Gymnasium Hohenlimburg

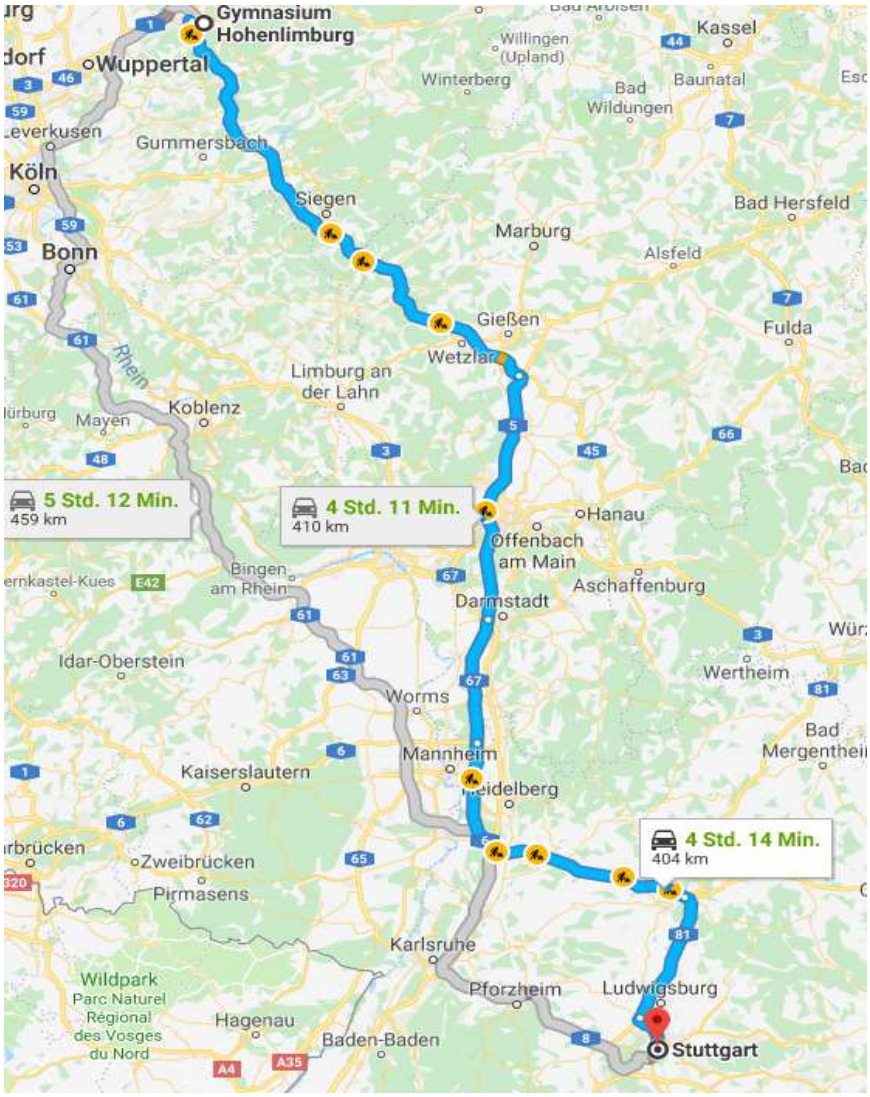
Stufe Q1 2018/19

Facharbeit im Leistungskurs Mathematik

Von Hohenlimburg nach Stuttgart

- Wie der Dijkstra Algorithmus wirklich weiter hilft.

*



Verfasser: Michael Wigond

Kursleiter: Herr Busch

Bearbeitungszeit: 22.12.2018-28.02.2019

Abgabetermin: 28.02.2019

Inhaltsverzeichnis

1. Einleitung / Problembenennung

„Ohne Faulheit kein Fortschritt“- Manfred Hausmann

Diese Facharbeit beschäftigt sich mit dem Thema der kürzesten Weg-Problematik. Der Mensch ist ein sehr effizientes Lebewesen, welches durch die Mathematik versucht alltägliche Probleme zu lösen (Modellierungskreislauf). Dies wird auch im Alltag, wo der Mensch beispielweise versucht abzukürzen und instiktiv dadurch den kürzesten Weg verwendet deutlich. Der menschliche Instikt kann sich jedoch nicht alle Verbindungen merken, geschweige denn den kürzesten Weg auf längeren Strecken herausfinden. Die Informatik, basierend auf mathematischen Methoden, löst so alltägliche Probleme durch Algorithmen, welche durch die Informatik verarbeitet und ausgerechnet werden können. Wie schon im Titel erwähnt, möchte ich die kürzeste Strecke zwischen dem Gymnasium Hohenlimburg und Stuttgart herausfinden.

Da ich zukünftig Informatik studieren möchte, liegt mir dieses Thema besonders am Herzen.

2. Definition Algorithmus

Die Algorithmik ist ein Teilgebiet der Numerik, welches durch theoretische Überlegungen reale Probleme modellieren und lösen kann.

Der Begriff Algorithmus kann durch die systematische Verabeitung von Daten beschrieben werden. "Ein Algorithmus ist eine eindeutige Beschreibung eines in mehreren Schritten durchgeführten (Bearbeitungs-) Vorganges" [1]. Geschichtlich hat der Algorithmus seine Wurzeln bei dem Mathematiker Muhammed ibn Musa abu Djafar alChoresmi. Der persisch-arabische Mathematiker veröffentlichte um etwa 800 nach Christus eine Aufgabensammlung für Kaufleute und Testamentsvollstrecker, welche die Probleme und dessen mathematische Lösung thematisiert. Diese Aufgabensammlung wurde im nachhinein als Liber Algorithmi ins Lateinische übersetzt. Die Bezeichnung Algorithmus ist ein Kunstwort aus dem griechischen Wort "arithmos" für Zahl und dem Namen des Mathematikers [2].

