UNIVERSITÄT
MANNHEIM

Seminar:
Mathematische Modellierung (B.Sc.)

Abschlussbericht

Optimale Routenplanung bei der Müllabfuhr

Minh Duc Bui

Mannheim
23. Dezember 2018

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich diese Seminararbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen-Hilfsmittel verwendet habe, und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Arbeit nicht, auch nicht auszugsweise, bereits für andere Prüfungen ausgefertigt wurde.

Minh Duc Bui

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the LP WIMA chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Minh Duc Bui

Inhaltsverzeichnis

1 Einleitung und Motivation

In jeder Stadt entstehen Abfälle, die beseitigt werden müssen. Bei jeder Fahrt der Müllabfuhr entstehen Kosten, die zum größten Teil von der Fahrtzeit abhängen. Diese Kosten wollen wir nun minimieren indem wir die schnellst mögliche Route befahren. Doch wie kann man diese mathematisch berechnen?
Allein in Mannheim wurden 108.269 Tonnen Müll (Biomüll, Grünabfälle und Papier) aus privaten Haushälter und Geschäften entsorgt, dass heißt 355 kg/Einwohner und Jahr oder auch ca. 500 Tonnen täglich! Am Ende der Arbeit wollen wir für die Müllabfuhr in einem Stadtteil in Mannheim eine optimale Route erstellen um so die Kosten der Müllbeseitigung minimieren [Quelle].

2 Beschreibung des Modells

2.1 Kriterien der Route

Bei der Erstellung einer Route der Müllabfuhr müssen verschiedene Kriterien erfüllt sein. Es müssen alle Straßen nur (mindestens) einmal befahren werden, da Müllfahrzeuge auf beiden Straßenseite den Müll auf einmal entsorgen können. Außerdem müssen wir Einbahnstraßen und Sackgassen, die in beiden Richtungen befahren werden, beachten. Das letzte Kriterium soll sein, dass unser Mullfahrzeug von der Müllhalde und wieder zurück zur selben Müllhalde fährt, dass heißt der Startpunkt und Endpunkt der Fahrt soll gleich sein. Wir nehmen an, dass wir nur ein einzelnes Müllfahrzeug besitzen, weshalb wir annehmen müssen, dass wir ein Straßennetz haben, wo wir jede Straße erreichen können.
Wir wollen in dieser Arbeit nur Straßennetze betrachten, die keine Einbahnstraßen besitzen um unser mathematisches Modell zu vereinfachen.

2.2 Mathematische Beschreibung

Wir wollen das Straßennetz einer Stadt bzw. eines Stadtteiles als ein ungerichteten Graph modellieren. Hierzu definieren wir einen ungerichteten Graph:

Definition (ungerichteter Graph). [1, S.56] *Ein* ungerichteter Graph *ist ein Tripel* $G = (V,E,\gamma)$ *mit folgenden Eigenschaften:*

(i) *die Knotenmenge* V *ist nicht leer*,

(ii) E *ist die Menge der Kanten*,

(iii) *es gilt* $V \cap E = \emptyset$,

(iv) *die Abbildung*

$$\gamma : E \to \{X : X \subseteq V \quad mit \quad 1 \leq |X| \leq 2\}$$

ordnet jeder Kante e ihre Endknoten $\gamma \subseteq V$ *zu.*

Jede Kante in einem ungerichtetem Graphen kann also immer nur in beiden Richtungen verlaufen. Durch solche Kanten können wir also jede Straße in unserem Modell darstellen, da wir keine Einbahnstraßen, die nur eine Richtung befahren werden dürfen, erlauben. Wir nehmen zusätzlich noch an, dass unser ungerichteter Graph, *zusammenhängend* ist, d.h. jeder beliebige Knoten muss durch jeden anderen Knoten erreichbar sein, da wir nur ein Müllfahrzeug haben, der alle Straßen befahren muss.

Quelle: [1]

Abbildung 1: Ungerichteter gewichteter Graph

Jeder Knoten wird als eine Kreuzung oder das Ende einer Sackgasse modelliert. Dadurch können wir jedes Straßennetz in unserem vereinfachten Modell als ein zusammenhängenden ungerichteten Graphen darstellen. Im Optimalfall können wir einen *Weg* finden, der alle Kanten genau einmal durchläuft.

Definition (Weg). [1, S.59] *Ein* Weg *in einem ungerichtetem Graphen* G = (V,E,γ) *ist eine endliche Folge* $P = (v_0, e_1, v_1, ..., e_k, v_k)$ *mit* $k \leq 0$, *wobei* $v_0, ..., v_k \in V$ *und* $e_1, ..., e_k \in$

[1] https://www.cyface.de/wp-content/uploads/2017/02/map.jpeg Letzter Zugriff: 22.12.18

E mit $v_{i-1}, v_i \in \gamma(e_i)$ *für* $i = 1, ..., k$.

