ulm university universität
uulm

Universität Ulm
Fakultät für Mathematik und
Wirtschaftswissenschaften

Vorhersage von Reisezeiten in städtischen Ballungsräumen

Diplomarbeit
in Wirtschaftsmathematik

vorgelegt von
Johannes Renfordt
am 26.6.2007

Gutachter

Jun.-Prof. Dr. Evgeny Spodarev
Prof. Dr. Franz Schweiggert

Danksagung

An dieser Stelle möchte ich mich bei allen, die zum Gelingen dieser Arbeit auf verschiedenste Weisen beigetragen haben, bedanken.

Allen voran danke ich Herrn Jun.-Prof. Dr. Evgeny Spodarev für die gelungene Betreuung der Arbeit. Darüberhinaus hat er mich am wissenschaftlichen Leben seines Instituts teilhaben lassen.

Als zweiter Gutachter hat sich Herr Prof. Dr. Franz Schweiggert zur Verfügung gestellt. Dafür gilt ihm mein herzlicher Dank.

Ich bedanke mich bei den Mitarbeitern des Instituts für Stochastik sowie des Instituts für Allgemeine Informationsverarbeitung. Sie standen mir in vielen Aspekten der Arbeit hilfreich zur Seite.

Meine Familie sowie viele Freunde und Bekannte haben mich nicht nur während der Diplomarbeitet unterstützt, sondern mich durch mein ganzes Studium begleitet. Ohne sie alle wäre vieles nicht möglich gewesen.

Besonderen Dank schulde ich Herrn Dr. Harald Bauer für seine Veranstaltung „Operations Research Praktikum“ im Wintersemester 2005 an der Universität Ulm. Ohne die dort angesprochenen Themen und Ideen hätte die vorliegende Arbeit sicher ganz anders ausgesehen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Die vorliegende Arbeit ist eingebunden in ein Projekt des Instituts für Stochastik an der Universität Ulm mit der Arbeitsgruppe Verkehrsstudien des Deutschen Zentrums für Luft- und Raumfahrt. Einen Überblick über das Projekt ist unter [geo] zu finden.
Viele Autofahrer benutzen heutzutage bereits Navigationssysteme. Diese bestehen aus zwei Komponenten: zum einen wird die aktuelle Position näherungsweise durch Satellitennavigationssysteme bestimmt, oft mit dem Global Positioning System, kurz GPS. Weiterhin sind in die Geräte Routing-Systeme integriert, die dem Autofahrer Aufgaben abnehmen: sie liefern eine Fahrtroute zu einem Ziel nach dem vom Anwender vorgegebenen Kriterien. Solche Kriterien könnten beispielsweise die kürzeste oder schnellste Route sein. Der Blick in den Atlas entfällt und auch das Suche nach kleinen Straßen in großen Städten kann entfallen: die in den Geräten enthaltenen Karten decken je nach Speichergröße oft ganze Länder vollständig ab.
Diese Arbeit stellt eine Möglichkeit vor, wie man solche Routenempfehlungen berechnen kann, nämlich durch den Ameisenalgorithmus. Dieser Algorithmus besitzt eine heuristische Komponente, d.h. er basiert auf Erfahrungen mit gleichgelagerten Problemstellungen. Eine Heuristik sagt aus, welche Eigenschaften eine gültige Lösung einer Problemstellung haben muss, damit sie typischerweise eine *gute* Lösung darstellt. Das Ziel von heuristischen Problemlösungen ist es, den Berechnungsaufwand für Lösungen deutlich zu reduzieren und dabei trotzdem akzeptable Lösungen zu erreichen.
Ameisenalgorithmen konstruieren Routen in Graphen. Die Entscheidung, welche Kanten in die Route aufgenommen werden, erfolgt dabei einerseits nach der heuristischen Information, andererseits nach den Erfahrungen zuvor konstruierter Routen. Durch wiederholte Konstruktion und Auswertung von Routen kann man so i.a. zu guten Lösungen des Problems gelangen. Dies wird beispielhaft an Fahrtempfehlungen im Innenstadtbereich von Berlin demonstriert.
In einem zweiten Teil der Arbeit wird der Einsatz von Ameisenalgorithmen in der Ver-

kehrssimulation demonstriert. Ein Datensatz vorhandener Verkehrsdaten, die von Taxis in Berlin aufgezeichnet wurden, steht zur Verfügung und wird mit simulierten Verkehrsdaten verglichen. Eine Anwendung dieser Simulation stellt folgendes Szenario dar: Autofahrer mit neuartigen Navigationssystemen bestimmen in regelmäßigen Abständen ihre Position, Geschwindigkeit und Fahrtrichtung und stellen diese Informationen einem zentralen Serversystem zur Verfügung. In diesem zentralen System werden nun die jeweiligen Daten extrapoliert, d.h. eine mögliche Fortsetzung der Fahrtroute und dort auftretende Geschwindigkeiten werden errechnet. Die gewonnenen Informationen können nun mit denen vieler anderer Autofahrer verknüpft und so zeitnah mögliche Staus erkannt werden. Der Server stellt nun seine errechneten zukünftigen Geschwindigkeiten den einzelnen Teilnehmern am System zur Verfügung, die anhand dieser Daten ihre individuelle Fahrtroute ggf. neu berechnen. Ein sich aus der Zusammenfassung der Daten ergebender potentieller Staupunkt kann somit entschärft werden und tritt möglicherweise gar nicht tatsächlich auf.

1.2 Gliederung

Im folgenden Kapitel werden einige Begriffe aus der Graphentheorie definiert und kurz erläutert, die später gebraucht werden.
Im dritten Kapitel wird anhand von Beispielen in die Probleme eingeführt, deren Ziel das Bestimmen eines kürzesten Weges ist. Manche dieser Probleme sind bislang nur besonders schwer lösbar, da sie zu den sogenannten NP-vollständigen Problemen gehören. Als Beispiel für ein solches Problem dient das Travelling Salesman Problem. Als ein Mittel der leichteren Berechnung von guten Lösungen von schwer lösbaren Problemen werden (Meta-) Heuristiken eingeführt.
Es folgt im vierten Kapitel eine Einführung in die Metaheuristik *Ameisenalgorithmus*. Dieser Ansatz entstammt der Natur: Ameisenstraßen sind oft vergleichsweise gerade und führen mehr oder weniger direkt auf das Ziel zu. Das Verhalten der Ameisen wird in ein Modell übertragen und kann auf verschiedene Kürzeste-Wege-Probleme angewendet werden. Am Beispiel des Travelling Salesman Problems wird der Ameisenalgorithmus genauer betrachtet.
Das fünfte Kapitel beschreibt die von den Taxen gesammelten Daten und stellt erste Auswertungen vor. Diese werden im folgenden sechsten Kapitel zur Bestimmung kürzester Wege durch den Ameisenalgorithmus aufgegriffen und verwendet. Dort wird auch

kurz auf die vorgenommene Implementierung eingegangen.
Die Simulation von Verkehrsdaten, die im siebten Kapitel beschrieben wird, basiert wiederum auf der Bestimmung kürzester Wege durch den Ameisenalgorithmus. Die erzeugten Daten werden mit den Originaldaten verglichen.
Das achte Kapitel rundet mit einem Ausblick über Erweiterungsmöglichkeiten die Arbeit ab.

2 Graphentheoretische Einführung

Unter Zuhilfenahme der Graphentheorie kann man Verkehrsnetze, die später untersucht werden, mathematisch beschreiben. In diesem Kapitel wird deshalb ein kleiner Grundstock an Begrifflichkeiten definiert, der beim Verständnis späterer Kapitel hilfreich sein wird.
Die folgenden Definition lehnen sich stark an [Vol91] an.

Grundlegende Definitionen

Zuallererst steht die Definition eines Graphen und weiterer grundlegender Begriffe an:

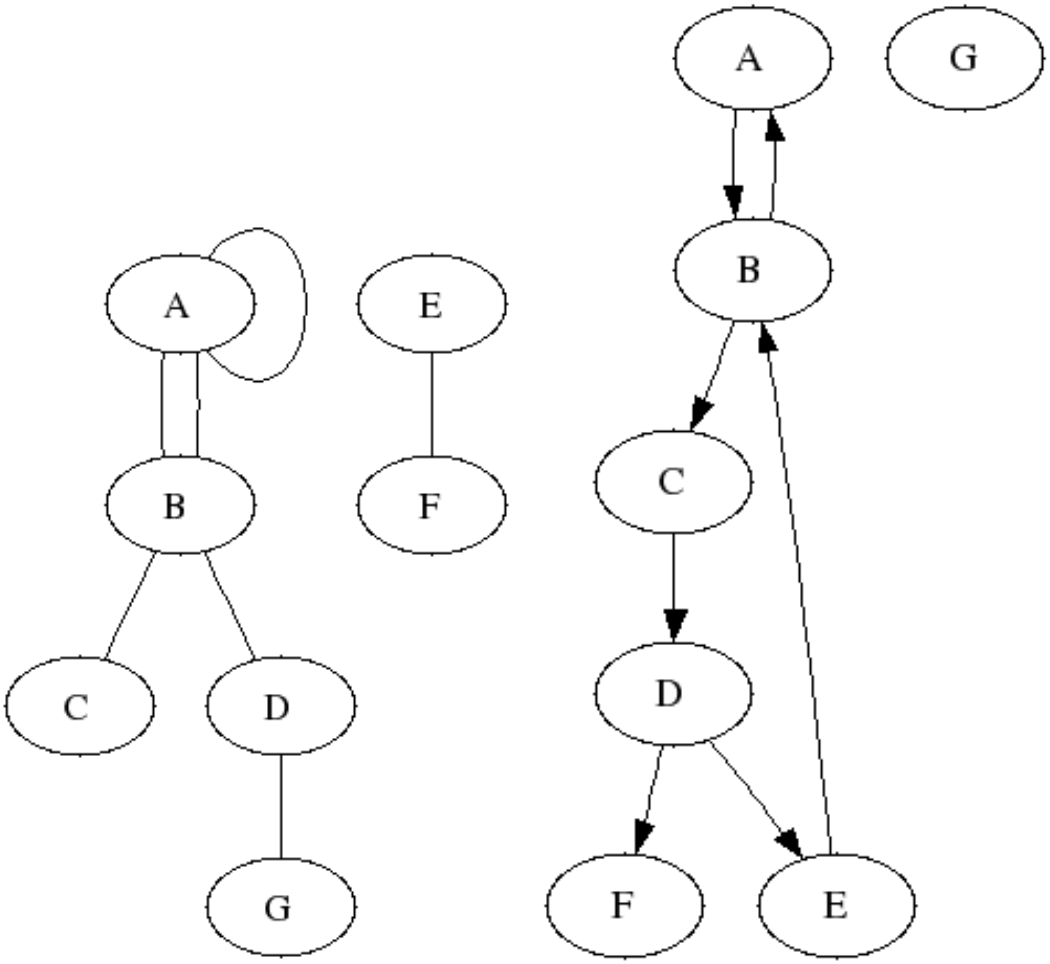
Definition 1 (Graph) *Seien E, K zwei nicht leere Mengen mit $E \cap K = \emptyset$ und $P_2(E) = \{X | X \subseteq E \text{ mit } 1 \leq |X| \leq 2\}$. Ist $g : K \to P_2(E)$ eine Abbildung, so heißt (E, K, g) ein (ungerichteter) Graph.*
Die Menge $E = E(G)$ wird Eckenmenge des Graphen genannt, ihre Elemente Ecken. Entsprechend heißt $K = K(G)$ Kantenmenge des Graphen und besteht aus den sogenannten Kanten.
Ist $E = K = \emptyset$, so ist der Graph ein leerer Graph.
Ein Nullgraph hingegen liegt vor, wenn $K = \emptyset$, aber $E \neq \emptyset$ gilt.

Schreibweise 1 *Im Folgenden wird für ungerichtete Graphen die Schreibweise*

$$G := (E, K, g) = (E(G), K(G)) = (E, K)$$

verwendet.

Definition 2 (Lagebeziehungen im Graphen) *Sei $G = (E, K)$ ein ungerichteter Graph. Ist $k \in K$ mit $g(k) = \{x, y\}$, so heißen x und y Endpunkte der Kante k; die Kante k inzidiert die Ecken x und y.Gilt $x = y$, so heißt k Schlinge.*
Zwei Ecken werden als benachbart bezeichnet, falls sie durch eine gemeinsame Kante verbunden werden. Besitzt eine Ecke keine benachbarte Ecken, so heißt sie isolierte Ecke.



(a) Beispiel eines ungerichteten Graphen (b) Beispiel eines gerichteten Graphen

Abbildung 2.1: Beispiele für Graphen

Definition 3 (schlichte Graphen, vollständige Graphen) *Werden in einem Graphen $G = (E, K)$ je zwei Ecken durch höchstens eine Kante verbunden, so wird der Graph schlicht genannt.*
Sind in einem schlichten Graphen alle Ecken paarweise benachbart, so heißt dieser auch vollständiger Graph.

Abbildung 2.1(a) zeigt ein Beispiel eines ungerichteten Graphen. Da dort nicht alle Ecken benachbart sind, ist der Graph nicht vollständig. Weiterhin ist er nicht schlicht, da die Ecken A und B durch zwei Kanten verbunden sind. Die Ecke A ist zudem durch eine Schlinge mit sich selbst verbunden. Eine isolierte Ecke gibt es nicht.

Definition 4 (gerichteter Graph) *Seien E, B zwei nicht leere Mengen mit $E \cap B =$*

$\emptyset$, sei $h : B \to E \times E$ eine Abbildung. Man nennt dann (E, B, h) einen gerichteten Graphen oder auch Digraphen. Wie zuvor werden die Elemente von E Ecken genannt, di Elemente von B heißen Bögen.

Bemerkung 1 *Bögen werden oft auch gerichtete Kanten genannt. Die Begriffe der Ecken- und Kantenmengen sowie die Definition des leeren Graphen und des Nullgraphen kann man ohne weiteres von ungerichteten Graphen auf Digraphen übertragen.*

Schreibweise 2 *Für Digraphen wird auch die Schreibweise*

$$D = (E, B, h) = (E(B), B(D)) = (E, D)$$

verwendet.

Definition 5 (Lagebeziehungen im Digraphen) *Sei $D = (E, B)$ ein gerichteter Graph. Ist $k \in B$ mit $h(k) = (x, y)$ eine Kante des Graphen, so heißt x der Anfangs- und y der Endpunkt des Bogens k. Analog zum ungerichteten Graphen heißt eine Kante Schlinge, wenn x mit y übereinstimmt.*
Ist eine Ecke weder Anfangs- noch Endpunkt aller Kanten der Kantenmenge, dann wird diese Ecke als isolierte Ecke bezeichnet.

Abbildung 2.1(b) auf der vorherigen Seite zeigt ein Beispiel eines gerichteten Graphen. Im Gegensatz zum ersten Beispiel eines ungerichteten Graphen gibt es nun eine isolierte Ecke, nämlich die Ecke G. Die Richtung einer Kante wird durch die Pfeilspitze gekennzeichnet: beispielsweise die Kante, die die Ecken D und F verbindet, geht von D aus und führt zu F. Eine Kante in umgekehrter Richtung gibt es jedoch nicht.

Definition 6 (endliche Graphen) *Ein Graph G bzw. Digraph D heißt endlich, falls $|E(G)| < \infty$ und $|K(G)| < \infty$ bzw. falls $|E(D)| < \infty$ und $|B(D)| < \infty$.*

In der Folge wird immer von einem endlichen (Di-) Graphen ausgegangen.

Beschreibung eines Graphen

Zur nicht-graphischen Darstellung eines Graphen kann man Adjazenzmatrizen verwenden:

Definition 7 (Adjazenzmatrix) *Sei $G = (E, K)$ ein Graph mit $E = \{x_1, \ldots, x_n\}$ und $K = \{k_1, \ldots, k_m\}$. Weiter bezeichne $m_G(x_i, x_j) = m(x_i, x_j)$ die Anzahl der Kanten, die x_i und x_j verbinden, wobei Schlingen doppelt gezählt werden. Dann bezeichnet man die $n \times n$ Matrix*

$$A_G = A = (m(x_i, x_j))_{i,j=1}^{n}$$

als Adjazenzmatrix des Graphen G.
Sei $D = (E, B)$ ein Digraph mit $E = \{x_1, \ldots, x_n\}$ und $B = \{k_1, \ldots, k_m\}$. Mit $m_D(x_i, x_j)$ sei die Anzahl der Bogen von x_i, nach x_j bezeichnet, wobei Schlingen einfach gezählt werden. Dann bezeichnet man die $n \times n$ Matrix

$$A_D = A = (m_D(x_i, x_j))_{i,j=1}^{n}$$

als Adjazenzmatrix des Digraphen D.

Bemerkung 2 *Besitzt ein (Di-) Graph vergleichsweise wenig Kanten, ist also die Adjazenzmatrix nur dünn besetzt, kann man die Kanten zur einfacheren Darstellung auch in Listenform angeben.*

Wege in einem Graphen

Definition 8 (Kantenfolge) *Sei $G = (E, K, g)$ ein Graph und $k_1, \ldots, k_p \in K$ dergestalt, dass $g(k_i) = \{a_{i-1}, a_i\}$ für $i = 1, \ldots, p$ gilt. Dann heißt $(k_1, \ldots, k_p)$ Kantenfolge von a_0 nach a_p der Länge p. Dabei wird a_0 der Anfangs- und a_p der Endpunkt der Kantenfolge genannt.*
Eine Kantenfolge wird als geschlossen bezeichnet, falls $a_0 = a_p$. Falls die Gleichheit nicht gilt, also $a_0 \neq a_p$, heißt die Kantenfolge offen.
Mit $K((k_1, \ldots, k_p))$ seien die Kanten der Kantenfolge $(k_1, \ldots, k_p)$ bezeichnet.

Schreibweise 3 *Für die Kantenfolge $(k_1, \ldots, k_p)$ kann man auch schreiben:*

$$Z = (k_1, \ldots, k_p) = (a_0, \ldots, a_p) = (a_0, k_1, a_1, \ldots, k_p, a_p).$$

Definition 9 (Kantenzug, Weg) *Sind in einer Kantenfolge alle Kanten paarweise verschieden, so spricht man von einem Kantenzug. Gilt weiter, dass alle Ecken eines Kantenzuges paarweise verschieden sind, dann wird der Kantenzug Weg genannt.*

Diese Definitionen von Kantenfolgen, -zügen und Wegen lassen sich auch auf Digraphen übertragen.
Anhand der Abbildung 2.1(a) auf Seite 5 ein kleines Beispiel: Die Kantenfolge (A, A, B, C) ist ein Kantenzug, da keine Kante mehrfach auftritt, jedoch kein Weg, da die Ecke A zweimal besucht wird. Die Kantenfolge (A, B, D, G) hingegen ist ein Weg.

Satz 1 *Sei $D = (E, K)$ ein Graph, sei Z eine offene Kantenfolge in D von a_0 nach a_p. Dann existiert ein Weg W von a_0 nach a_p mit $K(W) \subseteq K(Z)$.*

Anstelle eines Beweises soll hier ein praktisches Beispiel ausreichen:
Enthält eine Kantenfolge eine Ecke zweifach, kann man die Elemente, die zwischen dieser zweifach vorkommenden Ecke auftreten, entfernen. So kann man z.B. aus der Kantenfolge $(1, 2, 3, 4, 5, 2, 6)$ den Weg $(1, 2, 6)$ konstruieren. Kommt eine Ecke mehr als zweimal vor, entfernt man alle Elemente zwischen dem ersten und letzten Auftreten dieser Ecke. Falls eine Kantenfolge mehrere Ecken mehrfach enthält, ist der entstehende Weg nicht mehr notwendigerweise eindeutig. Am Beispiel $(1, 2, 3, 4, 5, 2, 5, 6, 7)$ wird deutlich, dass sowohl $(1, 2, 5, 6, 7)$ als auch $(1, 2, 3, 4, 5, 6, 7)$ Wege sind, die aus der ursprünglichen Kantenfolge entstehen können.

Definition 10 (Zusammenhangskomponenten I) *Sei $G = (E, K)$ ein Graph.*
Dann werden zwei Ecken x und y zusammenhängend genannt, falls ein Weg von x nach y in G existiert. Dadurch wird auf den Ecken des Graphen $E(G)$ eine Äquivalenzrelation definiert. Jeder von einer Äquivalenzklasse induzierte Teilgraph heißt Zusammenhangskomponente des Graphen G.
Durch $\kappa := \kappa(G)$ wird die Anzahl der Zusammenhangskomponenten des Graphen G bezeichnet. Gilt $\kappa = 1$, so wird der jeweilige Graph als zusammenhängend bezeichnet.

Im Beispielgraph aus Abbildung 2.1(a) auf Seite 5 gibt es zwei Zusammenhangskomponenten: die Ecken E und F sind von den übrigen separiert, jedoch untereinander verbunden. Der Graph ist also nicht zusammenhängend. Ebenso sind die Ecken A, B, C, D, G untereinander verbunden. Es gilt also $\kappa(Beispielgraph) = 2$.
Für Digraphen kann man ähnliche Begriffe definieren:

Definition 11 (Zusammenhangskomponenten II) *Sei $D = (E, B)$ ein Digraph, seien $x, y \in E$ zwei Ecken. Falls ein orientierter Weg mit der Anfangsecke x und Endecke y existiert, so wird y als von x aus erreichbar bezeichnet.*
Ist sowohl x von y als auch y von x aus erreichbar, dann werden die beiden Ecken x und y als stark zusammenhängend bezeichnet.