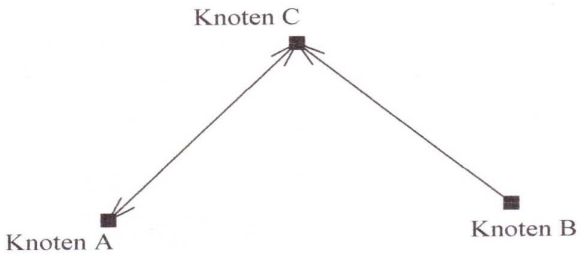
---

[1] Saake, Gunter und Sattler, Kai-Uwe: "Algorithmen und Datenstrukturen, Eine Einführüng mit Java", 3.,überabeitete Auflage , Heidelberg 2006, S.3

[2] Vgl.: Saake (2006) , S.5

2.1 Algorithmische Graphentheorie

"Jedes System, das aus diskreten Zuständen oder Objekten und Beziehungen zwischen diesen besteht, kann als Graph modelliert werden. Diese Darstellung ermöglicht den Einsatz graphentheoretischer Algorithmen." - Volker Turau, Christopher Weyer



Die obige Grafik dient zur Demonstration und mithilfe dieser möchte ich die grundlegenden Fachbegriffe der Graphentheorie erläutern. Man spricht bei den Objekten auch von Knoten, welche auch wie festgelegte Punkte zu behandeln sind. Zwischen den Knoten bestehen Beziehungen, beispielweise von dem Knoten B zu dem Knoten C. Diese Beziehungen werden als Kanten deklariert. Die algorithmische Graphentheorie lässt sich in zwei Funktionsweisen unterteilen. Einerseits gibt es die gerichteten Graphen, welche die einseitige Beziehung zwischen zwei Knoten darstellt (B→C). Andererseits existieren aber auch ungerichtete Graphen (A↔B). Ihre Eigenschaft liegt darin, dass die Kante zwischen den beiden Knoten in beide Richtungen verwendet werden kann. "Eine der wichtigsten Problemstellungen für gewichtete Graphen ist das Finden kürzester Wege" [3]. Der günstigste Weg kann der kürzeste, der schnellste bzw. der sicherste sein. In Kapitel 2.6 stelle ich den Vorreiter Algorithmus dieser besonderen Klasse vor, welcher durch die Nutzung von gewichteten Graphen den kürzesten Weg berechnen kann. Gerichtete Graphen werden durch die allgemeine Schreibweise so erklärt:

$$G = (V, E, \gamma) \quad 1.1$$

Ein gerichteter Graph ist ein 3-Tupel aus V, welches für die endliche Menge von Knoten steht. Die Menge der Kanten wird durch E beschrieben und eine dazugehörige Gewichtsfunktion, die mit der Variabel γ deklariert wird [4]. (E muss in dem Fall eine natürliche Zahl sein)

[3] Saake (2006) , S.442

[4] Saake (2006) , S.443

$$P = \{(v_1 , v_2),(v_2 ,v_3), \dots , (v_{n-1} ,v_n)\} \quad 1.2$$

P steht für das englische Wort path (Pfad). Ein Pfad ist eine Liste von aneinanderstoßenden Kanten, welche durch 1.1 gegeben sind.

$$w(P) = \sum_{(i=1)}^{(n-1)} \gamma\left(v_i, v_{(i+1)}\right) \quad 1.3$$

Durch die Aufsummierung der einzelnen Kantenlängen kann das Gewicht bzw. die Länge eines Pfades bestimmt werden. (siehe 1.3) "Die Distanz zweier Punkte d (u,v) ist nun das Gewicht des kürzesten Pfades von u nach v" [5]. Falls es jedoch zwischen zwei Punkten keinen kürzesten Weg gibt, gibt es zwei Möglichkeiten d (u,v) zu lösen. Die erste Variante ist: es gibt keinen Weg von u nach v und somit ist die Lösung der Gleichung $+\infty$. Die andere Variante geht davon aus, dass ein Weg zwischen u und v vorhanden ist, aber dieser nicht der kürzeste ist. Für diesen Fall erhalten wir als Lösung $-\infty$. Für die Beantwortung meiner Leitfrage ist ein Grundsatz von Bedeutung: Es können mehrere Pfade zwischen zwei Punkten, welche die gleiche Länge haben, existieren. Falls dieser Fall eintrifft, existiert kein kürzester Weg.

2.2 Greedy-Algorithmen

Das Grundprinzip der Greedy-Algorithmen ist, dass sie versuchen durch die Optimierung jedes Teilschritts so gierig wie möglich zu sein. Auf die Realität bezogen kann man durch die Verwendung solcher Algorithmen Arbeitszeit und Kosten sparen. Ein passender Realtitätsbezug, ist zum Beispiel die Herausgabe von Rückgeld. Bei der Herausgabe von 89 Cent, wäre die optimale Reihenfolge 89=50+20+10+5+2+2. Verallgemeinert wird vom Startwert in jedem Bearbeitungsschritt durch die optimale Lösung das Problem so gierig wie möglich gelöst. In unserem Beispiel wird die größtmögliche Münze abgezogen.

In der Algorithmik bauen in der Regel verschiedene Algorithmen aufeinander auf, um möglichst effektiv ein Problem zu lösen. Der Dijkstra Algorithmus arbeitet erst wenn alle Knoten und Kanten gegeben sind. Im Folgenden stelle ich zwei Durchsuchungsalgorithmen, die möglichst kostengünstig und gründlich alle Knoten und ihre Kanten erkunden können, vor.

[5] Saake (2006) , S.443

2.3 Breitensuche

Die Breitensuche (BFS: Breadth-first Search) ist ein einfacher Algorithmus für das Durchsuchen eines Graphen und gleichzeitig ein Vorbild für viele Graphenalgorithmen. "Für einen gegebenen Graphen G= (V, E) und einen ausgezeichneten Startknoten s erforscht die Breitensuche systematisch alle Kanten von G, um alle Knoten zu "entdecken", die von s aus erreichbar sind" [6]. Zwei weitere postive Aspekte sind, dass gleichzeitig die Distanz zwischen den Knoten berechnet wird und ein Breitensuchbaum mithilfe einer Adjazenmatrix erstellt, in dem alle erreichbaren Knoten enthalten sind, wird. Dieser Breitensuchbaum kann die Beziehungen zwischen den Vorgängern u und den Nachfahren v veranschaulichen. Des weiteren werden zunächst alle Knoten mit der Distanz k von s entdeckt, bevor irgendein anderer Knoten mit der Distanz k+1 erkundet werden kann. Diese Annahme erklärt auch den Namen der Breitensuche. Zusammenfassend ist die Breitensuche eine sehr logische und systematische Möglichkeit einen Graphen effizient zu erkunden und dessen Status zu ermitteln.

2.4 Tiefensuche

Die Tiefensuche (DFS:Depth-first Search) ist eher eine vorsichtige, konservative Strategie für das Durchsuchen eines Graphen. Es werden erst die Kanten, die von dem zuletzt entdeckten Knoten v ausgehen, welcher noch nicht vollständig erkundet wurde, untersucht. Sobald alle Kanten von v untersucht wurden, kehrt man zu dem Knoten von dem der Knoten v entdeckt wurde zurück. Dieses Prozedere wird solange fortgesetzt bis alle Knoten entdeckt wurden. Falls Knoten unentdeckt bleiben, wird ein neuer Startknoten, bis man alle Knoten entdeckt hat, festgelegt. Der große Unterschied zwischen der Breitensuche und der Tiefensuche ist, dass bei der Tiefensuche mehrere Vorgängerteilgraphen erstellt werden [7]. Das bedeutet, dass man mehrere Bäume hat um die Verbindungen nachzuvollziehen. Summa summarum ist die Tiefensuche ein konservativer, eher umständlicher Durchsuchungsalgorithmus, welcher nicht so praxisnah ist, wie die Breitensuche .

