Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Design einer Mitfahrerbörse auf Basis graphentheoretischer Verfahren

Gunnar Steffen
am 29. Oktober 2004

Studiengang Softwaretechnik

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Guido Pfeiffer

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Gunnar Steffen

Design einer Mitfahrerbörse
auf Basis graphentheoretischer Verfahren

Design einer Mitfahrerbörse auf Basis graphentheoretischer Verfahren

Stichworte Graphentheorie, Modellbildung, dynamischer Graph, spontane Fahrgemeinschaften

Zusammenfassung

Die Hauptverkehrsadern der meisten Ballungszentren sind zu den Rushhours überlastet. Ein Ansatzpunkt, die Strassen zu entlasten, ist die Förderung von Fahrgemeinschaften. In dieser Arbeit wird ein System zur spontanen Vermittlung von Fahrgemeinschaften entworfen. Die angebotenen Mitfahrgelegenheiten können durch einen dynamischen Graphen abgebildet werden. Ein wesentlicher Bestandteil der Modellbildung ist es, die dynamischen Aspekte soweit zu kapseln, dass bekannte, effiziente Shortest-Path-Algorithmen mit geringfügigen Modifikationen für das Routenmatching eingesetzt werden können. Anhand einer prototypischen Implementierung wird gezeigt, dass das Problem des Routenmatching effektiv gelöst werden kann und die Vermittlung spontaner Mitfahrgelegenheiten somit technisch realisierbar ist.

Design of an instant ride sharing application based on methods of operational research

Keywords Operational research, data modelling, dynamic networks, instant ride sharing, casual car pooling

Abstract

The main roads of most urban areas suffer from traffic congestion during peak hours. One approach to relieve this strain on mobility is to expedite the formation of ride sharing pools. In this study, a system for instant ride sharing was developed. The offered rides can be represented as dynamic networks. An integral part of model building is to encapsulate the dynamic aspects in a way that allows the use of established, slightly modified, effi-cient Shortest-Path-Algorithms for route matching. As demonstrated by the developed prototype, the problem of route matching can be solved efficiently. Thus, an application for instant ride sharing is feasible from the technical point of view.

Inhaltsverzeichnis

1 Einleitung

Der Berufsverkehr hat in den letzten Jahren besonders in den Ballungsräumen erheblich zugenommen. Die Folge sind Staus, Umweltverschmutzung und Stress.

1.1 Motivation

In einer verkehrswissenschaftlichen Studie (Halbritter u. a. 2002) heißt es: „Säßen zum Beispiel in jedem Auto zwei Personen statt einer, würde sich die Fahrzeit im Berufsverkehr um ein Drittel reduzieren, bei einem realistischeren Wert von durchschnittlich 1,4 Personen immer noch um ein Fünftel, einhergehend mit Emissionsminderungen von 20 bis 35 Prozent.“

Nach einer empirischen Studie des Deutschen Verkehrssicherheitsrates (Welk und Kage 2000) leiden Pendler, die den eigenen Pkw nutzen, häufig unter Konzentrationsstörungen und Nervosität. Für den Arbeitgeber bedeutet das Vorhalten großflächiger Parkplätze einen beträchtlichen Kostenfaktor. Das betriebliche Mobilitätsmanagement wird als innovativer Ansatz gesehen, der es durch Organisation und Einführung neuer Dienstleistungen ermöglichen soll, dass sowohl Arbeitnehmer als auch Arbeitgeber profitieren.

Eine der in diesem Zusammenhang immer wieder vorgeschlagenen Maßnahmen ist die Förderung von Fahrgemeinschaften. Die Bildung von Fahrgemeinschaften im herkömmlichen Sinn bedeutet aber für alle Teilnehmer das Eingehen von Verbindlichkeiten. Die Zunahme von flexiblen Arbeitszeiten steht im Gegensatz dazu. Mit Zunahme der Freizeitaktivitäten beinhaltet der Tagesablauf vieler Berufstätiger immer weniger Pendlervorgänge im eigentlichen Sinn, sondern besteht vielmehr aus Wegketten und lässt sich deshalb in der Regel nicht mit dem herkömmlichen Konzept von Fahrgemeinschaften vereinbaren.

Ein Ansatz, die Nutzung von Fahrgemeinschaften mit dem heutigen Mobiltätsverhalten in Einklang zu bringen, ist die Echtzeitvermittlung von Fahrgemeinschaften. Ähnlich wie bei der Benutzung von Taxis soll es möglich sein, zeitnah Mitfahrgelegenheiten zu finden und in Anspruch zu nehmen. Durch Einbindung des öffentlichen Personennahverkehrs kann die Erfolgswahrscheinlichkeit, eine passende Verbindung zu finden, deutlich erhöht werden. Zudem sollte die Fahrgemeinschaftsvermittlung nicht auf den betrieblichen Einsatz beschränkt

werden. Das Angebot einer öffentlichen Mitfahrerbörse bringt einen wesentlich größeren Kreis von Nutzern und damit ein besseres Streckenangebot mit sich. Nur durch das öffentliche Angebot einer Mitfahrerbörse besteht die Möglichkeit, auch die durch Freizeitaktivitäten verursachten Wege über Mitfahrgelegenheiten abzudecken. Um eine echte Alternative zum eigenen PKW bieten zu können, muss der Koordinationsaufwand für Personen, die eine Mitfahrgelegenheit suchen, möglichst gering sein. Aber auch für Fahrer sollte das Angebot von Mitfahrmöglichkeiten Vorteile bringen. Insbesondere dürfen aber für Fahrer keine Nachteile in Form von Wartezeiten bzw. Umwegen entstehen.

1.2 Zielsetzung

In dieser Arbeit wird die technische Realisierbarkeit eines Systems zur Vermittlung spontaner Mitfahrgelegenheiten untersucht. Durch das Routenmatching wird eine Kombination aus angebotenen Routen ermittelt, die einen Weg vom Startpunkt einer Anfrage zu ihrem Zielpunkt bilden. Die graphentheoretische Verfahren zur Realisierung des Routenmatchings sind hier besonders wichtig.

Ziel ist es, ein performantes, objektorientiertes Modell zur Darstellung von dynamischen Graphen zu entwickeln, in das die angebotenen Routen übertragen werden und auf dem aufbauend verschiedene Algorithmen implementiert werden können. Vor dem Hintergrund der Mitfahrerbörse werden die Vor- und Nachteile verschiedener Shortest-Path-Algorithmen erörtert.

Die Datengrundlage des Systems wird von einem Geoinformationssystem gebildet, das den genauen Verlauf der angebotenen Strecken ermittelt. Das geplante System soll in kürzester Zeit qualitativ hochwertige Ergebnisse liefern. Um eine gute Ausnutzung der Angebote zu gewährleisten, werden die Angebote als Streckennetz betrachtet. Das Ein- und Aussteigen ist dabei an jedem beliebigen Punkt einer Strecke möglich. Das Systemdesign soll von dem verwendeten Geoinformationssystem unabhängig sein.

1.3 Gliederung

In Kapitel 2 (Analyse) wird zunächst ein Anforderungskatalog erstellt und anhand von Szenarien und Usecases konkretisiert.

Die Gesamtarchitektur des Systems wird in Kapitel 3 (Design) vorgestellt. Neben der Entwicklung eines Oberflächenprototyps werden in diesem Kapitel ausführlich das graphentheoretische Modell sowie die Eignung verschiedener Shortest-Path-Algorithmen besprochen.

Daraufhin wird in Kapitel 4 (Evaluierung) das entwickelte Modell unter dem Aspekt der Performance untersucht. Diese Untersuchung beinhaltet zum einen die Messung konstruierter Testfälle, zum anderen die Messung von Testfällen, denen Daten aus der Metropolregion Hamburg zu Grunde liegen.

Abschließend wird in Kapitel 5 (Diskussion) auf Grund der vorliegenden Messergebnisse sowie des Anforderungskatalogs die Realisierbarkeit des Systems in verschiedenen Ausprägungen diskutiert.

2 Analyse

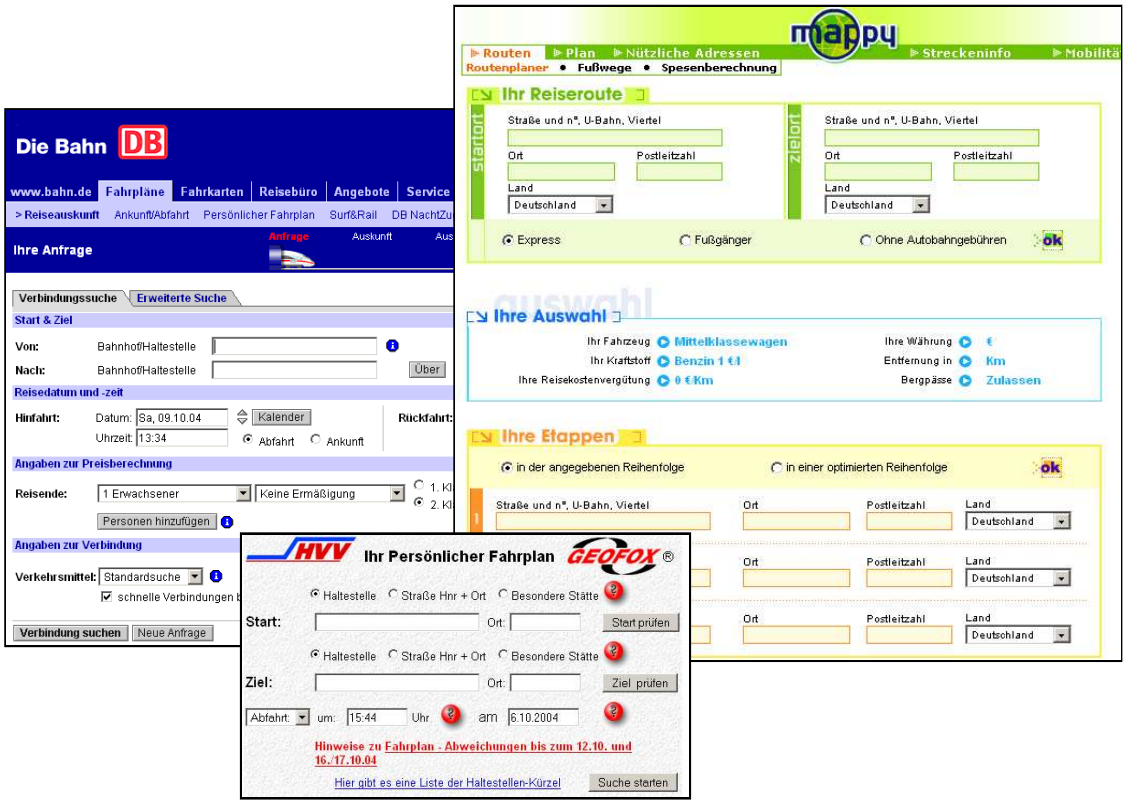


Abbildung 2.1: Online Fahrtplanung: Benutzerschnittstellen etablierter Systeme

Systeme, mit denen man Reisen online planen kann, sind heutzutage weitgehend etabliert. Routenplaner, Fahrplanauskünfte von Nahverkehrsverbänden oder das Buchungssystem der Deutschen Bahn AG sind bekannte Vertreter. In Abbildung 2.1 sind die entsprechenden Seiten der Bahn[1], des Routenplaners mappy[2], und der Fahrplanauskunft des HVV[3] dargestellt.

[1] Onlinebuchungssystem der Deutschen Bahn AG, erreichbar unter: www.bahn.de
[2] Mappy, erreichbar unter: www.mappy.de
[3] Fahrplanauskunft des HVV, erreichbar unter: www.geofox.de

Gemeinsam haben alle diese Systeme, dass sie darauf ausgelegt sind, von jedermann genutzt zu werden. Von den Benutzern wird keinerlei spezifisches Fachwissen verlangt. Je nach speziellem Einsatzbereich wird versucht, dem Benutzer möglichst wenige Angaben abzuverlangen und - soweit möglich - sinnvolle Voreinstellungen anzubieten. Da es sich bei der Mitfahrerbörse um ein System mit ähnlichem Benutzerkreis handelt, sollte auch die angebotene Benutzerschnittstelle ähnlich aufgebaut sein.

Da die Vermittlung der Mitfahrgelegenheiten spontan erfolgen soll, ist es besonders wichtig, dass das System für den Benutzer in nahezu jeder Situation zugänglich ist. Das System soll daher insbesondere auch mobile Endgeräte unterstützen. Eine erste Vision des zu entwickelnden Systems wird in Abbildung 2.2 gezeigt.

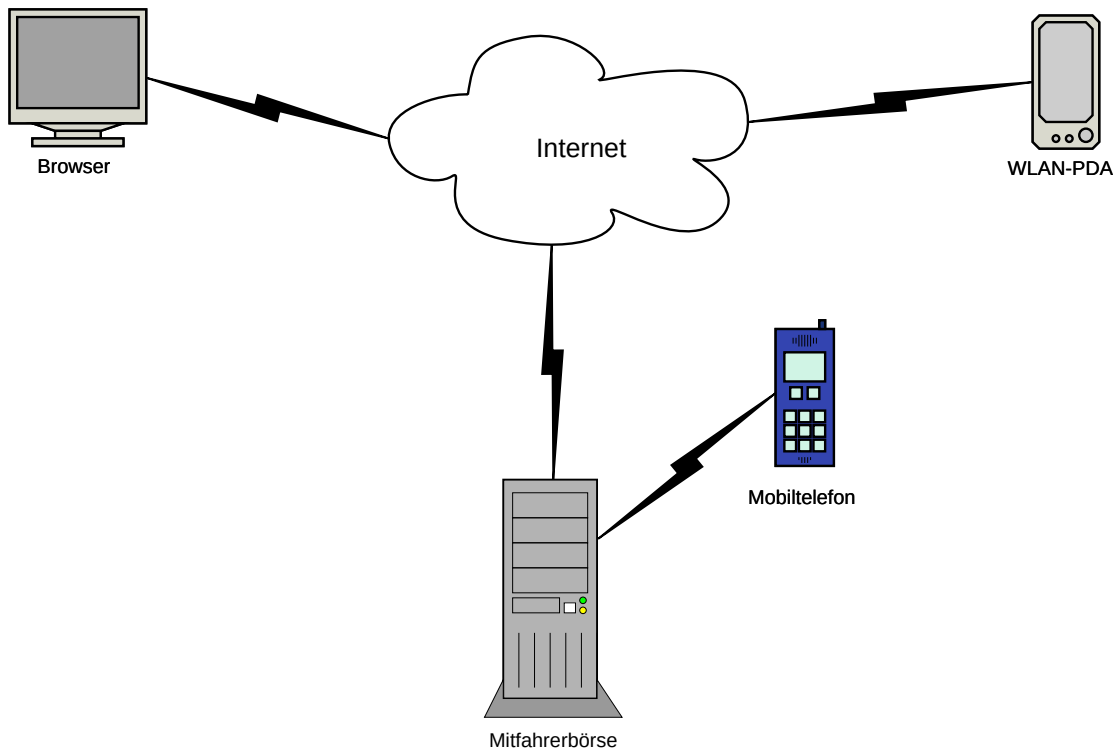


Abbildung 2.2: Kommunikationsmodell

2.1 Anforderungen

Das System muss in der Lage sein, Mitfahrgelegenheiten zu vermitteln. Dabei soll es einen Reisewunsch in mehrere Abschnitte aufteilen und den Wechsel von Verkehrsmitteln unterstützen können. Lösungen mit wenigen Umstiegspunkten sind aber zu bevorzugen, da häufiges Umsteigen aufgrund des erhöhten Koordinationsaufwands vom Benutzer als unangenehm empfunden wird. Weiterhin soll das Ergebnis durch Präferenzen des Benutzers beeinflussbar sein[4]. Der Wechsel von Verkehrsmitteln bezieht sich zunächst nur auf die Benutzung verschiedener privater PKW's. Eine Einbindung des öffentlichen Personennahverkehrs soll aber möglich sein.

Da es sich bei der Mitfahrerbörse um einen On-demand-Service handelt, sind für die Qualität des Systems die Ermittlungsgeschwindigkeit und die Güte der Ergebnisse von entscheidender Bedeutung. Es besteht sonst die Gefahr, dass durch kurze Wartebereitschaft seitens der Benutzer mögliche Vermittlungen nicht zustande kommen.

Aus Schätzungen der eNotions Gmbh (Homann 2004) geht hervor, dass eine städtische Mitfahrerbörse mindestens 5000 Mitglieder haben muss, damit ein flächendeckender Service angeboten werden kann. Es wird davon ausgegangen, dass von diesen 5000 Mitgliedern ungefähr die Hälfte die Mitfahrerbörse für den täglichen Arbeitsweg in Anspruch nimmt. Die daraus resultierenden Anfragen verteilen sich auf jeweils ca. 3 Stunden zu den Rushhours. Es kann weiterhin davon ausgegangen werden, dass Streckenangebote und Streckengesuche jeweils 50 Prozent der Anfragen ausmachen. Aus diesen Schätzungen ergibt sich, dass etwa alle 5 Sekunden eine Anfrage an das System gestellt wird. In einer Großstadt wie Hamburg haben die zurückgelegten Strecken eine durchschnittliche Länge von 14 Kilometern und werden in etwa 30 Minuten zurückgelegt. Das bedeutet, dass ein Angebot nach ca. 30 Minuten bei einer Suche nicht mehr berücksichtigt werden muss und aus dem Pool der aktiven Routen entfernt wird. Aus diesen Annahmen folgt, dass eine Mitfahrt in einem Netz aus ca. 600 Routen gefunden werden muss. Weiterhin muss eine Nachfrage in unter 5 Sekunden abgearbeitet werden, um jeweils alle aktuell angebotenen Routen berücksichtigen zu können. Dauert die Bearbeitung einer Nachfrage länger, steigt zudem die Wahrscheinlichkeit, dass das ermittelte Fahrzeug die Stelle bereits passiert hat, an der der Mitfahrer zusteigen soll. Den vorangegangenen Rechnungen liegen allerdings nur Durchschnittswerte zu Grunde. Um den Service der Mitfahrerbörse in gleichbleibender Qualität gewährleisten zu können, muss das System genügend Reserven haben, um auch das Anfragevolumen in Spitzenzeiten verarbeiten zu können.

Da die Mitfahrerbörse nicht von geschultem Fachpersonal, sondern von Gelegenheitsnutzern in Anspruch genommen wird, muss das Userinterface alle technischen Details abstrahieren.

[4] Zum Beispiel sollen auf Benutzerwunsch nur Nichtraucherfahrzeuge bei der Ergebnisfindung berücksichtigt werden.

Der User darf im Verlaufe der Nutzung nur mit ihm vertrauten Begriffen konfrontiert werden. Insbesondere gilt dies für die Benutzung der systemintern verwendeten Koordinatendarstellungen. Koordinatenpunkte müssen für den Benutzer in Adressen umgewandelt werden.

Das System soll für Benutzer möglichst jederzeit zugänglich sein. Neben der Unterstützung von Webbrowsern sollten daher insbesondere auch mobile Endgeräte für die Benutzerkommunikation in Betracht gezogen werden. Vielversprechend sind in diesem Zusammenhang zum einen WLan-fähige Geräte, zum anderen aber auch Mobiltelefone. Wie auch in (Resnick 2003) beschrieben wird, sind Mobiltelefone gerade für Fahrer die einzige Möglichkeit, um mit dem System der Mitfahrerbörse zu kommunizieren, da sie in der Regel während der Fahrt über wartende Mitfahrer informiert werden.

Um Mobiltelefone komfortabel zu unterstützen, wird die Benutzung eines Callback-Systems in Betracht gezogen. Benutzer können durch ihre Telefonnummer eindeutig identifiziert werden. Der Benutzer wählt eine Nummer, gefolgt von der Identifikationsnummer des Startpunktes und der Identifikationsnummer des Zielpunktes. Das Callback-System stellt eine Anfrage an die Mitfahrerbörse und teilt dem Benutzer ermittelte Mitfahrgelegenheiten durch Rückruf oder SMS mit. Mögliche Start- und Zielpunkte können dabei vom Benutzer über das Internet verwaltet werden.