Analog zum Fall des ungerichteten Graphen ist der starke Zusammenhang eine Äquivalenzrelation auf $E(D)$. Bezeichnet man durch E_i die disjunkten Äquivalenzklassen, so wird der Teilgraph $D[E_i]$ - d.h. D wird auf die Ecken, die in E_i zu liegen kommen und die Bögen, die nur Ecken aus E_i verbinden, eingeschränkt - eine starke Zusammenhangskomponente von D genannt. Falls es für den Digraphen D nur eine starke Zusammenhangskomponente gibt, dann wird dieser Digraph auch als stark zusammenhängend bezeichnet.

Die folgende Definition einer Brücke ist in der Form nur auf ungerichtete Graphen anwendbar.

Definition 12 (Brücke) *Sei $G = (E, K)$ ein ungerichteter Graph. Eine Kante $k \in K(G)$ heißt Brücke, falls $\kappa(G) < \kappa(G - k)$.*

Wie bereits oben erwähnt, ist der Graph aus Abbildung 2.1(a) auf Seite 5 nicht zusammenhängend, es gilt $\kappa(Beispielgraph) = 2$. Entfernt man nun die Kante zwischen D und G, gibt es eine isolierte Ecke, nämlich G. Damit erhöht sich die Anzahl der Zusammenhangskomponenten auf drei, die Kante (D, G) ist also eine Brücke.
Zwar ist der Begriff Brücke auf Digraphen nicht anwendbar, aber man kann sich trotzdem ein Analogon vorstellen. Durch das Entfernen einer Brücke aus einem Teilgraphen ist der Zusammenhang des Teilgraphen nicht mehr gegeben. Im obigen Beispiel ist der Teilgraph $(E(Beispielgraph) - \{E, F\}, K(Beispielgraph) - \{(E, F)\})$ zusammenhängend. Nach der Entfernung der Kante zwischen den Ecken D und G ist er es nicht mehr. Ähnlich verhält es sich im gerichteten Graphen aus Abbildung 2.1(b) auf Seite 5: Dort ist der Teilgraph $(E(Beispielgraph) - \{F, G\}, K(Beispielgraph) - \{(D, G)\}$ stark zusammenhängend. Entfernt man aus diesem Teilgraphen eine beliebige Kante, geht der starke Zusammenhang verloren. Die entfernte Kante hat also eine Funktion ähnlich der einer Brücke in einem ungerichteten Graphen inne.

Bewertete Graphen

Bisher wurden nur (Di-) Graphen betrachtet, deren Kanten abgesehen von den Ecken, die sie verbinden, keinerlei Informationen trugen. In der folgenden Definition wird eine Verallgemeinerung eingeführt.

Definition 13 (Bewerteter Graph) *Sei $G = (E, K)$ ein Graph und $\varrho : K \to \mathbf{R}$ eine Abbildung. Dann wird $G = (E, K, \varrho)$ ein bewerteter Graph und $\varrho(k)$ die Bewertung oder*

Länge einer Kante $k \in K(G)$ genannt.
Ist $k \notin K(G)$, so definiert man $\varrho(k) = \infty$.

Die Bezeichner Bewertung und Länge lassen schon erkennen, dass sie Begriffe der Anwendungen widerspiegeln: die Länge einer Kante mag man mit der Entfernung oder dem zeitlichen Abstand der verbundenen Ecken assoziieren, die Bewertung einer Kante kann als Kosten der Nutzung einer Kante interpretiert werden. In manchen Fällen wird man die Bewertung der Kanten einschränken, z.B. auf $\mathbf{R}_+$. Je nach Anwendung mag das verschieden ausfallen bzw. ineinander übergehen: Kosten können als negative Gewinne interpretiert werden usw.
Für schlichte bewertete Graphen kann man eine Bewertungsmatrix mit den Informationen über die Bewertungen der Kanten darstellen. Sie ist ähnlich zur Adjazenzmatrix:

Definition 14 (Bewertungsmatrix) *Sei $G = (E, K, \varrho)$ ein schlichter bewerteter Graph mit $E = \{x_1, \ldots, x_n\}$. Die quadratische $n \times n$ Matrix*

$$B_G = B = (\varrho(x_i x_j))_{i,j=1}^{n}$$

wird Bewertungsmatrix des Graphen G genannt. Dabei soll $\varrho(x_i x_j)$ die Bewertung der Kante sein, die x_i mit x_j verbindet, falls die beiden Ecken adjazent sind. Ist dies nicht der Fall, wird der entsprechende Eintrag auf ∞ gesetzt.

Analog kann man die Bewertungsmatrix auch für schlichte Digraphen definieren. Diese wird dann aber in aller Regel nicht mehr symmetrisch sein.

Definition 15 (Länge einer Kantenfolge) *Sei $G = (E, K, \varrho)$ ein bewerteter Graph, sei $(k_1, \ldots, k_p)$ eine Kantenfolge in G. Dann wird durch*

$$\varrho(k_1, \ldots, k_p) := \sum_{i=1}^{p} \varrho(k_i)$$

die Länge der Kantenfolge definiert.

Definition 16 (Abstand zweier Ecken) *Sei ein bewerteter Graph $G = (E, K, \varrho)$, seien $x, y \in E(G)$, sei $W(x, y)$ die Menge aller Wege in G, die x mit y verbindet. Dann ist der Abstand zwischen x und y definiert durch*

$$d_\varrho(x, y) := \min_{w \in W} \varrho(w).$$

Bemerkung 3 *Oft ist aus dem Kontext klar, wie die Bewertung ϱ einer Kante erklärt ist. In solchen Fällen ist es möglich, statt $d_\varrho(x, y)$ auch $d(x, y)$ zu schreiben.*

Anschaulich betrachtet stellt das Berechnen des Abstands zweier Ecken das Finden eines kürzesten Weges dar. Im folgenden Kapitel wird mehr darüber berichtet werden.

3 Kürzeste-Wege-Probleme

In vielen Bereichen unseres Lebens bewältigen wir (möglicherweise unbewußt) Probleme, die man graphentheoretisch durch das Suchen (und Finden) eines kürzesten Weges lösen kann:

Anwendungsbeispiel 1 *Eine Person möchte mit dem Zug von Ulm nach Hamburg reisen. Ein Blick auf den Abfahrtsplan für den Hbf Ulm zeigt ihr, dass es keine direkte Verbindung gibt, aber Fernverkehrszüge z.B. nach Mannheim, Frankfurt, Hannover und Berlin fahren. Unter Zuhilfenahme eines Kursbuches kann der Reisende dann die Verbindungen von allen von Ulm aus erreichbaren Städten nach Hamburg untersuchen. Zusätzlich wird er auch das eine oder andere Zwischenziel, das er von den aus Ulm erreichbaren Städten erreichen kann, untersuchen und diesen Vorgang möglicherweise öfter wiederholen. Auf diese Weise kann die Person alle Verbindungen zwischen Ulm Hbf und ihrem Hamburger Zielbahnhof ermitteln und nach ihren persönlichen Präferenzen (z.B. Aufenthaltsdauer in einem Umsteigebahnhof, Fahrpreis, Gesamtfahrzeit, gewünschte Ankunftszeit usw.) eine auswählen.*

Es ist heutzutage eher die Ausnahme, dass ein Reisender selbst mit einem Kursbuch eine kürzeste Zugverbindung ermittelt. Mitarbeiter in Reisezentren, elektronische Automaten und webbasierte Auskunftsdienste nehmen dem Reisenden diese Arbeit ab. Trotzdem wird dem Reisenden eine Auswahl an Verbindung vorgelegt werden, die auf o.a. Art ermittelt werden.
Graphentheoretisch betrachtet stellen die Bahnhöfe die Ecken dar. Eine Kante ist dementsprechend eine vorhandene umsteigefreie Verbindung zu einem anderen Bahnhof. Mit einer solchen Kante ist aber mehr als eine Information verknüpft: Ulm ist mit Stuttgart durch Züge verschiedener Kategorien verbunden, die a) alle dieselbe Kilometerzahl zwischen Ulm Hbf und Stuttgart Hbf zurücklegen, aber b) verschiedene Abfahrts- und Ankunftszeiten und ggf. auch c) verschiedene Fahrpreise aufweisen. Eine Möglichkeit, diese Mehrfachinformationen auf die Information einer Verbindung zu reduzieren besteht darin, für jede einzelne Verbindung eine eigene Kante zur Verfügung zu stellen.

Anwendungsbeispiel 2 *Ein Briefträger möchte die Post seines Zustellbezirks verständlicherweise in möglichst kurzer Zeit zustellen. Dabei ist ihm nicht unbedingt die kürzeste Route am Wichtigsten, sondern die praktikabelste: das Postaufkommen ist nicht jeden Tag gleich, daher möchte er eine flexible Route wählen, die ihm an vielen Tagen gute Dienste leistet. Zum Beispiel möchte er ein abgelegenes Haus, das durchschnittlich nur jeden zweiten Tag Post erhält, möglichst so in seine Route einbinden, dass er, wenn dort keine Post zugestellt werden soll, keinen Nachteil erleidet, Route günstig abkürzen kann sowie im Fall einer Zustellung möglichst wenig zusätzlichen Fußweg dorthin zurücklegen muss.*

In diesem Beispiel besteht der günstigste (Fuß-) Weg nicht notwendigerweise aus dem Weg mit der kürzesten Länge - einmal abgesehen davon, dass der Zusteller nicht notwendigerweise einen Weg im Graphen finden kann: in Sackgassen muss er ein Teilstück zurücklaufen. Hier würde man eher nach der Kantenfolge im Graphen suchen, die für ein durchschnittliches Postaufkommen einen zeitlich kürzesten Weg liefert.
Anschaulich betrachtet kann man die Ecken in diesem Beispiel mit den Häusern im Zustellbezirk und die Kanten mit Straßen und sonstigen Verbindungswegen identifizieren.

Anwendungsbeispiel 3 *Ein Besucher eines Internetcafes in Deutschland ruft den Inhalt eines Webservers auf, der an der US-amerikanischen Atlantikküste steht. Eine schnelle Verbindung vorausgesetzt, erscheint im Browser in Sekundenschnelle die angeforderte Seite. Würde der Nutzer nachforschen, würde er feststellen, dass die Anfrage an den Server einen i.a. vergleichsweise direkten Weg über den Atlantik genommen hat und nicht über Australien und/oder Südamerika geleitet wurde.*

Die Rechner, die Pakete, die z.B. beim Aufrufen einer Webseite entstehen, weiterleiten, kommunizieren auch testweise mit ihren Nachbarn, um unterbrochene Verbindungen oder solche mit einer deutlich geringeren Kapazität als bisher bekannt zu erkennen. Das Wissen um solche Veränderungen der Infrastruktur wird mit den bekannten Nachbarn geteilt, so dass sich solche Informationen durch das gesamte Netzwerk verteilen. Das geschieht natürlich zeitversetzt, aber i.a. wird z.B. die Anfrage des Nutzers des Internetcafes an die richtigen Stellen weitergeleitet, so dass die Antwort möglichst schnell auf dem anfragenden Rechner eintreffen wird.
Der Graph in diesem Beispiel ist dynamisch - sowohl die verbindenden Rechner als Ecken als auch die Verbindungen als Kanten können ausfallen.
Auf Unix-/Linuxsystemen kam man, die ggf. nötigen Rechte vorausgesetzt, den Weg

einer solchen Anfrage durch ein einfaches Kommando nachvollziehen. Im folgenden Beispiel wird das exemplarisch gezeigt:

```
renfordt@lagrange$ traceroute www.mathematik.uni-ulm.de
traceroute to www.mathematik.uni-ulm.de (134.60.54.6) [...]
 1  pa-gw.rz.uni-ulm.de (134.60.237.254) [...]
 2  239-gw-vrrp2.rz.uni-ulm.de (134.60.239.2) [...]
 3  theseus.mathematik.uni-ulm.de (134.60.54.12) [...]
```

Wie man an den Rechnernamen erkennt, verläßt die Anfrage aus dem WLAN-Netz der Universität Ulm auf dem Weg zur URL www.mathematik.uni-ulm.de die Rechner der Universität nicht. Bereits der dritte Rechner (theseus.mathematik.uni-ulm.de) kann die Anfrage beantworten. Es ist natürlich für einen Außenstehenden, der die interne Rechnerstruktur an der Universität Ulm nicht kennt, nicht möglich, dieses Ergebnis zu validieren, aber es ist augenscheinlich vernünftig.
Anhand der ersten beiden Beispiele wird im Folgenden gezeigt, wie man diese und ähnlich gelagerte Probleme allgemeiner formuliert und eine Lösung versuchen kann.

3.1 Der Algorithmus von Dijkstra

Wir betrachten im Folgenden einen Graphen $G = (E, K)$ mit $E = \{A, \ldots, G\}$ und der Bewertungsmatrix

$$B = \begin{pmatrix} \infty & 100 & 75 & \infty & \infty & 30 & \infty \\ 100 & \infty & \infty & \infty & 175 & \infty & \infty \\ 75 & \infty & \infty & 20 & \infty & 40 & \infty \\ \infty & \infty & 20 & \infty & 70 & \infty & 60 \\ \infty & 175 & \infty & 70 & \infty & \infty & \infty \\ 30 & \infty & 40 & \infty & \infty & \infty & 15 \\ \infty & \infty & \infty & 60 & \infty & 15 & \infty \end{pmatrix},$$

Abbildung 3.1 auf der nächsten Seite zeigt den Graphen in der bildlichen Darstellung. Gesucht sei z.B. der kürzeste Weg durch den Graphen G vom Ausgangspunkt A zum Endpunkt E. Analog zum ersten Anwendungsbeispiel mit dem Bahnreisenden kann man nun wie folgt vorgehen:

- Der Start ist erfolgt in Ecke A, die Liste der erreichbaren Punkte ist dann gegeben durch F (30 Einheiten entfernt), C (75) und B (100)

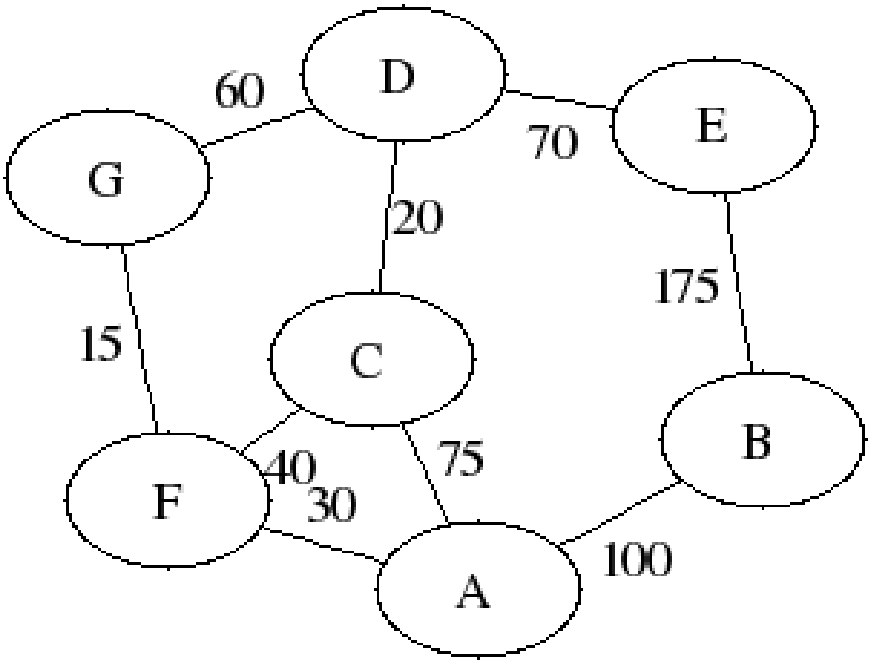


Abbildung 3.1: Der zur Bewertungsmatrix B passende Graph.

- F liegt zur Zeit am nächsten. Von dort kann man G in 15 Einheiten erreichen - G konnte bisher noch nicht erreicht werden und so wird G zu der Liste der erreichbaren Punkte hinzugefügt, und zwar mit 30 + 15 Einheiten. Diese 45 Einheiten setzen sich aus dem Abstand A zu F und dem von F zu G zusammen.

- Von F ist außerdem noch der Punkt C in 40 Einheiten erreichbar. Da der Weg von A nach C über F 5 Einheiten kürzer ist als die direkte Kante, wird der Eintrag für C in der Liste der erreichbaren Punkte geändert - die neue bekannte kürzeste Entfernung beträgt nur noch 70 Einheiten.

- Da F nun abgearbeitet ist, entfernt man F aus der Liste der erreichbaren Punkte.

- Von allen Einträgen in der Liste der erreichbaren Punkte hat nun G die kürzeste Entfernung. Von dort aus kann man den neuen Punkt D erreichen, dieser wird mit der Entfernung 30 + 15 + 60 = 105 Einheiten der Liste der noch zu besuchenden Punkt hinzugefügt. G wird wie zuvor schon F aus der Liste entfernt.

- Nun ist C von allen noch zu erreichenden Punkten am nächsten an A gelegen. Von C gelangt man in 20 weiteren Einheiten zum Punkt D. Damit wird D nun über F

und C kürzer erreichbar als über F und G. Die neue kürzere Entfernung ist nun 90 Einheiten, der Eintrag wird entsprechend geändert und C aus der Kandidatenliste gestrichen.

- Der nächste abzuarbeitende Punkt ist mit einer Entfernung von 90 Einheiten D. Von dort kann der Zielpunkt E erreicht werden, es sind von A bis dorthin 160 Einheiten und dies wird wieder in der Liste der erreichbaren Punkte vermerkt.

- Bevor nun der Zielpunkt E erreicht wird, kann man von A in kürzerer Entfernung den Punkt B erreichen. Da aber von A über B der Punkt E nur in der Entfernung von 175 erreicht werden kann, was mehr ist als in der Liste vermerkt, wird die Liste der noch zu besuchenden Punkte nur insofern geändert, als dass B dort gestrichen wird.

- Schließlich ist nur noch der Punkt E enthalten und die Entfernung von A nach E beträgt somit 160 Einheiten. Ein kürzerer Weg existiert nicht. Verfolgt man den Weg rückwärts, führt er über durch die Knoten A, F, C, D nach E.

Da der Zielpunkt E der letzte zu bearbeitende Punkt gewesen ist, wurde gewissermaßen als Nebeneffekt der Abstand sämlicher Ecken des Graphen von der Ausgangsecke A bestimmt. Wäre statt E der Zielpunkt G gewesen, hätte man früher die Bearbeitung abbrechen können - die Entfernungen zu den Punkten G, D, B und E wären nicht bestimmt worden.
Weiterhin kann man daraus, dass alle Ecken untersucht worden sind, schließen, dass der Graph G zusammenhängend gewesen sein muss. Gibt man sich keine konkrete Zielecke vor, sondern führt das Verfahren so lange fort, bis die Liste der noch erreichbaren Ecken leer ist, so gehören die tatsächlich besuchten Ecken zu einer Zusammenhangskomponente, ggf. nicht besuchte Ecken zu mindestens einer weiteren Zusammenhangskomponente.
Dieser Algorithmus wurde als erstes von Dijkstra 1959 und unabhängig davon von Dantzig 1960 beschrieben[1]. Es gibt eine Einschränkung, die im o.a. Beispiel aber stillschweigend erfüllt ist: alle Kantenbewertungen sollen echt positiv sein. Formal kann der Algorithmus - nach [Jun90] - wie folgt beschrieben werden:

Algorithmus 1 (Dijkstra) *Sei ein Graph $G = (V, E, \varrho)$ gegeben, dabei seien alle Kantenbewertungen nicht-negativ. Dann berechnet der folgende Algorithmus die Abstände*

[1] nach [Vol91]: Dijkstra, E.W.: A note on two problems in connexion with graphs. In: Numer. Math., Bd. 1, S. 269-271, 1959. bzw. Dantzig, G. B.: On the shortest route through a network. Management Sci., Bd. 6, S. 187-190, 1959/60.s

aller Ecken zu einer gegebener Ecke s:

1. $d(s) := 0, T := V$;
2. $\forall v \in V : d(v) := \infty$;
3. *solange* $T \neq \emptyset$:
   - *finde ein* $u' \in T : d(u') <= d(u) \forall u \in T$;
   - $T := T \backslash \{u'\}$;
   - $\forall u \in T : d(u) := \min(d(u), d(u') + \varrho((u', u)))$;

Dabei gibt $d(.)$ den Abstand zu der gegebenen Ecke s an. Ist $d(v) = \infty$, dann bedeutet das, dass die Ecke v nicht von der Ecke s erreicht werden kann.
Aus Sicht der Komplexitätstheorie kann man durch geschickte Implementierung von T als *priority queue* eine Komplexität von $O(|E| \log |V|)$ erreichen.

3.2 Das Travelling Salesman Problem

Das Travelling Salesman Problem (kurz TSP) - gelegentlich findet man auch den deutschen Begriff *Problem des Handelsreisenden* - eröffnet eine neue Dimension im Kürzesten-Wege-Problem. Dabei geht es darum, dass ein Handelsvertreter, von einer Stadt ausgehend in allen gegebenen Städten Kundenbesuche unternimmt, die er in beliebiger Reihenfolge durchführen kann. Dabei soll er jede Stadt genau einmal besuchen. Abschließend soll er in seine Ausgangsstadt zurückkehren. Nun stellt sich die Frage, in welcher Reihenfolge er die Städte aufsuchen soll, um eine kürzeste Route zu wählen.
Dabei geht man in aller Regel von einem ungerichteten bewerteten Graphen aus: die Städte entsprechen den Ecken, die Länge der jeweiligen Kanten den Entfernungen zwischen den benachbarten Städten bzw. den Kosten, die von der Benutzung der Kante verursacht werden. Damit das Problem sinnvoll gelöst werden kann, soll der Graph zusammenhängend sein. Oft betrachtet man einen vollständigen Graphen, d.h. von jeder Stadt aus kann man jede andere direkt erreichen. Das verhindert, dass - entgegen der Problemstellung- eine Stadt mehrfach besucht werden muss, um andere zu erreichen. Ist der Graph nicht vollständig, kann man die fehlenden Kanten künstlich einfügen. Die Länge dieser neuen Kanten bestimmt man zweckmäßigerweise durch den kürzesten Weg zwischen ihren Endpunkten im ursprünglichen Graphen. Dadurch ändert sich die Charakteristik des Graphen nicht und der Problemstellung ist Genüge getan. Im Folgenden

setzen wir einen vollständigen ungerichteten Graphen voraus.
Zulässige Lösungen des Problems lassen sich dadurch generieren, dass man die zu besuchenden Städte permutiert. Systematisches Ausprobieren aller Möglichkeiten führt zwar zur besten Rundreise, dauert aber bei einer größeren Anzahl an Städten sehr lange, denn die Anzahl der möglichen Rundreisen beträgt bei n Städten $\frac{(n-1)!}{2}$. Für nur 20 Städte ergeben sich also bereits knapp $6,08 \times 10^{16}$ Möglichkeiten. Ein Computer, der eine Million Kombinationen in der Sekunde überprüfen kann, würde also knapp 61 Mrd. Sekunden benötigen. Dies entspricht etwas mehr als 1.900 Jahren und stellt somit zwar eine theoretische Möglichkeit dar, die beste Route zu finden, ist aber eher selten eine praktische Alternative.

3.2.1 NP-Vollständigkeit

Das TSP ist also vorerst nur schwer zugänglich. Im Vergleich dazu sind die Kürzester-Weg-Probleme aus dem vorigen Beispiel deutlich angenehmer. Kann man nicht auch einen vergleichbaren Algorithmus zur Lösung des TSP in polynomialer Zeit angeben? Zur Beantwortung dieser Frage sind ein paar Begriffe aus der Komplexitätstheorie vonnöten (vgl. [Jun90], [Bran94]):

Definition 17 (Polynomiale Probleme) *Problemstellungen, die mit einem Algorithmus in polynomialer Komplexität der Eingangsdaten gelöst werden können, heißen polynomial.*

Bei Graphentheoretischen Problemen sind die Eingangsdaten in aller Regel die Mächtigkeit der Ecken bzw. die der Kanten.
Im Folgenden betrachten wir Entscheidungsprobleme. Dies sind Probleme, die als Antwort ein „ja“ oder „nein“ verlangen. Ein Beispiel dafür ist die Frage: Gibt es für einen gegebenen Graphen eine Route für einen Handlungsreisenden durch alle Knoten, die kürzer als eine gegebene Schranke M ist?

Definition 18 (Klassen von Entscheidungsproblemen) *Zwei Klassen von Entscheidungsproblemen seien wie folgt definiert:*

- *P gebe die Klasse der polynomialen Entscheidungsprobleme an,*
- *NP die Klasse der Entscheidungsprobleme, für die man eine vorgelegte Lösung in polynomialer Zeit überprüfen kann.*

Die Tatsache, dass $P \subset NP$ gilt, ist leicht einzusehen: ein polynomialer Algorithmus für ein Entscheidungsproblem, der eine gültige Lösung liefert kann auch dazu verwendet werden, diese Lösung zu verifizieren. Möglicherweise gilt $P \neq NP$, aber diese Vermutung ist weder bewiesen noch ist bekannt, ob sie überhaupt entscheidbar ist.
Die nächste Definition führt zu einer Klasse von Problemen aus NP, die vermuten läßt, dass $P \neq NP$ gilt:

Definition 19 *Ein Problem aus NP heißt* NP-vollständig*, falls aus der polynomialen Lösbarkeit dieses Problems die polynomiale Lösbarkeit aller Probleme aus NP folgt.*

Für das TSP gilt:

Satz 2 *Das TSP ist NP-vollständig.*

Es sind eine ganze Reihe von NP-vollständigen Problemen bekannt, bekannte Beispiele sind neben dem TSP das Rucksackproblem[2], das Problem der Färbung eines Graphen[3] und die Existenz eines Hamiltonschen Weges[4] in einem Graphen.
Es ist kein Algorithmus bekannt, der das TSP in polynomialer Zeit löst. Die Existenz eines solchen Algorithmus' hätte jedoch - gewissermaßen als Nebeneffekt - $P = NP$ zur Folge. Da kein polynomialer Algorithmus zur Hand ist, ist man gezwungen, sich nach Alternativen umzuschauen.

3.2.2 Approximative Algorithmen

Ein Ausweg besteht in der Suche nach Algorithmen, die zwar möglicherweise suboptimale Lösungen ergeben, deren maximaler Fehler aber für alle Ausprägungen eines Problemes bekannt ist. Formal kann diese Anforderung wie folgt beschrieben werden:

Definition 20 (Approximativer Algorithmus) *Sei ein Optimierungsproblem P gegeben sowie ein Algorithmus A, der für jeden einzelnen Fall E von P eine (nicht notwendigerweise optimale) Lösung generiert. Durch $\vartheta(E)$ sei der Wert der optimalen Lösung*

[2]Beim Rucksackproblem geht es darum, einen Rucksack mit verschiedenen Gegenständen zu füllen, die jeweils verschiedene Vermögenswerte mit einem dazugehörigen Gewicht besitzen. Das Ziel ist die Maximierung des Wertes des Rucksackinhaltes, ohne die Tragfähigkeit des Rucksackes (bzw. dessen Trägers) zu überschreiten.

[3]Das Problem der Färbung eines Graphen läßt sich bekannter auch als Problem der Färbung einer Karte formulieren: jedes Land soll eine Farbe zugewiesen bekommen, die von der Farbe jedes Nachbarlandes verschieden ist.

[4]Ein Hamiltonscher Weg in einem Graphen ist ein Weg der jede Ecke des Graphen besucht ohne Kanten mehrfach zu nutzen.

im Fall E, durch $\vartheta_A(E)$ der Wert im Fall E derjenigen Lösung bestimmt, die vom Algorithmus A erzeugt wird. Gilt für alle E die Ungleichung

$$|\vartheta_A(E) - \vartheta(E)| \leq \varepsilon\vartheta(E),$$

so nennt man A einen ε-approximativen Algorithmus für P.

Für Probleme, die in NP liegen, kann es lohnend sein, einen polynomialen ε-approximativen Algorithmus zu finden. Jedoch gilt der folgende Satz, ein Ergebnis von Sahni und Gonzales (1976)[5]:

Satz 3 *Wenn es einen ε-approximativen polynomialen Algorithmus für das TSP gibt, dann gilt $P = NP$.*

Für ein spezielleres TSP, bei dem für den Graphen die Dreiecksungleichung der Kantengewichte gilt (ΔTSP), kann man jedoch einen $\frac{1}{2}$-approximativen Algorithmus mit Komplexität $O(n^3)$ angeben. Dieses von Christofides 1976 vorgestellte Verfahren ist jedoch vergleichsweise kompliziert und wird an dieser Stelle nicht dargestellt. Es ist nach [Jun90] nicht bekannt, ob es für das ΔTSP Approximationsalgorithmen mit $\varepsilon < \frac{1}{2}$ gibt.

3.3 Heuristiken und Metaheuristiken

Betrachtet man noch einmal das Anwendungsbeispiel 1 auf Seite 11, so kann man sich auch ein anderes Vorgehen vorstellen. Natürlich führt der Algorithmus von Dijkstra zu der optimalen Lösung bezüglich Reisezeit oder minimaler Schienenkilometerzahl. Dabei untersucht der Algorithmus aber auch Zugverbindungen, die sicher nicht Bestandteil einer *guten* Lösung sind. Eine Regionalbahnverbindung von Ulm in Richtung Bodensee wird sehr wahrscheinlich nicht Teil einer schnellsten Verbindung nach Hamburg sein. Durch eine geschickte Vorauswahl oder Bevorzugung der Wahlmöglichkeiten, die *offensichtlich* besser sind, könnte man durch solche Ausschlüsse den Raum, in dem eine Lösung gesucht wird, deutlich verkleinern. Weniger mögliche Lösungen bedeuten kürzere algorithmische Laufzeiten – aber mit etwas Pech hat man die eigentlich optimale Lösung dabei ausgeschlossen.
Ein solches Vorgehen wird Heuristik genannt, eine genaue Definition einer Heuristik existiert jedoch nicht. Man versteht darunter i.a. ein Verfahren, von dem bekannt ist, dass es zu einer vernünftigen Lösung vieler Instanzen eines Problems führt, von der

[5] nach [Jun90]: Sahni, Gonzales: P-complete approximation problems. In: J. ACM 23, S. 555-565

man aber keine allgemeinen Güteeigenschaften kennt. An folgenden Beispielen wird dies deutlicher:

- Für Zugverbindungen zwischen weit voneinander entfernten Städten wird eine kürzestmögliche Reisezeit in aller Regel durch Fernverkehrszüge erzielt;
- Bei ganzzahligen Optimierungsproblemen *schöner* Funktionen liegt das ganzzahlige Optimum oft in der Nähe des allgemeinen Optimums (z.B. liegt das Minimum der Funktion $f : \mathbf{R} \to \mathbf{R}, x \mapsto (x + \frac{1}{2})^2$ unter der Nebenbedigung $x \in \mathbf{N}$ in der Nähe des allgemeinen Mimimums von f). Ist das allgemeine Problem leichter lösbar als das entsprechende ganzzahlige, kann man durch diese Heuristik auf eine vernünftige Lösung des ganzzahligen Problems schließen;
- Beim TSP führt oft die Wahl der jeweils nächstgelegenen noch nicht besuchten Stadt zu einer vergleichsweise kurzen Route;
- Beim Rucksackproblem wird man vor allem solche Lösungen betrachten, bei denen zuerst wertvolle Gegenstände in den Rucksack gepackt werden.

Diese Heuristiken können leider beliebig schlecht sein. Für Heuristiken kann man für eine allgemeine Problemstellung keine Schranke bezüglich der Güte angeben, da sonst der heuristische Algorithmus in die Klasse der approximativen Algorithmen eingestuft werden würde.

3.4 Metaheuristiken

Die Begrifflichkeit der Metaheuristik ist i.a. nicht klar von der Heuristik abgegrenzt. Hier soll jedoch eine Metaheuristik ein Verfahren sein, das eine Heuristik nutzt, um eine ganze Klasse von Problemstellungen lösen zu können. Dazu ist es aber oft nötig, die zugrundeliegende Heuristik anzupassen. Im Vergleich zum Reisenden, der mit der Bahn von Ulm nach Hamburg reist, kann ein Autofahrer mit demselben Ziel mit der Fernverkehrszug-Heuristik wenig anfangen. An diesem einfachen Beispiel sieht man bereits, dass die Wahl der Heuristik unbedingt auf das Problem abgestimmt werden muss. In diesem Beispiel ist das nicht allzu schwer: eine mögliche Heuristik für den Autofahrer ist - analog zu der bevorzugten Nutzung von Fernverkehrszügen - die bevorzugte Nutzung von Autobahnen.
Um möglichst erfolgreich im Sinne der Suche nach der optimalen Lösung zu sein, wird

eine Metaheuristik deutlich mehr als einen Durchgang der zugrundeliegenden Heuristik vorsehen. Dabei geht es weniger darum, statt einer Lösung gleich mehrere zu präsentieren, sondern darum, aus den bisherigen Versuchen eine bessere Lösung zu kreieren.
In [Sch97] werden zwei Arten von Metaheuristiken unterschieden. Einerseits gibt es konstruktive Verfahren, die unter Zuhilfenahme der heuristischen Information von einer leeren Lösung ausgehend sukzessive eine gültige Lösung konstruieren, andererseits existieren Verfahren, die ausgehend von einer oder mehrerer Lösungen weitere heuristisch ableiten. Da ein Beispiel der konstruierenden Verfahren der Ameisenalgorithmus ist, der im folgenden Kapitel ausführlich besprochen wird, sollen an dieser Stelle drei Vertreter der zweiten Klasse von Metaheuristiken kurz vorgestellt werden (vgl. dazu auch [Hro01]). Dabei sei jeweils ein Minimierungsproblem P gegeben, der Wert einer Lösung α sei durch $\vartheta(\alpha)$ gegeben.

3.4.1 Lokale Suche und Tabu-Suche

Zu Beginn betrachten wir eine sehr einfache und übersichtliche Metaheuristik.

Algorithmus 2 (Lokale Suche)

1. *Finde eine zulässige Lösung α des Minimierungsproblems P;*
2. *Finde die beste zulässige Lösung β aus der Nachbarschaft von α;*
3. *Falls $\vartheta(\beta) < \vartheta(\alpha)$, dann setze $\alpha := \beta$ und weiter zu Schritt 2. Ansonsten* STOP.

Dabei kommt dem Begriff der Nachbarschaft einer Lösung entscheidende Bedeutung zu. Beim TSP kann man beispielsweise definieren, dass durch die Vertauschung zweier Städte eine neue Lösung entsteht, die zu der ursprünglichen benachbart ist. Optimiert man allgemein eine Funktion $f : \mathbf{R^n} \mapsto \mathbf{R}$, so kann man die Nachbarschaft einer Lösung $x_0 \in \mathbf{R}$ durch $Nachbarschaft_\varepsilon(x_0) := \{x \in \mathbf{R} | ||x - x_0|| < \varepsilon\}$ definieren. In diesem Fall wäre es praktisch nicht möglich, alle Punkte aus der Nachbarschaft zu testen. Abhilfe kann leicht durch einen Schnitt der Nachbarschaft mit einem geeigneten diskreten Raster geschaffen werden.
Als Heuristik kommt bei diesem Algorithmus die Suche in der Nachbarschaft zum Einsatz: wenn man eine Lösung gefunden hat, wird versucht, diese nur wenig zu ändern, um die Güte der alten Lösung zu behalten und diese noch zu verbessern. Problematisch dabei ist aber, dass wenn die Nachbarschaft zu eng gewählt wird, der Algorithmus im erstbesten Minimum feststecken wird. Fasst man den Nachbarschaftsbegriff jedoch weiter, wird

die Menge der zu konstruierenden und zu überprüfenden Nachbarschaftslösungen sehr groß werden und das Einsparpotential gegenüber einer Absuche des gesamten Raumes aller möglicher Lösungen schrumpft. Das Verfahren der Lokalen Suche kann man durch mehrfaches Anwenden noch verbessern. Dabei sollten sich die Startpunkte nach Möglichkeit im Lösungsraum etwas verteilen.
Erweitern kann man die Lokale Suche wie folgt:

Algorithmus 3 (Tabu-Suche)

1. *Finde eine zulässige Lösung α des Minimierungsproblems P;*
2. *Setze $Tabu := \{\alpha\}, Stop := falsch, BesteLösung := \alpha$;*
3. *Finde die beste zulässige Lösung $\beta \in Nachbarschaft(\alpha) \backslash Tabu$; falls $\vartheta(\beta) < \vartheta(\alpha)$, dann setze $BesteLösung := \beta$; bestimme Tabu und Stop neu und setze $\alpha := \beta$;*
4. *Falls $Stop = falsch$ weiter mit Schritt 3, sonst* STOP.

Bezüglich der Update-Regel der Variablen *Tabu* und *Stop* gibt es verschiedenste Möglichkeiten. Setzt man *Stop* auf wahr, falls keine bessere Lösung in der Nachbarschaft der aktuellen Lösung gefunden wird, erhält man als Spezialfall die Lokale Suche. Alternativ kann man abbrechen, wenn sich für eine gewisse Anzahl an Schritten keine Verbesserung erzielen ließ. In der Variablen *Tabu* kann man bisher betrachtete Lösung speichern und nach einiger Zeit, z.B. nach $k \in \mathbf{N}$ Schritten, wieder zulassen.

3.4.2 Genetische Algorithmen

Einen vollkommen anderen Weg gehen die Genetischen Algorithmen. Wie der Name schon sagt, werden hier Ideen aus der Genetik aufgegriffen. Alle grundlegenden Informationen über Lebewesen sind in Genen codiert. Ein Teil der Gene einer Katze ist beispielsweise mit denen anderer Katzen identisch. Dies sorgt dafür, dass man, wenn man eine Katze kennt, eine andere Katze als solche zu erkennen vermag. Doch es gibt eine Reihe von individuellen Kennzeichen wie z.B. der Fellfarbe, der Ohrenform usw., die die Katzen unterscheidbar macht. Zeugen Katzen Nachkommen, werden diese die Gene ihrer Eltern erhalten. Dabei stehen zwei vollständige Sätze an Genen zur Verfügung, durch Rekombination dieser Ausgangsgene wird ein neuer, vollständiger Satz an Genen erzeugt. Dieser Vorgang passiert nicht ganz ohne Fehler: einzelnen Gene der Nachkommen werden auch mutieren. Insgesamt werden die Nachkommen ihren Eltern wenigstens

in Teilen ähnlich sehen, aber doch individuell sein.
Dies führt zu einer Metaheuristik, die das Verhalten der Natur nachstellt:

Algorithmus 4 (Genetischer Algorithmus)

1. *Initialisiere eine Ausgangspopulation;*
2. *Berechne für jedes Individuum eine sogenannte Fitness;*
3. *Generiere einen Pool von Eltern auf Basis der Fitness;*
4. *Erzeuge durch Rekombination der Gene der Eltern Nachkommen;*
5. *Mutiere zufällige Gene der Nachkommen;*
6. *Die Nachkommen ersetzen ausgeschiedene Individuen in der Population;*
7. *Solange die Abbruchbedingung nicht erfüllt ist, fahre mit Schritt 2 fort.*

Die Problemstellung bestimmt sehr stark die Darstellung einer Lösung in genetischer Form, den Rekombinations- und Mutationsmechanismus. Beim TSP kann das, nach [Wal05], wie folgt geschehen: Die zu besuchenden Städte werden in der Reihenfolge ihres Auftretens in der Lösung notiert. Um zwei dieser Lösungen zu rekombinieren, werden aus dem ersten Elternteil k zufällige Städte ausgewählt und als erster Teil der genetischen Struktur des Kinds abgelegt. Im nächsten Schritt werden dann die Städte, die im ersten Schritt nicht ausgewählt wurden, in der Reihenfolge ihres Auftretens im zweiten Elternteil an Kindlösung angehängt. Es entsteht dadurch auf jeden Fall eine zulässige Rundreise. Eine Mutation kann durch einfaches Vertauschen zweier Städte in der Kindlösung erfolgen, ohne die Zulässigkeit anzutasten.
Die Abbruchbedingung kann auch hier verschieden gehandhabt werden: sowohl statische Bedingungen wie z.B. „Abbruch nach der 15. Population“ als auch dynamische wie „Abbruch, wenn nach fünf Populationen keine Verbesserung der besten Lösung erfolgte“ kommen in Frage.

4 Ameisenalgorithmen

Ähnlich zu den Genetischen Algorithmen sind die Ameisenalgorithmen eng an die Natur angelehnt. Die Metaheuristik wurde von Marco Dorigo 1992 in seiner Dissertationsschrift [Dor92] veröffentlicht. Seitdem sind zahlreiche Publikationen erschienen, die sich mit Ameisenalgorithmen, dessen Varianten und Anwendungen beschäftigen. Einen ausführlichen Überblick stellt [Dor04] dar.

4.1 Biologie

Ameisen zeichnen sich weniger durch das einzelne Individuum aus, sondern sie treten in aller Regel als Ameisenstaat nach außen in Erscheinung. Betrachtet man einen Ameisenhügel im Wald, zeichnet er sich vor allem durch ein wirres Durcheinander aus. Alles scheint in Bewegung zu sein und Ameisen laufen übereinander hinweg. Trotzdem scheint das Konzept gut zu funktionieren, denn Ameisen finden sich in vielen Arten und an den verschiedensten Orten: in Wäldern, Wiesen, Wüsten, Städten und Gebirgen.
Aufgrund ihrer vergleichsweisen kleinen Gestalt können sich Ameisen optisch nur schlecht orientieren und oft ist ihr Sehvermögen auch nur mäßig ausgeprägt. Schon ein kleiner Grashalm verbirgt Dahinterliegendes. Ameisen vieler Arten kommunizieren deshalb indirekt über Duftstoffe, sogenannten Pheromonen, miteinander[1]. Das sind chemische Verbindungen, die eine Ameise ausbringt und andere Ameisen je nach Art des Pheromons zu Handlungen bewegt. Eine dieser Pheromonarten ist das sogenannte Wegpheromon. Wege zu Futterquellen werden von vielen Ameisen begangen, die dabei dieses Wegpheromon verbreiten. Weitere Ameisen folgen dann dieser Duftspur und finden so die Futterquelle. Im Laufe der Zeit verdunsten die Pheromone und die Spur eines nicht mehr begangenen Weges verschwindet langsam.
Eine einzelne Ameise braucht sich durch diesen Mechanismus nicht an einzelne Futterquellen zu erinnern - sie begibt sich einfach dorthin, wo schon viele andere Ameisen auf Futtersuche gewesen sind. Die Pheromonspuren stellen also ein kollektives Gedächtnis

[1] Daß es auch Arten gibt, die das nicht tun, erwähnt [Cru98].

(a) Ameisenhaufen (b) Futterquelle

Abbildung 4.1: Rote Waldameisen am Sellajoch auf etwa 2.000 m Höhe

der Ameisenkolonie dar.
In Abb. 4.1(a) ist ein Ameisenhaufen zu erkennen, eine dazugehörige Futterquelle in Abb. 4.1(b). In letzterer erkennt man bei genauem Hinsehen, dass Ameisen nur von der rechten unteren Ecke aus zu der Futterquelle gelangen. Von anderen Seiten wird die Futterquelle durch die Ameisen offenbar nicht erreicht.

