Institut für Formale Methoden der Informatik
Abteilung Algorithmik
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 19

Beschleunigte Berechnung von ressourcenbeschränkten kürzesten Wegen

Peter Vollmer

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Stefan Funke
Betreuer:	Dipl.-Biomath. Sabine Storandt
begonnen am:	14. Juni 2012
beendet am:	14. Dezember 2012
CR-Klassifikation:	G.2.2, G.1.6

Kurzfassung

Die Lösung NP-schwerer Probleme wie die ressourcenbeschränkten kürzesten Wege Berechnungen ist zur Zeit exakt nicht in akzeptabler Zeit möglich. Bisher lassen sich akzeptable Lösungen nur durch Abstriche im Hinblick auf den optimalen Pfad und lange Berechnungszeiten finden. In dieser Arbeit behandeln wir, wie durch Vorberechnung einer Contraction Hierarchy eine Beschleunigung einer Lösungsheuristik für ressourcenbeschränkte kürzeste Wege Berechnungen erreicht werden kann. Dazu haben wir ein Werkzeug erstellt, mit dem man die Vorberechnung vornehmen kann. Anschließend wurde auf den erstellten CH-Graphen getestet, wie erfolgreich die Beschleunigung ist. In unseren Messung wir, dass sich die Antwortzeiten um den Faktor 141-248 beschleunigen lassen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Während des Softwaretechnikstudiums wurden wir mit aktuellen Problemen der Informatik konfrontiert. Dies geschah beispielsweise in Studienprojekten, an denen 5-10 Studenten mitarbeiten. Dabei wird ein kompletter Projektzyklus durchlaufen, wodurch die Studenten Erfahrungen in der Projektarbeit sammeln können. In unserem Fall ging es um das Projekt "ToureNPlaner"[1], das als Zielsetzung die Erstellung eines Tourenplaners hatte, welcher nicht nur einfach kürzeste Wege Probleme lösen sollte, sondern auch NP-schwere Probleme wie das "ressourcenbeschränkte kürzeste Wege Problem" (CSP) oder das "Problem des Handelsreisenden". Unsere Aufgabe dabei war es, eine approximierte Lösungsheuristik für den CSP zu integrieren. Bei anschließenden Tests fiel auf, dass diese Heuristik ziemlich langsam ist. An dieser Stelle wurde unser Interesse geweckt, eine schnellere Heuristik zu integrieren.
Deshalb haben wir uns dafür entschieden, in dieser Bachelorarbeit zu untersuchen, in wieweit eine Beschleunigung mit einer *Contraction Hierarchy* (CH) für unsere Heuristik möglich ist. Zum besseren Verständnis haben wir in Kapitel 2 als Hintergrund die verwendeten Techniken erklärt. Anschließend werden die Voraussetzungen definiert, die erfüllt sein müssen, dass eine Vorberechnung möglich wird. Um einen größeren Überblick über die Ergebnisse der Berechnungen zu bekommen, werden in Kapitel 4 die Kontraktionszeiten der Graphen untersucht. Zur Untersuchung der Beschleunigung werden auf unterschiedlichen Graphen die Suchräume und die Antwortzeiten des CSP mit denen des CSPCH verglichen und schließlich zusammengefasst.

[1] http://tourenplaner.github.com/

2 Hintergrund

2.1 Das kürzeste Wege Problem und Dijkstras Algorithmus

Das kürzeste Wege Problem beschreibt die Suche nach dem kürzesten bzw. schnellsten Weg zwischen zwei oder mehreren Punkten (geordnet nach Besuchsreihenfolge) auf einem Straßennetz. Die formale Beschreibung und Lösung geschieht durch ein Optimierungsproblem auf einem Graphen:
dabei sei der Graph $G = (V, E)$, V eine endliche Menge von Knoten, E eine endliche Menge von Kanten und für E gilt zusätzlich das $E \subset V^2$. Außerdem definieren wir eine Funktion für die Kantenkosten (Kosten) $c : E \mapsto \mathbb{R}_0^+$, weil man davon ausgehen kann, dass gilt $\forall e \in E : c(e) \geq 0$, da für alle kürzesten bzw. schnellsten zurückgelegten Wege anzunehmen ist, dass sie nicht negative Kosten haben.
Unser Problem, den kürzesten bzw. schnellsten Weg π, von einem Startknoten(s) zu einem Zielknoten(t), zu finden, lässt sich nun als Minimierung über der Summe $\Sigma_{e \in \pi} c(e)$ darstellen.
Dijkstras Algorithmus (Dijkstra) ist einer der populärsten Algorithmen zur Bestimmung von kürzesten Wegen. Er arbeitet dabei auf Kanten, die nicht negative Kosten haben. Dabei lehnt sich der Dijkstra an der Tiefen-Suche und der Breiten-Suche an benutzt jedoch eine Prioritätswarteschlange U, anstatt der sonst üblichen Listen bzw. Kellern. U hält die unerschlossenen Knoten aufsteigend ihrer Distanz zu s und hält somit eine Art Frontlinie zu der Menge schon erschlossener Knoten. Um dies bewerkstelligen zu können, speichern wir für jeden Knoten $v \in V$ die minimale Distanz $d(v)$ von s nach v und speichern für jedes v den Vorgänger $p(v)$.
Zum Start vom Dijkstra setzen wir: $\forall v \neq s \in V : d(v) = \infty$, $d(s) = 0$, $p(v) = null$ und $U = \{s\}$. Von nun an wird in jedem Schritt aus U der Knoten v mit der niedrigsten $d(v)$ entnommen. Anschließend wird für jede ausgehende Kante $e(v, w) \in E$ überprüft, ob $d(v) + c(e) < d(w)$ ist. Ist das der Fall, so wird $d(w) := d(v) + c(e)$, w wird zu U hinzugefügt und $p(w) := v$. Dies geschieht, solange bis die Warteschlange leer ist. Sollte man nur den kürzesten Weg zu einem Ziel t suchen, reicht es, dass t aus U entnommen wurde. Denn wird ein Knoten v aus U entnommen, ist seine d(v) fertig berechnet.[Dij59] Anschließend kann man den Dijkstra beenden. Die Laufzeit des Dijkstra ist $O(nlog(n) + m)$. Dies ist aber nur möglich, wenn U als Fibonacci Heap strukturiert ist.[Boy96]

2.2 Das ressourcenbeschränkte kürzeste Wege Problem und eine Lösungsheuristik

Die folgende Darstellung der zu Grunde liegenden Probleme und Algorithmen folgt der aus [FS13].

Das ressourcenbeschränkte kürzeste Wege Problem (CSP) ist wie folgt charakterisiert: gegeben sei ein Graph G(V,E), eine Kostenfunktion $c : E \mapsto \mathbb{R}^+$ und eine Ressourcenverbrauchsfunktion $r : E \mapsto \mathbb{R}^+$ auf den Kanten. Eine Anfrage ist durch Anfangs- und Endknoten $s, t \in V$ und außerdem durch eine Ressourcengrenze spezifiziert. Das Ziel ist es, den Pfad mit den niedrigsten Kosten von s nach t zu finden, wobei die Ressourcengrenze nicht überschritten wird.
Die exakte Berechnung der Lösung ist NP-schwer und ist daher unter der Annahme P != NP nicht in akzeptabler (polynomieller) Zeit lösbar. Da auch in den meisten Anwendungsfällen eine approximierte, aber dafür schnell berechenbare, Lösung reicht, beschränken wir uns daher auf eine Heuristik, welche nur die Pfade betrachtet, die optimal sind für Kantengewichte $\lambda c(p) + (1 - \lambda)r(p)$, für ein gewisses $\lambda \in [0, 1]$. Diese Heuristik liefert eventuell für keine Wahl von λ die optimale Lösung.
Bei dieser Heuristik geht man mit einer binären Suche über das λ, um damit sich der bestmöglichen Lösung anzunähern. Zu Beginn werden dazu die Pfade für die Initial-Grenzen berechnet. Der Pfad mit $\lambda = 1$ entspricht dabei dem kürzesten bzw. schnellsten Pfad. Sollte der Ressourcenverbrauch dieses Pfads schon kleiner oder gleich der Ressourcengrenze sein, haben wir die Lösung gefunden und können die Heuristik beenden. Wenn nicht, überprüfen wir, ob für den Pfad mit $\lambda = 0$, der dem kürzesten bzw. schnellsten Pfad bezüglich des Ressourcenverbrauch entspricht, gilt, dass der Ressourcenverbrauch gleich der Ressourcengrenze ist. Sollte das so sein, haben wir unsere Lösung gefunden und können die Heuristik beenden. Sollte der Ressourcenverbrauch allerdings größer als die Ressourcengrenze sein, gibt es keinen Pfad, der mit seinem Ressourcenverbrauch unter unserer Ressourcengrenze liegen kann und wir können die Heuristik beenden. Sollte der Ressourcenverbrauch allerdings unter der Ressourcengrenze liegen, beginnen wir mit der binären Suche.
Dafür wird für jeden Iterationsschritt das neue λ berechnet mit: $\lambda = \frac{ObereGrenze + UntereGrenze}{2}$.
Danach wird jeweils der Pfad mit dem neuen λ berechnet und überprüft, ob dessen Ressourcenverbrauch gleich der Ressourcengrenze ist. Ist das so, so haben wir eine Näherungslösung gefunden und können die Heuristik beenden. Ist der Ressourcenverbrauch kleiner als die Ressourcengrenze, so wird die untere Grenze der Suche auf das aktuelle λ gesetzt. Ist der Ressourcenverbrauch allerdings größer als die Ressourcengrenze, so wird die obere Grenze der Suche auf das aktuelle λ gesetzt. Anschließend folgt die nächste Iteration.
Sollte man die Heuristik nun so ausführen, würde man häufig auf ein Terminierungsproblem stoßen, denn häufig findet man nicht das λ bei dem der Ressourcenverbrauch des Pfads gleich der Ressourcengrenze ist. Für eine weitere Abbruchbedingung kann man hierfür ein beliebig kleinen Abstand wählen, bei dem, sollte der Abstand der Grenzen kleiner oder gleich diesem Abstand sein, die Heuristik beendet wird. Als Lösung kann dann der letzte Pfad, dessen Ressourcenverbrauch unter der Ressourcengrenze liegt, dienen.

