Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 80

Verkehrsabhängiges Routing basierend auf TMC Nachrichten

Tim Sanwald

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Prof. Dr. Stefan Funke
Begonnen am:	18. Juni 2013
Beendet am:	18. Dezember 2013
CR-Nummer:	F.2.2

Kurzfassung

Diese Arbeit beschäftigt sich mit der Akquirierung von TMC-Verkehrsnachrichten und deren Verwendung bei einer verkehrsabhängigen Routenplanung auf Basis von OpenStreetMap Daten. Dafür wurde ein System erstellt, mit dessen Hilfe einkommende TMC-Nachrichten dekodiert und für die Routenplanung aufbereitet werden können.

Zu diesem Zweck wurde in dieser Arbeit ein Programm entwickelt, welches einkommende TMC Datenströme verarbeitet. Das Programm interpretiert die einkommende TMC-Nachricht und identifiziert die entsprechenden Kanten des Straßengraphen. Für die verkehrsabhängige Routenplanung wird entsprechend des Events eine Neugewichtung dieser Kanten durchgeführt. Darauf basierend wird eine graphische Oberfläche angeboten, welche die TMC-Nachrichten darstellt und mit der eine einfache Routenplanung durchgeführt werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

1.1 Überblick über das Dokument

Diese Arbeit gliedert sich in folgende Kapitel:

Kapitel 1 – Einleitung: enthält die Einleitung und den Überblick über die Arbeit. Die Aufgabenstellung wird definiert und die Motivation analysiert. Außerdem werden die verwendeten Bibliotheken vorgestellt.

Kapitel 2 – Grundlagen: definiert grundlegende Begriffe und beschreibt die verwendeten Systeme.

Kapitel 3 – Vorverarbeitung: beschreibt das erste Teilprogramm, welches die OSM-Daten in ein einheitliches Format für einen Graphen konvertiert.

Kapitel 4 – Hauptprogramm: zeigt das entstandene Hauptprogramm und beschreibt dessen Aufbau und Funktionsweise.

Kapitel 5 – Zusammenfassung und Ausblick: fasst die Kapitel 1-4 zusammen und gibt einen Ausblick über mögliche Weiterentwicklungen des Systems.

1.2 Motivation und Ziel

Zu Beginn oder zum Ende der Ferien sind Autobahnen vermehrt überfüllt. Die oftmals kilometerlangen Staus trüben dabei die Vorfreude auf den Urlaub. Doch nicht nur auf der Fahrt in den Urlaub wird der Stau zur nervlichen Belastung, auch der tägliche Weg zur Arbeit wird davon oftmals getrübt. Im Jahr 2012 standen Autofahrer in Deutschland im Schnitt 37 Stunden [Boc], im Großraum Stuttgart sogar bis zu 65 Stunden, pro Jahr im Stau und das, obwohl es meistens zeitsparender wäre den Stau zu umfahren. Für ortsunkundige Personen ist es meist ratsam im Stau zu warten, als sich unnötig zu verfahren. Um dieses Problem zu lösen, soll in dieser Arbeit ein Programm entwickelt werden, welches Realzeit-Verkehrsdaten akquiriert und damit eine verkehrsflussabhängige Planung von Routen ermöglicht. Die Verkehrsdaten sollen dabei mit dem Tool RDSD auf einem Server, mit angeschlossenem Radioempfänger, über rdsquery empfangen und an einen Client übertragen werden. Der Client soll die TMC-Nachrichten anschließend decodieren und die entsprechenden Straßenabschnitte in einem OSM-Graphen identifizieren, beziehungsweise deren Kosten aktualisieren. Des Weiteren soll eine einfache GUI entwickelt werden, mit Hilfe derer die TMC-Nachrichten visualisiert und eine einfache verkehrsabhängige Routenplanung realisiert werden kann.

1.3 Was ist OpenStreetMap?

OpenStreetMap [OSMb] ist ein Open Source Projekt, welches im Juli 2004 in London von Steve Coast gegründet wurde und befasst sich mit der Erstellung einer freien Weltkarte. Dafür werden weltweit Daten von Straßen, Eisenbahnen, Flüssen, Wälder, Häuser und weiteren Daten von einer Gemeinschaft gesammelt. Aktuell sind circa 1,5 Millionen registrierte Nutzer an diesem Projekt beteiligt, dabei steigt die Anzahl von Nutzern rasant an. Durch das Konzept des gemeinschaftsorientierten Sammelns von Geodaten können Veränderungen rasant in die OpenStreetMap Daten eingepflegt werden und damit eine höhere Aktualität, als bei kommerziellen Systemen erreicht werden.

Im Gegensatz zu anderen Kartenanbietern, wie Google, können von OpenStreetMap nicht nur die resultierenden Kartenkacheln frei benutzt werden, sondern auch die zugrunde liegenden Geodaten. Somit ist die Verwendung der Geodaten für einen eigenen Routing-Algorithmus gewährleistet.

1.4 Vorgehensweise

Im Verlauf dieser Bachelorarbeit wurden folgende Arbeitsschritte, in der gegebenen Reihenfolge durchgeführt:

1. Einarbeiten in den Aufbau und die Datenstruktur von TMC und OSM (siehe Kapitel 2).
2. Parsen der Event Code List und der Location Code List (siehe Kapitel 4.3).
3. Einlesen der TMC-Daten (siehe Kapitel 4.5).
4. Erstellen eines grafischen Prototyps und Anzeigen der TMC-Events.
5. Vorverarbeitung der OSM-Daten zu Graphdaten (siehe Kapitel 3.3).
6. Parsen der Graphdaten und berechnen von Routen (siehe Kapitel 4.5.1).
7. Verändern der Graphdaten durch TMC-Events (siehe Kapitel 4.5).
8. Testen von geeigneten Konfigurationen, zur Veränderung des Graphen.

1.5 Verwendete Bibliotheken

Zur Entwicklung des Systems wurden verschiedene Bibliotheken verwendet, im Folgenden werden diese kurz vorgestellt:

OpenCSV
: Das Projekt OpenCSV [ope] bietet Java Objekte an, mit deren Hilfe es möglich ist InputStreams, welche im CSV Format vorliegen, einfach einzulesen. Dabei ist die Bibliothek einfach zu verwenden und auf das Wichtigste reduziert.

Trove

Trove [tro] ist eine Bibliothek für Java, welche effiziente Java Collections für primitive Datentypen anbietet. Dies beinhaltet die Implementierung von Collections, welche primitive Datentypen verwalten können. Dadurch ist es nicht mehr nötig für jeden Integer Wert ein eigenes Objekt zu erstellen, um dieses in einer Collection zu verwalten. Durch diese Fähigkeit wird der RAM Verbrauch drastisch reduziert und selbst größere Datenmengen von über eine Millionen Objekte können in einer Collection verwaltet werden.

JXMapViewer2

JXMapViewer2 ist ein Projekt, welches eine Java Swing Komponente anbietet, mit der die Darstellung einer kachelbasierten Karte erleichtert wird. Es kann dazu benutzt werden, Karten von verschiedenen Kartenservern anzuzeigen.

ParseOSM

ParseOSM ist ein Programm, welches am FMI der Universität Stuttgart entwickelt wurde. Mit Hilfe des Programms können OSM-Dateien eingelesen und in ein CSV ähnliches Format konvertiert werden.

2 Grundlagen

In diesem Kapitel werden die verwendeten Dienste und Techniken beschrieben. Darüber hinaus wird die grundlegende Basis für die weiteren Kapitel erarbeitet. In diesem Sinne werden wichtige Begriffe erklärt und definiert.

2.1 Radio Data System

Das Radio Data System, im Folgenden RDS abgekürzt, beschreibt ein Kommunikationsprotokoll zur Übertragung von Zusatzinformationen über das UKW-Radionetz. Über RDS werden verschiedene Arten von Informationen über das Programm, die Sendeanstalt oder die Zeit übertragen.

2.1.1 Entstehung des RDS

Das RDS wurde von der Europäischen Rundfunkunion zwischen 1983 und 1987 konzipiert und in der DIN EN 62106 [DIN] standardisiert. Die offizielle Einführung war am 1.April 1988. Die aktuell in Deutschland gültige Version ist die DIN EN 62106:2010-07. Anfangs wurde das RDS häufig in Autoradios verwendet. Durch die Sendung von alternativen Frequenzen ist es möglich die Frequenz ohne manuelles Eingreifen zu ändern, um einen eingestellten Sender zu verfolgen. Dies erspart das manuelle Suchen, falls das Fahrzeug den Sendebereich eines Senders verlässt.

2.1.2 Dienste des RDS

Das RDS ist ein allgemeines Konzept zur Übertragung von Daten und ermöglicht vielen Diensten auf diesem System aufzubauen. Im Folgenden werden einige dieser Dienste aufgelistet und kurz beschrieben.

Programme Service Name (PS)
: Der Programme Service Name ist einer der meist genutzten Dienste des RDS. Dieser ermöglicht die Übertragung des Sendernamens in bis zu acht alphanumerischen Zeichen.

Traffic Announcement (TA)
: Dieses Signal bewirkt eine Erhöhung der Lautstärke während der Durchsage von Verkehrsmeldungen und einen Wechsel von CD zum Radio und wieder zurück.

Traffic Programme (TP)
Dieses Signal bedeutet, dass der Sender den so genannten Verkehrsfunk anbietet, welcher Informationen über Staus und Gefahren durch ein TA-Signal ankündigt.

Traffic Message Channel (TMC)
Über den Traffic Message Channel werden Informationen über Verkehrsbeeinträchtigungen in digitaler Form gesendet. Für diese Arbeit ist TMC der wichtigste Dienst und wird aus diesem Grund ausführlich in Kapitel 2.2 beschrieben.

Alternative Frequency (AF)
Bei diesem Dienst wird eine alternative Frequenz für einen Radiosender angegeben, sodass diese bei Verlassen des Sendegebietes automatisch aufgerufen werden kann. Dies führt zu einer Erhöhung der Empfangsqualität des eingestellten Radiosenders.

Radio Text (RT)
Radio Text überträgt zeilenweise Zusatzinformationen mit maximal 64 Zeichen pro Zeile, wobei es hauptsächlich dazu benutzt wird den aktuellen Musiktitel und Interpreten zu übertragen.

Clock Time (CT)
Durch die Ausstrahlung des CT-Signals können Ungenauigkeiten der Uhr im Empfänger korrigiert werden. Durch dieses Signal werden bei vielen Pkws die Uhren automatisch durch den, im Radio verbauten, Empfänger eingestellt. Dadurch entfällt die manuelle Korrektur bei der Umstellung zwischen Sommer- und Winterzeit.

2.2 Traffic Message Channel

Der Traffic Message Channel, im Folgenden TMC, ist ein Dienst basierend auf dem RDS-System und dient der codierten Übertragung von Verkehrsmeldungen.

In der Regel werden TMC-Nachrichten kostenlos sowie unverschlüsselt versendet und sind damit von jedem Radioempfänger frei empfangbar. Es gibt allerdings auch sogenannte payTMC oder TMCpro Dienste, welche eine höhere Qualität der Verkehrsnachrichten versprechen, für welche aber ein entsprechendes Entgelt verlangt wird. PayTMC wird meistens verschlüsselt, kann allerdings auch eine eigene Location Code List („LCL", Kapitel 2.4) verwenden. Das Ziel dieser Arbeit ist die Entwicklung einer Open Source Software und daher wird das kostenlose TMC-Angebot in den folgenden Kapiteln verwendet.

2.2.1 Entstehung des TMC

Der TMC-Standard (ISO 14819 [ISO]) wurde bis zum 11. November 2007 von der nichtkommerziellen Organisation „TMC-Forum“ gepflegt und weiterentwickelt. TMC-Forum verstand sich als Diskussionsplattform für Verkehrsinformationen im Allgemeinen und bestand aus Dienstleistern, Endgeräte-, Automobil- und Kartenherstellern, öffentlichen Institutionen, Rundfunkbetreibern und weiteren Organisationen. Am 11. November 2007 schloss

sich TMC-Forum mit dem TPEG-Forum (Transport Protocol Experts Group) zusammen und bilden seitdem die TISA (Traveller Information Services Association) [tis]. Die TISA prüft und zertifiert Location Code Listen aus der ganzen Welt unter Berücksichtigung von eindeutig definierten Regeln. In Deutschland wird TMC seit 1997 über das FM Band übertragen. TMCpro wurde 2005 von T-Systems Traffic GmbH, einer Subfirma von T-Systems, entwickelt und zur Verfügung gestellt. Seit Januar 2009 wird TMCpro in Deutschland von Navteq angeboten.

2.2.2 Quellen

Abbildung 2.1 zeigt die Entstehung von TMCpro-Nachrichten durch die Nutzung unterschiedlicher Datenquellen. Als mögliche Quellen für TMC-Nachrichten gelten die „Verkehrszentrale der Polizei“, die „Stationären Erfassungs-Systeme“ und das „Floating Car"Programm. Jedoch werden die Daten der „Stationären Erfassungs-Systeme (SES)“ und der „Verkehrszentrale der Polizei“ nicht verschlüsselt und auch über die kostenlose Variante von TMC angeboten.

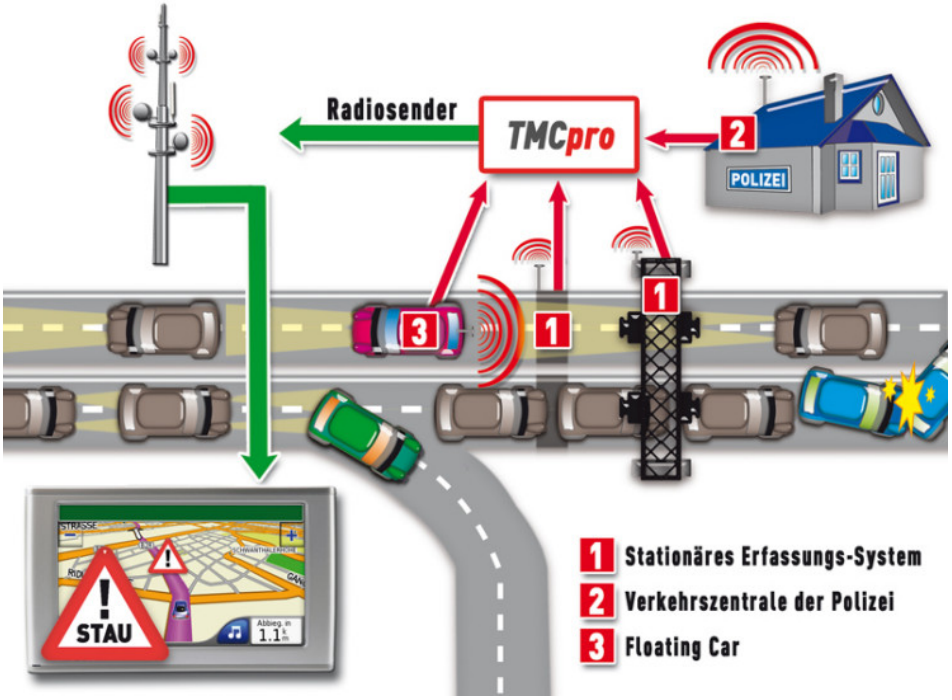

Abbildung 2.1: Übersicht über die Entstehung und Verwendung von TMC-Nachrichten [tmc].

SES ist ein komplexes System aus Sensoren (IR/Radar-Sensoren, Induktivschleifen), welche entlang der Autobahnen und Bundesstraßen installiert sind. Diese messen den Verkehrsfluss, berechnen die Durchschnittsgeschwindigkeit, erfassen die vorbeifahrenden Fahrzeugklassen

und schicken diese Informationen an eine SES-Zentrale. Diese prüft die gesammelten Daten auf Plausibilität und Integrität und verarbeitet die Daten unter anderem zu TMC-Nachrichten. Diese werden an die regionalen Rundfunkanstalten weitergegeben, welche die Daten über den FM-Funk versenden.

Die TMC-Nachrichten können somit von einem RDS-fähigen Radio oder Navigationssystem empfangen und zur Routenplanung oder zum Anzeigen von Gefahren eingesetzt werden. In Deutschland senden hauptsächlich die öffentlichen Rundfunkanstalten TMC-Nachrichten aus, dazu gehören unter anderem der Bayrische Rundfunk, der Südwestrundfunk, der Mitteldeutsche Rundfunk und der Norddeutsche Rundfunk.

2.3 RDS for Linux

„RDS for Linux“ [ber] ist ein Open Source Projekt zum Empfangen, Dekodieren und Anzeigen von RDS Daten der analogen Radiostationen. Dabei werden Sendernamen, Zeitdaten, Radiotext und Verkehrsmeldungen unterstützt. Das Projekt besteht aus den folgenden drei Komponenten:

rdsd
: Der Radio Data System Daemon (RDSD) ist ein Daemon [dae], welcher RDS Daten eines angeschlossenen Empfängers erhält und dekodiert. Außerdem dient RDSD als Server für Clients, welche über eine TCP/IP-Verbindung oder über Unix Domain Socket auf die codierten RDS-Daten zugreifen können.

rdsquery
: Das Programm rdsquery ist ein Kommandozeilenprogramm, welches librds zum Empfangen, Decodieren und Anzeigen von codierten Nachrichten eines rdsd verwendet.

librds
: Librds ist die zugrunde liegende Bibliothek für rdsd oder auch rdsquery. Diese bietet grundlegende Funktionen zum Dekodieren und Auslesen der RDS Daten. Des Weiteren ist librds für die Clientverbindung zwischen dem rdsquery und dem rdsd verantwortlich.

2.3.1 Format

Über einen Unix Domain Socket können die dekodierten Daten von rdsd abgerufen werden. Dabei wird jede neu empfangene TMC-Nachricht als eine Zeile dargestellt. Jede Zeile beginnt mit einem Identifier gefolgt von individuellen Informationen. Im Folgenden werden die unterschiedlichen Events erklärt und das jeweilige Format beschrieben.

Listing 2.1 Beispielhafter Ausschitt aus einem TMC-Stream.

```
1  S evt=1025 loc=30742 ext=1 dur=0 dir=1 div=0
2  GF evt=108 loc=11650 ext=7 CI=2 dir=1
3  GS CI=2 GSI=0 F1=461 F2=22528
4  GF evt=243 loc=12516 ext=3 CI=3 dir=1
5  GS CI=3 GSI=0 F1=1680 F2=53248
6  GF evt=101 loc=11517 ext=1 CI=4 dir=0
7  GS CI=4 GSI=0 F1=552 F2=0
8  S evt=103 loc=11960 ext=3 dur=0 dir=1 div=0
9  Y provider=SWR
10 T CI=1 8559 058f d313
11 T CI=1 8559 058b d363
12 S evt=104 loc=30616 ext=5 dur=0 dir=1 div=0
13 S evt=112 loc=12522 ext=4 dur=0 dir=0 div=0
```

Singleevent (S)

Format: S <Event> <Location> <Extend> <Duration> <Direction> <DiverseAdvice>
Beispiel: S evt=1025 loc=30742 ext=1 dur=0 dir=1 div=0

Der Identifier S definiert ein Singleevent. Es beschreibt meist TMC-Nachrichten, welche an einen bestimmten Ort gebunden sind, darunter fallen auch Unfälle oder Baustellen. „evt“ beschreibt hierbei den entsprechenden Event-Code in der Eventlist und „loc“ den Code aus der LCL. Der Wert von „extend“ definiert die Ausdehnung und damit die Anzahl an betroffenen TMC-Locations. Duration entspricht der Gültigkeitsdauer der aktuellen Meldung und kann als Freitext angegeben sein. Die Duration wurde jedoch bei keinem, der in dieser Arbeit überwachten, Sender verwendet, sodass der Standardwert von 15 Minuten benutzt wurde. Die letzte Angabe des Singleevents „div“ (Diversive Advice) definiert eine mögliche Umfahrungsempfehlung, diese Angabe wurde allerdings bei keinem der Sender ausgesendet.

Groupevent (GF)

Format: GF <Event> <Location> <Extent> <ContinuityIndex> <Direction>
Beispiel: GF evt=701 loc=10884 ext=1 CI=1 dir=0

Ein Groupevent beschreibt TMC-Nachrichten, welche zu viele Informationen für ein Singleevent besitzen. Ein Groupevent besteht aus einem Groupevent und bis zu vier Folgeevents. Die Identifizierung der Folgeevents folgt durch die Definition des „CI“ Wertes, dieser liegt zwischen eins und sechs und wird nach jedem GF-Event verändert. Äquivalent zur Definition des S-Events werden auch bei dem GF-Event der Eventcode, der Locationcode, die Ausdehnung und die Richtung spezifiziert. Der Eventcode sowie der Locationcode, die Ausdehnung und die Richtung werden über die gleichen Parameter wie bei einem Singleevent definiert.

Folgeevent (GS)

Format: GS <ContinuityIndex> <GroupSequenceIdentifier> <F1> <F2>
Beispiel: GS CI=4 GSI=0 F1=552 F2=0

Ein Folgeevent definiert zusätzliche Informationen eines Goupevents. Die Referenzierung zu dem entsprechenden Groupevent folgt durch die Wahl des CI Wertes. Der Group Sequence Identifier (GSI) beschreibt die Anzahl der noch folgenden GS-Events zum entsprechenden GF-Event, ein Wert von 0 besagt damit, dass das GF-Event mit diesem Folgeevent vollständig spezifiziert ist. Die Werte von F1 und F2 geben zusätzliche Informationen an. Ein Beispiel für ein vollständiges GF-Event wird in Listing 2.2 gegeben.

Listing 2.2 Vollständiges GF-Event mit dazugehörigen GS-Events.

```
1 GF evt=701 loc=11635 ext=0 CI=3 dir=1
2 GS CI=3 GSI=2 F1=812 F2=34325
3 GS CI=3 GSI=1 F1=507 F2=54222
4 GS CI=3 GSI=0 F1=1358 F2=16384
5 GS CI=6 GSI=0 F1=514 F2=16640
```

Verlüsseltes Event (T)

Format: T <ContinuityIndex> <verschlüsselt>
Beispiel: T CI=1 8559 058f d313

Ein verschlüsseltes Event beschreibt ein Event, welches nur mittels einer kostenpflichtigen TMCpro Lizenz entschlüsselt werden kann. Da das Ziel dieser Arbeit eine Open Source freie Software darstellt, kann dieses Event nicht weiter behandelt werden.

Provider Event(Y)

Format: Y <Provider>
Beispiel: Y provider=SWR

Ein Y-Event definiert die Sendeanstalt der TMC Daten über den Wert des „Provider“ Feldes. Dieses Event bietet keine Informationen für eine verkehrsabhängige Routenplanung und wird aus diesem Grund im Programm eingelesen aber nicht weiter verwertet.

2.4 Location Table und Eventlist

Die Location Table (auch „Location Code List“ oder LCL genannt) wird in Deutschland von der Bundesanstalt für Straßenwesen (BASt) veröffentlicht und enthält aktuell circa 45 000 Locations. Neben der LCL wird auch die Event Code List (ECL) von der BASt veröffentlicht und ist auf der Webseite der Behörde anforderbar. Die Listen werden jährlich von den einzelnen Länderbehörden und -organisationen zusammengestellt und aktualisiert. Für das Projekt in dieser Arbeit wurde die LCL in der Version 12.0 und die ECL in der Version 4.01 verwendet.

Die LCL enthält drei Arten von Locations: Gebietslocations zum Beispiel Städte, lineare Locations zum Beispiel Straßen und Punktlocations zum Beispiel Anschlussstellen. Für Autobahnen und Bundesstraßen sind alle relevanten Punkte in der LCL abgebildet, andere Kategorien von Straßen werden jedoch nicht abgebildet, da durch die maximale Datenlänge der Locationscodes der größtmögliche Index mit circa 65 000 definiert ist und allein für Bundesstraßen und Autobahnen 45 000 benötigt werden. Die Tabellen werden zur Decodierung von TMC-Nachrichten benötigt.

Die LCL überführt die, in den S- und GF-Events angegebenen Locations, in Koordinaten, legt den Nachfolger/Vorgänger fest und beschreibt diese mit Namen und anderen Daten. Mit Hilfe der ECL kann das Event für die TMC-Nachricht bestimmt werden. Durch die Angabe einer Beschreibung für jedes Event kann die Ursache beziehungsweise die Wirkung der TMC-Nachricht bestimmt werden.

In Tabelle 2.1 werden die, für diese Arbeit, relevanten Spalten aus der LCL aufgelistet. Im Folgenden werden der Aufbau der LCL und die relevanten Spalten näher erläutert.

Spalte (Nr.)	Spalte	Name des Attributes	Beispiel
1	A	LocationCode	11593
2	B	Typ	P1
10	J	Negativ_Offset	11592
11	K	Positiv_Offset	11594
29	AC	X-Koordinate	843700
30	AD	Y-Koordinate	4897460

Tabelle 2.1: Ausschnitt aus der Location Code List 2013.

Die LCL besteht aus 45 Spalten mit bis zu 65 000 Zeilen. In Spalte 1 der LCL steht die eindeutige Nummer der Location, durch diese ID kann eine Location einem S- oder GF-Event zugeordnet werden. Die ID ist immer positiv definiert. Die Spalte „Typ“ beschreibt den Locationstyp, gültige Werte des Typs sind P für eine Punktlocation, L für eine lineare Location oder A für eine Gebietslocation. Des Weiteren folgt dem Wert des Typs eine Zahl, welche die Kategorie angibt, das Beispiel aus Tabelle 2.1 definiert mit dem hier nicht gezeigten Untertyp ein Autobahndreieck. Die Spalten „Negativ_Offset“ und „Positiv_Offset“ definieren die nachfolgende und vorangehende Location über ihren LocationCode, dies wird

zur Definition der betroffenen Locations von TMC-Nachrichten mit einem extend Wert von eins oder größer benötigt. Dabei wird der „Positiv_Offset“ für einen direction Wert von 0 verwendet, im Fall von direction=1 wird der „Negativ_Offset“ verwendet. Die Anzahl der Iterationen wird über den extend Wert angegeben. Die Werte der Spalten „X-Koordinate“ und „Y-Koordinate“ geben die geographische Länge und die geographische Breite der Location an. Diese sind im Format von WGS84 [WGS] ohne vorangestellte Nullen angegeben. Mit diesen Informationen lässt sich die, im Beispiel der Tabelle 2.1, gegebene Location als Autobahndreieck bei Karsruhe, zwischen der A8 und der A5, mit dem Vorgänger „Ettlingen“ und dem Nachfolger „Karlsruhe-Mitte“ beschreiben.

Die ECL besteht aus mehreren Tabellen, die wichtigste ist die Eventlist, ein Ausschnitt ist in Tabelle 2.2 dargestellt.

Spalte (Nr.)	Spalte	Name des Attributes	Beispiel
1	A	Zeile	472
2	B	Text CEN-English	flooding. Slow traffic expected
3	D	Text (German)	Hochwasser, dichter Verkehr zu erwarten
6	G	Code	933

Tabelle 2.2: Ausschnitt aus der Event Code List 2013.

Darüber hinaus enthält die ECL weitere Tabellen für die Optionalen Messages aus den GS-Events und Tabellen, welche zusätzliche Informationen zu den Events, wie zum Beispiel „bitte vorsichtig fahren“ oder „Abstand halten“ bereit stellen. In der Eventlist sind jedoch die, für die verkehrsabhängige Routenplanung relevanten, Daten enthalten. Darin finden sich die Codes aus den TMC-Nachrichten und deren deutsche und englische Beschreibungen. Über die Zeile G „Code“, welcher eindeutig für jedes Event ist, kann ein Event einer TMC-Nachricht zugeordnet werden, falls der Wert dem Wert „evt“ der TMC-Nachricht entspricht. Über die Spalten „B“ und „D“ kann zusätzlich die Beschreibung des Events in englischer oder deutscher Sprache abgerufen werden. Die einzelnen Events sind unterteilt in verschiedene Bereiche, wie zum Beispiel „Verkehrslage“, „Erwartete Verkehrslage“, „Baustellen“ oder „Gefährliche Situationen“. Events der gleichen Kategorie können jedoch durchaus unterschiedliche Auswirkungen auf die Verkehrssituation haben. So kann es Baustellen geben, welche eine Stausituation erzeugen und damit eine geringere Durchschnittsgeschwindigkeit aufweisen, als die für Baustellen geltenden 80 km/h.

2.5 Graphrepräsentation

Da der gesammte Straßengraph Deutschlands über 20 Millionen Knoten und über 40 Millionen Kanten enthält, ist die Wahl der richtigen Graphrepräsentation äußerst bedeutsam. Einfache Graphrepräsentationen, wie eine Adjazenzmatrix, Adjazenzliste oder Inzidenzmatrix, benötigen bei einer so hohen Anzahl von Knoten und Kanten zu viel Platz im Arbeitsspeicher und sind teilweise zu ineffizient um darauf eine Route zu berechnen. Aus

diesen Gründen wurde eine Graphdarstellung mit Offsetarray gewählt. Diese Repräsentation besteht aus zwei Komponenten, dem Offsetarray und einer Tabelle der Kanten, bestehend aus Startknoten, Zielknoten und Kantenkosten. Im Folgenden wird der Aufbau dieser Repräsentation näher beschrieben, außerdem werden häufig verwendete Methoden, wie das Suchen einer Kante zwischen zwei Knoten oder das Zählen ausgehender Kanten eines Knotens, bei dieser Graphrepräsentation erklärt.

In der Kantentabelle (Tabelle 2.3) beschreibt jede Zeile eine Kante, welche einen fortlaufenden Index, den Quellknoten, den Zielknoten und die Kantenkosten beinhaltet. Diese Tabelle wird nach Quellknoten sortiert und von 0 bis m-1 durchnummeriert, wobei m die Anzahl der Kanten beschreibt.

Kantenindex	Quellknoten	Zielknoten	Kantenkosten
0	0	0	100
1	0	4	130
2	0	4	80
3	1	7	30
4	2	7	30
5	2	7	30
6	2	7	30

Tabelle 2.3: Beispiel einer Kantentabelle.

Das Offsetarray wird entsprechend dem Beispiel in Tabelle 2.4 erstellt:

Knotenindex	Offset
0	0
1	3
2	4
3	7

Tabelle 2.4: Beispiel eines Offsetarrays.

Dabei wird der Wert der Spalte „Offset“ entsprechend dem Index der ersten ausgehenden Kante des Knotens in der Kantentabelle gesetzt. Sollte dieser keine Kanten besitzen, so wird der Index der ersten Kante, welche einen Quellknoten mit Index > dem aktuellen besitzt, gespeichert. Somit ergibt sich die Bedeutung des Offset Wertes. Über diesen Wert ist es möglich, alle ausgehenden Kanten des entsprechenden Knotens zu finden. Dies wird realisiert indem der Offset Wert des eigenen Index und der Offset Wert des nachfolgenden Index verwendet wird. Diese geben den Bereich der ausgehenden Kanten in der Kantentabelle an. Damit erhält man alle Kanten zu einem bestimmten Knoten, wenn eine bestimmte Kante gesucht ist, kann über alle Kanten iteriert und der Zielknoten dieser mit dem des gesuchten Knotens abgeglichen werden.

3 Vorverarbeitung

In dieser Arbeit wird für die Routenplanung ein Graph, wie in Kapitel 2.5 beschrieben, benötigt. Dieser Graph soll aus Daten des OpenStreetMap Projektes erstellt werden. Dazu wird die .osm Datei des jeweiligen Bereichs zuerst mit dem Programm „ParseOSM“ (siehe Kapitel 1.5) bearbeitet. Als Resultat entsteht eine Datei im Format, welches in Kapitel 3.1 beschrieben wird. Im Bearbeitungsschritt der Vorverarbeitung soll dieses Format überführt werden, sodass der Graph im Hauptprogramm effizient einlesbar ist. Dieses Zielformat wird in Kapitel 3.2 definiert. In Kapitel 3.3 werden die benötigten Schritte zur Konvertierung des Formats dargestellt.

3.1 Datenformat vor der Vorverarbeitung

Das Datenformat nach der Bearbeitung mit ParseOSM ist eine Art CSV Dokument mit einem Leerzeichen als Trennsymbol für Spalten. Dieses Dokument besteht aus drei Abschnitten, den Nodes, Ways und Relations. Ein beispielhafter Ausschnitt aus einer Datei ist in Listing 3.1 gegeben. Im Folgenden werden die einzelnen Abschnitte beschrieben und das jeweilige Format der Zeile erklärt.

Listing 3.1 Beispielhafter Ausschitt aus einem OSM-File nach der Bearbeitung mit ParseOSM.

```
1  node 163424 48.8139739 9.0636351
2  node 163819 49.3394243 8.4756554
3  node 238187 48.4618626 10.1479113 "pk" "110.0 km" "highway" "milestone"
4  node 238199 48.4650524 10.1066354 "TMC:cid_58:tabcd_1:LocationCode" "12543"
5  node 238208 48.4638928 10.0970747 "TMC:cid_58:tabcd_1:NextLocationCode" "12544"
6  node 255897521 48.9514255 8.2154020 "TMC:cid_58:tabcd_1:Direction" "positive"
7  node 434459926 48.4974160 9.8150332 "TMC:cid_58:tabcd_1:PrevLocationCode"
8  ...
9  way 42994186 538492467 538492468 538492462 "highway" "path"
10 way 42994196 538492497 538492498 "highway" "residential"
11 way 150970740 1638264857 1638264829 "oneway" "yes"
12 way 150970761 1638264812 1638264831 "restricted
13 ...
14 rel 945816 "TMC:cid_58:tabcd_1:LocationCode" "11593" "node" "441043" "node" "440990"
15 rel 877942 "TMC:cid_58:tabcd_1:NextLocationCode" "11594" "node" "451374" "way" "485641"
16 rel 589541 "TMC:cid_58:tabcd_1:Direction" "negative" "node" "597846"
17 rel 784512 "TMC:cid_58:tabcd_1:PrevLocationCode" "11592" "node" "978642"
```

3.1.1 Elemente

Node

Beispiel: node 163379 48.6414783 9.3149001 “highway“ “motorway_junction“
Format: node <id> <latitude> <longitude> (<tagkey> <tagvalue>)*

Ein Node ist eines der Kernelemente der OpenStreetMap Daten. Nodes definieren Punkte über ihren Längen- und Breitengrad. Aktuell setzen sich die OSM Daten aus über zwei Milliarden Nodes zusammen. Ein Node kann ein einzelnes Objekt beschreiben, dies kann eine Telefonzelle, ein Teil eines Ways oder einer Relation sein. Für die Verwendung als Straßengraph werden alle Nodes benötigt, welche Teil eines Ways sind. Dieser Way muss dafür eine Straße repräsentieren.

Bei jedem Node beginnt die Zeile mit „node“, gefolgt von einer ID, welche den Node eindeutig identifiziert. Danach wird dessen geographische Breite und geographische Länge angegeben. Die Funktion des Nodes wird durch Key-Value-Paare definiert, diese werden auch Tags genannt. Für die Routenplanung relevante Tags können aus Tabelle 3.1 entnommen werden.

Typ	Key	Value	Kurzbeschreibung
Way Node Relation	highway	motorway(-link)	Autobahn bzw. Autobahn-Zubringer
		primary(-link)	Bundesstraße
		secondary(-link)	Landstraße
		tertiary(-link)	Kreisstraße
		unclassified	Schmale Gemeindestraße
		residential	Straße in Wohngebieten
		road	unbekannte Klassifikation
		living street	Verkehrsberuhigter Bereich
		path	Allgemeiner (schmaler) Weg
Way	oneway	yes	Einbahnstraße
Way	access	no	Benutzung nicht erlaubt

Tabelle 3.1: Auflistung der, für die Routenplanung relevanten Tags.

Durch die Tags wird außerdem angegeben, ob ein Node eine Location aus der LCL repräsentiert. Dazu enthält der Node einen der in Tabelle 3.2 aufgeführten Tags.

Typ	Key	Value	Kurzbeschreibung
Way Node Relation	TMC:cid_58:tabcd_1:LocationCode	Integer	LocationCode aus LCL
Way Node Relation	TMC:cid_58:tabcd_1:Direction	0	Positive Richtung
Way Node Relation	TMC:cid_58:tabcd_1:Direction	1	Negative Richtung
Way Node Relation	TMC:cid_58:tabcd_1:NextLocationCode	Integer	Nächster LocationCode
Way Node Relation	TMC:cid_58:tabcd_1:PrevLocationCode	Integer	Vorheriger LocationCode

Tabelle 3.2: Auflistung aller TMC Informationen enthaltender Tags.

Dabei wird ein LocationCode allerdings nicht eindeutig einem Node in den OSM Daten zugeordnet. Dies ist der Tatsache geschuldet, dass Autobahnen oder auch Bundesstraßen in OSM in die einzelnen Verkehrsrichtungen aufgeteilt werden. Für einen LocationCode, beispielsweise des Typs „Autobahnkreuz“, können aufgrund dieser Aufteilung mehrere Nodes auf dieselbe TMC-Location referenzieren.

Way

Beispiel: way 164441804 1760796060 1760796065 “highway“ “motorway“
Format: way <id> (<NodeIndex>)* (<tagkey> <tagvalue>)*

Ein Way beschreibt einen Streckenzug bestehend aus bis zu 2000 Nodes. Bei jedem Way beginnt die Zeile mit einem „way“, gefolgt von der ID des Ways. Nachfolgend wird eine Liste der Nodes durch deren IDs definiert, diese Liste definiert den Streckenzug des Ways. Das Beispiel beschreibt demnach einen direkten Weg vom Node mit Index 1760796060 zum Node mit Index 1760796065. Anschließend werden, äquivalent zu den Nodes, Tags für den Way angegeben. Ein Way repräsentiert genau dann einen Weg, wenn dieser einen der Tags aus Tabelle 3.1 enthält.

Relation

Beispiel: rel 945819 “TMC:cid_58:tabcd_1:LocationCode“ “7127“ mem “node“ “440990“ mem “node“ “441043“
Format: rel <id> (<tagkey> <tagvalue>)* (mem <way | node | rel> <id> (forward | backward))*

Eine Relation definiert einen Zusammenhang zwischen mehreren Nodes, Ways und Relations. Relationen werden verwendet um logische oder geografische Beziehungen zwischen Elementen zu modellieren.

Eine Relation wird mit dem Schlüsselwort „rel“ begonnen, gefolgt von einer eindeutigen ID. Anschließend werden die Tags, gefolgt von einer Liste von „Membern“ definiert. Ein Member stellt eine Referenz auf einen Node, einen Way oder eine Relation durch Angabe des Schlüsselworts und dessen ID dar. Falls eine Location einem Node nicht eindeutig zugeordnet werden kann, so wird diese Location über eine Relation beschrieben. Dabei enthalten die Tags den LocationCode und durch die Member werden die entsprechenden Nodes referenziert.

3.2 Datenformat für den Straßengraph

Damit das Einlesen des Graphen in das Routingprogramm möglichst schnell ablaufen kann, wird ein spezielles Format verwendet. Dabei wird die Anzahl der Knoten in der ersten Zeile der Datei angegeben. Diese wird verwendet um das Offsetarray mit fester Größe zu initialisieren. In der zweiten Zeile wird die Anzahl der Kanten angegeben, dadurch kann auch eine Kantentabelle mit fester Größe erstellt werden. Anschließend werden die Knoten mit ihren Tags aufgelistet, danach werden die Kanten ergänzt. In Listing 3.2 wird der Aufbau einer Graphdatei dargestellt. Die Knoten sind von Null ab durchnummeriert und die Kanten nach ihren Quellknoten sortiert. Das Format der Knoten und Kanten wird im Folgenden näher spezifiziert.

Knoten

Beispiel: “39“ “48.5157409“ “9.0868602“ “TMC cid_58:tabcd_1:LocationCode“ “41381“
Format: <index> <latitude> <longitude> (<tagkey> <tagvalue>)*

Knoten werden durch ihren Index spezifiziert. Zu jedem Knoten werden darüber hinaus die dazugehörige geographische Breite und Höhe angegeben, sodass der Abstand zwischen zwei Knoten berechnet werden kann. Weiter werden die benötigten Tags für die Routenplanung angegeben. Eine Liste dieser Tags ist in Tabelle 3.2 gegeben.

Kanten

Beispiel: “2“ “529197“ “130“
Format: <Quellknoten> <Zielknoten> <Geschwindigkeit>

Kanten werden durch ihren Quell- und Zielknoten definiert. Beide Werte geben

Listing 3.2 Ausschnitt aus einer Graphdatei

```
"3362416"
"6850936"
"0" "48.6674214" "9.2445576"
"1" "48.6694744" "9.2432712"
"2" "48.6661923" "9.2515362"
"3" "48.6656273" "9.2552693" "TMC:cid_58:tabcd_1:LocationCode" "15381"
"4" "48.6651480" "9.2586365"
"5" "48.6478807" "9.3334938"
"6" "48.6524704" "9.3366919"
"7" "48.6424647" "9.3313676"
...
"3362412" "49.1343169" "10.0351992"
"3362413" "49.1344326" "10.0350535"
"3362414" "49.1343129" "10.0353149"
"3362415" "48.5894634" "8.1174674"
"0" "2441372" "70"
"0" "2441373" "70"
"1" "375177" "70"
"1" "413252" "70"
"2" "343888" "70"
"2" "529197" "130"
"3" "343890" "70"
"3" "529199" "50"
"4" "343892" "70"
"4" "347623" "70"
"5" "346851" "100"
...
```

die Referenz auf den Index eines Knoten an. Für die Berechnung der schnellsten Route im Hauptprogramm wird neben dem Quell- und dem Zielknoten auch die maximale Geschwindigkeit in km/h benötigt.

3.3 Ablauf

Für die Transformation der OSM-Daten (siehe Kapitel 3.1) zu Graphdaten (siehe Kapitel 3.2) wurde das Java Programm graph-constructing.jar entwickelt, dieses kann folgendermaßen aufgerufen werden:

java -jar graph-constructing.jar <osmDatei> <outputDatei>

Der erste Parameter beschreibt den Pfad der OSM-Datei, welche durch ParseOSM erzeugt wurde. Der zweite Parameter definiert den Pfad, unter welchem der Graph gespeichert werden soll. Das Programm iteriert die OSM-Datei in dessen Ausführung zweimal.

In der ersten Iteration werden die, für den Straßengraphen, benötigten NodeIDs bestimmt. Des Weiteren werden die relevanten Relations erkannt und für die zweite Iteration abgespeichert.

In der zweiten Iteration werden die Nodes zu den gespeicherten IDs gesucht und in das Graphformat übertragen. Anschließend werden die Informationen aus den Relations an die entsprechenden Nodes hinzugefügt. Nach dem Einlesen der Nodes werden die Ways in Kanten aufgeteilt. Durch die Tags der Ways wird zudem die Geschwindigkeit der Kanten festgelegt.

Im Folgenden wird die Durchführung beider Iterationen näher beschrieben.

3.3.1 Ablauf der ersten Iteration

Eine der Aufgaben der ersten Iteration ist die Erkennung der NodeIDs, welche für das Parsen von Ways benötigt werden. Damit diese IDs identifiziert werden können, werden für jeden eingelesenen Way alle dazugehörigen NodeIDs in einem Way-Objekt gespeichert. Wenn dieser Way eine Straße repräsentiert, werden die entsprechenden IDs für die zweite Iteration in der Liste usedNodes gespeichert. Die Liste usedNodes enthält nach der ersten Iteration alle NodeIDs, für die der referenzierte Node in den Straßengraph exportiert werden muss. Für das Parsen der Straßen ist somit gewährleistet, dass alle benötigten Nodes definiert sind.

Eine weitere Aufgabe der ersten Iteration ist das Abspeichern der Relations, welche Informationen zur Identifizierung von TMC-LocationCodes enthalten. Diese Informationen müssen an die entsprechenden Nodes hinzugefügt werden, sodass diese im Programm zur Decodierung der TMC-Nachrichten verwendet werden können. Dabei werden für jeden Node einer Relation mit dem Tag „TMC cid_58:tabcd_1:LocationCode“, die NodeID und die Relation in einer HashMap gespeichert. Über diese HashMap kann bei der zweiten Iteration bestimmt werden, ob für einen Node zusätzliche TMC-Informationen in einer Relation vorliegen. Der Quellcode zu diesem Ablauf ist in Listing 3.3 abgebildet.

3.3.2 Ablauf der zweiten Iteration

Nachdem in der ersten Iteration alle relevanten Informationen gesammelt wurden, wird in der zweiten Iteration das Parsen der Nodes als Knoten und der Ways als Kanten realisiert.

Dabei werden die Nodes eingelesen und überprüft, ob deren ID in der Liste usedNodes enthalten ist. Sollte dies der Fall sein, muss der Node in der resultierenden Graphdatei enthalten sein, da dieser Teil einer Straße ist. Darüber hinaus wird überprüft, ob für den Node zusätzliche Informationen in einer Relation gegeben sind und diese, wenn vorhanden, angehängt. Des Weiteren wird die ID transformiert, um eine laufende Nummerierung der Knoten zu gewährleisten. Diese Transformation wird abgespeichert, sodass beim Parsen der Ways die Referenzen auf die Nodes aktualisiert werden können. Danach besitzt der

Listing 3.3 Quellcode für die erste Iteration.

```
while ((line = readerIt1.readNext()) != null) {
    // Zeile enthält ein Way Element
    if (line[0].equals("way")) {
        Way newWay = new Way(line, true);
        // Way ist ein relevantes Objekt für den Straßengraph
        if (newWay.isWay) {
            usedNodes.addAll(newWay.nodes);
        }
        // Zeile enthält ein Relation Element
    } else if (line[0].equals("rel")) {
        Relation rel = new Relation(line);
        if (!rel.nodes.isEmpty()) {
            for (Long id : rel.nodes) {
                nodeIdToRelation.put(id, rel);
            }
        }
    }
}
```

Node das Format, wie in Kapitel 3.2 beschrieben und wird abschließend in der Graphdatei gespeichert.

Für das Parsen der Ways müssen zuerst die IDs der Knoten transformiert werden, dazu wird die entsprechende Transformation für einen Node gesucht und die ID damit ersetzt. Des Weiteren wird die Relevanz des Ways für den Straßengraph entschieden, diese ist gegeben, falls der Way den Tag „highway“ enthält. Außerdem wird geprüft, ob es sich um eine Einbahnstraße handelt, da in diesem Fall die Kanten bezüglich der Reihenfolge der Nodes erstellt werden. Ist dies nicht der Fall, werden zudem Kanten für den Pfad zurück erstellt. Für den Straßengraph muss zusätzlich die erlaubte Maximalgeschwindigkeit eines Ways definiert werden. Dies geschieht indem der Wert des Tags „highway“ ausgelesen und, wie in Tabelle 3.3 dargestellt, konvertiert wird.

Straßentyp	Geschwindigkeit (km/h))
motorway	130
motorway(-link), primary(-link)	100
secondary(-link), tertiary(-link), trunk	70
unclassified, residential, road	50
living street, service, path	30

Tabelle 3.3: Geschwindigkeit der OSM Straßentypen [FaP].

Sobald dies geschehen ist werden die erstellten Kanten in einer Liste gespeichert. Diese Liste wird in regelmäßigen Abständen, nach Quellknoten sortiert, in Auslagerungsdateien geschrieben, dadurch wird der Speicherverbrauch reduziert.

3.3.3 Abschluss

Nachdem alle Knoten und Kanten erstellt wurden, müssen diese in das endgültige Format übertragen werden, dafür werden zuerst alle Auslagerungsdateien der Kanten mittels Mergesort Algorithmus [mer] sortiert. Dieser Algorithmus wurde so implementiert, dass aus jeweils zwei sortierten Eingangsdateien eine sortierte Ausgabe entsteht. Dies wird baumartig für alle Auslagerungsdateien durchgeführt. Zunächst wird eine Graphdatei im Format aus Kapitel 3.2 erstellt, dazu wird die Knotenanzahl zusammen mit der Kantenanzahl in die Ausgabedatei geschrieben. Anschließend werden die Knoten hinzugefügt. Zuletzt werden die sortierten Kanten aus der Auslagerungsdatei in die Graphdatei übertragen und die Datei geschlossen.

4 Hauptprogramm

Im Folgenden wird das Java Programm „routing.jar“ und dessen Implementierung näher beschrieben. Dazu wird zunächst dargestellt, wie das Programm gestartet werden kann und welche Funktionen damit verknüpft sind. Des Weiteren wird der Aufbau der Oberfläche erklärt. Anschließend werden das Einlesen und Verarbeiten des Graphs, der LCL, der ECL und der TMC-Nachrichten beschrieben.

4.1 Einstellungen

Das Programm bietet die Möglichkeit TMC-Nachrichten gefiltert abzuspeichern und eine einfache Routenplanung anhand dieser Verkehrsmeldungen durchzuführen. Für die Einstellung des Programms können folgende Startparameter verwendet werden:

-t <“path/to/file”>
: „-t“ bietet die Möglichkeit zur Angabe einer TMC-Datei für das Einlesen aus RDSD-Rohdaten.

-g <“path/to/file”>
: Der Parameter „-g“ dient zur Definition des Straßengraphs über den Pfad der Graphdatei.

-s <“path/to/file”>
: „-s“ bestimmt die Eingabe des TMC-Streams als serialisierte Daten, diese werden durch den Parameter „-o“ erstellt. Sollte kein Parameter „-t“ oder „-s“ definiert sein, so erwartet das Programm die TMC-Nachrichten über einen Unix Domain Socket.

-o <“path/to/file”>
: „-o“ beschreibt den Ausgabepfad für gefilterte TMC-Nachrichten als serialisierte Daten.

-ci <“path/to/file”>
: „-ci“ definiert die Eingangsdatei des TMC Caches. Dieser verringert die Dauer des Einlesens von TMC-Nachrichten, indem die veränderten Kanten durch TMC-Nachrichten gecached werden. Damit wird eine erneute Neuberechnung dieser Kanten übersprungen. Dabei ist der Cache jedoch abhängig vom verwendeten Graph, sodass beim Einlesen derselbe Graph verwendet werden muss. Dabei wird nicht überprüft, ob die Datei mit dem angegebenen Graphen erstellt wurde.

-co <“path/to/file”>

„-co“ definiert die Ausgabedatei des TMC Caches, dieser wird im Intervall von 10 Minuten und vor der Beendigung des Programms erstellt.

-i no

Der Parameter „-i no“ startet das Programm als Konsolenanwendung ohne Benutzeroberfläche. Dies dient der Reduzierung des verwendeten Arbeitsspeichers, falls das Programm dazu verwendet wird TMC-Nachrichten aufzuzeichnen und serialisiert abzuspeichern.

Das Programm kann zur Filterung und Darstellung der RDSD-Daten verwendet werden. Die Filterung ist notwendig, da das Programm RDSD Events häufig mehrfach sendet, dies führt zu einem massiven Overhead. Dieses Verhalten kann daran beobachtet werden, dass der RDSD pro Minute über zehntausend Nachrichten versendet, über TMC allerdings maximal eine Nachricht pro Sekunde übertragen werden kann.

Im Folgenden werden zwei verschiedene Aufrufe angegeben, mit welchen das Programm gestartet werden kann.

```
rdsquery -s server.testserver.de -t tmc -c 0 | java -jar routing.jar -g germany.txt -o
                                    serialized.ser -co serializedCache.ser -i no

java -Xmx6G -jar routing.jar -g germany.txt -s serialized.ser -ci serializedCache.ser
```

Der erste Aufruf bewirkt eine Aufnahme eines TMC-Streams aus RDSD in die gefilterte serialisierte Datei. Dazu wird gleichzeitig auch der Cache des Graphs gespeichert. Letzterer definiert den Start des Programms mit der Benutzeroberfläche und den serialisierten Daten. Diese Konfigurationen können nacheinander ausgeführt werden.

4.2 Benutzeroberfläche

Die Benutzeroberfläche (GUI) ist verantwortlich für die Darstellung der eingelesenen TMC-Nachrichten und dient zur Durchführung einer einfachen Routenplanung. Zudem werden Informationen zu Events und zur Route angezeigt. Die Oberfläche ist unterteilt in vier Hauptbestandteile (siehe Abbildung 4.1), auf welche im Folgenden eingegangen wird.

Kartendarstellung (1)

Die Kartendarstellung zeigt die TMC-Nachrichten als Warnsymbole an, färbt die veränderten Straßenabschnitte Rot ein und stellt die berechnete Route als blauen Streckenzug dar. Für die Routenplanung ist die Angabe einer Start- und Zielposition per Mausklick möglich. Die Route wird bei Angabe der Zielposition automatisch berechnet und dargestellt, dabei werden die aktuell angezeigten TMC-Nachrichten mit berücksichtigt.

Abbildung 4.1: Oberfläche des entwickelten Routenplaners, unterteilt in die vier Hauptbestandteile.

Eventbeschreibung (2)

Im Bereich der Eventbeschreibung werden detailliertere Informationen der, dem Mauszeiger am nächst gelegenen, TMC-Nachricht gegeben. Dazu gehört die veränderte Maximalgeschwindigkeit, die Straßenbezeichnung und eine kurze Beschreibung des Events.

Routeninformationen (3)

Im Falle einer geplanten Route wird in diesem Bereich zusätzlich zur Distanz auch eine erwartete Reisedauer angegeben.

Zeitachse (4)

Zur Analyse der TMC-Nachrichten kann die, vom Programm verwendete, Zeit über diese Zeitachse verändert werden. Damit kann ein Verlauf der TMC-Nachrichten über mehrere Tage oder Wochen hinweg dargestellt und untereinander verglichen werden.

4.3 Einlesen von ECL und LCL

Beim Start des Programms werden zunächst die ECL und die LCL von der Klasse „Parser.java“ aus dem Paket „sanwaltm.java.tmcrouting.util“ in Objekte eingelesen. Die ECL und die LCL liegen als CSV-Dateien in der routing.jar vor, sodass diese nicht extern geladen werden müssen. Das Einlesen und Verarbeiten von ECL und LCL sind nahezu identisch, insofern wird im Folgenden beispielhaft das Parsen der ECL beschrieben.

Die ECL wird zeilenweise eingelesen, dazu wird die Bibliothek OpenCSV (siehe Kapitel 1.5) verwendet. Dazu wird für jede Zeile, welche einen Eventcode enthält, ein Objekt des Typs „Event“ erstellt. Für dieses eingelesene Event wird anschließend die maximal zulässige Geschwindigkeit und die dazugehörige Knotenverzögerung bestimmt. Dazu werden die Werte aus der Tabelle 4.1 verwendet. Die Beschreibung des Events wird dabei von oben nach unten mit der Tabelle verglichen, falls die deutsche Beschreibung den String enthält, wird die entsprechende Geschwindigkeit im Event notiert. Diese Reihenfolge ist bei einigen Events notwendig, wie beispielsweise beim Event mit dem Code 722, dieser enthält die Beschreibung „Baustelle, 10 km stockender Verkehr“, dabei ist der limitierende Faktor nicht die Baustelle, sondern der stockende Verkehr. Anschließend wird das Event in einer Liste gespeichert, sodass beim Parsen der TMC-Nachrichten danach gesucht werden kann.

Eventbeschreibung enthält	**Geschwindigkeit in km/h**	**Knotenverzögerung in s**
stau	10	60
stockender verkehr	30	40
dichter verkehr	60	35
reger verkehr	80	30
baustelle bauarbeiten brückenarbeiten fahrbahnerneuerung markierungsarbeiten	60	40
verkehrsstörung	30	10
unfall	30	40

Tabelle 4.1: Tabelle zum Erkennen der Geschwindigkeit für eine Eventbeschreibung.

4.4 Einlesen des Graphen

Für die Routenplanung muss der Graph in das Programm eingelesen werden. Dieser wird im Format nach Kapitel 2.5 gehalten. Zur Realisierung des Formats wird der Graph wie folgt eingelesen und für die Verwendung optimiert:

1. Initialisieren des Offsetarrays und der Kantentabelle durch das Auslesen der Knotenanzahl und Kantenanzahl.
2. Einlesen der Knoten. Dabei werden die longitude und latitude Werte des Knoten gespeichert. Einfügen der Locationcodes, falls der Knoten eine TMC-Location repräsentiert.
3. Verarbeiten der Kanten:
   a) Speichern von Quellknoten, Zielknoten und der Kantengeschwindigkeit.
   b) Berechnen der euklidischen Distanz zwischen Quell- und Zielknoten.
4. Erstellen des Offsetarrays.
5. Berechnung der größten Starke Zusammenhangskomponente (SZK) und Deaktivierung der Knoten, welche nicht in der SZK enthalten sind.
6. Unterteilung der Knoten anhand eines Gitters.
7. Suche des nächsten Nachbarn der Locations, welche noch keinen Referenzknoten besitzen.

Die Schritte 1.-4. dienen dem grundlegenden Einlesen des Graphformats. Die Schritte 1.-4. werden, wie in Kapitel 2.5 beschrieben, bearbeitet. Die Schritte 5.-7. beschreiben die Optimierung des Graphs, diese dient verschiedenen Aspekten.

Zum einen wird der Aufwand durch die Berechnung der größten SZK eines Durchlaufs des Dijkstra-Algorithmus reduziert. Zum anderen wird sichergestellt, dass der Nutzer der Software bei einer Routenplanung nur Knoten als Start- beziehungsweise Zielposition erhält, welche ein Weg verbindet. Die Berechnung der SZKs geschieht mittels einer iterativen Implementierung des „Algorithmus von Tarjan zur Bestimmung starker Zusammenhangskomponenten“ (vgl. [Tar72]). Anschließend wird die größte gefundene SZK bestimmt und jeder Node, der nicht in dieser enthalten ist, für die Routenplanung und nearest neighbor Suche deaktiviert.

Bei der Suche nach den Start- und Zielknoten einer Routenplanung durch den Benutzer, muss für jeden der beiden Punkte der geographisch nächste Knoten im Graph gesucht werden. Mit einer einfachen Iteration durch alle Knoten auf dem Deutschlandgraph dauert diese Suche annähernd eine Minute, dies schadet der User Experience. Um diese Dauer zu reduzieren, wird in Schritt 6 ein Gitter über den Graph gelegt und Knoten in den Zellen des Gitters gesammelt. Falls bei der Routenplanung der nearest neigbor zu einem Knoten gesucht wird, kann die Zelle der dieser zugeordnet wird nach einem nächsten Knoten durchsucht werden. Ist kein Knoten in der Zelle enthalten, wird die Suche auf umliegende Zellen erweitert, bis ein Knoten gefunden wurde. Dies reduziert den Aufwand einer Suche erheblich und verbessert damit die User Experience.

4.5 Verarbeitung von TMC-Nachrichten

Die Verarbeitung der TMC-Nachrichten unterteilt sich in zwei Abschnitte, die Analyse der Nachricht und die Aktualisierung des Straßengraphen.

In der Analyse werden die einkommenden TMC-Nachrichten eingelesen und als Objekt gespeichert. Dazu werden über die Werte von „loc“ und „evt“ die ensprechenden Event- und Locationinformationen aus der ECL bzw. LCL bestimmt. Anschließend wird untersucht, ob das Event eine Kopie eines aktuellen Events ist. Falls diese Kopie eine Überschneidung der Gültigkeitsdauer mit dem bekannten Objekt besitzt, werden die Events und deren Dauer zu einem Event zusammengefasst. Andernfalls wird das TMC-Event zu der Liste der eingelesenen Nachrichten hinzugefügt.

In der zweiten Phase wird der Straßengraph an die neuen TMC-Nachrichten angepasst. Dafür werden zunächst die, von der TMC-Nachricht, beeinflussten Locations über den „ext“ und den „dir“ Wert der Nachricht, mit Hilfe der LCL, ermittelt. Zur Bestimmung der nächsten Location wird für einen „dir“ Wert von Eins der negative Offset Wert aus der LCL verwendet, für einen „dir“ Wert von Null wird der positive Offset verwendet. Der „loc“ Wert gibt die erste Location an und der „ext“ Wert die Anzahl an Wiederholungen des entsprechenden Offsets.

Zur Verbindung der Locations müssen die Kanten identifiziert werden. Dazu werden im Straßengraph die entsprechenden Knoten für die Start- und Zielpositionen gesucht und eine Route mittels Dijkstra-Algorithmus berechnet. Der daraus resultierende Pfad markiert die Kanten, welche durch die TMC-Nachricht verändert werden müssen. Dafür wird die Geschwindigkeit der Kante mit der Geschwindigkeit aus dem Event ersetzt. Falls die TMC-Nachricht nur eine Location betrifft, so wird der entsprechende Knoten mit dem Wert der Verzögerung aus dem Event verlangsamt. Dadurch erhöhen sich die Kosten jeder Kante, welche diesen Knoten als Ziel haben.

Diese Veränderungen werden gecached, sodass die Anpassungen auf den Graphen, bei Verwendung der Zeitleiste, für den Nutzer des Programms verzögerungsfrei dargestellt werden. Abschließend wird die Benutzeroberfläche aktualisiert.

Im Folgenden wird eine kurze zusammenfassende Übersicht des Ablaufs dargestellt:

1. Einlesen der TMC-Nachricht.
2. Suchen und Hinzufügen der Location- und Eventinformationen.
3. Hinzufügen des Events zur Liste aller TMC-Nachrichten. Sollte das Event eine Kopie eines schon bekannten Events sein, so wird nur der Gültigkeitsbereich des bekannten Events erweitert und das neue Event nicht zur Liste hinzugefügt.
4. Berechnen der beteiligten TMC-Locations.
5. Suchen der, für diese Nachricht zu aktualisierenden, Kanten.
6. Anpassen der Kantenkosten.

7. Cachen der gefundenen Kanten.
8. Update der Benutzeroberfläche.

4.5.1 Berechnung von Routen

Für die Berechnung von Routen wurde ein Dijkstra-Algorithmus implementiert. Dieser Algorithmus wurde durch eine Erweiterung der Kostenfunktion zur Berücksichtigung von Verkehrsmeldungen angepasst. Die Kostenfunktion setzt sich aus den Kosten der entsprechenden Kante und der Verzögerung des Knotens durch TMC-Nachrichten zusammen. Die Kosten für eine Kante stellt die erwartete Reisedauer zwischen Quell- und Zielknoten dar. Dazu wird die euklidische Distanz zwischen Quell- und Zielknoten berechnet und mit der Geschwindigkeit der Kante dividiert. Als Resultat liefert der Dijkstra-Algorithmus ein Array von KnotenIDs, welche die schnellste Route darstellen. Für die beschriebene Kostenfunktion werden die aktuell auf der Benutzeroberfläche angezeigten Verkehrsmeldungen berücksichtigt. Anhand von Abbildung 4.2 wird die Auswirkung auf die Route im Vergleich zu Abbildung 4.3 ersichtlich.

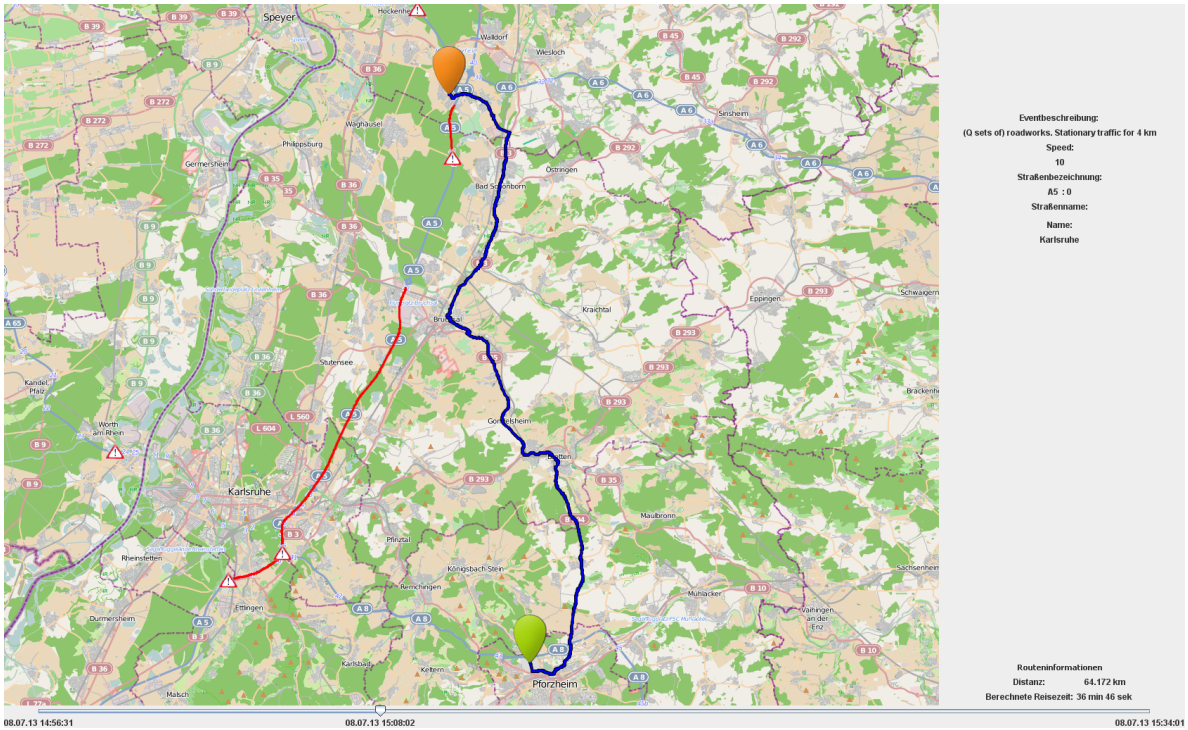

Abbildung 4.2: Stauumfahrung zwischen Pforzheim und Sankt Leon-Rot.

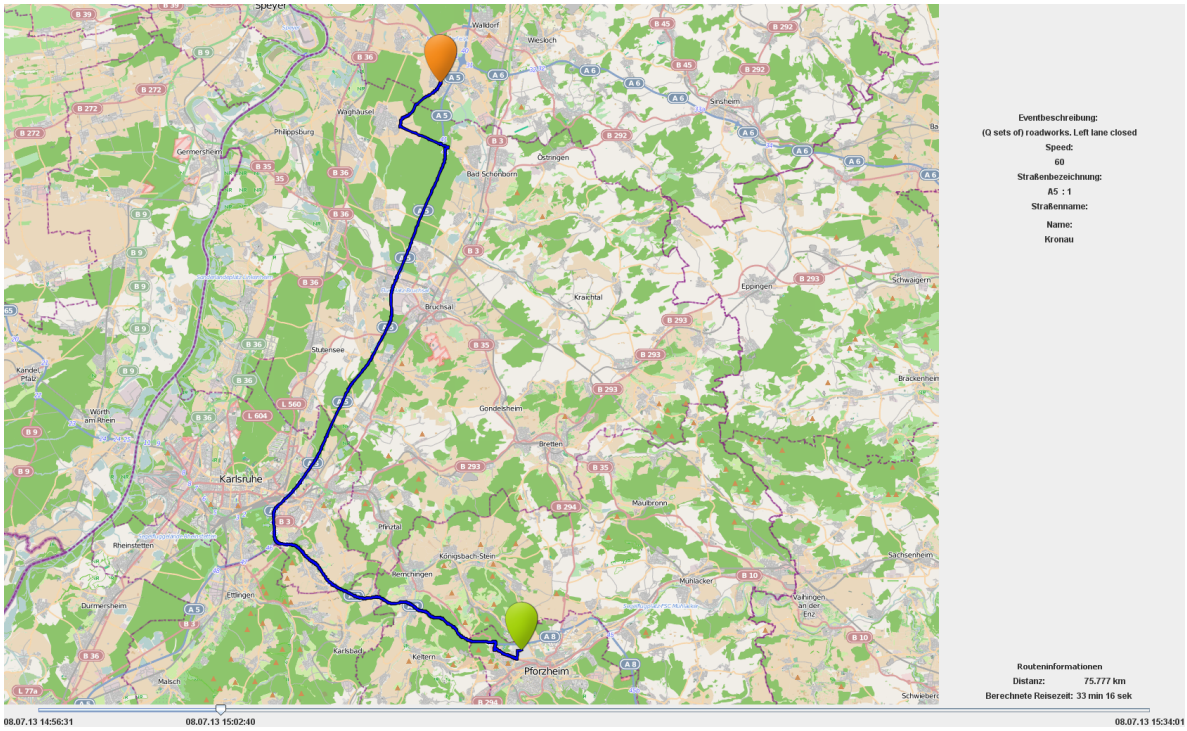

Abbildung 4.3: Route zwischen Pforzheim und Sankt Leon-Rot.

Die TMC-Nachricht hat die Route in Abbildung 4.3 verlangsamt, sodass die alternative Route in Abbildung 4.2 berechnet wurde.

5 Zusammenfassung und Ausblick

Das Ziel der vorliegenden Bachelorarbeit war die Entwicklung eines lauffähigen Systems, welches einkommende TMC Datenströme verarbeitet, den OpenStreetMap Straßengraph neu gewichtet und eine einfache grafische Routenplanung ermöglicht.

Der Source Code des entwickelten Systems wurde auf Bitbucket gehostet und ist über git abrufbar [Rou].

Das entstandene Programm zeigt, dass eine verkehrsabhängige Routenplanung auf Basis von OSM-Daten möglich ist, aber mit einigen Problemen behaftet ist. So ist die Transformation von TMC-Locations zu OSM-Knoten nicht ohne Probleme realisierbar und kann zu leicht verfälschten Ergebnissen führen. Auch die Problematik, dass OSM-Daten eventuell fehlerhaft oder unvollständig sind, lässt sich nur mit Einschränkungen im Routenplaner beheben. Doch durch die kontinuierliche Verbesserung der OSM-Daten könnten diese Probleme in Zukunft behoben werden. Jedoch ist es auch jetzt schon möglich die TMC-Nachrichten für eine Routenplanung auf OSM-Basis in Deutschland einzusetzen. Dieser Sachverhalt wird durch das entstandene System dargelegt.

Ausblick

Erweiterungsmöglichkeiten

Im Folgenden werden verschiedene Möglichkeiten der Erweiterung des Programms und Lösungsansätze dafür gegeben.

Neues Schema für TMC Daten in OSM

Seit einiger Zeit wird im OpenStreetMap Projekt diskutiert, ob und wie das Schema für TMC Informationen in OSM geändert werden soll. Hierzu hat die Firma infoware GmbH ([inf]) mit der Unterstützung der Geofabrik GmbH [geo] ein verbessertes Tagging Schema für TMC Daten entwickelt [OSMa]. Dieses Schema basiert auf den Erfahrungen von infoware im Bereich der Verkehrsabhängigen Routenplanung mit OSM Daten. Mit dem neuen Tagging Schema soll das Hinzufügen von neuen Informationen sowie das Beheben von Fehlern verbessert werden. Im Falle einer Änderung des Schemas müsste das Parsen der OSM Daten in der Vorverarbeitung und auch im Hauptprogramm angepasst werden.

Statistische Auswertung

Um eine Routenplanung auch für zukünftige Routen mit Verkehrsinformationen zu planen, müsste das Programm insofern erweitert werden, sodass die TMC-Nachrichten statistisch ausgewertet werden können. Dazu müssen die TMC-Nachrichten in einer Datenbank gespeichert und analysiert werden. Auf Grundlage dieser Analyse könnten Prognosen für den Verkehrsfluss und passende Events dazu erstellt werden.

Verbesserte Routensuche für TMC-Nachrichten

Da in OSM nicht alle TMC-Locations aufgeführt sind, werden manche Locations aktuell anhand des nächsten Knotens referenziert. Dies kann jedoch auch fehlerbehaftet sein, da so auch Knoten von umliegenden Landstraßen oder ähnlichen als TMC-Location identifiziert werden können. Durch dieses Verhalten ist es möglich, dass durch TMC-Nachrichten die falschen Kanten verändert werden. Um dies zu verhindern, könnten die nächsten 10 Knoten gesucht werden und die Route mit dieser Menge an Start- oder Zielknoten zu berechnen.

Transport Protocol Experts Group

TPEG [TPE] ist ein offener Internationaler Standard zum Aussenden von sprachunabhängigen und multimodalen Verkehrs- und Reiseinformationen. Dieser wurde von der Transport Protocol Experts Group entwickelt. So kann TPEG, anders als TMC, auch im öffentlichen Personenverkehr verwendet werden. TPEG kann über verschiedene Übertragungskanäle gesendet und empfangen werden. TPEG wird als Nachfolger des hier verwendeten TMC gehandelt und bietet im Vergleich zu TMC einige Verbesserungen, unter anderem bietet TPEG eine höhere Datenübertragungsrate als TMC, sodass mehr Events gesendet werden können.

Literaturverzeichnis

[ber] RDS for Linux. URL http://rdsd.berlios.de/. (Zitiert auf Seite 12)

[Boc] J. Bock. Stuttgart liegt bei Staus vorn. URL http://www.stuttgarter-nachrichten.de/inhalt.nirgendwo-dauert-s-laenger-stuttgart-liegt-bei-staus-vorn.41d731e3-ea66-49d7-aeaf-42ead5cecbc8.html. (Zitiert auf Seite 5)

[dae] Daemons. URL https://wiki.archlinux.de/title/Daemons. (Zitiert auf Seite 12)

[DIN] DIN 62106 Spezifikation des Radio-Daten-Systems (RDS). URL http://www.dke.din.de/cmd?artid=127407747&contextid=dke&subcommitteeid=54759035&bcrumblevel=1&level=tpl-art-detailansicht&committeeid=54738887&languageid=de. (Zitiert auf Seite 9)

[FaP] Algorithmen für OpenStreetMap Daten. URL http://www.fmi.uni-stuttgart.de/alg/lehre/ws12/algorithms-for-openstreetmap-data/. (Zitiert auf Seite 25)

[geo] geofabrik. URL http://www.geofabrik.de/. (Zitiert auf Seite 35)

[inf] infoware. URL http://www.infoware.de/. (Zitiert auf Seite 35)

[ISO] ISO 14819 Coding protocol for RDS-TMC. URL hhttp://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=59230. (Zitiert auf Seite 10)

[mer] Mergesort Algorithmus. URL http://de.wikipedia.org/wiki/Mergesort. (Zitiert auf Seite 26)

[ope] opencsv. URL http://opencsv.sourceforge.net/. (Zitiert auf Seite 6)

[OSMa] DE:Proposed features/New TMC scheme. URL http://wiki.openstreetmap.org/wiki/DE:Proposed_features/New_TMC_scheme. (Zitiert auf Seite 35)

[OSMb] OpenStreetMap. URL http://www.openstreetmap.org/. (Zitiert auf Seite 6)

[Rou] Vorverarbeitung und Hauptprogramm TMC-Routing. URL https://bitbucket.org/sanwaltm/tmc-routing/overview. (Zitiert auf Seite 35)

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal*, 1972. (Zitiert auf Seite 31)

[tis] TISA. URL http://www.tisa.org/. (Zitiert auf Seite 11)

[tmc] Stauwarnung: So funktioniert TMC. URL http://www.computerbild.de/artikel/cb-Tests-Navigation-Preiswerte-Navis-schon-ab-70-Euro-2074907.html. (Zitiert auf Seite 11)

[TPE] TPEG Verkehrs- und Reiseinformationen. URL http://www.irt.de/de/themengebiete/digitaler-hoerfunk/tpeg-verkehrs-und-reiseinformationen.html. (Zitiert auf Seite 36)

[tro] Trove High Performance Collections for Java. URL http://trove.starlight-systems.com/. (Zitiert auf Seite 7)

[WGS] World Geodetic System 1984. URL http://de.wikipedia.org/wiki/WGS_84. (Zitiert auf Seite 16)

Alle URLs wurden zuletzt am 16. 12. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift