

RUPRECHT-KARLS-UNIVERSITÄT
HEIDELBERG

Bachelorarbeit

Erstellung eines Routing-Profiles für Feuerwehrfahrzeuge auf Basis von OpenStreetMap-Daten

Amandus Stefan Butzer

Fakultät für Chemie und Geowissenschaften
Geographisches Institut
Abteilung Geoinformatik

Betreut von
Prof. Dr. Alexander Zipf

30. September 2017

Zusammenfassung

Deutsche Kurzfassung

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

Abstract in English Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst, noch nicht anderweitig zu Prüfungszwecken vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt, sowie wörtlich und sinngemäße Zitate als solche gekennzeichnet habe.

Heidelberg den 30. September 2017

.....
(Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis	5
Abkürzungsverzeichnis	5
1 Einleitung	6
1.1 Motivation	6
1.2 Zielsetzung	7
2 Theoretische Grundlagen	8
2.1 Graphentheorie	8
2.1.1 Gerichtete Graphen	9
2.1.2 Gewichtete Graphen	9
2.1.3 Bau eines Graphen aus OpenStreetMap Daten	11
2.2 Routing	12
2.2.1 Shortest Path Problem	13
2.2.2 Dijkstra-Algorithmus	13
2.2.3 Speedup Techniken	15
2.3 Isochronen Berechnung	17
2.3.1 Gitterbasierter Ansatz	17
2.3.2 Dreiecksvermaschung	18
2.3.3 Formenbasierter Ansatz	18
3 Generierung des Routing-Profiles/Methodik?	19
3.1 Informations Erhebung	19
3.2 Aufbau graph backend	19
3.3 Limitierende Faktoren	19
3.4 Erweiternde Faktoren	19
4 Ergebnisse	20
5 Fazit	21
6 Future Work or Ausblick	22

Abbildungsverzeichnis

1	Ein simpler Graph G	8
2	Zwei isomorphe Graphen G und H	10
3	Ein gerichteter Graph G	11
4	Ein gewichteter Graph G	11
5	Simplifizierung eines OSM Datensatzes	12
6	Ein gewichteter und gerichteter Graph G	14
7	Kompletter Durchlauf eines Dijkstra Algorithmus	15
8	Speedup Techniken für den Dijkstra Algorithmus	16
9	Marching Squares Algorithmus	17

Abkürzungsverzeichnis

APSP All Pairs Shortest Path Problem 13

FFL Freiwilligen Feuerwehr Lützelburg 6, 7

GIS Geoinformationssysteme 12

ORS Openrouteservice 6, 7, 18

OSM OpenStreetMap 11

SPP Single Pair Problem 13, 15

SSP Single Source Problem 13–15, 17, 18

TSP Traveling Salesman Problem 13

1 Einleitung

1.1 Motivation

Zum Zeitpunkt der Erstellung dieser Arbeit ist der Verfasser in der Geoinformatik Abteilung des Geographischen Instituts der Ruprecht-Karls-Universität Heidelberg als wissenschaftliche Hilfskraft im Projekt Openrouteservice (ORS) tätig. Dieser bietet neben einer Fülle von Webdiensten, wie Geocoding und Routing, auch einen Isochronen-Service an. Mit Isochronen werden Orte bestimmt, die von einem Standort aus in einer bestimmten Zeit erreicht werden können. Für Unternehmen ist so eine Analyse zum Beispiel bei der Standortwahl zur Berechnung von Umsatzpotentialen interessant. Arbeitnehmer können über Isochronen geeignete Wohnorte für eine zukünftige Arbeitsstelle ermitteln. Bus und Bahn richten an Isochronen ihre Verkehrsnetze aus oder legen Tarifzonen damit fest. Für Polizei, Rettungsdienst sowie Feuerwehr geht es vor allem um das Einhalten amtlich vorgegebener zeitlicher Hilfsfristen. Diese stellen eine bedeutende Eigenschaft für die Planung und Qualität der Einsätze von Feuerwehr und Rettungsdienst dar.

Der Brandschutz ist im Gegensatz zum Rettungsdienst eine kommunale Aufgabe und unterliegt nur in manchen Bundesländern bestimmten Standards (Stein 2015). Daher bedienen sich diese Organisationen unterschiedlicher Hilfsmittel um Bedarfspläne für ihren Standort zu erstellen.

Da mit dem Isochronen-Service jene zeitlichen Erreichbarkeitsanalysen durchgeführt werden können, ist dieser für die Erstellung eines Brandschutzbedarfsplans der Feuerwehr geeignet. Jedoch kann der Dienst in seiner bisherigen Form noch nicht alle erforderlichen Anforderungen für Einsatzfahrzeuge erfüllen. Um diese Nachteile zu überwinden, wurde eine Forschungs Kooperation mit der Freiwilligen Feuerwehr Lützelburg (FFL) eingegangen.

§35 Abs. 1 StVO:

”Von den Vorschriften dieser Verordnung sind die Bundeswehr, die Bundespolizei, die Feuerwehr, der Katastrophenschutz, die Polizei und der Zolldienst befreit, soweit das zur Erfüllung hoheitlicher Aufgaben dringend geboten ist.”

