Dijkstra-basiertes Mapmatching

Bachelorarbeit
von

Peter Maucher

An der Fakultät für Informatik
Institut für Theoretische Informatik

Erstgutachter:	Prof. Dr. Dorothea Wagner
Zweitgutachter:	Prof. Dr. Peter Sanders
Betreuende Mitarbeiter:	Tim Zeitz
	Tobias Zündorf

Bearbeitungszeit: 1. Juli 2018 – 31. Oktober 2018

Selbstständigkeitserklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, 31. Oktober 2018

Abstract

In this thesis we present presents a new approach on the map-matching problem based on Dijkstra's algorithm. Given a sequence of measured positions of an object traversing a road network and a graph representing that network the objective is to find the most likely path of said object. This problem is called the map-matching problem. Dijkstra's Algorithm solves the shortest-path problem efficiently, so the idea is to build a shortest-path problem instance where the result matches the result of the map-matching problem. Therefore, the edge-weights in the graph are modified with respect to the measured positions. Those modifications are based on line integrals over the edge in the graph. We show that our algorithm is efficient by comparing it to an unmodified version of Dijkstra's Algorithm. It has high-quality results on real world data as shown by comparing to a standard map-matching algorithm called ST-Matching.

Zusammenfassung

In dieser Arbeit beschreiben wir einen neuen Ansatz zur Lösung des Mapmatching-Problems basierend auf Dijkstras Algorithmus. Zu einem sich in einem Straßennetzwerk bewegenden Objekt werden Positionsmessdaten aufgezeichnet. Im Mapmatching-Problem wird zu diesen Messdaten ein möglichst plausibler Pfad in dem Graphen gesucht, der das Straßennetzwerk repräsentiert. Dijkstras Algorithmus löst das kürzeste-Wege-Problem effizient. Unser Ansatz erweitert diesen Algorithmus, indem die Kantengewichte abhängig von den gemessenen Daten verringert werden. Mit den geänderten Kantengewichten entsteht eine Lösung des Mapmatching-Problems. Zur Anpassung der Gewichte werden Wegintegrale über die Graphkante verwendet. Wir zeigen die Effizienz unseres Algorithmus auf echten Daten, indem wir ihn mit einem unmodifizierten Dijkstra vergleichen. Die Genauigkeit wird durch den Vergleich mit einem Standardalgorithmus für Mapmatching mit Namen ST-Matching gezeigt.

Inhaltsverzeichnis

1. Einleitung

Moderne Autos sind mit verschiedenen Sensoren ausgestattet, für Navigationssysteme auch mit Positionssensoren. Diese nutzen zum Beispiel das US-amerikanische Global Positioning System (GPS) oder das europäische Galileo-System. Die Ortsdaten, die diese Geräte liefern, rauschen teilweise um mehrere Meter. Verschiedene Dienstleister sammeln periodisch diese Positionsdaten und verknüpfen sie zu Messrouten. Die zeitliche Auflösung dieser Routen kann sehr dünn sein. In unseren Daten beträgt die Auflösung 15 s. Auf einer Autobahn bei 120 km/h entspricht eine zeitliche Auflösung von 15 s einer räumlichen Auflösung von 500 m. Die durchschnittliche Kantenlänge im OpenStreetMap-Straßengraphen für Europa beträgt im Vergleich dazu 20 m.

Für die Nutzung der Daten ist es oft nötig, die Messdaten auf den Straßengraphen zurückzurechnen und den Pfad im Graphen zu erhalten. Dies ist zum Beispiel interessant, um Staus zu erkennen, oder für die Verkehrsplanung, um die Nutzung einer Straße zu ermitteln. In beiden Fällen reicht es nicht, nur die räumliche Dichte von Punkten zu messen, denn es könnten auch Autos auf Parallelstraßen unterwegs oder am Straßenrand geparkt sein. Diese Frage nach der im Graph gewählten Route wird als Mapmatching-Problem bezeichnet. Diese Arbeit stellt einen Lösungsalgorithmus für dieses Problem vor. Dieser Ansatz basiert auf Wegintegralen über die Kanten, abhängig von den Messpunkten. Um die Messpunkte herum werden geometrische Formen gelegt, durch die ein Höhenverlauf einer Funktion gegeben ist. Es wird dann das Integral der Kante durch den Höhenverlauf gebildet. Dabei wird das ursprüngliche Gewicht der Kante in die Berechnung mit einbezogen, sodass die Geschwindigkeitsinformation erhalten bleibt. Durch diese zusätzliche Information ist es möglich, vermutlich plausiblere Fälle zu bevorzugen. Der Lösungsalgorithmus ist insbesondere dahingehend entwickelt, bei der Unterteilung von Kanten kein anderes Ergebnis zu liefern. Aus diesem Grund ist es möglich, den Graph in einem Vorberechnungsschritt räumlich zu unterteilen. Ein solcher Graph kann zum Beispiel in eine räumliche Datenstruktur wie einen Quadtree einfügt werden.

1.1. Related Work

Navigationssysteme benötigen Lokalisierungssysteme wie GPS, nach [DoD08] weist dieses in 95 % der Fälle einen räumlichen Fehler von unter 8 m auf.

Einen Überblick über Mapmatching-Verfahren zur Verwendung in Navigationssystemen gibt [QON07]. Dieses Paper fokussiert sich auch darauf, wie die aktuelle Position des Fahrzeuges korrekt auf die Straße abgebildet werden kann. Es werden Ansätze für Mapmatching

vorgestellt, die auf geometrischen, topologischen und probabilistischen Ansätzen beruhen. Für diese Arbeit sehr wichtig ist ein Paper, dass abhängig von den Verbindungen der Punkte Kanten auswählt [LZZ+09]. Der dort beschriebene Algorithmus stellt den Konkurrenten, gegen den der in dieser Arbeit beschriebene Algorithmus verglichen wird. Das Paper stellt einen Ansatz für sehr schlechte zeitliche Auflösung vor, der auf der Konstruktion von Kandidatenpfaden basiert. Dabei werden zunächst für jeden Messpunkt Knoten im Straßengraphen ausgewählt, die durch Abstand plausibel sind. Anhand dieser Knoten werden mögliche Pfade konstruiert, die diese Punkte enthalten. Diese Pfade werden dann abhängig von den räumlichen Korrelationen und den Kantengewichten gewichtet. Der Pfad mit dem besten Gewicht wird als Ergebnis gewählt.

In dieser Arbeit werden kürzeste Routen berechnet. Dazu nutzen wir den bekannten Algorithmus von Dijkstra [Dij59]. Dieser wird bereits in mehreren Ansätzen für Mapmatching-Algorithmen genutzt. Das Paper [SWP06] beschreibt einen Ansatz, in dem um die Messpunkte die Bereiche markiert werden, die innerhalb der Zeit überhaupt erreicht werden können. Dieser Bereich wird als Kandidatengraph gespeichert. Zur Konstruktion dieser Bereiche wird Dijkstras Algorithmus verwendet. Mithilfe eines auf Fréchet-Distanzen basierenden Algorithmus wird dann auf dem vorberechneten Graphen die wahrscheinlichste Route berechnet. Die Fréchet-Distanz [AG95] ist ein Maß für die Ähnlichkeit von Kurven. Das Paper [YW04] kommt dem Ansatz dieser Arbeit nahe, auch hier werden mithilfe von Wegintegralen die Kantengewichte verändert. Dazu werden in den gemessenen Daten die dem Ursprung und Ziel der Graphkante nächsten Punkte gesucht und die Zeitpunkte dazu notiert. Um das Gewicht der Kante zu berechnen, wird über den euklidischen Abstand des Polygonzugs der Messdaten zur Kante integriert. Danach wird durch die Zeitdauer geteilt, der für die Kante gebraucht wird, um damit orthogonale Kanten zu bestrafen.

1.2. Beitrag

In unserer Arbeit soll ein neuer Mapmatching-Algorithmus beschrieben und evaluiert werden. Dazu wird Dijkstras Algorithmus auf einer angepassten Instanz eines Straßengraphen genutzt. Diese Anpassung verringert das Gewicht bestehender Kanten abhängig von den umliegenden Messpunkten. Dazu wird, wie in [YW04], das Wegintegral über die betrachtete Kante gebildet. Es gibt allerdings im Detail Unterschiede. Diese Arbeit geht davon aus, dass der Zeitabstand der Messpunkte ähnlich ist, und nutzt die daraus resultierende räumliche Information, um die Zeit abzubilden. Der Ansatz in [YW04] geht von zeitlich sehr hoch aufgelösten Daten im Sekundenbereich aus und evaluiert daher gar nicht, wie gut der Algorithmus Lücken in den Daten überbrückt. Zusätzlich wird der Einfluss der Messpunkte nicht betrachtet, die außerhalb des Zeitintervalls der Kante liegen. Der in dieser Arbeit vorgestellte Algorithmus ist für zeitlich deutlich schlechter aufgelöste Daten ausgelegt. Obwohl unsere Daten eine zeitliche Auflösung von 15 Sekunden haben sollten, gibt es regelmäßige Unterbrechungen in den Daten. Der Algorithmus wird daher auch für zeitlich verrauschte Daten evaluiert. Zusätzlich definiert dieser Algorithmus einen Einflussbereich, für den die Punkte relevant sind. Außerhalb wird das Ursprungsgewicht der Kanten genutzt. Das Gewicht im Konkurrenzansatz ist unbeschränkt und ist außerdem nicht von dem Ursprungsgewicht der Kante abhängig. Damit gibt es dort keine Möglichkeit, durch bestehende Informationen plausiblere Pfade zu wählen oder Lücken in den Daten zu überbrücken.

1.3. Aufbau der Arbeit

Die Arbeit beschreibt zuerst in Kapitel 2 die Grundlagen der Arbeit. In Abschnitt 2.1 werden Definitionen eingeführt, die für die Arbeit benötigt werden, und das Problem

formalisiert. Im Kapitel 3 werden beide in dieser Arbeit evaluierten, mathematischen Ansätze beschrieben. Der programmatische Aufbau des Algorithmus wird in Kapitel 4 ausgeführt. Der erfolgreiche, zweite Ansatz wird in Kapitel 5 experimentell evaluiert. Den Abschluss der Arbeit bietet eine Zusammenfassung in Kapitel 6. Hier wird außerdem ein Ausblick gegeben, wie die Ergebnisse unserer Arbeit fortgeführt werden können. Im Anhang in werden mathematische Formeln für den zweiten Ansatz beschrieben.

2. Preliminaries

In diesem Kapitel werden zunächst für die Arbeit notwendige Definitionen eingeführt, insbesondere Koordinaten und Graphen. Basierend darauf wird dann das Problem beschrieben und daraus der grundlegende Ansatz entwickelt. Um die im Folgenden Kapitel 3 entwickelten detaillierteren Ansätze zu verstehen, werden am Schluss noch einige Anforderungen an den Algorithmus formuliert.

2.1. Definitionen

In der Arbeit werden Ortsdaten in Form von Koordinaten verwendet. Eine Koordinate ist ein Punkt in der zweidimensionalen Ebene bezüglich eines orthonormalen Koordinatensystems. Differenzen zwischen Punkten sind mathematische Vektoren $v \in \mathbb{R}^2$ mit den üblichen Verknüpfungen komponentenweise Addition $+$ und Skalarmultiplikation $\cdot$ ausgestattet. Punkte können durch Addition mit Vektoren verrechnet werden, das Ergebnis ist wieder ein Punkt. Zusätzlich gibt es auf Vektoren eine Norm $\|\cdot\|$ und zwischen je zwei Punkten oder Vektoren eine Distanzmetrik $\mathrm{dist}(\cdot,\cdot)$. Die Norm ist die bekannte 2-Norm $\|v\| = \sqrt{v_1^2 + v_2^2}$, die Distanzmetrik ist die durch die Norm induzierte Metrik $\mathrm{dist}(p_1, p_2) = \|p_2 - p_1\|$.

Ein gerichteter Graph $G := (V, E)$ ist ein Tupel aus der Knotenmenge V und der Kantenmenge E. Die Kantenmenge E besteht dabei aus Knotentupeln, also $E = \{(v_1, v_2) \mid v_1, v_2 \in V\}$. Eine Kante $e = (v_1, v_2) \in E$ zeigt von ihrem Ursprung, dem Knoten v_1 zu ihrem Ziel v_2. Gilt außerdem $v_1 \neq v_2\ \forall (v_1, v_2) \in E$, ist der Ursprung also immer vom Ziel verschieden, dann enthält der Graph keine Schleifen und heißt einfacher, gerichteter Graph. Wir betrachten im Folgenden ausschließlich einfache, gerichtete Graphen.

Um einen solchen Graphen zur Lösung von Routenplanungs- und Mapmatching-Problemen nutzen zu können, werden Ortsangaben und Informationen über die nötige Fahrzeit im Straßennetzwerk benötigt. Das Straßennetzwerk wird als Graph repräsentiert. Die Ortsangaben sind Graphknoten zugeordnet. Es gibt daher eine Funktion pos, die jedem Knoten eine Koordinate zuweist. Für die Ortsinformationen einer Kante $e = (v_1, v_2)$ wird linear zwischen Ursprung und Ziel interpoliert $\mathrm{pos}(e, t) = \mathrm{pos}((v_1, v_2), t) = \mathrm{pos}(v_1)(1 - t) + \mathrm{pos}(v_2)(t)$, dabei gilt $0 \leq t \leq 1$. Um die Einbettung dieser Kante anzugeben, nutzt man das Bild von $[0, 1]$, $\mathrm{pos}(e, [0, 1]) = \mathrm{pos}((v_1, v_2), [0, 1]) =: \mathrm{pos}(e)$. Dies entspricht der Menge aller Punkte auf der Verbindungsstrecke zwischen v_1 und v_2. Diese Strecke $\mathrm{pos}(e)$ wird genutzt, um den Graphen zu zeichnen, und gibt Informationen über den Verlauf der Straßen des zugehörigen Straßennetzwerks. Die Kanten werden als gerade angenommen. Zum Abbilden von Kurven

müssen solange Knoten auf dem realen Verlauf der Kurve eingefügt werden, bis die Kurve hinreichend genau abgebildet ist. Die Fahrzeiten werden in den Kanten gespeichert. Es gibt entsprechend eine Funktion $w : E \to \mathbb{R}_{\geq 0}$, die für eine Kante das Kantengewicht ausgibt. Die Fahrzeit einer Kante wird durch eine Funktion berechnet. In realen Graphen tritt der Fall einer Fahrzeit von 0 nie auf. Im weiteren Verlauf dieser Arbeit gibt es allerdings Fälle, in denen 0 als Fahrzeit beziehungsweise Kantengewicht Sinn ergibt. Im Folgenden werden wir bei Funktionen, die Kanten erwarten, an deren Stelle gleichbedeutend zwei Knoten als Funktionsparameter nutzen. Wir schreiben zum Beispiel $w(v_1, v_2)$ statt $w((v_1, v_2))$.

Da der Graph gerichtet ist, kann man Vorgänger und Nachfolger eines Knotens v bestimmen, Nachfolger sind Knoten u, für die eine Kante mit Ursprung v und Ziel u existiert. Vorgänger von v sind Knoten, die als Nachfolger v haben. Die Menge der Nachfolger von v ist also $\text{succ}(v) \coloneqq \{u \in V \mid (v, u) \in E\}$, die Menge der Vorgänger $\text{pred}(v) \coloneqq \{w \in V \mid (w, v) \in E\}$.

Eine Folge $p \coloneqq v_1 v_2 \dots v_n$ der auf dem Weg besuchten Knoten heißt Pfad, die Zahl n heißt Länge des Pfades. Für einen Pfad muss gelten, dass $(v_i, v_{i+1}) \in E \; \forall 0 \leq i < n$ gilt, die Knoten müssen also durch Kanten verbunden sein. Für Pfade wird der Begriff der Einbettung übernommen, indem die Einbettungen der Kanten vereinigt werden: $\text{pos}(p) \coloneqq \cup_{i=0}^{n-1} \text{pos}(v_i, v_{i+1})$. Das Gewicht des Pfades entsteht als Summe der Kantengewichte: $w(p) \coloneqq \sum_{i=0}^{n-1} w(v_i, v_{i+1})$.

Es kann gewünscht sein, Kanten zu unterteilen, ohne die Geometrie zu verändern. Dies ist zum Beispiel sinnvoll, wenn Kanten in eine Beschleunigungsstruktur eingefügt werden sollen. Wird zum Beispiel ein Quadtree genutzt, so müssen die Kanten für einfachen Zugriff an den Grenzen der Quadtree-Kacheln unterteilt werden. Wir nennen eine solche Unterteilung lineare Einbettung, wenn die Einbettung des Pfades $\hat{e}(e)$ der Einbettung von e entspricht. Ein Graph ist eine lineare Unterteilung von sich selbst, mit $\hat{e} = \text{id}, \hat{e}(v_1, v_2) = v_1 v_2$.

Dies kann auf Graphen übertragen werden. Um den feiner aufgelösten Graphen $\hat{G} \coloneqq (\hat{V}, \hat{E})$ des Graphen $G \coloneqq (V, E)$ zu definieren, werden zwei injektive Funktionen $\hat{v}$ und $\hat{e}$ benötigt. Diese übersetzen vom groben in den feinen Graph. Die Funktion $\hat{v} : V \to \hat{V}$ übersetzt die Knoten, dabei sollen die Positionsangaben erhalten bleiben, also $\text{pos}(\hat{v}(v)) = \text{pos}(v) \forall v \in V$. Die Funktion $\hat{e}$ transformiert die Kanten. Da neue Knoten eingefügt werden, bildet $\hat{e}$ von E in die Pfade von $\hat{G}$ ab. Hier werden die durch $\hat{v}$ eingefügten Knoten benötigt. Die Funktion $\hat{e}$ erhält die Kantengewichte, es gilt also $w(\hat{e}(e)) = w(e) \forall e \in E$. Damit haben kürzeste-Wege-Probleme in G und $\hat{G}$ äquivalente Lösungen. Für die Nutzung im Mapmatchingproblem muss das Gewicht von e zusätzlich gleichmäßig auf die neu entstehenden Kanten in $w(\hat{e}(e))$ aufgeteilt werden.

Ein typisches Problem, dass auf einem gewichteten Graphen gelöst werden muss, ist das kürzeste-Wege-Problem beziehungsweise kürzeste-Pfade-Problem. Die Problemstellung ist, zwischen einem Ursprungsknoten v_1 und einem Zielknoten v_n den Pfad mit geringstem Gewicht zu finden. Da die Gewichtsfunktion in $\mathbb{R}_{\geq 0}$ abbildet, kann kein Knoten v mehrfach auf diesem Pfad vorkommen. Der hypothetische Teilpfad, der mit dem ersten Vorkommen von v anfängt und mit dessen letzten Vorkommen endet, hat ein Gewicht ≥ 0. Damit kann der Teilpfad aus dem Ergebnis entfernt werden kann. Dieses Problem wird durch Dijkstras Algorithmus [Dij59] effizient gelöst. Dieser löst das Problem für einen Graphen $G = (V, E)$ in $\mathcal{O}(|V| \log(|V|) + |E|)$.

2.2. Problemdefinition

Ein Objekt bewegt sich durch ein real existierendes Straßennetzwerk. Da es sich in unserem Fall meist um ein Auto handelt, werden wir das Objekt im Folgenden Auto nennen. Gegeben sind eine Reihe von inexakten Positionsmessungen des Autos, zusammen mit Zeitstempeln dieser Positionsmessungen. Gegeben ist weiterhin ein Graph, der das Straßennetzwerk

modelliert. Aus diesen Eingaben soll der gefahrene Pfad des Autos im Straßennetzwerk rekonstruiert werden. Ausgegeben wird ein Pfad im Graph.

Eine Folge $M \coloneqq (c_1, t_1)(c_2, t_2) \dots (c_n, t_n)$ aus gemessenen möglicherweise inexakten Koordinaten $c_i = (x, y)_i$ und geordneten Zeitstempeln t_i heißt Messreihe. Die einzelnen Tupel (c_i, t_i) heißen Messpunkte. Wählt man die Messpunkte (c_i, t_i) als Knoten und verbindet die aufeinanderfolgenden Messpunkte $(c_i, t_i), (c_{i+1}, t_{i+1})$ mit Kanten, so erhält man einen Polygonzug. Wir schreiben für diesen Polygonzug auch P_M. Außerdem benötigen wir die Differenzen der Folgenglieder in zeitlicher $\Delta \operatorname{time}_M(i) = t_{i+1} - t_i$ und räumlicher $\Delta \operatorname{loc}_M(i) = c_{i+1} - c_i$ Dimension.

Formal haben wir zu einem Graphen $G \coloneqq (V, E)$ und einer Messreihe M das match(G, M)-Problem. Der gefahrene Pfad der Messreihe im Graphen ist die Lösung dieses Problems.

2.3. Ansatz

Zur effizienten Lösung des match-Problem soll auf die Geschwindigkeit von Dijkstras Algorithmus gesetzt werden. Dazu soll der Algorithmus selbst unverändert laufen, nur die Gewichtsfunktion w des Graphen soll durch eine abgewandelte Gewichtsfunktion $\bar{w}$ ausgetauscht werden. Diese Funktion wird so gewählt, dass eine Lösung des kürzeste-Wege-Problems bezüglich $\bar{w}$ eine Lösung des Problems match(G, M) ist.

Eine triviale solche Funktion $\bar{w}$ wäre, einen bestehenden Mapmatching-Algorithmus (*) wie in Abschnitt 1.1 vorgestellt zu nutzen: Sei p der Lösungspfad für (*), und für $\bar{w}(e)$ gelte (2.1). Offensichtlich ist der einzige Pfad in G, der nicht Gewicht unendlich hat, die Lösung.

$$\bar{w}(e) = \begin{cases} 0, & e \in p \\ \infty, & \text{sonst} \end{cases} \tag{2.1}$$

In Dijkstras Algorithmus wird immer der Knoten mit dem kleinsten Gewicht aus der Priorityqueue genommen. In dem Beispiel werden genau $|p|$ Knoten aus der Queue genommen, entsprechend schnell wird die Lösung berechnet. Wir nehmen an, dass sich dieser Vorteil in den hier beschrieben Algorithmus überträgt. Kanten, die nach der Berechnung ein kleines Gewicht haben, liegen nahe an den Messpunkten und sind daher bereits gute Kandidaten für den gesuchten Pfad. Da von den Zielen dieser Kanten aus neue Kanten gesucht werden, werden hier wieder Kanten gewählt, die nahe am Polygonzug liegen. Die weiterführenden Kanten, die am besten auf den Polygonzug passen, haben dann wieder ein sehr kleines Gewicht und werden entsprechend schnell abgearbeitet. Der Algorithmus sucht daher bevorzugt in Richtung des Polygonzugs. Da nach dieser Annahme der Suchraum des Algorithmus einem Kanal entlang des Polygonzugs folgt, nennen wir diesen Effekt Kanaleffekt.

2.4. Anforderungen

Eine wichtige Designentscheidung für den Algorithmus ist, invalide unter linearen Unterteilungen zu sein. Dies erhöht die Flexibilität der mit dem Graphen möglichen Berechnungen. Wie schon bei der Definition der linearen Unterteilungen beschrieben, erlaubt dies zum Beispiel das Einfügen in geometrische Datenstrukturen wie einen Quadtree. Die Anforderung macht es zusätzlich einfach, das Ergebnis stetig bezüglich nicht linearer Unterteilungen zu machen. Damit liefert der Algorithmus vergleichbare Ergebnisse, unabhängig davon, ob ein etwas gröber aufgelöster oder sehr fein aufgelöster Graph als Eingabe verwendet wird.

Die andere dem Ansatz zugrunde liegende Annahme ist, dass Autos rationales Verhalten zeigen. Wir setzen dies gleich mit der Vermutung, dass Autos mindestens für längere Abschnitte Routen minimalen Gewichts, also kürzester Fahrzeit fahren. Wir fordern deshalb, dass die Fahrzeit der ursprünglichen Gewichtsfunktion w in die Berechnung der neuen Gewichtsfunktion $\bar{w}$ eingehen. Für die Lösung folgt daraus, dass bei zweien durch den Abstand plausiblen Routen die schnellere gewählt wird. Besteht zum Beispiel die Wahl zwischen Autobahn und Landstraße, sollte die Lösung die Autobahn benutzen. Diese Annahme erschwert das korrekte Lösen von Messreihen, bei denen diese Annahme verletzt ist. Es können zum Beispiel Fehler auftreten, wenn ein Autofahrer die Autobahn nicht benutzen möchte. Fährt er in diesem Fall aber parallel zur Autobahn auf einer normalen Straße, so wählt ein auf dieser Annahme basierender Algorithmus möglicherweise die falsche Strecke. Ein weiteres Problem ist, dass Schleifen nach dieser Annahme niemals valide sind. Ein kurzer Abstecher zum Supermarkt wird daher von einem solchen Ansatz ignoriert.

Der Algorithmus soll invariant unter linearen Unterteilungen sein. Die in dieser Arbeit gewählte Implementierung dieser Anforderung ist das Wegintegral, da das Integral abzählbar oft unterbrechbar ist, ohne dass sich das Ergebnis ändert: $\int_a^b f(t)dt = (\int_a^c f(t)dt)+(\int_c^b f(t)dt)$ Die Funktion $\bar{w}$ soll aus dem Wegintegral über die Einbettung der Kante hervorgehen. Integriert wird eine zweidimensionale, stetige Funktion f. Damit ist ein Ziel der Arbeit, eine solche stetige Funktion f zu finden, sodass $\bar{w}$ eine Lösung des Problems in 2.2f ist. Die Funktion $\bar{w}$ soll außerdem die ursprüngliche Gewichtsfunktion w beachten. Durch den Ansatz des Integrals von $f = \bar{w}$ muss w in f eingehen.

Im folgenden Kapitel werden zwei Ansätze beschrieben, eine solche Funktion f zu konstruieren. Es wird theoretisch überprüft, ob sie sich für das match-Problem eignen.

3. Ansätze

Die Kantengewichte werden mithilfe von Wegintegralen durch eine zweidimensionale Funktion verringert. Der Inhalt dieses Kapitels ist die Konstruktion solcher Funktionen, damit sie sich zum Mapmatching eignen. Dazu wurden zwei Konstruktionsansätze evaluiert, die hier beschrieben werden sollen.

3.1. Gaußkurven

3.1.1. Idee

Die Messpunkte weisen einen örtlichen Fehler auf. Dieser Fehler wird als stochastisches Rauschen im zweidimensionalen Raum angenommen, dass sich durch Gaußglocken modellieren lässt. Es wird daher versucht, aus diesen Glocken eine Gewichtsfunktion zu entwickeln. Das stochastische Rauschen kann um den Punkt als Wahrscheinlichkeitsverteilung modelliert werden, also muss der Mittelpunkt der Glocke gleichzeitig dem Messpunkt entsprechen.

Eine Gaußglocke im Zweidimensionalen berechnet sich aus den Parametern x, μ, σ^2, wie in Gleichung 3.1 zu sehen. Dabei ist $\mu \in \mathbb{R}^2$ der Mittelpunkt der Glocke und entspricht dem Messpunkt. Die Dichtefunktion f wird an dem Punkt $x \in \mathbb{R}^2$ ausgewertet und σ^2 ist die Varianz, von der die Streckung der Glocke abhängt.

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\operatorname{dist}(x,\mu)^2}{2\sigma^2}\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\|x-\mu\|^2}{2\sigma^2}\right) \tag{3.1}$$

Durch die Messreihe ist μ bereits festgelegt und x ist die Variable, über die nach dem in Kapitel 2.4 beschriebenen Ansatz integriert werden soll.

Damit Dijkstras Algorithmus sinnvoll funktioniert, dürfen die Kantengewichte nicht negativ sein. Als erster Ansatz wurde versuchsweise das Kantengewicht einer unendlichen Gerade spezifiziert. Die Begründung dafür war, dass die Gaußglocke ebenfalls eine unendliche Ausdehnung hat. Das Kantengewicht durch einen Messpunkt sollte minimal sein, also sollte eine Kante durch den Messpunkt Gewicht 0 erhalten. Für solche Kanten k durch den Messpunkt μ mit Länge len, Einbettung k_s und Gewicht w soll also 3.2 gelten:

$$\lim_{\text{len}\to\infty} \int_{k_s} f(x)\mathrm{d}x = 0 \tag{3.2}$$

Der Einfluss der Gaußkurve nimmt mit steigendem Abstand zum Mittelpunkt ab und wird unendlich weit entfernt zu null. Eine von der Gaußglocke nicht beeinflusste, also unendlich ferne Kante soll ihr Ursprungsgewicht behalten.

Aus diesen Designentscheidungen leiten wir die folgende Konstruktion her. Damit eine von der Gaußglocke nicht betroffene Kante ihr Ursprungsgewicht behält, wird als Grundfunktion $f_{\text{base}} = \frac{1}{\text{len}}$ gewählt, mit dem Integral $\int_{k_s} \frac{1}{\text{len}} \mathrm{d}x = 1$. Das entspricht dem Integral von $-\infty$ bis ∞ der Dichtefunktion der eindimensionalen Gaußkurve, dass unabhängig von σ^2 auch gleich 1 ist. Das Gewicht der Kante kann an beide Funktionen multipliziert werden und bleibt daher an dieser Stelle unbetrachtet. Es bietet sich an, die Gesamtfunktion einer Kante bezüglich eines Punktes μ als $f_{\text{result}} = \frac{1}{\text{len}} - f(x \mid \mu, \sigma^2)$ zu wählen.

Es muss nur noch σ^2 passend bestimmt werden. Der erste Ansatz dafür war, das stochastische Rauschen des GPS-Signals als Startwert zu verwenden. Diese beträgt laut [DoD08] in 95 % der Fälle unter 8 Meter, in den Versuchen wurde großzügig mit einer Standardabweichung $\sigma = 15\ m$ gerechnet. Bei der Normalverteilung befindet sich mehr als 99,99 % der Wahrscheinlichkeitsmasse im Interval $[\mu - 4\sigma, \mu + 4\sigma]$, daher wird die Normalverteilung außerhalb von 4σ als null angenommen. Um negative Kantengewichte zu verhindern, wurden Teilintegrale der Kanten bei null abgeschnitten.

Bei ersten Versuchen stellte sich heraus, dass die Kantengewichte nicht wie erwartet verkleinert wurden. Dies ließ sich auf die Größe der Gaußglocken zurückführen. Diese wären viel zu klein, um einen hinreichenden Einfluss auf den Pfad zu nehmen. Um dieses Problem zu beheben, sollte daher σ^2 so erhöht werden, dass mehr Kanten innerhalb des relevanten Einflussbereiches liegen. Als neue Idee wurde demzufolge diskutiert, den Abstand der Messpunkte in die Berechnung der Standardabweichung mit einzubeziehen. Bei der Diskussion wurden, noch vor einer stehenden Implementierung, theoretische Probleme entdeckt, die im Folgenden beschrieben sind.

3.1.2. Theoretische Probleme

An diesem Ansatz und den zugrundeliegenden Annahmen stellen sich mehrere Probleme heraus:

Unendliche Kanten haben Gewicht 0

Diese Anforderung erzeugt Probleme, denn sie erzwingt, dass es negative Kantengewichte gibt. Dies lässt sich mit der Form der Gaußkurve begründen. Die Kurve ist stetig und nicht konstant. Das finale Gewicht soll null sein, $\frac{1}{\text{len}}$ ist aber echt größer 0. Die Gesamtfunktion nähert sich für den Abstand gegen unendlich beliebig nahe an $\frac{1}{\text{len}}$ an. Entsprechend muss es mindestens ein Gebiet geben, in dem $f_{\text{result}}(x) < 0$ gilt. Dementsprechend ist auch das Integral kleiner null. Dies lässt sich in den Abbildungen 3.1 und 3.2 nachvollziehen. Abbildung 3.1 zeigt in Blau die Gewichtsfunktion einer unbetroffenen Kante und in Gelb die Gewichtsfunktion einer komplett betroffenen Kante. In Rot ist der Verlauf des Integrals eingezeichnet, welches am Ziel der Kante den Wert null hat. Damit ist die Anforderung bezüglich des Kantengewichts anscheinend erfüllt.

Man kann schon am Verlauf der gelben Funktion erkennen, welches Problem das weiter oben beschriebene Begrenzen der Integralwerte hat: Damit am Ziel der Kante das Gewicht null herauskommt, muss die Gaußfunktion negative Werte annehmen. Dies ist auch grafisch in Abbildung 3.2 demonstriert. In Gelb ist wieder der Gewichtsverlauf der Kante eingezeichnet, in Rot das dazugehörige Integral. Erwartungsgemäß hat dieses Integral den Wert null. Nach der in Abschnitt 2.4 beschriebenen Unterteilungseigenschaft soll das Ergebnis invariant unter linearen Unterteilungen sein. Wir unterteilen die Kante jetzt so, dass genau an den Vorzeichenwechseln der gelben Funktion Knoten eingefügt werden. In diesem Fall kommt im negativen Bereich das oben erwähnte Abschneiden zum Tragen und führt zum Verlauf der magentafarbenen Funktion. Das dazugehörige Integral ist grün eingezeichnet und offensichtlich nicht gleich zum roten Integral. Damit ist diese provisorische Lösung ausgeschlossen.

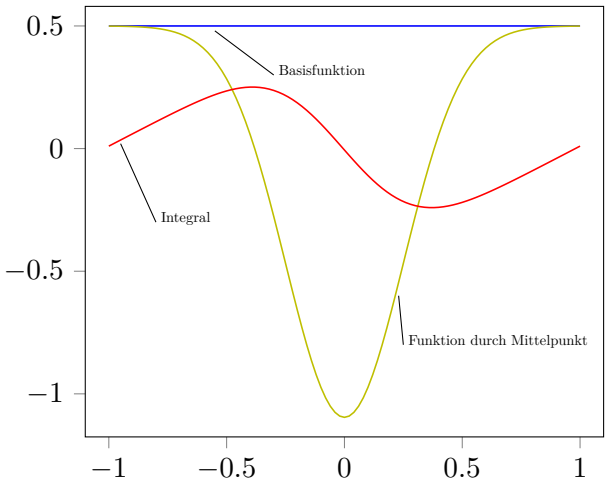


Abbildung 3.1.: Die Gewichtsfunktion f_{result} und ihr Integral für eine Kante durch den Mittelpunkt

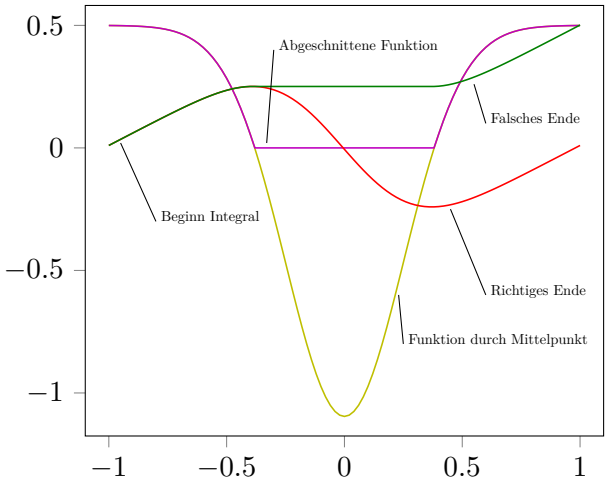


Abbildung 3.2.: Die Gewichtsfunktion f_{result} und die Gewichtsfunktion dreier am Nullstellenübergang abgeschnittener Kanten. Das Integral der abgeschnittenen Kante ist nicht identisch.

Es wurden mehrere alternative Ansätze diskutiert, dieses Problem zu beheben. Der einfache Ansatz wäre gewesen, das Problem zu ignorieren. Dijkstras Algorithmus funktioniert eingeschränkt weiterhin, solange es keine negativen Kreise gibt, negative Kantengewichte sind aber in Ordnung. Für diesen Ansatz hätte allerdings sichergestellt werden müssen, dass keine negativen Kreise entstehen können. Ein Gegenbeispiel dazu ist ein Kreisverkehr, der geschlossen in dem Gebiet liegt. Abhängig von der Wahl von σ ist dies nicht unwahrscheinlich. Zusätzlich ist es mit negativen Kantengewichten nicht mehr möglich, Dijkstras Algorithmus vorzeitig abzubrechen, wenn der Zielknoten das erste Mal gefunden wurde. Dieser Lösungsansatz ist also nicht praktikabel.

Die zweite Idee war, das überschüssige Gewicht der Kanten zu akkumulieren. Bei späteren Kanten mit positivem Kantengewicht wird das gespeicherte Gewicht dann zur weiteren Verringerung der Kante verwendet. Allerdings hat auch dies mehrere Probleme. Wie schon die erste Idee scheitert auch diese Idee an Kreisverkehren. Kann in diesen in beide Richtungen gefahren werden, ist es nicht eindeutig, in welche Fahrtrichtung der Algorithmus zuerst weitersucht. Dies führt dazu, dass im Zweifel ein zu kleines Gewicht zum Weitersuchen genutzt wird. Wie bei dem vorherigen Ansatz gibt es also wieder Probleme mit negativen Kreisen. Zusätzlich wird aus dem effizienten Dijkstra ein multikriterieller Algorithmus, der weniger effizient ist.

Aus diesem Grund wurde die Anforderung Unendliche Kanten haben Gewicht 0 aufgegeben.

Verknüpfung der Integrale

Für den nächsten Ansatz wurde versucht, negative Funktionswerte und damit negative Kantengewichte zu vermeiden. Der kleinstmögliche Funktionswert der neu zu konstruierenden Funktion f_{result} ist daher 0. Das Maximum der Gaußkurve ist am Punkt μ, also fordern wir $f_{\text{result}}(\mu) \stackrel{!}{=} 0$. Die Gaußglocke wird durch einen Vorfaktor so skaliert, dass diese Forderung gilt. Wir fordern darüber hinaus wieder, dass nicht betroffene Kanten ihr ursprüngliches Gewicht behalten sollen. Die Funktion für diese Kanten ist daher wieder $f_{\text{base}} = \frac{1}{\text{len}}$. Wir wählen als Gesamtfunktion f_{result} wieder eine Funktion von der Form $f_{\text{result}} = f_{\text{base}} = \frac{1}{\text{len}} - \alpha \cdot f(x \mid \mu, \sigma^2)$, wobei wir das α so wählen, dass Gleichung 3.3 gilt. Die mit $*$ markierte Gleichheit gilt per Konstruktion oben.

$$\min_{x \in \mathbb{R}^2} f_{\text{result}}(x) \stackrel{*}{=} f_{\text{result}}(\mu) \stackrel{!}{=} 0 \tag{3.3}$$

Wie schon im ersten Ansatz ist auch hier mit der Wahl von σ^2 und durch die Forderung oben die Gaußkurve eindeutig festgelegt. Es soll sich wie in Abschnitt 2.3 beschrieben ein Kanal ausbilden. Dafür müssen sich die Gaußkurven so überschneiden, dass ihre Werte einen Einfluss aufeinander ausüben. Es stellt sich daher die Frage, wie man die Gaußkurven der verschiedenen Messpunkte miteinander verknüpft.

Zum einen ist dies additiv möglich, indem man die Gesamtwerte der Funktionen $f_{\text{result}_{\mu_i}}$ zusammenaddiert. Dabei tritt das Problem auf, dass viele Punkte in der Nähe das Gewicht an x erhöhen, da der Beitrag dieser Punkte ≥ 0 ist. Um dies zu vermeiden, könnte der Beitrag der Punkte mit einem Vorfaktor versehen werden. In diesem Fall haben Punkte mit großem Abstand einen zu starken und gleichzeitig Punkte mit geringem Abstand einen zu geringen Einfluss auf das Ergebnis. Die Gewissheit an einem Messpunkt, dass dies ein Punkt auf dem genommenen Pfad ist, ist maximal, daher sollte das Gewicht hier minimal sein. Ein anderer Messpunkt mit additivem Gewicht verhindert diese Einschätzung.

Die andere, angedachte Alternative war, die Gewichte der Punkte miteinander zu multiplizieren. Dies hätte den Vorteil, dass alle Punkte automatisch ein kleineres Gewicht hätten und dass Multiplikation gut zu der Exponentialfunktion in der Gaußglocke passen

würde. Dazu müsste allerdings das Integral $\int_{k_s} \prod_I (1 - \alpha * f(x \mid \mu_i, \sigma^2))\mathrm{d}x$ gelöst werden, was aufgrund der Tatsache, dass die Normalverteilung nicht geschlossen integrierbar ist, nicht trivial ist.

Aus diesen Gründen wurde die Idee, Normalverteilungen zu nutzen, aufgegeben und stattdessen der in der folgenden Sektion beschriebene Ansatz evaluiert.

3.2. Kantenzuggewichte

Die Messreihe ist zeitlich sortiert und es kann der Polygonzug P_M wie in Kapitel 2.2 beschrieben konstruiert werden. Die Kanten zwischen den Messpunkten geben eine Approximation der Richtung des Autos an. Da das Auto den im Straßengraphen vorhandenen Kanten folgt, sollten die Kanten des Polygonzugs den Kanten des Straßengraphen der Richtung nach entsprechen. Die Länge der Polygonzugkante ist von der Geschwindigkeit abhängig. Bei gleichem zeitlichen Abstand der Punkte steigt deren räumlicher Abstand bei höherer Geschwindigkeit.

Schon im Abschnitt 3.1 über die Gaußkurven ist beschrieben, dass in einem bestimmten Umkreis um die Punkte der Einfluss dieser Punkte groß ist, in größerer Entfernung dagegen nicht mehr. Das Gebiet, in dem dieser Einfluss nicht vernachlässigbar ist, nennen wir Einflussbereich. Die Idee dieses Ansatzes ist wieder die Konstruktion eines Einflussbereiches aus den Knoten des Polygonzugs. Im Gegensatz zu den Gaußkurven in Abschnitt 3.1 wird diesmal eine geometrische Form konstruiert und erst im zweiten Schritt daraus eine Funktion.

Die Größe des Einflussbereiches

Für die Ausdehnung des Einflussbereichs ist es nicht sinnvoll, harte Infima und Suprema anzugeben. Ein solcher Schritt würde die Möglichkeiten der konstruierbaren Formen einschränken. Es kann aber versucht werden, unscharfe Schranken anzugeben. Der Einflussbereich jedes Messpunktes sollte mindestens so groß sein, dass er die Verbindungskanten zu seinen Nachbarn komplett enthält. Diese sind die beste Richtungsangabe, die aus den Daten konstruiert werden kann. Damit ist eine untere Schranke gefunden. Eine obere Schranke zu wählen ist schwierig, da auf diese Weise die möglichen Formen eingeschränkt werden könnten. Trotzdem sollte der Einflussbereich nicht beliebig weit ausgedehnt sein, da dies zum einen die Rechenzeit steigert. Zum anderen gehen wir davon aus, dass zwischen zwei Messpunkten keine großen Abweichungen von der Luftlinie gemacht werden. Die Begründung ist, dass Straßen recht gerade verlaufen. In den betrachteten Daten ist der Zeitabstand zu gering, als dass das Auto mehr als eine Kreuzung nehmen könnte. Daher scheidet auch diese Begründung aus.

Linie

Die genannte untere Schranke besteht aus den Verbindungskanten zwischen den Messpunkten. Da integriert wird, ist per Annahme die Gewichtsfunktion außerhalb der Linie gleich der bekannten Basisfunktion f_{base}. Auf der Messkante ergibt die Gewichtsfunktion null. Das Ursprungsgewicht der Kante wird aus der Gleichung faktorisiert und hier nicht näher betrachtet. Es sind dann 3 Fälle zu unterscheiden:

1. Die Einbettung der Graphkante e geschnitten mit der Einbettung der Messkante m ist unendlich groß:
   Dann liegen die Kanten mindestens teilweise aufeinander und der Ursprungs- und Zielpunkt der entstehenden Schnittkante sind auch Ursprungs- beziehungsweise Zielpunkt einer der Ursprungskanten. Das zu integrierende Gewicht w der Graphkante

ist dann $w(x) = \begin{cases} 0, & x \in \text{pos}(e) \cap \text{pos}(m) \\ f_{\text{base}}, & \text{sonst} \end{cases}$. Durch das Integral bleibt als Gewicht der Kante die Länge des Teils der Kante, die nicht im Schnitt liegt.

Dieser Fall ist allerdings unwahrscheinlich, da auch bei dem schwächsten Rauschen die Kanten nicht mehr parallel sind oder den gleichen Punkt schneiden, und es so nur maximal einen Schnittpunkt gibt.

2. Die Kanten schneiden sich in einem Punkt:
   Die zu integrierende Funktion ist für alle bis auf eine endliche Menge von Punkten gleich dem Kantengewicht, das Integral hat also den Wert des Kantengewichtes.
3. Die Kanten schneiden sich nicht. Auch in diesem Fall ist das Integral das Kantengewicht.

Die neue, durch Auswertung des Integrals entstehende Gewichtsfunktion hat also fast immer das ursprüngliche Kantengewicht und löst damit nicht das Mapmatching-Problem.

Die Linie ist daher wie andere eindimensionale Funktionen ungeeignet.

Kreis

Der Kreis ist eine einfache, zweidimensionale Form. Um die untere Schranke zu erfüllen, muss die Einbettung der Kanten einer Teilmenge des Kreises entsprechen.

Man kann die Berechnung der Integrale vereinfachen, indem das Gewicht am Mittelpunkt gleich null und auf dem Rand gleich dem Kantengewicht gesetzt wird. Dies hat den zusätzlichen Vorteil, dass am Messpunkt das Gewicht minimal ist. Damit beide Verbindungskanten abgedeckt sind, muss der Radius dem Maximum der Abstände zu den Nachbarmesspunkten betragen. Für die Werte dazwischen wird auf dem Radius linear skaliert. Wir vernachlässigen in den Formeln das Ursprungsgewicht der Gerade, da dies ein konstanter Faktor ist. Der Funktionswert an dieser Stelle ist also nur vom Abstand Mittelpunkt↔Funktionswert abhängig:

$$f_{p_o}(p) = \frac{1}{r} \cdot \begin{cases} \frac{dist(p_0,p)}{\text{len}} & dist(p_0,p) < r \\ \frac{r}{\text{len}} & dist(p_0,p) \geq r \end{cases} \tag{3.4}$$

Der Faktor $\frac{1}{r}$ dient der Normierung, sodass der äußere Rand zur Umgebung gleichgesetzt wird, der Rand hat das Gewicht $\frac{1}{\text{len}}$. Das Wegintegral außerhalb des Kreises ist also $\int_e \frac{1}{\text{len}} \mathrm{d}t = 1$.

Diese Funktion lässt sich integrieren, da es sich um eine zusammengesetzte Funktion aus einem Kegel um p und einer Ebene handelt. Mithilfe des Schnittpunkts der Kante mit dem Rand des Kreises kann das Integral als Integral der Kante durch den Kegel und Integral der Kante durch die Ebene berechnet werden. Die Details des Integrals durch den Kegel sind samt Formeln im Anhang unter A.2 beschrieben.

Diese Form hat allerdings folgende Probleme:

- Der Einfluss auf die Kanten hängt nur von einem Nachbarn im Polygonzug ab, der Andere wird ignoriert. Dies kann dazu führen, dass bei Unterschieden im Abstand ein Nachbar übermäßig viel Einfluss gewinnt und so der andere Messpunkt irrelevant wird. Dies ist zum Beispiel der Fall, wenn bei fünf Knoten $a \dots e$ die Abstände ab und de sehr groß, aber bc und cd klein sind. Liegt c nicht auf der Verbindungslinie ae, wird dessen Position nicht beachtet. Es ist außerdem möglich, dass sehr viele Messpunkte Einfluss auf eine Kante nehmen. Dies ist zum Beispiel bei einer Folge von Punkten (x_i, y_i) der Fall, mit $x_i = 0$ und $y_i = \frac{y_{i-1}}{2}$. Die Messpunktfolge hat einen Häufungspunkt in $(0, 0)$ und durch die Radiuskonstruktion sind alle Messpunkte an dem Gewicht von $(0, 0)$ beteiligt.

- Es gibt keine explizite Belohnung für das Folgen der Richtung des Polygonzugs. Die Einbettung der Kante des Polygonzugs ist eine Approximation der Fahrtrichtung des Autos. Die Information über die Fahrtrichtung geht verloren. Der Kreis ist nur von den Punkten abhängig.

Rechteck

Eine weitere einfache Form ist das Rechteck. Zur Lösung dieses Problems kann diese Form in zwei Varianten eingesetzt werden, punktorientiert und kantenorientiert.

Im punktorientierten Fall wird die gleiche Idee wie bei dem Kreis verfolgt, das Rechteck wird um einen Messpunkt herum konstruiert. Um die untere Schranke zu erfüllen, muss das Rechteck die Kante des Polygonzugs überdecken. Der Rand des Rechteckes muss dann durch die benachbarten Messpunkte verlaufen. Das reicht noch nicht, um das Rechteck eindeutig zu orientieren. Eine Forderung kann sein,a dass das Rechteck minimal sein soll. In diesem Fall liegen beide Nachbarn und der Messpunkt selbst auf dem Rand. Der Punkt, an dem das Gewicht am niedrigsten ist, liegt also auf dem Rand. Der Einfluss dieses Punktes ist dann nur auf der Seite, auf der das Rechteck liegt, gegeben. Ein Pfad, der knapp auf der anderen Seite entlang verläuft, wird also mit dem Ursprungsgewicht der Kanten gewichtet. Um dieses Problem zu beheben, kann das Rechteck an der Seite gespiegelt werden, auf der Messpunkt liegt. Dabei tritt allerdings das Problem auf, dass ein Pfad, der weit von den ursprünglichen Wegpunkten entfernt liegt, bezüglich dieses Messpunktes genauso bevorteilt wird. Bei einem Zickzack-Muster von Punkten werden so Pfade plausibel, die den Abstand des Zickzackmusters in die andere Richtung verfolgen.

Aus diesem Grund betrachten wir kantenorientierte Rechtecke: Der im Paragraphen „Linie“ beschriebene Ansatz, nur der Kante des Polygonzugs zu folgen, scheitert an der Eindimensionalität ebendieser. Wir verbreitern diese Linie, sodass ein Rechteck entsteht. Die Polygonzugkante erhält wie oben das Gewicht null, zum Rand hin wird linear skaliert. Aus Symmetriegründen sollte der Abstand parallel zu der Linie auf beiden Seiten gleich groß sein, genauso die Verlängerung der Linie zu Ursprung und Ziel. Diese Abstände zu wählen ist allerdings nicht trivial, um Stetigkeit der zu integrierenden Funktion zu gewährleisten, müssen beide Abstande echt größer null sein. Die Rechtecke allein werden also auch verworfen.

Baseballstadion

Aus den oben genannten Gründen funktionieren die Formen nicht, bieten allerdings einige Vorteile, die wünschenswert sind:

1. Der Kreis ist einfach und definiert durch zwei Punkte seine gesamte Form. Beim Integral werden zusätzlich leichte Abweichungen vom Polygonzug kaum, große Abweichungen dagegen deutlich bestraft.
2. Das kantenorientierte Rechteck betont das Folgen des Polygonzugs.

Wir vereinen diese Vorteile in einer neuen Form, die wir Baseballstadion nennen. Es wird abhängig von vier aufeinanderfolgenden Knoten $v_1v_2v_3v_4$ im Polygonzug eine Form konstruiert, die sich auf die Kante $(v_2, v_3) =: e$ im Polygonzug P_M bezieht. Wie bei der kantenorientierten Konstruktion für das Rechteck ist das Gewicht der Punkte, die auf der Einbettung von e liegen, gleich null. Es wird ebenfalls wieder senkrecht zu e zum Rand linear interpoliert. Die äußeren Punkte v_1 und v_4 gehen nur durch den Abstand zu v_2 beziehungsweise v_3 ein, ihre relative Lage wird ignoriert. Wir nutzen diesen Abstand, um Kreise K_2, K_3 um v_2 und v_3 zu legen, mit dem Abstand als Radius. Der Radius von K_2 beträgt also $r(K_2) = dist(v_1, v_2)$, der von K_3 entsprechend $r(K_3) = dist(v_3, v_4)$. Diese

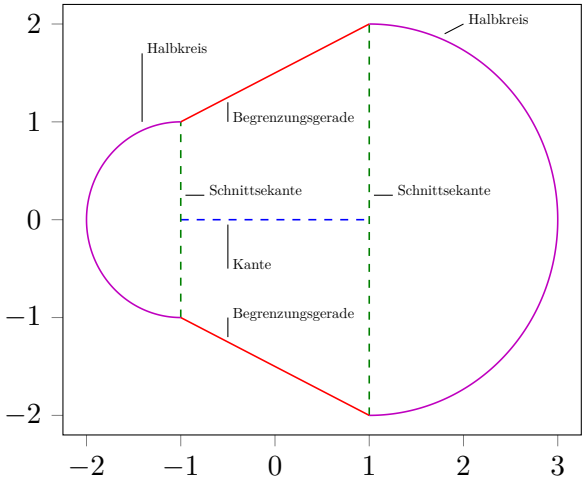


Abbildung 3.3.: Beschreibung des Baseballstadions.

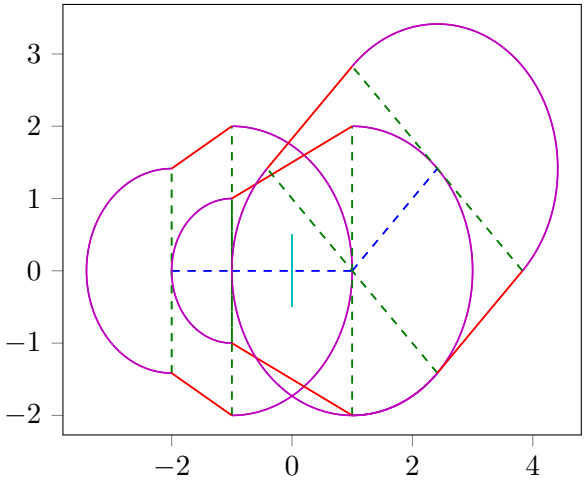


Abbildung 3.4.: Sammlung aus 3 Baseballstadien und einer Geraden.

Kreise werden in Halbkreise zerschnitten, sodass die Schnittsekante senkrecht zu e ist. Diese Schnittsekante verläuft durch den Mittelpunkt des Kreises. Die Schnittpunkte der Schnittsekante mit dem Halbkreis werden mit dem entsprechenden Punkt des anderen Halbkreises verbunden. Es entsteht damit eine Form, die durch die Halbkreise und die Verbindungslinien der Halbkreise begrenzt wird. Abbildung 3.3 zeigt diese Konstruktion.

Für eine Kante ist es interessant zu wissen, wie viele Punkte sie beeinflussen, da dadurch die Laufzeit mitbestimmt wird. Im Normalfall sind das die zwei Punkte, zwischen denen die Kante liegt. Diese erzeugen drei Stadien, davon zwei durch die Halbkreise und eins durch das Trapez in der Mitte, wie in Abbildung 3.4 gezeigt wird.

Des Weiteren fällt auf, dass die Breite des Vierecks, dass die Kante umgibt, vom Abstand beider Randknoten abhängt. Liegen detaillierte räumliche Informationen vor, gibt es also eine hohe räumliche Auflösung, und die Flanken sind steil. Dies hat den Effekt, dass bei guter Informationslage weiter entfernte Kanten schlecht gewichtet werden. Bei größerem Abstand der Messpunkte sinkt die Steigung der Flanken, damit auch der Malus weiter entfernter Kanten. Ist die räumliche Auflösung so hoch, dass das Rauschen der Messpunkte relevant wird, so liegt eine hohe Konzentration der Punkte vor. In diesem Fall kann das Rauschen durch andere Punkte ausgeglichen werden. Diese Konstruktion erzwingt bei guter Auflösung, dass der Pfad sich eng an den gegebenen Pfad hält, bei großem Abstand wird diese Anforderung etwas abgeschwächt.

Das Integral durch dieses Stadion setzt sich aus mehreren Teilen zusammen, den bereits beschriebenen Integralen durch die Halbkreise und die Ebene, und den Integralen durch die Seitenflächen. Die Berechnung des Integrals wird in drei Schritten durchgeführt. Zuerst wird die Graphkante mit den Kanten des Baseballstadions geschnitten und die Schnittpunkte sortiert. Zwischen den Schnittpunkten wird dann das geeignete Integral ausgerechnet. Wird mehr als ein Baseballstadion geschnitten, so werden alle Schnittpunkte zusammengefasst. In den Abschnitten, die durch mehr als ein Stadion abgedeckt sind, wird der Durchschnitt der Integrale gebildet. Zuletzt wird dann die Summe der so berechneten Teilintegrale ausgegeben.

Die Integrale werden hier für eine Kante k mit Länge len und Gewicht w angegeben. Das Integral durch den Kreis muss in zwei Fälle unterschieden werden. Liegt der Kreismittelpunkt auf der Kante, so entspricht der Höhenverlauf im Kreis der Geraden $f(x) = mx = \frac{w}{r}x$. Das Integral ist entsprechend $F(x) = \frac{w}{2r}x^2$. Verläuft die Kante nicht durch den Punkt, so entspricht der Höhenverlauf der Hyperbel $\hat{f}(y) = a\sqrt{q + \frac{y^2}{b^2}}$. Der Parameter a entspricht dem Abstand von Gerade und Punkt. Der Parameter b lässt sich aus der Steigung m berechnen als $m = \frac{a}{b} \Leftrightarrow b = \frac{a}{m}$. Das Integral dieser Funktion $\hat{f}(y)$ ist $\hat{F}(y) = \frac{1}{2}a(y + \sqrt{\frac{y^2}{b^2} + 1} + b\sinh^{-1}(\frac{y}{b}))$.

Für den Fall der Seitenflächen wird der Höhenverlauf proportional zu der Funktion $f(t) = \frac{a+bt}{c+et}$ dargestellt. Dabei geben die Terme $a+bt$ und $c+et$ den Abstand zur Polygonzugkante an, im Falle von $a+bt$ ist dies der Abstand der Graphkante, im Falle von $c+et$ der Abstand der Begrenzungsgerade. Es gilt dabei für alle Punkte innerhalb des Stadions $a + bt \leq c + et$. Der Quotient $\frac{a+bt}{c+et}$ ist daher immer kleiner oder gleich eins. Das Integral dieser Funktion $f(t)$ ist $F(t) = \frac{(ae-bc)\log(c+et)+ebt}{e^2}$. Tritt der Fall $e = 0$ auf, so ist dieses Integral nicht wohldefiniert. In diesem Fall muss das Integral $\int \frac{a+bt}{c+0}\mathrm{d}t = \frac{(a*t)+\frac{b}{2}t^2}{c}$ gebildet werden. In einem validen Stadion kann nie e und c gleichzeitig null sein. Die Herleitungen für diese Konstruktion finden sich im Anhang unter A.

4. Algorithmus

Dieses Kapitel beschreibt den Aufbau des Algorithmus, der in der Arbeit zum Einsatz kommt. Dabei werden neben Abweichungen zum beschriebenen Algorithmus auch die tatsächliche Implementierung der mathematischen Berechnungen erklärt.

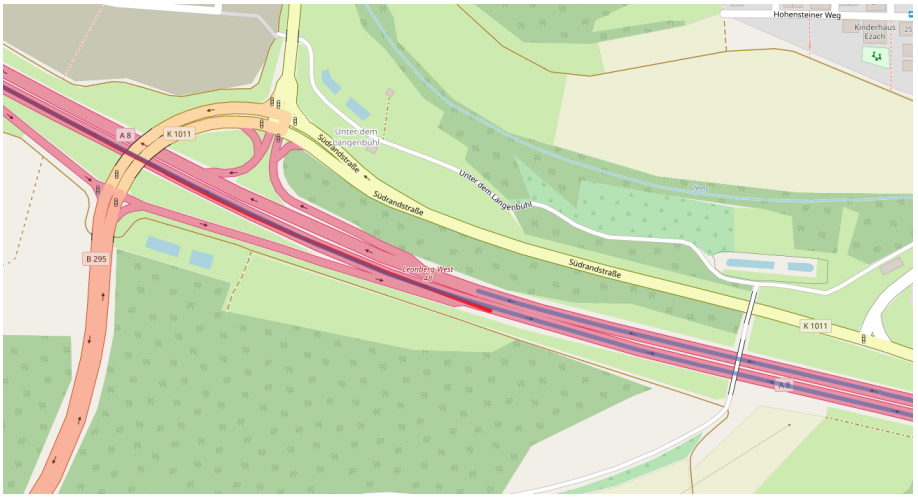
Der in dieser Arbeit evaluierte Ansatz nutzt Dijkstras Algorithmus [Dij59] zur Suche von kürzesten Wegen zwischen einem Startknoten und einem Zielknoten. Der prinzipielle Aufbau dieses Algorithmus ist wie folgt: Der Algorithmus beginnt bei dem Startknoten. Vom Startknoten aus werden dessen Nachbarn abgesucht und einer Kandidatenmenge hinzugefügt. Der nächste Knoten, also der Knoten mit dem kürzesten Pfad vom Startknoten, aus der Kandidatenmenge wird als besucht markiert. Dessen Nachbarn werden danach der Kandidatenmenge hinzugefügt. Findet sich ein Pfad zu einem Knoten in der Kandidatenmenge, der ein kleineres Gewicht vom Startknoten aus besitzt, so wird das Gewicht des Knotens verringert. Es wird abgebrochen, wenn der Zielknoten aus der Kandidatenmenge entfernt wird.

Die Suche von kürzesten Pfaden erfordert ein Kantengewicht für jede Kante, das zu einem Pfadgewicht aufsummiert werden kann. Dazu stellt ein Graph eine Funktion `getWeight` zur Verfügung, die einer Kante das Gewicht zuweist. Repräsentiert der Graph ein Straßennetzwerk, so gibt diese Funktion eine vorher abgespeicherte Fahrzeit für die Kante aus. Führt man auf einem solchen Graph Dijkstras Algorithmus aus, ist das Ergebnis entsprechend ein Weg kürzester Fahrzeit. In anderen Graphinstanzen sind auch andere Gewichte denkbar, zum Beispiel Kosten für das Traversieren der Kante.

Zur Lösung unseres Problems wird die `getWeight`-Methode so modifiziert, dass das Kantengewicht klein ist, wenn die Kante wahrscheinlich auf dem gesuchten Pfad liegt. Dazu wird das Kantengewicht wie in Abschnitt 3.2 angepasst. Eine praxisnäherere Implementierung wird in Algorithmus 4.2 gezeigt. In den Preliminaries, in Abschnitt 2.3 wird aufgeführt, dass konzeptionell allen Kanten ein Gewicht zugeordnet ist. Da wir im Algorithmus nur wenige Kanten betrachten, berechnet unsere Implementierung den Wert so spät wie möglich. Konkret ist dies der Zeitpunkt, an dem das Gewicht der Kante in der Implementierung abgefragt wird.

Wie oben beschrieben wurde der Algorithmus ursprünglich mit einem Start- und Zielknoten ausgeführt. Als Startknoten wurde der Knoten gewählt, der dem Startpunkt der Messreihe am nächsten ist. Als Zielknoten wurde entsprechend der Knoten gewählt, der am nächsten am Zielpunkt der Messreihe liegt. Durch die Diskretisierung des Graphen kann es passieren, dass der nächste Knoten gar nicht auf der Straße liegt, die in der Messreihe genommen wurde.

Abbildung 4.1.: Zielpunkt liegt auf der gegenüberliegenden Seite der Autobahn
Rot: der Polygonzug; Blau: der gefundene Pfad



In Abbildung 4.1 liegt der Zielknoten auf der gegenüberliegenden Seite einer Autobahn, in Abbildung 4.2 auf einer komplett falschen Straße. Dieses Problem lässt sich beheben. Statt fest den nächsten Knoten zu nehmen, wählt man einen naheliegenden Knoten aus einer Start- beziehungsweise Zielmenge. Aus diesem Grund wurde die unmodifizierte Version von Dijkstras Algorithmus erweitert, sodass von einer Startmenge in eine Zielmenge gesucht wird. Dieser Anpassung ist die Multisource-Multitarget-Version von Dijkstras Algorithmus. Diese Mengen sollen so initialisiert werden, dass jeweils mindestens ein Knoten auf dem erwarteten Pfad liegt. Dazu werden um Start- und Zielpunkt Knoten gewählt und diesen Mengen hinzugefügt.

In der Implementierung werden Kreise um den Start- beziehungsweise Endknoten der Polygonzüge gelegt. Der Radius dieser Kreise hängt vom Abstand zu den Nachbarn im Polygonzug ab. Dazu wird der Abstand mit einem konstanten Faktor c multipliziert, und die Kreise um Start- beziehungsweise Zielpunkt herum konstruiert. Die gefundenen Punkte werden mit ihrem Abstand gewichtet. Dieses Gewicht wird so angepasst, dass auf dem Kreis mit Radius c das zeitliche Gewicht genau dem zeitlichen Abstand zweier Punkte entspricht. Sollte sich in dem Umkreis kein Punkt finden lassen, so wird der nächste Nachbar angefragt und die Menge damit gefüllt. Da in diesem Fall nur ein Knoten gewählt werden kann, ist das Gewicht irrelevant und wird mit 0 initialisiert.

Der Pseudocode für die erweiterte Version von Dijkstras Algorithmus ist unter Algorithmus 4.1 zu finden. Der Algorithmus benötigt ein Kantengewicht, dieses wird in der Funktion `getWeight` berechnet. Diese Anpassung behebt die oben beschriebenen Probleme, wie die Abbildungen 4.3 und 4.4 zu entnehmen ist.

Wie in den Definitionen im Abschnitt 2.3ff beschrieben ist, werden die Messpunkte genutzt, um damit Integrale zu berechnen. Die Gewichtsbestimmung besteht daher aus mehreren Teilen. Zuerst werden die relevanten Messpunkte gesammelt und daraus die Baseballstadien konstruiert. Die Kante wird dann mit allen Stadien geschnitten. Für jeden gefundenen Abschnitt werden die Kantengewichte berechnet und alle Abschnitte kombiniert. Jeder dieser Schritte soll möglichst performant sein. Für die Messpunktsammlung wird daher ein Quadtree verwendet, der die Anfragen in logarithmischer Zeit bewältigt. Da der Quadtree nur für eine Anfrage in einem bestimmten Umkreis vorbereitet ist, wird der Graph vorberechnet. Die Kantenlänge ist danach hinreichend klein, sodass mit einem festen Suchradius alle relevanten Punkte gefunden werden können. Längere Kanten werden entsprechend in diesem Vorberechnungsschritt gekürzt. Der verwendete Graph ist also eine

Abbildung 4.2.: Zielpunkt liegt auf einer Autobahnbrücke
Rot: der Polygonzug; Blau: der gefundene Pfad

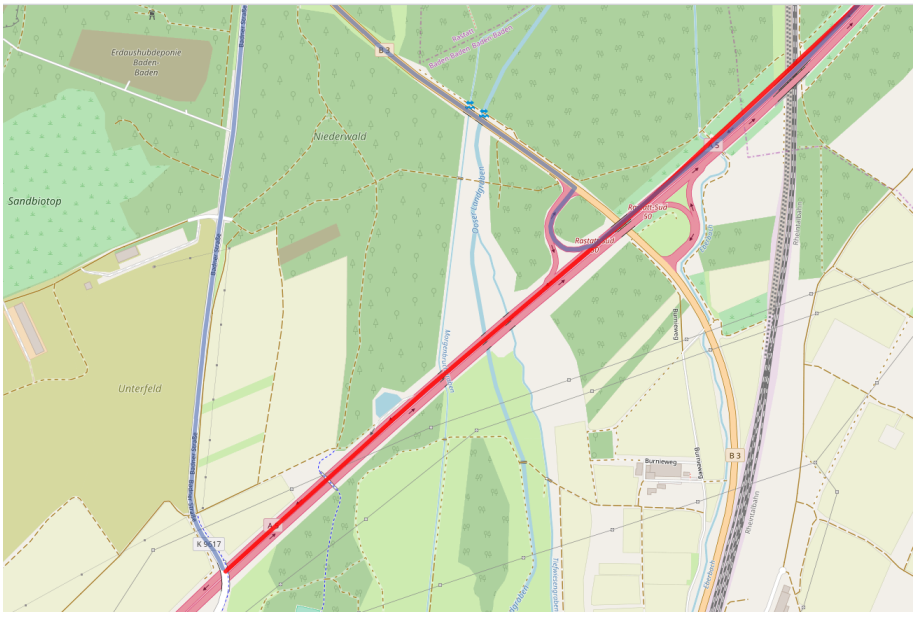
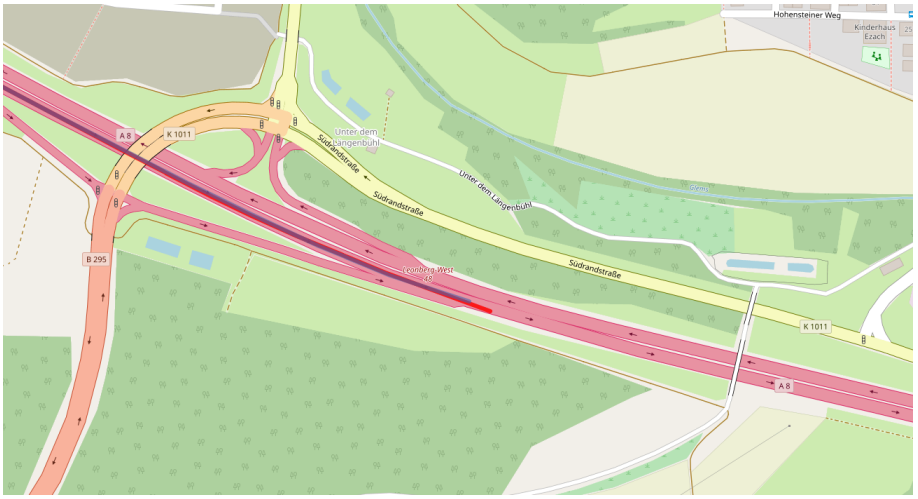


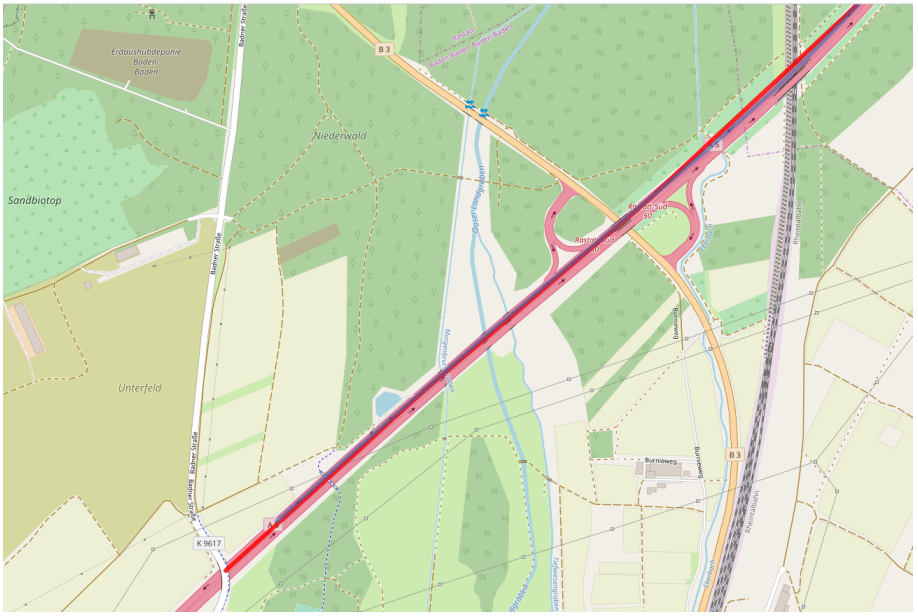
Abbildung 4.3.: Behobenes Ende von 4.1, mit dem Zielknoten auf der anderen Autobahnseite; Rot: der Polygonzug; Blau: der gefundene Pfad



Algorithm 4.1: Multisource-Multitarget-Dijkstra

```
   input  : Graph: Ein Graph auf dem Gerechnet wird
   input  : SourceMap: Knoten mit ihren Startgewichten
   input  : TargetMap: Knoten, die als Ziele gelten können, mit ihren Zielmali
   output : path: Pfad in Graph
 1 q ← PriorityQueue of [ℕ, ℕ] Paar: Gewicht für Sortierung, und der dazugehörige
     Vertex ;
 2 foreach vertex,weight ← SourceMap do
 3     Initalisierung mit bekannten Gewichten;
 4     q.insert(weight, vertex);
 5 while !q.empty() do
 6     weight, vertex ← q.pop() ;
 7     if TargetMap.contains(vertex) then
 8         ein mögliches Ziel gefunden ;
 9         target ← vertex;
10         break;
11     foreach neighbour ← Graph.neighbours(vertex) do
12         Falls Knoten noch nicht eingefügt, ist das Gewicht ∞;
13         if q.get(neighbour) > weight + getWeight(Graph, vertex,neighbour) +
            TargetMap.getOrElse(neighbour) then
14             neighbour.predecessor ← vertex;
15             setze das Gewicht des Nachbars (neu), aus Gewicht des Vorgängers,
                dem Kantengewicht, und einem möglichen Malus;
16             q.relax(neighbour, weight + getWeight(Graph,vertex,neighbour) +
                TargetMap.getOrElse(neighbour, 0)) ;
17 path ← List of Vertices ;
18 while !SourceMap.contains(target) do
19     path.insert(target) ;
20     target ← target.predecessor;
21 return path
```

Abbildung 4.4.: Behobenes Ende von 4.2, mit dem Zielknoten auf der Autobahnbrücke; Rot: der Polygonzug; Blau: der gefundene Pfad



lineare Unterteilung des ursprünglichen Graphen. Um die Verrechnung der Schnitttests zu beschleunigen, werden diese einer geordneten Menge (`std::set`) hinzugefügt, sodass die Punkte entlang der Kante durchiteriert werden können. Ist der Kantenabschnitt im Einflussbereich mehr als eines Baseballstadions, so wird das Gewicht aufsummiert und durch die Anzahl der Stadien geteilt. Damit geht der Einfluss anderer Stadien nicht verloren. Dieser Algorithmus ist in 4.2 beschrieben.

Algorithm 4.2: getWeight-Funktion

```
   input  : graph: Der Graph auf dem gerechnet wird
   input  : from: Der Startknoten
   input  : to: Der Zielknoten
   input  : tree: Quadtree mit den Messpunkten
   input  : maxrange: die Suchentfernung, die im Worstcase nötig ist
   output : Gesamtgewicht
 1 edge ← graph.getEdge(from, to) ;
 2 weight ← graph.getEdgeInfo(edge) ;
 3 berechne die Vereinigung der Punkte von Start und Ziel:
     points ← merge(tree.get(from, maxrange), tree.get(to, maxrange)) ;
 4 set ← geordnetes Set Schnittfaktoren zusammen mit einer Funktion, die das
     Integral berechnet ;
 5 foreach point ← points do
 6     stadion ← makeStadionAround(point) ;
 7     if edge.intersects(stadion.semicirle) then
 8         bei dem Schnitttest werden auch die Faktoren berechnet, die für das Integral
             benötigt werden. Diese sind in A.2 beschrieben.
 9         start, stop, factors ← edge.intersect(stadion.semicirle) ;
10         set.insert(start, stop, semicircleFunction(factors, weight)) ;
11     if edge.intersects(stadion.trapezoid) then
12         bei dem Schnitttest werden auch die Faktoren berechnet, die für das Integral
             benötigt werden. Diese sind in A.3 beschrieben.
13         start, stop, factors ← edge.intersect(stadion.trapezoid) ;
14         set.insert(start, stop, trapezoidFunction(factors, weight)) ;
15 weight ← 0 ;
16 foreach start, stop, function ← Interval in set, unterteilt bei Überlappungen do
17     val ← function(start, stop) / Anzahl der momentan betrachteten Intervalle ;
18     weight ← weight + val;
19 return weight ;
```

5. Experimentelle Evaluation

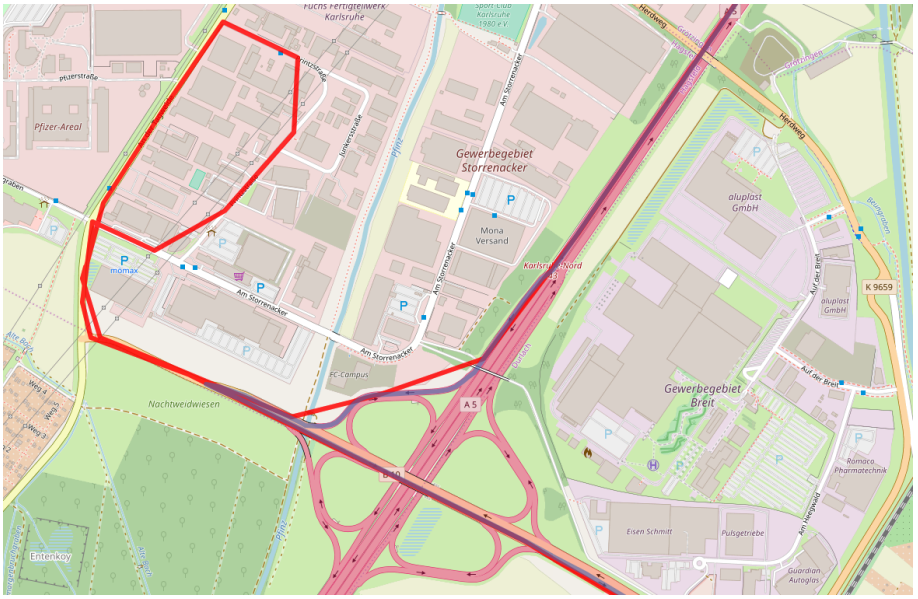
In diesem Kapitel stellen wir die experimentelle Evaluation unserer in Kapitel 3 beschriebenen Algorithmen vor. Es wurden sowohl der Ansatz mit Gausskurven in Abschnitt 3.1 als auch der Ansatz mit Kantenzuggewichten in Abschnitt 3.2 evaluiert. Der in Abschnitt 3.1.2 beschriebene Ansatz mit Gausskurven scheitert allerdings bereits theoretisch. Wir werden die Ergebnisse hier daher nicht weiter beschreiben.

5.1. Testmethodik

Um die Güte des Algorithmus zu bewerten, wird der Algorithmus gegen vorhandene Daten der Firma BMW getestet. Diese Daten wurden von der Firma für Forschungszwecke zur Verfügung gestellt und sind leider nicht öffentlich zugänglich. Es handelt sich um 6067 valide Messreihen mit zwischen 2 und 337 Punkten. Es sind mindestens 2 Punkte in der Messreihe nötig, um ein Baseballstadion aufzubauen, da die äußeren beiden Punkte durch Verlängerung der inneren Verbindungskante hergestellt werden können. Es werden also aus dem Datensatz alle Messreihen genutzt, deren Länge mindestens 2 beträgt. Zusätzlich sind in den Daten Messreihen, die keine örtliche Veränderung zeigen. Ist daher die Startmenge gleich der Zielmenge, wird die Messreihe übersprungen. Es sollen alle Caches, Branch Predictors und weitere Prozessorelemente auf den auszuführenden Code vorbereitet werden, bevor die eigentlichen Tests starten. Dazu werden vor den gewerteten Testläufen einige Tests ausgeführt, deren Ergebnisse verworfen werden. Die Routen werden dabei nach Länge sortiert und jede 2^k-te Route durchlaufen, um so auf lange und auf kurze Routen vorzubereiten. Um Ausreißer herausmitteln zu können, werden alle Testfälle fünfmal wiederholt.

Die Tests werden ausgeführt auf einer Maschine mit `Intel(R) Xeon(R) CPU E5-1630 v3`-CPU bei `3.70GHz`, einer Quadcore-CPU mit Haswell-Microarchitektur. Die Maschine ist ausgestattet mit `128GB` RAM. Das Betriebssystem ist `openSUSE Leap`, Version `15.0`, dem ein Linuxkernel der Version `4.12.14-lp150.12.19-default` zugrunde liegt. Der Code wird durch `clang++` in Version `7.0.0` gebaut, mit den Compileroptionen `-stdlib=libc++ -std=c++17 -O3 -march=haswell -ftree-vectorize -ffast-math`. Der verwendete Compiler benötigt das `-stdlib=libc++`-Flag, um `C++17-std::variant`s korrekt zu unterstützen. Da in der Integration viel mit Fließkommawerten gerechnet wird, wird zusätzlich das Flag `-ffast-math` aktiviert.

Abbildung 5.1.: Abzweigung von der Autobahn und später wieder darauf



Der zugrundeliegende Straßengraph ist eine OpenStreetMap[1]-Karte mit Stand vom 1.10.2018. Aus dieser Karte wurden die Straßendaten mit voller Auflösung extrahiert. Die von BMW gestellten Daten stammen aus dem Umkreis von Karlsruhe, insbesondere finden sich auch Punkte in Frankreich. Aus diesem Grund wurde eine Straßenkarte von ganz Europa zugrunde gelegt. Diese wurde bei dem Anbieter Geofabrik[2] von deren Downloadserver[3] heruntergeladen. Alle in dieser Arbeit gezeigten Karten, auch gezeigte Bilder, basieren auf OpenStreetMap-Daten. Für die Kartendaten gilt © OpenStreetMap contributors.

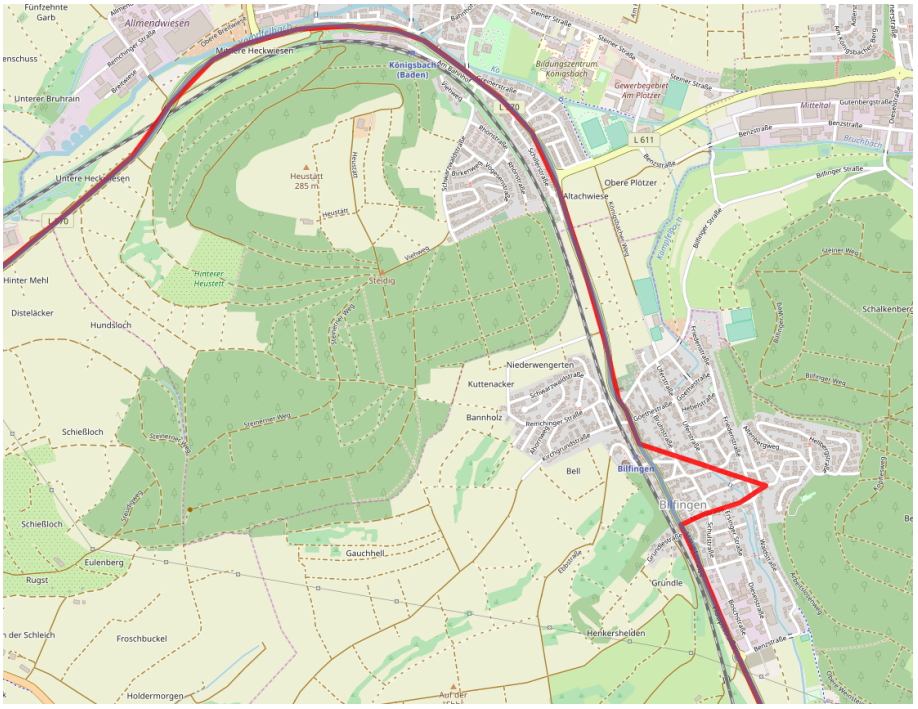
5.2. Limitierungen des Algorithmus

Im Verlauf der Evaluation sind einige Probleme aufgefallen, die durch den grundlegenden Ansatz bedingt sind. Der Algorithmus hat das Problem, keinen Schleifen folgen zu können, die über den gleichen Knoten zurück auf die zu folgende Route kommen. Dies tritt zum Beispiel auf, wenn Autos einen Abstecher zum Einkaufen oder zur Tankstelle machen, und lässt sich auch in Abbildung 5.1 in den Daten finden. Solche Schleifen haben ein positives Pfadgewicht. Der Pfad hat also ein kleineres Gewicht, wenn die Schleife nicht genommen wird. Ein Algorithmus, der kürzeste Pfade sucht, wird also keine Schleifen finden. Darüberhinaus ist in dem Algorithmus das Kantengewicht nur von der Position der Kante und der Messpunkte abhängig. Der kürzeste Pfad ist damit immer ohne die Schleife, denn das Gewicht der Schleife ist ≥ 0, die Schleife lohnt sich also nie.

Ein weiteres Problem des Algorithmus sind kurze Abweichungen von der Hauptrichtung, bei der nur ein Punkt nicht der Richtung des Algorithmus folgt, wie in Abbildung 5.2 Das Problem ist hier, dass die Baseballstadien einen großen Bereich der Messreihe ohne den Punkt auch abdecken, sodass auch dort das Gewicht gesenkt wird und sich das Folgen der Abweichung nicht lohnt. Mithilfe anderer Formen kann das Problem nur eingeschränkt behoben werden. Kurze Abzweigungen sind Kandidaten für Schleifen. Um von der Hauptrichtung abzuzweigen und später wieder darauf zu stoßen, wird wahrscheinlich eine Kreuzung genommen. In diesem Fall exisitiert eine Schleife, die an dieser Kreuzung startet. Gibt es einen anderen Pfad, so kann eine andere Form diese Einschränkung beheben.

[1] `https://www.openstreetmap.org/`
[2] `https://www.geofabrik.de/`
[3] `https://download.geofabrik.de/`

Abbildung 5.2.: Kurze Abweichung von der Hauptrichtung des Polygonzugs



5.3. Performance

Wir nehmen an, dass durch den Kanaleffekt die Anzahl der Queuepops im Vergleich zum normalen Dijkstra viel kleiner ist. Wir nehmen mit dieser Begründung auch an, dass die Laufzeit des Algorithmus deutlich kleiner ist als die des unmodifizierten Dijkstra. Wir vergleichen daher die Laufzeiten und Queuepops des Algorithmus mit den entsprechenden Werten eines unmodifizierten Dijkstra. Im Rahmen der Arbeit war es nicht möglich, den Code unseres Algorithmus bestmöglich zu optimieren. Die Laufzeit unseres Algorithmus kann also möglicherweise noch weiter verbessert werden.

Da es sich wie in Abschnitt 5.1 beschrieben um über 6000 Testfälle handelt, werden als Ergebnis hier nur statistische Überblicke gegeben. Die Testfälle sind absteigend nach Anzahl der Messpunkte in der Messreihe sortiert. Damit kann das Verhalten des Algorithmus bei langen und bei kurzen Messreihen verglichen werden. Es wurde in den Testläufen ein extremer Ausreißer gefunden. Die Laufzeiten beider sowohl unseres Algorithmus als auch des Referenzcodes sind mehr als eine Größenordnung größer sind als bei der nächstlangsamereren Messreihe. Es handelt sich um einen Abschnitt auf der Autobahn, bei dem die letzten Knoten nah beieinander liegen. In diesem Fall nutzt der Code den nächsten Nachbarn des Zielknotens. Dieser liegt auf einem Feldweg neben der Autobahn. Das Kantengewicht dieses Feldweges ist so groß, dass die Laufzeit des unmodifizierten Dijkstra bereits mehrere Sekunden beträgt. Diese Messreihe ist auch in Abbildung 5.3 zu sehen. Gut zu erkennen ist die Startmenge, die offensichtlich mehrere Startknoten enthält. Die Ergebnisse unseres Algorithmus und die der Multisource-Multitarget-Version von Dijkstras Algorithmus wählen verschiedene Startknoten.

In der ersten Tabelle 5.1 werden die Laufzeiten und die damit zusammenhängenden Queuepops gezeigt. Da die Laufzeiten und die Pops abhängig von der Länge des Polygonzugs stark schwanken, werden in der Tabelle die Werte logarithmisch dargestellt. Dabei wurden die Werte des Algorithmus durch die des Dijkstra geteilt, und dann der Logarithmus zur Basis 10 gebildet. Negative Werte sind also ein Vorteil für den Algorithmus, positive für den Dijkstra.

Abbildung 5.3.: Das Bild zeigt den Trace mit der schlechtesten Laufzeit. Der gefundene Zielknoten liegt auf einem Feldweg, in Rot zu sehen ist der eigentliche Trace auf der Autobahn, in Blau der gematchte Pfad, der nur auf Feldwegen ist, in Grün sieht man den normalen Dijkstra.

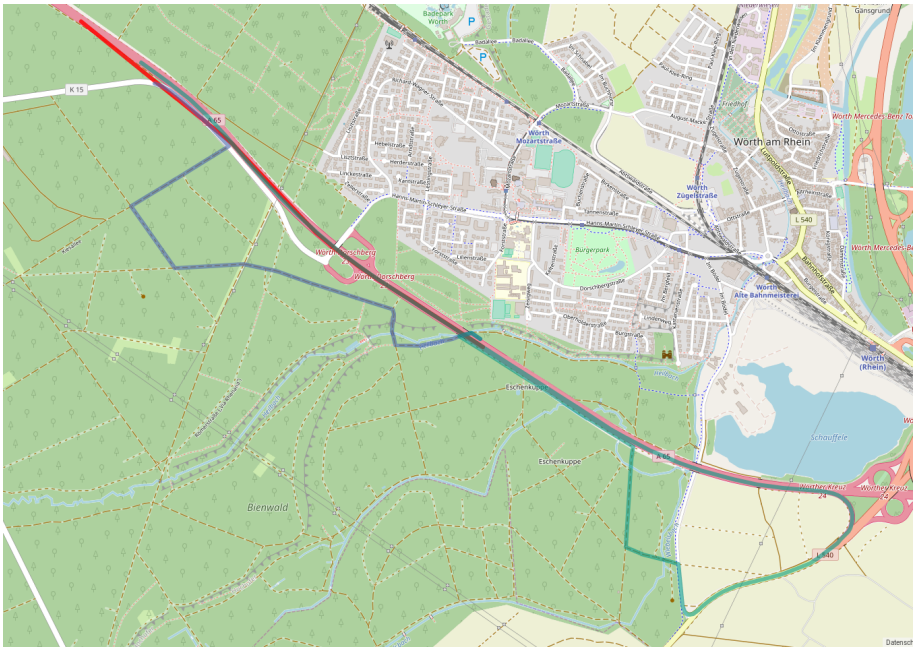


Tabelle 5.1.: Vergleich der Laufzeiten und Pops. Da die Messwerte sehr verschieden groß sind, wird der Quotient $\frac{\text{Wert Algorithmus}}{\text{Wert Dijkstras Algorithmus}}$ gebildet und davon der Logarithmus gebildet. Bei negativen angegebenen Zahlen hat also unser Algorithmus die kleineren Werte. Die angegebenen Zahlen sind aus der Menge der so berechneten Logarithmen entstanden. Gemessen wurde die Laufzeit t und die Anzahl der Queuepops p.

	Min	Max	Durchschnitt	Median	Standardabweichung
$\log(t)$	-2.23	1.18	-0.02	0	0.31
$\log(p)$	-3.13	1.60	-0.74	-0.67	0.55

Tabelle 5.2 zeigt einen Ausschnitt der Messreihen, in dem sowohl lange als auch kurze Reihen vertreten sind. Dazu wurden von den validen Traces, von 1 beginnend, immer die nächste Zweierpotenz genommen. Anhand dieser Tabelle lässt sich das Verhalten des Algorithmus in Bezug auf lange und kurze Messreihen abschätzen.

In den Tabellen 5.1 und 5.2 lässt sich erkennen, dass der Algorithmus durchschnittlich keine viel geringere Laufzeit als ein normaler Dijkstra hat. Dazu im Gegensatz steht die Anzahl der der Queuepops. Auch bei den extrem kurzen Messreihen, die man am Ende von Tabelle 5.2 sieht, wird immer noch ein deutlicher Speedup erreicht, und bei den längeren Traces sind alle Faktoren zwischen anderthalb und zweieinhalb Zehnerpotenzen. Der Algorithmus benötigt also deutlich weniger queuebezogene Operationen.

Bei den Minima und Maxima in Tabelle 5.1 fällt auf, dass es sowohl bei der Zeit, als auch bei den Queuepops starke Ausreißer in beide Richtungen gibt. Das Minimum bei der Zeit, genau wie bei den Queuepops, stellt Bestcaseszenarien für den Algorithmus dar. Das Maximum bei der Zeit hingegen stellt ein Bestcaseszenario für den Dijkstra. Bei vielen Messreihen gibt es den Fall, dass Anfang und Ende der Messreihe auf einer Autobahn liegen. In diesem Fall ist auch für eine unmodifizierte Multisource-Multitarget-Version von

Tabelle 5.2.: Performance in Abhängigkeit von der Länge der Messreihe. Dabei bezeichnet „Algo.“ unseren Algorithmus und „Dijkstra“ eine Multisource-Multitarget-Version von Dijkstras Algorithmus mit dem unveränderten Kantengewicht des Graphen

	Zeit(ms)			#Pops		
#Punkte	Algo.	Dijkstra	Speedup	Algo.	Dijkstra	Speedup
337	89	385	4.33	12486	1221335	97.82
336	78	316	4.05	9675	1028613	106.32
323	126	323	2.56	15763	1037250	65.80
290	27	445	16.48	5333	1340131	251.29
272	120	307	2.56	17369	1007579	58.01
232	17	276	16.24	2962	925318	312.40
201	34	79	2.32	5427	312662	57.61
167	46	182	3.96	9457	623697	65.95
135	11	151	13.73	3418	547930	160.31
93	14	29	2.07	3680	132040	35.88
57	2	26	13.00	805	118261	146.91
27	1	0	0.00	386	3700	9.59
9	0	0	0.00	55	82	1.49

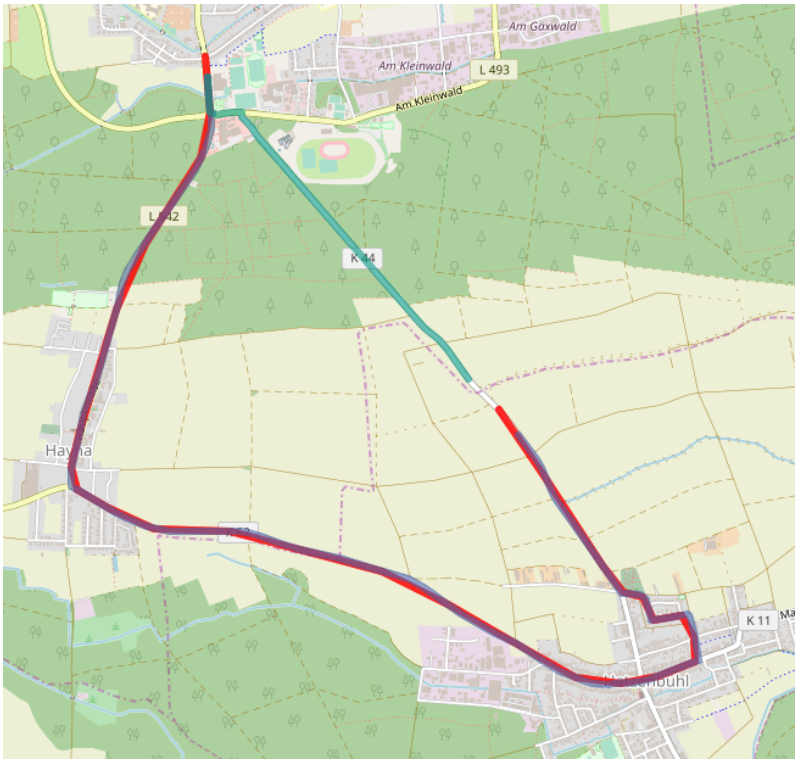
Dijkstras Algorithmus das benötigte Gewicht zum Zielknoten und daher die benötigte Laufzeit klein. In diesem Fall zeigt sich der Overhead unseres Algorithmus. Interessant ist der Fall, in dem unser Algorithmus mehr als eine Zehnerpotenz mehr Elemente aus der Priorityqueue nimmt als der Dijkstra. In diesem Fall verläuft die Messreihe in einer Form, dass unser Algorithmus durch die geringeren Gewichte anfängt zu suchen, der normale Dijkstra aber eine Alternativroute nehmen kann. Abbildung 5.4 zeigt ein solches Beispiel.

Anhand dieser Zahlen lässt sich folgern, dass die erwartete Einschränkung auf den Kanal funktioniert. Der Suchraum des Algorithmus ist, gerade bei längeren Messreihen, vor allem in dem Kanal entlang der Messroute. Allerdings ist die Diskrepanz zwischen zeitlichem Speedup und Queuepops sehr groß. Während der unmodifizierte Dijkstra nicht viel mehr als einen RAM-Zugriff durchführen muss, muss unser Algorithmus zur Kantenrelaxierung sowohl einen Schnittest durchführen als auch ein Integral berechnen. Die Berechnung dieser Werte bremst den Algorithmus. An dieser Stelle sei nochmals erwähnt, dass es im Rahmen dieser Arbeit leider nicht möglich war, die Berechnung durchzuoptimieren. Die starken Faktoren bei den Queuepops lassen aber hoffen, dass hier mit Optimierung noch viel verbessert werden kann.

5.4. Genauigkeit

Die Genauigkeit der Lösung soll durch den Vergleich des Polygonzugs mit dem gefundenen Pfad evaluiert werden. Dazu wird die diskrete Fréchet-Distanz zwischen Pfad und Polygonzug berechnet. Die Fréchet-Distanz ist ein Maß für den Abstand zweier Polygonzüge. Die diskrete Fréchet-Distanz bildet eine obere Schranke für die kontinuierliche Fréchet-Distanz, die sich leichter berechnen lässt. Der Nachteil dieser Lösung ist, dass der Fehler zur Fréchet-Distanz mit der Kantenlänge wächst. Dies ist bei den einzelnen Kanten des Polygonzuges ein nicht komplett zu vernachlässigender Faktor. Die diskrete Fréchet-Distanz hat Fehler in der Größenordnung der Kantenlänge. Allerdings ist der Abstand verschiedener plausibler Straßen recht groß. Trotz des Fehlers eignet sich die Distanz also, um die Plausibilität der Zahlen sicherzustellen.

Abbildung 5.4.: Der Algorithmus hat mehr Queuepops als die Multisource-Multitarget-Version von Dijkstras Algorithmus, da der Algorithmus ein valides Matching produziert.



Wir berechnen die diskreten Fréchet-Distanzen aller Polygonzüge zu unseren Pfaden, und präsentieren statistische Zahlen. Die Zahlen finden sich in Tabelle 5.3. Zusätzlich präsentieren wir die Ergebnisse der Pfade, die in Abschnitt 5.3 bereits als Beispiele genutzt wurden. Anhand dieser in Tabelle 5.4 gezeigten Daten lassen sich Rückschlüsse auf den Fehler anhängig von der Länge ziehen.

Tabelle 5.3.: Statistik der diskreten Fréchet-Distanz zwischen den Pfaden und dem Polygonzug

	Min	Max	Durchschnitt	Median	Standardabweichung
Fréchet-Distanz (in m)	0.00	11294.84	413.38	290.12	539.37

Tabelle 5.4.: Die diskreten Fréchet-Distanzen einiger Pfade

Route	1	2	4	8	16	32	64	128	256	512	1024	2048	4096
Fréchet-Distanz (in m)	1015	632	639	682	874	297	346	520	265	444	196	91	304

In Tabelle 5.3 lässt sich erkennen, dass der Durchschnitt der Werte in der Größenordnung einiger hundert Meter liegt. Der durch die Kantenlänge induzierte Fehler liegt bereits in dieser Größenordnung, daher ist dieser Wert plausibel. Spannend ist das Maximum, welches einige Kilometer beträgt. Diese Ausreißer entstehen durch Verläufe, die im Sinne unseres Algorithmus nicht valide sind. Ein Beispiel sind dafür sind Schleifen, wie in Abbildung 5.1 zu sehen. Bei Betrachtung der Polygonzüge in Tabelle 5.4 fällt auf, dass die Fehler recht zufällig verteilt sind. Dies lässt darauf schließen, dass die Evaluation mithilfe der diskreten Fréchet-Distanz in allen Pfaden die gleichen Probleme hat. Trotzdem sind, wie am Median zu erkennen, die meisten Pfade nah an den ursprünglichen Polygonzügen.

Der Konkurrenzansatz, mit dem unsere Lösung im Folgenden verglichen wird, bestätigt diese These weiter.

5.5. Vergleich mit anderer Lösung

Die Genauigkeit unseres Algorithmus soll mit dem in [LZZ+09] beschriebenen Ansatz verglichen werden. BMW hat dazu gematche Punkte zu den Polygonzugknoten bereits mit den Testdaten geliefert. Die komplette Route ist leider nicht Teil der Daten. Leider sind in den Daten neben validen Matches auch einige Messreihen, für die Punkte nicht gematched wurden. In diesem Fall ist ein Vergleich nicht möglich. Wir vergleichen daher die Messreihen, bei denen sowohl BMW als auch wir ein Matching berechnen konnten.

Wir berechnen für jeden Knoten des Polygonzuges den nächsten Punkt auf dem Pfad. BMW hat mit dem gleichen Verfahren Punkte auf dem Pfad berechnet. Wir berechnen dann den Durchschnitt des euklidischen Abstandes der von BMS und von uns berechneten Punkte. Wie bereits in den vorherigen Abschnitten werden wir hier zwei Tabellen präsentieren. In der einen Tabelle 5.5 werden Statistiken präsentiert. In der anderen, Tabelle 5.6 finden sich die Angaben einiger Pfade. BMW konnte leider nicht zu jedem Polygonzugknoten einen gematchten Punkt liefern. Die kompletten Polygonzüge, für dies gilt, wurden daher für dieses Kapitel als invalide verworfen. Wir sortieren daher wieder validen Pfade nach der Länge und geben jeden 2^k-ten Pfad an.

Tabelle 5.5.: Statistik über die typischen Abstände zwischen BMW und unserem Algorithmus

	Min	Max	Durchschnitt	Median	Standardabweichung
Distanz (in m):	0.00	30.32	5.10	3.00	4.55

Tabelle 5.6.: Die typischen Abstände ausgewählter, valider Pfade

Route	1	2	4	8	16	32	64	128	256	512
Distanz (in m)	1.34	1.45	5.04	1.42	1.13	2.12	1.72	2.17	2.23	6.29

Die in diesen Tabellen dargestellten Ergebnisse zeigen, dass ein Großteil der Pfade im Durchschnitt sehr nah an den Lösungen von BMW liegt. Mittelt man über den ganzen validen Pfad, verschwinden auch die Abweichungen unseres Algorithmus. Dieser Tabelle ist zu entnehmen, dass im Durchschnitt die von uns gefundenen Pfade denen von BMW entsprechen. Da die meisten der Messreihen von BMW als invalide nicht behandelt wurden, sind allerdings auch einige der für unseren Algorithmus problematischen Routen nicht in der geprüften Menge.

Unser Algorithmus nutzt eine Multisource-Multitarget-Version von Dijkstras Algorithmus. Am Anfang und Ende des Polygonzugs entstehen daher verhältnismäßig große Abstände. Die Start- beziehungsweise Zielknoten des gefundenen Pfades können mehrere hundert Meter von den Start- beziehungsweise Zielknoten des Polygonzugs entfernt sein. Der Einfluss dieser Punkte auf die Route soll betrachtet werden. Dazu wurden die Abstände aller Punkte gespeichert und notiert, ob es sich um einen Randpunkt handelt. Die Tabelle 5.7 zeigt den Einfluss innerer und äußere Punkte. Die Abstände im Inneren stimmen auf wenige Meter überein. Die Analyse der Tabelle 5.5 mit den Pfaddurchschnitten ist daher für die Punkte bestätigt. Die Fehler am Rand liegen im Bereich der erwarteten hunderen von Metern. Vergleicht man die in Tabelle 5.5 gebenen Durchschnitte der Pfade mit den den Durchschnitten der Punkte, erkennt man den Einfluss der äußeren Punkte.

Bei der Betrachtung der Maxima fallen die großen Abstände auf dem Pfad auf, die bei den Maxima der Pfade verschwinden. Wir vermuten, dass dies mit Pfaden zusammenhängt,

Tabelle 5.7.: Statistik über die typischen Abstände einzelner Punkte. Dies wird aufgeteilt auf alle Punkte, Punkte im Inneren des Polygonzuges und nur die Randpunkte des Polygonzuges.

	Min	Max	Durchschnitt	Median	Standardabweichung
Distanz aller Punkte (in m):	0.00	2278.67	9.82	1.16	61.39
Distanz innerer Punkte (in m):	0.00	2070.41	2.91	1.12	33.32
Distanz äußerer Punkte (in m):	0.02	2278.67	204.42	142.95	198.74

die im Sinne unserer Definition nicht valide sind. Ein Beispiel lässt sich in Abbildung 5.1 finden.

6. Zusammenfassung

Das Ziel der Arbeit war eine effiziente Lösung des Mapmatching-Problems zu finden. Diese sollte zusätzlich invariant unter linearer Unterteilung des Graphen sein, also bei Kantenunterteilungen mit neuen Knoten das gleiche Ergebnis liefern. Um dies sicherzustellen wurde auf Wegintegrale durch räumliche Funktionen zurückgegriffen. Die Forderung nach Invarianz unter Unterteilung stellte dabei sicher, dass Graphen in der Berechnung weiter unterteilt werden können. Die Begründung für die Integrale war, die Unterteilungen mathematisch elegant umsetzen.

In der Arbeit wurden zwei Familien räumlicher Funktionen betrachtet, durch die integriert werden kann. Der erste Ansatz basierte auf zweidimensionalen Gaußglocken, die als stochastisches Rauschen um die Messpunkte herum orientiert waren. Bei dem Versuch diese Glocken zu implementieren, wurden mathematische Probleme mit der Verrechnung mehrerer Glocken gefunden. Damit wurde dieser Ansatz verworfen und stattdessen auf geometrische Formen zurückgegriffen. Es wurde die Forderung aufgestellt, dass die Verbindungskanten des Polygonzugs ein minimales Gewicht haben sollen. Um diese Kante wurde eine Form konstruiert. Diese hängt nicht mehr nur von einem Punkt ab sondern zusätzlich von seinen Nachbarn. Dieser zweite Ansatz war für viele Fälle erfolgreich.

Die experimentelle Evaluation des Algorithmus hat zu kleineren Anpassungen des Ansatzes geführt. Der ursprüngliche Ansatz, von genau einem Startknoten zu genau einem Zielknoten zu rechnen, wurde verworfen. Diese Knoten liegen nicht notwendigerweise auf der gesuchten Route. Stattdessen wurde eine Erweiterung von Dijkstras Algorithmus mit mehreren Startknoten und mehreren Zielknoten genutzt. Die andere Anpassung betraf den zugrundeliegenden Graphen. Bei der Berechnung des Kantengewichts müssen alle im Umkreis liegenden Punkte betrachtet werden, und diese sollen mit einem möglichst kleinen Suchradius in einem Quadtree gesucht werden. In einer ersten Iteration wurde daher bei Kanten, die diese Länge überschritten, über die Kante linear iteriert. Es wurde daher der Graph so angepasst, dass zu lange Kanten unterteilt wurden.

Im Zuge der Evaluation wurden zwei designbedingten Limitierungen aufgetan. Das eine Problem tritt bei Messreihen auf in denen Zykel vorkommen. Da ein Dijkstra genutzt wird kann dieser nur zykelfreie Pfade finden. Ein in dieser Richtung unmodifizierter Dijkstra kann in solchen Fällen also keine korrekten Ergebnisse liefern. Ein verwandtes Problem tritt durch die gewählte geometrische Form auf. Bei sehr kurzen deutlichen Abweichungen von Messpunkten von der umgebenden Trajektorie wird durch die Konstruktion das direkte Folgen der Trajektorie lukrativ.

Bei der Betrachtung der Performance wurde festgestellt, dass unser Algorithmus viel weniger Queuepops als Dijkstras Algorithmus benötigt. Leider folgte daraus nicht, dass die Laufzeit im gleichen Maße steigt. Es stellte sich heraus, dass die Laufzeit unseres Algorithmus durch die nötigen Berechnungen beschränkt ist. Für jede Kante, die abgesucht wird, müssen Schnitttests und Integrale berechnet werden. Die Genauigkeit wurde durch die Berechnung eines Abstandsmaßes und den Vergleich mit einem Konkurrenzalgorithmus [LZZ+09] evaluiert. Das Abstandsmaß stellte sich als nicht sehr gut geeignet heraus, bekräftigte aber innerhalb der Einschränkungen das Ergebnis. Der Vergleich mit [LZZ+09] lieferte deutlich bessere Ergebnisse, und bestätigte unser Matching.

Ausgehend von den Ergebnissen dieser Arbeit kann in mehrere Richtungen weitergeforscht werden. Eine offensichtliche Richtung ist das Beheben der vorhandenen Mängel des Algorithmus. Dabei ist insbesondere die Frage interessant, ob der Algorithmus so angepasst werden kann, dass er Zykel matchen kann.

Eine andere Fragestellung ist, ob der Algorithmus durch eine andere Wahl der geometrischen Form verbessert werden kann. Ein Integrieren durch das Baseballstadion erfordert Schnitttests mit zwei Kreisen und fünf Geraden und erzeugt im schlimmsten Fall vier verschiedene zu integrierende Funktionen. Dabei kann betrachtet werden, ob andere Strukturen bei gleicher Genauigkeit eine höhere Performance oder bei nicht niedrigerer Performance eine höhere Genauigkeit bieten können.

Literaturverzeichnis

[AG95] Helmut Alt und Michael Godau: *Computing the Fréchet distance between two polygonal curves.* Transportation Research Part C: Emerging Technologies, 5(1&2):75 – 91, 1995, ISSN 1793-6357. `https://www.worldscientific.com/doi/abs/10.1142/S0218195995000064`.

[Dij59] Edsger W. Dijkstra: *A Note on Two Problems in Connexion with Graphs.* Numerische Mathematik, 1:269–271, 1959.

[DoD08] United States of America Department of Defense: *GLOBAL POSITIONING SYSTEM STANDARD POSITIONING SERVICE PERFORMANCE STANDARD*, September 2008. `https://www.gps.gov/technical/ps/2008-SPS-performance-standard.pdf`.

[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, Seiten 352–361, New York, NY, USA, 2009. ACM, ISBN 978-1-60558-649-6. `http://doi.acm.org/10.1145/1653771.1653820`.

[QON07] Mohammed A. Quddus, Washington Y. Ochieng und Robert B. Noland: *Current map-matching algorithms for transport applications: State-of-the art and future research directions.* Transportation Research Part C: Emerging Technologies, 15(5):312 – 328, 2007, ISSN 0968-090X. `http://www.sciencedirect.com/science/article/pii/S0968090X07000265`.

[SWP06] R. Salas, C. Wenk und D. Pfoser: *Addressing the Need for Map-Matching Speed: Localizing Global Curve-Matching Algorithms.* In: *18th International Conference on Scientific and Statistical Database Management(SSDBM)*, Band 00, Seiten 379–388, Juli 2006. `doi.ieeecomputersociety.org/10.1109/SSDBM.2006.11`.

[YW04] Huabei Yin und O. Wolfson: *A weight-based map matching method in moving objects databases.* In: *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, Seiten 437–438, June 2004.

Anhang

A. Formeln

Der Kern der Arbeit ist die Beschreibung des Algorithmus, die mathematischen Grundlagen dazu sind in den Anhang ausgelagert. Beschrieben werden, die Schnitttests mit dem Baseballstadion und die Integrale durch den Halbkreis und die Seitenflächen.

A.1. Schnitttests

Es müssen sieben Schnitttests ausgeführt werden, um die Integrationsparameter zu bestimmen. Zum einen müssen die Halbkreise geschnitten werden, zum anderen die Seitengeraden. Da die Schnitttests in der zweidimensionalen Ebene ausgeführt werden, gibt es bei den Kreisen nur die Möglichkeiten:

1. Die Gerade schneidet den Kreis gar nicht $\Rightarrow$ 0 Schnittpunkte
2. Die Gerade tangiert den Kreis $\Rightarrow$ 1 doppelter Schnittpunkt
3. Die Gerade schneidet den Kreis echt $\Rightarrow$ 2 disjunkte Schnittpunkte

Die Fälle 2 und 3 können zusammengefasst werden, indem der doppelte Schnittpunkt als zwei Schnittpunkte betrachtet wird. Der eigentliche Schnitttest besteht aus zwei Schritten. Zuerst wird überprüft, ob ein Schnitt existiert. Danach wird die Position der Punkte berechnet. Dazu wird zuerst die Gerade so verschoben, dass der Ursprung des Koordinatensystems auf dem Mittelpunkt des Kreises liegt, danach wird überprüft, ob es Schnittpunkte gibt. Ist dies der Fall, werden die Schnittpunkte bestimmt und die Koordinatentransformation wird rückgängig gemacht. Dies ist als Pseudocode in 6.1 dargestellt.

Bei den Geraden gibt es im zweidimensionalen wieder drei Fälle der Anzahl der Schnittpunkte:

1. Die Geraden sind identisch, liegen also aufeinander. In diesem Fall gibt es unendlich viele Schnittpunkte.
2. Die Geraden sind parallel, aber nicht identisch. In diesem Fall gibt es keine Schnittpunkte
3. Die Geraden schneiden sich in genau einem Punkt.

Im Gegensatz zu den Kreisen ist es in diesem Fall nicht möglich Fälle zusammenzulegen. In dem Code wird zuerst der Fall abgeprüft, ob die Geraden parallel sind, also die Fälle 1 oder 2 eintreten. Ansonsten wird der einzige Schnittpunkt ausgerechnet. Dies wird im Pseudocode 6.2 dargestellt.

Algorithm 6.1: Schnitttest zwischen Kreis und Linie

input : $\mathsf{circ} \coloneqq (\mathsf{center}, \mathsf{r})$ ein Kreis mit Mittelpunkt center und Radius r
input : $\mathsf{edge} \coloneqq (\mathsf{head}, \mathsf{tail})$ eine Gerade mit Ursprung head und Ziel tail
output : false wenn kein Schnittpunkt, $(\mathsf{p}_1, \mathsf{p}_2)$ als Schnittpunkte sonst

1 *Punkte und Vektoren bestehen immer aus* (x, y)*-Paaren* ;
2 $\mathsf{edge} \leftarrow (\mathsf{edge.head} - \mathsf{center}, \mathsf{edge.tail} - \mathsf{center})$;
3 $\mathsf{dir} \leftarrow \mathsf{edge.head} - \mathsf{edge.tail}$;
4 $\mathsf{det} \leftarrow \mathsf{edge.head.x} \cdot \mathsf{edge.tail.y} - \mathsf{edge.head.y} \cdot \mathsf{edge.tail.x}$;
5 $\mathsf{discriminant} \leftarrow \mathsf{r}^2 \cdot \|\mathsf{dir}\|^2 - \mathsf{det}^2$ *Wert zur Überprüfung des Schnittpunkts* ;
6 **if** $\mathsf{discriminant} < 0$ **then**
7 **return** false ;
8 $\mathsf{root} \leftarrow \sqrt{\mathsf{discriminant}}$;
9 $\mathsf{edgeCenter} \leftarrow (\mathsf{det} \cdot \mathsf{dir.y}, -\mathsf{det} \cdot \mathsf{dir.x})$;
10 $\mathsf{offset} \leftarrow (\mathrm{sgn}(\mathsf{dir.y}) \cdot \mathsf{dir.x} \cdot \mathsf{root}, \mathsf{dir.y} \cdot \mathsf{root})$ *im Falle einer Tangente gilt* $\mathsf{offset} = (0, 0)$;
11 $\mathsf{p}_1 \leftarrow \frac{\mathsf{edgeCenter} - \mathsf{offset}}{\|\mathsf{dir}\|^2} + \mathsf{center}$;
12 $\mathsf{p}_2 \leftarrow \frac{\mathsf{edgeCenter} + \mathsf{offset}}{\|\mathsf{dir}\|^2} + \mathsf{center}$;
13 **return** $(\mathsf{p}_1, \mathsf{p}_2)$;

Algorithm 6.2: Schnitttest zwischen zwei Linien

input : $\mathsf{edge}_1 \coloneqq (\mathsf{head}_1, \mathsf{tail}_1)$ Eine Kante mit Ursprung head_1 und Ziel tail_1
input : $\mathsf{edge}_2 \coloneqq (\mathsf{head}_2, \mathsf{tail}_2)$ Andere Kante mit Ursprung head_2 und Ziel tail_2
output : false wenn kein Schnittpunkt gefunden, identical wenn gleich, p als Schnittpunkt

1 $\mathsf{delta} \leftarrow \mathsf{head}_1 - \mathsf{head}_2$;
2 $\mathsf{dir}_1 \leftarrow \mathsf{tail}_1 - \mathsf{head}_1$;
3 $\mathsf{dir}_2 \leftarrow \mathsf{tail}_2 - \mathsf{head}_2$;
4 $\mathsf{det} \leftarrow \mathsf{dir}_1\mathsf{.x} \cdot \mathsf{dir}_2\mathsf{.y} - \mathsf{dir}_1\mathsf{.y} \cdot \mathsf{dir}_2\mathsf{.x}$;
5 **if** $\mathsf{det} = 0$ **then**
6 *paralleler Fall* ; **if** $\mathsf{edge}_1 = \mathsf{edge}_2$ **then**
7 **return** identical ;
8 **return** false ;
9 $\mathsf{scalar} \leftarrow <(-\mathsf{dir.y}, \mathsf{dir.x}), \mathsf{delta}>$ *Skalarprodukt* ;
10 **return** $\mathsf{edge}_1 + \mathsf{scalar} \cdot \mathsf{dir}_1$

A.2. Integral durch den Halbkreis

Nimmt man den Mittelpunkt eines Kreises und interpoliert die Höhe von diesem aus zum Rand, so erhält man einen Kegel. Um daraus den Höhenverlauf einer Geraden zu erhalten, betrachtet man einen Kegelschnitt. Die Begründung dafür ist, dass die Gerade in der $\mathbb{R}^2$-Ebene liegt, die Höhe ist senkrecht zu der $\mathbb{R}^2$-Ebene. Die Höhe und die Gerade spannen daher eine neue Ebene auf, die den Kegel schneidet, und die per Konstruktion senkrecht zur $\mathbb{R}^2$-Ebene steht. Es gibt sechs Arten in der eine Ebene einen Kegel schneiden kann, und zwar:

1. in einem einzelnen Punkt. Dies würde erfordern, dass die Ebene die Seiten nicht schneidet, was in unserem Fall nicht möglich ist, da die Ebene senkrecht steht.
2. in einer Parabel. Da für eine Parabel die Ebene die Seite des Kegelpunktes ändern müsste, ist auch dies in unserem Fall nicht möglich.
3. in einem Kreis. Auch in diesem Fall steht das im Widerspruch zu der Eigenschaft.
4. in einer einzelnen Gerade. In diesem Fall müsste die Ebene parallel zu einer Geraden im Kegel sein. Dies widerspricht per Konstruktion der Eigenschaft.
5. in einer Hyperbel. Dies ist tatsächlich möglich.
6. in zwei sich im Mittelpunkt schneidenden Geraden. Auch dieser Fall kann eintreten.

In Fall 6 schneidet die Ebene den Mittelpunkt des Kreises. Die Schnittgeradn verlaufen vom Mittelpunkt aus zum Rand des Kegels. Im Baseballstadion ist die geschnittene Gerade entweder identisch zum Trenner zwischen Halbkreis und Seitenflächen oder schneidet im Mittelpunkt den Trenner. Im ersten Fall müssen beide Geraden integriert werden, im letzten Fall nur eine. Die Geraden haben das Aussehen $f(x) = mx + b$ mit der Steigung m und einer Höhenverschiebung b. Die Höhenverschiebung kann auf 0 gesetzt werden, wenn man den Ursprung der Geraden auf den Mittelpunkt verschiebt. Die Steigung m kann man berechnen, indem man den Radius des Kreises r und das Gewicht außerhalb betrachtet, das kantenabhängig ist. Das Gewicht außerhalb ist $w = \frac{w_{\text{edge}}}{\text{len}}$, mit w_{edge} dem Kantengewicht und len der Länge der Kante. Die Steigung ist daher $m = \frac{w}{r}$. Das Wegintegral von x_1 zu x_2 bei passend verschobenen $x_{1,2}$ ist also $\int_{x_1}^{x_2} f(x)\mathrm{d}x = \frac{1}{2}m(x_2^2 - x_1^2)$. Dieses Integral wird im Betrag genommen, da es von der Richtung der Kante unabhängig sein soll.

Betrachtet man hingegen den Fall der Hyperbel in Punkt 5, wird das Integral komplizierter. Eine Hyperbel hat in der ersten Hauptlage die Form $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$, wobei sich die Hyperbel asymptotisch der Geraden $y = \pm\frac{b}{a}x$ annähert. Da die Gerade der oben beschriebenen Gerade entspricht, kann b aus a bestimmt werden. In unserem Fall entspricht y dabei dem Parameter auf der Geraden und x der Höhe. Daher gilt, dass die im vorherigen Abschnitt konstruierte Gerade f hier von y nach x abbilden muss. Aus diesem Grund ist $m = \frac{a}{b}$, a erhält man als den Abstand der Geraden zum Mittelpunkt. Man erhält also die Funktion, die die Gerade auf die Höhe abbildet, indem man die Hyperbel in eine Funktion $\hat{f} : y \mapsto x$ abbildet. Diese Umformung wird in Gleichung 6.1 beschrieben.

$$\begin{aligned} \frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \Leftrightarrow \frac{x^2}{a^2} = 1 + \frac{y^2}{b^2} \Leftrightarrow \\ x^2 = (1 + \frac{y^2}{b^2})a^2 \Leftrightarrow x = \hat{f}(y) = a\sqrt{1 + \frac{y^2}{b^2}} \end{aligned} \tag{6.1}$$

Computergestützt[1] wurde als das Integral dieser Funktion $\int \hat{f}(y)\mathrm{d}y = \hat{F}(y) = \frac{1}{2}a(y + \sqrt{\frac{y^2}{b^2} + 1} + b\sinh^{-1}(\frac{y}{b}))$ gefunden. Das Integral wurde händisch geprüft.

[1] `https://www.wolframalpha.com/input/?i=a*sqrt(1%2Bx%5E2%2Fb%5E2)+dx`

A.3. Integral durch die Seitenflächen

Im Baseballstadion, siehe Abbildung 3.3, gibt es neben den Halbkreisen am Ende auch das Trapez in der Mitte. Dieses ist unterteilt durch die Kante des Polygonzugs in der Mitte, zu den Halbkreisen abgetrennt durch Schnittsekanten und nach außen durch Begrenzungsgerade genannte Linien. Da an der Polygonzugkante m in der Mitte eine Unstetigkeit in der Ableitung ist, werden beide Seiten einzeln integriert. Dafür wird die Kante, über die zu integrieren ist, so transformiert, dass sie über den Abstand zur Polygonzugkante in der Mitte dargestellt wird. Wir nennen diese Kante im Folgenden Integralkante k_I. Sowohl die Kante des Polygonzugs als auch die Integralkante sind Geraden. Die Abstandsfunktion ist also wieder eine Gerade, hat also die Form Abstand $= y = a + bt$, mit einem initialen Abstand a und einer Veränderung b. Der Laufparameter t wird so skaliert, dass er für das Wegintegral genau von 0 bis 1 läuft. Dies hat den Vorteil, dass a der Abstand von Polygonzugkante und Ursprung der Integralkante ist und b der Abstand von Polygonzugkante und Ziel der Integralkante abzüglich a. Es gilt also $a = \text{dist}(m, k_I(0)), b = \text{dist}(m, k_I(1)) - a$. Mit dem gleichen Verfahren wird die Begrenzungsgerade dargestellt. Es werden die Punkte berechnet, die in der Verlängerung von Ursprung und Ziel der Integralkante senkrecht zur Kante des Polygonzugs stehen. Diese nennen wir (p_1, p_2), und berechnen Parameter c und e als $c = \text{dist}(m, p_1)$ beziehungsweise $e = \text{dist}(m, p_2) - \text{dist}(m, p_1) = \text{dist}(m, p_2) - c$. Dies ergibt die zweite Abstandsgerade $y = c + et$. Die Höhe, normiert auf das Interval $[0, 1]$ lässt sich für jeden Integralkantenpunkt berechnen als $\frac{a+bt}{c+et}, t \in [0, 1]$. Das Integral ist $\int \frac{a+bt}{c+et} \mathrm{d}t = \frac{(ae-bc)\log(c+et)+ebt}{e^2}$. Auch dieses Integral wurde computergestützt[2] berechnet und händisch überprüft. Es wird in dieser Gleichung ausschließlich durch e^2 geteilt, es muss also der Sonderfall $e = 0$ separat gehandhabt werden. Mit der gleichen Konstruktion wird die Funktion $\frac{a+bt}{c}, t \in [0, 1]$ integriert, mit $\int \frac{a+bt}{c+0t} = \int \frac{a+bt}{c} \mathrm{d}t = \frac{(a*t)+\frac{b}{2}t^2}{c}$. Sind sowohl e als auch c gleich 0, so ist das Baseballstadion unendlich dünn und daher nicht wohldefiniert. Die andere Stelle, an der die Funktion indefinit sein könnte, ist im Fall $-et \geq c$, in diesem Fall wäre das Argument des Logarithmus ≤ 0. Das Ergebnis ist im Reelen also undefiniert. Aufgrund der Definition von e ist nur der Fall $e = c, t = 1$ denkbar, falls die Polygonzugkante ihr Ziel in einem unendlich kleinen Kreis an einem der Enden des Baseballstadions hat. Damit dieser Wert allerdings auftritt, muss auch die Kante durch diesen Punkt verlaufen, es gilt also auch $-b = a$. Der Vorfaktor $(ae - bc) \stackrel{!}{=} (ae - (-a)(-e)) = (ae - ae) = 0$ ist null, der ganze Term hat also Wert 0, auch für den Limes $t \to 1$.

[2] `https://www.wolframalpha.com/input/?i=(a%2Bbx)+%2F(c%2Bex)+dx`