Identifikation und Visualisierung von gestauten Verkehrsmustern in Straßennetzen mit Hilfe von FCD

Masterthesis

zur Erlangung des akademischen Grades

Master of Engineering „M.Eng.“

Reg.-Nr. TM06/16/SS2008

Technische Fachhochschule Wildau

Fachbereich Ingenieurwesen

Studiengang Telematik

Eingereicht von:	Gabriel Voigt
Geb. am:	29.11.1983
Betreuer:	Prof. Dr. Anselm Fabig Prof. Dr. Stefan Brunthaler M.Eng. Sten Ruppe
Themenstellender Betrieb:	Deutsches Zentrum für Luft- und Raumfahrt in der Helmholtz-Gemeinschaft Institut für Verkehrssystemtechnik

Wildau, 13.02.2009

Deckblatt

Verlängerung

Bibliographische Beschreibung und Referat

Voigt, Gabriel

Identifikation und Visualisierung von Verkehrsknotenpunkten in Straßennetzen mit gestauten Verkehrsmustern

Masterthesis, Technische Fachhochschule Wildau 2008-2009, 134 Seiten, 44 Abbildungen, 5 Tabellen, 32 Quellenangaben, 1 Beilage.

Ziel:

Erstellen einer wiederverwendbaren Schnittstelle für die Darstellung von Kartenmaterial und Nutzung dieser Schnittstelle zur Visualisierung von gestauten Verkehrsmustern.

Für die Darstellung wird Kartenmaterial in Form von Vektorkarten in ein komprimiertes Format überführt. Ein Softwaremodul zum Laden dieser Karten wird implementiert. Die Karten werden mit Hilfe einer Schnittstelle für die Visualisierung dargestellt. Floating Car Data werden ausgewertet, um Knotenpunkte in Straßennetzen zu erkennen, an denen es Verkehrsbeeinträchtigungen gibt. Diese Knotenpunkte werden mit Hilfe der erstellten Schnittstelle zur Visualisierung im Straßennetz dargestellt.

Inhalt:

Beschreibung von Grundlagen für die Identifikation und Bewertung von Knotenpunkten in Straßennetzen.

Beschreibung von Floating Car Data.

Untersuchung von Kartenmaterial der Firma NAVTEQ.

Konzeptionelle Beschreibung der Softwarekomponenten zur Kartenkompression, Kartenverwendung, Kartendarstellung und FCD-Auswertung.

Programmdokumentation der Softwaremodule.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Abschlussarbeit selbstständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Wildau, den 13.02.2009

Unterschrift

Begriffserklärungen

RMI / Remote Method Invocation: Eine Java Technologie, um entfernte Methoden aufzurufen. Hiermit können Methoden von einem Programm über ein Netzwerk aufgerufen werden, die auf einem entfernten Endgerät ausgeführt werden.

FCD: Abkürzung für den Begriff „Floating Car Data“ (engl.: „probe vehicle Data“)

FCD-Positionen: FCD-Rohdaten, einzelne Geopositionen, die von bestimmten Fahrzeugen zu einem bestimmten Zeitpunkt übermittelt werden und keinen Bezug zum Kartenmaterial besitzen

FCD-Treffer: FCD-Positionen, die einer betrachteten Stelle im Kartenmaterial, hier im Regelfall Knotenpunkte, zugeordnet werden und somit Bezug zum Straßensystem bekommen

MySQL: Frei verfügbares Datenbanksystem auf SQL-Basis zum Speichern großer Datenmengen

GUI: Abkürzung für den Begriff „Graphical User Interface“, zu deutsch: Grafische Benutzerschnittstelle

PDA: Abkürzung für den Begriff „Personal Digital Assistant“, ein sehr kleiner, portabler Computer, auch „Handheld PC“ oder „Pocket PC“ genannt

DLR: Abkürzung für „Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmoltz Gesellschaft“

DLR-TS: Institut für Verkehrssystemtechnik im DLR

Verkehrsmanagementzentrale (VMZ-Berlin): Siemens-Tochterunternehmen mit Sitz in Berlin, hat Aufgaben bezüglich der Schaltung von Verkehrsinformationen und Verkehrsanalyse

TMC: Abkürzung für den Begriff „Traffic Message Channel“, ein System zur Übertragung von Verkehrsbeeinträchtigungen über das Radio

GPS: Abkürzung für den Begriff „Global Positioning System“, ein System zur weltweiten Positionsbestimmung

ÖPNV: Abkürzung für den Begriff „Öffentlicher Personen-Nahverkehr“, eine Zusammenfassung der öffentlichen Verkehrsmittel wie Linienbus oder Bahn

Geoposition: Die Angabe eines Punktes durch Längen- und Breitengrad, der einen Punkt auf der Erdoberfläche repräsentiert

Inhaltsverzeichnis

1 Einleitung

Mobilität ist die Fähigkeit, ungebunden und beweglich zu agieren. Für den Menschen ist der Begriff Mobilität, in seiner Ausprägung als Bezeichnung für die Möglichkeit, ortsungebunden zu sein, in der heutigen Zeit zu einem wichtigen Schlagwort geworden. Der Mensch empfindet, bewusst oder unbewusst, die Möglichkeit, sich frei bewegen zu können, als eines seiner höchsten Güter.

Dieses Empfinden spiegelt die gesellschaftliche, wirtschaftliche und technologische Entwicklung der letzten Jahrzehnte wieder. In vielen Bereichen von Gesellschaft und Technologie sind Gründe für den ständig wachsenden Bedarf an Mobilität zu finden, die Einflussfaktoren darstellen, welche in der Summe zu einem großen Ganzen führen. Der wachsende Bedarf an Mobilität führt zu einem ständigen Wachstum des öffentlichen und individuellen Personenverkehrs sowie des Güterverkehrs.

Die gesellschaftlichen Gründe sind vielfältig. Ein Einflussfaktor ist beispielsweise die derzeitige Arbeitsmarktsituation. Diese führt zum Umstand, dass das Wechseln der Arbeitsplätze häufig geschieht, während ein ständiger Wechsel der Wohnsituation für Menschen unzumutbar ist. Zudem ist die Wahl des Wohnortes ein komplexes Problem, das sich nicht zwangsläufig auf die Entfernung zum Arbeitsplatz bezieht. Oftmals leben Menschen heute nicht dort, wo sie arbeiten. Ein großer Teil der Arbeitnehmer fährt täglich mehrere Kilometer zum Arbeitsplatz. Auch von staatlicher Seite erhalten die Pendler Unterstützung in Form der Pendlerpauschale. Ein großer Teil dieser Personen nutzt den motorisierten Individualverkehr, fährt also beispielsweise mit dem Auto zur Arbeit. Diese tägliche Bewegung von vielen Menschen führt zu einer drastischen Zunahme der Auslastung des Straßenverkehrs. Worte wie „Berufsverkehr“ und „Rushhour“ sind seit Jahren stehende Begriffe in Deutschland.

Der technologische Fortschritt bezüglich der Kommunikationsmöglichkeiten führt zudem auch dazu, dass Menschen ihre Freizeitgewohnheiten ändern. Gerade viele junge Menschen legen weite Strecken zurück, um ihren Freizeitbeschäftigungen nachzugehen, seien dies die Fahrten zum Sport, in den Urlaub, zu weit entfernt wohnenden Freunden und Bekannten oder zu Orten für die Abendgestaltung.

Eine ständige Zunahme des Beförderungsentgeltes für die öffentlichen Verkehrsmittel sowie der Umstand, dass durch die technologische Entwicklung für die meisten Menschen ein eigenes Kraftfahrzeug finanzierbar ist, führt dazu, dass viele Menschen den Individualverkehr dem öffentlichen Personennahverkehr bevorzugen. Auch fällen viele Menschen die Entscheidung, ob sie eine Strecke mit Hilfe des öffentlichen Personennahverkehrs oder mit dem Auto zurücklegen, nach subjektiven Gesichtspunkten. So liegt es für Viele nahe, dass sie eine Strecke schneller, und vor allem bequemer mit dem eigenen Auto zurücklegen können.

Eine rasante Zunahme ist auch im Sektor des Straßengüterverkehrs zu beobachten. Immer

mehr Güter werden mit Lastkraftwagen auf den europäischen Straßen transportiert.

Ein Beispiel für die Zunahme der Auslastung von Straßennetzen: von 1992 bis 2006 stieg die Verkehrsleistung im motorisierten Individualverkehr von ca. 740 Mrd. Personenkilometer auf ca. 900 Mrd. Personenkilometer. [18] Es legen also immer mehr Menschen weite Strecken mit dem Auto oder dem Motorrad zurück. Die Folge dieser Umstände ist zunächst eine ständige Zunahme der Anzahl an Fahrzeugen, die sich auf den Straßen befinden. Mehr Fahrzeuge führen zu einer höheren Auslastung des Straßennetzes. So stieg zum Beispiel die durchschnittliche tägliche Verkehrsstärke auf deutschen Autobahnen von 1993 bis 2006 von 42.700 Kraftfahrzeugen auf 48.100 Fahrzeuge pro Tag [29]. Der Trend der Zunahme von Fahrleistungen auf anderen Straßentypen ist ähnlich. Die Folge des dichten Verkehrs sind Staus. Ende 2007 wurde geschätzt, dass „Staus auf einer Länge von 7500 Kilometern [...] täglich den Verkehr auf Europas Hauptstraßen“ behindern (zit. Christian Piehler, Programmdirektor Verkehr des DLR, 2007, [1]). Die Folgen eines hohen Aufkommens von Stau liegen auf der Hand. Zum einen sind dies eine hohe Umweltbelastung, zum anderen eine Einschränkung der Mobilität. Verspätetes Ankommen am Zielort ist der Anfang, bei einer weiteren Zunahme von Staus wäre jedoch „Stillstand mit massiven Folgewirkungen für unsere Wirtschaft [...] die Folge“ (zit. Michael Glos, Bundesminister für Wirtschaft und Technologie, 2007, [1]).

Um dem entgegen zu wirken, bieten sich zahlreiche Ansätze an. Zum einen wäre ein attraktiveres öffentliches Nahverkehrssystem ein Anlass für viele Menschen, das Auto stehen zu lassen. Dies hängt jedoch von zahlreichen wirtschaftlichen Faktoren ab, die nicht ohne Weiteres beeinflussbar sind.

Eine Alternative, das Verkehrsnetz zu entlasten, ohne den Verkehr zu reduzieren, wäre der Ausbau des Straßennetzes. Ein weiterer Ausbau der europäischen Straßen ist jedoch nur bedingt möglich. Straßen haben eine relativ hohe geographische Ausdehnung. Nicht überall jedoch existiert der Platz, um weitere Straßen bauen zu können. Naturgegebenheiten und vorhandene Bebauung in Städten verhindern eine große Erweiterung des Netzes. Der Kostenfaktor und der Aufwand für den Straßenbau spielen ebenso eine nicht zu unterschätzende Rolle.

Optimierungen des Verkehrsablaufes sind auch mit der vorhandenen Verkehrsinfrastruktur möglich. Große Verbesserungsmöglichkeiten ergeben sich beispielsweise durch die Optimierung der Steuerung von Lichtsignalanlagen im innerstädtischen Bereich. Unter Anderem wird in diesem Bereich vom Institut für Verkehrssystemtechnik des Deutschen Zentrum für Luft- und Raumfahrt (DLR-TS) im Rahmen des Projektes „ORINOKO“ geforscht.

Weiterhin ist es möglich, den Verkehr mit verbesserten, dynamischen Zielführungssystemen zu beeinflussen. Navigationssysteme, die Reiserouten dynamisch aus aktuellen Verkehrssituationen errechnen, können ihre Benutzer schneller und effizienter zum Ziel führen. Staus können umfahren, weniger stark ausgelastete Straßen als Ausweichrouten genutzt werden.

Auch andere Optimierungsmöglichkeiten sind denkbar. Eines jedoch haben alle diese Ansätze gemeinsam: Sie setzen eine genaue Kenntnis der Probleme voraus. Stau ist ein Phänomen, das nicht ganzheitlich und allgemein in einem Verkehrsnetz auftaucht, sondern geographisch und zeitlich eher punktuell. Es gibt regelmäßig auftretende Probleme, wie Stau zur Zeit des Berufsverkehrs und auf Hauptverkehrsstraßen. Verursacht werden diese unter Anderem durch schlechte Regelungen des Verkehrsablaufes oder das Fehlen von Alternativrouten. Unregelmäßig auftretende Probleme sind ebenfalls möglich und werden beispielsweise durch Verkehrsunfälle oder Baustellen verursacht.

Zur Erkennung von Problemen, oder allgemein zur Bestimmung des Zustands der Verkehrssituation, gibt es viele Möglichkeiten. An Straßen und Kreuzungen können Anlagen wie Zählschleifen installiert werden, die einen Aufschluss über die Anzahl und die Geschwindigkeit der vorbei fahrenden Fahrzeuge geben können. Solche Anlagen können jedoch nur punktuell eingesetzt werden, da ihre Installation, Wartung und Auswertung einen hohen Aufwand und hohe Kosten erzeugen. Gerade in finanziell schwächeren Ländern ist zudem eine flächendeckende Erfassung des Verkehrs mittels solcher Anlagen undenkbar.

Eine immer noch häufig eingesetzte Möglichkeit zur Bewertung von Kreuzungen ist die manuelle Verkehrszählung. Über einen Zeitraum von zumeist einigen Stunden werden dabei Fahrzeuge manuell von damit beauftragen Personen gezählt. Dies ist somit sowohl zeitlich als auch geographisch nur sehr punktuell möglich und setzt voraus, dass vor der Zählung bekannt ist, welche Stelle im Verkehrsnetz ein Problem darstellen könnte.

Mit Satellitenbildern wäre es möglich, zumindest Momentaufnahmen der Verkehrssituation auszuwerten, dies auch flächendeckend. Dies ist jedoch teuer und sehr aufwendig. Zudem stößt man hier sehr schnell an technische Grenzen. Staus können erfasst werden, jedoch nicht regelmäßig, sondern nur zu einem Zeitpunkt. Geschwindigkeitsmessungen oder die Analyse von Reisezeiten sind damit zum jetzigen Zeitpunkt quasi unmöglich.

Ein Ansatz, den unter Anderem das DLR seit einigen Jahren verfolgt, ist die Analyse der Verkehrssituation aus so genannten „Floating Car Data“ (im englischen Sprachraum: „probe vehicle data“). „Floating Car Data“, oder kurz „FCD“, ist eine Bezeichnung für eine Ansammlung von Positionsdaten bestimmter Fahrzeuge. Wenn von einem Fahrzeug bekannt ist, zu welchem Zeitpunkt es sich an welcher Position aufgehalten hat, lassen sich daraus viele interessante Schlüsse ziehen. Dies geht von einer Schätzung der Reisezeiten für eine Route bis hin zu einer automatisierten Erkennung von Straßen oder Straßenabschnitten, die in digitalem Kartenmaterial nicht oder nur unvollständig erfasst sind. Existieren solche Daten von einer ganzen Fahrzeugflotte, kann überall dort, wo sich die Fahrzeuge der Flotte bewegen, die Verkehrssituation analysiert werden. Ist die Flotte groß genug und bewegen sich die Fahrzeuge auf sehr unterschiedlichen Wegen, kann dieser Ansatz zu einer nahezu ganzheitlichen, flächendeckenden Verkehrsbeobachtung führen [21].

Genau an diesem Punkt setzt diese Arbeit an. Im Institut für Verkehrssystemtechnik des DLR wird seit einiger Zeit daran geforscht, wie „Floating Car Data“ ausgewertet werden können,

um einen wirtschaftlichen Nutzwert zu erzeugen. DLT-TS stehen Daten aus Taxiflotten einiger Städte bereit, unter Anderem aus Berlin, Hamburg und Nürnberg. Bereits seit Jahren existiert ein System, mit dessen Hilfe es möglich ist, Reisezeiten für bestimmte Abschnitte eines Verkehrsnetzes zu bestimmen. Auch für die Optimierung von Ampelsteuerungen auf Basis von FCD als Eingangsdaten wird geforscht. Die Bewertung von Verkehrssteuerungssystemen gehört ebenso zu den Aufgabenbereichen von DLR-TS.

„Was wir sicher wissen, ist, dass an den Stellen im Straßennetz, wo Verkehrsprobleme auftreten, auch überdurchschnittlich viele FCD-Positionen zu finden sind“ (zit. Dr. Peter Wagner, Wissenschaftler im DLR, [22])

Um die Verkehrslage in einem Straßennetz besser einschätzen zu können, ist es nötig, die Probleme erkennen zu können. Probleme in Straßennetzen tauchen vor allem dort auf, wo mehrere Straßen aufeinander treffen oder sich Straßenverläufe verändern. Das bedeutet, dass vor allem innerstädtische Kreuzungen, aber auch Fußgängerampeln oder Spurverengungen Punkte sind, an denen Staus auftreten können. Diese Punkte lassen sich unter dem Begriff „Knotenpunkte“ zusammenfassen.

Eine Beobachtung, die in der Forschung mit FCD gemacht wurde, ist, dass sich Fahrzeugpositionen immer an bestimmten Stellen zu häufen scheinen. So liegt es nahe, dass sich solche Häufungen für Analysezwecke nutzen lassen können. Dies ist die Basis für diese Masterarbeit. In Zusammenarbeit mit dem DLR soll eine Untersuchung erfolgen, ob es möglich ist, mit einfachen Mitteln herauszubekommen, an welchen Stellen in einem Verkehrsnetz Probleme auftauchen. Dies geschieht unter der Betrachtung von Fahrzeugpositionen, deren geographischer Häufung und ihrer Zuordnung zu Knotenpunkten in einem Straßennetz.

Im Voraus ist nicht bekannt, mit welcher Methodik solch eine Analyse erfolgen kann, und welche Ergebnisse erzielt werden können. Ist es repräsentativ, Fahrzeugpositionen aus den FCD in Knotenpunktnähe zu zählen und daraus auf die Situation an diesem Punkt zu schließen? Wie kann „Knotenpunktnähe“ definiert und bestimmt werden? Was sagen die Geschwindigkeiten der Fahrzeuge aus, die sie zu dem Zeitpunkt hatten, als sie die Position sendeten?

Ein Weg, um diese und andere Fragen zu klären, ist die Erstellung einer modularen Software, mit deren Hilfe mehrere verschiedene Wege getestet werden können, die eine Auswertung von „Floating Car Data“ im Hinblick auf gestaute Verkehrsmuster erlauben. Modularität bedeutet, in diesem Fall, dass der Teil der Auswertung austauschbar ist. Dies dient dazu, mehrere Wege der Analyse zu betrachten, auszuwerten und zu vergleichen, um darauf schließen zu können, ob und wie gut sich eine Art der Analyse für diese Aufgabe eignet.

Eine Anforderung für die Software ist eine Visualisierung der Auswertungsergebnisse. Ein Teil der Arbeit, der mindestens die Hälfte des praktischen Anteils ausmacht, beschäftigt sich somit mit den Grundvoraussetzungen für die Visualisierung von Straßennetzen. Um eine

Verkehrslage geeignet visualisieren zu können, benötigt man eine wiederverwendbare Softwarekomponente, welche Straßenkarten darstellen kann. Am geschicktesten ist dies mit Vektorkarten zu ermöglichen, da diese weniger Speicher benötigen und genauer sind als Rasterkarten. Zudem lassen sich Knotenpunkte in Straßennetzen mit Hilfe von Vektorkarten leicht finden und identifizieren.

Die Benutzung von Vektorkarten zur Darstellung wiederum setzt voraus, dass Kartenmaterial in geeigneter Form vorliegt.

Der praktische Anteil dieser Arbeit stellt sich also durch 2 Teile dar, deren Umfang und Aufwand ähnlich ist: Zum einen ist es die Kompression und Verarbeitung von Kartenmaterial, sowie die Erstellung eines Softwaremoduls zum Darstellen der Karten. Zum anderen wird die Software zur Bearbeitung der Problemstellung erstellt, die auf die Darstellungskomponente zurückgreift.

Über dieses Dokument

Die Masterarbeit zum Thema „Identifikation und Visualisierung von Knotenpunkten in Straßennetzen mit gestauten Verkehrsmustern“ ist sehr stark geprägt von einem hohen praktischen Anteil, der sich in der Konzeption, Umsetzung und Programmierung der entsprechenden Softwaremodule wiederfindet. Zur Unterstützung der praktischen Arbeit ist zusätzlich jedoch auch einiges an Hintergrundwissen erforderlich. Im Aufbau und Inhalt dieses Dokumentes spiegelt sich der Anteil der einzelnen Bearbeitungsschritte wieder.

Der erste Teil dieses Dokumentes setzt sich mit den relevanten Grundlagen auseinander. Was sind eigentlich Knotenpunkte in Straßennetzen und wie könnten sich gestaute Verkehrsmuster darauf beziehen, wie ist Kartenmaterial aufgebaut, was sind „Floating Car Data“ im Sinne der Anwendung im DLR und was sind Grundlagen für Identifikation und Bemessung von Verkehrsknoten? Diese Fragen werden im theoretischen Teil dieses Dokuments erläutert.

Eine Beschreibung dessen, was an Konzeption und Umsetzung hinsichtlich der Softwarekomponenten erfolgt ist, folgt als nächster Teil im Dokument und ist der Kern dieser Arbeit. Hier findet sich eine Beschreibung zu Anforderungen, Konzepten, Ideen und zur grundsätzlichen Umsetzung des Problems in die Software. Es wird auch geklärt, wie die einzelnen Analysemethoden zur Aufgabenstellung entwickelt wurden, beziehungsweise wie sie funktionieren. Dieser Teil gibt Aufschluss über die Art und das Ergebnis der praktischen Arbeit.

Der dritte Teil ist als eine Dokumentation des Quellcodes, der Verarbeitung der Software so wie eine Beschreibung von essenziellen Algorithmen, die in der Software Anwendung finden, zu sehen. Dieser Teil ist vor allem interessant für die Aspekte der Informatik in dieser Arbeit. Aufgrund des hohen Umfangs der Software ist nicht jede Zeile des Quelltextes hier im Detail erläutert, es wird aber ein Überblick über Aufbau, Abläufe und Funktionsweise gegeben. Mehr Details lassen sich im Zusammenhang mit dem Quellcode selbst ermitteln, der zu

diesem Zweck ausführlich kommentiert wurde.

An letzter Stelle folgt eine Übersicht über das Ergebnis, und welche Zukunftsaussichten die Arbeit bietet.

2 Aufgabenstellung

Die Aufgabe, die dieser Arbeit zugrunde liegt, ist es, eine Software zu erstellen, mit deren Hilfe gestaute Verkehrsmuster in Straßennetzen mit Hilfe von „Floating Car Data“ identifiziert und visualisiert werden können.

2.1 Aufgabe

Die Aufgabe gliedert sich in mehrere Teilbereiche. Dabei sind jeweils Teile der Aufgabe unabhängig von anderen Teilen, dienen aber als Grundlage für diese.

Zunächst sind einige theoretische Betrachtungen erforderlich. Neben Grundlagen zur Programmierung und mathematischen Grundlagen sind dies die Analyse von NAVTEQ-Kartenmaterial und eine theoretische Betrachtung von möglichen Knotenpunkten, die sich in realen Straßennetzen finden. Das Kartenmaterial soll so weit betrachtet werden, dass ein Verständnis über Aufbau und Inhalt der digitalen Karten von NAVTEQ gewährleistet werden kann. Knotenpunkttypen sollen sehr grob kategorisiert werden, um ein Vorwissen zu ermöglichen, welche Knotenpunkte in Straßennetzen sich im Ergebnis wiederfinden könnten.

Der weit größere praktische Anteil beinhaltet zunächst die Erstellung einer Java-Schnittstelle zur Visualisierung von digitalem Kartenmaterial. Diese Schnittstelle soll einigen Bedingungen entsprechen. Sie soll modular aufgebaut sein, so dass sich Lademodul und Darstellung trennen lassen. Sie soll so effizient sein, dass sie auch auf leistungsschwächeren Geräten wie PDAs lauffähig ist. Zudem soll das Darstellungsmodul in der Hinsicht wiederverwendbar sein, dass es nach außen hin Methoden zur Verfügung stellt, die, möglichst allgemein, Funktionalität für viele verschiedene Visualisierungsaufgaben realisieren.

Das digitale Kartenmaterial soll den Bedürfnissen einer Visualisierung angepasst und so komprimiert werden, dass es zum einen auf Datenträgern mit geringem Datenvolumen (beispielsweise SD-Karten mit 1 Gigabyte Speicher) abgelegt werden und zum anderen schnell und gezielt geladen werden kann.

Die Kartenladeschnittstelle soll vom Rest der Visualisierung gekapselt sein. Sie soll einen Prototyp für eine allgemein verwendbares Modul darstellen, von dem bestimmte Teile austauschbar sind. Das Ziel diese Austauschbarkeit ist es, eine Erweiterung einzuplanen, mit der verschieden komprimierte Karten über verschiedene Wege (beispielsweise auf der selben Maschine, über ein Netzwerk,...) geladen werden können. Implementiert werden soll dabei nur der Teil, der für die Bewältigung der eigentlichen Aufgabe notwendig ist.

Die Floating Car Data, die dem DLR zur Verfügung stehen, sollen analysiert werden. Für ein Beispielgebiet sollen die Daten ausgewertet werden, so dass sich mit ihrer Hilfe Verkehrsknotenpunkte mit gestauten Verkehrsmustern herausfinden lassen. Die Daten sollen aus der Datenbank geladen und verarbeitet werden. Es soll ein Algorithmus gefunden werden, mit dem sich diese Aufgabe möglichst gut bewältigen lässt. Dieser soll unter

Einbeziehung von Knotenpunkten, nicht jedoch mittels Betrachtung von Straßenkanten, arbeiten und möglichst einfach funktionieren. „Knotenpunkte“ sind hier zunächst die Bezeichnung für Knoten in digitalen Straßennetzen allgemein. „Gestaute Verkehrsmuster“ in dem Sinne sind eine Bezeichnung für Ansammlungen von Positionen aus den Daten, die sich an bestimmten geographischen Orten häufen. Es soll jedoch auch ein möglichst guter Bezug zur realen Verkehrssituation hergestellt werden. So soll das Ergebnis möglichst so sein, dass sich aus gestauten Verkehrsmustern Knoten mit realen Verkehrsproblemen ergeben.

Die identifizierten Knotenpunkte sollen mit Hilfe des Kartendarstellungsmoduls in einer Software sichtbar gemacht werden.

2.2 Eingrenzung des Themas und der Forschungsarbeit

Eine Betrachtung des Themas zeigt, dass eine Eingrenzung bezüglich der Gesamtproblematik der Verkehrsbeobachtung notwendig ist. An dieser Stelle wird die Aufgabenstellung „Identifikation von Knotenpunkten in Straßennetzen mit gestauten Verkehrsmustern“ daher genau abgegrenzt.

Probleme in Straßennetzen, die den Verkehrsablauf beeinträchtigen, sind vielseitig und haben verschiedene Ursachen. Staus an sich sind ein Problem, das aus mehreren verschiedenen Gründen auftreten kann. Im innerstädtischen Bereich treten Staus oft zur Berufsverkehrszeit auf, also Vormittags, wenn viele Pendler zur Arbeit fahren, und Nachmittags, wenn sie zum Wohnort zurückkehren. Sind die Schaltungen von Lichtsignalanlagen an Kreuzungen schlecht auf die Verkehrssituation eingestellt, kann es dadurch zu langen Staus in der Innenstadt kommen. Auch temporäre Probleme wie Baustellen, Verkehrsunfälle oder Straßensperrungen treten auf und können Stau verursachen. Außerorts, beispielsweise auf Autobahnen, sind in der Regel die Ursachen für Stau eher temporärer Natur, wie Baustellen und Unfälle. Außer dem Stau können auch andere Störungen in der Verkehrsinfrastruktur auftauchen, wie zum Beispiel Straßenschäden.

Prinzipiell gilt für die Aufgabenstellung dieser Arbeit, dass Stau das Verkehrsproblem ist, das analysiert werden soll. Hierbei werden allerdings nur die Auswirkungen betrachtet, nicht jedoch die Ursachen. Für eine Vollständige Betrachtung müsste zunächst differenziert werden, welcher Stau an welcher Stelle aus welchem Grund auftritt. Ziel dieser Arbeit ist es jedoch, Stellen, an denen Stau auftritt, zu identifizieren, ohne eine Betrachtung der Gründe.

Für die Betrachtung der Gesamtproblematik müsste zudem differenziert werden, auf welchen Straßentypen Probleme auftreten. Landstraßen, Autobahnen oder Straßen in der Stadt sind grundsätzlich verschieden, was ihre Verkehrssituation betrifft. Im Rahmen dieser Arbeit wird jedoch nicht differenziert, um welchen Straßentyp es sich handelt. Der Grund dafür liegt in der Betrachtung des Gebietes, das exemplarisch bearbeitet wird. Nürnberg ist ein Stadtgebiet, somit dominieren hier die Probleme, die innerstädtisch auftreten. Diese sind als

so kritisch zu bewerten, dass sie in der Auswertung dominant sind und Probleme, die auf nahen Autobahnen oder Landstraßen auftreten, nicht ins Gewicht fallen. Die in dieser Arbeit entwickelten Methoden beziehen sich außerdem allgemein auf die Analyse der Verkehrssituation in Stadtgebieten.

Differenziert werden kann bei der Betrachtung einer Auswertung basierend auf FCD auch im Hinblick auf die Art der Daten. Es gibt die Möglichkeit, Daten zu fusionieren, das bedeutet, für die Verkehrslageerfassung auch andere Datenquellen mit einzubeziehen. Beispiele sind Zählschleifen- oder Verkehrskameradaten. Im Rahmen dieser Arbeit werden lediglich die Daten betrachtet, die als reine „Floating Car Data“ vorliegen. FCD können aus verschiedenen Quellen stammen. Für die Forschungsarbeit im DLR gibt es eine Kooperation mit Taxiunternehmen, die FCD zur Verfügung stellen. Andere Fahrzeugflotten oder mögliche Quellen werden nicht berücksichtigt. Die Daten sind eventuell anfällig für Fehler. Daher können die Daten vor verarbeitet werden, um mögliche Fehlerquellen wie fehlende Positionen zu minimieren. In der Anwendung in diesem Beispiel jedoch werden die Daten nicht aus einer Datenquelle gewonnen, die vor verarbeitete Daten liefert. Die Basis für die Auswertung bilden Daten, die direkt aus der jeweiligen Taxizentrale an das DLR gesandt werden. Da Postionsdaten von Taxis auch Unterschiede aufweisen können, wie zum Beispiel die Häufigkeit, mit der Positionen gesendet werden, wird diese Arbeit an einem konkreten Beispiel umgesetzt. Dieses sind die Daten aus der Nürnberger Taxizentrale.

Es ist jedoch nicht auszuschließen, dass das Ergebnis dieser Arbeit mit Daten aus alternativen Datenquellen funktioniert, bzw. mit einem geringen Aufwand umgearbeitet werden kann, so dass sich alternative Datenquellen nutzen lassen.

In einem persönlichen Gespräch mit Mitarbeitern der Verkehrsmanagementzentrale (VMZ) in Berlin Tempelhof zeigte sich, dass vor allem die Anwendung einer Verkehrsanalyse bezüglich der Einordnung der Problemstellung eine große Rolle spielt [30]. Das bedeutet, dass im Voraus zu bestimmen ist, aus welchem Grund die Ergebnisse zu ermitteln sind und welcher Art die Ergebnisse sein sollen, um festzulegen, welche Wege bei der Analyse eingeschlagen werden können.

Eine Kategorisierung diesbezüglich ist Unterteilung in „Live-Systeme“ und die Auswertung historischer Daten. Soll eine Auswertung der Verkehrssituation so erfolgen, dass Probleme zeitnah erkannt werden und auf diese Probleme reagiert werden kann, spricht man von einem „Live-System“. Hierbei werden nur aktuelle Daten ausgewertet, dies geschieht regelmäßig. Ergebnisse müssen hierbei zum nächstmöglichen Zeitpunkt abrufbar sein. Anwendungsgebiet könnte beispielsweise die Erfassung der Verkehrslage für eine gezielte Verkehrslenkung sein. Maßnahmen zur Verkehrslenkung könnten hier die Schaltung eines Verkehrsleitsystems oder die gezielte Taktung von Lichtsignalanlagen sein. Einen anderen Grund für das Betreiben eines „Live-Systems“ stellt die Anwendung aktueller Verkehrsdaten in Navigationssystemen dar. Es ist mit Hilfe von FCD möglich, die Routenplanung so zu beeinflussen, dass der schnellste Weg zu einem Ziel aus der aktuellen Verkehrslage

berechnet wird.

Gründe für die Betrachtung und Auswertung von historischen Daten unterscheiden sich von der Anwendung eines „Live-Systems“. Werden Daten aus der Vergangenheit betrachtet, können andere Problemstellungen bearbeitet werden. So kann man aus der Betrachtung der Verkehrssituation über einen langen Zeitraum, der in nicht allzu weit entfernter Vergangenheit liegt, schließen, an welchen Stellen im Verkehrsnetz immer wieder Probleme auftreten. Eine Kenntnis über immer wieder kehrende Probleme kann dazu führen, dass gezielte Maßnahmen wie die Planung neuer Straßen, die Optimierung von Lichtsignalanlagen oder das Ändern von Verkehrsführungen eingeleitet werden können. Auch ist es denkbar, anhand historischer Daten die Verteilung von TMC-Punkten zu optimieren beziehungsweise für Gebiete ohne festgelegte TMC-Punkte solche neu zu erstellen.

Die Bearbeitung dieser Aufgabe erfolgt mit Hilfe historischer Daten. Ziel ist es, Knotenpunkte mit regelmäßig auftretenden Verkehrsproblemen zu ermitteln und zu visualisieren. Dem kommt der Umstand zu Gute, dass bei einem großen Auswertungszeitraum die hohe Menge der Eingangsdaten sich positiv auf das Ergebnis auswirkt. Eine Anwendung in einem „Live-System“ ist nicht vorgesehen. Die Herangehensweise an die Aufgabe wie auch die Art der Auswertungsalgorithmen sind dafür eher ungeeignet.

Wichtig für die Anwendung einer solchen Analyse ist auch die Art des Ergebnisses, das die Analyse liefert. In vielen Anwendungsbereichen sind dabei bestimmte Teilergebnisse besonders wichtig. Zunächst kann ein Knotenpunkt in einem Straßennetz verschiedene Dinge darstellen. Es gibt beispielsweise Knotenpunkte die Kreuzungen symbolisieren, Teile von Kreuzungen oder Übergänge von einem Straßentyp in einen Anderen. Wichtig kann bei der Auswertung sein, um was für eine Art von Knotenpunkt es sich handelt. Handelt es sich um eine Kreuzung, muss für einige Anwendungsgebiete differenziert werden:

- Tritt das Problem im Zufluss oder im Abfluss der Kreuzung auf?
- Welche Abbiegerichtungen sind betroffen?
- Ist dies eine Kreuzung mit Lichtsignalanlage oder eine Kreuzung ohne LSA?

Dies ist hilfreich bei der Verkehrslenkung oder der gezielten Bewertung eines Verkehrsproblems an einer Kreuzung. Die Aufgabenstellung dieser Arbeit ist es jedoch, lediglich Knotenpunkte mit gestauten Verkehrsmustern zu ermitteln und zu bewerten, ob Probleme auftreten bzw. wie stark Probleme vorhanden sind. Dies spiegelt sich in einer Bewertung eines Knotens von „unkritisch“ bis „kritisch“ wieder, wobei diese Bewertung keinen Aufschluss über die genaue Art des Problems liefert.

Ein generelles Problem, welches in dieser Arbeit nicht behandelt wird, ist der Rückstau über mehrere Knotenpunkte. Eine Darstellung des Problems zeigt folgende Skizze:

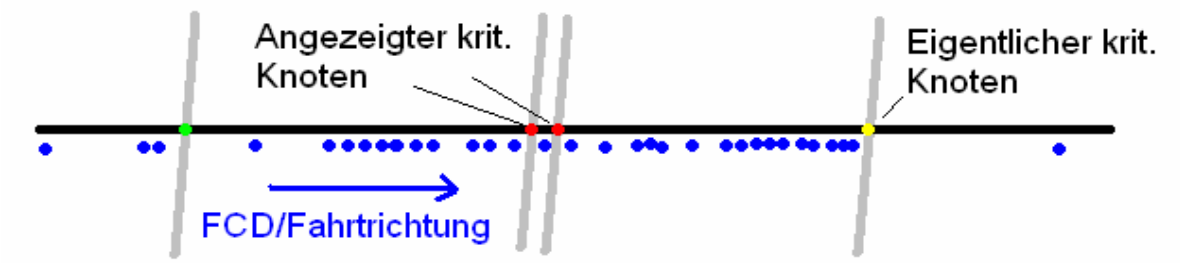

Abbildung 1: Skizzierung des Rückstauproblems bei Nichtbetrachtung von Straßenkanten

Dargestellt ist hier skizzenhaft ein Ausschnitt eines Straßennetzes. Grün, gelb und rot eingefärbt sind Knotenpunkte, die als „leicht kritisch“, „mittelmäßig kritisch“ und „sehr kritisch“ bezeichnet werden können. Blau eingezeichnet sind festgestellte Fahrzeugpositionen, die Fahrtrichtung soll generell Richtung Osten sein. Der östliche Knoten ist der Knoten, der Stau verursacht. Dem mittleren Knoten, der ein gestautes Verkehrsmuster aufweist, ohne Ursache für den Stau zu sein, sind jedoch mehr Treffer zugeordnet. Damit zeigt er ein kritischeres Muster als der eigentlich verursachende Knoten. Ohne die Betrachtung von Straßenkanten kann jedoch nicht bestimmt werden, ob 2 Knoten verbunden sind oder nicht. Somit muss gesagt werden, dass zwar Knoten, die gestaute Verkehrsmuster aufweisen, identifiziert werden können, diese jedoch nicht mit Verursachern für gestaute Verkehrsmuster gleichzusetzen sind.

3 Grundlagen

3.1 Straßennetze und kritische Knotenpunkte

Bei der Identifizierung und Visualisierung von Knotenpunkten mit gestauten Verkehrsmustern muss zunächst betrachtet werden, was Knotenpunkte eigentlich sind. Im Rahmen dieser Arbeit kann das Wort „Knotenpunkt“ zwei verschiedene Bedeutungen haben.

Im Straßen- und Verkehrswesen ist ein Knotenpunkt ein Ort, an dem mehrere Straßen aufeinander treffen, also eine Kreuzung oder eine Einmündung. Solche Knotenpunkte können eine der 12 folgenden Grundformen haben [2]:

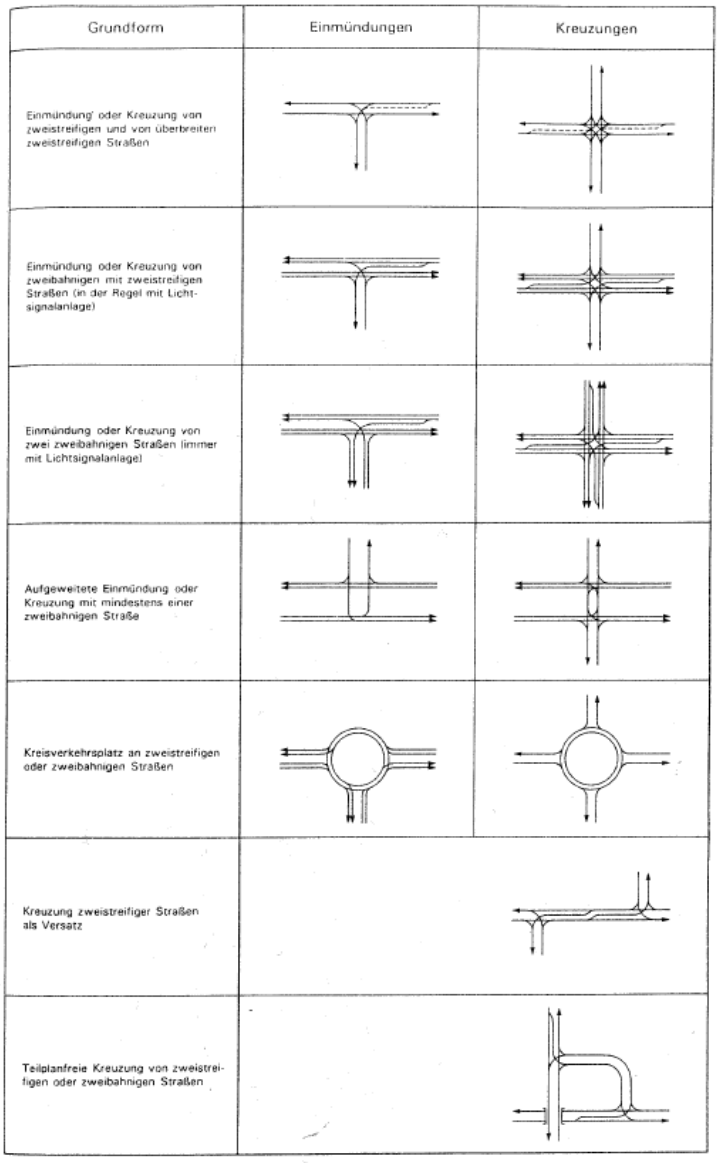

Abbildung 2: Grundformen von Strassenverkehrsknotentypen [2]

Neben diesen Grundformen sind auch leicht abweichende Ausprägungen möglich. Die geographische Ausdehnung solcher Kreuzungen und Einmündungen kann stark differieren. Abhängig ist die Ausdehnung vom Knotentyp, der Anzahl der Spuren, den Straßentypen der im Knoten mündenden Straßen und der Fahrbahnbreite. Die Ausdehnung kann relevant sein, wenn versucht wird, FCD-Positionen durch die Festlegung eines bestimmten Umkreises um einen Knoten diesem zuzuordnen.

Laut EAHV [2] ist die empfohlene Standardfahrstreifenbreite innerorts von der Flächenverfügbarkeit und des Anteils des vorgesehenen Linienbus- und Schwerlastverkehrs abhängig und beträgt zwischen 2,75 und 3,50 Meter. Für eine optisch gegliederte zweistreifige Fahrbahn (eine Straße mit 2 Richtungen, jeweils einer Spur und ohne bauliche Trennung) ist inklusive der Seitenstreifen eine Breite von 6,50 bis 8,50 Metern empfohlen. Einbahnstraßen sind in der Regel 3,50 Meter breit. Vierstreifige Fahrbahnen können, abhängig von der Breite des Mittelstreifens, eine Breite von mehr als 15 Metern aufweisen. Der maximale Durchmesser der Kreuzung ist also größer als 20 Meter. Größer noch ist die Ausdehnung von Kreisverkehrsplätzen. Im EAHV sind diese mit mit einem Außendurchmesser von 25 bis 40 Metern als „kleine Kreisverkehrsplätze“ kategorisiert, „große Kreisverkehrsplätze“ können einen noch größeren Radius aufweisen.

Für eine genaue Auswertung ist es zudem zu beachten, dass der Verkehr an Knotenpunkten in Straßennetzen entweder über eine Lichtsignalanlage, durch Verkehrsschilder oder die „Rechts vor Links“-Regel geregelt sein kann. Unterschieden wird generell nur in Knotenpunkte mit Lichtsignalanlage oder Knotenpunkte ohne Lichtsignalanlage. Für diese Knotenpunkte gibt es verschiedene Bewertungskriterien. Bei der Suche nach gestauten Verkehrsmustern kann jedoch auch generell von lichtsignalgesteuerten Knotenpunkten ausgegangen werden, da die mittleren Wartezeiten an diesen im Regelfall generell größer sind [3].

In der Praxis sind es unter Umständen diese Knotenpunkte, deren Auslastung bemessen werden soll. Dies würde bedeuten, dass Kreuzungen und Einmündungen gesucht werden müssen, die gestaute Verkehrsmuster aufweisen. Die Auswertung im Rahmen dieser Arbeit bezieht sich aber auf „Knotenpunkte“ in einem anderen Sinne.

Knotenpunkte sind, im graphentheoretischen Kontext, auch geographische Punkte im Kartenmaterial. Sie enthalten Informationen über ihre geographische Lage, also ihren genauen Längen- und Breitengrad, und gegebenenfalls zusätzlich Informationen über ein- und ausgehende Straßenkanten, ihre Bedeutung im Straßennetz, die Ausstattung mit einer Lichtsignalanlage oder Ähnliches. Im Kartenmaterial haben Knotenpunkte die Aufgabe, Straßenkanten zu verbinden. Dies kann zum Beispiel im Falle der Repräsentation einer Kreuzung geschehen. Treffen zwei Straßen ohne bauliche Trennung an einer Kreuzug aufeinander, kann diese Kreuzung durchaus durch einen einzelnen Knotenpunkt im Graphen repräsentiert werden. In diesem Fall sind der Knotenpunkt im straßenverkehrstechnischen

Sinn sowie im Kartenmaterial identisch. Beispiele, bei denen es Unterschiede zwischen Knotenpunkten im Kartenmaterial und im Verkehrsnetz gibt, sind Kreuzungen oder Einmündungen von Straßen mit baulicher Trennung.

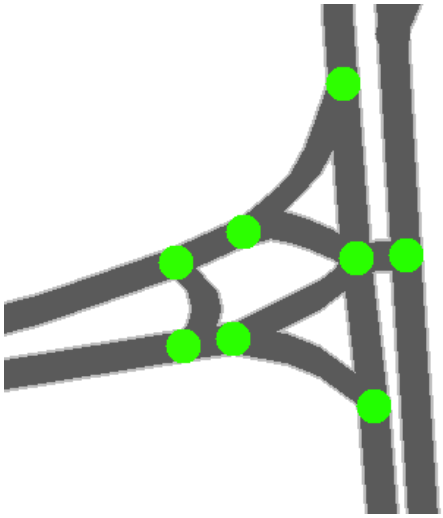

Abbildung 3: Beispiel für Knotenpunkte in einer Straßeneinmündung

In dieser Grafik ist zu sehen, dass eine Einmündung oder Kreuzung aus mehreren Knoten bestehen kann. In diesem Fall wird eine Einmündung durch 6 Knoten im Kartenmaterial (grün) repräsentiert. Da die zuführende Straße aus Richtung Westen eine Wendefahrbahn vor der Einmündung enthält, die zur Einmündung gehört, sind es sogar 8 Knoten im Kartenmaterial. Dies ist notwendig, um die bauliche Eigenart der Einmündung korrekt abbilden zu können. Die verschiedenen Fahrtrichtungen einer Straße sind durch einen Mittelstreifen getrennt und es gibt baulich getrennte Abbiegespuren für die aus Richtung Westen zuführende Straße.

Knotenpunkte im Kartenmaterial können jedoch auch andere Funktionen als die Repräsentation von Kreuzungen und Einmündungen haben. Sie können auch zwei Straßenkanten verbinden, die zur gleichen Straße gehören. Grund dafür könnten eine eine Fußgängerlichtsignalanlage oder eine Änderung der Kantenattribute sein (wenn beispielsweise auf einem Stück einer Straße eine niedrigere Geschwindigkeit erlaubt ist als auf dem Rest der Straße).

In dieser Arbeit, so nicht explizit anders erwähnt, ist mit der Bezeichnung „Knotenpunkt“ ein geographischer Punkt im Kartenmaterial gemeint.

Die Betrachtung solcher Knotenpunkte macht auch in Bezug auf die Aufgabenstellung Sinn. So kann es bei Kreuzungen, die im Kartenmaterial aus mehreren Knotenpunkten bestehen, auch bei einzelnen Knoten zu gestauten Verkehrsmustern kommen, wenn beispielsweise nur eine einzelne zuführende Straßenkante oder eine einzelne Abbiegerichtung von einem

Verkehrsproblem betroffen ist. Auch Fußgängerampeln können zur Entstehung gestauter Verkehrsmustern beitragen, beziehungsweise können sich solche Muster in deren Nähe befinden.

3.2 Grundlagen für die Bemessung von Verkehrsknoten

Eine wichtige Grundlage für die Entscheidung, ob Verkehrsknotenpunkte kritisch ausgelastet sind, ist das Finden eines Maßes zur Bewertung von Verkehrsknoten. Einen Ansatz dazu liefert das „Handbuch für die Bemessung von Straßenverkersanlagen“ (HBS) [3]. Das Handbuch liefert Empfehlungen zur Bewertung vieler möglicher Größen für verschiedene Arten von Streckenabschnitten und Verkehrsknotenpunkten (hier: Kreuzungen und Einmündungen).

Relevant sind hier die Qualitätsstufen des Verkehrsablaufs im Bezug auf Verkehrsknoten mit und ohne Lichtsignalanlage. Bei Knotenpunkten ohne Lichtsignalanlage ergeben sich die Qualitätsstufen aus folgender Tabelle:

Grenzwerte der mittleren Wartezeit für die Qualitätsstufen

QSV	Mittlere Wartezeit w [s]
A	≤ 10
B	≤ 20
C	≤ 30
D	≤ 45
E	> 45
F	-[1)]

1) Die Stufe F ist erreicht, wenn der Sättigungsgrad größer als 1 ist.

Abbildung 4: Qualitätsstufen des Verkehrsablaufs bei Kontenpunkten ohne Lichtsignalanlage [3]

Der Sättigungsgrad, wichtig für Stufe F, bedeutet, dass die Auslastung der Straße höher ist als ihre Kapazität.

$$Sättigungsgrad = \frac{Verkehrsstärke}{Kapazität}$$

Die Qualitätsstufen haben folgende Bedeutung:

Stufe A	Die Mehrzahl der Verkehrsteilnehmer kann nahezu ungehindert den Knotenpunkt passieren. Die Wartezeiten sind sehr gering.
Stufe B	Die Fahrmöglichkeiten der wartepflichtigen Kraftfahrzeugströme werden vom bevorrechtigten Verkehr beeinflusst. Die dabei entstehenden Wartezeiten sind gering.
Stufe C	Die Fahrzeugführer in den Nebenströmen müssen auf eine merkbare Anzahl von bevorrechtigten Verkehrsteilnehmern achten. Die Wartezeiten sind spürbar. Es kommt zur Bildung von Stau, der jedoch weder hinsichtlich seiner räumlichen Ausdehnung noch bezüglich der zeitlichen Dauer eine starke Beeinträchtigung darstellt.
Stufe D	Die Mehrzahl der Fahrzeugführer muss Haltevorgänge, verbunden mit deutlichen Zeitverlusten, hinnehmen. Für einzelne Fahrzeuge können die Wartezeiten hohe Werte annehmen. Auch wenn sich vorübergehend ein merklicher Stau in einem Nebenstrom ergeben hat, bildet sich dieser wieder zurück. Der Verkehrszustand ist noch stabil.
Stufe E	Es bilden sich Staus, die sich bei der vorhandenen Belastung nicht mehr abbauen. Die Wartezeiten nehmen sehr große und dabei stark streuende Werte an. Geringfügige Verschlechterungen der Einflussgrößen können zum Verkehrszusammenbruch führen. Die Kapazität wird erreicht.
Stufe F	Die Anzahl der Fahrzeuge, die in einem Verkehrsstrom dem Knotenpunkt je Zeiteinheit zufließen, ist über ein längeres Zeitintervall größer als die Kapazität für diesen Verkehrsstrom. Es bilden sich lange, ständig wachsende Schlangen mit besonders hohen Wartezeiten. Diese Situation löst sich erst nach einer deutlichen Abnahme der Verkehrsstärken im zufließenden Verkehr wieder auf. Der Knotenpunkt ist überlastet.

Tabelle 1: Bedeutung der Qualitätsstufen der Verkehrsablaufs bei Knoten ohne Lichtsignalanlage [3]

Ausschlaggebend für die Bewertung ist hierbei die mittlere Wartezeit, die ein Fahrzeug verbringt, bevor es den Knotenpunkt passieren kann. Aus praktischen Gründen wird in der Regel der Mittelwert der entstandenen bzw. gemessenen Wartezeiten betrachtet. Laut HBS ist hierbei zu beachten, dass aufgrund unterschiedlicher Rangordnung und Auslastung jede zuführende Straße einzeln betrachtet werden muss. Im Bezug auf diese Arbeit ist dies aufgrund der unterschiedlichen Ausprägungen von Kreuzungen und der damit verbundenen differierenden Anzahl von Knotenpunkten im Kartenmaterial jedoch nur eingeschränkt möglich.

Generell lässt sich aussagen, dass ein Knotenpunkt dann eine kritische Qualität des Verkehrsablaufs hat, wenn die mittlere Wartezeit höher ist als 30 Sekunden (Stufe D). Hier ist von deutlichen Zeitverlusten die Rede, wenn auch von einem noch stabilen Zustand.

Von dieser Kategorisierung unterscheidet sich die Bemessung der Qualitätsstufen des Verkehrsablaufes für Knotenpunkte mit Lichtsignalanlage. Diese sind folgender Tabelle zu entnehmen:

QSV	Zulässige mittlere Wartezeit w [s]				Prozentsatz der Durchfahrten ohne Halt [%]
	straßen-gebundener ÖPNV	Fahrrad-verkehr	Fußgänger-verkehr[1)]	Kraftfahrzeug-verkehr (nicht koordinierte Zufahrten)	Kraftfahrzeug-verkehr (koordinierte Zufahrten)
A	≤ 5	≤ 15	≤ 15	≤ 20	≤ 95
B	≤ 15	≤ 25	≤ 20	≤ 35	≤ 85
C	≤ 25	≤ 35	≤ 25	≤ 50	≤ 75
D	≤ 40	≤ 45	≤ 30	≤ 70	≤ 65
E	≤ 60	≤ 60	≤ 35	≤ 100	≤ 50*
F	> 60	> 60	> 35	> 100	< 50*

1) Zuschlag von 5 s bei Überquerung von mehreren Furten
* Koordinierung unwirksam

Abbildung 5: Qualitätsstufen des Verkehrsablaufs bei Kontenpunkten mit Lichtsignalanlage [3]

Für die Bewertung ausschlaggebend sind wiederum die mittleren Wartezeiten. Im Regelfall sind Taxis, welche im Rahmen dieser Arbeit die FCD liefern, an die gleichen Wartezeiten gebunden, wie der restliche Kraftfahrzeugverkehr. Nur in Ausnahmefällen, wie zum Beispiel an Kreuzungen mit Busspuren, die eine Mitnutzung durch Taxis erlauben, gelten die Wartezeiten für straßengebundenen ÖPNV. Bei der allgemeinen Betrachtung von Knotenpunkten mit Lichtsignalanlage ist nicht von koordinierten Zufahrten auszugehen. Die Angabe „mittlere Wartezeit“ unterscheidet sich von der Angabe bei Kreuzungen ohne LSA. Sie setzt sich zusammen aus der Grundwartezeit w_I und der Reststauwartezeit w_{II} [3]. die Grundwartezeit ist die mittlere Zeit, die Fahrzeuge aufgrund der Sperrung des Knotenpunktes durch die Lichtsignalanlage warten müssen. Reststauwartezeit ist die Zeit, die durch Fahrzeuge verursacht wird, die den Knotenpunkt nicht während der Freigabezeit des Knotens (Grünphase) passieren können, sondern mehrere Schaltumläufe benötigen. Die Wartezeit ergibt sich aus $w = w_I + w_{II}$.

Die Bedeutung der verschiedenen Qualitätsstufen ist folgender Tabelle zu entnehmen:

Stufe A	Die Mehrzahl der Verkehrsteilnehmer kann ungehindert den Knotenpunkt passieren. Die Wartezeiten sind kurz.
Stufe B	Alle während der Sperrzeit ankommenden Verkehrsteilnehmer können in der nachfolgenden Freigabezeit weiterfahren oder –gehen. Die Wartezeiten sind kurz.
Stufe C	Nahezu alle während der Sperrzeit ankommenden Verkehrsteilnehmer können in der nachfolgenden Freigabezeit weiterfahren oder –gehen. Die Wartezeiten sind spürbar. Beim Kraftfahrzeugverkehr tritt im Mittel nur geringer Stau am Ende der Freigabezeit auf.
Stufe D	Im Kraftfahrzeugverkehr ist ständiger Reststau vorhanden. Die Wartezeiten für alle Verkehrsteilnehmer sind beträchtlich. Der Verkehrszustand ist noch stabil.
Stufe E	Die Verkehrsteilnehmer stehen in erheblicher Konkurrenz zueinander. Im Kraftfahrzeugverkehr stellt sich ein allmählich wachsender Stau ein. Die Wartezeiten sind sehr lang. Die Kapazität wird erreicht.
Stufe F	Die Nachfrage ist größer als die Kapazität. Die Fahrzeuge müssen bis zu ihrer Abfertigung mehrfach vorrücken. Der Stau wächst ständig. Die Wartezeiten sind extrem lang. Die Anlage ist überlastet.

Tabelle 2: Bedeutung der Qualitätsstufen der Verkehrsablaufs bei Knoten mit Lichtsignalanlage [3]

Ähnlich wie bei der Betrachtung von Knotenpunkten ohne LSA ist eine Auslastung der Stufe D oder höher als „kritisch“ zu betrachten. Daraus folgt, dass eine mittlere Wartezeit von mehr als 50 Sekunden einen schlechten Verkehrsablauf bedeutet (Angabe nicht koordinierte Zufahrten bei Kraftfahrzeugverkehr). Der Verkehrszustand wird hier jedoch ebenfalls noch als „stabil“ bezeichnet.

Eine Schwierigkeit ergibt sich aus den Differenzen der Bewertungen beider Knotentypen. Beträgt die mittlere Wartezeit an Knotenpunkten beispielsweise 51 Sekunden, bedeutet das bei Knottenpunkten mit LSA eine Qualität der Stufe D, bei Knoten ohne LSA jedoch schon Stufe E. Wird dieser Unterschied nicht beachtet, muss ein Wert geschätzt werden, der sich im Rahmen dieser Differenz bewegt. In einer Auswertungsmethode in dieser Arbeit werden die Wartezeiten indirekt als Bewertungkriterium herangezogen. Ausgehend von einem Sendeintervall für FCD-Positionen von 30 Sekunden verbringt ein Fahrzeug ca. 31 bis 89 Sekunden an der gleichen Stelle, wenn es 2 Positionen sendet. Im Mittel bedeuten 2 Positionen an einem Knoten also 60 Sekunden Wartezeit. Nach der Einteilung laut HBS ist hier als schon von beeinträchtigtem Verkehrsablauf auszugehen.

3.3 Floating Car Data

3.3.1 Definition und Anwendung im DLR

„Floating Car Data“, oder Kurz FCD, ist ein Begriff aus dem europäischen Raum. Im US-amerikanischen Raum gibt es dafür die Bezeichnung „Probe Vehicle Data“, oder kurz PVD. Übersetzt bedeutet „Floating Car Data“ so viel wie „Daten aus fließendem

Kraftfahrzeugverkehr“ oder „Daten fahrender Autos“. Gemeint sind hier Daten, die geographische Positionen von Fahrzeugen enthalten, zuzüglich bestimmter Eigenschaften dieser Fahrzeuge bezüglich des Verkehrsflusses. FCD sind auf bestimmte Fahrzeuge bezogen, deren Positionen gemessen werden. Zudem enthalten sie einen Zeitstempel, der den Zeitpunkt der Aufnahme einer Position bestimmt. Weitere Informationen wie Momentangeschwindigkeit des Fahrzeuges können in den Daten enthalten sein.

FCD lassen sich durch die Methode der Aufnahme in 2 Kategorien einteilen [4]. Man unterscheidet in passive und aktive FCD. Passive FCD werden durch spezielle Zählschleifen aufgenommen. Diese Zählschleifen werden an bestimmten Punkten in einem Straßennetz aufgestellt und registrieren das Vorbeifahren bestimmter Fahrzeuge. Fahrzeuge, die erfasst werden sollen, müssen über einen Transponder verfügen, mit dessen Hilfe sie identifizierbar sind.

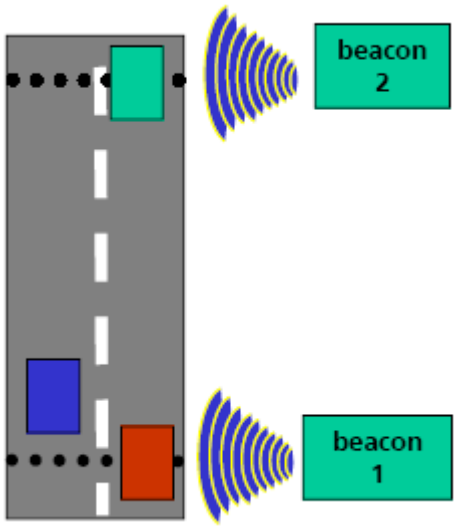

Abbildung 6: Schema der Aufnahme passiver Floating Car Data [4]

Der Vorteil ist, dass die Ausstattung der Fahrzeuge mit einem Transponder sehr kostengünstig ist. Zudem ist diese Methode sehr genau, da Fehler in der Positionsbestimmung entfallen. Das Aufstellen der „Beacons“, also der Zählschleifen, die das Vorbeifahren registrieren, ist jedoch mit einem hohen Kostenaufwand verbunden, der steigt, je besser die erreichte Abdeckung sein soll.

Aktive FCD im Gegensatz dazu benötigen eine Ausstattung der Fahrzeuge, welche kostenintensiver ist. Grundlage ist zunächst eine selbstständige Positionsbestimmung der Fahrzeuge, welche in der Praxis im Regelfall durch satellitengestützte Navigationssysteme Systeme wie GPS erfolgt. Die Positionen, die von den Fahrzeugen gemessen werden, müssen zudem über ein drahtloses Kommunikationsnetzwerk an eine zentrale Stelle gesandt werden, wo sie gespeichert und ausgewertet werden können. Solche Netzwerke können beispielsweise Betriebsfunk oder eine Übertragung per GSM sein. Im Gegensatz zu

passiven FCD entfallen hier die hohen Kosten für die Infrastruktur, lediglich das Kommunikationsnetzwerk und ein Server zum Sammeln der Daten müssen betrieben werden. In der Praxis lassen sich hier Daten von Fahrzeugflotten verwenden, deren Positionen zu Dispositionszwecken ohnehin gesammelt werden. Damit wird das System sehr kostengünstig.

Grundlage dieser Arbeit sind aktive FCD.

Die einfache und kostengünstige Positionsbestimmung für Fahrzeuge ist der Grundstein für das Sammeln von aktiven FCD. Im Zeitraum zwischen 2000 und 2001 wurde nach einem Beschluss der US-amerikanischen Regierung die künstliche Ungenauigkeit des Systems GPS abgeschaltet. Zuvor waren die zivilen Signale des Systems zum Schutz vor Terrorismus künstlich verschlechtert, um beispielsweise eine Steuerung für Langstreckenraketen mit diesem Signal zu verschlechtern. Da GPS nun jedoch eine relativ genaue Positionsbestimmung ermöglicht, ist mit diesem Zeitpunkt die Nutzung von aktiven FCD relevant geworden. (laut Dr. Michael Wößner, [23], liegt die Genauigkeit bei ca. 12 Metern, wobei technisch bedingt unterschiedliche Genauigkeiten möglich sind und in der Literatur unterschiedliche Angaben zur Genauigkeit auftauchen) In den letzten Jahren ist der Markt, der auf der Nutzung von GPS basiert, rapide gewachsen [4]. Viele Unternehmen mit Fahrzeugflotten haben diese mit Systemen ausgestattet, die durch die Positionsbestimmung via GPS und einem Kommunikationsnetzwerk das Erfassen der Positionen ihrer Fahrzeuge und damit ein besseres Flottenmanagement ermöglichen. Dies betrifft beispielsweise Spediteure und Taxiunternehmen.

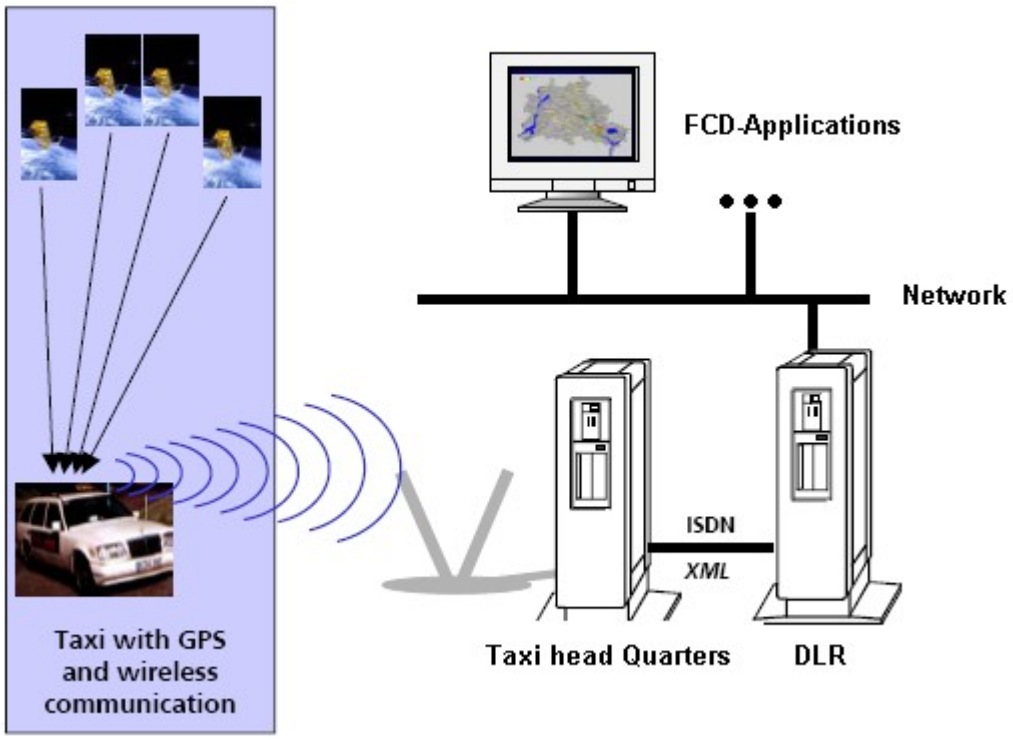

Abbildung 7: Schematische Darstellung der FCD-Prozesskette im DLR [4] (modifiziert)

Durch die Daten, die in Taxiunternehmen gesammelt werden, ergibt sich eine Basis für die Forschung zum Thema FCD zur Verkehrslageerfassung in Großstädten. Im Jahre 2001 startete das DLR in Berlin ein Pilotprojekt mit einem Taxiunternehmen [4]. 300 Taxis lieferten jeden Monat ungefähr 2 Millionen GPS-Positionen. Über eine XML-Schnittstelle werden diese Daten via ISDN an das DLR gesandt und dort zur Weiterverwendung in einer Datenbank gespeichert. Zur Auswertung der Daten und zur Verbesserung der Genauigkeit wird ein Mapmatching-Algorithmus verwendet, der die Positionen Straßenkanten zuordnet. Seither wird intensiv mit diesen Daten geforscht, in Bereichen wie dynamischen Zielführungssystemen und Analyse der Verkehrslage. Mit der Zeit kamen weitere Bereiche wie die Städte Wien, Nürnberg und Hamburg hinzu.

In Nürnberg beträgt die Größe der Taxiflotte ca. 500 Fahrzeuge, deren Positionsdaten zur Verfügung stehen [4]. Mit einer Größe von weniger als einem Viertel der Fläche von Berlin ist die FCD-Abdeckung damit wesentlich dichter als in Berlin, was Nürnberg zu einem guten Testgebiet für FCD-Anwendungen macht. In Nürnberg ist die Zahl der Kraftfahrzeuge, die täglich die Stadtgrenzen passieren, bis 1991 stetig angestiegen. Seit den 90er Jahren sind dies ca. 600.000 Fahrzeuge an jedem Werktag [24][25]. Zum Vergleich: Nürnberg ist eine Stadt mit 500.000 Einwohnern. Dies bringt ein hohes Verkehrsaufkommen mit sich, was wiederum schnell zu Verkehrsbeeinträchtigungen führen kann. Durch diese Umstände wurde Nürnberg als dasjenige Gebiet gewählt, mit dessen Daten diese Arbeit erstellt wurde.

So nicht anders erwähnt, bezieht sich diese Arbeit immer auf das Gebiet von Nürnberg und dessen unmittelbarer Umgebung.

Unabhängig vom Gebiet speichert das DLR die FCD Positionen mit den Angaben ID des Fahrzeuges, Datum und Zeit des Eintrags, Längen- und Breitengrad der Position, Status des Fahrzeuges, Winkel des Fahrzeuges (Fahrtrichtung) und Geschwindigkeit des Fahrzeuges. Je nachdem, welche Daten vorhanden sind, können einzelne Werte der Einträge 0 sein. In Nürnberg ist dies bei der Fahrtrichtung der Fall, da diese nicht angegeben ist. Einige Beispieleinträge aus Nürnberg zeigt folgende Tabelle:

Fahrzeug-ID	Datum/Zeit	Längengrad	Breitengrad	Status	Geschw.	Winkel
154	2008-10-01 00:00:28.0	11.0807	49.4467333333333	70	2	0
474	2008-10-01 00:00:30.0	11.0752833333333	49.4468333333333	65	1	0
498	2008-10-01 00:00:30.0	11.0829333333333	49.4912166666667	70	51	0

Tabelle 3: Beispieldatensätze aus der FCD-Datenbank, Originaldaten

Der Status, den der Wert des Feldes „Status“ dabei repräsentiert, hängt vom Dispositionssystem der Taxizentrale ab. In diesem Fall handelt es sich um ein Flottenmanagementsystem der Firma Microtek [32]. Der Status ist folgender Tabelle zu entnehmen:

Status-Code	Bedeutung
65	Angemeldet (ohne weitere Bedeutung)
70	Frei
83	Wartend am Halteplatz
90	Besetzt mit Fahrziel

Tabelle 4: Übersetzung der Statuscodes in den FCD

3.3.2 Qualität der Daten

In der Theorie lässt sich aus FCD gut auf die Verkehrslage des betrachteten Gebiets schließen [21]. In der Praxis muss jedoch beachtet werden, dass FCD einer ganzen Reihe von Fehlerquellen und Ungenauigkeiten unterliegen. Diese spielen eine wesentliche Rolle in dem Ergebnissen dieser Arbeit.

Die erste mögliche Fehlerquelle ist die Positionsbestimmung mit Hilfe von GPS. Diese Art der Positionsbestimmung unterliegt Fehlerquellen, welche an dieser Stelle nur kurz erwähnt werden.

- Zur Positionsbestimmung sind 4 GPS-Satelliten mit direkter Sichtlinie notwendig. Diese spannen zum Empfänger hin eine Pyramide auf. In Betracht der Winkel lässt sich sagen: Je größer das Pyramidenvolumen, desto höher die Genauigkeit, weil die Winkel besser sind. Sind die Winkel zu klein oder zu groß, kommt es zu einer Verschlechterung der Genauigkeit („dilution of precision“).
- Durch die Troposphäre und die Ionosphäre, Schichten der Erdathmosphäre, kann es zu einer unbekannten Verzögerung in der Signallaufzeit kommen. Dies kann die Positionsbetimmung ebenfalls etwas ungenauer machen.
- Die Erde ist keine ideale Kugel bzw. hat eine sehr unregelmäßige Oberfläche, weshalb die Satellitenbahnen, beeinflusst durch unterschiedlich starke Gravitation, keine idealen Ellipsen sind. Durch Ellipsoiden wird die Erdoberfläche approximiert, dies ist aber nicht zu 100% genau. Auch dies führt zu Ungenauigkeiten.
- Durch Bebauung kann es im Stadtgebiet zu Reflexionen des GPS-Signals an Häuserwänden kommen. Diese führen zu einer Verzögerung der Signallaufzeit, was wiederum zu einer Verringerung der Genauigkeit führt.
- Trotz der hohen angestrebten Genauigkeit haben GPS-Satelliten in der Regel einen geringfügigen Uhrenfehler, der sich auf die Genauigkeit der Positionsbestimmung auswirken kann.

Generell kann gesagt werden, dass die mögliche Ungenauigkeit der Positionsbestimmung mit Hilfe von GPS in etwa 12 Meter beträgt [23].

Eine weitere Fehlerquelle ist das Fahrverhalten der Taxifahrer. Hier muss nun unterschieden werden, welchen Status ein Taxi zum Zeitpunkt der Aufnahme einer Position hat. Ein Taxi kann sich auf einer Leerfahrt befinden. Mit diesem Status fahren Taxifahrer in der Regel langsamer als möglich, um Ausschau nach möglichen Fahrgästen zu halten. Sie geben hier keine FCD-Positionen ab, aus denen auf die reale Verkehrslage geschlossen werden kann. Haben Taxis den Status „wartend am Halteplatz“, stehen sie und liefern überhaupt keine Informationen über die Verkehrslage. Ist ihr Status unbekannt, sind Rückschlüsse auf ihr Fahrverhalten nur schwer zu ziehen, was wiederum eine Bestimmung der Verkehrslage mit Hilfe dieser Daten erschwert. Ausschlaggebend sind einzig und allein die Positionen, die gesendet werden, während das Fahrzeug mit einem Fahrgast besetzt ist und zu einem Zielort unterwegs. Mit diesem Status verhalten sich Taxis in der Regel wie alle anderen Verkehrsteilnehmer, die in einer möglichst kurzen Zeit ihren Zielort erreichen wollen. Eine Einschränkung ist hierbei jedoch das Verhalten am Anfang und am Ende der Fahrt, also während der Fahrgast sein Ziel nennt oder für die Fahrt bezahlt. Hier kann es dazu kommen, dass das Fahrzeug steht, obwohl die Verkehrslage ein flüssiges Fahren zulassen würde. Eine weitere Einschränkung ist, dass die Fahrer, je nach System, gegebenenfalls manuell den Status verändern müssen. Wenn sie dies vergessen, kann hier ein falscher Status gesendet werden.

Bezüglich einer Auswertungsmethode spielt das Sendeintervall der Positionen eine Rolle als weitere Fehlerquelle. In Nürnberg wird im Regelfall alle 30 Sekunden eine Position gesendet. Durch verschiedene Fehler kann es allerdings dazu kommen, dass entweder 2 Positionen direkt nacheinander gesendet werden, oder das Zeitintervall deutlich größer ist. Dies kann zu einer Verfälschung des Ergebnisses führen.

Um die Daten aus Gründen des Datenschutzes zu anonymisieren, werden die Fahrzeug-IDs der Positionsdaten, welche von den Taxizentralen gesendet werden, in bestimmten Abständen gewechselt. Um eine Rückverfolgung zu erschweren, wird nicht bekannt gegeben, in welchen Zeitabständen und nach welchem System IDs vergeben werden. Werden diese IDs ohne Betrachtung dieser Wechsel zu Auswertungszwecken herangezogen, kann auch dies Fehler mit sich bringen.

In einigen Städten, unter Anderem in Nürnberg, sind die Momentangeschwindigkeiten der Fahrzeuge während des Sendens einer Position bekannt. Zu beachten ist jedoch, dass dies nur Momentaufnahmen sind. Wenn also ein Fahrzeug während des Sendens gerade die Geschwindigkeit 0 hat, bedeutet das nicht, dass ein Stau aufgetreten ist. Während des Wartens an einer Lichtsignalanlage beispielsweise steht das Fahrzeug, egal wie viele Zyklen es warten muss. Bei der Betrachtung dieser Geschwindigkeiten muss beachtet werden, dass es sich hierbei nicht um Durchschnittsgeschwindigkeiten handelt. Ein Fahrzeug kann auch 2 Positionen mit der Geschwindigkeit 0 nacheinander senden, die aber so weit auseinander liegen, dass eine bestimmte Wegstrecke zurückgelegt wurde.

Generell ist es schwierig zu bestimmen, in wie weit sich diese Fehlerquellen auf die Ergebnisse auswirken. Die Betrachtung der Geschwindigkeit lässt sich bei bestimmten Auswertungsmethoden vermeiden, während der Fehler durch das Wechseln der Fahrzeugidentifikationsnummern mit der Vergabe neuer, eigener IDs eliminiert werden kann. Die anderen Fehler treten punktuell und unregelmäßig auf, weshalb es schwer bis unmöglich ist, ein Maß für die Fehler zu finden.

Allgemein kann gesagt werden, dass die Fehlerrate sinkt, je mehr Daten betrachtet werden. Dabei wesentlich ist die Anzahl der betrachteten Fahrzeuge [5]. Eine Aussage über die Fehleranfälligkeit der FCD, die nur mittelbar mit den Methoden dieser Arbeit in Verbindung steht, liefert eine Untersuchung des DLR im Jahr 2008. Hier wurde die allgemeine Fragestellung betrachtet, wie viele Fahrzeuge benötigt werden, um gute Ergebnisse bei der Ermittlung von Reisegeschwindigkeiten zu erzielen. Basis dafür sind jedoch, im Gegensatz zu dieser Arbeit, ein Mapmatching-Verfahren und eine Glättung der Tagesganglinien der Geschwindigkeiten. Basis für die Auswertung ist die bestehende FCD-Prozesskette, die im Institut für Verkehrssystemtechnik im DLR die Daten auswertet und nutzbar macht. Von daher kann nur auf eine sehr allgemeine Qualität der Daten geschlossen werden.

Betrachtet wurde dies mittels einer Simulation [21]. Zur Simulation verwendet wurde die Software „Sumo“, die einen Verkehrsablauf darstellen kann und virtuelle Positionsdaten für Fahrzeuge erzeugt. Die Rahmenbedingungen der Simulation waren folgende:

- Betrachtet wurde ein Teil der Stadt Nürnberg, der 2037 Knoten und 3648 Kanten umfasst. Es handelt sich um den Stadtteil rings um das Messegelände, der in etwa 10% der Nürnberger Stadtgebiets ausmacht.
- der Simulationszeitraum für eine Messung betrug 60 Minuten.
- Simuliert wurden 10.604 Fahrzeuge auf Routen, die nach Angaben aus Zählschleifendaten des Gebiets erstellt wurden.
- Es wurden Fahrzeuge zufällig ausgewählt, welche Positionen als simulierte FCD abgaben. Die Anzahl der gewählten Fahrzeuge wurde verändert, um verschiedene Ausstattungsgrade betrachten zu können. (Ausstattungsgrad: Prozentsatz der Fahrzeuge, welche FCD senden)
- Die Sendeintervalle von FCD waren ebenfalls variabel, um verschiedene Intervallgrößen untersuchen zu können.

Die So erzeugten Daten wurden mit der FCD-Prozesskette des Institut für Verkehrssystemtechnik des DLR ausgewertet. Aus dieser Untersuchung ergaben sich Beobachtungen, welche sich in folgenden Grafiken widerspiegeln:

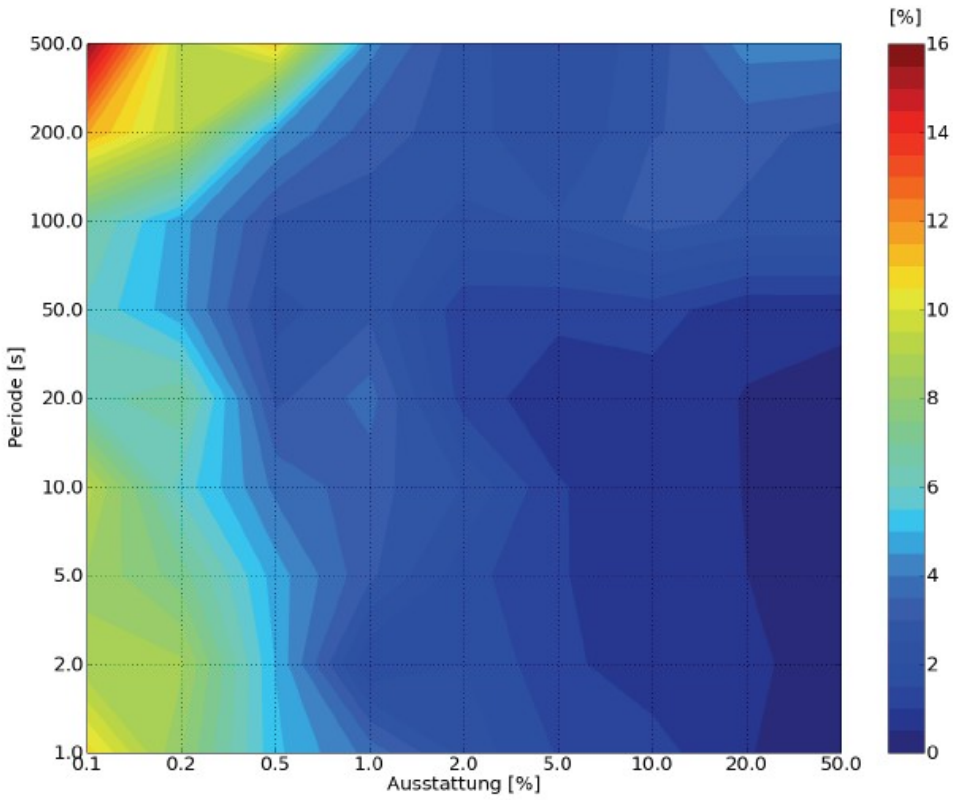

Abbildung 8: Abweichung der Geschwindigkeiten zwischen simulierten FCD und simuliertem Verkehr [21]

Betrachtet wurde hier die Abweichung der Geschwindigkeiten zwischen simuliertem Verkehr und der FCD-Auswertung. Diese Abbildung zeigt, dass ein Ausstattungsgrad von mehr als

0,5% bereits gute Ergebnisse liefert. Ideal ist ein Sendeintervall für FCD-Positionen von 10-50 Sekunden [21]. In der Realität liegt das Sendeintervall bei ca. 30 Sekunden. Interessant ist zudem eine Auswertung, die sich auf die Anzahl der erfassten Kanten des Straßennetzes bezieht:

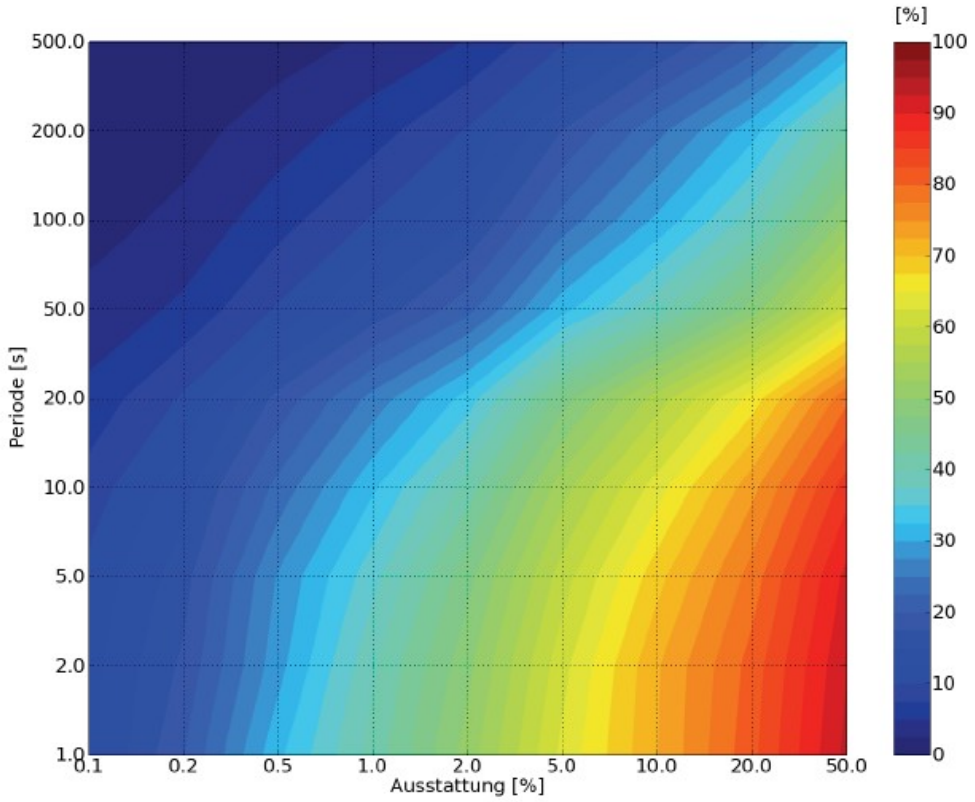

Abbildung 9: Relative Anzahl erfasster Kanten des Auswertungsgebiets der Simulation [21]

Diese Grafik zeigt, wie viele Kanten in der Simulation von den Meßfahrzeugen erfasst wurden. Dies ist wieder stark abhängig von Ausstattungsgrad und des Sendeintervalls. Bei einem Sendeintervall von 30 Sekunden werden laut dieser Auswertung in der Regel nicht alle Kanten erfasst. Erst bei hohen Ausstattungsgraden (über 10%) werden die meisten Kanten erfasst.

In der Realität ist der betrachtete geographische Raum jedoch deutlich größer, bei in etwa der gleichen Anzahl an Fahrzeugen. In der Simulation führte ein Ausstattungsgrad von 0.5% - 1% zu einem guten Ergebnis in der Betrachtung der Fehler bei der Bestimmung von Reisegeschwindigkeiten. Für die vollständige Erfassung der Kanten wurde ein höherer benötigter Ausstattungsgrad ermittelt. In Wirklichkeit wird im betrachteten Raum von ganz Nürnberg und Umgebung jedoch gerade ein Ausstattungsgrad von ca. 0,1% erreicht. Hierbei spielt jedoch folgende Betrachtung eine Rolle: Der Raum Nürnberg ist im Gesamten ca. 10 mal so groß wie das in der Simulation betrachtete Gebiet. Die Taxiflotte aus Nürnberg, welche dem DLR die Daten zur Verfügung stellt, umfasst ca. 500 Fahrzeuge. In der

Simulation wurden ca. 50 Fahrzeuge, die FCD senden, als benötigte Anzahl festgestellt. Der Ausstattungsgrad alleine ist jedoch nicht unbedingt ein Maß für die Qualität. Bei 500 Fahrzeugen aus einer Gesamtmenge von 500.000 ist der Ausstattungsgrad gerade 0.1%, bei 50 Fahrzeugen aus 5.000 wäre er mit 1% entsprechend besser. Ausgehend vom gleich großen betrachteten Gebiet liefern aber 500 Fahrzeuge mehr Positionen und eine damit bessere Abdeckung.

Es lässt sich anhand der Ergebnisse dieser Simulation jedoch feststellen, dass mit den vorhandenen Daten eine annährend realistische Verkehrslageerfassung möglich ist. Um einen möglichst hohen Grad der Erfassung von Kanten zu erhalten, sollte der Betrachtungszeitraum jedoch auf jeden Fall deutlich größer als 60 Minuten sein, um mehr Positionen zu erhalten.

Da im Rahmen dieser Arbeit nur Rohdaten verwendet werden, ist es wichtig, zu beachten, dass die Genauigkeit und die Qualität des Ergebnisses stark von der Qualität der Eingangsdaten abhängig ist. Fraglich ist, in wie weit die Fehlerquellen ohne die Verwendung eines Mapmatching-Verfahrens eliminiert bzw. die Ausprägung der Fehler verringert werden kann. Zu beachten ist, dass im Fokus der Arbeit relativ einfache Auswertungsalgorithmen stehen und nicht auf Vorverarbeitung der Daten und Verringerung der Fehler eingegangen wird. Vielmehr handelt es sich hier um eine Studie, mit der ein neuer, einfacher Ansatz zur Analyse eines bestimmten Merkmals der Verkehrssituation überprüft wird. Die Betrachtung der Fehler hingegen ist ein hoch komplexes Thema, welches zu umfangreich ist, um es in diese Arbeit vollständig einfließen zu lassen.

3.4 Grundlagen für die Zuordnung von FCD-Treffern

Straßennetze können sehr komplex aufgebaut sein. Es gibt verschiedene Ansätze, mit denen FCD-Positionen zu einem Knotenpunkt zugeordnet werden können. Für eine Zuordnung von FCD-Positionen zu Knotenpunkten muss eine Möglichkeit gefunden werden, mit der diese Aufgabe unabhängig von Ausdehnung, Art und Aussehen von Straßen, Kreuzungen oder Einmündungen durchgeführt werden kann. Ohne Betrachtung von Mapmatching-Verfahren bietet hier eine Betrachtung mit Hilfe eines Voronoi-Diagramms einen guten Ansatz.

3.4.1 Voronoi-Diagramm

[19][20] Um 1907 wurde das Voronoi-Diagramm vom ukrainischen Mathematiker Georgy Fedoseevich Voronoi erfunden. Er betrachtete ein Problem, welches schon von dem Mathematiker Dirichlet um 1850 diskutiert wurde. Gegeben seien N Punkte **p** in einer Ebene. Jeder Punkt **p** definiert eine Region, die sich näher an **p** befindet als an jedem anderen der N – 1 Punkte. Diese Region wird als Voronoi-Region bezeichnet. Der euklidische Abstand eines jeden möglichen Punktes in einer Voronoi-Region zu deren Punkt **p** ist kleiner als der euklidische Abstand zu einem anderen Punkt **p**.

Ein Beispiel für ein solches Diagramm zeigt folgende Abbildung:

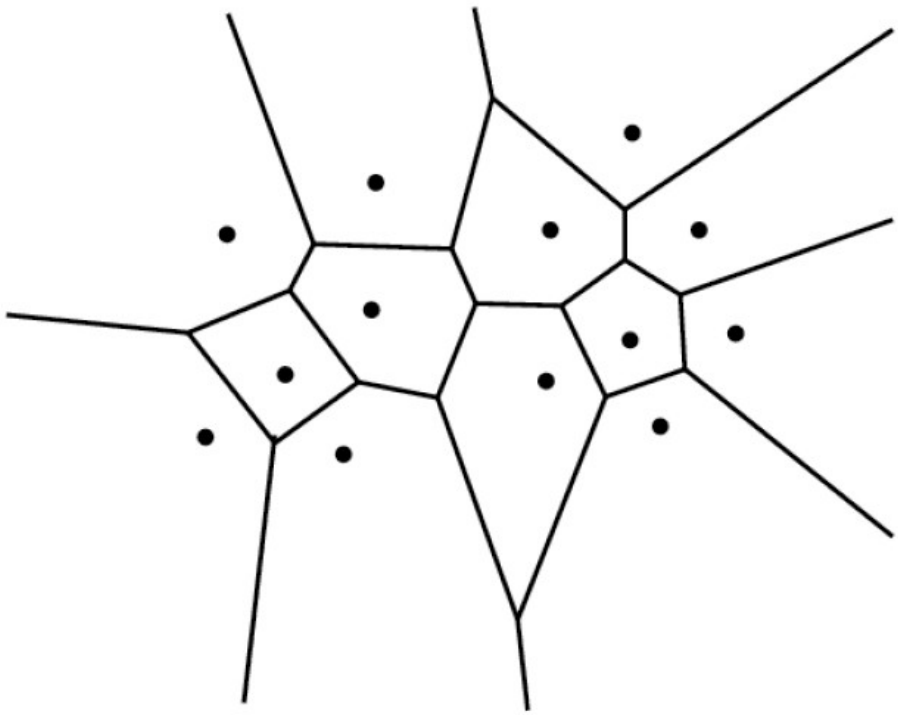

Abbildung 10: Beispiel eines Voronoi-Diagramms [19]

Ein Voronoi-Diagramm in der Ebene besitzt folgende Eigenschaften:

- Voronoi-Regionen werden von Geradenstücken, Voronoi-Kanten genannt, begrenzt.
- Endpunkte dieser Geraden sind Knoten. Liegen Punkte **p** auf einem Kreis, der keinen weiteren Punkt **p** umschließt, ist der Mittelpunkt des Kreises der Knoten zwischen den Grenzen der Regionen um die Punkte **p**.
- Liegen genau m Punkte **p** auf einem Kreis, der keinen weiteren Punkt **p** umschließt, hat der Knoten zwischen ihren Regionen den Grad m, das bedeutet, dass sich m Voronoi-Kanten sowie Voronoi-Regionen an diesem Knoten treffen.
- Voronoi-Regionen sind entweder konvex oder offen in die Unendlichkeit.
- Eine Voronoi-Region ist genau dann offen in die Unendlichkeit, wenn ihr Punkt **p** auf der konvexen Außenhülle der Punktemenge liegt.
- Voronoi-Kanten haben überall den gleichen euklidischen Abstand zu beiden Punkten **p** der Regionen, die sie begrenzen.

Voronoi-Diagramme werden oftmals in Verbindung mit so genannten Delaunay-Diagrammen betrachtet. Delaunay-Diagramme sind Graphen, in denen jeder Punkt **p** mit benachbarten Punkten **p** verbunden ist, so dass durch die Kanten eine Menge von Dreiecken entsteht. Dabei gilt, dass innerhalb eines Umkreises eines jeden Dreiecks kein weiterer Punkt **p** liegen

darf. Für den Zusammenhang von Voronoi- und Delaunay-Diagrammen gilt:

- Voronoi-Diagramm und Delaunay-Diagramm sind duale Graphen.
- Voronoi-Kanten zwischen 2 benachbarten Punkten **p** sind senkrecht zur Delaunay-Kante dieser beiden Punkte, also der Gerade, die diese beiden Punkte verbindet.
- Die Knoten des Voronoi-Diagramms sind genau die Umkreismittelpunkte der Dreiecke im Delaunay-Diagramm.
- Jede Voronoi-Kante schneidet genau eine Kante des Delaunay-Diagramms und umgekehrt.

Auf Straßennetze lassen sich Voronoi-Diagramme wie folgt anwenden: Seien die Knoten im Graphen des Kartenmaterials die Punkte **p** im Voronoi-Diagramm. Dann umschließt jeden Knotenpunkt eine Voronoi-Region. Unter der Voraussetzung, alle Straßenkanten zwischen 2 Knotenpunkten seien gerade, liegt der Teil einer Straßenkante, der sich näher an einem der beiden Knotenpunkte an den Enden der Kante befindet, in der Voronoi-Region dieses Knotens.

Besonders bemerkbar macht sich dies bei geraden Straßen mit einem Mittelstreifen, da hier die beiden Fahrspuren unterschiedliche Voronoi-Regionen haben. Bei Straßenkanten, die Kurven beinhalten, kann diese Betrachtung zu falschen Zuordnungen führen. Dies ist jedoch ohne Betrachtung der Straßenkanten an sich eine nicht eliminierbare Fehlerquelle.

Diese Betrachtung ist die Grundlage für eine Möglichkeit, FCD-Positionen zu Knotenpunkten im Straßennetz zuzuordnen.

3.5 NAVTEQ-Kartenmaterial

3.5.1 NAVTEQ

NAVTEQ ist ein US-amerikanisches Unternehmen, welches 1985 gegründet wurde [6]. Nach eigenen Angaben ist NAVTEQ führender Hersteller von digitalem Kartenmaterial, das in vielen Navigationssystemen in Europa und den vereinigten Staaten von Amerika eingesetzt wird.

NAVTEQ war bis 2007 Lieferant des Navigationssystemherstellers TomTom, welcher im Moment einer der führenden Hersteller von Navigationssystemen ist. (2007 übernahm TomTom jedoch den Kartenmaterialhersteller Teleatlas) [7].

Andere Hersteller von Navigationssystemen, die NAVTEQ einsetzen, sind z.b. Garmin, Falk und Medion. [8][9][10]

Laut Unternehmens-Homepage werden die Karten von eigenen Mitarbeitern mittels GPS-Messungen aufgenommen. Es sind jährlich laut eigenen Angaben mehr als 700 Feldforscher unterwegs, die das Material aufnehmen und die verschiedenen Merkmale der Kartenkomponenten bestimmen.

3.5.2 Auswahl des Kartenherstellers

Bei der Entscheidung, welches Kartenmaterial verwendet werden soll, müssen zunächst mögliche Alternativen betrachtet werden.

OpenStreetMap ist ein OpenSource Projekt [11], in dem Kartendaten zusammengetragen werden. Diese Karten sind frei verfügbar und kostenlos nutzbar. Das Projekt basiert darauf, dass jede Person, die sich an der Erstellung von weltweit freiem Kartenmaterial beteiligen möchte, neue Straßen eintragen kann. Ein entscheidender Vorteil neben der Gebührenfreiheit ist die ständige Aktualisierung des Materials, weshalb permanent aktuelles Material verfügbar ist. Ein Nachteil, wie bei allen OpenSource Projekten, ist, dass die Qualität nicht vollständig sichergestellt ist und keine Gewährleistung auf die Richtigkeit der Daten gegeben werden kann. So könnten unter anderem falsche Straßen eingetragen worden sein.

Teleatlas ist ein Unternehmen, das sich auf der eigenen Webseite ebenfalls als Marktführer bezeichnet. Seit 2007 gehört Teleatlas dem Navigationssystemhersteller TomTom.

Für die Arbeit des Instituts für Verkehrssystemtechnik des DLR werden vorrangig Karten von NAVTEQ verwendet. Dies bietet Vorteile für die Arbeit und die Weiternutzung der Softwarekomponenten.

Zunächst ist das Kartenmaterial im Institut vorhanden, es muss kein neues Material gekauft

werden. Zudem spielt ein wesentlicher Faktor eine Rolle, der Teil der Eigenschaften von digitalem Kartenmaterial ist. Das Straßennetz in digitalen Karten besteht zum wesentlichen Teil aus Kanten und Knoten. Zusammen ergeben die Kanten und Knoten Polygone und Polygonzüge, die unter anderem Straßen repräsentieren. Jede(r) dieser Kanten und Knoten (sowie der Straßennamen, ect.) hat eine eigene ID, also eine Nummer, über die jeder Teil der jeweiligen Komponente des Kartenmaterials eindeutig identifizierbar ist. In der Zusammenarbeit zwischen mehreren Teilprojekten ist es daher notwendig, einheitliches Kartenmaterial zu verwenden. Sollen zum Beispiel in einer Kartendarstellung Kanten oder Knoten hervorgehoben werden, die in einem anderen Teilprojekt identifiziert wurden (Beispiel: Staudarstellung aus Zählschleifendaten), so funktioniert dies nur, wenn die IDs einheitlich und eindeutig sind. Da sowohl die Darstellung als auch die komprimierten Karten, die Teilergebnis dieser Arbeit sind, zusammen mit anderen Projekten verwendet werden sollen, gibt es bezüglich der Wahl des Kartenmaterials keine andere Möglichkeit, als das Material von NAVTEQ zu verwenden.

3.5.3 Beschaffenheit des NAVTEQ-Kartenmaterials

Als Basis dieser Arbeit dient NAVTEQ-Kartenmaterial von Deutschland. Für die Entwicklung wurde ein älteres Paket verwendet, das Kartenmaterial aus dem ersten Quartal von 2005 enthält. Auf die Verwendung dieser Karten bezieht sich die gesamte Arbeit bezüglich des Kartenmaterials.

Bei Betrachtung des Kartenmaterials zu beachten ist die Art der Speicherung von Geokoordinaten. Alle Koordinaten beziehen sich auf das Kartendatum WGS84, sind jedoch im Format 1 Einheit = 0,00001 Grad Länge oder Breite vorhanden. Es sind nur ganzzahlige Werte angegeben. Damit wird (als Referenz: am Äquator) bezüglich der Längengrade eine Genauigkeit von ca. 1,11 Metern erreicht:

$$40.007.800\,m/360\,Grad = 111132{,}778\,m/Grad$$

$$111132{,}778\,m/Grad * 0{,}00001\,Grad/Einheit = 1{,}1113278\,m/Einheit$$

Davon unterscheidet sich die theoretische Genauigkeit bezüglich der Längengrade im Bereich anderer Breitengrade aufgrund der Form der Erde.

Das Kartenmaterial befindet sich auf 4 CD-ROMs mit insgesamt 8 Gzip-Archiven. Diese beinhalten das Kartenmaterial, unterteilt in 8 Gebiete:

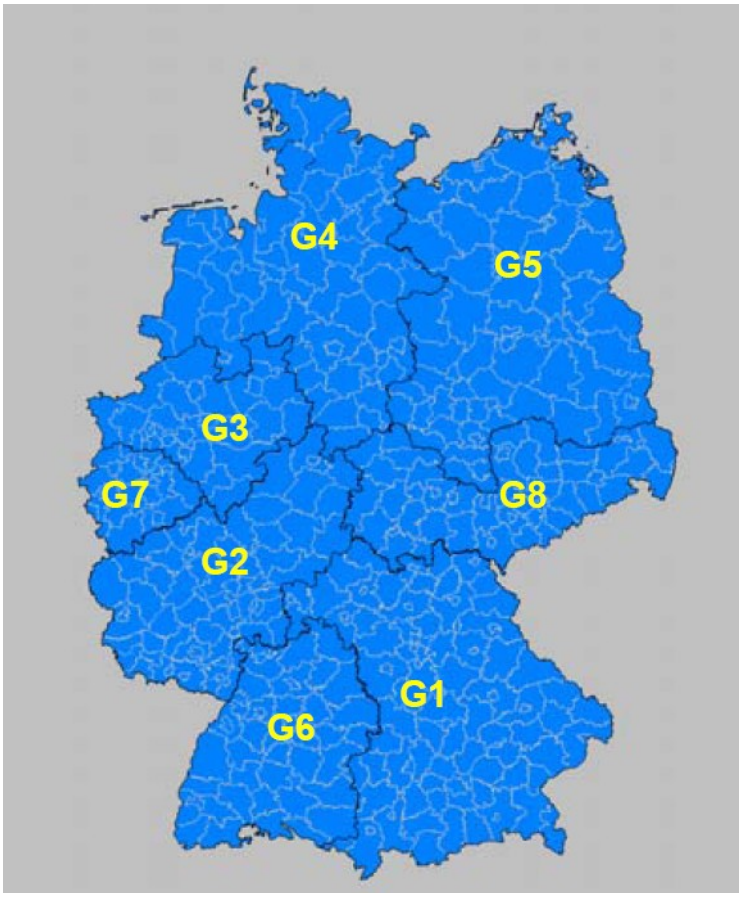

Abbildung 11: Unterteilung des Kartenmaterials von NAVTEQ [12]

Für die Entwicklung der Kartenkompression sowie der Kartendarstellung wurde zunächst das Gebiet „G5“, also Mecklenburg-Vorpommern, Brandenburg, Berlin und Sachsen-Anhalt betrachtet.

Der Inhalt der Gzip-Archive ist jeweils eine Textdatei im ASCII-Format, so genannte GDF-Dateien. Das Format GDF ist im Standard 14825 nach ISO beschrieben [17] und wurde geschaffen, um Kartenmaterial zu vereinheitlichen. Jede Datei enthält zunächst einen Header, in dem allgemeine Informationen zur GDF-Datei enthält (wie in etwa eine Zuordnung von Namen für „Points of Interest“ in verschiedenen Sprachen). Diesem Header folgt die eigentliche Beschreibung des Kartenmaterials.

Das Kartenmaterial ist von seiner geographischen Struktur her in verschiedene Regionen unterteilt. Diese Regionen sind meist Landkreise oder kreisfreie Städte, beispielsweise „Berlin“, „Dahme-Spreewald“ oder „Brandenburg an der Havel“. Eine Region wird von einer Zeile mit dem Namen der Region eingeleitet. Bis zur nächsten Zeile mit einem Regionsnamen folgen alle Informationen zur Region.

Die Informationen in den Regionen selbst werden von so genannten „Records“ repräsentiert.

Die wichtigsten Records sind:

„XYZREC“: Diese Records enthalten Informationen über Geokoordinaten. Sie enthalten eine ID und ein oder mehrere Paare von Geokoordinaten (jeweils Längengrad und Breitengrad).

„KNOTREC“: Hier befinden sich die Informationen für Knotenpunkte, die wichtig für geographische Punkte in der Karte sind, also für „Points of Interest“ oder Start- bzw. Endknoten für Kanten.

„NEDGEREC“: Edge-Records halten die Informationen für Kanten. Diese Kanten können Straßen repräsentieren, aber auch andere Kanten wie Ränder von Gebieten (z.B. Wasserpolygone).

„FACEREC“: Face-Records fassen so genannte „faces“ zusammen, also Flächen wie Seen, Flüsse oder Waldgebiete.

Diese 4 Records repräsentieren die Geometrie des Kartenmaterials. Zusätzlich zur Geometrie sind noch weitere Informationen enthalten. Dafür gibt es zunächst die Attribut-Records. Attribute werden in „DSATREC“ genannten Records gespeichert, die ihrerseits wieder auf „NAMEREC“-Einträge für Namen und „TIMEREC“ genannte Records für Zeitbeschränkungen (wie z.B. zeitlich begrenzte Abbiegeverbote) verweisen.

Die Zuweisung der Attribute zur Geometrie erfolgt in den „Feature Records“. „POFREC“ („point feature Record“), „LINFREC“ („Line Feature Record“) und „ARFREC“ („Area Feature Record“) verweisen jeweils auf die Attribute und die Geometrie von Punkten, Kanten und Gebieten.

Zuletzt gibt es noch den Record für „komplexe Features“, „COMPFREC“. Dieser speichert komplexe Informationen und kann auf die anderen Feature Records sowie auf Attribut-Records verweisen. Länder beispielsweise müssen in komplexen Records gespeichert werden.

Die Records an sich sind dabei eine Art zusammengesetzte Datentypen, die mehrere Zahlen oder Strings enthalten können. Ähnlich wie bei relationalen Datenbanken sind verschiedene Records über IDs einzelner Einträge miteinander verknüpft. Eine Übersicht gibt folgende Grafik [12]:

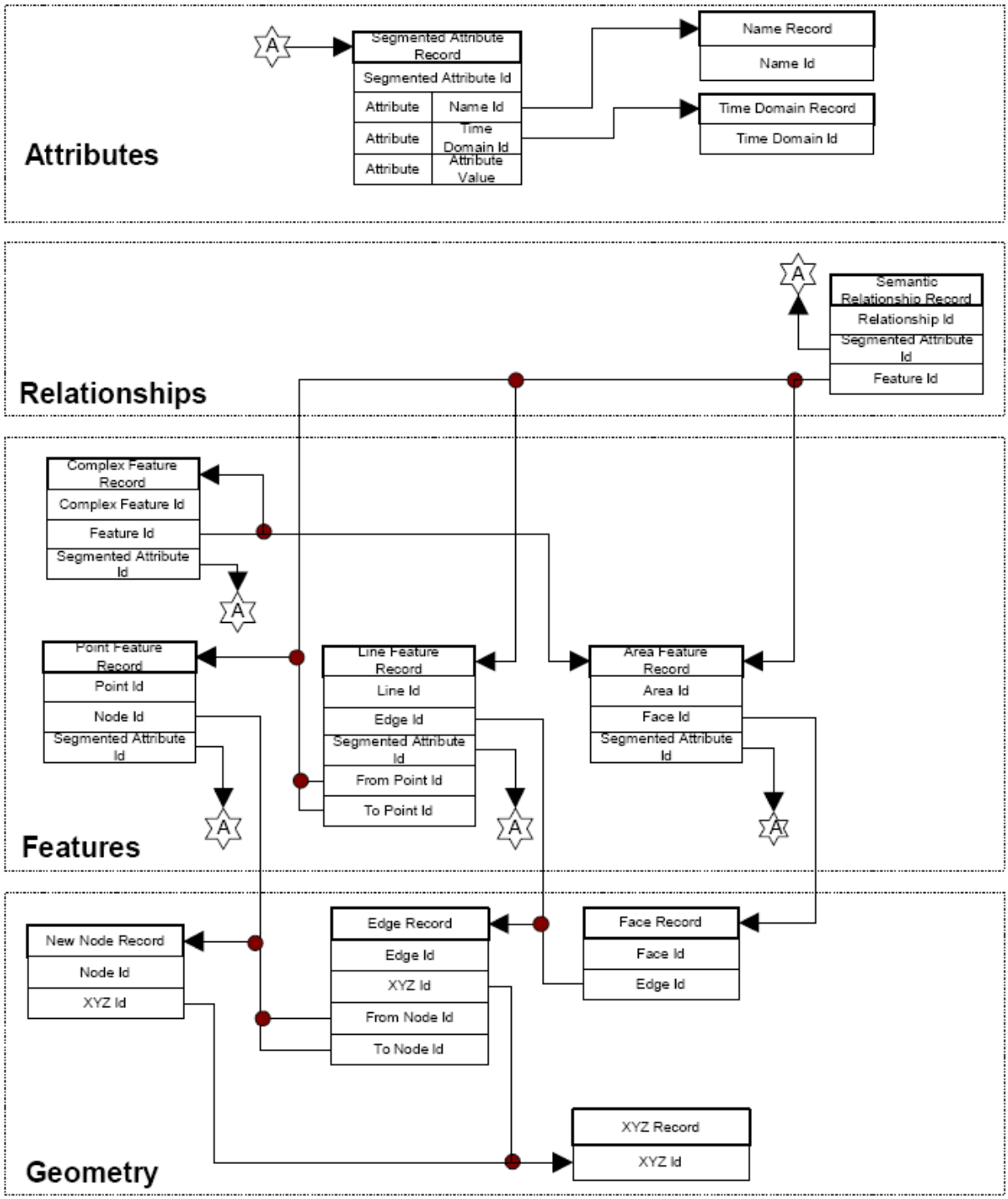

Abbildung 12: Übersicht über die Zusammenhänge in NAVTEQ-GDF Dateien [12]

Aus dieser Grafik lässt sich erkennen, wie die einzelnen Records miteinander verknüpft sind. Ein Beispiel dafür ist eine Straßenkante. Die Straße ist in einem „Line Feature Record“ gespeichert. Sie referenziert sowohl einen Start- als auch Endpunkt, gleichzeitig aber ein „Edge Record“, das wiederum auf den Anfangs- und Endknoten als „Node Record“ zeigt. Das „Node Record“ enthält die IDs der Geokoordinaten des Anfangs- und Endpunktes im

„XYZ Record“, aber auch die der „Point Features“ von Start- und Endpunkt. Da eine Kante aber auch ein Polygonzug sein kann (z.B. bedingt durch Kurven im Straßenverlauf), kann die Kante im „Edge Record“ ebenfalls ein „XYZ Record“ referenzieren, das dann die Geokoordinaten des Kantenverlaufs enthält.

Hieraus ergibt sich eine mögliche Mehrfachnutzung von Einträgen in bestimmten Records, z.B. also eine mehrfache Referenzierung einer Geokoordinate, die mehrere Funktionen gleichzeitig repräsentiert (als Beispiel: Unterbrechung einer Straße mit Aufteilung in mehrere Kanten und Repräsentation einer Ortsgrenze).

Wie bereits beschrieben, werden die Daten in Textdateien im ASCII-Format gespeichert. Eine kurze Zusammenstellung als Beispiel für den möglichen Inhalt einer GDF-Datei zeigt dieser Screenshot:

```
1601000018745OBRANDENBURG·AN·DER·HAVEL·····0¶
23·597584690·····1···1264390···5241920·····0·····0¶
23·597584691·····3···1264401···5241931·····0···1264423···5241961····1¶
00·····0···1264497···5242075·····0·····0¶
23·597584695·····1···1264495···5241891·····0·····0¶
23·597584699·····34···1264408···5242219·····0···1264391···5242234····1¶
00·····0···1264363···5242301·····0···1264357···5242309·····0·····1¶
00···1264342···5242316·····0···1264318···5242321·····0···1264244·····1¶
00···5242329·····0···1264191···5242339·····0···1263892···5242402·····1¶
00·····0···1263813···5242416·····0···1263752···5242435·····0·····1¶
00···1263733···5242443·····0···1263699···5242458·····0···1263652·····1¶
00···5242484·····0···1263624···5242496·····0···1263585···5242490·····1¶
00·····0···1263568···5242490·····0···1263271···5242502·····0·····1¶
00···1263234···5242499·····0···1263213···5242495·····0···1263147·····1¶
00···5242476·····0···1263047···5242445·····0···1262893···5242407·····1¶
00·····0···1262841···5242397·····0···1262833···5242397·····0·····1¶
00···1262807···5242393·····0···1262765···5242388·····0···1262738·····1¶
00···5242408·····0···1262723···5242417·····0···1262676···5242464·····1¶
00·····0···1262643···5242501·····0···1262636···5242505·····0·····1¶
00···1262618···5242509·····0···1262528···5242519·····0·····0¶
23·597584700·····1···1264435···5242207·····0·····0¶
25··60659568··60659568·····2·····0¶
25··60659572··60659572·····2·····0¶
25··60659573··60659573·····2·····0¶
25··60659574··60659574·····2·····0¶
25··60659575··60659575·····2·····0¶
25··60659577··60659577·····2·····0¶
25··60659578··60659578·····2·····0¶
25··60659579··60659579·····2·····0¶
24··53426775·····60676424··60676447·····2·····0¶
24··53426790·····60671721··60676447·····2·····0¶
24··53426793··60669454··60669452··60669455·····2·····0¶
24··53426798··60676493··60676494··60676495·····2·····0¶
24··53426799·····60676494··60676495·····2·····0¶
24··53426904··60676465··60676193··60676464·····2·····0¶
24··53428419··60664950··60664949··60665104·····2·····0¶
24··53428473··60677266··60677265··60677267·····2·····0¶
24··53428476·····60673371··60673256·····2·····0¶
24··53428494··60914250··60914251··60914280·····2·····0¶
```

Abbildung 13: Screenshot einer GDF Datei, im Editor geöffnet mit angezeigten Leerzeichen und Zeilenumbrüchen

Die erste Zeile repräsentiert Anfang einer neuen Region, in diesem Beispiel die Region „Brandenburg an der Havel“.

Ab der zweiten Zeile sind Beispiele für „XYZ Records“ zu sehen. Diese beginnen immer mit der Zahl „23“. Geht ein solcher Record über mehrere Zeilen, enden alle Zeilen, die eine Folgezeile haben, mit einer 1, während die nächste Zeile mit einer 00 beginnt. Nach der Bestimmung des Record-Typs folgt die ID des Records. Darauf sieht man die Paare der Geokoordinaten, die zum Record gehören.

Analog dazu repräsentieren die mit 25 beginnenden Records die Einträge im „KNOTREC“, mit Knoten-ID und ID des „XYZ Records“.

Die mit 24 beginnenden Zeilen enthalten Kanten, also Einträge im „NEDGEREC“, mit ID, optionaler ID der Zwischenkoordinaten (für Polygonzüge) und ID von Start- bzw. Endpunkt.

Ähnlich wie diese Beispiele sind alle Records abgespeichert.

Wie zu erkennen ist, bietet die Abspeicherung in dieser Form mehrere Nachteile bezüglich des Speicherplatzaufwandes.

- Zum einen ist die Abspeicherung im ASCII-Format sehr ineffizient. Während ein Long-Wert binär 64 Bit Speicherplatz benötigt, sind im ASCII-Format aufgrund von 20 möglichen Stellen bereits 160 Bit erforderlich ($20 * 8 = 160$).
- Zum anderen benötigt ebenfalls jedes vorhandene Leerzeichen einen Speicherplatz von 8 Bit. Bei der anfallenden Anzahl von Leerzeichen ist der Speicherplatzbedarf hier enorm.
- Des weiteren ist zu erkennen, dass die Speicherung von Knoten oft überflüssig ist, da diese „XYZ Records“ mit gleichen IDs referenzieren.
- Die Zeilenumbrüche innerhalb der Records verursachen außerdem noch weitere Zeichen, die eingespart werden könnten.

Um speichereffizient mit dem Kartenmaterial arbeiten zu können, sollte es aus diesen Gründen umgeschrieben bzw. komprimiert werden. Je nach Anwendungsgebiet gibt es hier mehrere Möglichkeiten.

3.6 Programmierung

Der Hauptanteil der Arbeit ist eine praktische Umsetzung einer Kartenkompression, einer Kartenladeschnittstelle, eines Moduls zur Visualisierung von Kartenmaterial und eines Tools zur Auswertung von FCD nach Gesichtspunkten der Identifikation von gestauten Verkehrsmustern.

3.6.1 Die Sprache Java

Zur Umsetzung des Programmierteils wurde die Sprache Java verwendet. Die Entwicklung der Sprache begann ursprünglich 1990 durch James Gosling unter dem Namen „Oak“ [13]. Relevanz bekam die Sprache, als die Firma Netscape im Jahre 1995 Java in ihren Browser „Netscape Navigator“ integrierte. 1996 erschien das erste JDK (Java Development Kit). Java wird von der Firma „Sun Microsystems“ weiterentwickelt und ist seit 2007 ein Open Source Projekt.

Bei Java handelt es sich nicht um eine native Sprache, bei der Quellcode durch einen Compiler in Maschinencode umgesetzt wird, der von der Art des Zielsystems abhängig ist. Durch einen Java Compiler wird so genannter Bytecode erzeugt, der über den Zwischenschritt durch eine virtuelle Maschine auf dem Zielsystem zur Laufzeit übersetzt wird. Die virtuelle Maschine enthält einen „Just-in-time-Compiler“, der zur Laufzeit das Programm in Maschinencode übersetzt.

Durch diese Eigenschaft lassen sich Java-Programme in unterschiedlichen Umgebungen einsetzen. In der Regel können Java-Programme ohne Weiteres z.B. auf Linux- oder Windows PCs, PDAs, Macintosh Computern und Anderen ausgeführt werden. Der Einsatz von Java ist nicht auf Applikationen beschränkt. Es können Applets in Webbrowsern ausgeführt werden, „Servlets“ und „Java Server Pages“ sorgen für Einsatzmöglichkeiten zur Webprogrammierung, mit Hilfe von „Java Beans“ lassen sich verteilte Anwendungen realisieren. Der Schwachpunkt von Java ist dagegen die fehlende Hardware- und Betriebssystemnähe. Da die Sprache unabhängig von der Laufzeitumgebung aufgebaut ist, können Betriebssystemspezifische Funktionen oder Hardware nicht direkt angesprochen werden.

Durch die universellen Einsatzmöglichkeiten der Sprache ist sie u.a. gut geeignet für Forschungsprojekte. In der Forschungsarbeit müssen bestimmte Sachverhalte, die erforscht werden sollen, einfach und effizient umgesetzt werden können. Für immer wiederkehrende Aufgaben ist es von Vorteil, wenn bereits vorhandene Softwaremodule und Programmteile wiederverwendet werden können, unabhängig vom Zielsystem. Die Wiederverwendbarkeit des Quellcodes steht im Fokus einiger im Rahmen dieser Arbeit entwickelter Programmteile. Um Software einheitlich gestalten, entwickeln und wiederverwenden zu können, arbeitet das Team für FCD vom Institut für Verkehrssystemtechnik des DLR seit einiger Zeit einheitlich mit Java. Dies ermöglicht die gemeinsame Verwendung von einheitlichen Bibliotheken.

Im Rahmen dieser Arbeit eignet sich Java sehr gut für die Umsetzung der Anforderungen. Es müssen keine betriebssystemspezifischen Aufgaben ausgeführt oder Hardware angesprochen werden. Die einzige notwendige externe Schnittstelle ist die durch die Verwendung einer MySQL-Datenbank erforderliche MySQL-Schnittstelle, welche kein Problem für die Programmierung darstellt.

3.6.2 PDA Programmierung und Besonderheiten im Umgang mit der CrEme-VM

Zu besonderen Demonstrations- und Versuchszwecken stehen dem DLR PDAs vom Typ „O2 XDA II“ zur Verfügung. Solche Anwendungsgebiete könnten beispielsweise Feldtests zur Erforschung von Bewegungsräumen von Testprobanden oder zur Erprobung neuartiger Navigationssysteme sein. In der Vergangenheit sind solche Tests bereits durchgeführt worden, und auch in Zukunft sind Feldtests mit Navigationssystemen zu erwarten.

Ein XDA II ist ein „Personal Digital Assistant“ (PDA) mit integriertem Telefon, also eine Art „Smartphone“. Es besitzt einen Intel „XScale“ Prozessor mit 400 MHz Taktfrequenz und verfügt über einen RAM Speicher von 128 MB [14]. Dieser Speicher wird gleichzeitig als Arbeitsspeicher und als Speicher für Daten verwendet. Da sich dieser Speicher im Falle eines entladenen Akkus selbstständig löscht, ist ein SD-Kartenslot für die permanente Speicherung von Daten integriert. Das Betriebssystem ist „Windows Mobile“ von der Firma Microsoft.

Bisher wurde bei Navigationssystemen innerhalb des DLR auf die Verwendung von Rasterkarten zurückgegriffen, was sich auf Dauer als unbefriedigend darstellte. Die Aufgabenstellung schließt die Programmierung eine Visualisierungskomponente für Vektorkarten mit ein. Eine Anforderung ist die Wiederverwendbarkeit dieser Komponente. In Hinblick auf die mögliche Verwendung der Kartendarstellung mit Hilfe eines PDA sind von vornherein einige zusätzliche Anforderungen entstanden, die sich aus der Leistung des O2 XDA II ergeben.

Zunächst ist das Gerät mit seinem 400 MHz Prozessor im Vergleich zu einem PC relativ langsam. Bei der Programmierung ist daher besonders auf Effizienz und Geschwindigkeit zu achten. Der Hauptspeicher, von dem nur ein Teil als Arbeitsspeicher verwendet werden kann, ist mit 128 MB sehr klein, was einen Umgang mit großen Datenmengen sehr kritisch macht. SD-Karten als Möglichkeit, größere Datenmengen zu speichern, besitzen unterschiedliche Kapazitäten. Gängige Größen von Karten, die dem DLR zur Verfügung stehen, liegen bei 512 MB oder 1024 MB. Es sind aber auch weit größere SD-Karten im Handel erhältlich. Ziel bei der Kartenkompression in dieser Arbeit ist es jedoch, bei einer Abspeicherung des Kartenmaterials für ganz Deutschland, 1 Gigabyte nicht zu überschreiten.

Auf einem O2 XDA II ist keine Java Laufzeitumgebung vorinstalliert. Um Java auf diesen Geräten verwenden zu können, kann die Java Virtual Machine „CrE-ME“ der Firma NSIcom

im Evaluations-Modus verwendet werden [15]. Diese VM weist die Besonderheit auf, dass sie von Funktionsumfang und Bibliotheken her in etwa der Java-Version 1.3 entspricht. Dies muss bei der Programmierung beachtet werden. Leider entfällt bei der Verwendung der „CrE-ME“ VM zum Beispiel die Benutzung von Generics und Java-Annotations, da diese nicht interpretiert werden können. Auch andere spezielle Befehle können zu Fehlern führen, was die Programmierung von Java-Software für diese Virtual Machine einschränkt. Solche Einschränkungen finden sich im Quellcode des Kartenlademoduls und der Kartendarstellung wieder.

3.6.3 Besonderheiten im Umgang mit Kartenmaterial und FCD

Sowohl bei der Verwendung von Vektorkarten als auch bei der Untersuchung von FCD fallen sehr große Datenmengen an. Sämtliche Programme und Module, die im Rahmen dieser Arbeit erstellt wurden, können gegebenenfalls mehr als 100 MB Platz im Hauptspeicher benötigen. Zunächst ist beim Start der Programme zu beachten, dass mittels der Java-VM-Optionen der zugesicherte Arbeitsspeicher erhöht wird, z.B. durch den Befehl „-Xmx1024m“.

Bei der Programmierung sind einige Überlegungen wichtig, um mit großen Datenmengen umgehen zu können. Im Wesentlichen gibt es in Java die Möglichkeiten, viele Daten in Arrays, Vector-Objekten oder Hashtables zu speichern. Dabei muss überlegt werden, welche Möglichkeit gebraucht wird. Der Grund ist der Overhead von Objekten. Die Auswirkung des Overheads von Java-Objekten kann in einem simplen Vergleich des Speicheraufwands von 1.000.000 Variablen des simplen Typs „int“ und des zugehörigen Wrapper-Objekts „Integer“ verdeutlicht werden. Mit Hilfe des Testprogrammes „Speichertest“ wurden 2 Arrays mit den entsprechenden Typen erzeugt. Im Windows-Taskmanager wurde die Speicherdifferenz abgelesen. Das Ergebnis ist, dass ein „int“ ca. 4 byte belegt, während ein „Integer“-Objekt bereits ca. 20 byte im Speicher benötigt. In diesem Falle wird also das fünffache an Speicher benötigt, was bei sehr vielen Einträgen eine große Erhöhung des Speicheraufwands bedeutet. (siehe Testprogramm „Speichertest.java“ in der Beilage)

Arrays haben die geringste Größe, sofern sie einfache Datentypen beinhalten. Hier hat nur das Array selbst einen Overhead, die einzelnen Elemente nicht. Der Nachteil ist jedoch, dass Arrays relativ starre Konstrukte sind. Will man Arrays benutzen, muss vorher bekannt sein, wie viele Daten gespeichert werden sollen. Beispielsweise bei einer festen Anzahl von FCD-Positionen, die sich nicht mehr verändert, können diese verwendet werden.

Vector-Objekte dagegen speichern jeden Eintrag als Objekt und weisen somit einen Overhead für jedes Element auf. Sie sind zur Laufzeit beliebig veränderbar, können vergrößert und verkleinert werden. Vector-Objekte können verwendet werden, wenn Elemente durch Iteration nacheinander abgearbeitet werden sollen. Um ein spezielles Element zu suchen, wird jedoch viel Zeit benötigt.

Für die Speicherung von Kanten und Knoten des Kartenmaterials lassen sich daher am besten Hashtable-Objekte verwenden. Hashtables haben die Eigenschaft, dass sich ihre

Elemente durch Schlüssel identifizieren lassen. Vom Schlüssel eines Elements kann eine Java-Hashtable intern durch die Hashfunktion auf die Position des Elements zurück rechnen, was einen sehr schnellen Zugriff ermöglicht. [26] Auf Kanten und Knoten muss im Regelfall mit Hilfe der ID schnell zugegriffen werden können.

Alle 3 Konzepte wurden an den entsprechenden Stellen in der Software verwendet.

4 Anforderungen, Konzept und Lösung der Kartenkompression und Kartendarstellung

4.1 Kartenkompression

Die Kompression des NAVTEQ-Kartenmaterials ist ein wichtiges Hilfsmittel für die Darstellung von Kartenmaterial. Wie in Punkt 3.5 beschrieben, benötigt das Kartenmaterial in seiner Ursprungsform viel Speicherplatz und ist komplex aufgebaut, was eine Auswertung erschwert. Daher sollte es in eine andere Form gebracht werden, um dargestellt werden zu können.

4.1.1 Anforderungen und Vorbetrachtungen

Bei der Wahl der Kartenkompression muss darauf geachtet werden, welche Anforderungen erfüllt werden müssen. Im Rahmen dieser Masterarbeit sind die Grundanforderungen an das Kartenmaterial:

- Es sollen nur Straßen angezeigt werden. Andere Karteninformationen (z.B. Informationen über Waldgebiete und Flüsse) entfallen.
- Das Kartenmaterial soll möglichst wenig Speicherplatz benötigen.
- Kartendaten sollen schnell geladen werden können.
- Ein geographisch differenziertes Laden soll möglich sein.

Die Grundanforderungen führen zu konkreten Vorbetrachtungen, wie das Kartenmaterial komprimiert werden soll.

Speicherplatzkritisch ist die Kompression, weil die Anforderung gegeben ist, dass das Kartenmaterial auf kleinen Speichermedien, wie zum Beispiel einer SD-Karte, die sich mit einem PDA (hier: O2 XDA II) zusammen verwenden lässt, abgelegt werden kann. Gängige Größen für solche SD-Karten, liegen momentan bei ca. 1 Gigabyte (Stand 2008). Es sind auch größere SD-Karten erhältlich, da die Entwicklung von Speichermedien permanent voran schreitet, 1 Gigabyte wurde jedoch als Grenze festgelegt, um eine möglichst gute Effizienz zu erreichen.

Zum Darstellen der Karten ist es zudem wichtig, das Material schnell und differenziert laden zu können. „Differenziert“ in diesem Sinne bedeutet, dass das Material nach zwei Gesichtspunkten komprimiert und sortiert abgelegt werden muss.

Zum einen ist es beim Anzeigen eines Kartenausschnittes nicht notwendig, Kartenmaterial zu laden, welches sich außerhalb des anzuzeigenden Gebietes befindet. Um kein unnötiges Material zu laden, können die Kartendaten in ihrer Struktur in rechteckige Teile zerlegt werden, so dass von der Größe des anzuzeigenden Bereiches schnell zurückgerechnet werden kann, welche Teile einer Karte geladen werden müssen. Im Originalformat sind

NAVTEQ-Karten jedoch nach Regionen gespeichert. (Diese rechteckigen Teile werden innerhalb dieses Dokuments als „Rasterteile“ bezeichnet.)

Zum anderen kann das Kartenmaterial abhängig von der Größe des anzuzeigenden Kartenausschnittes betrachtet werden. Straßenkanten können in verschiedene Kategorien unterteilt werden, je nach ihrer Art. So gibt es zum Beispiel Autobahnen, Landstraßen und Nebenstraßen in Ortschaften. Beim Anzeigen eines kleinen Kartenausschnittes sind im Regelfall alle Straßen relevant, so zum Beispiel, wenn eine Route oder ein Straßennetz im Detail angezeigt werden soll. Wird ein großer Ausschnitt angezeigt, um beispielsweise einen Überblick über eine große Region wie Berlin oder ganz Deutschland darzustellen, sind Nebenstraßen und regionale Hauptstraßen zumeist eher unwichtig, bzw. sogar störend. Sie können den Überblick verschlechtern und benötigen Platz im Hauptspeicher bzw. Bandbreite bei der Übertragung in einem Netzwerk. Eine Abhilfe schafft hier die Möglichkeit, das Kartenmaterial zusätzlich zur geographischen Unterteilung außerdem noch nach Straßentypen getrennt abzuspeichern. So können beispielsweise Autobahnen, Kraftfahrstraßen und Bundesstraßen gesondert von den Nebenstraßen geladen werden.

Als Anforderung an den Prototypen für die Kartenkompression sind lediglich die Abspeicherung von Informationen über Straßennetze gegeben. Platz kann also gespart werden, indem Informationen über Orts- und Landesgrenzen, Flüsse und Seen sowie Wälder, Stadtgebiete, Eisenbahnschienen und „Points of Interest“ nicht abgelegt werden. Hierdurch verringert sich die Größe des abzulegenden Kartenmaterials.

Anforderung ist es somit, ein Tool zu schaffen, welches NAVTEQ-Kartenmaterial nach diesen Gesichtspunkten abspeichert.

4.1.2 Grundkonzept

Die Kompression erfolgt in mehreren Schritten, die hier beschrieben werden.

Zunächst wird ein Kartenausschnitt gewählt, der das Gebiet umfasst, welches komprimiert werden soll. Dazu ist es notwendig, grob die Start- und Endkoordinaten des Gebietes in Y- und X-Richtung (Längen und Breitengrade) anzugeben, sowie die Aufteilung der Karte in einzelne Rasterteile festzulegen. Das bedeutet: Unterteilt man Deutschland (ca. 6° - 15° östliche Länge und 45° - 60° nördliche Breite) in ein Raster von 100 * 300 Teilen, erhält man ein Raster mit einer ungefähren Kantenlänge von 5500 Meter pro Rasterteil (differiert in der Ost-West Richtung, abhängig vom Breitengrad).

Dieses Raster wird für die Komprimierung vorbereitet, so dass für jede Geokoordinate bestimmt werden kann, in welchen Teil des Rasters das ihr zuzuordnende Merkmal des Kartenmaterials (Straße, Knotenpunkt) einzuordnen ist.

Danach wird das Kartenmaterial, nach einzelnen Regionen gestaffelt, ausgelesen. Jede der Regionen wird einzeln komprimiert. Das komprimierte Material wird dem Raster beziehungsweise dem schon vorhandenen komprimierten Material hinzugefügt. Zunächst

werden nur Straßen behandelt. Für jede Kante im Straßennetz wird festgestellt:

- Welchen Straßennamen hat die Kante?
- Welche Knoten und Zwischenknoten hat die Kante?
- In welchem Teil im Raster muss die Kante gespeichert werden bzw. gibt es Überschneidungen, so dass die Kante gegebenenfalls in 2 oder 4 Rasterteilen gleichzeitig gespeichert werden muss?
- Ist diese Kante, zur Festlegung von Richtungen, im Material doppelt vorhanden, und muss für eine Darstellung eventuell nur einmal gespeichert werden?
- Welchen Straßentyp hat die Kante?

Mit diesen Informationen können nun Straßennamen, Knotenpunkte, Zwischenknoten und die Kante gespeichert werden.

Nicht alle Informationen, die eine Kante enthält, sind für eine Darstellung relevant. So müssen für eine Anzeige zum Beispiel die Reihenfolge der Hausnummern, die erlaubten Geschwindigkeiten oder die Beschränkungen für bestimmte Fahrzeugtypen nicht gespeichert werden. Durch das Weglassen dieser Informationen kann Platz gespart werden (verlustbehaftete Kompression). Die wichtigen Informationen über Kanten, Knoten und Straßennamen werden nun, mit Trennzeichen versehen, als Zeichenketten im Bereich ihrer jeweiligen Rasterteile abgespeichert.

4.1.3 Vom GDF zum geladenen Kartenmaterial

Zur Kompression des Kartenmaterials wäre es möglich, das Kartenmaterial direkt aus den GDF-Dateien von NAVTEQ zu laden. Dies hat mehrere Nachteile.

Zum einen ist das Kartenmaterial sehr groß. Beim Laden von Dateien mit Größen von mehreren hundert Megabyte würde es bei der Kompression mit Hilfe eines Java-Programms schnell zum Überlaufen des Hauptspeichers kommen.

Zum anderen ist das Kartenmaterial in seinem Aufbau sehr komplex, was es aufwendig macht, die relevanten Informationen heraus zu filtern und in geeigneter Form abzuspeichern.

Zur Arbeit mit dem Kartenmaterial wurde im Institut für Verkehrssystemtechnik des DLR bisher ein eigenes Format verwendet, welches das Material in besser auswertbarer Form enthält. Es enthält die Kanten nach zwei verschiedenen Schemata abgespeichert, „splitted“ und „unsplitted“. „Splitted“, im Gegensatz zum Orignialformat, enthält keine Kanten, die Polygonzüge sein können. Kanten, die aus mehreren Polygonen bestehen, werden hier in einzelne „Unterkanten“ zerlegt. Das bietet zwar den Vorteil, dass die Kanten eventuell leichter zu laden und darzustellen sind, jedoch benötigen geteilte Kanten mehr Speicherplatz, und es müssen eigene Kanten-IDs vergeben werden. Dies schafft Inkompatibilität zum Originalformat. „Unsplitted“ ist die Methode, nach der auch das Originalformat abspeichert ist. Die IDs sind hier eindeutig, und Kanten, die für die Darstellung

von Kurven Polygone enthalten, verweisen auf separate Zwischenknoten. Im Folgenden wird nur auf das Format eingegangen, welches Zwischenknoten enthält.

Dieses Format besteht aus Verzeichnissen, die nach den Namen der jeweiligen Regionen benannt sind (die Regionen richten sich hierbei nach den Regionen, nach denen die originalen GDF-Dateien unterteilt sind). Diese Verzeichnisse beinhalten mehrere Textdateien. Diese speichern, getrennt voneinander, Knoten, Kanten, Namen (z.B. Straßennamen), andere Polygone (Bebauung, Waldgebiete, Wasser,...), „Points of Interest“, Zeitbeschränkungen und verbotene Manöver.

Die Textdateien enthalten Datensätze, die zeilenweise abgelegt sind (also Zeichenketten, getrennt durch Zeilenumbrüche). Jede der Zeilen enthält die einzelnen Attribute des Datensatzes, getrennt durch Tabulatoren. Das erste Attribut ist hierbei immer die einzigartige ID des Datums. Als Beispiel hier die Repräsentation einiger Knotenpunkte:

```
#·NODE_ID   »   IS_BETWEEN_NODE»amount_of_geocoordinates   »   x1   »   y1   »   [x2   »   y2   »   ...   »   xn   »   yn]¶
60454406   »   0   »   1   »   1190815»5196007¶
60454407   »   1   »   2   »   1190890»5196042»1190927»5196052¶
60454411   »   0   »   1   »   1190986»5196063¶
60454412   »   1   »   4   »   1191008»5196069»1191168»5196135»1191261»5196157»1191428»5196158¶
60454420   »   1   »   2   »   1191591»5196132»1191509»5196150¶
60454421   »   0   »   1   »   1191660»5196114¶
```

Abbildung 14: Ausschnitt aus dem internen Kartenformat des Institut für Verkehrssystemtechnik des DLR

Für das Laden dieses Formats gibt es eine bereits existierende Java-Klasse. Diese benötigt den Pfad des Verzeichnisses, in dem die Karten abgelegt sind. Sie kann aus diesem Format die Kanten, Knoten und Namen als Hashtables laden. Dies vereinfacht die weiterführenden Schritte der Kompression und lässt sich gut als Basis verwenden. Der Nachteil an dieser Klasse ist jedoch, dass immer das gesamte Kartenmaterial für einen Abschnitt geladen werden muss, was sehr speicheraufwendig ist.

Um dieses Format verwenden zu können, muss zunächst das Kartenmaterial aus den GDF-Dateien ins DLR-interne Format überführt werden. Dazu sind zwei Tools notwendig, welche ebenfalls DLR intern bereits existierten.

Das Erste ist ein „GDFHack“ genanntes Tool. Es unterteilt die GDF-Dateien, wie sie im Auslieferungszustand vorliegen, in kleinere GDF-Dateien, nach einzelnen Regionen. Mit diesem Werkzeug muss das Material zunächst zerlegt werden, um es speichereffizient auswerten zu können. Die Behandlung einer kompletten GDF-Datei würde den Rahmen des Arbeitsspeichers eines normalen PCs sprengen.

Das zweite Tool ist der so genannte „GDF-Extraktor“. Dieser erzeugt letztendlich das DLR-interne Kartenformat aus den einzelnen GDF-Dateien.

Beide Tools wurden zur Verwendung analysiert und leicht angepasst, um eine Basis für die Kompression des Kartenmaterials zu erhalten.

4.1.4 Lösungsmöglichkeiten zur effektiven Komprimierung

Das Weglassen ausgewählter Informationen alleine verringert das Datenvolumen des Kartenmaterials beachtlich. Auch die Überführung in ein Format, das keine unnötigen Leerzeichen und Zeilenumbrüche beinhaltet, macht die Abspeicherung effektiver. Für eine weitere Verringerung des Speichervolumens sind jedoch noch weitere Schritte zur Kompression, vor allem von ganzzahligen Werten, die in Strings gespeichert werden, notwendig. Dies führte zum ersten Ansatz zur weiteren Kompression.

In Textdateien sind sowohl beliebige Zeichenketten als auch Zahlen im ASCII-Format gespeichert. Zahlen lassen sich in Ziffern unterteilen. Eine Ziffer ist ein Zeichen mit 10 unterschiedlichen Ausprägungen, im ASCII-Format kann jedes Zeichen jedoch 256 unterschiedliche Werte haben. Hier ist zu erkennen, dass ein Ablegen im ASCII-Format einen unnötigen Platzbedarf erzeugt. Binär abgelegt benötigt beispielsweise ein Long-Wert 64 Bit, als ASCII-Zeichen aufgrund von 20 möglichen Zeichen bis zu 160 Bit.

Daher erscheint es zunächst als sinnvoll, das Kartenmaterial (vor allem Kanten und Knoten, welche nur Zahlen enthalten) im Binärformat abzuspeichern. Dies spart Platz gegenüber der Darstellung im ASCII-Zeichensatz. Der erhebliche Nachteil dieser Art von Speicherung ergibt sich jedoch aus der Beschaffenheit der Knoten. Ein Knoten kann eine beliebige Anzahl von Koordinatenpaaren haben. Beim sequenziellen Auslesen von Daten aus einer Binärdatei in Java muss jedoch bekannt sein, welcher Datentyp als jeweils nächstes gelesen werden muss. Eine Navigation in einer Binärdatei mit Java ist kompliziert und nicht ohne Weiteres möglich, daher müssen Binärdateien immer als Ganzes gelesen werden. Für eine Unterteilung in Rasterteile funktioniert dies also nur, wenn ein Teil komplett in einer Datei gespeichert wird.

Eine Möglichkeit, gute Ergebnisse in der Komprimierung zu erzielen, und trotzdem ein gut funktionierendes zeilenweises Auslesen einer Datei zu ermöglichen, ist die Darstellung von Zahlen im ASCII-Format, allerdings in komprimierter Form. Anstatt als Kette von Zeichen zur Basis 10 (Ziffern) wäre es mit dem ASCII theoretisch möglich, Zahlen als Kette von Zeichen zur Basis 256 darzustellen. Leider entfallen einige Zeichen des ASCII, weil sie Steuerzeichen sind oder das Trennen von Strings mit Hilfe von Tabs und Zeilenumbrüchen unmöglich machen. Es bleibt nach dem Aussortieren kritischer Zeichen jedoch noch ein relativ großer Zeichensatz übrig. Verwendet wurde in einem Experiment zu dieser Art von Kompression ein Satz von 184 Zeichen. Mittels wiederholter Modulodivision mit 184 lässt sich eine Zahl so in mehrere Zahlen mit einer Größe von 0-183 zerlegen, welche dann mit Hilfe einer Tabelle, die den in entsprechenden Zeichensatz enthält, in ASCII-Zeichen umgewandelt werden können.

Um die Effektivität dieser Methode zu prüfen, wurde experimentell ein Vergleich mit der Java eigenen Speicherung in Binärdateien durchgeführt. Dazu wurde das Programm „Kompressionstest“ verwendet, als Eingangsdaten dienten Knotenpunkte aus dem Raum Berlin. Die Ausgabedatei war bei der Kompression von 61991 Knotenpunkten sogar geringfügig effektiver. Während die Dateigröße bei binärer Speicherung 1.368 Kilobyte

betrug, wurde mit der Stringkompression eine Größe von 1.194 Kilobyte erreicht (siehe Testprogramm „Kompressionstest.java“ in der Beilage). Dies resultiert unter anderem auch daraus, dass bei der binären Speicherung nicht alle Zahlen immer den kleinstmöglichen Typ annehmen können, da die Größe der Zahlen nicht bekannt ist.

Im Zuge der Abspeicherung nach separat ladbaren Rasterteilen musste nun ein geeignetes Konzept für diese Unterteilung gefunden werden.

Der erste Ansatz war das Erstellen eines Rasters durch Unterteilung in viele kleine Textdateien. Dies bringt einen großen Nachteil mit sich. Bei einem Raster von 100 * 300 wären schon 30000 einzelne Dateien notwendig. Bei einer zusätzlichen Aufteilung in 5 verschiedene Straßentypen wären es 150000 Dateien, jeweils für Kanten, Knoten und Straßennamen, somit also 450000 Dateien. Im Regelfall haben viele Dateisysteme mit so großen Dateimengen in einem Ordner Probleme. Das Laden wäre sehr ineffizient, ebenso die Portierbarkeit des Kartenmaterials. Dieser Ansatz wurde daher nicht umgesetzt.

Ein zweiter Ansatz war die Unterteilung des Kartenmaterials in Raster innerhalb einer Datei. Dies geschieht durch das Einfügen von Zeilen, die einzelne Rasterteile innerhalb einer Datei trennen, und das sortiere Einfügen des Kartenmaterials in diese Bereiche. Wenn die byteweisen Positionen der Anfänge dieser Bereiche separat abgespeichert werden, kann mittels der Java-Klasse „RandomAccesFile“ an die jeweilige Position in der Datei gesprungen und die Folgezeilen können geladen werden. Um die Bereiche noch in Straßentypen zu unterteilen, genügt es, die Straßen nach Straßentyp sortiert einzufügen (Autobahnen an erster, Nebenstraßen an letzter Stelle). Dadurch kann das Laden abgebrochen werden, wenn alle benötigten Straßen geladen wurden. Dieser Ansatz wurde weiterverfolgt programmiertechnisch umgesetzt.

Die Nachteile dieses Verfahrens liegen zum einen in der Umsetzung des „RandomAccesFile“ in Java. Ein gepuffertes Lesen ist mit „RandomAccessFile“ ist nicht ohne Weiteres möglich. Einlesen ohne Lesepuffer hingegen ist nicht performant. Um gepuffert einlesen zu können, muss die Open Source Bibliothek „Unified I/O“ [27] verwendet werden. Ein Test auf dem PDA zeigte, dass das Lesen trotzdem sehr aufwendig ist und lange dauert.

Eine weitere Möglichkeit ist es, die Daten in ZIP-Dateien abzulegen. ZIP-Dateien können von Java mit einer Standardbibliothek gelesen werden. Der Ansatz war, die Rasterteile mit einzelnen Dateien zu repräsentieren, welche in eine ZIP-Datei gespeichert werden. Dadurch werden die Probleme von Dateisystemen mit großen Dateimengen umgangen. Auch auf die Kompression der einzelnen Zeichenketten kann hier verzichtet werden, da ZIP-Dateien die Daten ohnehin komprimieren. Die ZIP-Kompression in Java ermöglicht die Verwendung verschiedener Kompressionsstufen. Für die Entwicklung wurde für die Entwicklung die Kompressionsstufe 0 der Java Bibliothek gewählt. Diese Stufe entspricht jedoch nicht der Kompressionsstufe 0, die für das das ZIP-Format spezifiziert ist, da diese nur speichern, jedoch nicht komprimieren würde [28]. Theoretisch ist aber eine höhere Kompressionsstufe möglich.

Ein Problem, welches bei diesem Verfahren auftritt, ist die enorme Anzahl der Einträge innerhalb der ZIP-Datei. Auf einem PDA ließ sich die ZIP-Datei wegen dieser Größe nicht öffnen. Um dies zu umgehen, kann man die Einträge so auf verschiedene ZIP-Dateien verteilen, dass ein übergeordnetes Raster entsteht. Wenn man also Deutschland in 100 * 300 Teile teilen will, kann man beispielsweise pro ZIP-Datei 10 * 10 Rasterteile ablegen, aufgeteilt in 10 * 30 ZIP-Dateien. Dadurch erhält man sowohl für Kanten als auch für Knoten je 300 ZIP-Dateien mit jeweils 500 Einträgen. Dies ist sowohl eine Größe, die von Dateisystemen leicht zu verwalten ist, auch verringert sich die Ladezeit bei einer Darstellung, da die ZIP-Dateien effizienter durchsucht werden können.

Um diese Verfahren zahlenmäßig gegenüberstellen zu können, wurden einige kleine Beispieltests durchgeführt. Komprimiert wurde jeweils das Gebiet „G5“, also Berlin, Brandenburg, Mecklenburg-Vorpommern und Sachsen-Anhalt, in allen Fällen mit einem vorbereiteten Raster, welches ganz Deutschland unterteilen könnte. Dieses Raster bedeutet jedoch, dass ein gewisser Overhead dadurch anfällt, dass auch nicht genutzte Rasterteile durch leere ZIP-Dateien oder Trennzeilen in Textdateien repräsentiert werden.

Gemessen wurde bei der Kompression jeweils der Zeitaufwand für das Komprimieren vom internen DLR Format aus, auf einem PC mit einer 3 GHz Intel CPU und 1,5 Gigabyte Arbeitsspeicher. Anschließend wurde die Größe des komprimierten Kartenmaterials auf dem Datenträger gemessen.

Geladen wurde jeweils dieser Kartenausschnitt (ein Teil von Berlin, rund um die Stadtautobahn Avus). Verwendet wurde hierzu eine frühe Entwicklungsversion des Kartendarstellungsmoduls. Das Material wurde in ein Raster von 100 * 300 Teile zerlegt, die Kantenlänge eines Rasterteils beträgt rund 5,5 Km. Zu sehen ist der experimentell geladene Ausschnitt des Kartenmaterials in folgendem Screenshot:

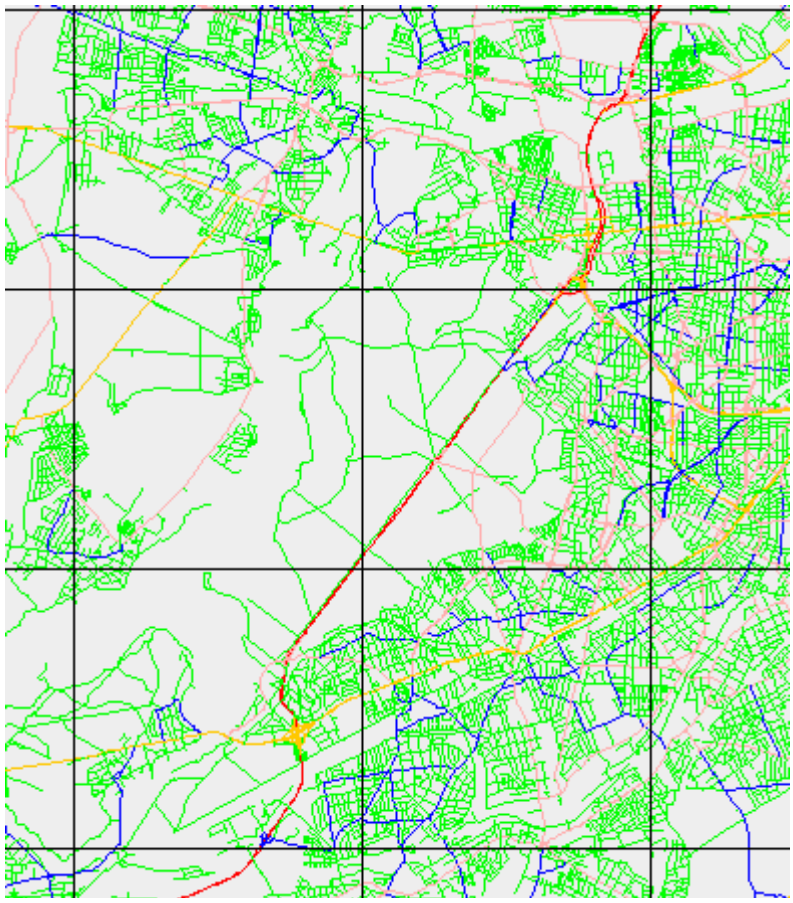

Abbildung 15: Beispielhaft geladener Kartenausschnitt Berlins zur Feststellung der Ladezeit von unterschiedlich komprimiertem Kartenmaterial mit angezeigtem Raster

Die Ergebnisse des Tests stehen in folgender Tabelle:

	Kompressionszeit	Ladezeit PC	Größe
Textdatei	Ca. 1 Stunde	-	73,8 MB
Textdatei komprimiert	Ca. 1 Stunde	3,9 Sekunden	43 MB
ZIP-Datei	Ca. 3 Stunden	4,9 Sekunden	56,7 MB
ZIP-Datei (geteilt)	Ca. 13 Minuten	1,4 Sekunden	57,5 MB

Tabelle 5: Größe und Ladezeiten von beispielhaft komprimiertem Kartenmaterial

Nach der Ladezeit auf dem PC zu urteilen ist das Verfahren des Ladens aus einer geteilten ZIP-Datei (doppelte Rasterung) am effektivsten.

Entscheidend für dieses Verfahren ist die Wahl der Rastergröße. Sind die Rasterteile zu

groß, müssen für kleine Kartenausschnitte zu viele Daten geladen werden. Bei zu kleinen Rasterteilen entstehen zu viele ZIP-Einträge beziehungsweise zu viele Dateien, außerdem sind viele Straßenkanten, die in mehreren Rasterteilen auftauchen, redundant gespeichert. Die Rastergröße ist demnach so zu wählen, dass ein optimaler Mittelweg gefunden wird. Im Beispiel des Gebiets der Bundesrepublik Deutschland ist eine Rastergröße von ca. 5,5 km, also der Unterteilung in 100*300 Rasterteile, ein guter Kompromiss. Ein Kartenausschnitt von dieser Größe benötigt relativ wenig Speicher, das Raster ist aber nicht zu groß für eine effektive Speicherung.

4.2 Kartenlademodul

4.2.1 Anforderungen und Grundkonzept

Im Laufe der Arbeit hat sich gezeigt, dass es von Vorteil ist, das Laden des Kartenmaterials in ein eigenes Modul auszulagern, statt es direkt in der Visualisierung zu implementieren. Dies hat mehrere Gründe.

Zum einen ist die übersichtliche Strukturierung des Codes besser, wenn diese Aufgabe in einem eigenen Projekt (Projekt hier: Zusammenstellung von Java-Code) organisiert ist.

Weiterhin ist es angedacht, das komprimierte Kartenmaterial auch in weiteren Projekten, die mit Vektorkarten arbeiten müssen, zu verwenden – auch ohne die Bindung an die Visualisierung, für die das Lademodul innerhalb dieser Arbeit benötigt wird.

Als alleinstehendes Modul ist die Ladeschnittstelle wiederverwendbar. Sie ist so angelegt, dass sie mit einem Minimum an Aufwand in Folgeprojekte eingebunden werden kann.

Außerdem ist das Modul so angelegt, dass es über mehrere Varianten des Kartenladens verfügen kann. Erreicht wird dies über den Einsatz von Interfaces. Das hat seinen Ursprung in möglichen Arten, wie Kartenmaterial geladen werden soll. Verdeutlicht werden kann das an einigen Beispielen:

- Für das Laden des Kartenmaterials als Hintergrundprozess für eine Darstellung auf einem beliebigen Endgerät mit geringer Speicherplatzanforderung empfiehlt es sich, das Laden aus ZIP-Dateien in einem eigenen Thread zu organisieren. Die Übergabe erfolgt direkt über Objektreferenzen bzw. Methoden.
- In einem Java-Applet, dass das Material beim Initialisieren nicht gleich mit lädt, kann es notwendig werden, Material aus ZIP-Dateien von einem Server auf einen Client (hier: das Applet) zu übertragen, möglichst in einem eigenen Thread.
- Ein schnelleres Laden auf einem beliebigen Endgerät zur sofortigen Verarbeitung erfordert gegebenenfalls das Laden aus unkomprimiertem Material im gleichen Thread, also blockierend.

Implementiert wurde das Laden ohne einen seperaten Thread aus ZIP-Dateien, für eine

Darstellung des Materials. Anforderung war es aber auch, den Code von vornherein möglichst so zu gestalten, dass andere Methoden möglichst einfach implementiert werden können.

Der Funktionsumfang umfasst das Laden eines Kartenausschnittes nach Angabe der Koordinatenintervalle in x- und y-Richtung. Zusätzliche Angaben sind das Laden eines maximalen Straßenlevels (Nur Autobahn, Autobahn-Bundesstraßen, Autobahn-Nebenstraßen ect.) und das optionale Ignorieren von Zwischenknoten (die nur für die detaillierte Darstellung benötigt werden, aber beim Laden größerer Kartenausschnitte Zeit und Speicher verbrauchen). Das Kartenlademodul ist so angelegt, dass es das benötigte Kartenmaterial nicht nur lädt, sondern auch im Speicher hält. Um nicht (mehr) benötigtes Kartenmaterial löschen und so den Speicher freigeben zu können, gibt es eine Methode „cleanup“, die alles Material außerhalb eines bestimmten Bereiches löscht. Zur Verfügung gestellt werden Straßenkanten, Knoten und Straßennamen in Java-Hashtables.

Implementiert sind außerdem die Klassen für Knoten und Kanten, die nach dem Laden in den Hashtables vorliegen.

4.3 Kartendarstellung

4.3.1 Anforderungen

Die Kartendarstellung dient innerhalb dieser Arbeit zur Visualisierung von Kartenmaterial, sowie zur Visualisierung der identifizierten Knotenpunkte. Damit ist sie Kernbestandteil der Arbeit. Es ist die Grundlage für das Visualisieren von bestimmten Knotenpunkten in Straßennetzen, diese Netze übersichtlich darzustellen.

Das Kartendarstellungsmodul ist aber nicht nur im Rahmen dieser Arbeit wichtig. Vielmehr soll nach der Arbeit für weitere Visualisierungsaufgaben in der Verkehrsforschung durch DLR-TS ein Tool vorhanden sein, welches imstande ist, Kartenausschnitte mit Hilfe von Vektorgrafiken darzustellen. Bislang gab es im Institut nur die Darstellung mit Hilfe von Rasterkarten, oder aber eine rudimentäre Darstellung von Vektorkarten, welche aufgrund des Ladens von kompletten Gebieten in den Speicher schlecht skalierbar ist. Zudem ist die bisherige Darstellung sehr mit dem Code des nutzenden Programms verwoben und von daher nicht ohne weiteres eigenständig nutzbar.

Daraus, dass die Kartendarstellung möglichst allgemein funktionieren soll, ergeben sich die Anforderungen und das Konzept für die Umsetzung der Kartendarstellung.

Zunächst muss das Modul gekapselt sein, so dass es einfach in andere Projekte einzubinden ist, ohne den Quellcode verändern zu müssen.

Zum anderen muss das Modul einen möglichst großen Funktionsumfang haben, der vom einbindenden Projekt gut steuerbar ist. Dazu gehören Funktionen wie:

- Festlegung der Farbe, der Fahrbahnbreite und des Rahmens für Straßen

- Ein- und auszoomen sowie scrollen in bestimmte Richtungen
- Einfärben bestimmter Knoten und Kanten
- Wahl des Gebietes für die Darstellung

Dazu kommen Anforderungen wie Speicher- und Zeiteffizienz, so dass die Darstellung komfortabel verwendbar ist.

4.3.2 Grundkonzept

Die Kapselung des Moduls zu erreichen ist mit Vererbung gelöst. Indem die Hauptklasse eine von JPanel erbende Klasse ist, lässt sich das Modul einfach in jeder Swing-GUI einbinden. In dieser Klasse werden nach außen hin die notwendigen Methoden zur Verfügung gestellt.

Interessant sind die Fragen der Performance. Das Zeichnen von einem Kartenausschnitt mit sehr vielen Straßenkanten bringt einen hohen Rechenaufwand mit sich. Es sind für jede Kante mehrere Schritte zum Zeichnen notwendig, wie das Umrechnen der Geokoordinaten in Pixelkoordinaten, das Bestimmen der Zugehörigkeiten zwischen Kanten und Knoten und das Zeichnen der Polygone, die eine Kante darstellen.

Vom Speicher her ist die Darstellung ebenfalls kritisch. Das Darstellen von relativ großen, detaillierten Kartenausschnitten benötigt schnell einen Speicher von mehr als 100 MB. Das ist eine Größenordnung, die die Voreinstellung der Java-Virtual Machine überschreitet. Dies hat mehrere Gründe.

Zum einen ist die Anzahl von Straßenkanten in einem Kartenausschnitt wie Berlin oder Nürnberg, also allgemein in Stadtgebieten, sehr hoch, wenn alle Straßenlevel, inklusive der kleinen Nebenstraßen, dargestellt werden. Zum anderen haben Java-Objekte einen Java-spezifischen Overhead (siehe Kapitel 3.6). Bei der Repräsentation von Knoten und Kanten in Objekten entsteht so schnell ein sehr hoher Speicherbedarf.

Um dies gering zu halten, gibt es den Ansatz, bei bestimmten Zoomstufen nur bestimmte Straßenlevel anzuzeigen. Stellt man einen sehr großen Kartenausschnitt dar (Beispiel: ganz Berlin und ein wenig der Umgebung, also ca. 50 * 50 Kilometer, oder mehr), so ist es beispielsweise nur erforderlich, die Autobahnen darzustellen. Höhere Straßenlevel würden zur Unübersichtlichkeit der Karte führen und sind bei solch hohen Zoomstufen für den Betrachter in der Regel irrelevant.

Das Kartendarstellungsmodul ist demzufolge so aufgebaut, dass der Benutzer angeben kann, ab welcher Zoomstufe welches Straßenlevel irrelevant ist und nicht angezeigt werden soll. Die Zoomstufe ist hierbei die Einheit Pixel/Meter.

Weiterhin führt das Weglassen von Zwischenknoten zur Steigerung der Speichereffizienz. Erst bei höheren Zoomstufen bekommen diese Knoten eine Bedeutung. Wird ein großer Kartenausschnitt angezeigt, sind beispielsweise Kurven von Autobahnabfahrten ohnehin

nicht erkennbar, da sie oft kleiner als ein einzelner Pixel sind. So kann mit Hilfe der entsprechenden Funktionalität im Lademodul eingestellt werden, ab welcher Zoomstufe Zwischenknoten nicht geladen und gespeichert werden sollen.

Wenn der Benutzer den Kartenausschnitt viel hin- und her bewegt oder viel ein- und auszoomt, entsteht schnell viel nicht benötigtes Kartenmaterial. Es ist daher sinnvoll, Kartenmaterial, welches weit ab vom angezeigten Kartenausschnitt liegt, aus dem Speicher zu entfernen. Hierzu wird die Methode „cleanup“ des Lademoduls in einer geeigneten Art und Weise verwendet. So werden alle Kartendaten, die Objekte außerhalb einer bestimmten Entfernung zum aktuellen Kartenausschnitt repräsentieren, aus dem Speicher entfernt.

Die Geschwindigkeit der Darstellung ist für den Benutzer ebenso von Bedeutung wie der Speicherplatzbedarf. Wenn bei jeder Anwendung vom Zoomen oder Scrollen ein so hoher Aufwand entsteht, dass der Benutzer dabei durch lange Wartezeiten behindert wird, ist dies schnell unzumutbar.

Daher ist es wichtig, das rechenzeitaufwändige Zeichnen von Kartenausschnitten und die Anzahl der Ladevoränge zu minimieren. Hierbei kommen mehrere Konzepte zum Tragen.

Das Zeichnen des Kartenausschnittes wird nur dann erledigt, wenn tatsächlich neues Kartenmaterial geladen wurde. Das Zeichnen erfolgt auf einem Zwischenbild, das, so lange kein neues Kartenmaterial geladen ist, vergrößert, verkleinert oder hin- und hergeschoben wird. Ist neues Material geladen, so wird das Bild mit dem aktuellen Material neu gezeichnet.

Das Laden von Kartenausschnitten ist so organisiert, dass das Modul maximal einen Ladevorgang gleichzeitig ausführt. Dies beseitigt den Konflikt mit Benutzereingaben von schneller Abfolge. Als Beispiel: Der Benutzer zoomt in schneller Abfolge 4 mal und scrollt danach 5 mal um 10% nach Norden. Theoretisch müsste das Kartenmaterial nun 9 mal neu geladen werden.

Das Lademodul startet den Vorgang jedoch nur bei der ersten Benutzereingabe. Kommt eine zweite Benutzereingabe hinzu, wird ein erneutes Laden hinten angestellt. Ist der erste Ladevorgang beendet, startet der zweite. Bei jeder folgenden Benutzereingabe werden nur noch die Ladeparameter für den zweiten, also den wartenden, Ladevorgang verändert. Erfolgen die 9 Wechsel des Kartenausschnittes aus dem Beispiel also tatsächlich während des ersten Ladevorganges, wird der zweite Ladevorgang mit den Parametern, die aus der letzten Benutzereingabe zustande kommen, ausgeführt. Zwischendurch angeforderte Ladevorgänge sind irrelevant und werden ignoriert.

Aus Gründen der Kompatibilität ist das Rendering mit Hilfe der Standardbibliotheken von Java implementiert. Alle Kanten werden nacheinander auf das Zwischenbild gezeichnet. Dabei werden Parameter wie Farbe und breite für die einzelnen Straßenlevels berücksichtigt.

5 Ermittlung von kritischen Knoten

Der Hauptteil der Arbeit ist die Identifikation von Verkehrsknoten, welche gestaute Verkehrsmuster aufweisen, mit Hilfe der „Floating Car Data“. Diese müssen in einer Art und Weise ausgewertet werden, die einen Rückschluss darauf zulässt, wie die Verkehrssituation an Knotenpunkten in einem Straßennetz aussieht.

Die grundsätzliche Herangehensweise an diese Aufgabe ist die Zuordnung von FCD-Positionen zu Knotenpunkten und die daraus folgende Kategorisierung in kritische und nicht kritische Knotenpunkte.

Geopositionen aus den FCD, die die Rohdaten darstellen, werden in den folgenden Ausführungen als „FCD-Positionen“ bezeichnet. FCD-Positionen, die bereits Knotenpunkten zugeordnet wurden, werden hier zur Unterscheidung „FCD-Treffer“ genannt.

5.1 Anforderungen und Vorbetrachtungen

Im Vordergrund der Entwicklung des Programms zur Identifikation von kritischen Verkehrsknoten steht der experimentelle Aspekt verschiedener Methoden. Da das Thema in dieser Art und Weise zunächst einmal neu ist und sich gegenüber bisher verwendeter Methoden zur Verkehrslageerfassung abgrenzt, war im Vorfeld nicht bekannt, welche Methoden wie gut und wie zuverlässig funktionieren.

Daraus ergibt sich, dass mehrere Methoden implementiert werden mussten, um Schritt für Schritt zu einem Ergebnis zu kommen, indem die Ausgaben der einzelnen Methoden analysiert und verglichen wurden.

Zum Teil sind die Methoden abhängig von den FCD, die für eine Auswertung zur Verfügung stehen. Diese Arbeit wurde am Beispiel der Stadt Nürnberg ausgearbeitet. Andere Städte stellen jedoch andere Positionsdaten zur Verfügung, was Unterschiede in deren Auswertung mit sich bringt.

Es ist zum Beispiel überall möglich, FCD Positionen Knoten zuzuordnen und aus zu zählen. Nicht überall jedoch stehen Informationen über die Geschwindigkeiten der Fahrzeuge bereit. Im Beispiel Nürnberg werden die Geschwindigkeiten direkt im GPS-Empfänger berechnet. Andere Städte stellen aber Geschwindigkeitsdaten zur Verfügung, die durch andere Verfahren berechnet werden und daher gegebenenfalls nicht für eine Betrachtung der Momentangeschwindigkeiten herangezogen werden können.

Auch die Behandlung von Fahrzeugidentifikationsnummern unterscheidet sich. In Nürnberg werden beispielsweise die Fahrzeugidentifikationsnummern regelmäßig gewechselt, allerdings werden Informationen über diese Wechsel aus Datenschutzgründen nicht zur Verfügung gestellt. Es ist also nichts über das Intervall der Wechsel oder Ähnliches bekannt.

Die Intervalle, in denen die Positionsdaten gesendet werden, können sich ebenfalls

unterscheiden. Von daher wird man mit einigen Auswertungsmethoden in jeder Stadt andere Ergebnisse erhalten. Diese Umstände müssen in der Herangehensweise an die Entwicklung möglicher Auswertungsmethoden beachtet werden.

Die Auswertung muss variabel sein, was nicht nur die Methodik der Auswertung, sondern auch das Laden der Daten betrifft. Zum einen soll die Anwendung auch außerhalb des DLR Netzwerkes, das Zugriff auf die Datenbank ermöglicht, ausführbar sein. Zum anderen sollen auch gegebenenfalls vor prozessierte Daten verwendet werden können.

Die Auswertung soll nach Eingabe einiger Parameter automatisiert erfolgen. Der Zeitraum, aus dem die Daten verwendet werden, soll hierbei für verschiedene Tests anpassbar sein.

Mit Hilfe der Kartendarstellung soll ein Überblick über ein Stadtgebiet gegeben werden. In der Karte dieses Gebiets soll eingezeichnet sein, wo eventuell kritische Knotenpunkt liegen; sie sollen also eingefärbt werden, je nach Grad des anzunehmenden Verkehrsproblems von Grün (kein Verkehrsproblem) über Gelb bis Rot (gestautes Verkehrsmuster).

Grundsätzlich gilt für den Rahmen dieser Arbeit, dass mit Daten aus der Stadt Nürnberg gearbeitet wird, so ist auch das Programm an diese angepasst.

5.2 Lösungsansätze

Um zunächst einen groben Ansatz für die Auswertung zu bekommen, kann die Aufgabe in zwei elementare Bereiche getrennt werden.

Der erste der beiden Bereiche ist die Zuordnung von FCD-Positionen zu den entsprechenden Knotenpunkten, zu denen sie „gehören“. Anders ausgedrückt: Wenn heraus gefunden werden soll, an welchen Kreuzungen in einem Straßennetz ein erhöhtes Verkehrsaufkommen verzeichnet werden kann, so muss zunächst analysiert werden, welche Positionen aus dem Datensatz der Floating Car Data welcher Kreuzung zuzuordnen sind.

Anhand dieser Zuordnung kann dann die Auswertung der Daten erfolgen. Der zweite Bereich hat die Aufgabe, mit Hilfe der FCD-Treffer für jeden Knotenpunkt zu ermitteln, ob dieser zu den kritischen Punkten gehört oder nicht, bzw. wie kritisch das Verkehrsmuster an diesem Knoten ist.

Die initiale Idee für diese Arbeit war das Rastern des Kartenmaterials und das Einteilen der FCD-Positionen in die einzelnen Kartenraster. Anschließend sollte zugeordnet werden, welche Knoten in jeweils einem Teil des Rasters liegen und wie viele Treffer ihnen somit zugeordnet wurden, als Maß für das Verkehrsaufkommen. Aus Betrachtungen dieses Verfahrens entwickelten sich weitere Ideen und Schritten für die Analyse.

5.2.1 Zuordnung von FCD-Positionen zu Knotenpunkten

Im folgenden Abschnitt wird zunächst erläutert, welche Methoden implementiert wurden, um FCD-Positionen zu Knotenpunkten im Kartenmaterial zuzuordnen. Dabei führten die

Erkenntnisse aus jeder Methode zur Entwicklung der jeweils nächsten, da sich durch einfache Beobachtungen Schwächen zeigten.

5.2.1.1 Zuordnung der FCD-Positionen durch einfaches Rastern

Die Zuordnung von FCD-Positionen aus der ersten Idee ist zunächst eine sehr einfache Methode. Das Gebiet der Stadt Nürnberg wurde dazu in annähernd gleich große Teile unterteilt, indem die Fläche gerastert wurde. Diese Rasterung bedeutet eine Unterteilung des gesamten Stadtgebiets in Quadrate mit einer Kantenlänge von 50 Metern. 50 Meter sind dabei eine geographische Ausdehnung, in die in etwa eine Kreuzung passt, die jedoch so klein gewählt ist, dass nur in wenigen Fällen mehrere Kreuzungen ins gleiche Rasterquadrat fallen (siehe Kapitel 3.1). Mit Hilfe dieses Rasters kann nun ermittelt werden, in welches Teil des Rasters eine FCD-Position fällt. Somit können alle FC-Daten des Untersuchungszeitraumes in ihr Teil des Rasters geordnet werden. Anschließend wird ermittelt, welche Knotenpunkte sich in diesem Raster befinden. Jedem Knotenpunkt aus dem Rasterteil werden nun alle FCD-Positionen des selben Rasterteils zugeordnet.

Theoretisch ist diese Methode jedoch ungenau. Sie hat bestimmte konzeptionelle Nachteile. Zum einen ist es schwer, die geographische Ausdehnung festzulegen, mit der die Größe des Rasters sinnvoll festgelegt werden kann. 50 Meter sind hierbei nur ein Wert, der so gewählt wurde, dass eine große Kreuzung mit einem Rasterteil abgedeckt werden kann, jedoch ein Rasterteil nur selten mehr als eine Kreuzung umfasst. Allerdings ist solch ein Wert sehr schwer festzulegen, da Kreuzungen und andere knotenartige Strukturen in Straßennetzen mit sehr unterschiedlichen Ausdehnungen sowie in sehr unterschiedlicher Nähe zueinander auftreten können.

Hieraus ergibt sich gleich das nächste Problem, dargestellt in folgender Grafik:

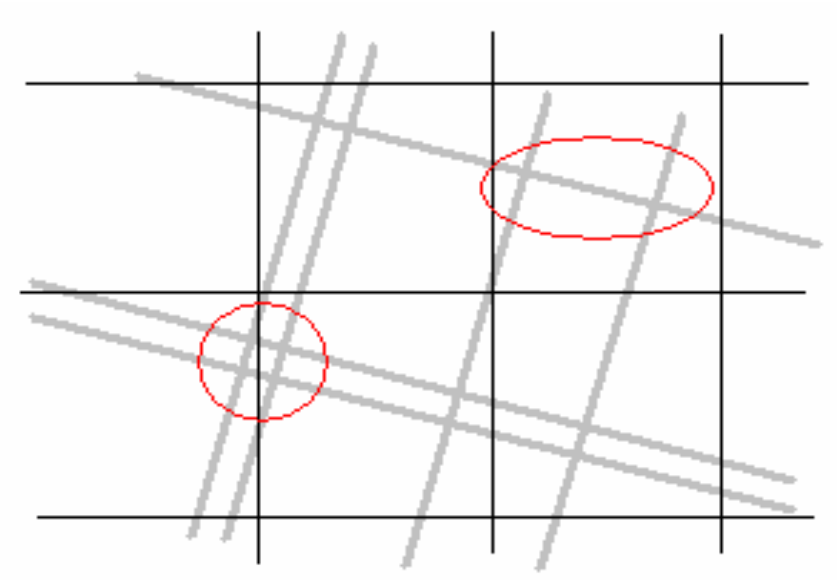

Abbildung 16: Theoretisch auftretende Zuordnungsprobleme. Grau: Straßen, Schwarz: Raster, Rot: Problemkreuzungen

Ein gleichmäßiges Raster als solches kann niemals so gewählt werden, dass die einzelnen Rasterteile zuverlässig Kreuzungen umfassen. Es kann daher auftreten, dass entweder die Grenze zwischen 2 Rasterteilen genau durch eine Kreuzung verläuft (in der Grafik links dargestellt, hier eine Kreuzung zwischen 2 Straßen mit baulicher Trennung), oder 2 Kreuzungen in ein Rasterteil fallen (in der Grafik rechts dargestellt). Von daher ist diese Methode sehr ungenau und unzuverlässig.

5.2.1.2 Zuordnung aller FCD-Positionen in Knotennähe

Um die falsche Zuordnung der Potitionen zu Kreuzungen zu umgehen, die in der Rastermethode auftaucht, gibt es die Möglichkeit, für jeden Knotenpunkt alle FCD-Positionen zu betrachten, die in der Nähe des Knotens liegen. Diese Zuordnungsmethode hat einen Hintergrund: Wenn man eine Kreuzung betrachtet, und deren Verkehrsaufkommen beurteilen möchte, bietet es sich an, alle FCD Positionen in ihrer Umgebung zuzuordnen. Somit können Staus erfasst werden, die nicht direkt am Knotenpunkt, sondern in seiner Umgebung auftauchen, also zum Beispiel auf zuführenden Kanten. Als „Umgebung“ wird hier ein vordefinierter Wert bezeichnet, der den Radius eines Umkreises um den entsprechenden Punkt darstellt. In der Umsetzung ist dies der Radius von 100 Metern, als geschätzter Wert. Dieser ist so groß gewählt, dass auch bei einem längeren Stau noch viele FCD-Positionen zum entsprechenden Knoten zugeordnet werden, nach Möglichkeit aber nur die Positionen, die auch tatsächlich zur Kreuzung gehören.

Diese Methode bietet mehrere Vorteile. Zum einen gibt es eine Ungenauigkeit von GPS-Empfängern. Bei einem Umkreis um einen Knoten fällt jedoch eine etwaige Ungenauigkeit einer FCD-Position nicht so schwer ins Gewicht, da die Zuordnung eher grob denn genau erfolgt. Zum anderen haben auch Fahrzeuge, die nicht direkt am eigentlichen Knoten einen FCD-Treffer erzeugen, unter Umständen einen Einfluss auf diesen Knoten. Dies geschieht beispielsweise, wenn sich 2 Kreuzungen in unmittelbarer Nähe befinden, und der Rückstau der einen Kreuzung so groß ist, dass er einen Stau an der vorherigen Kreuzung verursacht. Die Frage ist in diesem Falle jedoch, ob beide Knotenpunkte als „kritisch“ einzuordnen sind, oder ob eigentlich nur der Knoten, der den Stau verursacht, als kritisch einzustufen ist.

Das eigentliche Problem dieser Methode lässt sich erkennen, indem man mit dieser Zuordnungsmethode FCD-Positionen von Taxis auswertet, die an Taxiständen stehen:

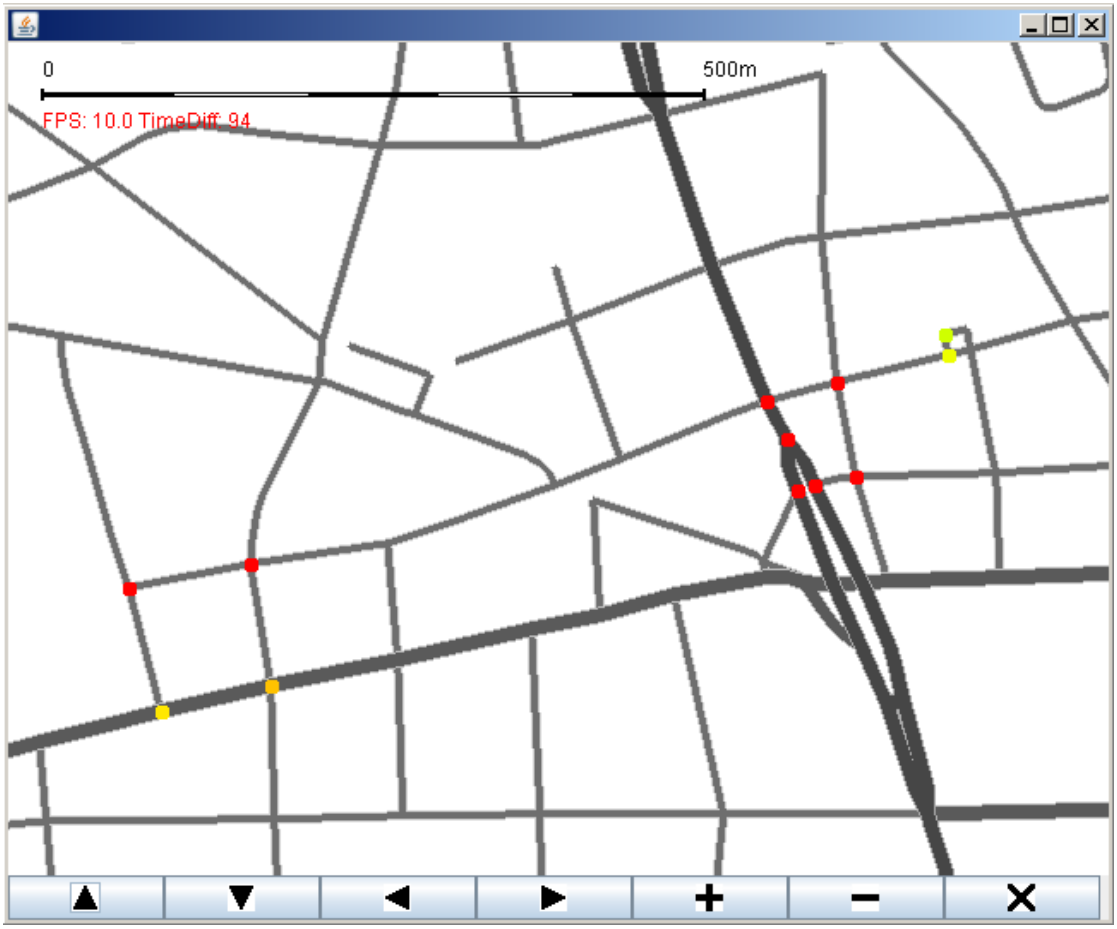

Abbildung 17: Ausschnitt Nürnberg, wartende Taxis. Rot: Viele Positionen, Grün: wenige Positionen

In diesem Beispiel wurden nur FCD Positionen ausgewertet, die den Status „wartend am Halteplatz“ haben. Es sind also nur Positionen von Taxis zu sehen, die an einem Taxistand stehen. Unabhängig vom Maß der Auswertung bedeutet ein roter Punkt, dass viele Treffer gezählt wurden, und ein grüner Punkt, dass wenige Treffer vorhanden sind.

Hierbei ist gut zu sehen, dass durch diese Auswertungsmethode auch Kreuzungen FCD-Treffer zugeordnet bekommen, die mit dem eigentlichen Knoten nichts zu tun haben. Bei einem Vergleich mit der im Bild östlich dargestellten Kreuzung mit dem gleichen Ausschnitt von Google Earth wird dies besonders deutlich:

Abbildung 18: Kartenausschnitt aus Nürnberg von Google Earth, Stand 01.12.2008. Rot: Taxistand mit wartenden Taxis [31]

Im Bild zu sehen sind wartende Taxis. Daraus wird deutlich, wo sich der Taxistand in der Realität befindet. Im Vergleich mit dem Screenshot aus der Auswertung wird deutlich, dass die wartenden Taxis den falschen Knotenpunkten zugeordnet sind.

Dies macht besonders deutlich, dass diese Zuordnungsmethode mit hoher Wahrscheinlichkeit dazu führt, dass auch falsche Knotenpunkte als „kritisch“ identifiziert werden.

5.2.1.3 Zuordnung aller FCD-Positionen zum nächstgelegenen Knoten

Die Ungenauigkeit der zweiten Methode zeigt, dass eine Zuordnung gefunden werden muss, die jede FCD-Position nur genau einem Knotenpunkt zuordnet, und dies möglichst genau. Damit fallen FCD-Positionen, die auf mehrere Kreuzungen übergreifenden Rückstau schließen lassen, für diesen nicht mehr ins Gewicht. Es lässt sich so besser darauf schließen, an welchen Knotenpunkten Probleme auftreten, nicht aber jedoch, welche Knotenpunkte für den Rückstau verantwortlich sind.

Ohne Kanten und Fahrtrichtungen zu kennen, geht dies am besten, wenn man für jede FCD-Position den Knotenpunkt sucht, der ihr geographisch am nächsten liegt. Diese Methode

ordnet die FCD-Treffer den Kreuzungen so zu, dass dies äquivalent zu einem Voronoi-Diagramm geschieht (siehe Kapitel 3.4). Folgende Abbildung zeigt skizzenhaft, wie FCD-Positionen den Knotenpunkten zugeordnet werden:

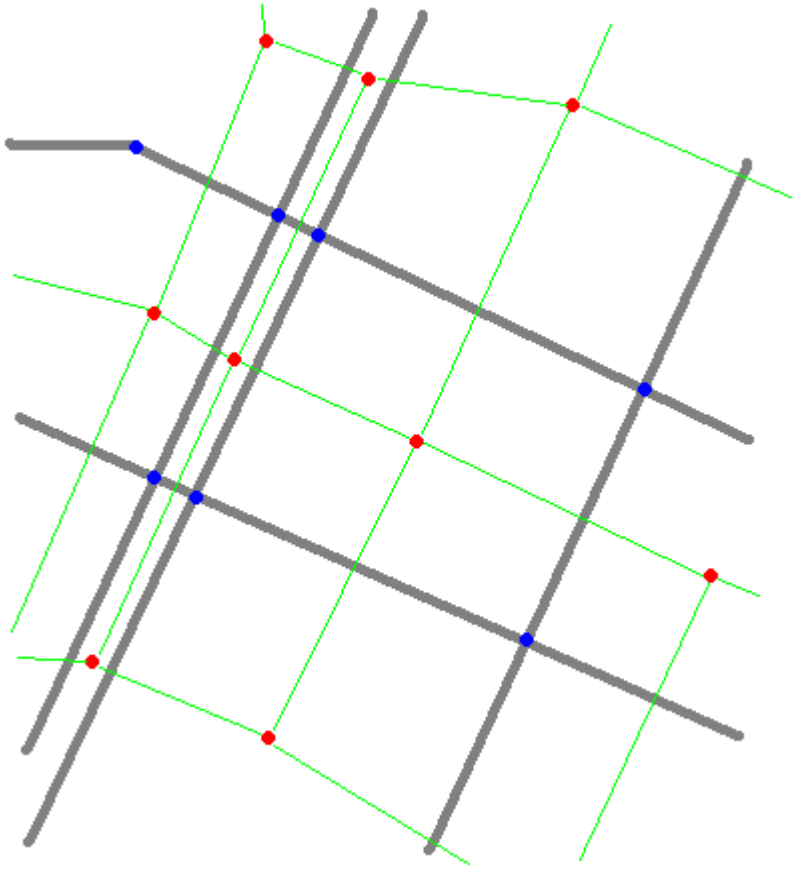

Abbildung 19: Skizze für die Zuordnung von Positionen mittels Voronoi-Diagramm. Grau: Straßenkanten, Blau: Straßenknoten, Rot: Voronoi-Knoten, Grün: Voronoi-Kanten

Diese Skizze zeigt beispielhaft Straßenkanten und -knoten, einen kleinen Ausschnitt aus einem Straßennetz. Außerhalb liegende Knoten werden nur beispielhaft angenommen. Knoten sind blau gefärbt, Straßen grau.

Über dieses Straßennetz lässt sich theoretisch ein Voronoi-Diagramm bilden. Dieses wird in der Grafik durch rote Punkte als Voronoi-Knoten und grüne Linien als Voronoi-Kanten dargestellt. Voronoi-Kanten spannen hierbei Voronoi-Ebenen auf.

Das Voronoi-Diagramm als solches ist nicht implementiert. Es kann aber eines verdeutlichen: Eine Voronoi-Ebene markiert das Gebiet, in dem alle möglichen in dem Gebiet liegenden Punkte demjenigen Straßenknoten zugeordnet werden, der ihnen am nächsten liegt.

Der Vorteil dieser Zuordnung ist am besten an der beispielhaften Straße mit baulicher

Trennung zu erkennen. Durch die Zuordnung jeder FCD-Position zum nächstgelegenen Knotenpunkt werden bei Straßen, die ähnlich dem Beispiel sind, sogar die Fahrtrichtungen mit berücksichtigt. Dies geht natürlich davon aus, dass die FCD-Positionen genau sind. GPS-Ungenauigkeit kann hier zu einem falschen Ergebnis führen, wenn eine FCD-Position in die falsche Voronoi-Ebene fällt. Handelt es sich, wie im Beispiel, um eine Straße mit baulicher Trennung, wird im Kartenmaterial eine Kreuzung durch 2 verschiedene Knotenpunkte dargestellt. Jeder dieser Punkte hat seine eigene Voronoi-Ebene, die ihre die entsprechende Straßenkante einschließt. Problematisch wird diese Zuordnung allerdings, wenn eine Straße mit baulicher Trennung eine Kurve beinhaltet. Durch die Nicht-Betrachtung von Straßenkanten ist dieser Fehler allerdings nicht auszuschließen.

Die Voronoi-Ebene schließt nicht nur die zum Knoten führende Kante, sondern auch die vom Knoten ausgehende Kante mit ein. Allerdings ist im Falle einer Kreuzung mit Lichtsignalanlage die Wahrscheinlichkeit einer FCD-Position auf der zuführenden Seite wahrscheinlicher, da Fahrzeuge hier stehen, und auf der abfließenden Kante in Bewegung sind. Eine Ausnahme ist ein Rückstau vom nächsten Knotenpunkt.

Vom theoretischen Ansatz her ist diese Methode die am besten geeignete, die im Rahmen dieser Masterarbeit realisiert wurde.

5.2.2 Auswertung von Knotenpunkten mit FCD-Treffern

Der zweite Teil der eigentlichen Auswertung ist die Bestimmung der kritischen Knotenpunkte, nachdem die FCD-Positionen als FCD-Treffer den einzelnen Kreuzungen zugeordnet wurden. Auch hier wurden mehrere Methoden implementiert, wobei die Erkenntnisse aus dem Ergebnis einer Methode zur Entwicklung der jeweils nächsten geführt haben.

5.2.2.1 Gemeinsame Parameter der Auswertungsmethoden

Um die Ergebnisse vergleichen zu können, wurden mit allen implementierten Methoden beispielhaft Auswertungen durchgeführt. Diese sind hier dokumentiert. Alle diese Auswertungen haben die gleichen Parameter und Randbedingungen, welche hier zunächst kurz erläutert werden.

In der Umsetzung haben alle Methoden gemeinsam, dass sie für jeden Knotenpunkt einen Wert von 0 bis 100 liefern. Dieser Wert wird mittels einer Farbskala von Grün (0) über gelb (50) bis Rot (100) im Kartenausschnitt dargestellt. Er gibt einen Prozentsatz an, wie „kritisch“ ein Knotenpunkt ist. Grundsätzlich gilt: je höher der Wert, desto eher ist eine kritische Verkehrssituation anzunehmen. Das Wort „kritisch“ hat in diesem Kontext die Bedeutung, dass laut Ergebnis einer Auswertungsmethode ein gestautes Verkehrsmuster an einem Knotenpunkt zu finden ist. In allen Methoden wird zudem ein Prozentsatz, der als gering einzustufen ist (beispielsweise 5% oder niedriger), ignoriert. Dies dient dazu, dass keine Knoten gefärbt werden, die sehr geringfügige, also „unkritische“ Werte aufweisen, sondern nur diejenigen, die als kritisch einzustufen sind, bzw. sich in der Nähe vom Zustand „kritisch“

befinden. Die Farbwerte werden von den Prozentwerten in das RGB-Modell umgerechnet. Bei 0%-50% „kritisch“ haben die Farben für rot, grün und blau Werte von 0-255, 255 und 0, bei 51%-100% „kritisch“ sind es 255, 255-0 und 0.

Grundsätzlich werden somit nur Knoten gefärbt, deren Betrachtung relevant ist. Die Skala von 0 bis 100 liefert dabei eine ungefähre Einteilung, wie stark das Verkehrsproblem am entsprechenden Knoten ist.

Anstelle einer Legende wird bei jeder Auswertungsmethode mindestens ein Fenster geöffnet, in dem ein Histogramm dargestellt wird. Unterschiedliche Auswertungsmethoden zeigen dabei unterschiedliche Histogramme. Für alle Histogramme ist die Einteilung in Klassen gleich. Histogrammklassen sind ein Prozent des maximalen, also kritischen, Wertes groß. Das bedeutet, dass für die Auswertung mittels eines Histogrammes die Ergebnismenge der Auswertung in 100 Klassen unterteilt wurde. Die Klassen werden in der x-Achse dargestellt und zeigen die Wertung der Knoten von „unkritisch“ bis „kritisch“. In der y-Achse wird die Anzahl der Knoten dargestellt, die eine Wertung bekommen haben, welche der entsprechenden Klasse zugeteilt wurde. Der Definitions- und Wertebereich eines Histogrammes wird automatisiert aus dem maximalen festgestellten Wert ermittelt. Die Beschriftung der Achsen erfolgt in den Grafiken in 25%-Schritten. Daher kann es vorkommen, dass Werte wie „15,75 Knoten“ auftauchen, die so aber nicht existent sein können. Dies beeinflusst jedoch nicht das Ergebnis, sondern nur die Darstellung der Achsenbeschriftung.

In den Histogrammen werden automatisch die Werte des arithmetischen Mittels und des Medians der Ergebnismenge berechnet. Das arithmetische Mittel wird aus den Eingangswerten berechnet, welche reelle Zahlen sein können. Der Median hingegen wird nach der Einteilung in Klassen berechnet und ist daher gerundet.

Für die Auswertung wurden Daten aus der Stadt Nürnberg verwendet. Betrachtet wird ein Ausschnitt aus dem Gebiet rund um Nürnberg mit einer Größe von ca. 18 km * 20 km. Der Auswertungszeitraum wurde rein zufällig ausgewählt und beträgt eine Woche, vom 1.10.2008 um 0:00:01 Uhr bis zum 8.10.2008 um 0:00:01 Uhr. In diesem Zeitraum gingen insgesamt 1.044.764 Positionsdaten von Taxis in der Datenbank ein. Für die Auswertung genutzt wurden nur Daten von Fahrzeugen, die den Status „besetzt mit Fahrziel“ haben. Dies sind 123.823 Positionen. Alle anderen Einträge wurden verworfen.

Diese Rahmenbedingungen sind nur exemplarisch und spiegeln sich in den folgenden Betrachtungen wieder. Die Auswertung kann ebenso gut mit anderen Daten und Parametern durchgeführt werden, wurde aber hier angepasst, um die Methoden in der Entwicklung vergleichen zu können.

In den folgenden Betrachtungen wurde immer die Zuordnungsmethode „Zuordnung aller FCD zum nächstgelegenen Knoten“ verwendet, da davon auszugehen ist, dass dies die genaueste Zuordnung ermöglicht.

5.2.2.2 Zählen der FCD-Treffer und Normierung zum maximalen gefundenen Wert

Die erste Methode geht davon aus, dass an Kreuzungen, an denen ein kritisches Verkehrsaufkommen zu beobachten ist, viele FCD-Positionen zu finden sind. „Gestaute Verkehrsmuster“ sind in dieser Betrachtung solche Muster, bei denen sich an Knotenpunkten viele FCD-Positionen befinden und bei der reinen Betrachtung der Häufungen ein gestautes Muster erkennen lassen. Der Wert in der Praxis ist jedoch fraglich, da im praktischen Sinne ein gestautes Verkehrsmuster ein solches ist, dass sich viele Fahrzeuge in Knotenpunktnähe stauen.

Diese Methode entspricht der ursprünglichen Idee zu dieser Masterarbeit. Für jeden Knotenpunkt wird gezählt, wie viele FCD-Positionen ihm zugeordnet wurden. Als kritischer Wert wird derjenige Wert angenommen, der am höchsten ist. Hat beispielsweise der Knoten mit den meisten Treffern 50 zugeordnete Positionen, so erhöht jeder einem Knoten zugeordnete FCD-Treffer die Bewertung des Knotens um 2% auf der Skala von „unkritisch“ bis „kritisch“. Berechnet wird die Bewertung nach folgender Formel:

$$Bewertung = \frac{(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 100$$

Der absolute Wert der FCD-Treffer pro Knoten wird nicht betrachtet, da nur das Verhältnis der FCD-Treffer pro Knotenpunkt zueinander für eine Bewertung wichtig ist.