Dieser kurze Absatz der Straßenverkehrs-Ordnung ermöglicht es Einsatzfahrzeugen sich unter Benutzung von Martinshorn und Blaulicht über je-

de Vorschrift im Straßenverkehr hinwegzusetzen. In einem Notfall hat das schnellste Erreichen des Zielorts eine höhere Priorität als Geschwindigkeitsbegrenzungen oder Fahrverbote. Bisher gibt es trotz einer großen Anzahl an Routing-Services keinen, der diese Tatsache berücksichtigt.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist in Kooperation mit der FFL zu Ermitteln, bis zu welchem Grad diese Notstandsvollmachten im Ernstfall in Anspruch genommen werden können. Auf Basis dieser Informationen soll dann ein prototypisches Routing-Profil für Einsatzfahrzeuge entwickelt werden. Die Implementierung wird aufgrund des Umfangs einer Bachelorarbeit auf eine Fahrzeugtyp der Feuerwehr begrenzt. Dabei handelt es sich um Löschfahrzeuge der Klassen LF8, LF8/6 und MLF zwischen 3,5 und 7,5 Tonnen (je nach Beladung). Diese Typen wurden von der FFL als erste Priorität empfohlen. Allerdings wird bei der Erstellung dieses Routing-Profils darauf geachtet, dass Erweiterungen für diverse Einsatzfahrzeuge einfach möglich sind.

Als Basis wird das Profil auf dem Backend¹ des bereits bestehenden Routing Service des ORS aufgebaut. Zusätzlich sollen Java Funktionen implementiert werden, die speziell auf das Emergency Profil zugeschnitten sind. Zur Darstellung wird das ORS Frontend² mit Hilfe der Java-Script Programmiersprache angepasst. Dadurch können die Ergebnisse in verständlicher und anschaulicher Weise präsentiert werden.

¹Programm und Datenbanken die sich zur Berechnung von Routen auf einem über das Internet ansprechbaren Server befinden

²Graphische Benutzeroberfläche der ORS-Website mit der Anfragen an das Backend gestellt und die Antworten dargestellt werden können

2 Theoretische Grundlagen

Als Grundlage für die Berechnung kürzester Wege in einem Straßennetzwerk wird die Graphentheorie als Teilgebiet der Mathematik herangezogen.

2.1 Graphentheorie

Ein Graph ist eine abstrakte Struktur, die Objekte und deren Verbindungen untereinander modellieren kann. Die in der Graphentheorie verwendeten Termini belaufen sich dabei auf Ecken (engl: *nodes* oder *vertices*) für Objekte und Kanten (engl: *edges*) für Verbindungen. (Kurt Mehlhorn 2008, S. 49)

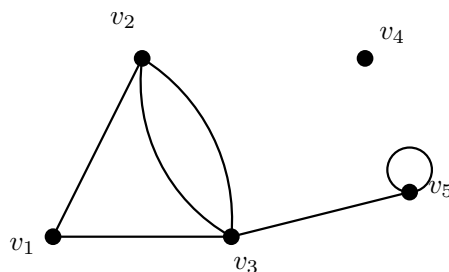
Mathematisch ausgedrückt ist ein Graph G die Funktion aus einer endlichen Eckenmenge V und einer endlichen Kantenmenge E (Aigner 2015, S. 4)

$$G = (V, E)$$

Für den Graphen in Abbildung 1 sehen V und E folgendermaßen aus:

$$V = \{(v_1, v_2, v_3, v_4, v_5)\}$$

$$E = \{(v_1v_2, v_1v_3, v_2v_3, v_2v_3, v_3v_5)\}$$



(a) G

$$\begin{matrix} & v_1 & v_2 & v_3 & v_4 & v_5 \\ \begin{matrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{matrix}$$

(b) Adjazenzmatrix von G

Abbildung 1: Ein simpler Graph G

Ein Vorteil von Graphen ist eine einfache Struktur. Dabei werden die Ecken als Punkte und die Kanten als Linien oder Pfeile dargestellt (Abbildung 1a) (Kurt Mehlhorn 2008, S. 49). Zwischen zwei Ecken können einfache, mehrfache oder keine Kanten bestehen. Darüber hinaus können sie mit sich selbst verbunden sein und eine sogenannte Schlinge bilden (v_5). Sind zwei Ecken durch eine Kante verbunden bezeichnet man sie als *adjazent* (benachbart). Ist eine Ecke der Start- oder Endpunkt einer Kante, werden beide Objekte als *inzident* zueinander bezeichnet. Ist eine Ecke zu keiner

Kante inzident heißt sie *isoliert*. Ein Graph der keine isolierten Ecken besitzt heißt *zusammenhängend*. (Aigner 2015, S. 4 f.)

Computer können Graphen sehr gut verarbeiten, da sich alle Ecken und Kanten in Form von Matrizen speichern lassen. Die Abbildung 1b zeigt die Adjazenzmatrix des Graphen aus 1a. Darin sind die Nachbarschaften für die jeweilige Eckenkombination gespeichert. Die Reihen sind die Start- und die Spalten die Endecken. Für eine existierende Kante wird eine 1 und für keine Verbindung eine 0 eingetragen. Die 2 in Spalte v_2 und Reihe v_3 zeigt die Mehrfachkante zwischen den beiden Ecken an. Für ungerichtete Graphen ist eine Kante von v_1 nach v_2 äquivalent mit einer Kante von v_2 nach v_1 . Daher ist die Adjazenzmatrix für ungerichtete Graphen spiegelsymmetrisch entlang der Hauptdiagonale ($v_1v_1 \rightarrow v_5v_5$). Der Speicherbedarf für eine Matrix folglich halbiert werden, da die eine Hälfte mit der Anderen rekonstruiert werden kann (Abb. 2c). Besitzt der Graph keine Schlingen, besteht die Hauptdiagonale lediglich aus Nullen und kann ebenfalls eingespart werden (Sven Oliver Krumke 2012, S. 19).

Im folgenden werden nur zusammenhängende Graphen ohne Schlingen betrachtet, da diese für viele Problemstellungen irrelevant sind (Aigner 2015, S. 4 f.).

Wenn zwei Graphen bei gleichbleibenden Nachbarschaften der Ecken aufeinander abgebildet werden können, spricht man von isomorphen Graphen Aigner 2015, S. 106. Daraus ergibt sich für isomorphe Graphen auch immer eine gleiche Adjazenzmatrix (Abb. 2).

