Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3528

Routenplanung mit Nebenbedingungen

Frederik Hartmann

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Stefan Funke
Betreuer/in:	Prof. Dr. Stefan Funke
Beginn am:	10. Juli 2013
Beendet am:	2. April 2014
CR-Nummer:	F.2.2, G.2.2

Kurzfassung

Um unterschiedliche Geschwindigkeiten beim energieoptimalen Routing mit zeitlichen Nebenbedingungen betrachten und optimieren zu können, erweitert diese Diplomarbeit das Constrained Shortest Path Problem um weitere Parameter bei den Kantenkostenfunktionen.

Nach der Erweiterung wird die Anwendbarkeit der Beschleunigungstechnik „Contraction Hierarchy“ gezeigt und mit dieser Technik das Problem in praktikabler Zeit gelöst.

Ebenfalls wird eine Heuristik entwickelt, die das Optimierungsproblem in kurzer Zeit löst und für viele praktische Anwendungsgebiete verwendbar ist.

Danksagung

Ich möchte mich an dieser Stelle bei allen Personen bedanken, die mich während meines Studiums und dem Verfassen der Diplomarbeit unterstützt und begleitet haben:

Ein besonderer Dank gilt Prof. Dr. Stefan Funke für ein hoch interessantes und aktuelles Thema und die Möglichkeit, eigene Ideen und Vorschläge zu verwirklichen. Ebenfalls möchte ich mich beim gesamten Institut für Formale Methoden der Informatik für die umfangreiche und wohlwollende Unterstützung und ein angenehmes Arbeitsklima bedanken.

Danken möchte ich auch meinen Eltern, die mir das Studium in dieser Form ermöglichten und mir immer mit Rat und Tat zur Seite standen.

Meiner Freundin danke ich für die Geduld und Gelassenheit vor allem während der Endphase dieser Diplomarbeit.

Inhaltsverzeichnis

1. Einleitung **11**

2. Routing mit Nebenbedingungen **13**
2.1. Vorgehensweise . . . 13
2.2. Definitionen . . . 14
2.2.1. Graph . . . 14
2.2.2. Straßentypen . . . 14
2.2.3. Kantenkostenfunktionen . . . 14
2.2.4. Weg . . . 15
2.2.5. Dominanz und pareto-optimale Mengen . . . 16
2.2.6. Constrained Shortest Path Problem . . . 17
2.2.7. Functional Constrained Shortest Path Problem . . . 18

3. Theoretische Grundlagen **19**
3.1. Addition von Kanten . . . 19
3.2. Dominanz der Geschwindigkeit . . . 22
3.3. $\mathcal{NP}$-Vollständigkeit des FCSP . . . 24

4. Energieoptimiertes Pkw Routing **27**
4.1. Die Straßengraphen . . . 27
4.1.1. Die verwendeten Graphen . . . 28
4.1.2. Straßentypen und Geschwindigkeiten . . . 28
4.2. Zeitkostenfunktion . . . 29
4.2.1. Zeitbereiche . . . 29
4.3. Die Energiekostenfunktion . . . 30
4.3.1. Energiebereiche . . . 31
4.4. Vergleich von Wegen . . . 32

5. Contraction Hierarchy **37**
5.1. Grundlagen . . . 37
5.1.1. Veränderung der Lösungsmenge . . . 37
5.1.2. Verknüpfung von Kanten . . . 38
5.2. Algorithmus . . . 38
5.2.1. Auswahl der zu kontrahierenden Knoten . . . 39
5.2.2. Kontraktion . . . 40
5.3. Aufteilung . . . 41

5.4. Ergebnisse . . . 42
5.4.1. Größe der Contraction Hierarchy . . . 42
5.4.2. Point to Point Querys . . . 42

6. Algorithmen **45**
6.1. FCSP Algorithmus . . . 45
6.2. CSP Algorithmus . . . 48
6.3. CSP-CH Algorithmus . . . 49
6.3.1. Laufzeiten . . . 52
6.3.2. Warteschlangenabfragen . . . 52
6.3.3. Laufzeiten und Heapabfragen im Vergleich . . . 54
6.4. CSP-CH Heuristik . . . 55
6.5. FCSP Lösung mit Hilfe der CSP Heuristik . . . 58
6.5.1. Ergebnisse und Fazit . . . 60

7. Zusammenfassung und Ausblick **63**

A. Anhang **65**
A.1. Tabellen . . . 66
A.1.1. Laufzeitanalyse Zielgeschwindigkeiten auf BW-CH . . . 66
A.1.2. Heuristik Verteilung . . . 67
A.1.3. FCSP Heuristikalgorithmus . . . 67

Literaturverzeichnis **69**

Abbildungsverzeichnis

2.1. Beispiel einer CSP Lösungsmenge . . . 17
2.2. Beispiel einer FCSP Lösungsmenge . . . 18

4.1. Typische Verbrauchsfunktion eines Pkw . . . 31
4.2. Beispiel Kantenvergleich . . . 33
4.3. Beispiel Kantenvergleich . . . 34

5.1. Beispiel Kantenkombination . . . 41

6.1. Suchraum CSP-CH . . . 50
6.2. Verteilung 1,005 Heuristik . . . 56
6.3. Laufzeitanalyse Zielgeschwindigkeiten . . . 57
6.4. Fehlende Monotonie des Energieverbrauch . . . 58
6.5. Reduktion Parametermenge durch Appxoimation . . . 59

Tabellenverzeichnis

4.1. Größe und Gebiet der verwendeten Straßengraphen . . . 28
4.2. Straßentyptabelle . . . 28
4.3. Zuordnung der OSM Typen zu Staßentypen . . . 29

5.1. Ergebnisse und Laufzeiten der CH Erstellung . . . 42
5.2. Point to Point Queryzeiten . . . 43

6.1. CSP-CH Queryzeiten bei 42 m/s . . . 52
6.2. Heap Abfragen und erfogreiche Wege bei 42 m/s . . . 52
6.3. Maximale Warteschlangengrößen und mittlere maximale Warteschlangengrößen bei 42 m/s . . . 53
6.4. Anzahl der Knoten mit pareto-optimalen Lösungsmengen . . . 53
6.5. label-setting und CSP-CH Queryzeiten bei 42 m/s . . . 54
6.6. label-setting und CSP-CH Queryzeiten bei 42 m/s . . . 54
6.7. CSP Heuristik Queryzeiten bei 42 m/s . . . 55

6.8. BW-CH FCSP Heuristik 61

A.1. Heuristikslaufzeiten auf BW-CH 66
A.2. 1,005 Heuristikverteilung 67
A.3. ST-CH FCSP Heuristiklösung 67
A.4. BW-CH FCSP Heuristiklösung 68
A.5. DE-CH FCSP Heuristiklösung 68

Verzeichnis der Algorithmen

3.1. CSP Transformationsalgorithmus 24
3.2. FCSP Lösungsalgorithmus 25

4.1. Dominanzalgorithmus 35

5.1. Aufbau der Contraction Hierarchy 38
5.2. Bestimmen der zu kontrahierenden Knoten 39
5.3. Kontraktion eines Knoten 40

6.1. FCSP-CH Disjkstra Modifikation 46
6.2. FCSP CSP Lösung 47
6.3. label setting Algorihmus 48
6.4. CSP-CH Algorithmus 51
6.5. FCSP Heuristik 60

1. Einleitung

Auf Grund steigender Energiekosten, knapper Ressourcen und steigender Umweltverschmutzung ist ein energiesparendes Verhalten wichtiger denn je. Ein bedeutender Teil des Energieverbrauchs entfällt auf Mobilität und Logistik, Bereiche, in denen der Verbrauch von fossilen Energieträgern trotz der beginnenden Elektrifizierung deutlich dominiert. Trotz der Empfehlung von Automobilverbänden und der Politik, Treibstoff zu sparen, ist die Funktion einer treibstoffsparenden Routen- und Geschwindigkeitswahl in üblichen Navigationsgeräten bisher nicht verfügbar.

Die vorliegende Diplomarbeit befasst sich mit der Entwicklung eines Verfahrens zum Routing mit Nebenbedingungen unter Berücksichtigung einer Geschwindigkeitsoptimierung. Bisherige Lösungen basierten auf einer festen Geschwindigkeit und suchten für diese Geschwindigkeit den effizientesten oder kürzesten Weg mit entsprechenden Nebenbedingungen. Da dieses Verfahren jedoch die Möglichkeit, auf Autobahnen ebenfalls langsamer zu fahren, nicht berücksichtigte, führte dies unter Umständen zu dem Fall, dass eine umwegige Landstraße der Autobahn vorgezogen wurde, da diese langsamer zu befahren und damit energieeffizienter war.

Durch die in der Zukunft erwartete Einführung von autonomen Fahrzeugen nimmt dieses Thema sogar noch an Bedeutung zu: Durch die Möglichkeit, die gewünschte Ankunftszeit bereits vor Fahrantritt zu wählen, wird das Fahrzeug dann in die Möglichkeit versetzt, den energieeffizientesten Weg auszusuchen und eventuell durch Straßenbelastung verlorene Zeit durch eine Geschwindigkeitsanpassung wieder einzuholen.

Ziel dieser Arbeit ist die Entwicklung eines Routingverfahrens, um durch eine Fahrzeit-, Geschwindigkeit- und Routenanpassung den Kraftstoffverbrauch des Individualverkehrs zu optimieren.

Beitrag der Arbeit

In der Arbeit wird das bekannte Constrained Shortest Path (CSP) Problem zu einem Functional Constrained Shortest Path (FCSP) Problem erweitert, indem den Kantenkostenfunktionen weitere Parameter hinzugefügt werden.

Nach der Erweiterung wird gezeigt, dass das Beschleunigungsverfahren der Contraction Hierarchy auf das FCSP mit speziellen Kantenkostenfunktionen anwendbar ist und die dadurch erstellte Contraction Hierarchy für alle Parameter gültig ist.

Mit Hilfe des durch die Contraction Hierarchy beschleunigten Graphen wird danach ein Algorithmus entwickelt, der das FCSP Problem durch eine Reduktion auf das CSP Problem löst.

2. Routing mit Nebenbedingungen

Um Fahrkosten zu verringern und die Umweltbelastung zu senken, wird ein Verfahren gesucht, das es ermöglicht, unterschiedliche gefahrene Geschwindigkeit bei der Berechnung von kürzesten Wegen mit energetischen oder zeitlichen Nebenbedingungen im Straßennetz zu betrachten.

Im Gegensatz zu dem normalen Constrained Shortest Path Problem, bei dem für jede Kante zwei feste Kantenkosten definiert sind, müssen hierzu zwei von der variablen Fahrgeschwindigkeit abhängende Kantenkostenfunktionen verwendet werden, da sich sowohl die Fahrzeit als auch der Energieverbrauch in Abhängigkeit der Geschwindigkeit ändern.

2.1. Vorgehensweise

Da das CSP und damit auch das FCSP $\mathcal{NP}$-vollständig ist, ist die garantierte Lösung in polynomieller Zeit nicht möglich. Es ist jedoch möglich, abhängig von der Probleminstanz, deutliche Verbesserungen der benötigten Rechenzeit durch die Verwendung von verschiedenen Speed-Up Techniken zu erreichen.

Nach der Einführung und den Definitionen im Verlaufe dieses Kapitels wird in Kapitel 3 die $\mathcal{NP}$-Vollständigkeit bewiesen und das theoretische Fundament für die Anwendung der Speed-Up Techniken gelegt.

Nach der Modellierung des Problems in Kapitel 4 wird in Kapitel 5 das Verfahren der Contraction Hierarchy [GSSD08] auf die Verwendung im Functional Constrained Shortest Path Routing bei passenden Kantenkostenfunktionen angewendet.

Anhand der erstellten Contraction Hierarchy, die für alle möglichen Geschwindigkeiten gültig ist, wird dann im Kapitel 6 ein Algorithmus entwickelt, der über eine Begrenzung der möglichen Parameter das FCSP Problem auf das CSP Problem zurückführt und genau löst. Durch eine Heuristik wird die Laufzeit mit geringen Genauigkeitsverlusten so verkürzt, dass das Verfahren für viele Anwendungsfälle verwendbar ist.

2.2. Definitionen

Der besseren Übersicht halber werden hier alle wichtigen Bezeichnungen und Definitionen eingeführt.

2.2.1. Graph

Der in dieser Arbeit verwendete Graph $G(V, E, T, C)$ ist über die endliche Knotenmenge V, die endliche Kantenmenge E, die Straßentypmenge T und die Menge der Kantenkostenfunktionen C definiert und repräsentiert einen vereinfachtes Straßennetz. Eine gerichtete Kante $e = (s, t, l_1, l_2, \ldots, l_n), n = |T|$ ist über den Start und Zielknoten $s, t \in V$ und einer Menge an Längen $l_1, \ldots, l_n$ entsprechend der Straßentypen definiert.

Eine Kante mit drei Straßentypen könnte wie folgt definiert sein: $(4, 25, 0, 0, 250)$. Dies würde einer Kante zwischen dem Knoten 4 und 25 mit einer Länge von 250 Metern vom Straßentyp 3 entsprechen.

Definition 2.2.1. *Eine Kante ist einfach, wenn nur eine einzige Straßentyplänge größer 0 ist. Kanten, die nicht einfach sind, sind komplex.*

Einfache Kanten repräsentieren üblicherweise normale Straßen zwischen Punkten. Komplexe Kanten entstehen bei der Zusammenfassung von mehreren einfachen Kanten.
Die Funktion $quelle(e)$ ordnet einer Kante e den Quellknoten und die Funktion $ziel(e)$ ihren Zielknoten zu. Die Funktionen $d_r(e)$ mit $r \in T$ ordnet der Kante e ihrer Länge des Straßentypen r zu.

2.2.2. Straßentypen

In einem Straßennetz gibt es üblicherweise verschiedene Straßentypen mit unterschiedlichen Eigenschaften wie zum Beispiel der Anzahl an Fahrspuren, die maximale und minimale erlaubte Geschwindigkeit oder die Kapazität. Bedeutsam sind im Kontext dieser Arbeit die maximale und minimale erlaubte Geschwindigkeit.

Wir betrachten die Menge der Straßentypen T als eine vereinfachte Abbildung der realen Straßentypen, die Funktionen $v_{max}(r), v_{min}(r), r \in T$ repräsentieren die maximale und minimale erlaubte Geschwindigkeit von Straßentyp r. Die absoluten Geschwindigkeitsgrenzen v_{min} und v_{max} werden über minimale und maximale Geschwindigkeit aller Straßentypen definiert, die ein echtes Geschwindigkeitsintervall $v_{min}(r) < v_{max}(r)$ haben.

2.2.3. Kantenkostenfunktionen

Die Menge der Kantenkostenfunktionen C ist die Menge der Funktionen, die eine Kante auf entsprechende Kantenkosten abbildet. Im weiteren Verlauf der Arbeit sind besonders die Zeitkostenfunktion c_t und die Energiekostenfunktion c_f von besonderer Bedeutung.

Zeitkostenfunktion c_t

Die Kantenkostenfunktion $c_t(e, v)$ entspricht der Fahrzeit einer Kante $e \in E$ in Abhängigkeit der gefahrenen Geschwindigkeit v.

$$c_t(e, v) = \sum_{r \in T} \frac{d_r(e)}{\tilde{v}_r} \text{ mit } \tilde{v}_r = min(max(v, v_{min}(r)), v_{max}(r))$$

Es wird angenommen, dass auf allen Straßentypen einer Kante die gleiche Geschwindigkeit gefahren wird, soweit dies möglich ist. Ist v außerhalb des erlaubten Geschwindigkeitsintervall, wird mit dem erlaubten Maximum oder Minimum gefahren. Dass dies keine Einschränkung der pareto-optimalen Lösungsmenge bewirkt, ergibt sich durch Satz 3.2.1.

Energiekostenfunktion c_f

Die Kantenkostenfunktion $c_f(e, v)$ entspricht der Energiekostenfunktion einer Kante e in Abhängigkeit der gefahrenen Geschwindigkeit. Sie ist über eine kantenunabhängige Energiekostenfunktion $f(v)$ definiert.

$$c_f(e, v) = \sum_{r \in T} d_t(e) f(\tilde{v}_r) \text{ mit } \tilde{v}_r = min(max(v, v_{min}(r)), v_{max}(r))$$

Es wird vorausgesetzt, dass $f(v)$ im Intervall $[v_{min}, v_{max}]$ ***stetig***, ***monoton steigend***, ***konvex*** und mindestens einmal ***differenzierbar*** ist. Dies ist für sehr viele praktische Verbrauchsfunktionen in den interessanten Geschwindigkeitsbereichen gegeben.

2.2.4. Weg

Ein Weg w zwischen zwei Knoten s und t wird als Aneinanderreihung von Kanten $w = e_0 e_1 \dots e_n$ definiert, für die gilt: $quelle(e_0) = s$, $ziel(e_n) = t$ und $ziel(e_n) = quelle(e_{n+1})$.
Wir definieren die Kosten eines Weges w für die der Kantenkostenfunktion c_t und c_f als:

$$c_t(w, v) = \sum_{e \in w} c_t(e, v)$$

$$c_f(w, v) = \sum_{e \in w} c_f(e, v)$$

Aus den Sätzen 3.1.1 und 3.2.1 folgt, dass die Einschränkung, alle Kanten eines Weges soweit möglich mit der gleichen Zielgeschwindigkeit zu fahren, nicht zu einer Änderung der pareto-optimalen Lösungsmenge führt.

2.2.5. Dominanz und pareto-optimale Mengen

Die Lösungsmenge L eines CSP ist eine pareto-optimale Menge aus Tupeln, welche die Kosten eines Weges repräsentieren. Eine pareto-optimale Menge enthält nur Wege, die nicht durch andere Wege der Menge dominiert werden.

Definition 2.2.2. *Einen Weg w_1 dominiert einen Weg w_2 genau dann, wenn er für keine Kantenkostenfunktion größere Kosten und für mindestens eine Kantenkostenfunktion geringere Kosten als w_2 hat. Es gilt also:*

$$w_1 < w_2 \rightarrow \forall c_i \in C : c_i(w_1) \leq c_i(w_2) \wedge \exists c_i \in C : c_i(w_1) < c_i(w_2)$$

Eine solche Dominanz- und Mengendefinition lässt sich jedoch nicht auf die Lösungsmenge eines FCSP anwenden, die aus Tupeln von Funktionen in Abhängigkeit weiterer Parameter als der Kante besteht. Wir erweitern hierzu den Dominanzbegriff auf eine Funktionsmenge.

Definition 2.2.3. *Ein Weg w_1 dominiert Weg w_2 genau dann, wenn es keine Parametermenge P gibt, mit der w_2 gleiche und für mindestens eine Funktion geringere Kantenkosten als w_1 mit allen gültigen Parametermengen hat.*

Diese erweiterte Dominanzdefinition ermöglicht es, die pareto-optimale Lösungsmenge des FCSP zu definieren.

Definition 2.2.4. *Eine pareto-optimale Funktionsmenge F enthält nur Tupel aus Funktionen, die für mindestens eine Parameterkombination nicht durch andere Funktionstupel mit beliebigen Parameterkombinationen aus F dominiert werden.*

2.2.6. Constrained Shortest Path Problem

Das Constrained Shortest Path Problem[Hen86] ist als Suche nach dem kürzesten Weg zwischen zwei Knoten in Bezug auf die Kantenkostenfunktion c_0 und Einschränkungen auf anderen Kantenkostenfunktionen definiert.

Gegeben ist üblicherweise ein Graph $G(V, E, C)$, Start- und Zielknoten $s, t \in V$, eine zu optimierende Kantenkostenfunktion $c_0 \in C$ und Einschränkungen G für alle Kantenkostenfunktionen $c_n \in C \setminus c_0$ in der Form $c_n \leq g_n, g_n \in \mathbb{N}$.

Gesucht ist der Weg von s nach t, der alle Einschränkungen erfüllt und die geringsten Kantenkosten für c_0 hat. Wählt man die Einschränkungen g_n nicht fest, führt dies zu einer $|C|$-dimensionalen, pareto-optimalen Lösungsmenge in Abhängigkeit der g_n. Abbildung 2.1 ist ein Beispiel einer punktweisen, pareto-optimalen Lösungsmenge einer CSP-Instanz.

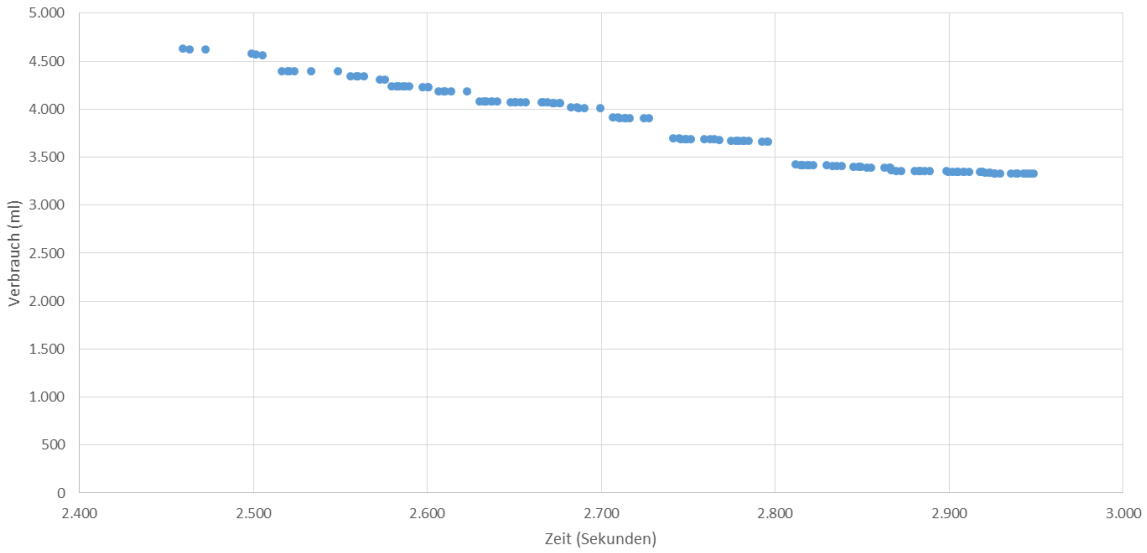


Abbildung 2.1.: Beispiel einer CSP Lösungsmenge mit 120 unterschiedlichen Wegen zwischen zwei Knoten. Jeder Punkt entspricht einem Weg zwischen Start und Ziel.

2.2.7. Functional Constrained Shortest Path Problem

Das Functional Constrained Shortest Path Problem erweitert das CSP um Kantenkostenfunktionen, die nicht die Kantenmenge E als Quellmenge haben. Die Kantenkostenfunktionen eines FCSP bilden beispielsweise die Menge $E \times \mathbb{N}$ (Kante $\times$ Geschwindigkeit) auf die Menge der natürlichen Zahlen ab. Gegeben ist ein Graph $G(V, E, C)$, Start- und Zielknoten $s, t \in V$, eine zu optimierende Kantenkostenfunktion $c_0 \in C$ und Einschränkungen für alle Kantenkostenfunktionen $c_n \in C \setminus c_0$ in der Form $c_n \leq g_n, g_n \in \mathbb{N}$.

Gesucht ist der Weg von s nach t mit einer entsprechenden Parametermenge für alle Kantenkostenfunktionen, der die geringste Kantenkostensumme von c_0 hat. Wählt man die Einschränkungen g_n nicht fest, führt dies zu einer $|C|$-dimensionalen, pareto-optimalen Lösungsmenge aus Funktionen in Abhängigkeit der g_n, siehe dazu auch Abbildung 2.2.

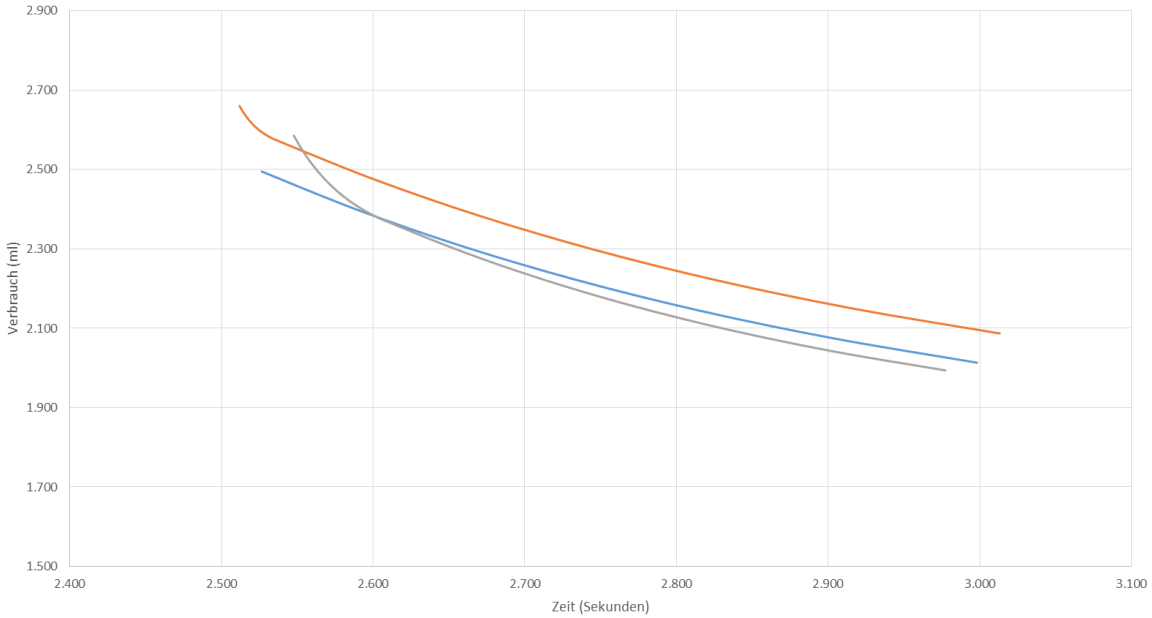


Abbildung 2.2.: Beispiel einer FCSP Lösungsmenge mit drei unterschiedlichen Wegen zwischen zwei Knoten. Jede Funktion entspricht einem Weg zwischen Start und Ziel.

3. Theoretische Grundlagen

Dieses Kapitel beweist verschiedene Annahmen aus den Definitionen und legt den Grundstein für die Verwendung der Contraction Hierarchy. Am Ende des Kapitels wird die $\mathcal{NP}$-Vollständigkeit des allgemeinen FCSP gezeigt.
Die Beweise beziehen sich speziell auf die Definitionen der Kantenzeitkostenfunktion c_t und der Kantenenergiekostenfunktion c_f.

3.1. Addition von Kanten

Die Verknüpfung von Kanten hat bei der Verwendung von vielen Speed-Up Techniken eine große Bedeutung. Wir definieren die Verknüpfung der Kante $e_0 = (u, v, l_1, \ldots, l_n)$ und der Kante $e_1 = (v, w, m_1, \ldots, m_n)$ mit $n = |T|$ zur Kante e_2 wie folgt:

$$e_0 \circ e_1 = e_2 = (u, w, l_1 + m_1, \ldots, l_n + m_n)$$

Wir nehmen an, dass auf allen Kanten eines Weges der pareto-optimalen Lösung soweit möglich die gleiche Zielgeschwindigkeit gefahren wird. Dann gilt für die Kantenkostenfunktion ein Linearitätsprinzip:

$$c_t(e_0, v) + c_t(e_1, v) = c_t(e_2, v)$$

$$\sum_{r \in T} d_r(e_0) f(\tilde{v}_r) + \sum_{r \in T} d_r(e_1) f(\tilde{v}_r) = \sum_{r \in T} (d_r(e_0) + d_r(e_1)) f(\tilde{v}_r)$$

$$\sum_{r \in T} \frac{d_r(e_0)}{\tilde{v}_r} + \sum_{r \in T} \frac{d_r(e_1)}{\tilde{v}_r} = \sum_{r \in T} \frac{d_r(e_0) + d_r(e_1)}{\tilde{v}_r}$$

$$\tilde{v}_r = min(max(v, v_{min}(r)), v_{max}(r)) \text{ mit } r \in T$$

Offensichtlich können bei gleicher Geschwindigkeit Kanten entsprechend ihrer Straßentypkomponenten addiert werden. Es wird jetzt gezeigt, dass jeder Weg, der Element der pareto-optimalen Lösungsmenge ist, die gleichen Geschwindigkeitsparameter für jeden Straßentyp haben muss.

Satz 3.1.1. *Die Kantentypgeschwindigkeiten sind für jede Kante eines pareto-optimalen Weges identisch.*

Beweis. Ohne Beschränkung der Allgemeinheit wird der Satz für den Straßentyp r bewiesen.

Auf Grund der Monotonie der Verbrauchsfunktion $f(x)$ gilt:

$$f(x) \leq f(x + \epsilon) \text{ für } \epsilon \geq 0$$

Nach Konvexitätsdefinition gilt für jedes $k \in [0, 1]$:

$$f(kx + (1 - k)y) \leq kf(x) + (1 - k)f(y)$$

Zu zeigen ist, dass für zwei Kanten e_0, e_1 der Energieverbrauch bei unterschiedlichen Geschwindigkeiten v_0, v_1 nicht niedriger als der Energieverbrauch mit einer Durchschnittsgeschwindigkeit v bei gleicher Fahrzeit sein kann.

Aus der Definition der pareto-optimalen Lösungsmenge folgt dann, dass Wege mit unterschiedlichen Geschwindigkeiten für den gleichen Kantentyp nicht in der pareto-optimalen Lösungsmenge liegen können. Es ist demnach zu zeigen, dass gilt:

$$c_f(d_r(e_0), v_0) + c_f(d_r(e_1), v_1) \geq c_f(d_r(e_0), v) + c_f(d_r(e_1), v)$$

Aus der Definition der Kantenenergiekostenfunktion erhält man:

$$f(v_0)d_r(e_0) + f(v_1)d_r(e_1) \geq f(v)d_r(e_0) + f(v)d_r(e_1)$$

Da die Fahrzeiten identisch sein müssen, kann die Durchschnittsgeschwindigkeit über die Längen beider Kanten und den entsprechenden Geschwindigkeiten errechnet werden. Aus der Fahrtzeit t

$$t = \frac{d_r(e_0) + d_r(e_1)}{v} = \frac{d_r(e_0)}{v_0} + \frac{d_r(e_1)}{v_1}$$

erhalten wir die Durchschnittsgeschwindigkeit v:

$$v = \frac{v_0 v_1 (d_r(e_0) + d_r(e_1))}{v_0 d_r(e_1) + v_1 d_r(e_0)}$$

Zu zeigen ist jetzt:

$$f(v_0)d_r(e_0) + f(v_1)d_r(e_1) \geq f(v)(d_r(e_0) + d_r(e_1))$$

$$f(v_0) \underbrace{\frac{d_r(e_0)}{d_r(e_0) + d_r(e_1)}}_{k} + f(v_1) \underbrace{\frac{d_r(e_1)}{d_r(e_0) + d_r(e_1)}}_{1-k} \geq f(v)$$

$$kf(v_0) + (1-k)f(v_1) \geq f(v)$$

mit $k \in [0,1]$ weil $d_r(e) > 0$.

