technische universität
dortmund

Bachelorarbeit

ÖPNV Verspätungsvorhersage mit probabilistischen graphischen Modellen

Lukas Heppe
Oktober 2016

Gutachter:
Prof. Dr. Katharina Morik
Dr. Thomas Liebig

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz (LS-8)
http://www-ai.cs.uni-dortmund.de/index.html

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Viele Fahrzeuge des öffentlichen Nahverkehrs kommen heutzutage zu spät[1]. Dafür gibt es verschiedene Gründe, wie z.B. hohes Verkehrs- und Personenaufkommen oder den Fall, dass ein Fahrzeug auf ein anderes Fahrzeug wartet [12]. Hohes Verkehrsaufkommen sollte aufgrund von Feierabend- oder Schichtwechselverkehr in regelmäßigen Mustern auftreten. Diese Muster sollen in dieser Arbeit erkannt werden, um darauf aufbauend Vorhersagen über die Abweichung der Fahrzeit zu anderen Tagen zu generieren. Solche Vorhersagen können von Routingprogrammen genutzt werden, um Routen unter Einrechnung von prognostizierten Verspätungen zu berechnen [11]. Somit können unnötige Umsteige- und Reisezeiten minimiert werden.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist es, ein System für Warschau zu entwickeln, welches anhand von GPS-Daten die Abweichungen der Fahrtzeiten vom Fahrplan der Stadtbahn in Echtzeit ermittelt und mittels dieser Verspätungen ein Modell lernt, das wiederum solche dynamisch vorhersagt. Als Modell dient ein graphisches Modell namens Spatio-Temporal-Random-Fields [15], welches in der Lage ist, räumliche und zeitliche Abhängigkeiten mithilfe von Zufallsvariablen zu modellieren. Anhand dieses Modells sollen Vorhersagen über die Abweichungen der Fahrtzeiten unter Berücksichtigung von sowohl historischen als auch Echtzeitdaten getroffen und diese in einem geeignetem Datenformat per API bereit zu stellen. Dieses Modell wurde gewählt, da es bereits in der Arbeit [10] erfolgreich bei Modellierung von Straßenverkehr genutzt wurde.

[1] `http://haltestelle.tagesspiegel.de/` (zuletzt besucht am 11.10.2016)

1.3 Aufbau der Arbeit

In dem folgenden Kapitel werden andere Ansätze aus dem Bereich der Fahrzeitvorhersage vorgestellt. Anschließend wird eine Einführung in graphische Modelle, insbesondere der Spatio-Temporal-Random-Fields [15] und Vorhersagen anhand dieser gegeben. In Kapitel 4 wird die während der Arbeit genutzte und entwickelte Softwarearchitektur anhand eines Prozesses präsentiert. Dieses Kapitel wird auch näher auf die genutzten und produzierten Daten eingehen. Im vorletzten Kapitel werden Experimente zur Vorhersage der Verspätungen anhand des Modells durchgeführt und diese ausgewertet. Abschließend gibt es eine Zusammenfassung der Arbeit und Ergebnisse sowie ein Ausblick auf mögliche Verbesserungen.

Kapitel 2

Verwandte Arbeiten

Dieses Kapitel gibt einen Einblick in andere aktuelle Arbeiten und Ansätze im Bereich der Verspätungsvorhersage bzw. der Reisezeitvorhersage. Abschließend werden diese Arbeiten mit dieser Arbeit verglichen. Bevor es jedoch einen Einblick gibt, werden die Begrifflichkeiten *Trip* und *Route* eingeführt. Unter einer Route wird ein Weg verstanden, den ein Fahrzeug regelmäßig fährt. Dabei spielt die Richtung keine Rolle. Ein Trip ist eine Reise entlang der Haltestellen einer Route in eine spezifische Richtung. Außerdem sind Trips immer mit Abfahrtszeiten an den jeweiligen Haltestellen verbunden.

2.1 Traveling Time Prediction in Scheduled Transportation with Journey Segments

In der Arbeit von Gal [5] wird anhand zweier Ansätze die Fragestellung behandelt, wie die Reisezeit von einer Start- zu einer Endhaltestelle für eine vordefinierte Route ermittelt werden kann. Der erste Ansatz fällt in den Bereich der $Queing-Theorie$, der zweite in den Bereich *Maschinelles Lernen*. Beide Ansätze beruhen auf dem *segmentierten Journey−Log*. Das segmentierte Journey-Log beinhaltet historische Beobachtungen für vergangene Trips, zusammen mit deren Trip-Pattern und Trip-Events. Ein Trip-Pattern repräsentiert die Route eines Trips. Trip-Events sind vergangene Stopevents, die die jeweilige Haltestelle zusammen mit dem Zeitstempel enthalten.

Modellierung des Problems

Die Reisezeit von Stop w_1 bis zu dem Stop w_n wird als die Sequenz über die Haltestellen $< w_1, ..., w_n >$ zu der Abfahrtszeit t_{w_1} als Zufallsvariable $T(< w_1, ..., w_n >, t_{w_1})$ modelliert. Dazu wurde für T ein Modell entwickelt, welches die Abhängigkeiten der besuchten Stops eines Fahrzeuges abbildet. In diesem Modell werden die Journeys als Segmente zwischen jeweils zwei Stops modelliert.
Anhand des segmentierten Journey-Logs können die Reisezeiten als Summe der Reisezeit

pro Segment modelliert werden. Als Annahme gilt, dass die Reisezeit eines Segmentes unabhängig von den Linien ist, welche dieses Segment durchfahren. Basierend auf dem segmentierten Journey-Log ergibt sich als Reisezeit für alle Haltestellen [5]:

$$T(< w_1, ...w_n >, t_{w_1}) = T(< w_1, w_2 >, t_{w_1}) + ... + T(< w_{n-1}, w_n >, t_{w_1}) \tag{2.1}$$

mit:

$$t_{w_{n-1}} = t_{w_1} + T(< w_1, w_{n-1} >, t_{w_1}). \tag{2.2}$$

Ansatz 1 : Queing-Theorie

Der erste Ansatz fällt in den Bereich der Queing-Theorie und ist das Snapshot-Prinzip nach Whitt [20]. Es wird die Annahme getroffen, dass ein Fahrzeug die selbe Zeit durch ein Segment benötigt, wie das Fahrzeug, welches zuletzt das Segment passiert hat. Aus dieser Annahme heraus wird ein Single-Segment-Snapshot-Predictor θ_{LBTS} gebildet, der für ein Segment die Reisezeit des letzten Busses wiedergibt, der vor der Zeit t_{w_1} das Segment passiert hat.

$$\theta_{LBTS}(< w_i, w_{i+1} >, t_{w_1}) \tag{2.3}$$

Nun soll basierend auf dem θ_{LBTS}-Predictor für eine Fahrt über mehrere Segmente gebildet werden. Dafür ergibt sich der Snapshot-Predictor θ_{LBTN} wie folgt:

$$\theta_{LBTN}(< w_1, ..., w_n >, t_{w_1}) = \sum_{i=1}^{n} \theta_{LBTS}(< w_i, w_{i+1} >, t_{w_1}) \ . \tag{2.4}$$

Um den Wert von θ_{LBTS} zu ermitteln, wird für das entsprechende Segment die letzte Fahrt, die vor der Zeit t_{w_1} lag, im segmentieren Journey-Log herausgesucht. Als Reisezeit ergibt sich die Differenz der Ausfahrts- und Einfahrtszeit aus dem Segment. Der Vorteil dieses Ansatzes ist, dass er nur die vergangenen Reisezeiten der Busse in einem segmentierten Journey-Log benötigt, jedoch sind dessen Vorhersagen nicht immer plausibel, gerade dann wenn das Netzwerk unter starker Last steht [16].

Ansatz 2 : Maschinelles Lernen

Basierend auf dem segmentiertem Journey-Log-Modell wird ein Machine-Learning-Predictor $\theta_{ML}(w_i, w_{i+1}, t, \hat{t}_w)$ für jedes Segment entlang der Route konstruiert. Anhand der einzelnen θ_{ML} kann die Reisezeit $T(< w_1, ...w_n >, t_{w_1})$ wie folgt modelliert werden:

$$T(< w_1, ...w_n >, t_{w_1}) = \sum_{i=1}^{n-1} \theta_{ML}(w_i, w_{i+1}, t_{w_1}, \hat{t}_{w_i}) \tag{2.5}$$

mit:

$$\hat{t}_{w_i} = t_{w_1} + \theta_{ML}(w_i, w_{i+1}, t_{w_1}, t_{w_1}) \tag{2.6}$$

für die geschätzte Eintrittszeit des Segments i.
Als Eingabe bekommt ein einzelner Predictor die Reisezeit des letzten Busses durch das spezifische Segment $\theta_{LBTS}(< w_i, w_{i+1} >, t_{w_1})$, das Intervall zwischen $\hat{t}_{w_i}$ und der Zeit, zu der der letzte Bus das Segment verlassen hat, den jeweiligen Wochentag sowie die Uhrzeit. Seine Ausgabe ist die Reisezeit des jeweiligen Segmentes.
Das verwendete Modell ist ein Ensemble aus M Entscheidungsbäumen [7]. Die Vorhersage eines Ensembles ist die gewichtete Vorhersage aller Bäume des Ensemble:

$$\Psi_M(.) = \sum_{m=1}^{M} \lambda_m \psi_m(.) \tag{2.7}$$

mit λ_m als Gewichtung des Baumes m und ψ_m als Vorhersage.

2.2 Predicting Bus Arrivals Using OneBusAway Real-Time Data

Die Arbeit von Baker und Nied [1] zielt darauf ab, Vorhersagen über die Verspätung an kommenden Haltestellen einer Busfahrt zu generieren. Ihre Daten bekommen sie aus einem Echtzeitsystem, welches sie alle 1 bis 3 Minuten bereitstellt. Diese beinhalteten Informationen über die TripId, den Längen- als auch Breitengrad, die zurückgelegte Strecke sowie Verspätung der aktuellen Fahrt. Als Features für die Vorhersage der Abweichung werden die oben genannten Daten sowie die aktuelle Uhrzeit und der Wochentag einbezogen.
Um die Vorhersagen zu generieren, werden auch hier 2 Ansätze verfolgt, eine k-Nearest-Neighborhood-(kNN) [7] und eine Kernel-Regressions-Methode [2]. Bei der kNN-Methode wird die Distanz zu allen Punkten des Trainingdatensatzes berechnet und als Vorhersage die durchschnittliche Abweichung der k nächsten Punkte genommen. Dies wird für alle möglichen k's durchgeführt und als Modell das k genommen, welches den geringsten Validierungsfehler aufweist.
Bei der Kernel-Regression wird jeder Punkt x des Trainingsets im Vergleich zu dem Testpunkt y gewichtet. Als Verspätung wird der gewichtete Durchschnitt aller Punkte genommen. Als Kernelfunktion dient:

$$kernel(x, y) = e^{\frac{-distance(x,y)^2}{\sigma^2}} \ . \tag{2.8}$$

Auch hier wurden alle möglichen Kernelweiten σ getestet und diejenige mit dem niedrigsten Validierungsfehler gewählt.
Für eine Verbesserung der Vorhersagen wurden die einzelnen Features unterschiedlich gewichtet. Dazu wurde per stochastischem Gradientenverfahren über den Trainingsdatensatz iteriert und der Fehler, unter Vorhersage des Features, mit unterschiedlichen Gewichten minimiert.

2.3 Artificial Neural Network Travel Time Prediction Model for Buses Using Only GPS Data

Das Ziel der Arbeit von Gurmu und Fan [6] ist es, basierend auf GPS-Koordinaten und einem Aritificial-Neural-Network (ANN) Vorhersagen über die Reisezeit zu einer noch kommenden Haltestelle einer bereits gestarteten Fahrt vorherzusagen. Dazu nutzen sie die Daten des Automatic-Vehicle-Location-Systems (AVL) in Macae, Brasilien. Dieses stellt Informationen über die GPS-Position, Geschwindigkeit eines Fahrzeuges, Datum, Uhrzeit und den Ankunftszeiten an bereits passierten Haltestellen bereit, welche für ein halbes Jahr aufgezeichnet wurden. Bevor auf diesen Daten das gleich vorgestellte Modell trainiert wurde, wurde die zeitliche Dimension auf halbstündliche Intervalle reduziert. Für jedes dieser Intervalle wurde die durchschnittliche Reisezeit der entsprechenden Fahrten berechnet.
Das ANN wurde mit drei Layern modelliert. Ein Layer für den Eingabevektor, ein Hiddenlayer und einen Outputlayer. Als Features dienten das jeweilige Intervall des Tages, die Id der Haltestelle i sowie j und die Reisezeit von der ersten bis zur i-ten Haltestelle. Die Ausgabe Y des ANN war die Reisezeit von der i-ten zur j-ten Haltestelle.
Für einen Bus k, welcher an der Haltestelle 0 in einem Intervall t losfährt und sich aktuell an der GPS-Position c befindet, nachdem er die Haltestelle i passiert hat, lässt sich die Reisezeit von der Position c zur Haltestelle j wie folgt berechnen:

$$TT_{cj}^{k} = TT_{ij}^{k} - TT_{ic}^{k} \tag{2.9}$$

mit TT_{ij}^{k} als vorhergesagte Reisezeit zwischen den Haltestellen i und j und TT_{ic}^{k} als Reisezeit bis zum Punkt c von der Haltestelle i. Da das Fahrzeug k aktuell an der Position c ist, wird der Wert TT_{ic}^{k} als Differenz zwischen der aktuellen Zeit und Ankunftszeit an Haltestelle i gebildet. Um den Wert TT_{ij}^{k} zu bestimmen, wurde das ANN zur Hilfe herangezogen. Dazu werden dem ANN die Features als Vektor übergeben und dieser klassifiziert. Der Wert des Outputlayers wurde für TT_{ij}^{k} genutzt. Um die Verspätung an der j-ten Haltestelle zu berechnen, muss der TT_{cj}^{k} mit der bereits vergangen Fahrzeit zum Punkt c addiert werden und die Differenz zwischen diesem Wert und der planmäßigen Ankunftszeit gebildet werden.

2.4 Verbindung

Die anderen Arbeiten wurden vorgestellt, da sie andere Ansätze zur Vorhersage der Reisezeit bzw. Abweichung der Ankunftszeit präsentieren. Sie basieren auf unterschiedlichen Verfahren, wie z.B. Queing-Theorie aber auch Methoden des Maschinellen Lernens, wie Decision-Trees, Nearest-Neighbourhood-Methoden, Aritificial-Neural-Networks oder Kernel-Regression. Alle diese Arbeiten stellen Methoden vor, in denen Modelle auf historischen Daten gelernt werden und zusammen mit Echtzeitevents für genauere Vorhersagen kom-

biniert werden. Anhand der Arbeiten wurden verschiedene Möglichkeiten vorgestellt, um sowohl die Zeit als auch die Verspätungen zu modellieren. Die zeitliche Dimension kann sowohl fahrtenweise als auch als Intervall abgebildet werden. Außerdem können die Modelle über eine Linie oder mehrere Linien gebildet und die Zustände stetig oder diskret modelliert werden. Für diese Arbeit wird eine fahrtenweise Abbildung der Zeit gewählt, da so genaue Vorhersagen über einzelne Fahrten möglich sind und die Vorhersagen nicht über den Durchschnitt eines Intervalls gebildet werden. Darüber hinaus werden in dieser Arbeit linienunabhängige Modelle genutzt und zwar zwei Modelle pro Route, jeweils eins für die Hin- und Rückrichtung. Der Grund dafür ist, dass so die Linien miteinander verglichen werden und die Modelle einfacher ausgetauscht werden können. Die Zustände sind diskret gewählt, da die vorliegende Modellimplementierung nur mit diesen umgehen kann.