2.1.1 Gerichtete Graphen

Im Gegensatz zu einem ungerichteten Graphen können bei einem gerichteten Graphen Kanten lediglich in einer Richtung durchlaufen werden. Die Kanten werden daher durch Pfeile anstatt Linien dargestellt.

$$G = (V, R)$$

Demnach ist auch

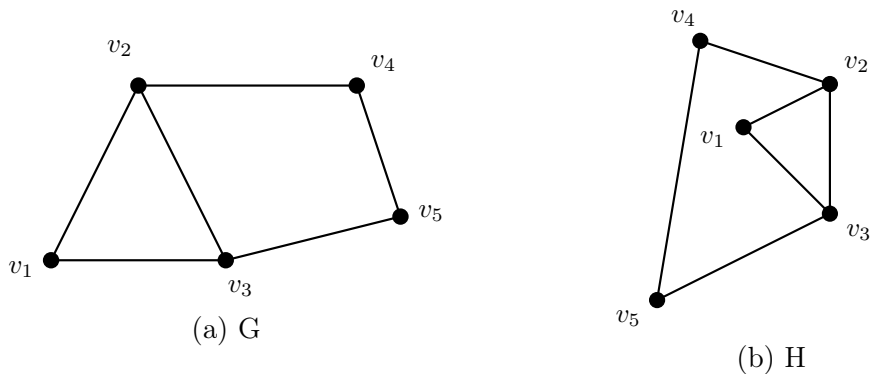
2.1.2 Gewichtete Graphen

In dieser Arbeit bezeichnet der Begriff *gewichteter Graph* einen Kantengewichteten Graphen, bei dem jeder Kante ein Wert c zugewiesen wird.

$$G = (V, E)$$

mit

$$c : R \rightarrow \mathbb{R}$$



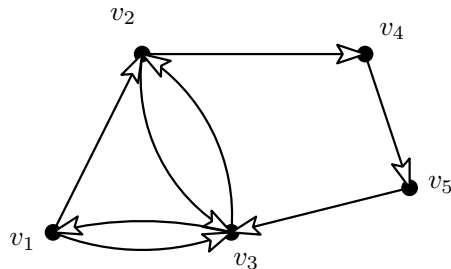
$$\begin{array}{c}
 v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \\
 \begin{pmatrix}
 \ddots & 1 & 1 & 0 & 0 \\
 & \ddots & 1 & 1 & 0 \\
 & & \ddots & 0 & 1 \\
 & & & \ddots & 1 \\
 & & & & \ddots
 \end{pmatrix}
 \end{array}$$

(c) Adjazenzmatrix von G und H

Abbildung 2: Zwei isomorphe Graphen G und H

Neben dem Kanten-gewichteten gibt es auch Knoten-gewichtete Graphen, bei welchen entsprechend die Knoten gewichtet werden. Diese werden aber nur für wenige Problemstellungen benötigt und sind hier nicht von Belang. Gewichtete Graphen können gerichtet und ungerichtet sein. Ein klassisches Beispiel hierfür ist der Linien-Netzplan einer Straßenbahn, bei dem die Knotenpunkte einzelne Haltestellen darstellen und die Kantengewichte die benötigten Minuten beinhalten.

Mit gewichteten Graphen können diverse Problemstellungen gelöst werden, zum Beispiel die Bestimmung maximaler (Durch-)Flüsse in Rohrsystemen oder das Berechnen kürzester Wege. Andererseits kann ein gewichteter Graph für Routing-Zwecke eingesetzt werden, indem räumliche Positionen sowie Eigenschaften von Straßen in der Datenstruktur eines Graphen gespeichert werden.

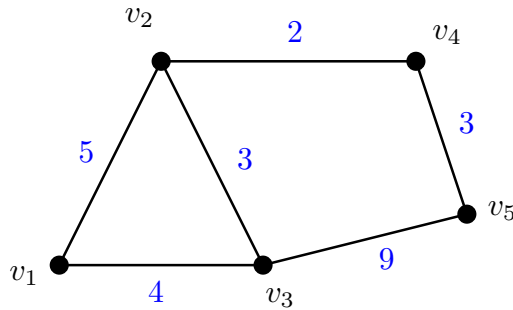


(a) G

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	0	0	1	1	0
v_3	1	1	0	0	0
v_4	0	0	0	0	1
v_5	0	0	1	0	0

(b) Adjazenzmatrix von G

Abbildung 3: Ein gerichteter Graph G



(a) G

	v_1	v_2	v_3	v_4	v_5
v_1	0	5	4	0	0
v_2	5	0	3	2	0
v_3	4	3	0	0	9
v_4	0	2	0	0	3
v_5	0	0	9	3	0

(b) Adjazenzmatrix von G

Abbildung 4: Ein gewichteter Graph G

2.1.3 Bau eines Graphen aus OpenStreetMap Daten

Die OpenStreetMap (OSM) Datenstruktur kann als Graph abgebildet werden. Hier werden Punktobjekte als *Nodes*(Knoten) und Linienobjekte wie Straßen als *Ways*(Wege) bezeichnet. Ein Way ist dabei die Verbindung zwischen zwei Nodes. Zusätzlich gibt es *Relations*(Relationen) die einem Set aus Nodes und Ways einen funktionalen Zusammenhang zuschreiben. Für Straßennetze kann dies durchaus hilfreich sein um beispielsweise unterschiedliche Segmente eines Autobahnsegmentes zusammenzufassen oder um Abbiegebeschränkungen an Kreuzungen zu beschreiben (OpenStreetMap-Wiki 2015).