2.3 Contraction Hierarchy(CH)

Die folgende Darstellung der zu Grunde liegenden Probleme und Algorithmen folgt der aus [FS13] und [GSSD08]. Die Grundidee von Contraction Hierarchy ist, zusätzliche Kanten(Shortcuts) in einen Graphen einzufügen, die es erlauben, bei Anfragen mehrere Knoten zu überspringen und damit die Anfragen zu beschleunigen, da weniger Knoten angeschaut werden müssen.
Das Erstellen der Shortcuts passiert in der Vorberechnung und zwar in Leveln. Für jedes Level wird aus dem Graphen ein unabhängiges Set gewählt und alle Knoten darin kontrahiert. Jeder in einem Level kontrahierte Knoten bekommt die Level Nr. zugewiesen. Kontrahiert man nun einen Knoten v, muss für jeden Pfad uvw gelten, dass die Distanz von u nach w vor und nach der Kontraktion gleich ist. Von nun an muss die Kante $e = (u, w)$ hinzugefügt werden, sollte der kürzeste Weg von u nach w uvw sein. Die Kosten von e sind die Verkettung der Kosten von (u, v) und (v, w). Existiert allerdings ein Pfad von u nach w, dessen Kosten kleiner denen von uvw sind, ein sogenannter *Witnesspath*, so wir dieser Shortcut nicht hinzugefügt. Gefunden wird so ein Witnesspath typischerweise mit einem Dijkstra von u nach w. Nach der Kontrahierung aller Knoten in einem Level, werden die erstellten Shortcuts zum Graphen hinzugefügt und darauf das nächste Level gerechnet. Dies geschieht solange, bis keine Knoten mehr vorhanden sind. Aus den gewonnenen Shortcuts und G wir ein neuer Graph G' abgeleitet, welcher damit alle ursprünglichen Knoten und Kanten und zusätzlich die neuen Shortcuts enthält. Eine Kante $e = (v, w)$, egal ob Shortcut oder nicht, wird als aufwärts bezeichnet, sollte v ein kleineres Level besitzen als w. Andernfalls wird e als abwärts bezeichnet. In G' existiert für jedes Paar von Knoten s und t ein kürzester Pfad von s nach t, der in einen Aufwärts- und einen Abwärtspfad geteilt werden kann.
S-t-Anfragen lassen sich durch die Möglichkeit dieser Unterteilung bidirektional beantworten. Dies geschieht, indem man von s ausgehend, nur ausgehende Aufwärtskanten betrachtet und von t ausgehend rückwärts nur eingehende Abwärtskanten betrachtet.
Die Kontraktion muss nicht bis zum Ende gerechnet werden um Korrektheit zu erhalten. Dann muss allerdings für alle nicht kontrahierten Knoten das Level auf ∞ gesetzt werden. Dadurch bleibt die aufwärts- abwärts Eigenschaft der optimalen Pfade und der Einsatzmöglichkeit eines bidirektionalen Dijkstra in CH-Graphen erhalten. Unter einem vorzeitigem Abbruch leidet jedoch die Effizienz der bidirektionalen Dijkstra.

3 Anwendung von CH auf CSP Heuristik

Um die Beschleunigung der CSP-Heuristik zu erreichen, wollen wir die Beschleunigungstechnik der Contraction Hierarchies anwenden. Dazu wird bei der Kontraktion von einem Knoten v, für alle Pfade uvw überprüft, ob uvw ein möglicher kürzester Weg ist. Dazu nutzen wir ein an die CSP-Heuristik angelehntes Verfahren, um für einen Pfad uvw ausschließen zu können, dass ein kürzester Weg über v geht. Sollte nun doch ein kürzester Weg von u nach w über v gehen, wird der Shortcut (u, w) zum neuen Graphen hinzugefügt. Dies wird solange gemacht, bis alle Knoten kontrahiert wurden. Analog zu 2.3 kann auch hier die Kontraktion schon vorzeitig beendet werden, dann muss allerdings das Level für nicht kontrahierte Knoten auf ∞ gesetzt werden. Auch hier kann eventuell die Effizienz leiden.

3.1 Überprüfung der Notwendigkeit eines Shortcuts

Das Problem ist es, zu entscheiden, ob ein Shortcut benötigt wird oder nicht. Um dieses Problem exakt zu lösen, müssten wir für sämtliche λ überprüfen, ob ein Shortcut benötigt wird oder nicht. Durch eine geschickte Wahl von λ können wir mit nur endlich vielen λ-Berechnungen die exakte Entscheidung treffen.

Bei der Entscheidung, ob ein Shortcut benötigt wird oder nicht, treten mehrere Fälle auf; Der erste Fall (vgl. Abbildung 3.1) ist, dass der Referencepath uvw dem Witnesspath entspricht. Sollte das so sein, so existiert ein λ, für das gilt, dass ein kürzester Weg über unseren Referencepath geht und somit als Shortcut benötigt wird.

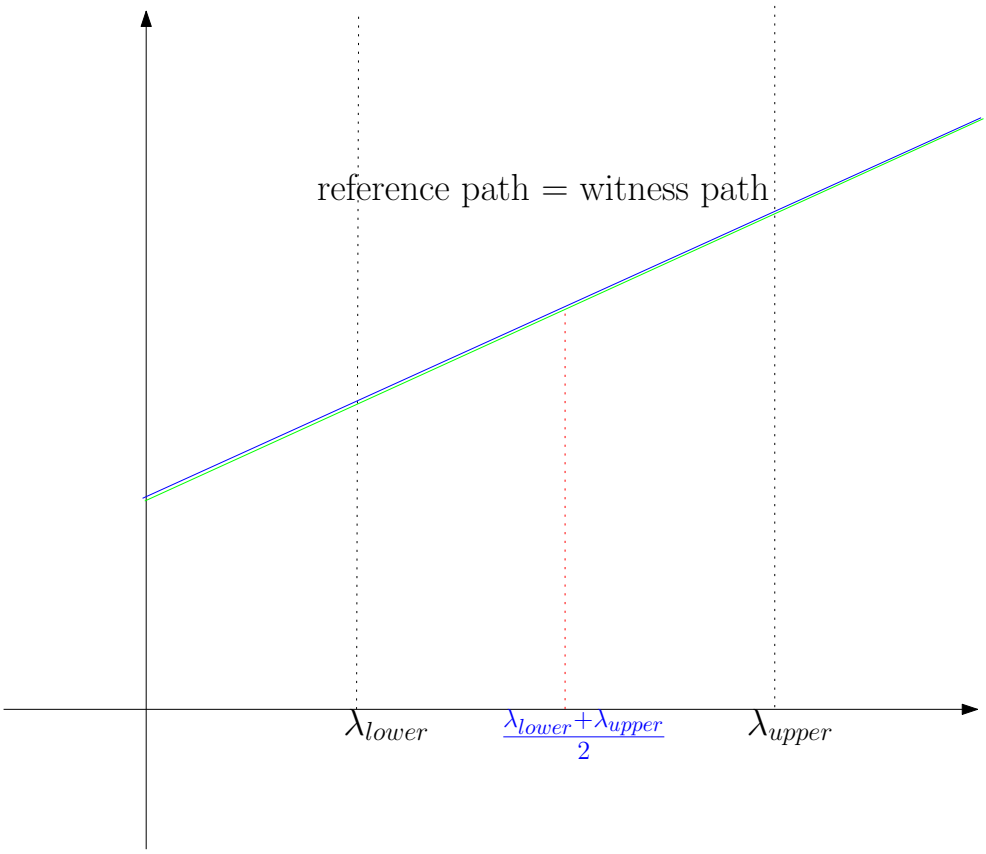

Abbildung 3.1: Referencepath gleich dem Witnesspath

Sollte der Referencepath ungleich dem Witnesspath sein, müssen wir überprüfen, ob der Witnesspath den Referencepath dominiert (vgl. Abbildung 3.2), dass heißt der Witnesspath ist im gesamten Suchraum kürzer als der Referencepath bezüglich der Kombination aus Resoucenverbrauch und Kantenkosten. Dies überprüfen wir, indem wir überprüfen, ob die Kantenkosten des Witnesspaths kleiner gleich den Kantenkosten des Referencepaths sind und der Ressourcenverbrauch des Witnesspaths kleiner gleich dem Ressourcenverbrauch des Referencepath. Sollte das der Fall sein, können wir entscheiden, dass wir den Referencepath nicht als Shortcut benötigen, da für jedes λ klar ist, dass nie ein kürzester Weg über den Referencepath geht.