---

[6] Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Algorithmen – eine Einführung. Oldenbourg, München, Wien 2004, (Originaltitel: Introduction to algorithms. Übersetzt von Karen Lippert, Micaela Krieger-Hauwede), S.603

[7] Vgl.: Cormen (2004) , S. 613

2.5 Adjazenzmatrix-Darstellung

Die Adjazenzmatrix-Darstellung ist eine Vereinfachung zur Darstellung von Graphen. Wir setzen voraus, dass die Knoten in beliebiger Weise von 1 bis |V| nummeriert sind. Für den Graphen G, welcher aus einer |V| × |V|-Matrix besteht, gilt $A = (a_{ij})$ [8] . Für diese Matrix gibt es zwei Lösungsansätze: Entweder gibt es eine Verbindung zwischen i und j dann wird in die Matrix eine 1 geschrieben oder es gibt keine Verbindung. In diesem Fall schreibt man in der Regel eine 0.

2.6 Algorithmus von Dijkstra

Zur Lösung meiner Leitfrage habe ich für den Dijkstra Algorithmus zur Weg-Optimierung entschieden. "Der Algorithmus von Dijkstra als erster Wegoptimierungs-Algorithmus ist die Basis für die meisten Algorithmen, wie z.B. der Bellman-Ford-Algorithmus, Algorithmus von Floyd und Warshall und A* und dessen Erweiterung IDA*" [9]. Dieser Wegoptimierungs-Algorithmus basiert auf seinen Erfinder Edsger W. Dijkstra. Der Niederländer, Dijkstra gilt als Wegbereiter der strukturierten Informatik, welche in den 1970er stark an Popularität gewonnen hat. Des weiteren gehört der Algorithmus zur Klasse der Greedy-Algorithmen, ist verwandt mit der Breitensuche und lässt sich durch eine Adjazenzmatrix bzw. Adjazenzliste darstellen [10]. Die Relevanz des Dijkstra Algorithmus wird heutzutage noch klar, dadurch das der IDA* Algorithmus für die professionelle Spieleentwicklung und die Entwicklung künstlicher Intelligenz verwendet wird. Im folgenden werde ich die einzelnen Arbeitsschritte des Algorithmus erklären [11]:

1. Man initialisiert alle Knoten mit der Distanz „unendlich“.
2. Der Startknoten wird auf die Warteliste gesetzt.
3. Die Distanz zu dem Startknoten ist 0.
4. Der Knoten mit der kleinsten Distanz wird auf die Warteliste gesetzt.
5. Distanz des Nachbarknoten = Distanz des aktuellen Knoten + Verbindung zwischen Nachbar- und aktuellem Knoten.

---

[8] Cormen (2004) , S. 601

[9] Phillip Erler, Wegoptimierung mit dem Dijkstra-Algorithmus https://www.pplusplus.lima-city.de/lib/data/wegoptimierung/Wegoptimierung.pdf , 20.02.2019 , S.4

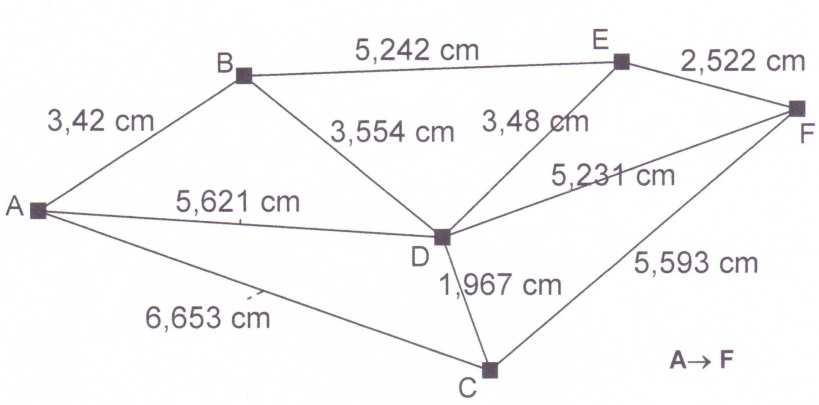
[10] Vgl.: Maurice Duvigneau, Kürzeste Wege in Graphen , http://fuzzy.cs.ovgu.de/studium/graph/txt/duvigneau.pdf , 20.02.2019 S.6

[11] Vgl. (Schema): Saake (2006), S.443ff

6. Wenn eine so berechnete Distanz kleiner ist, als der vorhandene Wert, dann wird diese neue Distanz beibehalten.
7. Alle benachbarten Knoten werden auf die Warteliste gesetzt, wenn sie noch nicht dran sind.
8. Schritt 4 bis 7 wird so oft wiederholt bis alle Knoten auf der Warteliste waren.

Nach dem Durchlaufen des Algorithmus ist der kürzeste Weg zu jedem Knoten vom Startknoten bekannt.

2.6.1 Beispiel anhand des Dijkstra-Algorithmus



In meinem Beispiel sollte der kürzeste Weg von A nach F ermittelt werden. Die Grafik wurde durch EUKLID Dynageo erstellt und die Kantengewichte dadurch berechnet. Durch die Erstellung einer Prioritätswarteschlange Q= (x: y) konnte der Algorithmus einfach auswertet werden. X steht in diesem Falle für den Namen des ausgewählten Knoten und Y für die Distanz [12]. Damit man weiß von welchem Knoten aus die Kantengewichte betrachtet werden, verändere ich die Prioritätswarteschlange ein wenig: Q(i) = (x: y). Das i steht für den Knoten bei dem man momentan ist.

Q= (A:0), (B:∞), (C:∞), (D:∞), (E:∞), (F:∞)

Ich starte beim Punkt A und gebe für die Distanz eine 0 an. Da ich mich in der Initialphase befinde sind alle anderen Punkte ∞ weit entfernt.

Q(A)= (B:3,42), (C:6,653), (D:5,621), (E:∞), (F:∞)

In der ersten Phase gibt es drei Kanten vom Knoten A, um andere Knoten zu erreichen. Ich habe jetzt die Möglichkeit von A aus zu B, C und zu D zu kommen.

[12] Saake (2006) , S.444ff

Q(B)= (C:6,653), (D:5,621), (E:8,662), (F:∞)

Nun nehme ich B als nächsten Knoten, da er die kürzeste Kante besitzt. Von B aus kann man nun den Knoten E erreichen. Der Wert von C verändert sich nicht, weil es keine Verbindung zwischen B und C gibt. Außerdem übernimmt man den Wert von D, da die Verbindung über B länger ist mit 6,974 als die Verbindung von A nach D.