Um aus den OSM Daten einen routingfähigen Graphen zu erhalten, müssen zuerst alle benutzbaren Nodes und Ways extrahiert werden und werden anhand ihrer OSM-Tags identifiziert. Dazu gehören alle Arten von Straßen und Wegen sowie als befahrbar gekennzeichneten Ways (zum Beispiel asphaltiert, allerdings ohne Straßentyp). Für das Routing sind vor allem Verbindungspunkte wie Kreuzungen, Ab- und Auffahrten etc. sowie Sackgassen interessant. Daher werden diese *Tower Nodes* aus den importierten Daten ermittelt. Anschließend werden die Straßen anhand der Verbindungspunkte

segmentiert. Danach werden die Verbindungen zwischen den Tower Nodes berechnet und anhand der Distanz gewichtet. Das Grundgerüst des eigentlichen Routing-Graphen ist damit erstellt. Einbahnstraßen und Abbiegebeschränkungen werden berücksichtigt und geben die Richtung der Edges an (Rehrl u. a. 2012). Die Punkte zwischen zwei Tower Nodes werden *Pillar Nodes* genannt. Sie werden als *WayGeometry* auf der jeweiligen Edge gespeichert, da sie nicht für den Routing Vorgang benötigt werden (Abbildung 5). Das Routing ist dadurch um ungefähr das 8-fache schneller (Karich 2016). Relevante Attribute wie Geschwindigkeit oder Straßentyp werden vereinheitlicht und als *Flags* auf der Edge gespeichert. Diese sind für die individuelle Gewichtung bei der Routenfindung interessant. (Mehr zu Gewichtung später in Kapitel 3.2(stimmt die reihenfolge max?). Zuletzt wird der Graph abgespeichert und ist für Routing Abfragen bereit (Rehrl u. a. 2012).

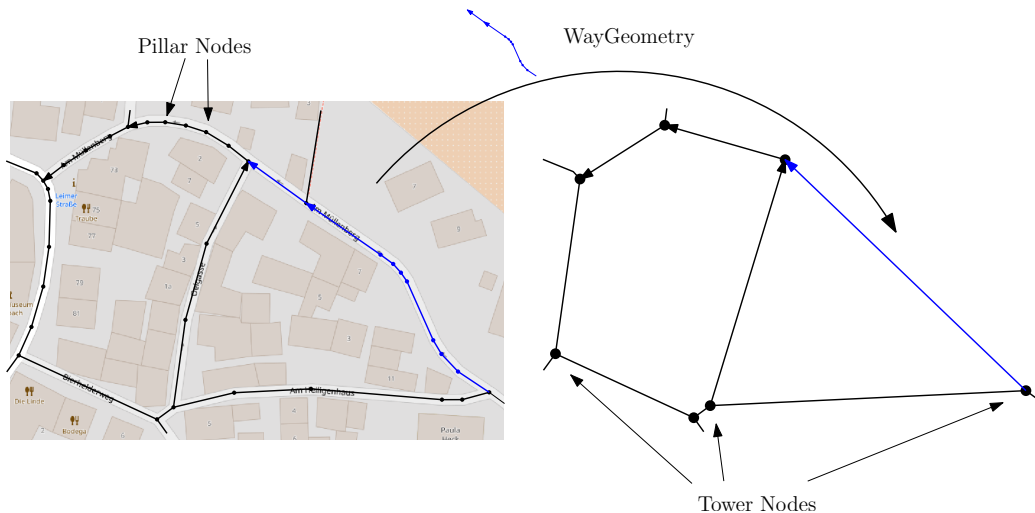


Abbildung 5: Simplifizierung eines OSM Datensatzes

2.2 Routing

Routing bezeichnet den Vorgang in einem Netzwerk Wege zu finden, auf denen Datenpakete entlang gesendet werden können. Diese Definition bezieht sich vor allem auf elektronische Datennetzwerke wie das Telefonnetz oder das Internet. Im Fachbereich der Geoinformationssysteme (GIS) werden hauptsächlich Straßennetze für Routing-Analysen verwendet (Wolfgang Kresse 2012, S. 165). Ein Weg P (*engl.: path*) von einem Startecke s zu einer Zielecke z ist eine Folge von benachbarten Ecken mit s als erster Ecke und z als letzter Ecke der Folge (Abb). Die Weglänge entspricht in einem gewichteten Graphen der Summe aller Kantengewichte.

Eine der wichtigsten Netzwerk-Analyse-Operationen ist die Berechnung des kürzesten Weges zwischen zwei Knoten. Jedes Navigationssystem muss diese Aufgabe erfüllen können. Ein kürzester Weg hat demnach die Eigenschaft, dass die Summe aller Kantengewichte, in anderen Worten die Kosten des Weges, minimal gegenüber allen anderen Wegen im Graphen ist.

2.2.1 Shortest Path Problem

Die nächstliegende zu beleuchtende Problemstellung ist das *Shortest Path Problem* welches aber nicht mit dem *Traveling Salesman Problem (TSP)* verwechselt werden sollte. Beim TSP ist die kürzeste Tour auf einem Graphen $G = (V, E) = K_n^3$ mit der Gewichtsfunktion $c : E \rightarrow \mathbb{R}_+$ gesucht, die jede des Graphen besucht (Sven Oliver Krumke 2012, S. 135). Das Shortest Path Problem lässt sich in drei Typen untergliedern denen jeweils ein gerichteter Graph $G = (V, R)$ mit der Gewichtsfunktion $c : R \rightarrow \mathbb{R}$ zugrunde liegt. Beim *Single Pair Problem (SPP)* wird der kürzeste Weg von einer Ecke a zu einer anderen Ecke b mit $a, b \in V$ gesucht. Das *Single Source Problem (SSP)* möchte den kürzesten Weg einer Ecke a zu jeder anderen Ecke ermitteln (Formel ?). Das *All Pairs Shortest Path Problem (APSP)* sucht den kürzesten Weg von jeder Ecke zu jeder anderen Ecke in V (Sven Oliver Krumke 2012, S. 169 f.).