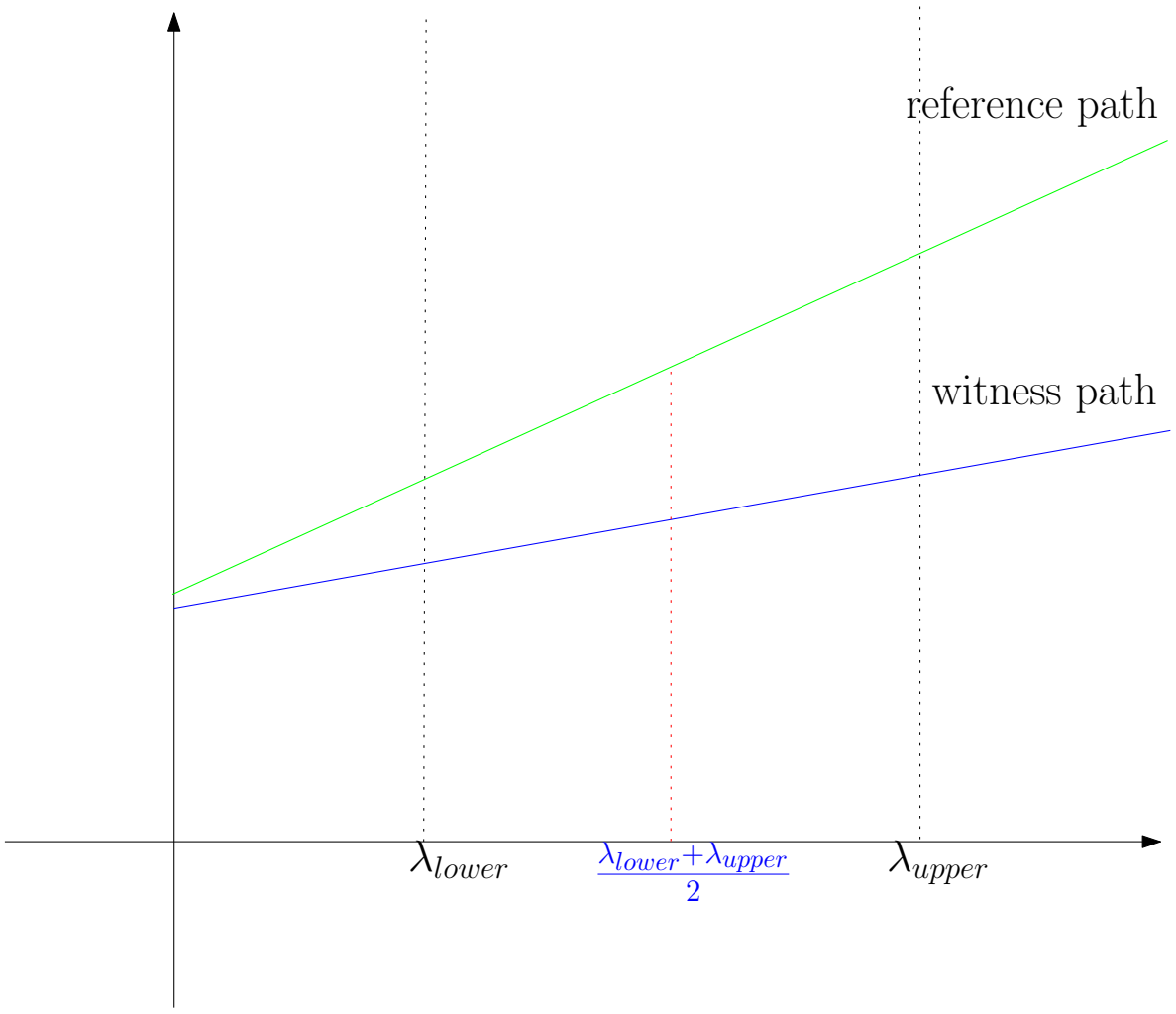

Abbildung 3.2: Witnesspath dominiert den Referencepath

Für die weiteren Fälle benötigen wir das λ des Schnittpunkts (λ_s) der beiden Pfade:
$\lambda_s * (Kosten_{wit} - Res_{wit}) + Res_{wit} = \lambda_s * (Kosten_{ref} - Res_{ref}) + Res_{ref}$
$$\Rightarrow \lambda_s = \frac{Res_{ref} - Res_{wit}}{Kosten_{wit} - Kosten_{ref} - Res_{wit} + Res_{ref}}$$

Sollte das λ_s kleiner sein als das λ_{lower} unseres Suchraums (vgl. Abbildung 3.3), so muss im gesamten Suchraum der Witnesspath den Referencepath dominieren, ansonsten träte Fall 1 ein und der Witnesspath wäre gleich dem Referencepath. Da der Referencepath nun im gesamten Suchbereich dominiert wird, brauchen wir ihn nicht als Shortcut hinzu zu nehmen, da für kein λ gelten kann, dass ein kürzester Weg über den Referencepath verläuft.

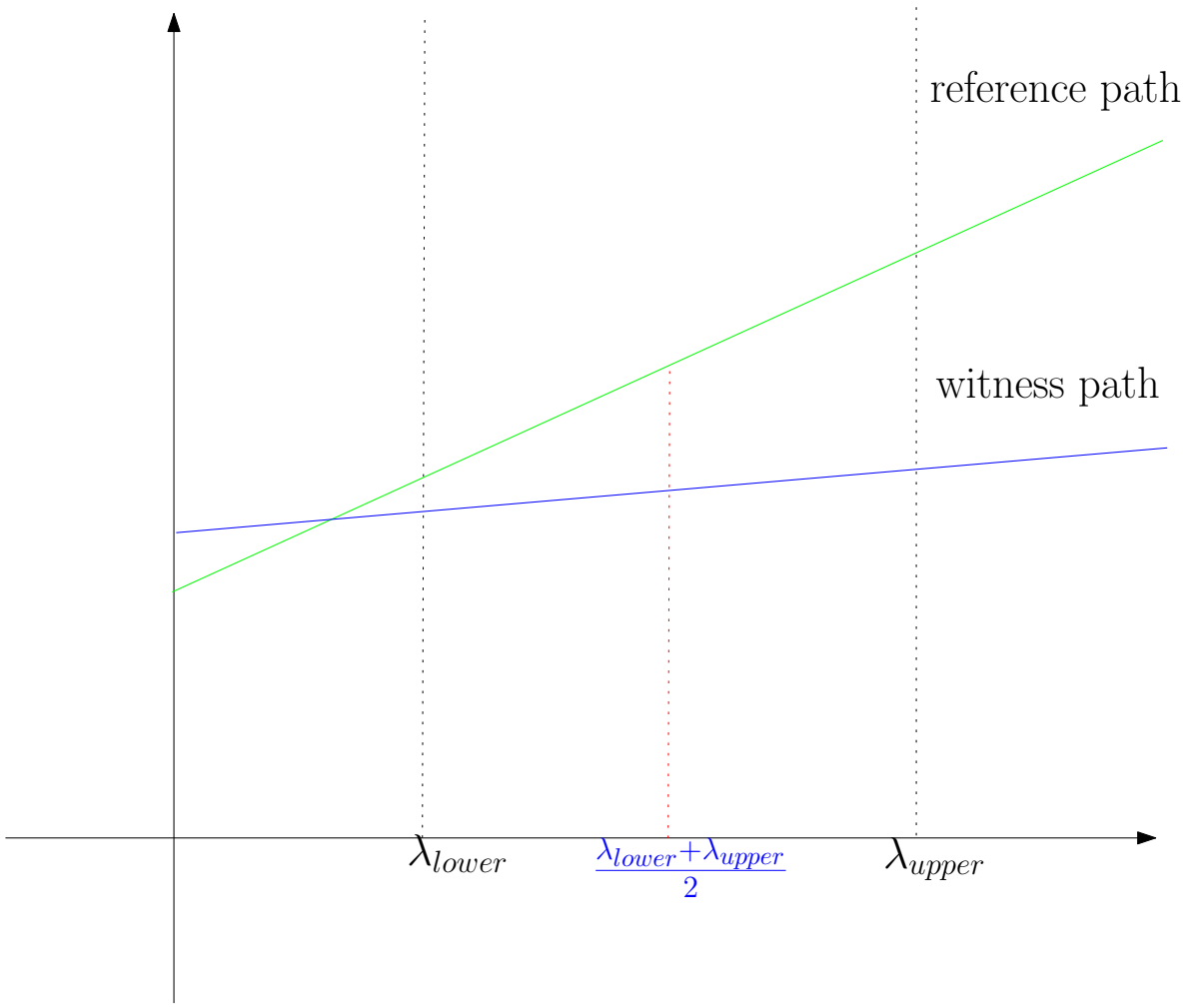

Abbildung 3.3: Referencepath schneidet Witnesspath links der unteren Grenze

Sollte das λ_s größer sein als das λ_{upper} unseres Suchraums (vgl. Abbildung 3.4), so muss im gesamten Suchraum der Witnesspath den Referencepath dominieren, ansonsten träte Fall 1 ein und der Witnesspath wäre gleich dem Referencepath. Da der Referencepath nun im gesamten Suchbereich dominiert wird, brauchen wir ihn nicht als Shortcut hinzu zu nehmen, da für kein λ gelten kann, dass ein kürzester Weg über den Referencepath verläuft.

