Bachelorarbeit

Semantischer Vergleich von Fahrplandaten

Phong Tran

Albert-Ludwigs-Universität Freiburg
Technische Fakultät
Lehrstuhl für Algorithmen und Datenstrukturen

16. Juni 2017

Abgabedatum

16. 06. 2017

Gutacherin

Prof. Dr. Hannah Bast

Betreuer

Patrick Brosi

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum	Unterschrift

Abstract

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung eines Programms, dass den Vergleich von zwei Fahrplandaten im GTFS (General Transit Feed Specification) Format ermöglicht. Viele Verkehrsunternehmen veröffentlichen ihre Fahrpläne als GTFS Feeds. Möchte man nun zwei Feeds zusammenfügen kann es zu Überschneidungen bei den Trips kommen, falls diese Feeds das gleiche Gebiet abdecken. Unser Programm vergleicht zwei gegebene Feeds und gibt Informationen zu abweichenden Servicezeiten, fehlenden Fahrten und Haltestellen aus. Diese Informationen können dann anschließend von weiteren Programmen verarbeitet werden.

Danksagungen

Ich möchte mich bei Frau Prof. Dr. Hannah Bast für die Bereitstellung dieses Themas bedanken. Großer Dank gilt auch meinem Betreuer Patrick Brosi für seine Unterstützung und Ideenvorschläge. Ich möchte mich zudem bei meiner Familie und Freunden für ihre Unterstützung während meines Studiums bedanken.

Inhaltsverzeichnis

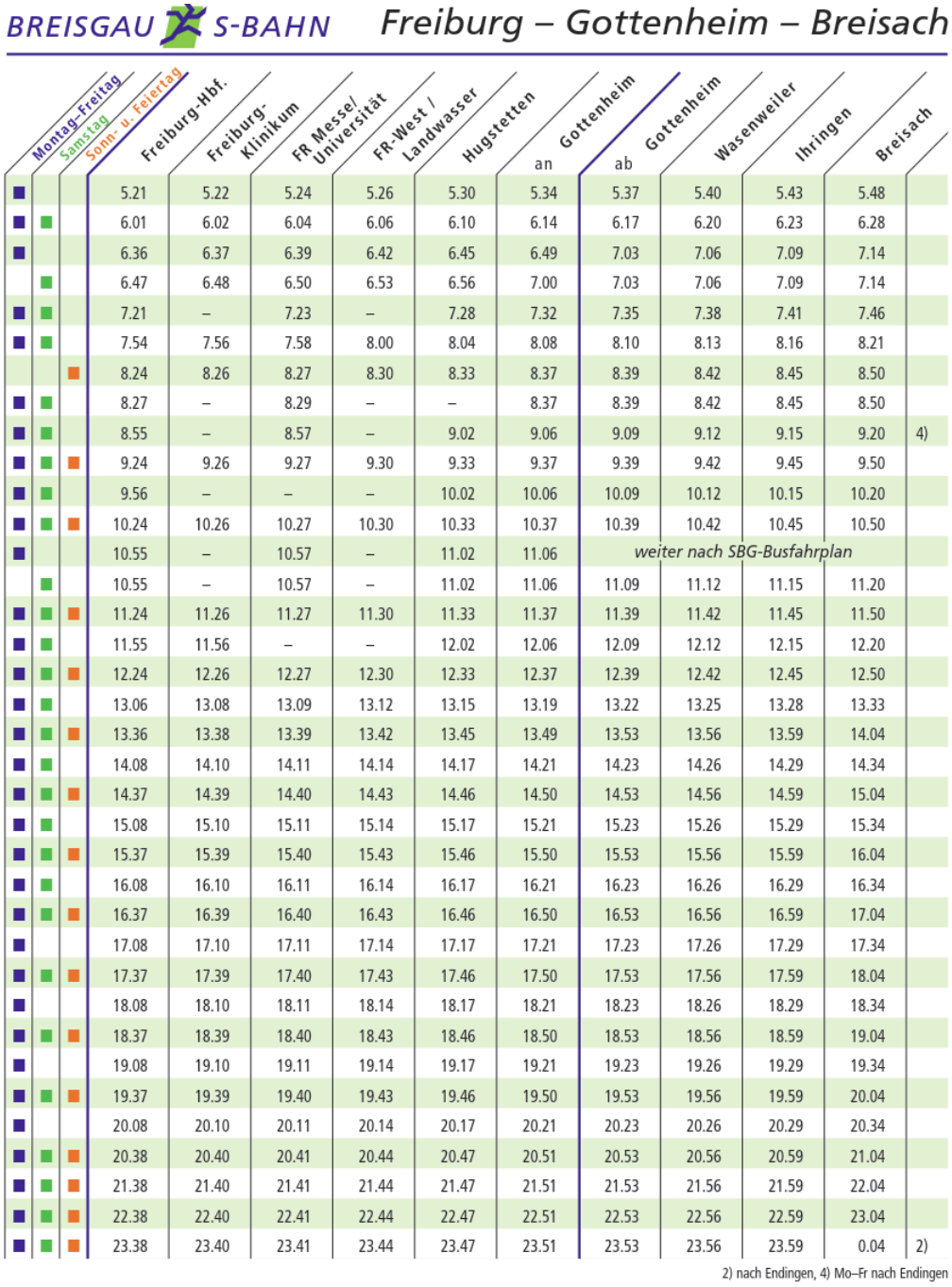
1 Einleitung

1.1 Motivation

Die Planung einer Reise mittels öffentlicher Verkehrsmitteln ist im Vergleich zur Vergangenheit deutlich komfortabler geworden. Früher musste man sich noch den Reiseplan mühselig mit einem Fahrplanheftchen zusammenstellen. Wann fährt der Zug ab und von welchem Gleis? Muss man während der Reise noch umsteigen, so muss man sich noch zusätzlich informieren. Oft kann es sein, dass die Informationen nicht an einem Ort zentriert sind. Dadurch wird zusätzliche Zeit benötigt um den optimalen Plan zu erhalten. Im Zeitalter des Internets und der Smartphones ist die Planung deutlich angenehmer. Fast jedes größere Verkehrsunternehmen verfügt über eine Webseite oder mobile Applikation mit der man sich schnell einen Reiseplan erstellen kann. Man gibt seine gewünschte Abfahrt- bzw. Ankunftszeit, den Start und das Ziel an, anschließend bekommt man sofort einen optimalen Reiseplan mit allen nötigen Angaben angezeigt. Viele Unternehmen verwenden das GTFS-Format um ihre Fahrplandaten zu repräsentieren und zu veröffentlichen. Die Informationen werden in einem Feed gespeichert, der dann für weitere Applikationen verwendet werden kann. Ein großer Vorteil des GTFS-Formates ist, dass im Prinzip jeder einen Feed erstellen kann. So hat sich in den letzten Jahren eine große Community gebildet die Feeds aus Drittquellen erzeugt und bereitstellt.

1.2 Problemstellung

Hat man nun zwei Feeds von verschiedenen Anbietern die das selbe Gebiet abdecken, so kann es zu Doppelungen der Informationen kommen. Als Beispiel sei hier die Breisgau-S-Bahn genannt. In den Abbildung 1 und 2 sind Fahrpläne der Breisgau-S-Bahn GmbH und der Regio-Verkehrsverbund Freiburg GmbH für die Route *Freiburg - Gottenheim - Breisach* zu sehen. In dieser Arbeit entwickeln wir ein Tool, welches die Unterschiede bei den Trips (ein Trip ist eine einzelne Fahrt) zweier Feeds ausgibt. Dabei handelt es sich z.B. um fehlende Trips, Service-Tage oder Haltestellen.



BREISGAU S-BAHN

Freiburg – Gottenheim – Breisach

Montag–Freitag	Samstag	Sonn- u. Feiertag	Freiburg-Hbf.	Freiburg-Klinikum	FR Messe/Universität	FR-West / Landwasser	Hugstetten	Gottenheim an	Gottenheim ab	Wasenweiler	Ihringen	Breisach	
■			5.21	5.22	5.24	5.26	5.30	5.34	5.37	5.40	5.43	5.48	
■	■		6.01	6.02	6.04	6.06	6.10	6.14	6.17	6.20	6.23	6.28	
■			6.36	6.37	6.39	6.42	6.45	6.49	7.03	7.06	7.09	7.14	
	■		6.47	6.48	6.50	6.53	6.56	7.00	7.03	7.06	7.09	7.14	
■	■		7.21	–	7.23	–	7.28	7.32	7.35	7.38	7.41	7.46	
■	■		7.54	7.56	7.58	8.00	8.04	8.08	8.10	8.13	8.16	8.21	
		■	8.24	8.26	8.27	8.30	8.33	8.37	8.39	8.42	8.45	8.50	
■	■		8.27	–	8.29	–	–	8.37	8.39	8.42	8.45	8.50	
■	■		8.55	–	8.57	–	9.02	9.06	9.09	9.12	9.15	9.20	4)
■	■	■	9.24	9.26	9.27	9.30	9.33	9.37	9.39	9.42	9.45	9.50	
■	■		9.56	–	–	–	10.02	10.06	10.09	10.12	10.15	10.20	
■	■	■	10.24	10.26	10.27	10.30	10.33	10.37	10.39	10.42	10.45	10.50	
■			10.55	–	10.57	–	11.02	11.06	*weiter nach SBG-Busfahrplan*				
	■		10.55	–	10.57	–	11.02	11.06	11.09	11.12	11.15	11.20	
■	■	■	11.24	11.26	11.27	11.30	11.33	11.37	11.39	11.42	11.45	11.50	
■	■		11.55	11.56	–	–	12.02	12.06	12.09	12.12	12.15	12.20	
■	■	■	12.24	12.26	12.27	12.30	12.33	12.37	12.39	12.42	12.45	12.50	
■	■		13.06	13.08	13.09	13.12	13.15	13.19	13.22	13.25	13.28	13.33	
■	■	■	13.36	13.38	13.39	13.42	13.45	13.49	13.53	13.56	13.59	14.04	
■	■		14.08	14.10	14.11	14.14	14.17	14.21	14.23	14.26	14.29	14.34	
■	■	■	14.37	14.39	14.40	14.43	14.46	14.50	14.53	14.56	14.59	15.04	
■	■		15.08	15.10	15.11	15.14	15.17	15.21	15.23	15.26	15.29	15.34	
■	■	■	15.37	15.39	15.40	15.43	15.46	15.50	15.53	15.56	15.59	16.04	
■	■		16.08	16.10	16.11	16.14	16.17	16.21	16.23	16.26	16.29	16.34	
■	■	■	16.37	16.39	16.40	16.43	16.46	16.50	16.53	16.56	16.59	17.04	
■			17.08	17.10	17.11	17.14	17.17	17.21	17.23	17.26	17.29	17.34	
■	■	■	17.37	17.39	17.40	17.43	17.46	17.50	17.53	17.56	17.59	18.04	
■			18.08	18.10	18.11	18.14	18.17	18.21	18.23	18.26	18.29	18.34	
■	■	■	18.37	18.39	18.40	18.43	18.46	18.50	18.53	18.56	18.59	19.04	
■			19.08	19.10	19.11	19.14	19.17	19.21	19.23	19.26	19.29	19.34	
■	■	■	19.37	19.39	19.40	19.43	19.46	19.50	19.53	19.56	19.59	20.04	
■			20.08	20.10	20.11	20.14	20.17	20.21	20.23	20.26	20.29	20.34	
■	■	■	20.38	20.40	20.41	20.44	20.47	20.51	20.53	20.56	20.59	21.04	
■	■	■	21.38	21.40	21.41	21.44	21.47	21.51	21.53	21.56	21.59	22.04	
■	■	■	22.38	22.40	22.41	22.44	22.47	22.51	22.53	22.56	22.59	23.04	
■	■	■	23.38	23.40	23.41	23.44	23.47	23.51	23.53	23.56	23.59	0.04	2)

2) nach Endingen, 4) Mo–Fr nach Endingen

Abbildung 1: Fahrplan der Breisgau-S-Bahn GmbH für die Strecke *Freiburg - Gottenheim - Breisach*

729 Freiburg (Breisgau) Hbf - Gottenheim - Breisach

Fahrplanauskunft RVF Tel.: (01805/779966); Internet: www.rvf.de; Breisgau-S-Bahn GmbH, Bahnhof, Üsenberger Str. 9, 79346 Endingen a.K., Tel.: 07821/9960770, http://www.breisgau-s-bahn.de