Die Schwäche dieser Methode zeigt sich bei einer Auswertung mit den beschriebenen Parametern, zu erkennen an folgendem Screenshot:

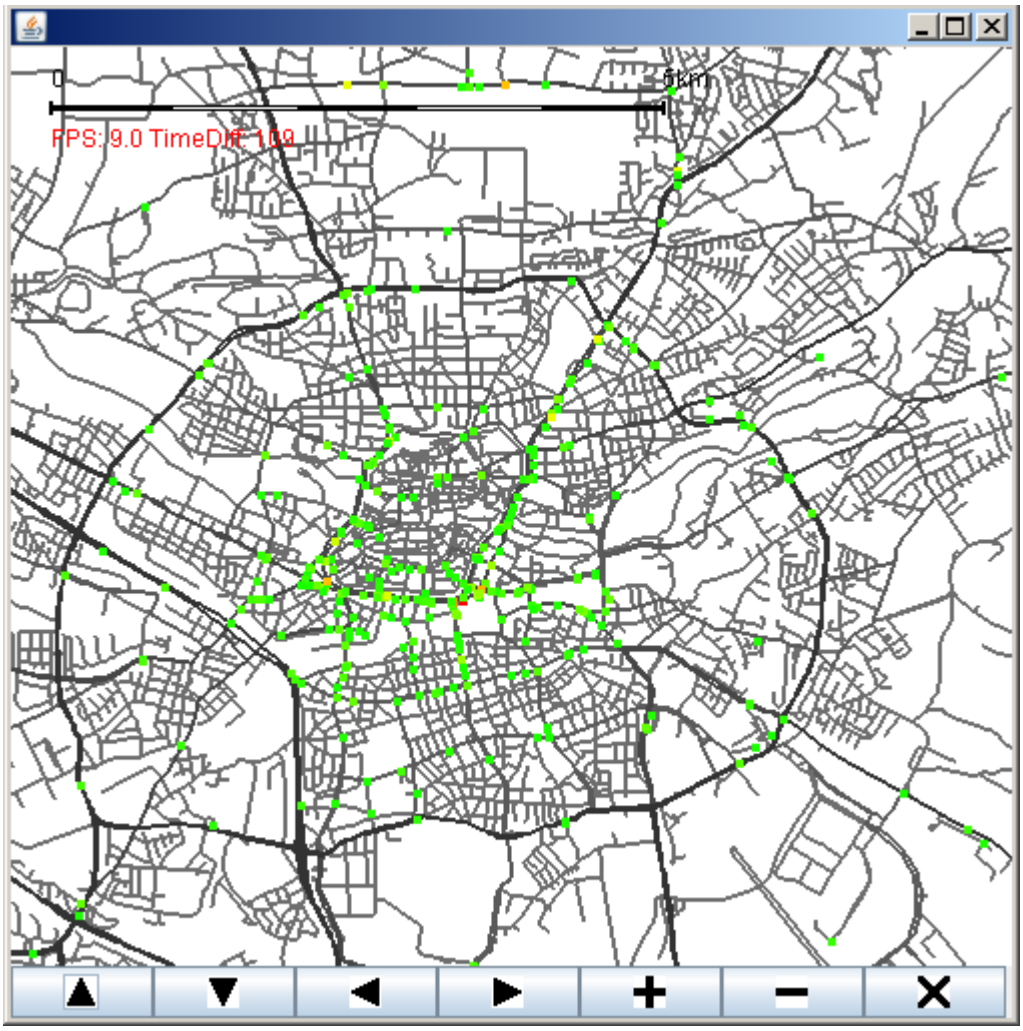

Abbildung 20: Screenshot der Software mit einem Ausschnitt aus Nürnberg. Kategorisierung der Knotenpunkte nach Auszählung der Treffer.

Es ist zu sehen, dass es viele Knotenpunkte gibt, an denen Treffer festgestellt wurden Es gibt nur wenige „Ausreißer“ mit vielen Treffern, aber viele Knoten mit wenigen Treffern. Deutlich wird dies vor allem auch, wenn man die Verteilung der Knotenbewertungen betrachtet. Die folgende Grafik zeigt die Verteilung der Knotenbewertungen des vorangegangenen Kartenausschnittes. Der größten Klasse des Histogramms bei ca. 6% des Maximalwertes an FCD-Treffern pro Knoten wurden 63 Knotenpunkte zugeordnet. Nur wenige Knoten haben über 50% der Maximalwertes, nur ein Knoten hat die vollständig kritische Bewertung von 100%. Knoten mit einer Bewertung unter 5% wurden nicht betrachtet:

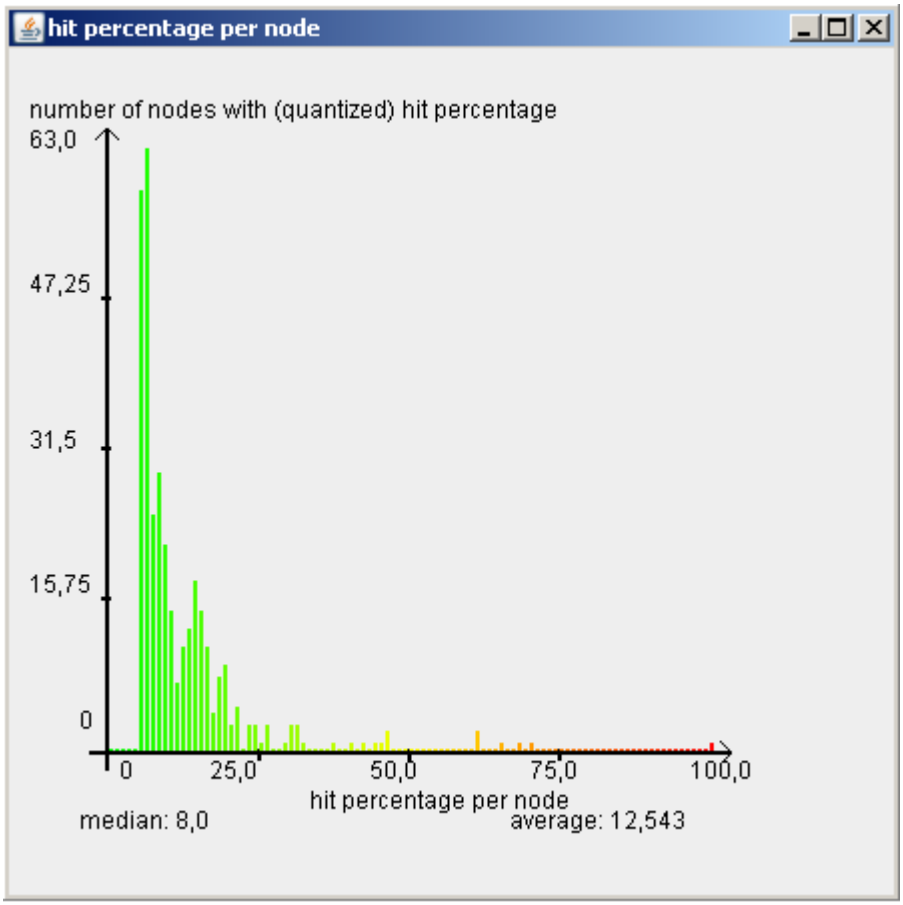

Abbildung 21: Verteilung der FCD-Treffer FCD-Treffer pro Knotenpunkt im Histogramm

Es ist zu erkennen, dass nur ein Knoten die 100 Prozent erhalten hat, die Mehrheit der Knoten jedoch als „unkritisch“ eingestuft wird. Dies lässt Schlüsse auf die Qualität dieser Methode zu. Deutlich ist auf jeden Fall, dass für eine Stadt von der Größe Nürnbergs zu wenige Knoten eine kritische Verkehrsbelastung aufweisen, als dass das Ergebnis realistisch erscheinen könnte.

Das erste Problem, dass dieses Ergebnis aufzeigt, ist dieser „Ausreißer“. Die Gründe für die Existenz eines Knotens, der so viel mehr FCD-Treffer aufweist als alle Anderen, ist im Nachhinein kaum nachvollziehbar. Es könnte sich hier um einen Taxistand oder Ähnliches handeln, oder ein sehr dicht befahrenes Gebiet. Das Problem an einem solchen „Ausreißer“ ist jedoch, dass damit die Bewertung von Knotenpunkten, die ebenfalls viele Treffer erhalten haben, abgewertet wird. Dies wird aus dem Median deutlich, der gerade bei 8% liegt, ebenso wie das arithmetische Mittel mit 12,5%. Der Großteil der Bewertungen liegt unter 25%. In der Kartendarstellung sind so geringe Werte kaum von Werten mit einer Bewertung von 0 zu unterscheiden, können aber im Bezug auf die Praxis durchaus eine Anzahl an FCD-Treffern haben, die sich in einer schlechten Verkehrssituation widerspiegeln.

Dies führt zu Anätzen für die Eliminierung solcher „Ausreißer“.

5.2.2.3 Zählen der FCD-Treffer und Normierung nach einem festgelegten Schwellwert

Die zweite Methode basiert auf der Idee, einen Schwellwert festzulegen, der den kritischen Wert repräsentiert. Die maximale Anzahl von FCD-Treffern pro Knotenpunkt, die die Einteilung in die Klassen definiert, wird damit verändert. „Kritischer“ Wert ist nun nicht mehr der Wert des Knotens mit den meisten Treffern, sondern ein bestimmter Prozentsatz dieses Wertes. In der Implementierung sind dies 25% des Maximalwertes. Vom Ablauf her arbeitet die Methode identisch zur Methode „Zählen der FCD-Treffer und Normierung zum maximalen gefundenen Wert“. Der Bewertungsmaßstab ist jedoch anders, was zu anderen Formeln, und in der Konsequenz auch zu einer anderen Einteilung in Klassen des Histogrammes führt. Das Spektrum der Bewertungen wird im Bereich zwischen einem Treffer pro Knotenpunkt und dem maximalen Wert (25% der maximalen Treffer pro Knoten) gespreizt. Die Formeln für diese Berechnungen sind:

Allgemein:

$$Bewertung = \frac{(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 100 * \frac{100}{Schwellwert}$$

beziehungsweise mit der Festlegung von 25% als Maximalwert:

$$Bewertung = \frac{(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 400 \quad .$$

Da somit Werte von bis zu 400% entstehen können, werden alle Werte über 100 auf 100 herunter gesetzt.

Mit dieser Methode kann ein „Ausreißer“ eliminiert werden. Die Verteilung stellt sich wieder in einem Histogramm dar. Eingangsparameter für diese Beispielauswertungen waren wieder die zuvor festgelegten Daten. Minimaler Schwellwert sind 10%, daher werden Knotenbewertungen kleiner als 2,5% des maximalen Wertes abgeschnitten.

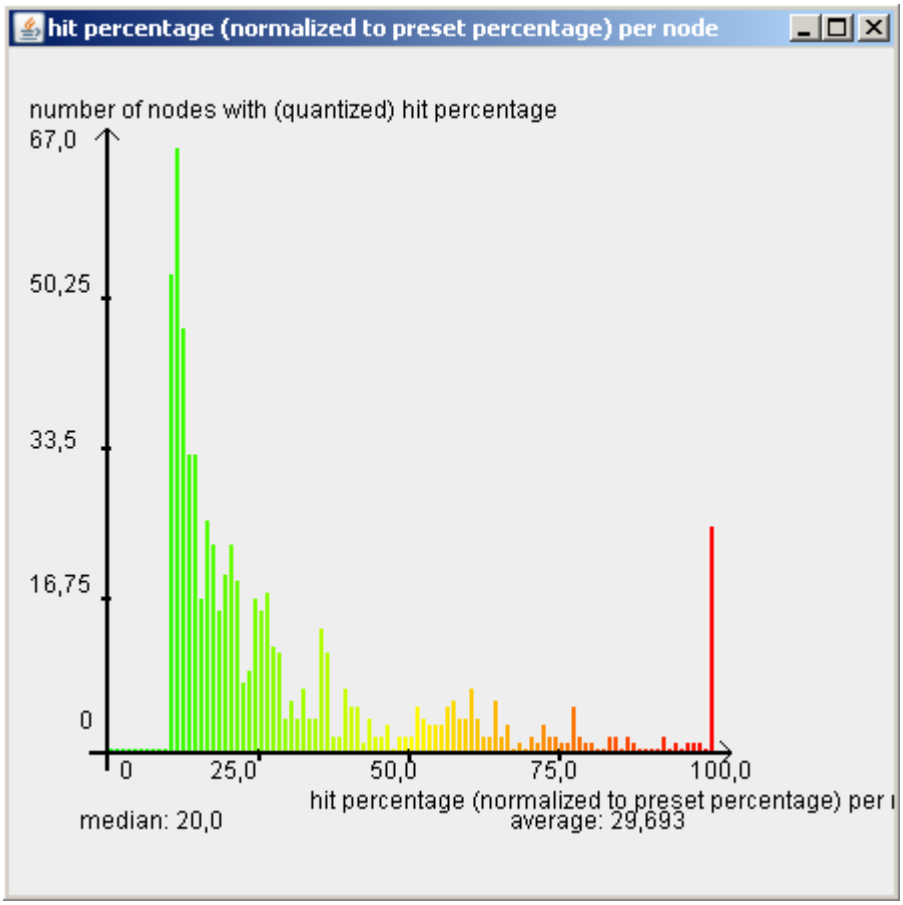

Abbildung 22: Verteilung der FCD-Treffer FCD-Treffer pro Knotenpunkt im Histogramm, mit 25% als maximalem Schwellwert

Zu sehen ist die Klasse bei ca. 11% des kritischen Wertes mit 67 Knotenpunkten als größte Klasse. Rund 25 Knotenpunkte haben die maximale Bewertung erhalten, gelten also als „vollständig kritisch“.

Die Ergebnismenge ist nun so verteilt, dass nicht nur ein einzelner Ausreißer, sondern viele Knoten, die in der Nähe des Schwellwertes oder darüber liegen, als „kritisch“ eingestuft werden. Im Bereich zwischen „kritisch“ und „unkritisch“ ist die Verteilung nun gleichmäßiger, daher werden Knoten mit vielen Treffern, die jedoch noch weit vom Maximum entfernt sind, differenzierter dargestellt.

Allerdings hat diese Methode im wesentlichen noch zwei Nachteile: Zum einen wird nun bei Knoten, die einen Wert von mehr als 25% der maximalen Treffer erhalten haben, nicht mehr differenziert, wie „kritisch“ sie sich zueinander verhalten. Zum zweiten sind diese 25% ein Wert, der sich aus der Verteilung der ersten Methode ergibt, jedoch keinen Anspruch auf Allgemeingültigkeit hat. Um ein praxisrelevantes Ergebnis zu erhalten, müsste also zunächst untersucht werden, welcher Prozentsatz hier einen Schwellwert darstellt, der sich für gute eine Spreizung eignet.

Dies lässt den Schluss zu, dass sich eine lineare Einteilung der Knotenbewertungen nur

bedingt für diese Aufgabe eignet.

5.2.2.4 Zählen der FCD-Treffer und logarithmische Einteilung

Ein mathematisches Verfahren, um „Ausreißer“ zu eliminieren, ist eine logarithmische Einteilung. Der Logarithmus bietet den Vorteil, dass die Einteilung nicht linear erfolgt, sondern hohe Werte dichter beieinander liegen, während niedrigere Werte gespreizt werden. Dies hat den Vorteil, dass sich im „kritischen“ Bereich die Knotenbewertungen, die im Verhältnis viele Treffer zugeordnet bekommen haben, sammeln. Deutlich wird das, wenn man den Logarithmus dualis aller FCD-Treffer pro Knoten in einem Histogramm darstellt:

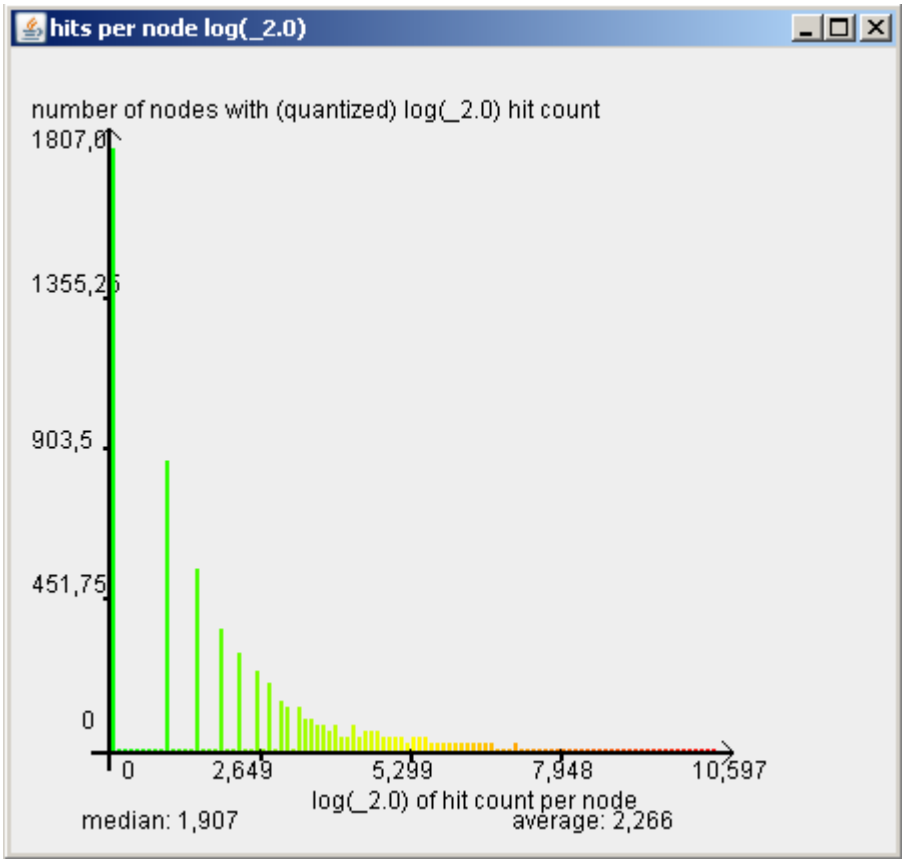

Abbildung 23: Verteilung der FCD-Treffer pro Knotenpunkt im Histogramm, logarithmisch eingeteilt

Das Histogramm enthält die logarithmierte Anzahl FCD-Treffer für alle Knoten im Straßennetz, denen mindestens 1 FCD-Treffer zugeordnet wurde:

$$Bewertung = \log_2(Anzahl\ der\ FCD-Treffer\ der\ Knoten)$$

Die Ausgangsdaten und Parameter sind wiederum die gleichen, festgelegten Daten wie bei allen anderen Methoden. In der x-Achse dargestellt ist der absolute Wert, den der Logarithmus dualis für jeden dieser Knoten ergibt. Der maximale Wert ist der Logarithmus des Knotens, der die meisten Treffer erhalten hat. Deutlich zu Erkennen sind die Spreizung

des unteren Bereiches sowie die Stauchung im Bereich mit vielen FCD-Treffern pro Knoten. Dies ist die Ausgangsbasis für die logarithmische Bewertungsmethode.

In dieser Methode werden die FCD-Treffer pro Knotenpunkt zunächst, wie in den vorigen Methoden auch, ausgezählt. Der maximale Wert, der einem Knoten zugeordnet wurde, wird zur Basis 2 logarithmiert und als Maximalwert angenommen. Die logarithmierten Werte der anderen Knoten werden im Verhältnis dazu gesetzt:

$$Bewertung = \frac{\log_2(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{\log_2(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 100$$

Es sind jedoch nicht alle Knoten von Relevanz, da durch das Logarithmieren der untere Bereich stark gespreizt ist. Das Einfärben aller Knoten würde daher unnötig zur Unübersichtlichkeit des Ergebnisses führen. Daher werden für die Darstellung die Werte unter 25% des Maximums der logarithmischen Werte ignoriert.

Das Ergebnis zeigt folgendes Histogramm:

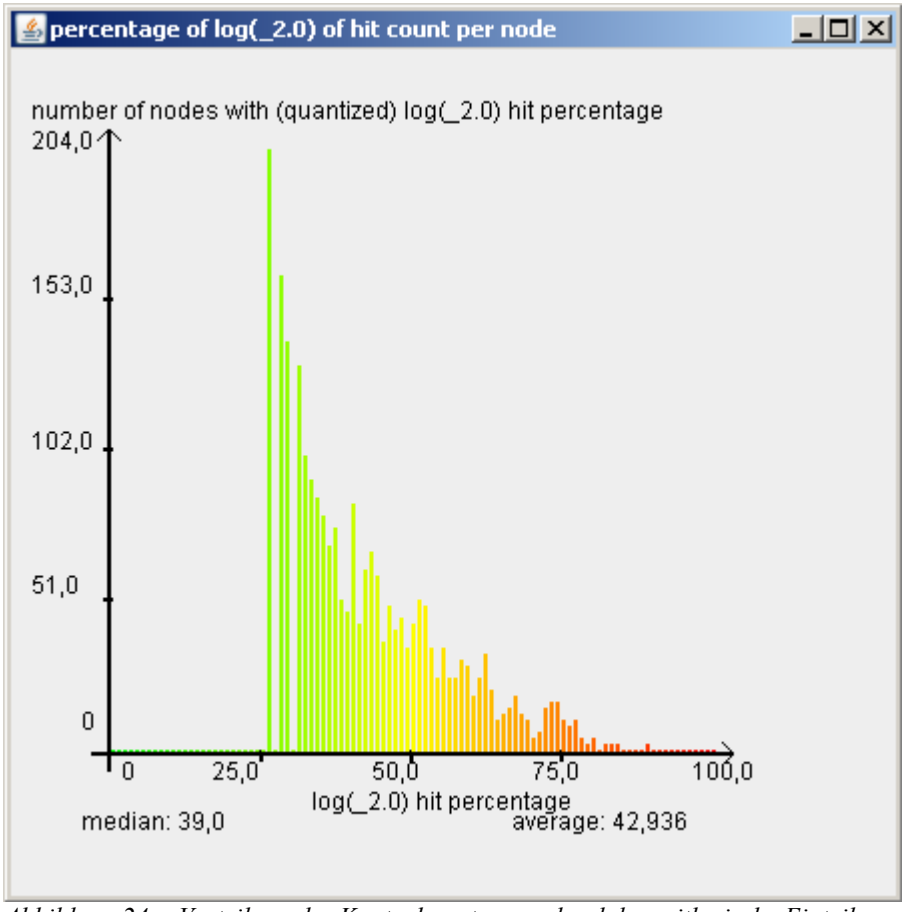

Abbildung 24: Verteilung der Knotenbewertungen durch logarithmische Einteilung im Histogramm

In dieser Grafik lässt sich erkennen, dass gegenüber der linearen Einteilung wesentlich mehr Knoten in den „kritischen“ Bereich fallen. Das Gefälle ist weniger stark als bei linearer

Einteilung, die Spreizung im unteren Bereich ist differenzierter und im „kritischen“ Bereich sind weniger Lücken zwischen den Klassen vorhanden. Die Beschriftung der Y-Achse zeigt, dass überhaupt mehr Knoten gefärbt wurden.

Interessant ist in diesem Zusammenhang auch die Darstellung dieser Auswertung in der Karte:

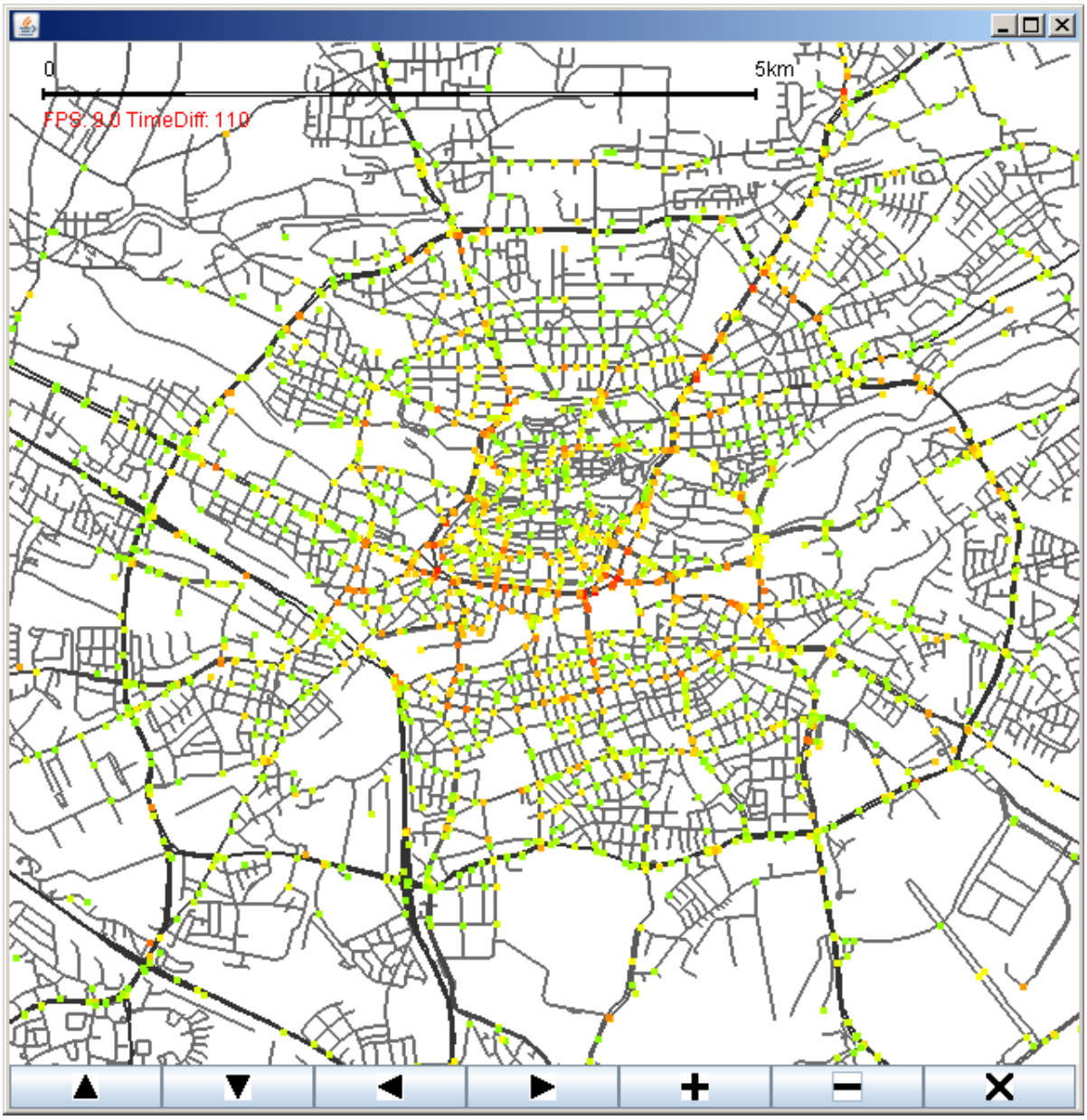

Abbildung 25: Screenshot der Software mit einem Ausschnitt aus Nürnberg. Kategorisierung der Knotenpunkte nach Auszählung der Treffer, logarithmisch skaliert.

Es ist deutlich zu sehen, dass viele Knoten „kritische“ Werte aufweisen, und diese vor Allem direkt in der Innenstadt und im Bereich großer Kreuzungen zu finden sind. Im Äußeren Stadtgebiet haben die Knoten eher als „unkritisch“ zu klassifizierende Werte.

Von den ersten 3 Methoden zeigt diese das Ergebnis, das am besten der Erwartung entspricht. Die logarithmische Einteilung scheint sich also tatsächlich zu eignen, um gestaute Verkehrsmuster zu ermitteln und darzustellen. Allerdings weisen die ersten 3 Methoden mehrere gemeinsame Schwächen auf.

Bei allen diesen Methoden wird davon ausgegangen, dass eine Ansammlung vieler FCD-Positionen auch ein kritisches Verkehrsaufkommen bedeutet. Es gibt jedoch keine Garantie dafür, dass das auch in der Realität zutrifft.

Die bisherige Aussage des Ergebnisses ist eigentlich nur, dass man feststellen kann, in der Nähe welcher Knotenpunkte besonders viele FCD-Positionen zu finden sind. Im Kontrast dazu stehen ein paar rein theoretische Überlegungen, was diese Positionen aussagen könnten.

Zum einen ist es möglich, dass es im Stadtgebiet besonders stark frequentierte Routen für Taxis gibt. Wenn eine Route sehr häufig befahren wurde, befinden sich in ihrer Umgebung viele FCD-Positionen. Dies heißt aber noch lange nicht, dass sich entlang dieser Route viele Knotenpunkte befinden, an denen es Verkehrsprobleme gibt. Als Beispiel kann hierzu die Umgebung eines Bahnhofes betrachtet werden. Viele Zugreisende fahren mit dem Taxi weiter zu ihrem Zielort, was eine hohe Dichte an Taxis in Bahnhofsnähe zur Folge hat.

Zudem ist es möglich, dass ein Taxifahrer mehrmals am Tag die selbe Route fährt, wodurch entlang dieser Route mehrere Positionen anfallen. Andere Routen werden gegebenenfalls aber sehr selten gefahren, obwohl sich dort vielleicht viele Kreuzungen mit hohen Verkehrsdichten befinden.

Diese Probleme zeigen, dass eine komplett andere Einordnung erforderlich untersucht werden sollte. Dies führt zu den Versuchen mit weiteren Bewertungsmethoden.

5.2.2.5 Auswertung der Durchschnittsgeschwindigkeiten

Ein Ansatz, der nicht direkt auf die Anzahl von FCD-Positionen eingeht, ist die Auswertung der Momentangeschwindigkeiten.

Dieser Ansatz ist nicht allgemein verwendbar. Es liegen dem DLR beispielsweise Datensätze aus Hamburg vor, die keine Geschwindigkeitsangaben enthalten. Auch in anderen Gebieten könnte dies der Fall sein. Von daher ist diese Methode nur als kleiner „Exkurs“ neben der eigentlichen Arbeit zu betrachten.

Für das betrachtete Gebiet Nürnberg können jedoch Geschwindigkeiten ausgewertet werden. Zu jeder vorhandenen FCD-Position gibt es eine Angabe, wie schnell das Fahrzeug, das die Position gesendet hat, zum Zeitpunkt des Sendens gefahren ist. Daraus ergibt sich die Möglichkeit, nachzuvollziehen, an welchen Knotenpunkten Fahrzeuge wie schnell gefahren sind.

Die Methode des Auswertens der Durchschnittsgeschwindigkeiten addiert die

Geschwindigkeiten aller FCD-Treffer, die einem Knotenpunkt zugeordnet wurden, und dividiert sie durch die Anzahl der zugeordneten Treffer. Das Ergebnis für jeden Knotenpunkt ist das arithmetische Mittel der Geschwindigkeiten:

$$\bar{v} = \frac{1}{n} \sum_{i=1}^{n} v_i$$

Mit n zugeordneten Positionen für einen Knotenpunkt, die die Geschwindigkeiten v_i enthalten.

Über den betrachteten Zeitraum werden also die Durchschnittsgeschwindigkeiten in Knotennähe betrachtet. Zur Ermittlung, welche Durchschnittsgeschwindigkeit als kritisch und welche Durchschnittsgeschwindigkeit als unkritisch einzustufen ist, werden die Werte 0 km/h (kritisch) und 15 km/h (unkritisch) verwendet. Alle Knoten mit Geschwindigkeiten über 15 km/h werden nicht eingefärbt.

Diese Einteilung erfolgt aufgrund folgender Betrachtung: Laut HBS ist die Wartezeit an Knotenpunkten mit Lichtsignalanlage unkritisch (Stufe C auf einer Skala von A bis F), wenn die mittlere Wartezeit kleiner als 50 Sekunden ist. In 50 Sekunden des Wartens werden im Schnitt eine bis zwei FCD-Positionen von einem Fahrzeug gesendet. Während des Wartens hat das Fahrzeug eine Geschwindigkeit von weniger als 1 km/h, im Datensatz in der Regel gerundet als 0 km/h angegeben. Durchfahrende Fahrzeuge haben eine Geschwindigkeit von mehr als 1 km/h. Bei koordinierter Zufahrt zum Knoten ist der Prozentsatz der Durchfahrten ohne Halt größer als 75%. Sowohl der Mittelwert als auch der Median der Geschwindigkeiten in den FCD-Positionen liegt in der Nähe von 33 km/h. Dadurch, dass wartende Fahrzeuge an Kreuzungen Positionen erzeugen, auch wenn die Knotenpunkte als unkritisch einzustufen sind, wird dieser Wert halbiert. Also werden Knotenpunkte, deren Durchschnittsgeschwindigkeiten kleiner sind als die Hälfte der gesamten Durchschnittsgeschwindigkeit, als kritisch eingestuft. Alle Knotenpunkte, deren Geschwindigkeiten höher sind, werden nicht betrachtet, um das Ergebnis auf das Wesentliche zu reduzieren.

Grundsätzlich sind diese Werte jedoch geschätzte Versuchswerte und keinesfalls als definitive Werte zu sehen, alleine, da sich Knotenpunkttypen und Art des zufließenden Verkehrs ohne die Betrachtung von Straßenkanten nicht ermitteln lassen.

Mit den gleichen Parametern wie bei den anderen Methoden wurde ein Test dieser Methode beispielhaft durchgeführt. Eine Darstellung des Ergebnisses in der Karte sieht aus wie folgt:

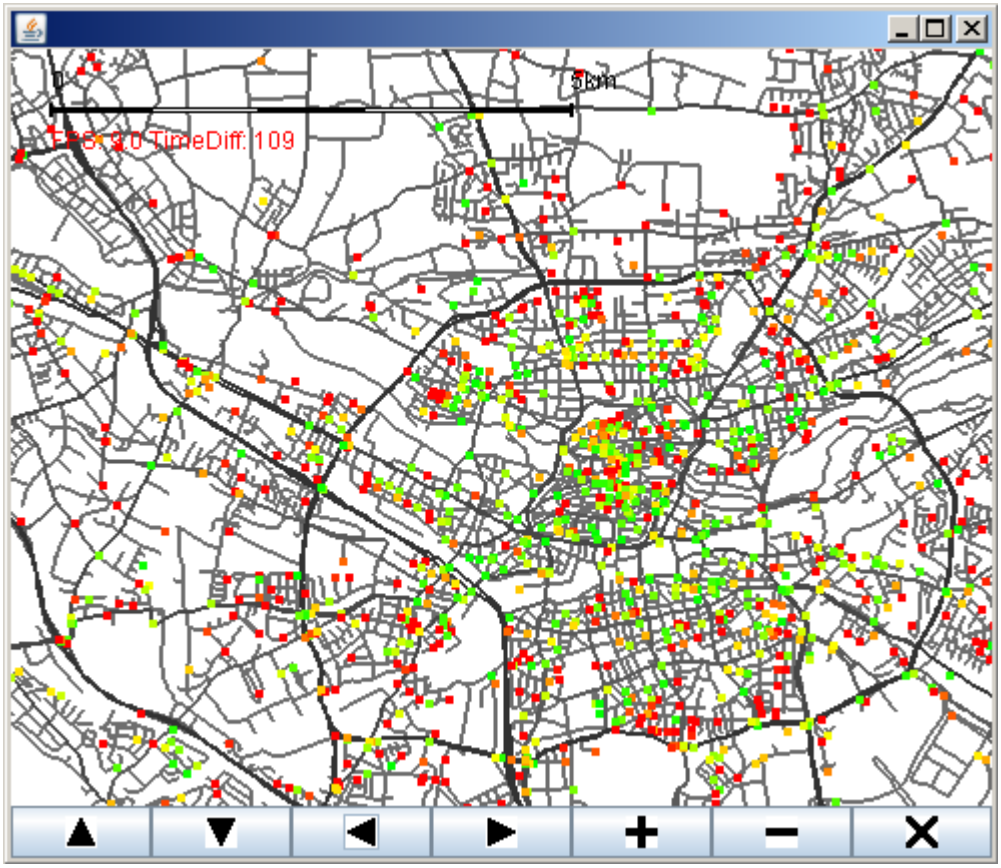

Abbildung 26: Screenshot der Software mit einem Ausschnitt aus Nürnberg. Kategorisierung nach Durchschnittsgeschwindigkeiten, linear skaliert.

Deutlich in diesem Bild zu sehen ist, dass viele, sehr ungleichmäßig verteilte Knotenpunkte rot eingefärbt sind, was mit dem Zustand „kritisch“ gleichzusetzen ist. Die zugehörige Verteilung sieht aus wie folgt:

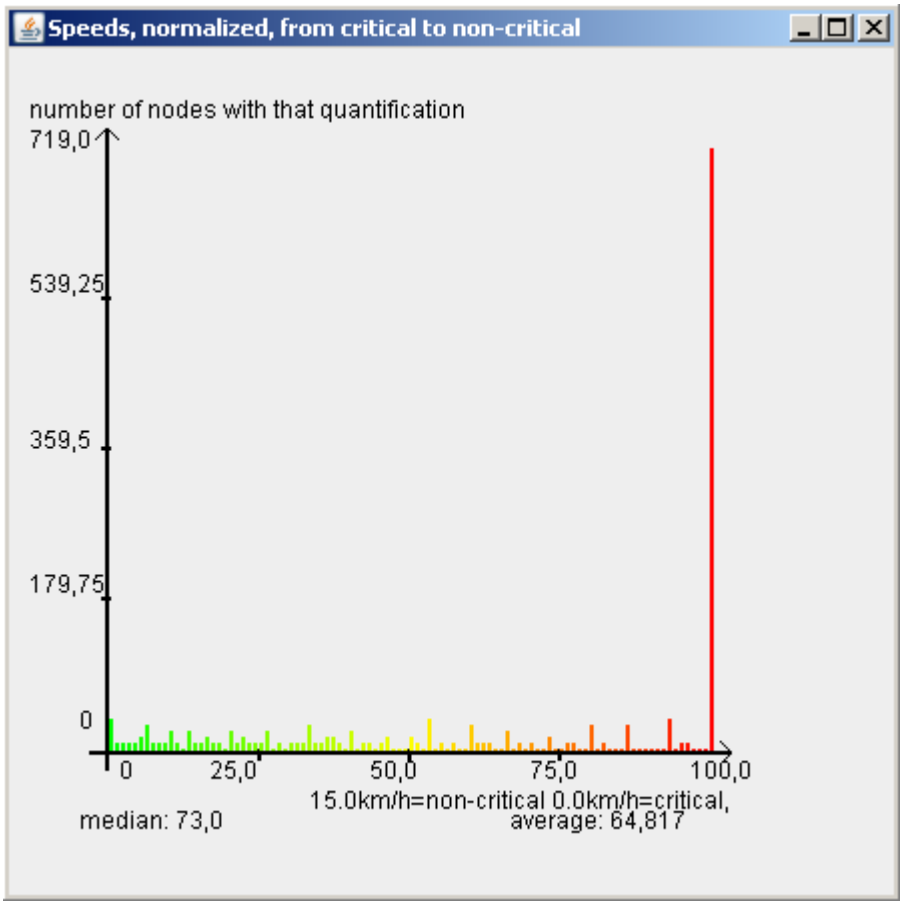

Abbildung 27: Verteilung der Knotenpunkte mit kritischen Durchschnittsgeschwindigkeiten

Das Histogramm zeigt die Verteilung der Knoten mit den Bewertungen von 0, was 15 km/h entspricht, als „unkritischer“ Wert, bis hin zu 100, was dem „kritischen“ Wert von 0 km/h entspricht.

In diesem Beispiel wird deutlich, dass die Mehrheit der Knotenpunkte eine Bewertung von 100% „kritisch“ erhält, also von 0 km/h. In der Kartendarstellung ist ebenfalls zu sehen, dass diese Bewertung auch bei Knotenpunkten auftritt, die Nebenstraßenkreuzungen oder Sackgassen darstellen. Auf den Hauptstraßen sind die Knoten jedoch zumeist als „unkritisch“ gekennzeichnet.

Die Frage nach der Herkunft dieser Verteilung führt zunächst zu einer Betrachtung der Geschwindigkeiten in den Eingangsdaten.

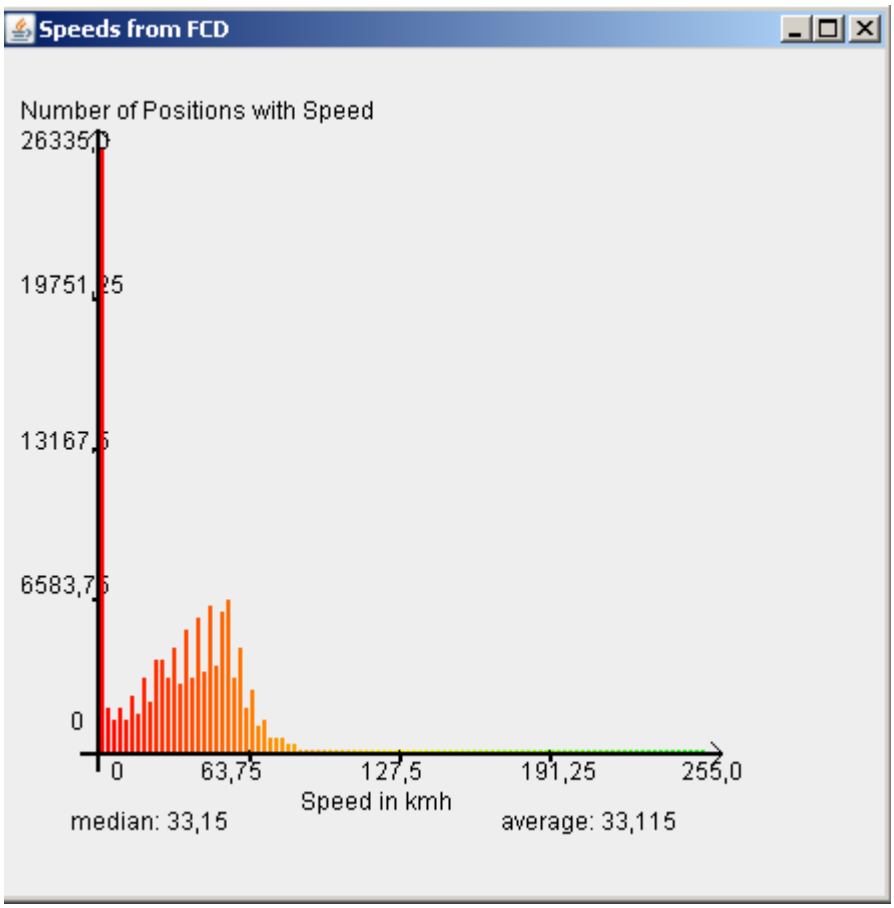

Abbildung 28: Verteilung der Geschwindigkeiten aus den Floating Car Daten

Dieses Histogramm zeigt alle FCD-Positionen aus der Datenbank. In der y-Achse steht die Anzahl der Positionen, die x-Achse stellt die Klassen der Geschwindigkeiten der Positionen dar. Die Klassenbreite beträgt, bedingt durch die Maximalgeschwindigkeit, 2,55 km/h.

Es sind Daten mit Geschwindigkeiten von 0 bis 255 km/h vorhanden. Sehr deutlich ist, dass trotz des Medians bei 33,15 km/h einem sehr großen Teil der Positionen eine Momentangeschwindigkeit von 0 km/h zugeordnet ist.

Davon ausgehend kann man in einer theoretischen Überlegung auf ein Problem dieser Auswertungsmethode schließen.

Zunächst ist es der Fall, dass nur ca. alle 30 Sekunden eine FCD-Position gesendet wird. Daraus ergibt sich der theoretische Fall, dass die Wahrscheinlichkeit sehr hoch ist, dass eine Momentangeschwindigkeit in Kreuzungsnähe genau 0 km/h beträgt, weil ein Fahrzeug gerade zum Zeitpunkt des Sendens steht und wartet. Die Wahrscheinlichkeit jedoch, dass eine Geschwindigkeit während des Durchfahrens an der selben Kreuzung aufgenommen wird, ist eher gering, da eine Durchfahrt sehr schnell vonstatten geht, und das Fahrzeug beim Senden einer Position eventuell schon in der Nähe eines Anderen Knotenpunktes ist. So kann es beispielsweise sein, dass 5 Fahrzeuge eine Kreuzung ohne Wartezeit passieren,

ihre Positionen aber vor und nach der Kreuzung senden, so dass ihre Geschwindigkeiten der Kreuzung nicht zugeordnet werden. Wenn ein Fahrzeug dagegen kurz warten muss und 0 km/h sendet, ist die Durchschnittsgeschwindigkeit am Knotenpunkt 0 km/h. In der Realität ist der Knotenpunkt unkritisch und zeigt kein gestautes Verkehrsmuster, die Auswertung jedoch ordnet den Punkt in einen kritischen Bereich ein.

Bei anderen Eingangsdaten, beispielsweise solchen mit sehr kurzen und konstanten Sendeintervallen bei den FCD-Positionen (beispielsweise 5 Sekunden) oder geeignet vorverarbeiteten Daten wäre diese Methode gegebenenfalls brauchbar, mit den vorhandenen Daten aber liefert sie kein realitätsnahes Ergebnis.

5.2.2.6 Auswertung der Anzahl der erzeugten Positionen pro Fahrzeug, linear eingeteilt

Wie bereits beschrieben liefern weder das Auswerten der Geschwindigkeiten, noch das reine Zählen der entstandenen zugeordneten Positionen theoretisch ein objektives Ergebnis. Letztere Methode scheitert daran, dass nicht bestimmt wird, ob bestimmte Straßen nur deshalb viele kritische Knoten erzeugen, weil sie häufig befahren werden, während andere Straßen wenig befahren werden, durchaus aber kritische Knoten enthalten können.

Den Schlüssel zu einer möglichen Lösung dieses Problems liefert eine ungefähre Einschätzung, welche Zeit Fahrzeuge beim Warten vor einer Kreuzung verbringen. Laut HBS ist eine mittlere Wartezeit dann schon kritisch, wenn sie zwischen 50 und 70 Sekunden beträgt (bei unkoordinierter Zufahrt). Wenn man von einem genauen Sendeintervall der FCD-Positionen von 30 Sekunden ausgeht, sendet ein Fahrzeug im Mittel also 2 oder 3 Positionen (oder mehr) an einem Knotenpunkt, wenn es einer kritischen Wartezeit unterliegt.

Eine FCD-Position enthält eine eindeutige Fahrzeugidentifikationsnummer. Anhand dieser Identifikationsnummern kann berechnet werden, wie viele Fahrzeuge an der Erzeugung der FCD-Treffer für einen Knotenpunkt beteiligt sind. Als Beispiel: Liegen an einem Knotenpunkt 10 FCD-Treffer vor, die 4 unterschiedliche IDs aufweisen, hat jedes dieser Fahrzeuge im Schnitt 2,5 FCD-Positionen in Knotennähe erzeugt. Damit ist von einer hohen Wartezeit auszugehen.

Diese Auswertungsmethode berechnet den Wert für die Einordnung der Knotenpunkte genau nach diesem Prinzip. Die Anzahl der FCD-Treffer für jeden Knotenpunkt wird ermittelt und durch die Anzahl der für die Treffer verantwortlichen Fahrzeugidentifikationsnummern dividiert:

$$Bewertung = \left(\frac{\frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}}{Maximum\left(\frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}\right)}\right) * 100$$

Ergibt sich dabei ein Wert von 2, müssen die Fahrzeuge theoretisch zwischen 30 und 90 Sekunden im Mittel in Knotennähe verbracht, wahrscheinlich also gewartet haben. Werte von

weniger als 2 werden ignoriert.

Mit den gegebenen Parametern wurde wiederum ein beispielhaft ein Test durchgeführt, dessen Ergebnis in der Kartendarstellung wie folgt aussieht:

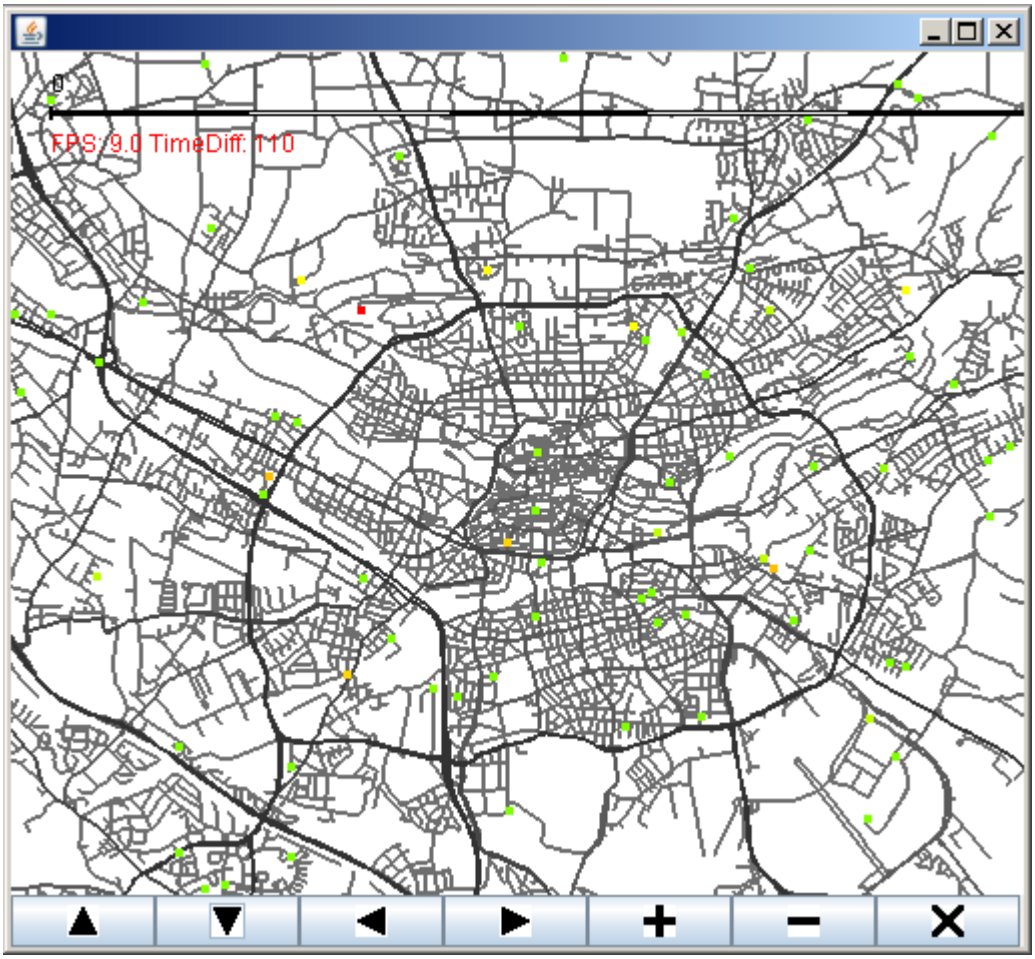

Abbildung 29: Screenshot der Software mit einem Ausschnitt aus Nürnberg. Kategorisierung nach Treffer pro Fahrzeugidentifikationsnummer, linear eingeteilt

In diesem Screenshot der Software ist eine Visualisierung der Methode zu sehen. Grün eingefärbt sind dabei Knotenpunkte, die 2 FCD-Treffer pro Fahrzeugidentifikationsnummer haben. Rot eingefärbt ist das Maximum von Treffern pro Fahrzeugidentifikationsnummer. In der Beispielauswertung wurden 137 Knotenpunkte mit durchschnittlich 2 FCD-Treffern pro ID erkannt. Die Verteilung zeigt folgendes Histogramm:

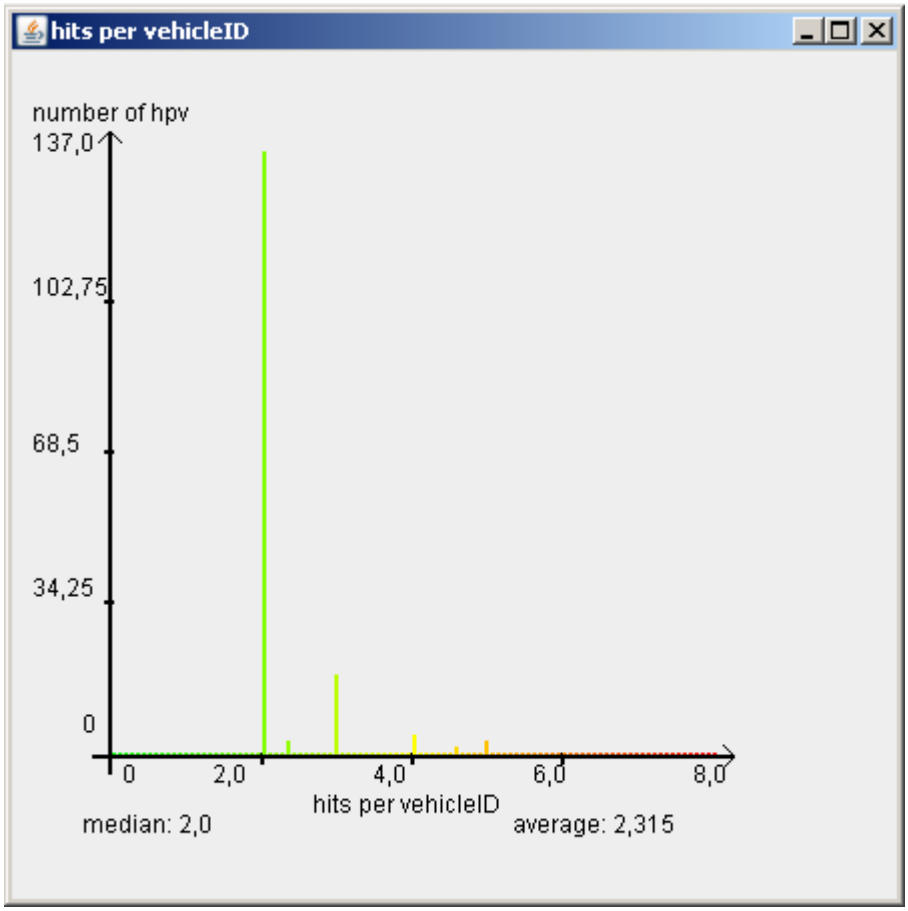

Abbildung 30: Verteilung der FCD-Treffer pro Fahrzeug ID, linear eingeteilt

Zu sehen sind die absoluten Werte der FCD-Treffer pro Fahrzeug als Klasseneinteilung, sowie die Anzahl der Knoten, deren Bewertung genau in eine dieser Klassen fällt. 137 knoten haben die genau 2 Treffer pro Fahrzeug-ID. Das Maximum liegt bei 8 FCD-Treffern pro ID.

Die Auswertungsmethode eliminiert das Problem, dass höher frequentierte Routen auch bei weniger Stau mehr Positionen aufweisen könne. Theoretisch sollte sie damit die objektivste und genaueste Methode, die im Rahmen dieser Arbeit entwickelt wurde, sein. Eine Schwäche sind jedoch die Eingangsdaten, da Positionen nicht immer genau im Intervall von 30 Sekunden gesendet werden, sondern sich gerade bei den Intervallen Fehler finden. Auch spielt der Umstand eine Rolle, dass nicht bekannt ist, in welcher Situation genau die Daten gesendet werden. Bei 30 Sekunden Sendeintervall können 2 Treffern pro Fahrzeugidentifikationsnummer also bedeuten, dass das Fahrzeug 31 Sekunden gewartet hat, ebenfalls wären aber auch z.B. 89 Sekunden Wartezeit möglich. Dies führt zu Ungenauigkeiten. Im Groben kann aber davon ausgegangen werden, dass bei einem langen Auswertungszeitraum, der viele FCD-Positionen nach sich zieht, sich die hierbei entstehenden Fehler durch große Häufigkeiten der unterschiedlichen Fehlerausprägungen verkleinern.

5.2.2.7 Auswertung der Anzahl der erzeugten Positionen pro Fahrzeug, logartihmisch eingeteilt

Ein Nachteil bei der linearen Einteilung ist zudem der Faktor der möglichen entstehenden Ausreißer. Extreme Werte können dazu führen, dass Knoten mit weniger extremen zugeordneten Werten unterschätzt oder fälschlicher Weise zu „unkritisch“ bewertet werden.

Eine gute Möglichkeit, die „Ausreißer“ zu eliminieren, ist das Einordnen auf einer logarithmischen Skala. Das bedeutet hier, dass von jedem Wert „FCD-Treffer pro Fahrzeugidentifikationsnummer“ der Logarithmus dualis gebildet wird. Damit wird jeder Wert von 2 Treffern pro ID zu einem Wert von 1, niedrigere Wertebereiche werden gespreizt, höhere Wertebereiche komprimiert.

Die Methode arbeitet äquivalent zur vorherig beschriebenen Methode. Lediglich das Logarithmieren hat Einfluss auf die Formel:

$$Bewertung = \left(\frac{\log_2 \frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}}{\log_2 Maximum\left(\frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}\right)}\right) * 100$$

Das Ergebnis eines Tests mit den selben Eingangsdaten und Parametern sieht ähnlich aus:

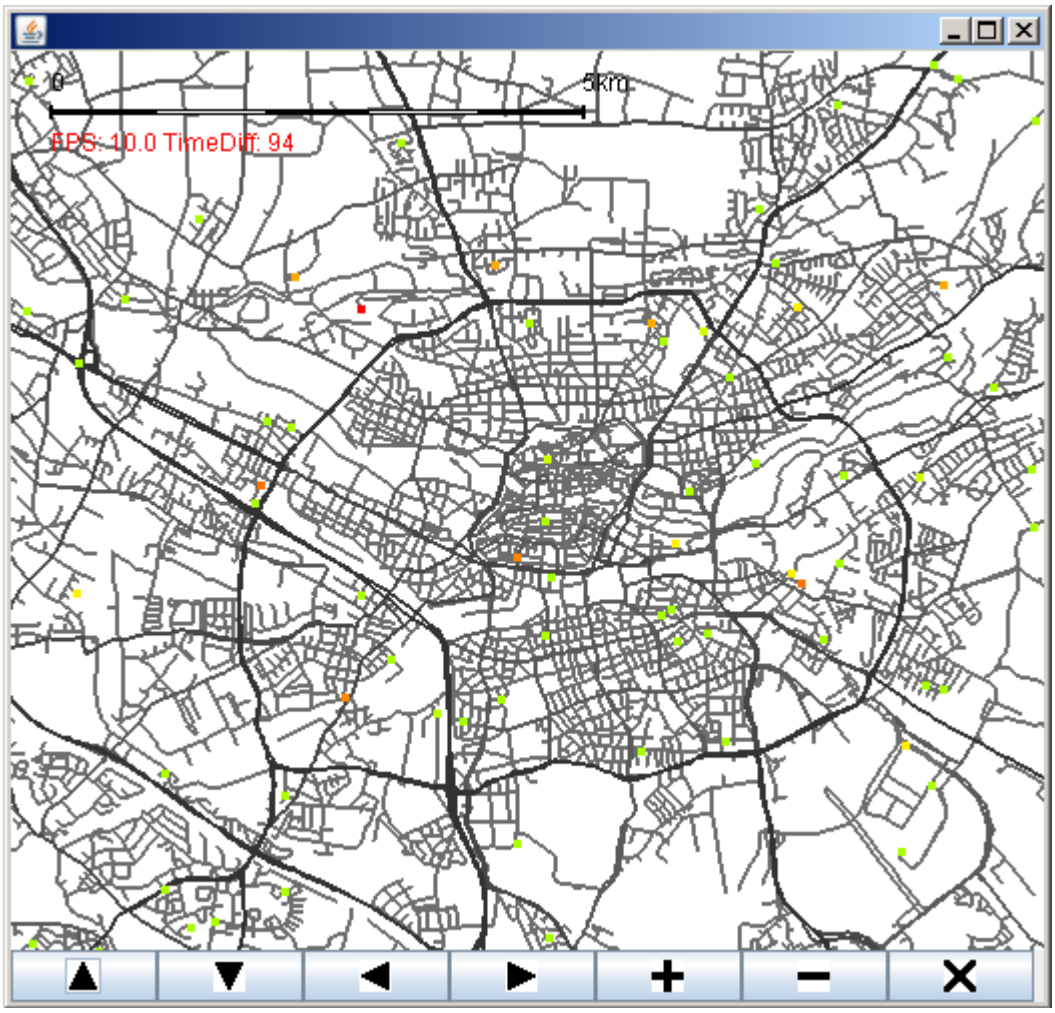

Abbildung 31: Screenshot der Software mit einem Ausschnitt aus Nürnberg. Kategorisierung nach Treffer pro Fahrzeugidentifikationsnummer, logarithmisch eingeteilt

Zu erkennen ist der Unterschied in der Farbgebung. Durch die logarithmische Skalierung ist diese differenzierter. Die Verteilung sieht aus wie folgt:

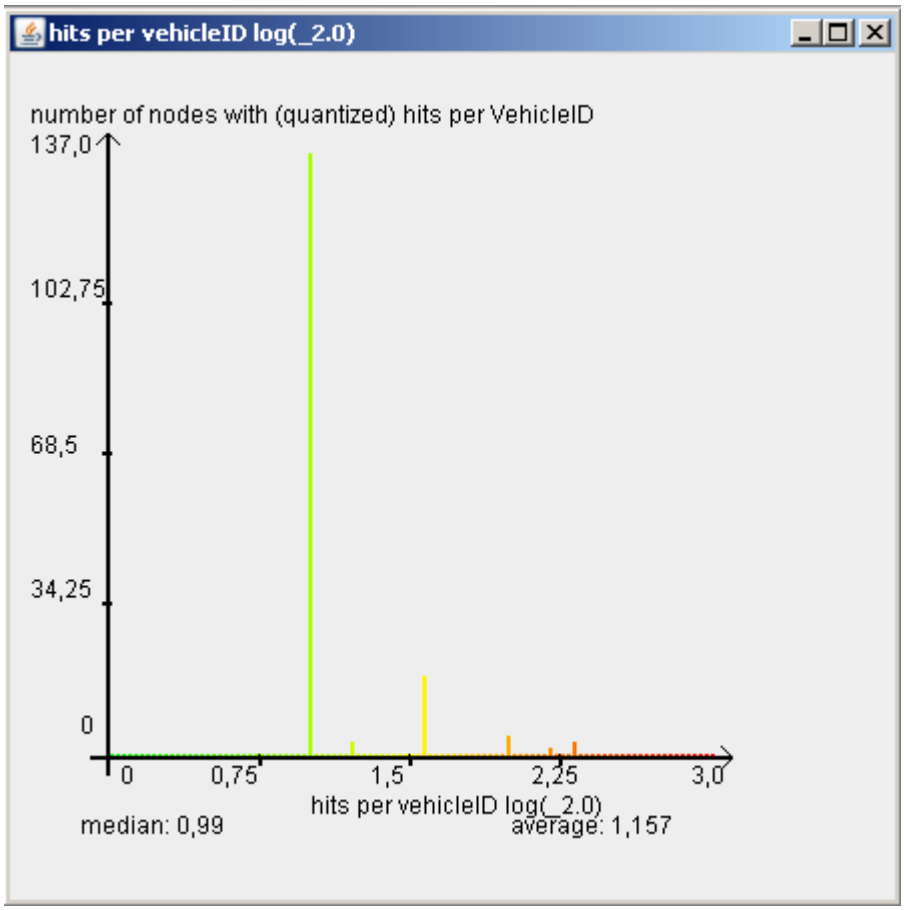

Abbildung 32: Verteilung der Knotenpunkte mit kritischen Werten von FCD-Treffern pro Fahrzeug ID

Zu sehen sind die logarithmierten Werte der FCD-Treffer pro Fahrzeug als Klasseneinteilung, sowie die Anzahl der Knoten, deren Bewertung genau in eine dieser Klassen fällt. Abgesehen von der Farbgebung gleicht das Ergebnis dem der vorangegangen Methode.

Bei beiden Methoden zu erkennen ist, dass sich die als „kritisch“ bewerteten Knotenpunkt nicht, wie bei den ersten 3 Auswertungsmethoden, im Bereich von Kreuzungen oder Hauptstraßen finden lassen. Aus sind es deutlich weniger Knotenpunkte, die „kritische“ Bewertungen erhalten haben. Somit ist es zunächst fraglich, ob die Zuordnung von FCD-Treffern pro Fahrzeug wirklich besser funktioniert, als das reine Zählen der FCD-Treffer.

Eine weitere Fehlerquelle der beiden Methoden ist die Bestimmung des minimalen Schwellwertes. Je nach Betrachtungsweise kann es sich ergeben, dass auch ein anderer Wert als 2 FCD-Treffer pro Fahrzeugidentifikationsnummer die Schwelle zum kritischen Bereich darstellt. Gerade bei der logarithmischen, aber auch bei der linearen Einfärbung, hat jedoch der Schwellwert keinen Einfluss auf die Färbung der Knoten. Wird der Schwellwert gesenkt, werden entsprechend mehr Knoten eingefärbt, zusätzliche Knoten erhalten jedoch nur niedrigere Werte und erhöhen so die Anzahl der grün eingefärbten Knoten.

5.2.3 Vorverarbeitung

Für die Auswertungsmethode mittels Bestimmung der FCD-Treffer pro Fahrzeugidentifikationsnummer ergibt sich ein Problem aus den Rohdaten. Die Rohdaten enthalten Geopositionen mit eindeutigen IDs. Allerdings ist zwar bekannt, dass diese IDs regelmäßig gewechselt werden, jedoch nicht das Zeitintervall, in welchem das geschieht. Damit treten im Wesentlichen 2 mögliche Fehlerquellen auf. Zum einen kann es sein, dass der Wechselzeitraum so groß angelegt ist, dass ein Fahrzeug mit der selben ID eine Route während des Auswertungszeitraumes mehrmals abfährt. Dadurch könnte sich ein erhöhter Wert an FCD-Treffern pro Fahrzeug-ID ergeben. Zum anderen kann es ebenfalls sein, dass bei einem längeren Betrachtungszeitraum eine Fahrzeug-ID mehrmals auftaucht, und zwar für verschiedene Fahrzeuge.

Eine Abhilfe für dieses Problem bietet die Vorverarbeitung der FCD-Positionen durch das Vergeben neuer, eigener IDs. Von 1 an können für alle Positionen neue Fahrzeugidentifikationsnummern vergeben werden. Dieses geschieht nach folgendem Prinzip: nacheinander folgende Positionen von einem Fahrzeug, also mit der gleichen ID, erhalten eine neue, frei erfundene Nummer. Taucht 10 Minuten oder länger keine Position dieses Fahrzeuges mehr im Datensatz auf, wird für diese ursprüngliche ID zukünftig eine neue Nummer vergeben. Wechselt das Fahrzeug den Status, auch innerhalb dieses Zeitraumes, bekommt es ebenfalls eine neue ID. Somit wird annähernd sichergestellt, dass ein Fahrzeug mit der selben ID jede Route nur einmal abfährt, und damit jeden Knotenpunkt pro zurück gelegtem Weg nur einmal besucht. Fehlerquellen sind hier nur unterlassene Statuswechsel der Fahrzeuge oder unvorhersehbare Routenplanungen der Fahrer.

Nicht implementiert, jedoch denkbar, sind auch andere Vorverarbeitungsschritte. Die Floating Car Daten in ihrer Reinform haben mehrere mögliche Fehler.

Zum einen kann es passieren, dass 2 Positionen vom gleichen Fahrzeug unmittelbar nacheinander gesendet werden. Dies geschieht selten, kann aber das Ergebnis verfälschen. Möglich wäre hier eine Eliminierung doppelt auftretender Positionen, wobei hier die Unterschiede zwischen den beiden Positionen genau betrachtet werden müssten.

Zum anderen können Lücken zwischen Positionen entstehen, die deutlich größer als 30 Sekunden sind. Entweder könnte man diese Daten eliminieren oder man interpoliert sie und ersetzt die fehlenden Werte durch berechnete Zwischenpositionen. Dies allerdings kann wiederum zu einer Verfälschung des Ergebnisses führen.

Schließlich liegt eine mögliche Fehlerquelle noch im Verhalten von Taxifahrern. Ausgewertet wird zwar prinzipiell (in den Beispielen) nur der Status „besetzt mit Fahrziel“, jedoch kann auch dieser Ungenauigkeiten enthalten – diese entstehen, wenn ein Fahrgast zu steigt und das Taxi nicht sofort los fährt, oder die Route beendet ist und das Taxi noch steht, während der Fahrgast bezahlt. Auch hier gibt es noch theoretische Ansätze, die Rohdaten entsprechend zu bereinigen.

6 Programmierung und Umsetzung

Dieses Kapitel beschreibt die Umsetzung der Programmteile in der Programmiersprache Java.

6.1 Tool zur Kartenkompression

Das Tool zu Kartenkompression ist ein Java-Programm mit einer einfachen GUI, die über eine Maske eine einfache Eingabe der Kompressionsparameter erlaubt. Das Programm lädt das Kartenmaterial aus einem Ordner, der per Dateidialog geöffnet werden kann. In diesem Ordner müssen sich in Unterordnern die Gebiete im alten Format von DLR-TS befinden, die komprimiert werden sollen. Nach Eingabe der Parameter werden diese für die nächste Verwendung gespeichert. Es folgt eine kurze Übersicht über die Programmierung des Tools, die vor allem die relevanten Schritte genauer beschreibt.

6.1.1 Graphical User Interface

Die GUI ist eine einfache, von „JFrame“ abgeleitete Klasse. Die Klasse „MainFrame“ mit einigen Textfeldern und Checkboxen ermöglicht die Eingabe der Parameter für die Kompression. Sie haben folgende Funktionen:

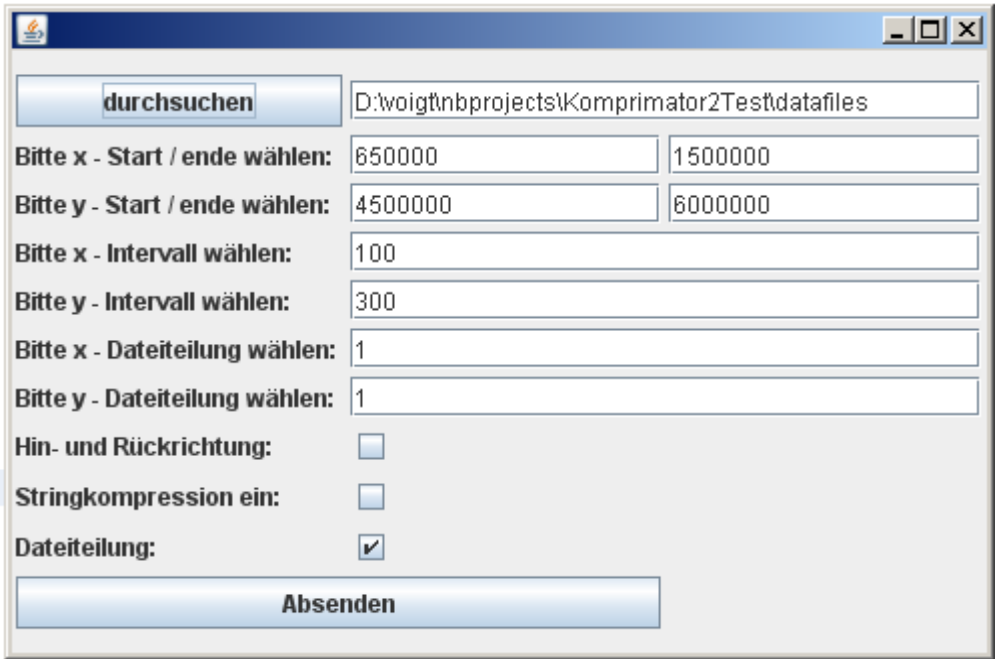

Abbildung 33: Screenshot der Eingabemaske für die Kompressionsparameter

In der oberen Zeile wird mit Hilfe des Dateidialogs der Quellordner für das Kartenmaterial angegeben. Die nächsten beiden Zeilen erfassen das Intervall in Geokoordinaten (NAVTEQ-Format) des maximalen zu komprimierenden Gebietes. Im Screenshot zu sehen ist Deutschland in sehr grob gewählten Grenzen.

Mit Hilfe der nächsten 4 Zeilen können die Parameter für Teilung innerhalb einer Datei und Teilung in verschiedene Dateien angegeben werden. Es wird bestimmt, in wie viele Teile (Rasterteile) das Kartenmaterial geteilt werden soll. Teilt man beispielsweise Deutschland in 10*10 Teile pro Datei und 10*30 Dateien, erhält man eine Teilung in 100*300 Rasterteile.

Die 3 Checkboxen am Ende ermöglichen die Bestimmung, ob nur eine Richtung oder 2 Richtungen integriert werden sollen (Für die Darstellung ist für eine Kante jeweils nur eine Richtung wichtig. Original liegen jedoch 2 Kanten für die meisten Straßen vor, eine für jede Richtung, wobei die Polygone identisch sind), ob die Stringkompression eingeschaltet werden soll und ob eine Dateiteilung vorgenommen wird (wählt ein anderes Kompressionsverfahren aus).

Der Absenden-Button startet die Kompression.

Die GUI basiert auf dem Java eigenen GUI-Toolkit „Swing“ und verwendet das Gridbaglayout, mit der Klasse „GUIutils“ ist dies recht einfach implementiert.

Da dieses Tool nur zu wissenschaftlichen Zwecken benutzt werden wird, sind mögliche Fehleingaben nicht abgefangen. Einfachheit der Programmierung, da diese Software nicht von Endbenutzern verwendet werden soll. Fehleingaben würden Exceptions auslösen.

6.1.2 Speicherung der Einstellungen

Für das Speichern der Parameter existiert die Klasse „CompressionOptions“. In dieser Klasse werden die Parameter wie Teilung in Dateien und in Teile innerhalb der Dateien gespeichert. Die Klasse besitzt die beiden Methoden „loadFromFile()“ und „saveToFile()“, mit denen es möglich ist, die Parameter in eine Textdatei zu speichern bzw. aus einer Textdatei zu laden. Dies hat den Vorteil, dass die Textdatei, in Verbindung mit dem komprimierten Kartenmaterial, Aufschluss darüber gibt, wie das Material geteilt und komprimiert wurde. So kann das Material automatisiert auf die richtige Art und Weise geladen werden.

6.1.3 Kompression

Für die Kompression gibt es verschiedene Klassen, die das Kartenmaterial mit unterschiedlichen Methoden komprimieren. Die Hauptklasse ist die Klasse „Compressor“, von ihr erbt die Klasse „FileSplitZipCompressor“. Die Klasse „Compressor“ komprimiert nur über die Stringkompression, die Klasse „FileSplitZipCompressor“ über das Speichern in ZIP-Dateien. Die Anwendung von Vererbung begründet sich dadurch, dass beide Klassen teilweise gleiche Aufgaben haben. So müssen z.B. beide Klassen das Kartenmaterial laden.

Um die Vererbung zu ermöglichen, sind die Attribute und Methoden in der Klasse „Compressor“ als „protected“ definiert (statt „private“).

Beide Klassen beinhalten eine Methode „doCompression()“, die von der GUI aus aufgerufen wird. Übergeben werden ihr der Pfad zu den Quelldateien und ein Objekt der Klasse

„CompressionOptions“, in dem die Parameter für die Kompression gespeichert sind. Diese Methode ruft sequenziell weitere Methoden auf, die die Kompression vorbereiten und durchführen.

Zunächst wird die Methode „makeIntervalSteps()“ aufgerunfen.

In der Klasse „Compressor“ werden in dieser Methode 2 Textdateien angelegt, eine für Knoten und eine für Kanten. Je nach der Unterteilung in Rasterteile werden diese Dateien mit Zeilen gefüllt, die die einzelnen Rasterteile voneinander teilen. Eine solche Zeile hat die Form

„ 650000_4500000>“.

Jede Zeile wird von einem TAB-Zeichen eingeleitet, enthält die Startkoordinaten in x- und y-Richtung, getrennt von einem Unterstrich, die als kleinste mögliche Koordinaten eines Rasterteils gelten.

Der Grund für dieser Vorbereitung ist es, die Ausgabedateien so vorzubereiten, dass Kanten und Knoten, die einem bestimmten Rasterteil zugeordnet werden, unterhalb der die Rasterteile bestimmenden Zeilen abgelegt werden können. So können Rasterteile später einzeln ausgelesen werden, indem von einer entsprechenden Trennzeile zur nächsten alle Zeilen gelesen und interpretiert werden. Da dabei die anderen Zeilen in der Datei ignoriert werden können, werden Speicher und Zeit beim Lesen des komprimierten Materials gespart.

In der Klasse „FileSplitZipCompressor“ legt diese Methode ZIP-Dateien anstatt von Textdateien an. In dieser Klasse ist auch die Speicherung von Straßennamen implementiert, so wird auch für die Straßennamen eine Datei vorbereitet. DieZIP-Dateien werden initial mit der Java-internen Kompressionsstufe „0“ erstellt, um später schnellstmöglich ausgelesen werden zu können. Es werden so viele ZIP-Dateien angelegt, wie als Dateiteilung angegeben wurde. Diese Dateien erhalten ihren Namen entsprechend wie die Trennzeilen in der auf Textdateien basierenden Kompression, zusätzlich mit dem Suffix „_edges_zip.zip“, „_nodes_zip.zip“ und „_names_zip.zip“. Als Beispiel:

„650000_5250000_edges_zip.zip“

Die Zahlen sind hier jeweils wieder die kleinstmöglichen enthaltenen Koordinaten.

Anstatt Textdateien mit zeilenweiser Trennung anzulegen, werden bei dieser Kompressionsmethode Einträge, so genannte „ZipEntries“ angelegt. Diese entsprechen in einem ZIP-Archiv enthaltenen Dateien. Diese Einträge gliedern die ZIP-Dateien in kleinere Teile, welche die Rasterteile darstellen. Diese sind nicht nur nach Rasterteilen, sondern ebenfalls nach Straßenlevels getrennt. Die „ZipEntries“ sind einfache Dateien, die als ASCII Dateien beschrieben und gelesen werden können.

Durch einfache Berechnung kann für jede Geokoordinate (im Zusammenhang mit dem Straßenlevel) nun bestimmt werden:

- zu welcher Datei gehört sie?

- zu welchem Eintrag gehört sie?

Dazu sind nur die Informationen aus der Klasse „CompressionOptions“ notwendig. In so vorbereitete Dateien können nun Kanten und Knoten gespeichert werden. Aus diesen lässt sich schnell lesen, da für jeden Kartenausschnitt bei einer Darstellung schnell errechenbar ist, welche ZIP-Dateien und welche ihrer Einträge geöffnet werden müssen.

Um Redundanzen bei Straßennamen zu vermeiden, werden Straßennamen nur mit den Dateischritten aufgeteilt, und nicht noch einmal nach Einträgen. Dies ist aufgrund der geringen Anzahl von Straßennamen nicht notwendig und spart zudem noch mit ZIP-Dateien verbunden Overhead.

Sind die Ausgabedateien vorbereitet, wird das Verzeichnis mit dem Quellmaterial ausgelesen und die Pfade der Unterordner sowie die Namen der entsprechenden Regionen werden in einer globalen Variable abgelegt (Methode „readFolderList()“).

Über diese Verzeichnisse wird nun iteriert, die nächsten Schritte werden für jedes Verzeichnis einzeln ausgeführt. Die gleichzeitige Bearbeitung des gesamten Materials wäre zu speicheraufwendig.

Zunächst wird jedes Verzeichnis in der Methode „readMapMaterialFolder()“ gelesen. Dies geschieht mit Hilfe der Klasse „Kachelreader“, einer älteren Klasse, die das Quellmaterial interpretieren kann. Ein „Kachelreader“-Objekt hält die Informationen für eine Region in Hashtables gespeichert. Der „Kachelreader“, der eine Region gelesen hat, wird als globale Variable abgelegt. Für jede weitere Region wird er jedoch überschrieben.

Da eine Region, je nach Größe, eventuell in mehrere Rasterteile unterteilt werden muss (oder Rasterteile Teile von verschiedenen Regionen enthalten können), muss nun für jede Kante und ihre Knotenpunkte bestimmt werden, in welches Rasterteil sie gehört. Problematisch ist hierbei, dass eine Kante durch mehrere Rasterteile hindurch gehen kann, weshalb sie dann mehrfach gespeichert werden muss. Diese Bestimmung erfolgt in der Methode „prepareEdges()“. Alle im Kachelreader geladenen Kanten werden abgearbeitet, wie auch die zugehörigen Knoten. Anhand der Geokoordinaten der Knoten wird nun festgelegt, in welche Rasterteile die Kante und ihre Knoten gespeichert werden müssen.

Für jedes betroffene Rasterteil wird eine Kante der Methode „storePreparedEdge()“ übergeben, zusammen mit einem String, der das Rasterteil identifiziert. In dieser Methode wird die Kante, wie auch ihre Knoten, einer globalen Hashtable zugeordnet (Schlüssel sind die Strings, die die Rasterteile identifizieren, Werte sind weitere Hashtables. Diese enthalten dann direkt die Kanten oder Knoten als Werte, als Schlüssel deren IDs).

Ist eine Region nach diesem Schema unterteilt, kann das enthaltene Material persistent gespeichert werden. Hier ergeben sich wieder Unterschiede zwischen „Compressor“ und „FileSplitZipCompressor“. Bei beiden wird die Methode „storeIntoFile()“ aufgerufen. Diese funktioniert sowohl für Knoten als auch für Kanten, was beim Aufruf explizit mit angegeben werden muss.

In „Compressor“ wird die entsprechende Ausgabedatei und mit ihr alle gegebenenfalls bereits gespeicherten Kanten und Knoten vollständig in ein Vector-Objekt eingelesen. Dies kostet leider viele Resourcen und viel Rechenzeit, aber ein sortiertes Einfügen in eine Datei ist sonst leider technisch nicht möglich.

Für jede der vorbereiteten Trennzeilen, vorhanden als Einträge im Vector-Objekt, wird nun geprüft, ob ihren Teil betreffendes Kartenmaterial in den Hashtables vorhanden ist. Wenn ja, wird dieses Material hinter dieser Zeile in den Vector eingefügt. Dies geschieht allerdings nach Straßenlevels sortiert, „0“ (Autobahn) als erstes, „4“ (Nebenstraße) als letztes. Diese Sortierung sorgt später dafür, dass beim Laden bestimmter Levels das Lesen beim erreichen eines zu geringen Levels abgebrochen werden kann.

„FileSplitZipCompressor“ geht mit den Kanten und Knoten relativ analog vor, speichert allerdings in den dafür vorbereiteten ZIP-Dateien. Die ZIP-Dateien und ihre Einträge werden geladen, die Einträge eingelesen. Gibt es für einen Eintrag neues einzufügendes Material, wird dies an den Eintrag angehangen. Hierbei werden sowohl die Zuordnung in der Hashtable als auch das Straßenlevel betrachtet. Sind alle ZIP-Einträge bearbeitet, wird die komplette ZIP-Datei anschließend neu geschrieben.

„FileSplitZipCompressor“ enthält noch eine weitere Methode, um die Straßennamen abzuspeichern. Dies ist in „Compressor“ nicht implementiert. Die Methode heißt „storeStreetnamesIntoFile()“. Sie geht die die Namen speichernden ZIP-Dateien durch, lädt sie und sucht für jede, ob neue Namen hinzu kommen. Diese werden angefügt, danach werden die ZIP-Dateien neu geschrieben.

Beide Klassen rufen aus der Methode „storeIntoFile()“ heraus die Methoden „compressEdge()“ oder „compressNode()“ auf. Diese Methoden sind dafür verantwortlich, dass ein Objekt vom Typ „Edge“ oder „Node“ in einen String umgewandelt wird. Hierbei werden alle relevanten Attribute mit Tabulatoren getrennt hintereinander gefügt. Ist die Stringkompression aktiviert, werden die Strings vorher komprimiert.

In diesen Methoden erfolgt auch das gezielte Auslassen nicht benötigter Attribute, um Speicherplatz zu sparen.

Die Stringkompresion erfolgt in der statischen Klasse „Compression“. Sie kann mit den Methoden „compString()“ und „decompString()“ Long-Werte in speziell für die Speicherung im ASCII-Format komprimierte Strings umwandeln bzw. von komprimierten Strings in Long-Werte.

Beim ersten Aufruf einer der Methoden wird eine Art „Pseudo-ASCII-Tabelle“ erstellt. Diese ist ein Array, das als Elemente Character-Werte enthält, mit fortlaufenden Zahlen als Index. In dieser Umsetzung enthält das Array 184 Zeichen. Bei der Erstellung werden viele Sonder- und Steuerzeichen ausgelassen, da diese zu Lesefehlern führen können. (z.B. Zwischen den ASCII-Zeichen 247 und 255). Auch werden alle Whitespace-Zeichen ausgelassen.

Die Methode compString komprimiert eine Zahl i zu einem String. Positive zahlen werden

zunächst mit -1 multipliziert. Mit Festlegung, dass die Variable „radix“ so groß ist wie die Anzahl der Elemente der Pseudo-ASCII-Tabelle (also 184) und „digits“ die „Pseudo-ASCII-Tabelle“ selbst, führt folgender Algorithmus die Kompression durch:

```
while (i <= -radix) {
        int pos = (int) -(i % radix);
        buf[charPos--] = digits[pos];
        i = i / radix;
    }
```

Die Ausgangszahl wird also jeweils modulo-dividiert. Der Rest wird mit Hilfe von „digits“ in einen Character umgewandelt und dieser einem Character-Array zugeordnet. Dann wird die Ausgangszahl unter Nichtbeachtung des Restes dividiert. Die verbleibende Zahl wird wiederum modulo-dividiert, usw.

Zum Schluss wird das Character-Array in einen String umgewandelt.

Die Dekomprimierung in der Methode „decompString()“ erfolgt einfacher. Der komprimierte String wird von hinten („rechts“) nach vorne („links“) Zeichen für Zeichen durchlaufen. Jedes Zeichen wird durch „Ablesen“ in der Pseudo-ASCII-Tabelle in eine Zahl umgewandelt, diese wird mit dem Abstand vom „rechten Rand“ des Strings potenziert und auf das Gesamtergebnis aufaddiert. Dadurch erhält man die Ausgangszahl zurück. Ist der „linke Rand“ des Strings ein Minuszeichen, wird die Zahl mit -1 multipliziert.

Auf diese Art und Weise lassen sich Zahlen in ASCII komprimieren, wobei die Größe theoretisch in etwa der des Binärformats entspricht (siehe Kapitel 4.1.4).

6.1.4 Vorbereitung für schnelles Auslesen der Textdateien

Bei der Kompressionsmethode mit Textdateien ist es für ein schnelles Lesen wichtig, dass die Positionen in der Datei, bei welchen die Rasterteile beginnen, vor dem Lesen bekannt sind. Mit der Klasse „RandomAccessFile“ kann man so zu einer bestimmten Stelle (Anfang des Rasterteils) springen und von dort aus weiter lesen, vorherige Zeilen können ignoriert werden.

Diese Positionen werden extra gespeichert, in hier so genannten „lookup Files“. Die Klasse „LookupCreator“ führt diese Aufgabe aus, wenn die Methode „createLookupFile()“ aufgerufen wird. Sie durchsucht die bei der Kompression mit der Klasse „Compressor“ entstandenen Textdateien nach den Trennzeilen. Für jede Trennzeile speichert sie den String, der ihr Rasterteil repräsentiert, und zusätzlich die aktuelle Position des Dateizeigers. Beide Informationen werden gemeinsam in extra Textdateien, den „Lookup-Dateien“ abgelegt. Vor dem Einlesen des Kartenmaterials können nun die „Lookup-Dateien“ in Hashtables eingelesen werden, die für jedes Rasterteil dann sofort dessen Position innerhalb der Textdatei wiedergeben können.

6.2 Kartenlademodul

Das Kartenlademodul ist eine Ansammlung von mehreren Klassen, die meist auf Interfaces basieren. Sie abstrahieren Schritt für Schritt die einzelnen Ebenen, auf denen sich die Methodik für das Laden von Karten unterscheiden kann. Das Kartenlademodul ist so aufgebaut, dass es in seiner Funktionsweise so variabel wie möglich ist. Einen Überblick über die Interfaces und Beispielimplementierungen gibt folgende Grafik:

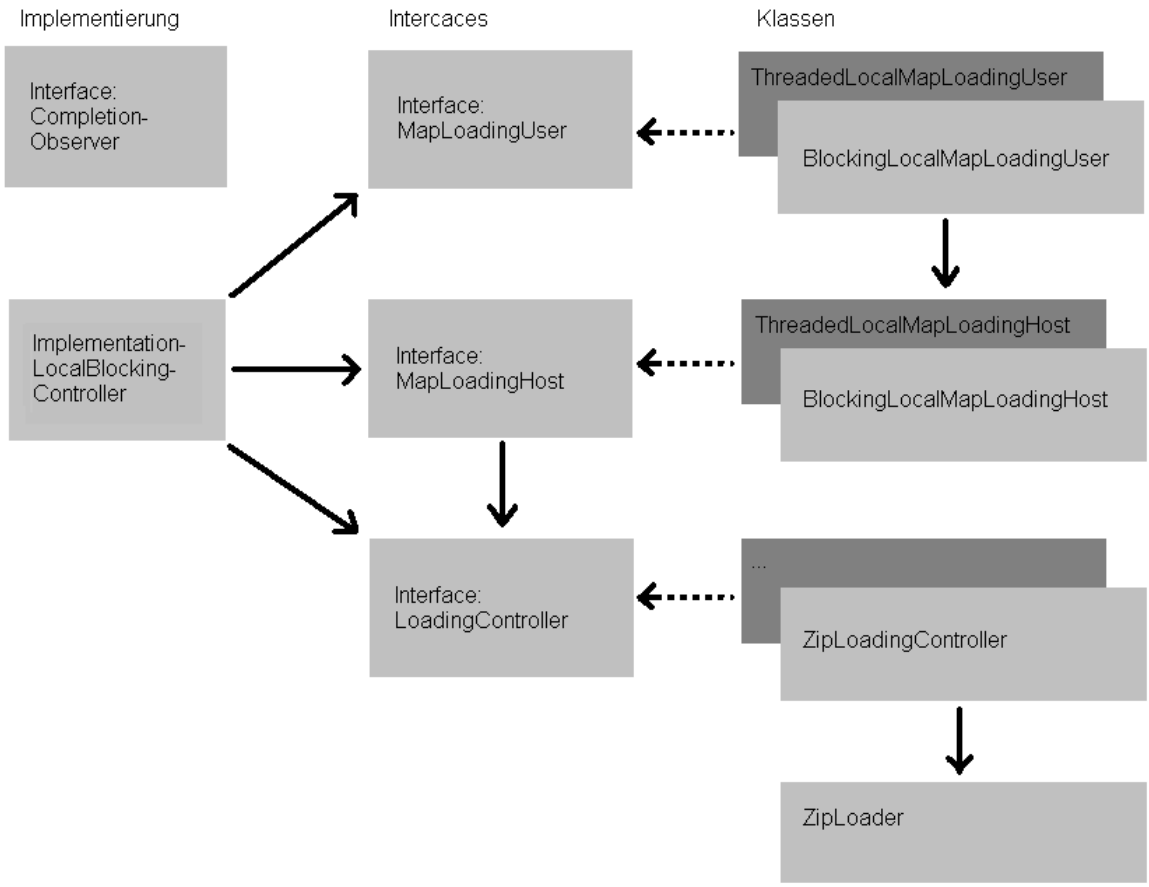

Abbildung 34: Klassendiagramm des Kartenlademoduls

Durchgehende Pfeile in der Darstellung bedeuten Referenzen, unterbrochene Pfeile stellen Implementierungen eines Interfaces dar.

Auf der linken Seite zu sehen ist die Klasse „ImplementationLocalBlockingController“, die die implementierten Klassen zur Vereinfachung der Benutzung instanziert und zusammenfasst. Über Interfaces hält sie Referenzen auf die Klassen, die die einzelnen Schritte des Ladevorganges abstrahieren. Das Interface „CompletionObserver“ kann von der das Lademodul benutzenden Software implementiert werden, um per Observer-Prinzip Informationen über beendete Ladevorgänge zu erhalten.

Um verschiedene Arten der Übertragung des Kartenmaterials zu ermöglichen, gibt es die

Interfaces MapLoadingHost und MapLoadingUser. Wie genau die Implementierungen dieser Klassen miteinander kommunizieren, ist mit Absicht nicht definiert. Zum einen kann dies der direkte Weg über einen Methodenaufruf sein, zum anderen wären aber auch eine Socketkommunikation oder eine Kommunikation mittels „Remote Method Invocation“ (RMI) denkbar.

„LoadingController“ ist für den den eigentlichen Ladevorgang verantwortlich. Implementiert sind die Klassen „BlockingLocalMapLoadingUser“, „BlockingLocalMapLoadingHost“, „ZipLoadingController“ und „ZipLoader“.

Um die Funktionsweise und das Programmierkonzept darzustellen folgt eine Beschreibung der einzelnen Interfaces und deren Implementierungen.

6.2.1 MapLoadingUser

Implementierungen von MapLoadingUser sind die Klassen, die von Programmen, die die Ladeschnittstelle benutzen, aufgerufen werden.

MapLoadingUser implementierende Klassen stellen die Methode „loadMapPart()“ mit den Parametern für die Intervalle von Geokoordinaten in x- und y- Richtung, dem maximalen Straßenlevel und einer flag für das Ignorieren von Zwischenknoten bereit. Diese Methode hat keinen Rückgabewert, da nicht definiert ist, ob das Laden synchron oder asynchron erfolgt. Ein Aufruf der Methode startet lediglich das Laden.

Die Feststellung, dass das Laden beendet ist, erfolgt nach dem Oberver-Prinzip. Mit Hilfe des Interfaces „CompletionObserver“ kann eine Klasse, die das Lademodul benutzt, sich mit der Methode „addCompletionObserver()“ beim MapLoadingUser als Observer registrieren. Dieser benachrichtigt alle bei ihm registrierten Observer, wenn das Laden beendet ist.

Das geladene Kartenmaterial kann in Form von Hashtables mit den Methoden „getCurrentEdgeHash()“, „getCurrentNodeHash()“ und „getCurrentNameHash()“ abgerufen werden. Übergeben wird das Material, das die Klasse von MapLoadingUser momentan gespeichert hat.

Geladenes Material wird prinzipiell unbegrenzt lange im MapLoadingUser gespeichert. Da bei einer Darstellung oder anderen Aufgaben die Ausprägung des Benötigten Materials wechseln kann, weshalb immer mehr Speicher für immer mehr Material benötigt wird, gibt es die Methode „cleanup()“. Diese bekommt, ähnlich wie „loadMapParts()“ die noch benötigten Koordinaten und das noch benötigte Level übergeben. Alles Material, was nicht diesen Parametern entspricht, wird gelöscht.

Weil ein MapLoadingUser–Objekt gleichzeitig ein Observer für MapLoadingHost sein kann, ist auch die Methode „notifyCompletion“ notwendig. In einer netzwerkbasierten Kommunikation würde diese jedoch nicht benutzt werden.

Zunächst implementiert ist das Laden lokal und blockierend, also ohne die Benutzung eines

separaten Threads. Die Klasse „BlockingLocalMapLoadingUser“ implementiert alle Methoden des Interfaces und greift beim Laden direkt auf „BlockingLocalMapLoadingHost“ zu. BlockingLocalMapLoadingUser bearbeitet in den Methoden das Material und überführt es von der in Rasterteile unterteilten Struktur in unsortierte Hashtables, verwaltet intern jedoch das Kartenmaterial in Rasterteilen. Dies ist notwendig, um bei einem Aufruf von „cleanup“ das Material schneller und effizienter verwalten zu können. Beim Laden wird nur Material nachgeladen, welches noch nicht vorhanden ist. Um diese Gegebenheiten feststellen zu können, muss über verschiedene Ebenen das Materials iteriert werden.

6.2.2 MapLoadingHost

MapLoadingHost ist das Interface für die Klassen, die den Host für das Laden repräsentieren, sei dies lokal oder über ein Netzwerk.

Um die Vielfältigkeit in der Anwendung zu ermöglichen, sind im Interface nur 2 Methoden festgeschrieben.

„addObserver()“ fügt einen Observer hinzu, der dann benachrichtigt wird, falls eine Implementierung des Interfaces einen Ladevorgang beendet hat.

„setLoadingController()“ fügt einer Implementierung einen „LoadingController“ hinzu. Diese ist verantwortlich für das kontrollierte Laden von Kartenmaterial aus einer nicht näher definierten Quelle.

Als Implementierung ist von Anfang an eine Klasse BlockingLocalMapLoadingHost als Gegenstück zum BlockingLocalMapLoadingUser umgesetzt.

In dieser Umsetzung wird bereits im Konstruktor ein LoadingController verlangt.

BlockingLocalMapLoadingHost besitzt die Methode „loadMaterial()“, die das Laden startet. Als Parameter übergeben werden wieder das maximale Straßenlevel sowie ein Boolean-flag für das Ignorieren von Zwischenknoten. Allerdings wird, um das benötigte Material zu identifizieren, ein Vector-Objekt übergeben. In diesem Vector-Objekt müssen sich für die Funktionalität des Ladens die Namen der Rasterteile als Strings befinden. Dies hat den Grund, dass das zu ladende Material in dieser Ebene komplexer sein kann als ein einfaches „Rechteck“. Wenn ein benötigter Kartenausschnitt simpel vergrößert wird, dann dehnt sich das Material in alle Himmelsrichtungen aus, währen das Material im ursprünglichen Rechteck nicht erneut geladen werden muss. So muss also bereits im MapLoadingUser das Rechteck in die Namen der Rasterteile überführt worden sein.

Um dies zu ermöglichen, benötigt eine auf BlockingLocalMapLoadingHost zugreifende Klasse die Angaben über die Beschaffenheit der Komprimierung des Kartenmaterials, gespeichert in einem Objekt der Klasse „CompressionOptions“. Mit der Methode „getOptions()“ stellt BlockingLocalMapLoadingHost eine Möglichkeit bereit, an diese Angaben zu kommen.

Wenn das Kartenmaterial geladen wurde, kann es mit „askForResults()“ abgerufen werden. Diese Methode übergibt eine Hashtable, die das Material enthält.

Um die in Java nicht implementierte Rückgabe von mehreren Rückgabewerten zu umgehen, wird das Kartenmaterial in einer Hashtable übergeben. Die Schlüssel werden bereits im LoadingController festgelegt.

Implementiert wurden auch 2 Klassen, die das Laden in einem separaten Thread ermöglichen. Diese funktionieren zum größten Teil identisch. Sie nennen sich „ThreadedLocalMapLoadingUser“ und „ThreadedLocalMapLoadingHost“.

Das Besondere im ThreadedLocalMapLoadingHost ist die Nebenläufigkeit, so dass das Laden hier geschieht, ohne andere Programmteile zu blockieren. In der Methode „loadMaterial()“ wird dabei ein neuer Thread erstellt und gestartet, der dann das Laden mittels des LoadingControllers erledigt. Wichtig ist, dass dieser Thread nur in einer Instanz simultan laufen soll. Dafür sorgt ein Boolean-Wert, der den Thread so lange schlafen lässt, bis sein Vorgänger beendet ist. Der Booleanwert „working“ wird vor dem Laden auf „true“ gesetzt, bei der Abholung des Materials auf „false“. Das darf erst nach der Übergabe des Materials passieren, weshalb das Konstrukt „try{ }finally{ }“ hier notwendig wird.

Diese Art der Imlementierung hat den großen Nachteil, dass eine Art Warteschlange entstehen kann, wenn mehrfach nacheinander das Laden aufgerufen wird, ohne zwischendurch auf Ergebnisse zu warten. Hierbei ist nicht deterministisch, welcher der anstehenden Aufrufe wann abgearbeitet wird. Beim Benutzen dieser Implementierung ist darauf zu achten und vorsichtig damit umzugehen. Da diese Implementierung nicht benötigt wurde, befindet sie sich im Entwicklungsstadium.

6.2.3 LoadingController

Das Interface „LoadingController“ dient als Klasse für die Vereinheitlichung des Ladens von Kartenmaterial, das in Rasterteile unterteilt ist. Es stellt die Methode „loadRasterParts()“ mit den Parametern für das maximale Straßenlevel, die Rasterteile im Vector und einem Boolean-flag für das Ignorieren von Zwischenknoten zur Verfügung.

„askForOptions()“ liefert ein Objekt vom Typ „CompressionOptions“ mit den Kompressionsparametern.

Implementiert ist ein die Klasse „ZipLoadingController“ für das Laden aus ZIP-Dateien. Der ZipLoadingController erfordert im Konstruktor den Pfad zum Kartenmaterial. Er kapselt das eigentliche Laden, welches in der Klasse „ZipLoader“ realisiert ist. Der ZipLoadingController wandelt die Ergebnisse vom Laden auch in der Art und Weise um, dass Namen, Kanten und Knoten, die sich in Hashtables befinden, in eine übergeordnete Hashtable gespeichert werden. Schlüssel sind hier die Strings „nodes“, „edges“ und „names“. Damit kann mit einer Rückgabe das gesamte geladene Material übergeben werden.

Da die Speicherung des Kartenmaterials in ZIP-Dateien die Teilung in Rasterteile und

zusätzlich Dateiteile erfordert, kann der Zipcontroller umrechnen, welche Dateien geöffnet werden müssen, um die angeforderten Rasterteile zu laden.

Die Klasse „ZipLoader“ bekommt im Konstruktor den Pfad zum Kartenmaterial übergeben. Sie kann mit den Methoden „loadEdgesFromFile()“, „loadNodesFromFile()“ und „loadNamesFromFile()“ das Material laden und gibt es direkt zurück. Bis hierhin werden auch das maximale Straßenlevel und das Ignorieren von Zwischenknoten als Parameter übergeben, hier dienen diese dazu, den entstehenden Zeitaufwand zu minimieren. In dieser Klasse entsteht durch das Laden und Parsen der Dateien der eigentliche Aufwand des Ladenvorgangs.

6.2.4 Andere Klassen

Die Klasse „ImplementationLocalBlockingController“ dient nur als Zusammenfassung der implementierten Klassen, also für das lokale und blockierende Laden. Hier werden die Konstruktoren aufgerufen und die Klassen zusammengeführt. Für den LocalMapLoadingUser gibt es eine get-Methode, um auf seine Methoden Zugriff zu ermöglichen.

„MyEdge“ und „MyNode“ sind Repräsentation für Kanten- und Knotenobjekte. Namen hingegen sind immer Strings.

„CompressionOptions“, „Scaling“ und „Comprimation“ entsprechen den in der Kompression schon benutzten Klassen.

6.3 Kartendarstellungsmodul

Das Kartendarstellungsmodul besteht aus 3 Klassen.

Die Hauptklasse des Moduls ist „MapPanel“. Sie ist von der Swing-Klasse JPanel abgeleitet, das in eine andere GUI eingebunden werden kann. Sie enthält wenig Logik und hält viele öffentliche Methoden für die Steuerung der Kartendarstellung bereit.

Die GUI-Klasse für die Kartendarstellung ist die Klasse „DrawCanvas“. Sie ist zuständig für das Rendering des Straßennetzes.

Das Fachkonzept ist in der Klasse „DrawController“ organisiert. „DrawController“ lädt, hält und organisiert das Kartenmaterial sowie die Parameter für die Darstellung.

Die Verbindung und das Zusammenspiel der Klassen ist in folgendem Klassendiagramm sichtbar:

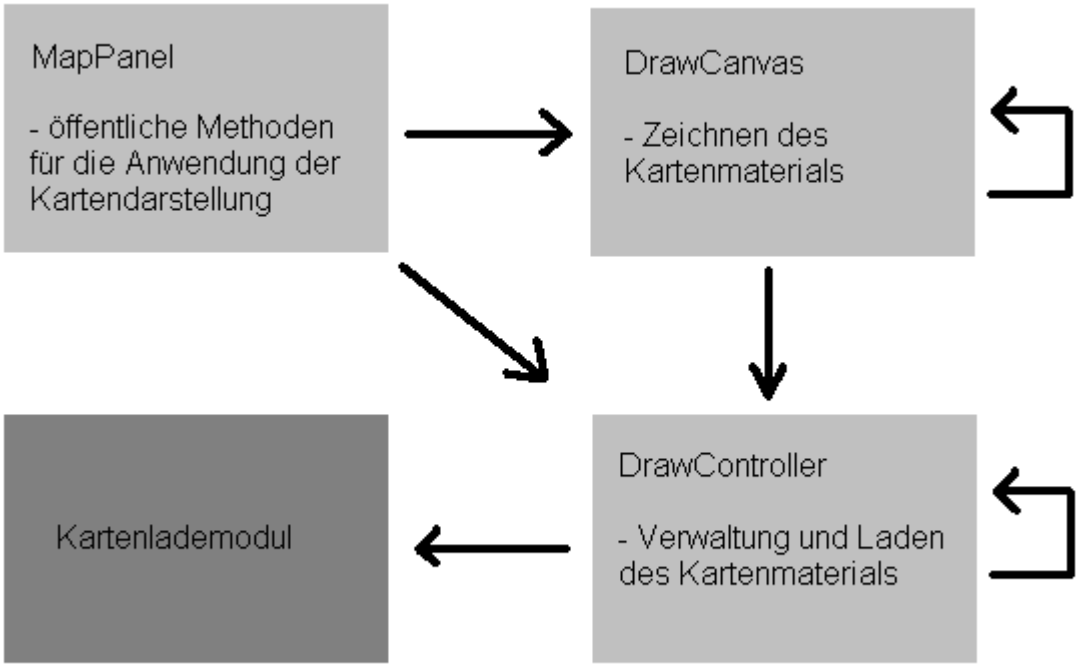

Abbildung 35: Klassendiagramm des Kartendarstellungsmoduls

Auffällig sind hier die Referenzen von „DrawController“ und „DrawCanvas“ aus sich selbst. Diese sind notwendig, um innerhalb von internen Threads atomare Abschnitte mit Hilfe von „synchronized“ zu ermöglichen.

6.3.1 MapDrawPanel

Diese Klasse hält die Referenzen zu „DrawCanvas“ und „DrawController“. Sie kapselt das Canvas für die Darstellung und zeigt es an. Zudem befinden sich mehrere öffentliche Methoden in dieser Klasse, die die Einstellung der Parameter ermöglichen. Diese Methoden enthalten nur wenig Logik, so wie zum Beispiel die Umrechnung von Richtung und Prozentsatz in Geokoordinatendifferenz für das Scrollen.

6.3.2 MapDrawCanvas

Das Canvas unterstützt von der Konzeption her primär „active Rendering“, das bedeutet, dass mit einer konstanten Framerate das Bild immer wieder neu gezeichnet wird. Hierzu ruft ein Thread immer wieder die Methode „renderAll()“ auf, in der das Offset und die Skalierung für das Zwischenbild errechnet werden und das Bild entsprechend auf das Canvas gezeichnet wird. Parameter zu dieser Berechnung kommen aus der Klasse „DrawController“. Zudem wird hier die Methode „createMapImage()“ aufgerufen.

CreateMapImage() überprüft anhand eines Boolean im „DrawController“, ob neues Kartenmaterial geladen wurde. Ist dies der Fall, bedeutet das, dass das Zwischenbild neu

gezeichnet werden muss.

Dazu erforderlich sind die Parameter wie Bildschrimgröße, Pixel pro Meter, Größe des Kartenausschnittes u.s.w.. Da das Zeichnen etwas länger dauern kann und in einem seperaten Thread erledigt wird, müssen diese Parameter eigenständig für das Zeichnen gespeichert werden. Erst, wenn das Zeichnen des neuen Bildes beendet ist, werden die Paramter für das aktuelle Zwischenbild mit denen für das neue Zwischenbild überschrieben. Die Parameter können sich während des Zeichnens ändern, also erfolgt die Zuweisung in einem synchronisierten Block.

Alle vorhandenen Kanten werden nun gezeichnet. Dabei gibt es eine spezielle Reihenfolge, die bestimmt, wie das Überlagern der Straßenkanten im Ergebnis aussieht:

- Zeichne zuerst die höchsten Level, also die Nebenstraßen, falls vorhanden
- Wenn Straßen einen Rand haben sollen, zeichne zuerst die Ränder für alle Straßen des Levels
- Zeichne dann den Kern der Straßen (weniger breit)
- Fahre mit dem nächsten Straßenlevel fort

Für jeden Zeichenvorgang wird die Methode „drawEdge()“ aufgerufen, mit den Parametern für die Kante, das Level, das Graphics2D-Objekt des neuen Zwischenbildes, die Knoten und die Bildgröße.

Ist das Zeichnen aller Straßen beendet, wird das alte Zwischenbild mitsamt dessen Parametern durch das Neue überschrieben.

„drawEdge()“ zerlegt die Straßenkanten in ihre Polygone, um diese einzeln zu zeichnen. Hierbei ist es von Relevanz, ob eine Straßenkante einen Zwischenknoten besitzt oder nicht. Zwischenknoten haben, je nach Richtung der Straßen, einen Polygonzug darstellende Koordinaten entweder vom Anfang bis zum Ende der Kante oder umgekehrt. Negative Kanten (-IDs) haben die den positiven Kanten (-IDs) entgegen gesetzte Richtung. Haben Straßenkanten keine Zwischenknoten oder sollen Zwischenknoten ignoriert werden, wird eine Straßenkante nur vom Start- bis zum Endpunkt gezeichnet.

Nach dem Zeichnen der Kanten wird „drawName()“ aufgerufen, um den Straßennamen zu zeichnen, wenn die Straßenkante eine gültige ID dafür enthält.

Für jedes Polygon wird die Methode „drawStreet()“ aufgerufen, mit den Geokoordinaten des Start- und Endpunktes des Polygons.

Hier werden zunächst die Geokoordinaten nach dem Prinzip

$$(Geokoordinate - Minimale\ Geokoordinate\ des\ Bildes) * (Pixel\ pro\ Geokoordinate)$$

errechnet, und zwar in Längen- und Breitengradrichtung (x- und y- Richtung) getrennt. Dann wird die Dicke des zu zeichnenden Polygons (Straßen werden nicht als Linie, sondern als dicke Linie aus Polygonen gezeichnet) bestimmt, abhängig davon, ob das Polygon ein Rand

oder ein Kern der Straße ist. Die Farbe wird der entsprechenden Voreinstellung nach zugewiesen und das Polygon wird gezeichnet, mit „drawThickLine()“. Diese Methode zeichnet 2 sich überlagernde Polygone mit einem Offset, einmal in x- und einmal in y-Richtung von der Breite der Straße. Zur Erklärung dieses Vorganges folgende Skizze:

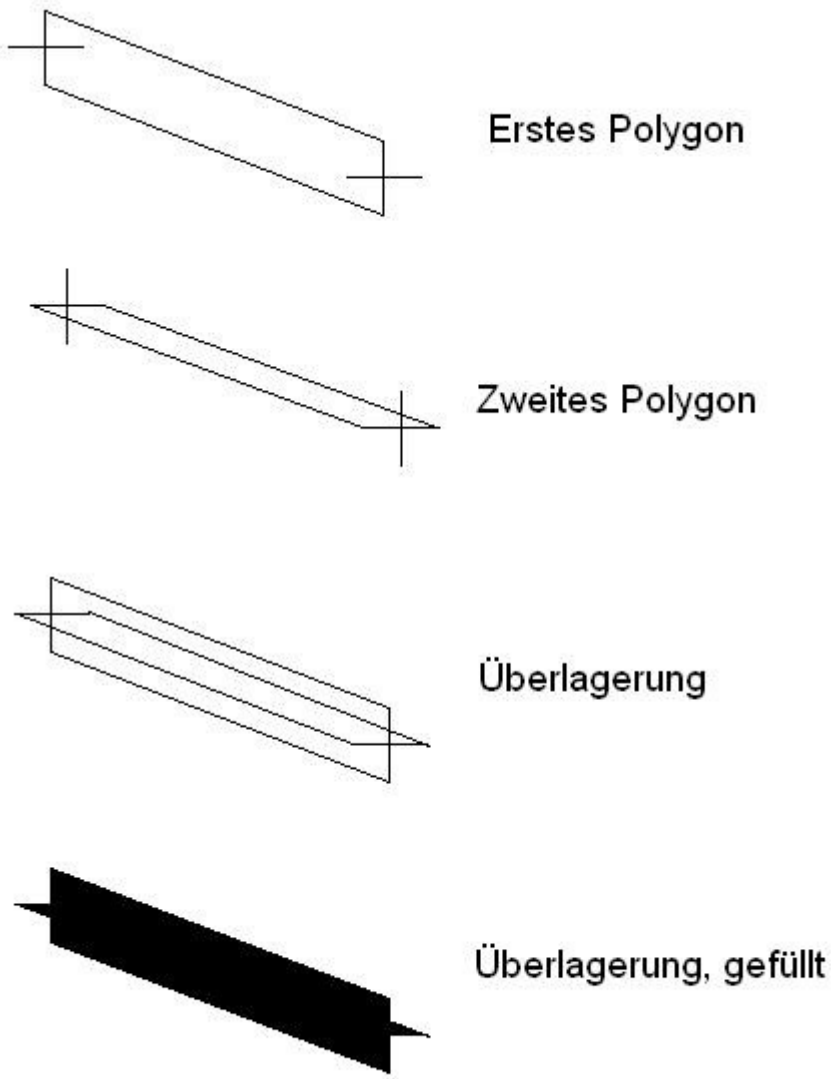

Abbildung 36: Skizze zum Prinzip der Darstellung dicker Linien mit Hilfe der Methoden eines Graphics-Objektes

Diese Skizze zeigt schnell die Einfachheit, aber auch den Nachteil dieser Art, Straßenkanten zu zeichnen. Der Hauptnachteil liegt darin, dass die Breite der Straße nur eine ungefähre Approximation ist, denn schräge Straßen werden schmaler gezeichnet als waage- oder senkrechte.

Der zweite Nachteil ist die weniger gut aussehende Darstellung von Straßenkanten, die enden, ohne in eine andere Straßenkante überzugehen.

Die schönere Alternative wäre gewesen, aus den Geokoordinaten von Start- und Endpunkt den Winkel der Straße zu errechnen, mit

$$\tan(\alpha) = \frac{x2 - x1}{y2 - y1} \quad ,$$

den Winkel der Stirnseite aus zu rechnen. Damit wären Start- und Endpunkte der Stirnseite berechenbar. Mit der Rückrechnung aus diesem Winkel könnte ein Polygon zwischen den beiden Stirnseiten der zu zeichnenden Straßenkante gezeichnet werden. Schließlich könnten um Start- und Endpunkt mit einem Kreis mit dem Radius

$$r = \frac{Straßenbreite}{2}$$

überzeichnet werden. Allerdings kostet diese Berechnung so viel Zeitaufwand, dass sie für ihren Nutzen nicht verhältnismäßig erscheint. Im Ergebnis sieht die Straßenzeichnung jedoch trotz der starken Vereinfachung gut aus.

Straßennamen werden in der Methode „drawStreetName()“ gezeichnet. Wenn Start- und Endpunkt einer Straßenkante weit genug auseinander liegen, wird in die Mitte der Straßenkante der Name gezeichnet. Dies ist eine sehr einfache Variante, die Straßen zu beschriften. Allerdings reicht sie zunächst für den Zweck dieser Arbeit aus.

6.3.3 DrawController

Die Klasse „DrawController“ enthält das Fachkonzept für die Kartendarstellung.

Kern ist die Methode „drawCoordRect()“. Sie bekommt die Geokoordinaten eines zu zeichnenden Kartenausschnittes sowie die Höhe und Breite der Zeichenfläche übergeben. Sie errechnet zur weiteren Verwendung folgende Werte:

- Startkoordinate in x- und y- Richtung
- Endkoordinate in x- und y-Richtung
- Intervall der Geokoordinaten in x- und y- Richtung
- Pixel pro Genkoordinate in x- und y- Richtung
- Pixel pro Meter (in beiden Richtungen gleich)
- Kartenformat
- Bildformat
- Meter pro Geokoordinate in x- und y- Richtung

Die Meter pro Geokoordinate sind in y- Richtung immer gleich, in x- Richtung jedoch unterschiedlich (aufgrund der Kugelform der Erde). Als Basis für die Berechnung wird die Mitte des y-Geokoordinatenintervalls genommen.

Die Methode entscheidet anhand der Voreinstellung auch, welches Straßenlevel geladen werden soll. Sie stößt das Laden des neuen Kartenmaterials an.

Genutzt wird die Methode auch von der Methode „setPixelPerMeter()“. Hier werden aus der Änderung der Zoomstufe die neuen Geokoordinaten des Ausschnittes berechnet, und „drawCoordRect()“ wird aufgerufen.

„drawCoordRect()“ ruft die Methode „loadMapParts()“ auf. Hier wird ein Thread erstellt, der „loadMapPart()“ vom „MapLoadingUser“ aus dem Kartenlademodul aufruft.

Es existieren 2 Boolean-flags, die in einem synchronisierten Block verändert werden. Die eine heißt „loadingLocked“ und wird gesetzt, wenn ein Thread das Laden anstößt. Damit ist das Laden für weitere Anfragen blockiert. Wird „loadMapPart()“ trotzdem ein weiteres mal ausgeführt, wird eine Schleife, die den 2. Thread schlafen lässt, so lange ausgeführt, bis „loadingLocked“ wieder false ist. Des Weiteren setzt diese Schleife ein flag „alreadyWaiting“. Das bedeutet, dass bei einem 3. Aufruf die Ladeparameter für den 2. Aufruf verändert werden, die Methode danach aber verlassen wird. So kann nur ein Ladevorgang mit den neuesten Ladeparametern warten und ein einzelner kann ausgeführt werden.

Die Klasse „DrawController“ implementiert das Interface „CompletionObserver“ aus dem Lademodul, und damit die Methode „isComplete()“. Sie dient somit als Observer. Hat das Kartenlademodul nun neues Material geladen, wird „processNewMapParts()“ aufgerufen. Hier wird das geladene Kartenmaterial in Straßenlevel unterteilt und somit zum Abruf vom Canvas vorbereitet, der Boolean, der angibt, dass neues Material eingetroffen ist, wird gesetzt, und das Laden wird freigegeben.

Außerdem wird die Methode „cleanUp()“ des „MapLoadingUser“ aufgerufen, um das Kartenmaterial aufzuräumen.

6.4 FCD-Auswertung und Darstellung der Auswertungsdaten

Beim Programm zur Auswertung der „Floating Car Data“, Ermittlung der kritischen Knotenpunkte und deren Darstellung handelt es sich um ein Java-Programm, welches das Kartenlademodul und das Kartendarstellungsmodul verwendet. Eine Gesamtübersicht zeigt, wie die bisher beschriebenen Elemente hier verwendet werden:

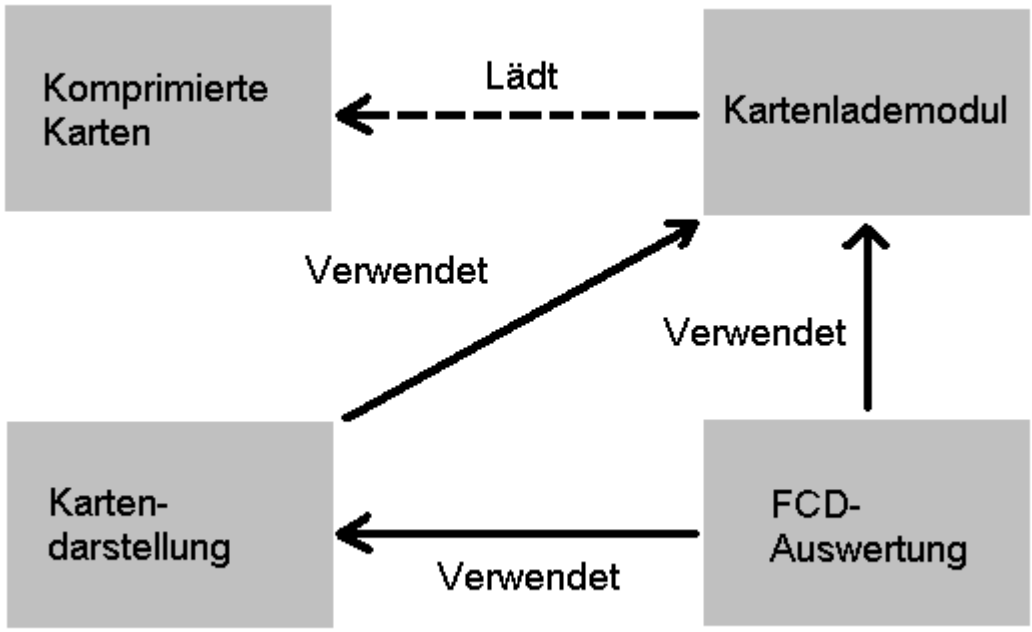

Abbildung 37: Übersicht über die Verwendung der erstellten Module

Die mit dem Kompressionstool erstellten Karten können mittels des Kartenlademoduls geladen werden. Die Kartendarstellung nutzt das Lademodul, um das Kartenmaterial zu laden und anschließend darzustellen.

Um die Modularität zu bewahren, werden die Karten auch von der FCD-Auswertung mittels Lademodul geladen. Diese werden für die Auswertungsalgorithmen benötigt. Die Kartendarstellung übernimmt in der FCD-Auswertung nur die Darstellung der Kanten und Knoten.

Um es zu ermöglichen, dass in der FCD-Auswertung verschiedene Auswertungsmethoden verwendet werden können, ist das Programm modular aufgebaut. Eine Übersicht gibt das folgende Bild:

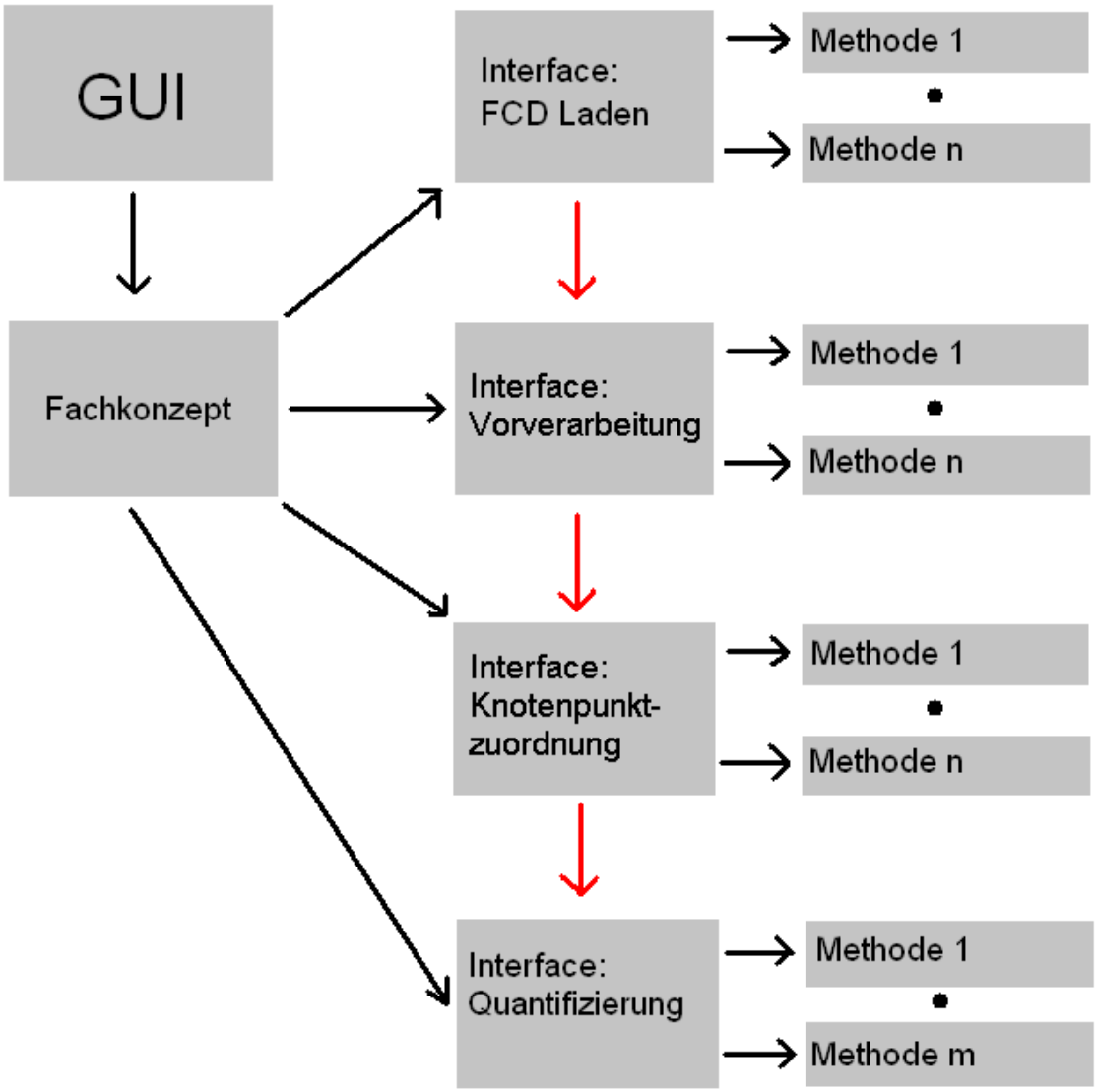

Abbildung 38: Übersicht über die Zusammenhänge und einzelnen Teile der FCD-Auswertung

Dieses Bild zeigt sehr schematisch die Zusammenhänge der Einzelnen Klassen und Interfaces des Programms.

Für einer Vereinfachung der Benutzereingaben und die Darstellung des Kartenmaterials existieren 2 GUI-Klassen, hier als „GUI“ zusammengefasst. Um die einzelnen Schritte der Auswertung auszulagern und von der GUI zu trennen, gibt es eine Fachkonzeptklasse, die nacheinander die einzelnen Verarbeitungsschritte aufruft. Die Reihenfolge der Verarbeitungsschritte ist hier mit roten Pfeilen gekennzeichnet.

Der erste Schritt für eine Auswertung ist das Laden der Floating Car Daten. Dies erfolgt im Regelfall über das Laden aus einer MySQL-Datenbank. Allerdings ist es zu

Demonstrationszwecken außerhalb des gesicherten Netzwerkes des DLR notwendig, eine alternative Lademethode anzubieten. Implementiert ist so auch das Laden aus einer Textdatei. Um die beiden Lademethoden kompatibel zu kapseln, implementieren sie ein einheitliches Interface.

Anschließend folgt (optional) die Vorverarbeitung der Floating Car Data. Hier ist zwar nur eine Vorverarbeitungsmethode implementiert, denkbar sind jedoch auch weitere, für den eventuellen Einsatz in der Zukunft. Auch diese Implementierung geschieht über ein Interface.

Ebenfalls über ein Interface implementiert sind die verschiedenen Methoden für die Zuordnung von FCD-Positionen zu Knotenpunkten, ebenso die Auswertung der zugeordneten Daten. Somit sind sämtliche Auswertungsmethoden austauschbar und kombinierbar, um unterschiedliche Ergebnisse nachvollziehen zu können.

Sind die Verarbeitungsschritte durchlaufen, können die so gewonnenen Daten in der GUI dargestellt werden.

6.4.1 Graphical User Interface

Es existieren 2 GUI Klassen, die unterschiedliche Aufgaben haben.

Zum einen gibt es die Klasse „StartFrame“, die lediglich dazu dient, Benutzereingaben für bestimmte Parameter der Auswertung zu vereinfachen. Diese Klasse ist so ausgelegt, dass sie relativ einfach funktioniert und dem Benutzer für das Testen der verschiedenen Methoden das Einstellen der Auswertungsparameter erleichtert. Da das Programm nur zu Forschungszwecken verwendet wird und nicht von Endbenutzern verwendet werden soll, wurde auf das Abfangen von Fehleingaben verzichtet.

Die Klasse ist von JFrame abgeleitet. Sie stellt ein Fenster dar, in dem die Parameter eingestellt werden können:

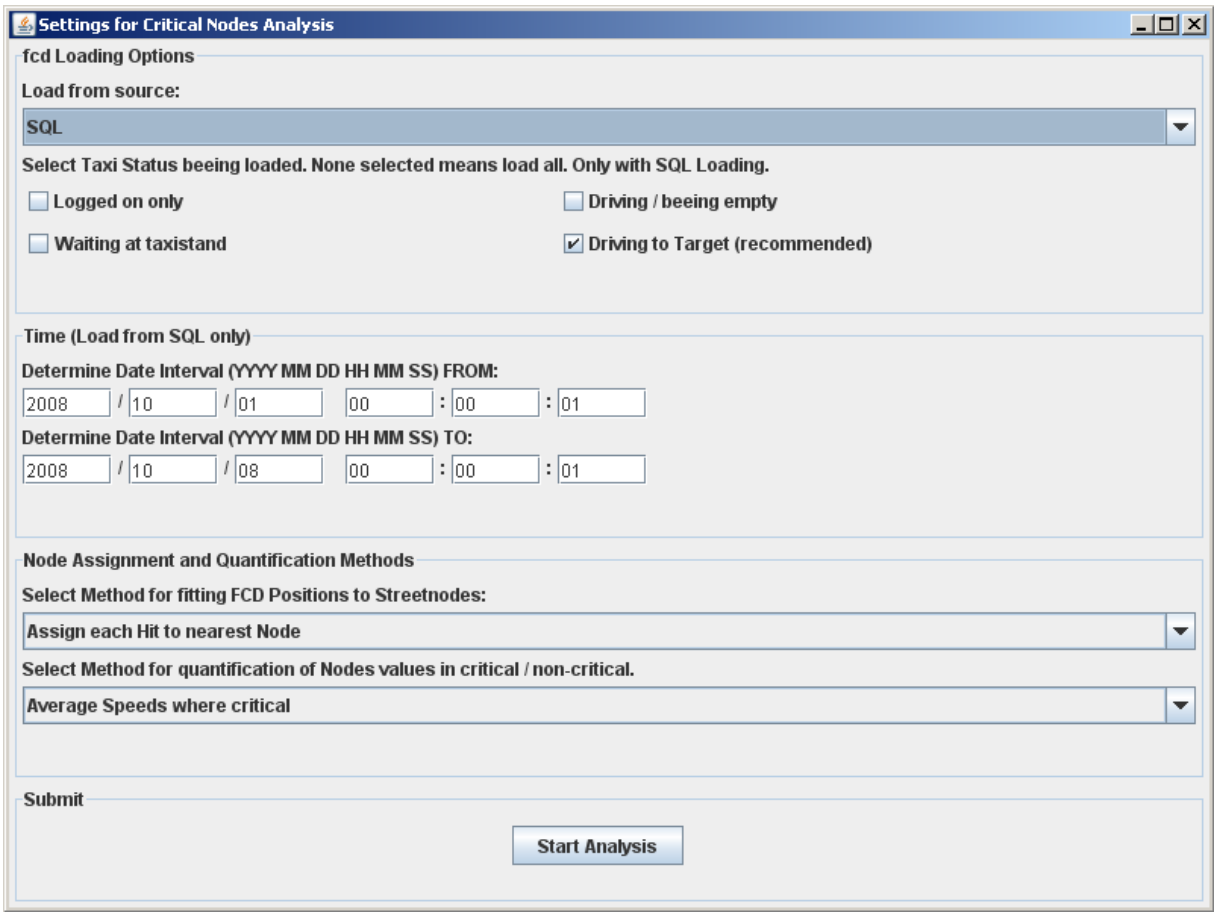

Abbildung 39: Abbildung 17: Screenshot des Startfensters, Klasse "StartFrame"

Die Eingabefelder sind durch JPanels in bestimmte Bereiche sortiert. Zunächst wird eine Auswahlliste vom Typ „JComboBox“ angezeigt, die eine Festlegung der Lademethode für die Floating Car Data ermöglicht. Danach folgen 4 Checkboxen, die eine Einschränkung der Statusnummern für die Positionen ermöglichen.

Für die Eingabe des Zeitraumes der Ausertung kann das Datum mittels jeweils 6 Textfeldern eingegeben werden.

Es folgen 2 Auswahllisten vom Typ „JComboBox“, mit denen die Zuordnungsmethode für FCD-Positionen zu Knotenpunkten sowie die Auswertungsmethode ausgewählt werden können. Zum Ende folgt ein Button, der das Starten der Auswertung ermöglicht.

Alle GUI-Elemente werden im Konstruktor mittels GridBagLayout erzeugt, positioniert und befüllt. Ein Druck auf den Button zum Starten ruft über einen ActionListener die Methode „startAnalysis()“ auf. Hier werden die GUI Elemente ausgewertet, was die Benutzereingaben festlegt. Der „MainFrame“-Konstruktor wird nun mit allen Parametern aufgerufen.

„MainFrame“ ist das Hauptfenster des Programmes. Die Klasse „MainFrame“ ist ebenfalls eine von JFrame abgeleitete GUI-Klasse und enthält das Hauptfenster der Anwendung. Da das Programm so ausgelegt ist, dass jeweils nur eine Analyse pro Programmstart

durchgeführt wird, enthält sie nur einen Konstruktor. Im Konstruktor wird das Fenster erzeugt und das Kartendarstellungs-Panel hinzugefügt. Zudem erhält das Fenster noch 7 JButtons, über die eine Steuerung der Kartendarstellung möglich ist. Für die Kartendarstellung werden einige Parameter gesetzt, wie die Bestimmung des anzuzeigenden Kartenausschnitts und die Einfärbung des Straßenkanten in Graustufen, zur besseren Visualisierung. Angezeigt wird das Fenster in dieser Form:

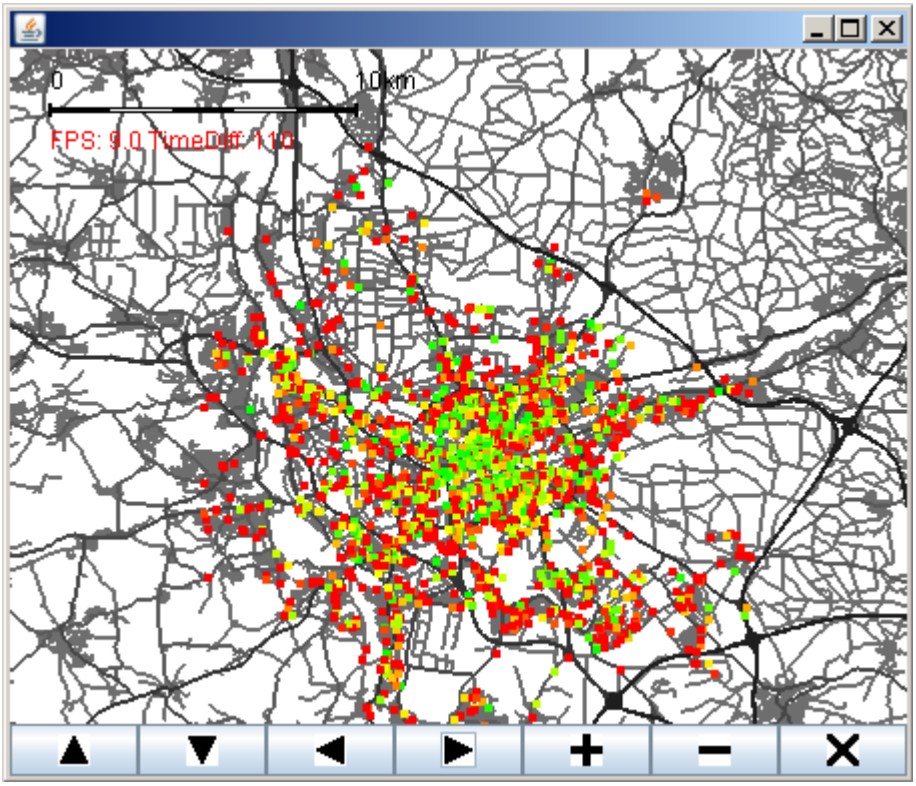

Abbildung 40: Screenshot des Hauptfensters, Klasse "MainFrame"

Ebenfalls im Konstruktor geregelt wird das Einfärben der Knotenpunkte in einer Schleife. Ausschlaggebend dafür ist die Analyse, die in der Fachkonzeptklasse „AnalysisController“ zusammengefasst wird.

6.4.2 Fachkonzept

Die erste Stufe des Fachkonzepts ist die Klasse „AnalysisController“. Diese Klasse hat 2 Aufgaben, die auch im Quellcode getrennt sind.

Die erste Aufgabe ist im Konstruktor implementiert. Im Konstruktor wird mittels switch-Anweisungen festgelegt, welche Methoden für das Laden, die Zuordnung und die Auswertung von Floating Car Data verwendet werden. Die Methoden werden mit statischen Konstanten identifiziert, die teilweise in der Klasse selbst, teilweise in der Klasse „FCDLoader“ zu festgelegt sind, um die Auswahl zu vereinfachen.

Die zweite Aufgabe erfolgt durch Aufruf der Methode „loadAndAnalyse()“. In dieser Methode wird die gesamte Prozesskette des Ladens und Analyiserens durch Aufrufen der entsprechenden Klassen und Methoden durchlaufen. Dieses sind im Wesentlichen 4 Schritte:

1. Laden des Kartenmaterials mittels Kartenlademodul
2. Laden der Floating Car Data aus Datenbank oder Textdatei
3. Hinzufügen der FCD- und Kartendaten zur Zuordnungsmethode, gegebenenfalls Vorverarbeitung
4. Zuordnen der FCD-Positionen zu Knoten und Analysieren der FCD-Treffer

Zurückgegeben wird eine Hashtable, die, im Falle einer erfolgreichen Analyse, Knoten-IDs und zugehörige Bewertungen in Form von double-Werten von 0 bis 100 enthält.

Die einzelnen Aufgaben werden nicht von der Klasse selbst übernommen, sondern von eigenen Klassen, die dies der Übersichtlichkeit und der Austauschbarkeit wegen aus dem Hauptablauf kapseln.

6.4.3 Datenhaltung – Laden der Floating Car Data

Um die Austauschbarkeit zu gewährleisten, werden die verschiedenen Arten des Ladens der Daten in der abstrakten Klasse „FCDLoader“ zusammengefasst. Hier wird zunächst definiert, welche Form die geladenen Daten haben. Aufgrund des hohen Speichervolumens der Floating Car Data wurde als Datenstruktur hier eine Ansammlung von Arrays gewählt. Diese enthalten:

- Longitude und Latitude der Geoposition
- Identifikationsnummer
- Geschwindigkeit
- Stauts
- Winkel und
- Zeitstempel

für jede FCD-Position. Wichtig dabei ist, dass eine gleiche Indexnummer in jedem Array den entsprechenden Wert für die selbe FCD-Position liefert. Übersichtlicher wäre es an dieser Stelle gewesen, statt einfacher Arrays Objekte zu verwenden, die die FCD-Positionen repräsentieren. Das Datenvolumen hätte jedoch aufgrund des Overheads von Java-Objekten zu Speicherproblemen führen können.

Festgelegt ist des weiteren die Methode für das Laden, „loadFCD()“, inklusive ihrer benötigten Parameter. Get-Methoden für die geladenen Daten ersparen das erneute Festlegen dieser Methoden in den Subklassen. Dies ist auch der Grund, dass hier eine

abstrakte Klasse verwendet wurde anstatt eines Interfaces.

Direkt implementiert sind die Subklassen „SQLFCDLoader“ und „TextFileFCDLoader“. Diese lassen sich auch über eine Referenz auf die Klasse „FCDLoader“ verwenden.

„SQLFCDLoader“ ist die Klasse, die das Laden aus der SQL-Datenbank ermöglicht. Sie implementiert das Interface „SQLData“, in dem Datenbankname, Benutzername, Passwort und IP der Datenbank angegeben sind. „SQLFCDLoader“ verwendet das Paket „java.sql“. Aus den Angaben des Auswertungszeitraumes, des Tabellennamens und der Statusrestriktionen wird ein String zusammengesetzt, der eine SQL-Datenbankabfrage repräsentiert. Statusrestiktionen sind dabei auf das Nürnberger System zugeschnitten. Sollen z.B. nur Positionen geladen werden, bei denen die Taxis den Status „frei“ (70) und „besetzt mit Fahrziel“ (90) angegeben haben, wird am Ende des Statements der String

„AND ((status = 70) OR (status = 90))“

angehängt. In anderen Städten könnten die Statusziffern vom Nürnberger System differieren. Hierbei wäre es möglich, eine entprechend angepasste Klasse für ein anderes System zu schreiben, unter Verwendung des bereits existierenden Codes.

Nach dem Laden der Daten werden diese zunächst in Vector-Objekten zwischengespeichert, um nach Ermittlung der Anzahl der Daten in die Arrays übertragen werden zu können.

„TextfileFCDLoader“ arbeitet äquivvalent dazu, mit dem Unterschied, dass zunächst alle Daten zeilenweise aus einer Textdatei ausgelesen werden. Zeilen werden mit einem StringTokenizer in die einzelnen Felder getrennt. Danach wird anhand von Datum und Statusrestriktionen entschieden, welche Daten zunächst in Vector-Objekte, nach Ermittlung der Anzahl in Arrays übernommen werden.

6.4.4 Vorverarbeitung

Um mehrere Vorverarbeitungsmethoden zu ermöglichen, wurde die Vorverarbeitung ebenfalls über das Interface „PreProcessing“ austauschbar gemacht. Das Interface enthält die Definition für die Methode „processFCD()“ und die get-Methoden für die einzelnen Arrays.

Um es zu ermöglichen, dass mehrere Vorverarbeitungsschritte nacheinander, nur eine Vorverarbeitung oder gar keine Vorverarbeitung erfolgen, ist das Interface so ausgelegt, dass die Form der Eingangsdaten gleich der Form der Ausgangsdaten ist. Das heißt, es werden genau die Arrays verwendet, die schon aus der Klasse „FCDLoader“ bekannt sind.

Als einzige Subklasse ist die Klasse „AssignNewIDs“ implementiert. „AssignNewIDs“ hat die Aufgabe, neue Fahrzeug-IDs zu verteilen, die ein mehrfaches Auftauchen der Ursprungs-Fahrzeug-ID verhindern, und Fahrzeugen neue Identifikationsnummern geben, sobald sie den Status wechseln bzw. über längere Zeit keine Position senden.

Der Algorithmus durchläuft zwei ineinander geschachtelte Zählschleifen. Die äußere Schleife

durchläuft so lange alle FCD-Positionen, bis eine noch nicht verarbeitete Position gefunden wurde. Verarbeitete Positionen haben die ID -1, nicht verarbeitete Positionen die originale ID. Wird eine nicht verarbeitete ID gefunden, wird eine neue ID festgelegt und die innere Zählschleife beginnt.

Die innere Zählschleife soll die neue ID allen Positionen zuordnen, die zu der letzten in der äußeren Schleife gefundenen ID gehören. Die Zugehörigkeit wird nach 3 Kriterien bestimmt, die nacheinander ausgeschlossen werden.

Das erste Kriterium ist die Zeit. Die Eingangsdaten sind immer nach der Zeit geordnet. Taucht eine Position auf, die mehr als ein bestimmtes Zeitintervall von der letzten verarbeiteten Position entfernt ist, kann keine zugehörige Position mehr auftauchen. Die Schleife bricht ab.

Das zweite Kriterium ist die gleiche Original-ID. Wird eine andere ID gefunden, gehört diese nicht zur aktuell zu verarbeitenden ID, und die innere Schleife fährt so lange fort, bis die aktuelle ID gefunden wurde.

Das letzte Kriterium ist der Status. Eine Position gehört nur so lange zur aktuellen neuen ID, wie kein Statuswechsel für die originale ID vorgenommen wurde. Ist der Staus ein Anderer als der Originalstatus, bricht die Schleife ab.

Geschachtelte Schleifen über viele Datensätze erzeugen meist einen hohen Aufwand. Im „worst case“ ist der Aufwand auch hier sehr hoch. Dieses wäre der Fall, wenn Alle Fahrzeuge über den gesamten Betrachtungszeitraum kontinuierlich Positionen mit dem gleichen Status erzeugt hätten. In diesem Fall wäre der Aufwand circa

$$((m+1)*n)*c \quad ,$$

mit

$$m = Anzahl\ der\ Fahrzeuge\ ,\ n = Anzahl\ der\ Positionen\ ,\ c = ungefährer\ Rechenaufwand \quad .$$

Damit also nicht quadratisch.

In der Praxis sind aber Statuswechsel und große Zeitlücken vorhanden, so dass die innere Schleife im Regelfall schnell zum Abbruch kommt, und die äußere Schleife viele Positionen überspringt. Bei der Ausführung kostet der Algorithmus auch bei größeren Datensätzen verhältnismäßig wenig Zeit.

6.4.5 Zuordnung von Positionen zu Knotenpunkten

Nach der Vorverarbeitung erfolgt die Zuordnung von FCD-Positionen zu Knotenpunkten. Es sind 3 verschiedene Zuordnungsmethoden implementiert, die jedoch von der Anwendung her äquivalent verwendbar sein sollen. Daher haben die Zuordnungsmethoden das gemeinsame Interface „Assignment“. In diesem Interface ist beschrieben, welche Methoden eine Klasse haben muss, die FCD-Positionen zu Knotenpunkten zuordnet. Dies sind Methoden für das Übergeben der Kartendaten (Kanten und Knoten) sowie das Setzen der

FCD-Positionen in Form von Arrays, das Zuordnen an sich und das Auslesen einiger statistischer Werte, wie z.B. die Anzahl der „Outliers“ (Positionen, die außerhalb des zu analysierenden Bereiches liegen).

Der Rückgabewert der Methode „analyse()“ ist eine Hashtable. Diese enthält als Schlüssel die KnotenID und als Wert einen Vector, der „FCDHit“ Objekte enthält. „FCDHit“ Objekte enthalten Informationen zu einer zugeordneten Position, die wichtig für die weitere Verarbeitung sind. Dieses sind Fahrzeug-ID, Status und Geschwindigkeit der Position. Diese Datenstruktur ist sehr speicheraufwendig, erfüllt aber die Anforderung, dynamisch zu sein, da die Anzahl der FCD-Treffer für einen Knoten nicht im Voraus bekannt ist.

„SimpleRasterisationAssignment“ ist die erste implementierte Zuordnungmethode. Sie geht davon aus, dass das gesamte zu betrachtende Gebiet mit einem Raster unterteilt wird, und Knotenpunkte sowie FCD-Positionen, die sich in einem gemeinsamen Rasterteil befinden, zueinander zugeordnet werden.

Im ersten Schritt erstellt die Methode „analyse()“ ein zweidimensionales Array, das Vector-Objekte enthält. Es repräsentiert das Raster. In den Vector-Objekten werden die Indizes der FCD-Positionen abgelegt, die die jeweilige Stelle in den Eingangsdaten-Arrays repräsentieren. Die Indizes für das zweidimensionale Array werden dabei aus den Geopositionen der FCD-Positionen errechnet.

Im zweiten Schritt geht eine Schleife alle Knotenpunkte des Kartenmaterials durch. Aus den Geopositionen der Knotenpunkte wird wieder errechnet, an welcher Stelle des Rasters sich die FCD-Positionen befinden. Diese werden dem Knoten dann zugeordnet, indem die Hashtable für die Rückgabe entsprechend befüllt wird.

Der Aufwand ist bei dieser Zuordnungsmethode

$(m*c1)+(n+c2)$ mit

$m = Anzahl\ der\ FCD-Positionen\,,\ n = Anzahl\ der\ Knotenpunkte\,,$

$c1 = ungefährer\ Rechenaufwand\ der\ ersten\ Schleife\,,$

$c1 = ungefährer\ Rechenaufwand\ der\ zweiten\ Schleife$.

Die zweite Zuordnungsmethode ist in der Klasse „HitsInNodeProximityAssignment“ implementiert. Diese Klasse ordnet jedem Knoten alle FCD-Positionen mit einer bestimmten maximalen Entfernung zu. Theoretisch müssten hier für jeden Knotenpunkt alle FCD-Positionen überprüft werden. Dies würde einen Aufwand von

$m*n*c$ mit

$m = Anzahl\ der\ FCD-Positionen\,,\ n = Anzahl\ der\ Knotenpunkte\,,$

$c = ungefährer\ Rechenaufwand$

bedeuten. Bei den extremen Datenmengen könnte dies dazu führen, dass die Zuordnung unnötiger Weise mehrere Stunden oder Tage benötigt.

Um die ungefähre Lage der FCD-Positionen von vornherein besser bestimmen zu können, werden diese wieder in ein zweidimensionales Array verteilt, das ein Raster repräsentiert. Die Seiten eines Rasterteils müssen dabei größer sein als die maximale Zuordnungsdistanz. Diese Einteilung geschieht analog zur ersten Zuordnungsmethode.

Im zweiten Schritt wird für jeden Knotenpunkt die Position des Rasterteils errechnet. Da der Knoten auch am Rand des Rasterteils liegen kann, werden auch die 8 umliegenden Rasterteile betrachtet. Für diesen Knoten wird nun die euklidische Distanz zu allen Positionen der betreffenden 9 Teile des Rasters berechnet. Alle Positionen, die innerhalb des festgelegten Radius liegen, werden dem Knoten zugeordnet. Bei der Berechnung wird die approximierte Distanz in Metern betrachtet, nicht die Distanz in Geokoordinaten.

Die Klasse „NearestNodeAssignment“ enthält die dritte Zuordnungsmethode. Diese ordnet jeder FCD-Position dem nächstliegenden Knotenpunkt zu.

Der theoretische Aufwand wäre dabei wieder sehr groß, da für eine genaue Zuordnung jede FCD-Position mit jedem Treffer verglichen werden müßte, also

$m * n * c$ mit

$m = Anzahl\ der\ FCD - Positionen\ ,\ n = Anzahl\ der\ Knotenpunkte\ ,$

$c = ungefährer\ Rechenaufwand$.

Zur Verringerung dieses Aufwands kann jedoch das Kartenmaterial in ein Raster unterteilt werden, um für jede FCD-Position die Anzahl der in Frage kommenden Knoten einzuschränken. Nachteil ist, dass FCD-Positionen, die eine größere Distanz zum nächsten Knoten haben, als die Länge einer Seite des Rasters beträgt, gegebenenfalls keinem Knoten zugeordnet werden können. Voreingestellte Länge eines Rasterteils sind ca. 200 Meter.

Die Verteilung der Knotenpunkte auf das Raster geschieht, indem IDs der Knotenpunkte in ein Array geschrieben werden, das Vector-Objekte enthält. Jeder Knoten wird seinem Rasterteil sowie den 8 umliegenden Rasterteilen zugeordnet. Jedes Teil des Rasters enthält somit die IDs von allen Knoten, die sich im selben bzw. in einem der umliegenden 8 Rasterteile befinden.

Im zweiten Schritt wird über alle FCD-Positionen iteriert. Für jede FCD-Position wird ermittelt, welche Knoten für eine Zuordnung in Frage kommen. Dazu wird der Vector aus dem Raster betrachtet, der die IDs der umliegenden Knoten enthält. Die Distanz zwischen den Knoten und einer Position wird vergleichen. Dem Knoten, der die geringste Distanz zur Position hat, wird diese als FCD-Treffer zugeordnet, indem wiederum die Hashtable für die Rückgabe entsprechend befüllt wird.

Da der Zeitaufwand trotz des Rasterns der Knoten noch recht hoch ist, wird auf der Konsole der Fortschritt ausgegeben. Dies erfolgt in 5%-Schritten, die sich aus der Anzahl der abgearbeiteten FCD-Positionen ergeben.

6.4.6 Auswertung von Knotenpunkten mit FCD-Treffern

Für das Auswerten von Knotenpunkten, denen FCD-Treffer zugeordnet sind, wurden 6 Methoden implementiert. Diese werden über das Interface „Quantification“ zusammengefasst, um einheitlich und austauschbar zu sein. Das Interface definiert lediglich die Methode „quantify()“mit Parametern und Rückgabewert. Eingabeparameter ist eine Hashtable, die Knoten-IDs als Schlüssel und Vector-Objekte mit FCDHit-Objekten als Elementen enthält, also entsprechend dem Rückgabewert der Zuordnungsmethoden. Rückgabewert des Auswertung ist eine Hashtable mit KnotenIDs als Schlüssel und einem Double-Wert, der eine Angabe von 0 („unkritisch“ bis 100 „kritisch“) repräsentiert.

Die Auswertungsmethoden lassen sich in 3 Bereiche teilen, wobei die Auswertungsmethoden eines Bereiches leichte Differenzen in der Bewertung enthalten dies sind:

- Zählen der FCD-Treffer mit linearer Bewertung nach Normierung, linearer Bewertung nach Normierung auf maximalen Schwellwert und logarithmisch eingeteilter Bewertung,
- Berechnen der Durchschnittsgeschwindigkeiten mit linearer Einteilung und
- Errechnen der FCD-Treffer pro Fahrzeug-ID mit linearer und logarithmischer Einteilung

Der Ablauf der Auswertung ist im Groben bei allen Auswertungsmethoden gleich und erfolgt in 2 Schritten:

1. Iterieren über alle Knoten mit FCD-Treffern zur Bestimmung des Maximums des Bewertungskriteriums
2. Iterieren über alle Knoten, wobei das Bewertungskriterium für jeden Knoten ins Verhältnis zum Maximum gesetzt wird

Die ersten 3 Auswertungsmethoden sind in den Klassen „HitNumberPercentageQuantification“, „HitNumberPresetMaximumPercentageQuantification“ und „HitNumberLogarithmicQuantification“ implementiert. Diese bewerten die absolute Anzahl von FCD-Treffern, die jeder Knoten erhalten hat.

Zunächst wird bei allen Methoden das Maximum von FCD-Treffern ermittelt. Dieses ist gleich der maximalen Größe eines der Vector-Objekte, die sich in der Hashtable der Eingangsdaten befinden. Bei der logarithmischen Einteilung wird anstatt der absoluten Vektorgröße deren Logarithmus dualis als Maximum gewertet.

Nach der Ermittlung des Maximums wird für jeden Knoten die Zahl der erhaltenen FCD-Treffer ermittelt, wodurch der Knoten eine Bewertung erhalten kann.

In der Klasse „HitNumberPercentageQuantification“ wird dabei die Bewertung durch die Formel

$$Bewertung = \frac{(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 100$$

berechnet. Werte unter 5 werden verworfen.

„HitNumberPresetMaximumPercentageQuantification“ benutzt einen voreingestellten Schwellwert, um Ausreißer zu eliminieren. Dieser ist in der Implementierung als 25% des Maximums festgelegt. Die Formel erweitert sich damit auf

$$Bewertung = \frac{(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 100 * \frac{100}{Schwellwert}$$ bezie

hungsweise

$$Bewertung = \frac{(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 400$$. Da somit Werte

von bis zu 400 entstehen können, werden alle Werte über 100 auf 100 heruntergesetzt.

Die Klasse „HitNumberLogarithmicQuantification“ benutzt den Logarithmus dualis, rechnet also

$$Bewertung = \frac{\log_2(Anzahl\ der\ FCD-Treffer\ der\ Knotens)}{\log_2(Maximale\ Anzahl\ FCD-Treffer\ für\ einen\ Knoten)} * 100 \quad .$$

Das jeweilige Ergebnis wird in einer Hashtable zurückgegeben.

„AverageSpeedsQuantification“ ist die Klasse, die die Durchschnittsgeschwindigkeiten der Knoten berechnet. Für diese Auswertungsmethode muss kein Maximum bestimmt werden, da kritische und unkritische Geschwindigkeit vordefinierten Werten entsprechen. In der Implementierung wird 15 km/h als unkritisch und 0 km/h als kritisch angenommen.

Für jeden Knoten wird die Durchschnittsgeschwindigkeit ermittelt und bewertet. Die Ermittlung der Durchschnittsgeschwindigkeit erfolgt dabei nach der Formel

$$\frac{1}{Anzahl\ der\ FCD-Treffer\ des\ Knoten} \sum_{i=1}^{Anzahl\ der\ FCD-Treffer\ des\ Knoten} (Geschwindigkeit_i) \quad .$$

Um eine Bewertung von 0 bis 100 zu ermöglichen, wobei 100 „kritisch“ und 0 „unkritisch“ ist, wird folgende Formel verwendet:

$$Bewertung = 100 - (\frac{100}{(unkrit.\ Wert - krit.\ Wert)}) * (Durchschn.\ Gesch.\ d.\ Kn. - krit.\ Wert) \quad .$$

Mit den vorgegebenen Werten entspricht dies:

$$Bewertung = 100 - \frac{100}{15} * Durchschnittliche\,Geschwindigkeit\,des\,Knotens \quad .$$

Werte über 100 werden auf 100 normiert, Werte unter 0 (ausgehend von Geschwindigkeiten, die höher sind als der unktitische Wert) werden verworfen. Das jeweilige Ergebnis wird in einer Hashtable zurückgegeben.

Die Klassen „HitsPerVehiclePercentageQuantification“ und „HitsPerVehicleLogarithmicQuantification“ enthalten die Auswertungsmethoden, die nach FCD-Treffern pro Fahrzeug-ID bewerten.

Im ersten Schritt wird für jeden Knoten über dessen Vector mit den Eingangsdaten iteriert. Jede ID wird in eine Hashtable geschrieben. Dabei wird der Effekt genutzt, dass eine Hashtable einen Wert überschreibt, wenn der Schlüssel für den Wert bereits existiert. Die Größe der Hashtable ist somit gleich der Anzahl der vorhandenen Fahrzeug-IDs für den Knoten. Die Anzahl der gefundenen FCD-Treffer wird nun durch die Anzahl der vorhandenen Fahrzeug-IDs dividiert. Der größte gefundene Wert wird als Maximalwert gespeichert. Bei der logarithmischen Einteilung wird der Logarithmus dualis anstatt des absoluten Wertes der FCD-Treffer pro FahrzeugID verwendet.

Im zweiten Schritt der Auswertung wird wieder über alle Knoten mit FCD-Treffern iteriert. Für jeden Knoten leitet sich die Bewertung nun von folgender Formel ab:

$$Bewertung = \left(\frac{\frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}}{Maximum\left(\frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}\right)}\right) * 100$$

für die linerare und:

$$Bewertung = \left(\frac{\log_2 \frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}}{\log_2 Maximum\left(\frac{FCD-Treffer\ pro\ Knoten}{Anzahl\ der\ Fahrzeug-IDs}\right)}\right) * 100$$

für die logarithmische Variante.

Die Bewertungen werden als Werte mit den Knotenpunkt-IDs als Schlüssel in einer Hashtable gespeichert und zurückgegeben.

6.4.7 Histogramm

Zur Darstellung und Visualisierung der Ergebnisse existieren 2 Klassen, mit deren Hilfe sich ein Histogramm darstellen lässt. Diese heissen „Histogram“ und „HistogramFrame“.

„Histogram“ ist die Klasse, die die Werte eines Histogramms berechnet und speichert. Sie bekommt im Konstruktor ein Vector-Objekt von Double-Werten übergeben. Diese Werte werden nun in 101 Klassen aufgeteilt, so dass die Klassenbreite 1% entspricht, von 0 bis

100.

Im ersten Schritt wird über alle Eingangswerte iteriert, um das Maximum zu finden.

Alle Werte werden nun in ein Array aus 101 int-Werten aufgeteilt, und zwar, indem für jeden Wert die entsprechende Stelle im Array incrementiert wird. Der Index des Array ist dabei der Wert, der sich aus

$$100 * (\frac{Eingangswert}{Maximalwert})$$

ergibt (gerundet per cast auf int).

Gleichzeitig werden alle Werte addiert, um sie dann durch die Anzahl der Werte zu dividieren. Dies ergibt das arithmetische Mittel.

Zum Schluß wird über die Elemente des Array iteriert, wobei die Wert von einer Variablen „medianCounter“ abgezogen werden. Diese wurde als Hälfte der Anzahl der Eingangselemente initialisiert. Ist „medianCounter“ 0, wird die Schleife abgebrochen. Der Aktuelle Index zeigt nun auf das Element des Array, in dem die Klasse des Medianwertes liegt. Somit ergibt sich ein approximierter Medianwert.

„Histogram“ verfügt ebenfalls über die entsprechenden get-Methoden, um die errechneten Werte außerhalb der Klasse verfügbar zu machen.

„HistogramFrame“ ist eine von JFrame abgeleitete Klasse. Sie dient dazu, ein Histogramm graphisch dartzustellen.

Dem Konstruktor wird ein Objekt der Klasse „Histogram“ übergeben, sowie Strings für die Beschriftung von Titel, x-Achse, y-Achse und einem boolean, der angibt, in welcher Richtung die Klassen des Histogramms von Rot nach Grün eingefärbt werden sollen.

Im Konstruktor wird ein Canvas mit überschriebener Paint()-Methode erzeugt. Die Paint()-Methode ruft „paintHistogramImage()“ auf. Hier wird das Histogramm gezeichnet. Mit dargestellt werden Median, Mittelwert und Achsenbeschriftung in 1/4-Schritten. Die Farbgebung wird durch die Methode „calculateColor()“ berechnet.

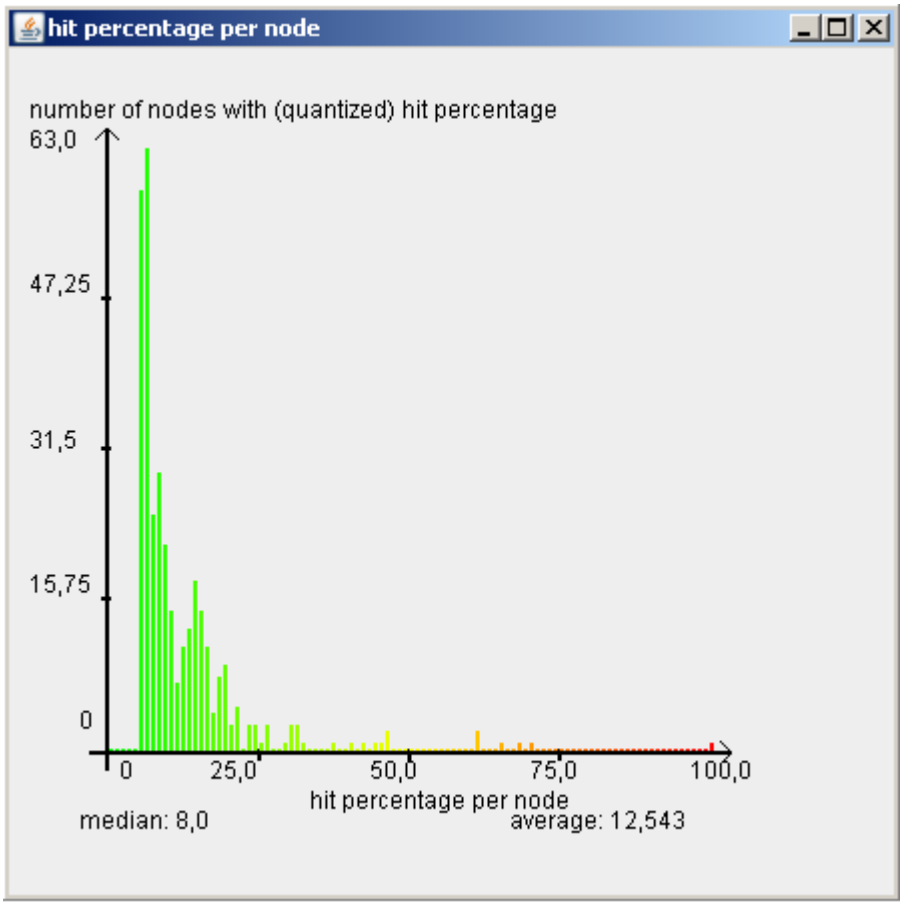

Abbildung 41: Beispiel für ein gezeichnetes Histogramm

Dieses Bild zeigt ein Beispiel für ein Histogramm, das mittels eines „HistogramFrame“ dargestellt wird.

Histogramme werden von den Auswertungsmethoden aufgerufen, um die Verteilung der auf der Karte dargestellten Daten zu visualisieren und gleichzeitig eine Legende zu ersetzen, so dass der Betrachter das Ergebnis besser einordnen kann.

7 Einordnung der Ergebnisse der Knotenpunktanalyse

Die Anforderung, Knotenpunkte im Kartenmaterial, die im Hinblick auf FCD gestaute Verkehrsmuster aufweisen, zu identifizieren, ist durch die Funktionsweise der Software erfüllt. Im Rahmen der möglichst einfachen Betrachtung von Knotenpunkten unter Nichtbetrachtung der Straßenkanten, was auch Mapmatching-Verfahren ausschließt, liefert die Software Ergebnisse mit unterschiedlichen, differenziert zu betrachtenden Methoden. Das Ziel, experimentell zu bestimmen, was mit solchen Methoden möglich ist, wurde erfüllt. Die Ergebnisse lassen sich, theoretisch betrachtet, nachvollziehen.

Allerdings liefert eine theoretische Betrachtung noch keine Aussage über den Realitätsbezug der Ergebnisse. Ein Realitätsbezug ist jedoch schwer herstellbar, da die Vergleichsmöglichkeiten sehr beschränkt sind.

Für eine Betrachtung des Realitätsbezuges müssen zunächst die Methoden für die Zuordnung von FCD-Positionen zu Knotenpunkten betrachtet werden. Ausschlaggebend in der Betrachtung der Ergebnisse ist die Methode „Zuordnung der FCD-Positionen zum nächstliegenden Knoten“. Der Grund für die Verwendung dieser Methode ist, dass diese Art der Zuordnung ohne Betrachtung von Straßenkanten am genauesten ist. Umfasst eine Kreuzung oder Einmündung im Kartenmaterial mehrere Knotenpunkte, so werden die Positionen dem Nächstliegenden zugeordnet. Unter der Voraussetzung, dass die Positionen genau sind, sollte damit auch verhindert werden, dass FCD-Positionen falschen Knoten, wie beispielsweise bedingt durch die Fahrtrichtung, zugeordnet werden. Falsche Zuordnungen lassen sich jedoch nicht ausschließen. Auch Fahrzeuge, die einen Knoten bereits passiert haben, könnten diesem zugeordnet werden. Die Methoden „Zuordnung durch einfaches Rastern“ und „Zuordnung aller Positionen in Knotennähe“ sind dennoch wesentlich ungenauer und Fehleranfälliger. Sie sind in Betrachtung des Ergebnisses somit lediglich als experimentelle Zwischenschritte in der Entwicklung zu sehen.

Zu betrachten sind nun die Auswertungsmethoden. Zunächst lässt sich zur Methode des Auswertens der Durchschnittsgeschwindigkeit eine Aussage treffen. Bedingt dadurch, dass die Sendeintervalle der FCD-Positionen für diesen Algorithmus zu groß sind, um repräsentativ zu sein, und die Aufnahme einer Position zu oft in einem Moment stattfindet, in dem das Fahrzeug steht, ist die Methode mit den reinen Rohdaten unbrauchbar. Betrachtet werden müssten hier optimaler Weise ganze Trajektorien geglätteter Daten, um ein gutes Ergebnis zu erhalten. Eine Sichtprüfung des Ergebnisses zeigt, unabhängig vom Betrachtungszeitraum, meist eine Häufung von als kritisch klassifizierten Knoten im Bereich von Nebenstraßen, während Hauptstraßen weniger kritische Punkte enthalten. In der Realität sind Nebenstraßen jedoch weitaus seltener von Verkehrsproblemen betroffen als die großen Hauptstraßen.

Bleibt der Vergleich des reinen Auszählens und Einordnens der Anzahl der FCD-Treffer mit der Bestimmung von Treffern pro Fahrzeug. Theoretisch betrachtet sollte die Methode, die

von den Positionen pro Fahrzeug ausgeht, objektiver und genauer sein. Bei einer Sichtprüfung liefern beide Methoden, im Gegensatz zur Auswertung der Momentangeschwindigkeiten, glaubhafte Ergebnisse, wobei die Auswertung von Positionen pro Fahrzeug erstaunlich wenige Punkte liefert, die als „kritisch“ eingeordnet werden.

Als Vergleichsmöglichkeit soll nun die Bewertung der Verkehrslage basierend auf einer grundsätzlich anderen Methode herangezogen werden. Eine Möglichkeit dazu bietet die für die Stadt Nürnberg vom DLR entwickelte Verkehrslageerfassungssoftware „QS-Tool“. Ebenfalls auf FCD basierend stellt diese Software die Verkehrslage in der Stadt Nürnberg graphisch dar. Eine Variante dazu ist die Darstellung einer Auswertung, basierend auf Momentangeschwindigkeiten von FCD-Positionen. Die FCD-Positionen wurden in der FCD-Prozessklette vor verarbeitet und mittels eines Mapmatching-Verfahrens den Kanten zugeordnet. In einem Test werden die FCD eines gesamten Tages, beispielhaft am 05.05.2008, ausgewertet und die Ergebnisse anhand der graphischen Darstellungen verglichen.

Das „QS-Tool“ liefert folgende Ausgabe:

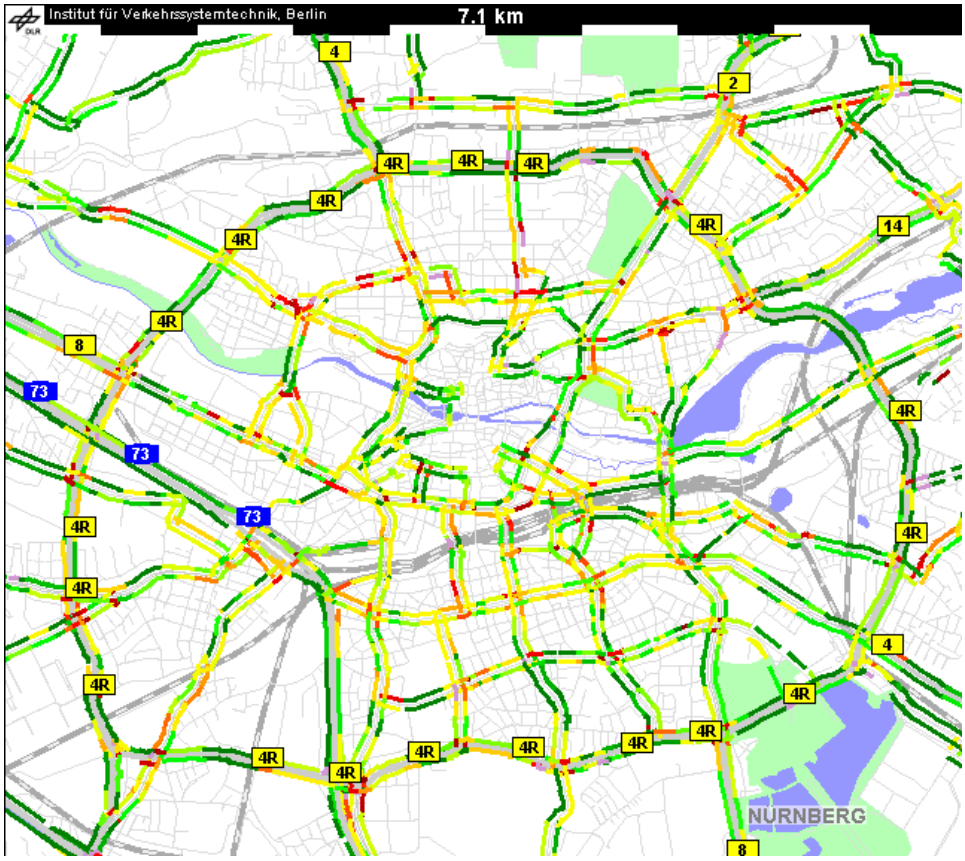

Abbildung 42: Verkehrslage in Nürnberg laut QS-Tool, aggregierte Daten vom 5. Mai 2008

Im Vergleich dazu die Ausgabe der Knotenpunktanalyse durch Zählen der FCD-Treffer vom gleichen Zeitraum:

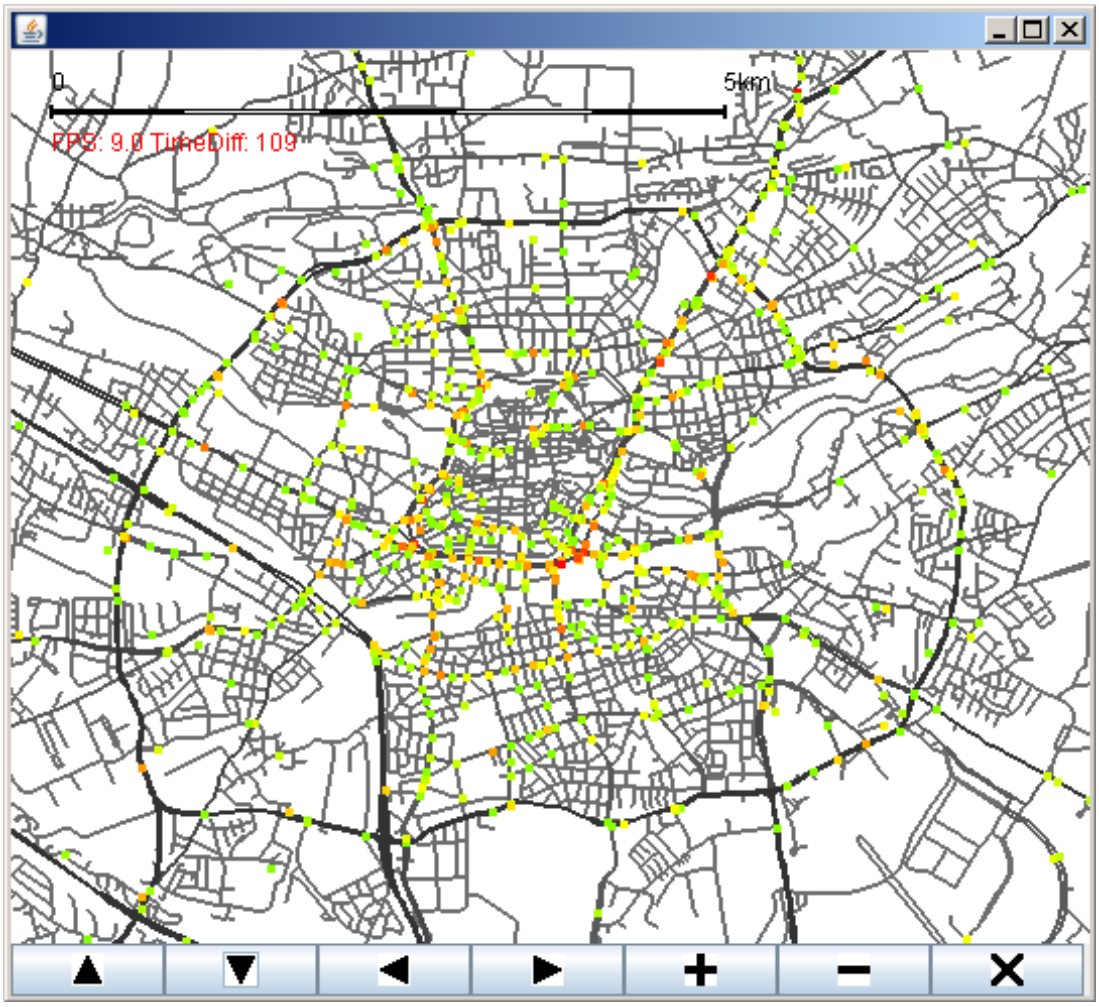

Abbildung 43: Durch Zählen der FCD-Treffer als "kritisch" identifizierte Knotenpunkte in Nürnberg, 5. Mai 2008

Unabhängig von der Skalierung und Einfärbung der Ergebnisse lässt sich erkennen, dass es trotz der unterschiedlichen Art der Auswertung (Zählen der FCD-Positionen bzw. Bewertung durch Momentangeschwindigkeiten) viele Übereinstimmungen in beiden Darstellungen gibt. Besonders auffällig sind die rot eingefärbten Punkte in der Darstellung der Knotenpunktanalyse. In ihrer Nähe lassen sich im QS-Tool violett eingefärbte Kanten erkennen.

Als Vergleich dazu die Auswertung der FCD-Treffer pro Fahrzeug:

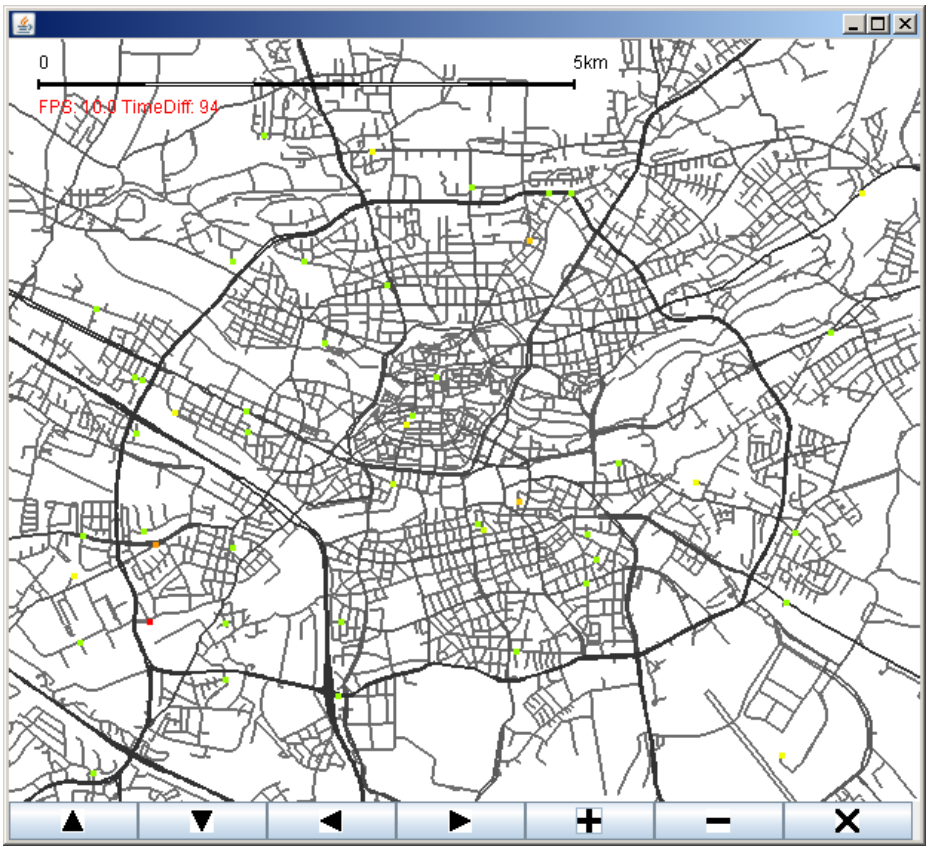

Abbildung 44: Durch Auswertung der FCD-Treffer pro Fahrzeug als "kritisch" identifizierte Knotenpunkte in Nürnberg, 5. Mai 2008

Es ist auffällig, dass es in dieser Darstellung keine Übereinstimmungen mit dem QS-Tool gibt. Die Annahme, dass die Auswertung der FCD-Treffer pro Fahrzeug zu einem besseren Ergebnis führe, ist demzufolge falsch.

Die Gründe dafür könnten in verschiedenen Bereichen zu suchen sein. Eine nahe liegende Vermutung ist die Ungenauigkeit der FCD-Rohdaten im Bezug auf ihr Sendeintervall. Wenn punktuell an Knoten viele FCD-Treffer zugeordnet werden, die nahezu unmittelbar nacheinander gesendet wurden, steigt der Wert der Treffer pro Fahrzeug. Auch Statuswechsel der Taxis, die nicht angegeben wurden (Zusteigen eines Fahrgastes, Aussteigen,...) können mitverantwortlich für das Ergebnis sein. Dies zeigt, dass eine bessere Vorverarbeitung der Daten für die Weiterverfolgung dieses Ansatzes unbedingt notwendig wäre.

Der Vergleich des Auszählens der FCD-Treffer mit der Darstellung des QS-Tools zeigt jedoch, dass auch mit einer sehr einfachen Methode Knotenpunkte mit gestauten Verkehrsmustern in Straßennetzen zumindest annähernd realistisch identifizieren kann (Dies setzt die Genauigkeit des QS-Tools voraus. Da aber ein anderer Aspekt der FCD im QS-Tool verarbeitet wurde, erscheint das Ergebnis im Vergleich als glaubwürdig.).

Bezüglich der Genauigkeit der Auswertung könnte nun überprüft werden, ob die IDs der

Knotenpunkte, die identifiziert wurden, sich mit den Knotenpunkt-IDs der kritisch bewerteten Kanten aus dem QS-Tool decken.

Der Vergleich der Methoden lässt jedenfalls die Aussage zu, dass ein Algorithmus gefunden wurde, der gestaute Verkehrsmuster in Straßennetzen identifiziert. Durch die Art der Auswertung kann also darauf geschlossen werden, dass Stellen in einem Straßennetz, in dem sich die FCD-Positionen aus den Rohdaten häufen, tatsächlich mit Verkehrsbeeinträchtigungen belastet sind. Die so ermittelten Knotenpunkte können also genutzt werden, um weitere Betrachtungen, beispielsweise in anderen Forschungsprojekten, durchzuführen.

8 Zusammenfassung und Zukunftsaussichten

Das Ziel der Arbeit war es, Knotenpunkte mit gestauten Verkehrsmustern in einem Straßennetz zu finden und zu visualisieren.

8.1 Kartenmaterial und Visualisierung

Als Grundlage war es Teil der Aufgabe, einer wiederverwendbare Schnittstelle für die Visualisierung von Vektorkarten zu erstellen, die mit komprimiertem Kartenmaterial umgehen kann. Ergebnis dieser Teilaufgabe sind 3 Softwarekomponenten, die als Prototypen zu betrachten sind.

Die Kartenkompression funktioniert in der angedachten Art und Weise und komprimiert Kartenmaterial auf einen Bruchteil seiner Ausgangsgröße, bringt es gleichzeitig auch in eine Form, in der es sich nach geographischen Gebieten differenziert laden lässt. Weitere Schritte in der zukünftigen Arbeit mit der Kartenkompression sind davon abhängig, in welcher Form Kartenmaterial in Zukunft benötigt wird und welche Informationen enthalten sein müssen. Mit diesen Anforderungen ließe sich das Kompressionstool anpassen und weiter verwenden. Der große Vorteil gegenüber der bisherigen Speicherung nach Regionen liegt in der Möglichkeit, Kartenmaterial großer Gebiete differenziert verwenden zu können.

Das Kartenlademodul implementiert eine Methode zum Laden des Kartenmaterial innerhalb einer Software, die auf der selben Maschine läuft, auf der das Kartenmaterial gespeichert ist. Denkbar wäre hier eine Erweiterung, die es ermöglicht, für den Einsatz in Applets das Kartenmaterial über ein Netzwerk zu übertragen. Das Modul ist so allgemein gehalten, dass eine Erweiterung ohne Weiteres implementierbar ist. Der Nachteil am modularen Aufbau dieser Softwarekomponente ist die Mehrstufigkeit ihrer Arbeitsweise. Eine einfachere Implementierung könnte unter Umständen Rechenzeit sparen, wenn weniger Verarbeitungsschritte notwendig wären. Diese Softwarekomponente könnte für einen zeitkritischen Einsatz in Hinsicht auf die Performance noch weiter optimiert werden.

Gleiches gilt für die Visualisierungsschnittstelle. Sie ist ebenfalls als Prototyp zu betrachten. Aus dem Kartenzeichnen ohne Hilfe von Schnittstellen wie OpenGL ergibt sich ein Geschwindigkeitsmalus, dafür ist das Modul theoretisch auf allen grafikfähigen Endgeräten mit Java-Unterstützung einsetzbar. Die implementierte Funktionalität des Moduls ist universell und, wie auch die Kompressionsmethode, auf die Verwendung mit Kartenmaterial großer Gebiete zugeschnitten. Für den Einsatz in Navigationssystemen wären einige Erweiterungen, wie zum Beispiel eine Darstellung in einem „pseudo-3D-Modus“ denkbar.

8.2 Knotenpunktanalyse

Hauptbestandteil der praktischen Arbeit war die Aufgabe, Knotenpunkte in Straßennetzen zu finden, die gestaute Verkehrsmuster aufweisen. Einschränkungen waren die reine

Betrachtung von Straßenknoten ohne Auswertung von Straßenkanten sowie die Verwendung von unprozessierten Rohdaten. Durch experimentelle Implementierung verschiedener Auswertungsmethoden konnten einige Ansätze zur Lösung dieser Aufgabe ausprobiert werden. Einer der Ansätze zeigt ein Ergebnis, das sich im Vergleich mit bisherigen Verkehrslagedarstellungen aufgrund von FCD als realistisch einstufen lässt.

Verwendbar sind die Ergebnisse, um einen Überblick über die Verkehrslage eines kleineren Zeitraumes (Tage bis Wochen) in einer Stadt zu geben, aus der FCD vorhanden sind. Das Ergebnis liefert Knotenpunkte im Kartenmaterial. Ein solcher Überblick ist für weitere Forschungsansätze verwendbar. Im DLR wird beispielsweise an einem Algorithmus zur Staulängenschätzung gearbeitet. Die identifizierten Knotenpunkte mit dem Ergebnis dieser Arbeit könnten als Eingangsdaten dienen, um mögliche relevante Punkte, für die eine Staulängenschätzung sinnvoll ist, von vornherein zu bestimmen.

Denkbar wäre auch eine erweitere Verkehrslagedarstellung mit Hilfe des Programms. Hieraus könnten sich Optimierungsmöglichkeiten für den Verkehrsablauf in bestimmten Gebieten ergeben.

Weiterhin ist denkbar, die Identifizierung als Basis zur Erstellung von TMC-Punkten für Gebiete ohne solche zu verwenden. Dies würde jedoch eine Erweiterung der Funktionalität bezüglich näherer Informationen zu den Knotenpunkten erfordern.

Falls ein Einsatzgebiet feststeht, für das die Knotenpunktanalyse verwendet werden soll, muss aber gegebenenfalls die Software angepasst werden, so dass beispielsweise die IDs der ermittelten Knoten unabhängig von der Darstellung ausgegeben werden können.

Bezüglich der Genauigkeit zeigen alle implementierten Auswertungsmethoden ihre Unzulänglichkeiten. Durch den modularen Aufbau der Software könnten weitere Schritte implementiert werden, um mit Fehlern besser umgehen zu können. Beispiel dafür sind weitere Implementierungen für die Vorverarbeitung. Das Glätten, Interpolieren und statistische Berichtigen der FCD-Rohdaten könnte Fortschritte in der Genauigkeit bringen.

Ein logischer Schritt für die Verbesserung der Zuordnung der FCD-Treffer wäre das Einbeziehen von Straßenkanten und ein damit verbundenes Map-Matching Verfahren. Dies könnte in der Konsequenz dazu führen, dass weniger FCD-Positionen falschen Fahrtrichtungen oder falschen Knotenpunkten allgemein zugeordnet würden. Mit Hilfe eines solchen Verfahrens könnten dann auch zusätzliche Informationen wie kritische Abbiegerichtungen und Rückstau über mehrere Knoten gewonnen werden. Da bereits ein solches Verfahren im DLR existiert, wäre es sinnvoll, Synergien zwischen beiden Verfahren zu finden und zu nutzen.

Schlussendlich ist es eine Frage des Bedarfs und der nachvollziehbaren Güte des Ergebnisses, in wie weit es praktisch genutzt werden kann. Dass theoretisch eine Lösung für das Problem mit einem solchen einfachen Ansatz möglich ist, wurde aber gezeigt.

9 Quellenverzeichnis

[1]: Deutsches Zentrum für Luft- und Raumfahr e.V. (DLR) in der Helmholtz-Gemeinschaft; **DLR-NACHICHTEN – Maganzin des Deutchen Zentrums für Luft- und Raumfahrt, Sonderheft Verkehr, November 2007;** *Zeitschrift*; Deutsches Zentrum für Luft- und Raumfahr e.V. (DLR) in der Helmholtz-Gemeinschaft 2007

[2]: Forschungsgesellschaft für Strassen- und Verkehrswesen, Arbeitsgruppe Strassenentwurf; **Empfehlungen für die Anlage von Hauptverkehrsstraßen EAHV 93;** *Buchquelle*; Forschungsgesellschaft für Strassen- und Verkehrswesen 1993

[3]: Forschungsgesellschaft für Straßen- und Verkehrswesen; **Handbuch für die Bemessung von Straßenverkehrsanlagen HBS;** *Buchquelle;* Forschungsgesellschaft für Straßen- und Verkehrswesen; FGSV Verlag 2005

[4]: Schäfer, Ralf-Peter; Thiessenhusen, Kai-Uwe; Brockfeld, Elmar; Wagner, Peter; **A traffic information system by means of real-time floating-car data;** *wissenschaftliche Veröffentlichung;* ITS World Congress 20022002; Institute of Transportation Systems, German Aerospace Centre, 2008

[5] Wagner, Peter; Brockfeld, Elmar; Krajzewicz, Daniel; Krieg, Sascha; Neumann, Thorsten; Sohr, Alexander; **How many probe-vehicle data are needed for traffic management applications?;** *Wissenschaftliche Abhandlung, unveröffentlicht*; Institute of Transportation Systems, German Aerospace Centre, 2008

[6]: NAVTEQ; **Webseite NAVTEQ***; Internetquelle*; NAVTEQ; 2008; www.navteq.com; Zugriffsdatum: 12.11.2008

[7]: TomTom International BV; **Webseite TomTom;** *Internetquelle;* TomTom International BV; 2008; http://www.meintomtom.de; Zugriffsdatum: 12.11.2008

[8]: Garmin; **Webseite Garmin;** *Internetquelle;* Garmin Ltd. 2008; http://www.garmin.com; Zugriffsdatum: 12.11.2008

[9]: Falk Marco Polo Interactive GmbH; **Webseite Falk Navigationssysteme;** *Internetquelle;* Falk Marco Polo Interactive GmbH 2008; http://navigation.falk.de; Zugriffsdatum: 11.12.2008

[10]: MEDION Electronics Ltd., **Webseite Medion Navigationssysteme;** *Internetquelle;* MEDION Electronics Ltd. 2008; http://www.mediongopal.co.uk; Zugriffsdatum: 11.12.2008

[11]: Autoren Unbekannt; **Webseite Openstreetmap – FAQ;** *Internetquelle;* OpenStreetMap 2008; http://www.openstreetmap.de/faq.html; Zugriffsdatum: 12.11.2008

[12]: NAVTEQ, **NAVTEQ Kartenmaterial - Customer Technical Reference Guide (CTRG);** *Technische Dokumentation*; NAVTEQ 2004;

[13]: Christian Ullenboom; **Java ist auch eine Insel;** *Buchquelle;* Galileo Computing 2007

[14]: **O2 XDA II Handbuch;** *Technische Dokumentation*; Herausgeber nicht angegeben;

2003

[15]: NSIcom; **Webseite NSIcom**; *Internetquelle*; NSIcom 2009; Zugriffsdatum: 05.01.2009

[16]: Tele Atlas; **Webseite Teleatlas**; *Internetquelle*; Tele Atlas BV 2008; http://www.teleatlas.com; Zugriffsdatum: 12.11.2008

[17]: International Organization for Standardization; **Onlinekatalog ISO, Seite Spezifikation GDF;** *Internetquelle;* 2009, International Organization for Standardization; http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=30763;

[18]: Statistisches Bundesamt; **Verkehr in Deutschland 2006;** *wissenschaftliche Veröffentlichung/Buchquelle;* Statistisches Bundesamt, Wiesbaden 2006

[19]: Th. Ottmann, Albert-Ludwigs-Universität Freiburg; **Analytische Geometrie SS 99 – Das Voronoi-Diagramm;** *Vorlesungsskript;* 1999, http://ad.informatik.uni-freiburg.de/bibliothek/books/ad-buch/k7/slides/11.pdf; Zugriffsdatum: 26.01.2009

[20]: William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery; **Numerical Recipes, The Art of Scientific Computing**, *Buchquelle*; Third Edition; 2007; Cambridge University Press

[21]: Krieg, Sascha; **Diplomarbeit Simulation und Verkehrslageerfassung:**

Evaluierung von Floating-Car-Daten; *Diplomarbeit;* Deutsches Zentrum für Luft- und Raumfahr e.V. (DLR) in der Helmholtz-Gemeinschaft, 2008

[22]: Dr. Peter Wagner; *persönliche Mitteilung*; Deutsches Zentrum für Luft- und Raumfahr e.V. (DLR) in der Helmholtz-Gemeinschaft; 2008

[23]: Wößner, Michael; **Webseite - Fehlerquellen bei der GPS Positionsbestimmung**; *Internetquelle*; Stand 13.02.2008; http://www.kowoma.de/gps/Fehlerquellen.htm; Zugriffsdatum 28.01.2009

[24]: Stadt Nürnberg; **Webseite des Verkehrsplanungsamtes der Stadt Nürnberg**; *Internetquelle*; http://nuernberg.de/internet/verkehrsplanung/; Zugriffsdatum: 28.01.2009

[25]: Stadt Nürnberg, Baureferat, Verkehrsplanungsamt; **Querschnitszählung 2007**; http://nuernberg.de/imperia/md/content/internet/ref6/vpl/querschnittszaehlung_2007.pdf; *veröffentlichte Statistik*; Zugriffsdatum: 28.01.2009

[26]: Sun Microsystems; **Java API Dokumentation**; *Technische Dokumentation*; Bestandteil des „Java Development Kit“, Onlineversion von J2SE 1.3: http://java.sun.com/j2se/1.3/docs/api/index.html; Zugriffsdatum: 30.01.2009

[27]: Andrey Kusnetsov; **Webseite „Imagero“**; *Technische Dokumentation/Internetquelle*; http://reader.imagero.com/uio/; Zugriffsdatum: 02.02.2009

[28]: PKWARE; **Homepage des Unternehmens PKWARE, Seite .ZIP Application Note;** *Technische Dokumentation/Internetquelle*; http://www.pkware.com/support/zip-application-

note; Zugriffsdatum: 02.02.2009

[29]: Bundesministerium für Verkehr, Bau- und Stadtentwicklung; **Verkehr in Zahlen 2007/2008;** *wissenschaftliche Veröffentlichung/Buchquelle;* DVV Media Group GmbH | Deutscher Verkehrs-Verlag, 2007

[30]: Dr.-Ing. Ralf Kohlen; *persönliche Mitteilung*; VMZ Berlin Betreibergesellschaft mbH; 2008

[31]: Google; **Google Earth;** *Software;* Google, 2008; http://earth.google.de/ Zugriffsdatum: 1.12.2008

[32]: Microtek; **Webseite der Firma Mikrotek s.r.l**; *Internetquelle*; Microtek s.r.l., 2009; http://www.microtek.ud.it/ Zugriffsdatum: 12.02.2009

10 Abbildungsverzeichnis

11 Tabellenverzeichnis

12 Beilagenverzeichnis

- CD-ROM mit folgendem Inhalt:
  - lauffähige Demonstrationsversion des Programms zur FCD-Auswertung
  - lauffähige Demonstrationsversion des Kartendarstellungsmoduls
  - Quellcode des Kartenkompressionsprogramms
  - Quellcode des Kartenlademoduls
  - Quellcode des Kartendarstellungsmoduls
  - Quellcode der Auswertungssoftware
  - Quellcode der Testprogramme „Speichertest“ und „Kompressionstest“
  - Beispiel Kartenmaterial Region Nürnberg als geteiltes GDF
  - Beispiel Kartenmaterial Region Nürnberg als altes Format DLR-TS

Das Kartenmaterial (Demonstrationsversionen, Testprogramme, Beispiele für Kartenmaterial) der beiliegenden CD-ROM ist auf Basis von NAVTEQ-Daten erstellt. Die Kartengrundlage darf nicht für kommerzielle Zwecke eingesetzt werden und verbleibt im Eigentum des DLR (Deutsches Zentrum für Luft- und Raumfahrt, Institut für Verkehrssystemtechnik).