Karlsruhe Institute of Technology

Algorithmische Werkzeuge zur Analyse von Routingdaten basierend auf historischen Fahrzeugtrajektorien

Bachelorarbeit
von

Florian Krone

An der Fakultät für Informatik
Institut für Theoretische Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. Peter Sanders
Betreuender Mitarbeiter:	Tim Zeitz

Bearbeitungszeit: 01. Dezember 2018 – 01. April 2019

Statement of Authorship

I hereby declare that this document has been composed by myself and describes my own work, unless otherwise acknowledged in the text.

Karlsruhe, 31. März 2019

Abstract

Correct routing services are of increasing importance, as more and more people rely on them. In this thesis, we therefore present algorithms to examine routing services for errors. The errors we want to find are missing roads, forbidden maneuvers and deviations in the predicted journey time. For this we use historical vehicle trajectories. Our approach is to reproduce the trajectories by calculating a route using the routing data, starting at the beginning of the trajectory and ending at the destination of the trajectory. We use the Fréchet distance to check if the calculated route matches the trajectory. By identifying the points where the replication fails, we can find possible errors in the routing data. We propose a clustering strategy to filter out failed replications due to GPS-Outliers as well as irrational behaviour by the driver and identify points where there is significant indication of erroneous routing data.
In addition, based on the intermediate results of our algorithm, we have developed measures that allow to compare the quality of different routing services.

Deutsche Zusammenfassung

Korrekte Routing-Services werden immer wichtiger, da sich immer mehr Menschen auf sie verlassen. Daher stellen wir in dieser Arbeit Algorithmen vor, die Routing-Services auf Fehler untersuchen. Die Fehler, die wir finden wollen, sind fehlende Straßen, verbotene Manöver und Abweichungen in der vorhergesagten Fahrtdauer. Dazu nutzen wir historische Fahrzeugtrajektorien. Unser Ansatz ist es, die Trajektorien nachzubilden, indem wir mit den Routingdaten, von Start bis Ziel der Trajektorie, eine Route berechnen. Zur Überprüfung der Nachbildung nutzen wir die Fréchet-Distanz. Durch die Identifizierung der Stellen, an denen die Nachbildung fehlschlägt, können wir mögliche Fehler in den Routingdaten finden. Wir schlagen eine Clustering-Strategie vor, um Punkte herauszufiltern, an denen die Nachbildung durch GPS-Ausreißer oder irrationales Verhalten des Fahrers nicht möglich ist. Dadurch identifizieren wir Punkte, an denen Fehler in den Routingdaten wahrscheinlich sind.
Zusätzlich haben wir auf Basis der Zwischenergebnisse unseres Algorithmus Maßzahlen entwickelt, die einen Vergleich der Qualität verschiedener Routing-Services ermöglichen.

Inhaltsverzeichnis

1. Einleitung

Im Jahr 2017 wurden in Deutschland 642,4 Milliarden Kilometer mit PKW zurückgelegt [Ver]. Viele dieser Fahrten werden durch Autos, Smartphones oder andere Geräte als Trajektorien aus GPS-Punkten protokolliert. Diese Informationen wollen wir nutzen, um Routingdaten zu verbessern. Dadurch kann nicht nur die Nutzerzufriedenheit erhöht werden, sondern z.B. auch der verzögerungsfreie Lieferverkehr effizienter gestaltet werden. Entscheidende Eigenschaften für gute Routingdaten sind für uns, dass alle Straßen in der Karte enthalten sind und die Routen keine verbotenen Manöver vorschlagen. Wichtig ist also z.B., dass eine neu gebaute Umgehungsstraße genutzt wird und der Verkehr nicht durch den entsprechenden Ort geleitet wird oder die Routen nicht vorschlagen, durchgezogene Linien zu überfahren. Eine weitere wichtige Eigenschaft ist die Korrektheit der vorhergesagten Reisezeiten.
Es gibt eine große Zahl verschiedener Routing-Services. Unser Tool soll mit einer möglichst großen Menge dieser Services kompatibel sein. Daher stellen wir als einzige Bedingung an einen Routing-Service eine minimale Schnittstelle, die zu je einem Start- und Zielpunkt eine Route liefert.
Unser Algorithmus benötigt eine Menge von Fahrzeugtrajektorien und evaluiert damit einen Routing-Service. Dazu versucht der Algorithmus die einzelnen Trajektorien durch Anfragen an den Routing-Service nachzubilden. Ist dies nicht möglich, so vermuten wir zunächst einen Fehler in der Trajektorie, z.B. durch ein nicht erlaubtes Manöver während der Fahrt. Erst wenn mehrere Trajektorien an einer Stelle nicht nachgebildet werden können, gehen wir von einem Fehler in den Routingdaten aus.
Zusätzlich vergleichen wir die Fahrtdauer der Trajektorien mit der von dem Routing-Service vorgeschlagenen Fahrtdauer. Da wir die Trajektorien durch Routen nachbilden, können Abweichungen in der Fahrtdauer nicht durch die Nutzung anderer Straßen entstehen. Daher können wir von Abweichungen der Fahrtdauer direkt auf Abweichungen der tatsächlichen zur vorhergesagten Geschwindigkeit des Fahrzeugs schließen.
Zur Überprüfung, ob eine Route und eine Trajektorie übereinstimmen, nutzen wir die Fréchet-Distanz, da diese, im Gegensatz zu anderen Ähnlichkeitsmaßen für Kurven, wie z.B. die Hausdorff-Distanz, die Reihenfolge der Punkte auf einer Kurve betrachtet [AKW04].

Verwandte Arbeiten

GPS-Daten können auf viele verschiedene Arten genutzt werden um Informationen zu gewinnen. Dazu gehören Echtzeitvorhersagen von Fahrtdauern auf Basis von schon gefahrenen Trajektorien [WZX14], die Echtzeiterkennung von Staus [ZZY+14] und ungewöhnlichem

Fahrverhalten z.B. durch Unfälle [PZWS13] sowie der Aufbau eines eigenen Routing-Services auf Basis von Trajektorien [YZXS11]. Eine Übersicht findet sich in [Zhe15]. Insgesamt gibt es nur relativ wenige Arbeiten zur Informationsgewinnung aus Fahrzeugtrajektorien, da kaum GPS-Daten für Fahrzeuge öffentlich verfügbar sind.
Viele Arbeiten um Fahrzeugtrajektorien gibt es zum Thema Map-Matching. Dabei wird versucht für eine Trajektorie zu rekonstruieren, auf welchen Straßen das Fahrzeug gefahren ist. Es gibt verschiedene Ansätze wie etwa mit einem Hidden Markov model [NK09], eine Kombination aus räumlicher Ähnlichkeit und der Geschwindigkeit [LZZ+09] sowie rein geometrische Ansätze, die z.B. die Fréchet-Distanz als Ähnlichkeitsmaß verwenden [BPSW]. Alle uns bekannten Ansätze haben gemein, dass sie den zugrundeliegenden Graphen, der die Straßen modelliert, kennen. Ein ähnliches Problem ist es, zu Kurven übereinstimmende Pfade in Graphen zu finden [AERW03, AW12, MSSZZ14]. Durch unsere Entscheidung, die Routingdaten hinter einem Routingorakel zu verstecken, haben wir jedoch keinen direkten Zugriff auf den zugrundeliegenden Graphen.
Ein zentrales Problem in unserer Arbeit ist es, partielle Übereinstimmungen von Kurven zu finden, da eine Trajektorie und die dazu berechnete Route in vielen Fällen nicht komplett übereinstimmen. Ein häufiger Ansatz ist es dabei, dass nur von einer der beiden Kurven eine Teilkurve betrachtet wird. Also zu Kurven P, Q eine Teilkurve $R \subseteq P$ gesucht wird, sodass R und Q übereinstimmen [AG95, MSSZZ14]. Ein weiterer Ansatz ist es, Teile der einen Kurve zu überspringen und jeweils durch eine Strecke zwischen zwei Punkten zu ersetzen [DHP13]. Beide Ansätze sind für unser Problem nicht Praktikabel, da wir nicht übereinstimmende Abschnitte in der Mitte erwarten, diese aber nicht einfach durch eine Strecke ersetzen können. Unter der Verwendung der L_1 oder L_∞-Norm und von Streckenzügen wurde das Problem, partielle Übereinstimmung von Kurven zu finden, bereits gelöst, allerdings mit einer Laufzeit von $O(mn(m+n)log(mn))$, wobei n und m die Längen der Streckenzüge sind [BBW09]. Wir haben uns daher entschieden, einen schnelleren Ansatz zu verwenden, der allerdings nur partielle Übereinstimmungen am Anfang und Ende der Kurven finden kann. Außerdem verwenden wir die L_2-Norm.

Unser Beitrag

Unser Beitrag ist erstens ein neuer Ansatz zum Finden partieller Übereinstimmungen in Kurven, ausgehend vom Start und Ziel der beiden Kurven, auf Basis der Fréchet-Distanz. Dabei wird entweder die Übereinstimmung der Kurven festgestellt oder zwei Punkte auf den Kurven identifiziert, bis zu dem bzw. ab dem diese übereinstimmen. Zweitens stellen wir einen Algorithmus zur Bewertung beliebiger Routing-Services vor.

2. Voraussetzungen

Das Ziel dieser Arbeit ist es, Fehler in Routing-Services zu finden. Zur Überprüfung eines Service benutzen wir Fahrzeugtrajektorien. Diese werden im Folgenden nur als *Trajektorien* bezeichnet.
Die Grundidee des von uns entwickelten Algorithmus ist es zu versuchen, die Trajektorien durch Routen nachzubilden.

Definition 2.1. *Eine* Punktefolge *ist eine Folge von Tripeln* (Latitude, Longitude, Zeit). Routen *und* Trajektorien *sind Punktefolgen.*

Unser Algorithmus soll dabei unabhängig von dem jeweiligen zu evaluierenden Service sein. Dazu verbergen wir je einen Service hinter einem *Routingorakel.*

Definition 2.2. *Ein* Routingorakel *ist eine Abbildung*

$$r : \mathbb{R}^2 \times \mathbb{R}^2 \to [(\mathbb{R}^2 \times \mathbb{N})]$$

von zwei Koordinatenpunkten auf eine Route.

Zunächst benötigen wir eine Überprüfung, ob eine Route und eine Trajektorie übereinstimmen. Dafür lassen wir die Zeit zunächst außen vor und arbeiten ausschließlich geometrisch.

Definition 2.3.

(1) Seien $a, b \in \mathbb{R}^3$, dann ist durch

$$S[a, b] := \{a + \lambda(b - a) : \lambda \in [0, 1]\}$$

die Strecke zwischen a und b gegeben.

(2) Seien $x_0, ..., x_m \in \mathbb{R}^3$, dann heißt die Vereinigung der Strecken

$$S[x_0, ..., x_m] := \bigcup_{j=1}^{m} S[x_{j-1}, x_j]$$

der Streckenzug *durch $x_0, ..., x_m$.*
Als Parametrisierung von $S[x_0, ..., x_m]$ verwenden wir die Abbildung $S : [0, m] \to S[x_0, ..., x_m]$ mit $S(k) = x_k, k \in \{0, ..., m\}$. Alle weiteren Punkte sind durch lineare Interpolierung festgelegt.

Zu einer Punktefolge kann ein Streckenzug konstruiert werden, indem die Zeit ignoriert wird und die Koordinaten vom WGS84-System in ein geozentrisches Koordinatensystem übertragen werden. Der durch eine Trajektorie bzw. Route definierte Streckenzug heißt *Trajektoriezug* bzw. *Routenzug.*

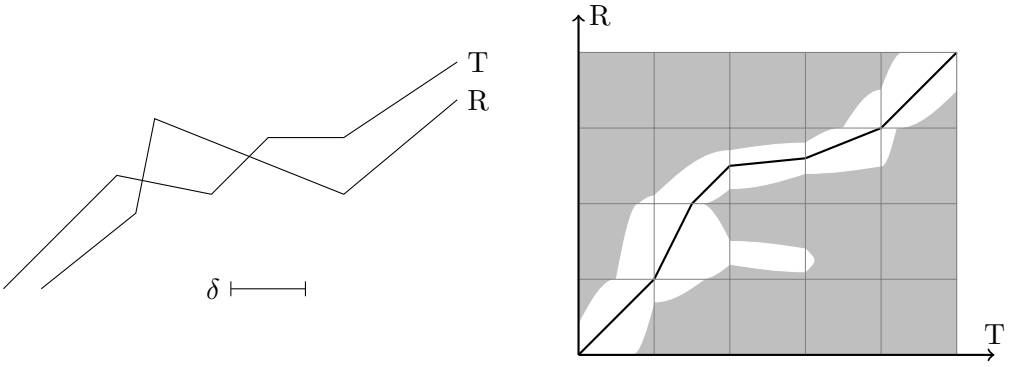


Abbildung 2.1: Streckenzüge T, R und das zugehörige free-space Diagramm mit einem festen δ

2.1 Fréchet-Distanz

Für zwei Streckenzüge gibt es mit der *Fréchet-Distanz* ein bekanntes Ähnlichkeitsmaß [Fré06]. Wir verwenden immer die kontinuierliche Fréchet-Distanz und werden diese in der Arbeit daher nur als Fréchet-Distanz bezeichnen. Für ein Paar von Trajektorie und Route kann mit der Konvertierung zu einem Trajektorie- und Routenzug die Fréchet-Distanz genutzt werden, um zu überprüfen, ob diese übereinstimmen. Da der konkrete Wert der Fréchet-Distanz dazu nicht benötigt wird, reicht es, das Entscheidungsproblem zu lösen.

Definition 2.4. *Seien* $T : [a_1, a_2] \to \mathbb{R}^3$ *und* $R : [b_1, b_2] \to \mathbb{R}^3$ *Kurven mit* $a_1 < a_2 \in \mathbb{R}$ *und* $b_1 < b_2 \in \mathbb{R}$. *Dann heißt*

$$F(T, R) = \inf_{\alpha,\beta} \max_{x \in [0,1]} ||T(\alpha(x)) - R(\beta(x))||_2$$

die Fréchet-Distanz *von T und R, wobei* $\alpha : [0, 1] \to [a_1, a_2]$ *und* $\beta : [0, 1] \to [b_1, b_2]$ *stetig, monoton steigend und surjektiv sind und über alle Reparametrisierungen reichen.*

Eine intuitive Definition der Fréchet-Distanz ergibt sich, wenn man sich vorstellt, dass auf der einen Kurve ein Mensch läuft, der seinen Hund an der Leine ausführt, der auf der anderen Kurve läuft. Beide können beliebig schnell, aber ausschließlich vorwärts gehen. Die Fréchet-Distanz ist dann die minimal ausreichende Länge der Leine.

2.1.1 Lösen des Entscheidungsproblems

Das Entscheidungsproblem zur Fréchet-Distanz ist gegeben durch:

Gegeben: Kurven T und R und ein $\delta \geq 0$
Entscheide, ob $F(T, R) \leq \delta$

Alt und Godau haben einen Algorithmus vorgestellt, der das Entscheidungsproblem für Streckenzüge T und R mit t bzw. r Punkten und einem festen $\delta \geq 0$ löst [AG95]. Dazu führen sie zunächst ein sogenanntes free-space Diagramm F_δ ein. Dies ist gegeben durch: $F_\delta := \{(i, j) \in [0, t] \times [0, r] \mid ||T(i) - R(j)||_2 \leq \delta\}$. Abbildung 2.1 zeigt ein solches Diagramm. Eine Zelle $C_{i,j}$ des Diagramms, wie in Abbildung 2.2 dargestellt, beschreibt, welche Punkte der Strecken $S[R(i), R(i + 1)]$ und $S[T(j), T(j + 1)]$ einen Abstand von höchstens δ haben.
Die Ränder einer Zelle sind immer Intervalle, da hier eine Punkt-Strecken-Distanz betrachtet wird. Alt und Godau beweisen, dass der Inhalt einer Zelle immer konvex ist. Somit müssen nur die Ränder der Zellen berechnet werden.
Sie führen eine Notation ein, die auch in dieser Arbeit verwendet wird. So bezeichnen

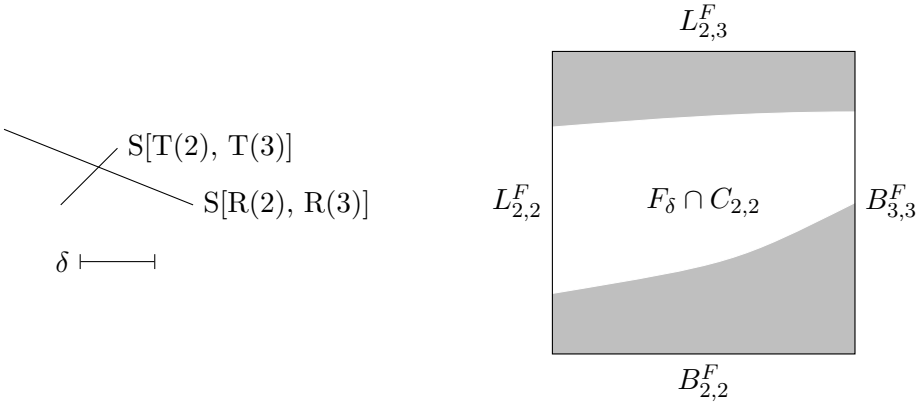


Abbildung 2.2: Eine Zelle des free-space Diagramms und die zugehörigen Strecken

Algorithm 2.1: Decide Fréchet-Distance

Input: Polygonal Chains T, R of length t, r, Deviation δ
Data: Matrices $L^F, L^R \in t \times (r-1)$, $B^F, B^R \in (t-1) \times r$
Output: Decision: Boolean

```
1  foreach (i, j) ∈ (0..t) × (0..r − 1) do
2      compute L^F_{i,j}
3  foreach (i, j) ∈ (0..t − 1) × (0..r) do
4      compute B^F_{i,j}
5  for i ← 0..t do
6      determine B^R_{i,0}
7  for j ← 0..r do
8      determine L^R_{0,j}
9  for i ← 0..t do
10     for j ← 0..r do
11         construct L^R_{i+1,j} and B^R_{i,j+1} from L^R_{i,j}, B^R_{i,j}, L^F_{i+1,j}, B^F_{i,j+1}
12 return (r, t) ∈ L^R_{r,t}
```

$L^F_{i,j}$ und $B^F_{i,j}$ den linken bzw. unteren Rand einer Zelle $C_{i,j}$ von F_δ. Durch $L^F_{i+1,j}$ und $B^F_{i,j+1}$ ist entsprechend der rechte bzw. obere Rand der Zelle gegeben.
Um $L^F_{i,j}$ zu berechnen, werden der Punkt $p := T(i)$ und die Strecke $s := S[R(j), R(j+1)]$ benötigt. Zur Berechnung wird eine Kugel mit Radius δ um p gelegt und diese mit s geschnitten. $B^F_{i,j}$ kann analog, mit dem Punkt $R(j)$ und der Strecke $S[T(i), T(i+1)]$, berechnet werden.
Weiter erkennen sie, dass $F(R,T) \leq \delta$ genau dann gilt, wenn es eine Kurve in F_δ von $(0,0)$ bis (t,r) gibt, die in beiden Koordinaten monoton ist. In Abbildung 2.1 ist eine solche Kurve dargestellt. Weiter definieren sie mit dieser Kurve
$R_\delta := \{(i,j) \in F_\delta \mid$ es existiert eine monotone Kurve in F_δ von (0,0) bis (i,j)$\}$ als Menge aller Punkte in F_δ, die von $(0,0)$ aus erreichbar sind. Für die Lösung des Entscheidungsproblems ist es nicht wichtig, wie die Kurve verläuft, sondern nur, ob sie existiert. Somit reicht es zu prüfen, ob $(t,r) \in R_\delta$ gilt.
Analog zum free-space Diagramm werden für eine Zelle des reachable Diagramms die Bezeichnungen $L^R_{i,j}$ und $B^R_{i,j}$ eingeführt.
Zur Lösung des Entscheidungsproblems ergibt sich dann Algorithmus 2.1. Dieser erhält zwei Streckenzüge T, R sowie ein $\delta \geq 0$ als Eingabe und bestimmt, ob $F(T,R) \leq \delta$ ist.

Zunächst berechnet der Algorithmus die Kanten aller Zellen des free-space Diagramms $L^F_{i,j}$ bzw. $B^F_{i,j}$. Dazu wird, wie oben beschrieben, je ein Punkt des einen Streckenzuges und eine Strecke des jeweils Anderen benötigt.
Als Nächstes werden der untere und linke Rand des reachable Diagramms berechnet. Dazu wird nur der entsprechende Rand des free-space Diagramms benötigt. Ein Punkt auf dem unteren Rand ist genau dann durch eine monotone Kurve von $(0, 0)$ aus erreichbar, wenn alle vorherigen Punkte des Randes im free-space Diagramm enthalten sind. Für den linken Rand gilt dies analog.
Anschließend werden die restlichen Kanten aller Zellen des reachable Diagramms berechnet. Die Kanten $L^R_{i+1,j}$ und $B^R_{i,j+1}$ werden aus $L^R_{i,j}, B^R_{i,j}, L^F_{i+1,j}$ und $B^F_{i,j+1}$ konstruiert. Wenn $B^R_{i,j}$ nicht leer ist, so gilt $L^R_{i+1,j} = L^F_{i+1,j}$. Ansonsten entspricht der kleinste Punkt von $L^R_{i+1,j}$ dem kleinsten Punkt von $L^R_{i,j}$. Sind $L^R_{i,j}$ und $B^R_{i,j}$ beide leer, so gilt dies auch für $L^R_{i+1,j}$. Analog kann auch $B^R_{i,j+1}$ berechnet werden.
Zuletzt muss nur noch geprüft werden, ob $(t, r) \in R_\delta$ gilt. Genau wenn dies der Fall ist, gilt auch $F(T, R) \leq \delta$.
Der beschriebene Kugel-Strecken-Schnitt und die Konstruktion von $L^R_{i+1,j}$ und $B^R_{i,j+1}$ können in konstanter Zeit ausgeführt werden. Daher kann sowohl die Konstruktion des free-space, als auch des reachable Diagramms in $O(tr)$ Zeit ausgeführt werden. Damit ergibt sich auch als Gesamtlaufzeit $O(tr)$.

3. Der Algorithmus

Im folgenden Kapitel werden nacheinander drei Algorithmen vorgestellt, die anschließend zu einem Algorithmus integriert werden, den wir verwenden, um Fehler in Routing-Services zu finden. Die Kernidee ist es dabei, zu tatsächlich gefahrenen Strecken Routen bei einem Service anzufragen und schließlich die gefahrenen Strecken durch Routen nachzubilden. Zur Überprüfung der Nachbildung stellt der erste Algorithmus eine Beziehung zwischen je einem Trajektoriezug und einem Routenzug her. Die anderen beiden Algorithmen nutzen diese Beziehung, um mögliche Fehler in den Routingdaten zu finden. Beide Algorithmen geben eine Liste mit Koordinatenpunkten aus, an denen mögliche Fehler identifiziert wurden. Anschließend wird vorgestellt, wie Ausreißer aus diesen Punkten herausgefiltert werden und Punkte identifiziert werden, an denen sich mit hoher Wahrscheinlichkeit Fehler in den Routingdaten befinden.

3.1 Abbildung von Trajektoriezügen auf Routenzüge

Definition 3.1.

(1) Seien $a, b \in \mathbb{R}$ mit $a \leq b$. Dann bezeichnet

$$I[a,b] := \{[x,y] \mid a \leq x \leq y \leq b\}$$

die Menge aller Teilintervalle von $[a,b]$.

(2) Sei $\beta : [0,1] \to [a,b]$ mit $a < b \in \mathbb{R}$ stetig, surjektiv und monoton steigend. Dann nennen wir

$$B : I[0,1] \to I[a,b], B(X) := \{\beta(y) \mid y \in X\}$$

die Intervallfunktion *von β.*

(3) Sei $\alpha : [0,1] \to [a,b]$ mit $a < b \in \mathbb{R}$ stetig, surjektiv und monoton steigend. Dann heißt

$$\alpha^{-1} : [a,b] \to I[0,1], \alpha^{-1}(x) := \{y \mid \alpha(y) = x\}$$

die Intervallinverse *von α.*

(4) Seien T, R Streckenzüge, dann definieren wir

$$R_{T,R} := \left\{ (\alpha, \beta) \mid \text{mit } \alpha, \beta \text{ gilt } \max_{x \in [0,1]} ||T(\alpha(x)) - R(\beta(x))||_2 = F(T,R) \right\}$$

als die Menge aller Reparametrisierungen, für die die Distanz nach Definition 2.4 minimal wird.

(5) Seien T, R Streckenzüge, dann ist durch

$$f_{T,R} : [0, |T| - 1] \to 2^{I[0,|R|-1]},\ f_{T,R}(x) := \left\{ B(\alpha^{-1}(x)) \mid (\alpha, \beta) \in R_{T,R} \right\}$$

eine Abbildung von jedem Punkt in T auf Abschnitte von R gegeben.

Für zwei Streckenzüge T, R sind die Reparametrisierungen $(\alpha, \beta) \in R_{T,R}$ nicht notwendigerweise streng monoton. Insbesondere ist also α nicht notwendigerweise invertierbar. Daher definieren wir in (3) eine Intervallinverse. Da α surjektiv und monoton steigend ist, ist $\alpha^{-1}(x)\,(x \in [a, b])$ immer ein Intervall. Zusätzlich müssen wir nun β auf Intervalle erweitern. Dies tun wir in (2). Wieder folgt aus der Surjektivität und Monotonie, dass $B(x)\,(x \in I[0, 1])$ ein Intervall ist. Da wir auf Streckenzügen arbeiten, können wir davon ausgehen, dass $R_{T,R}$ nicht leer ist.
Um mit der Abbildung $f_{T,R}$ zu arbeiten, benötigen wir zunächst ein $\delta \geq 0$ als maximalen Abstand zweier Punkte, an dem wir diese, unter Berücksichtigung von Abweichungen wie z.B. GPS-Rauschen, noch als übereinstimmend betrachten. Die Wahl des δ betrachten wir in Abschnitt 3.1.2.
Theorem 3.2 zeigt, dass es einen direkten Zusammenhang von $f_{T,R}$ und dem Entscheidungsproblem der Fréchet-Distanz gibt. Für zwei Streckenzüge T, R und ein $\delta \geq 0$ ist die Fréchet-Distanz $F(T, R)$ von T, R genau dann höchstens δ, wenn für alle Punkte k von T und alle Punkte l von R, die in $f_{T,R}(k)$ enthalten sind, die Distanz $||T(k) - R(l)||_2$ höchstens δ ist.

Theorem 3.2. *Seien T, R Streckenzüge und $\delta \geq 0$, dann gilt*

$$\forall k \in [0, |T| - 1]\, \forall l \in \left(\bigcup_{I \in f_{T,R}(k)} I \right) : ||T(k) - R(l)||_2 \leq \delta \iff F(T, R) \leq \delta$$

Beweis. Sei $F(T, R) \leq \delta$.
Aus der Definition der Fréchet-Distanz (2.4) folgt direkt

$$\forall (\alpha, \beta) \in R_{T,R}\, \forall x \in [0, 1] : ||T(\alpha(x)) - R(\beta(x))||_2 \leq \delta \tag{3.1}$$

Seien $k \in [0, |T| - 1]$ und $l \in \left(\bigcup_{I \in f_{T,R}(k)} I \right)$ beliebig, dann folgt mit der Definition von $f_{T,R}$

$$\exists (\alpha, \beta) \in R_{T,R} : l \in B(\alpha^{-1}(k)) \tag{3.2}$$

$$\implies \exists (\alpha, \beta) \in R_{T,R}\, \exists x \in \alpha^{-1}(k) \subseteq [0, 1] : \beta(x) = l \tag{3.3}$$

$$\implies \exists (\alpha, \beta) \in R_{T,R}\, \exists x \in [0, 1] : \alpha(x) = k \wedge \beta(x) = l \tag{3.4}$$

$$\overset{3.1}{\implies} ||T(k) - R(l)||_2 \leq \delta \tag{3.5}$$

Die Schritte zu 3.3 und 3.4 folgen jeweils direkt aus den Definitionen der Intervallfunktion und der Intervallinverse (Definition 3.1 (2) und (3)).
Da l und k beliebig gewählt wurden, gilt

$$\forall k \in [0, |T| - 1]\, \forall l \in \left(\bigcup_{I \in f_{T,R}(k)} I \right) : ||T(k) - R(l)||_2 \leq \delta$$

Sei $F(T,R) > \delta$.
Seien $(\alpha, \beta) \in R_{T,R}$ beliebig. Mit der Surjektivität von α und β sowie der Definitionen der Intervallinversen und der Intervallfunktion gilt

$$\forall x \in [0,1]\, \exists k \in [0, |T|-1] : x \in \alpha^{-1}(k) \tag{3.6}$$

$$\implies \quad \forall x \in [0,1]\, \exists k \in [0, |T|-1]\, \exists l \in \left(\bigcup_{I \in f_{T,R}(k)} I \right) : \alpha(x) = k \wedge \beta(x) = l \tag{3.7}$$

Mit der Definition der Fréchet-Distanz gilt

$$\exists x \in [0,1] : ||T(\alpha(x)) - R(\beta(x))||_2 > \delta \tag{3.8}$$

$$\overset{3.7}{\implies} \quad \exists k \in [0, |T|-1]\, \exists l \in \left(\bigcup_{I \in f_{T,R}(k)} I \right) : ||T(k) - T(l)||_2 > \delta \tag{3.9}$$

□

3.1.1 Komprimierung von Trajektoriezügen

Unser Ziel ist es nun, einen Trajektoriezug T zu komprimieren, indem wir nur einige Punkte von T speichern und zwischen den Punkten Routen bei einem Routingorakel anfragen. Die Routenzüge, die sich daraus ergeben, sollen T nachbilden. Die Nachbildung überprüfen wir mit der Abbildung f und einem festen $\delta \geq 0$.
Wir treffen nun die Annahme, dass aus einer Menge an Trajektoriezügen die meisten Fahrten sinnvoll sind, also z.B. keine unnötigen Umwege enthalten. Auf Basis dieser Annahme bewerten wir einen hinter einem Routingorakel versteckten Service als gut, wenn wir für eine Menge mit Trajektoriezügen die einzelnen Züge stark komprimieren können, also nur wenige Punkte speichern müssen.

Definition 3.3. *Seien T, R Streckenzüge und $\delta \geq 0$. Dann bezeichnet*

$$M^{\delta}_{T,R} := \left\{ x \mid \exists y \in \left(\bigcup_{I \in f_{T,R}(x)} I \right) : ||T(x) - T(y)||_2 > \delta \right\}$$

die Menge aller Punkte, an denen T und R um mehr als δ voneinander abweichen.

Zur Komprimierung von T fragen wir bei einem Routingorakel r eine Route vom ersten bis zum letzten Punkt von T an, die wir in einen Routenzug R umwandeln. In der Menge $M^{\delta}_{T,R}$ fassen wir aufeinanderfolgende Punkte zu Intervallen zusammen. Damit erhalten wir $n-1$ Intervalle. Die Mittelpunkte der Intervalle bezeichnen wir als $x_1, x_2, ..., x_{n-1}$. Zusammen mit $x_0 = 0$ und $x_n = |T| - 1$ haben wir $n+1$ Punkte, die wir zur Komprimierung benutzen. Für je zwei aufeinanderfolgende Punkte $T(x_i), T(x_{i+1})\, (i \in \{0, 1, ..., n\})$ fragen wir bei r eine Route R_i an. Wir bezeichnen mit $T_i := T\big|_{[x_i, x_{i+1}]}$ den Teil des Trajektoriezugs zwischen x_i und x_{i+1}. Gilt $F(T_i, R_i) > \delta$, so unterteilen wir T_i rekursiv weiter, bis für alle Stücke die Fréchet-Distanz höchstens δ ist. Insgesamt ergeben sich somit k Zwischenpunkte, an denen wir den Trajektoriezug unterteilen müssen, um ihn nachbilden zu können. Dadurch ergeben sich $k+1$ Anfragen an das Routingorakel r.

3.1.2 Wahl des Deltas

Durch den Parameter δ wird bestimmt, wie weit zwei Punkte auf zwei Streckenzügen maximal voneinander entfernt sein dürfen, um noch als übereinstimmend angesehen zu

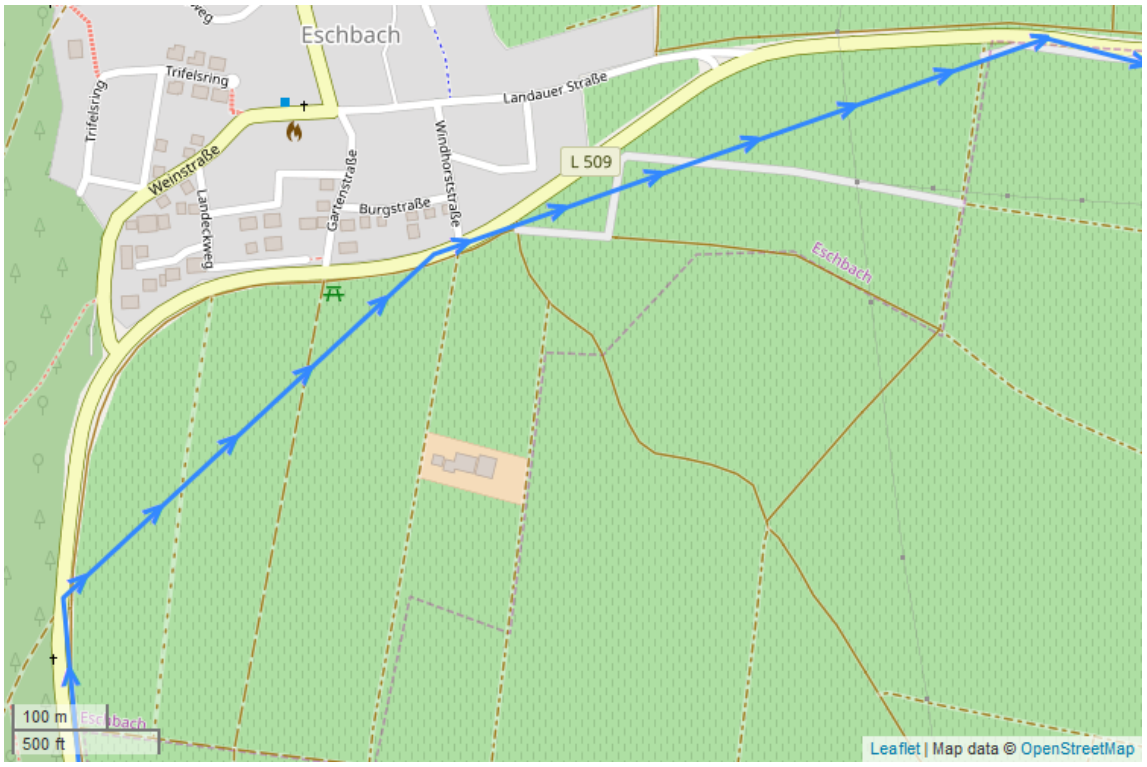


Abbildung 3.1: Abweichung einer Trajektorie von der Straße

werden. Dies muss nicht nur Abweichungen durch GPS-Rauschen berücksichtigen, sondern auch das in Abbildung 3.1 gezeigte Phänomen. Dadurch, dass ein Trajektoriezug nur zu diskreten Zeitpunkten die tatsächliche Position des Fahrzeugs darstellt und zwischen diesen Punkten interpoliert, kann der Trajektoriezug von der Straße abweichen, die das Fahrzeug gefahren ist. Bei festen zeitlichen Abständen zwischen den Punkten einer Trajektorie ist der räumliche Abstand von der Geschwindigkeit des Fahrzeugs abhängig. So ist in Abbildung 3.1 ein δ von über 100 Metern erforderlich, in Stadtgebieten kann dies jedoch zu falschen Übereinstimmungen führen. Wir bestimmen deshalb für jeden Punkt des Trajektoriezugs ein eigenes δ, welches der maximalen Distanz zu den beiden benachbarten Punkten des Trajektoriezugs entspricht.

Definition 3.4. *Sei T ein Streckenzug mit $|T| \geq 2$, dann definieren wir*

$$\delta_T : [0, |T| - 1] \to \mathbb{R}$$

$$\delta_T(x) := \begin{cases} d(T(0), T(1)) & x = 0 \\ d(T(|T|-2), T(|T|-1)) & x = |T| - 1 \\ max(d(T(x-1), T(x)),\ d(T(x), T(x+1))) & x \in \{1, 2, ..., |T| - 2\} \\ \delta_T(\lfloor x \rfloor) & sonst \end{cases}$$

als variables δ zu T.
Dabei bezeichnet d eine Abstandsfunktion. Wir verwenden dazu Orthodrome, auch bekannt als great-cirlce distance.

Auf Basis des variablen δ können wir eine Abwandlung des Entscheidungsproblems der Fréchet-Distanz definieren.

Definition 3.5. *Seien T, R Streckenzüge und δ_T das zu T gehörige variable δ. Dann bezeichnet*

$$F^{\delta}_{T,R} := \exists \alpha, \beta \, \forall x \in [0, 1] : ||T(\alpha(x)) - R(\beta(x))||_2 \leq \delta(x)$$

die boolesche Funktion zum Entscheidungsproblem der Fréchet-Distanz mit variablem δ. Dabei sind α und β Reparametrisierungen von T bzw. R.

3.1.3 Berechnen der Abbildung

Im Folgenden geben wir einen Algorithmus an, der zu einem Trajektoriezug T und einem Routenzug R eine zu $f_{T,R}$ ähnliche Abbildung $g^{\delta}_{T,R}$ berechnet, die unseren Anforderungen für die weiteren Algorithmen genügt.

Definition 3.6. *Seien T, R Streckenzüge, wobei $|T| \geq 2$ und sei δ_T das variable δ zu T, dann ist*

$$g^{\delta}_{T,R} : \{0, 1, ..., |T| - 1\} \to 2^{I^{[0,|R|-1]}}$$

eine Abbildung der definierenden Punkte von T auf Abschnitte von R mit folgenden Eigenschaften:

(I) $\forall x \in \{0, 1, ..., |T| - 1\}\ \forall y \in \left(\bigcup_{I \in g^{\delta}_{T,R}(x)} I\right) : ||T(x) - R(y)||_2 \leq \delta_T(x)$

(II) $\forall x \in \{0, 1, ..., |T| - 1\}\ \forall y \in \left(\bigcup_{I \in g^{\delta}_{T,R}(x)} I\right) : F^{\delta}(T|_{[0,x]}, R|_{[0,y]})$

Für einen Punkt $k \in \{0, 1, ...|T| - 1\}$ des Trajektoriezugs enthält $g^{\delta}_{T,R}(k)$ alle Abschnitte in R, die mit T rund um k zusammen verlaufen. Gilt $g^{\delta}_{T,R}(k) = \emptyset$, so verlaufen T und R rund um den Punkt k nicht zusammen oder k liegt nach einem Punkt, an dem T und R nicht zusammen verlaufen. Aus Bedingung (II) folgt direkt, dass, wenn für einen Punkt x gilt $g^{\delta}_{T,R}(x) = \emptyset$, für alle weiteren Punkte $y > x$ folgt $g^{\delta}_{T,R}(y) = \emptyset$ $(x, y \in \{0, 1, ..., |T| - 1\})$.
In seltenen Fällen bildet $g^{\delta}_{T,R}(k)$ auf mehrere Intervalle ab. Dies bedeutet, dass nicht eindeutig ist, welches Stück von R zusammen mit T um den Punkt k verläuft. $g^{\delta}_{T,R}(k)$ enthält dann alle möglichen Stücke. Algorithmus 3.1 bestimmt $g^{\delta}_{T,R}$. Dazu gehen wir ähnlich vor, wie in Algorithmus 2.1. Zunächst berechnen wir alle Kanten des free-space Diagramms, allerdings unter Berücksichtigung des variablen δ_T. Für die Kante $L^F_{i,j}$ wird um den Punkt $T(i)$ eine Kugel mit Radius $\delta_T(i)$ gelegt und diese mit der Strecke $S[R(j), R(j + 1)]$ geschnitten. Für die Kante $B^F_{i,j}$ wird analog dazu um den Punkt $R(j)$ eine Kugel mit Radius $\delta_T(i)$ gelegt und diese mit der Strecke $S[T(i), T(i + 1)]$ geschnitten.
Anschließend wird genau wie in Algorithmus 2.1 das reachable Diagramm berechnet. Für einen Punkt $k \in \{0, 1, ...|T| - 1\}$ des Trajektoriezugs kann $g^{\delta}_{T,R}(k)$ dann direkt aus der k-ten Spalte des reachable Diagramms $L^R_{k,j}$ gelesen werden. Alle zusammenhängenden Bereiche bilden je ein Intervall in $g^{\delta}_{T,R}(k)$. Durch die Konstruktion des free-space Diagramms mit dem variablen δ gilt Bedingung (I). Die Bedingung (II) folgt aus der Konstruktion des reachable Diagramms.

3.2 Bewertung von Routingdaten

Basierend auf der Abbildung g wird nun versucht, Fehler in den Routing-Services zu finden. Zunächst geht es dabei um fehlende Kanten. Aus den Zwischenergebnissen der Berechnungen versuchen wir danach überschüssige Kanten in den Routingdaten zu finden. Anschließend stellen wir einen Algorithmus vor um Fehler in der Metrik zu finden. Der erste Algorithmus arbeitet dabei auf genau einem Trajektoriezug und der zweite auf genau einer Trajektorie, deshalb wird im Anschluss beschrieben, wie beide Algorithmen kombiniert werden können. Zuletzt wird vorgestellt, wie die Ergebnisse, die durch eine Menge an Trajektorien gewonnen wurden, geclustert werden.

Algorithm 3.1: CALCULATE MATCHINGS

Input: Trajectory-Chain $T : List$ of length t, Route-Chain $R : List$ of length r, Deviation $\delta_T : List$ of length t
Data: Matrices $L^F, L^R \in t \times (r-1)$, $B^F, B^R \in (t-1) \times r$
Output: Function $f : List$

```
// Calculation of the free-space Diagram
```
foreach $(i,j) \in (0..t) \times (0..r-1)$ **do**
  $L^F_{i,j} \leftarrow$ computeFreeSpace$(T[i], R[j], R[j+1], \delta[i])$

foreach $(i,j) \in (0..t-1) \times (0..r)$ **do**
  $B^F_{i,j} \leftarrow$ computeFreeSpace$(R[j], T[i], T[i+1], \delta[i])$

```
// Initializing the left and bottom border of the reachable Diagram
```
for $i \leftarrow 0..t-1$ **do**
  $B^R_{i,0} \leftarrow$ initializeReachable()

for $j \leftarrow 0..r-1$ **do**
  $L^R_{0,j} \leftarrow$ initializeReachable()

```
// Calculation of the reachable Diagram
```
for $i \leftarrow 0..t-1$ **do**
  for $j \leftarrow 0..r-1$ **do**
    $L^R_{i+1,j}, B^R_{i,j+1} \leftarrow$ constructReachable$(L^R_{i,j}, B^R_{i,j}, L^F_{i+1,j}, B^F_{i,j+1})$

```
// Calculation of the Matches
```
for $i \leftarrow 0..t$ **do**
  $f[i] \leftarrow$ constructMatching(L^R_i)

Algorithm 3.2: CALCULATE TOPOLOGICAL ERRORS

Input: Trajectory-Chain $T : List$ of length t, Deviation $\delta_T : List$ of length t
Output: Errorpoints $E : List$

```
1  R ← calculateRoute(T[0], T[t − 1])
2  g ← Calculate Matchings(T, R, δ)
3  g_rev ← Calculate Matchings(T.reverse(), R.reverse(), δ_T.reverse())
4  if ∄k ∈ {0, 1, ..., t − 1} : g(k) = ∅ ∨ g_rev(k) = ∅ then
5      return ∅
6  if t = 2 then
7      return {T[0]}
8  Split1 ← min_{k∈{0,1,...,t−1}} k : g(k) = ∅
9  Split2 ← t − 1 − (min_{k∈{0,1,...,t−1}} k : g_rev(k) = ∅)
10 Mid ← (Split1 + Split2)/2
11 Start, End ← T.split(Mid)
12 δ_T^1, δ_T^2 ← δ_T.split(Mid)
13 Points1 ←Calculate Topological Errors(Start, δ_T^1)
14 Points2 ←Calculate Topological Errors(End, δ_T^2)
15 return Points1 + Points2
```

3.2.1 Bestimmen von fehlenden Kanten

Zur Bestimmung fehlender Kanten in den Routingdaten nutzen wir Algorithmus 3.2, der eine Liste mit Punkten berechnet, an denen eine Kante fehlen könnte. Dieser erhält als Eingabe einen Trajektoriezug T sowie das zugehörige δ_T. Bei einem Routingorakel r wird zu T eine Route vom ersten bis zum letzten Punkt von T angefragt. Die Route wird direkt in einen Routenzug R umgewandelt.
Da T, R und δ_T im Folgenden fest sind, werden wir, der Kürze halber, die Abbildung $g^{\delta}_{T,R}$ nur als g bezeichnen. Mit Algorithmus 3.1 berechnen wir g. Zusätzlich berechnen wir g_{rev}, indem wir Algorithmus 3.1 mit den invertierten Streckenzügen als Eingabe aufrufen.
Gilt für jeden Punkt $k \in \{0, 1, ..., |T| - 1\}$ mit $g(k) \neq \emptyset$ und $g_{rev}(k) \neq \emptyset$, so stimmen T und R in Bezug auf δ_T überein. In diesem Fall können wir keine fehlende Kante bestimmen. Mit g bestimmen wir den ersten Punkt k, an dem T und R nicht mehr zusammen verlaufen, indem wir das kleinste k suchen, an dem gilt $g(k) = \emptyset$. Genauso bestimmen wir mit g_{rev} den letzten Punkt k_{rev}, ab dem T und R wieder zusammen verlaufen. Dadurch haben wir bestimmt, in welchem Bereich T und R nicht zusammen verlaufen. In der Mitte dieses Bereichs unterteilen wir nun T und setzen dies rekursiv fort, bis wir T komplett durch Routen nachgebildet haben. In diesem Fall haben wir eine Komprimierung für T, wie in Abschnitt 3.1.1 besprochen, gefunden.
Wenn T und R nicht übereinstimmen und $|T| = 2$ gilt, so können wir T nicht weiter unterteilen. In diesem Fall besteht hier die Möglichkeit einer fehlenden Kante. Daher fügen wir den Punkt $T(0)$ in die Liste mit Fehlerpunkten ein.

3.2.2 Bestimmen von überschüssigen Kanten

Mit Algorithmus 3.2 können fehlende Kanten im Graphen eines Routing-Services bestimmt werden. Mit einer kleinen Erweiterung können auch Kanten gefunden werden, die fälschlicherweise im Graphen enthalten sind und verbotene Manöver beschreiben. Dazu gehören zum Beispiel verbotene Abbiegevorgänge oder das Überfahren von durchgezogenen Linien. Sei T ein Trajektoriezug und R ein Routenzug, der vom ersten bis zum letzten Punkt von T angefragt wurde. Enthalte R außerdem ein verbotenes Manöver. Wenn ansonsten keine Fehler in der Route sind, dann liegen die in Algorithmus 3.2 berechneten Punkte $Split1$

Algorithm 3.3: CALCULATE METRIC ERRORS

Input: Trajectory $T : List$ of length t, Deviation $\delta_T : List$ of lenght t, MinError ϵ
Output: Errorpoints $E : List$

```
1  R ← calculateRoute(T[0], T[t − 1])
   // Conversion of T, R to Trajectory-Chains is implied here.
2  g ← Calculate Matchings(T, R, δ_T)
3  for i ← 0..t − 1 do
4      Δa ← getTime(T[i + 1]) − getTime(T[i])
5      foreach I1 ∈ g(i) do
6          foreach I2 ∈ g(i + 1) do
7              Δp ← calculateTime(R, I2) − calculateTime(R, I1)
8              SymmetricError ← (Δa − Δp)/max(Δa, Δp)
9              if |SymmetricError| > ε then
10                 E ← E + T[i]
```

und $Split2$ unmittelbar vor bzw. nach dem verbotenen Manöver. Daher berechnen wir eine zusätzliche Liste mit Fehlerpunkten, die alle Punkte $Split1$ enthält für die die Distanz zu $Split2$ höchsten einem festen ϵ entspricht. Wir wählen $\epsilon = 200\,Meter$.

3.2.3 Bestimmen von Fehlern der Metrik

Zur Berechnung von fehlerhaften Kantengewichten wird nun zu einer Trajektorie T eine Route R bei einem Routingorakel angefragt. Die mit Algorithmus 3.1 berechnete Abbildung $g^{\delta}_{T,R}$ zwischen den zugehörigen Streckenzügen übertragen wir auf die Punktefolgen und bezeichnen sie der Kürze halber nur als g. Der folgende Algorithmus 3.3 betrachtet nur Trajektorien und Routen, die entsprechend der Abbildung übereinstimmen.
Es wird zunächst davon ausgegangen, dass $\forall i : |g(i)| = 1$ gilt. Für einen Punkt k der Trajektorie wird der zeitliche Abstand Δa zum nächsten Punkt berechnet. Für beide Punkte wird der Mittelpunkt von $g(k)$ bzw. $g(k+1)$berechnet und dort die Zeit der Route abgelesen, ggf. muss diese interpoliert werden. Die Differenz der beiden Routenzeiten nennen wir Δp. Es gilt immer $\Delta a, \Delta p \geq 0$. Wir berechnen den symmetrischen Fehler $s := (\Delta a - \Delta p)/max(\Delta a, \Delta p)$. Dessen Aussagekraft ist als relativer Fehler unabhängig von der jeweiligen Fahrtdauer und beschränkt auf das Intervall $[-1, 1]$. Außerdem berücksichtigt der symmetrische Fehler halb so lange Vorhersagen genau so stark wie doppelt so lange Vorhersagen. Diese unterscheiden sich lediglich im Vorzeichen. Ist das Auto ungefähr so schnell gefahren, wie von der Route vorhergesagt, so gilt $s \approx 0$. Gilt dies nicht, so merken wir uns k als Fehlerpunkt, sowie Δa und Δp zur späteren Verarbeitung. Als Grenze ϵ für Fehlerpunkte wählen wir $|s| > 0{,}2$.
Gibt es für einen Punkt mehrere Intervalle, so wird jede Kombination mit dem nachfolgenden Punkt betrachtet.

3.2.4 Zusammenfügen der Algorithmen

Für unsere Implementierung haben wir Algorithmus 3.3 in Algorithmus 3.2 integriert. Dabei wird, immer wenn in einem rekursiven Aufruf ein Trajektorie- und ein Routenzug übereinstimmen, eine abgewandelte Version von Algorithmus 3.3 aufgerufen, die einen Routenzug als Parameter erhält und nicht über ein Routingorakel anfragt. Mit der in Abschnitt 3.2.2 beschriebenen Erweiterung von Algorithmus 3.2 erhalten wir somit drei Listen mit Koordinatenpunkten, eine für mögliche fehlende Kanten, eine für mögliche fehlerhafte Kantengewichte und eine für mögliche überschüssige Kanten. Alle Punkte

konvertieren wir in das WGS84-System. Zusätzlich wird ausgegeben, wie viele Anfragen an das Routingorakel nötig waren, um einen Trajektoriezug durch Routenzüge nachzubilden, bzw. falls eine Nachbildung nicht möglich war, wie viele Anfragen benötigt wurden um alle Fehlerpunkte zu finden.

3.2.5 Clustern der Fehlerpunkte

Bisher wurde jede Trajektorie für sich alleine betrachtet. Im Folgenden werden immer die mit einer Menge von Trajektorien berechneten Fehlerpunkte betrachtetet. Ein Fehlerpunkt alleine weist noch nicht auf einen Fehler in den Routingdaten hin, vielmehr kann ein Auto zu schnell gefahren oder an einer Stelle abgebogen sein, wo dies nicht erlaubt ist. Erst wenn viele Trajektorien einen Fehlerpunkt an dieser Stelle liefern, kann ein Fehler in den Routingdaten erwartet werden. Daher ist das Clustern der Fehlerpunkte notwendig.
Von den in Abschnitt 3.2.4 zusammengefügten Algorithmen erhalten wir drei Mengen mit Koordinatenpunkten. Diese betrachten wir im Folgenden getrennt voneinander. Um Cluster in den Punkten zu finden, verwenden wir den DBSCAN-Algorithmus [EKSX96]. Dieser erhält zusätzlich zu den Fehlerpunkten P zwei Parameter ϵ und $MinPts$. Zu einem Punkt $p \in P$ wird die ϵ-Nachbarschaft $N_\epsilon(p) = \{q \in P \mid distanz(p,q) \leq \epsilon\}$ berechnet. Als Distanzfunktion verwenden wir Orthodrome, auch bekannt als great-circle distance. Ist $|N_\epsilon(p)| \geq MinPts$, so liegt p in einem Cluster. Das Cluster wird rekursiv auf alle Punkte $q \in N_\epsilon(p)$ ausgeweitet. Gilt auch $|N_\epsilon(q)| \geq MinPts$, so wird q dem Cluster zugefügt und die rekursive Erweiterung auf alle Punkte $r \in N_\epsilon(q)$ fortgesetzt. Alle Punkte, für die die ϵ-Nachbarschaft überprüft wird, werden als besucht markiert. Ist die rekursive Erweiterung eines Clusters abgeschlossen, so wird mit einem neuen unmarkierten Punkt gestartet. Dies wird so lange fortgesetzt, bis alle Punkte aus P markiert sind.
Wir erweitern den DBSCAN um eine nachträgliche Filterung der Cluster. Da für eine Trajektorie mehrere Fehlerpunkte berechnet werden können, kann es Cluster geben, die nur aus Punkten von einer oder weniger Trajektorien bestehen. Wir filtern daher alle gefundenen Cluster, ob von mindestens $minTrjs$ verschiedenen Trajektorien Punkte in dem Cluster enthalten sind.
Somit ergeben sich drei Parameter $\epsilon, minPts, minTrjs$ für das Clustern. Diese müssen wir abhängig von der Größe der Eingabedaten wählen, daher werden diese in Kapitel 4 besprochen.
Der abgewandelte Algorithmus 3.2 berechnet zwei Mengen an Fehlerpunkten. Diese clustern wir einzeln mit unserem verändert DBSCAN. Eine weitere Bearbeitung ist nicht notwendig. Die Zeitabweichungen, die durch Algorithmus 3.3 bestimmt wurden, clustern wir mit unserem veränderten DBSCAN. Anschließend filtern wir die Cluster noch weiter. Interessant sind für uns nur Cluster, in denen die Abweichung in eine Richtung geht. Ein Cluster, in dem die Hälfte der Autos schneller war als von der Route bestimmt und die andere Hälfte langsamer, zeugt nicht von einem Fehler in der Metrik, sondern von unterschiedlichen Verkehrssituationen. Zu jedem Fehlerpunkt haben wir Δa und Δp gespeichert. Nun können wir wieder den symmetrischen Fehler $s := (\Delta a - \Delta p)/max(\Delta a, \Delta p)$ berechnen. Wie oben beschrieben, hat dieser gegenüber einem normalen relativen Fehler den Vorteil, dass er auf $[-1, 1]$ beschränkt ist und eine halb so lange Vorhersage genau so schlecht ist wie eine doppelt so lange Vorhersage. Für alle in dem Cluster enthaltenen Trajektorien T berechnen wir das mittlere s, s_T. Über alle Trajektorien des Clusters können wir nun das mittlere s_T, s_{mean} berechnen. Durch den Zwischenschritt, s_T zu berechnen, geht jede Trajektorie gleich stark in die Abweichung des Clusters ein. Gilt $|s_{mean}| > 0{,}2$, so erwarten wir an diesem Cluster einen Fehler in der Metrik. Ist $s_{mean} < 0$, so fahren die Autos schneller als erwartet. Wir bezeichnen diese Cluster als negative Cluster. Ist $s_{mean} > 0$, so fahren die Autos langsamer als erwartet und wir bezeichnen die Cluster als positiv.

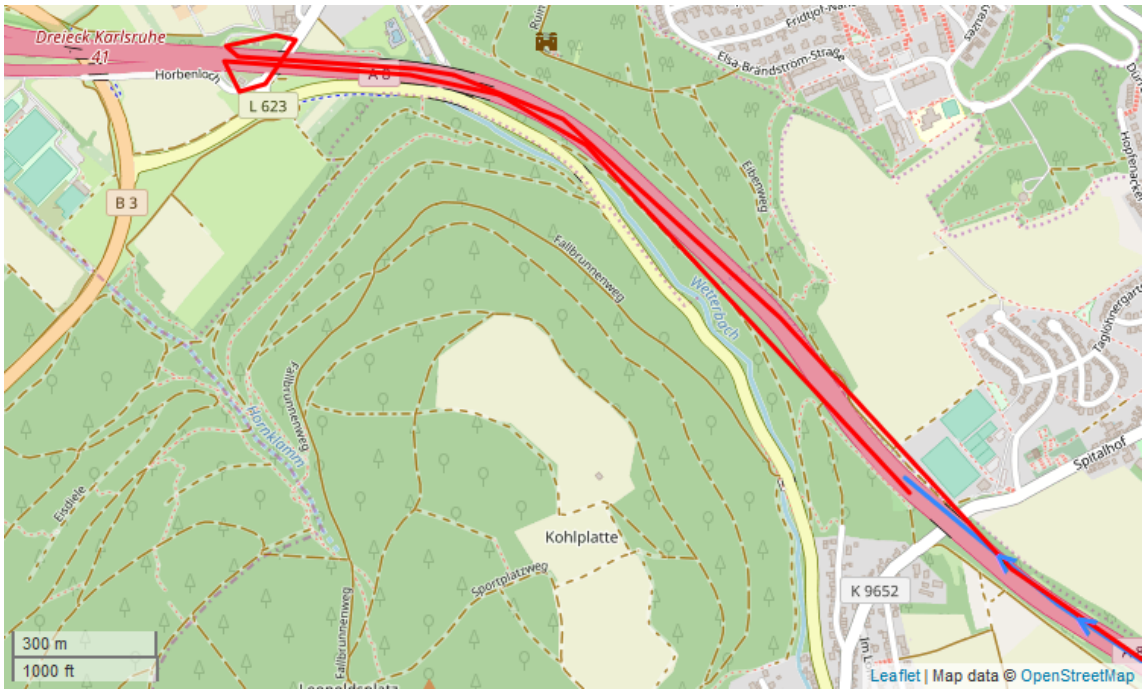


Abbildung 3.2: Eine Route, die über die Trajektorie hinausgeht

3.3 Probleme in der Praxis

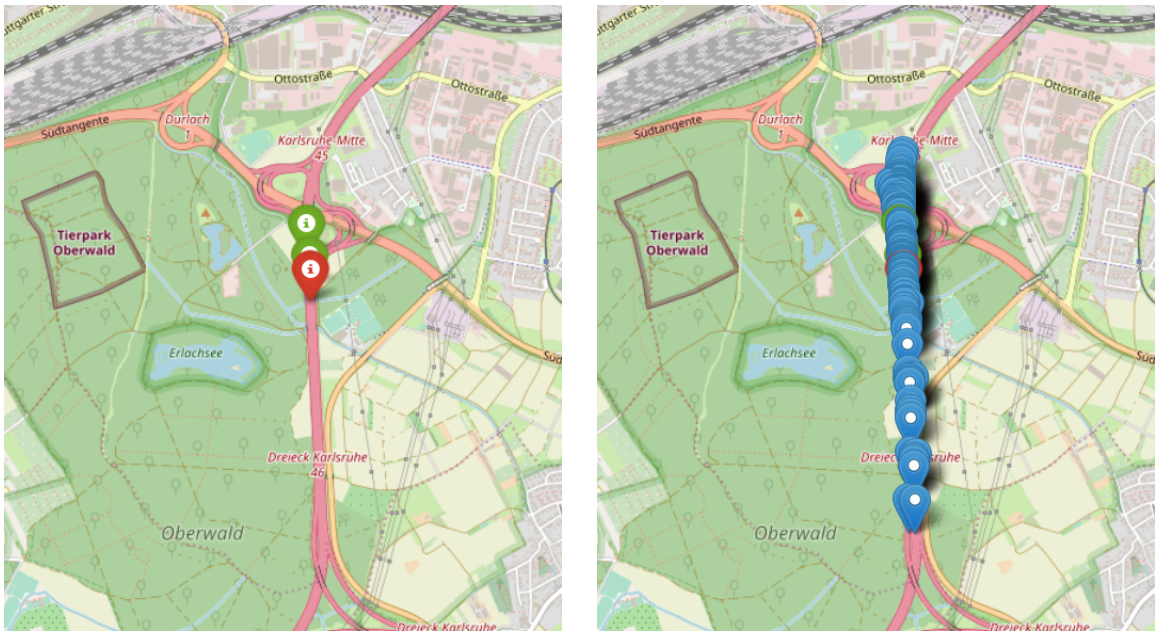
Bisher sind wir in diesem Kapitel von einem Routingorakel ausgegangen, das zu zwei Punkten immer eine verbindende Route berechnen kann. In der Praxis ist dies jedoch nicht immer der Fall und wir müssen für Algorithmus 3.2 Strategien definieren, um mit verschiedenen Fehlern umzugehen.
Falls das Routingorakel zu einer Trajektorie keine Route berechnen kann, so unterteilen wir die Trajektorie in der Mitte und setzten die Rekursion fort. Sollte die Trajektorie nicht weiter unterteilt werden können, so fügen wir wie gehabt einen Fehlerpunkt in die Liste ein. Sollte das Routingorakel zu einer Trajektorie eine Route liefern, deren Start- und Zielpunkt jeweils über 5 km vom Start bzw. Ziel entfernt ist, so wird die Berechnung für diese Trajektorie abgebrochen. Dies kann passieren, wenn das Routingorakel keine Routingdaten im Bereich der Trajektorie kennt und deshalb von der nächsten bekannten Straße aus das Routing startet.
Ein weiteres Problem, das auftreten kann, ist, dass jeder Punkt der Trajektorie auf einen Teil der Route abgebildet werden kann, die Route jedoch über die Trajektorie hinausgeht. Abbildung 3.2 zeigt, dass dies passieren kann, wenn das Routingorakel eine Route für die falsche Autobahnrichtung berechnet. In blau ist die Trajektorie dargestellt und in rot die Route. In diesem Fall beenden wir die Berechnung, da die Trajektorie komplett nachgebildet wurde und wir entsprechend keinen Fehlerpunkt finden können.

4. Evaluation

Im folgenden Kapitel wird zunächst experimentell überprüft, wie gut der Algorithmus bestimmte Fehler in Routing-Services finden kann. Anschließend wird der Algorithmus auf mehrere Routing-Services angewendet.

4.1 Testumgebung

Zur Evaluation haben wir lokal einen Routing-Service aufgesetzt. Dieser benutzt CCHs zur schnellen Berechnung der kürzesten Wege. Die an den Service übergebenen Koordinaten werden jeweils auf den nächsten Knoten abgebildet. Zur Verbesserung der Abbildung werden auf langen Kanten zusätzliche Knoten eingefügt, sodass zwischen zwei Knoten maximal 100 Meter liegen. Wir benutzen zwei verschieden Graphen, die wir als Routingdaten verwenden. Erstens benutzen wir einen Graphen von HERE WeGo in der Version 2016R4, der das westeuropäische Straßennetz abbildet und uns von der Firma BMW zu wissenschaftlichen Zwecken zur Verfügung gestellt wurde. Leider ist der Graph nicht öffentlich verfügbar. In diesem Graphen ist für jede Kante eine von fünf Straßenklassen hinterlegt. Dadurch können wir Autobahnkanten identifizieren. Insgesamt sind $0,75\%$ der Kanten Autobahnkanten. Zweitens benutzen wir einen Graphen der 9th DIMACS Implementation Challenge [9th]. Wir verwenden den Europa Graphen, der ursprünglich von der PTV AG zur Verfügung gestellt wurde. Auch dieser Graph bildet das westeuropäische Straßennetz ab. Beide Graphen sind mit einer free-flow-Metrik ausgestattet, gehen also von optimalen Verkehrsbedingungen aus.
Als Trajektorien haben wir einen Satz mit 266332 Fahrten aus Europa verwendet. Auch diese wurden uns von der Firma BMW zu wissenschaftlichen Zwecken zur Verfügung gestellt und sind nicht öffentlich verfügbar. Zur Anonymisierung der Daten wurde von jeder Fahrt jeweils die erste und die letzte Viertelstunde gelöscht. Eine Fahrt aus diesem Satz dauert im Schnitt etwas 8,6 Minuten. Jede Trajektorie besteht aus mindestens drei Koordinatenpunkten. Zwischen zwei aufeinanderfolgenden Punkten liegen mindestens 20 Meter und es vergehen im Schnitt etwa 13,4 Sekunden, aber maximal 60 Sekunden. Die Trajektorien decken hauptsächlich Autobahnen ab, daher konzentrieren wir uns in der Evaluation auf diese.
Als Testsystem wurde eine Maschine mit einer `Intel(R) Xeon(R) CPU E5-1630 v3 @ 3.70GHz` Quadcore-CPU mit `128 GB` Hauptspeicher verwendet. Auf der Maschine läuft ein `openSUSE Leap 15.0` als Betriebssystem. Unser Algorithmus ist in Rust implementiert und wurde mit dem Compiler `rustc 1.34.0-nightly` mit der Option `--release` übersetzt.



(a) Grün: Endpunkte der gelöschten Kante, Rot: Mittelpunkt des Clusters

(b) Blau: Alle gefundenen Fehlerpunkte

Abbildung 4.1: Finden einer gelöschten Kante der A5 zwischen Karlsruhe Mitte und Dreieck Karlsruhe

4.2 Evaluation des Algorithmus

Zur Evaluation des Algorithmus haben wir den Graphen von HERE WeGo eingesetzt. In den Graphen wurden gezielt Fehler eingebaut, die der Algorithmus finden soll. Dazu wurden getrennt Tests ausgeführt zu fehlenden Kanten, zusätzlichen Kanten und Zeitabweichungen.

4.2.1 Fehlende Kanten

Um fehlende Kanten zu finden, werden, wie in Abschnitt 3.2.5 beschrieben, die vom Algorithmus 3.2 berechneten Fehlerpunkte mit einem veränderten DBSCAN geclustert. Für diese Tests haben wir als minimale Größe für Nachbarschaften fünf und als maximalen Abstand für Punkte 200 Meter festgelegt. Zusätzlich sollen mindestens fünf verschiedene Trajektorien in jedem Cluster enthalten sein.
Als ersten exemplarischen Test haben wir eine Kante der A5 in der Nähe von Karlsruhe gelöscht. Abbildung 4.1(a) zeigt mit grünen Markierungen die Endpunkte der Kante. Die Fehlerpunkte, die der Algorithmus ausgegeben hat, wurden geclustert. In Rot ist der Mittelpunkt des Clusters zu sehen. Alle Fehlerpunkte des Clusters sind in 4.1(b) abgebildet. Offensichtlich ist der Algorithmus also in der Lage, die gelöschte Kante zu identifizieren.
Anschließend haben wir zufällige Kanten gelöscht und überprüft, wie viele der Kanten unser Algorithmus findet. Dazu haben wir zunächst einen Durchlauf auf einem unveränderten Graphen gestartet, um herauszufinden, wie viele Cluster mit möglichen fehlenden Kanten von Anfang an gefunden werden. Es werden 143 Cluster gefunden. Wie oben beschrieben, enthält unser Datensatz an Trajektorien hauptsächlich Autobahnfahrten. Deswegen löschen wir nur Autobahnkanten. Dabei haben wir zunächst alle Kanten betrachtet und anschließend nur Kanten, die von Knoten ausgehen, in deren Umkreis von 500 Metern Punkte von mindestens fünf verschiedenen Trajektorien liegen. Für beide Versuche haben wir in mehreren Ausführungen zwischen 0,25% und 5% der Kanten gelöscht. Tabelle 4.1 zeigt die Ergebnisse. Unser Algorithmus konnte jeweils anteilig mehr Kanten finden, wenn wir nur

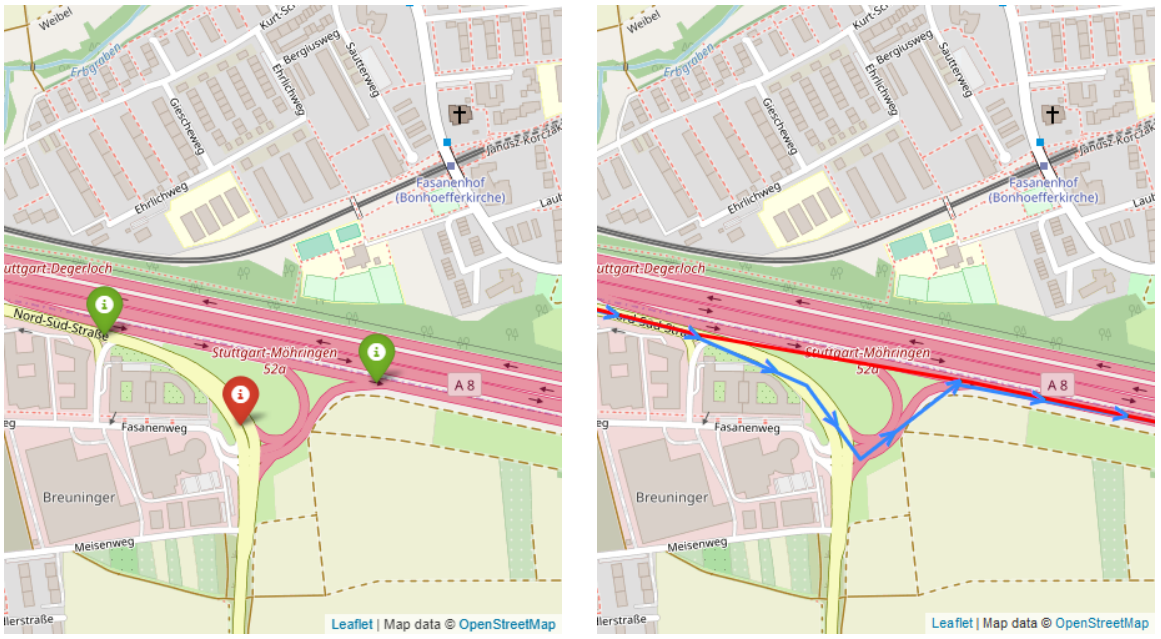
Tabelle 4.1: Übersicht der gefundenen gelöschten Autobahnkanten.

Betrachtete Knoten	Prozent gelöscht	Gefundene Cluster	Neue Cluster	Gelöschte Kanten	Prozent gefunden
Alle	0,25	740	597	1878	31,79
Alle	0,5	1329	1186	3757	31,57
Alle	1	2543	2400	7515	31,94
Alle	2	4105	3962	15031	26,36
Alle	3	5249	5106	22546	22,65
Alle	4	6077	5934	30062	19,74
Alle	5	6568	6425	37578	17,10
Nah an Trajektorien	0,25	740	597	999	59,76
Nah an Trajektorien	0,5	1347	1204	1998	60,26
Nah an Trajektorien	1	2396	2253	3996	56,38
Nah an Trajektorien	2	4134	3991	7993	49,93
Nah an Trajektorien	3	5170	5027	11989	41,93
Nah an Trajektorien	4	5971	5828	15986	36,46
Nah an Trajektorien	5	6369	6226	19983	31,16

Kanten betrachtet haben, die von Knoten ausgehen, in deren Nähe Trajektorien verlaufen. Dies entspricht unseren Erwartungen, da unsere Clustering-Strategie voraussetzt, dass ein Fehler mit mindesten fünf verschiedenen Trajektorien gefunden wird. Insgesamt konnten wir bis zu 60% der gelöschten Kanten finden. Interessant ist, dass wir anteilig deutlich weniger der gelöschten Kanten finden, wenn wir mehr Kanten löschen. Dies liegt daran, dass die berechneten Fehlerpunkte für gelöschte Kanten, die nah beieinander liegen, zu einem Cluster zusammengefasst werden. Je mehr Kanten wir löschen, desto höher ist die Wahrscheinlichkeit, dass einige dieser Kanten nah beieinander liegen. Dadurch können wir anteilig nicht alle Kanten finden. Hinzu kommt, dass unsere Filterung der Knoten nicht sicherstellt, dass die Kanten, die wir löschen, tatsächlich Straßen entsprechen, die von mindestens fünf Trajektorien benutzt werden. Wir erwarten eine weitere Verbesserung des Ergebnisses, wenn mit den Trajektorien zuerst ein Map-Matching ausgeführt wird und nur Kanten gelöscht werden, die von mindestens fünf Trajektorien benutzt werden. Aus Zeitgründen können wir eine solche Evaluation leider nicht vornehmen.

4.2.2 Verbotene Manöver

Verbotene Manöver, die das Routing vorschlägt, entsprechen falschen Kanten im Graphen. Deswegen fügen wir in unseren Routinggraphen eine Kante ein. Zu Identifikation der überschüssigen Kante nutzen wir die Punktepaare, die Algorithmus 3.2 identifiziert, an denen eine Trajektorie und eine Route sich trennen, bzw. ab wo sie wieder zusammen verlaufen. Wie in Abschnitt 3.2.5 beschrieben, filtern wir die Menge der Paare, bei denen die beiden Punkte nah beieinander liegen. Anschließend clustern wir die Punkte mit unserem veränderten DBSCAN. Auch für diese Tests haben wir als minimale Größe für Nachbarschaften fünf, als maximalen Abstand für Punkte 200 Meter und als Mindestanzahl an Trajektorien pro Cluster fünf festgelegt.
Als Test haben wir eine Kante in der Nähe von Stuttgart hinzugefügt. Durch die Kante gibt es eine zusätzliche Auffahrt auf die A8. Abbildung 4.2(a) zeigt in Grün die Endpunkte der zusätzlichen Kante und in Rot den Mittelpunkt des gefundenen Clusters. In Abbildung 4.2(b) ist in Blau eine Trajektorie zu sehen, die die normale Autobahnauffahrt nutzt, in Rot ist die dazu angefragte Route zu sehen, die die neu eingefügte Kante benutzt. Der Algorithmus hat also einen Fehler in der Nähe unserer eingefügten Kante identifiziert.



(a) Grün: Endpunkte der hinzugefügten Kante, Rot: Mittelpunkt des Cluster

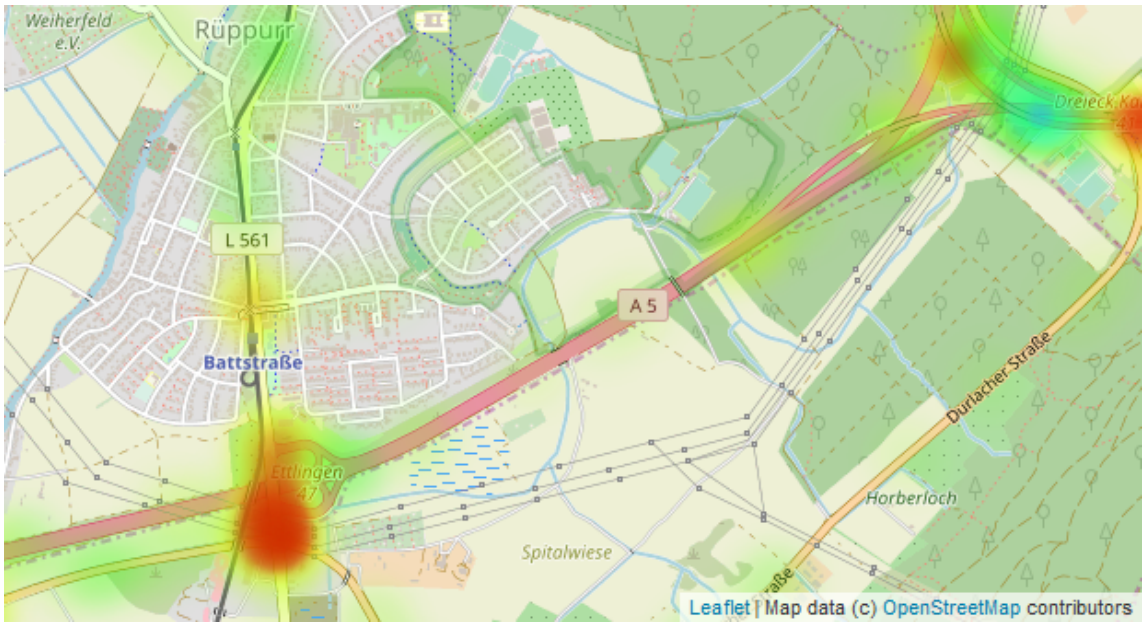
(b) Blau: Eine Trajektorie, die von der Nord-Süd-Straße auf die A8 auffährt, Rot: Die angefragte Route zu der Trajektorie

Abbildung 4.2: Finden einer hinzugefügten Auffahrt auf die A8 bei Stuttgart

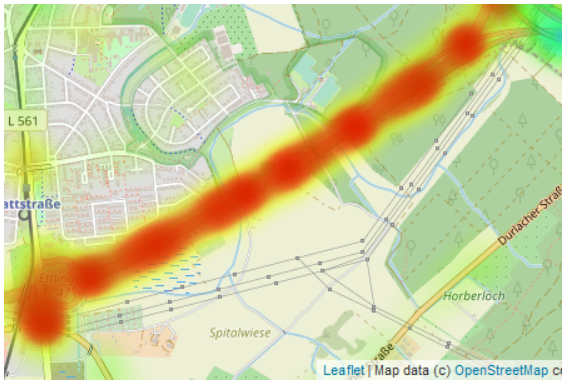
Die Identifizierung verbotener Manöver haben wir erst im Nachhinein aus Zwischenergebnissen unseres Algorithmus entwickelt. Daher haben wir keine experimentelle Evaluierung mit zufällig eingefügten Kanten vorgenommen.

4.2.3 Zeitabweichungen

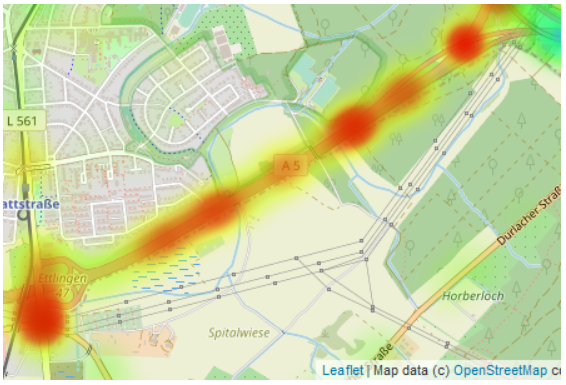
Die durch Algorithmus 3.3 berechneten Punkte mit möglichen Fehlern in der Metrik visualisieren wir mit einer Heatmap. Zu jedem Fehlerpunkt haben wir die gemessene Fahrtdauer Δa und die vorhergesagte Fahrtdauer Δp gespeichert. Nun berechnen wir für jeden Punkt die Abweichung $r := (\Delta a - \Delta p)$. Als Intensität des Punktes setzen wir 1, falls $r > 0$ bzw. -1 falls $r < 0$. Würden wir die Intensität von der Stärke der Zeitabweichung abhängig machen, so würden Bereiche mit geringen Zeitabweichungen aus zwei Gründen als weniger intensiv dargestellt werden: Erstens die geringere Intensität jeden Punktes und zweitens ist davon auszugehen, dass der Fehler mit weniger Trajektorien gefunden wird, es also weniger Punkte gibt. Ist $p > 0$, war also das Auto der Trajektorie langsamer als vom Routingorakel vorhergesagt, so sprechen wir von einer positiven Abweichung. Entsprechend bezeichnen wir Abweichungen mit $p < 0$ als negative Abweichungen. In der Heatmap visualisieren wir positive Abweichungen mit einem Spektrum von Gelb über Orange bis Rot und negative Abweichungen mit einem Spektrum von Grün über Cyan bis Blau. Abbildung 4.3(a) zeigt alle gefundenen Zeitabweichungen auf der A5 zwischen Ettlingen und Dreieck Karlsruhe. Hier gibt es kaum Abweichungen, die Autos fahren also fast alle etwa so schnell, wie unser Routingorakel dies vorhersagt. Für die Abbildungen 4.3(b) und 4.3(c) haben wir das Kantengewicht verringert. Daher verkürzt sich die vorhergesagte Fahrtdauer des Routingorakels und es kommt zu positiven Abweichungen. Entsprechend haben wir für die Heatmaps in den Abbildungen 4.3(d) und 4.3(e) die Kantengewichte vergrößert und es kommt zu positiven Abweichungen.
Gut sichtbar ist, dass durch die großen Veränderungen in den Abbildungen 4.3(b) und 4.3(d) von unserem Algorithmus entsprechend mehr Zeitabweichungen gefunden werden, aber auch die kleineren Abweichungen noch gefunden werden können.



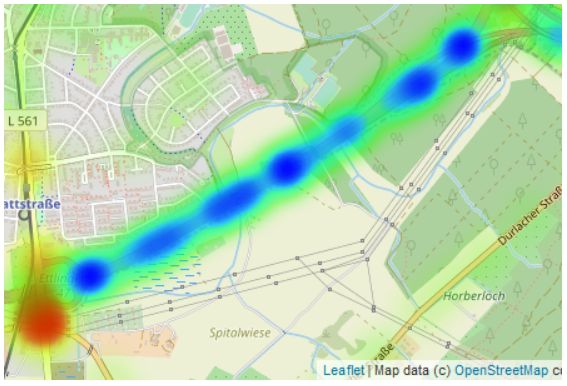
(a) Keine Veränderung der Kantengewichte



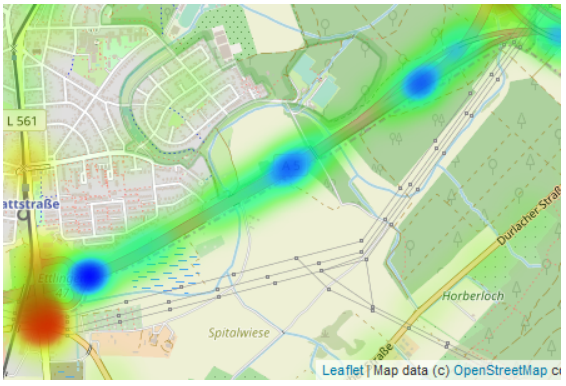
(b) Multiplizieren der Gewichte mit 0,5



(c) Multiplizieren der Gewichte mit 0,75



(d) Multiplizieren der Gewichte mit 1,5



(e) Multiplizieren der Gewichte mit 1,25

Abbildung 4.3: Heatmaps der gefundenen Zeitabweichungen auf der A5 zwischen Ettlingen und Dreieck Karlsruhe mit verschiedenen Veränderungen der Kantengewichte. In Gelb bis Rot sind positive und in Grün bis Blau negative Abweichungen dargestellt. Je intensiver die Farben sind, desto mehr Abweichungen wurden zusammengefasst.

Tabelle 4.2: Übersicht der gefundenen veränderten Kanten. Es wurden nur Knoten betrachtet, in deren Nähe mindestens fünf verschiedene Trajektorien verlaufen. Von diesen Knoten wurde das Gewicht von 1% der Autobahnen verändert. Zusätzlich wurden jeweils drei Folgekanten verändert.

Faktor	Differenz positive Cluster	Differenz negative Cluster	Veränderte Abschnitte	Prozent gefunden	Nachbildungen möglich
0,5	402	-166	3535	16,07	212717
1	0	0	0	-	212708
1,5	-193	297	3535	13,86	212664
2	-271	422	3535	19,60	212610
4	-338	394	3535	20,71	212005
6	-360	141	3535	14,17	211226
8	-392	11	3535	11,40	210544

Anschließend haben wir zufällige Fehler in die Kantengewichte eingebaut. Eine visuelle Analyse der Ergebnisse ist damit kaum möglich, daher haben wir die Fehlerpunkte, wie in Abschnitt 3.2.5 beschrieben, geclustert. Als Parameter haben wir $\epsilon = 200\,Meter$, $minPts = 5$, und $minTrjs = 5$ gewählt. Die anfängliche Ausführung mit dem unveränderten Graphen hat 12887 positive und 2260 negative Cluster ergeben. Zur Auswahl der Kanten haben wir, basierend auf den Erfahrungen mit fehlenden Kanten, direkt nur Autobahnkanten, ausgehend von Knoten, in deren Umkreis von 500 Meter mindestens fünf verschiedene Trajektorien verlaufen, benutzt. Von diesen Kanten haben wir zufällig 1% ausgewählt. Da der Here-Graph eine gute Modellierung des Straßenverlaufs enthält, beschreibt eine Kante jeweils nur ein kurzes Stück des Straße. Daher verändern wir nicht nur das Gewicht der ausgewählten Kanten, sondern auch der nächsten drei Autobahnkanten. Dabei wählen wir als nächste Kante, sofern diese existiert, immer die erste ausgehende Autobahnkante. Wenn sich zwei solche Abschnitte überschneiden, fassen wir diese als einen Abschnitt zusammen. Für einen Faktor größer als eins erwarten wir weniger positive Cluster und mehr negative Cluster, für Faktoren kleiner als eins entsprechend umgekehrt. Für unsere Experimente haben wir einen festen Satz an Kanten mit verschiedenen Faktoren multipliziert. Die Tabelle 4.2 zeigt die Ergebnisse. Für die Berechnung des Anteils der Veränderungen, der gefunden wurde, gehen wir davon aus, dass verschwundene Cluster der einen Richtung nicht zu Clustern der anderen Richtung geworden sind. Dabei wird sichtbar, dass wir kaum mehr als 20% der Veränderungen finden konnten. Dies lässt sich im Wesentlichen auf vier Gründe zurückführen. Wie schon in Abschnitt 4.2.1 besprochen, garantiert unsere Filterung der Knoten nicht, dass tatsächlich fünf Trajektorien die durch die Kante repräsentierte Straße benutzt haben und zusätzlich können nah beieinanderliegende Veränderungen zu einem Cluster zusammengefasst werden. Hinzu kommt nun, dass wir auch auf der unveränderten Karte bereits viele Zeitabweichungen finden. Wenn wir eine Kante mit einem Faktor größer als eins multiplizieren, deren Gewicht wir vorher schon als zu groß identifiziert hatten, so können wir keinen neuen Fehler entdecken. Weiter haben wir für die Filterung alle Trajektorien benutzt, für die Berechnung der Zeitabweichungen benutzen wir jedoch nur Trajektorien, die nachgebildet werden konnten. Abbildung 4.4 zeigt ein Beispiel, warum dies zum Problem werden kann. Dadurch, dass der Here-Graph bei Drusenheim keine Modellierung der A35 enthält, können die Trajektorien, die dort entlangführen, nicht nachgebildet werden und entsprechend kann unser Algorithmus keine Zeitabweichungen finden. Da zu erwarten ist, dass die meisten Trajektorien, die weiter nördlich bei den beiden anderen veränderten Abschnitten die A35 benutzen, auch weiter südlich dort fahren, können auch diese Veränderungen nicht gefunden werden.
Interessant ist weiterhin, dass bei einer starken Vergrößerung der Kantengewichte insgesamt

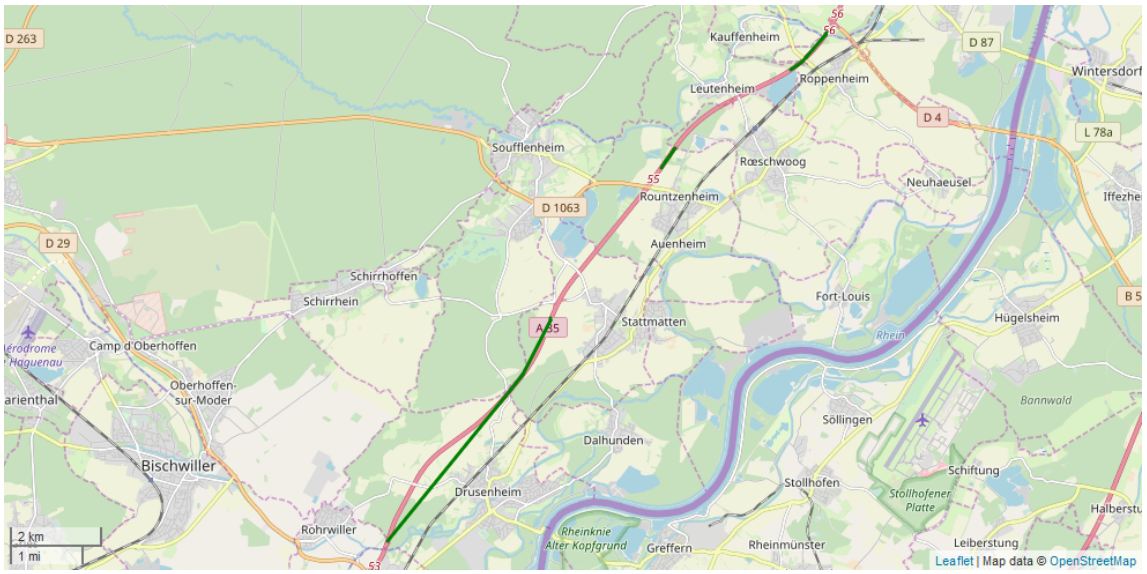


Abbildung 4.4: Drei zufällig ausgewählte Bereiche zur Veränderung des Kantengewichts auf der A35 zwischen Straßburg und Rastatt.

weniger negative Cluster gefunden werden. Dies lässt sich jedoch leicht verstehen mit der letzten Spalte der Tabelle. Sichtbar ist, dass unser Algorithmus wesentlich weniger Trajektorien durch Routen nachbilden kann, wenn für einzelne Bereiche das Kantengewicht stark erhöht wird. Dies liegt daran, dass der Routing-Service keine Routen mehr über einige dieser Kanten berechnet. Daran wird deutlich, dass die Analyse fehlender Kanten und fehlerhafter Kantengewichte nicht so klar trennbar ist, wie wir dies angenommen hatten. Weiter ist die genaue Unterscheidung der Fehler ohne explizites Wissen über den Graphen vermutlich schwierig.

4.2.4 Laufzeit

Die Laufzeit unseres Algorithmus ist stark von der Dauer der Anfragen an das Routingorakel abhängig. Dies gilt insbesondere dann, wenn eine Netzwerkanfrage zur Kommunikation mit dem Orakel notwendig ist. Daher haben wir die Dauer der Anfragen separat gemessen und anschließend von der Gesamtlaufzeit unseres Algorithmus für eine Trajektorie abgezogen. Abbildung 4.5 zeigt die Laufzeit in Abhängigkeit von der Länge der eingegeben Trajektorie, gemessen in der Anzahl der Koordinatenpunkte. Unterschieden wird zwischen Trajektorien, zu denen übereinstimmende Routen angefragt werden konnten und Trajektorien, für die dies nicht möglich war. Für beide Kategorien sind quadratische Regressionen dargestellt, die den Verlauf gut nachbilden. Im Schnitt ist die Laufzeit unseres Algorithmus also quadratisch in der Größe der eingegebenen Trajektorie. Gut sichtbar ist, dass die Laufzeit für Trajektorien, die nicht nachgebildet werden können, häufig höher ist, als für solche, die nachgebildet werden können. Dies liegt daran, dass meistens ein tieferer rekursiver Abstieg nötig ist, um festzustellen, dass ein Abschnitt nicht nachgebildet werden kann, als um eine Route zu finden, die einen Abschnitt nachbildet. Hinzu kommt, dass Trajektorien potenziell auch an mehreren Stellen nicht durch Routen nachgebildet werden können. Die theoretische Worst-Case-Laufzeit würde auftreten, wenn ein Routingorakel zu jedem Stück einer Trajektorie immer eine Route liefert, die zwar in der Nähe liegt, sodass die Berechnung nicht abgebrochen wird, jedoch kein Stück der Trajektorie nachbildet. Dies ist in unseren Daten nie aufgetreten und in einem realistischen Szenario auch sehr unwahrscheinlich. Daher haben wir eine Analyse der Worst-Case-Laufzeit nicht durchgeführt.
Als Nächstes haben wir die Laufzeit von Algorithmus 3.1 einzeln gemessen, da dieser den Grundbaustein unserer Anwendung bildet. Zusätzlich ist die Laufzeit hier lediglich von der

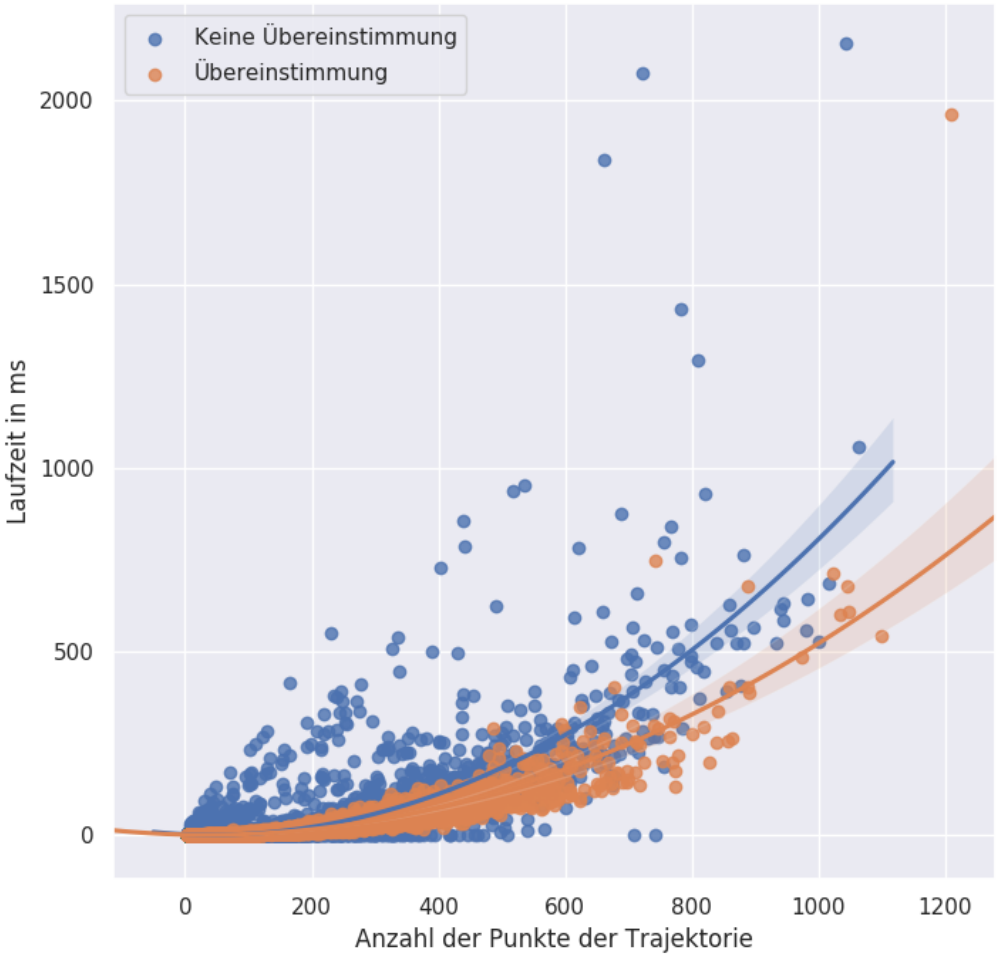


Abbildung 4.5: Laufzeit der kombinierten Algorithmen in Abhängigkeit von der Anzahl der Punkte der Trajektorie. Die Dauer der Anfragen an das Routingorakel wurde von der Laufzeit abgezogen. Zusätzlich sind quadratische Regressionen eingezeichnet.

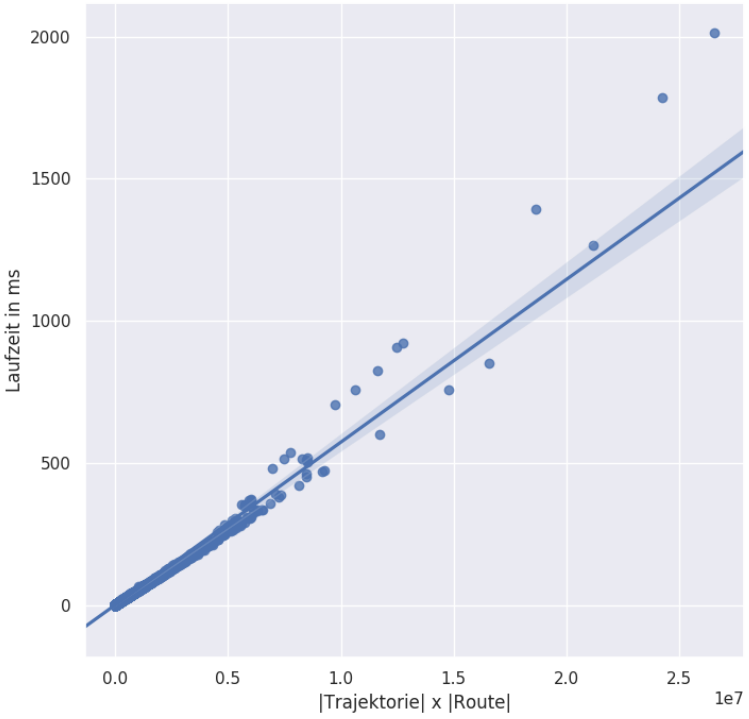
Größe der Eingabe abhängig und nicht davon, ob eine Nachbildung des Trajektoriezugs möglich ist. In Abbildung 4.6 ist die Laufzeit in Abhängigkeit der multiplizierten Länge des Trajektoriezugs $|T|$ und des Routenzugs $|R|$ dargestellt. In Abbildung 4.6(a) ist $|T| \times |R|$ linear skaliert. Anhand der Regressionsgerade ist gut sichtbar, dass Algorithmus 3.1 linear mit $|T| \times |R|$ skaliert. Durch die logarithmische Skalierung von $|T| \times |R|$ in Abbildung 4.6(b) wird deutlich, dass die Berechnung für die meisten Eingaben in wenigen Millisekunden abgeschlossen wird. Insgesamt wurde die Berechnung für 99% der Eingaben in weniger als zehn Millisekunden abgeschlossen.

4.3 Evaluation einiger Routing-Services

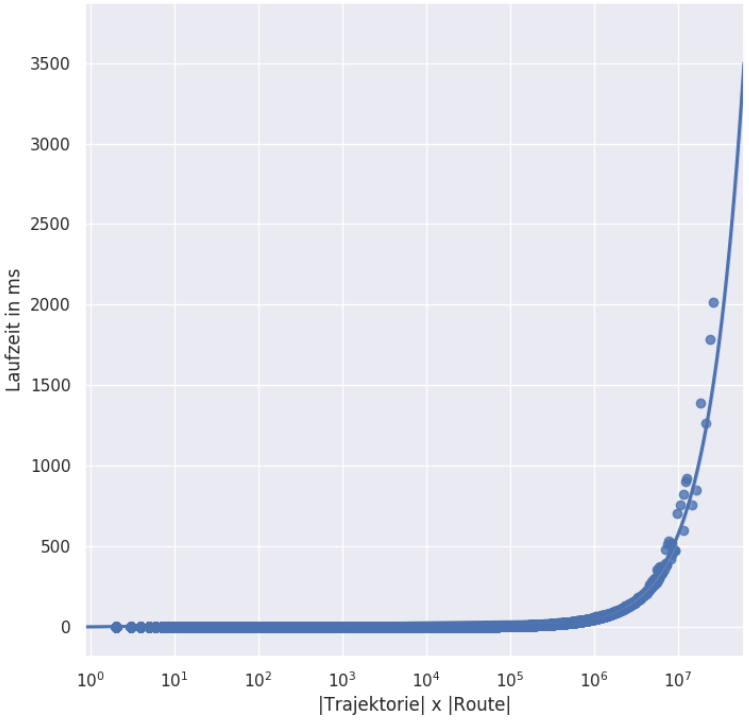
Exemplarisch diskutieren wir hier einige der gefundenen Fehler, die unser Tool auf unseren unmodifizierten Testdaten gefunden hat. Anschließend betrachten wir Abweichungen in der erwarteten und der erzielten Fahrtdauer. Zu beachten ist dabei, dass wir nur die Fahrtdauern von Routen und Trajektorien betrachten, die von unserem Algorithmus als übereinstimmend identifiziert worden sind, Abweichungen also nicht durch die Wahl anderer Straßen entstehen können. Zuletzt vergleichen wir die evaluierten Services.

4.3.1 Here-Graph

Als ersten Service evaluieren wir den auch zur Evaluation des Algorithmus eingesetzten Graphen von HERE WeGo mit einer free-flow-Metrik. Ohne Veränderungen des Graphen ergeben sich 143 Cluster mit möglichen fehlenden Kanten. Der größte Teil davon entsteht durch eine fehlende Modellierung des Straßenverlaufs an diesen Stellen im Graphen. Abbildung 4.7 zeigt eine Auswahl dieser Fehler. In Blau ist jeweils die Trajektorie dargestellt, zu der eine Route angefragt wurde, die in Rot dargestellt ist. Die Routen weichen deutlich von der Straße ab. In Anbetracht unseres Ziels, fehlende Kanten zu finden, sind dies false positives. Dennoch sind dies, wenn die Routingdaten zur Navigation eingesetzt werden sollen, Fehler in den Daten.
Eine Auswahl der tatsächlichen Fehler, die unser Algorithmus gefunden hat, findet sich in Abbildung 4.8. Zu den Straßen wurden zur Visualisierung Routen angefragt. In allen Fällen werden Start- und Zielpunkt auf Nachbarstraßen gelegt und es wird dazwischen eine Route berechnet.
Wir erhalten 107 Cluster, an denen verbotene Manöver vorliegen könnten. Eine Auswahl der gefundenen Fehler ist in Abbildung 4.9 zu sehen. In den Abbildungen 4.9(a) und 4.9(b) schlägt der Routing-Service vor, durchgezogene Linien zu überfahren. In Abbildung 4.9(c) berechnet der Service eine Route, die vorschlägt links abzubiegen, wo dies verboten ist, und in Abbildung 4.9(d) schlägt der Service vor, auf der B19 zu wenden, anstatt die Abfahrt zu benutzen. Allerdings weisen nicht alle gefundenen Cluster tatsächlich auf Fehler hin. Die in Abbildung 4.10 gezeigten Trajektorien und Routen sind alle zulässige Fahrten, die nur in der Stadt kurz voneinander abweichen.
Abweichungen der Fahrtdauer von der vorhergesagten Dauer visualisieren wir mit einer Heatmap. Abbildung 4.11 zeigt einen Ausschnitt der Münchener Innenstadt mit einer entsprechenden Heatmap. Für fast alle größeren Straßen findet unser Algorithmus positive Abweichungen, also Fahrten, die länger dauern als vom Routingorakel vorhergesagt. Dies deckt sich mit unseren Erwartungen, da es sich beim Here-Graphen um eine sogenannte free-flow-Karte handelt, also kein Verkehr berücksichtigt wird und alle Kantengewichte von der erlaubten Geschwindigkeit ausgehen. In der Realität werden Fahrzeuge jedoch durch hohes Verkehrsaufkommen und Ampeln ausgebremst. Die gefundenen negativen Abweichungen entstehen vermutlich dadurch, dass viele Fahrzeuge vor einer grünen Ampel über die erlaubte Geschwindigkeit beschleunigen, um diese noch rechtzeitig zu erreichen. Sichtbar ist außerdem, dass auf vielen kleinen Straßen negative Abweichungen auftreten.

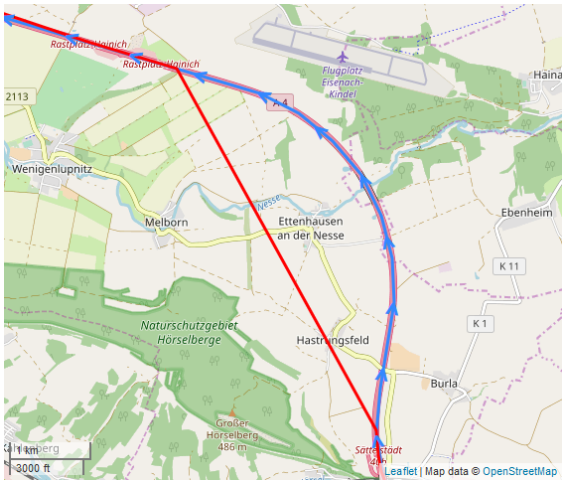


(a) Lineare Skalierung der Eingabegröße



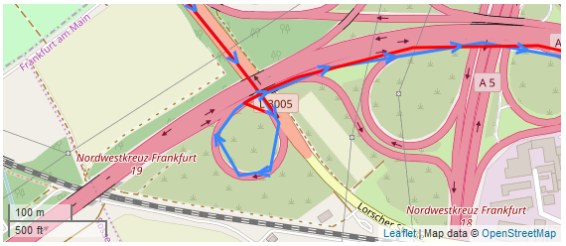
(b) Logarithmische Skalierung der Eingabegröße

Abbildung 4.6: Laufzeit von Algorithmus 3.1 in Abhängigkeit von der Anzahl der Punkte des Trajektoriezugs multipliziert mit der Anzahl der Punkte des Routenzugs. Zusätzlich ist eine lineare Regression eingezeichnet.



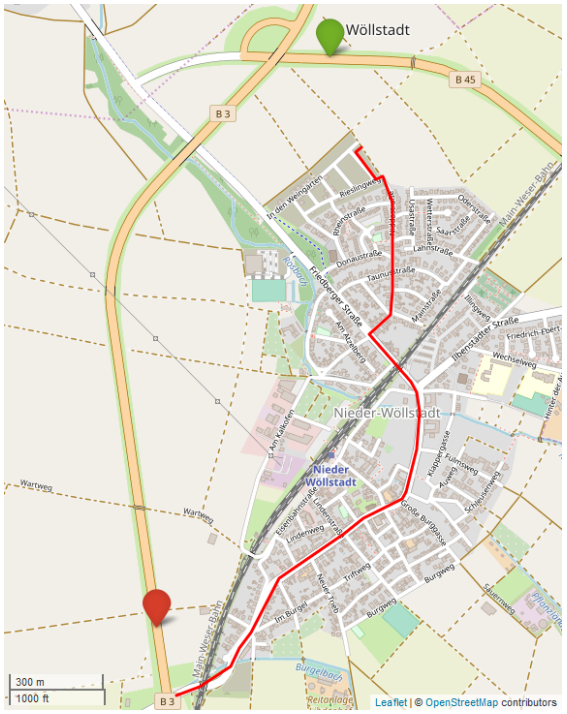
(a) Die A4 in der Nähe von Eisenach

(b) Die A5 in der Nähe von Wellington (England)

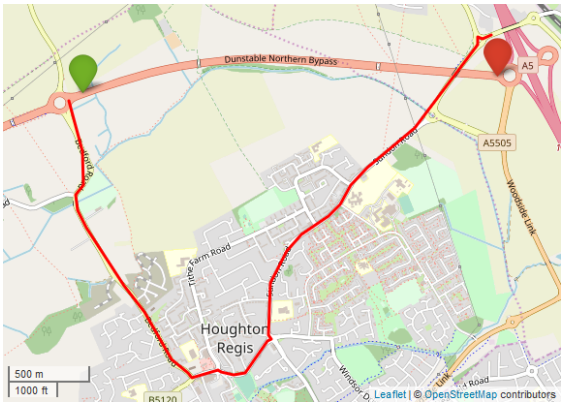


(c) Nordwestkreuz Frankfurt

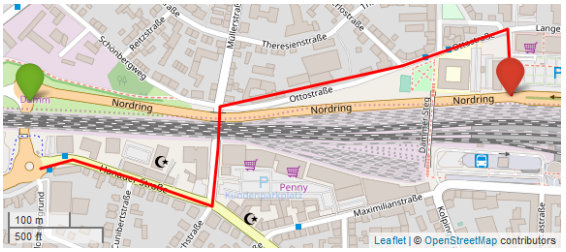
Abbildung 4.7: Trajektorien (blau), zu denen Routen (rot) mit dem Here-Graphen angefragt wurden. Die Routen weichen von der Straße ab.



(a) Die Umgehungsstraße um Wöllstadt bei Frankfurt

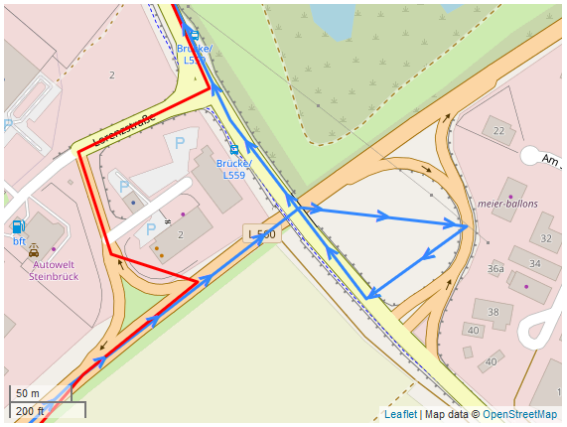


(b) Die A5 in der Nähe von Luton bei London

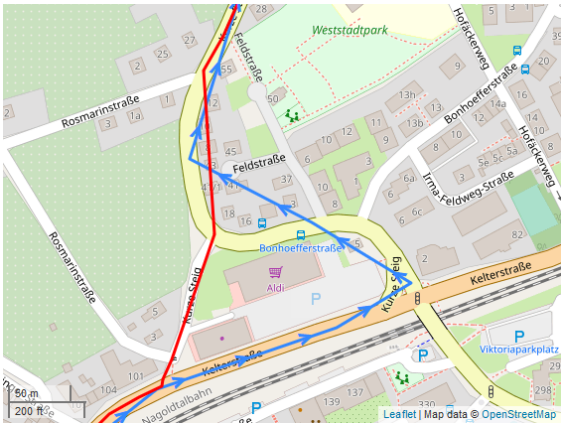


(c) Der Nordring in Aschaffenburg

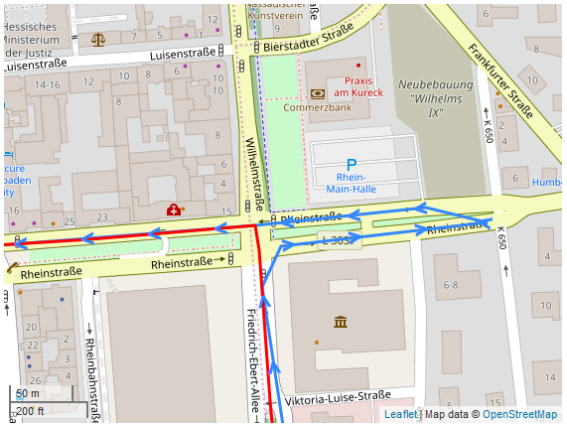
Abbildung 4.8: Beispiele von fehlenden Straßen im Here-Graphen, die unser Algorithmus gefunden hat.



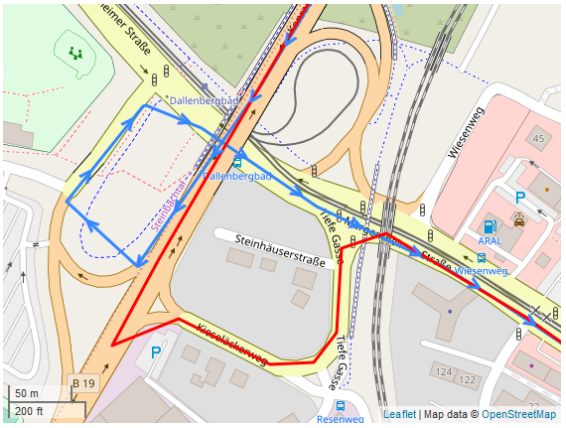
(a) Überfahren einer durchgezogenen Linie bei Blankenloch



(b) Überfahren einer durchgezogenen Linie in Pforzheim

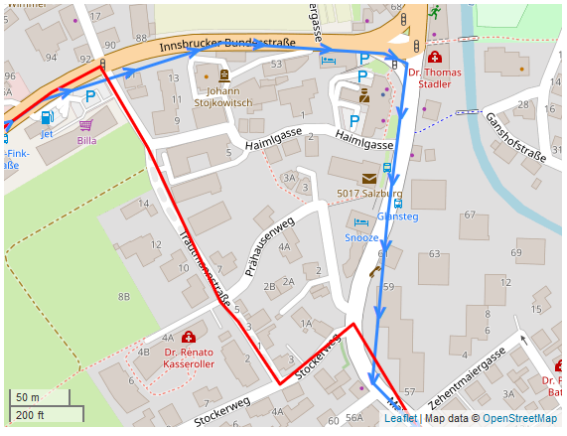


(c) Verbotenes Linksabbiegen in Wiesbaden

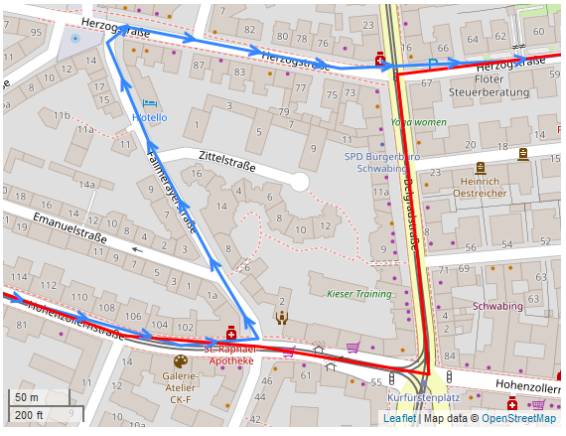


(d) Wendemanöver auf der B19 in Würzburg

Abbildung 4.9: Trajektorien (blau), zu denen Routen (rot) mit dem Here-Graphen angefragt wurden. Die Routen enthalten verbotene Manöver.



(a) Verschiedene zulässige Wege in Salzburg



(b) Verschiedene zulässige Wege in München

Abbildung 4.10: Trajektorien (blau), zu denen Routen (rot) mit dem Here-Graphen angefragt wurden. Unser Algorithmus erkennt fälschlicherweise verbotene Manöver.

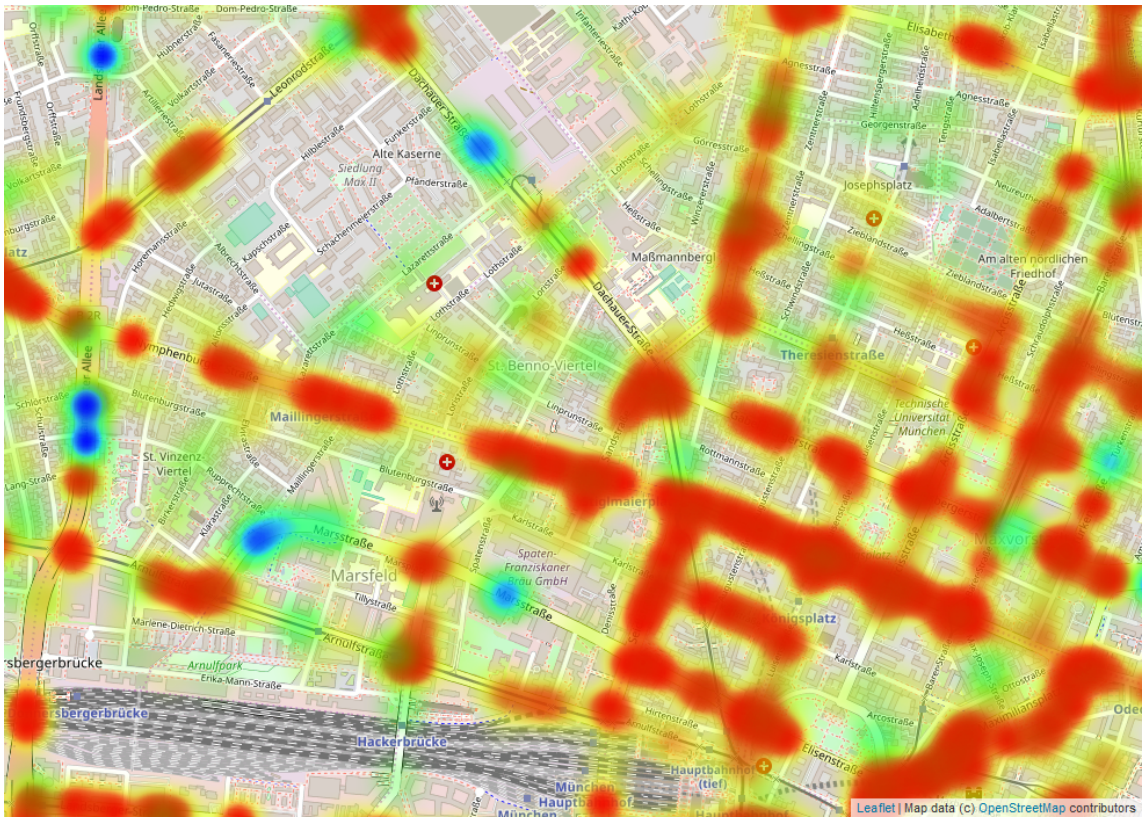


Abbildung 4.11: Karte der Münchener Innenstadt mit einer Heatmap der gefundenen Zeitabweichungen für den Here-Graphen. In Gelb bis Rot sind positive und in Grün bis Blau negative Abweichungen dargestellt. Je intensiver die Farben sind, desto mehr Abweichungen wurden zusammengefasst.

Die Markierungen dort sind kleiner, da weniger Fahrzeuge diese Straßen benutzen. Allerdings gehen wir auch hier davon aus, dass viele Fahrzeuge dort zu schnell fahren und deshalb negative Abweichungen auftreten. Auf der Stadtautobahn scheinen sich positive und negative Abweichungen etwa auszugleichen.
Zuletzt analysieren wir die vom Orakel mit dem Here-Graphen als Karte vorhergesagte Fahrtdauer für die Trajektorien. Abbildung 4.12 zeigt die vorhergesagte Zeit in Abhängigkeit zur Fahrtzeit der Trajektorie, zu der die Route angefragt wurde. In Blau wird zusätzlich die berechnete Regressionsgerade dargestellt. Vergleicht man diese mit der orangen Ideallinie, für die die Routenzeit der Trajektoriezeit entspricht, so stellt man leicht fest, dass die Routenzeit im Schnitt zu klein ist, die Fahrten also meistens länger dauern als vorhergesagt. Dies deckt sich mit unserer Analyse zu den Zeitabweichungen.
Zusätzlich haben wir die Trajektorien nach Fahrtdauern in die Kategorien weniger als 15 Minuten, 15 bis 60 Minuten und über 60 Minuten eingeteilt. In Abbildung 4.13 haben wir den symmetrischen Fehler in den Kategorien mit Box-Plots visualisiert. In allen Kategorien gibt es sehr hohe positive symmetrische Fehler. Dies lässt sich dadurch erklären, dass z.B. durch Staus die Trajektoriezeit sehr hoch werden kann, während die Routenzeit von optimalen Verkehrsbedingungen ausgeht. Sehr große negative symmetrische Fehler gibt es hingegen nur für Trajektorien bis 15 Minuten. Ein großer negativer symmetrischer Fehler kann nur auftreten, wenn ein Auto sehr viel schneller fährt, als auf einer Straße erlaubt ist, oder die erlaubte Geschwindigkeit auf einer Straße höher ist als im Graphen hinterlegt. Beide Ursachen sind nur auf kurzen Teilstrecken wahrscheinlich. Entsprechend kann ein großer negativer Fehler bei einer langen Fahrt ausgeglichen werden. Hinzu kommt, dass, wie in Abbildung 4.11 sichtbar, auf kleinen Straßen häufig negative Abweichungen auftreten.

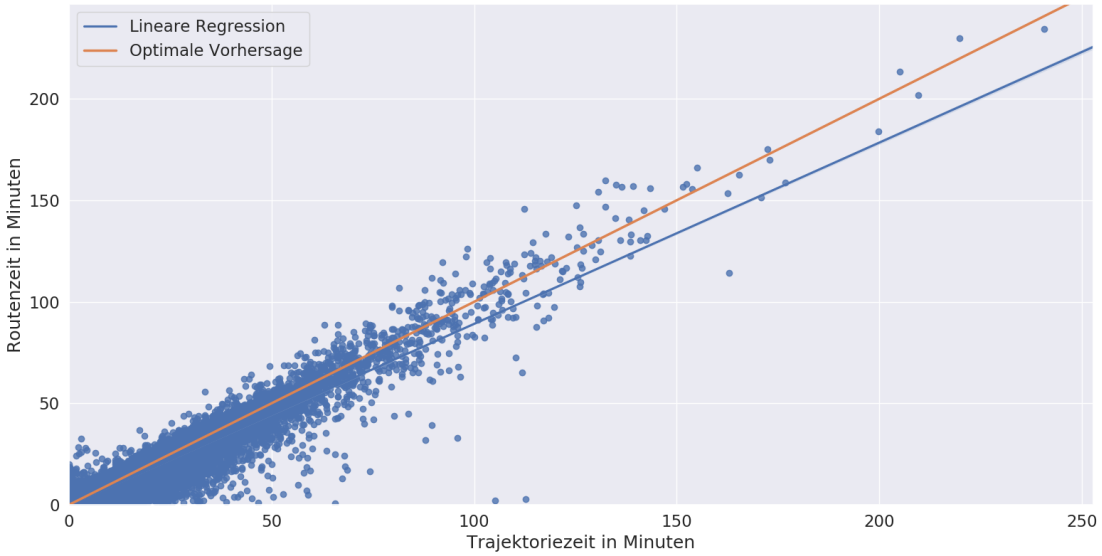


Abbildung 4.12: Vom Here-Graphen vorhergesagte Zeiten in Abhängigkeit zur tatsächlich benötigten Fahrtdauer. In Blau ist die lineare Regression eingezeichnet. Die orange Linie stellt den Optimalfall dar, dass genau die Trajektoriezeit vorhergesagt wird.

Der Anteil der Fahrt, der auf kleinen Straßen zugebracht wird, ist für kurze Fahrten höher als für lange Fahrten.
Mit zunehmender Fahrtdauer nähern sich sowohl der Median als auch die Whisker an einen Fehler von null an. Wir erklären dies damit, dass zum einen bei längeren Fahrten kleine Abweichungen ausgeglichen werden können, und zum anderen damit, dass auf Autobahnen und Landstraßen häufiger die Höchst- bzw. Richtgeschwindigkeit erreicht wird als bei Stadtfahrten. Trotzdem ist der Median auch für Fahrten über 60 Minuten noch positiv, die vorhergesagten Zeiten sind also auch hier meistens zu kurz.

4.3.2 Dimacs-Graph

Als zweiten Service nutzen wir unseren Routingalgorithmus mit dem Dimacs-Graphen. Unser Algorithmus findet 3400 Cluster mit möglichen fehlenden Kanten. Diese hohe Zahl ergibt sich im Wesentlichen dadurch, dass der Graph keine Modellierung des Straßenverlaufs enthält. Abbildung 4.14 zeigt beispielhaft zwei Fehler.
Allerdings finden wir auch im Dimacs-Graphen fehlende Kanten. Abbildung 4.15 zeigt dies beispielhaft. Die Auffahrt Karlsruhe-Nord der A5 wurde erst im Jahr 2007 eröffnet und ist deshalb nicht im Graphen enthalten [Kar07]. Die Umgehungsstraße um Münchingen ist nicht im Graphen enthalten, stattdessen werden Feldwege rund um den Ort genutzt.
Wir erhalten 61 Cluster mit möglichen verbotenen Manövern. Abbildung 4.16 zeigt zwei davon beispielhaft. Abbildung 4.16(a) enthält ein verbotenes Linksabbiegen vor der Rosensteinbrücke in Stuttgart, Abbildung 4.16(b) zeigt ein Wendemanöver auf dem Föhringer Ring in München.
Die von unserem Algorithmus gefundenen Zeitabweichungen analysieren wir, wie schon für den Here-Graphen, beispielhaft an einem Ausschnitt der Münchener Innenstadt. Der Dimacs-Graph arbeitet mit einer free-flow-Metrik, daher finden sich auf den meisten größeren Straßen positive Abweichungen, dargestellt in Rot. Besonders interessant ist die deutliche Abweichung auf der Stadtautobahn am linken Rand des Ausschnitts. Hier scheint das Kantengewicht deutlich zu klein gewählt zu sein oder auf der Stadtautobahn stauen sich die Fahrzeuge sehr häufig. Die wenigen intensiven negativen Abweichungen entstehen vermutlich dadurch, dass viele Fahrzeuge bei einer freien Straße über die erlaubte

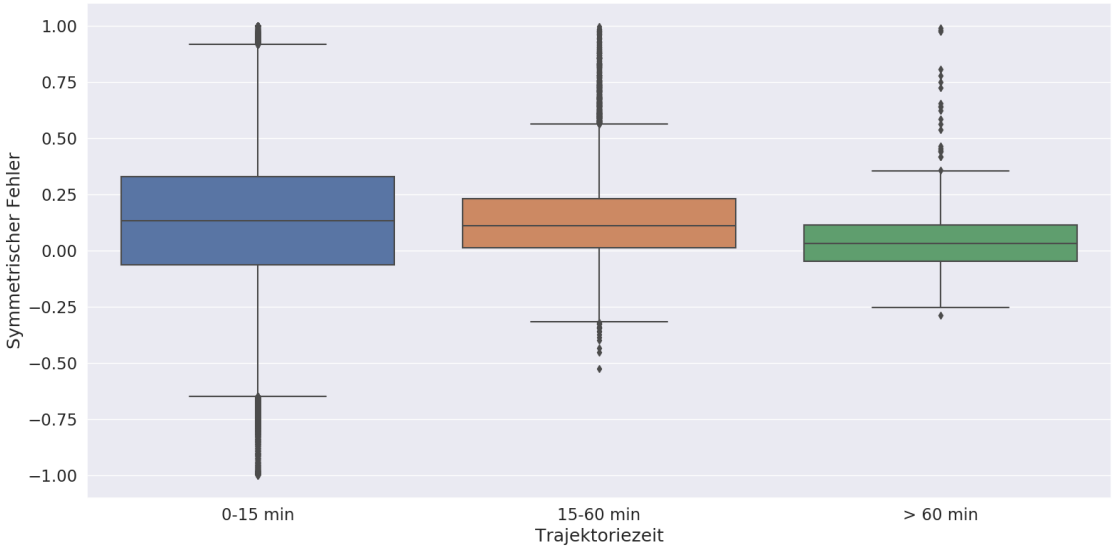
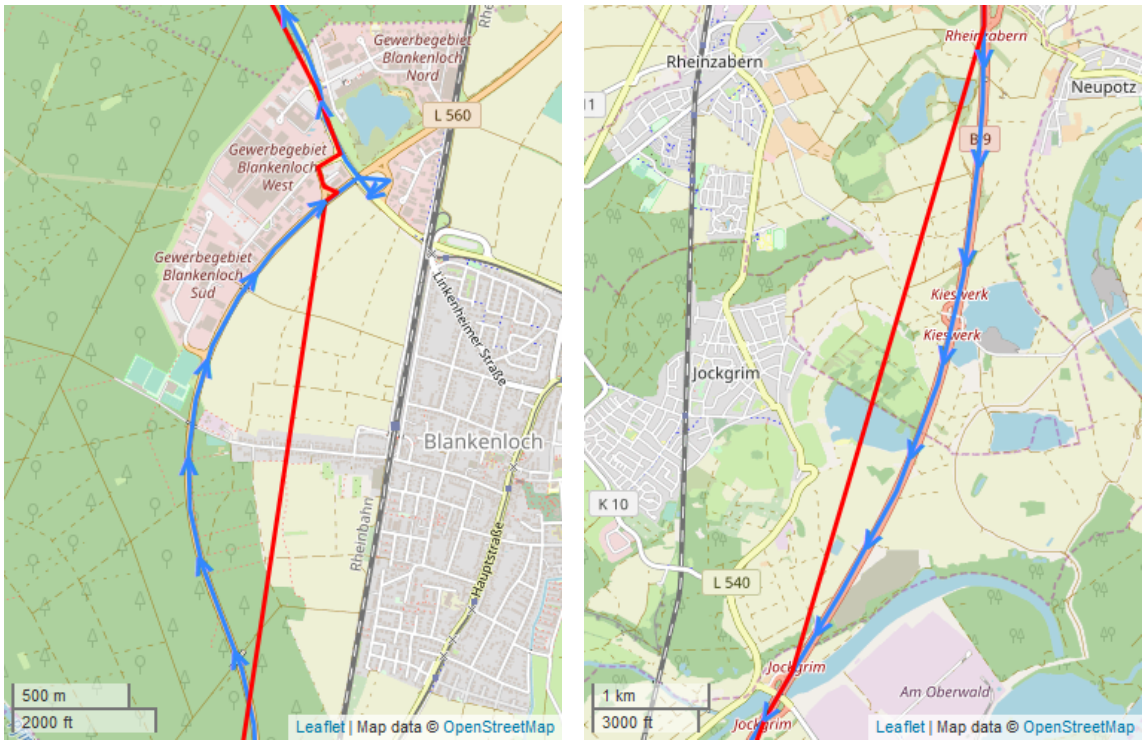


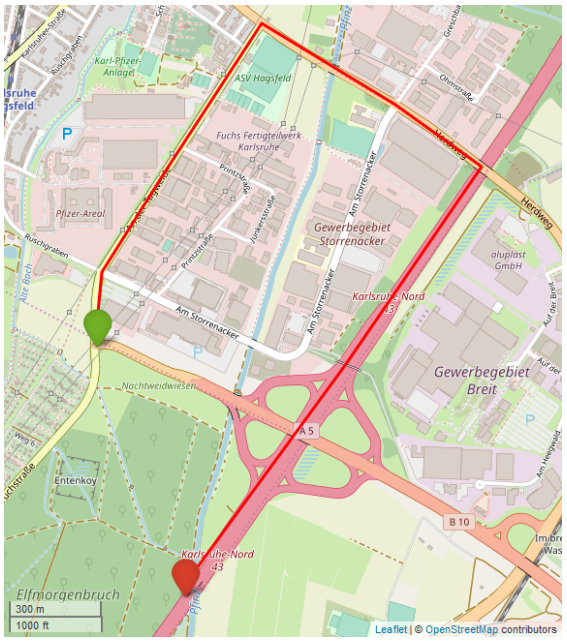
Abbildung 4.13: Box-Plots der symmetrischen Fehler für das Routingorakel mit dem Here-Graphen. Die Plots sind eingeteilt nach der Fahrtdauer der Trajektorien. Die Länge der Whisker ist der 1,5-fache Interquartilsabstand.



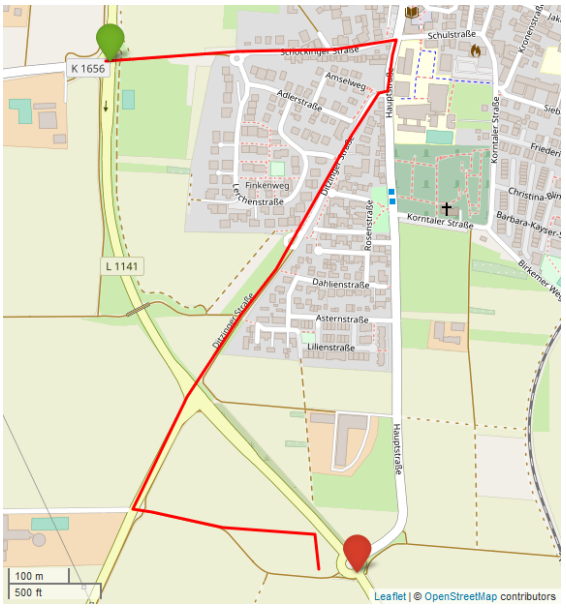
(a) Die L560 bei Blankenloch

(b) Die B9 bei Wörth

Abbildung 4.14: Trajektorien (blau), zu denen Routen (rot) mit dem Dimacs-Graphen angefragt wurden. Die Routen weichen von der Straße ab.

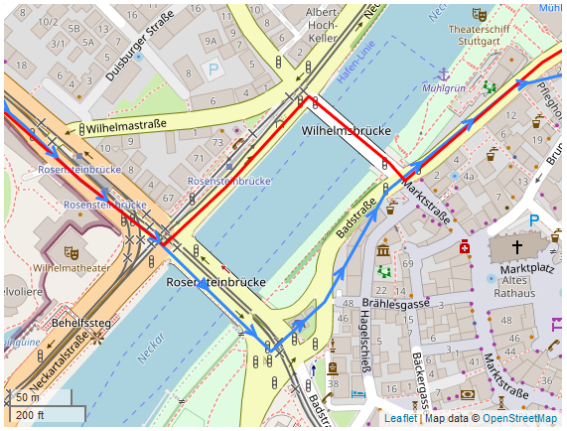


(a) Die Auffahrt 43 Karlsruhe-Nord der A5

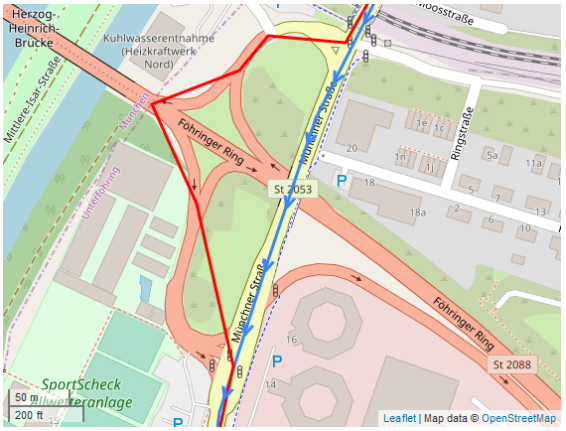


(b) Die Umgehungsstraße L1141 um Münchingen bei Stuttgart

Abbildung 4.15: Beispiel fehlender Straßen im Dimacs-Graphen, die unser Algorithmus gefunden hat.



(a) Die Rosensteinbrücke in Stuttgart



(b) Der Föhringer Ring in München

Abbildung 4.16: Trajektorien (blau), zu denen Routen (rot) mit dem Here-Graphen angefragt wurden. Die Routen enthalten verbotene Manöver.

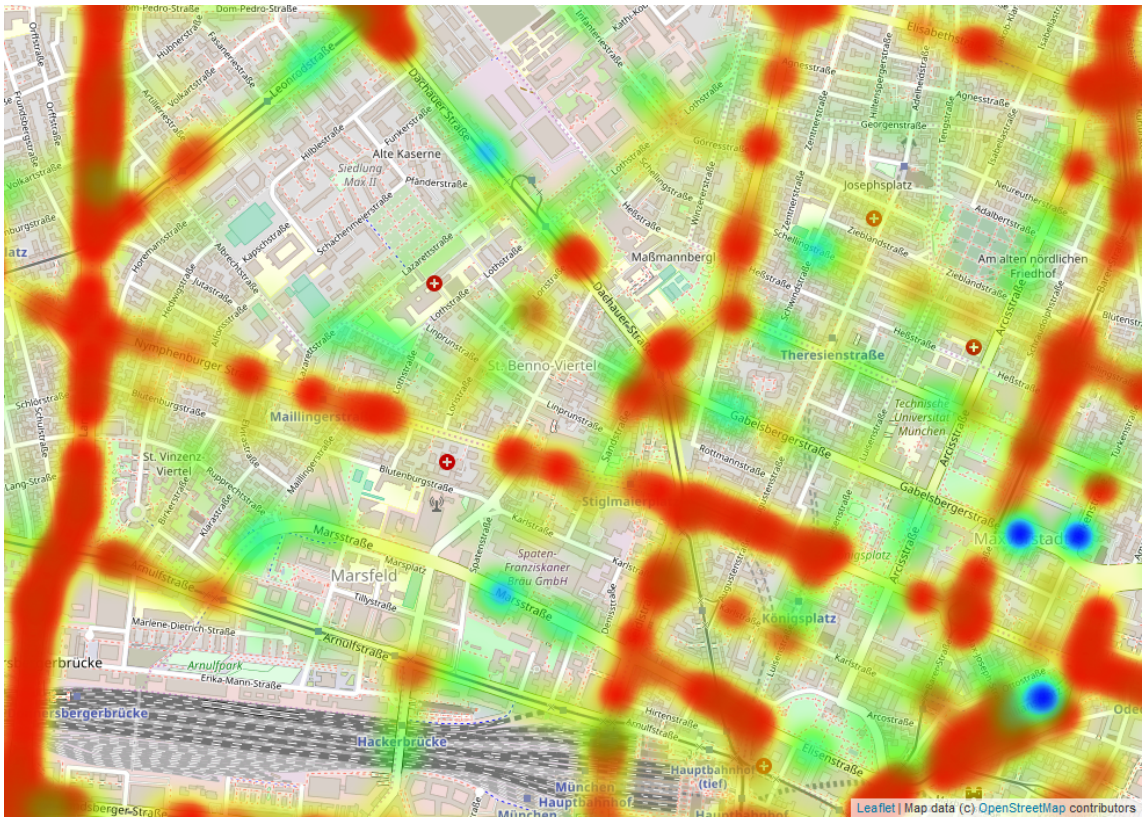


Abbildung 4.17: Karte der Münchener Innenstadt mit einer Heatmap der gefundenen Zeitabweichungen für den Dimacs-Graphen. In Gelb bis Rot sind positive und in Grün bis Blau negative Abweichungen dargestellt. Je intensiver die Farben sind, desto mehr Abweichungen wurden zusammengefasst.

Geschwindigkeitsbegrenzung beschleunigen. Die meisten dargestellten Abweichungen sind lediglich grün, es wurden also nur wenige Punkte in der Heatmap zusammengefasst. Daher ist es unklar, ob die Kantengewichte hier zu groß gewählt sind oder viele Fahrzeuge die Geschwindigkeitsbegrenzung überschreiten.
Zuletzt analysieren wir auch für den Dimacs-Graphen die von unserem Routingorakel vorhergesagten Fahrtdauern. Abbildung 4.18 zeigt die vorhergesagten Zeiten in Abhängigkeit von der für die Fahrt tatsächlich benötigten Zeit. In Blau ist zusätzlich die lineare Regression eingezeichnet. Es ist gut sichtbar, dass diese deutlich unter dem in Orange eingezeichneten Optimalfall liegt. Im Schnitt sind die Vorhersagen also, wie für eine free-flow Karte zu erwarten, zu kurz.
Für Abbildung 4.19 haben wir die Trajektorien nach der tatsächlich benötigten Fahrtdauer in die Kategorien bis 15 Minuten, 15 bis 60 Minuten und über 60 Minuten eingeteilt. Die Abbildung zeigt die Boxplots der berechneten symmetrischen Fehler zu den vorhergesagten Fahrtdauern. Gut sichtbar ist, dass sich die Fehler für die Kategorie bis 15 Minuten zwar gleichmäßig um den Optimalfall 0 verteilen, dabei aber betraglich sehr groß werden. Für längere Fahrten sehen wir die erwartete Verschiebung zu positiven Fehlern. Dies liegt erneut daran, dass es sich um eine free-flow Karte handelt. Um so verwunderlicher sind die vielen negativen Abweichungen für kurze Fahrten. Wir schließen daraus, dass die Kantengewichte für Stadtstraßen häufig zu groß gewählt sind. Dies deckt sich mit der Heatmap in Abbildung 4.17.

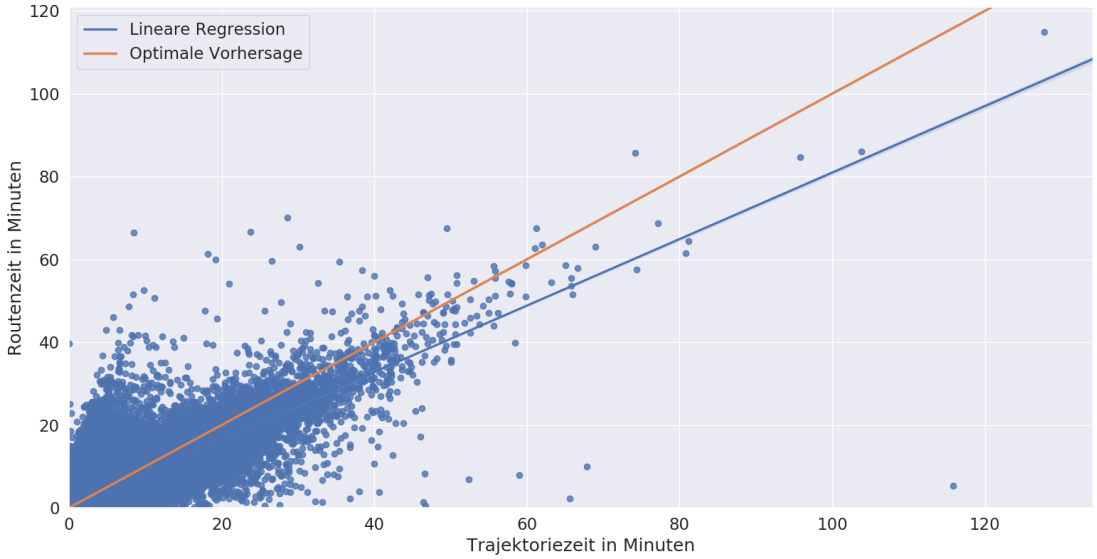


Abbildung 4.18: Vom Dimacs-Graphen vorhergesagte Zeiten in Abhängigkeit zur tatsächlich benötigten Fahrtdauer. In Blau ist die lineare Regression eingezeichnet. Die orange Linie stellt den Optimalfall dar, dass genau die Trajektoriezeit vorhergesagt wird.

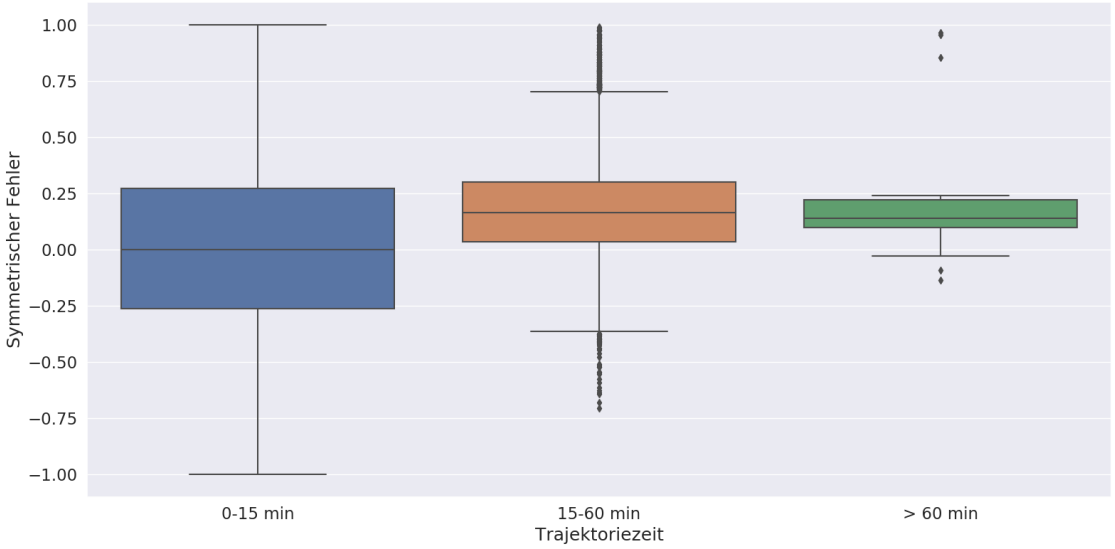


Abbildung 4.19: Box-Plots der symmetrischen Fehler für das Routingorakel mit dem Dimacs-Graphen. Die Plots sind eingeteilt nach der Fahrtdauer der Trajektorien. Die Länge der Whisker ist der 1,5-fache Interquartilsabstand.

Tabelle 4.3: Übersicht der benötigten Anfragen und gefundener Fehler für verschiedene Karten

	Here-Graph	Dimacs-Graph
Durchschnittliche Anfragen	2,04	4,18
Durchschnittlicher absoluter symmetrischer Fehler	0,26	0,31
Nachbildung möglich	212708	161674
Nachbildung nicht möglich	53624	104658
Mögliche fehlende Kanten	143	3400
Mögliche verbotene Manöver	107	61
Positive Zeitabweichungen	12887	7028
Negative Zeitabweichungen	2260	807

4.3.3 Vergleich der Routing-Services

Bisher haben wir alle Routing-Services für sich betrachtet. Unsere Ziel war es aber auch, einen Qualitätsvergleich zu ermöglichen. Dazu zeigt Tabelle 4.3 einige Maßzahlen. Für die ersten beiden Werte sind jeweils kleinere Zahlen besser. Die durchschnittlichen Anfragen entsprechen der in Abschnitt 3.1.1 beschriebenen Komprimierung. Für Trajektorien, die nicht nachgebildet werden können, verwenden wir die Zahl der Anfragen, die benötigt wird, um alle Stellen zu finden, an denen eine Nachbildung nicht möglich ist. Der Here-Graph schneidet hier deutlich besser ab als der Dimacs-Graph.
Als Nächstes betrachten wir den durchschnittlichen symmetrischen Fehler. Damit sich positive und negative Fehler nicht ausgleichen können, verwenden wir nur den Betrag des Fehlers. Auch hier schneidet der Here-Graph etwas besser ab, hat aber immer noch eine starke durchschnittliche Abweichung.
Die Anzahl der Trajektorien, die nachgebildet werden können, ist eine offensichtliche Maßzahl. Der Dimacs-Graph schneidet hier hauptsächlich durch die fehlende Modellierung des Straßenverlaufs deutlich schlechter ab.
Die weiteren Zahlen sind lediglich der Vollständigkeit halber aufgeführt und sind für einen direkten Vergleich nicht geeignet. So findet unser Algorithmus für den Dimacs-Graphen zwar wesentlich weniger mögliche verbotene Manöver, allerdings liegt dies unter anderem auch an der fehlenden Modellierung der Straßen. In Abbildung 4.14(a) schlägt die in Rot dargestellte Route vor, eine durchgezogene Linie zu überfahren. Für den Here-Graphen findet unser Algorithmus genau dieses Manöver, dargestellt in Abbildung 4.9(a). Durch die fehlende Modellierung der Straße vorher kann unser Algorithmus den Fehler für den Dimacs-Graphen nicht finden.
Die Zahl der gefundenen Zeitabweichungen lässt sich ebenfalls schwer vergleichen, da wir Zeitabweichungen nur für gelungene Nachbildungen betrachten. Daher folgt die höhere Zahl der Zeitabweichungen für den Here-Graphen unter anderem aus der höheren Zahl der Nachbildungen.
Eine Möglichkeit, diese Zahlen für einen Vergleich zu nutzen, wäre, sie in Relation zur insgesamt, über alle Trajektorien, nachgebildeten Strecke zu setzen.

5. Zusammenfassung

Das Ziel dieser Arbeit war es, einen Algorithmus zu entwickeln, der mit Hilfe von Fahrzeugtrajektorien Routingdaten analysiert. Dazu wurden drei Klassen an Fehlern bestimmt, die der Algorithmus finden soll. Diese sind fehlende Kanten, falsche Kanten und Fehler in der Metrik.
Unser Algorithmus wurde experimentell evaluiert. Dazu wurden in einen Routinggraphen Fehler eingebaut, die der Algorithmus finden sollte. Unser Algorithmus kann nur Fehler finden, an denen ausreichend Fahrten entlangführen. Wir hatten nur eine vergleichsweise kleine Menge an Fahrtdaten zur Verfügung. Diese hat die beste Abdeckung auf Autobahnen. Daher haben wir lediglich Autobahnkanten manipuliert. Dabei konnte unser Algorithmus einen Großteil der gelöschten Kanten als fehlend identifizieren. Die Identifikation von Kanten mit manipuliertem Gewicht hat sich als wesentlich schwieriger erwiesen. Wir sind dabei zu dem Schluss gekommen, dass unser Ansatz, lediglich auf den Ausgaben eines Routing-Services und ohne Wissen der Routingdaten zu arbeiten, Probleme hat, starke Abweichungen des Kantengewichts von fehlenden Straßen zu unterscheiden.
Zusätzlich wurde der Algorithmus auf zwei unveränderten Routinggraphen eingesetzt, um Fehler zu finden. Dabei hat sich herausgestellt, dass unser Algorithmus vermeintlich fehlende Kanten an Stellen identifiziert, an denen der Routing-Service keine Modellierung des Straßenverlaufs zurückliefert. Da dies dem Fall entspricht, dass an diesem Ort in den Routingdaten keine Straße verläuft, sondern daneben, sehen wir keine Möglichkeit diese Fehler zu unterscheiden.
Weiter hat unser Algorithmus auf beiden Karten verbotene Manöver, wie das Überfahren von durchgezogenen Linien, gefunden. Allerdings kommt es hier auch zu false positives. Ein besserer Clustering-Algorithmus wäre möglicherweise in der Lage, das Ergebnis zu verbessern. Vorstellbar wäre ein Ansatz, der die Mindestzahl der Trajektorien, die in einem Cluster enthalten sein müssen, abhängig macht von der Anzahl der Trajektorien, die an einer Stelle gefahren sind.
Für die Evaluation standen nur Routingdaten mit free-flow Metriken zur Verfügung. Eine Analyse der von unserem Algorithmus gefundenen möglichen Fehler in der Metrik hat sich daher als problematisch herausgestellt. Abweichungen, in denen die tatsächlichen Fahrtzeiten länger sind als die vorhergesagten, lassen sich meistens auf ein erhöhtes Verkehrsaufkommen zurückführen. Spannend wäre daher die Evaluation eines Routing-Service mit historischen Daten zum Verkehrsaufkommen und daran angepassten Fahrtzeiten. Zusätzlich ist es für Abweichungen, in denen die tatsächlichen Fahrtzeiten kürzer sind als die vorhergesagten, schwer zu unterscheiden, ob das Kantengewicht tatsächlich falsch ist oder das Fahrzeug lediglich die Geschwindigkeitsbegrenzung überschritten hat. Der oben ange-

sprochene bessere Clustering-Algorithmus könnte vermutlich auch hier zu Verbesserungen führen.
Zuletzt wurden einige Maßzahlen zum Vergleich von Routing-Services vorgestellt. Interessant wäre es als Nächstes, diese für etablierte Routing-Services zu berechnen und einen Vergleich anzustellen.

Literaturverzeichnis

[9th] *9th DIMACS Implementation Challenge: Shortest Paths*. `http://users.diag.uniroma1.it/challenge9/download.shtml`, besucht: 2019-03-22.

[AERW03] Helmut Alt, Alon Efrat, Günter Rote und Carola Wenk: *Matching planar maps*. Journal of Algorithms, 49(2):262–283, November 2003, ISSN 01966774.

[AG95] Helmut Alt und Michael Godau: *COMPUTING THE FRÉCHET DISTANCE BETWEEN TWO POLYGONAL CURVES*. International Journal of Computational Geometry & Applications, 05(01n02):75–91, März 1995, ISSN 0218-1959, 1793-6357.

[AKW04] Helmut Alt, Christian Knauer und Carola Wenk: *Comparison of Distance Measures for Planar Curves*. Algorithmica, 38(1):45–58, Januar 2004, ISSN 0178-4617, 1432-0541.

[AW12] Mahmuda Ahmed und Carola Wenk: *Constructing Street Networks from GPS Trajectories*. In: Leah Epstein und Paolo Ferragina (Herausgeber): *Algorithms – ESA 2012*, Lecture Notes in Computer Science, Seiten 60–71. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-33090-2.

[BBW09] Kevin Buchin, Maike Buchin und Yusu Wang: *Exact Algorithms for Partial Curve Matching via the Fréchet Distance*. In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, Seiten 645–654. Society for Industrial and Applied Mathematics, Januar 2009, ISBN 978-0-89871-680-1 978-1-61197-306-8.

[BPSW] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas und Carola Wenk: *On Map-Matching Vehicle Tracking Data*. Seite 12.

[DHP13] A. Driemel und S. Har-Peled: *Jaywalking Your Dog: Computing the Fréchet Distance with Shortcuts*. SIAM J. Comput., 42(5):1830–1866, Januar 2013, ISSN 0097-5397.

[EKSX96] Martin Ester, Hans Peter Kriegel, Jörg Sander und Xiaowei Xu: *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. Seite 6, Januar 1996.

[Fré06] M. Maurice Fréchet: *Sur quelques points du calcul fonctionnel*. Rend. Circ. Matem. Palermo, 22(1):1–72, Dezember 1906, ISSN 0009-725X.

[Kar07] *Karlsruhes neuer Autobahnanschluss | ka-news*, März 2007. `https://www.ka-news.de/region/karlsruhe/Durchbruch-im-Norden;art6066,56910?show=phf200732-85K`, besucht: 2019-03-15.

[LZZ+09] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang und Yan Huang: *Map-matching for low-sampling-rate GPS trajectories*. In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic*

Information Systems - GIS '09, Seite 352, Seattle, Washington, 2009. ACM Press, ISBN 978-1-60558-649-6.

[MSSZZ14] Anil Maheshwari, Jörg Rüdiger Sack, Kaveh Shahbaz und Hamid Zarrabi-Zadeh: *Improved Algorithms for Partial Curve Matching.* Algorithmica, 69(3):641–657, Juli 2014, ISSN 0178-4617, 1432-0541.

[NK09] Paul Newson und John Krumm: *Hidden Markov map matching through noise and sparseness.* In: *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '09*, Seite 336, Seattle, Washington, 2009. ACM Press, ISBN 978-1-60558-649-6.

[PZWS13] Bei Pan, Yu Zheng, David Wilkie und Cyrus Shahabi: *Crowd sensing of traffic anomalies based on human mobility and social media.* In: *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems - SIGSPATIAL'13*, Seiten 344–353, Orlando, Florida, 2013. ACM Press, ISBN 978-1-4503-2521-9.

[Ver] *Verkehrs- und Unfalldaten - Kurzzusammenstellung der Entwicklung in Deutschland.* `https://www.bast.de/DE/Publikationen/Medien/VU-Daten/VU-Daten.pdf?__blob=publicationFile&v=3`, besucht: 2019-02-20.

[WZX14] Yilun Wang, Yu Zheng und Yexiang Xue: *Travel time estimation of a path using sparse trajectories.* In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, Seiten 25–34, New York, New York, USA, 2014. ACM Press, ISBN 978-1-4503-2956-9.

[YZXS11] Jing Yuan, Yu Zheng, Xing Xie und Guangzhong Sun: *Driving with knowledge from the physical world.* In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '11*, Seite 316, San Diego, California, USA, 2011. ACM Press, ISBN 978-1-4503-0813-7.

[Zhe15] Yu Zheng: *Trajectory Data Mining: An Overview.* ACM Transactions on Intelligent Systems and Technology, 6(3):1–41, Mai 2015, ISSN 21576904.

[ZZY+14] K. Zheng, Y. Zheng, N. J. Yuan, S. Shang und X. Zhou: *Online Discovery of Gathering Patterns over Trajectories.* IEEE Transactions on Knowledge and Data Engineering, 26(8):1974–1988, August 2014, ISSN 1041-4347.