RVF

VERKEHRSHINWEIS		Montag - Freitag											NF SBG									
Freiburg (Breisgau) Hbf	ab	5.21	6.01	6.36	7.21	7.54	8.27	8.55	9.24	9.56	10.24	10.55		11.24	11.55	12.24	13.06	13.36	14.08	14.37	15.08	15.37
- Klinikum		5.22	6.02	6.37	|	7.56	|	|	9.26	|	10.26	|		11.26	11.56	12.26	13.08	13.38	14.10	14.39	15.10	15.39
- Neue Messe / Universität		5.24	6.04	6.39	7.23	7.58	8.29	8.57	9.27	|	10.27	10.57		11.27	|	12.27	13.09	13.39	14.11	14.40	15.11	15.40
- Freiburg West		5.26	6.06	6.42	|	8.00	|	|	9.30	|	10.30	|		11.30	|	12.30	13.12	13.42	14.14	14.43	15.14	15.43
Hugstetten		5.30	6.10	6.45	7.28	8.04	|	9.02	9.33	10.02	10.33	11.02		11.33	12.02	12.33	13.15	13.45	14.17	14.46	15.17	15.46
Gottenheim	an	5.34	6.14	6.49	7.32	8.08	8.37	9.06	9.37	10.06	10.37	11.06		11.37	12.06	12.37	13.19	13.49	14.21	14.50	15.21	15.50
Gottenheim	ab	5.37	6.17	7.03	7.35	8.10	8.39	9.09	9.39	10.09	10.39		11.10	11.39	12.09	12.39	13.22	13.53	14.23	14.53	15.23	15.53
Wasenweiler		5.40	6.20	7.06	7.38	8.13	8.42	9.12	9.42	10.12	10.42		▲11.16	11.42	12.12	12.42	13.25	13.56	14.26	14.56	15.26	15.56
Ihringen		5.43	6.23	7.09	7.41	8.16	8.45	9.15	9.45	10.15	10.45		▲11.20	11.45	12.15	12.45	13.28	13.59	14.29	14.59	15.29	15.59
Breisach	an	5.48	6.28	7.14	7.46	8.21	8.50	9.20	9.50	10.20	10.50		11.43	11.50	12.20	12.50	13.33	14.04	14.34	15.04	15.34	16.04

RVF

VERKEHRSHINWEIS		Montag - Freitag													NF SBG	Samstag							
Freiburg (Breisgau) Hbf	ab	16.08	16.37	17.08	17.37	18.08	18.37	19.08	19.37	20.08	20.38	21.38	22.38	23.38		6.01	6.47	7.21	7.54	8.27	8.55	9.24
- Klinikum		16.10	16.39	17.10	17.39	18.10	18.39	19.10	19.39	20.10	20.40	21.40	22.40	23.40		6.02	6.48	|	7.56	|	|	9.26
- Neue Messe / Universität		16.11	16.40	17.11	17.40	18.11	18.40	19.11	19.40	20.11	20.41	21.41	22.41	23.41		6.04	6.50	7.23	7.58	8.29	8.57	9.27
- Freiburg West		16.14	16.43	17.14	17.43	18.14	18.43	19.14	19.43	20.14	20.44	21.44	22.44	23.44		6.06	6.53	|	8.00	|	|	9.30
Hugstetten		16.17	16.46	17.17	17.46	18.17	18.46	19.17	19.46	20.17	20.47	21.47	22.47	23.47		6.10	6.56	7.28	8.04	|	9.02	9.33
Gottenheim	an	16.21	16.50	17.21	17.50	18.21	18.50	19.21	19.50	20.21	20.51	21.51	22.51	23.51		6.14	7.00	7.32	8.08	8.37	9.06	9.37
Gottenheim	ab	16.23	16.53	17.23	17.53	18.23	18.53	19.23	19.53	20.23	20.53	21.53	22.53	23.53	▲1.00	6.17	7.02	7.35	8.10	8.39	9.09	9.39
Wasenweiler		16.26	16.56	17.26	17.56	18.26	18.56	19.26	19.56	20.26	20.56	21.56	22.56	23.56	▲1.08	6.20	7.06	7.38	8.13	8.42	9.12	9.42
Ihringen		16.29	16.59	17.29	17.59	18.29	18.59	19.29	19.59	20.29	20.59	21.59	22.59	23.59	▲1.12	6.23	7.09	7.41	8.16	8.45	9.15	9.45
Breisach	an	16.34	17.04	17.34	18.04	18.34	19.04	19.34	20.04	20.34	21.04	22.04	23.04	0.04	1.24	6.28	7.14	7.46	8.21	8.50	9.20	9.50

RVF

		Samstag																				
Freiburg (Breisgau) Hbf	ab	9.56	10.24	10.55	11.24	11.55	12.24	13.06	13.36	14.08	14.37	15.08	15.37	16.08	16.37	17.37	18.37	19.37	20.38	21.38	22.38	23.38
- Klinikum		|	10.26	|	11.26	11.56	12.26	13.08	13.38	14.10	14.39	15.10	15.39	16.10	16.39	17.39	18.39	19.39	20.40	21.40	22.40	23.40
- Neue Messe / Universität		|	10.27	10.57	11.27	|	12.27	13.09	13.39	14.11	14.40	15.11	15.40	16.11	16.40	17.40	18.40	19.40	20.41	21.41	22.41	23.41
- Freiburg West		|	10.30	|	11.30	|	12.30	13.12	13.42	14.14	14.43	15.14	15.43	16.14	16.43	17.43	18.43	19.43	20.44	21.44	22.44	23.44
Hugstetten		10.02	10.33	11.02	11.33	12.02	12.33	13.15	13.45	14.17	14.46	15.17	15.46	16.17	16.46	17.46	18.46	19.46	20.47	21.47	22.47	23.47
Gottenheim	an	10.06	10.37	11.06	11.37	12.06	12.37	13.19	13.49	14.21	14.50	15.21	15.50	16.21	16.50	17.50	18.50	19.50	20.51	21.51	22.51	23.51
Gottenheim	ab	10.09	10.39	11.09	11.39	12.09	12.39	13.22	13.53	14.23	14.53	15.23	15.53	16.23	16.53	17.53	18.53	19.53	20.53	21.53	22.53	23.53
Wasenweiler		10.12	10.42	11.12	11.42	12.12	12.42	13.25	13.56	14.26	14.56	15.26	15.56	16.26	16.56	17.56	18.56	19.56	20.56	21.56	22.56	23.56
Ihringen		10.15	10.45	11.15	11.45	12.15	12.45	13.28	13.59	14.29	14.59	15.29	15.59	16.29	16.59	17.59	18.59	19.59	20.59	21.59	22.59	23.59
Breisach	an	10.20	10.50	11.20	11.50	12.20	12.50	13.33	14.04	14.34	15.04	15.34	16.04	16.34	17.04	18.04	19.04	20.04	21.04	22.04	23.04	0.04

RVF

VERKEHRSHINWEIS		Samstag NF Q SBG			Sonn- und Feiertag															NF SBG
Freiburg (Breisgau) Hbf	ab				8.24	9.24	10.24	11.24	12.24	13.36	14.37	15.37	16.37	17.37	18.37	19.37	20.38	21.38	22.38	23.38	
- Klinikum					8.26	9.26	10.26	11.26	12.26	13.38	14.39	15.39	16.39	17.39	18.39	19.39	20.40	21.40	22.40	23.40	
- Neue Messe / Universität					8.27	9.27	10.27	11.27	12.27	13.39	14.40	15.40	16.40	17.40	18.40	19.40	20.41	21.41	22.41	23.41	
- Freiburg West					8.30	9.30	10.30	11.30	12.30	13.42	14.43	15.43	16.43	17.43	18.43	19.43	20.44	21.44	22.44	23.44	
Hugstetten					8.33	9.33	10.33	11.33	12.33	13.45	14.46	15.46	16.46	17.46	18.46	19.46	20.47	21.47	22.47	23.47	
Gottenheim	an				8.37	9.37	10.37	11.37	12.37	13.49	14.50	15.50	16.50	17.50	18.50	19.50	20.51	21.51	22.51	23.51	
Gottenheim	ab	▲1.00			8.39	9.39	10.39	11.39	12.39	13.53	14.53	15.53	16.53	17.53	18.53	19.53	20.53	21.53	22.53	23.53	▲1.00
Wasenweiler		▲1.08			8.42	9.42	10.42	11.42	12.42	13.56	14.56	15.56	16.56	17.56	18.56	19.56	20.56	21.56	22.56	23.56	▲1.08
Ihringen		▲1.12			8.45	9.45	10.45	11.45	12.45	13.59	14.59	15.59	16.59	17.59	18.59	19.59	20.59	21.59	22.59	23.59	▲1.12
Breisach	an	1.24			8.50	9.54	10.50	11.50	12.50	14.04	15.04	16.04	17.04	18.04	19.04	20.04	21.04	22.04	23.04	0.04	1.24

Q = nicht 24.12. SBG = Busfahrt der SBG ▲ = Bus hält nicht direkt am Bahnhof NF = Niederflurfahrzeug

Reisegruppen mit mehr als 20 Reisenden müssen 1 Woche vor dem Reisetag angemeldet werden. Tel. 07821/9960770; Fax 07642 / 9124-30
Fahrradmitnahme im Rahmen des vorhandenen Platzes möglich.
Einstiegshilfe durch Zugführer im ersten Fahrzeug. Anmeldung: 07821/9960770

Abbildung 2: Fahrplan der Regio-Verkehrsverbund Freiburg GmbH für die Strecke *Freiburg - Gottenheim - Breisach*

2 Stand der Forschung

Nach ausführlicher Recherche konnten wir keine Forschung finden, die sich mit dem Thema unserer Arbeit auseinandersetzt. Wir möchten deshalb an dieser Stelle einige Arbeiten vorstellen die sich generell mit GTFS Feeds und deren Anwendung beschäftigen.

In 2013 veröffentlichen Antrim u.a. [1] eine Arbeit die Unternehmen bei der Erstellung und Verbreitung von GTFS Daten informiert. Dabei werden Programme gelistet mit denen es möglich ist die bestehenden Daten in das GTFS Format umzuwandeln. Eine weitere Möglichkeit sei die Erstellung und Instandhaltung der GTFS Daten an eine externe Firma auszulagern. Zur Verbreitung der GTFS Daten werden mögliche Risiken genannt, unter anderem rechtliche Risiken wegen ungenauer Daten. Nach Antrim u.a. glauben allerdings viele Unternehmen, dass die potentiellen Vorteile öffentlicher GTFS Daten, z.B. die Entwicklung neuer innovativer Applikationen, die Risiken überwiegen. Außerdem werden eine Vielzahl möglicher Verwendungszwecke der GTFS Daten angegben, beispielsweise Visualisierung der Daten und mobile Applikationen für Fahrplandaten. Ein weiterer wichtiger Verwendungszweck ist die Reise-/Routenplanung. Zu diesem Thema veröffentlichten Bast u.a. [2] eine Arbeit, die eine neue Methode zur Routenberechnung in öffentlichen Verkehrsnetzen vorstellte.

Die Methode basiert auf sogenannten *transfer patterns*. Ein *transfer pattern* besteht dabei aus der Startstation, der Sequenz von Umstiegen (engl.: transfer) und der Endstation. Aufgrund der Datengröße ist es nicht möglich alle optimalen *transfer patterns* zwischen allen Stationspaaren vorab zu berechnen und zu speichern. Stattdessen werden einzelne Teile von *transfer patterns* vorab berechnet. Durch Kombinierung dieser Einzelteile ist es möglich alle optimalen *transfer patterns* zu generieren. Bei einer Anfrage für eine Fahrt von A nach B werden alle möglichen Kombinationen berechnet, die einen *transfer pattern* von A nach B bilden. Anschließend wird mit allen erzeugten *transfer patterns* ein *query graph* gebildet und mit Hilfe eines Kürzesten-Weg Algorithmus kann die optimale Verbindung von A nach B gefunden werden.

In 2017 schrieb Zhang [3] eine Bachelorabeit über die Extrahierung öffentlicher Transitdaten aus OpenStreetMap (OSM). Der Prozess ist dabei in 3 Schritte unterteilt.