Kapitel 3

Probabilistische Graphische Modelle

Beim maschinellem Lernen geht es darum, dass Computer statistische Lernverfahren auf Daten anwenden, um automatisch Muster und Zusammenhänge zu erkennen [2]. Probabilistische Graphische Modelle sind ein Teilgebiet des Maschinellen Lernens und kombinieren Graphen- und Wahrscheinlichkeitstheorie. Graphen repräsentieren Wahrscheinlichkeitsverteilungen und deren komplexe Abhängigkeiten anhand von Zufallsvariablen.

3.1 Graphische Modelle

Ein Graph $G = (V, E)$ besteht aus einer Menge an Knoten V und einer Menge an Kanten $E \subset V \times V$. Die Kanten können sowohl gerichtet als auch ungerichtet sein. Sind die Kanten gerichtet, handelt es sich um ein Bayesian Network, ansonsten um ein Markov-Random-Field (MRF) [8]. In dieser Arbeit wird jedoch nur mit MRF's bzw. deren Reprametrisierung Spatio-Temporal-Random-Fields (STRF) [15] gearbeitet.
In einem probabilistischen graphischen Modell repräsentiert jeder Knoten $v \in V$ eine Zufallsvariable X_v, welche Werte aus ihrem Wertebereich $\mathcal{X}_v$ annimmt [2]. Der Wertebereich kann sowohl diskret als auch stetig sein. In dieser Arbeit wird mit einem diskreten Wertebereich gearbeitet. Die Kanten repräsentieren probabilistische Abhängigkeiten zwischen den Zufallsvariablen. Der gesamte Graph steht für die gemeinsame Wahrscheinlichkeitsverteilung $p(x)$ aller Zufallsvariablen X_v. Graphische Modelle sind durch einen Vektor θ parametrisiert. Dieser Vektor repräsentiert die Kantengewichte und ist immer spezifisch von den Daten abhängig, auf denen gelernt werden soll.

3.2 Eigenschaft der bedingten Unabhängigkeit

Für einen gegebenen Graphen G und drei Mengen A, B und C aus Knoten des Graphen, gilt die Eigenschaft der bedingten Unabhängigkeit [2]:

$$A \perp\!\!\!\perp B \mid C \tag{3.1}$$

genau dann, wenn alle Pfade aller Knoten $a \in A$ jeden Knoten $b \in B$ nur durch Knoten der Menge C erreichen können.

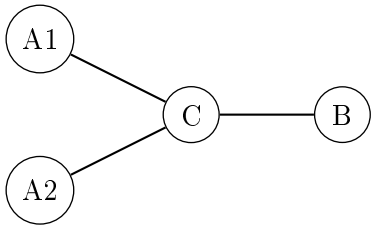


Abbildung 3.1: Beispiel bedingte Unabhängigkeit

Anhand des Graphen aus Abb. 3.1 wird ein Beispiel erläutert. Würden seine Knoten in die Mengen $A = \{A1, A2\}$, $B = \{B\}$ und $C = \{C\}$ geteilt werden, so wäre die Eigenschaft der bedingten Unabhängigkeit erfüllt.
Die Eigenschaft der bedingten Unabhängigkeit ist für alle Wahrscheinlichkeitsverteilungen $p(x)$ eines Modells erfüllt und somit das Bindeglied zwischen Graphen- und Wahrscheinlichkeitstheorie, da die Knoten Zufallsvariablen sind und so die Eigenschaft:

$$p(a, b|c) = p(a|c) \tag{3.2}$$

erfüllt ist. Für einen Graphen bedeutet dies, dass ein Knoten bzw. eine Menge von diesen unabhängig von allen Knoten außer seinen Nachbarn ist.

3.3 Wahrscheinlichkeitsverteilung eines MRF

Die Wahrscheinlichkeitsverteilung $p(x)$ eines MRF ergibt sich durch die Faktorisierung des Graphen. So kann $p(x)$ als Produkt von lokalen Funktionen, basierend auf der Graphstruktur, ausgedrückt werden.
Seien X und Y zwei Knoten eines Graphen G und es existiert keine Kante (x,y), dann gilt aufgrund von 3.2, dass X und Y bedingt unabhängig gegeben alle anderen Knoten im Graph sind. Daraus folgt, dass die Faktorisierung nicht gleichzeitig X und Y enthalten darf, da ansonsten die Eigenschaft aus 3.2 nicht für alle Wahrscheinlichkeitsverteilungen über dem Graphen gelten würde.
Somit kommen wir zu dem Konzept der *Clique*. Eine Clique C ist definiert als Teilmenge über die Knoten eines Graphen, in der jeder Knoten der Clique C eine Kante zu jedem anderem Knoten in C besitzt:

$$\forall\, v_1, v_2 \in C\ .\ \exists\, (v_1, v_2) \in E\ ,\ wenn\ v_1 \neq v_2 \tag{3.3}$$

Um eine *maximale Clique* handelt es sich, wenn der Clique kein weiterer Knoten hinzugefügt werden kann, da es sich sonst um keine Clique mehr handeln würde.
Die Wahrscheinlichkeitsverteilung eines ungerichteten Graphen wird anhand der Cliquen

ausgedrückt, welche durch die Graphstruktur gegeben sind. Jeder Clique wird eine *Potential−funktion* $\psi_c(x_c)$ zugeordnet. Alle Potentialfunktionen sind strikt positive Funktionen auf den Elementen von C mit dem Wertebereich $\mathbb{R}_+$.
Sei C eine Menge an maximalen Cliquen und x_c die Menge an Variablen einer Clique. Dann kann die gemeinsame Wahrscheinlichkeitsverteilung des graphischen Modells als Produkt von *Potentialfunktionen* $\psi_c(x_c)$ über die maximalen Cliquen des Graphen wie folgt gebildet werden [2] :

$$p(x) = \frac{1}{Z} \prod_C \psi_c(x_c). \tag{3.4}$$

Z dient als Normalisierungskonstante und wird auch Partitionsfunktion genannt und ist gegeben durch [2]:

$$Z = \sum_X \prod_C \psi_c(x_c). \tag{3.5}$$

3.4 Spatio-Temporal-Random-Fields

Spatio-Temporal-Random-Fields [15] sind ein raum-zeitliches graphisches Modell und eine Reparametrisierung von Markov-Random-Fields [8]. Sie bestehen aus einem Basisgraph $G_0 = (V_0, E_0)$, welcher die räumliche Struktur der Zufallsvariablen modelliert. Weiterhin besteht ein STRF aus T Snapshotgraphen $G_t = (V_t, E_t)$, welche den Zustand der räumlichen Struktur zum Zeitpunkt $t \in T$ modellieren. Außerdem besitzt der Graph eine Menge $E_{t-1;t} \subset V_{t-1} \times V_t$ an zeitlichen Kanten für alle Zeitschichten von 2 bis T. Für t = 1 gilt $E_{0;1} = \emptyset$. Diese Kanten modellieren die Abhängigkeiten zwischen zwei benachbarten Snapshotgraphen. Für beliebige t gilt, falls $h > 1$, dass $E_{t;t+h} = \emptyset$.
Der komplette Graph $G = (V, E)$ setzt sich aus den einzelnen Snapshotgraphen G_T in zeitlich aufsteigend geordneter Reihenfolge $t = 1, 2, ..., T$ zusammen, welche von den zeitlichen Kanten verbunden werden. Insgesamt ergibt sich als Menge der Knoten $V = \bigcup_{t=1}^{T} V_t$ und als Menge der Kanten $E = \bigcup_{t=1}^{T} \{E_t \cup E_{t-1;t}\}$.
Anhand der Knoten und Kanten lassen sich die Anzahl der Modellparameter d berechnen. Zur Vereinfachung wird davon ausgegangen, dass alle Knoten des Modells die selbe Zustandsmenge besitzen[2]. Dann kann d wie folgt ausgedrückt werden:

$$d = T|V_0||X_{v_t}| + [(T-1)(|V_0| + 3|E_0|) + |E_0|]|X_{v_t}|^2 \,. \tag{3.6}$$

Dieser Wert ergibt sich aus den möglichen Zuständen pro Knoten für jede Zeitschicht plus die möglichen Zustandskombinationen für die Kanten zwischen zwei Knoten.
Die Dichtefunktion der gemeinsamen Wahrscheinlichkeitsverteilung ist gegeben durch:

$$p_\theta(X = x) = \frac{1}{\Psi(\theta)} \prod_{v \in V} \psi_v(x) \prod_{(u,v) \in E} \psi_{(u,v)}(x) \,. \tag{3.7}$$

[2]Dies wird angenommen, da in dieser Arbeit nur mit Modellen mit den gleichen Zustandsmengen gearbeitet wird und die Formel so deutlich verkürzt. In anderen Settings könnte auch jeder Knoten eine unterschiedliche Menge an Zuständen besitzen.

mit $\Psi(\theta)$ als Log-Partition-Funktion :

$$\Psi(\theta) = log \int_X exp[\langle \theta, \phi(x) \rangle] v dx \tag{3.8}$$

X repräsentiert den zufälligen Zustand aller Knoten zu allen Zeitpunkten T und x ist eine spezielle Konfiguration aus X. Bei $\phi(x)$ handelt es sich um eine Indikatorfunktion, welche den Wert 1 zurück gibt, falls sich das STRF in dieser Konfiguration befindet, andernfalls gibt sie 0 zurück.

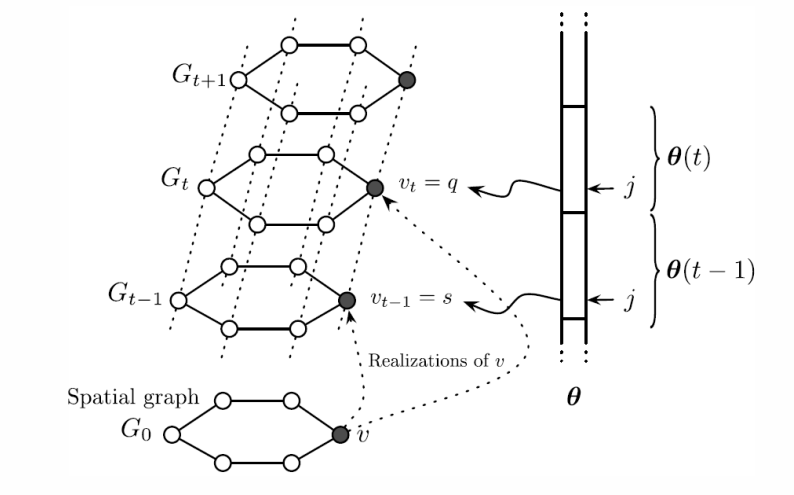


Abbildung 3.2: Spatio-Temporal-Random-Field [15]

Ein STRF ist schichtweise für jedes t durch $\theta(t)$ parametrisiert. Ein $\theta(t)$ setzt sich aus den einzelnen Parametervektoren $Z_{\cdot i} \in \mathbb{R}^d$ der vorherigen Schichten wie folgt zusammen:

$$\theta(t) = \sum_{i=1}^{t} \frac{1}{t-i+1} Z_{\cdot i} \tag{3.9}$$

Die komplette Parametrisierung ist durch die Konkatenation der einzelnen Parametervektoren zu allen Zeitschichten T gegeben:

$$\theta = \begin{bmatrix} \theta(1) \\ \theta(2) \\ \vdots \\ \theta(T) \end{bmatrix} \tag{3.10}$$

In Abb. 3.2 ist ein beispielhaftes Spatio-Temporal-Random-Field zu sehen. Sein Basisgraph besteht aus 6 Knoten. Auf diesem werden die restlichen T Zeitschichten gebildet, welche jeweils durch den Vektor $\theta(t)$ parametrisiert sind.

3.5 Inferenz

In diesem Kapitel werden die Vorhersagen anhand des probabilistischen graphischem Modells erklärt. Unter Inferenz wird sowohl die Ermittlung der Marginalhäufigkeiten als auch die Suche nach einer Maximum-A-Priori-Verteilung verstanden. Die Marginalhäufigkeiten können durch den Belief-Propagation-Algorithmus nach Pearl [14] ermittelt werden, jedoch nur für einen gerichteten Graphen. Deshalb wird auf den Sum-Produkt-Algorithmus [9] zurückgegriffen, welcher auf einem *Faktorgraph* aufgerufen wird. Zur Berechnung der Maximum-A-Priori-Verteilung wird eine Variante des Sum-Produkt-Algorithmus genutzt, der Max-Produkt-Algorithmus.

3.5.1 Faktorgraph

Ein Faktorgraph ist ein bipartiter Graph, der die Faktorisierung der Struktur eines Modells darstellt [9]. Sowohl gerichtete als auch ungerichtete Graphen können in einen solchen überführt werden. Faktorgraphen bestehen aus zwei Typen von Knoten. Dies sind die Knoten des jeweiligen zu konvertierenden Graphen, sowie die sogenannten Faktorknoten. Für jede Potentialfunktion $\psi_C(x_C)$ bzw. deren maximale Clique C wird ein Faktorknoten zu dem Faktorgraphen hinzugefügt. Die Kanten ergeben sich, indem die Knoten der Clique C mit dem jeweiligen Faktorknoten verbunden werden, der ihre Potentialfunktion repräsentiert.

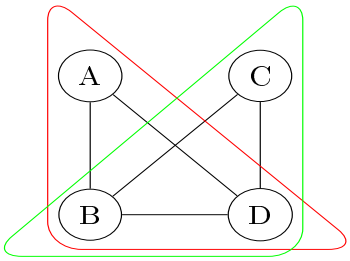


Abbildung 3.3: Zu faktorisierender Graph

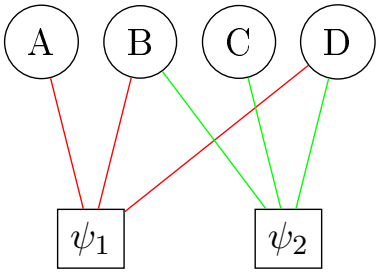


Abbildung 3.4: Faktorisierter Graph

Ein Beispiel ist in den Abbildungen 3.3 und 3.4 gegeben. Der Graph aus Abb. 3.3 besitzt die beiden maximalen Cliquen $C_1 = \{A, B, D\}$ und $C_2 = \{C, B, D\}$. Somit existieren 2 Faktoren ψ_1 und ψ_2, für welche ein Faktorknoten im Faktorgraph in Abb. 3.4 eingefügt wird. Diese werden als Rechteck dargestellt. Die Kanten ergeben sich wie oben beschrieben. Jeder Knoten bekommt eine Kante zu den Faktorknoten, zu dessen Clique er Mitglied ist. Sowohl gerichtete als auch ungerichtete Graphen können in Faktorgraphen überführt werden.

