

Inhaltsverzeichnis

1	Algorithmen	2
1.1	Graph Basierte Algorithmen	2
1.1.1	Dijkstra	2
1.1.2	A*	2
1.1.3	Bellman-Ford	2
1.2	Non Graph Basierte Algorithmen	3
1.2.1	RAPTOR	3
1.2.2	Connection Scan Algorithm	3
1.3	Transfer Pattern	4
2	Programme	6
2.1	Traintickets.to	6
2.2	Open Trip Planner	6
2.3	R5	7
3	Erkenntnisse	8
3.1	Daten	8
3.2	Algorithmen	8
3.3	Programme	8
4	AufgabeBA	11
4.1	AufgabenstellungBA	11
4.2	Lastenheft	12
4.3	Meilensteine	12
4.4	Zeitplan	13

1 Algorithmen

Die für einen Journey Planner zu lösende Aufgabe ist das Shortest-Path Problem. Um 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 Endpunkt mit dem niedrigsten kombinierten Gewicht.

1.1.1 Dijkstra

Der Dijkstra Algorithmus ist ein Single-Source-Shortest-Path Algorithmus. 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] [2]

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 Ziels führen präferiert und 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]

1.1.3 Bellman-Ford

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

node, 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. [2] [3]

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 ein 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 Runde werden alle Zuglinien gescannt, welche durch die Startstation verlaufen. Wenn eine andere Zuglinie die gescannte Zuglinie kreuzt, so wird die andere Zuglinie sowie 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 besitzt eine höhere Performanz als alle Graph Basierten Algorithmen. [4]

1.2.2 Connection Scan Algorithm

Der Connection Scan Algorithm, kurz CSA, ist schon vom Grundkonzept her 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 einem 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

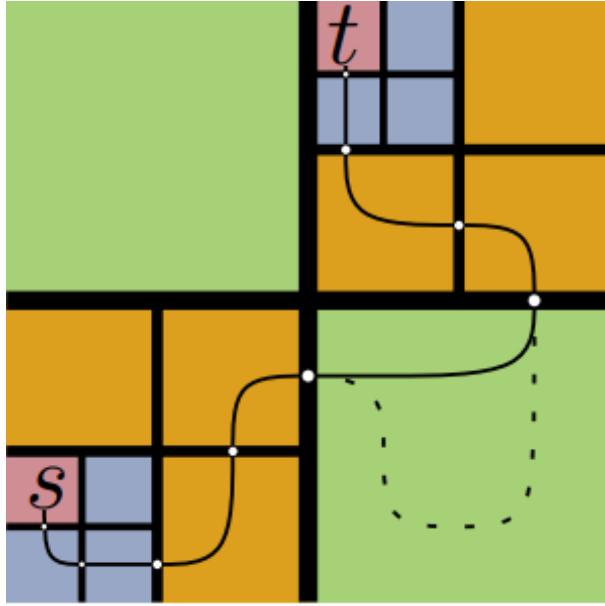


Abbildung 1: Veranschaulichung der Funktion des QuadTree-Preprocessings [5]

die 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. [5]

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 Dijkstra basierten Algorithmen haben jedoch den Nachteil, dass Query anfragen für landesweite Netzwerke immer noch mehrere Sekunden benötigen, weshalb der RAPTOR Algorithmus oder der CSA zu bevorzugen sind. [6]

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 Promille liegt. [7]

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-Lizenierung. Wir werden uns im Rahmen dieses Projektes auf die OpenSource Anwendungen beschränken.

2.1 Traintickets.to

Traintickets.to ist ein von Linus Norton [8] entwickelter Journey Planner für das englische Zugnetzwerk. Linus Norton ist CTO der Firma Assertis [9], welche das Projekt betreibt. Linus Norton erklärt den Programmierprozess in einem Blog [10]

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. [11] Diese werden dann von Hauptprogramm [12] gefiltert und über eine API [13] zur Verfügung gestellt. Assertis ist von der ATOC (Association of Train Operation Companies) lizenziert, so dass sie Ticketpreise anzeigen können sowie Tickets direkt verkaufen können. Es basiert auf PHP für den Algorithmus und Scala für die Transfer Pattern Generierung.