Anforderung	Priorität
Vermittlung spontaner Mitfahrgelegenheiten	hoch
Verteilung eines Reisewunsches auf mehrere Abschnitte	
Minimierung der Antworzeiten	
Wege mit wenig Umstiegspunkten werden bevorzugt	
Einfaches, verständliches Userinterface	
Ergebnis durch den Benutzer beeinflussbar	mittel
Benutzbarkeit über mobile Devices	
Keine technischen Mindestanforderungen an Clients	
Angebot alternativer Reiserouten	niedrig
Einbindung anderer Verkehrsmittel	

Tabelle 2.1: Übersicht über die Anforderungen

2.2 Szenarien

Die folgenden Szenarien zeigen drei Beispiele für die Vermittlung von Transporten. Szenario 2.2.1 stellt die in dieser Arbeit entwickelte Mitfahrerbörse zur spontanen Vermittlung von Mitfahrgelegenheiten dar. Szenario 2.2.2 stellt die Ausweitung der Mitfahrerbörse auf überregionale Angebote dar. In diesem Szenario mischen sich Aspekte spontaner Fahrtvermittlungen mit denen längerfristiger Planungen. In Szenario 2.2.3 wird am Beispiel eines Kurierdienstes die Übertragbarkeit der hier besprochenen Problemstellung auf Transporte von Gütern dargestellt.

2.2.1 Spontane Mitfahrt

Die Mitfahrerbörse stellt Vermittlungen von Mitfahrgelegenheiten zur Verfügung. Mitglieder können verschiedene Routen bei der Mitfahrerbörse registrieren. Eine Route besteht aus einem Start- und einem Zielpunkt sowie der Angabe, ob es sich um eine Fahrt oder eine Mitfahrt handelt. Für jede Fahrt wird dem Fahrer eine Wegbeschreibung zur Verfügung gestellt. Jede Route erhält eine ID.

Eine Person fährt mit dem Auto zum Sport. Da sie diese Route bei der Mitfahrerbörse registriert hat, aktiviert sie diese Route telefonisch. Über die Nummer kann das System feststellen, um welche Route es sich handelt. Es kommt ein Ansage: „Ihre Route wurde aktiviert“.

Eine weitere Person möchte nach der Arbeit nach Hause. Da sie diese Route bei der Mitfahrerbörse registriert hat, stellt sie telefonisch eine Anfrage an die Mitfahrerbörse. Über die Nummer kann das System feststellen, um welche Route es sich handelt.

Das System ermittelt daraufhin, ob auf Grund der aktivierten Routen eine Mitfahrt vermittelt werden kann. Neben den wegplanerischen Gegebenheiten werden dabei ebenso andere Punkte aus dem Nutzerprofil berücksichtigt wie z.B. „Raucher/Nichtraucher“. Die möglichen Ergebnisse sehen wie folgt aus:

1. Es wurde eine komplette Mitfahrgelegenheit gefunden:

   Von: Stresemannstrasse 375
   Nach: Wolkausweg 1
   Abfahrt: ca.17:30h
   Ankunft: ca.18:00h
   Sie werden verbunden.

2. Es wurde ein Mitfahrgelegenheit auf einer Teilstrecke gefunden:

    Von: Stresemannstrasse 375
    Nach: U Kellinghusenstrasse
    Abfahrt: ca.17:30h
    Ankunft: ca. 17:45h
    Fahren sie mit der U1 nach U Ohlsdorf weiter
    Sie werden verbunden.

3. Es wurde keine Mitfahrmöglichkeit gefunden:

    Fahren sie mit der S11 von S Holstenstraße nach S Rübenkamp.

2.2.2 Geplante Reise

Eine Mitfahrerbörse stellt Vermittlungen von Mitfahrgelegenheiten zur Verfügung. Mitglieder können verschiedene Routen bei der Mitfahrerbörse registrieren, die sie regelmäßig fahren. Eine Route besteht aus einem Start- und einem Zielpunkt. Für jede Fahrt wird dem Fahrer eine Wegbeschreibung zur Verfügung gestellt. Jede Route erhält eine ID.

Eine Person fährt am Wochenende von Hamburg nach Frankfurt. Sie bucht eine Bahnfahrt am Freitag um 18:24h. Gegen 17:15h stellt sie per Internet eine Anfrage an die Mitfahrerbörse:

Von: Tarpenbekstrasse 8 , 22848 Norderstedt
Nach: Hbf Hamburg
Zeit: sofort
Geschlecht des Fahrers: egal
Nichtraucherfahrzeug: ja

Das System ermittelt daraufhin die günstigste Möglichkeit für diese Anfrage. Die möglichen Ergebnisse sehen wie folgt aus:

1. Es wurde eine komplette Mitfahrmöglichkeit gefunden:

Von: Tarpenbekstrasse 8 , 22848 Norderstedt
Nach: Hbf Hamburg
Abfahrt: ca.17:30h
Ankunft: ca.18:10h
Fahrer: Mark Mustermann
Mobilnummer: 0172/123666
Der Fahrer wird benachrichtigt.

2. Es wurde ein Mitfahrmöglichkeit auf einer Teilstrecke gefunden:

Von: Tarpenbekstrasse 8 , 22848 Norderstedt
Nach: S Ohlsdorf
Abfahrt: ca.17:30h
Ankunft: ca. 17:45h
Fahrer: Mark Mustermann
Mobilnummer: 0172/123666
Der Fahrer wird benachrichtigt.

Von: S Ohlsdorf
Nach: Hbf Hamburg
Abfahrt: ca. 17:50
Ankunft: ca.18:05
Fahrer: S1

3. Es wurde keine Mitfahrmöglichkeit gefunden:

Von: U Ochsenzoll
Nach: Hbf Hamburg
Abfahrt: ca. 17:20
Ankunft: ca. 17:55
Fahrer: U1

Wenn ein privater Fahrer in den Fahrplan involviert ist, ist eine Absprache per Telefon über den genauen Treffpunkt notwendig. Dazu bekommen beide Parteien die jeweilige Telefonnummer. Am Hauptbahnhof Hamburg angekommen steigt die Person in ihren Zug und fährt Richtung Frankfurt. Am Frankfurter Hauptbahnhof angekommen kann sie per W-Lan eine Anfrage an die dortige Mitfahrerbörse stellen:

Von: Hbf Frankfurt
Nach: Berliner Allee 23, 63225 Langen
Zeit: sofort
Geschlecht des Fahrers: egal
Nichtraucherfahrzeug: ja

Das System ermittelt wieder die günstigste Möglichkeit, das Ziel zu erreichen. Die reisende Person kommt an ihrem Zielort an.

2.2.3 Kurierdienst

Ein Botendienst befördert Sendungen innerhalb eines Stadtgebiets. Kommt ein neuer Auftrag in der Zentrale an, wird vom System geprüft, welcher der Fahrer auf seinem aktuellen Weg in der Nähe des Abholortes vorbeikommt. Als nächstes wird geprüft, welche der Fahrer auf ihrem aktuellen Weg am Bestimmungsort der Sendung vorbeikommen. Handelt es sich dabei um den gleichen Fahrer, bekommt dieser den Auftrag, die Sendung abzuholen. Sollte es sich um einen anderen Fahrer handeln, wird geprüft, ob sich die Routen der Fahrer schneiden oder berühren. Sollte dieses der Fall sein, wird Fahrer eins beauftragt, das Paket abzuholen und an einem bestimmten Punkt an Fahrer zwei zu übergeben. Fahrer zwei wird beauftragt das Paket an einem bestimmten Punkt von Fahrer eins in Empfang zu nehmen und zu seinem Bestimmungsort zu bringen.

Beispiel 1.

> Fahrer eins befördert eine Sendung vom Wolkausweg in die Lohmühlenstraße um 13:30h. Ein zweiter Fahrer befördert eine Sendung aus der Bramfelder Landstraße in die Stresemannstraße um 13:50h.
>
> Um 13:35 geht ein weiterer Auftrag über eine Sendung vom AK-Barmbek in die Gärtnerstraße ein.
>
> Das System ermittelt, dass Fahrer eins die Sendung im AK-Barmbek einsammelt und sie ca. 14:00 am Bahnhof Barmbek an Fahrer zwei übergibt. Fahrer zwei liefert dann die Sendung auf seinem Weg zur Stresemannstraße in der Gärtnerstraße ab.

Beispiel 2.

> Fahrer eins befördert eine Sendung vom Wolkausweg in die Lohmühlenstraße um 13:30h. Ein zweiter Fahrer liefert gegen 14:00h eine Sendung im EkZ Hamburger Straße ab.
>
> Um 13:35 geht ein weiterer Auftrag über eine Sendung vom AK-Barmbek in die Gärtnerstraße ein.
>
> Das System ermittelt, dass Fahrer eins die Sendung im AK-Barmbek einsammelt und sie ca. 14:10 am EkZ Hamburger Straße an Fahrer zwei übergibt. Fahrer zwei liefert dann die Sendung in der Gärtnerstraße ab.

2.3 Usecases

Im folgenden Kapitel werden die Abläufe innerhalb der Mitfahrerbörse anhand von Usecases festgehalten. Zunächst wird das System als Übersicht dargestellt, darauf folgend werden die einzelnen Vorgänge näher erläutert.

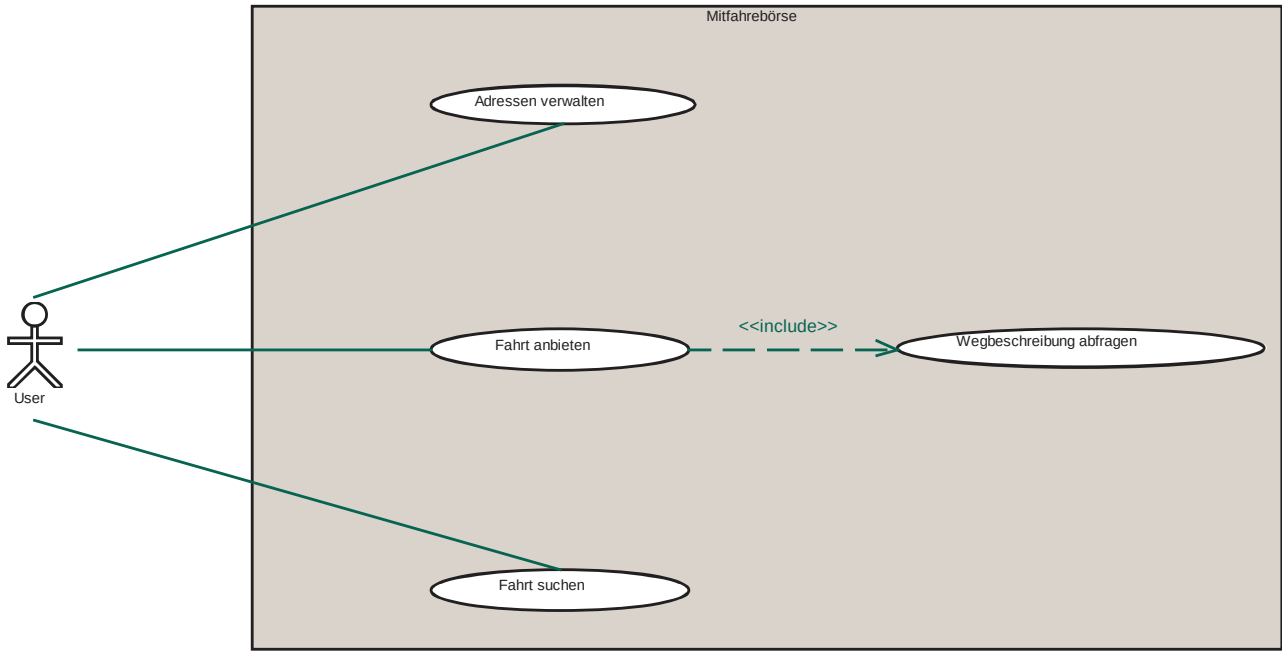


Abbildung 2.3: Usecase: Systemübersicht

Die Mitfahrerbörse vermittelt Mitfahrgelegenheiten. Ein Benutzer hat innerhalb des Systems die Möglichkeit,

- Adressen zu verwalten,
- Mitfahrgelegenheiten anzubieten und
- Mitfahrgelegenheiten zu suchen.

Wird eine Mitfahrgelegenheit angeboten, ermittelt das System dafür eine Route. Diese Route wird dem Benutzer angezeigt und ist für ihn verbindlich.

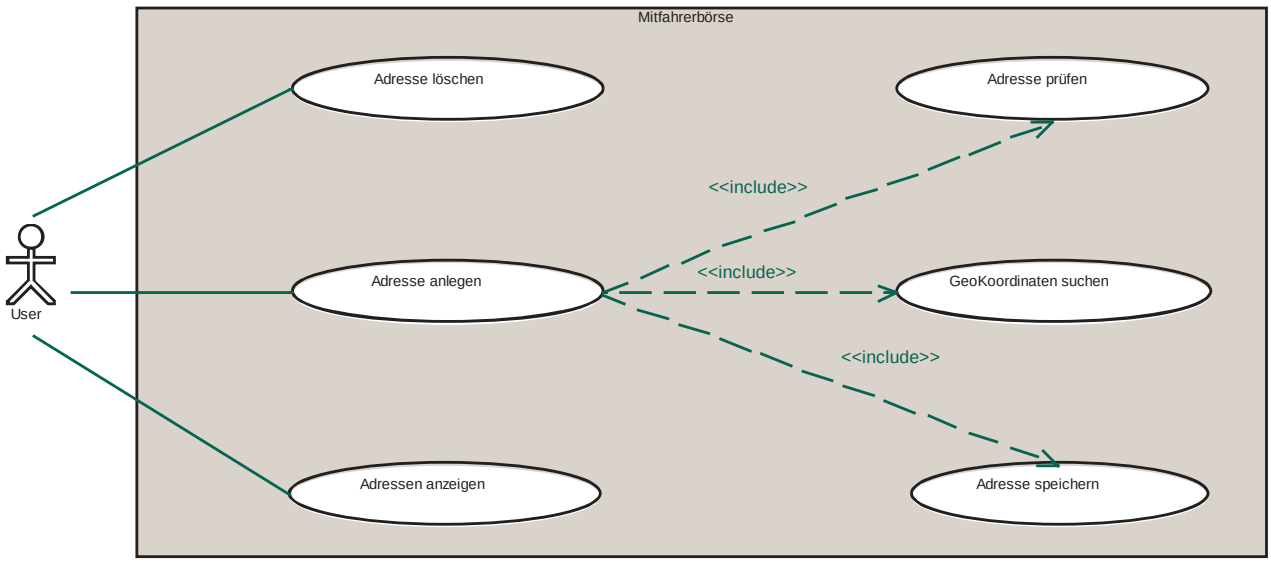


Abbildung 2.4: Usecase: Adressen verwalten

Die Verwaltung von Adressen beinhaltet die in Abbildung 2.4 dargestellten Vorgänge:

- Adressen anlegen: Legt ein Benutzer eine neue Adresse an, wird vom System zunächst geprüft, ob diese innerhalb des Gebietes der Mitfahrerbörse liegt. Ist diese Prüfung erfolgreich, werden die Geokoordinaten zu dieser Adresse gesucht. Die Adresse wird zusammen mit den ermittelten Geokoordinaten in der Datenbank gespeichert.
- Adressen löschen: Das System zeigt dem Benutzer eine Liste der für ihn gespeicherten Adressen. Daraufhin hat dieser die Möglichkeit, einzelne Einträge auszuwählen und diese aus der Datenbank zu löschen.

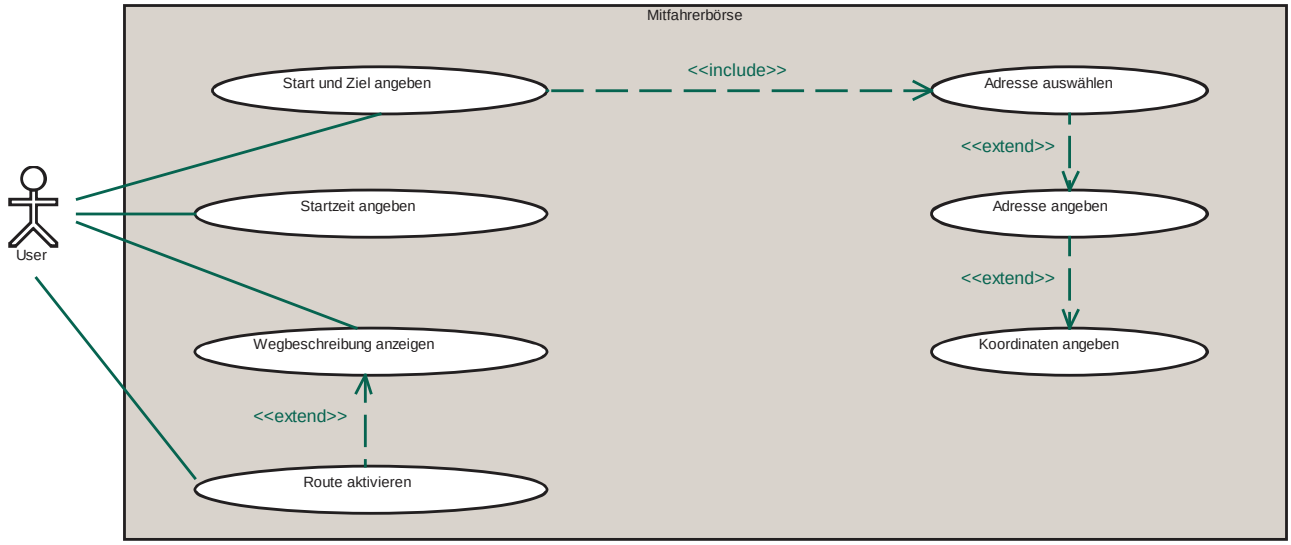


Abbildung 2.5: Usecase: Mitfahrgelegenheit anbieten

Das Anbieten von Mitfahrgelegenheiten beinhaltet die in Abbildung 2.5 dargestellten Vorgänge:

- Start und Ziel angeben: Start- und Zielpunkt können vom Benutzer entweder aus der Liste der für ihn gespeicherten Adressen gewählt oder auch neu eingegeben werden. Zu neu eingegebene Adressen müssen die Geokoordinaten ermittelt werden. Endgeräte mit GPS-Modul haben die Möglichkeit, die aktuelle Position in Form der Geokoordinaten als Startadresse anzugeben.
- Wegbeschreibung anzeigen: Dem Benutzer wird die vom Geoinformationssystem ermittelte Wegbeschreibung angezeigt. Diese Wegbeschreibung ist für ihn verbindlich, da sie als Grundlage der Vermittlung dient.
- Startzeit angeben: Der Benutzer muss eine verbindliche Startzeit angeben.
- Route aktivieren: Ist der Benutzer mit der vorgeschlagenen Route einverstanden, kann er sie akzeptieren. Damit wird sie zum Angebot freigegeben.

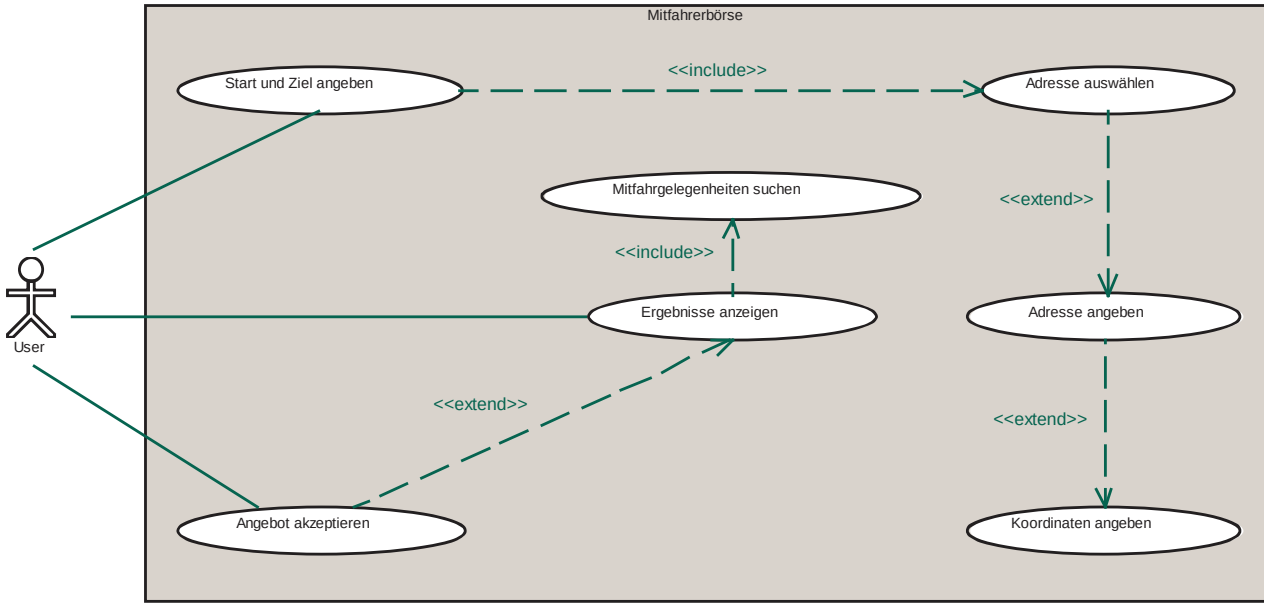


Abbildung 2.6: Usecase: Mitfahrgelegenheit suchen

Die Suche nach Mitfahrgelegenheiten beinhaltet die in Abbildung 2.6 dargestellten Vorgänge:

- Start und Ziel angeben: Start- und Zielpunkt können vom Benutzer entweder aus der Liste der für ihn gespeicherten Adressen gewählt oder auch neu eingegeben werden. Zu neu eingegebenen Adressen müssen die Geokoordinaten ermittelt werden. Der Benutzer bzw. der von ihm benutzte Client kann ebenfalls direkt Geokoordinaten angeben.
- Ergebnisse anzeigen: Nachdem das System nach passenden Mitfahrgelegenheiten gesucht hat, werden diese dem Benutzer angezeigt.
- Vorschlag akzeptieren: Ist der Benutzer mit einer der vorgeschlagenen Kombinationen von Mitfahrgelegenheiten einverstanden, kann er sie akzeptieren. Daraufhin erhält er die Kontaktdaten der Fahrer.