Wir nehmen zunächst an, dass $v_0 k + (1-k)v_1 \geq v$ gilt. Dann gilt auf Grund der Monotonie von f die Abschätzung:

$$f(v_0 k + (1-k)v_1) \geq f(v)$$

Wir wenden auf die linke Seite der Gleichung die Konvexitätsbedingung für f an und erhalten:

$$\underbrace{kf(v_0) + (1-k)f(v_1) \geq f(v_0 k + (1-k)v_1)}_{\text{Konvexitätsdefinition}} \geq f(v)$$

Damit wäre der gewünschte Beweis erbracht. Zu zeigen ist jetzt noch, dass $v_0 k + (1-k)v_1 \geq v$ gilt. Dazu wird $k, 1-k$ und die Durchschnittsgeschwindigkeit v eingesetzt.

$$v_0 k + (1-k)v_1 \geq v$$
$$\frac{v_0 d_r(e_0) + v_1 d_r(e_1)}{d_r(e_0) + d_r(e_1)} \geq \frac{v_0 v_1 (d_r(e_0) + d_r(e_1))}{v_0 d_r(e_1) + v_1 d_r(e_0)}$$
$$v_0 d_r(e_0) + v_1 d_r(e_1) \geq \frac{v_0 v_1 (d_r(e_0) + d_r(e_1))^2}{v_0 d_r(e_1) + v_1 d_r(e_0)}$$
$$(v_0 d_r(e_0) + v_1 d_r(e_1))(v_0 d_r(e_1) + v_1 d_r(e_0)) \geq v_0 v_1 (d_r(e_0) + d_r(e_1))^2$$
$$v_0 v_1 ((d_r(e_0))^2 + (d_r(e_1))^2) + d_r(e_0) d_r(e_1)(v_0^2 + v_1^2) \geq v_0 v_1 (d_r(e_0) + d_r(e_1))^2$$
$$d_r(e_0) d_r(e_1)(v_0^2 + v_1^2) + v_0 v_1 (d_r(e_0) + d_r(e_1))^2 - 2 v_0 v_1 d_r(e_0) d_r(e_1) \geq v_0 v_1 (d_r(e_0) + d_r(e_1))^2$$
$$d_r(e_0) d_r(e_1)(v_0^2 + v_1^2) - 2 v_0 v_1 d_r(e_0) d_r(e_1) \geq 0$$
$$v_0^2 + v_1^2 - 2 v_0 v_1 \geq 0$$
$$(v_0 - v_1)^2 \geq 0$$

□

3.2. Dominanz der Geschwindigkeit

Satz 3.2.1. *Ist ein Weg w Element der pareto-optimalen Lösungsmenge, wird auf allen Straßentypen die dominierende Geschwindigkeit v gefahren. Ist dies auf Grund der Geschwindigkeitsdefinitionsgrenzen eines Kantentyps nicht möglich, wird die nächst mögliche Geschwindigkeit gefahren.*

Beweis. Da jeder Weg nach Satz 3.1.1 als eine Kante dargestellt werden kann, ist es ausreichend, den Satz für eine Kante u zu zeigen, die Weg w repräsentiert.

$$f(v)d_1(u) + f(v+x)d_2(u) \leq f(v-y)d_1(u) + f(v+x+z)d_2(u) \text{ mit } x, y, z \geq 0$$

Da u pareto-optimal ist, kann die Geschwindigkeit durch die Fahrzeit fixiert werden:

$$\frac{d_1(u)}{v} + \frac{d_2(u)}{v+x} = \frac{d_1(u)}{v-y} + \frac{d_2(u)}{v+x+z}$$

Auf Grund der Konvexität kann die Funktion f an der Stelle $v+x$ durch das Anlegen einer Tangente an v nach unten abgeschätzt werden.

$$f(v-y)d_1(u) \geq (f(v) - f'(v)y)d_1(u)$$
$$f(v+x+z)d_2(u) \geq (f(v+x) + f'(v+x)z)d_2(u)$$

Dadurch erhält man folgende Ungleichung:

$$f(v)d_1(u) + f(v+x)d_2(u) \leq (f(v) - f'(v)y)d_1(u) + (f(v+x) + f'(v+x)z)d_2(u)$$
$$0 \leq f'(v+x)zd_2(u) - f'(v)yd_1(u)$$
$$f'(v)yd_1(u) \leq f'(v+x)zd_2(u)$$

Wenn gezeigt werden kann, dass $yd_1(u) \leq zd_2(u)$ gilt, ist der Satz bewiesen, da auf Grund der Konvexität von f gilt: $f'(v) \leq f'(v+x)$.
Es wird anstelle der Gültigkeit der Ungleichung $yd_1(u) \leq zd_2(u)$ die Gültigkeit der Ungleichung

$$\tilde{y}d_1(u) \leq zd_2(u)$$

mit einem geeignet gewählten $\tilde{y} \geq y$ gezeigt. Es wird nun ein passendes $\tilde{y}$ konstruiert.

Aus der Fahrtzeit $\frac{d_1(u)}{v} + \frac{d_2(u)}{v+x} = \frac{d_1(u)}{v-y} + \frac{d_2(u)}{v+x+z}$ ergibt sich:

$$y = \frac{d_2(u)v^2z}{d_1(u)(v+x)(v+x+z) + d_2(u)vz}$$

Da der Nenner von y nur positive Terme enthält, erhalten wir $\tilde{y}$ durch das Weglassen aller Terme ohne v:

$$\tilde{y} = \frac{d_2(u)vz}{d_1(u)v + d_2(u)z} \text{ mit } \tilde{y} \geq y$$

Einsetzen von $\tilde{y}$ liefert

$$\frac{d_1(u)d_2(u)vz}{d_1(u)v + d_2(u)z} \leq zd_2(u)$$

$$\frac{d_1(u)v}{d_1(u)v + d_2(u)z} \leq 1$$

□

3.3. $\mathcal{NP}$-Vollständigkeit des FCSP

Zur Einordnung der Schwierigkeit des Problems ist die Komplexitätsklasse entscheidend. Durch die Reduktion des normalen CSP als Entscheidungsproblem auf das FCSP als Entscheidungsproblem wird die $\mathcal{NP}$-Schwere von FCSP gezeigt. Nach [HZ80] ist das Entscheidungsproblem des Constrained Shortest Path Problem $\mathcal{NP}$-vollständig.

Lemma 3.3.1. *Das Entscheidungsproblem des Functional Constrained Shortest Path Problem ist $\mathcal{NP}$-schwer.*

Beweis. Wir führen den Beweis, indem wir einen Transformationsalgorithmus 3.1 angeben, der in polynomieller Zeit eine CSP in eine FCSP Instanz umwandelt.

Algorithmus 3.1 CSP Transformationsalgorithmus

procedure CSPTRANSFORM(V,E,C,c_0,G,$c_{0_{max}}$)
  for all $c \in C$ **do**
    $C_{FCSP} \leftarrow C_{FCSP} \cup (c(e,x) \rightarrow c(e))$
  end for
  $G_{FCSP} \leftarrow G$
  FCSP(V,E,C_{FCSP},c_0,G,$c_{0_{max}}$)
end procedure

Durch diese Transformation erhalten wir eine gültige FCSP Instanz. Wenn FCSP nicht $\mathcal{NP}$-schwer wäre, könnte dadurch CSP in polynomieller Zeit entschieden werden, was unter der Voraussetzung $\mathcal{P} \neq \mathcal{NP}$ nicht möglich ist. □

Nach der $\mathcal{NP}$-Schwere ist jetzt noch zu zeigen, dass das Entscheidungsproblem von FCSP Element von $\mathcal{NP}$ ist. Dazu raten wir die passende Parametermenge und transformieren dadurch das FCSP in ein CSP von dem die $\mathcal{NP}$-Vollständigkeit bekannt ist.

Lemma 3.3.2. *Das Entscheidungsproblem des Functional Constrained Shortest Path Problem ist Element von $\mathcal{NP}$.*

Beweis. Durch das Orakel O wird eine passende Parametermenge für das Entscheidungsproblem von FCSP geraten. Mit Hilfe dieser Parametermenge werden die Kantenkostenfunktionen mit Parametern wieder auf die einfachen Kantenkostenfunktionen des CSP reduziert.

Algorithmus 3.2 FCSP Lösungsalgorithmus

procedure TRANSFORMFCSP(V,E,C,c_0,G,$c_{0_{max}}$,O)
  $P \leftarrow$ durch Orakel O geratene Parametermenge
  for all $c \in C$ **do**
    $C_{CSP} \leftarrow C_{CSP} \cup (c(e) \rightarrow c(e, P))$
  end for
  $c_{0_{CSP}} \leftarrow (c_0(e) \rightarrow c_0(e, P))$
  CSP(V,E,C_{CSP},$c_{0_{CSP}}$,G)
end procedure

Der Algorithmus 3.2 ist Element von $\mathcal{NP}$, da CSP Element von $\mathcal{NP}$ ist und die Transformation an sich ebenfalls Element von $\mathcal{NP}$ ist. □

Durch Lemma 3.3.1 und 3.3.2 ergibt sich nach Definition:

Satz 3.3.1. *Das Entscheidungsproblem des Functional Constrained Shortest Path Problem ist $\mathcal{NP}$-vollständig.*

4. Energieoptimiertes Pkw Routing

Nach den theoretischen Grundlagen und Definitionen wird eine Einführung in die Modellierung des energieoptimierten Routings mit Zeitgrenzen gegeben.

Da die Bedeutung der gefahrenen Kilometer im praktischen Gebrauch oft geringer ist als die Fahrzeit oder der Energieverbrauch, wird die Streckenlänge nur zur Berechnung des entsprechenden Zeit- und Energieverbrauchs verwendet. Durch die verwendete Energiekostenfunktion ist eine realistische Umwandlung von Streckenlänge in fossilen Kraftstoff möglich.

4.1. Die Straßengraphen

Die vorliegende Arbeit verwendet als Datenquelle eine transformierte Deutschlandkarte aus dem OpenStreepMap Projekt. Aufgrund der Existenz von mehrfachen Kanten zwischen Knoten in den Rohdaten gehen wir von einem Multigraph $G(V, E)$ bestehend aus Knotenmenge E und Kantenmenge V aus. Die Kanten, welche direkt durch die Transformation aus den OSM Daten entstanden sind, haben keinen kontrahierten Knoten k und bestehen nur aus einem Straßentyp. Kanten, welche bei der Erstellung der Contraction Hierarchy entstanden sind, bestehen eventuell aus mehreren Straßentypen und haben als kontrahierten Knoten denjenigen Knoten zugeordnet, bei dessen Kontraktion sie entstanden sind.

- Knoten
  - KnotenID
  - Latitude
  - Longitude
  - Level
- Kanten
  - Startknoten
  - Zielknoten
  - Länge Innenstadt
  - Länge Landstraße
  - Länge Autobahn
  - Kontrahierter Knoten

4.1.1. Die verwendeten Graphen

Die in Tabelle 4.1 dargestellten Graphen repräsentieren Gebiete aus Deutschland und sind nicht notwendigerweise zusammenhängend. Bei allen Experimenten wurden Knotenpaare, zwischen denen kein Weg besteht, ignoriert. Der durchschnittliche Eingangs- und Ausgangsgrad liegt in allen unbearbeiteten Graphen ungefähr bei zwei.

Graph	Geographisches Gebiet	Knoten	Kanten
DE	Deutschland	19.478.240	39.454.253
BW	Baden-Württemberg	2.911.711	5.903.801
ST	Regionalbezirk Stuttgart	924.688	1.876.030

Tabelle 4.1.: Größe und Gebiet der verwendeten Straßengraphen

4.1.2. Straßentypen und Geschwindigkeiten

Das Straßennetz wird unter Berücksichtigung von realistischen Geschwindigkeit in drei Typen unterteilt. Eine feinere Unterteilung ist ohne weiteres möglich, erhöht jedoch die Anzahl der benötigten Fallunterscheidungen. Es werden hier speziell die innerstädtischen Straßen, Landstraßen und Autobahnen unterschieden. Tabelle 4.2 stellt die Straßentypen und ihre Geschwindigkeitsbereiche dar.

	Innenstadt in	Landstraße la	Autobahn a
v_{min}[km/h]	50	80	100
v_{max}[hm/h]	50	100	150
v_{min}[m/s]	14	22	28
v_{max}[m/s]	14	28	42

Tabelle 4.2.: Straßentyptabelle

Im weiteren Verlauf wird Länge immer in der Einheit Meter und Geschwindigkeit immer in der Einheit Meter pro Sekunde angegeben.

Die Zuordnung der OpenStreetMap Wegetype zu den Straßentypen erfolgte anhand von Tabelle 4.3.

OSM highway	Straßentyp
motorway	Autobahn
motorway_link	
trunk	
trunk_link	
primary	Landstraße
primary_link	
secondary	
secondary_link	
tertiary	
tertiary_link	
residential	Innenstadt
unclassified	
living_street	

Tabelle 4.3.: Zuordnung der OSM Typen zu Staßentypen

4.2. Zeitkostenfunktion

Für dieses Problem wird die Zeitkostenfunktion aus 2.2.3 ohne Änderungen verwendet, da die Einheiten ohne Umrechnungen kompatibel sind. Durch die definierten Kantentypen und Geschwindigkeitsbereiche ergibt sich damit eine anwendbare, entsprechend der Geschwindigkeit partiell definierte, aber stetige Kantenkostenfunktion.

4.2.1. Zeitbereiche

Durch die erlaubten Geschwindigkeiten kann für jede Kante das Zeitprofil errechnet werden, das aus der minimalen benötigten Zeit, dem Zeitpunkt, bei dem sowohl auf der Autobahn als auch auf der Landstraße die gleiche Geschwindigkeit gefahren wird, sowie der maximal benötigten Zeit besteht. Dieses Zeitprofil wird für die algorithmische Betrachtung eines Weges, beziehungsweise einer Wegmenge, benötigt, da diese Punkte den Wertebereich jeden Weges eingrenzen.

Die minimale Zeit $t_{min}(e)$ ist definiert als Zeit, welche mindestens benötigt wird, um den Weg oder die Kante e zu fahren, es wird auf jedem Kantentyp die entsprechende Maximalgeschwindigkeit gefahren. Die Zeit, bei der sowohl auf der Autobahnkomponente als auch auf der Landstraßenkomponente die gleiche Geschwindigkeit gefahren wird, ist als $t_{mid}(e)$ definiert. Die maximale Zeit $t_{max}(e)$ ist dementsprechend die Zeit, die benötigt wird, um die Kante e bei minimaler Geschwindigkeit auf allen Straßentypen zu fahren.

$$t_{min}(e) = \frac{d_{in}(e)}{v_{min}(in)} + \frac{d_{la}(e)}{v_{min}(la)} + \frac{d_a(e)}{v_{min}(a)}$$
$$t_{mid}(e) = \frac{d_{in}(e)}{v_{max}(in)} + \frac{d_{la}(e)}{v_{max}(la)} + \frac{d_a(e)}{v_{min}(a)}$$
$$t_{max}(e) = \frac{d_{in}(e)}{v_{max}(in)} + \frac{d_{la}(e)}{v_{max}(la)} + \frac{d_a(e)}{v_{max}(a)}$$

Diese Unterteilung funktioniert nur bei sich nicht überlappenden Geschwindigkeitsbereichen. Sollten sich Geschwindigkeitsbereiche überlappen, muss für die überlappenden Bereiche ebenfalls eine Fallunterscheidung mit einer entsprechenden Berechnung erfolgen.

Durch die Zeitpunkte lässt sich dann die gefahrene Zeit und Geschwindigkeit für jeden Kantenbestandteil berechnen.

$$t_{in}(t, e) = \frac{d_{in}(e)}{v_{max}(in)}$$
$$t_{la}(t, e) = \begin{cases} t - t_{min}(e) & \text{für } t_{min}(e) \leq t \leq t_{mid}(e) \\ \frac{d_{la}(e)}{v_{max}(la)} & \text{für } t_{mid}(e) < t \leq t_{max}(e) \end{cases}$$
$$t_a(t, e) = \begin{cases} \frac{d_{la}(e)}{v_{min}(la)} & \text{für } t_{min}(e) \leq t \leq t_{mid}(e) \\ t - t_{mid}(e) & \text{für } t_{mid}(e) < t \leq t_{max}(e) \end{cases}$$
$$e \in E \text{ und } t \in \mathbb{N}$$

4.3. Die Energiekostenfunktion

Nachdem die Zeitkostenfunktionen und die Zeitabschnitte definiert sind, werden wir die Energiekostenfunktionen berechnen. Als Grundlage wird hier eine Beispielfunktion der FU Berlin verwendet [FUB09], die in den interessanten Geschwindigkeitsbereichen positiv, stetig, konvex und monoton steigend ist. Hier kann jedoch jede andere positive, stetige, konvexe, monoton steigende und mindestens einmal differenzierbare Funktion verwendet werden.

$$f(v)[ml/m] : v \to 0.00625968v^2 - 0.11736v + 2.1714 + \frac{18\frac{1}{3}}{v}$$

Abbildung 4.1 stellt diese Funktion graphisch dar.

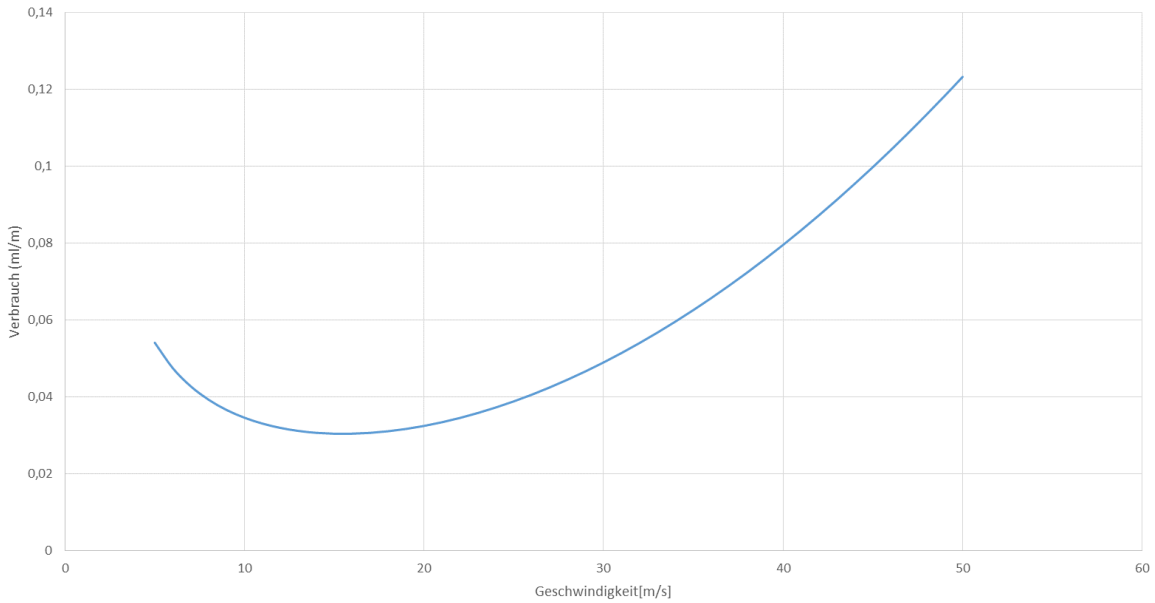


Abbildung 4.1.: Typische Verbrauchsfunktion eines Pkw

Zur Verringerung von numerischen Ungenauigkeiten wird der Verbrauch in ml/m statt in l/m angegeben. Diese Energiekostenfunktion ist spezifisch für jedes Fahrzeug und entsprechend den Gegebenheiten anzupassen. Ist eine Verbrauchsfunktion konvex, aber in einem Geschwindigkeitsbereich für einen Straßentyp nicht monoton steigend, kann die Minimalgeschwindigkeit des Straßentypes zum Minimum der Verbrauchsfunktion verschoben werden, da in diesem Fall eine Steigerung der Geschwindigkeit bei beiden Kantenkostenfunktionen eine Verbesserung darstellt.

4.3.1. Energiebereiche

Bedeutsam für die algorithmische Betrachtung sind besonders die Funktionen f_{min} und f_{max}. Diese Funktionen bilden einen Weg auf die entsprechenden minimalen und maximalen Kosten des Weges ab.

$$f_{min}(w) = \frac{d_{in}(w)}{v_{min}(in)} + \frac{d_{la}(w)}{v_{min}(la)} + \frac{d_a(w)}{v_{min}(a)}$$
$$f_{mid}(w) = \frac{d_{in}(w)}{v_{max}(in)} + \frac{d_{la}(w)}{v_{max}(la)} + \frac{d_a(w)}{v_{min}(a)}$$
$$f_{max}(w) = \frac{d_{in}(w)}{v_{max}(in)} + \frac{d_{la}(w)}{v_{max}(la)} + \frac{d_a(w)}{v_{max}(a)}$$

4.4. Vergleich von Wegen

Lemma 4.4.1. *Die Energiekostenfunktion in Abhängigkeit der Zeit ist für den Zeitbereich* $[t_{min}, t_{max}]$ *streng monoton fallend und innerhalb der Zeitbereiche* $[t_{min}, t_{mid}]$ *und* $[t_{mid}, t_{max}]$ *konvex.*

Beweis. Durch die Definition $Geschwindigkeit = \frac{Distanz}{Zeit}$ kann die Zeit direkt in die Energiekostenfunktion eingesetzt werden. Ein Anstieg der Zeit führt immer zu einer Verringerung der Geschwindigkeit.

Die Energiekostenfunktion besteht, abhängig von dem Zeitintervall, immer aus maximal einem von der Zeit abhängenden Summanden:

$$c_{f_t}(t,e) = \begin{cases} d_{in}(e)f(v_{max}(in)) + d_{la}(e)f(\frac{d_{la}(e)}{t_{la}(t,e)})) + d_a(e)f(v_{max}(a)) & \text{für } t \in [t_{max}(e), t_{mid}(e)] \\ d_{in}(e)f(v_{max}(in)) + d_{la}(e)f(v_{max}(la)) + d_a(e)f(\frac{d_a(e)}{t_a(t,e)})) & \text{für } t \in]t_{mid}(e), t_{min}(e)] \end{cases}$$

Da dieser Summand in Abhängigkeit der Zeit konvex und monoton fallend ist, ist die Energiekostenfunktion in Abhängigkeit der Zeit innerhalb aller Intervalle konvex und im gesamten Definitionsbereich monoton fallend.

□

Die in Lemma 4.4.1 gezeigte Eigenschaft wird verwendet, um eine approximierte Dominanzuntersuchung von Wegpaaren vorzunehmen. Auf Grund der Konvexität aller Geschwindigkeitsbereiche kann für jeden Geschwindigkeitsbereich durch ein Newtonverfahren eine Schnittpunktsuche durchgeführt werden.

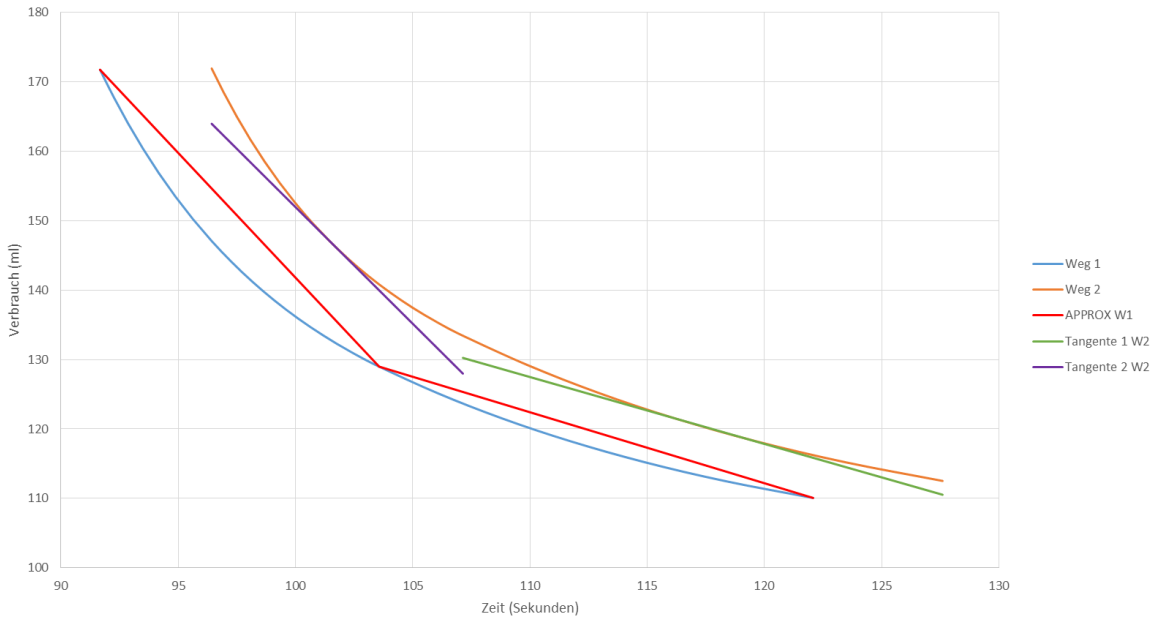


Abbildung 4.2.: Beispiel eines Kantenvergleichs von zwei Wegen: Weg 1 besteht in diesem Beispiel aus 1900 Metern Landstraße und 1000 Metern Autobahn, Weg 2 setzt sich aus 2100 Metern Landstraße und 900 Metern Autobahn zusammen.

Wie in Abbildung 4.2 deutlich wird, ist dieses Verfahren für ausreichend große Abstände zwischen den Wegen in kurzer Zeit durchführbar. Eine exakte Lösung eines Polynoms dritten Grades wäre wesentlich komplizierter, da auf Grund der Fallunterscheidungen für jeden Straßentyp eine solche Untersuchung nötig wäre. Sehr ähnliche Wege kann man dadurch jedoch nicht erkennen, wie in Abbildung 4.3 deutlich wird. Dies kann unter Umständen dazu führen, dass unnötige Abkürzungen beim Erstellen der CH angelegt werden, was sich negativ auf die Abfragegeschwindigkeit auswirken kann.

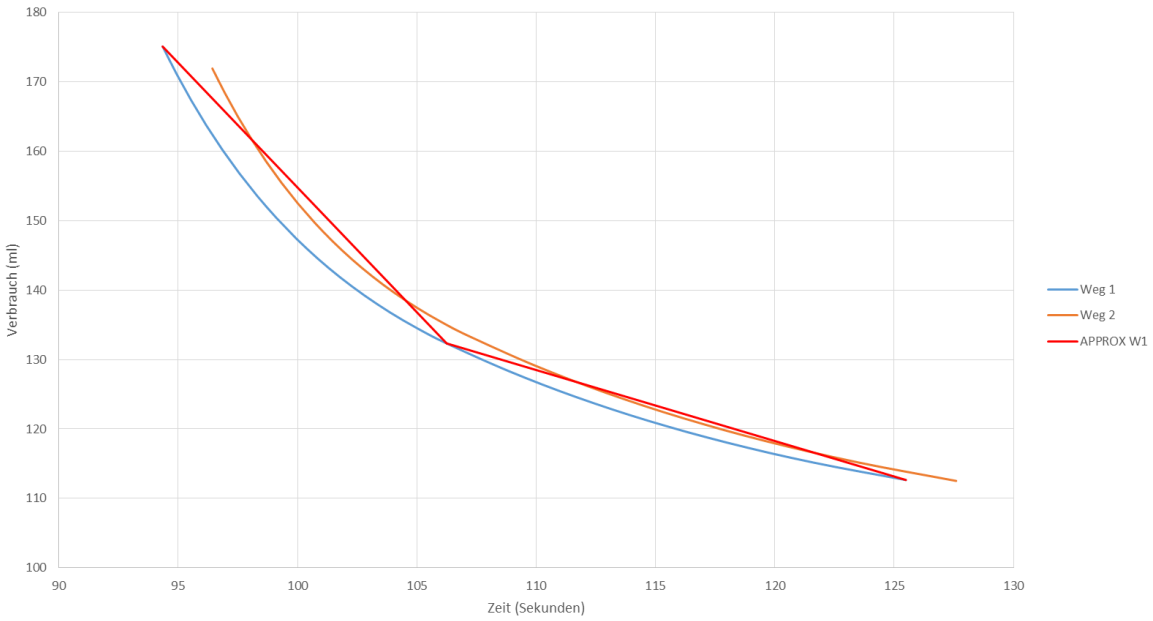


Abbildung 4.3.: Beispiel von zwei Wegen (0 1975 1000) und (0 2100 910), die mit dem Vergleichsalgorithmus falsch entschieden werden.

Bei der Untersuchung der Grundeigenschaften prüft der Algorithmus 4.1, ob $w1$ in kürzerer Zeit gefahren wird als $w2$. Ist dies zutreffend, kann $w2$ unter keinen Umstanden $w1$ dominieren. Ist der minimale Energieverbrauch von $w1$ niedriger als der minimale Energieverbrauch von $w2$, kann $w1$ ebenfalls nicht von $w2$ dominiert werden. Ist $w2$ für alle Straßentypen kürzer als $w1$, wird $w1$ dominiert.

Die Untersuchung der Grundeigenschaften ist für sehr viele Fälle bereits ausreichend, da alle Schleifen, Kreise oder an sich längere Wege sehr einfach durch die Untersuchung der Grundeigenschaften entschieden werden. Führt die Untersuchung der Grundeigenschaften nicht zu einer Entscheidung, wird durch ein Newton Verfahren ermittelt, ob sich die an $w1$ angelegten Tangenten in ihrem interessanten Gebiet ($[t_{min}, t_{mid}]$ bzw $[t_{mid}, t_{max}]$) mit der durch Sekanten approximierte Funktion von $w2$ schneiden.

Algorithmus 4.1 Algorithmus zum Entscheiden ob Weg $w1$ durch Weg $w2$ dominiert wird

```
procedure DOMINATED(w1,w2)
                                                  // Untersuchung der Grundeigenschaften
    if t_min(w1) < t_min(w2) then
        return false
    end if
    if f_min(w1) < f_min(w2) then
        return false
    end if
    if d_in(w2) ≤ d_in(w1) ∧ d_la(w2) ≤ d_la(w1) ∧ d_a(w2) ≤ d_a(w1) then
        return true
    end if
                                                  // Schnittpunktuntersuchung
    if w1 schneidet w2 then
        return false
    end if
    return true
end procedure
```

5. Contraction Hierarchy

Das Konzept der Contraction Hierarchy [GSSD08] ist ein relativ neues Verfahren, welches im Jahr 2008 an der Universität Karlsruhe zur Beschleunigung von normalen Distanzberechnungen in Graphen entwickelt wurde. In einem Vorbereitungsschritt werden alle Knoten eines Graphen nacheinander kontrahiert, ohne im Restgraphen die Struktur der kürzesten Wege zu verändern. Die Ermittlung der kürzesten Distanz erfolgt dann durch eine bidirektionale Anwendung von Dijkstra's Algorithmus [Dij59]. Eine genaue Dokumentation kann [GSSD08] und [Gei08] entnommen werden.

5.1. Grundlagen

Das Konzept der Contraction Hierarchy basiert auf der Eigenschaft, dass durch die korrekte Kontraktion von Knoten und dem Anlegen der nötigen Shortcuts keine Veränderung der Lösungsmenge erfolgt und dass die Verknüpfung von Kanten einfach ist. Vor der Anwendung der CH werden diese Voraussetzungen geprüft.

5.1.1. Veränderung der Lösungsmenge

Für das normale Shortest-Path Routing erfolgt die Prüfung, ob die Kontraktion eines Knotens n den kürzesten Pfad ändern, durch eine Distanzberechnung zwischen allen adjazenten Knotenpaaren. Ist der kürzeste Weg der beiden zu n adjazenten Knoten k und v unter Vermeidung des Knoten n größer als die Summe der Kosten von Kante (k, n) und (n, v), muss eine Abkürzung k, v mit den Kosten der beiden ersetzten Kanten eingeführt werden.

Wird eine CH für das FCSP aufgebaut, reicht es nicht, eine einfache Distanzberechnung durchzuführen, sondern es muss für jedes Knotenpaar eine entsprechende FCSP Instant gelöst werden, was im Allgemeinen nicht in polynomieller Zeit möglich ist. Durch die oft kleinen Distanzen zwischen s und t ist eine solche Berechnung aber oft in kurzer Zeit möglich.

5.1.2. Verknüpfung von Kanten

Das Verknüpfung von Kanten kann bei manchen Anwendungsfällen durchaus problematisch sein, da die Komplexität der Kanten mit jeder Verknüpfung steigen kann. Wenn man annimmt, dass zum Beispiel für jede Kante unterschiedliche Maximalgeschwindigkeiten gelten und diese zu bewahren sind, ist eine einfache Addition nicht möglich. Oft ist jedoch eine Anpassung der Graphstuktur oder der Kantenkostenfunktion möglich, um dieses Problem zu lösen (vergleiche dazu [BDSV09] und [EFS11]). Wie wir jedoch bereits in Satz 3.1.1 gezeigt haben, ist die Verknüpfung von Kanten ohne Änderung der pareto-optimalen Lösungsmenge möglich und erhöht nicht die Komplexität der Kanten.

5.2. Algorithmus

Der für diese Arbeit verwendete Algorithmus 5.1 folgt im Prinzip dem in [Gei08][3.1] definierten Algorithmus. Er unterscheidet sich jedoch durch das vorherige Anlegen einer Kontraktionsliste, die immer maximal 2% der aktuellen Knoten des Graphs enthält. Diese Liste ist durch die Konstruktion voneinander unabhängig und kann daher parallelisiert kontrahiert werden. Dies ist auf Grund der im Vergleich zur CH Berechnung für normale Distanzberechnungen wesentlich wichtiger, da der Kontraktionsschritt auf Grund der Komplexität des CSP beziehungsweise FCSP bedeutend länger dauern kann und dieser normalerweise zur Einordnung von Knoten simuliert wird.

Der folgende Algorithmus beschreibt das grundlegende Verfahrung der Erstellung einer Contraction Hierarchy, dessen einzelne Schritte im Folgenden noch genauer erläutert werden.

Algorithmus 5.1 Aufbau der Contraction Hierarchy

```
procedure buildCH(V,E)
    int[] levels ← new int[|V|]
    int level ← 1
    Set addEdges
    List contract
    while |N|>0 do
                                                    // Aufbau der Kontraktionsliste
        contract ← buildNodeList(V,E)
        while contract.size > 0 do
            n ← contract.removeHead
                                                    // Kontraktion des Knotes
            contract(V,E,n)
                                                    // Setzen des entsprechenden Kontraktionslevels
            levels[n] ← level
        end while
        level++
    end while
end procedure
```

5.2.1. Auswahl der zu kontrahierenden Knoten

In der grundlegenden Abhandlung [GSSD08] werden zwei wichtige Eigenschaften genannt, die für eine gute Auswahl der zu kontrahierenden Knoten zu erfüllen sind. Als wichtigste Eigenschaft wird die Kantendifferenz zwischen den Graph G vor der Kontraktion und dem Graph G' genannt, eine weitere wünschenswerte Eigenschaft ist eine möglichst gleichmäßige Verteilung der kontrahierten Knoten.

In [GSSD08] wurde die Kantendifferenz durch eine simulierten Kontraktion berechnet und bei jeder Änderung der angrenzenden Knoten oder Kanten neu berechnet. Dieses Verfahren ist für einfache Distanzberechnungen ausreichend performant, führt aber bei einer CSP CH zu Problemen, da für jeden adjazenten Knoten des zu kontrahierenden Knoten zu jedem anderen adjazenten Knoten eine CSP Instanz gelöst werden muss, um zu entscheiden, ob die Konkatenation der verbindenden Kanten Bestandteil der pareto-optimalen Lösungsmenge ist. Da diese Berechnungen je nach Probleminstanz sehr zeitintensiv sind, wird in Algorithmus 5.2 eine pessimistische Abschätzung der Kantendifferenz verwendet.

Algorithmus 5.2 Bestimmen der zu kontrahierenden Knoten

```
procedure BUILDNODELIST(V,E)
    PriorityQueue Q ← ∅
    Set adj ← ∅
    Set taken ← ∅
    List result ← ∅
                                                    // Bewertung für jeden Knoten ausführen
    for all n ∈ N do
        currentDegree ← 0
        adj ← ∅
                                                    // Grad eines Knotens ermitteln
        for all e(v, w) ∈ E : v = n do
            currentDegree + +
            adj.ADD(n)
        end for
                                                    // Kantendifferenz abschätzen
        Q.ADD(((adj.size−1)adj.size)/2 − currentDegree, n)
    end for
                                                    // Kontraktionsliste aufbauen
    while Q.size > 0 do
        n ← Q.REMOVE
                                                    // Markierte Knoten ignorieren
        if taken.CONTAINS(n) then
            CONTINUE
        end if
        result.ADD(o)
                                                    // Knoten im Umkreis markieren
        for all m ∈ N : m adjazent zu n do
            taken.ADD(m)
            for all o ∈ N : o adjazent zu m do
                taken.ADD(o)
            end for
        end for
    end while
    return result
end procedure
```

5.2.2. Kontraktion

Das Herzstück des Algorithmus zur Berechnung der Contraction Hierarchy ist die Kontraktion der Knoten. Hier wird für alle gültigen adjazenten Knotenpaare untersucht, ob die Kontraktion zu einer Veränderung der Lösungsmenge führen könnte. Ist das der Fall, werden die entsprechenden Abkürzungen eingefügt.

Algorithmus 5.3 Kontraktion eines Knoten

```
procedure CONTRACT(V,E,n)
    Set pareto
    Set addEdges
     // Für alle Kombinationen von Eingangs- und Ausgangskanten prüfen, ob sie pareto-optimal
sind
    for all e(v,w) ∈ E : w = n do
        for all f(x,y) ∈ E : x = n do
                                                  // Prüfen, ob die Kantenkombination pareto-optimal ist
            pareto ← FCSP(v,w)
            if pareto.contains(e + f) then
                addEdges.add(e + f)
            end if
        end for
    end for
                                                  // Kontrahierten Knoten aus dem Graph entfernen
    for all e(v,w) ∈ E : v = n ∨ w = n do
        E ← E/e
    end for
                                                  // pareto-optimale Kantenkombinationen in den Graph einfügen
    for all e(v,w) ∈ addEdges do
        E ← E ∪ e
    end for
    N ← N \ n
end procedure
```

Zum Berechnen der pareto-optimalen Lösungsmenge durch den Algorithmus 5.3 wird eine Modifikation des label-setting Algorithmus 6.3 verwendet, der mit Kantenkostenfunktionen umgehen kann. Wird bei der Berechnung der pareto-optimalen Lösungsmenge für einen Knoten die Zeitgrenze von 10 Sekunden überschritten, wird der Knoten nicht kontrahiert und bei den nächsten drei Knotenauswahlen ignoriert.

Beim Hinzufügen eines Knotens in die Lösungsmenge wird ignoriert, dass eine Kombination von zwei oder mehr Wegen gemeinsam einen anderen Weg dominieren können. Dieser Fall ist mit dem in Kapitel 4.4 eingeführten Wegvergleich nicht möglich, da dafür eine genaue Analyse der Schnittpunkte notwendig wäre.

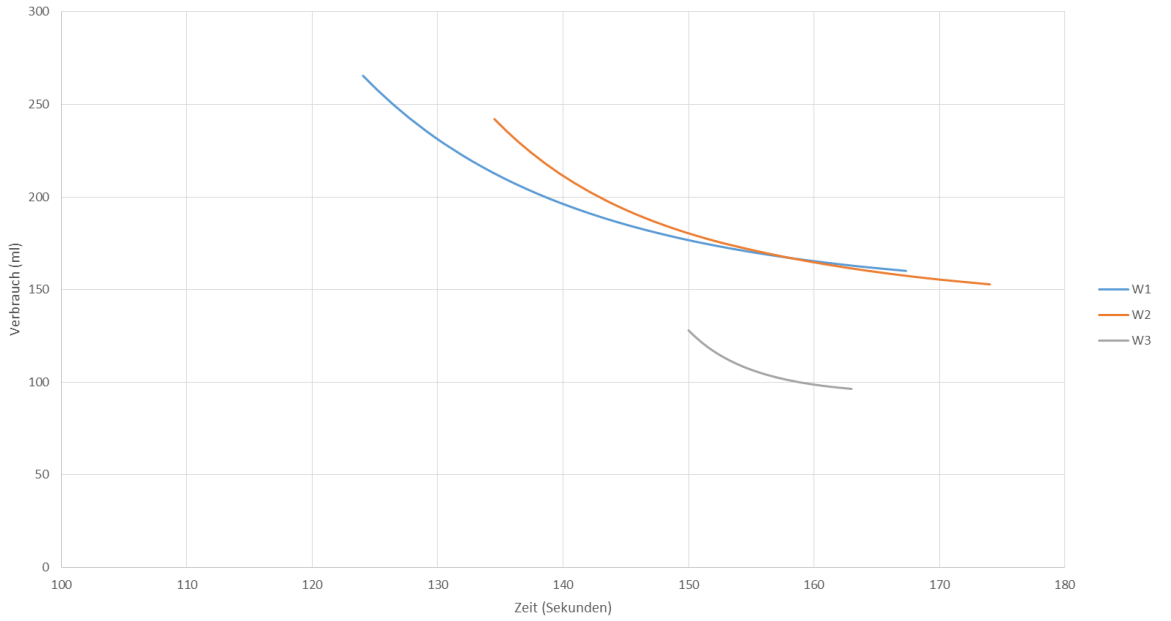


Abbildung 5.1.: Beispiel einer Wegkombination W1(70 2000 2000), W2(400 2100 1600) und W3(1600 600 600), bei der W2 als nicht dominiert gewertet wird, obwohl er von der Kombination von W1 und W3 dominiert wird.

Abbildung 5.1 verdeutlicht auch ein weiteres Problem: Verwendet man eine Modifikation des label-setting Algorithmus zum Lösen des FCSP Problem in Kombination mit komplexen Kantenkostenfunktionen ist es möglich, dass Wege nachträglich als dominiert markiert werden können, wenn man die Wege im Heap nach ihrer minimalen Zeit sortiert. Dies kann zu einer Kaskade von dominierten Wegen im Heap und anderen Knoten führen, die entfernt oder anderweitig markiert werden müssten.

5.3. Aufteilung

Da jede CH-Dijkstra Variante ausschließlich Kanten verwendet, die von einem Knoten zu einem Knoten mit größerem oder gleichem Level führt, können die Kanten in zwei Mengen zerlegt werden: Kanten, die einen Knoten mit einem Knoten von größerem oder gleichem Level verbinden, sind Bestandteil der Menge E und werden ausschließlich von dem vom Startknoten ausgehenden CH-Dijkstra verwendet. Kanten, die einen Knoten mit einem Knoten von niedrigerem oder gleichem Level verbinden, sind invertiert (Start und Zielknoten vertauscht) Bestandteil der Menge $Erev$ und werden ausschließlich vom Zielknoten ausgehenden CH-Dijkstra verwendet.

5.4. Ergebnisse

Für alle in 4.1 angegebenen Graphen wurde die CH berechnet. Alle Berechungen der Contraction Hierarchy der entsprechenden Graphen wurden parallelisiert mit 20 Threads auf einem Server mit 96GB Ram und 2 AMD Opteron 6100 (24 Kerne) durchgeführt. Alle Algorithmen wurden in Java 7 implementiert.

5.4.1. Größe der Contraction Hierarchy

Tabelle 5.1 stellt die Laufzeiten der Erstellung der Contraction Hierarchy und Anzahl der durch sie hinzugefügten Kanten dar.

Graph	Knoten	Kanten	Unkontrahierte Knoten	Laufzeit[s]
DE	19.478.240	39.454.253	19.478.240 (100%)	-
DE-CH	19.478.240	70.128.496	154.856(0,7%)	11.174
BW	2.911.711	5.903.801	2.911.711 (100%)	-
BW-CH	2.911.711	10.972.106	20.317(0,7%)	4.767
ST	924.688	1.876.030	924.688 (100%)	-
ST-CH	924.688	3.452.011	2.743 (0,7%)	2.408

Tabelle 5.1.: Ergebnisse und Laufzeiten der CH Erstellung

Da sich die Anzahl der Kanten im Vergleich zum Aufbau einer CH für das „shortest-path routing“ nicht deutlich erhöht haben [Sto12a][7.3], ist sowohl die Knotenauswahlheuristik als auch die Dominanzuntersuchung für Knoten ausreichend. Bei allen Graphen hat sich die Anzahl der Kanten weniger als verdoppelt. Die Kontraktion wurde abgebrochen, wenn die Anzahl der für einen Knoten hinzugefügten Kanten bei einem Kontraktionsvorgang nicht mehr mit dem Geschwindigkeitsgewinn der Kontraktion aufgewogen wurde.

5.4.2. Point to Point Querys

Da die Zeitgrenze hier unbestimmt und nur über einen Faktor definiert ist, wird vor jeder FCSP Query die maximale Zeit berechnet. Da die hier konstruierte CH alle Elemente der pareto-optimalen Lösungsmenge enthält, muss auch die schnellste Lösung enthalten sein. Durch die Existenz der kürzesten Lösung in der pareto-toptimalen Lösungsmenge ist es möglich, die identische CH zu verwenden, um mit einen normalen CH-Disjkstra die kürzeste Distanz zu berechnen.

Alle weiteren Berechnungen wurden auf einem Core i7 4770K (3.5 Ghz) mit 16GB Ram durchgeführt.

Graph	Mittlere Weglänge[s]	Mittlere Queryzeit[ms]	Mittlere Heap Abfragen
DE	10.316	4.700	10.500.000
DE-CH	10.316	162	287.100
BW	3950	504	1.489.000
BW-CH	3950	17	38.000
ST	2167	162	530.000
ST-CH	2167	4	6.900

Tabelle 5.2.: Point to Point Queryzeiten, Weglängen und Entfernungen aus dem Heap gemittelt über zufällige 10.000 Abfragen

Tabelle 5.2 stellt die Queryzeiten eines normalen Dijkstra's Algorithmus im Vergleich zu der beschleunigten CH-Variante dar. Trotz der groben Approximation der Kantendominanz und der Tatsache, dass im Vergleich zu einer normalen „shortest-path" CH mehr Kanten eingefügt werden, lassen sich doch aktzeptable „shortest-path" Queryzeiten erreichen. Die

6. Algorithmen

Nachdem die Contraction Hierarchy aufgebaut ist, wird nun das FCSP Problem mit den in Kapitel 4 eingeführten Kantenkostenfunktionen über eine Rückführung auf das CSP Problem gelöst. Um die praktische Anwendbarkeit zu verbessern, wird danach ein Heuristik eingeführt, der die Menge der möglichen Geschwindigkeiten einschränkt und dann mit diesen Geschwindigkeiten das FCSP annähert oder löst. Alle Berechnungen wurden auf einem Core i7 4770K mit 16GB Ram durchgeführt.

6.1. FCSP Algorithmus

Die Verwendung eines direkten FCSP-CH Algorithmus (siehe Algorithmus 6.1) ist auf Grund der Vielzahl an möglichen Wegen bereits bei kleineren Graphen und komplexen Kantenkostenfunktionen nicht möglich.

Es wird bereits bei dem kleinen Graphen des Regionalbezirks Stuttgart regelmäßig die Grenze von 100.000.000 Elementen im Heap überschritten. Bei der Verwendung von einfacheren Kantenkostenfunktionen und einer exakten Dominanzentscheidung sollte eine direkte FCSP Lösung dennoch möglich sein.

Algorithmus 6.1 FCSP-CH Disjkstra Modifikation

```
procedure FCSP-CH(V,E,Erev,s,t,t_max)
                                                  // Speicher für die pareto-optimalen Lösungsmengen
    List[] paretoUp ← new List[|V|]
    List[] paretoDown ← new List[|V|]
                                                  // Heap für Wege, sortiert nach minimaler Zeit
    Heap minWayHeap ← new Heap()
                                  // Wege im Heap sind definiert über (target, in, la, a, UP/DOWN)
    minWayHeap.Push(s,0,0,0,UP)
    minWayHeap.Push(t,0,0,0,DOWN)
    while |minWayHeap| > 0 do
        Way w ← minWayHeap.Pop
        if t_min(w) > t_max then
            Break
        end if
                                                  // Startseite des CH Dijkstra
        if w[4] = UP then
            for e(s,t,in,la,a) ∈ E : s = w[0] do
                // Setzen eines neuen Labels, wenn Weg Bestandteil der pareto-optimalen Lösung
                if !Dominated(e,paretoUp[w[0]]) then
                    paretoUp.Add(w[1],w[2],w[3])
                    minWayHeap.Push(e[1],e[2]+w[1],e[3]+w[2],e[4]+w[3],UP)
                end if
            end for
                                                  // Zielseite des CH Disjkstra
        else
            for e(s,t,in,la,a) ∈ Erev : s = w[0] do
                // Setzen eines neuen Labels, wenn Weg Bestandteil der pareto-optimalen Lösung
                if !DOMINATED(e,paretoDown[w[0]]) then
                    paretoDown.Add(w[1],w[2],w[3])
                    minWayHeap.Push(e[1],e[2]+w[1],e[3]+w[2],e[4]+w[3],DOWN)
                end if
            end for
        end if
    end while
    return BestSolution(paretoUp,paretoDown,t_max)
end procedure
```

Anstatt die FCSP Instanz direkt mit einem FCSP-CH Algorithmus zu lösen, ist es bei einer eingeschränkten, diskreten Parametermenge möglich, für jeden möglichen Parameter das CSP Problem zu lösen und die beste Lösung mit Parameter auszugeben. Da alle gültigen Geschwindigkeiten diskret und im Intervall zwischen 14 und 42 Meter pro Sekunde liegen, ist die Anzahl an möglichen Parametern diskret und beschränkt.

Der folgende Algorithmus 6.2 verwendet den CSP-CH Algorithmus 6.4, um eine passende FCSP Instanz zu lösen. Als t_{max} wird hier eine wählbare Zeitschranke verwendet.

Algorithmus 6.2 FCSP CSP Lösung

```
procedure FCSP(V,E,s,t,t_max,P)
                                          // L sei die Menge von gültigen Lösungen
                                          // P sei die Menge von gültigen Parametern
    L ← ∅
                                          // Finden aller Lösungen
    for all p ∈ P do
        L ← L∪CSP(V,E,s,t,t_max,p)
    end for
    result ← ∞
                                          // Auswahl der besten Lösung
    for all l ∈ L do
        if l < result then
            result ← l
        end if
    end for
    return result
end procedure
```

6.2. CSP Algorithmus

Ein sehr einfacher Algorithmus für die Lösung des CSP ist der sogenannte „label-setting“ Algorithmus, der eine Erweiterung von Dijkstra's Algorithmus darstellt und als Basis für den in Abschnitt 6.3 definierten Algorithmus dient. Zum Vergleich mit dem verwendeten CSP-CH Algorithmus wird hier eine Untersuchung der Laufzeit und der Heapabfragen auf verschiedenen Graphen durchgeführt. Der Algorithmus 6.3 ist eine leicht vereinfachte Darstellung der Implementierung.

Algorithmus 6.3 label setting Algorihmus

```
procedure LABEL(V,E,s,t,max)
                                          // Speicher für die pareto-optimalen Lösungsmengen
    int[] lastTime;
                                            // Heap für Wege, sortiert nach minimaler Energie
    Heap minWayHeap ← new Heap()
                                           // Wege im Heap sind definiert über (target, in, la, a)
    minWayHeap.PUSH(s,0,0,0)
    while |minWayHeap| > 0 do
        Way w ← minWayHeap.POP
        if w[0] = t then
            return w
        end if
        if lastTime[w[0]] >= time(w) then
            CONTINUE
        end if
        for e(s,t,in,la,a) ∈ E : s = w[0] do
            if !DOMINATED(e,pareto[w[0]]) then
                pareto.ADD(w[1],w[2],w[3])
                minWayHeap.PUSH(e[1],e[2]+w[1],e[3]+w[2],e[4]+w[3])
            end if
        end for
    end while
    return BESTSOLUTION(paretoUp,paretoDown,max)
end procedure
```

Eine Darstellung der Laufzeiten können in Abschnitt 6.3.3 gefunden werden. Leider ist dieser Algorithmus nicht ausreichend performant für viele Anwendungsgebiete.

6.3. CSP-CH Algorithmus

Für die partielle Lösung des FCSP Problems wird jedoch ein performanter Algorithmus zum Lösen des CSP Problems benötigt. Dieses Problem an sich ist natürlich weiterhin $\mathcal{NP}$-schwer, aber auf Grund der Eigenschaften eines Straßengraphen kommt diese Komplexität üblicherweise nicht in vollem Umfang zum Tragen. Der hier verwendete CSP-CH Algorithmus 6.4 ist eine Mischung aus einem CH-Dijkstra und dem label-setting Algorithmus, der entsprechend Wege statt Knotendistanzen verwendet.

Durch die Sortierung der Warteschlange (Heap) nach der Zeit eines Weges wird praktisch die pareto-optimale Lösungsmenge jedes Knoten von „links“ aus aufgebaut (vergleiche dazu Abbildung 2.1). Durch diese Sortierung muss der Weg w, der aus der Warteschlange gezogen wird, immer nur mit dem letzten Element der pareto-optimalen Lösungsmenge seines Zielknoten n verglichen werden: Da w auf jeden Fall langsamer ist als das letzte Element der Lösungsmenge von n, müsste er weniger Energie verbrauchen, um Bestandteil der pareto-optimalen Lösungsmenge von n zu sein.

Ein weiterer Vorteil der Sortierung nach der Zeit ist, dass auf dem höchsten Level der Contraction Hierarchy die Zeitgrenze dazu verwendet werden kann, den Suchradius deutlich zu verringern. Dazu wird an jedem Knoten die geringste Zeit gespeichert, die benötigt wird, um von dem Knoten zum Startpunkt s und zum Zielpunkt t zu gelangen. Die Zeit vom Startknoten wird durch die Aufwärtskomponente und die Zeit zum Zielknoten durch die Abwärtskomponente des CSP-CH Algorithmus definiert. Überschreitet die Summe der Zeiten eines Weg von s nach v und die Minimalzeit des Knoten v nach t die Zeitschranke, kann der Weg die Zeitschranke nicht mehr einhalten. Dies gilt identisch für die andere Komponente.

Abbildung 6.1 verdeutlicht die Verkleinerung des Suchraums: Ohne die oben genannte Abbruchbedingung müsste sowohl vom aufsteigenden wie auch vom absteigenden Teil des Algorithmus jeder Weg bis zur maximalen Zeitgrenze weitergeführt werden. So reduziert sich die Fläche des maximalen Suchraums um mehr als die Hälfte auf weniger als die beiden dunklen Kreise.

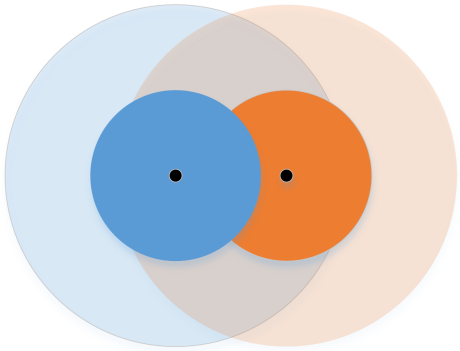


Abbildung 6.1.: Darstellung des maximalen Suchraums eines CSP-CH Algorithmus auf höchstem Level. In Hellblau und Hellorgange ist der maximale Suchradius einer Zeitgrenze von 1.2. Durch die Begrenzung konnte eine Verkleinerung des Suchraums um mehr als die Hälfte erreicht werden.

Algorithmus 6.4 CSP-CH Algorithmus

```
procedure CSP(V,E,Erev,s,t,max, v)
                                          // Energie des letzten Weges der pareto-optimalen Lösung, initialisiert mit ∞
    Int[] paretoUp ← new int[|V|]
    Int[] paretoDown ← new int[|V|]
                                          // Minimalzeit mit der ein Knoten erreicht werden Kann, initialiert mit −1
    Int[] timeUp ← new int[|V|]
    Int[] timeDown ← new int[|V|]
                                          // Heap für Wege, sortiert nach Zeit
    Heap minWayHeap ← new Heap()
                                          // Wege im Heap sind definiert über (target, in, la, a, UP/DOWN)
    minWayHeap.Push(s,0,0,0,UP)
    minWayHeap.Push(t,0,0,0,DOWN)
    while |minWayHeap| > 0 do
        Way w ← minWayHeap.Pop
        if c_t(w, v) > max then
            Break
        end if
                                          // Aufwärtskomponente des CH Dijkstra
        if w[4] = UP then
                                          // Kontrolle der Zeitschranke
            if level(w[0] = maximal ∧ c_t(w[4], v) + minDown[w[0]] > max then
                Continue
            end if
            if level(w[0] = maximal ∧ minDown[w[0]] = −1 ∧ 2 ∗ c_t(w, v) > max then
                Continue
            end if
                                          // Hinzufügen der neuen Wege in den Heap
            for e(s, t, in, la, a) ∈ E : s = w[0] do
                if lastUp[w[0]]>c_e(w, v) then
                    paretoUp.Add(w[1],w[2],w[3])
                    minWayHeap.Push(e[1],e[2]+w[1],e[3]+w[2],e[4]+w[3],UP)
                end if
            end for
                                          // Abwärtskomponente des CSP-CH Dijkstra
        else
                                          // Kontrolle der Zeitschranke
            if level(w[0] = maximal ∧ c_t(w[4], v) + minUp[w[0]] > max then
                Continue
            end if
            if level(w[0] = maximal ∧ minUp[w[0]] = −1 ∧ 2 ∗ c_t(w, v) > max then
                Continue
            end if
                                          // Hinzufügen der neuen Wege in den Heap
            for e(s, t, in, la, a) ∈ Erev : s = w[0] do
                if lastDown[w[0]]>c_e(w, v) then
                    paretoDown.Add(w[1],w[2],w[3])
                    minWayHeap.Push(e[1],e[2]+w[1],e[3]+w[2],e[4]+w[3],DOWN)
                end if
            end for
        end if
    end while
    return BestSolution(paretoUp,paretoDown,max,v)
end procedure
```

Die Funktion „BestSolution“ sucht aus der Menge der möglichen Lösungen die beste Lösung aus. Dazu wird entweder eine Liste an Knoten mit möglichen Lösungen gespeichert oder die Kombination aller Label aller Knoten durchgegangen.

6.3.1. Laufzeiten

Das folgende Experiment stellt die Queryzeiten der CSP Lösung in Abhängigkeit von ihrer Zeitschranke dar. Wie erwartet, steigt mit einer erweiterten Zeitschranke die Laufzeit des Algorithmus an, da die Abbruchsbedingung entsprechend später wirksam wird. Alle Berechnungen wurden mit 1.000, für Deutschland mit 100, zufällig ausgewählten Punktpaaren durchgeführt. Als Zeitschranke wird im folgenden immer ein Faktor verwendet, der die maximal mögliche Fahrzeit ausdrückt. Ein Faktor von 1,2 entspricht einem Zeitzuschlag von 20%. Es wird erwartet, dass eine Erhöhung der Zeitschranke zu einer Verlängerung der Laufzeit führt, da eine Erhöhung der Zeitschranke den maximalen Suchradius erhöht.

Graph	**Mittlere Queryzeit[ms] bei Zeitschranke**			
	$1,05$	$1,1$	$1,15$	$1,2$
DE-CH	252.992	328.027	392.766	570.691
BW-CH	5.674	7.995	13.357	24.914
ST-CH	447	662	1.015	1.532

Tabelle 6.1.: CSP-CH Queryzeiten bei 42 m/s, in Abhängigkeit der Zeitschranke, inklusive einer Distanzberechnung pro Query

Die in Tabelle 6.1 dargestellten Laufzeiten zeigen, dass die Lösung einer CSP Instanz auch auf Deutschland exakt möglich ist und die Laufzeit für viele Anwendungsfälle wie zum Beispiel der Logistikplanung ausreichend ist. Dennoch ist die Laufzeit für normale Routinganwendungen noch zu hoch.

6.3.2. Warteschlangenabfragen

Zur Bewertung eines Algorithmus, der auf Dijkstra's Algorithmus basiert, ist die Anzahl der Warteschlangenabfragen und die maximale Größe der Warteschlange wichtig. Hier wird die Anzahl der Warteschlangenabfragen, die durchschnittliche maximale Warteschlangengröße und die maximale Warteschlangengröße des ganzen Experiments dargestellt.

Graph	**Mittlere Warteschlangen Abfragen(% pareto-optimal)**			
	$1,05$	$1,1$	$1,15$	$1,2$
DE-CH	$1,2 \cdot 10^8 (25,2\%)$	$1,5 \cdot 10^8 (25,0\%)$	$1,9 \cdot 10^8 (24,7\%)$	$2,3 \cdot 10^8 (24,3\%)$
BW-CH	$6,6 \cdot 10^6 (23,0\%)$	$8,9 \cdot 10^6 (22,6\%)$	$1,2 \cdot 10^7 (22,1\%)$	$1,8 \cdot 10^7 (21,5\%)$
ST-CH	$7,4 \cdot 10^5 (28,7\%)$	$1,0 \cdot 10^6 (27,9\%)$	$1,4 \cdot 10^6 (27,0\%)$	$1,9 \cdot 10^6 (26,3\%)$

Tabelle 6.2.: Heap Abfragen und erfogreiche Wege bei 42 m/s

Graph	Maximale Warteschlangengröße (Mittlere maximale Warteschlangengröße)			
	$1,05$	$1,1$	$1,15$	$1,2$
DE-CH	$3,6 \cdot 10^7 (1,3 \cdot 10^7)$	$3,9 \cdot 10^7 (1,5 \cdot 10^7)$	$4,2 \cdot 10^7 (1,7 \cdot 10^7)$	$4,6 \cdot 10^7 (2,0 \cdot 10^7)$
BW-CH	$5,8 \cdot 10^6 (9,9 \cdot 10^5)$	$6,1 \cdot 10^6 (1,1 \cdot 10^6)$	$6,6 \cdot 10^6 (1,3 \cdot 10^6)$	$9,5 \cdot 10^6 (1,6 \cdot 10^6)$
ST-CH	$5,2 \cdot 10^5 (1,1 \cdot 10^5)$	$6,3 \cdot 10^5 (1,3 \cdot 10^5)$	$8,0 \cdot 10^5 (1,5 \cdot 10^5)$	$9,8 \cdot 10^5 (1,9 \cdot 10^5)$

Tabelle 6.3.: Maximale Warteschlangengrößen und mittlere maximale Warteschlangengrößen bei 42 m/s

Die in Tabellen 6.2 und 6.3 dargestellten Abfragen an die Warteschlange und die daraus resultierende Anzahl von Elementen in den pareto-optimalen Lösungsmengen erklären die hohe Laufzeit des Algorithmus. Selbst wenn es möglich wäre, bereits zu dem Zeitpunkt, bei dem ein Weg in den Heap eingefügt wird, zu wissen, ob der Weg Bestandteil einer pareto-optimalen Lösungsmenge ist, könnte dadurch die Laufzeit nicht wesentlich verringert werden.

Graph	Knoten mit pareto-optimalen Lösungsmengen (Durchschnittliche Größe)			
	$1,05$	$1,1$	$1,15$	$1,2$
DE-CH	$58.776(514)$	$62.591(599)$	$71.239(660)$	$74.630(750)$
BW-CH	$6.205(244)$	$6.698(300)$	$7.148(371)$	$7.564(511)$
ST-CH	$2.774(78)$	$2.942(95)$	$3.092(122)$	$3.229(155)$

Tabelle 6.4.: Anzahl der Knoten mit pareto-optimalen Lösungsmengen und Summe ihrer durchschnittlichen Größe

Die Anzahl der Knoten, die mindestens eine pareto-optimale Lösungsmenge haben und die durchschnittliche Größe dieser Lösungsmengen (dargestellt in Tabelle 6.4) zeigen die Komplexität des Problems. Durch die hohe Dichte der Lösungsmengen ergibt sich eine Großzahl an möglichen Wegen und damit eine höhere Laufzeit.

Der Vergleich mit einem strukturell ähnlichen Problem, Minimierung der Höhendifferenzen mit einer Distanzschranke [Sto12b], zeigt, dass sich die Anzahl der Heapabfragen und die Laufzeit in der gleichen Größenordnung befinden.

6.3.3. Laufzeiten und Heapabfragen im Vergleich

Zur Analyse des Laufzeitgewinns wurde auf verschiedenen Graphen ein Vergleich von verschiedenen CSP Lösungsverfahren durchgeführt. Als Basis dient für alle label-setting Algorithmen der Algorithmus 6.3 mit unterschiedlichen Heapsortierungen und Fragestellungen. Als Vergleich dazu wurden Berechnungen mit dem Algorithmus 6.4 angegeben. Auf eine Untersuchung der Algorithmen auf Deutschland musste verzichtet werden, da der verfügbare RAM nicht ausreichte, um den label-setting Algorithmus durchzuführen.

Graph	**label-setting**				**CSP-CH**	
Heapsortierung	nach Zeit		nach Energie		nach Zeit	
	Zeit[ms]	Abfragen	Zeit[ms]	Abfragen	Zeit[ms]	Abfragen
BW / BW-CH	331.143	$6,3 \cdot 10^8$	221.062	$4,6 \cdot 10^8$	17.911	$1,8 \cdot 10^7$
ST / ST-CH	27.010	$8,2 \cdot 10^7$	20.630	$6,0 \cdot 10^7$	1.730	$2,4 \cdot 10^6$
150K / 150K-CH	1.050	$3,6 \cdot 10^6$	750	$2,5 \cdot 10^6$	130	$8,0 \cdot 10^4$
15K / 15K-CH	681	$5,7 \cdot 10^4$	663	$4.6 \cdot 10^4$	119	$1.3 \cdot 10^3$

Tabelle 6.5.: label-setting und CSP-CH Queryzeiten bei 42 m/s für den energieeffizientesten Weg mit der Zeitschranke 1.2, inklusive einer Distanzberechnung pro Query, gemittelt über 1.000 (100 für BW/BW-CH) zufälligen Abfragen

Graph	**label-setting**			
Heapsortierung	nach Zeit		nach Energie	
	Zeit[ms]	Abfragen	Zeit[ms]	Abfragen
BW	207.396	$4,4 \cdot 10^8$	289.673	$7,2 \cdot 10^8$
ST	24.683	$5,4 \cdot 10^7$	36.874	$9.6 \cdot 10^7$
150K	737	$2,1 \cdot 10^6$	1.183	$3,9 \cdot 10^6$
15K	652	$4,0 \cdot 10^4$	673	$6.4 \cdot 10^4$

Tabelle 6.6.: label-setting Queryzeiten bei 42 m/s für den schnellsten Weg mit der Energieschranke 1.2, inklusive einer geringsten Energieberechnung pro Query, gemittelt über 1.000 (100 für BW) zufälligen Abfragen

Der in Tabelle 6.5 dargestellte Laufzeitvergleich verdeutlicht den Vorteil des CSP-CH Algorithmus. Sowohl die Queryzeiten als auch die Anzahl der Heapabfragen unterscheiden sich um mindestens eine Größenordnung zugunsten des CSP-CH Algorithmus. Aus den Ergebnissen in Tabelle 6.6 kann geschlossen werden, dass sich die Laufzeiten für eine Fragestellung nach dem kürzesten Weg mit einer Energieschranke und dem energieeffizientesten Weg mit einer Zeitschranke ähneln. Die Graphen 15K und 150K repräsentieren Ausschnitte aus Deutschland mit 14.951 beziehungsweise 149.303 Knoten und 30.734 bzw. 306.274 Kanten.

6.4. CSP-CH Heuristik

Bei kleinen Distanzen kann der Algorithmus 6.4 unverändert verwendet werden. Jedoch ist bei einer Laufzeit von mehr als 300 Sekunden pro CSP Instanz innerhalb von Deutschland die praktische Anwendbarkeit nur eingeschränkt gegeben. Akzeptiert man eine Verschlechterung der Genauigkeit, kann die Laufzeit des CSP-CH Algorithmus durch eine Heuristik deutlich verbessert werden. Hierzu verlangt man, dass der Energieverbrauch von jedem Element einer pareto-optimalen Lösungsmenge mindestens $x\%$ kleiner ist als der Energieverbrauch seines Vorgängers. Durch die Erhöhung der Schranke reduziert sich die Anzahl der möglichen Wege und damit die Laufzeit des Algorithmus. Wählt man $x = 0$, erhält man eine exakte Lösung der CSP-Instanz.

In Tabelle 6.7 wird nun die durchschnittliche Energieabweichung ΔE, die maximale Energieabweichung ΔE_{max} und das Laufzeitverhalten von verschiedenen Grenzen dargestellt. Auf Grund der hohen Laufzeiten des exakten CSP-CH Algorithmus wurde auf eine genaue Untersuchung von DE-CH verzichtet.

	ST-CH				**BW-CH**				**DE-CH**
Grenze	Queryzeit	ΔE	ΔE_{max}	ΔT	Queryzeit	ΔE	ΔE_{max}	ΔT	Queryzeit
1	1.500 (100%)	-	-	-	25.900 (100%)	-	-	-	570.691 (100%)
1,001	416 (28%)	0,009%	0,06%	-0.017%	4.100 (16%)	0,015%	0,08%	-0,016%	77.300 (14%)
1,005	146 (10%)	0,114%	0,91%	-0.174%	1.200 (5%)	0,152%	0,85%	-0,136%	20.200 (4%)
1,01	103 (7%)	0,300%	1,512%	-0.321%	692 (3%)	0,375%	1,83%	-0,405%	8.900 (2%)

Tabelle 6.7.: CSP Heuristik Queryzeiten[m/s] bei 42 m/s und Zeitgrenze 1,2, inklusive einer Distanzberechnung pro Query

Bereits durch Verwendung einer sehr genauen 1,005 Heuristik konnte die Laufzeit um einen Faktor 10, beziehungsweise 20 reduziert werden. Wie erwartet, liegt der Zeitverbrauch ΔT leicht unter der optimalen Lösung, während der Energieverbrauch leicht darüber liegt.

Um die Qualität und Verteilung der Heuristik bewerten zu können, wurden 1.000 Querys auf dem Graph ST-CH mit einer 0,5% Heuristik nach der Verteilung um das Optimum untersucht. Abbildung 6.2 stellt die Abweichung vom Optimum dar. Die Tabelle, der diese Abbildung zugrunde liegt, ist im Anhang A.2 zu finden.

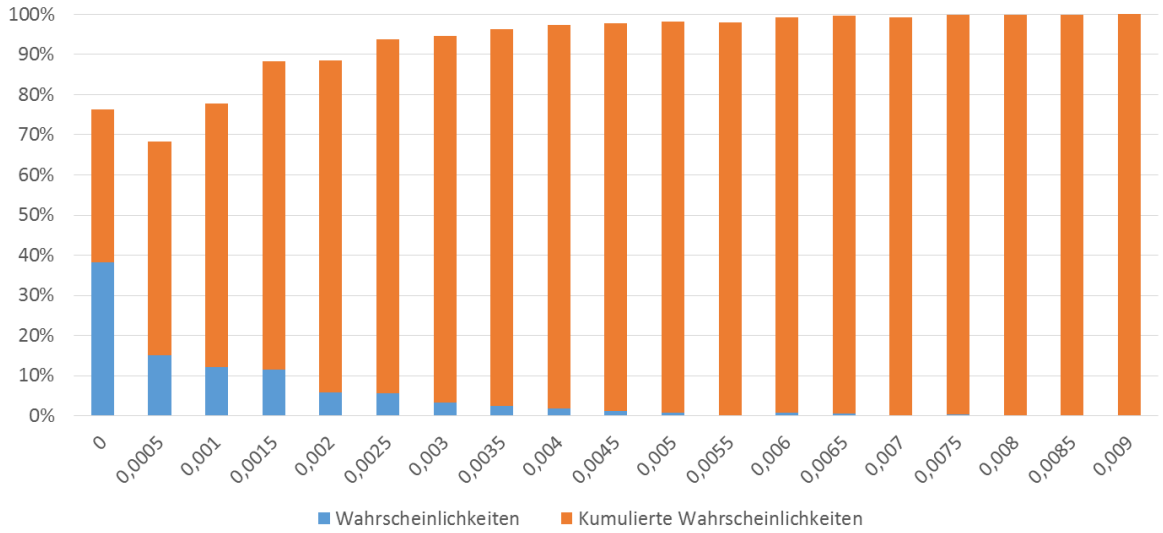


Abbildung 6.2.: Untersuchung der Verteilung der Ergebisse der Heuristik um das Optimum auf ST-CH mit 1.000 zufälligen Knotenpaaren

Die Verteilung zeigt, dass mehr als 98% aller Lösungen innerhalb der Grenze liegen, die für die pareto-optimalen Mengen definiert wurde und keine Lösung mehr als ein Faktor 2 von der Grenze abweicht. Die Qualität der Heuristik erfüllt damit die Anforderungen für viele praktische Anwendungen wie das Routing im Individualverkehr. Die wenigen Ausreißer, die außerhalb der Grenze liegen, können für solche Probleme oft ignoriert werden.

Da die Laufzeit der Heuristik auch von der maximalen Geschwindigkeit abhängt, weil diese ebenfalls den maximalen Suchradius erhöhen kann, wurden 1.000 CSP-CH Heuristikberechnungen für jede Geschwindigkeit zwischen 22 m/s und 48 m/s mit der Zeitgrenze 1,2 durchgeführt.

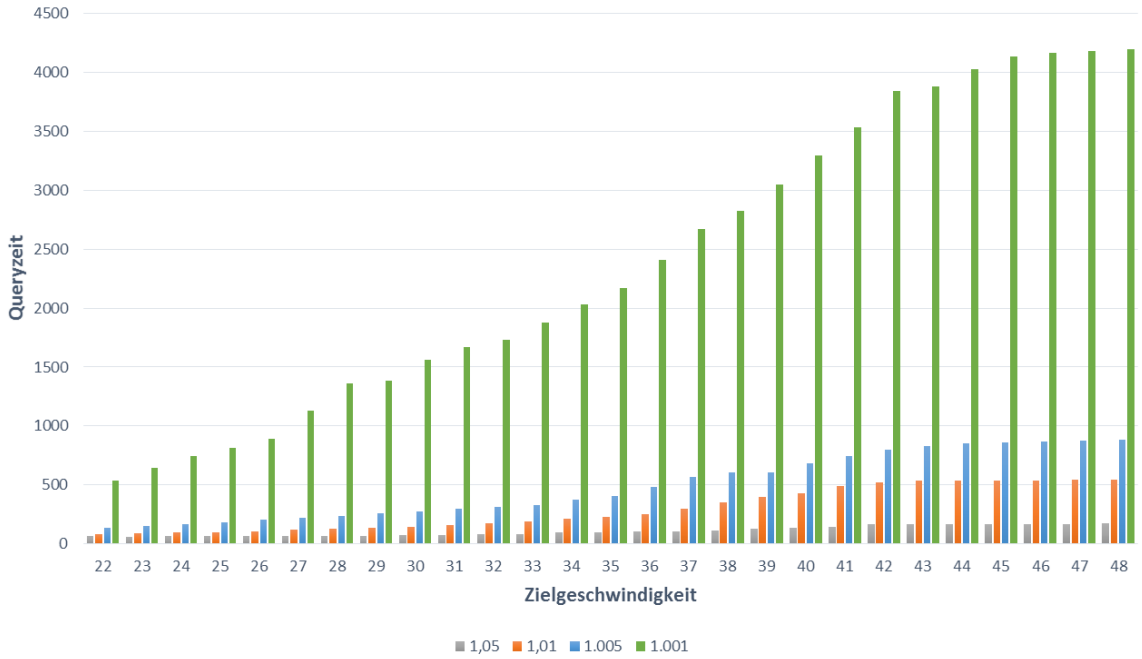


Abbildung 6.3.: Untersuchung der Queryzeit auf BW-CH mit 1.000 zufälligen Knotenpaaren mit verschiedenen Grenzen mit Distanzberechnung pro Query bei Zeitschranke 1,2

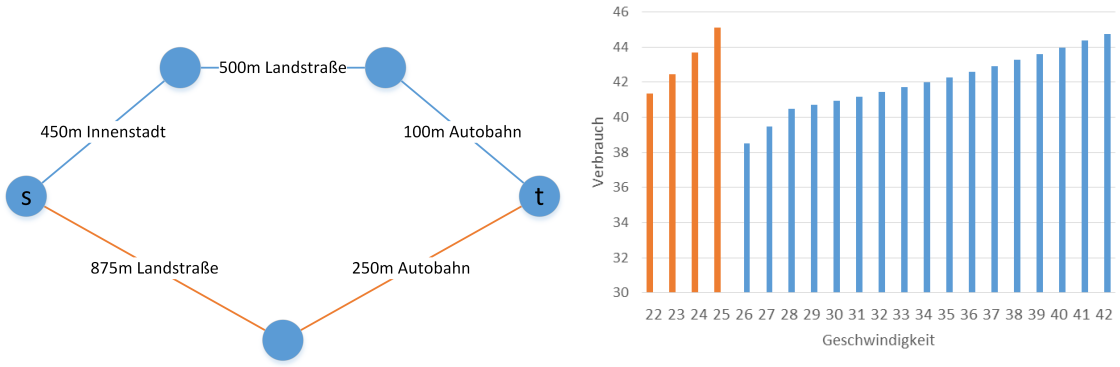
Die in Abbildung 6.3 dargestellten Laufzeiten unterscheiden sich wie erwartet recht deutlich. Je größer die Zielgeschwindigkeit ist, desto länger benötigt die Lösung der CSP Instanz, da bei einer niedrigeren Zielgeschwindigkeit die Zeitschranke schneller erreicht ist. Die genauen Werte können in tabellarischer Form im Anhang unter A.1.1 nachgeschlagen werden. Als Schätzwerte der Summe der Laufzeiten für jeden Parameter ergeben sich dann folgende Werte: 64.764 ms für Grenze 1,001, 13.215 ms für Grenze 1,005, 7.960 ms für Grenze 1,01 und 2.962 ms für Grenze 1,05.