3.5.2 Sum-Produkt-Algorithmus

Das Ziel des Sum-Produkt-Algorithmus ist es, die marginalen Häufigkeiten aller Knoten unter möglichen Observationen zu ermitteln [9]. Dazu sollen die Funktionen $p_i(x_i)$ ermittelt

werden, welche die marginalen Häufigkeiten der Zufallsvariable x_i repräsentieren. Dieses Ziel wird durch einen Message-Passing-Algorithmus erreicht. Jeder Knoten soll mit seinen Nachbarn interagieren und diese danach fragen, was sie denken, in welchem Zustand der Knoten selbst ist. Es existieren zwei Arten von Nachrichten, die von Variablen- zu Faktorknoten [9]:

$$\mu_{x\to\psi_C}(x) = \prod_{\psi_C\in n(x)\setminus\{\psi C\}} \mu_{\psi_C\to x}(x) \tag{3.11}$$

und die von Faktor- zu Variablenknoten [9]:

$$\mu_{\psi_C\to x}(x) = \sum_{\sim\{x\}} \left(\psi_C(X) \prod_{y\in n(\psi_C)\setminus\{x\}} \mu_{y\to\psi_C}(y) \right), \tag{3.12}$$

wobei die Funktion $n(x)$ die benachbarten Knoten von x zurückgibt, $\sum_{\sim\{x\}}$ für die Summierung über alle Knoten des Faktors außer x und X für $n(\psi_C)$ steht. Bei den Nachrichten handelt es sich um Funktionen über den Zustandsraum $\mathcal{X}_i$ von x.
Um die marginalen Häufigkeiten aller Knoten zu berechnen, wird ein Knoten als Wurzel ausgewählt. Anschließend werden die Blätter des Graphen mit 1 initialisiert und senden ihre Nachricht an ihre Nachbarn. Diese Nachrichten werden solange rekursiv weitergeleitet, bis der ausgewählte Wurzelknoten die Nachrichten aller seiner Nachbarn bekommen hat. Daraufhin sendet der Wurzelknoten eine Nachricht an alle seine Nachbarn zurück, welche wiederum rekursiv an alle Knoten weitergeleitet werden. Enthält der Graph Zyklen, wird der Vorgang mehrmals iterativ durchgeführt, bis sich die Funktionswerte minimal ändern. Dies dient als Konvergenzkriterium und sobald es erreicht wird, können die Marginalhäufigkeiten aller Knoten berechnet werden. Die Marginalhäufigkeit eines Knoten x ergibt sich als das Produkt der eintreffenden Nachrichten an seinen zugehörigen Faktorknoten:

$$p_i(x_i) = \psi_C(x_i) \prod_{i\in n(\psi_C)} \mu_{x\to\psi_C}(i) \tag{3.13}$$

Dieser Algorithmus kann auch bereits beobachtete Werte $x = v$ mit einbeziehen, indem die marginalen Häufigkeiten der Events $x \neq v$ auf 0 gesetzt werden. Somit ergibt sich die Wahrscheinlichkeitsverteilung $p(x|y)$, wobei y eine beliebige Menge an beobachteten Knoten sein kann.

3.5.3 Max-Produkt-Algorithmus

Der Max-Produkt-Algorithmus ist eine Abänderung des Sum-Produkt-Algorithmus [2, 9]. Sein Ziel ist es, eine Maximum-a-posteriori-Verteilung (MAP) x_{max} zu berechnen, welche jedem Knoten x des Graphen seinen Wert in der wahrscheinlichsten Gesamtkonfiguration zuordnet. Somit ergibt sich das Problem, die wahrscheinlichste Verteilung zu finden, wie folgt:

$$x_{max} = \arg\max_x p(x)\,, \tag{3.14}$$

wobei $p(x)$ auch $p(x|y)$ sein darf und y eine beliebige Menge an beobachteten Knoten.
Um diesen Vektor zu berechnen, wird der Sum-Produkt-Algorithmus leicht abgeändert genutzt. In der Formel 3.11 wird die $\sum$ durch die arg max-Funktion ersetzt, was zu folgender Nachricht für Faktor- an Variablenknoten führt:

$$\mu_{x\to\psi_C}(x) = \arg\max_x \prod_{y\in n(\psi_C(X))\setminus\{x\}} \mu_{y\to\psi_C(X)}(y). \tag{3.15}$$

Außerdem wird sich bei jeder Berechnung der Nachricht eines Faktors an eine Variable der Zustand gemerkt, welcher die Nachricht maximierte. Auch hier wird ein Knoten als Wurzel ausgewählt und anschließend beginnen die Blätter ihre Nachrichten rekursiv zu senden, bis der Wurzelknoten die Nachrichten aller seiner Nachbarn erhalten hat. Anschließend werden per Backtracing die Knoten zurückverfolgt, um deren wahrscheinlichsten Zustand in der Gesamtkonfiguration zu ermitteln. Bereits beobachtete Knoten verweilen in ihrem Zustand.

3.6 Lernen der Modellparameter

Um ein Modell an die Daten anzupassen, müssen die Modellparameter θ ermittelt werden. Dies kann mit einer $Maximum-Likelihood-Estimation(MLE)$ [19] erledigt werden. Sei $\mathcal{D} = \{x^1, x^2, ..., x^n\}$ ein Datensatz aus n Beobachtungen und jedes $x^i \in X$ eine Konfiguration über alle Knoten des Graphen. Dann ist die Likelihood $\mathcal{L}$ der Parameter θ für einen Datensatz $\mathcal{D}$ wie folgt definiert:

$$\mathcal{L}(\theta, \mathcal{D}) := \prod_{i=1}^{n} p_\theta(x^i). \tag{3.16}$$

Da die Formel aufgrund ihres Produktes jedoch teuer zu berechnen ist, wird oft auf die Log-Likelihood ℓ zurückgegriffen. Dazu wird der Logarithmus auf die Formel (3.10) angewendet, was zu dem folgendem Ausdruck für ℓ führt:

$$\ell(\theta, \mathcal{D}) := \frac{1}{N} \sum_{i=1}^{N} log\ p_\theta(x^i) \tag{3.17}$$

Indem die Log-Likelihood maximiert wird, werden die Parameter an die Wahrscheinlichkeitsverteilung der Events aus den Daten angepasst. Es wird davon ausgegangen, dass wenn der Ausdruck (3.11) maximiert ist, die Parameter optimal an die Daten angepasst sind [2]. Dies ist äquivalent zur Minimierung von $-\ell$.
Für ein durch Z parametrisiertes STRF ergibt sich das folgende Optimierungsproblem [15]:

$$\min_{Z\in R^{d`\times T}} h(Z) := -\ell(\theta(Z), \mathcal{D})\ , \tag{3.18}$$

welches durch das Verfahren des konjugierten Gradienten gelöst werden kann [13]. Die Optimierung wird beendet sobald die Gradientennorm unter ein ϵ fällt, sich die Log-Likelihood in zwei Schritten um weniger als ϵ ändert oder das Verfahren für eine maximale Anzahl an Iterationen lief.

Kapitel 4

Architektur

In diesem Kapitel wird eine Übersicht über die Architektur der implementierten Software gegeben. Zu Beginn wird das Echtzeit-Framework Streams [3] vorgestellt, welches benutzt wurde, um die Daten abzugreifen und zu verarbeiten. Anschließend werden die verwendeten Daten sowie der genutzte Prozess in Streams vorgestellt.

4.1 Streams-Framework

Streams ist ein Echtzeit-Framework zur Streamverarbeitung. Prozesse in Streams werden in XML modelliert und bestehen aus vier abstrakten Elementen. Diese sind Container, Streams, Services sowie Prozesse. Container stellen die Umgebung zum Ausführen bereit. Innerhalb dieser Container können Streams als Datenquellen definiert werden. Streams lesen aus beliebigen Quellen und erzeugen Dataitems. Ein Dataitem ist ein Objekt, bestehend aus einer Menge an (k, v) Key-Value Paaren. Diese Dataitems werden anschließend an einen Prozess weitergereicht, welcher die Items wiederum an seine Prozessoren weiterleitet. Diese Prozessoren leisten die eigentliche Datenverarbeitung. Weiterhin gibt es Services, die die verschiedensten Funktionen bereitstellen können. In dieser Arbeit werden sie z.B. genutzt, um sich Daten und Fahrpläne zu speichern und Zugriff auf die Modell mit ihren benötigten Funktionen bereitstellen. Das Streams-Framework wurde aufgrund seiner hohen Flexibilität gewählt, da Parameter in der XML-Deklaration und nicht direkt im Code geändert werden müssen.

4.2 Daten

Um die Verspätungen zu erkennen und zu prognostiezieren, wurden Daten aus verschiedenen Quellen und verschiedener Art benutzt.

4.2.1 API

Die benutzten Daten wurden von der Stadt Warschau per REST-API bereitgestellt. Sie liegen im JSON-Format vor und geben die Positionen der Stadtbahnen in Warschau in Echtzeit an. Ein API-Aufruf liefert eine Liste von JSON-Objekten, von denen jedes Objekt ein Fahrzeug repräsentiert. Ein einzelnes Objekt beinhaltet Informationen über die Linie, Brigade, Uhrzeit und die GPS-Position des Fahrzeuges. In Abbildung 4.1 ist ein solches JSON-Objekt zu sehen. Jedes JSON-Objekt eines API-Aufrufes wird zu einem Dataitem in Streams konvertiert. Ein Attribut eines Objektes wird genau zu einem Key-Value-Paar eines Dataitems konvertiert.

```
{"result":[
{"Status":"RUNNING",
"FirstLine":"4",
"Lon":21.0210075,
"Lines":"4",
"Time":"2016-07-21T16:14:47",
"Lat":52.2146454,
"LowFloor":true,
"Brigade":"12"},
{...},
]}
```

Abbildung 4.1: Ein Objekt eines API-Aufrufes

4.2.2 GTFS

Außer den Fahrzeugspositionsdaten werden zusätzlich die Fahrpläne benötigt, um einen Vergleich zur Ermittlung der Verspätung zu besitzen. Diese werden heutzutage oft im GTFS-Format modelliert. GTFS steht für „General Tranist Feed Specification“ und ist ein von Google ins Leben gerufene Datenformat. Ein GTFS-Feed ist eine Datenbank ähnliche Datenstruktur, welche aus mehreren CSV-Dateien besteht. Die einzelnen Dateien stehen durch ID‘s, wie in Abb. 4.2 gezeigt, im Zusammenhang und modellieren so die Abhängigkeiten zwischen Routen, Trips, Agenturen, Haltestellen und Abfahrtszeiten. Die hier verwendeten Fahrpläne wurden im Februar und März 2016 mit Hilfe des Conveyal GTFS-Editors[3] erstellt.

[3] `https://github.com/conveyal/gtfs-editor` (zuletzt besucht am 02.10.2010)

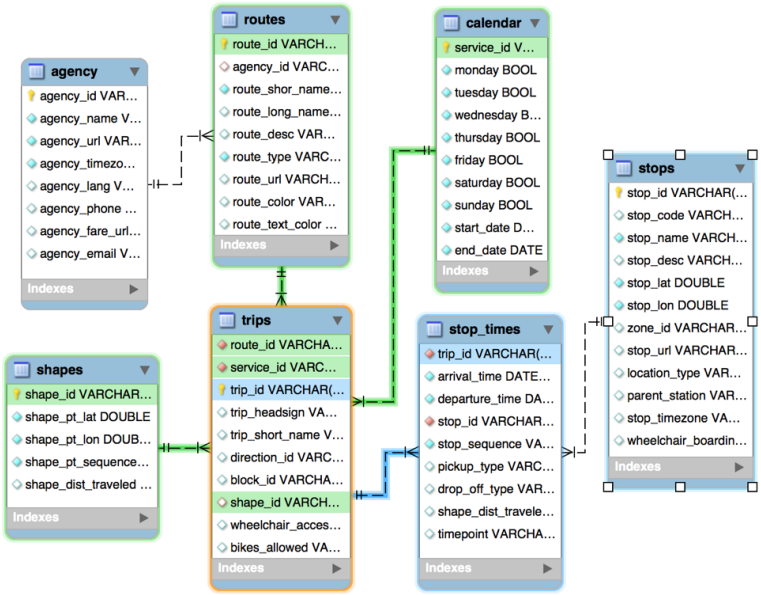


Abbildung 4.2: GTFS-Schema[4]

4.2.3 GTFS-Realtime

Das Ziel ist es, die Verspätungen im GTFS-Realtime-Format[5] auszugeben, da dieses das Standardinterface für Routingprogramme, wie beispielsweise OpenTripPlanner, ist um Routen unter Berücksichtigung von Verspätungen zu planen[6] . Ein GTFS-Realtime-Feed kann aus bis zu drei verschiedenen Feedtypen bestehen - ServiceAlerts, Vehicle Positions und TripUpdates. In dieser Arbeit wird der Realtime-Feed nur aus Trip-Updates bestehen. Der GTFS-Realtime-Feed beinhaltet zu Trips aus dem GTFS-Feed TripUpdates, welche die Verspätungen an den Haltestellen des Trips in Sekunden angibt. Ein Realtime-Feed muss immer in Abhängigkeit eines GTFS-Feedes erzeugt werden, damit die TripIds übereinstimmen. Außerdem sollte ein solcher GTFS-Realtime-Feed einmal die Minute aktualisiert werden. In Abb. 4.3 ist eine Instanz von OpenTripPlanner zu sehen, welche sich Daten über einen GTFS-Realtime-Server holt und diese in das Routing miteinbezieht. Hier wurde z.B. eine Verfrühung von 1 Minute für die Linie 15 prognostiziert.

4.2.4 Sonstige Daten

Weiterhin wurden zur Erkennung der Trips die Start- bzw. Endhaltestellen der jeweiligen Linien inklusive GPS-Bereichen benötigt. Diese stehen in einer Konfigurationsdatei und

[4] `https://www.atlantbh.com/wp-content/uploads/2016/07/Picture1.png` (zuletzt besucht am 02.10.2010)

[5] `https://developers.google.com/transit/gtfs-realtime/` (zuletzt besucht am 02.10.2010)

[6] `http://docs.opentripplanner.org/en/latest/Configuration/` (zuletzt besucht am 02.10.2010)

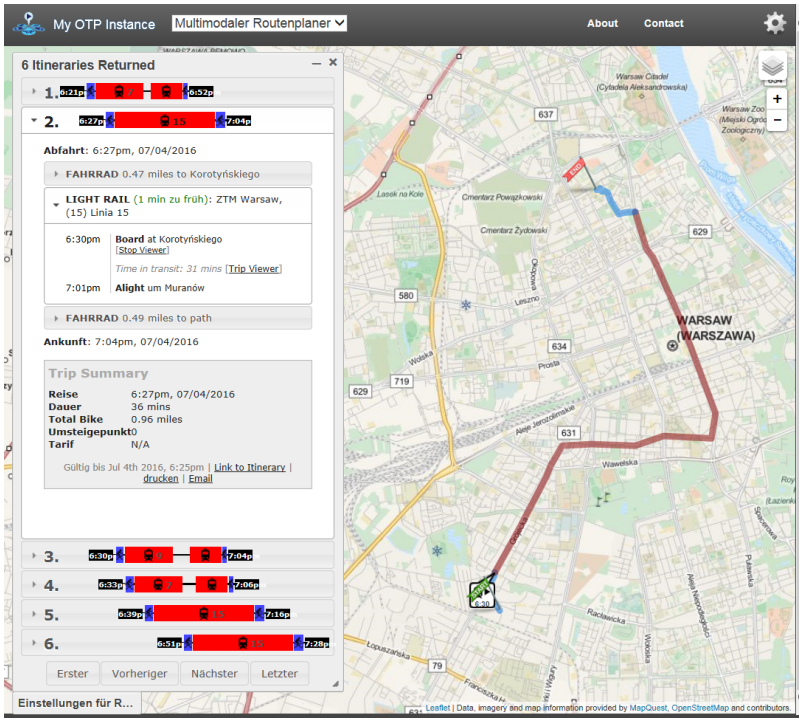


Abbildung 4.3: Verspätung in OpenTripPlanner

werden bei Programmstart eingelesen und erzeugt. Damit zwischen den STRF's unterschieden werden kann, werden sie mit einer Id versehen. Da ein STRF eine Strecke einer Route repräsentiert wurden ihnen Id's nach dem Schema $Linie\_FirstStopId\_LastStopId$ zugeordnet. Darüber hinaus müssen alle Trips eines STRF's indiziert werden, damit wenn beispielsweise eine Verspätung beobachtet wird, sie in dem Modell in der entsprechenden Zeitschicht gesetzt werden kann. Die Indizierung erfolgt, indem alle Trips für ein STRF eingelesen werden, sie aufsteigend nach ihrer Abfahrtszeit sortiert werden und anschließend von der ersten bis zur letzten Fahrt von 0 bis M durchnummeriert werden. Dies wurde ebenfalls vor dem Programmstart erledigt, so dass die Trips mit ihren zugehörigen Id in einer CSV-Datei liegen, welche nur noch eingelesen werden muss.

4.3 Streams Container

Dieses Kapitel wird den Ablauf in Streams näher legen. Der Prozess wird Schritt für Schritt mit Erläuterungen erklärt. Zu Beginn wird ein Überblick über den Prozess gegeben und anschließend werden seine einzelnen Schritte näher erläutert. In Abb. 4.4 ist ein Datenflussdiagramm des gesamten Prozesses dargestellt. Über die API treffen GPS-Daten aus Warschau ein, welche zunächst aufbereitet werden müssen. Anschließend müssen die GPS-Koordinaten Fahrten zugeordnet werden, um so Verspätungen zu berechnen. Diese Verspätungen werden genutzt, um ein Modell zu lernen, welches wiederum Vorhersagen über die

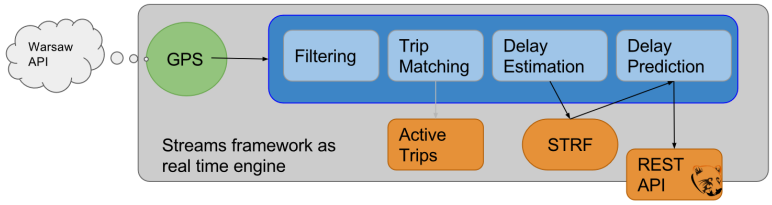


Abbildung 4.4: Prozess-Übersicht

Verspätung der zukünftigen Fahrten generiert. Die prognostizierten Verspätungen werden daraufhin per REST-API in einem geeigneten Datenformat bereitgestellt.

4.3.1 Stream

Der `TramStream` ließt alle 15 Sekunden die Daten eines API-Aufrufes ein. Als Intervall wurden 15 Sekunden gewählt, da nach dieser Zeit die API neue Daten bereitstellt. Die eingelesen Daten werden innerhalb des Streams-Frameworks zu Dataitems verarbeitet und an den eigentlichen Prozess weitergereicht. Bevor dieser jedoch erklärt wird, werden die fünf verschiedenen Services aus Abbildung 4.5 erläutert.

```
<stream id="jsonStream" class="ba.streams.TramStream" nextapicall="15" />
<property name="storePath" value="./data/graph/" />
<property name="serverPort" value="8080" />

<service id="gtfs" class="ba.service.GtfsService" />
<service id="trips" class="ba.service.TripService" />
<service id="deviation" class="ba.service.DeviationService" />
<service id="strf" class="ba.service.STRFService" />
<service id="predictions" class="ba.service.PredictionService" />
```

Abbildung 4.5: Stream und Services

4.3.2 Services

Der `GTFS-Service` stellt die Informationen über die Haltestellen und Fahrtzeiten bereit. Er ließt die Daten aus dem GTFS-Feed ein, sortiert die Abfahrtszeiten nach Fahrten und speichert die sortierten Abfahrtszeiten unter der Fahrt in einer `HashMap`. Dasselbe geschieht für Fahrten und Routen, um später einen performanteren Zugriff auf diese Daten zu ermöglichen. Über diesen Service kann weiterhin auf die GPS-Koordinaten der Haltestellen und die Zonen rund um die End- bzw. Starthaltestellen zugegriffen werden.
Der `Trip-Service` verwaltet mehrere Mengen von Trips. In ihm werden mögliche in Kürze startende, aktive und bereits vergebene Trips gespeichert. Die möglichen Trips werden benötigt, um startende Trips zu erkennen, die Aktiven, um einen GTFS-Realtime-Feed in Abhängigkeit der aktuellen fahrenden Trips zu erstellen und die benutzten Trips werden

dazu benutzt, dass keine TripId doppelt verteilt wird.
Der `Deviation-Service` merkt sich die beobachteten Verspätungen, welche im Laufe des Prozesses ermittelt werden. Die Verspätungen werden als Klasse `Deviation` modelliert. Eine Instanz dieser Klasse besitzt Einträge für n Knoten bei m Zeitschichten. Für jedes STRF wird eine Klasse `Deviation` in einer `HashMap` unter der StrfId gespeichert. Diese werden später dafür genutzt, um in den STRF's die beobachteten Variablen zu setzen, dass genauere Prognosen erzeugt werden können.
Der `STRF-Service` verwaltet die Modelle und stellt Funktionen auf ihnen bereit. Die Modellimplementierung[7] liegt nach [15] in C++ vor. Um eine Anbindung an die Modelle in Java zu ermöglichen, wurde eine Shared-Library `libstrf.so` mit dem Java-Native-Interface[8] erzeugt. Im späteren Verlauf der Arbeit wurde auch eine *Cuda*[9]-Version dieser Bibliothek implementiert, welche den Inferenz-Algorithmus auf einer GPU durchführt. Dies brachte einen deutlichen Geschwindigkeitsvorteil mit sich. Diese Library wird von der Klasse `STRFInterface` geladen, von welcher der `STRF-Service` eine Instanz besitzt. Die Modelle werden durch native Funktionsaufrufe im RAM erzeugt und geben ihre Speicheradresse als Rückgabewert zurück. Diese Speicheradressen werden wiederum in einer `HashMap` unter der StrfId gespeichert, um weitere Funktionsaufrufe auf den Modellen zu ermöglichen. Der Service stellt Funktionen zur Einlesung von Trainings- und Testdaten, Modelloptimierung, Setzen und Vergessen von beobachteten Knoten, Laden und Speichern von Modellen sowie der Vorhersage der wahrscheinlichsten Zustände bereit.
In dem `Prediction-Service` wird pro Modell eine Klasse `Prediction` gehalten, welche wiederum n Knoten bei m Zeitschichten besitzt. Ein Eintrag in dieser Klasse repräsentiert die Vorhersage der Verspätung an der spezifischen Haltestelle zu einer bestimmten Fahrt.

4.3.3 Initialisierung

In Abbildung 4.6 wird der erste richtige Schritt des Prozesses dargestellt. Der `OnStart`-Prozessor wird nur einmalig im Prozess ausgeführt. Innerhalb dieses werden zu Beginn die Modelle mit Hilfe des `STRF-Servies` durch den `InitSTRF`-Prozessor erzeugt. Falls die Modellparameter bereits gelernt wurden, können sie aus einer Datei eingelesen werden. Andernfalls werden für jedes Modell die jeweils zugehörigen Trainingsdaten eingelesen, aus diesen die empirischen Wahrscheinlichkeiten berechnet und dem Modell übergeben. Zusätzlich kann hier spezifiziert werden, ob die Modell nach Beendigung des Prozesses gespeichert werden sollen. Der kommende `OptimizeModel`-Prozessor trainiert die Modellparameter. Hier ist es möglich, eine maximale Anzahl an Iterationen für die Optimierung zu bestimmen sowie ein Epsilon, welches als Konvergenzkriterium ϵ für die Differenz der

[7] `https://bitbucket.org/np84/px` (zuletzt besucht am 02.10.2010)
[8] `https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/jniTOC.html` (zuletzt besucht am 02.10.2010)
[9] `http://www.nvidia.com/object/cuda_home_new.html` (zuletzt besucht am 02.10.2010)

```
<OnStart>
<ba.processor.strf.InitSTRF strfService="strf"
createSTRF="true" storeSTRF="true" modelPath="${storePath}" />
<ba.processor.strf.OptimizeModel
strfService="strf" iterations="10000" epsilon="0.00001" excecute="false" />
<ba.processor.server.StartServer iP="${serverIp}"
port="${serverPort}" baseUrl="delay" />
</OnStart>
```

Abbildung 4.6: Initialisierung

Likelihood des alten Optimierungsschrittes im Vergleich zu der des nächsten Schrittes gilt. Außerdem kann über die Variable *excecute* geregelt werden, ob dieser Prozessor ausgeführt werden soll oder nicht, z.B. für den Fall, dass die Modell bereits trainiert und nun nur geladen werden müssen. Anschließend wird ein Grizzly-Webserver gestartet, welcher die Vorhersagen per REST-API im GTFS-Realtime-Format bereitstellt. Die Vorhersagen werden aus einer Datei eingelesen, welche in einem in Sektion 4.3.8 beschriebenem Prozessor erstellt wird.

4.3.4 Datenaufbereitung

Von nun an werden die eigentlichen Dataitems verarbeitet. Zu Beginn müssen die Daten aufbereitet werden. Dazu werden unnötige Attribute, wie die Informationen, ob es sich um ein Niederflurfahrzeug handelt, und der Status, welcher immer RUNNING ist, entfernt. Die Values der Dataitems werden in ihre entsprechenden Datentypen in Java überführt und es werden führende Nullen bei Zahlen, sowie Leerzeichen um Strings entfernt. Außerdem werden Dataitems mit ungültigen Daten (z.B kein gültiges Datum / GPS liegt außerhalb von Warschau) aussortiert.

```
<RemoveKeys keys="Status,LowFloor" />
<WithKeys keys="Lon,Lat">
        <stream.parser.ParseDouble />
</WithKeys>
<WithKeys keys="Lines,Brigade,FirstLine">
        <ba.processor.clean.TrimAttribute />
</WithKeys>
<WithKeys keys="Brigade">
        <ba.processor.clean.RemoveLeadingZero />
</WithKeys>
<ba.processor.clean.FormatDate />
<ba.processor.clean.FilterGPS />
```

Abbildung 4.7: Aufbereitung der Daten

4.3.5 TripMatching

Um die Abweichungen ermitteln zu können, müssen die Dataitems zunächst ihren entsprechenden Trips aus dem GTFS-Feed zugeordnet werden. Diese Aufgabe wird von dem `MapTripId`-Prozessor übernommen. Sobald ein Dataitem bei diesem Prozessor ankommt, wird überprüft, ob es sich in einer *Baselocation* befindet. Bei einer Baselocation handelt es sich um eine Zone um die End- bzw. Starthaltestelle einer Route. Tritt der Fall ein, dass sich ein Fahrzeug in einer Baselocation befindet, wird zuerst überprüft, ob dieses Fahrzeug einen aktiven Trip beendet. Ist dies nicht der Fall, wird davon ausgegangen, dass in Kürze ein neuer Trip starten wird und es wird ein möglicher startender Trip angelegt. Dieser neu angelegte Trip beinhaltet das Datum, wann das Fahrzeug gesehen wurde, die FahrzeugId (zusammengesetzt aus "*Line_Brigade*") und eine Menge von möglichen Trips aus dem GTFS innerhalb eines 30-minütigen Intervalls. Ist ein Fahrzeug weder in seiner Start- noch in der Endhaltestelle, wird überprüft, ob zu diesem Fahrzeug ein möglicher startender Trip existiert. Tritt dieser Fall ein wird überprüft, ob das Fahrzeug seine Baselocation verlassen hat, indem überprüft wird, ob seine Distanz zu der Haltestelle größer als 100 Meter, aber gleichzeitig kleiner als 2 Kilometer ist (sonst würde die Möglichkeit bestehen, dass dieses Fahrzeug durch ein anderes ersetzt wurde und seine ID übernommen wurde). Sollte das

```
<!-- Maps a dataitem to a trip. -->
<ba.processor.tripmapper.MapTripId
        gtfsService="gtfs" tripService="trips" minDistanceThreshold="100"
        maxDistanceThreshold="2000" />
<!-- Computes deviations and sets observed vars in strf -->
<ba.processor.deviations.ComputeDeviations
        gtfsService="gtfs" deviationService="deviation" strfService="strf"
        tripService="trips" distanceAroundStop="150" />
```

Abbildung 4.8: Zuordnen der TripId und Ermittlung der Verspätung

Fahrzeug die Baselocation verlassen haben, wird ein neuer aktiver Trip im Tripservice angelegt. Ein aktiver Trip besitzt Informationen zu der Linie, FahrzeugId, dem gefahrenen Trip, dem zuletzt passierten Stop, sowie dem Datum seiner Erstellung und des letzten StopUpdates. Als Trip wird der Trip aus dem GTFS genommen, welcher zeitlich am nächsten für diese Linie von der aktuellen Baselocation aus startet, unter Berücksichtigung der bereits vergebenen Trips. Andernfalls wird geschaut, ob in den aktiven Trips bereits ein Fahrzeug mit der selben Id fährt und dessen TripId wird als TripId genutzt. Sollte kein Fall eintreffen, wird das Dataitem verworfen, da man keine Informationen zu einem (möglicherweise bald startendem) Trip besitzt und so keine TripID vergeben kann, anhand welcher die Verspätungen ermittelt werden könnten.

4.3.6 Verspätungsermittlung

Mit Wissen der TripId des Fahrzeuges kann überprüft werden, ob ein Fahrzeug eine Haltestelle passiert hat und ob es Verspätung hatte. Diese Aufgabe übernimmt der `ComputeDeviations`-Prozessor. Dieser Prozessor überprüft für eintreffende Dataitems, ob ihnen eine TripId zugeordnet wurde. Wenn unter dem Key „TripId“ eine gültige TripId aus dem GTFS-Feed steht, wird überprüft ob das Fahrzeug eine noch kommende Haltestelle passiert hat. Dies geschieht in zwei Schritten. Zuerst muss sich ein Fahrzeug in einem Radius von weniger als 150 Meter um die Haltestelle befinden und zweitens muss das Fahrzeug die Zone wieder verlassen, um eine Haltestelle passiert zu haben. Die Idee ist inspiriert aus dem Paper Mobility Stream Pattern Matching [4], wo lokale Events erkannt und global gesammelt werden. Andere Event-Detecition-Framworks werden in Solve Large Scale Learning [18] vorgestellt und bewertet. Besitzt das Fahrzeug eine gültige TripID, wird der TripService nach seinem zugehörigen aktiven Trip gefragt. Mit Hilfe des aktiven Trips können auf Basis der zuletzt passierten Haltestelle die noch zu passierenden Haltestellen ermittelt werden. Zuerst wird geprüft, ob zu der aktuellen Fahrt ein `StopEvent` existiert. Ein StopEvent repräsentiert den Fall, dass sich ein Fahrzeug in einem Radius von 150 Meter um eine Haltestelle befunden hat, und enthält Informationen über die Trip- und FahrzeugId, dir Haltestelle sowie die Uhrzeit, zu der es dort gesehen wurde. Existiert ein solches `StopEvent` wird überprüft ob sich das Fahrzeug nicht mehr in der 150 Meter Zone befindet. In diesem Fall wurde die Haltestelle passiert und ein `StopEvent` zurückgegeben. Um die `StopEvent`‘s zu erkennen, werden auf Basis der zuletzt passierten Haltestelle die noch kommenden Haltestellen vom GtfsService abgefragt. Für jede dieser Haltestellen wird eine `BoundingArea` erzeugt, welche die 150 Meter-Zone repräsentiert. Anschließend wird für jeden Stop überprüft, ob die aktuelle GPS-Koordinate des Dataitems innerhalb der zugehörigen `BoundingArea` liegt. Sollte eine Koordinate in einer dieser Zonen liegen, wird zu dem spezifischen Stop ein Stopevent generiert, welches von einem der nächsten eintreffenden Dataitems benutzt werden kann, um die Passierung des Stops zu erkennen. Handelt es sich um die letzte Haltestelle eines Trips, wird direkt das StopEvent zurückgegeben, da wenn das Fahrzeug die Zone wieder verlässt, es unter einer neuen TripId zurück in seine alte Richtung fährt und somit die Verspätung für einen falschen Trip aufgenommen würde. Sobald festgestellt wurde, dass eine Fahrstelle passiert wurde, kann die Abweichung vom Fahrplan berechnet werden. Jedoch wird zuerst der zuletzt passierte Stop des aktiven Trips aktualisiert. Daraufhin wird der GtfsService befragt, wann das Fahrzeug an dem passierten Stop nach Fahrplan halten sollte. Diese Zeit wird anschließend mit der Zeit des StopEvents verglichen und so wird die Abweichung als Differenz in Minuten zwischen den beiden Zeiten errechnet. Diese Differenz repräsentiert die Verspätung bzw. Verfrühung der jeweiligen Fahrt an der jeweiligen Haltestelle. Sie wird zum einen in dem DeviationService gespeichert, welcher seine Beobachtungen einmal die Stunde auf der Festplatte sichert. Somit

werden weitere Trainings- bzw. Testdaten generiert. Außerdem wird die jeweilige Beobachtung in seinen entsprechenden Zustand des STRF-Modells überführt und im Modell als beobachtet gesetzt. Wenn z.B. eine Verspätung von 3 Minuten beobachtet wird und das STRF 3 Zustände besitzt ($0 \rightarrow (-\infty,0)$, $1 \rightarrow [0,0]$, $2 \rightarrow (0,\infty)$) würde der StrfService den Knoten $(stop, trip)$ auf den beobachteten Zustand 2 setzen. Diese Intervalle wurden in den Experimenten an die entsprechenden Zustände angepasst.

4.3.7 Update und Übernahme der Vorhersagen

Um neue Vorhersagen zu generieren, wird der Max-Produkt-Algorithmus aus Kapitel 3.5.3 aufgerufen. Die Vorhersagen sollen, um den Spezifikationen des GTFS-Realtime-Feedes gerecht zu werden, einmal die Minute erneuert werden, wozu neue Vorhersagen benötigt werden. Der `OnMinuteChange`-Prozessor erkennt den Wechsel von einer auf die andere Minute und führt, wenn dieses Event eintrifft, die inneren Prozessoren aus. Von dem `STRFService` werden sich die STRF's gemerkt, in denen seit dem letzten Max-Produkt-Aufruf eine Variable gesetzt wurde. Für diese STRF's werden neue Vorhersagen von dem `UpdatePredictions`-Prozessor berechnet. Somit braucht der Max-Produkt-Algorithmus nur auf den STRF's aufgerufen werden, wo sich die MAP-Konfiguration aufgrund von neu beobachteten Werten ändert. Anschließend werden im `GetPredictions`-Prozessor die

```
<OnMinuteChange>
        <ba.processor.strf.UpdatePredictions
                strfService="strf" />
        <ba.processor.strf.GetPredictions
                predictionService="predictions" strfService="strf" />
</OnMinuteChange>
```

Abbildung 4.9: Update und Übernahme der Predictions

STRF's nach ihren wahrscheinlichsten Zuständen gefragt und diese werden in den `PredictionService` übernommen. Dieser stellt die Vorhersagen für die Erzeugung des GTFS-Realtime-Feedes im nächsten Schritt bereit.

4.3.8 Erzeugung GTFS-Realtime

Nachdem neue Vorhersagen von dem Modell bereit gestellt wurden, müssen diese in das GTFS-Realtime-Format übersetzt werden, um von Routingprogrammen, wie z.B. OpenTripPlanner verwendet werden zu können. Nach Definition des GTFS-Realtime-Feedes muss dieser jede Minute aktualisiert werden. Dies wird mit Hilfe des `OnMinuteChange`-Prozessors erreicht. Die inneren `UpdateRTFeed`-Prozessoren konstruieren den eigentlichen Realtime-Feed. Alle in 4.10 abgebildeten Prozessoren ähneln sich sehr und unterscheiden sich nur in der Menge an Trips, welche in den Feed miteinbezogen werden. Der Feed wird

```
<!-- once per minute the gtfs realtime feed is updated -->
<OnMinuteChange>
<ba.processor.gtfsrt.UpdateRTFeed
        predictionService="predictions" gtfsService="gtfs" />
<ba.processor.gtfsrt.UpdateCurrentRTFeed
        predictionService="predictions" tripService="trips" gtfsService="gtfs" />
<ba.processor.gtfsrt.UpdateAllTripsRTFeed
        predictionService="predictions" gtfsService="gtfs" />
<ba.processor.gtfsrt.UpdatePredRTFeed
        predictionService="predictions" gtfsService="gtfs" />
<ba.processor.gtfsrt.UpdateLineGtfsRt
        predictionService="predictions" line="33" gtfsService="gtfs" />
<ba.processor.gtfsrt.UpdateLineGtfsRt
        predictionService="predictions" line="26" gtfsService="gtfs" />
</OnMinuteChange>
```

Abbildung 4.10: Erzeugen eines neuen GTFS-RT-Feedes

erstellt, in dem über alle Trips aus der jeweils spezifizierten Menge iteriert wird und zu jeder Stopzeit jedes Trips die jeweilige vorhergesagte Verspätung in das entsprechende StopTimeUpdate eingefügt werden. Sobald alle Trips abgearbeitet wurden, wird der Feed im Protocol-Buffer-Format[10] gespeichert. Das Protocol-Buffer-Format ist ein Datenformat zur Serialisierung und sehr effizient, da es die Inhalte binär speichert. Um Daten im Protocol-Buffer-Format zu speichern, wird eine *.proto*-Datei benötigt. Diese ist ein Schema, wie die Daten zusammengesetzt und anschließend kompiliert werden. Der `AfterDay`-Prozessor

Algorithmus 1 Generierung von GTFS-Realtime

```
1:  procedure GENERATE GTFS-REALTIME
2:      f = generateFeedAndHeader()
3:      for trip t ∈ T do
4:          tu = createTripupdate()
5:          for st ∈ stoptimes(t) do
6:              d = delay(st)
7:              tu.putDelay(st,d)
8:              f.add(tu)
9:          end for
10:     end for
11:     safeFeedToFile(f)
12: end procedure
```

erkennt den Zeitpunkt, an dem die letzte Fahrt des Tages abgeschlossen ist. 30 Minuten später wird er ausgeführt. Seine inneren Prozessoren werden dazu genutzt, den Zustand des Programmes auf denselben Zustand zu setzen wie vor Programmstart, also es wer-

[10] `https://developers.google.com/protocol-buffers/` (zuletzt besucht am 02.10.2010)

```
<AfterDay gtfsService="gtfs">
        <!-- Resets observations in strf-model -->
        <ba.processor.strf.ResetObservations
                strfService="strf" />
        <!-- Resets the observations / pred / trips for the day -->
        <ba.processor.ResetServices tripservice="trips"
                deviationService="deviation" predictionService="predictions" />
</AfterDay>
```

Abbildung 4.11: Vergessen der Beobachtungen

den die beobachteten Werte sowohl im STRF als auch im `DeviationService` auf ihre Ausgangswerte (unbeobachtet) zurückgestzt. Darüber hinaus werden die Vorhersagen des `PredictionService` zurückgesetzt und morgens die wahrscheinlichsten Zustände ohne Beobachtungen übernommen und anschließend Stück für Stück wieder unter Beobachtungen optimiert.

4.4 Visualisierung der Ergebnisse

Zur Visualisierung der Ergebnisse wurde eine Webseite mit dem Cordova-Framework[11] programmiert. Das Framework ermöglicht es, plattformunabhängige mobile Applikationen und Webseiten in HTML, CSS und Javascript zu entwickeln und diese in native Applikationen zu überführen.

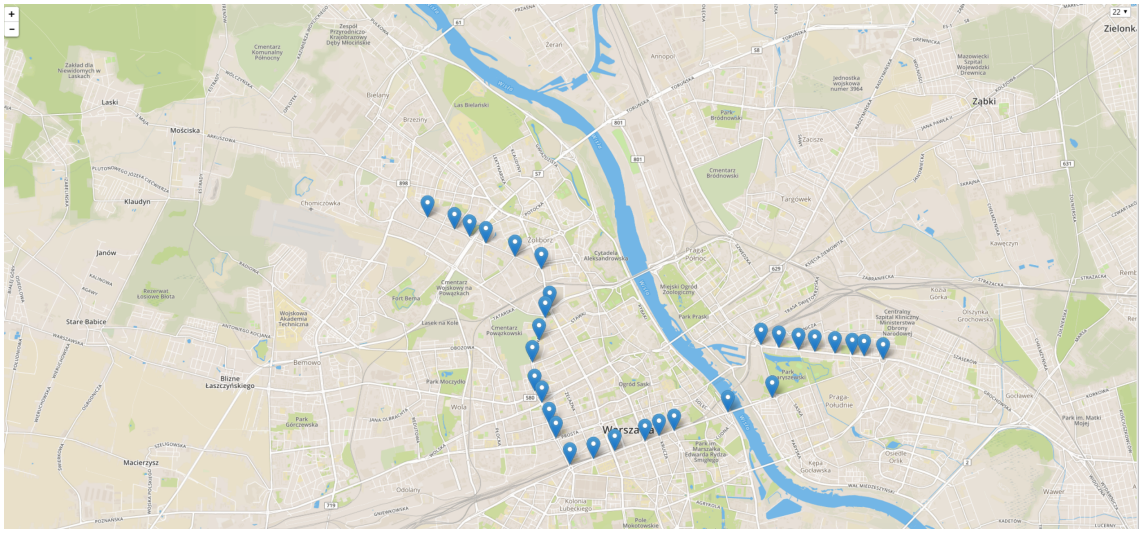


Abbildung 4.12: Haltestellen der Linie 22

Die Abbildung 4.12 zeigt die Startseite der Webseite. Zu sehen ist eine Karte, die, falls der Benutzer seinen Standort freigibt, auf den Nutzer fokussiert ist, andernfalls auf das Stadt-

[11] `https://cordova.apache.org/` (zuletzt besucht am 02.10.2010)

zentrum von Warschau. Die Karte wird von OpenStreetMap[12] bereitgestellt und durch das Leaflet-Framework[13] in die Seite eingebunden. In dem Select-Element oben rechts kann der Benutzer die Linie aussuchen, für die er die Verspätungen sehen möchte. Standardmäßig ist die Linie mit der niedrigsten Nummer ausgewählt. Sobald eine neue Linie ausgewählt wird, werden die Stops der alten Linie entfernt und die Haltestellen der neu ausgewählten Linie gerendert.

Durch einen Klick auf einen Marker öffnet sich ein Verspätungsgraph für die jeweilige Haltestelle. Dort werden für alle noch kommenden Fahrten des Tages die Verspätung der jeweiligen Fahrt an der ausgewählten Haltestelle in Minuten aufgelistet. Die Abbildung 4.13 zeigt einen solchen Graphen. Die dazu genutzten Daten werden von dem Server, welcher während der Initialisierung des Streams-Prozesses gestartet wird, im GTFS-Realtime-Format abgegriffen. Zur Dekodierung des binären Feedes wird die Bibliothek GTFS-Realtime-Bindings[14] genutzt. Sobald die Seite fertig geladen ist, werden die Daten geholt und linienweise aufbereitet.

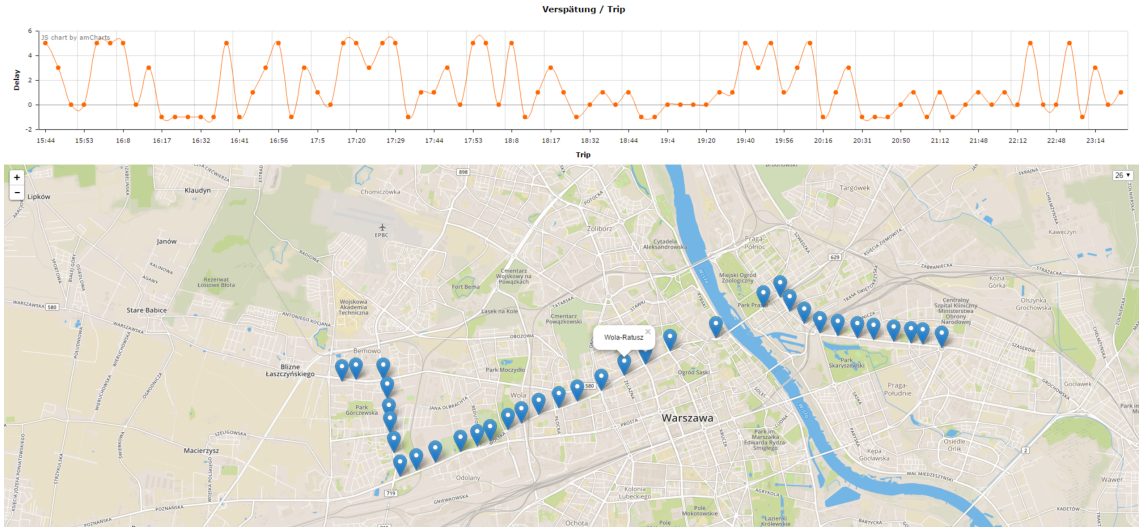


Abbildung 4.13: Verspätungsgraph einer Haltestelle

[12] `http://www.openstreetmap.org/` (zuletzt besucht am 02.10.2010)
[13] `http://leafletjs.com/` (zuletzt besucht am 02.10.2010)
[14] `https://github.com/google/gtfs-realtime-bindings` (zuletzt besucht am 02.10.2010)

Kapitel 5

Experimente

In diesem Kapitel werden die durchgeführten Experimente präsentiert. Vorher werden Maße zur Modellperformance sowie die genutzte Methode zur Ermittlung der Maße erläutert.

5.1 Methode zur Auswertung

Die Versuche wurden unter Verwendung einer Konfusionsmatrix ausgewertet. Da es sich um ein zeitlich-räumliches Modell handelt, wurden die einzelnen Zeitschichten nacheinander unter Beobachtung und Inferenz der jeweils vorherigen Zeitschichten ausgewertet. Der Algorithmus dazu wird in Algorithmus 2 vorgestellt. Benötigt wurden zur Auswertung das trainierte Modell sowie eine Liste von Testdaten. Jedes Element dieser Liste ist eine Matrix, deren Werte die Verspätungszustände an den einzelnen Haltestellen eines Trips repräsentieren. Zu Beginn wird in Zeile 3 das STRF nach seinen wahrscheinlichsten Zuständen gefragt. Anschließend wird über alle Tage, Fahrten und deren Stops iteriert. Zu jedem Stop wird die Wahrheit aus den Testdaten (Zeile 7) und der vorhergesagte Zustand aus den mls ermittelt. Diese werden daraufhin in Zeile 9 in der Konfusionsmatrix gesetzt und die Wahrheit wird dem STRF als beobachtete Variable gesetzt. Sobald eine Fahrt komplett abgearbeitet wurde, wird durch den Maximum-Product-Algorithmus eine neue MAP-Konfiguration ermittelt und als *mls* gesetzt. Diese Schritte werden solange wiederholt, bis alle Fahrten eines Tages abgearbeitet wurden. Nachdem ein Tag vergangen ist, werden die Beobachtungen zurückgesetzt und die initial am wahrscheinlichsten Werte als *mls* für den neuen Tag gesetzt. Sobald alle Elemente der Liste L gesehen wurden, werden die Ergebnisse der Matrix in eine Datei geschrieben. In dieser stehen Werte wie die *Accuracy* und *Precision* des Modells, welche im nächste Kapitel genauer erläutert werden.

5.1.1 Maße zur Modellperformance

Zur Evaluation der Modellqualität können unterschiedliche Werte herangezogen werden. Die möglichen Werte sind *accuracy*, *precision*, *recall* oder die $F - Measure$. Um die-

Algorithmus 2 Evaluierung eines STRF

```
1: procedure Evaluierung eines STRF
2:     input: L = Liste mit Testdaten, S = STRF-Modell
3:     mls = S.getMostLikliestStates()
4:     for l ∈ L do
5:         for trip ∈ l do
6:             for stop ∈ trip do
7:                 truth = l[trip][stop];
8:                 pred = mls[trip][stop];
9:                 cMatrix.add(truth,pred)
10:                S.observe(trip,stop,truth)
11:            end for
12:            S.maxProduct(S.getEdges())
13:            mls = S.getMostLikliestStates()
14:        end for
15:        S.resetObservation()
16:        S.maxProduct(S.getEdges())
17:        mls = S.getMostLikliestStates()
18:    end for
19:    cMatrix.evaluate(eval-file)
20: end procedure
```

se Werte zu berechnen, müssen anhand der Testdaten die $true-positives$(tn), $false-negatives$(fn), $false-positives$(fp) und die $true-negatives$(tn) ermittelt werden. Anhand dieser Werte können die Maße durch folgende Formeln errechnet werden [17] :

$$accuracy = \frac{tp + tn}{tp + fp + + fn + tn} \tag{5.1}$$

$$precision = \frac{tp}{tp + fp} \tag{5.2}$$

$$recall = \frac{tp}{tp + fn} \tag{5.3}$$

$$F - measure = \frac{(\beta^2 + 1) * precision * recall}{\beta^2 * precision + recall} \tag{5.4}$$

5.2 Experimente

In den Experimenten wurde die Vorhersage von Verspätungen anhand von Spatio-Temporal-Random-Fields untersucht. Jedes STRF ist ein Modell über die Verspätungen einer Stadtbahnlinie in eine Richtung. Als Basisgraph dient bei jedem STRF ein Kettengraph, der die

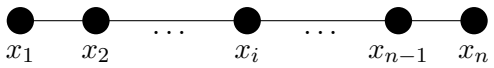


Abbildung 5.1: Kettengraph

Struktur der jeweiligen Haltestellen repräsentiert. In Abbildung 5.1 ist ein solcher Graph aus n Knoten zu sehen. Für die Anzahl der Layer wurde die Anzahl an Trips aus dem GTFS-Feed für diese bestimmte Strecke genommen.