Für eine Route von Startecke s zur Zielecke z ist das SPP demnach die richtige Wahl. Allerdings wird dazu das SSP herangezogen, welches mit dem Algorithmus von Dijkstra gelöst werden kann.

2.2.2 Dijkstra-Algorithmus

Der in 1959 von Edsger W. Dijkstra entwickelte Algorithmus (Dijkstra 1959) benötigt einen gewichteten Graphen ohne negative Kantengewichte⁴ sowie eine Startecke $s \in V$. Es gibt eine Warteliste W_s mit unmarkierten *gesichteten* Ecken. Dort sind für alle v die Kosten für den bisher kürzesten Weg von s und die jeweilige vorangehende Ecke auf diesem Weg gespeichert. Die Kosten können sich noch ändern wenn ein noch kürzerer Weg gefunden wird. Diese Liste enthält zu Beginn nur den Startpunkt s mit den trivialen Kosten 0. Es

³ K_n bezeichnet einen vollständigen Graphen bei dem jede Ecke aus V mit jeder anderen Ecke verbunden ist

⁴Das Problem bei Graphen mit negativer Gewichtung entsteht, wenn diese auf einer Schlinge oder der Kante eines Rings liegen. Sobald der Algorithmus den Zyklus erreicht, werden die Kosten für die Ecken des Zyklus immer geringer. Die Kosten für den kürzesten Weg nähern sich $-\infty$ während der Algorithmus in einer endlos Schleife läuft. Deswegen ist der Dijkstra nur für positive Kantengewichte anwendbar (Kurt Mehlhorn 2008, S. 194 f.)

gibt eine weitere Liste der endgültig kürzesten Wege K_s in der alle markierten Ecken gespeichert werden.

Dijkstra's Algorithmus markiert die Ecke mit den geringsten Kosten aus W_s und verschiebt diese nach K_s . Nun werden alle benachbarten Ecken gesichtet und die Kosten berechnet. Die Kosten und der Vorgänger werden in W_s gespeichert. Die Ecke mit den geringsten Kosten wird als nächstes markiert, da der Weg dorthin auf jeden Fall ein kürzester ist. Dieser Vorgang wird wiederholt bis alle Ecken aus W_s markiert wurden und W_s somit leer ist.

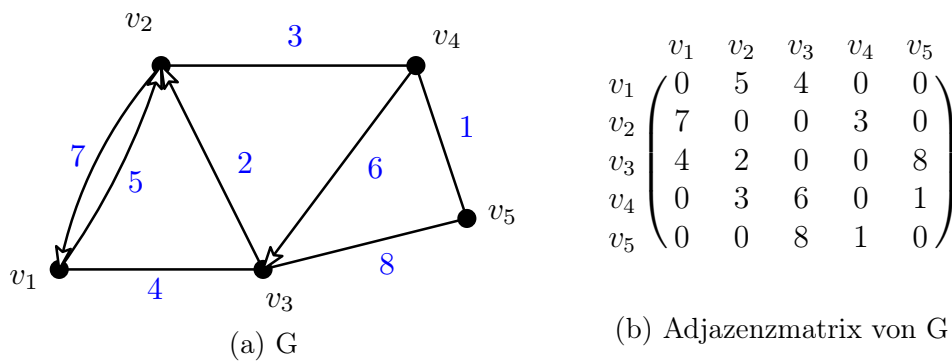
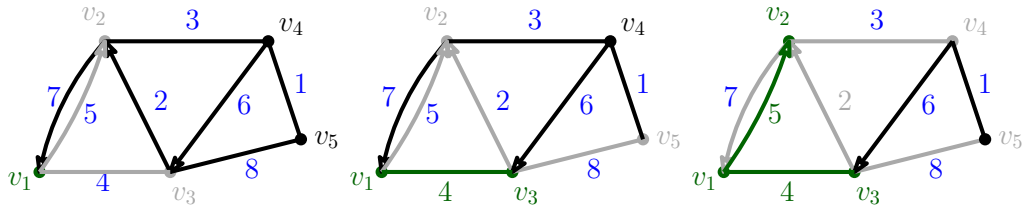


Abbildung 6: Ein gewichteter und gerichteter Graph G

Die Funktionsweise des Dijkstra Algorithmus wird am Graphen aus Abbildung 6 mit der Startecke $s = v_1$ veranschaulicht.

Die erste Ecke der Warteschlange ist $s = v_1$. Da s der einzige Eintrag ist, sind die Kosten automatisch am geringsten. s wird von W_s nach K_s verschoben, markiert und die erreichbaren Ecken gesichtet. Das sind v_2 mit den Kosten 5 und v_3 mit den Kosten 4. Beide werden mit v_1 als Vorgänger in W_s eingetragen. Im zweiten Durchgang wird v_3 (geringste Kosten in W_s) markiert und nach K_s verschoben. Über v_3 erreichbare Ecken sind v_5 mit $4 + 8 = 12$ Kosten und v_2 mit $4 + 2 = 6$ Kosten. Da bereits ein kürzerer Weg nach v_2 besteht wird der aktuelle Pfad über v_3 verworfen. Als nächstes wird v_2 nach K_s verschoben. Die einzige neue von dort gesichtete Ecke ist v_4 mit 8 Kosten. Bei der vierten Wiederholung wird v_4 ($8 < 12$) markiert. Es werden v_3 und v_5 gesichtet. Für v_3 besteht bereits ein kürzerer Weg. Für v_5 ist der neue Weg über v_4 mit den Kosten von 9 allerdings kürzer als der Weg über v_3 . v_5 wird aktualisiert und der längere Pfad verworfen. Im fünften Durchgang wird v_5 als letzte Ecke in W_s nach K_s verschoben. Von dort sind keine neuen Ecken sichtbar. Die Warteschlange ist leer und der Algorithmus damit beendet.