Zunächst werden die relevanten Informationen aus den OSM Daten ausgelesen und ein Netzwerk aufgebaut. Der nächste Schritt ist Fehler innerhalb des Netzwerkes zu beheben. Dazu zählen unter anderem die korrekte Reihenfolge, Lückenschließung und korrekte Klassifizierung von Knoten. Wurden die Fehler des Netzwerkes behoben, so wird als letzter Schritt ein GTFS Feed aus dem Netzwerk erstellt. Dabei werden anhand von einigen Regeln Pseudo-Ankunfts- und Abfahrtszeiten generiert.

Für die Evaluation unseres Tools vergleichen wir eine von Zhang's Tool generiertes Feed mit einem Feed aus offiziellen Fahrplandaten.

3 Algorithmen und Datenstrukturen

In diesem Kapitel gehen wir auf die relevanten Algorithmen und Datenstrukturen für das Programm ein. Im Abschnitt 3.6 geben wir einen kurzen Überblick zu den verwendeten externen Bibliotheken.

3.1 Hash-Tabelle

In einer Hash-Tabelle (oder auch Hash-Map) können Elemente gemeinsam mit einem Schlüssel gespeichert werden. Das Element lässt sich dann wiederum mit Hilfe des Schlüssels finden [4]. Mit einer Hash-Funktion wird ein sogannter Hash-Code ermittelt, der bestimmt wo sich das Element in der Hash-Tabelle befindet. Im Idealfall hat jedes Element einen anderen Hash-Code, sodass die Elemente in der Tabelle gleichmäßig verteilt sind. Es kann aber auch vorkommen dass zwei unterschiedliche Elemente den selben Hash-Code haben. Dieses Ereignis wird *Kollision* genannt. Tritt eine Kollision auf, so wird ein neuer Platz für das neue Element gesucht. Das ganze wird *Sondieren* genannt und es gibt dafür unterschiedliche Strategien, z.B. wird das Element in den nächsten freien Platz gesetzt [4]. Wir verwenden größtenteils Hash-Tabellen für unsere Datenstruktur um GTFS Feeds zu speichern, da viele Elemente des GTFS-Formats eine einzigartige *id* besitzen, welche wir als Schlüssel verwenden können.

Ein großer Vorteil von Hash-Tabellen ist die Laufzeit um ein Element zu suchen. Ist die Hash-Funktion gut gewählt und die Elemente sind gleichmäßig verteilt, haben wir eine konstante Lauftzeit von $O(1)$. Im schlimmsten Fall haben wir eine Laufzeit von $O(n)$. Das ist der Fall bei einer Hash-Funktion die für jedes Element den gleichen Wert ausgibt. Da dies aber eher die Ausnahme ist, haben wir generell eine konstante Laufzeit von $O(1)$.

In Abbildung 3 ist eine graphische Darstellung zu sehen. In diesem Beispiel werden Telefonnummern von Personen gespeichert. Dabei ist der Schlüssel der Name und die Telefonnummer ist der Wert. Mittels einer Hash-Funktion wird bestimmt in welchen Index der Tabelle das Schlüssel-Werte Paar gespeichert wird.

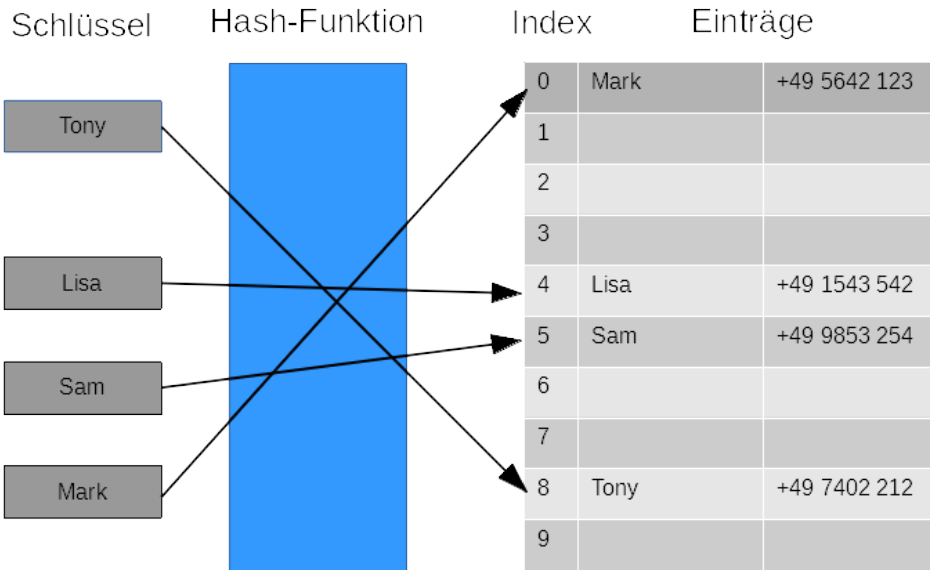


Abbildung 3: Graphische Darstellung einer Hashtabelle

3.2 Prioritätswarteschlange

Eine Prioritätswarteschlange (engl.: priority queue) speichert eine Menge an Elementen und gibt das Element mit dem minimalesten Wert aus. Im Prinzip ist eine Prioritätswarteschlange ein Array bei dem an der ersten Stelle das Element mit dem minimalsten Wert steht. Das Array ist dabei ein Heap. Wird ein Element entfernt so muss die Heap Eigenschaft repariert werden. Die Heap-Eigenschaft besagt, dass der Wert eines Knotens kleiner sein muss als der Wert seiner Kinder [5]. Wir verwenden die Prioritätswarteschlange in unserem Programm um die Ergebnisse der Vergleiche nach bestimmten Kriterien abzuarbeiten (siehe Abschnitt 4.2.2).

Die Laufzeiten für das Einfügen bzw. Löschen eines Elements hängen von der Laufzeit ab, die benötigt wird um die Heap Eigenschaft zu reparieren. Die Laufzeit der Reparatur beträgt $O(log\ n)$. Um das kleinste Element zu holen wird eine Laufzeit von $O(1)$ benötigt.

In Abbildung 4 ist die Reparatur eines Heaps zu sehen. Wird im oberen Baum nun der Wert am ersten Index entfernt, so wird der Wert vom letzten Index an die erste Stelle gesetzt. In diesem Fall ist es die Zahl 5. Nun ist allerdings die Heap-Eigenschaft verletzt, da beide Kinder (3 und 4) kleiner sind als der Elternknoten. Um die Heap-Eigenschaft wiederherzustellen, vertauschen wir das Kind mit dem kleineren Wert

mit dem Elternknoten.

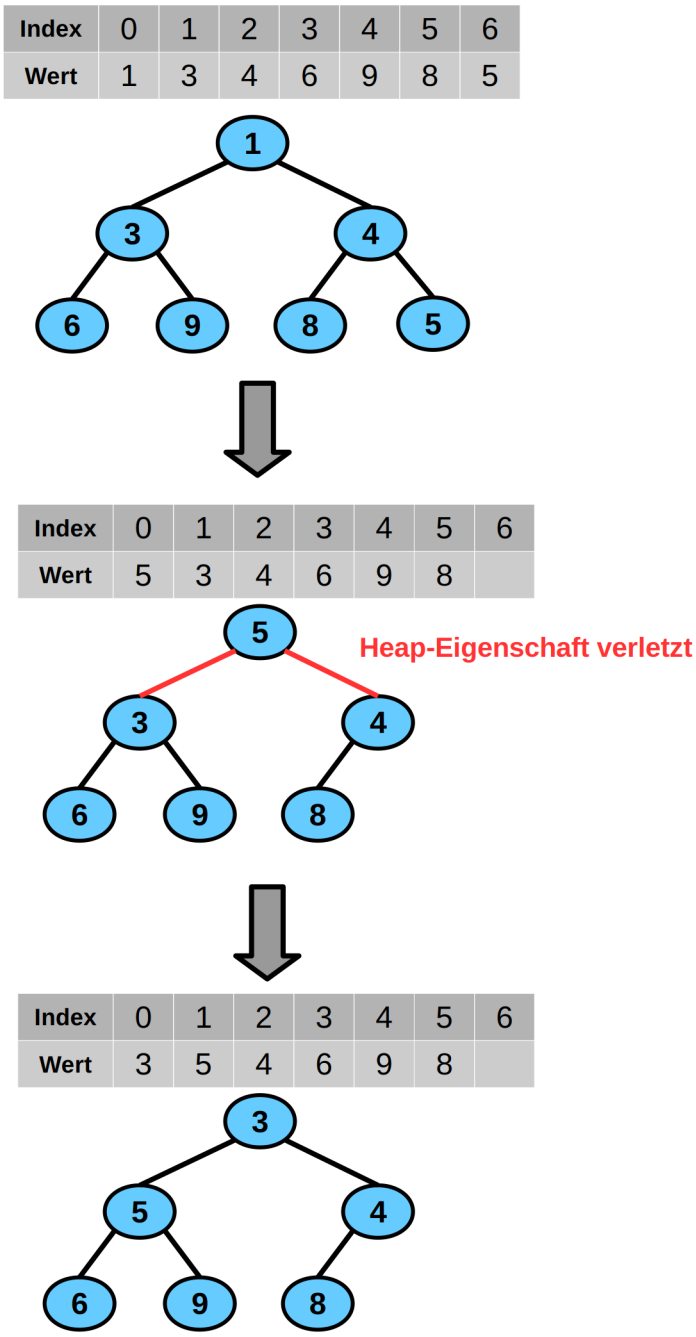


Abbildung 4: Graphische Darstellung einer Reparatur der Heap-Eigenschaft

3.3 GTFS-Format

GTFS steht für General Transit Feed Specification [6] und wurde ursprünglich von Google erstellt. Ein GTFS Feed ist eine Sammlung von CSV (comma-separated values) Dateien. Im Standard sind ingesamt 13 solcher Dateien angegeben. Davon sind 7 optional. Im Folgenden wird eine kurze Übersicht über die von uns verwendeten Dateien gegeben:

- **agency.txt:** Enthält Informationen zu den Agenturen, die Daten für den Feed bereitstellen. Jeder Agentur wird eine einzigartige *agency_id* zugewiesen.
- **calendar.txt:** Enthält Informationen zu den Servicezeiten. Dazu gehören ein Wochenplan an denen ein Trip verfügbar ist und eine Angabe zur Start- und Endzeit des Service. Jeder Service hat eine einzigartige *service_id*.
- **calendar_dates.txt (optional):** Enthält Tage an denen ein Trip anders verkehrt als in calendar.txt angegeben.
- **frequencies.txt (optional):** Enthält Trips die zu festen Intervallen fahren. Beispielsweise ein Bus der von 9 Uhr bis 10 Uhr alle 10 Minuten abfährt.
- **routes.txt:** Enthält Informationen zu Routen die von den Argenturen genutzt werden. Jede Route hat eine einzigartige *route_id*.
- **stops.txt:** Enthält Informationen zu den Haltestellen wie Name oder geographische Lage. Jede Haltestelle hat eine einzigartige *stop_id*.
- **stop_times.txt:** Definiert die Reihenfolge der Haltestellen eines Trips. Es wird zudem angegeben zu welchen Zeiten ein Trip diese Haltestelle anfährt und wieder verlässt.
- **trips.txt:** Enthält Informationen zu den Trips des Feeds. Ein Trip beschreibt eine einzelne Fahrt auf einer Route. Jeder Trip hat eine einzigartige *trip_id*.