5.2.1 Fünf Zustände

In den ersten Experimenten wurden zwei STRF's modelliert und diese getestet. Die beiden STRF's wurden für die Linie 26 (Wiatraczna → Os.Górczewska) und die Linie 33 (Metro Mlociny → Kielecka) erstellt. Für die Linie 26 ergab sich ein STRF mit einem Kettengraph von 33 Knoten, sowie 105 Zeitschichten. Der Graph der Linie 33 enthielt 26 Knoten und 165 Zeitschichten. Beide Modelle teilen sich dieselbe Zustandsmenge. Die Zustände sind von 0 bis 4 gelabelt und repräsentieren die folgenden Abweichungsintervalle :

- 0 → verfrüht
- 1 → pünktlich
- 2 → 1 - 2 Minuten zu spät
- 3 → 3 - 4 Minuten zu spät
- 4 → 5 + Minuten zu spät

Mit der in Kapitel 4 vorgestellten Architektur wurden die Verspätungen vom 20.6.2016-24.06.2016 und vom 27.06.2016 - 01.07.2016 aufgenommen. Für jeden Tag und jede Linie wurde eine Matrix der Größe $Stops * Trips$ generiert. Ein Eintrag in dieser Matrix stellt die Verspätung einer Fahrt an einer Haltestelle dar. Eine Spalte repräsentiert eine Fahrt.
Als Trainingsdaten dienten die Daten der Woche vom 20.6.2016 bis zum 24.06.2016. Aus diesen Daten wurden die empirischen Häufigkeiten der Zustandskombinationenen für jede Kante des STRF's ermittelt und dem Model als Vektor übergeben. Auf diesem wurden die Modelle mit dem in Kapitel 3.6 beschriebenen Verfahren optimiert. Als maximale Anzahl an Iterationen wurde 5000 gewählt und ϵ wurde auf 10^{-5} gesetzt.
Die gelernten Modelle wurden mit Hilfe einer Konfusionsmatrix und der Methode nach Algorithmus 2 ausgewertet. Als Testdaten dienten die Daten vom 27.06.2016 - 01.07.2016. Die Ergebnisse der Auswertung der beiden Modelle sind in den Tabellen 5.1 und 5.2 zu sehen.
Optimal wäre es, wenn das Maximum der i-ten Spalte in der i-ten Zeile liegt. Ein Blick auf die Konfusionsmatrix (Tabelle 5.1) der Linie 26 zeigt jedoch ungünstiges Verhalten. Für die Klasse 0 liegen eine große Menge an positiven Treffern vor, jedoch bleibt es bei dieser Klasse.

In allen anderen Klassen überwiegt die Menge an negativen Treffen. Insgesamt wurden von dem Modell fast nur Verfrühungen vorhergesagt. Ein Blick auf die Klasse 3 zeigt, dass diese nicht ein Mal von dem Spatio-Temporal-Random-Field vorhergesagt wurde. Im Vergleich mit der Konfusionsmatrix der Linie 33 (Tabelle 5.2) wird festgestellt, dass hier dasselbe Phänomen auftritt. Auch hier wurde die Klasse 3 nicht ein Mal vorhergesagt. Allerdings verhält sich dieses STRF ein wenig besser. Hier liegen die Maxima der Klassen $0, 1, 4$ auf der Diagonalen. Viele Werte der Klasse 1 wurden jedoch der Klasse 0 zugeordnet, so dass sich nur eine Precision von 56% ergibt. Die Klasse 4 kommt auch nur an eine Precision von 55%.

Tabelle 5.1: Konfusionsmatrix Linie 26

Pred / True	0	1	2	3	4	Precision
0	5273	75	2	0	16	0.98
1	1177	323	9	0	12	0.26
2	529	216	21	0	9	0.028
3	286	93	5	0	9	0. 0
4	998	161	3	0	271	0.23
Recall	0.63	0.32	0.59	0	0.89	

Tabelle 5.2: Konfusionsmatrix Linie 33

Pred / True	0	1	2	3	4	Precision
0	2575	502	25	0	18	0.82
1	898	1250	26	0	33	0.56
2	312	272	232	0	12	0.26
3	111	85	23	0	23	0
4	71	70	23	0	205	0.55
Recall	0.62	0.57	0.67	0	0.65	

5.2.2 Reduzierung der Zustände

Da sowohl das Modell der Linie 26 als auch das Modell der Linie 33 nicht zwischen der Klasse 3 und seinen benachbarten Klassen unterscheiden konnte, wurden die Klassen der Modelle geändert. In den Daten wurden daraufhin die Klassen 2 und 3 verschmolzen. Hierzu wurden zuerst alle Zustände der Klasse in die Klasse 2 überführt und anschließend wurde die Klasse 4 zu 3 abgeändert. Somit repräsentiert die Klasse 2 eine leichte bis mittlere Verspätung (1 - 4 Minuten Abweichung) und die Klasse 3 eine hohe Abweichung (5 +

Minuten). Ihre Struktur behielten die Modelle bei. Trainiert wurden sie nach dem selben Prinzip wie in 5.2.1.
Die Ergebnisse dieser Versuche sind in den beiden Tabellen 5.3 (Linie 33) und 5.4 (Linie 26) zu sehen. Dieses Mal wurde jede Klasse vorhergesagt, jedoch hat dies nicht viel bei der Güte der Modelle genutzt. In der Konfusionsmatrix der Linie 26 ist ein Overfitting an die Klasse 0 zu sehen. Dies bedeutet, dass für jede Klasse die Klasse 0 als der meist wahrscheinlichste Zustand vorhergesagt wurde. Auch bei der Linie 33 hat sich nich viel geändert. Die Precision der Klasse 0 ist zwar von 82% auf 90% gesprungen, jedoch brachte diese Verbesserung eine Verschlechterung der Klasse 1 um gut 25% mit sich. Klasse 3 blieb nach wie vor bei 55% und die neu zusammengelegte Klasse 2 erhielt eine Precision von 25%, welche fast identisch mit der Precision der Klasse 2 aus 5.2.1 ist (26%).

Tabelle 5.3: Konfusionsmatrix Linie 33 - 4 Zustände

Pred / True	0	1	2	3	Precision
0	2821	231	63	16	0.9
1	1435	686	61	46	0.31
2	711	136	601	47	0.25
3	94	26	45	205	0.55
Recall	0.56	0.64	0.64	0.65	

Tabelle 5.4: Konfusionsmatrix Linie 26 - 4 Zustände

Pred / True	0	1	2	3	Precision
0	5427	16	5	20	0.99
1	1448	138	26	10	0.08
2	1006	58	113	12	0.09
3	1129	11	31	363	0.24
Recall	0.6	0.62	0.64	0.89	

5.2.3 Mischung der Daten

Da sowohl die ersten Ergebnisse aus Sektion 5.2.1 als auch 5.2.2 nicht zufriedenstellend waren, wurden die Ursachen untersucht. Möglicherweise lagen dem Modell beim Training zu wenig Daten vor um Rückschlüsse auf Zusammenhänge zu ziehen oder die Verteilung der Zustände über die Daten der zwei Wochen war zu unterschiedlich.
Deshalb wurden dieses Mal zwei Änderungen an den Daten vorgenommen. Zum einen wurden anteilig mehr Daten als Trainingsdaten verwendet und die Daten wurden nicht mehr

zeitabhängig in Test- und Trainingsdaten aufgeteilt. Die Verteilung wurde jetzt von einem Skript übernommen, welches die Daten aus einem Ordner in eine Liste einließt, diese Liste mischt und in einem gegebenen Verhältnis in die entsprechenden Ordner für Test- und Trainingsdaten kopiert. Als Verhältnis wurde für diese Versuche 60% Trainings- und 40% Testdaten gewählt. Die räumliche und zeitliche Struktur der vorherigen Versuche wurde beibehalten. Ebenso das Lernverfahren und die Parameter. Diese Experimente wurden sowohl für die Linie 26 als auch die Linie 33 mit 4 und 5 Zuständen durchgeführt. Die Konfusionmatrizen für die Linie 33 befinden sich in den Tabellen 5.5 und 5.6; für Linie 26 in 5.7 und 5.8.
Bei dem Versuch, die Vorhersagen durch Mischung der Test- und Trainingsdaten zu verbessern, konnte für die Linie 33 bei 4 Zuständen keine Verbesserung erreicht werden. Die Precision der Klasse 2 ist zwar um 4% gestiegen. Dies ging jedoch mit einer Verschlechterung der Zustände 0, 1, 3 um bis zu 30% der Klasse 3 einher.

Tabelle 5.5: Konfusionsmatrix Linie 33 - 4 Zustände - gemischt

Pred / True	0	1	2	3	Precision
0	2469	360	30	4	0.86
1	1505	829	45	20	0.35
2	772	277	231	23	0.17
3	299	55	26	97	0.20
Recall	0.49	0.54	0.69	0.67	

In der Tabelle 5.6 sind die Ergebnisse des STRF's der Linie 33 mit 5 Zuständen zu sehen. Die Precision der Klasse 1 ist hier um 20% gestiegen, die der Klasse 0 jedoch um 13% gesunken. Bei den anderen Klassen gab es minimale Verbesserungen der Precision um 1%. Insgesamt ist dieses Modell besser zu bewerten, da sich bei 4 der 5 Klassen das Maximum auf der Diagonalen befindet.

Tabelle 5.6: Konfusionsmatrix Linie 33 - 5 Zustände - gemischt

Pred / True	0	1	2	3	4	Precision
0	1983	829	34	0	17	0.69
1	471	1836	53	0	39	0.76
2	285	410	266	0	22	0.27
3	114	120	35	4	47	0.01
4	87	95	29	0	266	0.55
Recall	0.67	0.55	0.63	1	0.68	

Die Tabelle 5.7 repräsentiert die Auswertung des Modells der Linie 26 für 5 Zustände. Dieses Modell liefert schlechte Werte für die Performance. Dies kann auf eine Überanpassung des Modells zurückgeführt werden. Ein Blick auf die Vorhersagen zeigt, dass fast nur der Zustand 0 bzw. 4 vorhergesagt wurde.

Tabelle 5.7: Konfusionsmatrix Linie 26 - 5 Zustände - gemischt

Pred / True	0	1	2	3	4	Precision
0	1635	0	6	7	1693	0.49
1	65	0	2	0	574	0
2	281	0	42	0	409	0.05
3	574	0	2	50	587	0.04
4	469	0	0	8	3581	0.88
Recall	0.54	0	0.8	0.77	0.52	

Auch für die Linie 26 gab es mit anderen Daten keine Verbesserungen. Precision und Recall waren beide niedrig und das Problem mit dem Zustand 2 bestand weiterhin.

Tabelle 5.8: Konfusionsmatrix Linie 26 - 4 Zustände - gemischt

Pred / True	0	1	2	3	Precision
0	1109	319	0	1	0.77
1	154	189	0	0	0.55
2	1448	4173	0	50	0.0
3	2021	2245	0	127	0.028
Recall	0.23	0.059	0	0.71	

Da die Ergebnisse aus Tabelle 5.6 im Vergleich zu den restlichen Versuchen gut aussahen, wurde probiert, diese noch einmal zu verbessern. Dazu wurde das Modell der Linie 33 mit 5 Zuständen und gemischten Daten mit einer höheren Anzahl an Iterationen gelernt. Das Ergebnis ist in Tabelle 5.9 zu sehen. Im Vergleich zu den alten Versuchen wurde für die Klassen 1,2,3 und 5 Verbesserungen der Precision und des Recalls im Bereich zwischen 4% und 7% gemessen.

Tabelle 5.9: Konfusionsmatrix Linie 33 - 5 Zustände - gemischt - weitere Optimierung

Pred / True	0	1	2	3	4	Precision
0	1458	431	14	0	12	0.76
1	268	1431	35	0	23	0.81
2	210	294	239	0	14	0.32
3	89	90	29	4	29	0.01
4	51	75	12	0	205	0.59
Recall	0.7	0.61	0.72	1	0.72	

5.2.4 Experimente auf neuen Daten

Da die Ergebnisse der bisherigen Versuche nicht sehr gut waren, wurde die Annahme untersucht, dass zu wenig Trainingsdaten genutzt wurden oder die Qualität dieser nicht ausreichend war. Aus diesem Grund wurden neue Daten aufgenommen, diesmal vom 18.08.2016 bis zum 17.09.2016. Die Daten aus den vorherigen Experimenten wurden nicht mitverwendet, da ein längerer Zeitraum zwischen der Aufzeichnung lag. Dieses Mal wurden die STRF's mit denselben fünf Zuständen wie in Kapitel 5.2.1 gelernt. Als Linien wurden die 7 (P+R Al.Krakowska → Kawęczyńska-Bazylika), 15 (P+R Al.Krakowska → Marymont-Potok), 22 (Piaski → Wiatraczna) und 35 (Wyścigi → Nowe Bemowo) gewählt.

Tabelle 5.10: Konfusionsmatrix Linie 7 - Neue Daten

Pred / True	0	1	2	3	4	Precision
0	6538	1314	165	0	167	0.79
1	729	436	44	0	42	0.35
2	612	192	95	0	53	0.1
3	426	46	69	0	56	0
4	183	7	29	0	100	0.31
Recall	0.77	0.22	0.23	0	0.24	

Alle dieser Konfusionsmatrizen ähneln sich sehr. Der Großteil der Werte befindet sich in der Klasse 0 - verfrüht. Dies ist jedoch die einzige Klasse, deren Precision über 30% liegt. Die Linien 7 und 15 weisen weiterhin das Problem auf, dass die Klasse 3 nicht einmal vorhergesagt wurde. Für die Linie 22 existiert dieses Problem für die Klasse 4. Dort wurde wie in Tabelle 5.12 zu sehen nur ein einziges Mal die Klasse 4 vorhergesagt.

Tabelle 5.11: Konfusionsmatrix Linie 15 - Neue Daten

Pred / True	0	1	2	3	4	Precision
0	8178	44	1	0	16	0.99
1	2319	127	0	0	6	0.05
2	1812	59	7	0	6	0.03
3	749	8	0	0	4	0
4	460	1	0	0	191	0.29
Recall	0.6	0.53	0.87	0	0.85	

Tabelle 5.12: Konfusionsmatrix Linie 22 - Neue Daten

Pred / True	0	1	2	3	4	Precision
0	5872	13	199	44	0	0.95
1	799	3	50	10	0	0.003
2	1008	0	118	27	1	0.1
3	1156	0	78	45	0	0.03
4	1055	0	9	2	0	0
Recall	0.59	0.18	0.26	0.35	0	

Tabelle 5.13: Konfusionsmatrix Linie 35 - Neue Daten

Pred / True	0	1	2	3	4	Precision
0	6186	43	57	165	233	0.92
1	1209	12	23	17	42	0.01
2	849	19	53	18	17	0.05
3	649	5	17	24	28	0.03
4	950	3	1	104	198	0.16
Recall	0.63	0.14	0.35	0.07	0.38	

5.2.5 Geänderte Zustandsaufteilung

Alle bisherigen Experimente wiesen die Gemeinsamkeit auf, dass der Großteil der Vorhersagen im Zustand 0 (zu früh) lag. Deshalb wurde dieser Zustand in zwei weitere aufgeteilt. Folgende Zustände ergaben sich durch die Anpassung:

- 0 → 5+ Minuten verfrüht
- 1 → 1 - 5 Minuten verfrüht
- 2 → pünktlich
- 3 → 1 - 4 Minuten verspätet
- 4 → 5+ Minuten verspätet

Es wurden erneut die neu aufgenommenen Daten aus Kapitel 5.2.4 genutzt und in diesen die Zustände an die neue Aufteilung angepasst. Bei diesen Experimenten wurden vier STRF's gelernt. Als Linien dienten die 15 und 22 aus dem vorherigen Kapitel 5.2.4 zum Vergleich und zusätzlich die Linien 18 (Żerań FSO → Służewiec) und 26 (Wiatraczna → Os.Górczewska).
Die folgende Tabelle 5.14 zeigt die Ergebnisse der Auswertung von Linie 15. Im Vergleich zu den Ergebnissen der selben Linie im vorherigen Kapitel hat sich einiges getan. Zum einen wurde jede Klasse vorhergesagt und die Precision verhält sich eindeutig besser. Hier liegt nur noch die Klasse 3 unter 50%, wohingegen es vorher 4 der 5 Klassen waren. Außerdem liegen bei diesem Modell die Maxima der Spalten auf der Diagonalen, jedoch liegen auch weiterhin einige Werte daneben.

Tabelle 5.14: Konfusionsmatrix Linie 15 - Neue Daten - 2 verfrühte Zustände

Pred / True	0	1	2	3	4	Precision
0	601	188	40	0	20	0.71
1	40	979	385	5	30	0.68
2	462	948	3307	40	803	0.59
3	180	128	320	197	432	0.15
4	426	252	479	120	1510	0.54
Recall	0.35	0.39	0.72	0.54	0.54	

Tabelle 5.15: Konfusionsmatrix Linie 18 - Neue Daten - 2 verfrühte Zustände

Pred / True	0	1	2	3	4	Precision
0	888	466	825	0	59	0.39
1	253	3319	1206	0	307	0.65
2	123	690	1461	2	549	0.51
3	39	289	242	0	93	0
4	39	330	401	0	232	0.23
Recall	0.66	0.65	0.35	0	0.18	

Tabelle 5.16: Konfusionsmatrix Linie 22 - Neue Daten - 2 verfrühte Zustände

Pred / True	0	1	2	3	4	Precision
0	998	46	157	0	16	0.82
1	180	474	640	0	28	0.36
2	176	211	1283	12	74	0.73
3	101	33	341	8	47	0.15
4	184	104	1527	20	756	0.29
Recall	0.61	0.54	0.32	0.2	0.82	

In Tabelle 5.15 sind die Ergebnisse für die Linie 18 zu sehen. Hier fällt wieder direkt das Problem auf, dass die Klasse 3 nur sehr selten vorhergesagt wurde. Auch die neue Klasse 0 erreicht nur 39% Precision und die Klasse 4 sogar nur 23%. Für die Linie 22 ergaben sich einige Verbesserungen. In den vorherigen Experimenten lag die Precision von 4 der 5 Klassen bei 10% oder weniger. Nun besitzt die schlechteste Klasse eine Precision von 15%. Die restlichen Klassen lagen zwischen 28% und 73%. Auch wenn dies immer noch keine wirklich guten Werte sind, ist es ein guter Fortschritt zu den vorherigen Auswertungen.

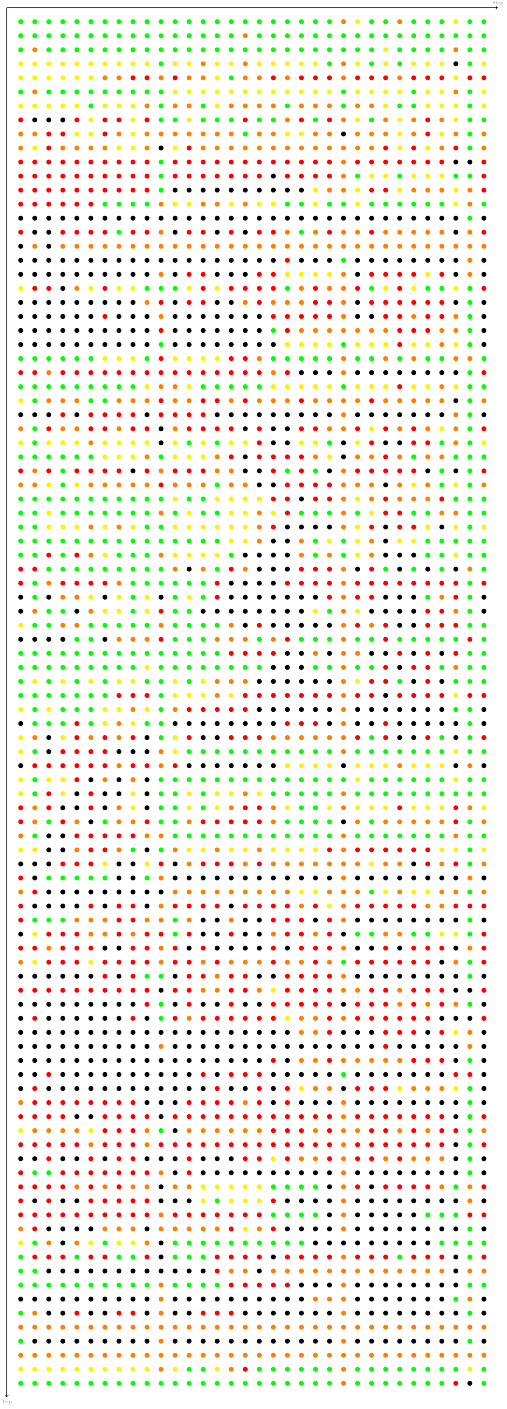
Tabelle 5.17: Konfusionsmatrix Linie 26 - Neue Daten - 2 verfrühte Zustände

Pred / True	0	1	2	3	4	Precision
0	614	233	531	0	109	0.41
1	2	641	1409	0	273	0.28
2	0	235	4510	1	1392	0.73
3	0	34	258	8	244	0.02
4	2	188	1894	6	2643	0.56
Recall	0.99	0.48	0.52	0.53	0.57	

Bei der Auswertung der Linie 26 fielen direkt die fehlenden Vorhersagen für den Zustand 3 auf. Dies führte zu einer Precision von nur 2% für diese Klasse. Auch die anderen Klassen waren nicht überzeugend. Alle außer der Klasse 2 lieferten nur Werte zwischen 28 und 56%. Die Klasse 2 konnte immerhin eine Precision von 73% aufweisen.
Auch diese Modelle waren nicht zufriedenstellend, jedoch ein Schritt in die richtige Richtung. Durch die Aufteilung der Klasse 0 konnte eine bessere Verteilung der Zustände zugunsten der restlichen Klassen erreicht werden.

5.3 Einfluss des Fahrplans

Da die Modelle jedoch weiterhin nicht brauchbar waren, wurde die Annahme getroffen, dass häufige Fahrplanwechsel und ein nicht aktueller Fahrplan Probleme bereiten. Diese Fahrplanwechsel wurden im September an drei Daten festgestellt, dem 05.09, 10.09 und 26.09. Durch eine Visualisierung des Fehlers pro Knoten sollte untersucht werden, ob Orte oder Zeiten existieren, für die es trotzdem funktioniert. Ein solches Diagramm zeigt alle Knoten eines STRF's und repräsentiert den Fehler pro Knoten als Accuracy. In Abbildung 5.3 ist die Bedeutung der einzelnen Farben dargestellt. Eine horizontale Linie an Knoten repräsentiert eine Fahrt, welche jeweils an dem linken Knoten startet. Die vertikale Achse repräsentiert die einzelnen Fahrten eines Tages, begonnen mit der ersten Fahrt am oberen Ende. Zu sehen ist, dass bei allen Modellen, außer dem aus Abb. 5.5, die ersten Fahrten gute Werte erreichen. Anschließend wird es eher durchwachsen und die Vorhersagen nehmen an Qualität ab. Zwischenzeitlich tauchen einige Fahrten auf, die gute Qualität aufweisen, jedoch nimmt diese mit zunehmender Fahrtdauer ab. Insgesamt spiegeln sich die schlechten der Konfusionsmatrizen auch in diesen Bildern wider, jedoch scheint das Verfahren trotzdem für frühe Trips und an einigen Orten zu funktionieren.



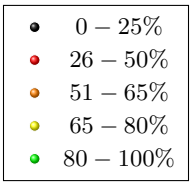


Abbildung 5.3: Accuracy pro Knoten

Abbildung 5.2: Fehler pro Knoten - Linie 15

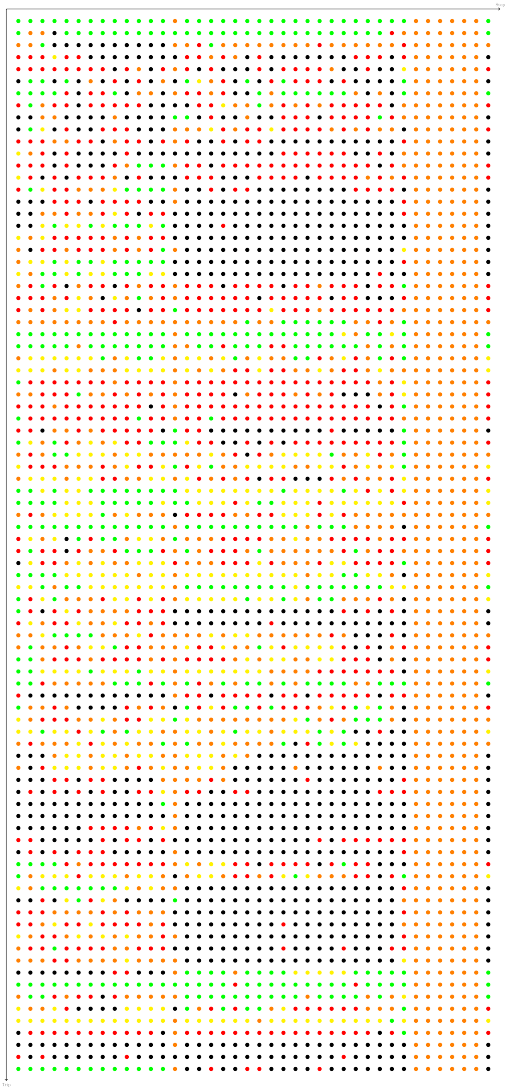


Abbildung 5.4: Fehler pro Knoten - Linie 18

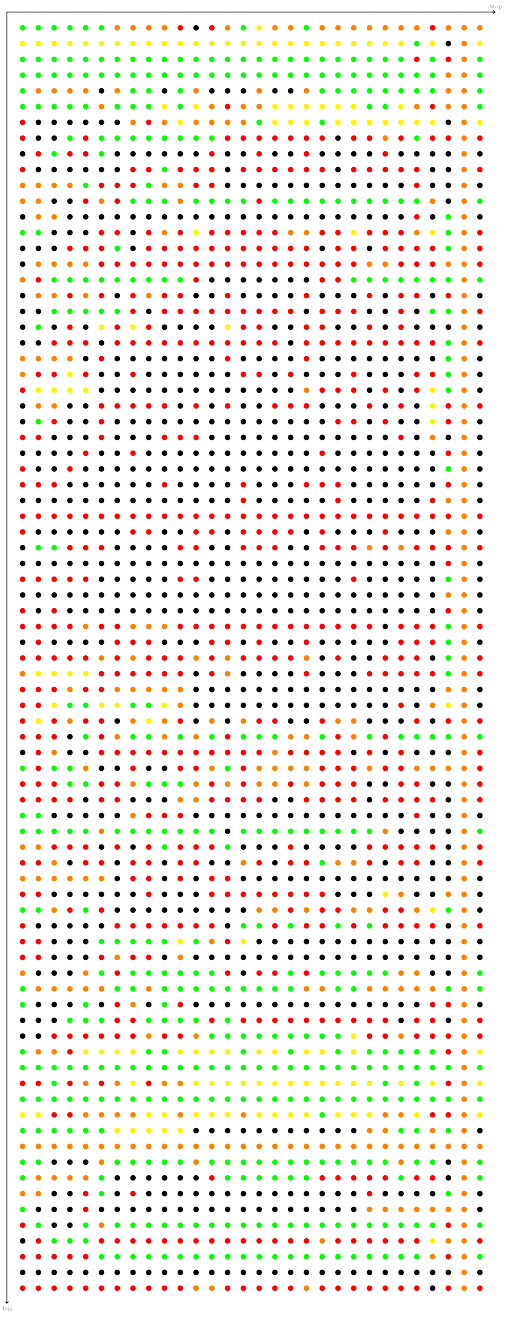


Abbildung 5.5: Fehler pro Knoten - Linie 22

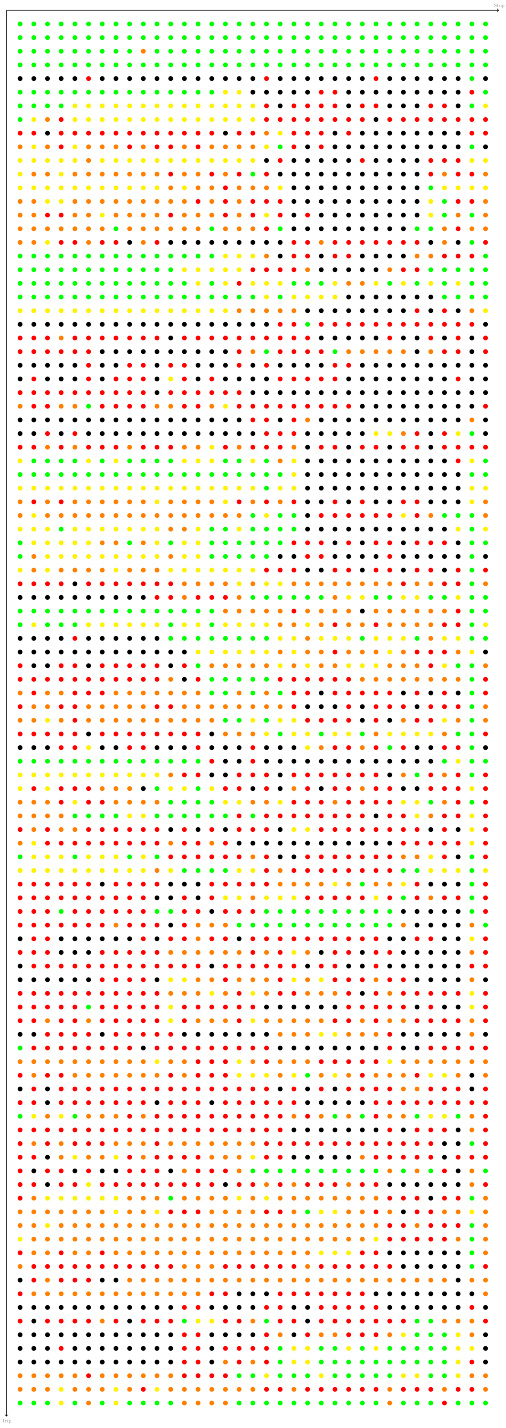


Abbildung 5.6: Fehler pro Knoten - Linie 26

5.4 Vergleich mit Gal

Um einen Vergleich zu bestehenden Arbeiten schaffen, wurde das bereits vorgestellte Verfahren der Queing-Theorie nach Gal [5, 16] implementiert. Als Netzwerk diente wie auch in den vorherigen Versuchen der Kettengraph der jeweiligen Linie. Der Algorithmus, nach dem dieses Verfahren ausgewertet wurde, wird in Algorithmus 3 präsentiert. Dort wird, wie auch in Algorithmus 5.1, über alle Datensätzen, deren Trips und Haltestellen iteriert. Um die Verspätung und Vorhersage an der Haltestelle i zu berechnen, wurde die Summe über die Reisezeit der einzelnen Segmente $(i, i-1)$ für alle $i > 0$ berechnet. In Zeile 10 wird die Reisezeit für ein Segment ermittelt.