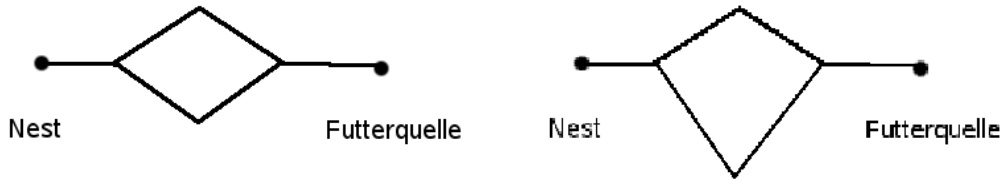
4.2 Ameisenstraßen

Es bilden sich häufig Ameisenstraßen zu ergiebigen Futterquellen. Diese zeichnen sich oft dadurch aus, dass sie einen vergleichsweise kurzen Weg vom Ameisenhaufen zu der Futterstelle darstellen, vergleiche Abb. 4.2 auf der nächsten Seite. Das ist besonders erwähnenswert, da ja, wie oben bereits beschrieben, der Sichtradius einer einzelnen Ameise eher gering ist. Wie kann es unter diesen Umständen zu diesem Ergebnis kommen?
Biologen haben dazu verschiedene Experimente durchgeführt[2]. Dabei wurden von einem Nest der untersuchten Ameisenart, die auf Pheromonbasis kommunziert, zwei gleich lange Wege zu einer Futterquelle zugelassen. Es wurde dabei auf den Wegen kein Pheromon aufgetragen, d.h., die ersten Ameisen konnten sich an keiner Erfahrung orientieren. Abb. 4.3(a) auf Seite 27 zeigt eine Versuchsskizze. Dabei ergab sich, dass nach einiger Zeit die Ameisen sich auf einen der beiden Wege konzentrierten, obwohl beide gleich lang waren. Zu Beginn wählten die Ameisen jeweilig zufällig einen der beiden Wege aus, da

[2]nach [Dor04]: Deneubourg, J.-L. et al.: The self-organizing exploratory pattern of the Argentine ant. In: Journal of Insect Behavior, Bd. 3, S. 159-168, 1990 sowie Goss,S. et al.: Self-organized shortcuts in the Argentine ant. In: Naturwissenschaften, Bd. 76, S. 579-581, 1989.

Abbildung 4.2: Ameisenstraße

noch kein Pheromon vorhanden war. Sobald einige Ameisen ihre zufällige Wahl getroffen hatten, wandten sich die weiteren Ameisen dem Weg zu, der zufällig von mehr Ameisen zu Beginn gewählt worden war, da nun entsprechende Pheromonkonzentrationen vorlagen. Eine ständig wachsende Wegpheromonkonzentration brachte nachfolgende Ameisen dazu, ebenfalls tendenziell diesen Weg anstatt den anderen zu wählen, so dass sich nach einiger Zeit ein Großteil der Ameisen auf einen Weg konzentrierte. Dieses Verhalten kann man mit *positiver Rückkopplung* beschreiben. Generell wurde dabei aber kein Weg bevorzugt; wiederholte Durchgänge des Experimentes ergaben, dass beide Wege gleich häufig bevorzugt wurden.
In einem zweiten Experiment wurde bei sonst gleichen Bedingungen wie zuvor ein Weg deutlich verlängert. In Abb. 4.3(b) auf der nächsten Seite ist die Skizze dieses Versuchsaufbau zu sehen. In fast allen Fällen endete das Experiment damit, dass sich wiederum ein Großteil der Ameisen auf einen Weg konzentrierte: den kürzeren. Auch dieses Verhalten läßt sich erklären: die ersten Ameisen haben keine Information und wählen zufällig einen der beiden Wege. Haben sie eine Futterquelle erreicht, kehren sie um. Unterstellt man, dass die Ameisen auf beiden Wegen gleich schnell vorankommen, erreichen



(a) Experiment 1 (b) Experiment 2

Abbildung 4.3: Skizzen zu den Versuchen mit der Wegfindung von Ameisen

die Ameisen zuerst wieder das Nest, die auf dem kürzeren Weg unterwegs waren. Diese bringen an der Verzweigung am Ende ihres Rückwegs bereits Pheromon auf, während die Ameisen, die zeitgleich mit ihnen gestart sind und den längeren Weg gewählt haben, sich noch auf dem Rückweg befinden. Weitere Ameisen, die sich nun auf den Weg machen, werden tendenziell nun den kürzeren Weg wählen und so werden sich die Ameisen wie im ersten Experiment nach einiger Zeit auf einen Weg konzentrieren. Entgegen dem ersten Experiment hat dieser Weg aber einen entscheidenden Vorteil: er ist der kürzere der beiden Alternativen.
Das zweite Experiment hat aber weiter gezeigt, dass es immer wieder einige wenige Ameisen gibt, die sich nicht von der hohen Pheromonkonzentration auf dem kürzeren Weg angezogen fühlen und den längeren Weg ausprobieren.

4.3 Modellbildung

Den Versuchsaufbau aus den beiden Experimenten kann man in graphentheoretische Begriffe fassen. Es liegt ein Graph mit zwei Ecken, nämlich dem Nest oder Ameisenhaufen einerseits und der Futterquelle andererseits vor. Beide Ecken werden durch zwei Kanten mit verschiedenen Längen verbunden. Da die Ameisen sich tendenziell für den Weg entscheiden, der die höchste Pheromonkonzentration aufweist, bietet es sich an, dies wie folgt zu modellieren: sei ein Graph $G = (E, K)$ gegeben, auf den Kanten des Graphen befinde sich eine jeweilige Pheromonkonzentration $\tau_{ij} > 0 \forall$ Kanten $(i, j) \in K$. Befindet sich eine Ameise zur Zeit in der Kante $i \in E$, deren Nachbarschaft nicht leer sei, so wählt sie die folgende Ecke $j \in E$ nach der Wahrscheinlichkeit

$$p_{ij} = \begin{cases} \frac{\tau_{ij}^{\alpha}}{\sum_{l \in N(i)} \tau_{il}^{\alpha}} & , \quad \text{falls } j \in N(i) \\ 0 & , \quad \text{falls } j \notin N(i) \end{cases} \tag{4.1}$$

Durch diese Übergangswahrscheinlichkeiten bewegt sich nun eine virtuelle Ameise ausgehend vom Startknoten zu dem festgesetzten Zielknoten, ein Weg im Graphen wird *konstruiert*. Dabei ist der Ansatz für die Übergangswahrscheinlichkeiten gerechtfertigt: die erwähnten Experimente mit den beiden möglichen Wegen haben zu einer sehr ähnlichen Übergangswahrscheinlichkeit geführt.
Im nächsten Abschnitt wird auf Basis dieser Entscheidungsregel ein Metaheuristischer Algorithmus präsentiert.

4.4 Algorithmus

Da es sich bei dem Ameisenalgorithmus um einen Metaheuristischen Algorithmus handelt, sei $\eta : E \times E \to \mathbf{R}_+$ eine Funktion, die angibt, wie gut die Benutzung der Kante zwischen zwei Ecken in das heuristische Lösungsschema hineinpaßt. Ein hoher Wert von $\eta_{ij} := \eta(i,j)$ sei dabei besser im Sinne der Heuristik als ein niedriger.
Sehr allgemein kann man die Metaheuristik *Ameisenalgorithmus* wie folgt beschreiben:

Algorithmus 5 (Ameisen-Metaheuristik)

1. *Initialisierung*
2. *Solange Terminierungsbedingung(en) nicht erfüllt sind*
   - *Konstruiere Lösungen unter Zuhilfenahme der Heuristik sowie des Pheromons*
   - *Verändere in Abhängigkeit der gefunden Lösungen den Pheromongehalt (sogenannte Update-Regel)*
   - *(optionale) Tätigkeiten, die nicht von den Ameisen erledigt werden können*

Für diesen allgemeinen Algorithmus ist es nicht von Bedeutung, ob erst alle Lösungen einer Iteration berechnet und danach die Pheromonintensität verändert wird oder ob dies bereits nach jeder Teillösung geschieht. Für jede Anwendung kann das nach den jeweiligen Erfordernissen gehandhabt werden.
Wichtig für die Konstruktion ist, das etwaige Nebenbedingungen eingehalten werden. So ist beispielsweise beim TSP die Auflage gegeben, alle Städte nur einmal zu besuchen und dann zur Ausgangsstadt zurückzukehren.
Am Beispiel des TSP in einem vollständigen ungerichteten Graphen $G = (E, K)$ erkennt man, wie flexibel anwendbar der Ameisenalgorithmus ist. Dabei gebe die Bewertung $\varrho((i,j)), (i,j) \in K$ an, wie weit die Ecken, d.h. die Städte, i und j voneinander entfernt seien.

4.4.1 Travelling Salesman Problem

- Zuerst steht die Wahl der Heuristik η an. Wie bereits erwähnt, führt oft die Wahl einer der nächstgelegenen noch nicht besuchten Städten oft zu einer guten Lösung. Für einen vollständigen Graphen kann man also die heuristische Funktion definieren durch $\eta_{ij} := \frac{1}{\varrho((i,j))}$.

- Kombiniert man diese Heuristik mit (4.1), kann man die Übergangswahrscheinlichkeit von der Stadt i zu Stadt j durch

$$p_{ij} = \begin{cases} \frac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l\in N(i)}\tau_{il}^{\alpha}\eta_{il}^{\beta}} & , \quad \text{falls } j \in N(i) \\ 0 & , \quad \text{falls } j \notin N(i) \end{cases} \tag{4.2}$$

  modellieren.

- Die einzige Nebenbedingung ist, dass jede Stadt nur einmal besucht werden soll. Es ist also eine Liste zu führen mit den Städten, die noch zu besuchen sind. Die Nachbarschaft einer Stadt ist dann durch diese Liste einfach gegeben. Ist die Liste leer, steht nur noch die Rückkehr zum Startort an.

- Eine Rundreise wird nun dadurch konstruiert, dass man eine Ameise von einer Stadt ausgehend alle weiteren mit den jeweiligen Übergangswahrscheinlichkeiten besuchen läßt.

- Üblicherweise konstruieren erst alle Ameisen einer Iteration eine Tour, bevor das Pheromonupdate vorgenommen wird. Dabei wird zuerst der alte Bestand an Pheromon auf allen Kanten des Graphen um den Verdunstungsfaktor $\varrho \in (0,1)$ verringert. In einem zweiten Schritt werden alle Rundreisen der Iteration ausge- und bewertet. Dabei wird jeder Tour ein Pheromonwert zugewiesen, der dann auf den Kanten, die zu der Tour gehören, den Bestand an Pheromon ergänzt.

- Es werden solange Iterationen vorgenommen, bis wahlweise eine bestimmte Anzahl an Iterationen durchlaufen wurde oder bis für eine gewisse Anzahl an Iterationen keine bessere Route gefunden wurde.

Etwas formaler aufgeschrieben, ergibt sich der Algorithmus wie folgt:

Algorithmus 6 (Ameisenalgorithmus: TSP)

1. *Initialisiere alle Kanten mit einem Pheromongehalt, setze Iterationsschritt* $:= 1$

2. *Für alle k Ameisen:*

   - *Konstruiere eine Tour T_k mit den Übergangswahrscheinlichkeiten gemäß (4.2)*
   - *Berechne die Länge L_k von Tour T_k.*

3. *Pheromonupdate:*

   - *Setze $\tau_{ij} := (1 - \varrho)\tau_{ij}$ für alle Kanten $(i, j) \in K$*
   - *Für alle zuvor konstruierten k Touren setze*

$$\Delta\tau_{ij}^{k} := \begin{cases} \frac{1}{L_k} & , \text{ falls Tour } k \text{ die Kante } (i,j) \text{ beinhaltet} \\ 0 & , \textit{ sonst} \end{cases} \tag{4.3}$$

   - *Setze*

$$\tau_{ij} := \tau_{ij} + \sum_{\mu=1}^{k} \Delta\tau_{ij}^{\mu} \text{ für alle Kanten } (i,j) \tag{4.4}$$

4. *Falls die Abbruchbedingung noch nicht erreicht ist, setze Iterationsschritt := Iterationsschritt + 1 und weiter mit Schritt 2, ansonsten* STOP.

4.5 Die Update-Regel

Der beschriebene Algorithmus stellt das sogenannte *Ant System*, kurz AS, dar. Das AS ist der Ausgangspunkt rund um die Entwicklung der Ameisenalgorithmen. Es wurde zu Beginn der 90er Jahre von Marco Dorigo entwickelt. Doch schon bald kam es zu ersten Weiterentwicklungen. Dabei wurde vielfach das Grundgerüst des soeben beschriebenen Algorithmus nicht verändert. Die Varianten unterscheiden sich vor allem durch die Variation der Update-Regel.

4.5.1 Elitäres Ameisensystem

Ebenfalls von Marco Dorigo vorgestellt wurde das Elitäre Ameisensystem. Es verhält sich bis auf das Pheromonupdate in (4.4) identisch zum AS. Zusätzlich dazu wird die bisher beste gefunden Tour T^{best} mitgeführt. Man setzt nun für alle Kanten $(i, j) \in K$

$$\Delta\tau_{ij}^{best} := \begin{cases} \frac{1}{L^{best}} & , \text{ falls Tour } T^{best} \text{ Kante } (i,j) \text{ beinhaltet} \\ 0 & , \textit{ sonst} \end{cases}$$

und wählt zusätzlich e elitäre Ameisen aus. Diese elitären Ameisen laufen nach den gewöhnlichen Ameisen die bisher beste gefundene Route ab und hinterlassen dort zusätzliches Pheromon. Die Updateregel (4.4) verändert sich also zu

$$\tau_{ij} := \tau_{ij} + \sum_{\mu=1}^{k} \Delta\tau_{ij}^{\mu} + e\Delta\tau_{ij}^{best} \text{ für alle Kanten } (i,j). \tag{4.5}$$

Ziel des Elitären Ameisensystem ist es, die Ameisen vor allem in der Nähe der bisher besten gefundenen Lösung suchen zu lassen. In vielen Fällen führt das zu einer besseren Lösung als beim AS.

4.5.2 Rangbasiertes Ameisensystem

Ende der 90er Jahre beschäftigte sich Bernd Bullnheimer sowohl mit der Adaption von Ameisenalgorithmen auf Probleme wie das *Vehicle Routing Problem*[3], vgl. [Bul99b], als auch mit neuen Varianten des Algorithmus' selbst. Dabei veröffentlichte er das Rangbasierte Ameisensystem in [Bul99a]. Im Gegensatz zu den beiden bisher vorgestellten Systemen gehen nun nicht mehr alle Ameisen einer Iteration mit ihren Routen in den Pheromongehalt ein. Lediglich die besten $\omega - 1$ Ameisen und die bisher beste gefundene Lösung werden zur Auffüllung der Pheromonkonzentration benutzt. Dabei wird der Pheromonwert der besten Ameise mit dem Faktor ω gewichtet, der der zweitbesten Ameise mit dem Faktor $\omega - 1$ usw., d.h. die r-te beste Ameise bringt einen Pheromongehalt in Höhe von $(\omega - r)\Delta\tau_{ij}^{r}$ ein. Damit ergibt sich die Update-Regel durch

$$\tau_{ij} := \tau_{ij} + \sum_{r=1}^{\omega-1} (\omega - r)\Delta\tau_{ij}^{r} + \omega\Delta\tau_{ij}^{best} \text{ für alle Kanten } (i,j). \tag{4.6}$$

Ebenfalls wie das Elitäre Ameisensystem verbessert das Rangbasierte Ameisensystem die Qualität der Lösungen im Vergleich zum AS spürbar. Dadurch, dass schlechte Lösungen erst gar nicht berücksichtigt werden, werden die Ameisen bei ihren Entscheidungen wesentlich besser in der Nähe bereits gefundener guter Lösungen gehalten um diese lokal zu optimieren.

[3] Das Vehicle Routing Problem stellt eine Verallgemeinerung des TSP dar. Dabei geht es darum, eine Anzahl an Kunden mit einem Fahrzeug mit begrenzter Frachtkapazität zu beliefern. Dem Fahrzeug steht es jederzeit offen, in einem Lager seinen Bestand wieder aufzufüllen. Ist die Kapazität des Fahrzeugs höher als die Nachfrage der Kunden, so erhält man als Spezialfall das TSP.

4.5.3 Weitere Varianten

Es gibt eine ganze Reihe weiterer Varianten und Algorithmen, die sich stark am Vorbild des Ameisenalgorithmus orientieren. Zu nennen sind hier das Ant Colony System und das Hypercube Framework, bei deren Entwicklung auch jeweils Marco Dorigo beteilgt war. Ein weiterer wichtiger Vertreter ist das MAX-MIN Ant System, das wesentlich von Thomas Stützle entwickelt wurde. Es beinhaltet recht intensive Veränderungen bezüglich des AS. U.a. wird nur die beste Ameise einer Iteration bzw. die bisher beste gefundene Tour gewertet und die Pheromonkonzentration auf den Kanten darf sich nur in einem Intervall $[\tau_{min}, \tau_{max}]$ bewegen (vgl. [Stü96]).

4.6 Pheromoninitialisierung

Ein Blick auf die Übergangswahrscheinlichkeiten wie in (4.2) offenbart: Hat eine in Frage kommende Kante zu der nächsten Stadt j eine Pheromonkonzentration $\tau_{ij} = 0$, so wird sie niemals gewählt werden. Möchte man diese Übergangswahrscheinlichkeiten anwenden, muss sichergestellt sein, dass alle Kanten mit einem initialen Pheromonwert versehen werden. Dieser Startwert sollte aber *vernünftig* sein, d.h., die Größenordnung muss richtig gewählt werden. Ist der Startwert sehr viel größer als die Pheromonwerte der später gefunden Touren, so fällt diese Erfahrung nicht ins Gewicht. Ist hingegen der Initialwert deutlich kleiner als die Werte der ersten Touren, so werden bereits in den ersten Iterationen nur selten Kanten gewählt, die bisher noch nicht versucht wurden. Es ist also darauf zu achten, dass ein auf die jeweilige Variante des Ameisenalgorithmus' und auf die Situation abgestimmter Startwert gewählt wird, um möglichst gute Ergebnisse zu erzielen.
Beim TSP kann beispielsweise die Nächster-Nachbar-Heuristik als Basis für den initialen Pheromonwert verwendet werden.

4.7 Parameterwahl

In [Dor92] wird auch ein Vorschlag zur Wahl *guter* Parameter für verschiedene Varianten des Ameisenalgorithmus' zur Lösung des TSP gegeben:

- $\alpha := 1, \beta \in [2, 5]$;
- beim AS als auch beim elitären Ameisensystem wird die Wahl des Verdunstungsfaktor $\varrho := \frac{1}{2}$ vorgeschlagen, für das Rangbasierte Ameisensystem $\varrho := \frac{1}{10}$;

- die Anzahl der Ameisen pro Iteration soll der Anzahl der zu besuchenden Städte entsprechen.

Für das TSP ist das natürlich ein guter Anhaltspunkt, wie Parameter sinnvoll gewählt werden können. Versucht man jedoch eine Implementierung des Ameisenalgorithmus für ein weniger gut untersuchtes Problem, wird man nicht umhinkommen, einige Versuche zur Parameterjustierung durchzuführen.

4.8 Parallele Implementierung

Die Ausprägungen AS, Elitäres Ameisensystem und Rangbasiertes Ameisensystem eignen sich vorzüglich zur parallelen Implementierung. Da innerhalb jeder Iteration die Ameisen vollkommen unabhängig voneinander agieren, kann man sie auf verschiedene Prozesse oder Threads verteilen. Dabei sollte jedoch sichergestellt sein, dass die konstruierten Routen in einem synchronisierten Objekt gesammelt werden. Liegen alle Touren vor, d.h. sind alle nebenläufigen Prozesse oder Threads beendet, kann dann wie gewöhnlich die Update-Regel ausgeführt werden.
Im MIN-MAX-System hingegen verändern die Ameisen unmittelbar beim Beschreiten einer Kante deren Pheromongehalt. Deshalb gibt es dort in aller Regel nur eine Ameise pro Iteration und zum Ausgleich deutlich mehr Iterationen. Die gerade beschriebene Parallelisierungsmöglichkeit besteht also nicht mehr.
Eine vollkommen andere Möglichkeit stellt der Versuch dar, mehrere virtuelle Ameisenkolonien das Problem lösen zu lassen. Ein ständiger Austausch über Pheromonkonzentrationen sorgt dabei aber für einen großen Zeitaufwand für Kommunikation zwischen den verschiedenen Kolonien. Wesentlich besser schneidet ein Abgleich der Ergebnisse mehrerer Kolonien in bestimmten Intervallen ab[4]. Beispielsweise tauschen sich dann die einzelnen Kolonien nach jeweils fünf erfolgten Iterationen aus.

[4] nach [Dor04]: Bullnheimer, B., Kotsis, G., Strauss, C.: Parallelization strategies for the Ant System. In: Leone, R. D., Murli, A., Pardalos, P., Toraldo G. (Hrsg.): High Performance Algorithms and Software in Nonlinear Optimization, Bd. 24 in Kluwer Series of Applied Optimization, Seite 87-100, Kluwer Academic Publishers, Dordrecht, 1998.

5 Verkehrsdatenlage im Zentrum Berlins

Seit Ende 2001 arbeitet das Institut für Stochastik, Fakultät für Mathematik und Wirtschaftswissenschaften an der Universität Ulm mit dem Zentrum für Luft- und Raumfahrt, Arbeitsgruppe Verkehrsstudien, in einem gemeinsamen Projekt an der Vorhersage von Verkehrszuständen in großstädtischen Ballungsräumen. Zur Verfügung stehen dafür Daten aus dem Berliner Verkehrsgeschehen, die von ca. 300 Berliner Taxis gesammelt wurden. Diese Daten sollen ein repräsentatives Bild des Verkehrsgeschehen in einem Beobachtungsfenster der ungefähren Größe 12 km $\times$ 12 km liefern. Da an den Wochenenden sich das Verkehrsaufkommen deutlich von dem an Werktagen unterscheidet, werden im Folgenden lediglich die Daten betrachtet, die an Werktagen aufgezeichnet wurden.

5.1 Taxipositionsdaten

Die Taxipositionsdaten werden erzeugt, indem periodisch die Zeit und der Ort (als Längen- und Breitenangabe) der Taxis automatisch aufgezeichnet wird. Als ein Attribut dieser Meldungen wird auch ein Status geführt; dieser gibt an, ob das Taxi beispielsweise gerade mit einem Kunden unterwegs ist, einen Kunden abholt oder etwa in einer Taxiwarteschlange steht. Zur Identifizierung der Taxis hat jedes Taxi eine eindeutige ID, die tagesabhängig vergeben wird. Für dieses Projekt liegen die so gesammelten Daten aus dem Zeitraum 1. Januar 2004 bis zum 31. März 2004 vor. Da z.B. Meldungen von Taxis, die sich gerade in einem Wartezustand befinden, für die Beobachtung des fließenden Verkehrs unerheblich sind, müssen die Daten bearbeitet werden. Zu diesem Zweck werden alle Daten eines Tages, die zu einer Taxi-ID gehören, zeitlich sortiert. Alle Messpunkte, die außerhalb des Beobachtungsfenster liegen, werden ersatzlos gestrichen. Der so entstandene Polygonzug wird nun unterteilt. Liegen beispielsweise zwei Messpunkte zu nah bzw. deutlich zu weit auseinander oder wechselt der Status, wird der Polygonzug unterteilt. Auf diese Weise entstehen sogenannte Trajektorien von Taxifahrten. Einzelne

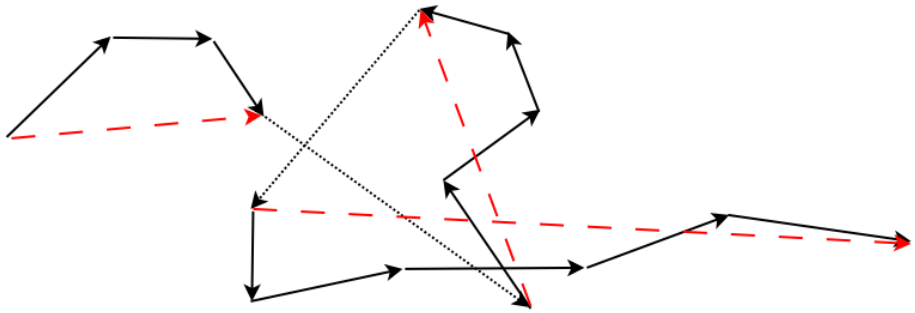


Abbildung 5.1: Beispiel eines Polygonzuges mit drei Trajektorien

Trajektorien stellen dabei z.B. eine Fahrt eines Taxis mit einem Fahrgast vom Flughafen Tegel zu einem Wohnhaus in Charlottenburg oder eine Anfahrt zu einem Kunden dar. Das Ziel der Unterteilung ist es, diese Sinneinheiten des gesamten Polygonzuges zu ermitteln. Abb. 5.1 zeigt ein Beispiel eines Polygonzuges (schwarze Pfeile), der unterteilt wurde. Zunächst einmal werden die gepunkteten Teile des Polygonzuges entfernt. Es bleiben drei Trajektorien, die sich in sogenannte Segmente unterteilen. Ein Segment ist der Teil der Trajektorie, der sich zwischen zwei Meldezeitpunkten befindet. Mit roten Pfeilen wird der Start- und Endpunkt einer jeden Trajektorie im Bild gekennzeichnet.
Die entstandenen Trajektorien und Segmente werden nun weiter untersucht. Verschiedene Merkmale werden betrachtet:

- die zeitlichen Abstände zweier zeitlich aufeinanderfolgender Trajektorien (unabhängig von der Taxi-ID);
- die geographischen Orte aller Messpunkte wurden aus dem System der Längen- und Breitengrade in das *UTM-Koordinatensystem* umgerechnet[1].

[1] UTM steht als Abkürzung für die *Universal Transverse Mercator*-Projektion. Koordinaten im UTM-System bestehen aus zwei Komponenten: einerseits ein sogenannter Ostwert, der den Abstand in Metern von einem zugehörigen Meridian angibt, andererseits aus dem sogenannten Nordwert, der den Abstand in Metern zum Äquator angibt. Die Projektion ist so angelegt, dass geographische Koordinaten eines Bereichs, der sogenannten Kachel, als in einer Ebene gelegen betrachtet werden können. Somit können die Daten euklidisch verrechnet werden, z.B. kann die Entfernung zweier Punkte der selben UTM-Kachel über den Satz des Pythagoras berechnet werden. Der relevante Mittelmeridian für die Berliner Daten ist der Meridian 15° östlicher Länge, aus dem Abstand zum Äquator ergibt sich für Berlin die Kachel 33U. Ein erster Orientierungspunkt zum UTM-System bietet [wik]. Zur Umrechnung der geographischen Daten in das UTM-System wurde das Tool *cs2cs*, Version 4.4.9 aus der proj.4-Bibliothek verwendet. Es kann unter http://www.remotesensing.org/proj/ (Stand vom 7. Juni 2007) heruntergeladen werden.

- der Winkel der Trajektorie, d.h. der Winkel zwischen der Verbindungsgeraden des ersten und letzten Messpunktes der Trajektorie zu einer Referenzgeraden. Hier wurde als Referenzgerade die Halbgerade gewählt, die parallel zum Äquator in Ostrichtung durch den Startpunkt führt (vgl. Abb. 5.2 auf der nächsten Seite);
- der Winkel eines Segments, der auf die gleiche Art und Weise gewonnen wird;
- aufgrund des Segmentwinkels wird das entsprechende Segment in einen sogenannten Sektor eingeordnet. Es gibt vier Sektoren, die den Fahrtrichtungen Nordost, Nordwest, Südwest und Südost entsprechen;
- die Differenz des Winkels eines Segments zu dem Winkel des Vorgängersegments;
- der Abstand der Messpunkte eines Segments, d.h. die Länge des Segments[2];
- der zeitliche Abstand der Messpunkte, die ein Segment bilden;
- aus der Länge des Segments und der Zeitdauer zwischen den Messpunkten wird die durchschnittliche Geschwindigkeit des Segments errechnet.

All diese Daten werden in einer Datenbank gespeichert. Die Struktur der beiden Tabellen *Trajectories* und *Segments* ist wie folgt:

```
mysql> describe Trajectories;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| trajectoryNumber | int(11)     | NO   | PRI |         |       |
| date             | datetime    | NO   |     |         |       |
| taxiID           | int(11)     | NO   |     |         |       |
| angle            | decimal(8,7)| NO   |     |         |       |
+------------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> describe Segments;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| taxiID             | int(11)     | NO   |     |         |       |
| trajectoryNumber   | int(11)     | NO   | PRI |         |       |
| segmentNumber      | int(3)      | NO   | PRI |         |       |
| date               | datetime    | NO   |     |         |       |
| timespan           | int(3)      | NO   |     |         |       |
| segmentStartEasting| decimal(11,3)| NO  |     |         |       |
```

[2]dabei wird als Abstand die Länge der Verbindungsgeraden beider Punkte verwendet. Diese Luftlinie muss nicht dem tatsächlichen Fahrweg des Taxis entsprechen

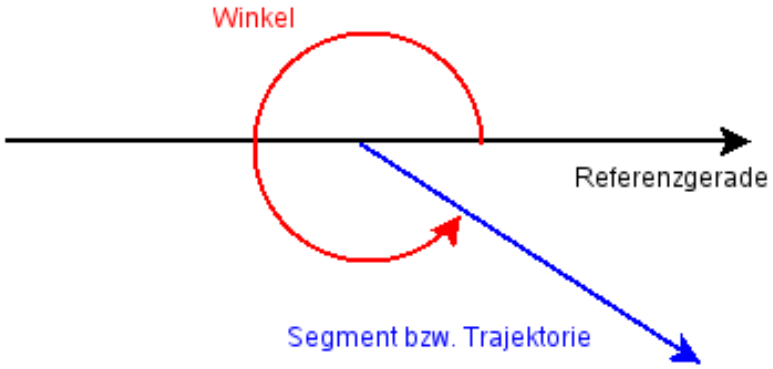


Abbildung 5.2: Skizze zur Berechnung des Winkels einer Trajektorie

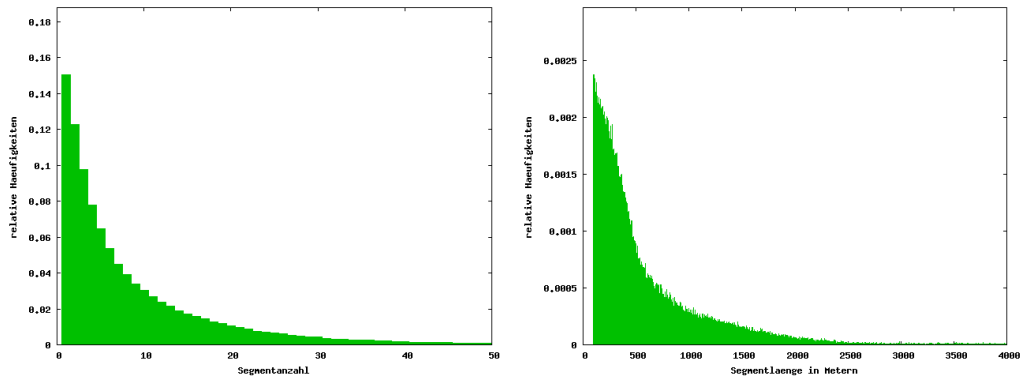
```
| segmentStartNorthing    | decimal(11,3) | NO |     |         |       |
| segmentEndEasting       | decimal(11,3) | NO |     |         |       |
| segmentEndNorthing      | decimal(11,3) | NO |     |         |       |
| velocity                | decimal(5,2)  | NO |     |         |       |
| velocityDifference      | decimal(5,2)  | YES|     | NULL    |       |
| segmentLength           | decimal(6,2)  | NO |     |         |       |
| segmentAngle            | decimal(8,7)  | NO |     |         |       |
| segmentAngleDifference  | decimal(8,7)  | YES|     | NULL    |       |
| sector                  | int(1)        | NO |     |         |       |
| state                   | int(2)        | NO |     |         |       |
+-------------------------+---------------+----+-----+---------+-------+
16 rows in set (0.00 sec)

mysql>
```

5.1.1 Auswertung der Merkmale der Trajektorien und Segmente

Die Kenngrößen, die unmittelbar zu den einzelnen Trajektorien bzw. der Segmente gehören, werden nun ausgewertet. Dazu werden aus den Daten Histogramme erstellt.

Anzahl der Segmente pro Trajektorie

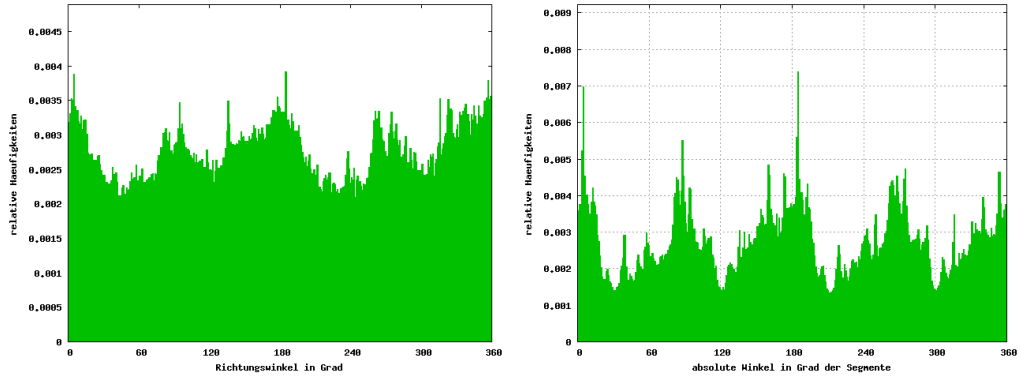
Die insgesamt 297.556 Trajektorien enthalten insgesamt 2.719.666 Segmente, im Mittel setzt sich also eine Trajektorie aus etwas mehr als 9 Segmenten zusammen. Das entsprechende Histogramm der Häufigkeiten der Segmentanzahl pro Trajektorie zeigt Abb. 5.3(a) auf der nächsten Seite.
Interessant ist zudem die Länge eines Segments. Abb. 5.3(b) auf der nächsten Seite stellt die Häufigkeiten der Länge in Metern eines Segments dar. Auffällig ist das Fehlen von Segmenten mit Längen kleiner als 100 Meter. Dies kommt dadurch zustande,



(a) relative Häufigkeit der Anzahl der Segmente pro Trajektorie

(b) relative Häufigkeit der Segmentlängen

Abbildung 5.3: Segmenteigenschaften



(a) der Trajektorien

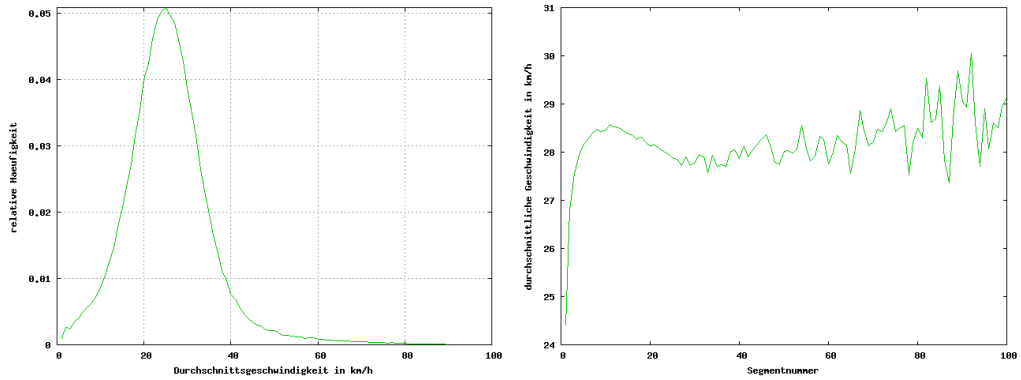
(b) der Segmente

Abbildung 5.4: Häufigkeitsverteilungen von Winkeln

dass Trajektorien, in denen zwei Messpunkte mit Abstand von weniger als 100 Metern vorkommen, getrennt werden.

Winkel von Trajektorie und Segmenten

Die Häufigkeitsverteilungen der Winkel von Trajektorien und Segmenten kann Abb. 5.4 entnommen werden. Diese Verteilungen sind vermutlich sehr stark von der zugrundeliegenden Stadt, der Lage typischer Fahrtziele von Taxen und natürlich auch von der Wahl des Beobachtungsfensters abhängig.



(a) der Trajektorien (b) der i-ten Segmente

Abbildung 5.5: Durchschnittsgeschwindigkeiten

Durchschnittsgeschwindigkeiten

Die Durchschnittsgeschwindigkeiten der Trajektorien, vgl. Abb. 5.5(a), werden durch das arithmetische Mittel der Geschwindigkeiten der einzelnen Segmente errechnet.
Zur Berechnung der Durchschnittsgeschwindigkeiten der i-ten Segmente werden die Geschwindigkeiten der i-ten Segmente aller Trajektorien arithmetisch gemittelt. Besitzt eine Trajektorie dieses Segment nicht, wird kein Wert verrechnet. Wie die Abb. 5.5(b) zeigt, ist diese Kurve zu Beginn recht glatt, in späten Segmenten zeigt sich jedoch ein sprunghaftes Verhalten. Dies erklärt sich durch die geringe Zahl an Trajektorien, die mehr als beispielsweise 40 Segmente besitzen.
Gut zu erkennen ist, dass die Durchschnittsgeschwindigkeit in den ersten Segmenten deutlich ansteigt, etwa im 10. Segment ein Maximum erreicht und danach wieder langsam abfällt (bis das Verhalten sprunghaft wird). Diese Charakteristik entspricht in etwa einer typischen Taxifahrt: Abholen des Fahrgastes in einem Wohngebiet mit kleinen Straßen, Aufsuchen größerer Straßen zum schnellen Transfer und Absetzen des Fahrgastes wieder in langsamer zu befahrenden Straßen.

5.1.2 Berechnung von Mittelwertkarten

Die soeben vorgestellten Eigenschaften betreffen nur einzelne Trajektorien oder Segmente und sind direkt aus den Datenbanktabellen ablesbar. In einem nächsten Schritt werden sogenannte Mittelwertkarten erstellt. Dazu wird das Beobachtungsfenster durch ein Bild mit $n \times m$ Pixel wiedergegeben. Da in dieser Arbeit ein quadratisches Beobachtungsfenster von $12km \times 12km$ Größe gewählt wurde, wird, um Verzerrungen zu

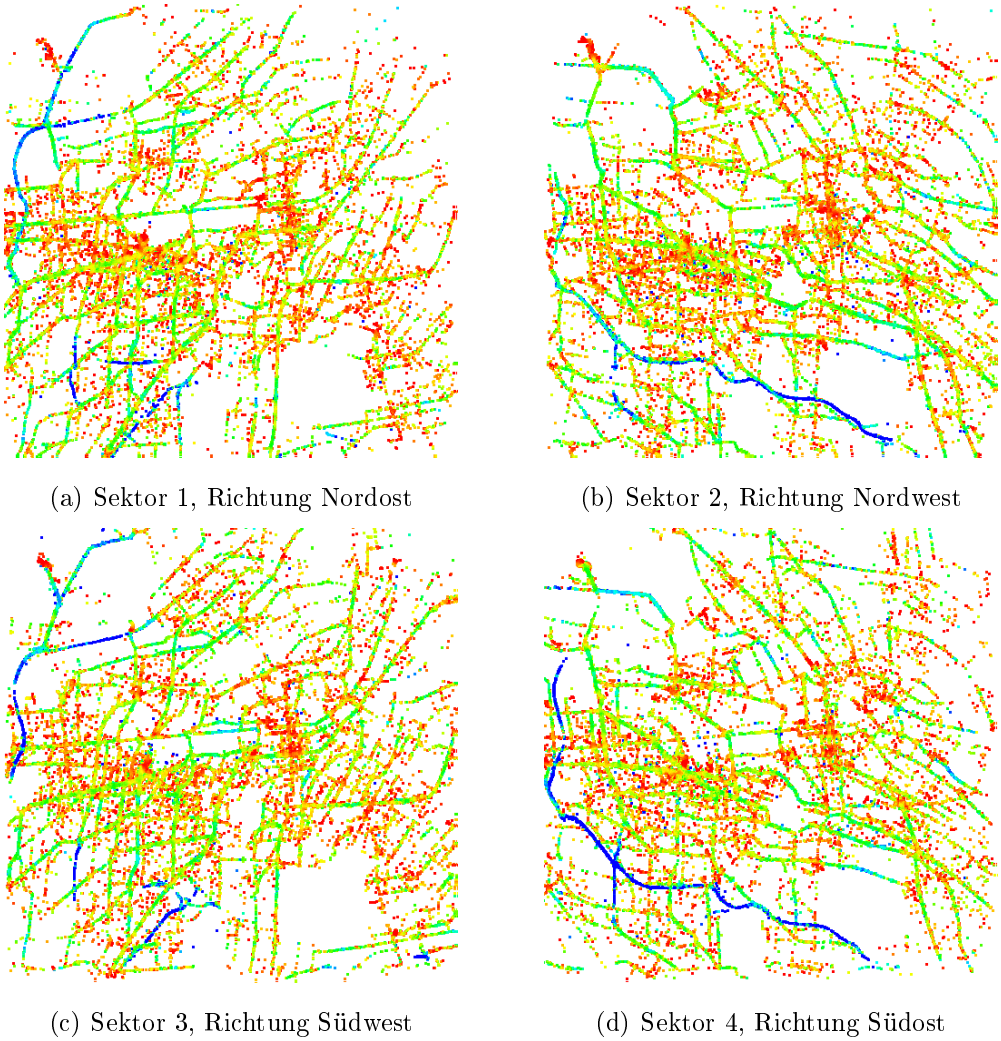
vermeiden, auch das Bild der Mittelwertkarten quadratisch gewählt. Zur Anwendung kommt eine Auflösung von 500×500 Pixeln. Ein Pixel entspricht damit einem Bereich von $24m \times 24m$.
Zuerst wird jedem Messpunkt ein Pixel mit einer gewissen Umgebung zugeordnet. Jedes Pixel aus dieser Umgebung erhält quasi die selbe Information wie das eigentliche Pixel. Die Umgebung ist jedoch klein gewählt, hier sind es Quadrate mit 5^2 Pixel. Zur Generierung der Mittelwertkarten werden nun einzelne Messpunkte nach Kriterien zusammengefaßt:

- die Messpunkte zu Segmentbeginn müssen in den selben zeitlichen Bereich fallen. Hier wurden Abschnitte von halben Stunden gewählt;

- die Richtung der Segmente werden gemäß den o.a. Sektoren unterschieden, da die Geschwindigkeit von Fahrzeugen in der einen Fahrtrichtung einer Straße in aller Regel nicht von der Geschwindigkeit entgegenkommender Fahrzeuge abhängt.

Schließlich werden noch die zeitlichen Abschnitte der Sektoren zu größeren zusammengefaßt: aus vier kleinen zeitlichen Abschnitten wird ein größerer errechnet. Beispielsweise werden zur Generierung der Mittelwertkarte von 12 - 13h eines Sektors die Mittelwertkarten 11.30h bis 12h, 12h bis 12.30h, 12.30h bis 13h sowie 13h bis 13.30h des selben Sektors verwendet. Dabei wird die zeitlich größere Mittelwertkarte pixelweise aus den kleineren zusammengesetzt. Die Gewichtung der einzelnen Pixel der kleineren Mittelwertkarten erfolgt dabei nach der Menge der eingeflossenen Messpunkte.
Auf diese Weise stehen schließlich 24 Mittelwertkarten für jeden Fahrtrichtungssektor zur Verfügung. Abb. 5.6 auf der nächsten Seite zeigt dies am Beispiel der Mittelwertekarten für den Zeitraum von 8h bis 9h morgens. In der Datenbank werden die mittleren Geschwindigkeitswerte natürlich als Zahlen in der Einheit $\frac{km}{h}$ vorgehalten. Zur geeigneten Darstellung als Bild wurden die Zahlen in Farben umgewandelt. Dabei steht rot für 0, blau für $60\frac{km}{h}$ oder mehr. Dazwischenliegende Mittelwerte werden linear in die Farben zwischen rot und blau umgewandelt.
Die ermittelten Mittelwertkarten stellen das langfristige Verhalten dar[3] und dienen weiteren Untersuchungen als Basis.

