Erstellung eines Routing-Profils auf Basis von OSM / Öffentlichen Daten für Feuerwehrfahrzeuge

Amandus Stefan Butzer

24. September 2017

Kurzfassung

Deutsche Kurzfassung

Abstract

Abstract in Englisch

Erklärung

Hiermit erkläre ich

Danksagungen

An dieser Stelle möchte ich all denen Danken..

Inhaltsverzeichnis

1 Einleitung

Zum Zeitpunkt der Erstellung dieser Arbeit ist der Verfasser in der Geoinformatik Abteilung des Geographischen Instituts der Ruprecht-Karls-Universität Heidelberg als wissenschaftliche Hilfskraft des openrouteservice (ORS) tätig. Der ORS bietet neben Geocoding, Routing und Location Service auch einen Isochrone Service an. Immer wieder wurden Anfragen bezüglich Erreichbarkeitsanalysen aus dem Rettungs- und Brandschutzwesen erhalten. Für Polizei, Rettungsdienst und Feuerwehr geht es vor allem um das Einhalten amtlich vorgegebener Hilfsfristen. Diese stellen die wichtigsten Eigenschaften 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 (vgl.Stein 2015).

Da mit dem Isochrones-Service des ORS Erreichbarkeitsanalysen durchgeführt werden, 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.

In dieser Arbeit wird daher ein Emergency-Routing-Profil in Kooperation mit der Freiwilligen Feuerwehr Lützelburg entwickelt. Die Implementierung ist auf eine Fahrzeugklasse der Feuerwehr begrenzt. Allerdings wird bei der Erstellung des Emergency Profils darauf geachtet, dass Erweiterungen für diverse Einsatzfahrzeuge sehr einfach möglich sind. (Aufgrund des geringen Umfangs einer Bachelor Arbeit kann das in dieser Arbeit jedoch nicht erarbeitet werden.)

Grundsätzlich basiert das Profil auf dem Backend des bereits bestehenden Routing Service des ORS. Zusätzlich werden Java Funktionen implementiert, die speziell auf das Emergency Profil zugeschnitten sind. Zur Darstellung wurde das ORS Frontend mit Java-Script angepasst. Dadurch können die Ergebnisse in verständlicher und anschaulicher Weise dargestellt werden. (weitere Infos über technic details?)

1.1 Motivation

STVO Anwendbar für fast alle Einsatzfahrzeuge

2 Theoretische Grundlagen

Als Grundlage für die Berechnung kürzester Wege bedienen wir uns eines Speicherform der Mathematik, dem Graphen.

2.1 Graphen

Ein Graph ist ein sehr nützliches Konzept um Objekte und deren Verbindungen untereinander zu modellieren. Die in der Graphentheorie verwendeten Termini belaufen sich dabei auf Knoten oder Ecken (engl: *nodes* oder *vertices*) für Objekte und Kanten (engl: *edges*) für Verbindungen. (Kurt Mehlhorn 2008, S. 49)

Ein Graph G ist die Funktion aus einer endlichen Eckenmenge V und einer endlichen Kantenmenge E

G = (V,E) (Graph definition with cite)

(simple graph picture)

Der Vorteil von Graphen ist die einfache Darstellung. Dabei werden die Knoten als Punkte und die Kanten als Linien oder Pfeile dargestellt (vgl Abb.) (Kurt Mehlhorn 2008, S. 49). Zwischen zwei Knoten können einfache, mehrfache oder keine Kanten bestehen. Darüber hinaus können sie mit sich selbst verbunden sein und eine Schlinge bilden. Sind zwei Knoten durch eine Kante verbunden bezeichnet man sie als *benachbart*. Das erfüllt bei einem ungerichteten und ungewichteten Graphen allerdings keinen Zweck, da eine Kante für das Anzeigen der Relation ausreichend ist(richtig?).

Es sollte aber nicht vergessen werden, dass ein Graph nicht die räumliche Position der Objekte sondern nur ihre Relation zueinander ausdrückt! Graphen können also komplett unterschiedlich aussehen und dennoch einander entsprechen. Wenn zwei Graphen bei gleichbleibenden Nachbarschaften der Ecken aufeinander abgebildet werden können spricht man von isomorphen Graphen Aigner 2015, S. 106.

2.1.1 Gerichtete Graphen

Für das Routing ist es wichtig von einem Knoten zu einem anderen zu gelangen. Im Gegensatz zu einem ungerichteten Graphen können bei einem gerichteten Graphen Kanten nur in einer Richtung durchlaufen werden. Die Kanten werden daher durch Pfeile anstatt Linien dargestellt.

G = (V,R)

(Abb gerichtet Graph)

2.1.2 Gewichtete Graphen

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

$G = (V, E) mit c : R- > R$

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

(Abb gewichtet Graph)

Mit gewichteten Graphen können diverse Problemstellungen gelöst werden, zum Beispiel die Bestimmung maximaler (Durch-)Flüsse in Rohrsystemen oder das Berechnen kürzester Wege.

2.1.3 Bau Graph aus OSM Data

Die OpenStreetMap(OSM) Datenstruktur und die Struktur eines Graphen sind im Prinzip gleich. Hier werden Punktobjekte als *nodes*(Knoten) und Linienobjekte wie Straßen als *ways*(Wege) bezeichnet. Ein way ist dabei die Verbindung zwischen zwei nodes. Zusätzich gibt es *relations*(Relationen) die einem Set aus nodes und ways einen funktionalen Zusammenhang zuschreiben. Für Straßennetze ist dies äußerst hilfreich um zum Beispiel unterschiedliche Segmente eines Autobahnstücks zusammenzufassen oder um Abbiegebeschränkungen an Kreuzungen zu beschreiben (Wiki 2015).

Um aus den OSM Daten einen routingfähigen Graphen zu erhalten müssen zuerst alle benutzbaren Nodes und Ways extrahiert werden. Diese werden anhand ihrer OSM Tags identifiziert. Dazu gehören alle Arten von Straßen und Wegen sowie als befahrbar gekennzeichnet Ways (zum Beispiel asphaltiert aber ohne Straßentyp). Für das Routing sind vor allem Verbindungspunkte wie Kreuzungen, Ab- und Auffahrten etc. und 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 nennt man *Pillar Nodes.*(Abb. Tower Nodes / Pillar nodes) Sie werden als *WayGeometry* auf der jeweiligen Edge gespeichert. Da Pillar Nodes somit nicht für den Routing Vorgang benötigt werden, ist dieser 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 ... aufbau graph backend(stimmt die reihenfolge max?. Zuletzt wird der Graph abgespeichert und ist für Routing Abfragen bereit (Rehrl u. a. 2012).

2.2 Routing

Routing ist ein Begriff aus der Nachrichtentechnik und bezeichnet die Bestimmung des günstigsten Weges für die Übertragung von Daten in einem Netzwerk (cite DUDEN). In der Navigation(Geoinformatik?) bedeutet Routing ebenfalls das Ermitteln eines besonders günstigen (häufig des schnellsten) Weges von Startpunkt s zu Zielpunkt z. Das Netzwerk in dem dieser Weg ermittelt werden soll ist in diesem Fall ein gerichteter und gewichteter Graph, der ein reales Straßen- oder Wegenetz repräsentiert.

Wenn die Kante eines solchen Graphen also eine Straße darstellt, beinhaltet das Kantengewicht dabei die Distanz. Die Richtung der Kante ist äquivalent zur erlaubten Fahrtrichtung zum Beispiel im Fall von Einbahnstraßen oder Autobahnen.

Ein kürzester Weg hat die Eigenschaft, dass die Summe aller Kantengewichte des Weges minimal gegenüber allen anderen Wegen ist.

2.2.1 Shortest Path Problem

Die nächstliegendste 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 formel $G = (V, E) = K_n$[1] mit der Gewichtsfunktion $c : E- > \mathbb{R}_+$ gesucht, die jede Ecke V 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 $c : R- > \mathbb{R}$ zugrunde liegt. Beim *Single Pair Problem*(SPP) ist der kürzeste Weg von einer Ecke a zu einer 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, 169f

Für ein Routing von Startecke s zur Zielecke z ist das SPP also die richtige Wahl. Allerdings führt die Lösung über das SSP, welches mit dem Algorithmus von Dijkstra gelöst werden kann.

Beispiel: Ein Anschauliches Beispiel ohne Rechenaufwand für die Lösung des SSPs erhält man, wenn man auf die Karte eines Straßennetzes Fäden auf jede Straße legt. Die Fäden werden an Kreuzungen und am Startpunkt verknotet. Die Fäden stellen den Routing Graph dar. Nun wird der Graph am Startpunkt angehoben. Sofern sich nichts verheddert hat, stellen alle straffen Fäden die kürzeste Route zu den darunter hängenden Knoten dar (Kurt Mehlhorn 2008, S. 191)

2.2.2 Dijkstra Algorithmus

Als Ausgangssituation haben wir einen gerichteten Graphen $G = (V, R)$ mit einer Gewichtsfunktion c: funktion [2] sowie einen Startpunkt $s \in V$. Dijkstra's Algorithmus markiert den Startpunkt und berechnet die Kosten für die Bewegung von s zu jeder benachbarten Ecke. Die berechneten Kosten werden zusammen mit der ID der vorangehenden Ecke für jede Ecke gespeichert und sind die bisherigen kürzesten Wege. Die Ecke mit den geringsten Kosten wird als nächstes markiert. Dieser Vorgang wird wiederholt und die Kosten writewrite

scan node with shortest distance (sum of edge weights) in first start point calculate distance to the endpoints of all outgoing edges write new minimum distance for nodes to queue minimum distance to an unscanned node equals the true shortest distance (proof Kurt Mehlhorn 2008, 197f) repeat until no unscanned nodes in queue

(Dijkstra 1959)

Das 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

[1] K_n bezeichnet einen vollständigen Graphen bei dem jede Ecke V mit jeder anderen Ecke verbunden ist.

[2] ohne negative Kantengewichte

in einer endlos Schleife läuft. Deswegen ist der Algorithmus nur für positive Kantengewichte anwendbar (Kurt Mehlhorn 2008, 194f).

Damit wurde das SSP gelöst und einen kürzeste Wege zu jeder Ecke des Graphen vom Startpunkt. Daraus ergibt sich auch die Lösung des SPP für den Startpunkt zu jeder anderen Ecke. 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 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 er den Zielpunkt erreicht hat (Kurt Mehlhorn 2008, S. 209).

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 Begin wird für jeden Startpunkt die initiale der umliegenden Nodes durchgeführt. Anschließend wird die Ecke mit der geringsten Distanz in beiden Warteschlangen markiert und aus der jeweiligen Warteschlange entfernt. Wird eine Node aus beiden Warteschlangen entfernt, werden W_s und W_z auf weitere übereinstimmende Nodes geprüft. Für jede Übereinstimmung wird die Distanz in beiden Instanzen Berechnet. Die Node ist Teil des kürzesten Weges wenn die Summe beider Distanzen minimal ist(Kurt Mehlhorn 2008, 209f).

Abbildung zwei Kreise, ein grosser kreis, einspaarung,

vergleich realität

A* goal directed Search

lenke suchraum richtung Ziel, ausbreitung nicht kreisförmig. für jede ecke wird die Distanz zum Ziel geschätzt Näher am Ziel liegende Punkte werden bevorzugt (this part...) Kurt Mehlhorn 2008, 210f

+ contraction hierarchies (Kurt Mehlhorn 2008, S. 212)

2.3 Isochronen Berechnung

Isochronen (Herkunft wort iso, chronos)

Wenn in einem gewichteten Graphen die Kanten die benötigte Zeit enthalten um von einem Knoten zum nächsten zu gelangen, können damit Isochronen berechnet werden. Isochronen sind Linien gleicher Zeit. Für die Berechnung wird ein Zentrum und das Zeitlimit benötigt. Das resultierende Objekt ist ein Polygon, welches jeden in gegebenem Zeitlimit erreichbaren Punkt beinhaltet. Isochronen können auf unterschiedliche Arten berechnet werden.

Benutzt routing

+ shaping (darstellung)

marching squares grid based (recursive grid?), delauny erklären shape based(triangles) + unsere implementierung shape based (points)

2.3.1 Gitterbasierter Ansatz

Beim *marching squares* algorithmus wird um das Zentrum ein Gitter über dem Graphen gebildet. Die Eckpunkte des Gitters erhalten dabei die Werte des nächsten Punktes auf dem Graphen. Anschließend werden auf den Kanten des Gitters diejenigen Punkte markiert, bei denen der Wert mit dem gesuchten Zeitlimit übereinstimmt. Die markierten Punkte werden verbunden und bilden schließlich die Isochrone.

(Abb. recursive grid / marching squares)

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.

2.3.2 Delaunay

TIN: Delaunay Triangulation (vgl Neis u. a. 2008) adapt explanation

2.3.3 Formenbasierter Ansatz

Die Implementierung des ORS zur Berechnung von Isochronen verwendet ist ein formen basierter Ansatz (wiederholung :/). Zuerst werden mit dem in Kapitel .. bereits ausführlich erklärten Dijkstra Algorithmus alle in gegebener Zeit erreichbaren Kanten markiert. Anschließend werden die geographischen Punkte der Geometrie der Kante extrahiert. Jede Kante des Graphen stellt ein Wegstück der realen Welt dar. Im Graphen können Kanten allerdings nur durch gerade Linien dargestellt werden. Die wirkliche Geometrie ist deswegen auf der Kante gespeichert. 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) Durch den Dijkstra Algorithmus können mit diesem Ansatz Distanzen bis zu 100km berechnet werden. Je nach Transportmittel entstehen also Unterschiedliche Zeitlimits für die Isochronen Berechnung: Mit dem Auto können bis zu einer Stunde, mit dem Rad bis zu fünf Stunden und zu Fuß sogar bis zu 20 Stunden abgefragt werden. Dieser Ansatz liefert präzise Ergebnisse bei schnellen Berechnungszeiten. Die Verwendung der alpha shape Bibliothek verhindert allerdings die Möglichkeit der Darstellung von nicht erreichbaren 'Löchern' innerhalb der Isochronen.

3 Generierung des Routing-Profils

3.1 Informations Erhebung

Fragebogen für Feuerwehr Lützelburg[3]

[3]Lützelburg ist eine stadt in Bayern

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

3.4 Erweiternde Faktoren

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 realitischere!

5 Fazit

Für diesen Zweck geeignet. Much more accurate than previous profile. Für Allgemeingültigkeit weitere Tests nötig

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.

[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.

[Wik15] OSM Wiki. *DE:Relationen*. 2015. URL: `https://wiki.openstreetmap.org/wiki/DE:Relationen`.