Traintickets.to besitzt zwei verschiedene Lizenzierungen. 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 in der Schweiz ohne Skalierung anwendbar ist. [14]

Nachteile der Traintickets.to Anwendung sind, dass modifizierte GTFS-Daten verwendet werden und diese so nicht direkt von der OpenData Plattform 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 [15] 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. [16]

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. [17]

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 im-

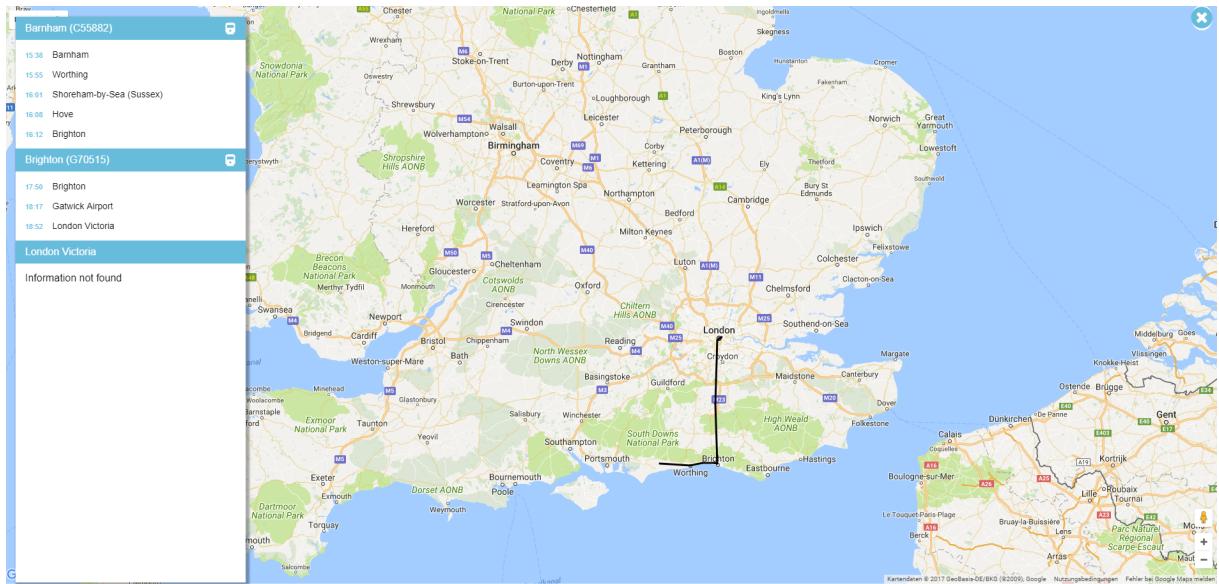


Abbildung 2: Traintickets.To Webseite [14]

plementiert 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 ausreichend getestet sind und dass das Programm unheimlich gross und verzweigt ist, so dass eine lange Einarbeitungszeit von Nötten 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 [18] 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-Algorithmus nicht auf den Dijkstra-Algorithmus aufbaut und auch keinen Graphen verwendet. [19]

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.

3 Erkenntnisse

3.1 Daten

Für unsere Anwendung ist der GTFS Datentyp dem HRDF Datentyp aus mehreren Gründen überlegen.

1. Alle zuvor genannten Programme basieren auf dem GTFS-Datentyp. Dies führt

3.2 Algorithmen

Alle von uns genannten Algorithmen wurden schon für Journey Planner verwendet. In den Papers des RAPTORS und des CSA wurden jeweils experimentelle Vergleiche mit Dijkstra basierten Algorithmen gezogen. In allen Testfällen hatten die Dijkstra basierten Algorithmen mindestens doppelt so lange Query-Zeiten. [5] [4]

Der RAPTOR-Algorithmus und der CSA sind sich in Vorgehen und Effizienz sehr ähnlich. Es gibt jedoch mehr Daten aus Experimenten für den CSA. Hannah Bast hat schon einen auf CSA und Transfer Pattern basierten Test mit dem Schweizer Schienennetz durchgeführt. Ihre Implementation hatte Query-Zeiten von wenigen Millisekunden. Dadurch können wir uns sicher sein, dass der CSA mit den Schweizer Daten kompatibel ist und genügend Skalierbarkeit mitbringt. [7]

Der CSA ist für unsere Arbeit am besten geeignet. Sollte jedoch durch andere Faktoren der RAPTOR-Algorithmus verlangt werden ist dieser auch akzeptabel.

3.3 Programme

Die drei Programme Traintickets.to, OTP und R5 wurden in Betracht gezogen.

R5 basiert zwar auf dem modernen RAPTOR Algorithmus, ist jedoch nicht für unser Projekt geeignet. Der Programmcode steht unter einer MIT-Lizenz. R5 wird von der Firma Conveyal [18] entwickelt. Wenn wir mit unserem Projekt auf das R5 Programm aufbauen, so besteht keine Chance dass unser Code in das Originalprogramm integriert wird.

Die beiden Programme Traintickets.to und OpenTripPlanner wurden genauer betrachtet. Die Programmierer beider Programme wurden von uns angeschrieben. Wir posteten eine Anfrage in die OTP Developer Mailing. Darin fragten wir an, ob sie zurzeit an einer Implementation des CSA in ihrem Programm arbeiten und wie sie zu der Idee unseres Projektes stehen. Nach nur kurzer Zeit bekamen wir eine Antwort. Eine Integration des CSA ist zurzeit nicht geplant. Projekte von neuen Personen sind gerne gesehen, jedoch wird der Code nur in das Hauptprogramm übernommen, wenn das OpenSource-Gremium den Code als gut erachtet. Dazu muss der Code den Programmrichtlinien entsprechen und der Strategie des OTP entsprechen. Die Strategie des OTP liegt darin, dass sie nicht ausschliesslich auf die schnellste Lösung setzen. Wichtiger ist, dass das Programm in Echtzeit auf Verspätungen und Fahrplanänderungen reagieren kann. Wir haben auch Linus Norton von Traintickets.to angeschrieben, haben jedoch keine Antwort erhalten.

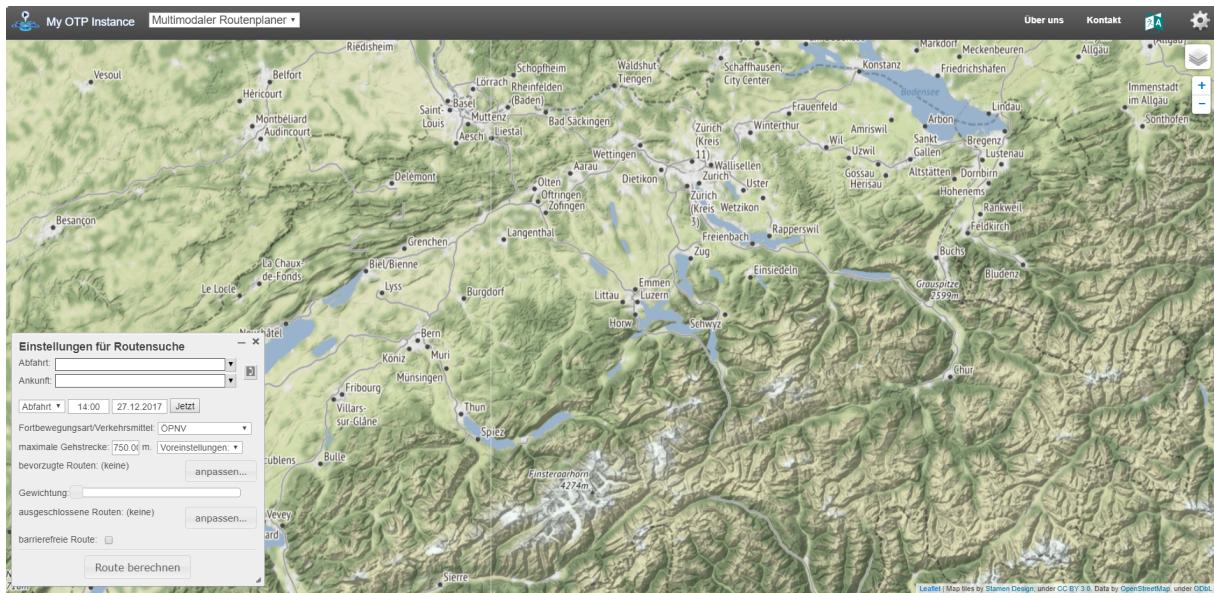


Abbildung 3: Implementation des OTP mit den Schweizer Daten

Um die Kompatibilität der Programme mit dem Schweizer System überprüfen versuchten wir die Programme mit den Schweizer GTFS-Daten zu betreiben. Traintickets.to lies sich nicht ohne weiteres auf das Schweizer System anwenden. Die GTFS-Daten konnten zwar in die für den CSA benötigte Form umgewandelt werden, jedoch funktionierte der darauffolgende CSA nicht. Wir konnten das Problem im Rahmen des Fachmodules leider nicht lokalisieren. Jedoch ist es von der Skalierbarkeit her möglich das Programm auf einer landesweiten Basis zu implementieren, da es zur Zeit auf dem Zugnetzwerk des United Kingdoms aufbaut. Auch die Implementation des OTP führte zu Problemen. Die Ausführung funktioniert mit GTFS-Daten, welche älter als der 27.09.2017 sind. Mit neueren GTFS-Files schlägt sie jedoch fehl. Wir konnten das Problem finden. Die SBB hat einige der Routen mit dem Routetype 1700 Miscellaneous bezeichnet. Das OTP Programm beachtete diesen Route-Type nicht, obwohl er GTFS konform ist. Wir meldeten das Problem an die Development-Mailing-List, wo man uns mitteilte, dass dieser Typ nicht unterstützt wird, da er nicht eindeutig einem Route-Type zugeordnet werden kann. Wenn wir OTP verwenden müssen wir dies für das Schweizer System selber implementieren. Die älteren GTFS-Daten funktionieren, da die SBB vor dem 27.09.2017 mit den Alten Route-Types arbeiteten und so den Route-Type 1700 nicht verwendeten. Die Implementation an sich funktionierte ohne Probleme und es konnten sogar die Schweizer OpenStreetMap-Daten integriert werden.

Wir verglichen die Programmstrukturen der beiden Programme. Traintickets.to besitzt einige Strukturprobleme. Linus Norton selbst sagte: 'The connection scan algorithm is inherently imperative and does not port well to Scala' (Linus Norton [?]) Es ist in mehrere Programme aufgeteilt, welche jeweils Teilaufgaben übernehmen. OTP besitzt einen sehr gut strukturierten Programmcode. Es ist nach Funktionen in verschiedene Ordner aufgeteilt, welche die jeweiligen Funktionen für die Aufgabe enthalten. Das Pro-

blem hierbei ist, dass OTP von mehr als 100 verschiedenen Entwicklern programmiert wurde. Es wurden schon mehr als 10'000 Commits durchgeführt. [16] Dadurch wird für uns das Verstehen des Programmes trotz der guten Struktur viel Zeit in Anspruch nehmen.

Aus diesen Erkenntnissen zogen wir, dass das OTP-Programm besser als Grundlage für unsere Bachelorarbeit geeignet ist. Es basiert nicht auf dem CSA. Wir werden den CSA für das OTP implementieren und dem OpenSource Gremium unsere Ergebnisse präsentieren, welche sie dann implementieren oder als Forschungswerte verwenden können.

4 AufgabeBA

4.1 AufgabenstellungBA

- Schreiben sie eine Implementation des CSA für den OpenTripPlanner. Dieser muss OTP-Development-Richtlinien konform sein.
- Implementieren sie die Schweizer GTFS und OpenStreetMap daten für den OTP, so dass Punkt zu Punkt Verbindungen in der Schweiz berechnet werden können.
- Implementieren sie den Schweizer realtime GTFS-Feed, so dass das Programm auf Verspätungen oder Fahrplanänderungen reagieren kann.
- Implementieren sie Skalierungsoptimierungen für das Programm, so dass es mit einem landesweiten System bessere Performanz liefert.
- Führen sie Performance Tests durch und vergleichen sie das Implementierte System mit dem auf Dijkstra basierenden OTP.
- Dokumentieren sie ihre Ergebnisse und schreiben sie einen ausführlichen Bericht.

4.2 Lastenheft

Funktionale Anforderungen		
Anforderung	Anforderungsart	Priorisierung
Der CSA muss als Berechnungsalgorithmus verwendet werden.	Muss	1
Es kann eine Verbindung zwischen 2 Stationen zu einem bestimmten Zeitpunkt errechnet werden.	Muss	2
Fusswege zu Stationen hin können miteinbezogen werden können.	Muss	3
Das Programm kann auf Verspätungen und Fahrplanänderungen reagieren.	Muss	4
Mehrere mögliche Verbindungen können angezeigt werden.	Muss	5
Fusswege zwischen verschiedenen Stationen können miteinbezogen werden.	Soll	6
Die Angezeigte Route soll den Fahrlinien, z.B. Schienen, folgen.	Kann	7

Nicht funktionale Anforderungen

- Der Programmcode muss den OTP-Development-Richtlinien entsprechen.
- Die Query-Zeit muss weniger als 1 Sekunde betragen.
- Die Preprocessing-Zeit muss weniger als 30 Minuten betragen.
- Die Query-Zeit soll schneller als die des originalen OTP sein.

4.3 Meilensteine

1. OTP Analysiert
2. OTP mit dem Schweitzer System implementieren
3. Basis CSA implementiert
4. CSA Spalierungsoptimierung implementiert
5. Performancetests durchgeführt

6. Bericht geschrieben

4.4 Zeitplan

Abbildungsverzeichnis

1	Veranschaulichung der Funktion des QuadTree-Preprocessings [5]	4
2	Traintickets.To Webseite [14]	7
3	Implementation des OTP mit den Schweitzer Daten	9

Literatur

- [1] R. Geisberger, "Advanced route planning in transportation networks," Feb. 2011. [Online]. Available: <http://algo2.iti.kit.edu/1814.php>
- [2] "Dijkstra's algorithm," Oct. 2013. [Online]. Available: <http://www.cse.unt.edu/tarau/teaching/AnAlgo>
- [3] "A survey paper of bellman-ford algorithm and dijkstra algorithm for finding shortest path in gis application," Feb. 2014. [Online]. Available: <http://www.ijpttjournal.org/volume-5/IJPTT-V5P402.pdf>
- [4] D. Delling, T. Pajor, and R. Werneck, "Round-based public transit routing, raptor," Jan. 2012. [Online]. Available: <https://www.microsoft.com/en-us/research/publication/round-based-public-transit-routing>
- [5] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner, "Connection scan algorithm," *CoRR*, vol. abs/1703.05997, 2017. [Online]. Available: <http://arxiv.org/abs/1703.05997>
- [6] H. Bast, M. Hertel, and S. Storandt, "Scalable transfer patterns," 2016. [Online]. Available: <http://pubs.siam.org/doi/abs/10.1137/1.9781611974317.2>
- [7] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger, "Fast routing in very larg public transportation networks using transfer patterns," 2010. [Online]. Available: <https://research.google.com/pubs/pub36672.html>
- [8] "Linus norton personenbeschreibung," Dec. 2017. [Online]. Available: <https://www.linusnorton.co.uk>
- [9] "Assertis - rail ticketing für online, mobile, digital." Dec. 2017. [Online]. Available: <https://assertis.co.uk>
- [10] "Entstehungsblog über traintickets.to von linus norton," Dec. 2017. [Online]. Available: <http://ljn.io/posts/so-you-want-to-build-a-journey-planner>
- [11] "Git-repository der patterngenerierung von traintickets.to," Dec. 2017. [Online]. Available: <https://github.com/open-track/transfer-pattern-generator-scala>
- [12] "Git-repository des hauptprogrammes von traintickets.to," Dec. 2017. [Online]. Available: <https://github.com/open-track/journey-planner>
- [13] "Git-repository der weboberfläche von traintickets.to," Dec. 2017. [Online]. Available: <https://github.com/open-track/traintickets.to>
- [14] "Traintickets.to mit dem uk-schienennetz," Dec. 2017. [Online]. Available: <http://traintickets.to>

- [15] “Webseite der maven-repository,” Dec. 2017. [Online]. Available: <https://maven.apache.org/guides/introduction/introduction-to-repositories.html>
- [16] “Webseite des otp-projektes,” Dec. 2017. [Online]. Available: <http://www.opentripplanner.org>
- [17] “Git-repository des otp-projektes,” Dec. 2017. [Online]. Available: <https://github.com/opentripplanner/OpenTripPlanner>
- [18] “Conveyal firmenwebseite,” Dec. 2017. [Online]. Available: <https://www.conveyal.com/trip-planning>
- [19] “Git-repository des r5-projektes,” Dec. 2017. [Online]. Available: <https://github.com/conveyal/r5/blob/master>