[3] beispielsweise kann man aus den Karten typische Staupunkte leicht als rote Bereiche identifizieren



(a) Sektor 1, Richtung Nordost

(b) Sektor 2, Richtung Nordwest

(c) Sektor 3, Richtung Südwest

(d) Sektor 4, Richtung Südost

Abbildung 5.6: Mittelwertkarten für den Zeitraum 8h bis 9h morgens

5.2 Straßengraph

Der Graph des Berliner Straßensystems ist gegeben durch eine Liste der Kanten. Die Ecken sind dabei durch eindeutige Nummern gekennzeichnet und besitzen als Attribut ihren geographischen Ort, der durch die Angabe des Längen- und des Breitengrades gegeben ist. Wie bei den aufgezeichneten Taxidaten wird diese Positionsangabe in das UTM-Koordinatensystem umgerechnet. Der Graph wird komplett in der Datenbank vorgehalten, erst beim Auslesen wird er auf das Beobachtungsfenster reduziert. Dadurch kann das Beobachtungsfenster flexibel gehandhabt werden.
Die Datenstruktur des Graphen liegt in der Datenbank wie folgt vor:

```
mysql> describe Vertices;
+-----------+-----------------+------+-----+---------+-------+
| Field     | Type            | Null | Key | Default | Extra |
+-----------+-----------------+------+-----+---------+-------+
| id        | int(11)         | NO   | PRI |         |       |
| easting   | decimal(11,3)   | NO   |     |         |       |
| northing  | decimal(11,3)   | NO   |     |         |       |
+-----------+-----------------+------+-----+---------+-------+
3 rows in set (0.04 sec)

mysql> desc Edges;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| start | int(11) | NO   | PRI |         |       |
| end   | int(11) | NO   | PRI |         |       |
+-------+---------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql>
```

Jeder Eckeneintrag besitzt also drei Felder: die ID sowie den Ost- als auch den Nordwert aus dem UTM-Koordinatensystem. Da die Kanten durch die Start- und Endecken beschrieben sind, handelt es sich hierbei um einen gerichteten Graphen. Einbahnstraßen sind damit einfach abzubilden, ebenso räumlich getrennte Fahrbahnen.
Die Kanten bestehen lediglich aus Geraden zwischen den Ecken - damit können erst einmal keine Kurven modelliert werden. Um eine Kurve zu erzielen, sind im Datensatz auch Ecken enthalten, die keine Kreuzungen darstellen, sondern lediglich den Kurvenverlauf approximieren. Dies gelingt recht gut, Abb. 5.7 auf der nächsten Seite zeigt den Graph des Berliner Straßennetzes eingeschränkt auf den Beobachtungsbereich. Zur besseren Orientierung sind drei Markierungen eingefügt: in der nordwestlichen Ecke befindet sich der Flughafen Berlin-Tegel, in der südöstlichen der Flughafen Berlin-Tempelhof. In der Bildmitte ist das Brandenburger Tor als Orientierungspunkt angegeben.
Es liegt in der Natur eines Straßennetzes, dass prinzipiell jeder Punkt von jedem anderen aus erreichbar ist, der dazugehörige Graph also stark zusammenhängend ist. Hier ergibt sich durch das Begrenzen des Graphen auf das Beobachtungsfenster die Situation, dass einige Ecken am Rand von den übrigen isoliert sind. Manche Punkte sind zudem nur in einer Fahrtrichtung zu erreichen. Sieht man von diesen Restriktionen einmal ab, so handelt es sich um einen Graphen ohne brückenähnliche Kanten, d.h. es gibt in aller

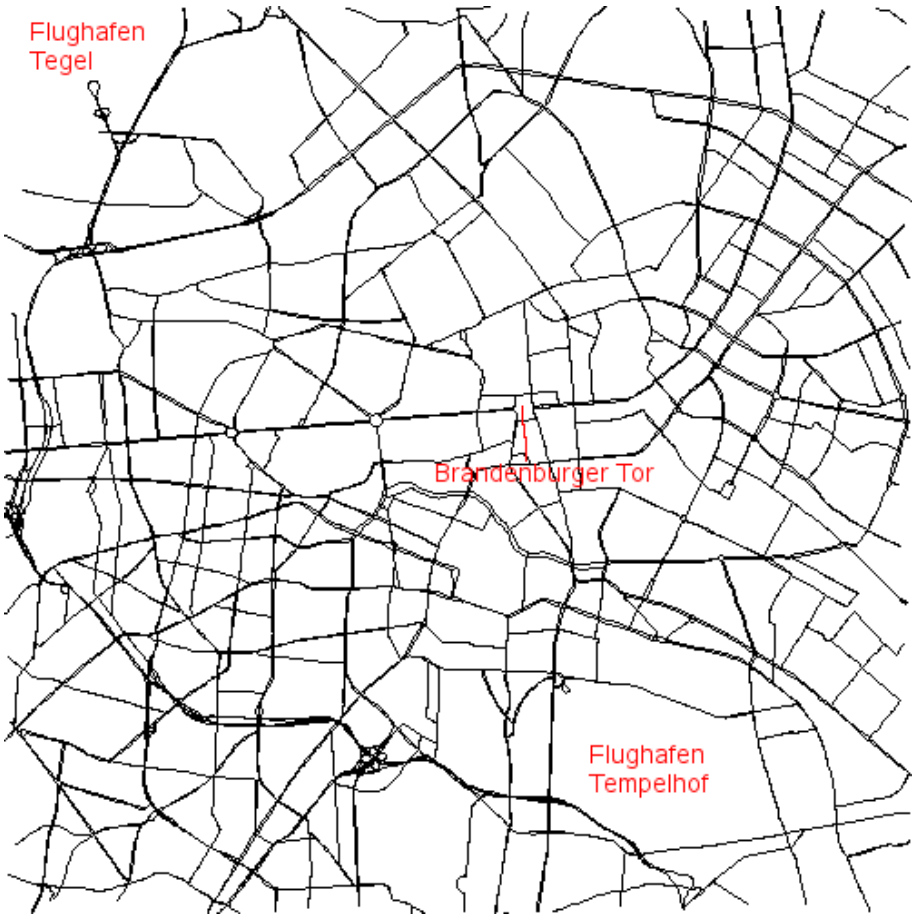


Abbildung 5.7: Der Graph des Straßensystems im Beobachtungsfenster

Regel sehr viele Wege von einem Ausgangs- zu einem Endpunkt.
Vergleicht man das Bild des Graphen mit einer Straßenkarte von Berlin, so stellt man fest, dass einige Kanten in Wohngebieten fehlen, die Menge der Kanten und Ecken ist also im Vergleich zur Realität unvollständig. Zudem sind in der Datenbank keine Informationen über die Ordnung der Straßen hinterlegt. Eine Kante, die ein Autobahnteilstück darstellt wird absolut gleich wie eine lokale Straße behandelt. Es gibt auch keine Informationen über die maximal mögliche Geschwindigkeit auf einer Straße.

6 Berechnung kürzester Verkehrswege mit Ameisenalgorithmen

In diesem Kapitel wird beschrieben, wie man mit Hilfe des Ameisenalgorithmus' in einem Straßengraphen kürzeste Wege finden kann. Als Basis dient dabei zum einen natürlich der Graph des Straßensystems. Die Länge einer Kante dieses Graphen bestimmt sich durch die Luftlinienentfernung der Kantenendstücke. Da das Interesse an einer schnellen Fahrtmöglichkeit und weniger an einer längenmäßig kurzen Verbindung besteht, kommt dazu noch die Information der Mittelwertkarten. Dadurch kann einer Kante eine Geschwindigkeit zugewiesen werden, die einerseits von der Uhrzeit und andererseits von der Fahrtrichtung abhängt. Zusammen mit der Länge einer Kante errechnet sich dann eine Anzahl von Sekunden, die für die Fahrt entlang der Kante aufgebracht werden muss.

6.1 Die Wahl der Heuristik

Bevor nun der Algorithmus beschrieben wird, ist die Wahl der Heuristik zu treffen. Diese Wahl fällt hier nicht allzu schwer, denn es gibt recht viele Kreuzungen, d.h. entsprechend auch viele Wahlmöglichkeiten. Dies wiederum führt dazu, dass man bei der Konstruktion eines Weges nicht bereits zu Beginn Entscheidungen für den restlichen Weg treffen muss, sondern auch später noch leicht korrigieren kann. Dadurch macht man wenig falsch, wenn man tendenziell auf den Zielpunkt zufährt. Auch bei den Taxidaten stellt man fest, dass die Taxen sich oft so verhalten. Betrachtet man die Winkeldifferenz zweier aufeinanderfolgender Segmente, also den Wert des Ausdruckes $AbsoluterWinkel(Segment_i) - AbsoluterWinkel(Segment_{i+1})$, so stellt man fest, dass diese Winkeldifferenz in aller Regel in dem Intervall $[-\frac{\pi}{2}, \frac{\pi}{2}]$ liegt. Abb. 6.1 auf der nächsten Seite zeigt die relative Häufigkeit von Winkeldifferenzen (umgerechnet in Grad). Insbesondere Taxifahrer sollten sich in ihrer Stadt sehr gut auskennen und auch die Ver-

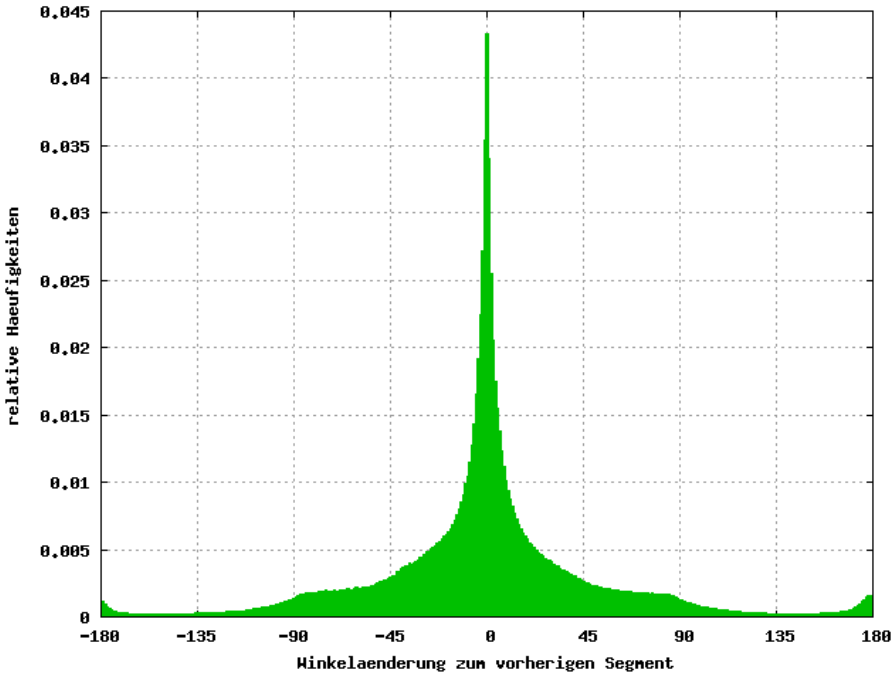


Abbildung 6.1: Relative Häufigkeiten von Winkeldifferenzen zweier aufeinanderfolgender Segmente

kehrslage so gut kennen, dass sie zu jeder Uhrzeit ihre Fahrgäste schnell ans Ziel bringen können. Orientiert sich die Heuristik an dem Verhalten der Taxifahrer, sollte sie ähnlich gut sein.

Wichtig für die praktische Umsetzung ist, dass die Funktion $\eta : K \mapsto \mathbf{R}_+$ möglichst einfach gewählt wird. Da es viele Ecken auf dem Weg zum Ziel gibt, wird eine Ameise sehr häufig die Funktion η auswerten müssen. Dazu wird folgender Ansatz gewählt: für alle zur Verfügung stehenden Kanten wird der Absolutbetrag des Innenwinkels zum Zielpunkt bestimmt. Darunter wird der Winkel zwischen einer Kante k und der Luftlinie von der Standortecke zum Zielpunkt verstanden, vgl. Abb. 6.2 auf der nächsten Seite. Dieser Winkel α_k liegt damit im Intervall $[-\pi, \pi]$. In die Berechnung der Heuristik η_k fließt er wie folgt ein: $\eta_k := 1 - \frac{|\alpha_k|}{\pi}$. Dieser Ausdruck ist symmetrisch bzgl. α_k, ebenso wie das Diagramm der Winkeldifferenzen in Abb. 6.1. Zudem ist der Ausdruck η_k durch den einfachen Ansatz leicht auszurechnen und ermöglicht so einen schnellen Programmablauf in der praktischen Implementierung.

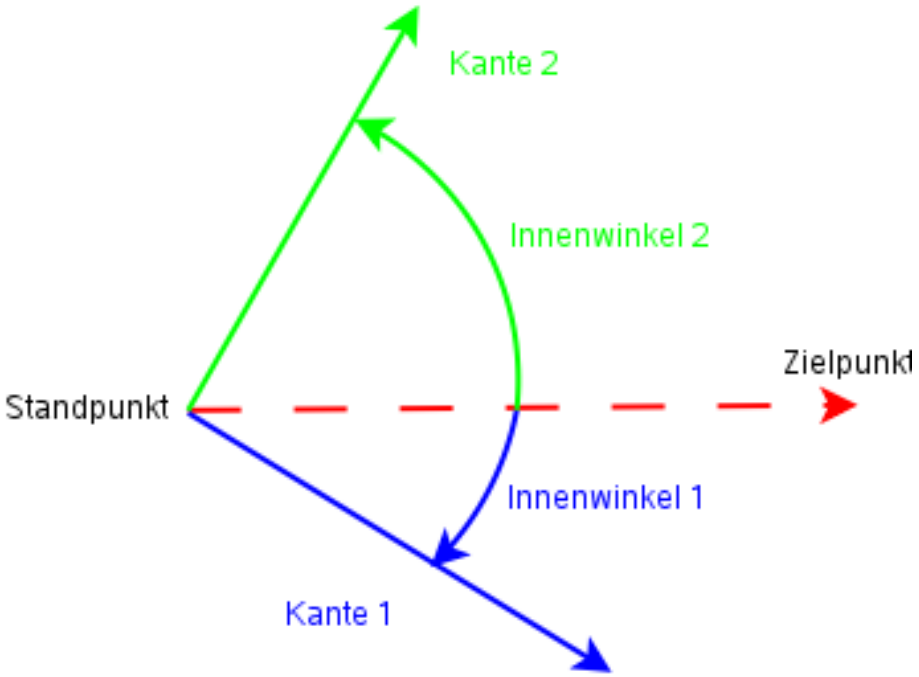


Abbildung 6.2: Skizze zur Bestimmung des Winkels zum Zielpunkt

6.2 Algorithmus

Der Algorithmus gleicht sehr dem Ameisenalgorithmus für das TSP. Unterschiede bestehen lediglich in der Heuristik η und den Nebenbedingungen: beim TSP soll eine Rundreise konstruiert werden, hier geht es lediglich um einen Weg zu einem Zielpunkt.

Algorithmus 7 (Ameisenalgorithmus: kürzester Weg)

1. *Initialisiere alle Kanten mit einem Pheromongehalt, setze Iterationsschritt* $:= 1$
2. *Für alle k Ameisen:*
   - *Konstruiere einen Kantenzug* K_k *von der Start- zur Zielecke mit den Übergangswahrscheinlichkeiten von Ecke i zu Ecke j*

$$p_{ij} = \begin{cases} \frac{\tau_{ij}\eta_{(i,j)}}{\sum_{l\in N(i)}\tau_{il}\eta_{(i,l)}} & , \text{ falls } j \in N(i) \\ 0 & , \text{ falls } j \notin N(i) \end{cases}$$

$$= \begin{cases} \frac{\tau_{ij}(1-\frac{|\alpha_{(i,j)}|}{\pi})}{\sum_{l\in N(i)}\tau_{il}(1-\frac{|\alpha_{(i,l)}|}{\pi})} & , \text{ falls } j \in N(i) \\ 0 & , \text{ falls } j \notin N(i) \end{cases}$$

- *Überführe den Kantenzug K_k in einen Weg W_k, vgl. Satz 1 auf Seite 8*
- *Berechne die Fahrtzeit in Sekunden L_k von Weg W_k.*

3. *Pheromonupdate:*

   - *Setze $\tau_{ij} := (1 - \varrho)\tau_{ij}$ für alle Kanten $(i, j) \in K$*
   - *Für alle zuvor konstruierten k Wege: setze*

$$\Delta\tau_{ij}^{k} := \begin{cases} \frac{1000}{L_k} & , \text{ falls Weg } k \text{ die Kante } (i,j) \text{ beinhaltet} \\ 0 & , \textit{ sonst} \end{cases}$$

   - *Setze $\tau_{ij} := \tau_{ij} + \sum_{\mu=1}^{k} \Delta\tau_{ij}^{\mu}$* für alle Kanten (i, j)

4. *Falls die Abbruchbedingung noch nicht erreicht ist, setze Iterationsschritt := Iterationsschritt + 1 und weiter mit Schritt 2, ansonsten* STOP.

Der Algorithmus verwendet das Ant System, aber es ist ein Wechsel zu anderen Update-Regeln - wie beim TSP beschrieben - ohne weiteres möglich.

6.3 Initialisierung

Wie bereits erwähnt, ist der Graph des Straßensystems nicht stark zusammenhängend. In der Initialisierung ist also abzuprüfen, ob die Zielecke von der Startecke überhaupt zu erreichen ist. Zudem muss natürlich auch der initiale Pheromongehalt ermittelt werden. Hier wird beides auf einmal erledigt: der Test auf starken Zusammenhang liefert auch einen Weg, der sich in der Größenordnung einer guten Lösung bewegen wird.
Ist sichergestellt, z.B. durch Analyse des Graphen im Vorfeld, dass der Graph stark zusammenhängend ist, kann auf diesen Test natürlich verzichtet werden. Die initiale Pheromonkonzentration kann dann durch Abschätzen ermittelt werden: die zeitliche Länge der Luftlinie zwischen Start- und Zielwert, $L_{Luftlinie}$, stellt einen Anhaltspunkt für die Größenordnung der optimalen Lösung dar. Möchte man etwas genauer agieren, vergleicht man in den Originaldaten die Luftlinie mit den tatsächlich gefahrenen Routen. Der Faktor λ, um den die Fahrtdauer durchschnittlich länger als die Dauer einer fiktiven Fahrt entlang der Luftlinie durchschnittlich ist, kann dann in die Abschätzung einfließen.

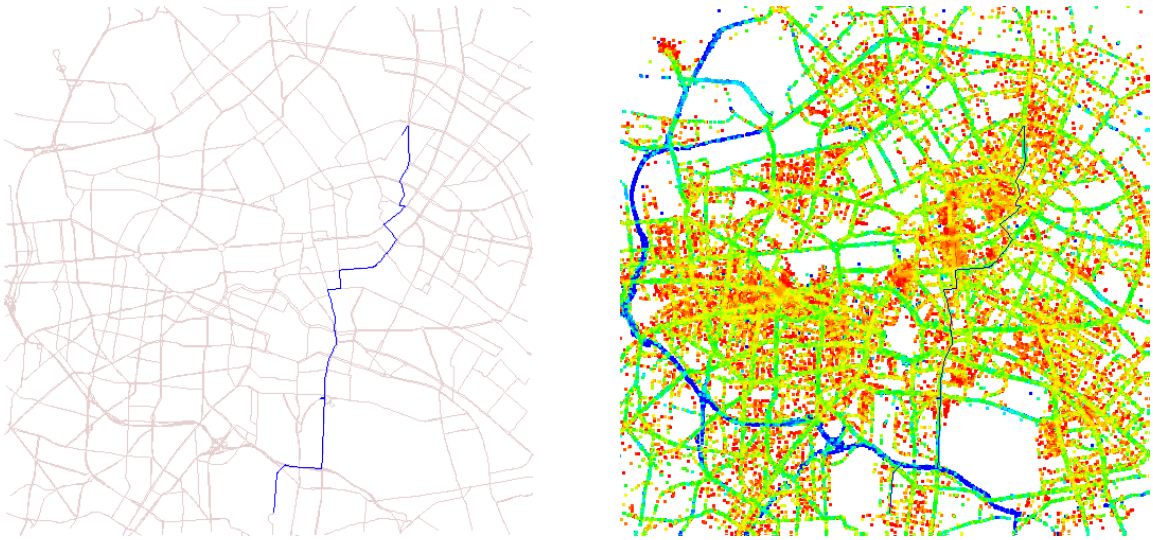
6.4 Ermittlung der Geschwindigkeit einer Kante

In vielen Fällen fallen Start- und Endecke einer Kante nicht auf das gleiche Pixel der Mittelwertkarten. In solchen Fällen kommt der Algorithmus nach Bresenham zum Einsatz, vgl. [Bre65], der eine häufig benutzte Möglichkeit darstellt, wie man die Pixel einer Verbindungslinie zweier Pixel zu wählen hat. Erschwerend kommt hinzu, dass in den Mittelwertkarten auch Pixel ohne Information existieren. Folgende Fälle können auftreten:

- Start- und Endecke fallen auf das gleiche Pixel:
  - ist der Geschwindigkeitswert des Pixels bekannt, ist dies die mögliche Geschwindigkeit der Kante;
  - fehlt diese Information, wird der Geschwindigkeitswert der vorigen Kante verwendet; ist noch gar kein Geschwindigkeitswert in der Route vorhanden, wird ein Wert von 25 $\frac{km}{h}$ zu Grunde gelegt
- sind die Pixel verschieden, wird mit dem erwähnten Algorithmus von Bresenham eine Liste aller Pixel erstellt, die diese Punkte verbinden;
  - kein Pixel hat einen zugeordneten Geschwindigkeitswert in der Mittelwertkarte: es wird wie oben der letzte bekannte Wert fortgeschrieben;
  - haben Pixel entsprechende Geschwindigkeitseinträge, wird von allen Einträgen das ungewichtete arithmetische Mittel als Geschwindigkeit der Kante zugeordnet.

6.5 Beispiel

Abb. 6.3 auf der nächsten Seite zeigt ein Beispiel, in dem mit dem Ameisenalgorithmus ein kurzer Weg zwischen zwei Punkten ermittelt wurde. Zur besseren Übersicht ist in Abb. 6.3(a) nur der Graph und in blau der gefundene Weg vom Startpunkt im rechten oberen Bereich zum Zielpunkt in der Mitte der unteren Bildkante eingezeichnet. Zugrunde liegen die Mittelwertkarten des zeitlichen Bereiches zwischen 21 und 22h. Der gefundene Weg ist 11.961 m lang und die ermittelte Fahrzeit beträgt 1.472 Sekunden, das sind knapp 25 Minuten. Damit beträgt die mittlere Geschwindigkeit auf diesem Weg fast 30 $\frac{km}{h}$. In Abb. 6.3(b) ist zusätzlich noch eine Geschwindigkeitsmittelwertkarte



(a) ohne unterlegte Mittelwertkarte (b) mit unterlegter Mittelwertkarte

Abbildung 6.3: Beispiel einer gefunden Route

unterlegt. Hier wurden die vier Karten der einzelnen Fahrtrichtungen jedoch aus Übersichtsgründen zu einer Karte verrechnet. Der Algorithmus verwendet jedoch nach wie vor die jeweiligen Geschwindigkeitskarten der Sektoren. Man erkennt, dass die gefundene Route die roten Bereiche, die eine sehr niedrige mittlere Geschwindigkeit darstellen, meidet, z.B. wird der Bereich in der Bildmitte leicht südöstlich umfahren.

6.6 Implementierung

Die Implementierung ist in der Programmiersprache Java vorgenommen worden und macht Gebrauch vom Programmierparadigma der Objektorientierung. Da der gesamte Quellcode mit knapp 7.000 Zeilen deutlich zu lang ist, um hier abgedruckt zu werden, folgt nur eine kurze Beschreibung der wichtigsten Pakete. Der gesamte Quellcode liegt der Arbeit auf CD-ROM bei.
Zu nahezu jedem Paket gehören auch Exceptions-Klassen, die jedoch in diesem Text nicht weiter diskutiert werden.

- Das Paket *SpatialObjects* beinhaltet lediglich eine Klasse *Point*. Dort werden alle Berechnungen durchgeführt, die auf UTM-Koordinaten basieren, wie z.B. die Entfernung eines Punktes zu einem gegebenen anderen oder auch die Berechnung des Innenwinkels (s.o.).

- Das Paket *graph* beinhaltet die Implementierung des Graphen. Die Klasse *Graph* dient dabei als erste Anlaufstelle für den Benutzer des Pakets. Intern werden dort alle Ecken vorgehalten, die mit einer eindeutigen Nummer versehen sind. Ausgehend von einer bekannten Ecke kann der Aufrufer beispielsweise mit der Methode *LinkedList<Integer> getNeighbours(int i)* eine Liste der Nummern der Ecken erhalten, die von der gegebenen Ecke i erreichbar sind. Eine Reihe von Methoden ruft entsprechende Methoden der Klassen *Vertex* und *Edge* auf, so dass allein mit den Methoden der Klasse *Graph* ein Großteil aller Informationen abgerufen werden kann.
  Großen Gebrauch davon macht die Klasse *Route*, die eine Route durch Ecken des Graphen repräsentiert. Innerhalb dieser Klasse wird z.B. die Fahrtzeit einer Route berechnet auf Basis der Kantenlänge, die im Graphenobjekt vorgehalten wird.

- Da die Kantenlänge zur Berechnung der Fahrtzeit entlang der Kante nicht ausreicht, wird im Paket *meanVelocities* eine Schnittstelle zur Datenbank vorgehalten, mit der die Informationen der Geschwindigkeitsmittelwertkarten abgerufen werden können. Dies erledigt die Klasse *MeanVelocities*. Zur Reduzierung der Rechenzeit werden alle abgefragten Geschwindigkeiten im Arbeitsspeicher zwischengespeichert: wird ein bereits abgefragter Wert erneut angefordert, wird nicht die Datenbank konsultiert, sondern der Wert direkt aus einem Hash an den Aufrufer zurückgegeben. Das erneute Auslesen des Wertes aus der Datenbank würde ein Vielfaches der Zeit benötigen, die durch die Verwaltung des Hashes tatsächlich anfällt.

- Das Paket *ants* zerfällt in zwei Sinnbereiche:

  - Die Klasse *Ant* ist von der Klasse *Threads* abgeleitet und dient der nebenläufigen Konstruktion einer Route von einem Start- zu einem Zielknoten gemäß des Algorithmus. Eine konstruierte Route, d.h. eine Instanz von *Route* wird in einer synchronisierten Liste abgelegt. Die Synchronisierung ist unabdingbar, da mehrere Threads ihre konstruierten Routen zeitgleich dort ablegen könnten.

  - Die abstrakte Klasse *Ants* stellt den Ameisenalgorithmus per se dar. Dementsprechend werden hier die wichtigsten Parameter dem Konstruktor übergeben, z.B. die Anzahl der Ameisen pro Iteration, der Verdunstungsfaktor ϱ, Referenzen auf den zugrundeliegenden Graph und die Mittelwerte und na-

türlich auch die Start- und Zielecke. Noch im Konstruktor wird der Pheromongehalt berechnet und das Graphenobjekt entsprechen präpariert. Kein Parameter ist jedoch die Abbruchbedingung. Der Anwender der Klasse kann gezielt durch den Aufruf der Methode *void iterate(int iterations)* die angegebene Anzahl von Iterationen durchführen lassen und so die Kontrolle über den Ablauf vollständig behalten; mehrere Aufrufe dieser Methode sind zulässig. Für jede Iteration wird die Anzahl der Ameisen aufgeteilt. Für jeden zur Verfügung stehenden Thread wird eine Instanz der Klasse *Ant* gestartet. Beispielsweise erhalten zwei Threads jeweils zehn Ameisen zur Berechnung, wenn pro Iteration 20 Ameisen eine Lösung konstruieren sollen. Nachdem alle Threads ihre Berechnungen beendet haben, wird das Sammelobjekt mit den berechneten Routen ausgewertet, d.h. die Update-Regel wird ausgeführt. Da für jede Variante des Ameisenalgorithmus diese anders formuliert ist, ist diese Methode abstrakt. Die drei genauer beschriebenen Varianten Ant System, Elitäres und Rangbasiertes Ameisensystem sind implementiert und füllen diese Methode entsprechend aus.
Dem Aufrufer stehen weiterhin Methoden zur Verfügung, die Kenndaten Länge und Fahrzeit der besten Route als auch diese selbst abzufragen.

7 Verkehrssimulation

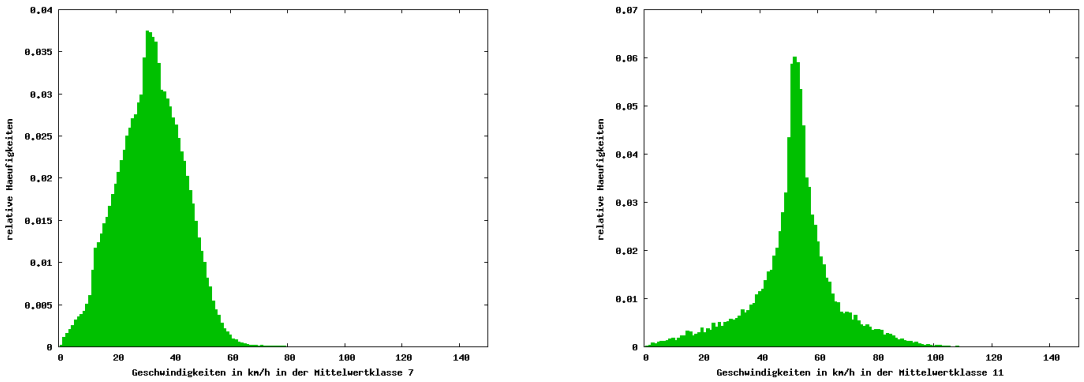
Die im vorigen Abschnitt vorgestellte Suche nach kurzen Wegen unter Zuhilfenahme des Ameisenalgorithmus' kann auch zur Simulation des Verkehrsgeschehen verwendet werden. Dabei wird ein Teil der Parameter der vorhandenen Taxi-Daten als Simulationsgrundlage verwendet, während die übrigen Parameter zum Vergleich der Ergebnisse dienen sollen.

7.1 Weitere Eigenschaften der Taxi-Daten

Es wurden bereits einige Eigenschaften der originalen Daten der Berliner Taxis beschrieben. Für die Simulation werden weitere benötigt.

7.1.1 Geschwindigkeitsmittelwertklassen

Zur Berechnung der Geschwindigkeitsmittelwerte werden, wie bereits erwähnt, die einzelnen Geschwindigkeiten der Segmente, die auf das gleiche Pixel, in das gleiche zeitliche Intervall und die gleiche Fahrtrichtung fallen, zu einem Geschwindigkeitsmittelwert zusammengefaßt. Aber auch umgekehrt kann man analysieren: welche Geschwindigkeiten waren beteiligt, so dass dieser Mittelwert dabei entstand? Dazu werden die Geschwindigkeitsmittelwerte in zwölf Klassen eingeteilt. Jede dieser Klassen deckt dabei einen Bereich von fünf $\frac{km}{h}$ ab, d.h. die erste Klasse umfaßt Geschwindigkeitsmittelwerte aus dem Intervall $[0\frac{km}{h}, 5\frac{km}{h})$, die zweite entsprechend das Intervall $[5\frac{km}{h}, 10\frac{km}{h})$ usw. Die zwölfte und letzte Klasse umfaßt einerseits ihre dementsprechendes Intervall $[55\frac{km}{h}, 60\frac{km}{h})$, darüber hinaus aber auch noch alle weiteren nicht berücksichtigten Geschwindigkeitsmittelwerte. Nun wird für jedes einzelne Segment notiert, in welche Geschwindigkeitsmittelwertklasse es fällt. Wertet man diese Information aus, gewinnt man ein Spektrum an Geschwindigkeiten, die zu den jeweiligen Mittelwerten führten. Abb. 7.1 auf der nächsten Seite zeigt beispielhaft die relativen Häufigkeiten der Geschwindigkeiten, die zu zwei ausgewählten Mittelwertklassen führen.



(a) Mittelwertklasse 7 (b) Mittelwertklasse 11

Abbildung 7.1: Zwei beispielhafte Mittelwertklassen

7.1.2 Geschwindigkeiten im ersten Segment

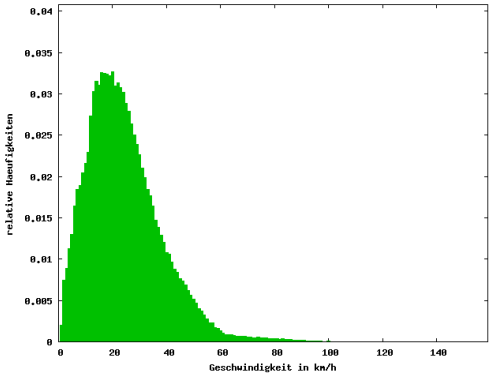
Eine weitere Charakteristik der Taxi-Daten findet in der Simulation Verwendung. Dabei handelt es sich um die Geschwindigkeit des ersten Segments jeder Trajektorie, vgl. Abb. 7.2(a) auf der nächsten Seite.

7.1.3 Abstand des Trajektorienstartpunktes zur nächsten Ecke im Graphen

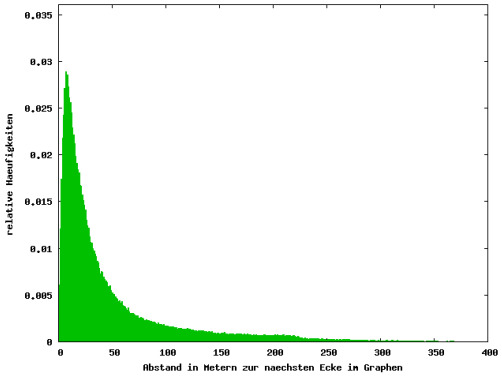
Wie bereits erwähnt, bildet der Graph das Berliner Straßensystem nicht vollständig ab. So mancher Messpunkt der Taxidaten wird also weder auf einer Ecke noch auf einer Kante des Graphen zu liegen kommen. Insbesondere ist davon natürlich auch der erste Messpunkt einer Trajektorie betroffen. Um später den Ameisenalgorithmus verwenden zu können, wird nun jedem Trajektorienstartpunkt die nächstgelegene Ecke im Graphen zugeordnet. Abb. 7.2(b) auf der nächsten Seite zeigt die relativen Häufigkeiten der Entfernung in Metern zu dieser Ecke. Ein Großteil der Trajektorien ist von den entsprechenden Ecken weniger als 50 m entfernt.

7.1.4 Der zeitliche Abstand zweier aufeinander folgender Trajektorien

Ebenfalls analysiert wurde der zeitliche Abstand zweier aufeinanderfolgender Trajektorien. Dazu wurden alle Trajektorien nach Datum und Uhrzeit sortiert und die Differenzen



(a) relative Häufigkeit der Geschwindigkeit im ersten Segment



(b) relative Häufigkeiten der Abstände in Metern zwischen Startpunkt einer Trajektorie und der zugeordneten Ecke im Graphen

Abbildung 7.2: Weitere Eigenschaften der Taxi-Daten

der Startzeiten notiert[1]. Diese Startzeitdifferenzen geben quasi die Intensität der Trajektorien an. Da, wie in Abb. 7.3(a) auf der nächsten Seite zu erkennen ist, die Anzahl der Trajektorien während der Nachtstunden geringer als zu anderen Tageszeiten ist, werden die Startzeitdifferenzen in Abhängigkeit der Stunden betrachtet. Abb. 7.3(b) auf der nächsten Seite zeigt beispielhaft die Startzeitdifferenzen der Trajektorien, die zwischen 8h und 9h morgens begannen.

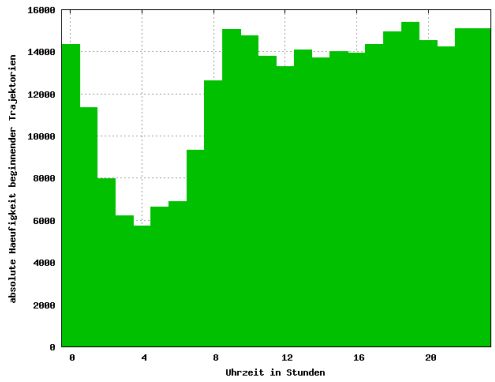
7.1.5 Zeitlicher Abstand zweier Messpunkte einer Trajektorie

Ebenfalls von Interesse für die Simulation ist der zeitliche Abstand zweier Messpunkte einer Trajektorie. Die Geräte, die in den Taxis die Position aufzeichnen, sind darauf angelegt, je nach Status des Taxis in unterschiedlichen Intervallen Daten aufzuzeichnen. Dies erklärt die Peaks, die in Abb. 7.4 auf der nächsten Seite zu erkennen sind.

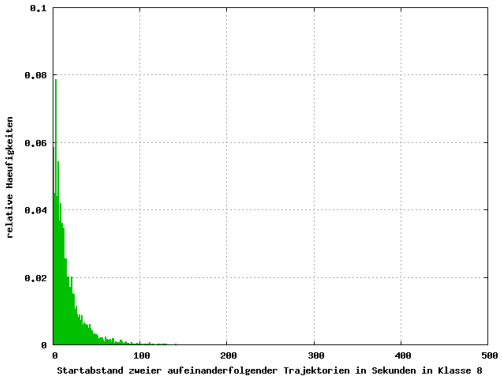
7.2 Vorgehen

Die Simulation unterteilt sich in zwei gedankliche Abschnitte, nämlich die Simulation einer einzelnen Trajektorie und zum anderen eine Simulation von vielen Trajektorien über eine gewisse zeitliche Dauer. Letztere nimmt dazu natürlich bezug auf die Simulation

[1] Falls zwischen zwei Trajektorien ein Wochenende lag, das wie bereits erwähnt nicht behandelt wird, wurde die Startzeitdifferenz verworfen



(a) absolute Anzahl an beginnenden Trajektoren zur angegeben Stunde



(b) relative Häufigkeiten der zeitlichen Abstände zweier Trajektorien in Sekunden zwischen 8h und 9h morgens

Abbildung 7.3: Intensität von Trajektorien

einer einzelnen Trajektorie.

7.2.1 Simulation einer Trajektorie

Um möglichst realistische Trajektorien zu simulieren, wird aus den Taxi-Daten eine Trajektorie zufällig herausgegriffen. Als Basis für die zu simulierende Trajektorie dient der Startort und der Winkel der existierenden Trajektorie. Der Startort ist, wie oben beschrieben, mit einer Ecke des Graphen verknüpft, die ihm am nächsten liegt. Diese Ecke dient als Startpunkt der Simulation. Um mit Hilfe des Ameisenalgorithmus einen kürzesten Weg zu finden, muss noch ein Zielpunkt bestimmt werden. Dieser berechnet

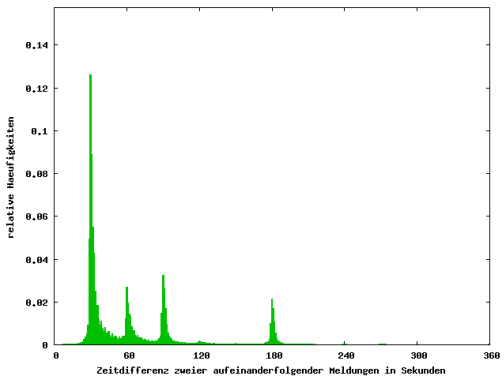


Abbildung 7.4: Abstand in Sekunden zwischen zwei Messpunkten einer Trajektorie

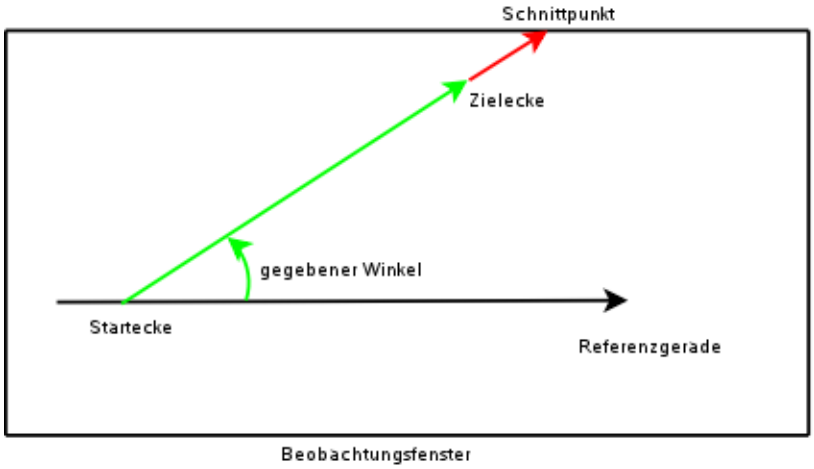


Abbildung 7.5: Skizze zur Bestimmung des Zielpunkt

sich wie folgt:

- zuerst wird der Schnittpunkt der Halbgeraden, die unter dem gegebenen Winkel den Ort der Startecke verläßt, mit dem Beobachtungsfenster berechnet;
- in einem zweiten Schritt wird die Strecke zwischen Startecke und dem Schnittpunkt auf $\frac{9}{10}$ ihrer Länge verkürzt, d.h. die Strecke endet nun nicht mehr im Schnittpunkt mit dem Beobachtungsfenster sondern innerhalb des Beobachtungsfensters;
- schließlich wird ausgehend vom neuen Streckenendpunkt die nächstgelegene Ecke im Graphen gesucht, sie stellt den Zielpunkt dar.

Die Abb. 7.5 zeigt eine Skizze des Vorgehens.
Durch das Versetzen des Zielpunktes vom Rand etwas mehr ins Innere des Beobachtungsfensters wird erreicht, dass sehr viel häufiger die Zielecke von der Startecke aus erreicht werden kann.
Nun wird durch den Ameisenalgorithmus ein möglichst schneller Weg zwischen Start- und Zielpunkt errechnet. Dieser Weg dient gewissermaßen als die Fahrtstrecke eines simulierten Taxis. Um auch Messpunkte zu generieren, wird auf Basis der relativen Häufigkeiten der Anzahl der Segmente einer Trajektorie für diese zu simulierende Trajektorie eine Segmentzahl zufällig gewählt. Nun werden solange Messpunkte entlang der Fahrtstrecke generiert, bis die erforderliche Segmentanzahl erreicht ist. Die Messpunkte werden wie folgt ausgewählt:

1. über die relativen Häufigkeiten der Geschwindigkeiten im ersten Segment wird die Startgeschwindigkeit zufällig gewählt;

2. anhand der zeitlichen Abstände zweier Messpunkte einer Trajektorie wird ein Abstand zufällig gewählt;

3. mit der aktuellen Geschwindigkeit wird nun der vom Ameisenalgorithmus vorgeschlagene Weg (weiter-) verfolgt, bis der Zeitabstand erfüllt ist. Es wird eine Meldung analog zu den originalen Taxi-Meldungen abgesetzt;

4. an diesem Ort wird nun die Geschwindigkeit aus den Mittelwertkarten eruiert und mit der bisherigen aktuellen Geschwindigkeit ungewichtet gemittelt und als neue Geschwindigkeit gewählt;

5. falls noch nicht alle Segmente konstruiert wurden, fahre fort bei Schritt 2, ansonsten STOP.

Sollte einmal der durch den Ameisenalgorithmus vorgeschlagene Weg nicht ausreichen, um alle geforderten Segmente zu konstruieren, so wird am Ende des Weges die Trajektorie beendet.
Abb. 7.6 auf der nächsten Seite zeigt ein Beispiel einer solcherart konstruierten Trajektorie. Der Startpunkt befindet sich am oberen, der Zielpunkt nahe des unteren Bildrandes. In grün ist der vom Ameisenalgorithmus berechnete Weg zwischen diesen beiden Punkten eingezeichnet. Durch die verbundenen blauen Quadrate sind die Messpunkte dargestellt. In diesem Fall ist der berechnete Weg lang genug gewesen, um alle Segmente aufnehmen zu können.

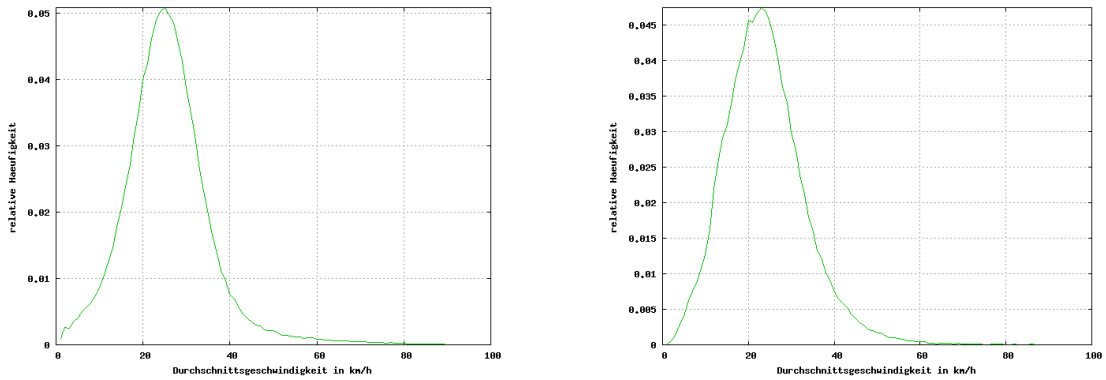
7.2.2 Simulation über einen Zeitraum

Bei der Simulation über einen gewissen Zeitraum hinweg werden fortlaufend einzelne Trajektorien simuliert. Dabei werden die Abstände der Trajektorien über eine zufällige Wahl eines Abstandes gemäß der relativen Häufigkeit von Startabständen zweier aufeinanderfolgender Trajektorien gewählt. Wie bereits erwähnt, sind diese relativen Häufigkeiten in den jeweiligen Stunden eines Tages verschieden. Durch Berücksichtigung dieses Gedankens werden z.B. nachts weniger Trajektorien als tagsüber simuliert, um so die Taxi-Daten möglichst gut nachzubilden.

Abbildung 7.6: Beispiel einer simulierten Trajektorie

7.3 Ergebnisse der Simulation

Es wurden 30 vollständige Werktage simuliert. Dabei wurden von der Simulation etwas mehr als eine Million Messpunkte aufgezeichnet. Diese Messpunkte wurden nun exakt der gleichen Bearbeitung wie die originalen Daten unterworfen. Es entstanden etwa 160.000 Trajektorien, die insgesamt aus etwa 800.000 Segmenten bestehen. Die durchschnitliche Anzahl der Segmente pro simulierter Trajektorie beträgt damit nur etwa fünf, während im originalen Datensatz durchschnittlich neun Segmente eine Trajektorie bilden. Diese Diskrepanz ist mehreren Umständen geschuldet: einerseits wird konstruktionsbedingt keine simulierte Trajektorie bis direkt zum Rand des Beobachtungsfensters führen können, da, wie oben beschrieben, der Zielpunkt ein Stück Richtung Mitte des Beobachtungsfensters zurückverlegt wird. Andererseits gibt es im originalen Datensatz Trajektorien mit deutlich mehr als 100 Segmenten. Dabei handelt es sich nicht um ein-



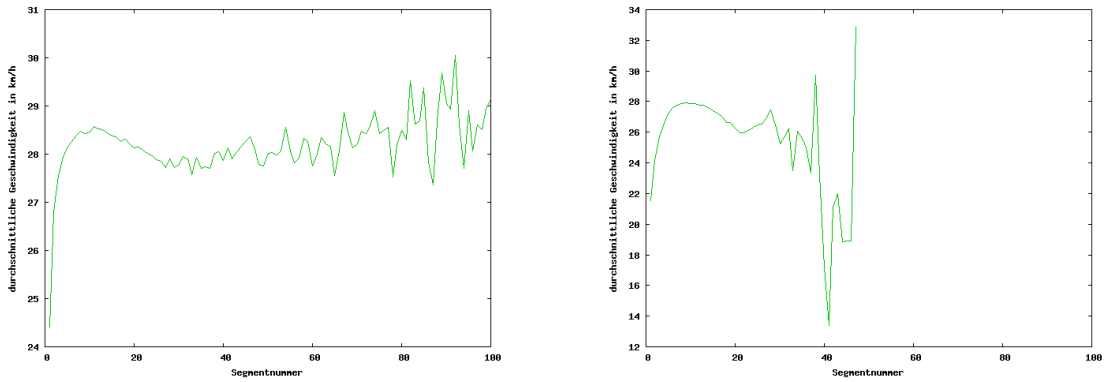
(a) originale Daten (b) simulierte Daten

Abbildung 7.7: Vergleich der Durchschnittsgeschwindigkeit tatsächlicher und simulierter Trajektorien

fache Fahrten, sondern um Rundfahrten durch das Beobachtungsfenster oder ggf. auch Fehlbedienungen des Aufzeichnungssystems[2]. In der Simulation existieren keine Trajektorien mit mehr als 47 Segmenten, d.h. die Konstruktion wird auch nicht viel mehr zulassen. Um viele Segmente einer Trajektorie überhaupt zu ermöglichen, muss die simulierte Trajektorie zwischen den diagonal entfernten Ecken des Randbereiches verlaufen. Zum Vergleich: in den originalen Daten existiert eine Trajektorie mit 383 Segmenten. Abgesehen von diesem Umstand zeigen die Bilder verschiedener Parameter eine sehr gute Übereinstimmung zwischen originalen und simulierten Daten. Als Vergleichsparameter dienen nur die Parameter, die nicht in die Simulation einflossen. Dargestellt werden immer zwei Diagramme, die zum einen den Parameter in den originalen als auch in den simulierten Daten zeigen.

Durchschnittsgeschwindigkeit der Trajektorien

Vergleicht man die beiden Diagramme in Abb. 7.7, so kann man nur sehr geringe Unterschiede feststellen. Die Form beider Kurven ist nahezu identisch und auch die maximale Häufigkeit liegt in etwa in der selben Geschwindigkeit vor.



(a) originale Daten (b) simulierte Daten

Abbildung 7.8: Vergleich der Durchschnittsgeschwindigkeit tatsächlicher und simulierter Segmente

Durchschnittsgeschwindigkeit der Segmente

Beide Diagramme sind in Abb. 7.8 zu sehen. Bis in etwa zum 20. Segment zeigen beide Abbildungen die gleiche Charakteristik. Da, wie oben bereits erwähnt, deutlich weniger Segmente pro simulierter Trajektorie im Vergleich zu den realen Trajektorien existieren, kommt es entsprechend früher zu dem sprunghaften Verhalten, das dann nach dem 39. Segment abbricht, da dann in keiner simulierten Trajektorie mehr Segmente vorhanden sind.

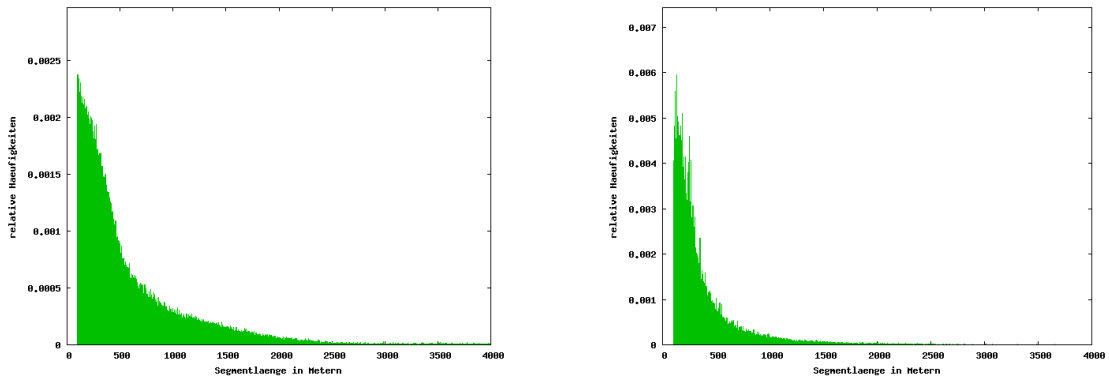
Länge eines Segments

Vergleicht man die Längen tatsächlicher und simulierter Segmente, so stellt man ausgehend von Abb. 7.9 auf der nächsten Seite fest, dass auch hier die Charakteristik der relativen Häufigkeiten gleich sind. Die simulierten Segmente scheinen jedoch tendenziell ein wenig kürzer zu sein. Die Unterschiede sind aber eher gering.

Differenzen der Segmentwinkel

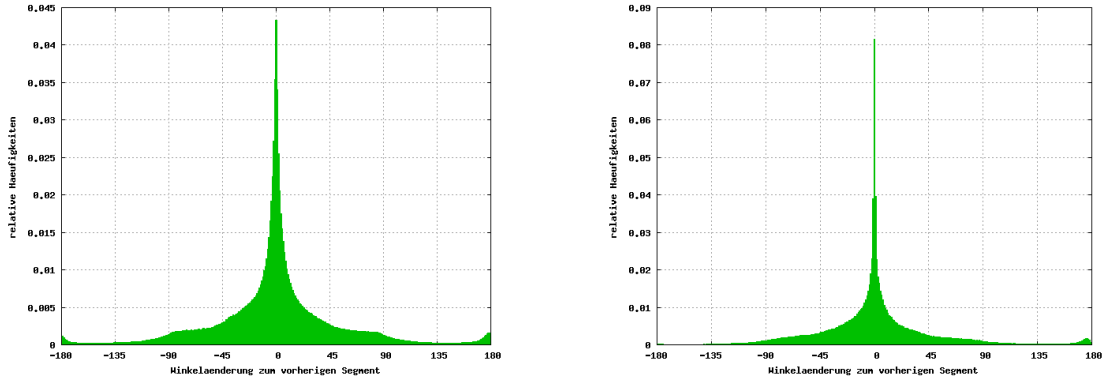
Von besonderer Bedeutung für die Güte der Simulation ist die relative Häufigkeit der Differenzen zweier aufeinanderfolgender Segmente.
Die Ecken des Graphen unterteilen den Weg von der Start- zur Endecke in Abschnitte.

[2] Ändert beispielsweise ein Fahrer nie den Status und bewegt sein Taxi hinreichend oft, so wird dieser Polygonzug nur sehr selten in Trajektorien unterteilt.



(a) originale Daten (b) simulierte Daten

Abbildung 7.9: Vergleich der Längen tatsächlicher und simulierter Segmente



(a) originale Daten (b) simulierte Daten

Abbildung 7.10: Vergleich der Winkeldifferenzen tatsächlicher und simulierter Segmente

Aus Sicht des Graphen ist keine feinere Unterteilung des Weges möglich. Die Messpunkte stellen auch eine Unterteilung des Weges dar, doch ist diese i.a. weniger fein als die durch Ecken. Andererseits sind die Messpunkte nicht an den Graphen geknüpft, d.h. ein Messpunkt wird nicht nur in Ecken auftauchen, sondern sogar oft mitten auf einer Kante des Graphen. Da in den Ecken die heuristische Information linear in den Entscheidungsprozess einfließt, stellt sich nun die Frage, welche Winkeldifferenzen zwischen zwei Messpunkten besonders häufig auftreten bzw. ob der lineare Einfluß der Heuristik Auswirkungen darauf haben kann. In Abb. 7.10 ist ganz klar zu erkennen, dass auch in den simulierten Segmenten die Winkeländerungen zum Vorgängersegment sehr ähnlich zu denen der Originaldaten verteilt sind. Auch der kleine Peak bei etwa 180° ist wieder-

um zu beobachten.
Insgesamt zeigen die simulierten Daten eine sehr gute Übereinstimmung zu den Originaldaten in den betrachteten Parametern. Einzig die Anzahl der Segmente pro Trajektorie sollte angepaßt werden. Entweder verbessert man die Qualität der Eingabedaten (z.B. durch Entfernung der unrealistischen Trajektorien mit mehr als 100 Segmenten) oder man setzt die zufällige gezogene Zahl der Segmente einer simulierten Trajektorie höher an, so dass sie der tatsächlichen Anzahl näher kommen wird.

8 Zusammenfassung und Ausblick

In der Arbeit wurde die Metaheuristik Ameisenalgorithmus ausführlich hergeleitet und besprochen. Anhand eines Graphen, der das Straßensystem abbildet, wurde die Möglichkeit vorgestellt, mit dem Ameisenalgorithmus das Kürzeste-Wege-Problem zu lösen. Dieses Verfahren diente in einem nächsten Schritt zur Simulation des Verkehrs in Ballungsräumen am Beispiel Berlin. Die Ergebnisse der Simulation erwiesen sich als sehr ähnlich zu den in der Vergangenheit beobachteten Daten.

Es gibt eine ganze Reihe von Dingen, die in der Fortführung dieser Arbeit betrachtet werden können. Einige wurden auch schon in den vorigen Kapiteln angedeutet.

Klassifizierung von Straßen

Im verwendeten Straßennetz wurden keine Unterschiede zwischen Autobahnen, Durchgangs- und Nebenstraßen gemacht. Tatsächlich spielt diese Unterscheidung eine große Rolle: die Geschwindigkeit auf höherrangigen Straßen ist i.a. höher als auf anderen Straßen. Je weiter nun Start- und Zielpunkt voneinander entfernt sind, desto eher wird zur schnellen Überbrückung auf hochrangige Straßen zurückgegriffen. Zu diesem Zweck enthalten beispielsweise Autoreiseatlanten in der Regel Übersichtskarten mit sehr großen Maßstäben, auf denen lediglich Autobahnen und wichtige Bundesstraßen enthalten sind. Zwar ist das Beobachtungsfenster in Berlin eher klein, aber innerhalb diesen Ausschnittes gibt es auch die Stadtautobahn und viele weitere Straßen mit mehreren Fahrspuren pro Richtung und Trennung der Richtungsfahrbahnen.
Falls man eine Klassifizierung von Straßen vornehmen kann, so bieten sich zwei verschiedene Möglichkeiten an, diese im Sinne des Ameisenalgorithmus' zu verwerten:

- Statt wie bisher alle Kanten des Graphen mit dem gleichen Pheromonwert zu initialisieren, können nun höherrangige Straßen bzw. die entsprechenden Kanten einen höheren Ausgangswert zugewiesen bekommen;
- Bislang fließt in die Heuristik nur der Winkel zwischen Kante und Verbindungslinie

zum Zielpunkt ein. Eine kleine Straße, die direkt in die Zielrichtung führt, wird somit tendenziell einem Autobahnstück vorgezogen, das vielleicht einen zugeordneten Winkel von 20° aufweist. Ob die Benutzung der Autobahn tatsächlich besser ist oder nicht, hängt sehr vom Abstand zum Ziel ab: befindet sich die Ecke, in der die Wahl auftritt, in unmittelbarer Nähe zum Ziel, ist vermutlich die Autobahn die schlechtere Wahl, da dort die nächste Ausfahrt vielleicht erst in 2 km Entfernung ist und man damit sich möglicherweise bereits wieder 1,7 km vom Ziel entfernt hat. Ist hingegen noch eine große Strecke zurückzulegen, kann man das auf der Autobahn oft in sehr viel kürzerer Zeit tun und somit einen etwas schlechteren Winkel zum Ziel in Kauf nehmen. Hier wäre also die heuristische Funktion η mit den drei Argumenten Winkel, Straßenklassifizierung und Luftlinienentfernung[1] zu bestimmen.

Bei beiden Möglichkeiten muss aber vermutlich sehr intensiv getestet werden, um gute Ergebnisse erzielen zu können.

Zusammenfassung von Kanten

Zur Kurvenmodellierung werden im bisher verwendeteten Graphen mehrere Ecken gebraucht, die aber nur jeweils eine weiterführende Kante aufweisen. Aus Sicht der Entscheidungsfindung wäre es wichtig zu wissen, wohin die Kandidatenkante tatsächlich führt, d.h. im Falle einer zusammengesetzten Kurve, wo bzw. in welcher Richtung die nächste Entscheidungsmöglichkeit liegt.

Andere Großstädte

Bisher wurden nur Daten aus dem Berliner Raum bzw. das Straßensystem Berlins betrachtet. Es bietet sich an, auch andere Städte bzw. Ballungszentren zu betrachten und zu testen, ob die Auswertung der Daten ähnliche Ergebnisse liefert. Wie bereits erwähnt besteht die Vermutung, dass die Winkel der Trajektorien bzw. die der Segmente stark von der Stadt bzw. dem Straßensystem dieser Stadt abhängen. Erst Vergleiche mit anderen Städten können eine empirische Bestätigung dieser Vermutung liefern.

[1] Die tatsächliche Entfernung soll erst noch bestimmt werden und so bietet sich die Luftlinienentfernung, ggf. versehen mit einem Korrekturfaktor, als Schätzer dafür an.

Lokale Optimierung von gefundenen Routen

Sofern es der Laufzeit des gesamten Ameisenalgorithmus nicht besonders abträglich ist, stellt eine Optimierung der von einer Ameise konstruierten Route z.B. durch Lokale Suche, vgl. Algorithmus 2 auf Seite 21, eine lohnenswerte Erweiterung des Ameisenalgorithmus' dar.

Verknüpfung von langfristigen Mittelwertkarten mit aktuellen Daten

In einer praktischen Anwendung dieser Arbeit wäre das langfristige Geschwindigkeitsmittel zwar sehr interessant, aber es sollten Möglichkeiten geschaffen werden, aktuelle Informationen in die Mittelwertkarten einfließen zu lassen. Dabei wird eine aktuelle Information eines Pixels nicht nur für das eine Pixel in der Mittelwertkarte relevant sein, sondern der Einflußbereich der Änderung ist zu berechnen und die langfristigen Informationen in diesem Bereich sind neu zu bewerten (vgl. [Brax05]).
Ist diese Verknüpfung von Daten möglich, so kann die Simulation des Verkehrsgeschehens, z.B. durch Extrapolation aktueller Fahrten der Verkehrsteilnehmer, direkt in aktuelle Verkehrskarten einfließen und die Berechnung individueller Fahrtrouten auf diesen zusammengesetzten Informationen beruhen.

Literaturverzeichnis

[Bran94] Brandstädt, A.: Graphen und Algorithmen. Teubner, Stuttgart, 1994.

[Brax05] Braxmeier, H., Schmidt, V., Spodarev, E.: Kriged Road-Traffic Maps. In: Springer Series: Advances in Spatial Science, S. 39 - 52, Berlin, 2005.

[Bre65] Bresenham, J.E.: Algorithm for computer control of a digital plotter. In: IBM Systems Journal, Bd. 4 (1), S. 2530, 1965.

[Bul99a] Bullnheimer, B., Hartl F., Strauss, C.: A new rank-based version for the Ant System: A computational study. In: Central European Journal for Operations Research and Economics, Band 7(1), Seite 25-38, 1999.

[Bul99b] Bullnheimer, B., Hartl F., Strauss, C.: An Improved Ant System algorithm for the Vehicle Routing Problem. In: Annals of Operations Research Bd. 89, Seiten 319 - 328.

[Cru98] Cruse, H., Dean, J., Ritter, H.: Die Entdeckung der Intelligenz oder Können Ameisen denken? Intelligenz bei Tieren und Menschen. Verlag C.H. Beck, München, 1998.

[Dor92] Dorigo, M.: Optimization, Learning and Natural Algorithms. Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milan, 1992.

[Dor04] Dorigo, M, Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, London, 2004.

[Hro01] Hromkovič, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation and Heuristics, Springer, Berlin, Heidelberg, New York et. al., 2001.

[Jun90] Jungnickel, D.: Graphen, Netzwerke und Algorithmen. BI-Wissenschaftsverlag Mannheim, Wien, Zürich, 1990.

[Sch97] Schöning, U.: Algorithmen - kurzgefaßt. Spektrum Akademischer Verlag, Heidelberg, Berlin, 1997.

[Stü96] Stützle, T., Hoos, H. H.: Improving the Ant System: A detailed Report on the MAX-Min Ant System. In: Technical report AIDA-96-12, FG Intellektik, FB Informatik, TU Darmstadt, 1996.

[Vol91] Volkmann, L.: Graphen und Digraphen. Eine Einführung in die Graphentheorie, Springer Verlag Wien, New York, 1991.

[Wal05] Walch, P.: Genetische Algorithmen zur Lösung von Stackelberg-Spielen. Diplomarbeit, Universität Ulm, 2005.

[geo] http://www.geostoch.de, 16. Juni 2007.

[wik] http://de.wikipedia.org/wiki/UTM-Koordinatensystem, 7. Juni 2007.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ich bin mir bewußt, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Ulm, 26. Juni 2007

(Johannes Renfordt)