Q(D)= (C:6,653), (E:8,662), (F:10,852)

Der nächst kürzeste Knoten ist D. Die Verbindungen von D nach C und E sind länger als die Verbindungen, welche ich bereits ermittelt habe, deshalb ändert sich an dem Wert nichts. Außerdem können wir erstmals den Zielknoten F erreichen. Dieser muss aber zwangsweise nicht der kürzeste sein und muss deswegen weiter geprüft werden.

Q(C)= (E:8,662), (F:10,852)

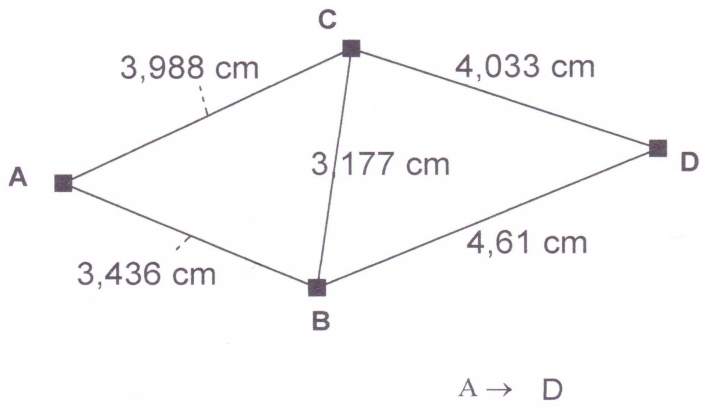
Über den Knoten C kann man nicht zu den Knoten E gelangen, darum bleibt der momentane Wert bestehen. Die zweite Verbindung um nach F zu gelangen ist mit 12,246 länger als die erste und ist somit nicht zielführend.

Q(E)= (F:10,852)

Der kürzeste Weg von A nach F ist mit 10,852km über den Knoten D (A→D→F).

2.6.2 Erklärung Notationssystem

Meiner Meinung nach, sind die kommerziellen Notationssysteme unübersichtlich und haben einen großen Nachteil. Man kann am Ende leicht das Ergebnis vom Startknoten zum Endknoten ablesen, sieht aber nicht die kürzesten Verbindungen auf dem Weg zu anderen Knoten. Dazu habe ich mir ein eigenes Notationssystem ausgedacht und werde es an einem leichten Beispiel erläutern.



Wie schon der Zeichnung entnommen werden kann, soll von dem Knoten A kostengünstig zu dem Knoten D gelangt werden.

Markiert	A	B	C	D
A	0A	3,436A	3,988A	∞
B		3,436A	3,988A	8,064B
C			3,988A	8,021C
D				8,021C

Diese Excel-Tabelle berechnet den kürzesten Weg. Diese Dastellungsweise hat einige Vorteile zur üblichen Notation. Der erste Vorteil wäre, dass man direkt erkennen kann welcher Knoten markiert ist. Außerdem wird durch die eingegrauten Zellen deutlich kenntlich gemacht, für welchen Knoten bereits der kürzeste Weg herausgefunden wurde. Der kürzeste Weg zwischen den einzelnen Knoten kann somit leicht abgelesen werden. Ein anderer Vorteil ist, dass nun erkennbar ist, wie ein bestimmer Knoten errericht werden konnte. Das kann beispielsweise durch die Bezeichung 8,064B nachvollzogen werden. Es ist erkennbar, dass über den Knoten B zu D gelangt wurde. Dies kann noch weiter bis zum Ursprung verfolgt und die komplette Route übersichtlich dargestellt werden. Das Ergebnis lautet 8,021C. Die optimierte Zielroute für das obige Beispiel lautet somit: A→C→D und hat die Strecke 8,021km.

3. Die kürzeste Strecke vom Gymnasium Hohenlimburg nach Stuttgart

Im folgenden Kapitel werde ich mich explizit mit meiner Leitfrage beschäftigen. Für die Beantwortung dieser Frage werde ich mich auf meine bereits erarbeiteten Methoden stützen.

3.1 Vorbereitung und Vorgehen

Um eine Frage dieser Größenordnung zu beantworten, verlangt es nach einer geregelten Vorbereitung. Dazu habe ich mir einige Regeln ausgedacht, um den Dijkstra Algorithmus zu modellieren. Die erste ist, dass ich immer vom Stadtzentrum als Knoten für meine Berechnung ausging. Die Strecken zwischen den einzelnen Knoten/Städte habe ich durch Google Maps errechnen lassen. Dabei habe ich immer die kürzeste Strecke berechnen lassen, weswegen der Spritverbrauch und die dafür benötigte Zeit irrelevant sind. Ein weiterer Punkt ist, dass ich bei meiner Berechnung von keinem Stau ausgegangen bin. Diese Überlegung würde das Verfahren sehr komplex und zeitpunktabhängig machen, da es durch Unfälle und Umwelteinflüsse zu temporären Stau kommen könnte.

Es gibt nur Direktverbindungen zwischen zwei Städten. Wie schon in 2.1 erwähnt, kann es dazu kommen das zwei gleich lange Verbindungen zwischen zwei Städten erfasst wurden. Dieser Fall ist bei meinen Berechnungen jedoch nicht eingetroffen. In Deutschland ist es teilweise auf der Autobahn erlaubt mit unbegrenzter Geschwindigkeit zu fahren. Für diesen Fall ging ich von der Richtgeschwindigkeit von 130 km/h aus.

Da ich die Strecke im Rahmen einer wissenschaftlichen Arbeit berechnete, werde ich mein Vorgehen genau beschreiben und durch meine Anlagen im Anhang dokumentieren. Mein erster Arbeitsschritt ist das Finden potentieller Kandidaten für Knoten. Dabei habe ich alle Städte die etwa auf dem Weg liegen und eine Einwohnerzahl von mindestens 200.000 Menschen als Knoten deklariert. Um dies ein wenig anschaulicher darzustellen, habe ich eine Deutschlandkarte mit dem deutschen Autobahnnetz und den ausgewählten Kandidaten angefertigt. Jede Stadt bzw. Kandidat hat dazu einen eigenen Buchstaben zur Vereinfachung bekommen. Mein nächster Schritt war es die einzelnen Stadtverbindungen in einer Datenbank überschaubar darzustellen und die Distanz auszurechnen. Diese Datenbank ist eine Modifizierung der Adjazenzmatrix. Meine Überlegung dahinter war, dass ich für die Verbindungen den tatsächlichen Wert eingebe und falls es keine sinnvolle Verbindung zwischen zwei Städten gibt, der Übersichtlichkeit halber ein X eintrage.