Algorithmus 3 Evaluierung der Daten nach Gal [5, 16]

1: **procedure** Evaluierung der Daten nach Gal
2: **input**: L = Liste mit Daten
3: **for** $l \in L$ **do**
4: **for** $t \in l$ **do** ▷ Iteriere über die Trips
5: **for** $stop \in t$ **do** ▷ Iteriere über die Stops
6: $i \leftarrow stop.sequenceId$
7: $travelTime \leftarrow 0$
8: $travelTimePred \leftarrow 0$
9: **while** $i > 0$ **do** ▷ Prediction und Fahrtdauer für Segmente $(i, i-1)$
10: $segmentTime = arr(i,t) + d(i,t) - arr(i-1,t) + d(i-1,t)$
11: $predTime = arr(i,t) + lastSeen(i,t) - arr(i-1,t) + lastSeen(i,t)$
12: $travelTime \leftarrow travelTime + segmentTime$
13: $travelTimePred \leftarrow travelTimePred + predTime$
14: $i = i - 1$
15: **end while**
16: $truth = diff(arr(0,t) + travelTime, arr(stop,t))$ ▷ Verspätung
17: $truth = diff(arr(0,t) + travelTimePred, arr(stop,t))$
18: $cMatrix.add(truth, pred)$
19: **end for**
20: **end for**
21: **end for**
22: cMatrix.evaluate(eval-file)
23: **end procedure**

Die Funktion $arr(i,t)$ gibt die Ankunftszeit und $d(i,t)$ die Verspätung an der Haltestelle i für den Trip t zurück. Als Reisezeit für ein Segment wird die Differenz zwischen der Ankunftszeit plus der Verspätung an der i-ten Haltestelle für den t-ten Trip und der Ankunftszeit plus Verspätung an der $i-1$-ten Haltestelle berechnet. Um die Vorhersage zu

bestimmen, wird mit der Funktion $lastSeen(i, t)$ die zuletzt beobachtete Verspätung an der Haltestelle i ermittelt, welche vor dem Trip t beobachtet wurde. Diese Zeiten werden statt der eigentlichen Verspätungen zur Zeit t genutzt, um eine Vorhersage über die Reisezeit des Segmentes zu treffen. Die einzelnen Reisezeiten der Segmente werden summiert und die Abweichung von der geplanten Reisezeit berechnet. Dasselbe wird für die vorhergesagte Reisezeit erledigt und diese werden schlussendlich in der Konfusionsmatrix als Wahrheit und Vorhersage gesetzt.
In den folgenden vier Tabellen sind die Ergebnisse der Auswertung zu sehen. Genutzt wurden die Testdaten aus Kapitel 5.2.5. Im Vergleich zu den Ergebnissen der letzten Versuche verschlechterte sich die Linie 15 (Tabelle 5.18) in vier Klassen. Die größte Verschlechterung fand in der Klasse 2 statt, dort sank die Precision von 68% auf nur 30%. Auch die restlichen Klassen, außer 2, brachten Verschlechterungen von 7% bis 26% mit sich. Nur für die Klasse 2 wurde eine Verbesserung der Precision erreicht. Dort stieg sie von 59% auf 82%.

Tabelle 5.18: Konfusionsmatrix - Linie 15 - Auswertung nach Gal

Pred / True	0	1	2	3	4	Precision
0	721	9	476	1	57	0.57
1	8	28	55	0	1	0.3
2	768	69	6608	43	508	0.82
3	1	6	43	5	2	0.09
4	93	3	771	0	350	0.29
Recall	0.45	0.24	0.83	0.1	0.38	

Die Auswertung der Linie 18 (Tabelle 5.19) weist im Vergleich ein anderes Verhalten auf. Vier der Klassen erreichten im Vergleich höhere Werte für die Precision. Der geringste Anstieg war für die Klasse 4 zu bemerken, dort stieg der Wert von 23% auf 26%. Außerdem stieg die Precision der Klasse 3 von 0% auf 8%. Dieses Mal wurden auch Werte für diese Klasse vorhergesagt, was in Kapitel 5.2.5 nicht der Fall war. Ferner stieg die Precision der Klasse 0 um 17% auf 56%. Jedoch brachten diese Verbesserungen einen Abstieg der Precision der Klasse 1 von 65% auf 9% mit sich. Die durchschnittliche Precision lag sowohl für diese Konfusionsmatrix als auch die aus Kapitel 5.2.5 bei $35,6\%$. Für die Linie 22 (Tabelle 5.20) bot sich wieder ein anderes Bild. Die einzige Verbesserung ergab sich für die Klasse 2, hier stieg die Precision von 73% auf 85%. Dies brachte jedoch Verschlechterungen der restlichen Klassen mit sich. Klasse 0 sank von 82% auf 53%, Klasse 1 von 36% auf 17%, Klasse 3 von 15% auf 5% und Klasse 4 sank um 16% auf 13%. Bei der Betrachtung der Ergebnisse für die Linie 26 (Tabelle 5.21) war ein ähnliches Verhalten wie in Tabelle 5.20 zu sehen. Nur die Klassen 2 und 3 erreichten eine minimale Verbesserung der Precision und zwar von 73% auf 82% für Klasse 2 und 2% auf 8% für die Klasse 3.

Tabelle 5.19: Konfusionsmatrix - Linie 18 - Auswertung nach Gal

Pred / True	0	1	2	3	4	Precision
0	702	5	478	4	59	0.56
1	1	9	85	1	3	0.09
2	798	119	6017	93	567	0.79
3	5	12	116	12	2	0.08
4	95	6	828	8	336	0.26
Recall	0.44	0.06	0.79	0.1	0.34	

Tabelle 5.20: Konfusionsmatrix - Linie 22 - Auswertung nach Gal

Pred / True	0	1	2	3	4	Precision
0	431	1	362	0	21	0.53
1	0	1	4	1	0	0.17
2	542	5	5352	15	408	0.85
3	1	0	18	1	0	0.05
4	46	2	504	0	84	0.13
Recall	0.42	0.11	0.85	0.06	0.16	

Tabelle 5.21: Konfusionsmatrix - Linie 26 - Auswertung nach Gal

Pred / True	0	1	2	3	4	Precision
0	501	3	695	1	47	0.4
1	3	11	65	1	0	0.14
2	823	89	8152	96	735	0.82
3	1	11	102	10	2	0.08
4	98	0	874	3	277	0.22
Recall	0.35	0.09	0.82	0.09	0.26	

Die restlichen Klassen sanken um 14% für die Klasse 1, 34% für die Klasse 4 und die Klasse 0 sank von 41% auf 4%.
Auch mit Gals Ansatz [5, 16] der Queing-Theorie konnten für die vier ausgewählten Linien keine zufriedenstellenden Ergebnisse erreicht werden. Die Ergebnisse liegen sogar noch hinter den STRF zurück. Lediglich für die Linie 18 gab es zum STRF vergleichbare Ergebnisse.

5.5 Bewertung der Ergebnisse

Leider lieferte keines der STRF-Modelle zufriedenstellende Ergebnisse. Entweder wurde eine Klasse von dem STRF ignoriert, es wurde primär nur eine Klasse vorhergesagt oder Maße zur Modellperformance lieferten schwache Werte. Um herauszufinden, ob die vorgestellte Methode oder die Daten das Problem waren, wurde auf denselben Daten die Methode von Gal zum Vergleich genutzt. Da auch hier keine guten Ergebnisse erzielt wurden, wurde vermutet, dass es Probleme mit den Daten gibt. Eine mögliche Ursache könnte der GTFS-Feed sein, welcher im Frühjahr des Jahres erstellt wurde. Das Problem könnte darin liegen, dass die Fahrpläne nicht mehr aktuell sind und somit ein Rauschen in die Vergabe der Fahrten kommt. Dies würde dazu führen, dass die Daten, auf denen die Modelle gelernt und getestet werden, selbst nicht genügend akkurat sind, um Vorhersagen über die Abweichung des Fahrplans zu treffen. Weitere mögliche Probleme sind häufige Wechsel der Fahrpläne (im September wurden Fahrplanwechsel am 05.09, 10.09 und 26.09 bemerkt). Tritt ein solcher Fahrplanwechsel auf, werden Datensätze mit unterschiedlichen Verteilungen generiert, da die Fahrzeuge an unterschiedlichen Zeiten an den Haltestellen sein sollen, jedoch immer mit derselben Zeit verglichen werden.

Kapitel 6

Fazit und Ausblick

Die Arbeit hat ein Verfahren zur Ermittlung von Verspätungen der Stadtbahnen in Warschau sowie ein graphisches Modell zur Vorhersage dieser vorgestellt. Leider waren die Ergebnisse schlecht, was möglicherweise an dem zu alten GTFS-Feed lag. Interessant wäre es die Modelle noch einmal auf den Verspätungsdaten eines AVL-Systems, welches auf einem aktuellen GTFS-Feed arbeitet, zu trainieren, auszuwerten und diese mit den aktuellen Ergebnissen zu vergleichen. Allerdings sollte die zeitliche Dimension dieses Mal in Intervallen modelliert werden, da die Modelle so trotz Fahrplanwechseln eine einheitliche Struktur beibehalten können. Somit wäre es ihnen egal, ob in dem jeweiligen Intervall ein oder mehrere Trips fahren. Jedoch sollten die Intervalle nur aus maximal 30 Minuten bestehen, damit nicht zu viele Trips in ein Intervall fallen.

6.1 Ausblick und mögliche Verbesserungen

Das System zur Verspätungserkennung könnte durch einige Maßnahmen verbessert werden. Durch Hilfe eines Partikelfilters, wie er in New-York von OneBusAway[15] genutzt wird, könnte die Tripvergabe verbessert werden. Dort werden neben Linien und zeitlichen Daten auch Geschwindigkeit, Richtung sowie Entfernung zu einer Haltestelle mit einbezogen und basierend auf Likelihoodschätzungen der wahrscheinlichste Trip an einen Bus verteilt. Weiterhin könnte die Erkennung von Stopevents durch die Einbeziehung von anderen Events verbessert werden. Ein mögliches Event wäre die Öffnung der Türen einer Tram, welches die genaue Stopzeit verraten würde. In Warschau existiert wohl eine solche API, welche jedoch kein Teil der Open-Data-Initiative[16] ist. Weiterhin wäre es wichtig den GTFS-Feed automatisch jeden Tag auf Veränderungen zu überprüfen und im Falle eines neuen Feedes, diesen einzulesen und als Vergleich für die Verspätung heranzuziehen.
Außerdem wäre es interessant die Modelle in einer anderen Struktur zu testen. Eine denk-

[15] `https://github.com/camsys/onebusaway-nyc/wiki/Inference-Engine` (zuletzt besucht am 02.10.2010)

[16] `https://api.um.warszawa.pl/` (zuletzt besucht am 02.10.2010)

bare Struktur wäre ein Liniengraph über die Segmente zwischen zwei Haltestellen, wie in [5], oder die Reduzierung der zeitlichen Dimension auf einige Intervalle am Tag, wie in der Arbeit [6]. Jedoch könnte dies auch Einbußen bei der Modellqualität mit sich bringen, wenn zu viele Fahrten in einem Intervall zusammengefasst werden. Zusätzlich könnte ein Modell für jeden Wochentag gelernt werden, diese dynamisch ausgewechselt und zur Vorhersage herangezogen werden.
Auch für die Webseite gibt es noch einige nicht umgesetzte Ideen. Die Busse könnten z.B. mit ihren jeweiligen Fahrten an ihrer Position auf der Karte angezeigt werden und sich, sobald die API neue Daten bereitstellt, bewegen. Weiterhin sollten auch einzelne Fahrten für eine Linie ausgewählt werden können und zu dieser könnten die Verspätungen an jeder Haltestellen als Graph dargestellt werden.

Abbildungsverzeichnis

List of Algorithms

Literaturverzeichnis

[1] Baker, CB und Alexander C Nied: *Predicting bus arrival using One Bus Away real-time data.* `http://homes.cs.washington.edu/~anied/papers/AConradNied_OneBusAway_Writeup_20131209.pdf`, 2014. [Online; besucht 11.10.2016].

[2] Bishop, Christopher M.: *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] Bockermann, Christian und Hendrik Blom: *The streams Framework.* Technischer Bericht 5, TU Dortmund University, 12 2012.

[4] Florescu, Simona claudia, Michael Mock, Christine Körner und Michael May: *Efficient Mobility Pattern Stream Matching on Mobile Devices.*

[5] Gal, Avigdor, Avishai Mandelbaum, François Schnitzler, Arik Senderovich und Matthias Weidlich: *Traveling time prediction in scheduled transportation with journey segments.* Information Systems, 2015.

[6] Gurmu, Zegeye Kebede und Wei David Fan: *Artificial neural network travel time prediction model for buses using only GPS data.* Journal of Public Transportation, 17(2):3, 2014.

[7] Hastie, Trevor J., Robert John Tibshirani und Jerome H. Friedman: *The elements of statistical learning : data mining, inference, and prediction.* Springer series in statistics. Springer, New York, 2009. Autres impressions : 2011 (corr.), 2013 (7e corr.).

[8] Kindermann, Ross, J. Laurie James Laurie Snell und American mathematical society: *Markov random fields and their applications.* Contemporary mathematics. Providence, R.I. American Mathematical Society, 1980.

[9] Kschischang, Frank, Senior Member, Brendan J. Frey und Hans-andrea Loeliger: *Factor Graphs and the Sum-Product Algorithm.* IEEE Transactions on Information Theory, 47:498–519, 2001.

[10] Liebig, Thomas und Katharina Morik: *Report on the integration and the evaluation of the alarming, visualization and prediction component.* Technischer Bericht FP7-318225 D5.2, INSIGHT Consortium, Dortmund, Germany, August 2015.

[11] Liebig, Thomas, Nico Piatkowski, Christian Bockermann und Katharina Morik: *Dynamic Route Planning with Real-Time Traffic Predictions.* Information Systems, 1(1):(in press), 2016.

[12] Müller-Hannemann, Matthias und Mathias Schnee: *Efficient timetable information in the presence of delays.* In: *Robust and Online Large-Scale Optimization*, Seiten 249–272. Springer, 2009.

[13] Nocedal, Jorge und Stephen J. Wright: *Numerical Optimization, second edition.* World Scientific, 2006.

[14] Pearl, Judea: *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Representation and Reasoning Series. Morgan Kaufmann, 1988.

[15] Piatkowski, Nico, Sangkyun Lee und Katharina Morik: *Spatio-temporal random fields: compressible representation and distributed estimation.* Machine Learning, 93(1):115–139, 2013.

[16] Senderovich, Arik, Matthias Weidlich, Avigdor Gal und Avishai Mandelbaum: *Queue mining–predicting delays in service processes.* In: *International Conference on Advanced Information Systems Engineering*, Seiten 42–57. Springer, 2014.

[17] Sokolova, Marina, Nathalie Japkowicz und Stan Szpakowicz: *Beyond Accuracy, F-score and ROC: A Family of Discriminant Measures for Performance Evaluation.* In: *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI'06, Seiten 1015–1021, Berlin, Heidelberg, 2006. Springer-Verlag.

[18] Souto, Gustavo und Thomas Liebig: *On Event Detection from Spatial Time series for Urban Traffic Applications.* In: Michaelis, Stefan, Nico Piatkowski und Marco Stolpe (Herausgeber): *Solving Large Scale Learning Tasks: Challenges and Algorithms*, Band 9580, Seiten 221–233. Springer International Publishing, 2016.

[19] Wainwright, Martin J. und Michael I. Jordan: *Graphical Models, Exponential Families, and Variational Inference.* Foundations and Trends® in Machine Learning, 1(1–2):1–305, 2008.

[20] Whitt, W.: *Stochastic-Process Limits: An Introduction to Stochastic-Process Limits and Their Application to Queues.* Springer Series in Operations Research and Financial Engineering. Springer New York, 2002.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 13. Oktober 2016

Lukas Heppe