Auf der folgenden Seite sind einige Auszüge der CSV Dateien. Die erste Zeile gibt dabei die Namen der Felder an, die durch ein Kommazeichen getrennt sind.

```
route_id, service_id, trip_id, trip_headsign, shape_id, direction_id
BNSF, A1, BNSF_BN1200_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1201_V1_A, Naperville, BNSF_OB_1,  0
BNSF, A1, BNSF_BN1202_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1204_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1205_V1_A, Aurora, BNSF_OB_1,  0
BNSF, A1, BNSF_BN1206_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1208_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1209_V1_A, Westmont, BNSF_OB_1,  0
BNSF, A1, BNSF_BN1210_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1211_V1_A, Aurora, BNSF_OB_1,  0
BNSF, A1, BNSF_BN1212_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1214_V1_A, Chicago Union Station, BNSF_IB_1,  1
BNSF, A1, BNSF_BN1215_V1_A, Westmont, BNSF_OB_1,  0
BNSF, A1, BNSF_BN1216_V1_A, Chicago Union Station, BNSF_IB_1,  1
```

Abbildung 5: Auszug aus *trips.txt*

```
trip_id, arrival_time, departure_time, stop_id, stop_sequence
BNSF_BN1200_V1_A,  04:30:00,  04:30:00,  AURORA,  1
BNSF_BN1200_V1_A,  04:39:00,  04:39:00,  ROUTE59,  3
BNSF_BN1200_V1_A,  04:44:00,  04:44:00,  NAPERVILLE,  4
BNSF_BN1200_V1_A,  04:50:00,  04:50:00,  LISLE,  5
BNSF_BN1200_V1_A,  04:53:00,  04:53:00,  BELMONT,  6
BNSF_BN1200_V1_A,  04:57:00,  04:57:00,  MAINST-DG,  7
BNSF_BN1200_V1_A,  04:59:00,  04:59:00,  FAIRVIEWDG,  8
BNSF_BN1200_V1_A,  05:01:00,  05:01:00,  WESTMONT,  9
BNSF_BN1200_V1_A,  05:04:00,  05:04:00,  CLARNDNHIL,  10
BNSF_BN1200_V1_A,  05:07:00,  05:07:00,  HINSDALE,  12
BNSF_BN1200_V1_A,  05:10:00,  05:10:00,  WESTSPRING,  14
BNSF_BN1200_V1_A,  05:14:00,  05:14:00,  LAGRANGE,  16
BNSF_BN1200_V1_A,  05:32:00,  05:32:00,  CUS,  28
```

Abbildung 6: Auszug aus *stop_times.txt*

```
stop_id, stop_name, stop_desc, stop_lat, stop_lon, zone_id
AURORA, Aurora, , 41.7608333, -88.3083333, H
ROUTE59, Route 59, , 41.7777778, -88.2086111, G
NAPERVILLE, Naperville, , 41.7797222, -88.1455556, F
LISLE, Lisle, , 41.7977778, -88.0719444, E
BELMONT, Belmont, , 41.7952778, -88.0380556, E
MAINST-DG, Downers Grove, , 41.7952778, -88.0097222, E
FAIRVIEWDG, Fairview Ave., , 41.7952778, -87.9936111, E
WESTMONT, Westmont, , 41.7955556, -87.9763889, D
CLARNDNHIL, Clarendon Hills, , 41.7969444, -87.9536111, D
WHINSDALE, West Hinsdale, , 41.7988889, -87.9452778, D
```

Abbildung 7: Auszug aus *stops.txt*

3.4 Datenstruktur zum Speichern eines GTFS Feeds

Unsere Datenstruktur richtet sich nach dem Konzept der objektorientierten Programmierung. Jede Zeile einer CSV Datei entspricht einem Objekt in der Struktur. Ein Objekt dient dabei als ein Behälter, der die Informationen aus den Feldern der CSV Datei speichert. In Abbildung 8 ist eine grafische Darstellung der Datenstruktur zu sehen. Folgende Objekte sind in der Struktur enthalten:

- **Stop:** Repräsentiert eine Haltestelle in *stops.txt*. Enthält eine Referenz auf eine Station (ein *Stop*-Objekt) falls es ein Teil der Station ist.
- **StopTime:** Repräsentiert eine Haltestelle mit An- und Abfahrtszeiten. Enthält eine Referenz auf ein *Stop*-Objekt.
- **Service:** Repräsentiert eine Servicezeit. Der Wochenplan ist als Boolean Array dargestellt, jeder Wochentag ist ein Index (Index 0: Montag, Index 1: Dienstag,...). Ist der Wert **true** so ist der Trip an dem Tag verfügbar. Start- und Enddatum des Wochenplans sind als String-Objekte angegeben. Jedes Service-Objekt enthält eine Hash-Map, in der angegeben ist an welchen Tagen sich die Verfügbarkeit des Dienstes vom regulärem Plan unterscheidet. Der Schlüssel der Hash-Map ist dabei das Datum. Der Wert gibt an, ob der Dienst an diesem Datum verfügbar ist.
- **Trip:** Repräsentiert einen Trip. Ein Trip enthält eine Liste von StopTime-Objekten. Diese Liste stellt die von dem Trip angefahrenen Haltestellen dar und ist nach der *stop_sequence* sortiert. Zudem wird ein Service-Objekt referenziert, das die Gültigkeitsdauer des Trips angibt.
- **Route:** Repräsentiert eine Route. Enthält eine Hash-Map mit allen Trips auf der Route. Der Schlüssel ist die *trip_id*. Der Wert ist das Trip-Objekt.
- **Agency:** Repräsentiert eine Agentur.
- **Feed:** Repräsentiert einen GTFS Feed. Enthält mehrere Hash-Maps für die unterschiedlichen Objekte. Der Schlüssel ist dabei die *id* des entsprechenden Objektes.
- **Frequency:** Repräsentiert einen Eintrag in *frequencies.txt*. Gibt einen Zeitraum (*startTime* und *endTime*) an, in der ein Trip in festen Zeitabständen (*headway*) startet. Beispielsweise sei die *startTime*=10:00:00, *endTime*=10:10:00 und

headway = 180. Dann würden zwischen 10:00 Uhr und 10:10 Uhr alle 3 Minuten ein neuer Trip starten. Insgesamt sind es 4 Fahrten.

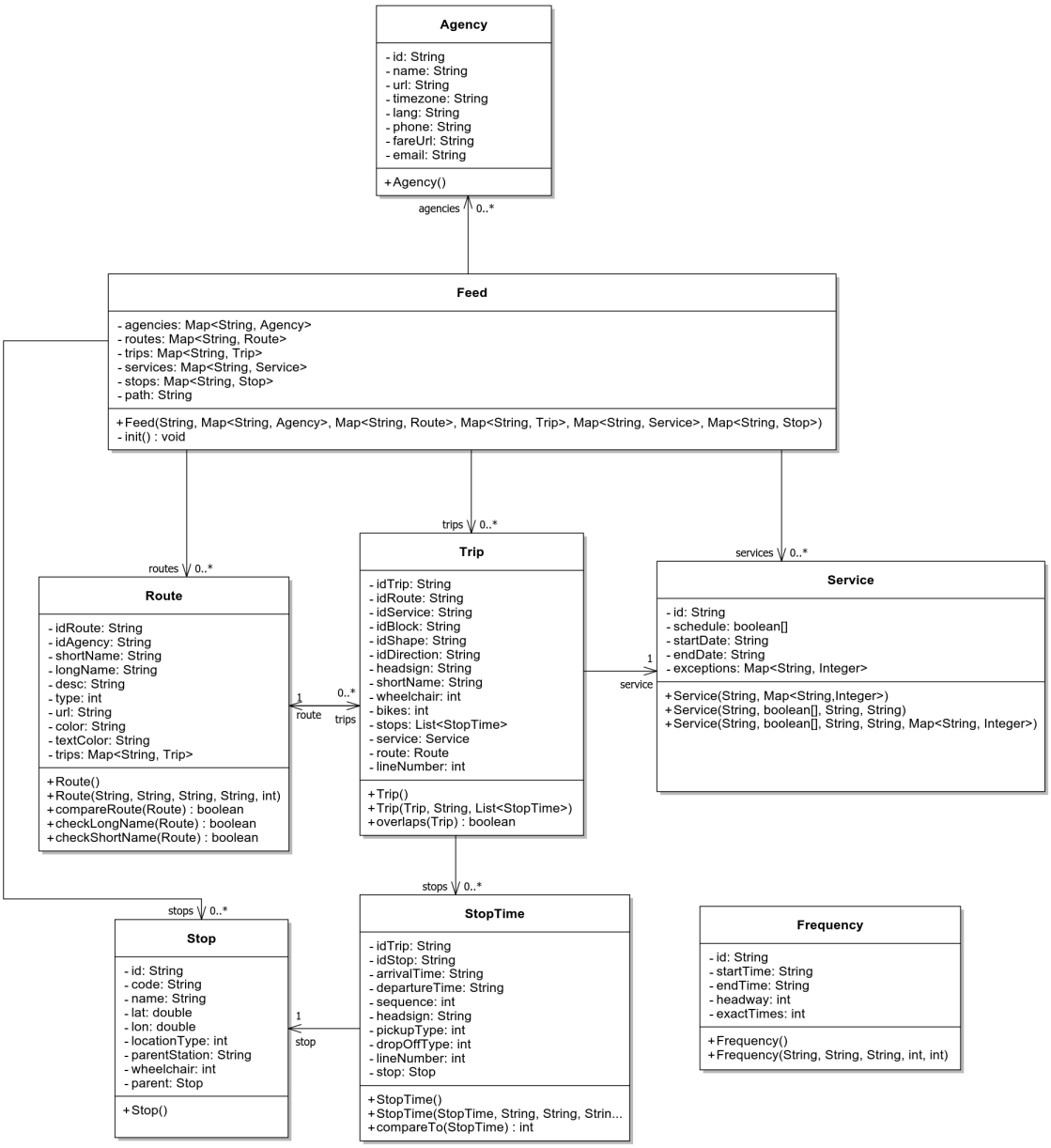


Abbildung 8: Klassendiagramm für die Datenstruktur eines Feeds.

3.4.1 Speichermanagement

Ein Großteil der Informationen in unserer Datenstruktur besteht aus String-Objekten. Unter diesen Objekten gibt es viele Doppelungen, z.B. die *trip_id* ist sowohl in *trips.txt* als auch in *stop_times.txt* enthalten. Beim Einlesen der Datei wird für jeder dieser Angaben ein neues String-Objekt erstellt, was schnell den Speicher füllen kann.

Um selbst große Feeds im Speicher lagern zu können, wird die *java.lang.String.intern()* Methode verwendet. Bei diesem Methodenaufruf wird zunächst in einem Pool von Strings kontrolliert ob dieser bereits vorhanden ist. Ist der String im Pool enthalten wird eine Referenz des Strings zurückgegeben. Ist der String nicht im Pool enthalten, wird er hinzugefügt und eine Referenz des Strings wird zurückgegeben [7].

In Tabelle 1 sind die Unterschiede beim Einlesen einer *stop_times.txt* Datei mit ca. 4,6 Millionen Zeilen zu sehen. Die Anzahl der String Instanzen konnte um ein vielfaches reduziert werden und der Speicherbedarf konnte von ca. 1,6 GB auf ca. 474 MB verringert werden.

		mit intern()	ohne intern()
String	Instanzen	454 332	18 673 234
	Größe (Bytes)	12 721 296	522 850 552
StopTime	Instanzen	4 666 384	4 666 384
	Größe (Bytes)	317 314 112	335 979 648
Bytes insgesamt		474 288 581	1 677 665 456
Instanzen insgesamt		5 597 533	42 035 223

Tabelle 1: Vergleichstabelle für *java.lang.String.intern()*.

3.5 Algorithmen

3.5.1 Longest Common Subsequence

Der Algorithmus des Longest Common Subsequence (LCS) wird dazu eingesetzt um die längste Teilfolge von zwei oder mehreren Sequenzen zu finden.
Sind zwei Folgen $A = (a_1a_2...a_m)$ und $B = (b_1b_2...b_n)$ gegeben, so gibt es eine Folge $C = c_1c_2...c_p$ die eine Teilfolge von A und B ist, sodass p maximal ist. Dazu werden die letzten Elemente der aktuellen Folgen $a_1a_2...a_i$ und $b_1b_2...b_j$ untersucht. Ist $a_i = b_j$ dann ist $c_p = a_i$ und es wird nur noch nach der restlichen Teilfolge $c_1c_2, ...c_{p-1}$ gesucht. Der LCS lässt sich dann durch $\text{LCS}(i, j) = LCS(i-1, j-1)+1$ berechnen. Ist $a_i \neq b_j$ dann ist die $\text{LCS}(i, j)$ die längere der Folgen $\text{LCS}(i-1, j)$ und $\text{LCS}(i, j-1)$. [8]

Bei der Implementierung wird eine $m \times n$ Matrix D erstellt, welche die Längen der Teilfolge C speichert. Ein Eintrag an der Zelle $d_{i,j}$ gibt dabei die Länge der Teilfolge $c_1c_2...c_k$ von $a_1a_2...a_i$ und $b_1b_2...b_j$ an. Der Inhalt der Matrix wird durch folgende Formel bestimmt:

$$d_{i,j} = \begin{cases} 0 & i = 0 \text{ oder } j = 0 \\ d_{i-1,j-1} + 1 & a_i = b_j \\ \max(d_{i-1,j}, d_{i,j-1}) & a_i \neq b_j \end{cases} \quad (1)$$

Die maximale Länge der Teilfolge befindet sich in der Zelle $d_{m,n}$ der Matrix. In den Zellen $d_{i,j}$ befindet sich die Länge der Teilfolge $c_1c_2...c_k$ von $a_1a_2...a_i$ und $b_1b_2...b_j$. Der Algorithmus hat eine Laufzeit von $O(mn)$, da wir eine $m \times n$ Matrix befüllen müssen. Wir verwenden diesen Algorithmus für die Erkennung von gemeinsamen Haltestellen zweier Trips (siehe Abschnitt 4.2.3).

Abbildung 9: Matrix für die Berechnung der LCS von den Folgen ABCD und AEBDA. Die rote Umrandung zeigt den Traceback an um die Zeichen der Teilfolge zu erhalten.

3.5.2 Levenshtein-Distanz

Die Levenshtein-Distanz gibt die minimale Anzahl der Operationen an (Löschen, Einfügen oder Ersetzen eines Zeichens), die notwendig sind um eine Zeichenkette in eine andere umzuwandeln.

Bei der Implementierung verwenden wir eine $m \times n$ Matrix D. Gegeben sind zwei Zeichenketten $A = a_1 a_2 ... a_{m-1}$ und $B = b_1 b_2 ... b_{n-1}$. Der Inhalt der Matrix wird durch folgende Formel bestimmt:

$$d_{i,j} = \begin{cases} |i| & j = 0 \\ |j| & i = 0 \\ d_{i-1,j-1} & a_i = b_j \\ \min \text{ von } \begin{cases} d_{i,j-1} + 1 \text{ (Löschen)} \\ d_{i-1,j} + 1 \text{ (Hinzufügen)} \\ d_{i-1,j-1} + 1 \text{ (Ersetzen)} \end{cases} & a_i \neq b_j \end{cases} \tag{2}$$

Die Zellen $m_{0,j}$ $(0 \leq j \leq n)$ werden mit j gefüllt. Dies stellt die Umwandlung einer leeren Zeichenkette in die Zeichenkette B dar, es müssen n Zeichen hinzugefügt werden. Analog werden die Zellen $m_{i,0}$ gefüllt. Dort wird die Zeichenkette A in eine

leere Zeichenkette umgewandelt indem man m Zeichen löscht [9].

Die minimale Anzahl an Operationen die notwendig sind befinden sich in der Zelle $D_{m,n}$ der Matrix. In den Zellen $M_{i,j}$ befindet sich die Anzahl der Operationen um die Zeichenfolge $a_1a_2...a_i$ in die Zeichenfolge $b_1b_2...b_j$ umzuwandeln. Die Laufzeit des Algorithmus beträgt $O(mn)$. Wir verwenden den Algorithmus um kleine Abweichungen bei den Namen der Routen (Abschnitt 4.2.1) und Haltestellen (Abschnitt 4.2.3) festzustellen.

	Ø	T	i	e	r
Ø	0	1	2	3	4
T	1	0	1	2	3
o	2	1	1	2	3
r	3	2	2	2	2

Abbildung 10: Matrix für die Berechnung der Levenshtein-Distanz für Tor und Tier.

3.5.3 Vincenty Formel

Vincenty's Formel besteht aus zwei iterativen Methoden (direkt und invers) zur Distanzberechnung von zwei Punkten auf einem Ellipsoid. Die direkte Methode berechnet den Zielpunkt von einem Startpunkt bei gegebener Richtung und Distanz. Die inverse Methode berechnet die Distanz zwischen zwei Punkten. Für diese Thesis ist die inverse Methode von Relevanz und wird im folgenden kurz beschrieben.

Zunächst wird die Längengradunterschied λ auf der Hilfskugel approximiert. $\lambda = L$ ist die erste Approximation. Dazu werden die Formeln (3) bis (9) iterativ wiederholt bis λ konvergiert und die Änderung in λ vernachlässigbar ist [10].

Anschließend kann die Distanz s zwischen den zwei Punkten mit der Formel (14) berechnet werden. Die berechnete Distanz ist bis auf 0,5 mm genau.

$$\sin^2\sigma = (\cos U_2 \sin \lambda)^2 + (\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos \sigma)^2 \tag{3}$$

$$\cos \sigma = \sin U_1 \sin U_2 + \cos U_1 \cos U_2 \cos \lambda \tag{4}$$

$$\tan\,\sigma = \frac{\sin\,\sigma}{\cos\,\sigma} \tag{5}$$

$$\sin\,\alpha = \cos\,U_1\,\cos\,U_2 \frac{\sin\,\lambda}{\sin\,\sigma} \tag{6}$$

$$\cos(2\sigma_m) = \cos\,\sigma - 2\sin\,U_1 \frac{\sin\,U_2}{\cos^2\alpha} \tag{7}$$

$$C = \frac{f}{16}\cos^2\alpha\Big[4 + f(4 - 3\cos^2\alpha)\Big] \tag{8}$$

$$\lambda = L + (1-C)f\sin\,\alpha\Bigg(\sigma + C\sin\,\sigma\Big[\cos\,(2\sigma_m) + C\cos\,\sigma\Big(-1 + 2\cos^2(2\sigma_m)\Big)\Big]\Bigg), \tag{9}$$

wobei f die Abflachung $f = (a-b)/a$, α der Azimut, σ der Winkelabstand zweier Punkte auf der Hilfssphäre, U_1, U_2 der reduzierter Breitengrad, L der Längengradunterschied und λ der Längengradunterschied auf der Hilfssphäre ist.

$$u^2 = \cos^2\alpha \frac{(a^2 - b^2)}{b^2} \tag{10}$$

$$A = 1 + \frac{u^2}{16384}\Bigg(4096 + u^2\Big[-768 + u^2(320 - 175u^2)\Big]\Bigg) \tag{11}$$

$$B = \frac{u^2}{1024}\Bigg(256 + u^2\Big[-128 + u^2(74 - 47u^2)\Big]\Bigg) \tag{12}$$

$$\begin{aligned}\Delta\sigma = B\sin\,\sigma\Bigg(&\cos(2\sigma_m) + \frac{1}{4}B\Big[\cos\Big(-1 + 2\cos^2(2\sigma_m)\Big)\\ &-\frac{1}{6}B\cos(2\sigma_m)\Big(-3 + 4\sin^2\sigma\Big)\Big(-3 + 4\cos^2(2\sigma_m)\Big)\Big]\Bigg)\end{aligned} \tag{13}$$

$$s = bA(\sigma - \Delta\sigma), \tag{14}$$

wobei a, b die große und kleine Halbachse des Ellipsoid sind und σ_m der Winkelab-

stand auf der Hilfssphäre vom Equator zum Mittelpunkt der Linie ist.

Wir verwenden Vincenty's Formel um die geographischen Positionen von zwei Haltestellen zu vergleichen (siehe Abschnitt 4.2.3). Eine Alternative zu Vincenty's Formel ist die Haversine Formel. Diese ist schneller zu berechnen, dafür aber nicht so präzise wie Vincenty's Formel. Haversine's Formel geht davon aus dass die Erde eine perfekte Kugel ist. Vincenty's Formel hingegen berücksichtigt, dass die Erde um den Äquator breiter ist.

3.6 Externe Bibliotheken

- **Apache Commons:**
  Apache Commons ist ein Projekt der Apache Software Foundation, die sich darauf fokusiert wiederverwendbare Java Komponenten zu erstellen [11]. Darunter ist eine Bibiothek enthalten welche eine Methode für die Berechnung der Levenshtein-Distanz bereitstellt.

- **uniVOcity-parsers:**
  Die uniVocity-parsers Bibiothek verwenden wir als Grundlage für das Einlesen eines Feeds. Sie stellt Komponenten zur Verfügung mit denen es möglich ist, den eingelesenen Inhalt direkt in Objekte umzuwandeln [12].

- **Java Geodesy Library for GPS – Vincenty's Formulae:**
  Diese Bibiothek implementiert den Algorithmus von Vincenty zur Berechnung von zwei Punkten auf der Erde. Diese Implementierung enthält unterschiedliche Ellipsoid Modelle, unter anderem das für uns interessante WGS84 Model [13].

4 Aufbau und Implementierung

In diesem Kapitel gehen wir genauer auf den Aufbau und die Implementierung unseres Programms ein. Das Programm besteht im wesentlichen aus zwei Teilen. Zunächst werden die Feeds eingelesen und für den sematischen Vergleich vorbereitet. Anschließend werden die Trips der Feeds miteinader verglichen.

4.1 Einlesen der GTFS Feeds

Unsere *FeedParser* Klasse erlaubt es einen GTFS-Feed als ZIP-Archiv oder Ordner einzulesen. Es werden zunächst die Dateien *agency.txt*, *stops.txt*, *routes.txt*, *calendar.txt* und *calendar_ dates.txt* eingelesen. Alle Objekte befinden sich in einer für die Klasse entsprechenden Hash-Map und können mit der ID des jeweiligen Objekts abgerufen werden. Im Anschluss wird die *stop_ times.txt* Datei eingelesen. StopTime-Objekte mit der gleichen *trip_ id* werden in eine Liste aufsteigend nach der *stop_ sequence* sortiert und stellen die Strecke eines Trips dar. Es wird zudem die Referenz auf das Stop-Objekt mit der angegebenen *stop_ id* gesetzt. Die Listen werden dann anschließend in eine Hash-Map untergebracht bei dem die *trip_ id* als Schlüssel dient.

Im nächsten Schritt werden die Trip-Objekte erstellt. Die Route, der Service und die StopTime Liste werden dem Trip-Objekt zugeordnet. Wurden alle Trips erstellt, so wird anschließend kontrolliert ob der Feed eine *frequencies.txt* Datei besitzt. Ist dies der Fall, werden Frequency-Objekte erstellt und weitere Trips zur Hash-Map hinzugefügt. Ist ein Trip in *frequencies.txt* aufgeführt, werden die absoluten Zeitwerte für An- und Abfahrt in den dazugehörigen StopTime-Objekten ignoriert. Stattdessen definieren sie die zeitliche Differenz zwischen jeder Haltestelle [14].

Dazu wird die StopTime-Liste des Trips kopiert und die An- und Abfahrtszeiten der StopTime-Elemente werden angepasst. Wir verwenden dafür eine verschachtelte for-Schleife. Zunächst werden *startTime* und *endTime* vom Frequency-Objekt in Sekunden nach Mitternacht umgewandelt (z.B. 03:00:00 wird zu 10 800). In der äußeren Schleife initialisieren wir unsere Zählvariable i mit der *startTime*. Die Schleife wird solange wiederholt wie $i < endTime$ ist. Nach jeder Iteration wird i um *headway* erhöht.

Zunächst wird das erste StopTime-Objekt angepasst, dabei wird die *arrival_time* und *departure_time* gleich *i* gesetzt. Anschließend durchlaufen wir die StopTime-Liste und berechnen die Zeiten für die restlichen Elemente. Dazu wird der zeitliche Abstand zur vorherigen Haltestelle aus der originalen Liste berechnet. Die neuen Zeiten eines StopTime-Objekts sind dann die neu berechneten Zeiten des vorherigen Objekts addiert mit dem zeitlichen Abstand. Nachdem alle StopTime-Elemente angepasst wurden, wird ein neuer Trip erstellt und zur Hash-Map hinzugefügt. Hinzugefügte Trips erhalten eine neue *trip_id*, die sich aus der alten *trip_id* und dem Zusatz *FREQi* zusammensetzt.

Wurden alle Trips erfolgreich eingelesen, werden anschließend noch Trip-Duplikate entfernt und Trips den entsprechenden Routen zugewiesen. Duplikate sind Trips, die sich nur anhand der *trip_id* unterscheiden. Im Anschluss wird überprüft ob es Duplikate bei den Routen selbst gibt. Zwei Routen sind dabei gleich wenn sie die selben Werte haben, mit Ausnahme der *route_id* und den Trips. Gibt es Duplikate, so werden alle Trips einer einzigen Route zugeordnet und die *route_id* und die Referenz auf das Route-Objekt angepasst. Anschließend werden die alle doppelten Routen entfernt, sodass nur noch eine Route mit sämtlichen Trips vorhanden ist. Zum Schluss gibt es noch eine kurze Ausgabe zu der Anzahl relevanten Objekte im Feed (siehe Listing 4.1).

```
1  Feed: F:\Downloads\GTFS\freiburg\gtfs.zip
2  Stops: 713
3  StopTimes: 234533
4  Services: 263
5  Routes: 24
6  Trips: 18847
7  ----------------------------------------
8  Feed: F:\Downloads\GTFS\freiburg\gtfs -freq-.zip
9  Stops: 698
10 StopTimes: 234533
11 Services: 261
12 Routes: 24
13 Trips: 18847
14 ----------------------------------------
```

Listing 4.1: Ausgabe der *FeedParser* Klasse.

4.2 Vergleich der GTFS Feeds

Der Vergleich der Feeds ist modular aufgebaut. Zunächst werden die Routen überprüft. Wurde eine Übereinstimmung gefunden, werden anschließend die Trips beider Routen verglichen. Die Überprüfung der Trips ist wiederum in zwei Teiluntersuchungen unterteilt, den Vergleich der Haltestellen und der Servicezeiten.

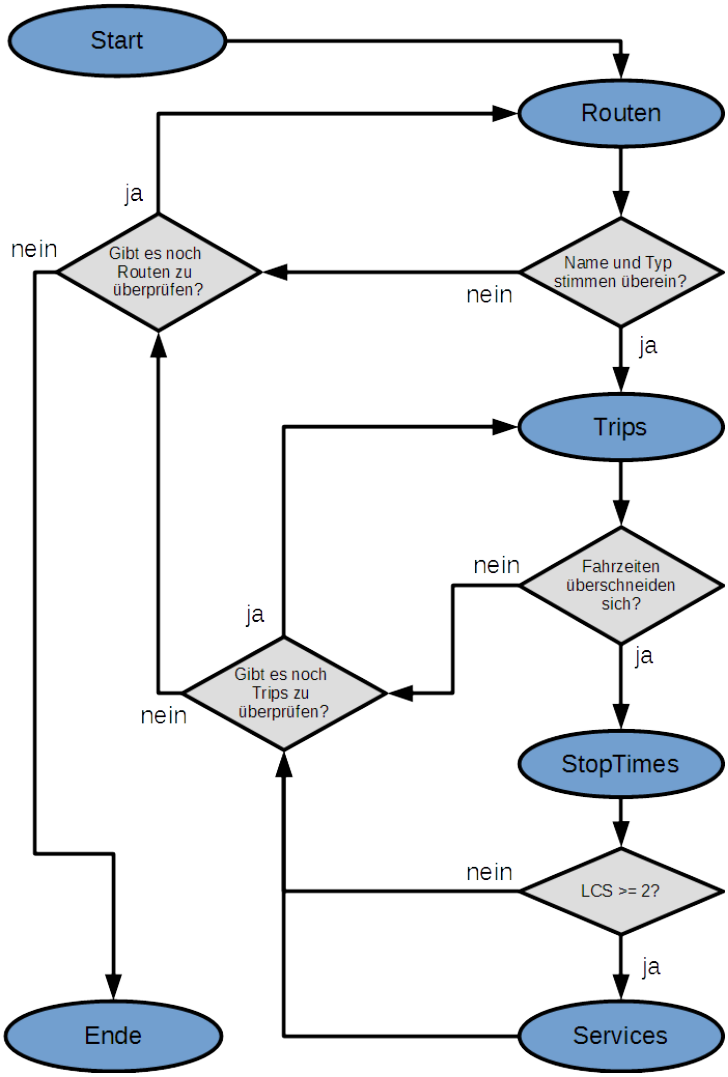


Abbildung 11: Grober Ablauf der Vergleiche

4.2.1 Vergleich von Routen

Es wird durch die Routen beider Feeds iteriert und verglichen ob der *route_type*, *route_short_name* und *route_long_name* übereinstimmen. Bei dem Vergleich der Namen wird die Levenshtein-Distanz verwendet um kleine Abweichungen zu berücksichtigen. Befindet sich die Anzahl der Operationen unter 25% der maximal möglichen Operationen, so werden die Namen als gleichwertig angesehen. Wurde eine Übereinstimmung gefunden so werden die Trips der Routen miteinander verglichen. Wurden alle Routen verarbeitet, so verbleiben nur noch Routen die jeweils im anderen Feed nicht vorkommen.

4.2.2 Vergleich von Trips

Der Vergleich von Trips ist in zwei voneinander unabhängige Vergleiche unterteilt. Es werden zunächst die Haltestellen der Trips verglichen. Gab es eine Übereinstimmung, so werden die Servicezeiten überprüft.

Ein Trip aus Feed A kann maximal einem Trip aus Feed B zugeordnet werden. Um die Suche zu beschleunigen werden die Trips vom Feed B umsortiert. Wir erstellen dafür eine neue Hash-Map. Der Schlüssel ist ein StopTime-Objekt und der Wert ist eine Liste von Trips, die das StopTime-Objekt referenzieren.

Nun können wir einen Trip aus Feed A nehmen und durch seine StopTime-Objekte iterieren. Befindet sich ein StopTime-Objekt in der Hash-Map, fügen wir die Trips in eine Liste für mögliche Übereinstimmungen ein.

Anschließend können wir durch diese Liste iterieren und die Trips vergleichen. Durch diese Methode können wir die Anzahl der zu untersuchenden Trips verringen, sodass nur die Trips verglichen werden die mindestens ein StopTime-Objekt gemeinsam haben.

Die Ergebnisse werden in einem *TripData*-Objekt gespeichert. Dieses Objekt enthält den Übereinstimmungstyp, die *trip_id's* beider Trips, zwei Listen für fehlende StopTime-Objekte der Trips, eine Hash-Map die Tage enthält an denen die Verfügbarkeit der Trips verschieden sind und eine prozentuale Angabe für die Übereinstimmung der StopTime-Objekte. Es gibt vier unterschiedliche Übereinstimmungstypen:

1. **FULL_MATCH:** Volle Übreinstimmung. Identische Liste von StopTime-Objekten und Servicezeiten.

2. **FULL_STOP_MATCH:** Liste von StopTime-Objekten ist identisch. Servicezeiten haben Unterschiede.

3. **MATCH:** Liste von StopTime-Objekten und Servicezeiten stimmen teilweise überein.

4. **NO_MATCH:** Keine Übereinstimmung bei den Listen von StopTime-Objekten. Servicezeiten werden nicht überprüft.

Stimmen Trips überein, wird das *TripData*-Objekt in eine Prioritätswarteschlange eingefügt. Wurden alle Trips von Feed A untersucht, wird die Prioritätswarteschlange verarbeitet. Um eine konsistente Ausgabe zu erhalten werden die *TripData*-Objekte nach den folgenden Kriterien sortiert:

1. Übereinstimmungstyp: Priorität nach der Reihenfolge wie oben beschrieben.

2. prozentualle Übereinstimmung der StopTime-Objekte: *TripData*-Objekte mit höherer Übereinstimmung haben Vorrang.

3. Anzahl der Tage an denen die Verfügbarkeit des Trips unterscheidet: *TripData*-Objekte mit niedrigerer Anzahl haben Vorrang.

4. ID's der Trips: *TripData*-Objekte werden alphabetisch nach den ID's sortiert falls alle vorherigen Kriterien übereinstimmen.

TripData-Objekte die bereits verarbeitete Trips enthalten werden ignoriert. Wurden alle Elemente aus der Prioritätswarteschlange verarbeitet verbleiben nur noch Trips für die keine Übereinstimmung gefunden wurde.

4.2.3 Vergleich von Haltestellen

Eine Übereinstimmung der Haltestellen liegt vor, wenn mindestens zwei StopTime-Objekte übereinstimmen. Zwei StopTime-Objekte stimmen überein wenn ihre An- und Abfahrtszeit identisch sind (*arrival_time*, *departure_time*), Stop-Objekte und einige optionale Felder, welche einen Defaultwert haben (*pickup_type*, *drop_off_type*), übereinstimmen. Ein Stop-Objekt stimmt mit einem anderen überein falls die Namen (*stop_name*) gleich sind bzw. die Levenshtein-Distanz unter 25% der maximalen Anzahl an Operationen liegt. Ein weiteres Kriterium ist die geographische Position der Haltestelle (*stop_lat*, *stop_lon*). Die Positionen dürfen maximal 5 Meter voneinander entfernt sein, um noch als gleiche Haltestelle angesehen zu werden. Die Distanz wird

mit Hilfe von Vincenty's Formel berechnet. Ist eine Haltestelle Teil einer Station müssen die Stationen ebenfalls übereinstimmen. Eine Station ist in diesem Fall ein weiteres Stop-Objekt.

Um die Anzahl der gleichen StopTime-Objekte zu bestimmen verwenden wir den Algorithmus des Longest Common Subsequence (LCS). Durch Backtracking erhalten wir die StopTime-Objekte der Teilfolge. Anschließend werden zwei Listen für StopTime-Objekte erstellt, welche jeweils nur in einem Feed vorkommen. Es wird zudem noch die prozentualle Übereinstimmung der StopTime-Objekte nach folgender Formel berechnet:

$$p = \frac{k}{M} * 100, \tag{15}$$

wobei p die prozentualle Übereinstimmungist, k die Länge der LCS und M die kleinere Größe der beiden StopTime-Listen.

4.2.4 Vergleich von Servicezeiten

Es werden zunächst die Ausnahmen beider Servicezeiten verglichen. Unterscheidet sich die Verfügbarkeit des Trips an einem Tag, so wird ein Eintrag in eine Hash-Map gemacht. Das Datum ist dabei der Schlüssel und der Wert ist ein „-“ String (Trip ist nur in Service A verfügbar) oder ein „+“ String (Trip ist nur in Service B verfügbar). Ist ein Datum nicht in beiden Ausnahmen enthalten, wird in den regulären Servicezeiten nach dem Datum gesucht. Befindet sich das Datum innerhalb der Start- und Endzeiten eines Service wird der Wochentag bestimmt. Es wird dann überprüft ob die Verfügbarkeit des Trips an dem Wochentag mit dem Eintrag im *schedule* Array des Service-Objekts übereinstimmt. Liegt das Datum außerhalb der Start- und Endzeiten eines Service, so wird das Datum nur in die Hash-Map eingetragen, falls der Service an dem Tage verfügbar ist. Eine graphische Darstelleung ist in Abbildung 12 zu sehen.

Wurden alle Ausnahmen bearbeitet, werden anschließend die regulären Servicezeiten untersucht. Beginnen bzw. enden die Servicezeiten an unterschiedlichen Tagen, so werden die Tage des Service der früher beginnt bzw. später endet der Hash-Map hinzugefügt, falls der Trip an dem Tag verfügbar ist und nicht in den Ausnahmen steht. Zu guter Letzt wird der Zeitraum kontrolliert an dem beide Services aktiv sind. Dazu wird durch die *schedule* Arrays beider Services iteriert. Unterscheiden sich die boolean-Werte eines Eintrags (Wochentag), werden alle Wochentage innerhalb der

Start- und Endzeiten berechnet und der Hash-Map hinzugefügt, sofern die Tage nicht in den Ausnahmen beider Services enthalten sind.

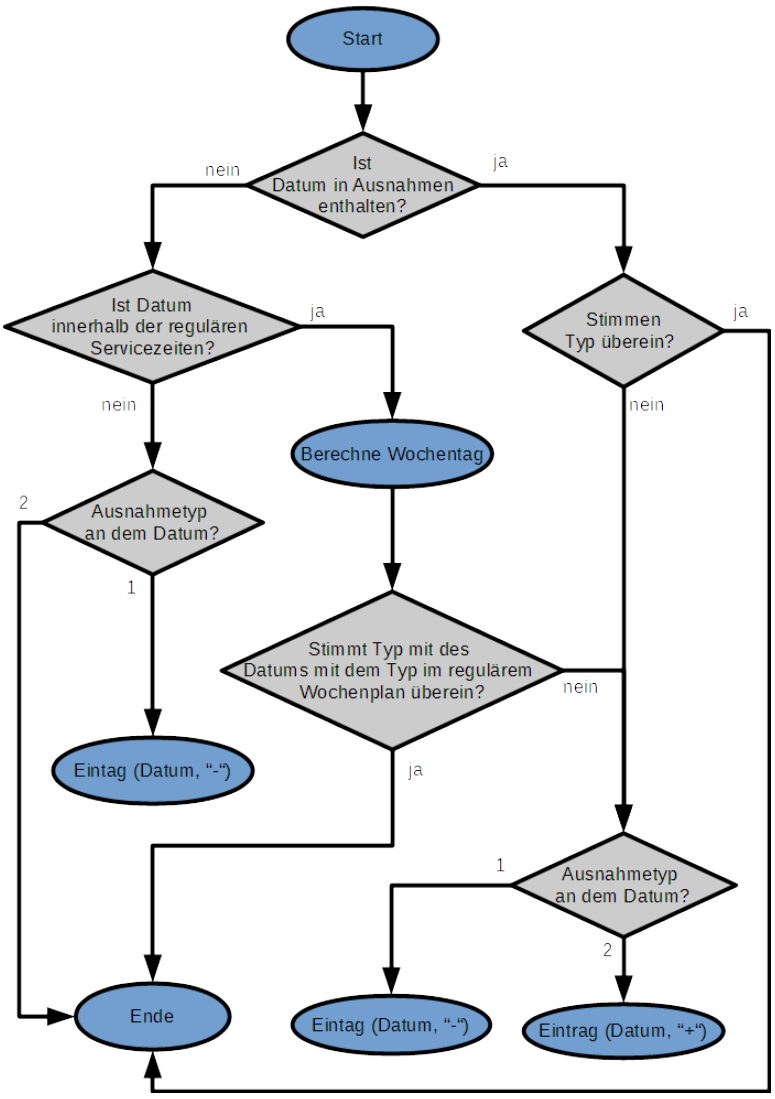


Abbildung 12: Ablauf für den Vergleich der Ausnahmen von Service A mit Service B. Vergleicht man die Service B mit Service A, müssen die „-“ und „+“ Strings vertauscht werden.

4.3 Ausgabemodi

Das Tool hat zwei unterschiedliche Ausgabemodi, eine aggregierte Ausgabe (Listing 4.2) und eine vollständige Ausgabe (Listing 4.3). In der aggregierten Ausgabe wird die Menge an identischen, teilweise übereinstimmenden und fehlenden Trips ausgeben. Zusätzlich wird die Anzahl der fehlenden Haltestellen (StopTime) und Routen ausgegeben. Bei der vollständigen Ausgabe werden hingegen mehr Informationen (*trip_id*, Name der Route, erste und letzte Haltestelle etc.) zu dem jeweiligen Trip ausgegeben. Dies beinhaltet eine Pfadangabe zur Datei inklusive Zeilenangabe wo man den Trip finden kann. Informationen zu fehlenden Haltestellen werden hier ebenfalls ausgegeben.

In den Zeilen 1 bis 6 in Listing 4.3 ist beispielsweise eine Teilübereinstimmung zu erkennen. Zeile 1 gibt an, dass sich die *trip_id's* unterscheiden. In Zeile 2 und 3 werden zwei Haltestellen angegeben die nur im ersten Feed (Feed A) vorkommen. In Zeile 4, 5 und 6 sind die Haltestellen angegebn die nur im zweiten Feed (Feed B) enthalten sind angegeben.

```
1 9 trip(s) stayed the same
2 1 trip(s) with same stop(s), 2 different service date(s)
3 2 partial trip match(es), 4 stop(s) removed, 3 stop(s) added, 2 different date(s)
4 4 trip(s) removed, 1 trip(s) added, 26 stop(s) removed, 4 stop(s) added
5 6 route(s) stayed the same, 0 route(s) removed, 0 route(s) added
```

Listing 4.2: Aggregierte Ausgabe von einem Vergleich zweier Feeds.

```
1  ~ trip; id_a: 39744A361B1981A; id_b: 39744A361B1981B
2  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedOne.zip\stop_times.txt:142: -stop; on trip: 39744A361B1981A
       ; id: 19016; name: Sutter Hill Transit Center, arr_time: 09:05:00, dep_time: 09:05:00
3  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedOne.zip\stop_times.txt:143: -stop; on trip: 39744A361B1981A
       ; id: 20474; name: Sutter Creek Auditorium, arr_time: 09:08:00, dep_time: 09:08:00
4  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedTwo.zip\stop_times.txt:143: +stop; on trip: 39744A361B1981B
       ; id: 19014; name: Safeway, arr_time: 09:20:00, dep_time: 09:20:00
5  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedTwo.zip\stop_times.txt:151: +stop; on trip: 39744A361B1981B
       ; id: 18993; name: Petkovich Park, arr_time: 09:47:00, dep_time: 09:47:00
6  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedTwo.zip\stop_times.txt:152: +stop; on trip: 39744A361B1981B
       ; id: 18994; name: 150 Main St., arr_time: 09:48:00, dep_time: 09:48:00
7  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedOne.zip\trips.txt:13: -trip; id: AAA-FREQ5; short_name: ;
       route_name: Sacramento Express, first_station: J St. & 3rd St., last_station: Murieta Dr. & Lone
       Pine Dr., start: 05:50:00, end: 06:20:00
8  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedOne.zip\trips.txt:13: -trip; id: AAA-FREQ4; short_name: ;
       route_name: Sacramento Express, first_station: J St. & 3rd St., last_station: Murieta Dr. & Lone
       Pine Dr., start: 05:40:00, end: 06:10:00
9  D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedOne.zip\trips.txt:13: -trip; id: AAA-FREQ3; short_name: ;
       route_name: Sacramento Express, first_station: J St. & 3rd St., last_station: Murieta Dr. & Lone
       Pine Dr., start: 05:30:00, end: 06:00:00
10 ~ trip; id_a: DIFFERENT_SERVICE; id_b: DIFFERENT_SERVICE2
11 D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedTwo.zip\trips.txt:4: +trip; on 22.03.2017; id:
       DIFFERENT_SERVICE2; short_name: ; route_name: Upcountry, first_station: Amador Station,
       last_station: Sutter Hill Transit Center, start: 13:10:00, end: 14:24:00
12 D:\Dokumente\Eclipse\DiffToolForGtfs\TestFeedOne.zip\trips.txt:4: -trip; on 24.03.2017; id:
       DIFFERENT_SERVICE; short_name: ; route_name: Upcountry, first_station: Amador Station,
       last_station: Sutter Hill Transit Center, start: 13:10:00, end: 14:24:00
```

Listing 4.3: Ausschnitt für vollständige Ausgabe von einem Vergleich zweier Feeds.

5 Evaluation

Das Programm wurde auf einem Computer mit einem Intel Core i5-4670K @ 3.40 GHz, 8 GB Arbeitsspeicher und Windows 10 64-bit ausgeführt. Um die Ausgabe unseres Tools zu validieren, überprüfen wir zwei semantisch gleichwertige Feeds in Tabelle 2. Einer der Feeds wurde mit dem Tool *gtfstidy* [15] aufgeräumt, d.h. wenn möglich wurden Trips zusammengefasst und in *frequencies.txt* dargestellt oder Services und Stops wurden entfernt, falls sie nicht referenziert wurden. Aus der Tabelle wird ersichtlich, dass beide Feeds trotz Unterschiede bei der Anzahl von Stops und Trips semantisch gleichwertig sind und die gleichen Trips enthalten.

Die Angabe der Laufzeit bezieht sich auf den aggregierten Modus. Im vollständigen Ausgabemodus ist die Laufzeit generell länger, da mehr Daten verarbeitet und ausgegeben werden müssen.

In den Tabellen 3 und 4 vergleichen wir die Feeds von der Metra, ein Nahverkehrsystem von Chicago. Dabei wird untersucht wie sich der Fahrplan von 2015 bis 2017 verändert hat. Änderungen von den Servicezeiten werden im Folgenden nicht explizit aufgeführt aufgrund der zeitlichen Distanz der Feeds.

Von 2015 auf 2016 stieg die Anzahl der Trips um 26,5% von 6972 auf 8817. Die Anzahl der Haltestellen (StopTime) stieg um 23,7% von 121836 auf 150716. 5938 Trips (67,35%) sind unverändert geblieben, an 839 Trips (9,52%) wurden die Haltestellen geändert und 2040 neue Trips (23,14%) kamen hinzu. 195 Trips wurden entfernt.

Von 2016 auf 2017 hingegen sank die Anzahl an Trips wieder um 12,67% auf 7700, die Haltestellen sind um 13,12% auf 130692 gesunken. 79,47% der Trips sind gleichgeblieben, 18,96% der Trips erhielten eine Änderung der Haltestellen und 1,58% der Trips in 2017 sind neu hinzugekommen. 1238 Trips wurden hingegen zum Vorjahr entfernt.

In der Tabelle 5 sind die Ergebnisse eines Vergleichs von zwei größeren Feeds der Empresa Municipal de Transportes de Madrid (EMT Madrid).
In Tabelle 6 vergleichen wir einen von Zhang's Tool [3] generierten Feed mit einem Feed

bestehend aus offiziellen Fahrplandaten der Deutschen Bahn für Südwestdeutschland. Wie zu erwarten gab es keine Übereinstimmungen, da die An- und Abfahrtszeiten nach bestimmten Regeln generiert wurden, beispielsweise beginnt eine Trip immer zur vollen Stunde und ist den Tag verfügbar. Beim Einlesen des Feeds wurden zudem 1608 Trips entfernt, weil diese Trips keine gültige Route besitzen (*route_id* existiert nicht oder nicht zugewiesen). Aufgrund dessen wurden diese Trips nicht beim Vergleich miteinbezogen und sind nicht in der Ergebnistabelle enthalten.

VAG Freiburg

Inhalte der Feeds	original	gtfstidy
Stops	713	698
StopTimes	234 533	234 533
Services	263	261
Routes	24	24
Zusätzliche Trips durch frequencies.txt	0	16 652
Trips	18 847	18 847

Unterschiede der Feeds		original	gtfstidy
Gleiche Routen		24 (100%)	24 (100%)
Gleiche Trips		18 847 (100%)	18 847 (100%)
Gleiche Trips mit unterschiedliche Servicezeiten	Trips	0 (0%)	0 (0%)
Trips mit teilweiser Übereinstimmung	Trips	0 (0%)	0 (0%)
	StopTimes	0 (0%)	0 (0%)
Exclusive Trips		0 (0%)	0 (0%)
Exclusive StopTimes		0 (0%)	0 (0%)
Exclusive Routen		0 (0%)	0 (0%)
Laufzeit (ms)		4029	

Tabelle 2: Ergebnistabelle zur Validierung des Tools

Metra Chicago

Inhalte der Feeds	April 2015	April 2016
Stops	239	239
StopTimes	121 836	150 716
Services	18	29
Routes	11	11
Zusätzliche Trips durch frequencies.txt	0	0
Trips	6972	8817

Unterschiede der Feeds		April 2015	April 2016
Gleiche Trips		0 (0%)	0 (0%)
Gleiche Routen		11 (100%)	11 (100%)
Gleiche Trips mit unterschiedliche Servicezeiten	Trips	5938 (85,2%)	5938 (67,3%)
Trips mit teilweiser Übereinstimmung	Trips	839 (12,0%)	839 (9,5%)
	StopTimes	1136 (0,9%)	1106 (0,7%)
Exclusive Trips		195 (2,8%)	2040 (23,1%)
Exclusive StopTimes		4326 (3,6%)	33 236 (22,05%)
Exclusive Routen		0 (0%)	0 (0%)
Laufzeit (ms)		67 847	

Tabelle 3: Ergebnistabelle für Metra Chicago Feed von April 2015 und April 2016

Metra Chicago

Inhalte der Feeds	April 2016	April 2017
Stops	239	239
StopTimes	150 716	130 962
Services	29	25
Routes	11	11
Zusätzliche Trips durch frequencies.txt	0	0
Trips	8817	7700

Unterschiede der Feeds		April 2016	April 2017
Gleiche Trips		0 (0%)	0 (0%)
Gleiche Routen		11 (100%)	11 (100%)
Gleiche Trips mit unterschiedliche Servicezeiten	Trips	6119 (69,4%)	6119 (79,5%)
Trips mit teilweiser Übereinstimmung	Trips	1460 (16,6%)	1460 (19,0%)
	StopTimes	7018 (4,7%)	6617 (5,1%)
Exclusive Trips		1238 (14,0%)	121 (1,6%)
Exclusive StopTimes		21 273 (14,1%)	1920 (24,9%)
Exclusive Routen		0 (0%)	0 (0%)
Laufzeit (ms)		74 560	

Tabelle 4: Ergebnistabelle für Metra Chicago Feed von April 2016 und April 2017

EMT Madrid

Inhalte der Feeds	März 2016	März 2017
Stops	4649	4686
StopTimes	5 537 600	5 625 726
Services	5	5
Routes	208	214
Zusätzliche Trips durch frequencies.txt	74 818	78 822
Trips	193 254	203 596

Unterschiede der Feeds		März 2016	März 2017
Gleiche Trips		0 (0%)	0 (0%)
Gleiche Routen		204 (98,1%)	204 (95,3%)
Gleiche Trips mit unterschiedliche Servicezeiten	Trips	6011 (3,1%)	6011 (3,0%)
Trips mit teilweiser Übereinstimmung	Trips	137 978 (71,4%)	137 978 (67,8%)
	StopTimes	766 419 (13,8%)	767 905 (13,6%)
Exclusive Trips		49 265 (25,5%)	59 607 (29,3%)
Exclusive StopTimes		1 428 298 (25,8%)	1 514 938 (26,9%)
Exclusive Routen		4 (1,9%)	10 (4,7%)
Laufzeit (ms)		1 450 260	

Tabelle 5: Ergebnistabelle für EMT Madrid Feed von März 2016 und März 2017

Nahverkehr der DB in Südwestdeutschland

Inhalte der Feeds	offiziell	Zhang
Stops	2202	1418
StopTimes	36 668	59 136
Services	2580	1
Routes	2579	132
Zusätzliche Trips durch frequencies.txt	0	0
Trips	2579	7896

Unterschiede der Feeds		offiziell	Zhang
Gleiche Trips		0 (0%)	0 (0%)
Gleiche Routen		2 (0,1%)	2 (1,5%)
Gleiche Trips mit unterschiedliche Servicezeiten	Trips	0 (0%)	0 (0%)
Trips mit teilweiser Übereinstimmung	Trips	0 (0%)	0 (0%)
	StopTimes	0 (0%)	0 (0%)
Exclusive Trips		2579 (100%)	7896 (100%)
Exclusive StopTimes		36668 (100%)	59136 (100%)
Exclusive Routen		2577 (99,9%)	130 (98,5%)
Laufzeit (ms)		1300	

Tabelle 6: Ergebnistabelle für Nahverkehr der DB in Südwestdeutschland

6 Zukünftige Erweiterungen

In diesem Kapitel möchten wir einige Anregungen für mögliche Erweiterungen unserer Arbeit geben. Zum einen könnten die restlichen optionalen Dateien eingelesen und verglichen werden. *shapes.txt* wäre hier wichtigste dieser Dateien. In *shapes.txt* wird der Pfad, den ein Trip nimmt anhand von mehreren geographischen Punkten beschrieben. Eine weitere Ergänzung wäre die Erstellung eines neuen GTFS Feeds, welche die Informationen der verglichenen Feeds vereint.

Ein weiteres Feature könnte ein Vergleich der Feeds auf Abdeckungsähnlichkeit sein. Dabei wird die Anzahl der Fahrten zwischen zwei Stationen gezählt. Dies kann als separates Modul entwickelt werden, da es unabhängig von den Vergleichen der Trips ist. Die Fahrten werden für jeden Feed unabhängig voneinander berechnet und die Ergebnisse werden als Stationspaare mit der Anzahl der Fahrten gespeichert. Nach der Berechnung beider Feeds müssen dann nur noch die gleichen Stationspaare gesucht und verglichen werden. Es bieten sich dabei zwei Herangehensweisen an:
Seien A, B, C Stationen und bilden den Trip $A \to B \to C$.

- Es werden nur benachbarte Stationspaare berechnet. Die Stationspaare sind (A, B) und (B, C).
- Es werden alle Stationspaare entlang eines Trips berücksichtigt. Die Stationspaare sind (A, B), (B, C) und (A, C).

6.1 Vergleich von Pfaden

Hier möchten wir zwei Möglichkeiten vorstellen um zwei Pfade vergleichen. Der erste Algorithmus nennt sich *Dynamic Time Warping* (DTW). Das Maß für die Ähnlichkeit zweier Pfade bzw. Kurven ist die minimale Gesamtabweichung der einzelnen geographischen Punkte. Ein weiteres Maß ist die *Fréchet-Distanz*, welche die maximale Distanz von zwei Punkten entlang der Pfade berechnet.

Die Änderungen am Tool selbst sind mit geringen Aufwand verbunden. Die zusätzli-

che Überprüfung lässt sich nach den Vergleichen der Haltestellen und Servicezeiten einbinden ohne diese zu beinträchtigen. Das *TripData*-Objekt müsste um ein zusätzliches Feld erweitert werden, der die Ergebnisse speichert. Als letzter Schritt müssten die Sortierkriterien für die Prioritätswarteschlange (siehe Abschnitt 4.2.2) angepasst werden.

6.1.1 Dynamic Time Warping

Dieser Algorithmus vergleicht zwei Sequenzen und berechnet anhand einer Kostenfunktion deren Ähnlichkeit. Dazu werden die Kosten der einzelnen Elemente lokal berechnet und das Minimum entnommen. Dabei sind zwei Elemente ähnlich zueinander wenn die Kostenfunktion gering ist. Anschließend wird die Summe der einzelnen minimalen Kosten gebildet.

In unserem Fall haben wir eine Sequenz von geographischen Punkten. Die Kosten sind der Abstand zwischen zwei Punkten, die wir mit Vincenty's Formel berechnen können. Durch den Algorithmus erhalten wir eine minimale Gesamtabweichung in Metern. Liegt diese Abweichung nun unter einem festgelegtem Wert, könnte man die Pfade als identisch betrachten.

Die Implementierung ist mittels dynamischer Programmierung möglich. Gegeben seien die Sequenzen $X = (x_1 x_2 ... x_n)$ und $Y = (y_1 y_2 ... y_m)$. Der Inhalt der Zelle $D_{n,m}$ einer $n \times m$ Matrix D gibt die Summe der Kosten an. Generell gibt $D_{i,j} = \mathrm{DTW}(i, j)$ mit $(1 \leq i \leq n)$ und $(1 \leq j \leq m)$ die Kosten der Sequenzen $x_1 x_2 ... x_i$ und $y_1 y_2 ... y_j$ an. Sei $dist(i, j)$ die Distanz von den Elementen x_i und y_j. Der Inhalt der Matrix wird nach der Formel 16 berechnet.[16]

$$d_{i,j} = \begin{cases} 0 & i = 0 \ \& \ j = 0 \\ \infty & i = 0 \ \& \ (1 \leq j \leq m) \\ \infty & j = 0 \ \& \ (1 \leq i \leq n) \\ dist(x_i, y_j) + \min\{d_{i,j-1}, d_{i-1,j}, d_{i-1,j-1}\} & (1 \leq i \leq n) \ \& \ (1 \leq j \leq m) \end{cases} \tag{16}$$

6.1.2 Fréchet-Distanz

Der Algorithmus berechnet die maximale lokale Distanz von zwei Punkten entlang der Pfade. Informell kann man sich die *Fréchet-Distanz* wie folgt vorstellen. Eine Person

geht mit seinem Hund an einer Hundeleine spazieren. Die Person bewegt sich auf einem Pfad, sein Hund bewegt sich auf einen anderen. Die *Fréchet-Distanz* ist dann die kürzeste Länge der Hundeleine damit beide Pfade abgelaufen werden können [17].

Auch hier ist die Implementierung mittels dynamischer Programmierung möglich und in Formel 17 dargestellt.

$$d_{i,j} = \begin{cases} dist(x_i, x_j) & i = 0 \ \& \ j = 0 \\ max\Big\{d_{i,0}, dist(x_i, y_0)\Big\} & (1 \leq i < n) \\ max\Big\{d_{0,j}, dist(x_0, y_j)\Big\} & (1 \leq j < m) \\ max\Big\{dist(x_i, y_j), min\Big\{d_{i-1,j}, d_{i,j-1}, d_{i-1,j-1}\Big\}\Big\} & (1 \leq i < n) \ \& \ (1 \leq j < m) \end{cases} \tag{17}$$

6.2 Generierung eines neuen Feeds

Um einen neuen Feed zu erstellen, benötigen wir zunächst eine Sammlung aller einzigartigen Trips beider Feeds. Dabei müssen wir beachten, dass keine *id's*, wie z.B. die *trip_ id*, doppelt vorkommen. Als ersten Schritt benötigen wir eine neue Hash-Map in der wir alle Trips speichern. Der Schlüssel ist dabei die *trip_ id*. Dadurch können wir auch einfach kontrollieren ob eine *trip_ id* bereits vorhanden ist. Trips für die kein passender Partner gefunden wurde, können der Hash-Map einfach hinzugefügt werden. Bei Trips mit vollständiger Übereinstimmung reicht es aus einen der beiden Trips zu nehmen. Bei Trips mit teilweiser Übereinstimmung müssen die Servicezeiten und Haltestellen der Trips vereinigt werden. Bei den Servicezeiten wäre eine Möglichkeit sich Service A zu nehmen und die Tage an denen der Trip nur im Service B aktiv ist zu den Ausnahmen von Service A hinzuzufügen.

Bei den Haltestellen gibt es drei Fälle: Die Haltestellen stimmen überein (ABC und ABC), die Haltestellen eines Trips sind eine Teilmenge des anderen (ABCD und ABC) oder es gibt Überschneidungen von Haltestellen (ABC und BCD).

In den ersten beiden Fällen reicht es aus einen Trip zu nehmen und seine Servicezeiten anzupassen. Im ersten Fall spielt es generell keine Rolle welchen von beiden man nimmt. Im zweiten Fall nimmt man den Trip der die Obermenge darstellt.

Im letzen Fall hingegen müssen die Listen der Haltestellen kombiniert werden. Dabei

müssen die *StopTime*-Objekte in eine neue Liste eingefügt und nach der *arrival_time* sortiert werden. Anschließend müssen die *stop_sequence* Werte angepasst werden. Die *trip_id* wird ebenfalls verändert, falls diese sich unterscheiden.

Wurden alle Trips zur Hash-Map hinzugefügt, kann man diese nun iterativ durchlaufen und alle Daten verarbeiten, da alle Objekte über ein Trip-Objekt ereichbar sind wie in Abbildung 8 dargestellt ist. Es wäre ebenfalls sinnvoll den erstellten Feed mittels dem Tool *gtfstidy* aufzuräumen.

Literaturverzeichnis

[1] A. Antrim and S. J. Barbeau, "The Many Uses of GTFS Data - Opening the Door to Transit and Multimodalapplications," tech. rep., Trillium Solutions, Inc., 2013.

[2] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger, "Fast Routing in Very Large Public Transportation Networks using Transfer Patterns," tech. rep., Albert-Ludwigs-Universität Freiburg, Google, Karlsruhe Institute of Technology, 2010.

[3] Z. Zhang, "Public-Transit Data Extraction from OpenStreetMap Data," Bachelorarbeit, 2017.

[4] A. Solymosi and U. Grude, *Grundkurs Algorithmen und Datenstrukturen in JAVA*. Springer Vieweg, 2014. S. 71.

[5] M. Dietzfelbinger, K. Mehlhorn, and P. Sanders, *Algorithmen und Datenstrukturen - Die Grundwerkzeuge*. Springer Vieweg, 2014. S.161 - S.164.

[6] General Transit Feed Specification. `https://developers.google.com/transit/gtfs/reference/`.

[7] Oracle, "Java API Specification." `http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#intern%28%29`.

[8] D. S. Hirschberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences," tech. rep., Princeton University, 1975.

[9] D. Logofatu, *Grundlegende Algorithmen mit Java*. Springer Vieweg, 2014. S.291.

[10] T. Vincenty, "Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations," tech. rep., Directorate of Overseas Serveys of the Ministry of OVerseas Development, 1975.

[11] Apache Commons. `http://commons.apache.org/`.

[12] uniVocity. https://www.univocity.com/.

[13] Java Geodesy Library for GPS – Vincenty's Formulae. https://github.com/mgavaghan/geodesy.

[14] Google Transit API, frequencies.txt. https://developers.google.com/transit/gtfs/reference/frequencies-file.

[15] P. Brosi, "gtfstidy." https://github.com/patrickbr/gtfstidy.

[16] M. Müller, *Information Retrieval for Music and Motion*. Springer, 2007. S.69-84.

[17] T. Eiter and H. Mannila, "Computing Discrete Fréchet Distance," tech. rep., Technische Universität Wien, 1994.