Wenn ich durch eine Verbindung weiter in den Norden Deutschlands kommen würde, wäre die Kante ebenfalls mit einem X zu deklarieren.

3.2 Auswertung

Um meine Leitfrage abschließend zu beantworten, war es erforderlich den Dijkstra Algorithmus auf mein Fallbeispiel anzuwenden. Dafür habe ich das Tabellensytem welches ich in 2.6.2 bereits schon erläutert habe, verwendet und bin auf folgendes Ergebnis gestoßen (siehe Anhang):

Hohenlimburg	A
Gummersbach	B
Siegen	L
Frankfurt am Main	O
Darmstadt	Q
Heidelberg	R

Heilbronn	U
Stuttgart	W

Die Strecke der Route beträgt 424km. Meine Überlegung war, dass ich mein Ergebnis mit einem ethablierten Routenplaner vergleichen müsse um eine Vorstellung darüber zu bekommen, wie gut meine Modellierung war. Laut Google Maps ist die kürzeste Route 405km lang. Das entspricht einer prozentualen Abweichung von 4,48%. Persönlich stellte sich mir die Frage woran es gelegen hat, dass diese Abweichung so hoch war. Dafür gab es eigentlich nur eine logische Erklärung, da sich die Route von Google Maps auch an den Städten orientiert hat. Durch die Annahme, dass das Stadtzentrum angefahren werden muss, ensteht unnötig gefahrene Strecke so als wenn man auf der Autobahn bleiben würde. Die Angaben der Kantenlängen führen wahrscheinlich nur minimal zu dieser Ungenauigkeit, weil diese nur mit einer Nachkommastelle notiert wurde.

3.3 Grenzen des Dijkstra Algorithmus

Ein Algorithmus ist eine sehr komplexe mathetische Arbeitsanweisung zur Modellierung von realen Problemen. Die Realtität ist jedoch so komplex, dass ein Algorithmus nicht das Problem im vollen Maße erfüllen kann. Ich möchte in diesem Abschnitt die Grenzen des Dijkstra Algorithmus erläutern. Das erste Problem ist, dass die Kantengewichte immer postiv sein müssen. Wenn teilweise die Kantengewichte negativ sind, muss man den Bellman-Ford-Algorithmus verwenden [13]. Ein weiteres Problem könnte bei größeren Strecken wie zum Beispiel von Deutschland nach China sein, dass wir zu viele Knoten und somit noch mehr unnötige Kanten haben. Dadurch würde die Berechnung durch Rechenmaschienen, welche im Vergleich zum Menschen sehr schnell sind, sehr viel Zeit in Anspruch nehmen (Zeitkomplexität) [14]. Die Lösung für dieses Problem könnte beispielweise der IDA*-Algorithmus schaffen. Außerdem müssen die Kantengewichte und die Knoten konstant bleiben. Wenn sich diese rein theoretisch sich verändern würden, kann der Dijkstra Algorithmus nicht auf diese Veränderung reagieren.

[13] Cormen (2004), S. 663

[14] Martin Dietzfelbinger, Kurt Mehlhorn, Peter Sanders: "Algorithmen und Datenstrukturen, Die Grundwerkzeuge", S.255

3.4 Alternativen zum Dijkstra Algorithmus

Wie schon in 3.3 erwähnt, hat auch der Dijkstra Algorithmus seine Probleme. Der Dijkstra Algorithmus gilt als Vorreiter Algorithmus für den kürzesten Pfad. Im Laufe der Geschichte der Graphenalgorithmen sind sehr viele Erweiterungen durch den Dijkstra Algorithmus entstanden. Ihre Erfinder haben es sich zur Aufgabe gemacht die Probleme des Dijkstra Algorithmus auszubessern und dafür Lösungen zu finden, diese auch zu programmieren. Eine grobe Übersicht über die Algorithmen welche in den letzten Jahrzehnten entwickelt wurden, um diesen Sachverhalt zu lösen, wurde von mir im Anhang aufgestellt. Ich möchte hier nur die wichtigsten kurz benennen und ihren Grundgedanken wiedergeben. Heutzutage ist der IDA* von besonderer Bedeutung, da er wie schon in 2.6 erwähnt für die professionelle Spieleentwicklung und künstliche Intelligenz verwendet wird. Der IDA* verwendet eine beschränkte Tiefensuche mit steigenden Schranken, dadurch werden die positiven Eigenschaften der Tiefensuche mit der Breitensuche verknüpft [15]. Der andere Algorithmus den ich als besonders wichtig empfinde, ist der Bellman-Ford Algorithmus. Dieser Algorithmus prüft im Gegensatz zum Dijkstra Algorithmus einige Male bis er zu einem Ergebnis kommt und kann deswegen mit negativen Kantengewichten umgehen. Sein großer Nachteil ist, dass er durch die vielen Durchläufe bei größeren Datenmengen eine Menge Zeit in Anspruch nimmt (Zeitkomplexität).

4. Ausblick: Pseudocode des Dijkstra Algorithmus

Definition des Pseudocode: "Hilfsmittel bei der Programmentwicklung zur verbalen Formulierung eines Algorithmus oder der Rohform eines Programms. Die Darstellungsform orientiert sich an der Schreibweise einer Programmiersprache" [16].

Man entwickelt Algorithmen um durch maschienelles Lernen, diese nachhaltig in einem Programm als Lösung des Problems zu haben. In einem Pseudocode sind überlicherweise noch Kommentare die diesen erklären. In dem Code sind die Kommentare durch ein // ,wie die Blockkommentare in der Programmiersprache Java, vom tatsächlichen Code getrennt.

---

[15] Vgl.: Malte Helmert, Grundlagen der Künstlichen Intelligenz 14. Klassische Suche: IDA, https://ai.dmi.unibas.ch/_files/teaching/fs14/ki/slides/ki14.pdf , 20.02.2019, S.5

[16] Prof. Dr. Richard Lackes, Gabler Wirtschaftslexikon, https://wirtschaftslexikon.gabler.de/definition/pseudocode-46714, 20.02.2019

In einem Quellcode haben diese Kommentare keine Funktion. Sie dienen nur dem Programmierer für das Verstehen des Codes. Ein Beispiel für solch einen Pseudocode könnte so aussehen [17].

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:             // Initialization
        dist[v] ← INFINITY                  // Unknown distance from source to v
        prev[v] ← UNDEFINED                 // Previous node in optimal path from source
        add v to Q                          // All nodes initially in Q (unvisited nodes)

    dist[source] ← 0                        // Distance from source to source

    while Q is not empty:
        u ← vertex in Q with min dist[u]    // Source node will be selected first
        remove u from Q

        for each neighbor v of u:           // where v is still in Q.
            alt ← dist[u] + length(u, v)
            if alt < dist[v]:               // A shorter path to v has been found
                dist[v] ← alt
                prev[v] ← u

    return dist[], prev[]
```

Erstmal wird der Graph initialisiert. Danach werden alle unbekannten Distanzen vom Startknoten zu n-beliebigen Knoten auf die Distanz von ∞ gesetzt. Daraufhin erstellt man ein Netzwerk aus den Kanten und Knoten. Der nächste Arbeitsschritt setzt die Distanz des Startknotens auf 0. Wenn diese Schritte durchgelaufen sind, wird der eigentlichen Dijkstra Algorithmus mit den mathematischen Vorrausetzungen aus 2.6 programmiert. Durch die Digitalisierung des Dijkstra Algorithmus können auch bei größeren Netzwerke bzw. Breitensuchbäume kostengünstig und schnell der kürzeste Weg bestimmt werden. Der Endverbraucher hat dadurch eine leicht zu verstehende Applikation und kann nun auch selber die kürzesten Wege berechnen. Dadurch hat die Mathematik es geschafft dieses Problem zu lösen.

[17] Sylvain Saurel, Calculate shortest paths in Java by implementing Dijkstra's Algorithm, https://medium.com/@ssaurel/calculate-shortest-paths-in-java-by-implementing-dijkstras-algorithm-5c1db06b6541, 20.02.2019

5. Erklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt habe und nur die im Literatur- und Quellenverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.

X

Michael Wigond

X. Literatur

Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Algorithmen – eine Einführung. Oldenbourg, München, Wien 2004, ISBN 3-486-27515-1, S. 598–604 (Originaltitel: Introduction to algorithms. Übersetzt von Karen Lippert, Micaela Krieger-Hauwede).

Martin Dietzfelbinger, Kurt Mehlhorn, Peter Sanders: "Algorithmen und Datenstrukturen, Die Grundwerkzeuge", Heidelberg 2014

Edsger W. Dijkstra: A note on two problems in connexion with graphs. In: Numerische Mathematik.1, 1959, S. 269–271

Gunter Saake, Kai-Uwe Sattler: Algorithmen und Datenstrukturen- eine Einführung mit Java, Heidelberg 2006

Internetquellen:

* Bild entnommen von Google Maps, Gymnasium Hohenlimburg nach Stuttgart, https://www.google.com/maps/dir/Gymnasium+Hohenlimburg,+Wiesenstra%C3%9Fe+27,+58119+Hagen/Stuttgart/@49.8143427,8.8171965,7z/data=!4m14!4m13!1m5!1m1!1s0x47b93ecf0139feff:0xdc44877c871192c5!2m2!1d7.5706011!2d51.3629534!1m5!1m1!1s0x4799db34c1ad8fd3:0x79d5c11c7791cfe4!2m2!1d9.1829321!2d48.7758459!3e0, 26.02.2019

Mixmaps Homepage, https://www.mixmaps.de/deutschland/karte.html, 26.02.2019

Ron Gutman, "How does the algorithm of Google Maps work?", https://www.quora.com/How-does-the-algorithm-of-Google-Maps-work, 26.02.2019

Maurice Duvigneau, Kürzeste Wege in Graphen, http://fuzzy.cs.ovgu.de/studium/graph/txt/duvigneau.pdf , 26.02.2019

Phillip Erler, Wegoptimierung mit dem Dijkstra-Algorithmus, https://www.pplusplus.lima-city.de/lib/data/wegoptimierung/Wegoptimierung.pdf , 26.02.2019

Malte Helmert, Grundlagen der Künstlichen Intelligenz 14. Klassische Suche: IDA, https://ai.dmi.unibas.ch/_files/teaching/fs14/ki/slides/ki14.pdf , 26.02.2019

Prof. Dr. Richard Lackes, Gabler Wirtschaftslexikon, https://wirtschaftslexikon.gabler.de/definition/pseudocode-46714 , 26.02.2019

Sylvain Saurel, Calculate shortest paths in Java by implementing Dijkstra's Algorithm, https://medium.com/@ssaurel/calculate-shortest-paths-in-java-by-implementing-dijkstras-algorithm-5c1db06b6541, 26.02.2019

Website der TU München, Der Dijkstra-Algorithmus - TUM - Mathematik – M9, https://www-m9.ma.tum.de/graph-algorithms/spp-dijkstra/index_de.html, 26.02.2019

Website der TU München, Der A*-Algorithmus - TUM - Mathematik – M9, https://www-m9.ma.tum.de/graph-algorithms/spp-a-star/index_de.html, 26.02.2019

Falls die Internetseiten nicht mehr online sind, können diese der beigelegten Datei des Datenträgers, unter "Internetverzeichnis", entnommen werden. Dort sind alle Internetseiten als PDF-Datei hinterlegt.

Anhang zu 2.6.1

Markiert	A	B	C	D	E	F
A	0A	3,420A	6,653A	5,621A	∞	∞
B		3,420A	6,653A	5,621A	8,662B	∞
D			6,653A	5,621A	8,662B	10,852D
C			6,653A		8,662B	10,852D
E					8,662B	10,852D
F						10,852D

Zusätzlich habe ich die Aufgabe in einer Tabelle ausgerechnet.

Anhang zu 3.1

Hagen	A
Gummersbach	B
Wuppertal	C
Neuss	D
Solingen	E
Remscheid	F
Düsseldorf	G
Leverkusen	H
Köln	I
Bergisch Gladbach	J
Bonn	K
Siegen	L
Gießen	N
Koblenz	M
Wiesbaden	N
Frankfurt	O
Mainz	P
Darmstadt	Q
Heidelberg	R
Kaiserslautern	S
Kalsruhe	T
Heilbronn	U
Pforzheim	V
Stuttgart	W

Jeder Stadt wurde ein Buchstabe zugeordnet. Dadurch wird die Dijkstra Tabelle deutlich übersichtlicher.

Dies ist eine Deutschlandkarte mit meinem Knotenpunkten und dem deutschen Autobahnnetz. Auf den folgenden Seiten ist eine Datenbank mit den Kantengewichten der einzelnen Verbindungen aufgeführt, welche mit Google Maps berechnet wurde.

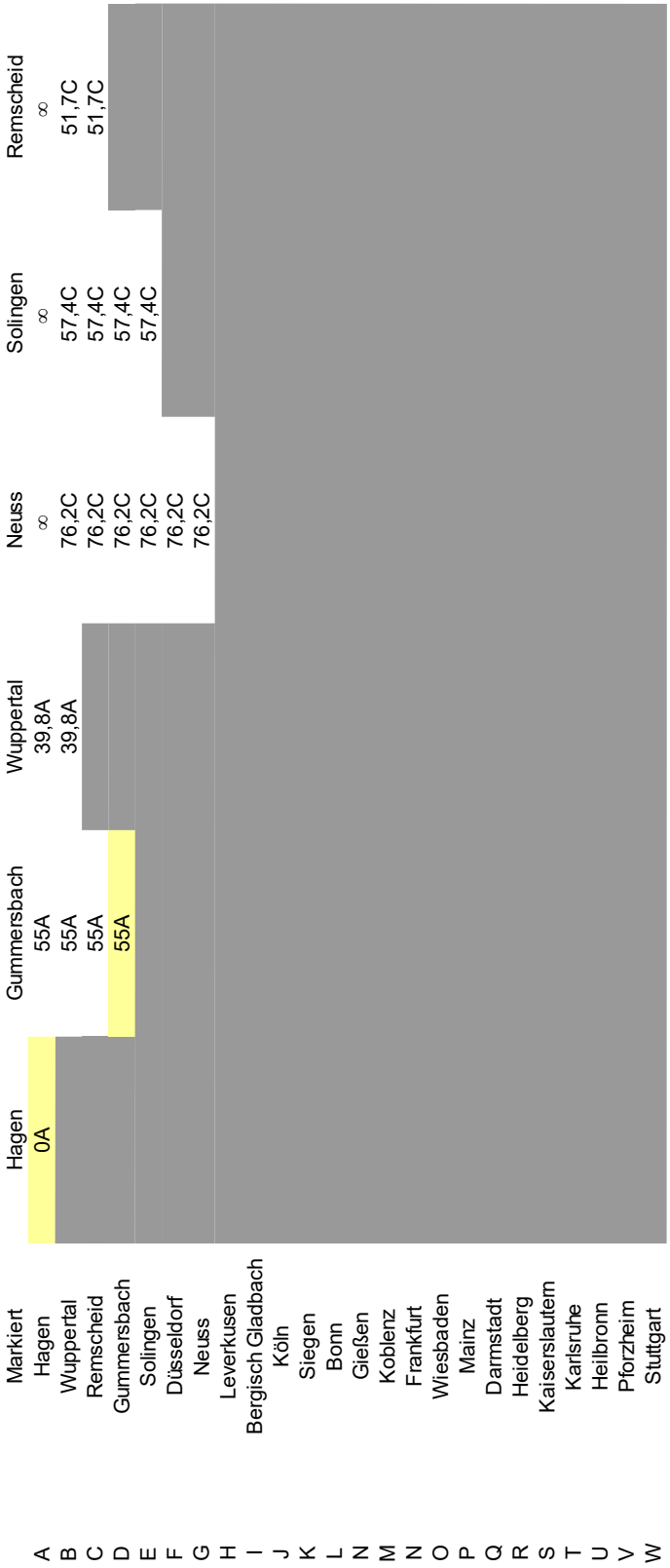
	Hagen	Gummersbach	Wuppertal	Neuss	Solingen	Remscheid	Düsseldorf	Leverkusen
Hagen	0	X	X	X	X	X	X	X
Gummersbach	55	0	X	90,7	56,3	43,8	X	X
Wuppertal	39,8	X	0	X	X	X	X	X
Neuss	X	X	36,4	0	X	X	10,3	X
Solingen	X	X	17,6	35,3	0	12,2	30,5	X
Remscheid	X	X	11,9	47,6	X	0	43,6	X
Düsseldorf	X	X	35,5	X	X	X	0	X
Leverkusen	X	X	X	57,7	26,4	25,8	57,8	0
Köln	X	X	X	39,6	X	X	X	15,9
Bergisch Gladbach	X	41,2	X	X	X	31,6	X	14,5
Bonn	X	60,5	X	X	X	X	X	X
Siegen	X	49,6	X	X	X	87,3	X	X
Gießen	X	X	X	X	X	X	X	X
Koblenz	X	X	X	X	X	X	X	X
Wiesbaden	X	X	X	X	X	X	X	X
Frankfurt	X	X	X	X	X	X	X	X
Mainz	X	X	X	X	X	X	X	X
Darmstadt	X	X	X	X	X	X	X	X
Heidelberg	X	X	X	X	X	X	X	X
Kaiserlautern	X	X	X	X	X	X	X	X
Kalsruhe	X	X	X	X	X	X	X	X
Heilbronn	X	X	X	X	X	X	X	X
Pforzheim	X	X	X	X	X	X	X	X
Stuttgart	X	X	X	X	X	X	X	X

	Köln	Bergisch Gladbach	Bonn	Siegen	Gießen	Koblenz	Wiesbaden	Frankfurt
Hagen	X	X	X	X	X	X	X	X
Gummersbach	X	42,7	X	X	X	X	X	X
Wuppertal	X	X	X	X	X	X	X	X
Neuss	X	X	X	X	X	X	X	X
Solingen	X	X	X	X	X	X	X	X
Remscheid	X	X	X	X	X	X	X	X
Düsseldorf	X	X	X	X	X	X	X	X
Leverkusen	X	X	X	X	X	X	X	X
Köln	0	15,4	X	X	X	X	X	X
Bergisch Gladbach	X	0	X	X	X	X	X	X
Bonn	28,8	X	0	X	X	X	X	X
Siegen	X	X	X	0	X	X	X	X
Gießen	X	X	138	78,1	0	X	X	X
Koblenz	109	X	90,1	104	105	0	X	X
Wiesbaden	X	X	X	122	64,7	72,1	0	38,9
Frankfurt	X	X	X	131	X	X	39,1	0
Mainz	X	X	X	X	X	86,5	13,2	43,6
Darmstadt	X	X	X	X	X	X	46,2	33,3
Heidelberg	X	X	X	X	X	X	X	X
Kaiserlautern	X	X	X	X	X	X	X	X
Kalsruhe	X	X	X	X	X	X	X	X
Heilbronn	X	X	X	X	X	X	X	X
Pforzheim	X	X	X	X	X	X	X	X
Stuttgart	X	X	X	X	X	X	X	X

	Mainz	Darmstadt	Heidelberg	Kaiserslautern	Kalsruhe	Heilbronn	Pforzheim	Stuttgart
Hagen	X	X	X	X	X	X	X	X
Gummersbach	X	X	X	X	X	X	X	X
Wuppertal	X	X	X	X	X	X	X	X
Neuss	X	X	X	X	X	X	X	X
Solingen	X	X	X	X	X	X	X	X
Remscheid	X	X	X	X	X	X	X	X
Düsseldorf	X	X	X	X	X	X	X	X
Leverkusen	X	X	X	X	X	X	X	X
Köln	X	X	X	X	X	X	X	X
Bergisch Gladbach	X	X	X	X	X	X	X	X
Bonn	X	X	X	X	X	X	X	X
Siegen	X	X	X	X	X	X	X	X
Gießen	X	X	X	X	X	X	X	X
Koblenz	X	X	X	X	X	X	X	X
Wiesbaden	X	X	X	X	X	X	X	X
Frankfurt	X	X	X	X	X	X	X	X
Mainz	0	X	X	X	X	X	X	X
Darmstadt	40,3	0	X	X	X	X	X	X
Heidelberg	X	57,9	0	88,1	X	X	X	X
Kaiserlautern	80,2	105	X	0	X	X	X	X
Kalsruhe	141	X	53,1	125	0	X	X	X
Heilbronn	X	X	67,9	X	X	0	X	X
Pforzheim	X	X	X	X	28,5	55,8	0	X
Stuttgart	X	X	X	X	X	53,3	49,3	0

Anhang zu 3.2

Im folgenden ist meine Tabelle zur Berechnung des kürzesten Weg dargestellt.



	Markiert	Hagen	Gummersbach	Wuppertal	Neuss	Solingen	Remscheid
A	Hagen	0A	55A	39,8A	∞	∞	∞
B	Wuppertal		55A	39,8A	76,2C	57,4C	51,7C
C	Remscheid		55A		76,2C	57,4C	51,7C
D	Gummersbach		55A		76,2C	57,4C	
E	Solingen				76,2C	57,4C	
F	Düsseldorf				76,2C		
G	Neuss				76,2C		
H	Leverkusen						
I	Bergisch Gladbach						
J	Köln						
K	Siegen						
L	Bonn						
N	Gießen						
M	Koblenz						
N	Frankfurt						
O	Wiesbaden						
P	Mainz						
Q	Darmstadt						
R	Heidelberg						
S	Kaiserslautern						
T	Karlsruhe						
U	Heilbronn						
V	Pforzheim						
W	Stuttgart						

	Markiert	Düsseldorf	Leverkusen	Köln	Bergisch Gladbach	Bonn	Siegen
A	Hagen	∞	∞	∞	∞	∞	∞
B	Wuppertal	75,3C	∞	∞	∞	∞	∞
C	Remscheid	75,3C	77,5F	∞	83,3F	∞	139F
D	Gummersbach	75,3C	77,5F	∞	83,3F	115,5B	104,6B
E	Solingen	75,3C	77,5F	∞	83,3F	115,5B	104,6B
F	Düsseldorf	75,3C	77,5F	∞	83,3F	115,5B	104,6B
G	Neuss		77,5F	115,8D	83,3F	115,5B	104,6B
H	Leverkusen		77,5F	93,4H	83,3F	115,5B	104,6B
I	Bergisch Gladbach			93,4H	83,3F	115,5B	104,6B
J	Köln			93,4H		115,5B	104,6B
K	Siegen					115,5B	104,6B
L	Bonn					115,5B	
N	Gießen						
M	Koblenz						
N	Frankfurt						
O	Wiesbaden						
P	Mainz						
Q	Darmstadt						
R	Heidelberg						
S	Kaiserslautern						
T	Karlsruhe						
U	Heilbronn						
V	Pforzheim						
W	Stuttgart						

	Markiert	Gießen	Koblenz	Wiesbaden	Frankfurt	Mainz	Darmstadt
A	Hagen	∞	∞	∞	∞	∞	∞
B	Wuppertal	∞	∞	∞	∞	∞	∞
C	Remscheid	∞	∞	∞	∞	∞	∞
D	Gummersbach	∞	∞	∞	∞	∞	∞
E	Solingen	∞	∞	∞	∞	∞	∞
F	Düsseldorf	∞	∞	∞	∞	∞	∞
G	Neuss	∞	∞	∞	∞	∞	∞
H	Leverkusen	∞	∞	∞	∞	∞	∞
I	Bergisch Gladbach	∞	∞	∞	∞	∞	∞
J	Köln	∞	202,4I	∞	∞	∞	∞
K	Siegen	182,7L	202,4I	226,6L	209,6L	∞	∞
L	Bonn	182,7L	202,4I	226,6L	209,6L	∞	∞
N	Gießen	182,7L	202,4I	226,6L	209,6L	∞	∞
M	Koblenz		202,4I	226,6L	209,6L	288,9M	∞
N	Frankfurt			226,6L	209,6L	253,2O	242,9O
O	Wiesbaden			226,6L		239,8N	242,9O
P	Mainz					239,8N	242,9O
Q	Darmstadt						242,9O
R	Heidelberg						
S	Kaiserslautern						
T	Karlsruhe						
U	Heilbronn						
V	Pforzheim						
W	Stuttgart						

	Markiert	Heidelberg	Kaiserslautern	Kalsruhe	Heilbronn	Pforzheim	Stuttgart
A	Hagen	∞	∞	∞	∞	∞	∞
B	Wuppertal	∞	∞	∞	∞	∞	∞
C	Remscheid	∞	∞	∞	∞	∞	∞
D	Gummersbach	∞	∞	∞	∞	∞	∞
E	Solingen	∞	∞	∞	∞	∞	∞
F	Düsseldorf	∞	∞	∞	∞	∞	∞
G	Neuss	∞	∞	∞	∞	∞	∞
H	Leverkusen	∞	∞	∞	∞	∞	∞
I	Bergisch Gladbach	∞	∞	∞	∞	∞	∞
J	Köln	∞	∞	∞	∞	∞	∞
K	Siegen	∞	∞	∞	∞	∞	∞
L	Bonn	∞	∞	∞	∞	∞	∞
N	Gießen	∞	∞	∞	∞	∞	∞
M	Koblenz	∞	∞	∞	∞	∞	∞
N	Frankfurt	∞	∞	∞	∞	∞	∞
O	Wiesbaden	∞	∞	∞	∞	∞	∞
P	Mainz	∞	320P	380.8P	∞	∞	∞
Q	Darmstadt	300,8Q	320P	380.8P	∞	∞	∞
R	Heidelberg	300,8Q	320P	353,9R	368,7R	∞	∞
S	Kaiserslautern		320P	353,9R	368,7R	∞	∞
T	Karlsruhe			353,9R	368,7R	382,4T	∞
U	Heilbronn				368,7R	382,4T	424U
V	Pforzheim					382,4T	424U
W	Stuttgart						424U

Y. Anhang zu 3.4

Die Mathematik hat viele Methoden, um auf eine Lösung zu kommen. Dieses Schaubild zeigt wie viele Graphalgorithmen für das Finden des kürzesten Weges existieren und Erweiterungen für die bereits vorhandene Algorithmen entwickelt wurden.