2.4 Planstrategien

Betrachtet man die Vermittlung von Mitfahrgelegenheiten als Planung einer Reise, kann diese Reise auf verschiedene Arten geplant werden. Im städtischen bzw. regionalen Einsatzgebiet befinden sich alle Angebote in einem Pool. Aus diesem Pool wird dann eine Kombination von Angeboten gesucht, mit der der Reiseweg beschritten werden kann. Es liegt also eine einstufige Planstrategie (siehe Abbildung 2.7 a) vor. Im überregionalen Bereich kann zwar prinzipiell genauso vorgegangen werden, doch bringt dieses Vorgehen gewisse Nachteile mit sich. Eine Planung kann zum Beispiel daran scheitern, dass kein Weg vom Startpunkt zum ersten Knoten gefunden wird. Dieses Teilstück kann aber vom Benutzer zum Beispiel durch den eigenen PKW oder durch ein Taxi bestritten werden. Es ist demnach sinnvoll, dem Benutzer auch solche Vorschläge zu unterbreiten, bei denen kein Ergebnis für eine der regionalen Strecken gefunden wurde.

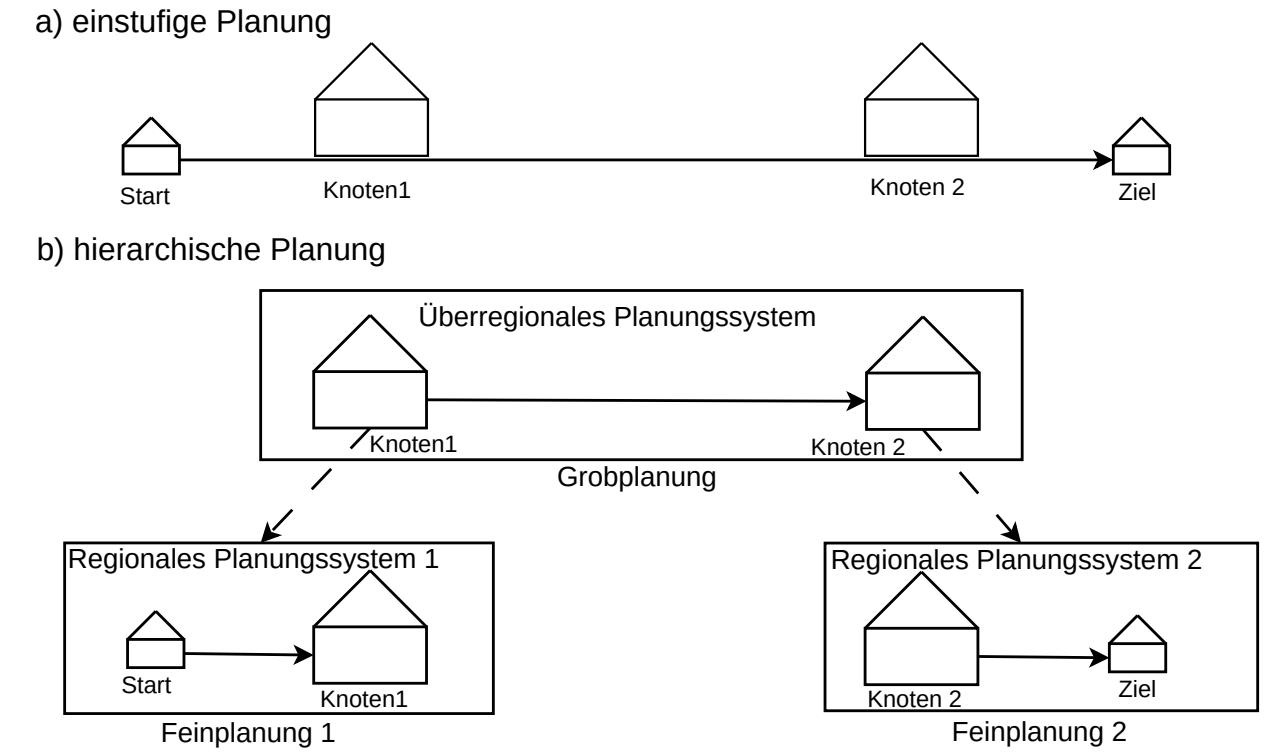


Abbildung 2.7: Planstrategien

In Szenario 2.2.2 wurde ein geplante Reise von 22848 Norderstedt (Region Hamburg) nach 63225 Langen (Region Frankfurt a.M.) beschrieben. Solche Reisen werden in der Regel nicht spontan, sondern einige Tage im Voraus gebucht. Das heißt, zum Zeitpunkt der Anfrage durch den Reisenden sind noch keine privaten, regionalen Angebote verfügbar. Eine einstufige Planstrategie versagt hier also vollständig.

Teilt man den Problemraum in mehrere Abschnitte auf, ergibt sich eine mehrstufige bzw. hierarchische Planstrategie (siehe Abbildung 2.7 b). Bei diesem Vorgehen wird nun zunächst eine Grobplanung vorgenommen. Das Ergebnis der Grobplanung beinhaltet zumindest die Überbrückung der überregionalen Strecke von Hamburg nach Frankfurt a.M. . Je nachdem, welches Verkehrsmittel zur Überbrückung dieser Strecke gewählt wurde, sind Start- und Zielpunkt innerhalb der Stadtgebiete unterschiedlich. Aufgrund dieses Planschrittes wird dann die Feinplanung innerhalb der Stadtgebiete vorgenommen. Da diese Planung erst unmittelbar vor Fahrtantritt geschehen kann, kann im Zuge der Grobplanung die Fahrt innerhalb der Stadtgebiete zunächst unter Berücksichtigung des Linienverkehrs stattfinden. Später können diese Planschritte dann noch einmal unter Berücksichtigung privater Fahrer stattfinden.

Ein weiterer Vorteil hierarchischer Planstrategien - geeignete Schnittstellen vorausgesetzt - ist, dass die einzelnen Planschritte von ansonsten unabhängigen Systemen durchgeführt und beliebig oft wiederholt werden können.

3 Design

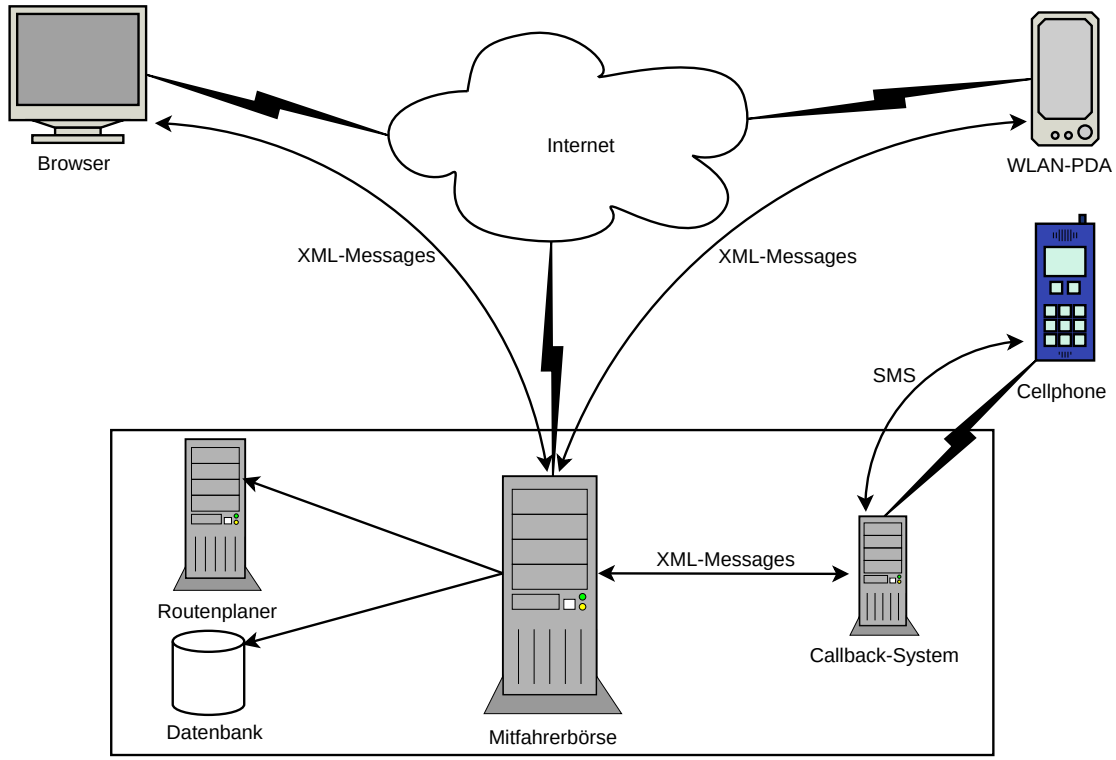


Abbildung 3.1: Kommunikationsmodell

In Abbildung 3.1 wird ein Überblick über das geplante System gegeben. Die Kommunikation mit den Clients soll über XML-Strings realisiert werden. Weiterhin wird auf ein Geoinformationssystem (GIS) und eine Datenbank zurückgegriffen. Für die Unterstützung von Mobiltelefonen wird ein Callback-System benötigt. Das Callback-System wird in dieser Arbeit aber nicht diskutiert, da es die gleiche Schnittstelle wie auch die Clients benutzten kann. Im

diesem Kapitel wird von den ermittelten Anforderungen ausgehend das System der Mitfahrerbörse entworfen. Zunächst wird beispielhaft ein ergonomisches Userinterface entwickelt. Dann werden die zur Realisierung des Systems notwendigen graphentheoretischen Methoden besprochen. Abschließend werden die Systemschnittstellen erläutert und das objektorientierte Modell vorgestellt.

3.1 Userinterface

Es gibt eine Reihe von Normen und Richtlinien für die ergonomische Dialoggestaltung. Besonders hervorzuheben ist die EN ISO 9241-10 aus dem Jahr 1996, in der allgemeine Ziele, Regeln und Empfehlungen festgehalten werden (vgl. Balzert 2000, S.551ff).

Sie erläutert folgende Grundsätze:

- Aufgabenangemessenheit,
- Selbstbeschreibungsfähigkeit,
- Steuerbarkeit,
- Erwartungskonformität,
- Fehlertoleranz,
- Individualisierbarkeit und
- Lernförderlichkeit

Viele dieser Regeln sind auf die Gestaltung von Expertensystemen ausgerichtet. Andere sind allgemeiner gültig und sollten auch bei der Gestaltung des Userinterface der Mitfahrerbörse, die hauptsächlich von Gelegenheitsnutzern verwendet wird, Beachtung finden. Insbesondere sollte hier auf Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Erwartungskonformität und Fehlertoleranz geachtet werden. Daher werden diese Punkte hier noch einmal wörtlich wiedergegeben:

Aufgabenangemessenheit

> „Ein Dialog ist aufgabenangemessen, wenn er den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen.“

Selbstbeschreibungsfähigkeit

> „Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird.“

Erwartungskonformität

> „Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht, z.B. seinen Kenntnissen aus dem Arbeitsgebiet, seiner Ausbildung und seiner Erfahrung sowie den allgemein anerkannten Konventionen“

Fehlertoleranz

> „Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.“

Folgend wird beispielhaft ein Webfrontend für die Mitfahrerbörse entwickelt. Dabei wurde auf Einhaltung der genannten Ergonomierichtlinien geachtet. Alle Adresseingaben werden zur Prüfung auf Korrektheit an das Geoinformationssystem durchgereicht.

In den Abbildungen 3.2, 3.3 und 3.4 werden die entsprechenden Dialoge zur Suche einer Mitfahrgelegenheit dargestellt. Zu diesem Zweck wurde zunächst auf ein schlichtes und klares Design geachtet. Zur guten Übersichtlichkeit wird auf eine einstufige Menüstruktur zurückgegriffen und darauf geachtet, dass die angezeigten Informationen möglichst immer komplett auf den Bildschirm passen.

Um dem Benutzer Eingaben zu erleichtern und Tippfehlern vorzubeugen, wird das Abspeichern von oft besuchten Adressen, wie in Abbildung 3.7 dargestellt, unterstützt. Diese Adressen können dann bei der Suche nach einer Mitfahrgelegenheit wieder übernommen werden[1]. In 3.3 wird das Ergebnis einer Suche dargestellt. Der Benutzer hat dann die Möglichkeit, die angebotene Lösung zu akzeptieren oder eine weitere Suche vorzunehmen. Nimmt der Benutzer das Angebot an, werden die Fahrten innerhalb der Mitfahrerbörse für ihn gebucht und, wie in Abbildung 3.4 dargestellt, die Kontaktdaten der Fahrer angezeigt. An diesem Punkt sollte dem Benutzer noch einmal die Möglichkeit gegeben werden, die Buchung zurückzuziehen, falls er mit den angezeigten Fahrern nicht fahren möchte. Das Zurückziehen sollte mit der Angabe einer Begründung versehen werden. Bei diesem Vorgang muss darauf geachtet werden, dass mit den Kontaktdaten der Fahrer kein Missbrauch getrieben werden kann. Eine denkbare Maßnahme in diesem Zusammenhang ist, dass ein Account gesperrt wird, sollte der Benutzer an diesem Punkt mehrmals hintereinander die Buchung zurückziehen.

[1] Auf Clients mit GPS-Modul wie zum Beispiel PDA’s sollte die Übernahme der aktuellen Position in das Startadressfeld unterstützt werden.

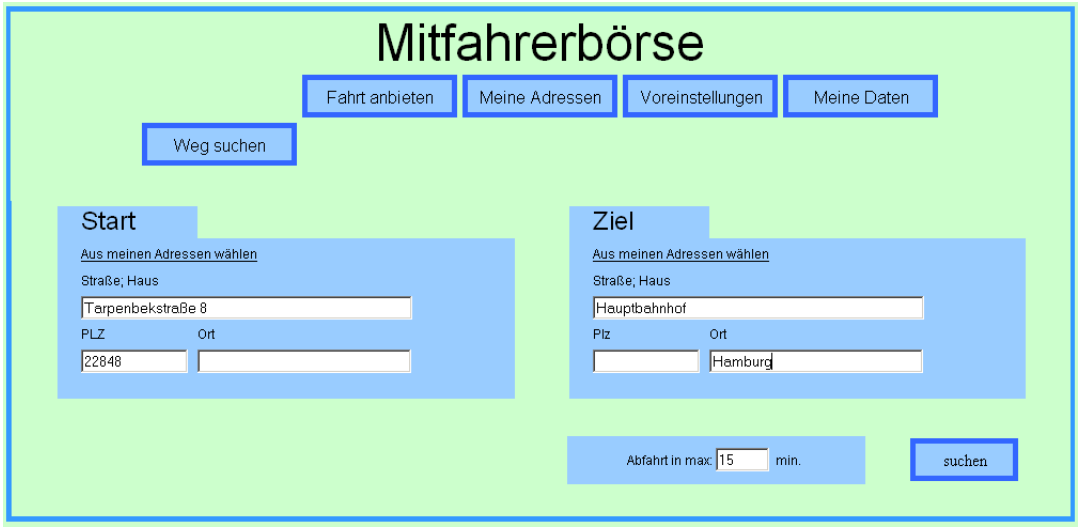


Abbildung 3.2: Screenshot: Mitfahrgelegenheit suchen (1)

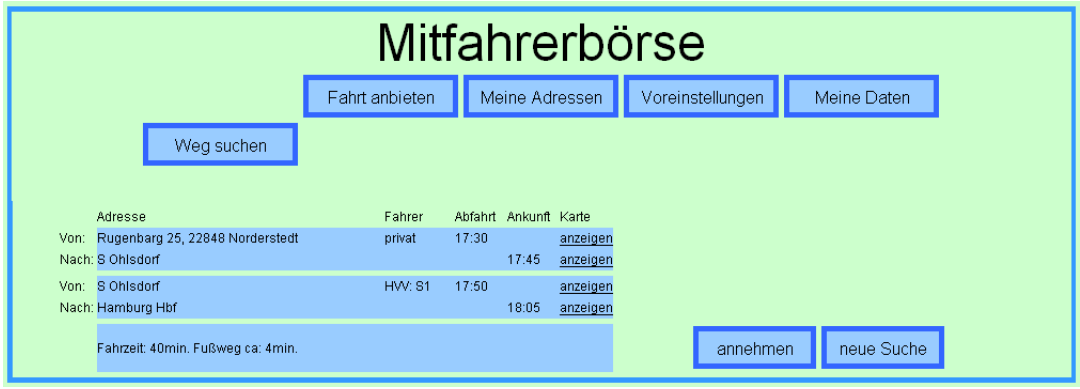


Abbildung 3.3: Screenshot: Mitfahrgelegenheit suchen (2)

Abbildung 3.4: Screenshot: Mitfahrgelegenheit suchen (3)

In den Abbildungen 3.5 und 3.6 werden die Dialoge zum Anbieten einer Mitfahrgelegenheit dargestellt. Auch hier wurden die bereits beschriebenen Designrichtlinien angewendet. Dem Anbietenden wird abschließend eine Wegbeschreibung angezeigt, die für ihn verbindlich ist. Hier ist es zudem sinnvoll, dem Benutzer den Ausdruck eines entsprechenden Kartenausschnitts mit eingezeichneter Route anzubieten.