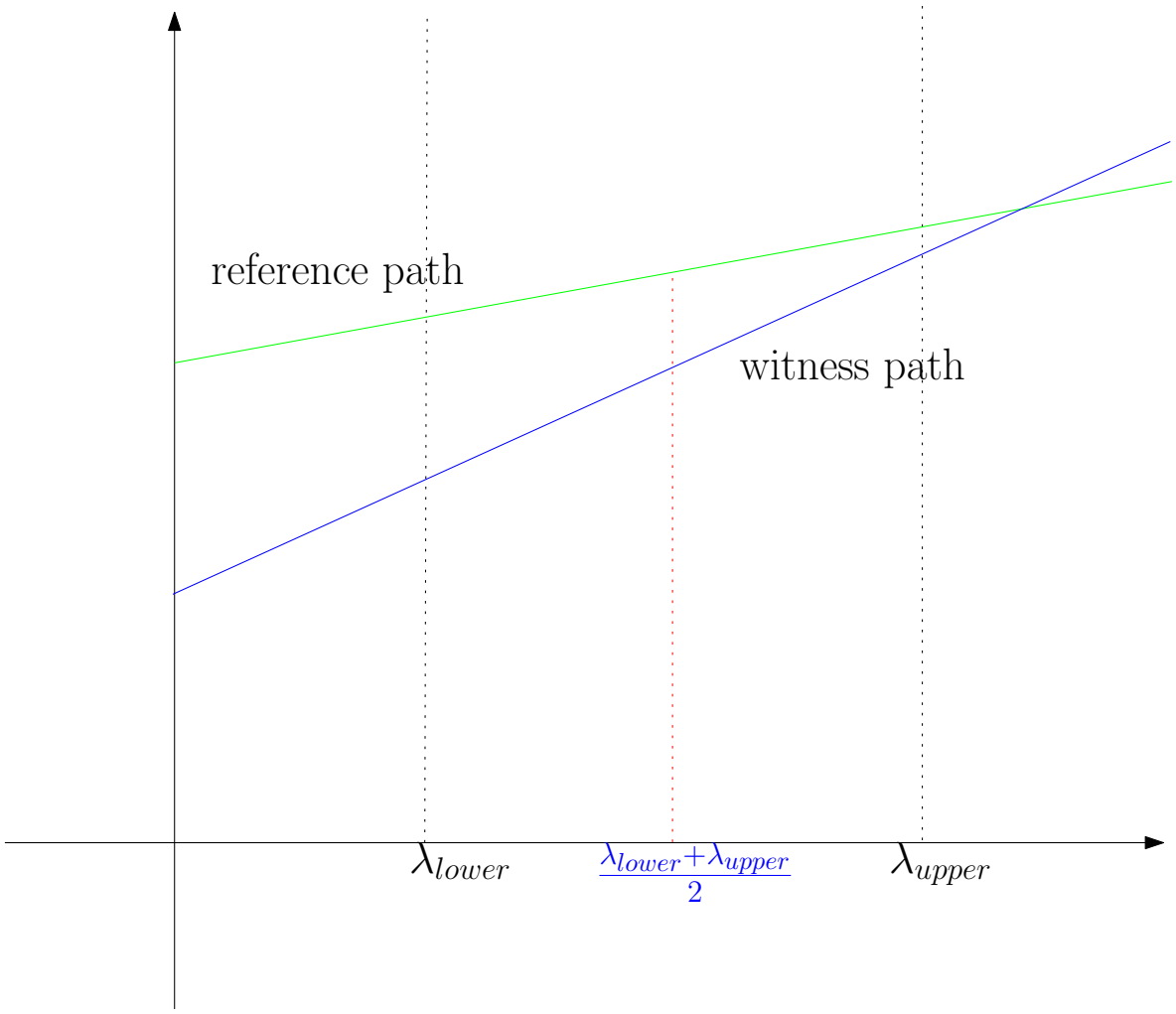

Abbildung 3.4: Referencepath schneidet Witnesspath rechts der oberen Grenze

Die interessanten Fälle sind, wenn wir den Suchraum eingrenzen können, aber nicht sagen können, ob der Referencepath ein Shortcut wird oder nicht.
Sollte sich der Referencepath in der linken Hälfte unseres Suchraums mit dem Witnesspath schneiden (vgl. Abbildung 3.5), so ist der rechte Teil des Suchraums dominiert von unserem Witnesspath, da ansonsten Fall 1 eintreten würde. Daher suchen wir nur noch im linken Teil des Suchraums, da es nur hier noch ein λ geben kann, für das ein kürzester Weg über den Referencepath gehen kann. Wir verschieben daher die obere Grenze unseres Suchbereichs λ_{upper} auf das bisherige λ und nehmen als neues λ die Mitte zwischen λ_{lower} und dem neuen λ_{upper}.

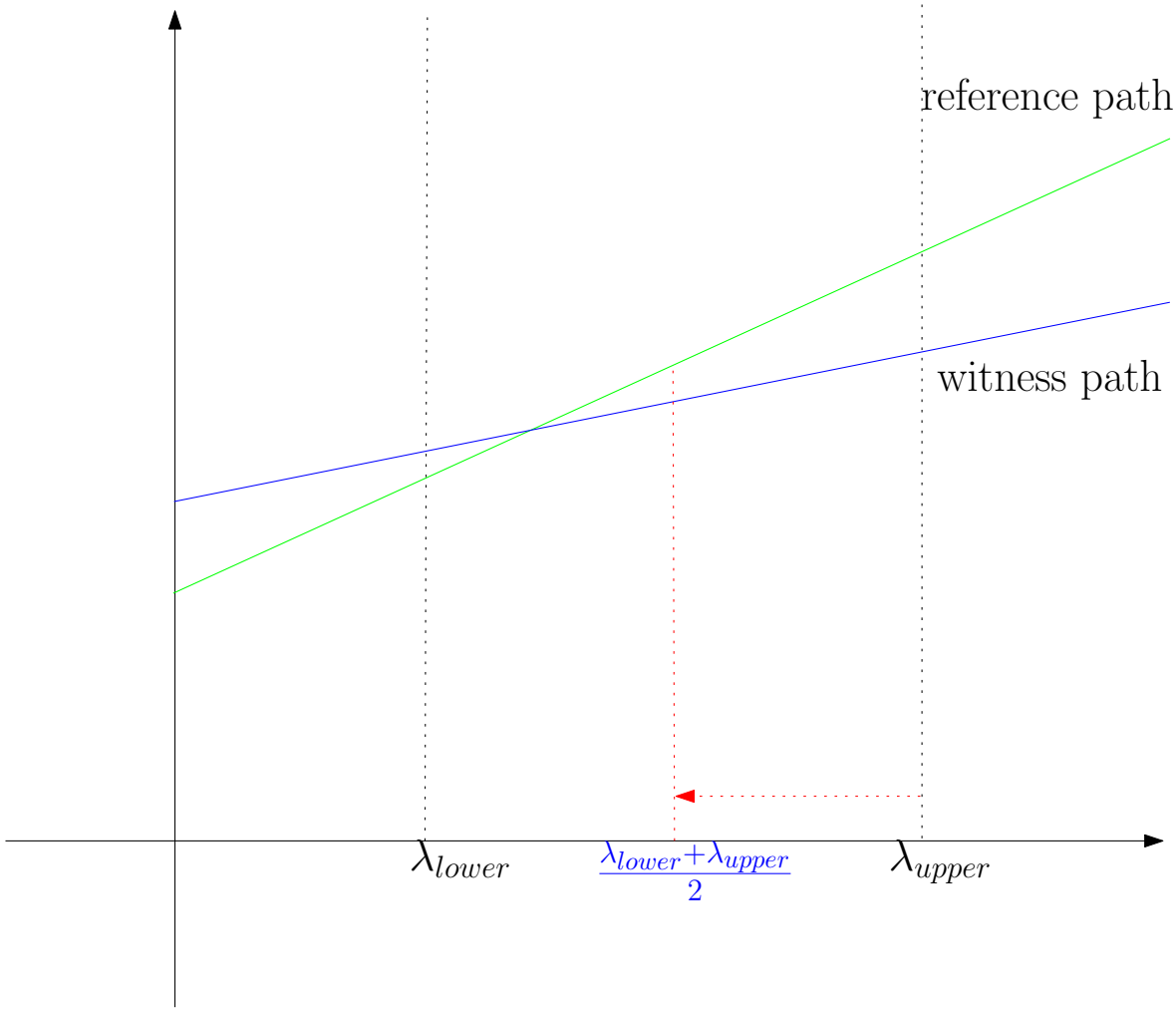

Abbildung 3.5: Referencepath schneidet Witnesspath in der linken Hälfte zwischen den Grenzen

Sollte sich der Referencepath in der rechten Hälfte unseres Suchraums mit dem Witnesspath schneiden (vgl. Abbildung 3.6), so ist der linke Teil des Suchraums dominiert von unserem Witnesspath, da ansonsten Fall 1 eintreten würde. Daher suchen wir nur noch im rechten Teil des Suchraums, da es nur hier noch ein λ geben kann, für das ein kürzester Weg über den Referencepath gehen kann. Wir verschieben daher die untere Grenze unseres Suchbereichs λ_{lower} auf das bisherige λ und nehmen als neues λ die Mitte zwischen λ_{upper} und dem neuen λ_{lower}.

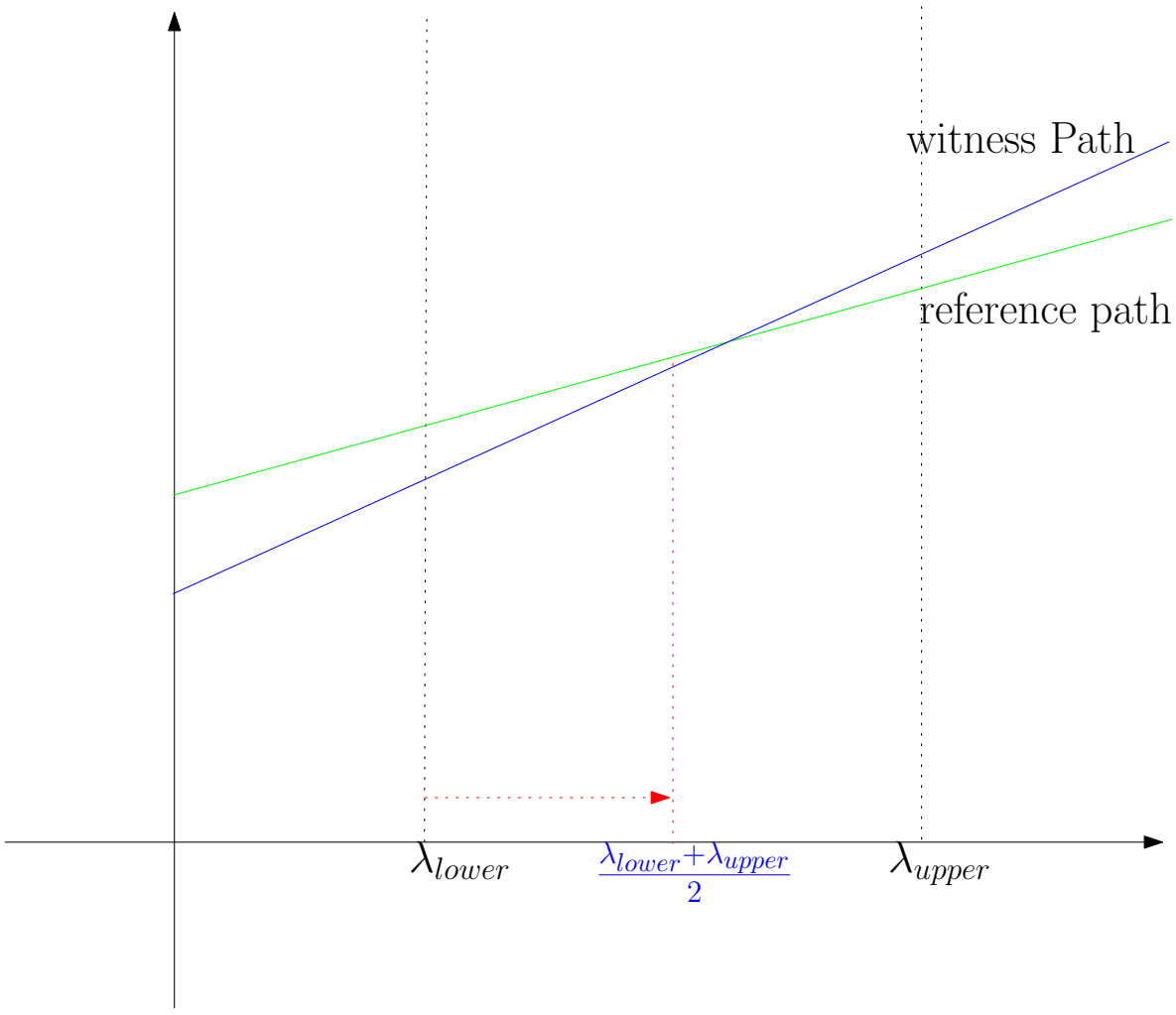

Abbildung 3.6: Referencepath schneidet Witnesspath in der rechten Hälfte zwischen den Grenzen

3.2 Algorithmus

Im Algorithmus 1 werden für einen gegebenen zu kontrahierenden Knoten, für alle Kombinationen aus dessen eingehenden und ausgehenden Kanten, mit dem Algorithmus 2 "computeShortcut" überprüft, ob dieser Shortcut benötigt wird.

Algorithm 1: computeShortcuts

```
   Input: Node nodeToContract
   Output: List<Edge>
1  List <Edge> shortcutsToAdd;
2  Edge shortcut;
3  begin
4      for all Pairs of inEdges and outEdges of nodeToContract do
5          shortcut ← computeShortcut(nodeToContract, inEdge, outEdge);
6          if shortcut ≠ Null then
7              shortcutsToAdd.append(shortcut);
8              shortcut ← Null;
9      return shortcutsToAdd;
```

Mit dem Algorithmus 2 wird mit Hilfe des Algorithmus 3 ”addShortcut” entschieden, ob ein Shortcut erstellt wird oder nicht. Der erstellte oder eben nicht erstellte Shortcut wird dann anschließend an den aufrufenden Algorithmus zurückgegeben.

Algorithm 2: computeShortcut

```
   Input: Node nodeToContract, Edge inEdge, Edge outEdge
   Output: Edge or Null
1  begin
2      Edge shortcut;
3      Boolean adden ← addShortcut(nodeToContract, inEdge, outEdge) ;
4      if adden then
5          shortcut ← new Edge(inEdge.source, outEdge.target,
           inEdge.length+outEdge.length, inEdge.resCost+outEdge.resCost, );
6      return shortcut;
```

Der Algorithmus 3 entscheidet, ob ein Shortcut benötigt wird oder nicht. Dies kann man durch geschicktes Ausprobieren von λ und dem Dijkstra, der für ein bestimmtes λ den kürzesten bzw. schnellsten Pfad berechnet, erreichen. Dabei wird zur Wahl der λ eine binäre Suche mit dem Initial Bereich [0,1] benutzt. Das erste zu überprüfende λ ist, wie auch für alle weiteren λ , $\frac{upperBound + lowerBound}{2}$. Anschließend wir der Dijkstra mit diesem λ ausgeführt. Die Ergebnisse werden mit allen der oben genannten Fälle verglichen und anhand derer entschieden, ob weiter gesucht werden muss oder ob man mit einem positiven oder negativen Ergebnis den Algorithmus beenden kann. Zur Berechnung des Schnittpunkts wird hier der Algorithmus 4 "ıntersect" benutzt. Das Ergebnis wird am Ende dem aufrufenden Algorithmus mitgeteilt.

Algorithm 3: addShortcut

```
   Input: Node nodeToContract, Edge inEdge, Edge outEdge
   Output: Boolean
 1 begin
 2     Boolean notFinish ← True;
 3     real upperBound ← 1;
 4     real lowerBound ← 0;
 5     real midLambda;
 6     Path refPath ← new Path(inEdge.length + outEdge.length, inEdge.resCost +
       outEdge.resCost);
 7     Path witPath;
 8     while notFinish do
 9         midLambda ← (upperBound + lowerBound)/2;
10         witPath ← shortestPath(inEdge.source, outEdge.target, midLambda);
11         if refPath.length == witPath.length && refPath.resCost == witPath.resCost then
12             return True;
13         else if refPath.length ≥ witPath.length && refPath.resCost ≥ witPath.resCost then
14             return False;
15         else
16             int intersectLamda ← intersection(witPath, refPath);
17             if intersectLambda < lowerBound || intersectLambda > upperBound then
18                 return False;
19             else if refPath.length - refPath.resCost < witPath.length -refPath.resCost then
20                 lowerBound ← midLambda;
21             else
22                 upperBound ← midLambda;
23     return Null;
```

Der Algorithmus 4 berechnet für zwei mathematische Funktionen, die den Reference und den Witnesspath repräsentieren, den Schnittpunkt. Dies ist eine einfache Schnittpunktberechnung zweier linearer Geraden.

Algorithm 4: intersection

Input: Path witPath, Path refPath
Output: real intersectLambda

```
1 begin
2     return (witPath.resCost-refPath.resCost) /
      (refPath.length-witPath.length-refPath.resCost+witPath.resCost);
```

4 Experimentelle Evaluation

Um unsere Ch-Berechnung zu evaluieren wurden die Testberechnungen auf einer Intel i7-2620M CPU mit 2.7 GHz (im Boost 3.4 GHz) und 8GB RAM durchgeführt. Diese Tests wurden auf kleineren Graphen ausgeführt, die aus den Open Street Map Daten extrahiert wurden. Die Details der Graphen "10k", "100k", "1m" und "hessen" sind wie folgt:

	10k	100k	1m	hessen
Knotenanzahl	11220	100242	999591	1121082
Kantenanzahl	24119	213096	2131490	2269020

Tabelle 4.1: Graph-Daten

4.1 Auswertung

Von besonderem Interesse ist es, zu analysieren, wie sich das Verhalten von CSP-Dijkstra, CSPCH-Dijkstra und der CSP-Heuristik mit und ohne CH bei unterschiedlichen Kontraktionsgraden ändert. Inbesondere der Vergleich der Antwortzeiten der CSP-Heuristiken sind interessant. Ebenfalls notwendige Untersuchungen sind, in wieweit sich der jeweilige Suchraum in Hinblick auf unterschiedliche Kontraktionsgrade ändert.

4.1.1 Kontraktionszeiten

Hier werden die Kontraktionszeiten (in s) für die unterschiedlichen Graphen, abhängig von deren Kontraktionsgrad, analysiert. Klar ist, dass größere Graphen länger brauchen, um kontrahiert zu werden. Dies kann hier bestätigt werden. Ein anderer Punkt ist die Veränderung der Kontraktionszeit im Hinblick auf den Kontraktionsgrad. Natürlich ist auch hier klar, dass ein höherer Kontraktionsgrad meist auch längere Zeit benötigt. Die Frage, die es zu beantworten galt war, wie viel? Beim "10k"-Graphen ist die Kontraktionszeit jedoch bei 99,8% höher als bei 99,9%, dies könnte an einer Schwankung durch den Turbo-Boost liegen. Wie man der Tabelle 4.2 leicht entnehmen kann, ist bei den kleineren Graphen, im Vergleich zu den größeren Graphen, nur eine leichte Steigerung der Sprünge am Ende zu beobachten. Beim "1m"- Graphen kommt es zwischen 99,8 % und 99,9% zu einer Steigerung um den Faktor 3,5. Von 99,9% nach 100% kommt es zu einer Steigerung um den Faktor 125. Das letzte 0.1% braucht somit mehr als die 99,9% vorher. Beim Graphen "hessen" haben wir zwischen 99,8% und 99,9% eine Steigerung um den Faktor 3 und zwischen 99,9% und

100% eine Steigerung um den Faktor 177. Damit braucht bei "hessen" das letzte 0,1% fast doppelt solange wie die 99,9% vorher. Bei dem Versuch für den Deutschlandgraphen eine CH zu berechnen dauerte der Grad 99,9% schon 90 min und der Grad 100% haben wir nach 1 Woche intensivsten Rechnens abgebrochen, da hier die verbrauchte Energie den Nutzen nicht aufwiegen kann.

	99 %	99,4 %	99,5 %	99,8 %	99,9 %	100%
10 k	0,74s	1,00s	1,01s	1,28s	1,20s	1,29s
100 k	1,44s	1,60s	1,70s	2,07s	3,01s	4,24s
1m	13,43s	17,37s	19,60s	38,78s	140,12s	17455,28s ($\approx 4,8h$)
hessen	12,50s	15,07s	15,89s	27,50	82,59s	14644,96s ($\approx 4h$)

Tabelle 4.2: Kontraktionszeiten

4.1.2 Antwortzeiten von Dijkstra-Anfragen bei verschiedenen Kontraktionsgraden

Welche Auswirkungen hat der Kontraktionsgrad auf die Antwortzeiten (in ms) eines bidirektionalen CSPCH-Dijkstra, bis er den kürzesten Weg von s nach t mit einem bestimmten λ gefunden hat? Diese Zeiten sind gemittelt über 10000 Anfragen. Die Annahme ist, dass die Antwortzeit mit dem Anstieg des Kontraktionsgrad sinkt. Im folgenden Diagramm 4.1 kann man erkennen, dass die Annahme bestätigt wird. Je höher der Kontraktionsgrad ist, desto niedriger sind die Anfragezeiten.

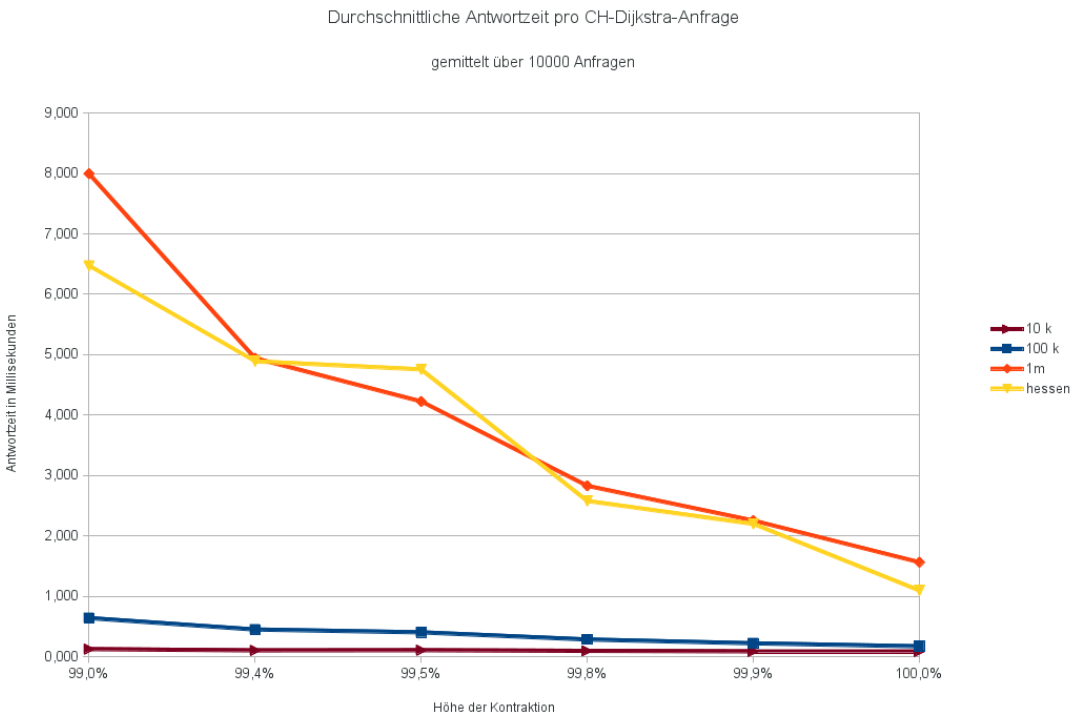

Abbildung 4.1: Antwortzeiten von Dijkstra-Anfragen

In Tabelle 4.3 sind zusätzlich die Antwortzeiten (gemittelt über 10000 Anfragen) der CSP-Dijkstra-Anfragen auf den Graphen ohne CH abgetragen. Mit dieser zusätzlichen Angabe kann man zeigen, dass sich selbst mit einem Kontraktionsgraphen von 99% eine 7- fache Beschleunigung bei dem kleinsten Graphen erreichen lässt. Bei den Größeren bewegt sich dieser Faktor zwischen 15 und 22, was eine deutliche Beschleunigung ist. An den Zeiten des langsamen, nicht beschleunigten Dijkstras (auf nicht CH-Graph), im Vergleich zu den Zeiten des CH-Dijkstras (auf CH-Graph mit Kontraktionsgrad 100%), lässt sich erkennen, dass Beschleunigungen um den Faktor 100 möglich sind.

	ohne CH	99 %	99,4 %	99,5 %	99,8 %	99,9 %	100%
10k	0,918ms	0,124ms	0,103ms	0,106ms	0,095ms	0,087ms	0,086ms
100k	8,935ms	0,638ms	0,448ms	0,402ms	0,283ms	0,220ms	0,170ms
1m	117,832ms	7,995ms	4,940ms	4,225ms	2,828ms	2,252ms	1,562ms
hessen	136,636ms	6,472ms	4,890ms	4,755ms	2,577ms	2,196ms	1,094ms

Tabelle 4.3: Antwortzeiten von Dijkstra-Anfragen

4.1.3 Suchraum einer Dijkstra-Anfrage

Durch das Einfügen von Shortcuts erhoffen wir uns, dass der Suchraum unseres Dijkstras kleiner und somit schneller wird. Unsere Annahme ist daher, dass je höher der Kontraktionsgrad ist, desto weniger Knoten und Kanten müssen bei einer $s - t - \lambda$-Anfrage berührt werden, um von s nach t zu gelangen.
In Diagramm 4.2 lässt sich erkennen, dass mit steigendem Kontraktionsgrad die Anzahl berührter Knoten (gemittelt über 10000 Anfragen) abnimmt. Bei größeren Graphen sinkt die Anzahl der berührten Knoten schneller. Man kann annehmen, dass mit dem Einfügen eines Shortcuts, im Verhältnis zu kleineren Graphen, eine größere Anzahl von Knoten übersprungen werden kann.

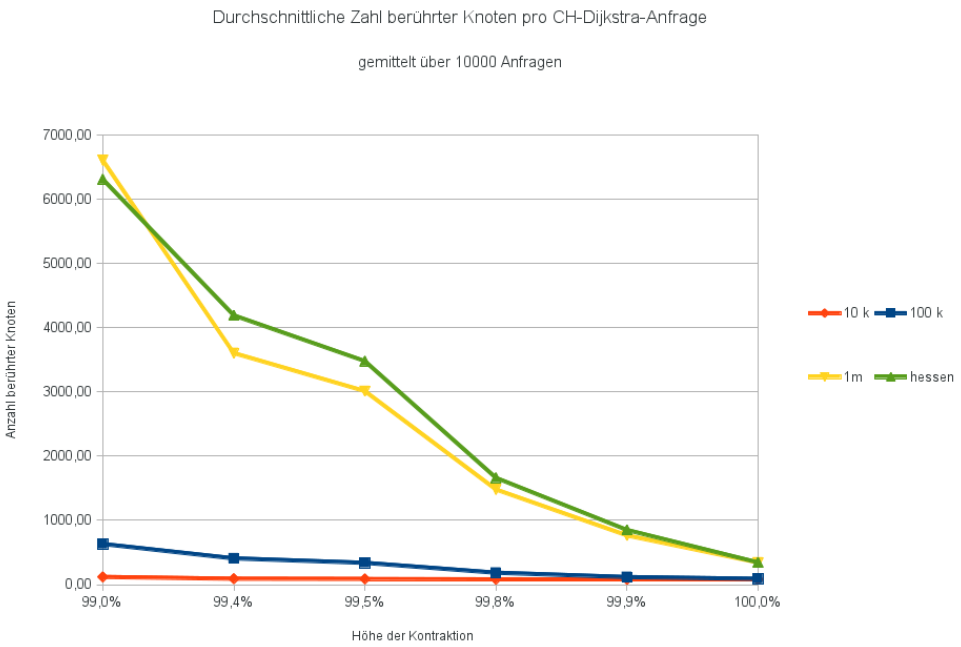

Abbildung 4.2: Berührte Knoten bei Dijsktra-Anfragen

In Tabelle 4.4 sind zusätzlich noch die Zahlen der berührten Knoten von Dijkstra-Anfragen (gemittelt über 10000 Anfragen) auf den Graphen ohne CH abgetragen. Dies zeigt uns, dass, zusätzlich zur Abnahme innerhalb der Ch-Graphen, die größte Abnahme vom Graphen ohne CH zu den CH-Graphen stattfindet. Hier bewegen sich die Faktoren zwischen 47 beim "10k"- Graphen und dessen CH-Graph mit 99% bzw. 1431 beim "1m"- Graphen und dessen

CH-Graph mit 100%. Dies zeigt, dass bei größeren Graphen eine größere Verringerung des Suchraums bezüglich der Zahl berührter Knoten erreicht wird.

	ohne CH	99 %	99,4 %	99,5 %	99,8 %	99,9 %	100%
10 k	5602,37	119,66	91,60	89,06	82,98	82,21	82,29
100 k	49308,01	632,01	408,92	339,30	185,37	118,14	90,56
1m	492433,33	6607,45	3604,46	3013,22	1481,63	765,76	343,98
hessen	570210,72	6311,65	4191,70	3479,87	1663,50	850,92	345,53

Tabelle 4.4: Berührte Knoten bei Dijsktra-Anfragen

In Diagramm 4.3 lässt sich erkennen, dass mit steigendem Kontraktionsgrad die Anzahl berührter Kanten (gemittelt über 10000 Anfragen) abnimmt. Bei größeren Graphen sinkt die Anzahl der berührten Kanten schneller. Man kann annehmen, dass mit dem Einfügen eines Shortcuts, im Verhältnis zu kleineren Graphen, eine größere Anzahl von Kanten übersprungen werden kann.

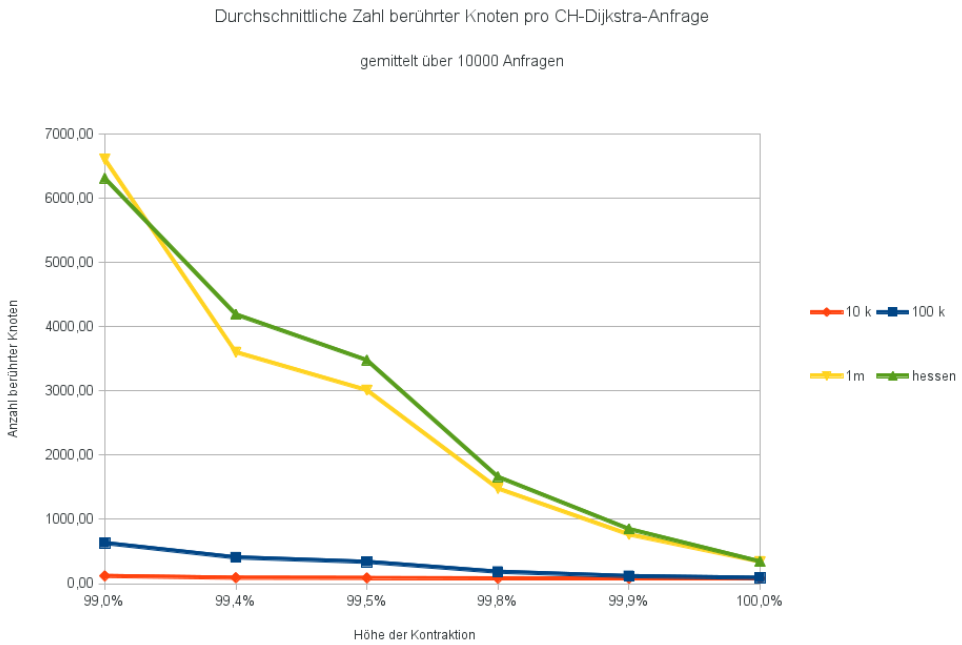

Abbildung 4.3: Berührte Kanten bei Dijkstra-Anfragen

In Tabelle 4.5 sind zusätzlich noch die Zahlen der berührten Kanten von Dijkstra-Anfragen (gemittelt über 10000 Anfragen) auf den Graphen ohne CH abgetragen. Dies zeigt uns, dass, zusätzlich zur Abnahme innerhalb der Ch-Graphen, die größte Abnahme vom Graphen ohne CH zu den CH-Graphen stattfindet. Hier bewegen sich die Faktoren zwischen 2 beim "10k"- Graphen und dessen CH-Graph mit 99% bzw. 16 beim "hessen"- Graphen und dessen CH-Graph mit 100%. Dies zeigt, dass bei größeren Graphen eine größere Verringerung des Suchraums bezüglich der Zahl berührter Knoten erreicht wird.

	ohne CH	99 %	99,4 %	99,5 %	99,8 %	99,9 %	100%
10 k	5804,79	2362.73	1776.43	1736.17	1509.06	1123.13	948.52
100 k	50394,86	7663,43	6128.41	5688,15	4494,90	3445,93	1474,82
1m	503708,87	92098,75	74392,75	71339,14	66096,83	70733,73	55511,51
hessen	582475,61	78394,55	66388,61	62320,49	54155,07	57275,48	36450,84

Tabelle 4.5: Berührte Kanten bei Dijkstra-Anfragen

4.1.4 Antwortzeiten von CSP-Anfragen

Welche Auswirkungen hat der Kontraktionsgrad auf die Antwortzeiten (in ms) der CSPCH-Heuristik, bis sie den kürzesten Weg von s nach t unter einer bestimmten Resourcengrenze gefunden hat? Diese Zeiten sind gemittelt über 1000 Anfragen. Die Annahme ist, dass die Antwortzeit mit dem Anstieg des Kontraktionsgrad sinkt. Im folgenden Diagramm 4.4 kann man erkennen, dass die Annahme bestätigt wird. Je höher der Kontraktionsgrad ist, desto niedriger sind die Anfragezeiten.

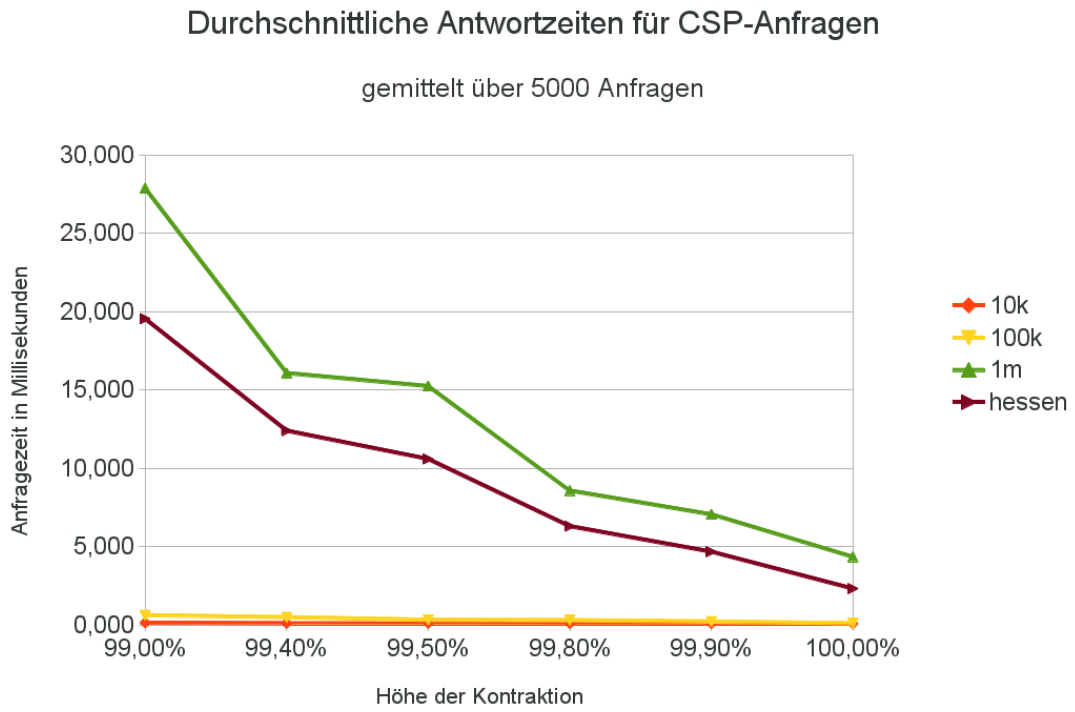

Abbildung 4.4: Antwortzeiten von CSP-Anfragen

In Tabelle 4.6 sind zusätzlich die Antwortzeiten (gemittelt über 1000 Anfragen) der CSP-Anfragen, auf den Graphen ohne CH, abgetragen. Mit dieser zusätzlichen Angabe kann man zeigen, dass sich selbst mit einem Kontraktionsgraphen von 99% eine Beschleunigung um den Faktor 9 bei dem kleinsten Graphen erreichen lässt. Bei den Größeren bewegt sich dieser Faktor zwischen 21 und 29, was eine deutliche Beschleunigung ist. An den Zeiten des langsamen, nicht beschleunigten Dijkstras (auf nicht CH-Graph), im Vergleich zu den Zeiten des CH-Dijkstras (auf CH-Graph mit Kontraktionsgrad 100%), lässt sich erkennen, dass sich Beschleunigungen zwischen einen Faktor 141 und 248 bei den größeren Graphen bewegen. Beim "10k"- Graphen ist nur eine Beschleunigung um den Faktor 14 festzustellen. Hier nehmen wir an, dass dieser Graph zu klein ist um eine wirkliche Beschleunigung zu erfahren. Außerdem vermuten wir, dass alle diese Werte etwas zu niedrig sind. Dies geschieht dadurch, dass bei unserem Test alle Anfragen zufällig produziert werden, dadurch die Ressourcenbeschränkung hoch genug wird, sodass der kürzeste Weg schon die maximal erreichbare Lösung ist und somit die Anfragen bei der ersten Iteration durchkommen.

	ohne CH	99 %	99,4 %	99,5 %	99,8 %	99,9 %	100%
10k	1,200ms	0,140ms	0,108ms	0,107ms	0,099ms	0,092ms	0,086ms
100k	15,946ms	0,627ms	0,492ms	0,331ms	0,309ms	0,220ms	0,109ms
1m	606,806ms	27,874ms	16,075ms	15,243ms	8,574ms	7,061ms	4,346ms
hessen	569,928ms	19,545ms	12,400ms	10,593ms	6,306ms	4,676ms	2,321ms

Tabelle 4.6: Antwortzeiten von CSP-Anfragen

4.1.5 Suchraum der CSP-Anfragen

Durch das Einfügen von Shortcuts erhoffen wir uns, dass der Suchraum unseres Dijkstras kleiner wird und somit auch der Suchraum des CSP. Unsere Annahme ist daher, dass je höher der Kontraktionsgrad ist, desto weniger Knoten und Kanten müssen bei einer CSP-Anfrage berührt werden, um von s nach t zu gelangen.
In Diagramm 4.5 lässt sich erkennen, dass mit steigendem Kontraktionsgrad die Anzahl berührter Knoten (gemittelt über 1000 Anfragen) abnimmt. Bei größeren Graphen sinkt die Anzahl der berührten Knoten schneller. Man kann annehmen, dass sich hier der kleinere Suchraum bezüglich der Knoten der einzelnen Dijkstras niederschlägt wodurch wir bei größeren Graphen, im Verhältnis zu kleineren Graphen, eine größere Abnahme haben.

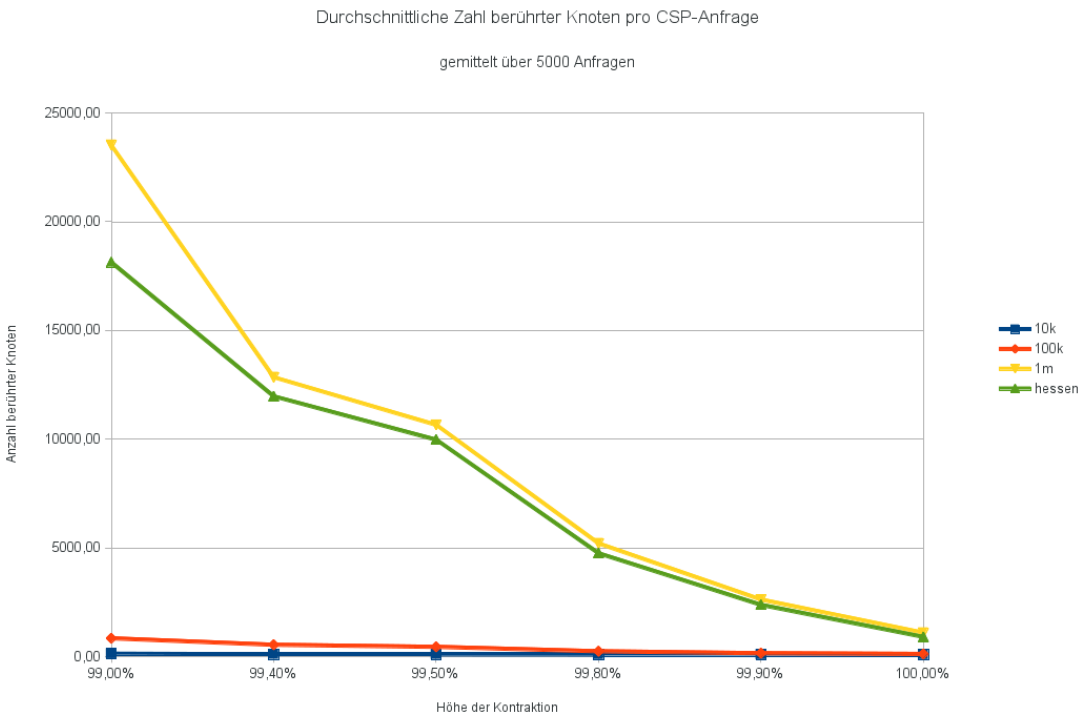

Abbildung 4.5: Berührte Knoten von CSP-Anfragen

In Tabelle 4.7 sind zusätzlich noch die Zahlen der berührten Knoten von CSP-Anfragen (gemittelt über 1000 Anfragen) auf den Graphen ohne CH abgetragen. Dies zeigt uns, dass, zusätzlich zur Abnahme innerhalb der Ch-Graphen, die größte Abnahme vom Graphen ohne CH zu den CH-Graphen stattfindet. Hier bewegen sich die Faktoren zwischen 48 beim "10k"-Graphen und dessen CH-Graph mit 99% bzw. 1869 beim "hessen"- Graphen und dessen CH-Graph mit 100%. Dies zeigt, dass bei größeren Graphen eine größere Verringerung des Suchraums bezüglich der Zahl berührter Knoten erreicht wird.

	ohne CH	99 %	99,4 %	99,5 %	99,8 %	99,9 %	100%
10k	6547,77	136,59	104,64	101,82	95,33	94,60	94,70
100k	66939,54	841,70	544,02	449,00	245,83	157,05	120,67
1m	1840945,79	23532,06	12849,29	10658,43	5209,50	2625,00	1098,46
hessen	1687509,24	18139,25	11973,50	9977,57	4755,33	2381,42	903,13

Tabelle 4.7: Berührte Knoten von CSP-Anfragen

In Diagramm 4.6 lässt sich erkennen, dass mit steigendem Kontraktionsgrad die Anzahl berührter Kanten (gemittelt über 1000 Anfragen) abnimmt. Bei größeren Graphen sinkt die Anzahl der berührten Kanten schneller. Man kann annehmen, dass sich hier der kleinere Suchraum bezüglich der Kanten der einzelnen Dijkstras niederschlägt, wodurch wir bei größeren Graphen im Verhältnis zu kleineren Graphen eine größere Abnahme haben.

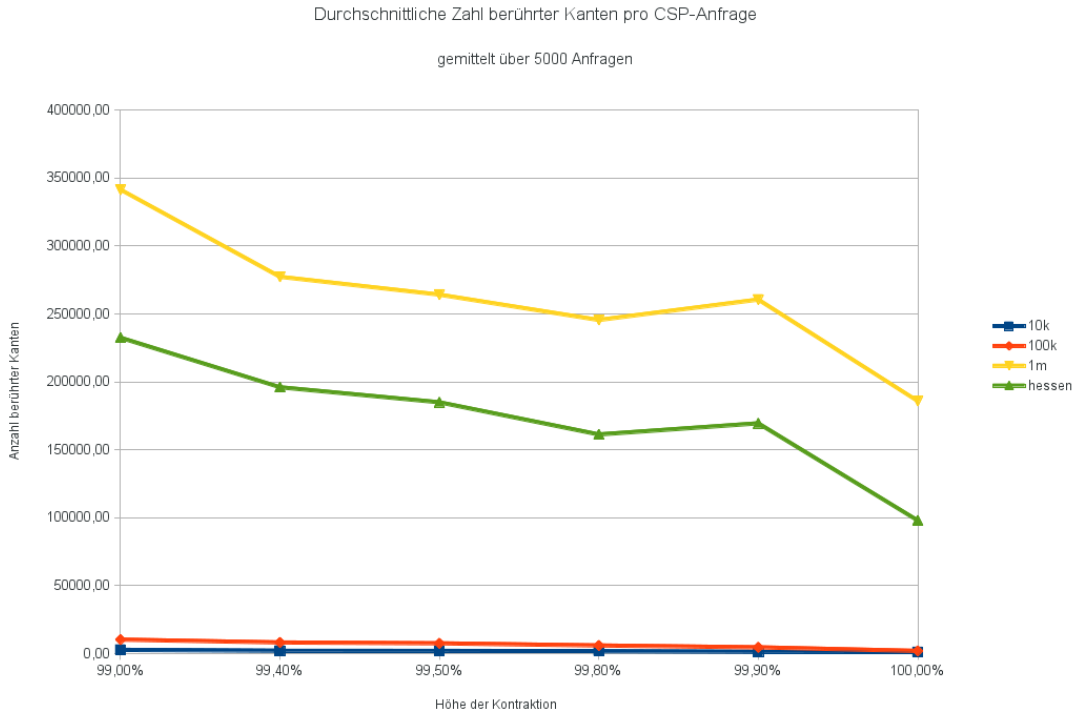

Abbildung 4.6: Berührte Kanten von CSP-Anfragen

In Tabelle 4.7 sind zusätzlich noch die Zahlen der berührten Knoten von CSP-Anfragen (gemittelt über 1000 Anfragen) auf den Graphen ohne CH abgetragen. Dies zeigt uns, dass, zusätzlich zur Abnahme innerhalb der Ch-Graphen, die größte Abnahme vom Graphen ohne CH zu den CH-Graphen stattfindet. Hier bewegen sich die Faktoren zwischen 48 beim "10k"-Graphen und dessen CH-Graph mit 99% bzw. 1869 beim "hessen"- Graphen und dessen CH-Graph mit 100%. Dies zeigt, dass bei größeren Graphen eine größere Verringerung des Suchraums bezüglich der Zahl berührter Knoten erreicht wird

	ohne CH	99 %	99,4 %	99,5 %	99,8 %	99,9 %	100%
10k	6776,91	2770,40	2077,83	2028,48	1761,47	1306,46	1099,68
100k	68394,45	10273,21	8207,90	7572,68	5992,84	4596,44	1976,22
1m	1885965,03	341378,80	277300,39	264162,29	245530,89	260547,10	185918,24
hessen	1725578,13	232486,01	196017,73	184946,96	161298,35	169476,45	98039,89

Tabelle 4.8: Berührte Kanten von CSP-Anfragen

4.1.6 Weitere Analysen von CSP-Anfragen

Um die Korrektheit unserer Beschleunigung zu bestätigen, kann zwei Eigenschaften von CSP-Anfragen überprüfen: das ist zum einen die Anzahl der Iterationen und zum anderen die Approximationsgüte einer Anfrage. Für beide Eigenschaften muss gelten, dass innerhalb einer Graphenfamile (alle 10k Graphen), egal für welchen Kontraktionsgrad (einschließlich des Graphen ohne CH), die Anzahl der Iterationen und die Approximationsgüte, bei der selben Anfrage, gleich sein müssen. Die Approximationsgüte gibt die obere Schranke für die Entfernung der gefundenen Lösung von der optimalen Lösung an. Die Berechnung der Approximation ist dabei der Quotient aus der Länge der letzten gefundenen Lösung, die nicht mehr unter die Resourcenbeschränkung gepasst hat, und der Länge unserer gefundenen Lösung.
Die Untersuchung dieser Eigenschaften an unsere CSP-Anfragen hat dies bestätigt.

4.1.7 Bedeutung

Abschließend können wir sagen, dass bezogen auf die Kontraktionszeit (vgl. 4.1.1) bei größeren Graphen die Kontraktion nur bis zum Grad 99,9% laufen sollte, da ansonsten der Aufwand nicht im Verhältnis zum Nutzen steht. Nimmt man noch die Antwortzeiten des beschleunigten CSPs (vgl. 4.1.4) dazu, kann man zwischen 99,9% und 100% nur noch knapp eine Verdoppelung der Geschwindigkeit erkennen.

5 Zusammenfassung und Ausblick

Was wir in dieser Arbeit erreichen wollten, war die Beschleunigung der CSP-Heuristik mit der Vorberechnung einer CH. Mit Vorberechnungen für kleinere Graphen wie "10k", "100k", "1m" und "hessen" ist es uns gelungen, in akzeptabler Zeit eine CH zu berechnen. Für größere Graphen konnte eine Beschleunigung um eine Faktor 141-248 erreicht werden. Bei einem Graphen der Größe von Deutschland war es uns nicht möglich, entsprechende Tests durchzuführen, da der höchste Kontraktionsgrad nach 1 Woche immer noch nicht fertig war. Hierbei sei auch zu bedenken, dass der Aufwand nicht den Nutzen übersteigt. In den meisten Fällen kann schon ein CH-Graph mit Grad 99,9% ausreichen, da auch dieser schon ein großes beschleunigendes Potenzial besitzt, das in den meisten Fällen ausreicht.

Ausblick

Da die Berechnung des Deutschlandgraphen zu lange gedauert hätte und damit auch entsprechende Tests nicht möglich waren, wurde diese Funktionalität nicht in den ToureNPlaner eingefügt. Für die Zukunft wäre dies denkbar, wenn die CH-Berechnung weiter beschleunigt wird. Dies könnte z.B. durch Parallelisierung erreicht werden. Auch könnte man versuchen, die Größe des CH-Graphen zu Minimieren. Eine denkbare Möglichkeit dies zu erreichen, wäre das Entfernen von doppelten Kanten. In weiteren Studien gilt es nun zu Untersuchen, ob diese Modifizierungen die Berechnung für solch große Graphen, wie dem Deutschlandgraphen, möglich machen.

Literaturverzeichnis

[Boy96] J. Boyer. The Fibonacci Heap. *Dr. Dobb's Journal, Januar 1997.*, 1996. URL http://www.drdobbs.com/database/algorithm-alley/184410119. (Zitiert auf Seite 9)

[Dij59] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1(0029-599X):269–271, 1959. doi: 10.1007/BF01386390. URL http://www.springerlink.com/index/10.1007/BF01386390http://www.springerlink.com/content/uu8608u0u27k7256/. (Zitiert auf Seite 9)

[FS13] S. Funke, S. Storandt. Polynomial-time Construction of Contraction Hierarchies for Multi-criteria Objectives. 2013. (Zitiert auf den Seiten 10 und 11)

[GSSD08] R. Geisberger, P. Sanders, D. Schultes, D. Delling. Contraction Hierarchies : Faster and Simpler Hierarchical Routing in Road Networks. *Experimental Algorithms*, 2(July):319–333, 2008. URL http://www.springerlink.com/index/j062316602803057.pdf. (Zitiert auf Seite 11)

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.

(Peter Vollmer)