Durch die Verwendung von gröberen Grenzen zum Ausschluss von Parametern wie auch der Begrenzung auf mögliche Wege wird die erwartete Laufzeit zur Lösung einer FCSP Instanz im weiteren Kapitel noch deutlich verbessert.

6.5. FCSP Lösung mit Hilfe der CSP Heuristik

Je nach Grenze kann direkt für alle möglichen Geschwindigkeiten das CSP Problem gelöst werden. Leider kann der Bereich der gültigen Parameter nicht durch eine Binärsuche eingeschränkt werden, wie Abbildung 6.4 zeigt.

Jeder Weg führt zwar mit abnehmender Geschwindigkeit zu einem niedrigeren Energieverbrauch, aber sobald die Fahrzeit durch die abnehmende Geschwindigkeit die Zeitgrenze überschreitet, ist der Weg nicht mehr gültig und ein anderer, eventuell energieintensiverer Weg, stellt das Optimum für diese Geschwindigkeit dar. Der Energieverbrauch ist im Verhältnis zur Zielgeschwindigkeit dementsprechend nicht monoton steigend.



(a) Straßengraph **(b)** Energieverbrauchsdiagramm

Abbildung 6.4.: Beispiel eines nicht monotonen Energieverbrauchs für zwei Wege (450 500 100), (0 875 250) und einer Zeitschranke von 55 Sekunden

Wird jedoch eine hohe Genauigkeit oder eine exakte Lösung benötigt, lassen sich dennoch gute Laufzeiten durch den Ausschluss von Geschwindigkeiten mit einer gröberen Grenze erreichen, indem durch eine Heuristik mögliche Fahrgeschwindigkeiten praktisch ausgeschlossen werden.

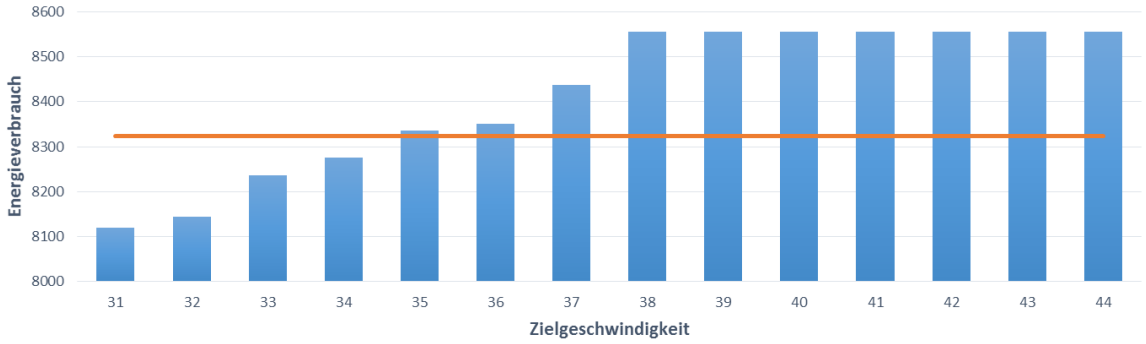


Abbildung 6.5.: Beispiel einer Lösungsmenge mit einer 2,5% Heuristik, welche zu einer Begrenzung der möglichen Parametermenge auf 4 Geschwindigkeiten(31-34) führt. Geschwindigkeiten unter 31 m/s führten zu keiner Lösung.

Das Beispiel aus Abbildung 6.5 verdeutlicht, wie eine grobe Abschätzung verwendet werden kann, um Geschwindigkeiten auszuschließen. Man betrachtet die beste Lösung als Richtwert und schließt über die Heuristik Geschwindigkeiten aus, die nicht oder nur extrem unwahrscheinlich zu einer optimalen Lösung führen können.

Der Algorithmus 6.5 verwendet eine $2,5\%$ Heuristik als Grenze, um die Menge der potentiellen Geschwindigkeiten so weit wie möglich zu reduzieren und berechnet dann mit der Menge der potentiellen Geschwindigkeiten die Lösung mit der gewünschten Heuristik.

Algorithmus 6.5 FCSP Heuristik

```
procedure FCSPHEUR(V,E, Erev, s,t,max,heur)
                                                    // Berechnen der Lösungsmenge
    for v = 48 → 22 do
        w ←CSPHEUR_CH(V,E,Erev,s,t,max, v,1.025)
        if VALID(W) then BREAK
        end if
        L ← L ∪ (v, w)
    end for
                                                    // Kleinste Lösung finden
    min ←MIN(L)
                                                    // Lösungen außerhalb der Grenze aussortieren
    for all l ∈ L do
        if l[1] > min ∗ 1.025 then
            L ← L \ l
        end if
    end for
                                                    // Mögliche Lösungen entsprechend der Grenze berechnen
    for all l ∈ L do
        if l[1] > min ∗ 1.025 then
            w ←CSPHEUR_CH(V,E,Erev,s,t,max, l[0],heur)
            R ← R ∪ (l[0], w)
        end if
    end for
                                                    // Beste Lösung ausgeben
    return MIN(R)
end procedure
```

Der Algorithmus beginnt bei der maximalen Zielgeschwindigkeit von 48 m/s und prüft alle Geschwindigkeiten, bis kein Weg gefunden werden kann, der die Zeitschranke einhält. Alle langsameren Geschwindigkeiten als diese Zielgeschwindigkeit können ebenfalls nicht mehr zu gültigen Wegen führen, da diese Wege sonst mindestens genauso schnell von einer schnelleren Zielgeschwindigkeit gefahren werden könnten und damit zu einer Lösung führen würden, die diese Zeitschranke einhält.

6.5.1. Ergebnisse und Fazit

Der wie oben angegeben modifizierte Algorithmus wird nun verwendet, um die energieeffizienteste Distanz und Geschwindigkeit zwischen zwei Punkten zu ermitteln. Die folgenden Experimente verdeutlichen den Einfluss der für den Ausschluss von möglichen Parametern verwendeten Grenze. Die verbleibenden Parameter sind die nach der Vorauswahl verbleibenden Parameter, für die eine genaue Untersuchung durchgeführt werden muss. In Tabelle 6.8 sind die untersuchten Parameter für den Graph BW-CH dargestellt. Eine vollständige Auflistung der Resultate ist im Anhang unter A.1.3 zu finden.

Heuristik	Grenze	Queryzeit (davon Grenze)	Verbleibende Parameter
1,005	1,05	13,4 (4,6)	7,4
1,005	1,025	13,1 (6,2)	4,8
1,005	1,01	16,7 (13,0)	3,6
1,005	-	20,0	-
1,001	1,05	32,3 (4,3)	7,4
1,001	1,025	24,9 (6,2)	4,8
1,001	1,01	27,7 (12,1)	3,6
1,001	-	81,8	-

Tabelle 6.8.: BW-CH FCSP Heuristik über 1.000 zufällige Punktpaare

Dieses Ergebnisse lassen den Schluss zu, dass die entsprechende Grenze in Abhängigkeit der entsprechenden Heuristik zu wählen ist. Je mehr Zeit auf die finalen Berechnungen verwendet wird, desto mehr Zeit sollte auch auf die Auswahl der potentiellen Parameter verwendet werden. Auch ist bei steigender Größe des Graphen die Verwendung der FCSP Heuristik im Gegensatz zu einer Lösung ohne vorherige Auswahl der möglichen Parameter zunehmend wichtig.

Abschließend lässt sich sagen, dass das Problem des energieoptimalen Routing mit zeitlichen Nebenbedingungen und variabler Geschwindigkeit durch die hier vorgestellten Algorithmen und Verfahren in praktikabler Zeit lösbar ist. Da speziell der FCSPapprox Algorithmus gut parallelisierbar ist, könnte dadurch noch eine deutliche Verbesserung der Laufzeit erzielt werden, sodass es durchaus realisch ist, das Routingproblem innerhalb Deutschlands in unter 20 Sekunden zu lösen.

7. Zusammenfassung und Ausblick

Durch die Erweiterung des Constrained Shortest Path Problem zum Functional Constrained Shortest Path Problem wurde eine Möglichkeit geschaffen, auch kompliziertere Fragestellungen mit weiteren Parametern wie zum Beispiel der Geschwindigkeit oder dem Gewicht direkt darzustellen.

In Verbindung mit dem Nachweis, dass die Beschleunigungstechnik der „Contraction Hierarchy“ bei speziellen Kantenkostenfunktionen auch auf das FCSP anwendbar ist, wurde eine Vorgehensweise entwickelt, die dazu verwendet werden kann, FCSP Instanzen, wie zum Beispiel das energieoptimierte Routing mit zeitlichen Nebenbedingungen und variabler Geschwindigkeit in praktikabler Zeit zu lösen.

Durch die Anwendung einer Heuristik, die mit dem Beschleunigungsverfahren kompatibel ist, konnten noch schnellere Laufzeiten realisiert werden.

Ausblick

Durch die fortschreitende Entwicklung in der Fahrzeugtechnik und Robotik wird sich in mittelbarer Zeit durch das autonome Fahren ein Umbruch in der Mobilität vollziehen.

In der Zukunft wird der Fahrer nur noch ein gewünschtes Fahrziel und eine Ankunftszeit eingeben und dem Fahrzeug die Routen- und Geschwindigkeitswahl überlassen. Diese Möglichkeit könnte genutzt werden, um den weltweiten Energieverbrauch des Individualverkehrs durch ein energiesparendes Routing zu senken.

Die in der vorliegenden Diplomarbeit entwickelten Grundlagen können bei der Entwicklung solcher Navigationssysteme dazu beitragen, dieses Problem exakt oder mit einer guten Heuristik zu lösen.

Wissenschaftlich kann die Erweiterung des Constrained Shortest Path Problem zum Functional Constrained Shortest Path Problem dazu genutzt werden, auch andere Probleme mit komplexeren Kantenkostenfunktionen oder kontinuierlichen Parametermengen darzustellen.

A. Anhang

A.1. Tabellen

A.1.1. Laufzeitanalyse Zielgeschwindigkeiten auf BW-CH

Geschwindigkeit	1,001	1,005	1,01	1,05
22	537	135	84	64
23	645	150	89	63
24	742	168	95	65
25	817	183	100	64
26	895	204	109	65
27	1.133	223	120	67
28	1.364	238	132	71
29	1.383	261	139	71
30	1.563	279	145	74
31	1.671	299	158	77
32	1.728	314	173	81
33	1.877	332	188	85
34	2.032	377	214	96
35	2.173	404	230	97
36	2.406	481	255	106
37	2.674	565	298	109
38	2.825	606	356	116
39	3.046	610	395	126
40	3.296	681	426	135
41	3.533	746	490	147
42	3.838	796	523	166
43	3.883	832	534	168
44	4.028	855	540	171
45	4.134	857	541	169
46	4.165	865	535	169
47	4.181	873	547	168
48	4.195	881	544	172

Tabelle A.1.: Heuristikslaufzeiten auf BW-CH

A.1.2. Heuristik Verteilung

Bereich	Anzahl
0% - 0,0005%	382
0,0005% - 0,0010%	151
0,0010% - 0,0015%	125
0,0015% - 0,0020%	114
0,0020% - 0,0025%	58
0,0025% - 0,0030%	55
0,0030% - 0,0035%	32
0,0035% - 0,0040%	24
0,0035% - 0,0040%	18
0,0035% - 0,0040%	11
0,0040% - 0,0045%	8
0,0045% - 0,0050%	2
0,0050% - 0,0055%	8
0,0055% - 0,0060%	3
0,0060% - 0,0065%	1
0,0065% - 0,0070%	3
0,0070% - 0,0075%	2
0,0075% - 0,0080%	1
0,0080% - 0,0085%	2

Tabelle A.2.: 1,005 Heuristikverteilung auf ST-CH über 1000 Querys

A.1.3. FCSP Heuristikalgorithmus

Heuristik	Grenze	Queryzeit (Grenze)	verbleibende Parameter
1,005	1,05	1,3 (1,3)	5,8
1,005	1,025	1,8 (1,5)	2,9
1,005	1,01	2,3 (2,1)	2,2
1,005	-	2,8	-
1,001	1,05	2,9 (1,3)	5,8
1,001	1,025	2,1 (1,5)	2,9
1,001	1,01	2,7 (2,1)	2,2
1,001	-	8,9	-

Tabelle A.3.: ST-CH FCSP Heuristiklösung über 1.000 zufällige Punktpaare

Heuristik	Grenze	Queryzeit (Grenze)	verbleibende Parameter
1,005	1,05	13,4 (4,6)	7,4
1,005	1,025	13,1 (6,2)	4,8
1,005	1,01	16,7 (13,0)	3,6
1,005	-	20,0	-
1,001	1,05	32,3 (4,3)	7,4
1,001	1,025	24,9 (6,2)	4,8
1,001	1,01	27,7 (12,1)	3,6
1,001	-	81,8	-

Tabelle A.4.: BW-CH FCSP Heuristiklösung über 1.000 zufällige Punktpaare

Heuristik	Grenze	Queryzeit (davon Grenze)	verbleibende Parameter
1,005	1,05	106,7 (30,3)	7,6
1,005	1,025	80,7 (49,1)	4,4
1,005	1,01	126,2 (109,6)	2,1
1,005	-	196,0	-
1,001	1,05	327,2 (30,3)	7,6
1,001	1,025	175,6 (49,1)	4,4
1,001	1,01	325,3 (109,6)	2,1
1,001	-	1009,8	-

Tabelle A.5.: DE-CH FCSP Heuristiklösung über 100 zufällige Punktpaare

Literaturverzeichnis

[BDSV09] G. V. Batz, D. Delling, P. Sanders, C. Vetter. Time-dependent contraction hierarchies. In *IN PROC. 11TH WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS (ALENEX*, S. 97–105. SIAM, 2009. (Zitiert auf Seite 38)

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. URL http://dx.doi.org/10.1007/BF01386390. 10.1007/BF01386390. (Zitiert auf Seite 37)

[EFS11] J. Eisner, S. Funke, S. Storandt. Optimal Route Planning for Electric Vehicles in Large Networks, 2011. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3637. (Zitiert auf Seite 38)

[FUB09] Kfz energetisch betrachtet, 2009. URL http://www.chemie.fu-berlin.de/chemistry/general/kfz-energetisch.html. (Zitiert auf Seite 30)

[Gei08] R. Geisberger. *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*. Diplomarbeit, Institut für Theoretische Informatik Universität Karlsruhe (TH, 2008. (Zitiert auf den Seiten 37 und 38)

[GSSD08] R. Geisberger, P. Sanders, D. Schultes, D. Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th International Conference on Experimental Algorithms*, WEA'08, S. 319–333. Springer-Verlag, Berlin, Heidelberg, 2008. URL http://dl.acm.org/citation.cfm?id=1788888.1788912. (Zitiert auf den Seiten 13, 37 und 39)

[Hen86] M. I. Henig. The shortest path problem with two objective functions. *European Journal of Operational Research*, 25(2):281 – 291, 1986. doi:http://dx.doi.org/10.1016/0377-2217(86)90092-5. URL http://www.sciencedirect.com/science/article/pii/0377221786900925. (Zitiert auf Seite 17)

[HZ80] G. Y. Handler, I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980. URL http://dblp.uni-trier.de/db/journals/networks/networks10.html. (Zitiert auf Seite 24)

[Sto12a] S. Storandt. Quick and Energy-efficient Routes: Computing Constrained Shortest Paths for Electric Vehicles. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '12, S. 20–25. ACM, New York, NY, USA, 2012. doi:10.1145/2442942.2442947. URL http://doi.acm.org/10.1145/2442942.2442947. (Zitiert auf Seite 42)

[Sto12b] S. Storandt. Route Planning for Bicycles — Exact Constrained Shortest Paths Made Practical via Contraction Hierarchy, 2012. URL http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4502. (Zitiert auf Seite 53)

Alle URLs wurden zuletzt am 17. 03. 2008 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift