Inhaltsverzeichnis

1 Algorithmen

Das für einen Jouney Planner zu lösende Problem ist das Shortest-Path Problem. Un dieses Problem zu lösen gibt es mehrere Herangehensweisen.

1.1 Graph Basierte Algorithmen

Graph-Basierte Algorithmen bilden einen Graphen aus Punkten. Die Verbindungen zwischen den Punkten werden mit einer Gewichtung versehen. Der Algorithmus iteriert nun über den Graphen und findet die Verbindung zwischen dem Start- und End-Punkt mit dem niedrigsten kombinierten Gewicht.

1.1.1 Dijkstra

Der Dijkstra Algorithmus ist ein SSingle-Source-Shortest-PathÄlgorithmus. Er berechnet den kürzesten Weg von einem Startnode zu jedem anderen Node im Graphen. Er iteriert über alle möglichen Verbindungen und speichert das Gewicht eines Zielnodes. Wenn eine weitere Verbindung zum gleichen Node gefunden wird so wird die Verbindung mit dem geringsten Gewicht behalten. Es wird immer die Node mit dem geringsten Gewichtswert zum Startnode als nächstes berechnet. Wenn auch der Zielnode bekannt ist wird der Algorithmus gleichzeitig vom Start und Zielnode aus gestartet, so dass sie sich in der Mitte treffen.

Der Dijkstra Algorithmus ist der Basisalgorithmus mit dem die anderen Algorithmen verglichen werden. Er ist konzeptionell einfach, jedoch in keiner Weise Optimiert. Die Laufzeit erhöht sich exponentiell mit der Anzahl der Nodes und Verbindungen, weshalb er für grosse Netzwerke ungeeignet ist.

1.1.2 A*

Der A* Algorithmus ist eine Erweiterung des Dijkstar Algorithmus. Er sucht nicht wie der Dijkstra Algorithmus linear in alle Richtungen, sondern gezielt in Richtung des Zielnodes. Dazu wird jedem Punkt im Graph eine Entfernung zum Zielnodes zugewiesen. Nun wird dieser Wert mit dem Gewichtswert des Abstandes zum Startnode kombiniert. Der als nächstes zu berechnende Punkt wird nun aufgrund dieses Wertes entschieden. Dadurch werden Verbindungen welche in Richtung des Zieles führen präferiert un es werden viele überflüssige Rechenschritte eingespart.

Obwohl der A* Algorithmus mehr Operationen pro Node durchführen muss hat er dennoch eine höhere Performance als der Dijkstra Algorithmus da viel weniger Nodes untersucht werden müssen. Der Nachteil vom A* Algorithmus ist, dass er mehr Memory benötigt.

1.1.3 Bellman-Ford

Der Bellman-Ford Algorithmus verwendet das gleiche Grundkonzept wie der Dijkstra Algorithmus. Er achtet jedoch nicht auf den geringsten Gewichtungswert zum Startno-

de, sondern geht der Reihe nach alle Nodes durch. Wenn nun für einen Node nicht alle Zwischennodes zum Starnode schon berechnet wurden, so scheint dieser Node unerreichbar. Um diese Problematik zu lösen wird dieser Prozess x-1 mal wiederholt, wobei x die Anzahl der Nodes ist.

Der Bellman-Ford algorithmus ist langsamer als der Dijkstra Algorithmus, da er mehrere Durchläufe über alle Nodes benötigt. Dies bietet ihm jedoch den Vorteil, dass auch eine negative Gewichtung einer Verbindung möglich ist.

1.2 Non Graph Basierte Algorithmen

In diesem Abschnitt werden Algorithmen erläutert, welche nicht auf das Grundkonzept des Dijkstra Algorithmus aufbauen.

1.2.1 RAPTOR

Der Round-Based Public Transit Routing Algorithmus, auch RAPTOR Algorithmus genannt, ist eine Rundenbasierter, auf öffentliche Verkehrsnetzwerke zugeschnittener Algorithmus welcher auf die vorgegebenen Zuglinien achtet.

Der RAPTOR Algorithmus durchläuft mehrere Runden um von der Startstation zur Zielstation zu finden. in der ersten Runden werden alle Zuglinien gescannt, welche durch die Startstation verlaufen. Wenn eine andere Zuglinie die gescannte Zuglinie Kreuzt so wird die andere Zuglinie sowei die Kreuzungsstation markiert. In der nächsten Runde werden nun alle markierten Zuglinien gescannt. Dieser Prozess wird so lange Weitergeführt, bis sich die Zielstation in einer gescannten Zuglinie befindet. Nun kann anhand der Kreuzungsstationen die Route erstellt werden.

Der RAPTOR Algorithmus hat besitzt eine höhere Performanz als alle Graph Basierten Algorithmen.

1.2.2 Connection Scan Algorithm

Der Connection Scan Algorithm, kurz CSA, ist schon vom Grundkonzept auf Zeitplanbasierte Netzwerke zugeschnitten. Er arbeitet mit Stations, Connections, Trips und Footpaths. Eine Connection ist die Verbindung zwischen zwei Stationen während ein Trip den gesamten Weg eines Zuges darstellt.

In einem Ersten Schritt werden die Daten in die benötigte Timetable-Form gebracht und alle Connections nach der Abfahrtszeit sortiert. Diese Schritte werden preprocessed. Danach wird über alle Connections iteriert. Eine Connection wird als erreichbar markiert, wenn sie an bereits als erreichbar Markierte Connections anschliesst oder mit der Startstation verbunden ist. Dies wird dann solange durchgeführt bis die Zielstation erreicht ist. Anschliessend wird von der Zielstation aus der verfolgte Weg zusammengesetzt, so dass ein Journey von der Startstation zur Zielstation entsteht.

Für grosse Netzwerke kann der CSA mit eine Quadtree-Preprocessing schritt erweitert werden. Dabei wird das Netzwerk in immer kleiner werdende Quadrate unterteilt. Für Quadrate welche nicht die Start- oder Zielstation enthalten müssen nun nur noch die

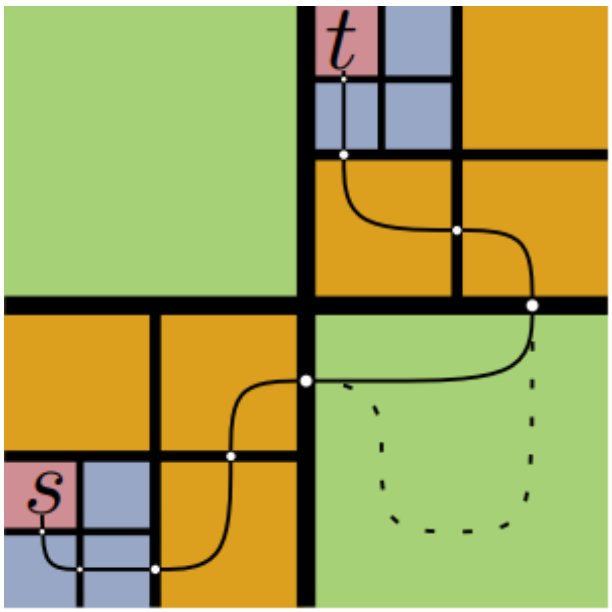

Abbildung 1: Veranschaulichung der Funktion des QuadTree-Preprocessings [1]

Connections beachtet werden, welche über die Ränder des Quadrates hinaus gehen. Somit können für Fernverbindungen die Lokalen Connections ignoriert werden.

Im gegensatz zum Dijkstra müssen beim CSA die Connections nicht nach dem nächsten Schritt durchsucht werden, da die Abfolge durch die Preprocessing-Schritte schon definiert wurde. Dies führt zu einer Performanceverbesserung im mehrstelligen Bereich. Ein Nachteil ist jedoch, dass Preprocessing-Schritte von nöten sind und so nicht in echtzeit auf Verspätungen reagiert werden kann. Da diese Schritte jedoch nur wenige Sekunden benötigen ist dies kein schwerwiegender Nachteil. Mit der Erweiterung des Quadtree-Preprocessings können auch Anfragen in landesweite Netzwerke in wenigen Millisekunden berechnet werden.

Im Vergleich zum RAPTOR Algorithmus bietet der CSA eine höhere Performance.

1.3 Transfer Pattern

Transfer Pattern ist eine Algorithmusstrategie welche auf Preprocessing setzt. Alle möglichen Verbindungen werden mithilfe eines Algorithmuses berechnet und in einem Datensatz gespeichert. Die eigentliche Anfrage beschränkt sich dann auf eine Query-Abfrage auf diesen Datensatz.

Für das Preprocessing können alle zuvor genannten Algorithmen verwendet werden. Die Dijksta basierten Algorithen haben jedoch den Nachteil, dass Query anfragen für landesweite Netzwerke immer noch mehrere Sekunden benötigen, wesshalb der RAPTOR Algorithmus oder der CSA zu bevorzugen sind.

Um das Preprocessing zu beschleunigen können mehrere Erweiterungen hinzugefügt werden. Eine davon ist die Verwendung von Hubs. Dabei werden die grössten Stationen als Hubs markiert. Zwischen diesen Hubs werden alle Transfer Pattern berechnet. Bei

Stationen welche keine Hubs sind werden nur die Verbindungen bis zum nächsten Hub berechnet. Sollte nach drei Zugwechseln kein Hub erreicht sein so wird die Verbindung verworfen. Dies kann zwar zu Fehlern führen, ist aber vernachlässigbar, da die Fehlerrate laut Experimenten bei drei Promill liegt.

Der Preprocessing kann je nach Grösse und Struktur des Netzwerks eine lange Zeit in Anspruch nehmen. Selbst mit allen Erweiterungen braucht das Preprocessing für das CH-Netzwerk vier Stunden. Dafür benötigen die Query-Anfragen nur wenige Millisekunden. Der Nachteil ist jedoch, dass mit Transfer Pattern nicht auf Zugverspätungen und Fahrplanänderungen reagiert werden kann.

2 Programme

Es gibt viele Journey-Planning-Softwares, doch nur wenige davon besitzen eine OpenSource-Lizensierung. Wir werden uns im Rahmen dieses Projektes auf die OpenSource Anwendungen beschränke.

2.1 Traintickets.to

Traintickets.to ist ein von Linus Norton [2] entwickelter Journey Planner für das englische Zugnetzwerk. Linus Norton ist CTO der Firma Assertis [3], welche das Projekt betreibt. [4]

Traintickets.to basiert auf einem modifizierten CSA kombiniert mit TransferPattern. In einem preprocessing Schritt werden aus einem modifizierten GTFS-datasheet die TransferPattern mit dem CSA generiert. Diese werden dann von Hauptprogramm gefiltert und über eine API zur Verfügung gestellt. Assertis ist von der ATOC (Association of Train Operation Companies) lizensiert, so dass sie Ticketpreise anzeigen können sowie Tickets direkt verkaufen können.

Traintickets.to besitzt zwei verschiedene Lizensierungen. Die TransferPattern generierung sowie das Filterprogramm stehen unter einer GNU GPLv3 Lizenz. Die WebAPI ist technisch gesehen ein eigenes Projekt und steht unter exclusive copyright.

Vorteil der Traintickets.to Anwendung ist, dass dieser schon Landesweit implementiert und getestet ist und deshalb auf der Schweiz ohne Skalierung anwendbar ist. [5] [6]

Nachteile der Traintickets.to Anwendung sind, dass modifizierte GTFS-Daten verwendet werden und diese so nicht direkt von der OpenData platform bezogen werden können, dass der Code schlecht Strukturiert ist und dass die Anwendung aufgrund des preprocessing-Aufbaus nicht auf Verspätungen und Fahrplanänderungen reagieren kann.

2.2 Open Trip Planner

Der OpenTripPlanner kurz OTP ist eine auf der Maven-Repository aufbauende Multimodale trip planning Software welche anfangs für Städte ausgelegt war, nun aber auch in ersten landesweiten Netzwerken Anwendung findet. Er wurde von einem OpenSource-Kollektiv aus mehr als 100 Personen in acht Jahren entwickelt.

Der OTP basiert auf dem A*-Algorithmus und verwendet GTFS-Daten und OpenStreetMap Daten in Form einer pbf-Datei. In einem preprocessing Schritt wird der Graph für den Algorithmus erstellt. Dieser kann in einer Datei gespeichert werden oder direkt im RAM des Servers gelagert werden. Selbiges wird für die OpenStreetMap-Daten gemacht. Während dem Betrieb kann der Graph angepasst werden, so dass das Programm auf verspätete Züge reagieren kann.

OTP steht unter einer GNU Lesser General Public License.

Vorteile des OTP sind, dass er in Echtzeit auf Fahrplanänderungen und Verspätungen reagieren kann, dass OTP schon seit mehreren Jahren in verschiedenen Städten implementiert und getestet ist und dass unsere Arbeit vom OpenSource-Kollektiv in den Hauptcode aufgenommen werden könnte.

Nachteile des OTP sind, dass landesweite Implementationen erst in der Beta-Phase sind und somit noch nicht ausreichen getestet sind und dass das Programm unheimlich gross und verzweigt ist, so dass eine lange Einarbeitungszeit von Nöten ist.

2.3 R5

Rapid Realistic Routing on Real-world and Reimagined networks oder kurz R5 ist ein multimodales Trip-Planning-Tool. Er wurde von der Firma Conveyal entwickelt und basiert auf dem OTP.

R5 hat die Grundstruktur des OTP übernommen, jedoch verwendet R5 den RAPTOR-Algorithmus anstelle des A*-Algorithmus. Dadurch wurden einige Programmstrukturen verändert, da der RAPTOR-Algorithm nicht auf den Dijkstra-Algorithmus aufbaut und auch keinen Graphen verwendet.

R5 steht unter der MIT License.

Vorteile des R5 sind, dass er in Echtzeit auf Fahrplanänderung und Verspätungen reagieren kann und dass der RAPTOR-Algorithmus eine bessere Performance als die Dijkstra-basierten Algorithmen bietet.

Nachteile des R5 sind, dass er unter einer MIT License steht und dass der R5 von einer Firma entwickelt wurde, welche unsere Arbeit nicht übernehmen werden.

Abbildungsverzeichnis

Literatur

[1] Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection scan algorithm. *CoRR*, abs/1703.05997, 2017.

[2] Linus norton personal description.

[3] Assertis - Rail ticketing for Online. Mobile. Digital., December 2017. https://assertis.co.uk/.

[4] So you want to build a journey planner.

[5] journey-planner: GTFS based journey planner, December 2017. original-date: 2017-12-11T18:36:53Z.

[6] Train Tickets, December 2017.