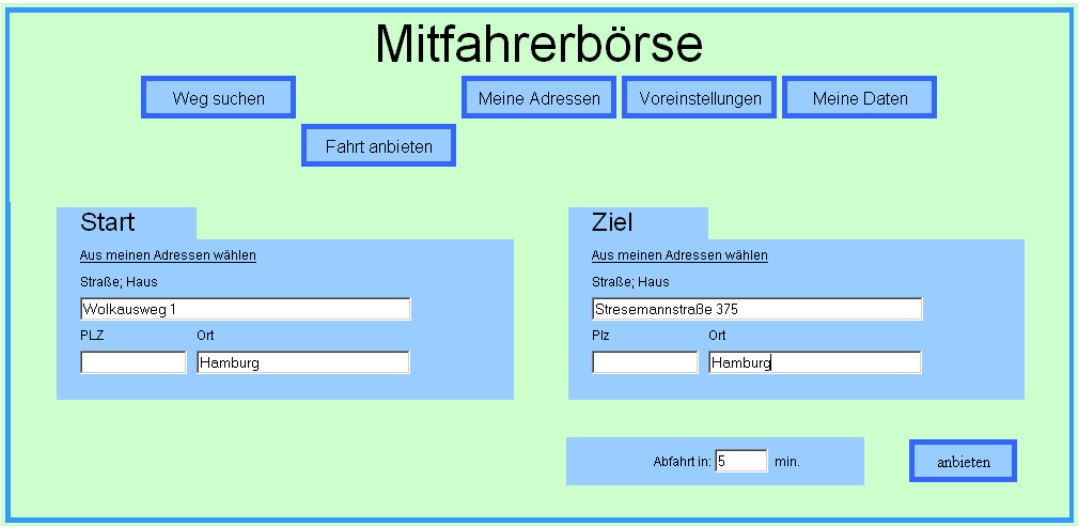


Abbildung 3.5: Screenshot: Fahrt anbieten

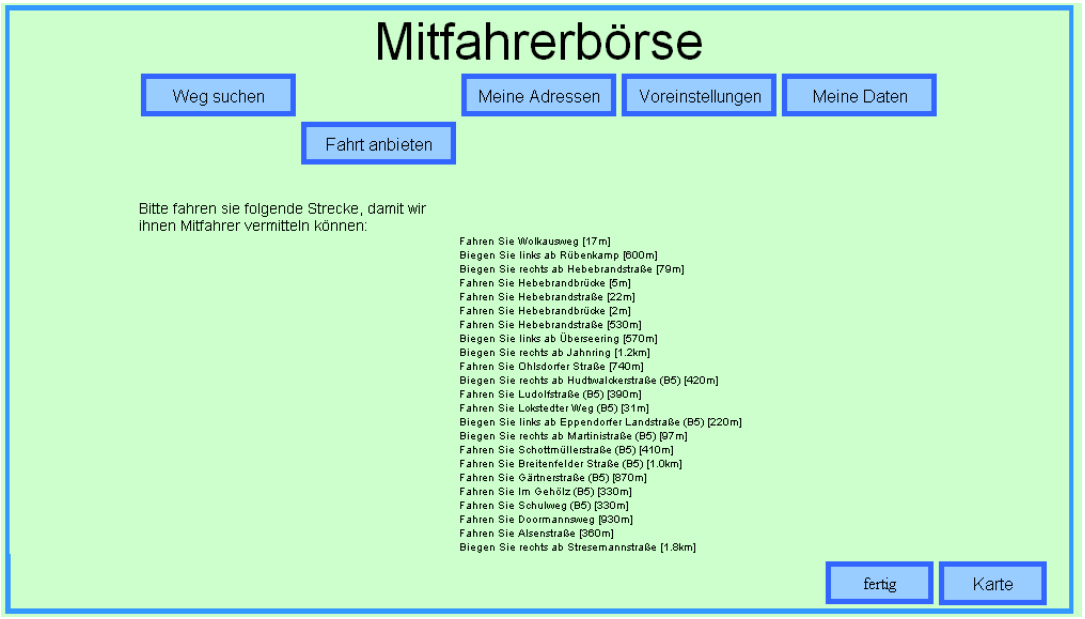


Abbildung 3.6: Screenshot: Fahrt anbieten (2)

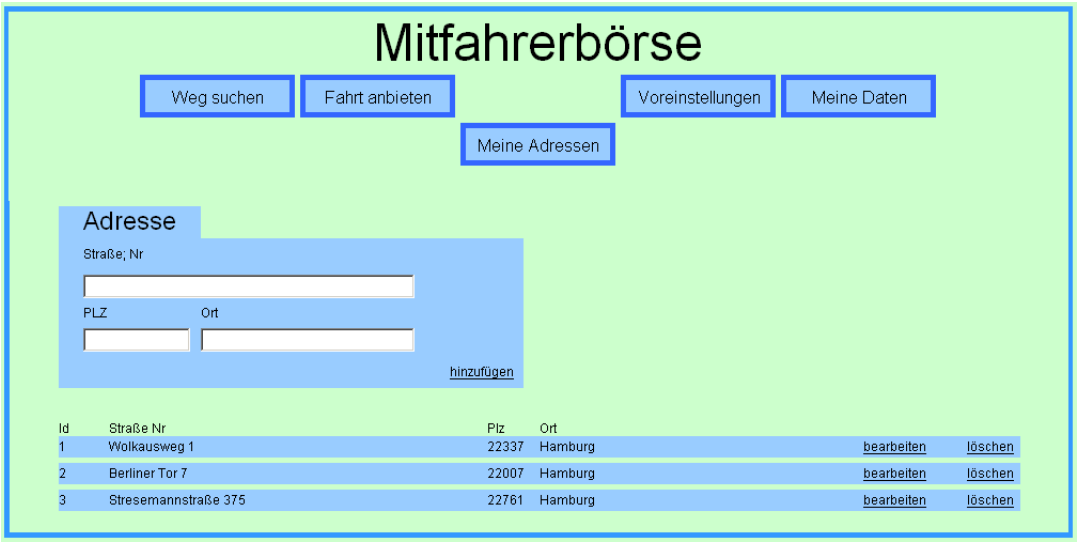


Abbildung 3.7: Screenshot: Adressen verwalten

3.2 Graphentheorie

Die Suche nach einer passenden Mitfahrgelegenheit lässt sich graphentheoretisch als Suche nach dem kürzesten Weg darstellen. Die klassischen Verfahren zur Suche des kürzesten Weges gehen von statischen Graphen mit festen Kantengewichten aus.

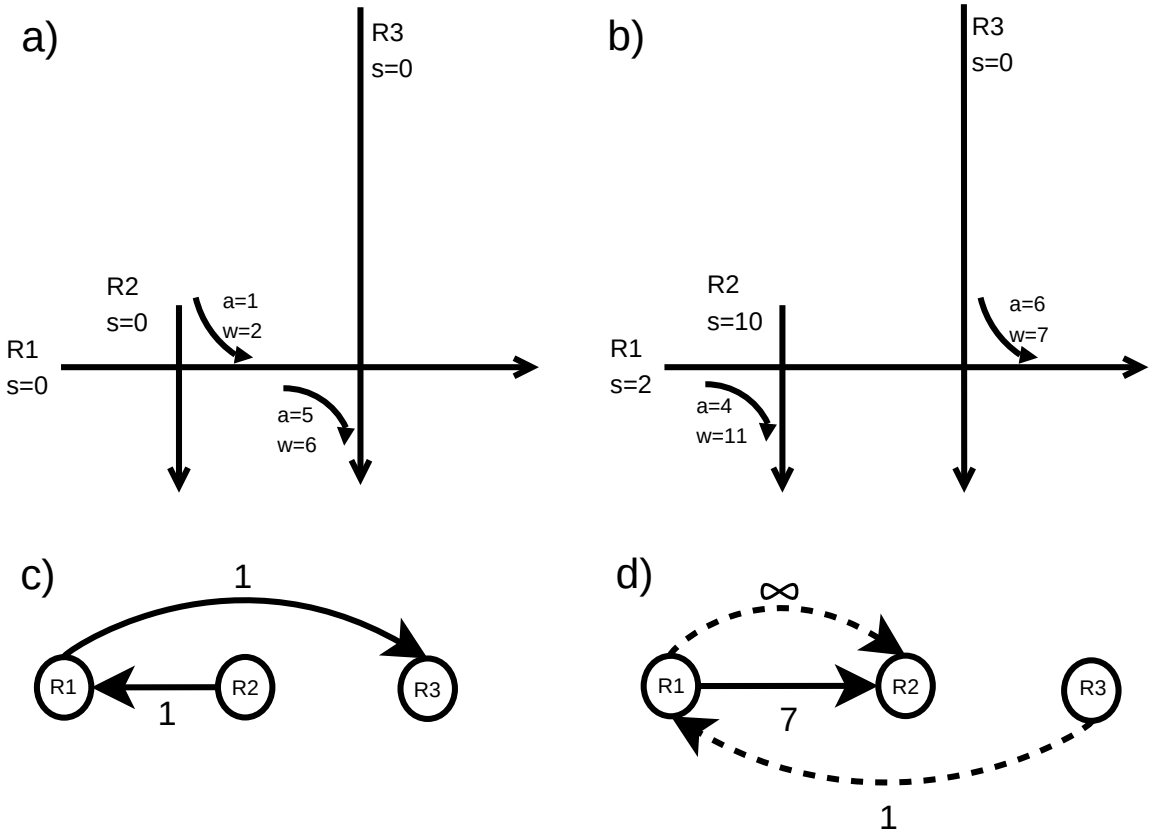


Abbildung 3.8: Dynamik durch Streckenverlauf und Zeit

Im Falle der Mitfahrerbörse handelt es sich aber um einen dynamischen Graphen mit Zeitbeschränkungen, also einen Graphen, der ständigen Änderungen unterliegt. Die Kantengewichte sind vom Zeitpunkt der Betrachtung sowie von der bisher durchlaufenen Kantenfolge abhängig. In Abbildung 3.8 ist ein einfaches Netz aus Routen dargestellt (a). Das gleiche Netz aus Routen wird mit anderen Startzeitpunkten gezeigt (b). Zu jedem dieser beiden Netze ist der daraus resultierende Graph[2] abgebildet (c/d). Zu erkennen ist, wie der Graph sich

[2]Dabei bilden die Routen die Ecken des Graphen, ihre Kreuzungspunkte die Kanten. Die Gründe für diese Form der Darstellung werden in Kapitel 3.2.1 näher erläutert.

in Abhängigkeit der Startzeitpunkte der einzelnen Routen ändert (c/d). Weiterhin ist zu sehen, wie sich die Kantengewichte je nach Ausgangspunkt ändern (d). Wenn R_1 die erste betrachtete Ecke ist, ist das Kantengewicht $R_1R_2 = 7$. Ist aber R_3 die erste betrachtete Ecke, ist das Kantengewicht $R_1R_2 = \infty$.

In dem gegebenen Beispiel werden die Kantengewichte durch die Wartezeiten an den Kreuzungspunkten der Routen festgelegt. Um jedoch allen Anforderungen gerecht zu werden, müssen in die Kantengewichte noch weitere Faktoren einbezogen werden können. Ein weiterer Faktor ist die Kapazität der angebotenen Strecken. Wird eine Mitfahrgelegenheit vermittelt, stehen die vermittelten Teilstrecken für weitere Anfragen gegebenenfalls nicht mehr zur Verfügung. Im Falle der Mitfahrerbörse hängt die Qualität der gefundenen Route nicht allein von der Fahrzeit, sondern auch von der Anzahl der Routenwechsel ab. Die Einschränkung der maximalen Anzahl von Routenwechsel kann je nach gewähltem Algorithmus z.B. durch eine Beschränkung der Suchtiefe erreicht werden.

Neben der Wahl eines geeigneten Algorithmus ist die Speicherung des Graphen entscheidend. Weit verbreitet sind zum Beispiel Matrixdarstellungen und Nachbarschaftslisten (vgl. Klauck und Maas 1999, S. 51ff). Eine Matrixdarstellung ist für größere dynamische Graphen in der Regel eher ungeeignet, da die Matrizen bei jedem Einfügen oder Entfernen einer Ecke komplett neu aufgebaut werden müssen. Zudem ist es in einer Matrixdarstellung nicht möglich, sowohl Mehrfachkanten als auch Kosten effizient abzubilden. Nachbarschaftslisten hingegen sind für dynamische Graphen besser geeignet, da das Einfügen und das Entfernen von Ecken hier keinen kompletten Neuaufbau der Liste erfordert. Ebenso können Kosten und Mehrfachkanten gleichzeitig abgebildet werden. Eine Berücksichtigung des zuvor durchlaufenen Kantenzugs ist in beiden Darstellungen zunächst nicht möglich.

Im folgenden Abschnitt wird ein geeignetes Speichermodell auf Basis von Nachbarschaftslisten vorgestellt; dabei werden die dynamischen Aspekte, soweit möglich, gekapselt. Daraufhin werden einige Algorithmen auf Anwendbarkeit für das entwickelte dynamische Modell untersucht.

3.2.1 Modellbildung

Gemäß den Anforderungen kann jeder Wegpunkt einer Route ein Start- oder (Teil-)Zielpunkt sein. Die Fragestellung an den Graphen lautet:
„Über welche Punkte komme ich mit welchen Routen vom Start- zum Zielpunkt?“
Demnach werden Kreuzungspunkte von jeweils zwei Routen als Ecke eines Graphen dargestellt. Die Kanten ergeben sich aus den jeweiligen Teilabschnitten der Routen. Fasst man sämtliche Wegpunkte einer Route zu jeweils einer Ecke zusammen und erzeugt die Kanten aus den Kreuzungspunkten der Routen, ergibt sich folgende neue Fragestellung an den Graphen:

„Welche Aneinanderreihung von Routen bringt mich von meinem Start- zu meinem Zielpunkt?“
Diese Fragestellung ist Ausgangspunkt für das im Folgenden entwickelte Modell.

Unabhängig vom verwendeten Algorithmus ist die Datenstruktur, durch die ein Graph dargestellt wird, entscheidend für die Performance der Implementation. Hierbei ist speziell das Aufsuchen aller Kanten aus einer Ecke kritisch. Die in der Literatur als Forward-Star bezeichnete Darstellung durch zwei Listen wird allgemein als die effizienteste Art der Repräsentation betrachtet (vgl. Zhan 1997). Die erste Liste ist eine Nachbarschaftsliste. In ihr wird pro Zeile eine Kante gespeichert. Diese Liste ist so sortiert, dass alle Kanten, die aus der gleichen Ecke herausführen, untereinander stehen. Die zweite Liste ist eine Indexliste. In ihr wird pro Ecke V_i festgehalten, bei welchem Index der Nachbarschaftsliste die erste Kante mit Startpunkt V_i liegt. Die Nachbarschaftsliste kann leicht um zusätzliche Felder erweitert werden. Bei dynamischen Graphen sind die Kosten variabel und können somit nicht im Vorweg ermittelt werden. Im Falle der Mitfahrerbörse sind für die Berechnung der Kosten der Punkt und die Zeit, an dem eine Route eine andere kreuzt, entscheidend. Nimmt man diese Informationen in die Nachbarschaftsliste auf, ergibt sich für den in Abbildung 3.8 (b) dargestellten Graphen die Tabelle 3.1[3].

index	departure	arrival	dep. node(dn)	arr. node(an)	dep. time(dt)	arr. time(at)
1	R_1	R_2	2	1	4	11
2	R_3	R_1	6	5	6	7

Tabelle 3.1: Forward-Star

Die Kosten $K_{R_iR_jR_p}$ stellen die Kosten R_iR_j unter der Bedingung, dass zuvor die Kante R_pR_i benutzt wurde, dar und berechnen sich wie folgt:

$$K_{R_iR_jR_p} = \begin{cases} at_{R_iR_j} - dt_{R_iR_j} & \text{wenn } at_{R_iR_j} > dt_{R_iR_j} \wedge (an_{R_pR_i} > dn_{R_iR_j} \vee R_p = nil) \\ \infty & \text{sonst} \end{cases} \tag{3.1}$$

Äquivalent zur Forward-Star-Darstellung eines Graphen ist die so genannte Backward-Star-Darstellung, bei der anstatt der Nachfolger einer Ecke ihre Vorgänger in die Nachbarschaftsliste aufgenommen werden. Durch die Dynamik des Graphen kann sich die Richtung einer Kante von Fall zu Fall ändern. Daher ist eine Kombination beider Darstellungsformen auch dann sinnvoll, wenn der Informationsgehalt der Liste sich dadurch nicht ändert. Die kombinierte Forward-Backward-Star-Darstellung wird in Tabelle 3.2 gezeigt. Die Zeilen eins und vier entsprechen den Einträgen der Forward-Star-Darstellung, die Zeilen zwei und drei den Einträgen einer Backward-Star-Darstellung.

[3] Departure bezeichnet hier den Punkt an dem eine Route verlassen wird, arrival den Punkt an dem eine Route betreten wird.

index	departure	arrival	dep. node(dn)	arr. node(an)	dep. time(dt)	arr. time(at)
1	R_1	R_2	2	1	4	11
2	R_1	R_3	5	6	7	6
3	R_2	R_1	1	2	11	4
4	R_3	R_1	6	5	6	7

Tabelle 3.2: Forward- /Backward-Star

Durch Aufnehmen der redundanten Informationen aus dem Backward-Star kann aber die Ausführungszeit der Kostenberechnung bei doppeltem Platzbedarf mit folgender Formel im Durchschnitt halbiert werden:

$$K_{R_i R_j R_p} = \begin{cases} at_{R_i R_j} - dt_{R_i R_j} & \text{wenn } at_{R_i R_j} > dt_{R_i R_j} \wedge (dn_{R_i R_p} > dn_{R_i R_j} \vee R_p = nil) \\ \infty & \text{sonst} \end{cases} \tag{3.2}$$

Zudem können durch die Vermischung der Forward-Star- und Backward-Star-Darstellung bidirektionale Algorithmen sehr viel effizienter implementiert werden. Bidirektionale Algorithmen suchen vom Start- und vom Endpunkt parallel und müssen somit die Kanten in beiden Richtungen durchlaufen.

Da die Kreuzungspunkte zwischen den Routen nicht bekannt sind, müssen diese bei der Aktivierung einer Route ermittelt werden. Eine Route besteht unter anderem aus einer geordneten Liste der Koordinatenpunkte, die sie durchläuft. Zu jedem Koordinatenpunkt wird festgehalten, wann er durchlaufen wird. Um nun die Nachbarschaftsliste aufzubauen, muss zu jedem Koordinatenpunkt einer Route geprüft werden, welche anderen Routen ihn gegebenenfalls noch durchlaufen. Da dieses Vorgehen sehr zeitaufwendig ist, gibt es ein Set mit den Koordinatenpunkten aller aktiven Routen. Bei Aktivierung einer Route werden alle ihre Koordinatenpunkte in dieses Set aufgenommen. Zu jedem in dem Set enthaltenen Koordinatenpunkt wird eine Liste mit Routen geführt, die diesen enthalten. Ein Ausschnitt aus dem Set, aus dem die in Tabelle 3.1 dargestellte Nachbarschaftsliste entstanden ist, ist in Tabelle 3.3 dargestellt. Schreib- und Lese-Operationen sind nach dem „Teile-und-herrsche-Prinzip“ (vgl. z.B. Aho u. a. 1974, S. 60ff) realisiert und haben somit eine Komplexität von $O(lnN)$. Die Nachbarschaftsliste kann parallel zu dem Koordinaten-Set aufgebaut werden. Somit ist das Modell voll dynamisch einsetzbar.

Mit Hilfe dieser Liste können später auch Start- und Ziel-Punkt für die Suche ermittelt werden, da diese zunächst ebenfalls als Koordinatenpunkte vorliegen.

Die bisher besprochene Darstellung in reiner Listenform entspricht natürlich keiner objektorientierten Denkweise. Ein analoges objektorientiertes Modell wird in Abbildung 3.9 dargestellt. Die Nachbarschaftsliste kann in der Route gespeichert werden, wodurch im Gegensatz

x	y	Route	node number	arr. time
0	2	R_1	0	2
1	2	R_1	1	3
2	1	R_2	2	12
2	2	R_1	2	4
2	2	R_2	1	11
2	3	R_2	0	10

Tabelle 3.3: Entry Liste

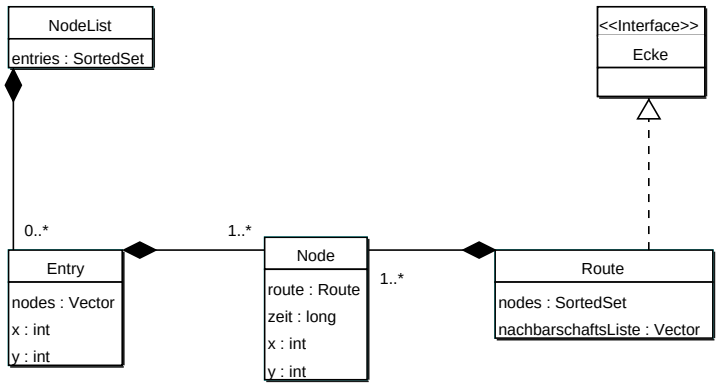


Abbildung 3.9: Objektorientiertes Modell

zum listenbasierten Forward-Star eine Indexsuche entfällt. Die Entry-Liste kann als „Sorted Set“ implementiert werden. Durch die Verwendung von Entry-Objekten wird das Modell handlicher. Entries können über die Koordinaten direkt mit Nodes verglichen werden. Auch hier entfällt eine Iteration.

Die Daten liegen nun in einer für die Suche optimierten Struktur bereit. Dafür wird ein erhöhter Platzbedarf des Speichermodells in Kauf genommen. Wie auch in (Vahrenkamp 2003, S. 34) diskutiert, führt dies aber zu erheblich schlankeren und damit effizienteren Suchalgorithmen. Traditionell wurde im Operational Research immer auf eine möglichst kompakte Darstellung geachtet, da in der Regel der Speicherplatz teuer und zudem sehr begrenzt war. Heutzutage sind Rechner mit Hauptspeicherkapazitäten von mehreren Gigabyte nicht unüblich und ermöglichen bei deren Ausnutzung sehr schnelle Suchverfahren.

3.2.2 Wahl des Algorithmus

Folgend werden einige bekannte Shortest-Path-Algorithmen auf Anwendbarkeit für das hier entwickelte dynamische Modell untersucht.

Klassifizierung von Shortest-Path-Algorithmen

Shortest-Path Algorithmen lassen sich nach den Problemstellungen in zwei Klassen einteilen:

- Single-source-shortest-path-Algorithmen (SSSP-Algorithmen)
  Diese Klasse von Algorithmen beantwortet das One-to-one-shortest-path-Problem. Ausgehend von einem Startknoten wird der kürzeste Weg zu einem Zielknoten gesucht. Dadurch werden gleichzeitig die kürzesten Wege zu verschiedenen anderen Knoten gefunden, da jeder Teilabschnitt des gesuchten kürzesten Weges wieder ein kürzester Weg sein muss. Eine Erweiterung ist das One-to-all-shortest-path-Problem, bei dem ausgehend von einem Startknoten die kürzesten Wege zu allen anderen Knoten des Graphen gesucht werden. Ein Beispiel hierfür ist die Breitensuche (vgl. z.B. Sedgewick 1992).
  Die SSSP-Algorithmen werden zudem in Label-Setting- und Label-Correcting-Algorithmen unterteilt. Speziell für das One-to-one-shortest-path-Problem wird davon ausgegangen, dass Label-Setting-Algorithmen effizienter sind, da sie abgebrochen werden können sobald das Ziel gefunden wurde. Der Dijkstra-Algorithmus gehört zur Klasse der Label-Setting-Algorithmen. Der Bellman-Ford-Moore-Algorithmus (siehe Bersekas 1993) ist der klassische Vertreter der Label-Correcting-Algorithmen.
- All-pairs-shortest-path-Algorithmen (APSP-Algorithmen)
  Diese Klasse von Algorithmen beantwortet das All-to-all-shortest-path-Problem. Es werden die kürzesten Wege von jedem Knoten in einem Graphen zu jedem anderen Knoten gesucht. Ein klassischer Vertreter ist der Floyd-Warshall-Algorithmus (vgl. z.B. Klauck und Maas 1999).

Ein All-pairs-shortest-path-Algorithmus wäre für das hier behandelte Problem natürlich sehr gut geeignet, da mit einem Durchgang des Algorithmus eine Vielzahl von Anfragen beantwortet werden kann. Leider haben aber die bekannten All-pairs-shortest-path-Algorithmen für große Eckenzahlen ein schlechtes Laufzeitverhalten. Der Floyd-Warshall-Algorithmus hat zum Beispiel eine Komplexität von $O(N^3)$. Vorausgesetzt, die Häufigkeit der Anfragen rechtfertigt die Berechnung aller Paare, ist es sinnvoller, einen Single-source-shortest-path-Algorithmus mit einer Komplexität von höchstens $O(N^2)$ mit jeder Ecke des Graphen als Startpunkt einmal zu durchlaufen.

Floyd-Warshall-Algorithmus

Der Floyd-Warshall-Algorithmus berechnet alle kürzesten Wege in gewichteten Graphen ohne Zyklen negativer Länge.
Der Algorithmus arbeitet mit zwei Matrizen, einer Distanzmatrix und einer Transitmatrix. Am

Ende des Algorithmus lassen sich aus der Distanzmatrix die Kosten des kürzesten Weges vom Start zum Ziel ablesen und aus der Transitmatrix der Weg rekonstruieren.
Die Distanzmatrix wird mit ∞ initialisiert.
Die Transitmatrix wird mit 0 initialisiert.
In die Distanzmatrix werden die Kosten aller bekannten direkten Verbindungen D_{ik} eingetragen.
Die Distanzmatrix gleicht nun der Darstellung des Graphen als Adjazenzmatrix.
Der Algorithmus verläuft wie folgt:

- Für jeden Eintrag in der Distanzmatrix wird schrittweise geprüft, ob sich die Kosten der direkten Verbindung D_{ik} durch den Weg über einen der anderen Knoten V_j verringern lassen.
- Führt die Benutzung des Weges über einen der anderen Knoten V_j zu einer Veringerung der Kosten, wird die Distanzmatrix mit dem berechneten Wert aktualisiert
  $D_{ik} = min(D_{ij} + D_{jk}, D_{ik})$
  und der Knoten V_j in die Transitmatrix als Vorgängerknoten des Zielknotens V_k eingetragen
  $T_{ik} = j$ falls $D_{ij} + D_{jk} < D_{ik}$.

Der Floyd-Warshall-Algorithmus ist auf das hier entwickelte dynamische Modell nicht anwendbar, da der Weg vom Start zum Ziel schrittweise in Teilabschnitte zerlegt wird. Die Kosten für die einzelnen Teilabschnitte werden unabhängig vom Gesamtkontext berechnet.

Dijkstra-Algorithmus

Der Dijkstra-Algorithmus berechnet kürzeste Wege in gewichteten Graphen ohne negative Kanten.
Zu jedem Knoten wird die Distanz zum Startpunkt gespeichert. Zu Beginn des Algorithmus ist diese Distanz für den Startknoten Null und für alle anderen unendlich.
Die Knoten werden in zwei Gruppen aufgeteilt, die Gruppe der temporär markierten Knoten und die Gruppe der permanent markierten Knoten. Zu Beginn des Algorithmus sind alle Knoten temporär markiert.
Um den kürzesten Weg konstruieren zu können, wird zu jedem Knoten der Vorgänger festgehalten.
Der Algorithmus verläuft wie folgt:

- Es wird der temporär markierte Knoten V_{min} mit der kürzesten Distanz zum Startknoten gewählt.

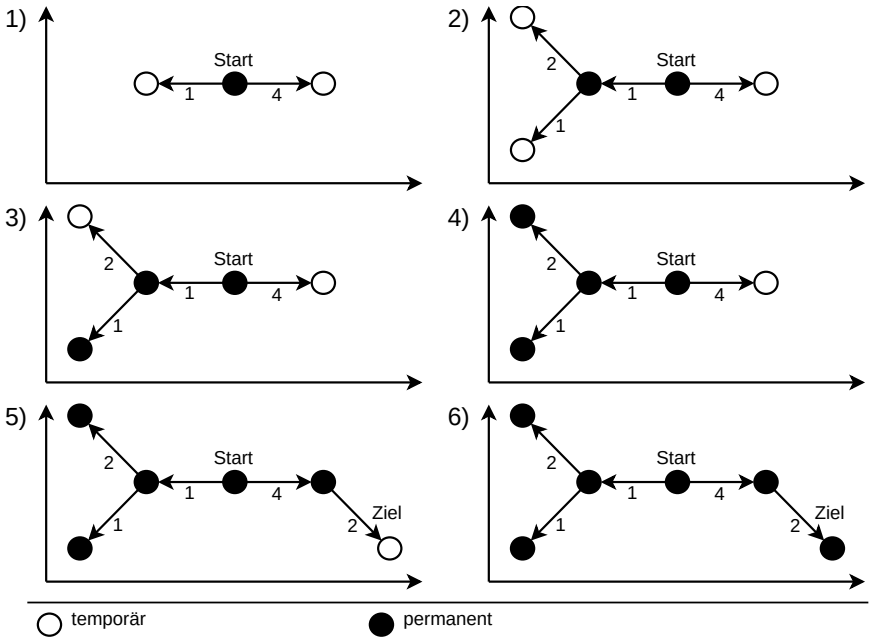


Abbildung 3.10: Ausbreitung Dijkstra

- Der Knoten V_{min} wird in die Gruppe der permanent markierten Knoten aufgenommen. Nun wird die Distanz zum Startknoten für alle Folgeknoten V_n von V_{min} neu berechnet, sofern diese nicht bereits permanent markiert sind. V_{min} wird als Vorgänger des jeweiligen Folgeknotens V_n festgehalten.
- Dieser Vorgang wiederholt sich solange, bis es keine temporär markierten Knoten mehr gibt.

Für die Suche nach dem temporären Knoten mit der jeweils kleinsten Distanz zum Startknoten gibt es verschiedene Vorgehensweisen. Als besonders effizient hat sich die Speicherung der temporär markierten Knoten in einem Radix- oder Fibonacci-Heap erwiesen (siehe z.B. Ahuja u. a. 1990, Hasselberg 2000).

Ford-Moore-Algorithmus

Der Ford-Moore-Algorithmus berechnet kürzeste Wege in gewichteten Graphen ohne Zyklen negativer Länge.
Im Gegensatz zum Dijkstra-Algorithmus wird hier nicht zwischen permanent und temporär markierten Knoten unterschieden.
Zu jedem Knoten wird die Distanz zum Startpunkt gespeichert. Zu Beginn des Algorithmus

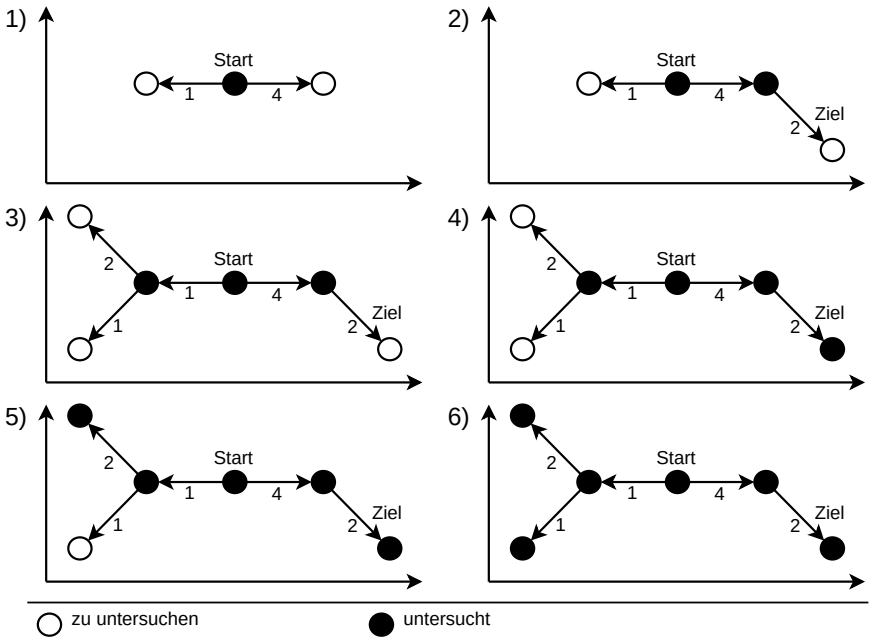


Abbildung 3.11: Ausbreitung Ford-Moore / FiFo

ist diese Distanz für den Startknoten Null und für alle anderen unendlich.
Es gibt eine Liste mit zu untersuchenden Knoten. Zu Beginn des Algorithmus enthält diese nur den Startknoten.
Um den kürzesten Weg konstruieren zu können, wird zu jedem Knoten der Vorgänger festgehalten.
Der Algorithmus verläuft wie folgt:

- Der erste Knoten V_{first} aus der Liste der zu untersuchenden Knoten wird entfernt.
- Nun wird die Distanz zum Startknoten für alle Folgeknoten V_n von V_{first} neu berechnet. V_{first} wird als Vorgänger des jeweiligen Folgeknotens V_n festgehalten. Der jeweilige Folgeknoten V_n wird, sofern er nicht schon enthalten ist, hinten in die Liste der zu untersuchenden Knoten eingefügt.
- Dieser Vorgang wiederholt sich solange, bis die Liste der zu untersuchenden Knoten leer ist.

Die hier vorliegende FiFo-Strategie zur Wahl des jeweils nächsten zu untersuchenden Knotens kann durch die Strategie von d'Esopo und Pape (vgl. Bersekas 1993) ersetzt werden, bei der ein Folgeknoten V_n nur dann am Ende der Liste eingefügt wird, wenn er zum ersten Mal untersucht wurde. Falls der jeweilige Folgeknoten schon mindestens einmal untersucht wurde, wird er am Anfang der Liste eingefügt und somit schnell wieder untersucht.

Der Ford-Moore-Algorithmus(FiFo) ist für das hier vorliegende Problem sehr gut geeignet, da er, wie in Abbildung 3.11 dargestellt, eine Breitensuch-Strategie verfolgt. Dadurch kann der Algorithmus einfach abgebrochen werden, sobald alle möglichen Wege mit einer festgelegten Zahl von Routenwechseln abgesucht worden sind.

A*-Algorithmus

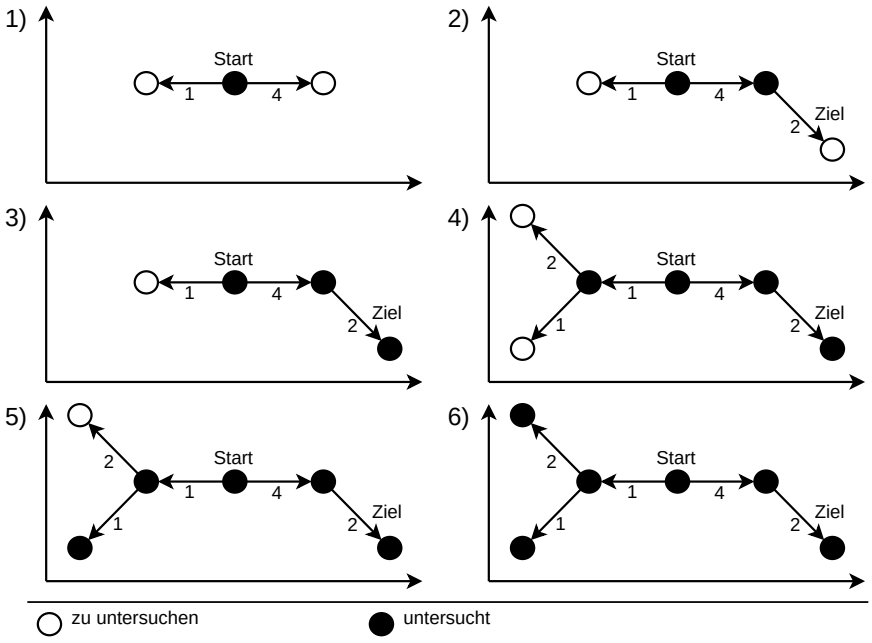


Abbildung 3.12: Ausbreitung A* / euklidische Distanz

Der A*-Algorithmus berechnet kürzeste Wege in gewichteten Graphen.
Zu jedem Knoten wird die Distanz zum Startpunkt gespeichert. Zu Beginn des Algorithmus ist diese Distanz für den Startknoten Null und für alle anderen unendlich.
Es gibt eine Liste mit zu untersuchenden Knoten. Zu Beginn des Algorithmus enthält diese nur den Startknoten.
Es gibt eine Schätzfunktion, die für jeden Knoten eine untere Schranke für die noch zu erwartendende Distanz zum Ziel liefert.
Um den kürzesten Weg konstruieren zu können, wird zu jedem Knoten der Vorgänger festgehalten.
Der Algorithmus verläuft wie folgt:

- Es wird der Knoten V_{min} aus der Liste der zu untersuchenden Knoten entfernt, bei

dem die Summe aus der auf ihn angewandten Schätzfunktion und der gespeicherten bisherigen Distanz am kleinsten ist.

- Nun wird die Distanz zum Startknoten für alle Folgeknoten V_n von V_{min} neu berechnet. V_{min} wird als Vorgänger des jeweiligen Folgeknotens V_n festgehalten. Der jeweilige Folgeknoten V_n wird, sofern er nicht schon enthalten ist, in die Liste der zu untersuchenden Knoten eingefügt. Ist V_n dagegen bereits in der Liste enthalten, wird die Distanz neu gesetzt und V_{min} als Vorgänger eingetragen unter der Vorraussetzung, dass der neue Wert besser ist.
- Dieser Vorgang wiederholt sich solange, bis die Liste der zu untersuchenden Knoten leer ist oder keiner der enthaltenen Knoten eine Verbesserung des bisher gefundenen Ergebnisses liefern kann.

Durch die Verwendung der Schätzfunktion ergibt sich eine gerichtete Suche in Richtung Zielknoten. Es handelt sich dabei um eine Heuristik: Es wird zwar zielgerichtet gesucht, aber keine Möglichkeit ausgeschlossen.
Verwendet man die euklidische Distanz als Schätzfunktion, kann der A*-Algorithmus den kürzesten Weg, wie in Abbildung 3.12 dargestellt, bezüglich der Anzahl der Umstiegspunkte unabhängig von den dabei entstehenden Kosten berechnen.

3.2.3 Anpassung und Optimierung der Algorithmen

Nachdem im letzten Abschnitt eine Reihe von Algorithmen identifiziert wurde, die auf den hier entwickelten dynamischen Graphen anwendbar sind, wird folgend beschrieben, inwieweit dazu Anpassungen erforderlich sind. Der Dijkstra-Algorithmus, der FordMoore-Algorithmus und der A-Stern-Algorithmus sind vom Ablauf her sehr ähnlich. Im wesentlichen unterscheiden sie sich durch die Strategie zur Auswahl des nächsten zu untersuchenden Knoten. Die Kosten der Kanten werden natürlich in allen Algorithmen benötigt. In dem hier besprochenen dynamischen Graph sind aber gerade diese Kosten variabel. Sie hängen unter anderem, wie bereits beschrieben, von der vorher benutzten Kante bzw. der zuvor besuchten Ecke ab. Der folgende Codeausschnitt zeigt den Dijkstra-Algorithmus für das entwickelte dynamische Modell:

```
1  while (!zuUntesuchendeEcken.isEmpty()) {
2    Route r1 = (Route) zuUntersuchendeEcken.extractMin();
3    r1.setVisited(true);
4    List t = r1.neighboursAfter(r1.getPredecessor());
5    for (Iterator iterator = t.iterator(); iterator.hasNext();) {
6      Route r2 = (Route) iterator.next();
7      long newCosts = r1.getCurrentCosts()
```

```
 8        + r1.costsFromTo(r1.getPredecessor(), r2);
 9      if (newCosts < r2.getCurrentCosts()) {
10        r2.setPredecessor(r1);
11        r2.setCurrentCosts(newCosts);
12      }
13    }
14  }
```

Für die Anpassung an das dynamische Modell sind zwei minimale Modifikationen nötig:

- In Zeile 4 werden die Kanten, die vom derzeitigen Standpunkt aus erreichbar sind, zur weiteren Untersuchung geholt. Dazu wird der Funktion jeweils die Vorgängerecke mitgegeben.
- In den Zeilen 8,9 werden die Kosten für die jeweils betrachtete Kante ermittelt. Zu diesem Zweck wird der Funktion jeweils die Vorgängerecke mitgegeben.

Beim Dijkstra-Algorithmus, der zur Familie der Label-Setting-Verfahren gehört, enthält die Liste der zu untersuchenden Knoten alle Ecken des Graphen. Dadurch ist zum einen der Aufwand zum Auffinden der Ecke mit geringsten Kosten recht groß, zum anderen müssen diese Ecken alle untersucht werden. Bei den Label-Correcting-Verfahren wird diese Liste nur mit dem Startknoten initialisiert. Im Verlauf des Algorithmus werden dann diejenigen Ecken hinzugefügt, die zur Ergebnisfindung beitragen können. Dadurch wird eine Ecke gegebenenfalls mehrfach untersucht. Kombiniert man nun die Vorteile dieser beiden Verfahren erhält man einen Algorithmus, der in dem hier entwickelten dynamischen Modell sehr gute Ergebnisse liefert. Er ist in dem folgenden Codeausschnitt dargestellt:

```
 1  while (!zuUntesuchendeEcken.isEmpty()) {
 2    Route r1 = (Route) zuUntesuchendeEcken.extractMin();
 3    r1.setVisited(true);
 4    List t = r1.neighboursAfter(r1.getPredecessor());
 5    for (Iterator iterator = t.iterator(); iterator.hasNext();) {
 6      Route r2 = (Route) iterator.next();
 7      if (!r2.isVisited()) {
 8        long newCosts = r1.getCurrentCosts()
 9           + r1.costsFromTo(r1.getPredecessor(), r2);
10        if (newCosts < r2.getCurrentCosts()) {
11          r2.setPredecessor(r1);
12          r2.setCurrentCosts(newCosts);
13          if (!zuUntesuchendeEcken.contains(r2)) {
14            zuUntesuchendeEcken.add(r2);
```

```
15            }
17          }
18        }
19      }
20    }
```

Die Liste der zu untersuchenden Knoten wird wie bei den Label-Correcting-Verfahren mit dem Startknoten initialisiert. Im weiteren Verlauf des Algorithmus werden wiederum diejenigen Ecken eingefügt die zur Ergebnisfindung beitragen können. Dabei wird aber sichergestellt, dass jede Ecke höchstens einmal untersucht wird. Es wird jeweils die Ecke untersucht, die die geringsten bisherigen Kosten enthält. Gerade in dem hier entwickelten dynamischen Graphen, in dem die Kantenkosten jedesmal neu berechnet werden müssen, kann so eine deutliche Leistungssteigerung erreicht werden.

3.3 Systemdesign

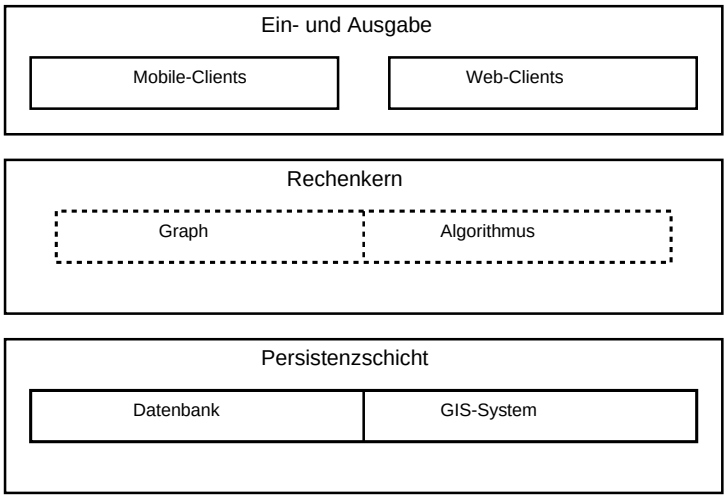


Abbildung 3.13: Übersicht über die Komponenten

Um Wartbarkeit und Flexibilität des Systems gewährleisten zu können, wird, wie in Abbildung 3.13 dargestellt ist, auf eine klassische Dreischichten-Architektur zurückgegriffen. Neben dem Rechenkern gibt es eine Persistenzschicht sowie eine Ein- und Ausgabeschicht. Der Rechenkern ist dabei nur für die Suche innerhalb des Wegenetzes verantwortlich, die zugrunde liegenden Routen werden von einem Geoinformationssystem geliefert. Das Geoinformationssystem kann als Teil der Persistenzschicht betrachtet werden, da es lediglich grundlegende Informationen liefert.

In Abbildung 3.14 sind die Schnittstellen zwischen den einzelnen Komponenten dargestellt. Die Schnittstelle für Clients wird durch den Controller realisiert. Der Controller stellt Methoden zum Anbieten und zum Suchen von Mitfahrgelegenheiten sowie zum Verwalten der Userdaten bereit. Die Schnittstelle zwischen Controller und Persistenzschicht wird durch die Interfaces „GIS“ und „DB“ festgelegt. Die Schnittstelle zwischen Controller und Rechenkern wird durch die Klassen „AbstractAlgorithm“ und „Graph“ gebildet. Das Geoinformationssystem, der konkrete Algorithmus sowie die Datenbank sind dadurch weitestgehend austauschbar.

Um auch mobile Clients mit eingeschränkten Ressourcen unterstützen zu können, verwendet der Controller clientseitig nur einfache Datenstrukturen. In den Klassendiagrammen werden zur besseren Verständlichkeit die Typen „Point“ und „Result“ benutzt, diese Typen werden bei der Implementierung durch XML-Strings zu ersetzt.

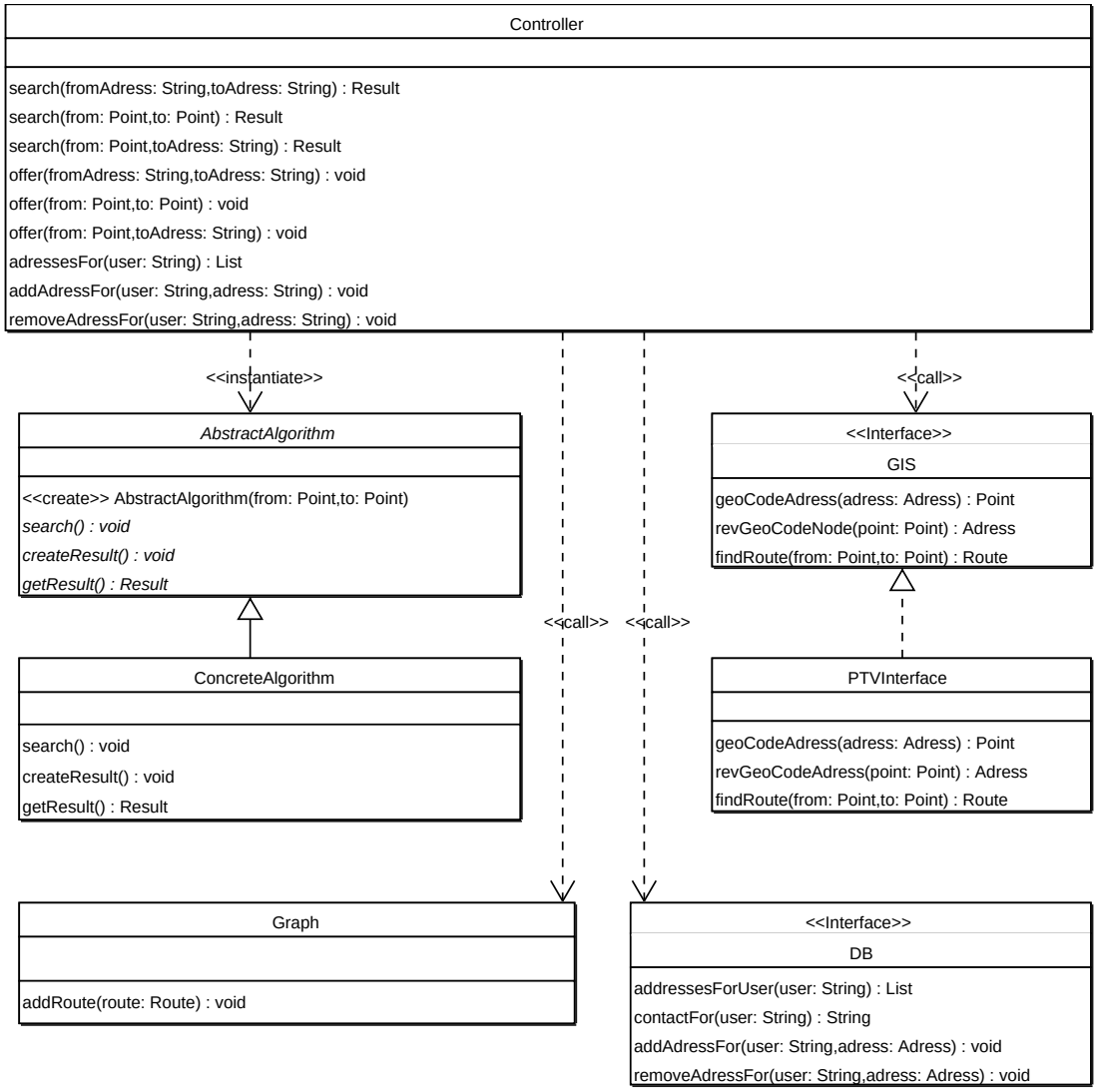


Abbildung 3.14: Klassendiagramm: Interfaces

3.3.1 Persistenzschicht

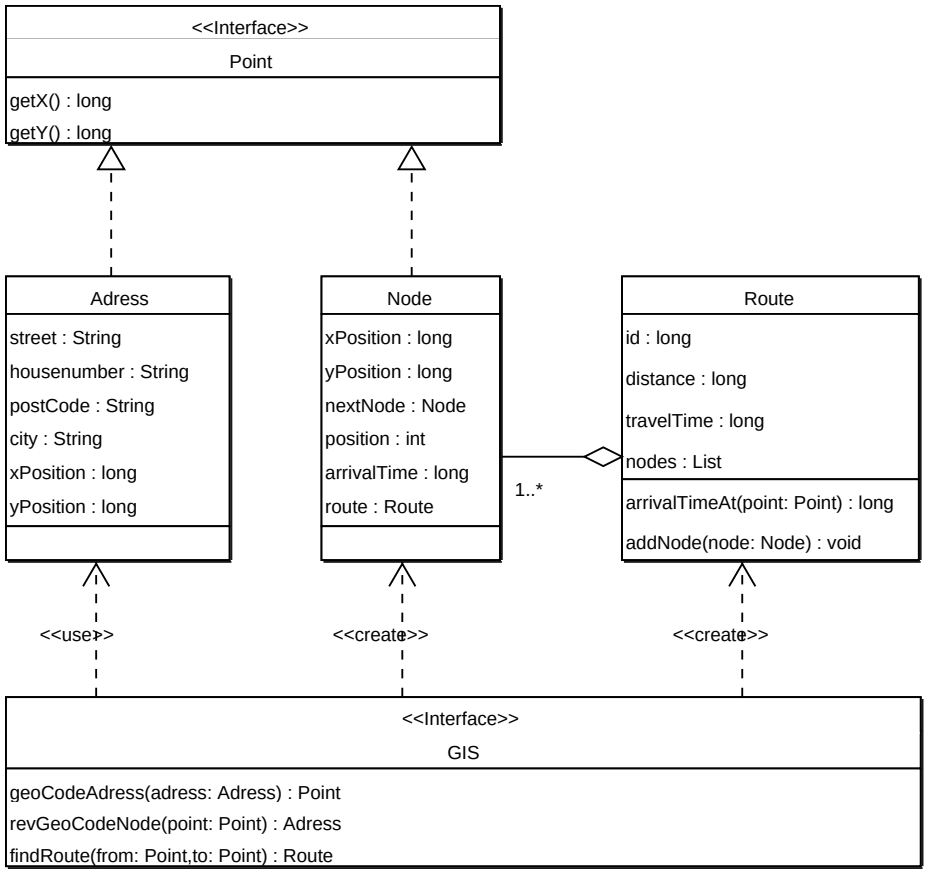


Abbildung 3.15: Klassendiagramm: GIS-Interface

An das Geoinformationssystem werden nur geringe Anforderungen gestellt. Folgende Anforderungen müssen erfüllt werden:

- Adressen müssen in Geo-Koordinaten aufgelöst werden können,
- Geo-Koordinaten müssen in Adressen umgewandelt werden können,
- Routen müssen als sortierte Liste von Geo-Koordinaten wiedergegeben werden.

Weiterhin sollte das Geoinformationssystem in der Lage sein, zu jedem Koordinatenpunkt einer Route die voraussichtliche Durchfahrtszeit anzugeben. Kann diese Anforderung nicht

erfüllt werden, lassen sich die Durchfahrtszeiten aus der Anzahl der Koordinatenpunkte und der Gesamtfahrzeit oder der Durchschnittsgeschwindigkeit schätzen. Aus Statistiken (siehe HansestadtHamburg 2001) geht hervor, dass die Durchschnittsgeschwindigkeit in Hamburg im Jahr 2001 ca. 28km/h betrug. Die Benutzung von Autobahnen oder Landstraßen muss gegebenenfalls berücksichtigt werden können.

Um ein Geoinformationssystem nutzen zu können, muss eine Schnittstellenklasse bereitgestellt werden. Die Funktionalität dieser Klasse wird in Abbildung 3.15 dargestellt. Insbesondere wird durch das Interface „GIS“ dafür gesorgt, dass die gelieferten Informationen in eine für den Rechenkern nutzbare Struktur gebracht werden.

Die Datenbank dient zur Speicherung von Userdaten. Neben den Kontaktdaten kann ein User mehrere häufig von ihm besuchte Adressen speichern. Das Speichern der Adressen ist für die Funktion des Systems nicht relevant, sondern dient einzig einer komfortableren Nutzung.

3.3.2 Rechenkern

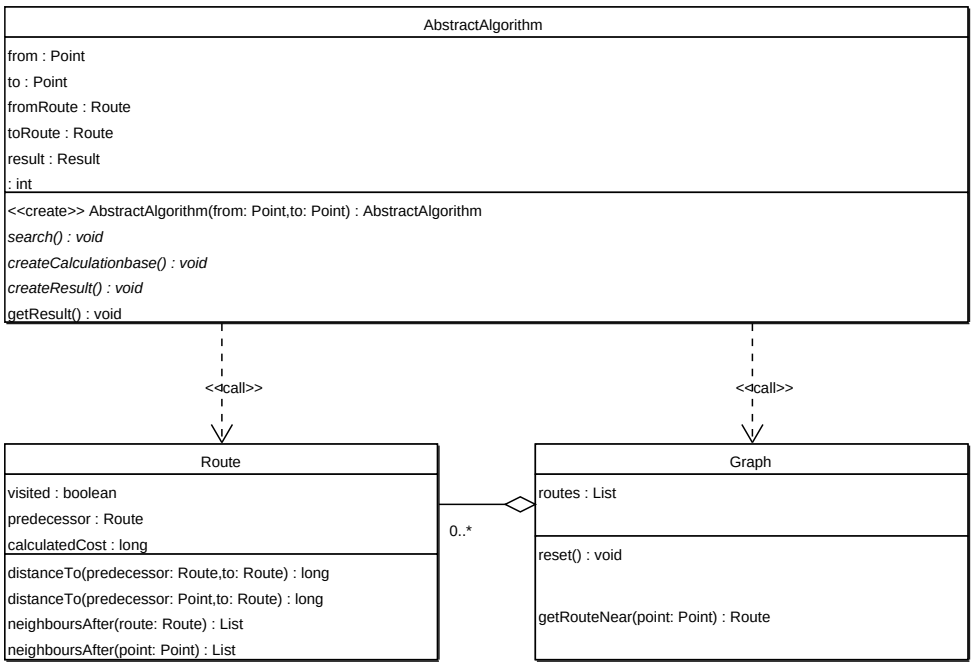


Abbildung 3.16: Klassendiagramm: Rechenkern

Der Rechenkern besteht im wesentlichen aus zwei Teilen: Einer Datenstruktur, die das Wegenetz repräsentiert, und einer Suchfunktion, die einen Weg durch das Wegenetz liefert (Siehe Abbildung 3.16). Das Wegenetz wird durch die Klassen „Graph“ und „Route“ realisiert. Die Klasse Graph hat im wesentlichen eine Containerfunktion. Die für eine Suche wichtigen Informationen werden direkt von den Routen geliefert. Eine Route kennt alle anderen Routen, die sie kreuzen, und kann die Kosten berechnen, die anfallen, um zu diesen zu wechseln. Weiterhin stellt sie einige Felder bereit, um einen Weg durch das Netz zu konstruieren. Da die Suchfunktion auf das spezielle Einsatzgebiet der Mitfahrerbörse abgestimmt werden muss, ist in der Klasse „AbstractAlgorithm“ nicht nur die Schnittstelle einer Suchfunktion definiert, sondern auch der in Abbildung 3.17 dargestellte allgemeine Ablauf einer Suche implementiert.

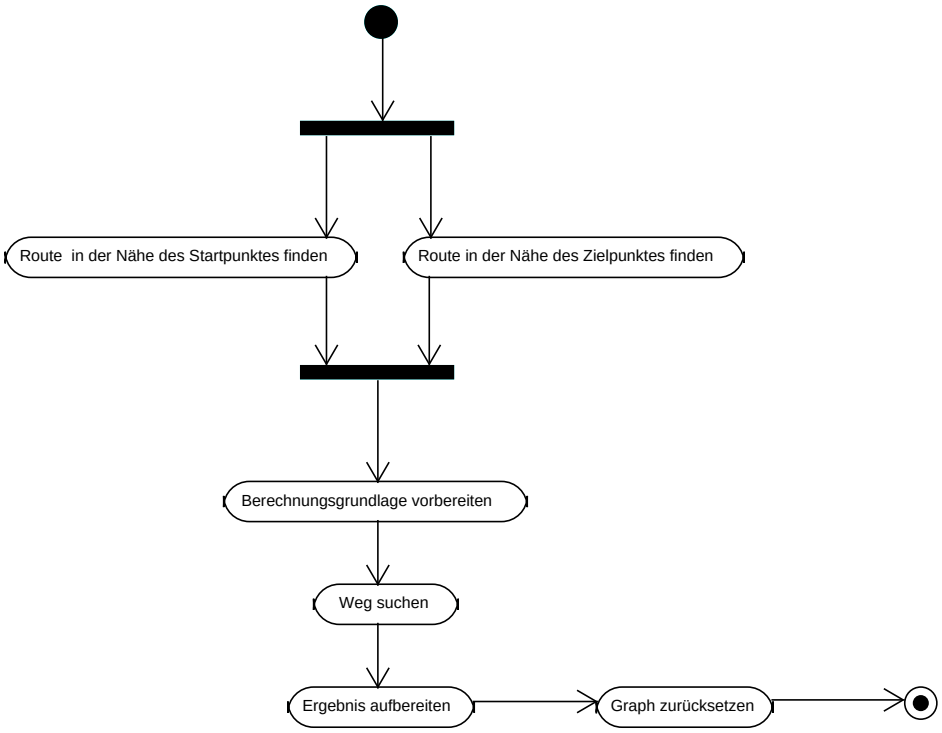


Abbildung 3.17: Aktivitätsdiagramm: Suche

In Abbildung 3.18 wird der Ablauf der Ergebnisaufbereitung zum Beispiel für den Dijkstra- oder den A*-Algorithmus dargestellt. Während der Ergebnisaufbereitung wird die verfügbare Kapazität auf den benutzten Teilstrecken verringert, damit für eine Fahrt nicht mehr Mitfahrer vermittelt werden können als freie Plätze verfügbar sind.

Für eine Mitfahrerbörse scheint es aber sinnvoll, pro angebotener privater Fahrt nur einen Fahrgast zu vermitteln, da es für diese Fahrer im Zweifelsfall unangenehm ist, auf einer Fahrt mehrere Personen, die an verschiedenen Punkten warten, mitzunehmen.

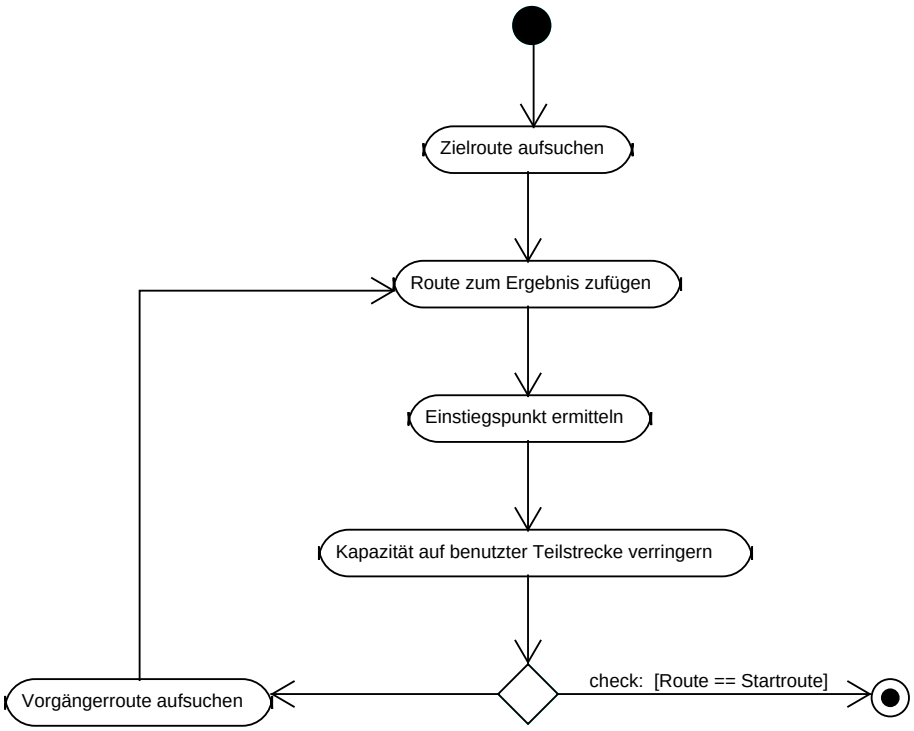


Abbildung 3.18: Ergebnis aufbereiten

3.3.3 Threadmodell

Neben den dynamischen Kantengewichten und den Kapazitäten der Routen ist das ständig wechselnde Angebot von Routen für die Dynamik des Graphen verantwortlich. Dieser Punkt kann aber vor den Suchalgorithmen völlig verborgen werden, indem der Zugriff auf den Graphen exklusiv gestaltet wird. Insgesamt gibt es drei verschiedene Funktionen, die auf den Graphen zugreifen:

- Neue Routenangebote werden zu dem Graphen hinzugefügt,
- Abgelaufene Routen werden aus dem Graphen entfernt,
- Suche in dem Graphen nach einer passenden Mitfahrgelegenheit.

In dem in Abbildung 3.19 dargestellten Petri-Netz ist der synchronisierte Zugriff auf den Graphen dargestellt. Die abgelaufenen Routen werden durch einen Thread entfernt, der etwa

alle Sekunde aktiv ist. Für jede Suche und jedes Routenangebot wird ein neuer Thread gestartet. Diese werden über eine FiFo-Warteschlange synchronisiert. Ziel muss es aber sein, sowohl die Suche als auch das Einstellen neuer Angebote so schnell abzuarbeiten, das sich in der Warteschlange im Regelfall höchstens eine Anfrage befindet.

Dieses Vorgehen trägt wesentlich dazu bei, dass die Suchalgorithmen sehr schlank implementiert werden können. Außerdem bringt ein paralleler Zugriff auf den Graphen kaum Vorteile, wenn die Warteschlangenlänge nur sehr kurz ist.

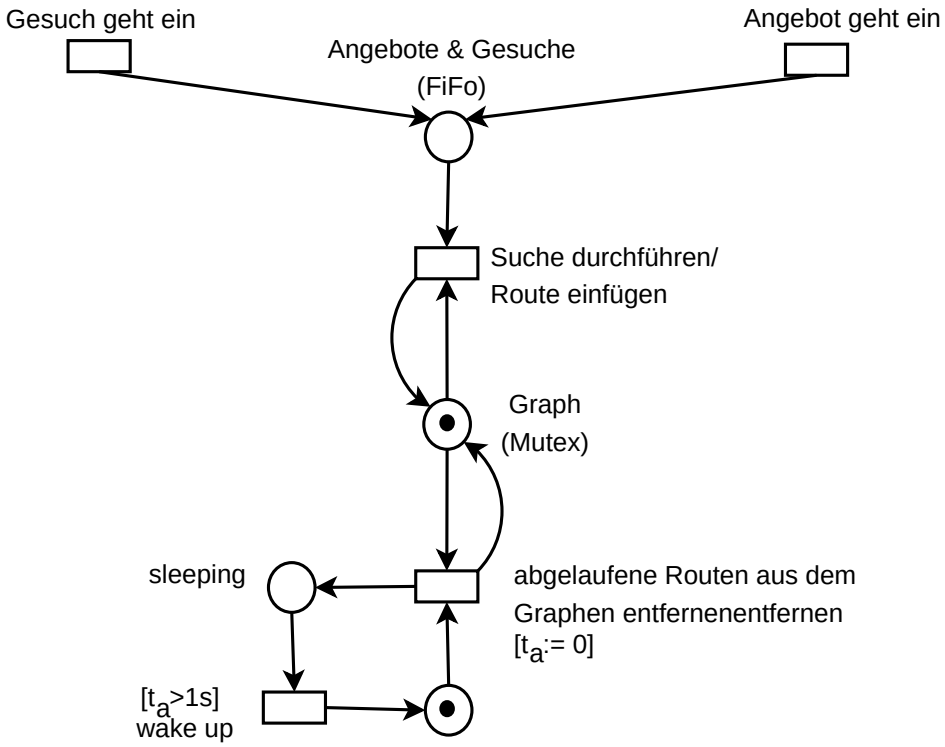


Abbildung 3.19: Threadmodell

3.4 Testsystem

Zum Testen wird eine Java/Swing-Applikation bereitgestellt, die die wesentlichen Funktionen des Systems veranschaulicht. Die Applikation, wie in Abbildung 3.20 dargestellt, ermöglicht es,

- neue Routen anzulegen,
- vorhandene Routen zu aktivieren/deaktivieren,
- einen Weg durch das Routennetz zu finden und das Ergebnis anzuzeigen,
- einen Algorithmus für die Suche auszuwählen und
- die aktiven Routen als Karte zu visualisieren.

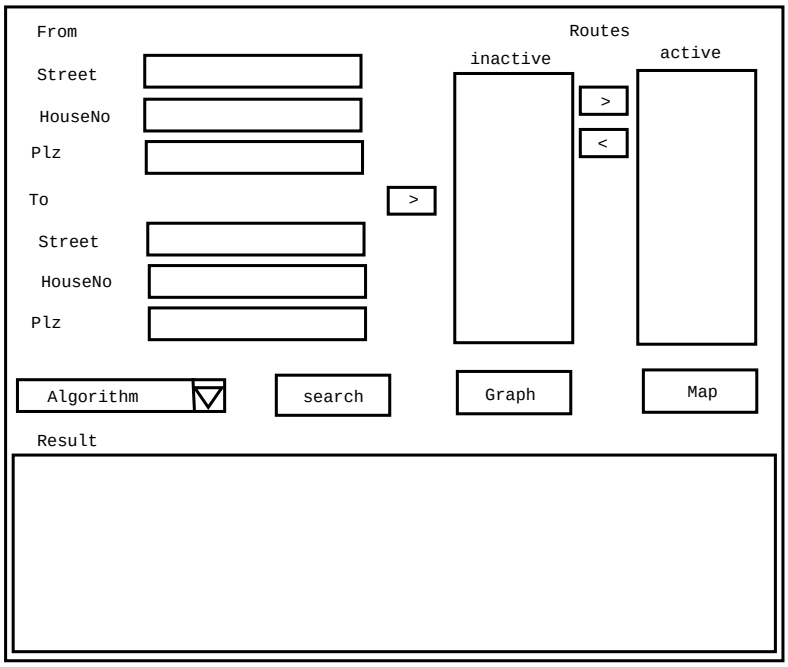


Abbildung 3.20: Screenshot: Testapplikation

4 Evaluierung

Alle in diesem Kapitel ausgewiesenen Werte wurden auf einem System mit folgenden Leistungsmerkmalen gemessen:
„Pentium M 1.4 GHz“
512Mb Arbeitsspeicher
„Microsoft Windows XP“

4.1 Implementierungshinweise

Das getestete System wurde unter Sun’s J2SE5 kompiliert.
Zeitmessungen erfolgten mit Hilfe der Klasse sun.misc.Perf, die Zugriff auf einen Highresolution-Timer bietet.

Die Performance des unter JAVA implementierten Prototypen hängt stark von den verwendeten JDK-Klassen ab.

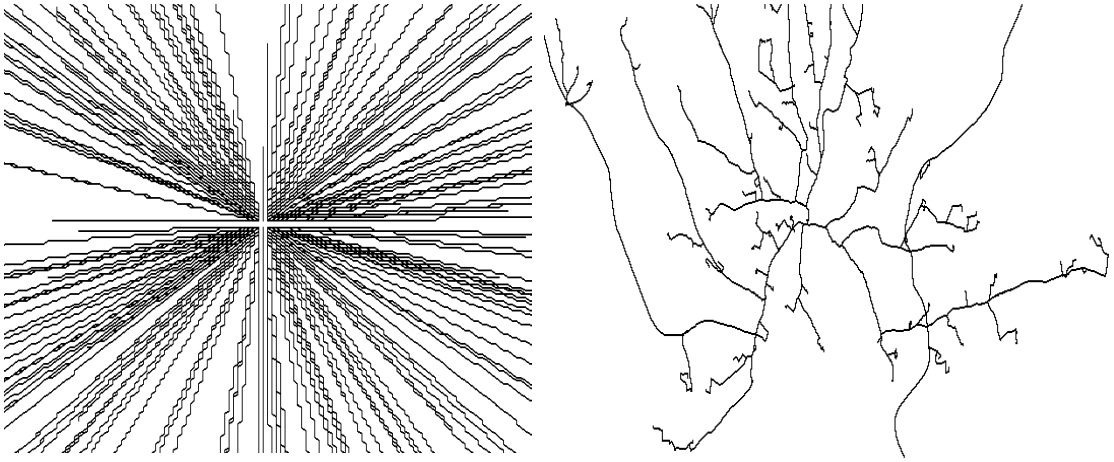
Wie auch in (sun microsystems, java.util.ArrayList) beschrieben, laufen die meisten Operationen auf einer Liste vom Typ ArrayList in linearer oder nahezu konstanter Zeit. Listen vom Typ LinkedList hingegen sind deutlich langsamer; besonders hervorzuheben ist hierbei, dass gerade das Einfügen von Elementen in der Mitte der Liste hier sehr viel langsamer ist und sich exponentiell zur Größe der Liste verhält. Dieses Verhalten ist im wesentlichen darauf zurückzuführen, dass der Zugriff auf Elemente mittels Indizes im Falle der LinkedList eine Iteration über deren Elemente erfordert (siehe sun microsystems, java.util.LinkedList). Die Verwendung von java.util.LinkedList ist demnach in der Regel zu vermeiden.

Weiterhin sollte die Verwendung rekursiver Methodenaufrufe sorgfältig geprüft werden. In Fällen, wo die Rekursionstiefe sehr groß werden kann, muss mit einem Überlauf des Call-Stacks gerechnet werden. In diesen Fällen sollte der rekursive Aufruf von vornherein durch eine Schleife ersetzt werden.

In der Java 2 Standard Edition 5.0 wurde im Vergleich zur vorherigen Version das Collection-Framework deutlich ausgebaut. Neben anderen Neuerungen gibt es nun eine Heap-basierte Priority Queue (java.util.PriorityQueue). Diese Priority Queue wird in einer der Implementierungen des Dijkstra-Algorithmus verwendet und mit anderen Algorithmen verglichen.

4.2 Beschreibung der Testmodelle

Für die Mitfahrerbörse wurden zwei Einsatzgebiete ermittelt. Zum einen der Einsatz als innerbetrieblicher Service, zum anderen der Einsatz als öffentliches Angebot. Für diese Einsatzgebiete wurden die folgenden Testszenarien festgelegt.



a) Luftlinien von ca. 200 konstruierten Mitfahrgelegenheiten

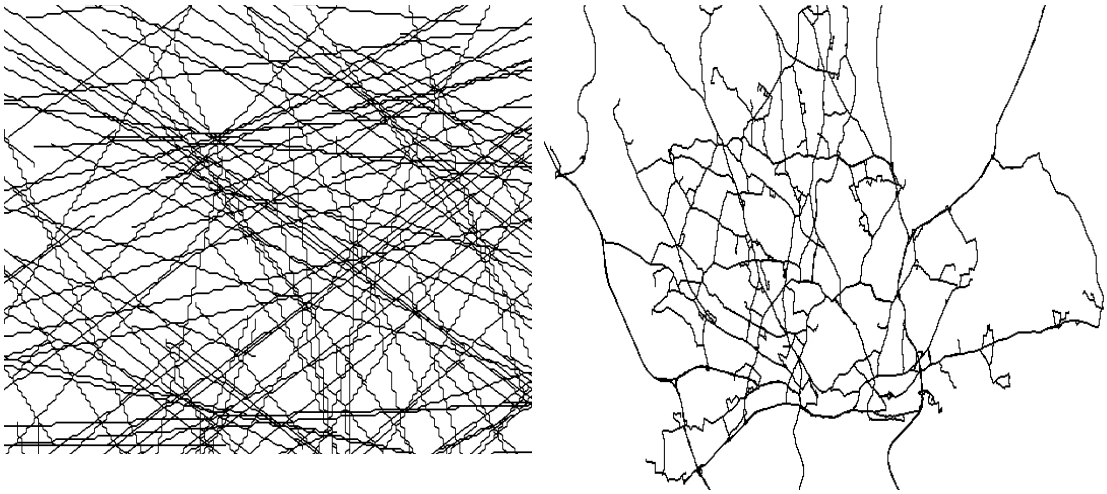
b) Ca. 200 durch einen Routenplaner ermittelte Fahrtwege

Abbildung 4.1: Testfall 1: typischer Routenverlauf einer betrieblichen Mitfahrerbörse

„Testfall 1“ spiegelt den Einsatz der Mitfahrerbörse bei einem großen Arbeitgeber wieder. Alle Angebote und Nachfragen haben das gleiche Ziel. Daraus ergibt sich eine sternförmige Anordnung der Routen. In Abbildung 4.1(a) sind ca. 200 zufällig gewählte Routen abgebildet, deren Verläufe den Luftlinien entsprechen. In Abbildung 4.1(b) dagegen sind ca. 200 Routen abgebildet, deren Start- und Zielpunkte Adressen aus dem Großbereich Hamburg sind. Die Verläufe der hier abgebildeten Routen wurden von einem Routenplaner (Map and Guide) ermittelt. Der Verlauf der Luftlinien verdeutlicht, dass eine Mitfahrgelegenheit in nahezu konstanter Zeit gefunden werden kann, da hier nur geprüft werden muss, ob eine Route zum Zeitpunkt der Anfrage in der Nähe des gewünschten Startpunktes verläuft. Der Verlauf in einem realen Straßennetz aber zeigt, dass viele Routen in weiten Teilen gleich verlaufen. Daraus ergeben sich eine große Anzahl unsinniger Umstiegsmöglichkeiten. Es ist also anzunehmen, dass einige Algorithmen in einem realen Straßennetz sehr viel schlechtere Ergebnisse liefern als die Betrachtung der Luftlinien vermuten lässt. Da die Algorithmen in dem entwickelten System sehr einfach austauschbar sind, sollte für diesen speziellen Fall ein Algorithmus implementiert werden, der einfach die Route des ermittelten Startpunktes

liefert. Die Mitfahrerbörse in einem betrieblichen Umfeld ist also ein vereinfachter Spezialfall der Mitfahrerbörse. Da dieses Szenario durch „Testfall 2“ mit abgedeckt wird, wird es nicht separat getestet.

„Testfall 2“ spiegelt den Einsatz der Mitfahrerbörse als öffentliches zugänglichen Service wieder. In Abbildung 4.2(a) sind ca. 200 zufällig gewählte Routen abgebildet, deren Verläufe den Luftlinien entsprechen. In Abbildung 4.2(b) dagegen sind ca. 200 Routen abgebildet deren Start- und Zielpunkte Adressen aus dem Großbereich Hamburg sind. Die Verläufe der hier abgebildeten Routen wurden von einem Routenplaner (Map and Guide) ermittelt. Die meisten Angebote haben unterschiedliche Ziele. Hieraus ergibt sich eine Netz von Routen. Da die Laufzeit von Shortest-Path-Algorithmen von der Anzahl der Ecken bzw. Kanten abhängt, ist der theoretische Testfall so konstruiert worden, dass es eine möglichst große Anzahl von Kreuzungspunkten gibt. Der Verlauf in einem realen Straßennetz hingegen zeigt, dass viele Routen auf weiten Teilen gleich verlaufen. Da es somit sehr viel weniger unterschiedliche Datenpunkte gibt, kann davon ausgegangen werden, dass der Speicherbedarf für reale Routen kleiner ist.



a) Luftlinien von ca. 200 konstruierten Mitfahrgelegenheiten

b) Ca. 200 durch einen Routenplaner ermittelte Fahrtwege

Abbildung 4.2: Testfall 2: typischer Routenverlauf einer öffentlichen Mitfahrerbörse

Während der Evaluierung wurden die in Tabelle 4.1 aufgelisteten Algorithmen untersucht.

DKS	Standardimplementierung des Dijkstra-Algorithmus
DKSLC	Implementierung des Dijkstra-Algorithmus als Label-Correcting Verfahren
DKSLCH	Implementierung des Dijkstra-Algorithmus als Label-Correcting Verfahren mit Heap (hier java.util.PriorityQueue)
DKSLCRH	Implementierung des Dijkstra Algorithmus als Label-Correcting Verfahren mit RadixHeap
FMB	Implementierung des Ford-Moore-Algorithmus mit FiFo-Strategie und Beschränkung der Suchtiefe auf 3
AS	Implementierung des A*-Algorithmus mit euklidischer Distanz als Schätzfunktion

Tabelle 4.1: Getestete Algorithmen

4.3 Laufzeit: Algorithmen und Modell

Das im vorigen Kapitel entwickelte Modell und die ausgewählten Algorithmen wurden anhand konstruierter Testdaten gemessen. Zu diesem Zweck wurden die Routen aus Linien mit zufällig gewählten Start- und Zielpunkten gebildet. Die konstruierten Routen decken eine Fläche von ca. $30km^2$ ab und haben jeweils eine Länge von bis zu $20km$. Der Verlauf der Routen entspricht dem Verlauf der Routen in Abbildung 4.2 (a). Durch dieses Vorgehen läßt sich die Performance des entwickelten Modells beurteilen, ohne dass die Ergebnisse durch den Overhead des Routensystems verwischt werden. Da das System unabhängig vom verwendeten Routensystem ist und verschiedene Routensysteme voraussichtlich unterschiedlich performant sind, ergeben die nachfolgend dargestellten Werte die obere Schranke für die Performance des Gesamtsystems.

Die in den Tabellen dargestellten Werte sind, soweit nicht anders angegeben, Mittelwerte aus jeweils 1000 Messungen.

In Tabelle 4.2 ist die Zeit, die zur Aktivierung einer Route benötigt wird, sowie die Zeit zum Auffinden des jeweiligen Start- und Zielpunktes in Abhängigkeit von der Größe des Graphen dargestellt.

In Tabelle 4.3 wird in Abhängigkeit der Größe des Graphen dargestellt, wie viele Ecken im Verlauf einer Suche durchschnittlich von den Algorithmen untersucht werden.

In Tabelle 4.4 wird in Abhängigkeit der Größe des Graphen dargestellt, wie lange die verschiedenen Algorithmen durchschnittlich für das Routenmatching benötigen.

Routen	Wegpunkte	activation time (max. in ms)	Start/Ziel finden (ms)	Kanten
200	29883	200	0,5	12308
400	57038	200	0,5	48194
600	85328	200	0,5	110304
800	111914	200	0,5	184488
1000	138566	200	0,5	284870
1200	164925	200	0,5	406016

Tabelle 4.2: Performance: SortedList

Die hier dargestellten Messungen zeigen, dass sowohl die Aktivierung der Routen wie auch das Auffinden des Start- und Zielpunktes einer Suchanfrage in nahezu konstanter Zeit erfolgt. Zum Auffinden der Kreuzungspunkte der Routen muss bei der Aktivierung einer Route für jeden ihrer Wegpunkte geprüft werden, ob ein Wegpunkt mit gleichen Koordinaten bereits im Graphen vorhanden ist. Diese Prüfung entspricht vom Aufwand her der Suche nach dem jeweiligen Start- und Zielpunkt.

Routen	DKS	DKSLC	DKSLCH	DKSLCRH	FMB	AS
200	200	23	23	23	7	35
400	400	35	35	35	3	54
600	600	63	63	63	2	106
800	800	91	91	91	10	170
1000	1000	102	102	102	13	206
1200	1200	115	115	115	18	239

Tabelle 4.3: Untesuchte Knoten in einem Netz aus konstruierten Routen

Routen	DKS	DKSLC	DKSLCH	DKSLCRH	FMB	AS
200	3,0	0,5	1,0	1,0	0,5	1,0
400	10,0	2,0	2,0	2,0	0,5	3,0
600	19,5	5,5	5,5	5,5	0,5	8,5
800	28,5	9,0	8,5	9,0	1,0	16,0
1000	39,0	11,5	11,0	11,5	1,5	22,0
1200	53,5	15,0	14,0	14,0	2,5	29,5

Tabelle 4.4: Laufzeit in einem Netz aus konstruierten Routen

Weiterhin zeigt sich, dass die Kombination des Dijkstra-Algorithmus mit der Label-Correcting-Strategie die Zahl der zu untersuchenden Knoten etwa um den Faktor 10 verringert. Auch der A*-Algorithmus untersucht deutlich weniger Knoten als der Standard-Dijkstra-Algorithmus. Am effektivsten ist aber eine Einschränkung der Suchtiefe, wie an den Werten des Ford-Moore-Algorithmus zu sehen ist.

Dass die Zahl der untersuchten Knoten sich direkt auf die Laufzeit auswirkt, ist eine bekannte Tatsache und der Hauptgrund, warum der A*-Algorithmus, der durch die Schätzfunktion eine gezielte Suche realisiert, in vielen Problemstellungen hervorragende Ergebnisse liefert. Hier wurde die euklidische Distanz als Schätzfunktion verwendet. Die Berechnung der euklidischen Distanz ist äußerst performant zu implementieren, eignet sich aber nicht besonders gut, um den Verlauf der Routen vorherzusagen. Dieses spiegelt sich auch in den Ergebnissen wieder. Entspricht der Verlauf der Routen wie in diesem Testfall der Luftlinie zwischen jeweils zwei Punkten, ließe sich im Zweifelsfall eine bessere Schätzfunktion implementieren. In einem realen Straßennetz aber ist der Verlauf einer Route nicht ohne weiteres vorhersagbar.

Wertet man die hier erzielten Ergebnisse als obere Schranke für die Performance des Gesamtsystems, bedeutet das, dass das System bei 1200 aktiven Routen pro Sekunde rund 5 Angebote oder - unter Verwendung des Dijkstra-Algorithmus - 70 Gesuche verarbeiten kann.

4.4 Laufzeitverhalten des Gesamtsystems

Zur Beurteilung des Gesamtsystems wurde dieses zusammen mit dem „eRouteServer“ und „eLocateServer“ der PTV-AG getestet. Das Routensystem lief dazu auf dem gleichen Rechner wie das hier entwickelte System. Die Routen wurden aus jeweils zwei von 150 zufällig aus dem Großraum Hamburg ausgewählten Adressen gebildet. Die Routen decken eine Fläche von ca. $60km^2$ ab und haben eine Länge von jeweils bis zu $30km$. Der Verlauf der Routen wurde durch das Routensystem bestimmt und entspricht dem Verlauf der Routen in Abbildung 4.2 (b).

Die in den Tabellen dargestellten Werte sind, soweit nicht anders angegeben, Mittelwerte aus jeweils 1000 Messungen.

In Tabelle 4.5 ist die Zeit, die zur Aktivierung einer Route benötigt wird, sowie die Zeit zum Auffinden des jeweiligen Start- und Zielpunktes in Abhängigkeit von der Größe des Graphen dargestellt.

In Tabelle 4.6 wird in Abhängigkeit der Größe des Graphen dargestellt, wie viele Ecken im Verlauf einer Suche durchschnittlich von den Algorithmen untersucht werden.

In Tabelle 4.7 in Abhängigkeit der Größe des Graphen dargestellt, wie lange eine die verschiedenen Algorithmen durchschnittlich für das Routenmatching benötigen.

Routen	Wegpunkte	activation time (max. in ms)	Start/Ziel finden (ms)	Kanten
200	28868	23 (+ca 400ms)	0,5 (+ca 400ms)	9948
400	58080	50 (+ca 400ms)	0,5 (+ca 400ms)	43638
600	87783	118 (+ca 400ms)	0,5 (+ca 400ms)	101667
800	118315	223 (+ca 400ms)	0,5 (+ca 400ms)	182920
1000	149212	314 (+ca 400ms)	0,5 (+ca 400ms)	292309
1200	178501	429 (+ca 400ms)	0,5 (+ca 400ms)	426497

Tabelle 4.5: Performance: SortedList

Routen	DKS	DKSLC	DKSLCH	DKSLCRH	FMB	AS
200	200	24	24	24	11	35
400	400	46	46	46	13	82
600	600	90	90	90	11	162
800	800	125	125	125	19	252
1000	1000	221	221	221	14	522
1200	1200	310	310	310	34	671

Tabelle 4.6: Untesuchte Knoten in einem realen Straßennetz

Routen	DKS	DKSLC	DKSLCH	DKSLCRH	FMB	AS
200	3,0	1,0	1,5	1,5	1,0	1,0
400	11,5	4,5	4,5	4,5	2,0	6,5
600	24,0	11,0	9,5	10,0	1,5	16,0
800	39,5	22,0	18,5	18,0	3,5	32,5
1000	61,5	44,5	37,5	35,5	3,0	79,5
1200	98,5	73,0	58,0	55,0	8,5	118,5

Tabelle 4.7: Laufzeit in einem realen Straßennetz

Im Vergleich zu den vorangegangenen Messungen fällt sofort auf, dass die Zahl der untersuchten Knoten während der Suche deutlich höher ist. Dieses ist auf die Strategie des Routensystems zurückzuführen, Verkehrsteilnehmer möglichst über Hauptstraßen zu leiten. Dadurch verlaufen zum einen viele Routen auf weiten Strecken gleich, zum anderen gibt es dadurch eine Reihe von Kreuzungspunkten, an denen sich sehr viele Routen kreuzen. Das Verhältnis der in den verschiedenen Algorithmen untersuchten Knoten bleibt aber unverändert.

Sehr auffällig ist zudem, dass die Aktivierungszeit hier nicht mehr konstant ist, sondern ansteigt. Auch dieses ist auf die Kreuzungspunkte zurückzuführen, da die Nachbarschaftslisten in sämtlichen betroffenen Routen aktualisiert werden müssen.

Das getestete System kann bei 1200 aktiven Routen pro Sekunde rund 1,2 Angebote oder - unter Verwendung des Dijkstra-Algorithmus - 2,2 Gesuche verarbeiten.

4.5 Speicherbedarf des Gesamtsystems

Abschließend wurde der Speicherbedarf des Gesamtsystems gemessen. Dazu wurde zum einen mit betriebssysteminternen Mitteln gemessen, zum anderen mit den Mechanismen, die die JavaVM zur Verfügung stellt. Beide Messungen eignen sich nicht dazu, Vergleiche mit anderen Darstellungsformen des Graphen anzustellen, da in beiden Fällen der Speicherbedarf des Interpreters mit gemessen wird. Diese Werte geben aber dennoch einen Anhaltspunkt für die benötigte Hardwareausstattung. In Abbildung 4.3 sind die Werte graphisch dargestellt, der Speicherbedarf für 1000 aktivierte Routen liegt deutlich unter 100 Megabyte. Es ist also nicht zu erwarten, dass der Speicherbedarf auf einem Serversystem ein Problem darstellt.

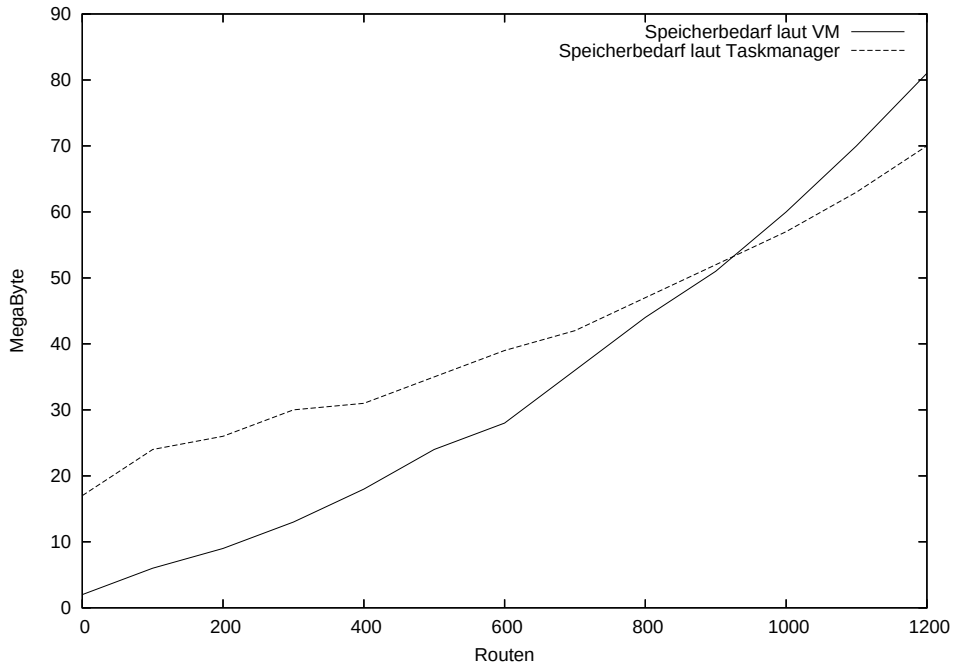


Abbildung 4.3: Hauptspeicherauslastung

5 Diskussion

Ziel dieser Arbeit war das Design einer Mitfahrerbörse auf Basis graphentheoretischer Verfahren. In Tabelle 5.1 werden nochmals die während der Analyse gefundenen Anforderungen aufgelistet und das Ergebnis dieser Arbeit zusammengefasst.

Anforderung	Priorität	erfüllt
Vermittlung spontaner Mitfahrgelegenheiten	hoch	ja
Verteilung eines Reisewunsches auf mehrere Abschnitte	hoch	ja
Minimierung der Antwortzeiten	hoch	ja
Wege mit wenig Umstiegspunkten werden bevorzugt	hoch	ja
Einfaches, verständliches Userinterface	hoch	ja
Ergebnis durch den Benutzer beeinflussbar	mittel	ja
Benutzbarkeit über mobile Devices	mittel	ja
Keine technischen Mindestanforderungen an Clients	mittel	ja
Angebot alternativer Reiserouten	niedrig	nein
Einbindung anderer Verkehrsmittel	niedrig	ja

Tabelle 5.1: erfüllte Anforderungen

Als Einsatzgebiet eines solchen Systems wurde zum einen der Einsatz in einem betriebliches Umfeld, zum anderen der Einsatz als öffentlich zugänglicher Service ermittelt. Dabei ist der Einsatz der Mitfahrerbörse als betriebsinterner Service aus graphentheoretischer Sicht einfach realisierbar. Da alle Benutzer das gleiche Ziel beziehungsweise den gleichen Startpunkt haben, muss lediglich geprüft werden welcher Fahrer in der Nähe des Wohnsitzes des jeweils suchenden Mitfahrers vorbeifährt. Zwischen zwei privaten Fahrern umzusteigen macht in diesem Fall keinen Sinn. Der Einsatz der Mitfahrerbörse in einem öffentlichen Umfeld hingegen bringt aus graphentheoretischer Sicht einige interessante Aspekte mit sich. Dadurch, dass es hier durchaus sinnvoll sein kann, verschiedene Abschnitte der Reise durch verschiedene Mitfahrgelegenheiten abzudecken, ergibt sich ein dynamischer Graph in dem der kürzeste Weg von einem bestimmten Startpunkt zu einem bestimmten Zielpunkt gefunden werden muss.

Ein wesentlicher Faktor ist dabei die Zeit, die benötigt wird, eine Suche durchzuführen. Da es für statische Graphen einige sehr guter Shortest-Path-Algorithmen gibt, wurde ein Mo-

dell zur Abbildung des sich aus den Fahrtangeboten ergebenden Graphen entwickelt, das die dynamischen Aspekte soweit kapselt, dass die meisten dieser Algorithmen ohne große Anpassungen anwendbar werden. In diesem Zusammenhang wird gezeigt,dass sich die Performance des Dijkstra-Algorithmus für das hier entwickelte dynamische Modell durch Umbau in ein Label-Correcting-Verfahren deutlich steigern lässt.

Die besten Ergebnisse werden aber, wie anhand der Evaluierung des Prototyps gezeigt wurde, mit dem Ford-Moore-Algorithmus erzielt. Mit einer Beschränkung der Suchtiefe auf maximal drei verschiedene Fahrer pro Reise kann die Suche nach einer passenden Kombination von Mitfahrgelegenheiten in einem Pool von 1000 Angeboten in rund 5ms berechnet werden. Im Rahmen der Analyse wurde gefordert, dass die Suche nach einer Mitfahrgelegenheit in einem Pool von 600 Angeboten in unter 5s berechnet wird. Selbst mit einer Antwortzeit des hier verwendeten Routenplaners von ca. 400ms ist die Suche damit rund 10 mal schneller als gefordert.

Einen Reisewunsch in mehrere Abschnitte aufzuteilen ist technisch realisierbar. Es gilt jedoch zu bedenken, dass ein Routenplaner die Fahrzeit aufgrund von Durchschnittsgeschwindigkeiten berechnet. Je mehr Umstiegspunkte in einer Reise enthalten sind, umso höher ist die Wahrscheinlichkeit, dass der Reiseplan nicht eingehalten werden kann. Je genauer die Schätzungen sind, desto wahrscheinlicher ist es, dass die Reise wie geplant verläuft. Um gute Schätzungen zu erhalten, ist die Kopplung mit einem kollektiven Telematiksystem zur Fahrzeitermittlung ein denkbarer Ansatzpunkt. Eine weitere Möglichkeit besteht darin, von Fahrern und Mitfahrern am Ende jeder Reise Fahrzeitangaben einzufordern. Aber selbst, wenn das System zuverlässige Ergebnisse über mehrere Teilabschnitte liefern kann, darf dabei nicht außer Acht gelassen werden, das hier Personen befördert werden und zu häufiges Umsteigen in der Regel als stressig empfunden wird und daher nicht zumutbar ist. Die Beschränkung der Suchtiefe durch den Ford-Moore-Algorithmus ist vor diesem Hintergrund also durchaus Sinnvoll.

Aufgrund der Performance und der Möglichkeit, die Suchtiefe einzuschränken, sollte also der Ford-Moore-Algorithmus bei einer Umsetzung der Mitfahrerbörse bevorzugt werden.

Öffentliche Verkehrsmittel können durch das flexible Datenmodell einfach berücksichtigt werden. Da der genaue Streckenverlauf für einen Benutzer nicht relevant ist, kann eine U-Bahn-Linie als Route dargestellt werden, deren Wegpunktliste nur die Geokoordinaten der einzelnen Haltestellen beinhaltet. Ein bisher offener Punkt in diesem Zusammenhang ist die Abbildung von Linienfahrplänen. Das periodische Einstellen einer solchen Route ist dabei ein denkbarer Ansatzpunkt.

Ein weiterer Faktor für den Erfolg vieler IT-Systemen ist ein einfach bedienbares Benutzerinterface. Für eine Mitfahrerbörse zur Vermittlung spontaner Mitfahrgelegenheiten ist dies aber unerlässlich, da ein Benutzer in dem Moment eine Anfrage an das System stellt, in dem er die Fahrt antreten möchte. Aus diesem Grund wurde während der Designs ein intuitiv zu

bedienender Webclient entworfen. Der Benutzer benötigt in diesem Fall also nur einen Browser, um Zugang zu dem System zu erhalten. Browser und Internetanschluss gehören auf Desktop Pc's heutzutage zur Standardausrüstung und sind auch für die meisten PDA's sowie für viele moderne Mobiltelefone verfügbar.

Das Angebot mehrerer alternativer Ergebnisse ist bisher noch offen, da die hier getesteten Algorithmen nur den besten Weg durch einen Graphen liefern. In diesem Zusammenhang muss auch nochmals die Strategie zur Reservierung von Fahrten überdacht werden.

Mit den genannten Punkten wurden alle notwendigen Anforderungen erfüllt, und somit das Ziel dieser Arbeit erreicht.

5.1 Ausblick

Diese Arbeit liefert die technische Grundlage, Transportprobleme verschiedener Art aus einem neuen Blickwinkel zu betrachten. Feldversuche müssen nun zeigen, ob ein System dieser Art angenommen wird. Das Angebot betrieblicher Mitfahrerbörsen ist dazu gut geeignet, da sich die Nutzer hier zwar untereinander nicht unbedingt kennen, aber immerhin durch den gemeinsamen Arbeitgeber verbunden sind. Eine Firma bietet daher auf menschlicher Ebene ein geeignetes Umfeld, das hier vorgestellte Konzept zu etablieren.

Da sich im Verlauf der Arbeit gezeigt hat, dass eine gute Schätzung der Durchfahrtzeiten sehr wichtig ist, sollten in diesem Bereich weitere Nachforschungen angestellt werden. Die Ortung von Mobiltelefonen ist in diesem Zusammenhang ein Punkt, der näher untersucht werden könnte.

In den Szenarien wurde der denkbare Transportverlauf innerhalb eines Kurierdienstes dargestellt. Analog dazu könnte das hier beschriebene System dazu genutzt werden, eine Reihe kurzfristig anfallender Transporte ressourcensparend zu planen. Dabei ist insbesondere die effiziente Abbildung der in der Logistikbranche oft sehr komplexen Tarifstrukturen interessant, da diese Tarife in die Berechnung der Kantenkosten einfließen müssen.

Die Erweiterung des Konzepts der Mitfahrerbörse auf ein überregionales Reiseportal ist ein weiterer Punkt, der interessant zu untersuchen wäre. Hierbei ist zum einen die Zusammenfassung verschiedener Systeme unter einer einheitlichen Benutzerschnittstelle interessant, zum anderen ist die Entwicklung effizienter parallelisierbarer Algorithmen zur Beantwortung des gegebenenfalls sehr hohen Anfragevolumens interessant.

Das hier entwickelte System kann gegebenenfalls zu einer effizienteren Ausnutzung der verfügbaren Transportmittel führen und somit den Verkehr auf den Straßen effektiv reduzieren.

A Anhang

A.1 Glossar

Adjazenzmatrix:

Sei $G = (V, E)$ ein gerichteter Graph und $n = \|V\|$ die Anzahl der Knoten V. Die Adjazenzmatrix $A = \{1, 0\}^{n \times n}$ des gerichteten Graphen G wird definiert durch:

$$A_{ij} = \begin{cases} 1 & \text{wenn } (i, j) \in E \\ 0 & \text{sonst} \end{cases} \tag{A.1}$$

In gewichteten schlichten Graphen kann in die Adjazenzmatrix an der Stelle A_{ij} das Gewicht der Kante E_{ij} eingetragen werden.

Euklidische Distanz:

Der Euklidische Abstand ist die mathematische Definition des normalen Abstands. Er ist als Distanzfunktion über zwei Vektoren definiert und berechnet sich als euklidische Norm des Differenzvektors zwischen den beiden Vektoren.

$$d(x, y) = |x - y| = \sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2} = \sqrt{\sum (x_i - y_i)^2} \tag{A.2}$$

Geoinformationssystem (GIS):

Ein Geoinformationssystem ist ein Informationssystem, mit dem raumbezogene Daten digital erfasst und redigiert, gespeichert und reorganisiert, modelliert und analysiert sowie alphanumerisch und graphisch präsentiert werden. Es vereint eine Datenbank und die zur Bearbeitung und Darstellung dieser Daten nützlichen Methoden (Kurzdefinition nach Fédération Internationale des Géomètres).

Prioritätswarteschlange (Priority-Queue)

Eine Prioritätswarteschlange ist eine Datenstruktur, die einer Queue oder einem Heap ähnelt. Sie stellt ebenfalls Methoden zum Einfügen und zum Entfernen von Elementen bereit. Anders als bei diesen einfachen Datenstrukturen wird aber bei einer Prioritätswarteschlange immer das Element mit der höchsten Priorität entfernt.

Radix Heap

Radix-Heaps stellen eine Möglichkeit zur effizienten Realisierung von Priority-Queues dar. Die Elemente werden in Buckets mit exponentiell ansteigendem Wertebereich vorsortiert. Bezeichnet C den Wertebereich aller Elemente, entspricht die amortisierte Laufzeit der langsamsten Zugriffsfunktion $O(logC)$.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

Aho u. a. 1974 AHO, Alfred V. ; HOPCROFT, John E. ; ULLMAN, Jeffrey D.: *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974

Ahuja u. a. 1990 AHUJA, Ravindra K. ; MEHLHORN, Kurt ; ORLIN, James ; TARJAN, Robert E.: Faster algorithms for the Shortest Path Problem. In: *Journal of the ACM* 37 (1990), S. S. 213 – 223

Balzert 2000 BALZERT, Helmut: *Lehrbuch der Softwaretechnik*. Heidelberg, Berlin : Spektrum, akad. Verlag, 2000

Bersekas 1993 BERSEKAS, Dimitri B.: A Simple and Fast Label Correcting Algorithm for Shortest Paths. In: *Networks* 23 (1993), S. S. 703–709

Halbritter u. a. 2002 HALBRITTER, Günter ; BRÄUTIGAM, Rainer ; FLEISCHER, Torsten: *Verkehr in Ballungsräumen*. Berlin : Erich Schmidt Verlag, 2002

HansestadtHamburg 2001 HANSESTADTHAMBURG: *Fahrgeschwindigkeiten im Straßenverkehr*. 09 2001. – URL `www.hamburg.de/fhh/aktuelle_meldungen/archiv_2001/september/pe_2001_09_11_bb_02.htm`. – Zugriffsdatum: 19.08.2004. – Pressemeldung der staatlichen Pressestelle der Freien und Hansestadt Hamburg

Hasselberg 2000 HASSELBERG, Stephan: *Some results on heuristical algorithms for shortest path problems in large road networks*, Mathematisch-Naturwissenschaftliche Fakultät der Universität zu Köln, Dissertation, 2000

Homann 2004 HOMANN, Klaus: *Der Individualverkehr wird öffentlicher, Ein neues Modell zur Vermeidung von PKW-Fahrleistungen*. 01 2004. – betriebsinterne Studie der eNotions GmbH

Klauck und Maas 1999 KLAUCK, Christoph ; MAAS, Christoph: *Graphentheorie und Operations Research für Studierende der Informatik*. 3. veränd. Auflage. Augsburg : Wißner, 1999

Resnick 2003 RESNICK, Paul: *SocioTechnical Support for Ride Sharing*. 2003. – URL `http://www.caup.umich.edu/acadpgm/urp/utepsymposium/papers.html`. – Zugriffsdatum: 24.10.2004

Sedgewick 1992 SEDGEWICK, Robert: *Algorithmen in C++*. Addison-Wesley, 1992

sun microsystems sun microsystems (Veranst.): *JavaTM 2 Platform Standard Edition API Specification*. 5.0. – URL `http://java.sun.com/j2se/1.5.0/docs/api/index.html`

Vahrenkamp 2003 VAHRENKAMP, Richard: *Quantitative Logistik für das Supply-chain-Management*. München, Wien : Oldenbourg Verlag, 2003

Welk und Kage 2000 WELK, Lars ; KAGE, Jana: Betriebliche Mobilität gestalten. In: *Aktuell, Informationen zum BG/DVR* 1 (2000), S. S. 13

Zhan 1997 ZHAN, F. B.: Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures. In: *Journal of Geographic Information and Decision Analysis* 1 (1997), Nr. 1, S. S. 69–82

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 29. Oktober 2004	
Ort, Datum	Unterschrift