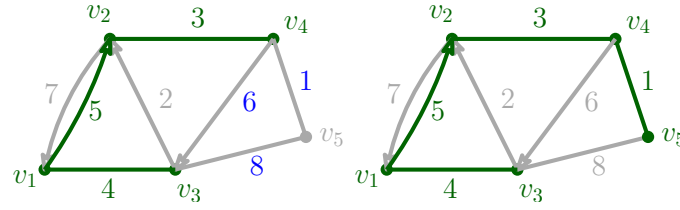
Damit wurde das SSP gelöst und einen kürzeste Wege zu jeder Ecke des



(a) 1. Iteration

(b) 2. Iteration

(c) 3. Iteration



(d) 4. Iteration

(e) 5. Iteration

	W_s	K_s
1.	$v_2 : 5(v_1) \quad v_3 : 4(v_1)$	$v_1 : 0(v_1)$
2.	$v_2 : 5(v_1) \quad v_5 : 12(v_3)$	$v_1 : 0(v_1) \quad v_3 : 4(v_1)$
3.	$v_5 : 12(v_3) \quad v_4 : 8(v_2)$	$v_1 : 0(v_1) \quad v_3 : 4(v_1) \quad v_2 : 5(v_1)$
4.	$v_5 : 9(v_4)$	$v_1 : 0(v_1) \quad v_3 : 4(v_1) \quad v_2 : 5(v_1) \quad v_4 : 8(v_2)$
5.	-	$v_1 : 0(v_1) \quad v_3 : 4(v_1) \quad v_2 : 5(v_1) \quad v_4 : 8(v_2) \quad v_5 : 9(v_4)$

(f) W_s und K_s für jede Iteration

Abbildung 7: Kompletter Durchlauf eines Dijkstra Algorithmus

Graphen vom Startpunkt. Daraus ergibt sich auch die Lösung aller SPP für s zu jeder anderen Ecke aus V . Es ist jedoch nicht sinnvoll für die Lösung eines SPP jedes mal das SSP für den kompletten Graphen zu berechnen. Das liefert nicht nur viele irrelevante Ergebnisse sondern kostet auch mehr Berechnungsressourcen und Zeit. Daher gibt es unterschiedliche Möglichkeiten den Dijkstra Algorithmus zu beschleunigen.

2.2.3 Speedup Techniken

Early Stopping: Der Algorithmus wird bisher jedes mal für den ganzen Graphen ausgeführt, obwohl oft nur die Route für einen kleinen Bruchteil des Graphen benötigt wird. Die einfachste Methode ist den Dijkstra zu stoppen, nachdem z erreicht wurde (8a).

Bidirectional Dijkstra: Beim bidirectional Dijkstra werden zeitgleich zwei Algorithmen nebeneinander ausgeführt. Einer auf s und einer auf z (auf einem umgekehrt gerichteten Graphen). Für beide Instanzen gibt es eine separate Warteschlange W_s und W_z . Zu Beginn wird für jeden Startpunkt die initiale der umliegenden Ecken durchgeführt. Anschließend wird die Ecke mit der geringsten Distanz in beiden Warteschlangen markiert und aus der jeweiligen Warteschlange entfernt. Wird eine Ecke aus beiden Warteschlangen entfernt, werden W_s und W_z auf weitere übereinstimmende Ecken geprüft. Für jede Übereinstimmung wird die Distanz in beiden Instanzen berechnet. Die Ecke ist Teil des kürzesten Weges wenn die Summe beider Distanzen minimal ist.

In der schematischen Abbildung 8b wird die Einsparung gegenüber dem Early Stopping deutlich. Der graue Bereich muss nicht berechnet werden. Die Bearbeitungszeit lässt sich also theoretisch um die Hälfte reduzieren⁵.

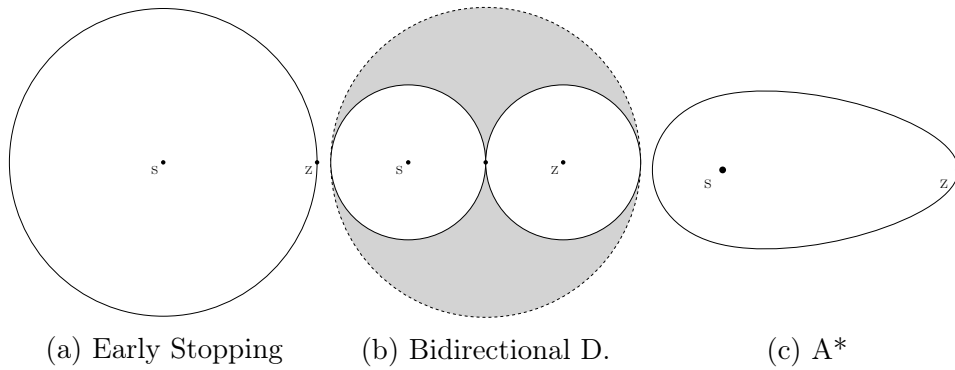


Abbildung 8: Speedup Techniken für den Dijkstra Algorithmus

A*: Der A* Algorithmus ist eine Variante des Dijkstras die den Suchraum in Richtung der Zielecke lenkt. Es wird durch eine Funktion für jede Ecke die Distanz zum Ziel geschätzt. Diese wird mit den Kantengewichten verrechnet damit Ecken in Zielrichtung früher markiert werden. Die standardmäßig kreisförmige Ausbreitung des Dijkstras wird mit dem A* zu einem Oval gestreckt. Da der Zielpunkt so früher erreicht wird, müssen weniger Iterationen durchgeführt werden.

Diese und weitere Möglichkeiten sind in Kurt Mehlhorn 2008, S. 209–213 ausführlich beschrieben.

⁵In der Realität wird dieser Wert selten erreicht. Bei Graphen mit einer höheren Ecken-dichte in Nähe von z könnte die Bearbeitungszeit sogar größer werden. Allgemein wird aber ein signifikanter Vorteil erzielt.

2.3 Isochronen Berechnung

Isochronen sind Linien gleicher Zeit (griech.: *iso* = gleich + *chronos* = Zeit).

Wenn in einem gewichteten Graphen die Kanten die benötigte Zeit enthalten, um von einem Knoten zum nächsten zu gelangen, können damit Analysen zur Erreichbarkeit durchgeführt werden. Dazu wird ein SSP für eine zentrale Ecke z mit einem gegebenen Zeitlimit t gelöst. Isochronen können mit unterschiedlichen Methoden berechnet werden. Das resultierende Objekt ist immer ein Polygon, welches jeden in gegebenem Zeitlimit erreichbaren Punkt beinhaltet.

2.3.1 Gitterbasierter Ansatz

Beim *Marching Squares* Algorithmus wird um das Zentrum ein Gitter über dem Graphen gebildet (9a). Die Eckpunkte des Gitters erhalten dabei die Werte des nächsten Punktes auf dem Graphen (9b). Anschließend werden auf den Kanten des Gitters diejenigen Punkte markiert, bei denen der Wert mit dem gesuchten Zeitlimit übereinstimmt. In Abbildung 9c wurde das für die Werte $t = 5$ und $t = 10$ durchgeführt. Die markierten Punkte werden verbunden und bilden schließlich die Isochrone.

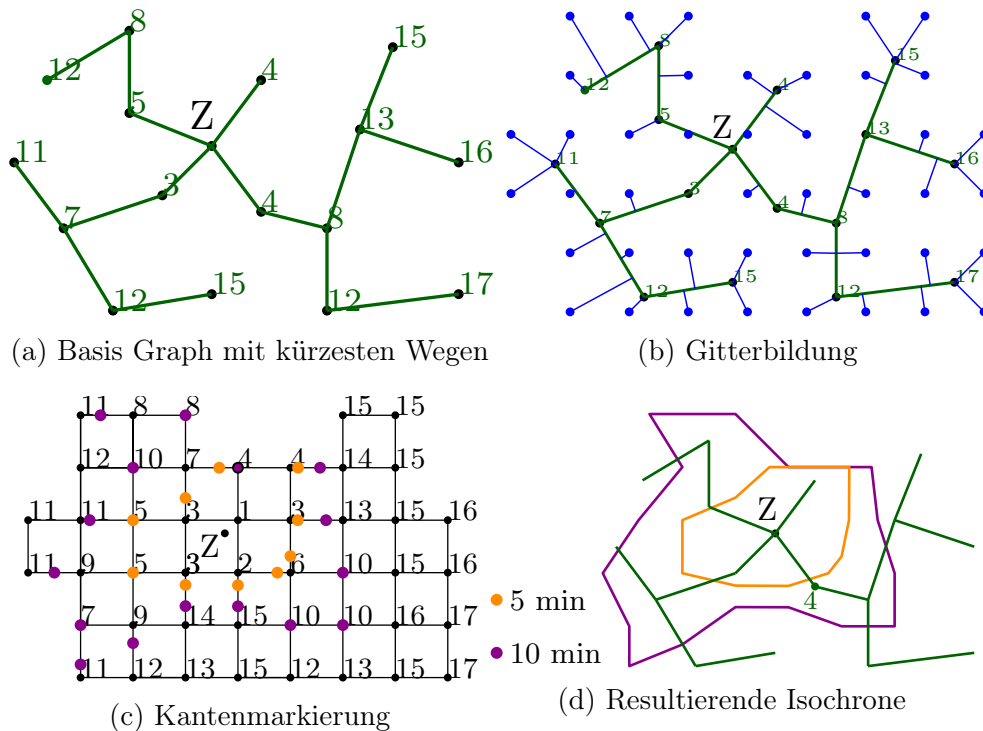


Abbildung 9: Marching Squares Algorithmus

Der Vorteil dieses Algorithmus ist, dass die Maschengröße des Gitters angepasst werden kann. Bei sehr kleinen Maschen liefert der Algorithmus ein sehr genaues Ergebnis. Allerdings werden dabei mehr Ressourcen zur Berechnung gebraucht. Daher können nur kleine Gebiete und geringe Zeitlimits berechnet werden. Bei weiten Maschen ist der Algorithmus dagegen sehr schnell und kann große Distanzen und lange Zeitspannen berechnen. Das Ergebnis ist dementsprechend aber auch ungenauer. Auch in Abbildung 9d wurde die Größe der Maschen unglücklich gewählt. Eine Ecke mit Abstand 4 Minuten vom Zentrum liegt dadurch außerhalb der 5-Minuten-Isochrone.

2.3.2 Dreiecksvermaschung

Neis u. a. 2008 beschreiben eine anschauliche Methode um Isochronen zu berechnen. Nach der Lösung des SSPs werden den Ecken des Graphen die geographische Koordinate der repräsentierten Kreuzung zugeteilt. Die Kanten werden nicht benötigt und daher entfernt. Es liegt also eine 3D-Punktwolke vor. Jeder Ecke wird nun die benötigte Zeit zugewiesen, mit der diese zu erreichen ist. Anschließend werden die Ecken nach dem Delaunay Triangulationsverfahren vermascht. Es entsteht eine Art Trichter mit der Startecke als Tiefpunkt auf der Höhe Null. Wird dieser Trichter auf Höhe des Zeitlimits geschnitten, entsprechen von oben betrachtet die Randkanten des Trichters der Isochrone.

2.3.3 Formenbasierter Ansatz

Die Implementierung des ORS zur Berechnung von Isochronen verwendet ist einen Form-basierten Ansatz. Zuerst werden mit dem in Kapitel 2.2.2 bereits ausführlich erklärten Dijkstra Algorithmus alle in gegebener Zeit erreichbaren Kanten markiert. Anschließend werden die geographischen Punkte (Pillar Nodes) aus der WayGeometry der Kante extrahiert (Siehe Kapitel 2.1.3 auf Seite 11). Um jeden der extrahierten Punkte wird ein kreisförmiger Pufferbereich gelegt. Dadurch können nahe beieinanderliegende Punkte übersprungen werden. Mit den verbleibenden Punkten wird eine Punktwolke generiert. Auf diese Punktwolke wird nun der Alpha-Shape Algorithmus (Zitat) angewandt um die Isochrone als Hülle um die Erreichbaren Wegsegmente zu zeichnen. (Abb. Pictures shape based stuff) Dieser Ansatz liefert präzise Ergebnisse bei schnellen Berechnungszeiten. Die Verwendung der Alpha-Shapes verhindert allerdings die Möglichkeit der Darstellung von nicht erreichbaren 'Löchern' innerhalb der Isochronen.

3 Generierung des Routing-Profiles/Methodik?

Die Grundsätzlichen Überlegungen für

3.1 Informations Erhebung

Fragebogen für Feuerwehr Lützelburg⁶

3.2 Aufbau graph backend

to allow different waytypes we have to accept them while creating the graph (kapitel bla)

weightings different weights for different profiles , apply weights to dijkstra?

for unusable edge -> weighting + unendlich (quelle ???) for emergency speed maps

first tests, feedback from firebrigade way too far.

calc weight for dijkstra calc millis for time add up

3.3 Limitierende Faktoren

geschwindigkeit limits hgv restrictions

3.4 Erweiternde Faktoren

Einbahnstraßen neue Wegtypen

⁶Lützelburg ist eine stadt in Bayern

4 Ergebnisse

Vergleiche zwischen Firetruck - Emergency Vehicle - Car - Heavy Vehicle + exemplarische reale Beispiele!

Hier ein paar räumliche Beispiele aussuchen und exemplarisch zeigen (Routing und Isochronen), welche Änderungen das Profil mit sich bringt, einerseits innerstädtisch, andererseits auch außerhalb der Stadt. Denn Änderungen als solches ist bisschen schwierig zu definieren. Die Jungs aus Lützelburg fragen, welches Gebiet mit den bereits vorhandenen Profilen wirklich schlechte Ergebnisse bringt und jetzt mit Emergency weitaus realistischere!

5 Fazit

Für diesen Zweck geeignet. Much more accurate than previous profile. Für Allgemeingültigkeit weitere Tests nötig Welches für weitere Fahrzeugklassen der Feuerwehr als auch des Rettungsdienstes oder der Polizei erweitert werden kann, da für alle die selbe Grundprämisse gilt (1.1).

6 Future Work or Ausblick

Suche nach Löschwasser quellen um den Zielpunkt (osm tag emergency=fire_hydrant) Beschleunigung, bisher nur Faktor, für genauere Berechnungen exakte Beschleunigungsdaten der Jeweiligen Fahrzeugklasse benötigt. Bremsweg, Kurvengeschwindigkeit, Beschleunigungsweg. rush hour , Tag und Nacht Unterscheidung (nachts weniger los auf Straßen Fußgängerzonen ...)

Literatur

- [Aig15] Martin Aigner. *Graphentheorie*. 2015.
- [Dij59] Edsger W. Dijkstra. „A note on two problems in connexion with graphs“. In: *Numerische Mathematik 1*. 1959.
- [Kar16] Peter Karich. *Low Level API*. 2016. URL: <https://github.com/graphhopper/graphhopper/blob/master/docs/core/low-level-api.md>.
- [Kur08] Peter Sanders Kurt Mehlhorn. *Algorithms and Data Structures*. 2008.
- [Nei+08] Pascal Neis u. a. „Webbasierte Erreichbarkeitsanalyse – Vorschläge zur Definition eines Accessibility Analysis Service (AAS) auf Basis des OpenLS Route Service“. In: *Aktuelle Arbeiten auf dem Gebiet der informations- und Messtechnik*. 2008.
- [Ope15] OpenStreetMap-Wiki. *DE:Relationen*. 2015. URL: <https://wiki.openstreetmap.org/wiki/DE:Relationen>.
- [Reh+12] Karl Rehrl u. a. „Evaluierung von Verkehrsgraphen für die Berechnung von länderübergreifenden Erreichbarkeitspotenzialen am Beispiel von OpenStreetMap“. In: *Angewandte Geoinformatik 2012*. 2012.
- [Ste15] Jochen Stein. *Qualitätskriterien für die Bedarfsplanung von Feuerwehren in Städten*. 2015. URL: <http://www.agbf.de/pdf/Fortschreibung%20der%20Empfehlung%20der%20Qualitaetskriterien%20fuer%20die%20Bedarfsplanung%20in%20Staedten%20Layout%20neu%202016.pdf>.
- [Sve12] Harmut Noltemeier Sven Oliver Krumke. *Graphentheoretische Konzepte und Algorithmen*. 2012.
- [Wol12] David M. Danko Wolfgang Kresse, Hrsg. *Handbook of Geographic Information*. 2012.

Danksagungen

An dieser Stelle möchte ich allen danken..

empty last page