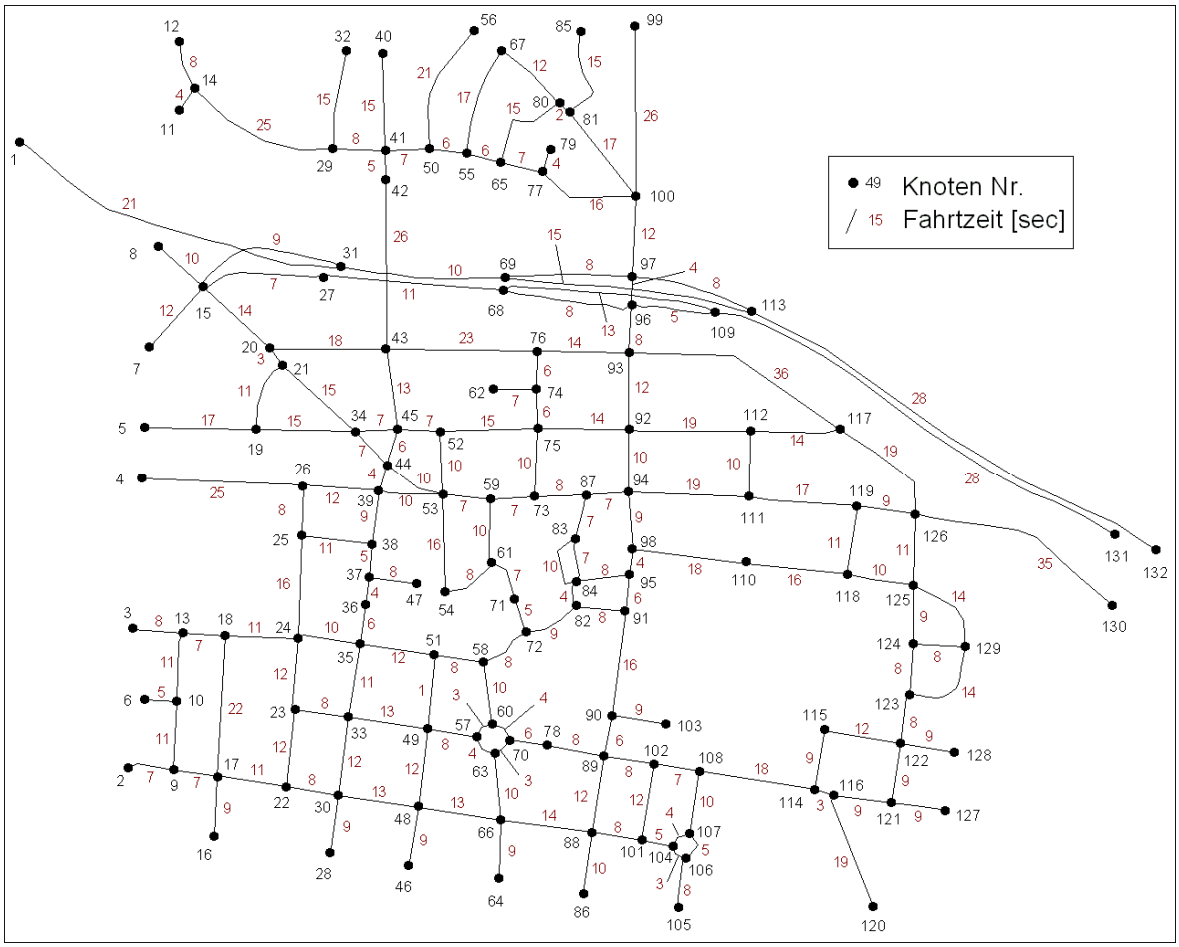


Abb. 9.4: Durchfahrzeiten für die Kanten des Testgebietes (auf ganze Zahlen gerundet)

Die Kantenlängen des Testgebietes erscheinen im Feld ’Meters’ der Straßendatenbank auf einen Zentimeter und die Sekunden im Feld ’Seconds’ auf eine Zehntelsekunde genau (vgl. Abb. 9.5).

Attributes of Strassen-Netz (Testgebiet)

Edge#	Fjunction	Tjunction	Oneway	Speedlimit_kmh	Meters	Seconds	FT_Seconds	TF_Seconds
1	15	7		50	171.60	12.4	12.4	12.4
2	23	33		50	115.53	8.3	8.3	8.3
3	33	49		50	173.28	12.5	12.5	12.5
4	49	57		50	105.86	7.6	7.6	7.6
5	70	78		50	79.01	5.7	5.7	5.7

Abb. 9.5: Kantenlängen und Durchfahrzeiten in der Straßendatenbank

Nach der vollständigen Erfassung der Kantenlängen und Durchlaufzeiten ist das Straßennetzwerk des Testgebietes für eine Routenoptimierung mit der Network Analyst - Extension geeignet.

10. Beispiele zur Routenplanung

In diesem Kapitel sollen die Möglichkeiten der Caritas-Routenplanung anhand von Beispielen erläutert werden. Ziel ist es, einen allgemeinen Überblick zu geben. Auf eine detaillierte Beschreibung, welche Menüpunkte, Buttons oder Tools wann und wo einzusetzen sind, wird hier verzichtet. (Eine sinnvolle Anleitung zur Nutzung des Programms sollte auf jeden Fall „online“ am Rechner erfolgen.)

10.1 Vorgehensweise in ArcView

Schritt 1: In den Views ’Testgebiet Tempelhof-Nord’ oder ’Bezirk Tempelhof und Ortsteil Lankwitz’ wird das Thema Straßennetz aktiviert und ein neues Routenproblem erzeugt, welches als Symbol unter dem Namen ’Route x’ in der Themenliste erscheint.

Schritt 2: Die Kundenadressen, zwischen denen die Rundreise optimiert werden soll, müssen dem Programm mitgeteilt werden. Man könnte für jede geplante Tour die Monatsliste mit allen Kunden so verkleinern, daß nur noch die gewünschten Kunden enthalten sind. Für jede Tour müßte eine neue Adressenliste erstellt und eingelesen werden, was ziemlich umständlich wäre. Da eine Caritas-Mitarbeiterin im Schnitt nur sechs bis acht Hausbesuche pro Tag macht, ist die Eingabe von Straßenname und Postleitzahl (letztere kann optional weggelassen werden) über die interaktive Adressenmaske (vgl. Abb. 7.25, S. 95) der einfachere und schnellere Weg. Außerdem bietet sich so die Möglichkeit, auf Änderungen in der Planung schnell und flexibel zu reagieren. Alte Adressen können bequem herausgenommen und neue hinzugefügt werden.

Schritt 3: Wenn die eingegebenen Kundenadressen als Symbole im View erschienen sind, wählt man zwischen Weg- oder Zeitoptimierung. Das Programm versucht die *beste* Route zu finden, wobei die Besuchsreihenfolge sich nach der Abfolge der eingegebenen Adressen richtet. (Diese können im Dialogfenster, vgl. Abb.10.2 nächste Seite, auch neu sortiert werden.) Zwei weitere Optionen sind wählbar: zum ersten hat man die Möglichkeit, die Tour wieder am Startort enden zu lassen. Zum zweiten kann man den Rechner veranlassen, nicht die vorgegebene Besuchsreihenfolge zu verwenden, sondern nach der Tour mit der besten (also kürzesten oder schnellsten) Reihenfolge zu suchen.

Schritt 4: Die gefundene Tour wird als Liniensymbolik mit Pfeilen auf dem Bildschirm dargestellt. Zusätzlich kann man sich eine detaillierte Textbeschreibung mit Straßennamen, Abbiegehinweisen, Wegstrecken oder Fahrtzeiten anzeigen lassen und bei Bedarf ausdrucken (vgl. Abb. 10.1 und 10.2 nächste Seite).

10.2 Zwei Beispiele im Testgebiet Tempelhof-Nord

10.2.1 Kundentour A (wegoptimiert in vorgegebener Reihenfolge)

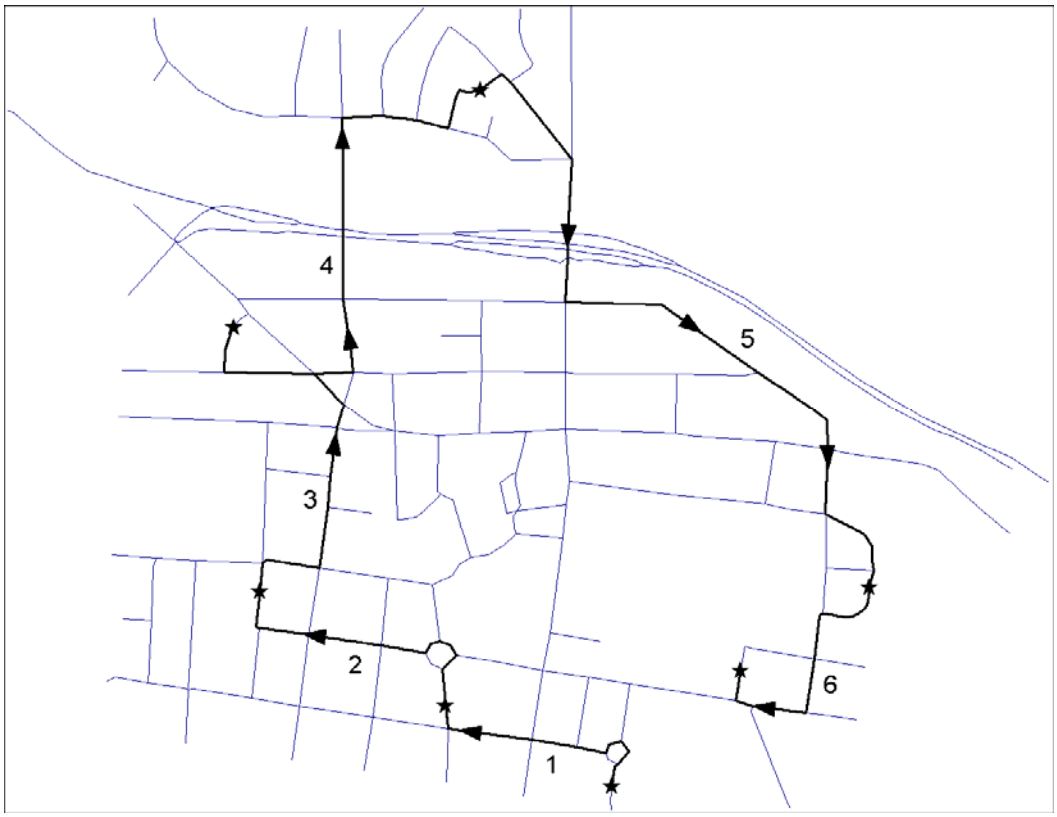


Abb. 10.1: Wegoptimierter Tourpfad im Straßenverkehrsnetz

Directions

Starting from Wenckebachstr. 19
Turn left onto Wenckebachstr.
Travel on Wenckebachstr. for 126 m
Turn left onto Metzplatz
Travel on Metzplatz for 50 m
Turn right onto Kaiserin-Augusta-Str.
Travel on Kaiserin-Augusta-Str. for 373 m
Turn right onto Blumenthalstr.
Travel on Blumenthalstr. for 57 m
Turn right into Blumenthalstr. 15

Starting from Blumenthalstr. 15
.....
.....
.....

Turn right into Götzstr. 57

Starting from Götzstr. 57
Turn right onto Götzstr.
Travel on Götzstr. for 160 m
Turn left onto Felixstr.
Travel on Felixstr. for 238 m
Turn right onto Albrechtstr.
Travel on Albrechtstr. for 167 m
Turn right onto Templerzeile
Travel on Templerzeile for 74 m
Turn left into Templerzeile 10

Total distance traveled is 6211 m
==========

Attributes of Weg-Optimierung

Shape	Path_id	F_label	T_label	F_cost	T_cost
PolyLine	1	Wenckebachstr. 19	Blumenthalstr. 15	0.000	606.224
PolyLine	2	Blumenthalstr. 15	Gäßnerweg 54	606.224	1329.384
PolyLine	3	Gäßnerweg 54	Arenholzsteig 5	1329.384	2347.849
PolyLine	4	Arenholzsteig 5	Siegertweg 5	2347.849	3764.936
PolyLine	5	Siegertweg 5	Götzstr. 57	3764.936	5571.713
PolyLine	6	Götzstr. 57	Templerzeile 10	5571.713	6211.318

Weg-Optimierung

Total route cost: 6211 m

Label	meters
Wenckebachstr. 19	0
Blumenthalstr. 15	606
Gäßnerweg 54	1329
Arenholzsteig 5	2348
Siegertweg 5	3765
Götzstr. 57	5572
Templerzeile 10	6211

Find best order
Return to origin
Directions...
Load Stops...
Save Stops...
Properties..
Number of stops: 7

Abb. 10.2: Die zur Kundentour gehörigen Attribute (oben), das Dialogfenster mit Einstellungsmöglichkeiten bzgl. der Optimierung und die Strecke textlich im Detail (links).

10.2.2 Kundentour B1 (zeitoptimiert in vorgegebener Reihenfolge)

Abb. 10.3: Zeitoptimierter Tourpfad im Straßenverkehrsnetz

Directions

```
Starting from Hoeppnerstr. 7
Turn left onto Hoeppnerstr.
  Travel on Hoeppnerstr. for 5 sec
Turn right onto Tempelhofer Damm
  Travel on Tempelh. Damm for 12 sec
Turn right onto BAB 100 Einf. T-Damm
  Travel on BAB 100 Einfahrt for 8 sec
Continue straight onto BAB 100
  Travel on BAB 100 for 10 sec
Continue straight onto BAB 100 Ausf.
  Travel on BAB 100 Ausfahrt for 9 sec
Turn left onto Schöneberger Str.
  Travel on Schöneberger Str. for 9 sec
Turn right into Schönebergerstr. 27

Starting from Schönebergerstr. 27
.....
.....

Turn right into Rothariweg 24

Starting from Rothariweg 24
Turn right onto Rothariweg
  Travel on Rothariweg for 16 sec
Turn right onto Bosestr.
  Travel on Bosestr. for 28 sec
Turn left onto Manteuffelstr.
  Travel on Manteuffelstr. for 73 sec
Continue straight onto Boelckestr.
  Travel on Boelckestr. for 5 sec
Turn right onto Hoeppnerstr.
  Travel on Hoeppnerstr. for 36 sec
Turn left into Hoeppnerstr. 7

Total distance traveled is 615 sec
==========
```

Attributes of Zeit-Optimierung

Shape	Path_id	F_label	T_label	F_cost	T_cost
PolyLine	1	Hoeppnerstr. 7	Schönebergerstr. 27	0.000	82.241
PolyLine	2	Schönebergerstr. 27	Richnowstr. 2	82.241	151.095
PolyLine	3	Richnowstr. 2	Luise-Henriette-Str. 3	151.095	214.058
PolyLine	4	Luise-Henriette-Str. 3	Ringbahnstr. 42	214.058	299.203
PolyLine	5	Ringbahnstr. 42	Zastrowstr. 5	299.203	363.091
PolyLine	6	Zastrowstr. 5	Albrechtstr. 117	363.091	457.103
PolyLine	7	Albrechtstr. 117	Kaiserin-Augusta-Str. 68	457.103	502.316
PolyLine	8	Kaiserin-Augusta-Str. 68	Rothariweg 24	502.316	555.423
PolyLine	9	Rothariweg 24	Hoeppnerstr. 7	555.423	727.265

Zeit-Optimierung

Total route cost: 00:12:07 hh:mm:ss

Label	hh:mm:ss
Hoeppnerstr. 7	00:00:00
Schönebergerstr. 27	00:01:22
Richnowstr. 2	00:02:31
Luise-Henriette-Str. 3	00:03:34
Ringbahnstr. 42	00:04:59
Zastrowstr. 5	00:06:03
Albrechtstr. 117	00:07:37
Kaiserin-Augusta-Str. 68	00:08:22
Rothariweg 24	00:09:15
Hoeppnerstr. 7	00:12:07

☐ Find best order
☑ Return to origin

Directions...
Load Stops...
Save Stops...
Properties..

Number of stops: 9

Abb. 10.4: Attribute, Dialogfenster u. Streckenbeschreibung

Kundentour B2 (zeitoptimiert in bester Reihenfolge)

Im obigen Beispiel war die Reihenfolge der Kundenbesuche vorgegeben, wobei die Tour wieder am Ausgangsort enden sollte (’Return to origin’). Bei einer Vereinbarung von festen Besuchszeiten, ist eine solche Auswahl sinnvoll. Es ist aber auch der folgende Fall denkbar: eine Krankenschwester möchte für alle Hausbesuche eines Tages die schnellste (oder kürzeste) Route wissen - die Besuchsreihenfolge spielt dabei keine Rolle (’Find best Order’). Erst nach der Planung teilt sie den Kunden die Termine mit. Das Ergebnis einer solchen Optimierung (für die Kundenadressen aus Beispiel B1) zeigt Abbildung 10.5.

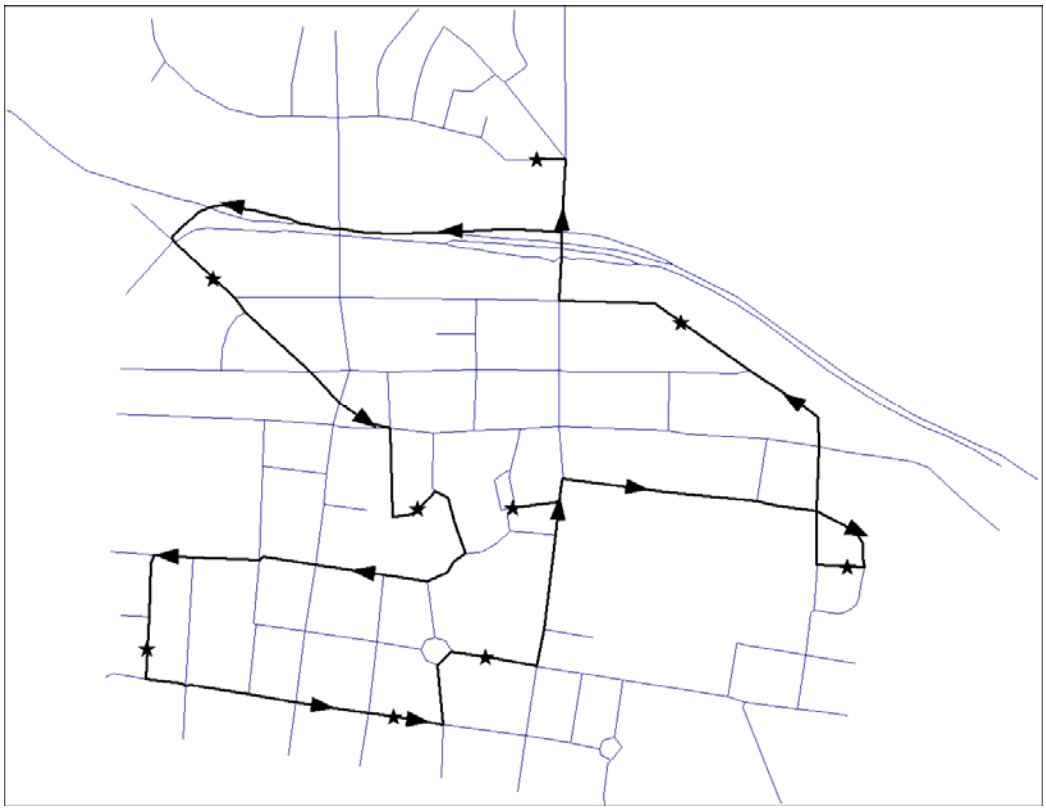


Abb. 10.5: Zeitoptimierte Reihenfolge der Kundenbesuche

Label	hh:mm:ss
Hoeppnerstr. 7	00:00:00
Schönebergerstr. 27	00:01:22
Richnowstr. 2	00:02:31
Rothariweg 24	00:04:17
Kaiserin-Augusta-Str. 68	00:05:10
Albrechtstr. 117	00:05:50
Luise-Henriette-Str. 3	00:06:52
Zastrowstr. 5	00:08:31
Ringbahnstr. 42	00:09:35
Hoeppnerstr. 7	00:10:36

Abbildung 10.6 (links) zeigt die Fahrtzeit für die Tour mit der besten Reihenfolge. Anstelle von 12:07 min werden jetzt nur noch 10:36 min benötigt. Führt man eine Wegoptimierung für das gleiche Beispiel durch, ergeben sich mit vorgegebener Reihenfolge 8853 m und mit bester Reihenfolge 7738 m. Besonders bei sehr langen Rundreisen (vgl. Kap. 10.4) kann die ’Find best Order’ - Option erhebliche Zeit- und Streckeneinsparungen bewirken.

10.3 Übersicht zur Erreichbarkeit der Caritas-Kunden

Um einen Überblick über die Verteilung der Caritas-Kunden (hier von Mai und September 2000) und deren Erreichbarkeit zu gewinnen, können mit Hilfe der Network-Analyst-Extension sogenannte 'Service Areas' generiert werden. Ausgehend vom Standort der Sozialstation werden Abstandszonen berechnet, die in gewissen Zeit- oder Kilometerintervallen auf dem Straßenverkehrsnetz erreichbar sind (vgl. Abb. 10.7 u. 10.8). Mit Hilfe solcher Graphiken können im Vorfeld der Tourenplanung z.B. benachbarte Kundenwohnorte, die auf günstigen Wegen kombinierbar sind, zu Gruppen zusammengefaßt werden (Cluster-Analyse).

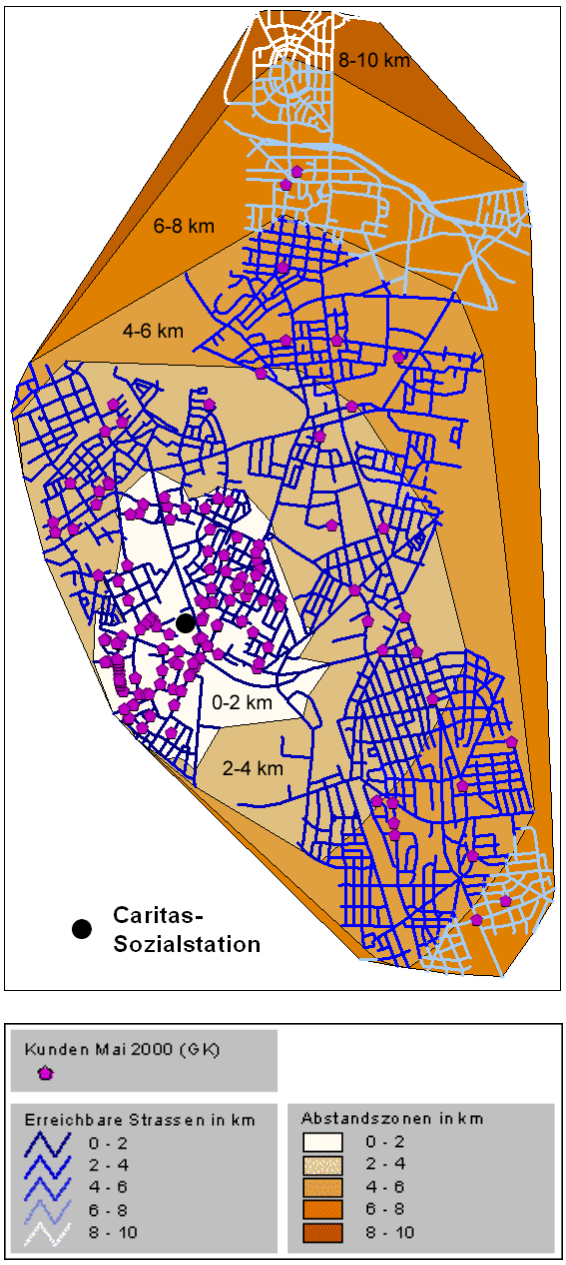


Abb. 10.7: Erreichbarkeit d. Kunden in km

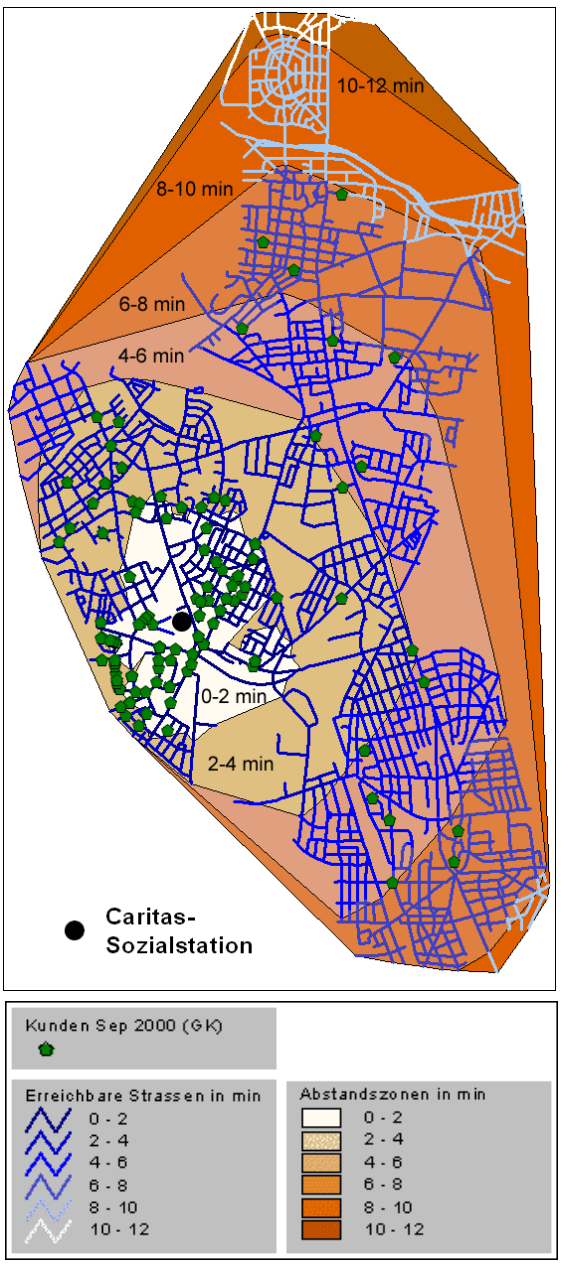


Abb. 10.8: Erreichbarkeit d. Kunden in min

10.4 Zwei Beispiele im Gesamtgebiet Tempelhof und Lankwitz

10.4.1 Kundentour C (wegoptimiert)

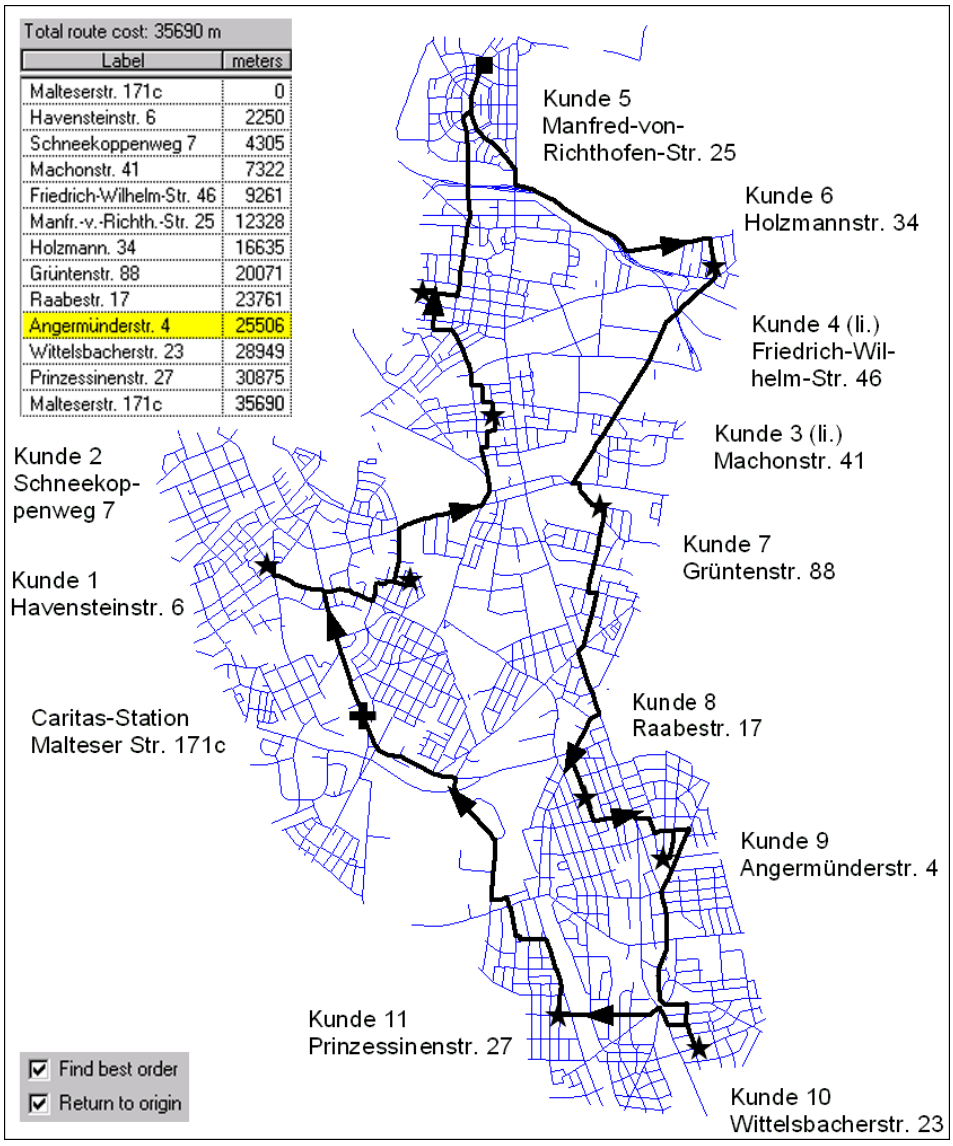


Abb. 10.9: Tourpfad und Reisekosten der einzelnen Streckenabschnitte

Die Routenoptimierung funktioniert für das Gesamtgebiet im Prinzip genauso wie für das Testgebiet. Für alle Kanten wurde mit den in Kapitel 9 erläuterten *Request.Object* – Statements eine Bewertung nach Wegstrecke und Fahrtzeit durchgeführt.

Die Ergebnisse im Gesamtgebiet sind allerdings nur bedingt für eine reale Tourenplanung einsetzbar. Verkehrstechnische Besonderheiten (Einbahnstraßen, gesperrte Straßen etc.) wurden hier nur sehr begrenzt (in den meisten Fällen gar nicht) berücksichtigt. Auf eine Modellierung der Abbiegevorgänge (die Erstellung eines Turntables) wurde komplett verzichtet. Zur Erzeugung von realistischen Rundreisen im Gesamtgebiet wäre eine vollständige Durchführung des Maßnahmenkataloges notwendig, wie sie für das Testgebiet beschrieben wurde.

10.4.2 Kundentour D (zeitoptimiert)

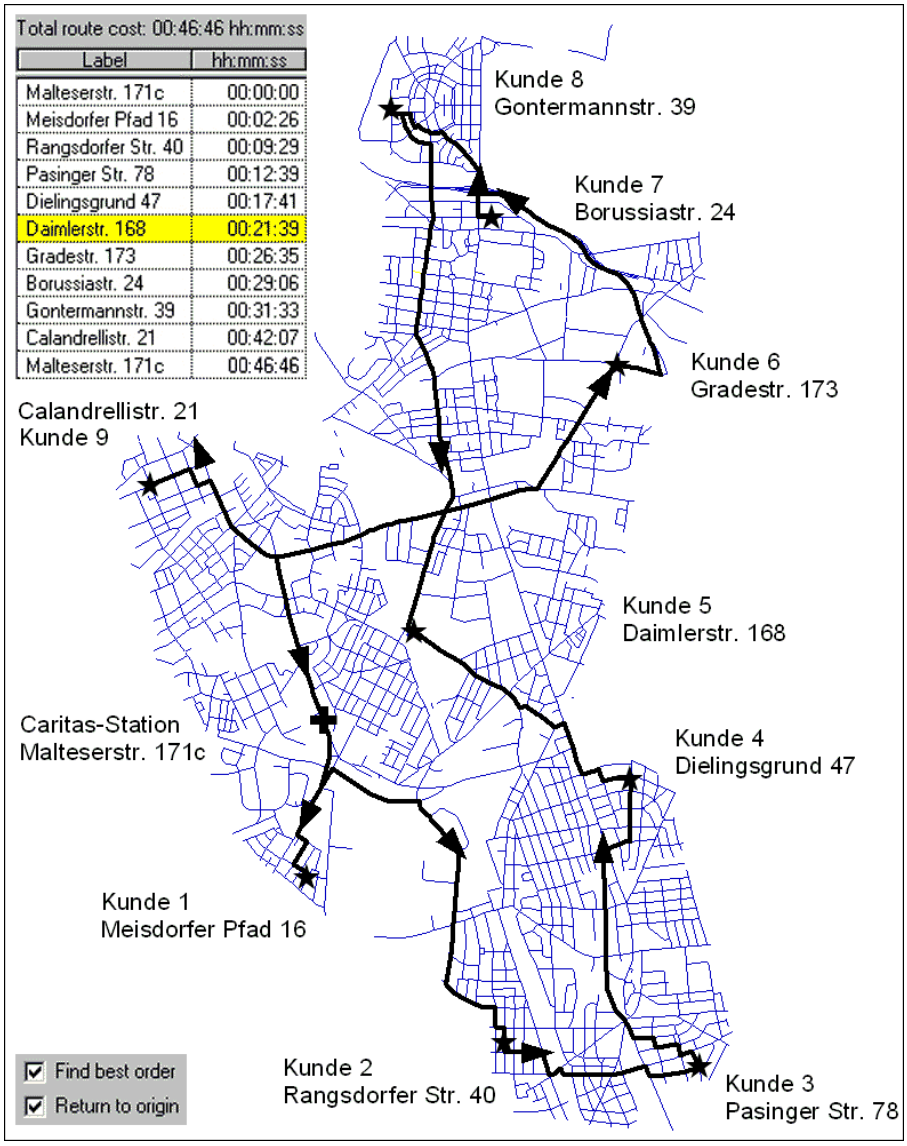


Abb. 10.10: Tourpfad und Reisekosten der einzelnen Streckenabschnitte

An dieser Stelle sei noch eine Besonderheit erwähnt, welche in bestimmten Fällen die Ergebnisse der zeitoptimierten Tourenplanung verbessern kann. Eine Kante von Knoten A nach Knoten B muß nicht zwangsläufig dieselben Kosten für beide Durchlaufrichtungen aufweisen. Der Weg von A nach B kann mit 15 sec bewertet werden und der Weg von B nach A mit 25 sec. Gründe für ein solches Vorgehen sind z.B. Bauarbeiten auf einer Straßenseite oder ein stärkeres Verkehrsaufkommen auf einer Spur während der Rushhour. Die Modellierung läßt sich mit Hilfe von zwei zusätzlichen Feldern (’FT_Seconds’ und ’TF_Seconds’) in der Straßendatenbank bewerkstelligen (vgl. Abb. 9.5, S. 112). Bei der Abfrage des Kostenfeldes in den Optimierungseinstellungen wird anstelle von ’Seconds’ dann einfach ’Seconds (directional)’ ausgewählt.

11. Zusammenfassung

11.1 Resümee

Durch Verwendung geeigneten Datenmaterials (Straßennetz und Wohnblöcke im Vektordatenformat) und kundenspezifischer Informationen (Tabellen mit Namen, Adressen und Altersangaben) kann unter Berücksichtigung von Verkehrsregeln eine effiziente Tourenplanung und Routenoptimierung mit ArcView durchgeführt werden. Rasterdaten (digitale Kartenblätter) ermöglichen eine rasche Überprüfung der Adressenzuweisung im Straßennetz und eine gute graphische Veranschaulichung der gefundenen Rundreisen.

Das Konzept der Routenoptimierung ist flexibel und läßt sich im Prinzip auf beliebig große Flächen (z.B. ganz Berlin) skalieren. Die für das Testgebiet beschriebenen Maßnahmen zur Erzeugung realistischer Touren können in anderen Städten genauso angewendet werden. Für größere Anwendungen wäre zu überlegen, ob Informationen zu Straßen- und Verkehrsregeln bereits in digitaler Form vorliegen und käuflich erworben werden können (eine selbständige Datenerhebung ist nicht praktikabel). Des weiteren sollte eine konsistente Straßendatenbank mit möglichst redundanzfreier Datenhaltung benutzt werden.

Für die weg- oder zeitabhängige Optimierung der Kundentouren verwendet die Network Analyst-Extension ein Tabu Search-Verfahren. Andere heuristische Methoden sind hier ebenso denkbar wie exakte Lösungsverfahren und könnten die Ergebnisse in mancher Hinsicht noch verbessern (vgl. Kapitel 7.2.1).

Die Caritas-Sozialstation in Berlin-Tempelhof benutzt für ihre Tourenplanung seit längerem das Programmsystem „Profsys“. Im Unterschied zu ArcView GIS ist Profsys allerdings nicht für eine Routenoptimierung ausgelegt, sondern dient eher zur Haltung von Stammdaten und Abrechnungszwecken. Fragestellungen wie „welche Krankenschwester muß für welchen Kundenbesuch welche Arzneien mitnehmen“ werden dort gespeichert. Eine Einzelplatzlizenz für dieses System kostet 5.000 DM. Ein Mitarbeiter der Station berichtete mir, daß von zehn verschiedenen Programmsystemen, die beim Kauf zur Auswahl standen, nur ein einziges Ansätze einer Routenoptimierung aufwies.

Das Basismodul von ArcView GIS, welches keine Spezialsoftware, sondern ein vielseitig einsetzbares Informationssystem darstellt, kostet 3.350 DM. Für die Network-Analyst Extension kommen noch einmal 3.900 DM hinzu. Mit diesen beiden Programmen kann eine effiziente Tourenplanung und Routenoptimierung zu einem vertretbaren Preis durchgeführt werden.

11.2 Ausblick

Wünschenswert wäre eine kontinuierliche Erhöhung der Benutzerfreundlichkeit von ArcView GIS seitens der Firma ESRI sowie eine weitere Verbesserung der Routenoptimierung hinsichtlich Genauigkeit und Zuverlässigkeit. Die in Kapitel 7.5.2 behandelten Probleme mit deutschen Adressenformaten (z.B. 'Hausnummern des Berliner Typs') müßten für großflächige Anwendungen gelöst werden.

Die Entwicklung von Systemen, welche raumbezogene Daten organisieren, verarbeiten und graphisch präsentieren, ist noch lange nicht abgeschlossen. Im Gegenteil, durch die rapide Ausweitung des Internets ergeben sich auch für den GIS-Bereich ungeahnte Möglichkeiten. Zu nennen sind Themen wie Vernetzung von Rechnern, Datenbankanbindung oder Webmapping.

Im Falle des Caritas-Verbandes könnten die Rechner der 23 Berliner Bezirksstationen für eine gemeinsame Tourenplanung miteinander vernetzt werden und Zugriff auf einen zentralen Datenbank-Server erhalten. Über Applikationen, wie die in Kapitel 7.2.1 vorgestellte HTML Image Mapper - Extension, könnte eine interaktive Routenoptimierung etabliert werden, welche für alle Stationen jederzeit online abrufbar wäre.

Interessante Optionen für die Tourenplanung auf einem ganz anderen Feld erschließen sich durch die Nutzung des Globalen Positionierungssystems (GPS). GPS ermöglicht eine satellitengestützte Ortsbestimmung auf der Erdoberfläche und wird für Navigationsaufgaben in den unterschiedlichsten Bereichen angewendet (See-, Luft- und Straßenverkehr). So nutzen z.B. Taxiunternehmen dieses System seit längerem erfolgreich für die Auftragsplanung und den Einsatz von Fahrzeugen.

11.3 Danksagung

Dank sagen möchte ich an dieser Stelle Herrn Gerd Schulze vom Statistischen Landesamt Berlin und Herrn Michael Kirstein von der Caritas-Sozialstation in Berlin-Tempelhof. Ohne Ihre unbürokratische Hilfe bei der Bereitstellung des Datenmaterials wäre dieses Projekt nicht möglich gewesen.

Abbildungsverzeichnis

Das Verzeichnis gibt eine Übersicht über das verwendete Bildmaterial. Die Abbildungsnummern richten sich nach den Hauptkapiteln und werden fortlaufend beziffert. Neben den Nummern erscheinen die Seitenzahlen, auf welchen die Abbildungen zu finden sind und ggf. Quellenangaben. Es treten drei Sorten von Abbildungen auf :

- Abbildungen ohne Quellenangabe und ohne Zusatz wurden von mir erstellt.
- Abbildungen mit Quellenangabe (und ohne Zusatz) wurden im Original übernommen.
- Abbildungen mit Quellenangabe und dem Zusatz ’mod’ stammen von den angegebenen Autoren, wurden von mir aber modifiziert oder komplett neu gezeichnet.

Wenn die Abbildungen sich auf im Literaturverzeichnis angegebene Quellen beziehen, sind weitere Informationen dort zu entnehmen. Andernfalls wird die Quelle (z.B. URL) an dieser Stelle notiert.

Kapitel 2

Kapitel 3

Kapitel 4

Kapitel 5

Kapitel 6

Kapitel 7

Kapitel 8

Kapitel 9

Kapitel 10

Literaturverzeichnis

BARTELME, N. (1999)

GIS-Technologie - Geoinformationssysteme, Landinformationssysteme und ihre Grundlagen, Springer-Verlag, Berlin

BILL, R. (1999 A)

Grundlagen der Geo-Informationssysteme, Band 1: Hardware, Software und Daten, 2.Auflage, Wichmann Verlag, Karlsruhe

BILL, R. (1999 B)

Grundlagen der Geo-Informationssysteme, Band 2: Analysen, Anwendungen und neue Entwicklungen, 2. Auflage, Wichmann Verlag, Karlsruhe

BRAUSE, R. (1991)

Neuronale Netze – Eine Einführung in die Neuroinformatik, B. G. Teubner, Stuttgart

BORNDÖRFER, R., GRÖTSCHEL, M., LÖBEL, A. (1999)

Der Schnellste Weg zum Ziel, Artikel SC 99-32, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, Download via http://www.zib.de/bib/pub/pw/index.de.html

BRONSTEIN, I., SEMENDJAJEW, K. (1979)

Taschenbuch der Mathematik, Ergänzende Kapitel, 19. Auflage, Teubner Verlagsgesellschaft, Leipzig

DIEDRICH, J., HOWITZ, M. (1996)

Vortrag über Algorithmen und Komplexität, Fachbereich Informatik der FH Zittau / Görlitz, http://www.inf-gr.htwzittau.de/~wagenkn/TI/Komplexitaet/Referate98

DIESTEL, R. (1996)

Graphentheorie, Springer-Verlag, Berlin

DOMSCHKE, W. (1982)

Logistik: Rundreisen und Touren, R. Oldenbourg Verlag, München

DOMSCHKE, W., DREXL, A. (1991)

Einführung in Operations Research, 2. Auflage, Springer-Verlag, Berlin

DOMSCHKE, W. (1996 A)

Online-Hilfe zum Programm TENOR BABE, Version 1.0, Darmstadt

Download via http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/tenor/index.htm

DOMSCHKE, W. (1996 B)

Online-Hilfe zum Programm TENOR TSP, Version 1.0, Darmstadt

Download via http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/tenor/index.htm

DOMSCHKE, W., KLEIN, R., SCHOLL, A. (1996)

Tabu Search, c't - Zeitschrift für Computer und Technik, Heft 12 / 96, S. 326

Download via http://www.bwl.tu-darmstadt.de/bwl3/welcome.htm

ESRI (1996 A)

Environmental Systems Research Inst., ArcView GIS - User Manual, Redlands, USA

ESRI (1996 B)

Environmental Systems Research Institute, Avenue - Customization and Application Development for ArcView, Redlands, USA

ESRI (1996 C)

Environmental Systems Research Institute, ArcView Spatial Analyst - User Manual Advanced Spatial Analysis Using Raster and Vector Data, Redlands, USA

ESRI (1996 D)

Environmental Systems Research Institute, ArcView Network Analyst - User Manual Optimum Routing, Closest Facility and Service Area Analysis“, Redlands, USA

BUDNIK, A., FILIPOVA, T. (1996)

Elastic Net Method for the Traveling Salesman Problem, Java-Applet
http://nuweb.jinr.dubna.su/~filipova/tsp.html

GLEIßENBERGER, F., REICH, L. (1996)

Vortrag über Algorithmen und Komplexität, Fachbereich Informatik der FH Zittau / Görlitz
http://www.inf-gr.htwzittau.de/~wagenkn/TI/Komplexitaet/Referate98

GLEMSER, M. (1998)

ArcView GIS Projekthandbuch, Institut für Photogrammetrie, Universität Stuttgart
Download via http://www.ifp.uni-stuttgart.de/education/gis_praktikum

GRÖTSCHEL, M., PADBERG, M. (1999)

Die optimierte Odyssee, Spektrum der Wissenschaft, Heft 4/99, S. 76
http://www.spektrum.de/themen/heft210499.html

HEINE, Y., VOGT, A. (1996)

Vortrag über Algorithmen und Komplexität, Fachbereich Informatik der FH Zittau / Görlitz
http://www.inf-gr.htwzittau.de/~wagenkn/TI/Komplexitaet/Referate98

JAKIMOVSKI, P. (1999)

Proseminar Künstliche Intelligenz – Heuristische Suche
http://www.rz.uni-karlsruhe.de/~ueh5

LIEBIG, W. (1997)

Desktop-GIS mit ArcView, Leitfaden für Anwender, 1. Auflage, Wichmann Verlag

LACKA, E., PENIUTOWSKA, M. (1996)

Vortrag über Algorithmen und Komplexität, Fachbereich Informatik der FH Zittau / Görlitz
http://www.inf-gr.htwzittau.de/~wagenkn/TI/Komplexitaet/Referate98

MADEJ, A., SIEKIERKA, A. (1996)

Vortrag über Algorithmen und Komplexität, Fachbereich Informatik der FH Zittau / Görlitz
http://www.inf-gr.htwzittau.de/~wagenkn/TI/Komplexitaet/Referate98

OTTO, T. (1994)

Reiselust: Travelling Salesman – Eine neue Stratgie für eine alte Aufgabe,
c’t - Zeitschrift für Computer und Technik, Heft 1/94, S. 188

RECHENBERG, I (1994)

Evolutionsstrategie ’94, Frommann-Holzboog, Stuttgart

REEKEN, M., SCHOLZ, E., GROßER, B., FROMMER, A., KRIVSKY, S. (1998)

Das Königsberger Brückenproblem, Bergische Universität-Gesamthochschule Wuppertal
http://www.matheprisma.uni-wuppertal.de/Module/Koenigsb/index.htm

STADTINFO VERLAG (1998)

Autofahrer-Atlas mit Straßenverkehrsregeln, Hausnummernabschnitten etc.
12. Auflage, Kartenstand November 1997, StadtINFO Verlag GmbH, Berlin

SCHOLL, A., KRISPIN, G., KLEIN, R., DOMSCHKE, W. (1997)

Branch and Bound, c’t - Zeitschrift für Computer und Technik, Heft 10/97, S.336
Download via http://www.bwl.tu-darmstadt.de/bwl3/welcome.htm

STREIT, U., UHLENKÜKEN, C. (1999)

ArcView - Online Tutorial, Vers. 1.0, Institut für Geoinformatik, Universtät Münster
http://castafiore.uni-muenster.de/vorlesungen/av_tutor_e/frames/fsteuer.htm

WENDT, O. (1995 A)

COSA: COoperative Simulated Annealing, Institutsbericht (Zusammenfassung der Dissertation, s.u.), Download via http://caladan.wiwi.uni-frankfurt.de/IWI

WENDT, O. (1995 B)

Naturanaloge Verfahren zur approximativen Lösung kombinatorischer Optimierungsprobleme (Dissertation an der Universität Frankfurt am Main), erschienen unter dem Titel „Tourenplanung durch Einsatz naturanaloger Verfahren“ im Gabler-Verlag, Wiesbaden

WIRTH, N. (1986)

Algorithmen und Datenstrukturen mit Modula-2, 4. Auflage, B.G. Teubner, Stuttgart

ZELEWSKI, S. (1989)

Komplexitätstheorie als Instrument zur Klassifizierung und Beurteilung von Problemen des Operations Research, Friedr. Vieweg & Sohn, Braunschweig

Zell, A. (1996)

Simulation neuronaler Netze, 1. unveränderter Nachdruck, Addison-Wesley, Bonn

SKRIPT 1

```
' ------------------------------------------------------------
' Installationsskript für die Extension "CSS_Temphof.avx"
' ------------------------------------------------------------
' (basiert auf dem ESRI-Systemskript "Extension Builder")
' ------------------------------------------------------------

if (av.getproject=nil) then return(nil) end

' SELF ist das Äquivalent für die Extension
theDocs = SELF.get(0)
theControlList = SELF.get(1)
theMenuList = SELF.get(2)
theToolMenuList = SELF.Get(3)
theProject = Av.getproject

' Dokumente (Docs = Views, Tables etc.) hinzufügen
for each adoc in theDocs
 theProject.addDoc(adoc)
end

' Controls hinzufügen
for each totalControl in theControlList

' Control Liste
   acontrol = totalControl.get(0)

' Physical Control
   theControl = totalControl.get(1)

' Control Index
   theCindex = totalControl.get(2)

' DocGUI (Graphic User Interface) finden
   theControlDoc=av.getproject.findGUI(aControl.get(0))
     if (theControlDoc=NIL) then
       MsgBox.Warning("Das GUI "+aControl.get(0)+" kann im aktuellen Projekt nicht gefunden
       werden.","Fehler im Skript")
      return(nil)
  end

' Control Set finden
  thecommand="av.getproject.findGUI("""+aControl.get(0)+""").Get"+acontrol.get(1)
  thescript1=Script.Make(thecommand)
  thecontrolset=thescript1.doit("")

' Controls zum Control Set hinzufügen
  theControlSet.Add(theControl,theCindex)
end


' Menüs hinzufügen
for each totalcontrol in theMenuList


' Control Liste
   acontrol = totalControl.get(0)
      mDoc = acontrol.get(0)
      mMenu = acontrol.get(1)
      mMenuItem = acontrol.get(2)

' Physical Control
   theControl = totalControl.get(1)

' Control Index
   theCindex = totalControl.get(2)

' DocGUI finden
   theControlDoc = av.getproject.findGUI(aControl.get(0))
     if (theControlDoc=NIL) then
       MsgBox.Warning("Das GUI "+aControl.get(0)+" kann im aktuellen Projekt nicht gefunden
       werden.","Fehler im Skript")
      return(nil)
end
```

```
theMbar = av.getproject.findGUI(mDoc).GetMenuBar
  themenu = theMbar.findbylabel(mMenu)
  if (themenu = NiL) then
      themenu = menu.make
      themenu.setlabel(mMenu)
      theMbar.add(themenu,999)
  end
  themenu.add(thecontrol, theCindex)
end

' Werkzeug Menüs hinzufügen
for each totalControl in theToolMenuList
' Control Liste
   acontrol=totalControl.get(0)
' Physical Control
   theControl = totalControl.get(1)
' Control Index
   theCindex=totalControl.get(2)

' DocGUI finden
   theControlDoc=av.getproject.findGUI(aControl.get(0))
     if (theControlDoc=NIL) then
       MsgBox.Warning("Das GUI "+aControl.get(0)+" kann im aktuellen Projekt nicht gefunden
werden.","Fehler im Skript")
       return(nil)
    end

' Control Set finden
thecommand = "av.getproject.findGUI("""+aControl.get(0)+""").Get"+acontrol.get(1)
  thescript1 = Script.Make(thecommand)
  thecontrolset = av.getproject.findGUI(aControl.get(0)).GetToolBar

' Controls zum Control Set hinzufügen
  theControlSet.Add(theControl,theCindex)
end
av.getproject.setmodified(true)

' Skripte fügen sich selbst hinzu

' ----------------------------------------------------------------------------> ENDE
```

SKRIPT 2

```
' ------------------------------------------------------------------------
' ------------------------------------------------------------------------
' Skript zur Erstellung der Extension "CSS_Temphof.avx", welche die
' ArcView-Benutzeroberfläche an die Routenoptimierungs-Applikation anpasst.
'
' Benötigt werden dafür insgesamt 3 verschiedene Skripte:
' CSS_Temphof_Make   (erstellt die Extension)
' CSS_Temphof_Inst   (installiert die Extension)
' CSS_Temphof_Uninst (deinstalliert die Extension)
'
' Die drei Skripte basieren auf dem ESRI-Systemskript "Extension-Builder".
' ------------------------------------------------------------------------
' ------------------------------------------------------------------------

' Name der Extension (wird ins ArcView-Systemverzeichnis EXT32 geschrieben,
' wo sich auch alle anderen Extensions befinden)
theExtensionFile = "$AVEXT/CSS_Temphof.avx"

' Name der Extension, welche in der Liste der verfügbaren Extensions erscheint
theExtensionName = "Caritas-Sozialstation Tempelhof"

' Beschreibung der Extension, welche bei Aktivierung im Dialogfeld erscheint
theDescription = "Routenoptmierung für die Caritas-Sozialstation (CSS) in Berlin-Tempelhof.
Anpassung der ArcView-Benutzeroberfläche, so daß nur notwendige Menüs, Buttons, Tools,
Views und Themes erscheinen."

' Version der Extension
theVersion = 1.0

' Installationsskript
InScriptName = "CSS_Temphof_Inst"
```

```
' Deinstallationsskript
UnScriptName = "CSS_Temphof_Uninst"

' ----------------------------------------------------------------------
' Alle Dokumente (Views, Tables etc.), die eingebunden werden sollen.
theDocs = {"Bezirk Tempelhof und Ortsteil Lankwitz", "Testgebiet Tempelhof-
Nord","Einwohner", "Kunden_Mai", "Kunden_Sep", "Statistik_Mai_Adressen",
"Statistik_Mai_Strassen", "Turntable_t"}

' ----------------------------------------------------------------------
' Liste der Buttons und Tools, die erscheinen sollen.
' Muster: {GUIname, ControlType, ScriptName}
' e.g. theControlList = { {"View", "ButtonBar", "View.ZoomIn"},
'                         {"View", "ToolBar", "View.Identify"} usw.}
theControlList = { {"View", "ToolBar", "Infos_zur_Anwendung"},
                   {"View", "ToolBar", "Turntable_deklarieren"} }

' ----------------------------------------------------------------------
' Liste der Tool-Menüs, die erscheinen sollen.
' Tool menus are specified by giving a GUI name and a script name of one
' of the tools in the tool menu. Using this to locate the tool menu ALL
' other tools in that menu and their scripts will be extracted.
' e.g. theToolMenuList = {{"View", "View.PointTool"}}
theToolMenuList = {}

' ----------------------------------------------------------------------
' the Menus to include (a list of menu items listing the doc, the top
' menu, and the script for the menu item)
' the MenuList {Doc name, Main Menu name , menu item Scriptname}
' e.g. the MenuList = {{"View", "File", "View.Export"}}
theMenuList = {}

' ----------------------------------------------------------------------
'the scripts to include, not in controls or menus
'e.g. theScripts = {"My.Script","View.export"}
theScripts = {"Knoten_uebertragen"}

' ----------------------------------------------------------------------
' the Dependencies
theDependencies={}
' ----------------------------------------------------------------------

' ----------------------------------------------------------------------
' AB HIER NICHT MEHR EDITIEREN!!!
' ----------------------------------------------------------------------
' Liste mit den benötigten Skripten
theNeeded = {}
for each ControlScript in theControlList
 theNeeded.add(ControlScript.get(2))
end
for each MenuScript in theMenuList
 theNeeded.add(MenuScript.get(2))
end

Totalscripts = thescripts.merge(theNeeded)
TotalScripts.removeduplicates

TheInstall = av.GetProject.FindScript(InScriptName)
if (theInstall = NIL) then
     Msgbox.Error("Das Installationsskript '"+InScriptName+"' wurde nicht
     gefunden.","Skript nicht gefunden")
  return(nil)
end

theUninstall = av.GetProject.FindScript(UnScriptName)
if (theUninstall = NIL) then
   Msgbox.Error("Das Deinstallationsskript '"+UnScriptName+"' wurde nicht gefunden.","Skript
   nicht gefunden")
  return(nil)
end

' Erstellen der Extension
MyExt = Extension.Make(theExtensionFile.asFilename, theExtensionName, theInstall,
theUninstall, theDependencies)
TheDocList = {}
TheTotalControls = {}
TheTotalMenus = {}
TheTotalToolMenus = {}
```

```
' Dokumentenliste einlesen
for each aDoc in theDocs
  if ((av.getproject.findDoc(aDoc) = "Null").NOT) then
     theDocList.Add(av.GetProject.FindDoc(aDoc))
  else
       MsgBox.Warning("Das Dokument '"+adoc.asstring+"' wurde im aktuellen Projekt nicht
       gefunden","Fehler im Skript")
    return(nil)
 end
end

' Hinzufügen der Dokumentenliste
MyExt.add(theDocList)
' Control Liste prozessieren
' Controls {GUI, ControlType, Scriptname}

for each aControl in theControlList
  theControlDoc = av.getproject.findGUI(aControl.get(0))
    if (theControlDoc = NIL) then
       MsgBox.Warning("Das GUI '"+aControl.get(0)+"' konnte im aktuellen Projekt nicht
       gefunden werden.","Fehler im Skript")
      return(nil)
    end
  thecommand="av.getproject.findGUI("""+aControl.get(0)+""").Get"+acontrol.get(1)
' --- msgbox aktiviert ---
  msgbox.info(thecommand,"")
  thescript1=Script.Make(thecommand)
  thecontrolset=thescript1.doit("")
  theFoundControl=theControlSet.FindbyScript(aControl.get(2))
    if (theFoundControl=NIL) then
       MsgBox.Warning("Keine 'Controls' gefunden für das Skript '"+aControl.get(2)+"' in
       der "+acontrol.get(1),"Warnung")
      return(nil)
    end
    if (theFoundControl.is(ToolMenu)) then
       MsgBox.Warning("Das Skript '"+aControl.get(2)+"' in "+acontrol.get(1)+" ist für ein
       Tool-Menü gedacht.",           "Wird nicht unterstützt")
      return(nil)
    end

theTotalControls.Add({aControl,thefoundControl,theControlSet.getcontrols.
find(thefoundcontrol)})
end

' Zur Extension hinzufügen
MyExt.add(theTotalControls)

' Menü Liste {Doc, Menu, MenuScript}
for each aMenu in themenulist
  mDoc = aMenu.get(0)
  mMenu = aMenu.get(1)
  mScript = aMenu.get(2)

  themDoc = av.getproject.findGUI(mDoc)
  if (themDoc = NIL) then
       MsgBox.Warning("Das GUI '"+mDoc+"' konnte im aktuellen Projekt nicht gefunden
       werden.","Fehler im Skript")
      return(nil)
 end

 theMbar = av.getproject.findGUI(mDoc).GetMenuBar
 themenu = theMbar.findbylabel(mMenu)
  if (themenu = NiL) then
      MsgBox.Warning("Das Menü namens '"+mMenu+"' ist nicht vorhanden.","Fehler im Skript")
      return(nil)
  end

themenucontrol = themenu.findbyScript(mScript)
  if (themenucontrol = NiL) then
       MsgBox.Warning("Das Skript '"+mScript+"' wurde nicht gefunden.","Warnung")
       return(nil)
   end
  themenuitemidx=themenu.getcontrols.find(themenucontrol)
  thetotalmenus.add({amenu,theMenuControl,theMenuItemIdx})

end
```

```
myext.add(thetotalmenus)

' Tool Menu List prozessieren
for each aControl in theToolMenuList
  theControlDoc = av.getproject.findGUI(aControl.get(0))
    if (theControlDoc = NIL) then
       MsgBox.Warning("Das GUI '"+aControl.get(0)+"' konnte im aktuellen Projekt nicht
       gefunden werden.","Fehler im Skript")
      return(nil)
   end

  theControlSet = theControlDoc.getToolbar
  theFoundControl = theControlSet.FindbyScript(aControl.get(1))
    if (theFoundControl = NIL) then
       MsgBox.Warning("Keine 'Controls' gefunden für das Skript '"+aControl.get(1)+"' in
       der "+aControl.get(1),"Warnung")
      return(nil)
    end
    if (theFoundControl.is(ToolMenu).not) then
       MsgBox.Warning("Das Skript '"+aControl.get(1)+"' ist nicht als Tool-Menü
       verwendbar.", "Fehler durch den Anwender")
      return(nil)
  end

theTotalToolMenus.Add({aControl,theFoundControl,theControlSet.getcontrols.
find(thefoundcontrol)})

end

myext.add(thetotalToolmenus)

' Script Liste prozessieren
for each ascript in TotalScripts
  if ((av.getproject.findscript(ascript) = "Null").NOT) then
      myExt.Add(av.GetProject.FindScript(ascript))
  else
       MsgBox.Warning("Das Skript '"+ascript+"' konnte im aktuellen Projekt nicht gefunden
       werden.", "Fehler im Skript")
      return(nil)
 end
end

myExt.SetAbout(theDescription)
myExt.SetExtVersion(theVersion)
myExt.Commit

' ----------------------------------------------------------------------------> ENDE




SKRIPT 3

' ---------------------------------------------------------------
' Deinstallationsskript für die Extension "Caritas-Sozialstation"
' ---------------------------------------------------------------
' (basiert auf dem ESRI-Systemskript "Extension Builder")
' ---------------------------------------------------------------

' SELF ist das Äquivalent für die Extension

theDocs = SELF.get(0)
theControlList = SELF.get(1)
theMenuList = SELF.get(2)
theToolMenuList = SELF.get(3)
theProject = Av.getproject

' Dokumente (Docs = Views, Tables etc.) entfernen
for each adoc in theDocs
 If (theProject.finddoc(adoc.getname)<>NIL) then
       TheAnswer = msgbox.yesno("Entfernen von Dokument     "+adoc.getname+"?", "Dokument
       entfernen?", TRUE)
  if (theAnswer = TRUE) then theProject.RemoveDoc(adoc)

  end
 end
end
```

```
' Controls entfernen
for each totalControl in theControlList
' Control List von der Extension holen
   acontrol=totalControl.get(0)
' Physical Control holen
   theControl = totalControl.get(1)
' Control Index holen
   theCindex=totalControl.get(2)

' DocGUI (Graphic User Interface) für Controls finden
   theControlDoc=av.getproject.findGUI(aControl.get(0))
     if (theControlDoc=NIL) then
       MsgBox.Warning("Das GUI "+aControl.get(0)+" konnte im aktuellen Projekt nicht
gefunden werden.","Fehler im Skript")
       return(nil)
     end

' Den adäquaten Control Set finden
thecommand="av.getproject.findGUI("""+aControl.get(0)+""").Get"+acontrol.get(1)
   thescript1=Script.Make(thecommand)
   thecontrolset=thescript1.doit("")

' Nachgucken, ob der Control Set in diesem Set ist, wenn ja: entfernen
   if (theControlSet.GetControls.find(theControl)<>NIL) then
      theControlSet.remove(theControl)
      if (thecontrol = "ToolBar") then
        theControlSet.selectdefault

        end
    end
end

' Menüs entfernen
for each totalcontrol in theMenuList

' Control Liste
    acontrol = totalControl.get(0)
      mDoc = acontrol.get(0)
      mMenu = acontrol.get(1)
      mMenuItem = acontrol.get(2)

' Physical Control
   theControl = totalControl.get(1)

' Control Index
   theCindex = totalControl.get(2)

' DocGUI finden
   theControlDoc = av.getproject.findGUI(aControl.get(0))
     if (theControlDoc = NIL) then
       MsgBox.Warning("Das GUI "+aControl.get(0)+" konnte im aktuellen Projekt nicht
       gefunden werden.", "Fehler im Skript")
       return(nil)
     end

  theMbar = av.getproject.findGUI(mDoc).GetMenuBar
  themenu = theMbar.findbylabel(mMenu)
  if (themenu = NiL) then
       MsgBox.Warning("Das Menü namens "+mMenu+" ist nicht vorhanden.", "Fehler  im
       Skript")
' --- return(nil) aktiviert ---
       return(nil)
  else
      thething = themenu.getcontrols.find(thecontrol)

  if (thething <> NIL) then
      themenu.remove(thecontrol)
    end

' --- msgbox aktiviert ---
     msgbox.info(themenu.GetControls.count.asstring,"")
     if (themenu.GetControls.count<1) then
      theMbar.remove(themenu)
    end
  end
end
```

```
for each totalControl in theToolMenuList
' Control Liste von der Extension holen
   acontrol = totalControl.get(0)

' Physical Control holen
   theControl = totalControl.get(1)

' Control Index holen
   theCindex = totalControl.get(2)

' DocGUI für Controls finden
   theControlDoc = av.getproject.findGUI(aControl.get(0))
     if (theControlDoc = NIL) then
        MsgBox.Warning("Das GUI "+aControl.get(0)+" konnte im aktuellen Projekt nicht
        gefunden werden.", "Fehler im Skript")
       return(nil)
     end

' Den adäquaten Control Set finden
   thecontrolset = av.getproject.findGUI(aControl.get(0)).GetToolBar

' Nachgucken, ob der Control Set in diesem Set ist, wenn ja: entfernen
    if (theControlSet.GetControls.find(theControl)<>NIL) then

      theControlSet.remove(theControl)
      theControlSet.selectdefault

   end
end

' Skripte löschen sich selbst
av.getproject.setmodified(true)

' ------------------------------------------------------------------------------> ENDE
```

SKRIPT 4

```
' ---------------------------------------------------------------------
' Skript zur Übertragung von Knoten- und Verteilung von Kantennummern
' (basiert auf dem ESRI-Systemskript "Copying over Node Numbers")
' ---------------------------------------------------------------------

' Dieses Skript überträgt die Knotennummern aus der Datei nodes.dbf (liegt
' im Verzeichnis strassen_t.nws) in die Attribut-Tabelle des Netzwerkthemas
' (hier: "Strassen-Netz") und verteilt die dazugehörigen Kantennummern.
' Zu der Tabelle werden die Felder Edge#, Fjunction und Tjunction  hinzugefügt.
' Damit das Skript korrekt ausgeführt werden kann, muss für das Netzwerkthema
' ein Netzwerk-Index-Verzeichnis existieren.

' View und Netzwerkthema benennen

aView = av.GetProject.FindDoc("Testgebiet Tempelhof-Nord")
aNetworkTheme = aView.FindTheme("Strassen-Netz")
aNetworkThemeFTab = aNetworkTheme.GetFTab

' nodes.dbf einlesen, VTab Objekt erstellen und Felder initialisieren

aNetworkIndexDir = aNetworkTheme.AsString.Substitute("shp", "nws")
aNodeFile = FN.Merge(aNetworkIndexDir, "nodes.dbf")
aNodeVTab = VTab.Make(aNodeFile, false, false)
aFjunction = aNodeVTab.FindField("Fjunction")
aTjunction = aNodeVTab.FindField("Tjunction")

' Felder Edge#, Fjunction und Tjunction zum Netzwerkthema FTab hinzufügen
' Feldlänge: 3 Integer-Zahlen, Genauigkeit: 0 Stellen nach dem Komma

aEdgeField = Field.Make("Edge#", #FIELD_LONG, 3, 0)
aFjunctionField = Field.Make("Fjunction", #FIELD_LONG, 3, 0)
aTjunctionField = Field.Make("Tjunction", #FIELD_LONG, 3, 0)
aFieldList = {aEdgeField, aFjunctionField, aTjunctionField}
aNetworkThemeFTab.SetEditable(True)
aNetworkThemeFTab.AddFields(aFieldList)
```

```
' mit nodes.dbf die Felder Edge#, Fjunction und Tjunction popularisieren

Count = 0
for each r in aNetworkThemeFTab
  aFromNodeNumber = aNodeVTab.ReturnValueNumber(aFjunction, Count)
  aToNodeNumber = aNodeVTab.ReturnValueNumber(aTjunction, Count)
  aNetworkThemeFTab.SetValueNumber(aFjunctionField, r, aFromNodeNumber)
  aNetworkThemeFTab.SetValueNumber(aTjunctionField, r ,aToNodeNumber)
  Count = Count + 1
  aNetworkThemeFTab.SetValueNumber(aEdgeField, r, Count)
end

aNetworkThemeFTab.SetEditable(False)

' ------------------------------------------------------------------------> ENDE
```

SKRIPT 5

```
' ------------------------------------------------------------
' Skript zur Deklarierung eines Turntables
' (basiert auf den ESRI-Systemskript "Declaring a Turntable")
' ------------------------------------------------------------

' Dieses Skript deklariert einen Turntable, d.h. der Network-Analyst- Extension werden die
' Zeiten für Abbiegevorgänge und verbotene Turns mitgeteilt. Nach jeder Änderung des
' Turntables muss das Skript erneut ausgeführt werden (Button 'T' in der Tool-Leiste).
' Für die Ausführung muss ein Projekt mit Turntable, View und Netzwerkthema geöffnet sein.

' View und Netzwerkthema benennen

aView = av.GetProject.FindDoc("Testgebiet Tempelhof-Nord")
aNetworkTheme = aView.FindTheme("Strassen-Netz")

' Netwerk-Definitions-Objekt erstellen

aNetwork = av.Run("Network.GetNetwork",{aNetworkTheme})
aNetDef = aNetwork.GetNetDef

' Turntable einlesen und deklarieren

aVTab = av.GetProject.FindDoc("Turntable_t").GetVTab
aDeclaredTurntable = aNetDef.SetTurnVTab(aVTab)

' Sicherstellen, dass der Turntable korrekt deklariert wurde

if (aDeclaredTurntable) then
  MsgBox.Info("Turntable wurde korrekt deklariert.", "Turntable Status")

' Fehlermeldung, wenn etwas schief gegangen ist

else
   MsgBox.Info("Turntable konnte nicht deklariert werden.", "Turntable  Status")
end

' ------------------------------------------------------------------------> ENDE
```