Wir können nun unser Modell zusätzlich spezifizieren durch Zuweisungen von Kantengewichten, sodass wir verschiedene Arten von Variablen darstellen können, wie z.B. die Weglänge, Durchfahrtszeit oder Bearbeitungszeit der Straße. Der Grund für die Spezifikation ist, dass wir nicht immer eine Route finden können, die jede Straße nur einmal befährt und wir so Straßen doppelt befahren müssen. Dadurch können sich Routen unterscheiden, falls wir entweder die Weglänge oder die Fahrtzeit minimieren wollen. Aber da wir bei jeder Straße genau einmal den Müll entsorgen müssen, können wir die Bearbeitungszeit ignorieren. So könnten die Kantengewichte in Abbildung 1 die Durchfahrtszeit abbilden.

Definition (Kantengewicht).[1, S.62] *Wir nennen eine Funktion c:* $E \to \mathbb{R}$ Kantengewichte *oder* Kostenfunktion *und schreiben*

$$c(G) = c(E(G)) = \sum_{e \in E(G)} c(e)$$

für das Gewicht des Graphen G, wobei für unsere Zwecke c: $E \to \mathbb{R}_{>0}$ *ausreicht.*

Unser Problem lautet also nun: Finde einen optimalen Weg, die die Summe der Gewichte von den Kanten minimiert, jede Kante mindestens einmal durchläuft und deren Startpunkt und Endpunkt identisch sind.
Einen *Weg*, der jede Kante in unserem zusammenhängendem Graphen genau einmal durchläuft, der jedoch noch nicht die Bedingung erfüllt, dass der Start- und Endpunkt gleich sind, nennt man auch *Eulerweg*. Da aber in unserem Modell der Anfangspunkt gleich dem Endpunkt sein soll, muss unser Eulerweg darüber hinaus noch ein *Kreis* sein. Ein Kreis ist ein *Weg*, dessen Anfangs- und Endpunkte übereinstimmen. Dieser spezielle Eulerweg nennt man auch eine *Eulertour*.

Definition (Euler-Graph). [2, S.58] *Ein Graph heißt* eulersch *oder ist ein* Euler-Graph, *wenn es in ihm eine* Euler-Tour *gibt. Ein* Euler-Kreis *oder eine* Euler-Tour *in einem Graphen ist ein Kreis, der jede Kante genau einmal enthält.*

Wenn wir also einen Eulertour finden können, der die Summe der Gewichte auf dem Weg minimiert, dann haben wir eine optimale Lösung zu unserem Problem gefunden!

Ich möchte hier anmerken, dass man zusätzlich zu unseren Modell Kanten, die nur

eine Richtung besitzen, aufnehmen kann. Durch solche Kanten kann man nun Eisenbahnstraßen oder auch Straßen, die zweimal befahren werden müssen modellieren. Ein Graph, die aus gerichteten und ungerichteten Kanten und Knoten besteht, heißt *ein gemischter Graph*. Damit könnte man jedes Straßennetzwerk präzise abbilden.

3 Lösungsansatz

3.1 Konstruktion von Eulergraphen und -touren

Die Fragen mit der wir uns in diesem Abschnitt befassen werden ist, ob es in einem beliebigen (ungerichteten) Graphen, der unser Straßennetz modelliert, eine Eulertour existiert und wenn ja, wie wir sie konstruieren. Bevor wir uns mit den Fragen beschäftigen, müssen wir noch den Begriff *Grad* einführen.

Definition (Grad).[2, S.5] *Sei G = (V,E,γ) ein Graph. Der* Grad *d(v) eines Knoten $v \in V$* ist die Anzahl des Auftretens von v als Endknoten einer Kante.

Der wichtigste Satz für die Antwort auf unsere Frage, ob es eine Eulertour existiert in einem Graphen, hatte schon Euler (1736) formuliert für sein berühmten *Königsberger Brückenproblem*. Mit diesem Satz können wir alle eulersche Graphen charakterisieren:

Satz.[2, S.59] *Ein zusammenhängender Graph ist genau dann eulersch, wenn jeder Knoten einen geraden Grad hat.*

Wir können zwei wichtige Erkenntnisse aus dem Satz ziehen: Wenn ein Graph eulersch ist, dann hat jeder Knoten ein geraden Grad. Die zweite Aussage ist für uns die wichtigere. Denn wir müssen nur überprüfen, ob der Grad jedes Knoten im Graph des Straßennetzes gerade ist und wissen, ob es eine Eulertour existiert. Bevor wir fortfahren, wollen wir diesen Satz noch beweisen. Dafür brauchen wir das Lemma:

Lemma.[2, S.59] *Gegeben sei ein Graph, in dem alle Knoten geraden Grad haben. Sei W ein Weg mit den Endknoten v_0 und v_k, der keine Kante doppelt durchläuft. Falls jede zu v_k inzidente Kante auch in W vorkommt, gilt $v_0 = v_k$. Der Weg W ist also ein Kreis.*

Den Beweis zu diesem Lemma findet man in der Quelle [2, S.59]. Wir können nun mit Hilfe des Lemmas den Satz beweisen:

Beweis:[2, S.59]
„⇒“: Wir beginnen mit der Hin-Richtung (Wenn der Graph G eulersch ist, dann hat

jeder Knoten geraden Grad): Sei T eine Eulertour in G, d.h. ein Kreis, der alle Kanten durchläuft. Sei v ein Knoten, der in der Tour k-mal vorkommt. Dann führt jedes Mal eine Kante in den Knoten hinein und eine Kante aus dem Knoten heraus. Alle Kanten, die zum Knoten v inzident sind, werden genau einmal besucht. Insgesamt sind es $2k$ Kanten, und damit ist der Grad am Knoten v gerade.
„⇐“: Wir wollen nun die Rückrichtung beweisen (Wenn jeder Knoten geraden Grad hat, dann existiert eine Euler-Tour): Betrachten wir dazu einen Weg
$W = v_0e_0v_1e_1...e_{k-1}v_k$ maximaler Länge in G, der keine Kante doppelt benutzt. Nach Lemma 4.2 ist W ein Kreis.
Fall 1: W benutzt alle Kanten. Dann ist W eine Eulertour und wir sind fertig.
Fall 2: Angenommen, es existiert eine Kante $e = (u, v) \in E$, die nicht in W enthalten ist. Die Idee besteht nun darin zu zeigen, dass wir einen Knoten in W finden können, der zu einer noch nicht besuchten Kante $e' = (v', v_i)$ inzident ist. Mit dieser Kante können wir dann den Weg W folgendermaßen verlängern:

$$W' := v'e'v_ie_iv_{i+1}...e_{k-1}v_ke_0v_1e_1...e_{i-1}v_i$$

Der Weg W' ist länger als W. Das ist aber ein Widerspruch zur Wahl von W, da W der längste Weg in G sein sollte. Wir müssen also noch zeigen, dass die Kante e' existiert. Nach unserer Fallunterscheidung existiert eine Kante $e = (u, v)$, die nicht auf dem Weg liegt. Falls e zu einem Knoten in W inzident ist, stellt e sofort die gesuchte Kante dar. Ist dies nicht der Fall, so existiert ein Weg $\overline{W}$ von v nach W, d.h. ein Weg in G, der nur Kanten enthält (folgt, weil der Graph zusammenhängend ist), die nicht in W liegen. Sei v_i der Endknoten von W'. Dann ist die gesuchte Kante e' die letzte Kante (v, v_i) auf dem Weg $\overline{W}$ mit $v' \in \overline{W}$ und $v_i \in W$.

Nachdem wir mit dem Satz überprüft haben, ob der vorliegende Graph eulersch ist, können wir uns jetzt überlegen, wie wir eine Eulertour darin finden. Später werden wir uns auch Graphen betrachten, die Knoten besitzen, die keinen geraden Grad haben.

3.2 Algorithmen zur Bestimmung von Eulertouren in Eulergraphen

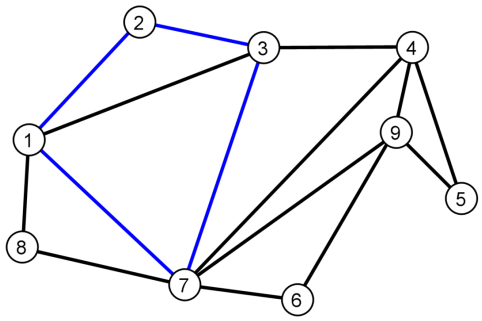
In diesem Abschnitt werden wir zwei verschiedene Algorithmen kennen lernen um eine Euler Tour zu finden in einem eulerschen Graph, wobei wir nun wissen, dass der Graph zusammenhängend und nur Knoten mit geradem Knotengrad besitzt. Die Eigenschaft wird uns sehr zu Nutze kommen, um die Korrektheit der Algorithmen zu beweisen. Wir nehmen an wir haben einen eulerschen Graphen *Sei G = (V,E,γ).*

3.2.1 Algorithmus von Hierholzer

Der Algorithmus wurde vom Mathematiker Carl Hierholzer im Jahr 1871 veröffentlicht. Die Idee des Algorithmus ist es, den gesamten Graph nach und nach in verschiedene kreisförmige Graphen zu zerlegen, die dann anschließend zu einem gesamten Kreis zusammenzufassen. Formal kann man den Prozess in 3 Schritten unterteilen:

1. Schritt:

- Wähle einen beliebigen Startknote $v_0 \in V$
- Wähle unbesuchte Kanten bis der Ausgangsknoten wieder erreicht ist. Der konstruierte Weg ist ein Kreis K.



Quelle: [2]

Abbildung 2: Wähle als Startknoten 2 aus. Der blaue Weg ist der Kreis K.

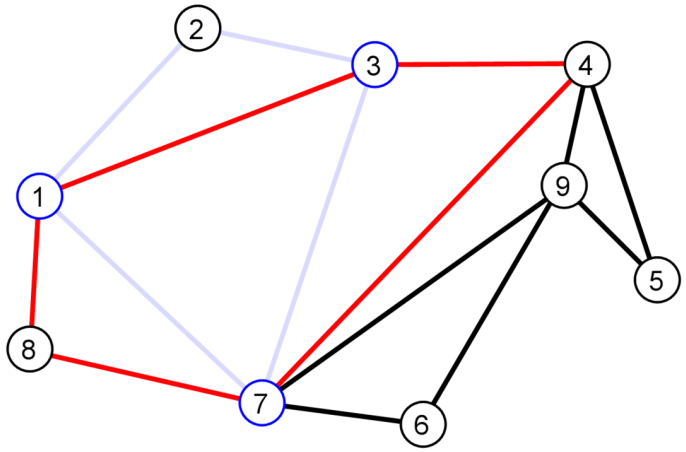
2. Schritt:

- Wenn K eine Eulertour ist, breche ab und K ist unsere Lösung. Ansonsten gehe zu Schritt 3.

3. Schritt:

- Setze $K^{'} = K$.
- Wähle Knoten w aus dem Kreis $K^{'}$, der eine unbesuchte Kante besitzt.
- Konstruiere wie im 1. Schritt für w einen Kreis $K^{''}$, wobei wir nur den Graphen ohne Kanten vom Kreis $K^{'}$ betrachten.

[2] https://de.wikipedia.org/wiki/Datei:Hierholzer(2).png Letzter Zugriff: 22.12.18



Quelle: [3]

Abbildung 3: Entferne K (in blau) und wähle Knoten 1,3 oder 7 für Schritt 1.

- Nun wollen wir den Kreis K neu konstruieren, indem er die Kreise $K^{'}$ und $K^{''}$ enthält: Gehe von $v_0 \in K^{'}$ entlang $K^{'}$ bis zum Knoten w. Durchlaufe dann den neuen Kreis $K^{''}$ einmal komplett durch und wieder bei w angekommen, durchläuft man den Rest von $K^{'}$. Bezeichne diesen Weg als K.
- Gehe zu Schritt 2.

[3] https://de.wikipedia.org/wiki/Datei:Hierholzer(3).png Letzter Zugriff: 22.12.18

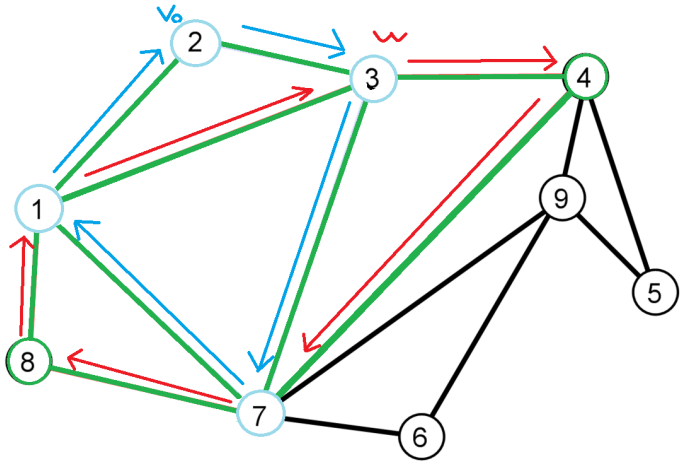


Abbildung 4: Der grüne Weg ist der neue konstruierte Kreis K.

Satz.[2, S.62] *Gegeben sei ein zusammenhängender Graph, in dem alle Kanten geraden Grad haben. Dann berechnet der Algorithmus von Hierholzer eine Euler-Tour.*

Beweis:[2, S.63] Wir müssen nach und nach alle Schritte durchgehen und uns vergewissern, dass jeder einzelne durchführbar ist.
Wir starten mit Schritt 1: Nach der Wahl eines beliebigen Knoten v_0 bestimmen wir einen Weg W, der keine Kanten doppelt enthält und nicht verlängerbar ist. Dann ist W nach Lemma 4.2 ein Kreis. Schritt 1 ist somit sinnvoll. In Schritt 2 gilt es zu überprüfen, ob der bisher gefundene Kreis K eine Euler-Tour ist. Dazu wird untersucht, ob jede Kante von G in K enthalten ist. Nur wenn dies der Fall ist, stoppt der Algorithmus, ansonsten nicht. Damit wird auf jeden Fall eine Eulertour berechnet.
Bei Schritt 3 müssen wir zunächst zeigen, dass ein Knoten $w \in K$ existiert, dessen Kante noch nicht besucht wurde. Damit Schritt 3 ausgeführt wird, muss es eine noch nicht besuchte Kante $e = (u, v) \in E(G) \backslash E(K)$ geben. Um den gewünschten Knoten w zu finden, unterscheiden wir zwei Fälle (ähnlich wie im Beweis zum Satz von Euler): *Fall 1* : u oder v sind in K enthalten. Dann entsprechen beide dem gesuchten Knoten.
Fall 2 : Weder u noch v sind in K enthalten. Da G zusammenhängend ist, existiert ein Weg W' von v nach W. Der letzte Knoten auf diesem Weg entspricht den Anforderungen.
Der letzte Teil in Schritt 3 besteht aus dem Zusammenkleben der Kreise K' und K''. (K'' ist ein Kreis, da $G \backslash K'$ nur Knoten mit geradem Grad hat und K'' wieder ein nicht verlängerbarer Weg ist.) Anhand der formalen Schreibweise beider Kreise sieht man sofort, dass K ein Kreis ist. Da der Kreis K'' mindestens eine neue Kante dem

Kreis K' hinzufügt, kann Schritt 3 nicht häufiger als $0.5 \cdot m$ durchgeführt werden. Der Algorithmus endet also immer, sodass seine Korrektheit bewiesen. ist.

Wir können also mit dem Algorithmus von Hierholzer eine Eulertour finden in einem eulerschen Graphen.

3.2.2 Algorithmus von Fleury

Ein weiterer Algorithmus zur Bestimmung der Eulertour lautet *Algorithmus von Fleury*. Im Gegensatz zum Hierholzer haben wir bei diesem Algorithmus bei der Konstruktion der Tour keine Unterbrechung. Der Algorithmus *zeichnet ohne den Stift abzusetzen*, jedoch ist er auch langsamer als der Algorithmus von Hierholzer [wikipedia]. Bevor wir die Struktur des Algorithmus verstehen können, müssen wir den Begriff *Brücke* einführen, da dieser eine wichtige Rolle im Algorithmus spielt.

Definition (Brücke). Eine Brücke ist eine Kante $e \in E$ bei deren Löschung der Graph in zwei Komponenten zerfallen würde.

Eulersche Graphen haben spezielle Eigenschaften in Bezug auf Brücken:

Lemma.[2, S.64] *Wenn in einem Graphen alle Knotengrade gerade sind, dann hat er keine Brücke.*

Auch zu diesem Lemma können sich interessierende Leser den Beweis in der Quelle [2, S.64] nachschlagen. Wir wissen also, dass wenn wir eine eulerschen Graphen haben, dann besitzt dieser keine Brücke. Auf der Grundlage dieses Lemmas ist der Algorithmus von Fleury konstruiert. Der Algorithmus verlängert nach und nach einen zum Anfang leeren Weg bis er eine Eulertour ist. Wir können den Algorithmus in 3 Schritte unterteilen:

1. Schritt: Beginne mit einer beliebigen Startknoten und wähle eine inzidente Kante.

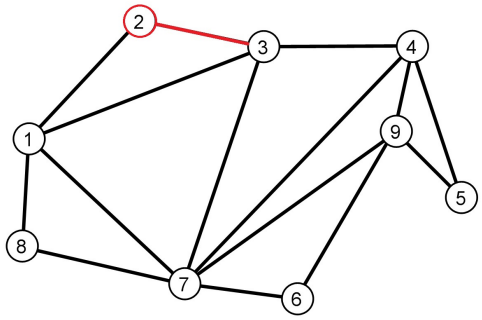


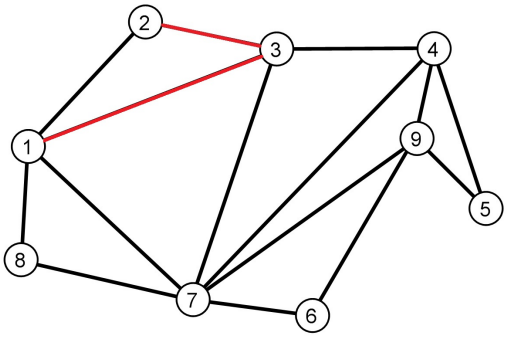
Abbildung 5: Wählen Startknoten 2 und die inzidente Kante zwischen Knoten 2 und 3.

2. Schritt: Wähle nächste unbesuchte Kante aus. Dabei sind Kanten zu wählen, die die 2 Bedingungen erfüllen, wobei die 2. Bedingung verletzt werden darf, wenn es keine andere Kanten gibt, die diese erfüllt:

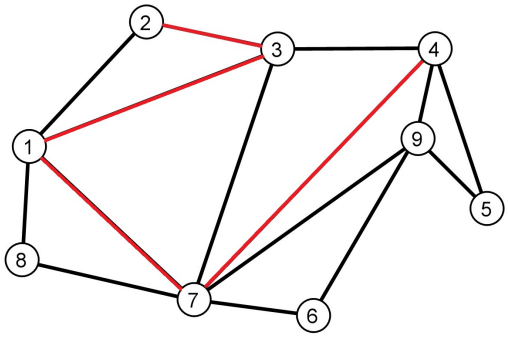
1. Inzident zu der zuletzt besuchten Kanten.

2. Kante (u, v) ist keine Brücke in dem Restgraphen, der aus allen noch nicht besuchten Kanten besteht.
Wir überprüfen diese Bedingung, indem wir nach einem (u, v)-Weg suchen im Restgraphen ohne die Kante (u, v).
Falls wir solch ein Weg finden können, z.B. mit der Tiefensuche, dann wissen wir, dass die Kante eine Brücke sein muss, denn wir können den gefundenen Weg (u, v) erweitern zu einem Kreis mit der Kante u, v.
Aber eine Kante ist keine Brücke, wenn sie auf einem Kreis liegt!

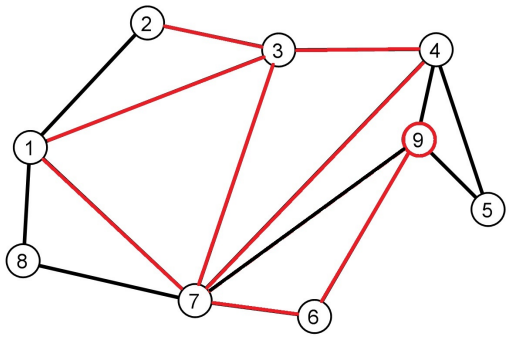
3. Schritt: Wiederhole Schritt 2 bis alle Kanten besucht worden sind.



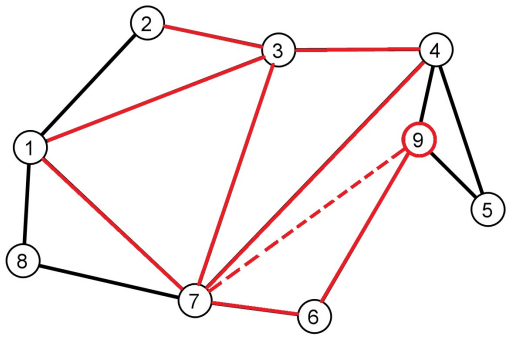
(a) Graph nachdem zwei Kanten gewählt wurden



(b) Graph nachdem vier Kanten gewählt wurden



(c) Graph nachdem acht Kanten gewählt wurden. Wir befinden uns im Knoten 9.



(d) Wahl der gestrichelten Linie verboten, da sonst der Restgraph in zwei unzusammenhängende Graphen zerfällt

Wir kennen nun zwei verschiedene Algorithmen um eine Eulertour in einem eulerschen Graphen zu finden. Das einzige Problem, dass uns noch bleibt ist: Unser Straßennetz muss kein eulerscher Graph sein! Klar, denn nicht jede Kreuzung hat eine gerade Anzahl an Straßen, d.h. wir haben Knoten, die einen ungeraden Grad haben.

3.3 Verbinden von ungeraden Knoten

Wir schauen uns in diesem Abschnitt den Fall an, wenn der Graph ungeraden Knoten hat. Wir merken zuerst an, dass es kein Graphen existiert, der eine ungerade Anzahl an Knoten mit ungeraden Grad hat. Daraus folgt der Satz:

Satz. [3, S.53] In jedem Graphen ist die Anzahl der Knoten mit ungeradem Grad gerade.

Beweis: Jede Kante hat einen Anfangs- und einen Endknoten, weshalb sich die Summe der Knotengrade um 2 erhöht, wenn wir eine Kante in unseren Graphen hinzufügen. Dadurch lässt sich leicht einsehen, dass die Summe der Knotengrade gleich der doppelten Anzahl der Kanten ist. Nehmen wir jetzt an, dass die Anzahl der Knoten mit ungeradem Grad ungerade ist, dann folgt, dass die Summe der Knotengrade ungerade ist. Widerspruch! Damit ist der Satz bewiesen. □

Die Idee ist es jetzt, die Knoten mit ungeraden Grad paarweise zu verbinden, wobei wir auf kürzestem Weg miteinander verbinden wollen um die Summe der Kantengewichte zu minimieren. Das funktioniert für jeden nicht eulerschen Graphen, da wir

wissen, dass es immer eine gerade Anzahl an Knoten mit ungeraden Grad hat. Falls die Verbindung durch andere Knoten mit geradem Grad führt, erhöht sich deren Grad immer um 2, da die Kante hineinführt, aber auch aus dem Knoten hinausführt. Wir haben durch das Verbinden von Knoten mit ungeraden Grad nun ein Graph erzeugt, der nur aus Knoten mit geraden Grad besteht.

Wir werden mithilfe vom Algorithmus von Dijkstra eine kürzeste Verbindung zwischen 2 ungeraden Knoten finden. Der Algorithmus verlangt jedoch, dass wir einen gerichteten Graphen vorliegen haben und das dieser keine Schlingel aufweist. Wir sehen aber, dass die Umwandlung in ein gerichteten Graphen keine Schwierigkeiten mitbringt und dass unser Straßennetzwerk keine Schlingel besitzen kann, denn solche Konstrukte nehmen keinen Einfluss auf unsere Routenplanung und wir können diese ignorieren.

Lösung: Finden einer optimaler Route bei der Müllabfuhr.

1) Wandel das Straßennetz in einen ungerichteten zusammenhängenden Graphen um, wobei Kreuzungen die Knoten und Straßen die Kanten darstellen sollen und Sackgassen sollen erstmal ignoriert werden.

2) Falls der Graph Knoten mit ungeraden Grad hat, dann betrachte diese Knoten und finde eine Verbindung mit der kleinsten Summe der Kantengewichte (mithilfe Algorithmus von Dijkstra), um unsere Route optimal zu halten.

3) Füge die Verbindung in Form von Kantenzügen in den Graphen hinzu.

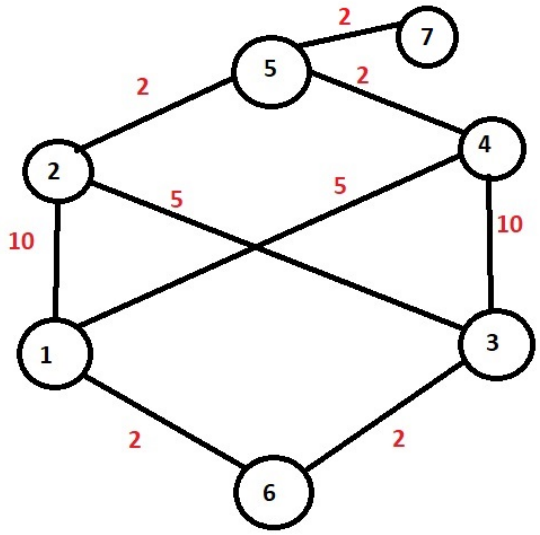
4) Bei Sackgassen müssen wir hinein- und hinausfahren. Also werden für Sackgassen doppelte Kanten eingezeichnet.

5) Wir haben nun einen Eulergraph und jede Eulertour ist optimal. Wir können unsere Algorithmen von Hierholzer und Fleury anwenden um eine Eulertour zu finden. Nun haben wir eine optimale Route gefunden und wir sind fertig!

4 Implementierung und Simulationsergebnisse

4.1 Testen der Algorithmen an einem Beispiel

Bevor wir uns ein Stadtteil von Mannheim anschauen werden wir unsere Algorithmen erstmal auf ein kleines Beispiel testen, wo die Lösung bekannt ist! Der folgende Graph hat offensichtlich eine Sackgasse zwischen Knoten 5 und 7. Außerdem besitzen Knoten 1,2,3 und 4 ungeraden Grad.



1. Schritt: Wir werden als erstes die Sackgassen des Graphen bestimmen durch die Funktion „sackgasse“. Nun können wir die entsprechenden Kanten in der Inzidenzmatrix löschen.

```
I =

     1     0     0     1     0     0     0     1     0
     1     0     1     0     0     1     0     0     0
     0     1     1     0     0     0     1     0     0
     0     1     0     1     1     0     0     0     0
     0     0     0     0     1     1     0     0     1
     0     0     0     0     0     0     1     1     0
     0     0     0     0     0     0     0     0     1
```

```
>> [gasse,sackG] = sackgasse(I);
>> gasse{1,1}

ans =

     5     7
```

Abbildung 6: Inzidenzmatrix + Sackgasse in Knoten

2. Schritt: Unser vorliegender Graph hat offensichtlich Knoten mit ungeradem Grad, weshalb wir den Graph umwandeln müssen in ein eulerschen Graphen, d.h ein Graph mit nur geradem Knotengrad. Wir müssen also als erstes alle Knoten mit ungeraden Grad suchen. Dies können wir mit dem Algorithmus „knotengrad“ erledigen.

```
>> I(:,sackG) = []

I =

     1     0     0     1     0     0     0     1
     1     0     1     0     0     1     0     0
     0     1     1     0     0     0     1     0
     0     1     0     1     1     0     0     0
     0     0     0     0     1     1     0     0
     0     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0
```

```
>> ungKnoten = knotengrad(I)

ungKnoten =

     1     2     3     4
```

Abbildung 7: Inzidenzmatrix ohne Sackgassen + Knoten mit ungeraden Grad

Nun wollen wir ein minimales Matching finden für die Knoten mit ungeraden Grad. Dafür habe ich die Funktion „minMatchin“ geschrieben, die auf Basis „maxWeight-Matching“ von Daniel Saunder[4] basiert. Wir wollen seine Funktion so modifiziert, sodass wir ein minimales Matching gegeben für bestimmte Punkte in einem Graphen ausrechnen können. Das Problem der Funktion „maxWeightMatching“ ist nämlich, dass diese für alle Knoten im Graphen ein maximales Matching ausrechnet. In unserem Fall wollen wir aber ein minimales Matching gegeben für **bestimmte Knoten**, und zwar die ungeraden Knoten, in einem Graphen ausrechnen.

Idee: Ein maximales Matching in ein minimales Matching umzuwandeln ist einfach, wir drehen alle Vorzeichen unserer Kantengewichte um. Wenn wir das maximale Matching auf die Kanten mit den neuen Gewichten anwenden, erhalten wir offensichtlich ein minimales Matching. Bleibt das Problem, dass wir nur bestimmte Punkte betrachten wollen.

Wir machen als erstes die folgende Überlegung: Wir berechnen ALLE Wege die es gibt zwischen allen ungerade Knoten. Wir führen (gedanklich) das gesuchte minimale Matching aus für nur ungerade Knoten. Wenn wir uns nun 2 vorher ungerade Knoten a und b, die verbunden wurden, betrachten, dann muss das minimale Matching einen der Wege ausgewählt haben, die wir vorher ausgerechnet haben, da wir schließlich alle Wege zwischen a und b berechnet haben, die es gibt. Das entscheidende ist, dass kein Weg im Verlaufe des Matching entfällt, d.h. wenn wir zwei Knoten verbunden haben, können wir immer noch jeden existierenden Weg laufen um die anderen Knoten zu verbinden. Wir können also alle Wege zwischen allen ungerade Knoten als eine Kante modellieren mit entsprechenden Kantengewichten! Das Mat-

[4]https://de.mathworks.com/matlabcentral/fileexchange/42827-weighted-maximum-matching-in-general-graphs Letzer Zugriff: 23.12.18

ching wird natürlich dann kein Unterschied sehen, als wenn wir die Wege als einzelne Kanten betrachten, solange die Kantengewichte übereinstimmen. Also können wir getrost unser Matching über die neuen Kanten laufen lassen. Das Problem dieser Methode ist es, dass wir einen sehr hohen Rechenaufwand brauchen, da wir alle Wege zwischen allen ungeraden Knoten berechnen müssen. Deshalb habe ich mir einen anderen Lösungsweg gesucht:

Wir betrachten den Ansatz von oben und modifizieren ihn. Wir haben alle möglichen Wege berechnet und haben diese als Kanten modelliert, aber daraus folgt, dass alle Wege zwischen 2 Knoten als parallele Kanten modelliert werden. Das minimale Matching wird natürlich nur die minimalste Kante zwischen 2 Knoten auswählen! Auf unseren ursprünglichen Graphen bedeutet das nun, dass wir den kürzesten Weg zwischen a und b gewählt haben. Wir können mit Dijkstra alle kürzesten Wege zwischen allen ungeraden Knoten berechnen und diese Wege als Kanten im neuen Graphen modellieren und nicht mehr alle möglichen Wege. Darauf wenden wir nun dann das minimale Matching an und am Ende müssen wir natürlich die ausgewählten Kanten wieder zurück transformieren in einen Weg im alten Graphen.

```
>> kanten = minMatching(I,ungKnoten,c);
>> kanten{1,1}

ans =

     3     6     1
```

```
>> kanten{1,2}

ans =

     4     5     2
```

Abbildung 8: Kanten die hinzugefügt werden müssen: Matching macht Sinn, da diese 2 Verbindungen minimal sind.

3. Schritt: Wir fügen die gefundenen Kantenzüge in unser Straßennetz zu mit der Funktion „kantenhinzufugen“.

4. Schritt: Der letzte Schritt um einen eulerschen Graphen zu erhalten ist, die Sackgassen durch doppelte Kanten darzustellen. Hinzufügen dieser Kanten erfolgt wieder mit „kantenhinzufugen“, wobei wir zweimal hinzufügen müssen, da wir die Sackgassen im 1. Schritt gelöscht haben.

```
K>> I = kantenhinzufugen(I,kanten);
 I = kantenhinzufugen (I, gasse);
 I = kantenhinzufugen (I, gasse);
K>> I

I =

     1     0     0     1     0     0     0     1     0     1     0     0     0     0
     1     0     1     0     0     1     0     0     0     0     0     1     0     0
     0     1     1     0     0     0     1     0     1     0     0     0     0     0
     0     1     0     1     1     0     0     0     0     0     1     0     0     0
     0     0     0     0     1     1     0     0     0     0     1     1     1     1
     0     0     0     0     0     0     1     1     1     1     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0     1     1
```

Abbildung 9: Endgültige Inzidenzmatrix ist richtig, leicht selber überprüfbar

5. Schritt: Wir haben nun einen Euler Graphen und können unsere 2 Algorithmen laufen lassen.

```
K>> kanten = hierholzer(I,1);
weg = kanten2knoten(I,kanten,1)

weg =

     1     2     3     6     1     6     3     4     5     2     5     7     5     4     1


K>> s = 1;
kanten = fleury(I,1);
weg = kanten2knoten(I,kanten,1)

weg =

     1     4     5     7     5     2     5     4     3     6     1     6     3     2     1
```

Abbildung 10: Hierholzer oben, Fleury unten

Wir können mit einer Zeichnung des Weges in den Graphen leicht einsehen, dass die beiden Wege der optimale Weg sein muss.

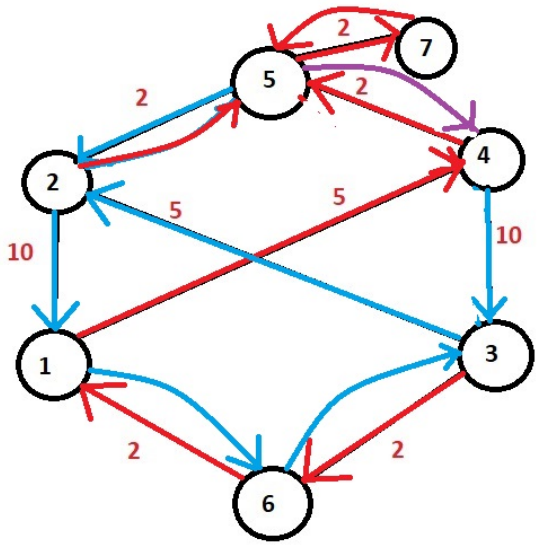


Abbildung 11: Weg von Fleury: 1. Mal bei Knoten: Rote Kante. 2. Mal: Blaue. 3. Mal : Violette

Unsere Algorithmen funktionieren also schon mal für einen überschaubaren Graphen!

4.2 Optimale Route in Gartenstadt

In diesem Abschnitt werden wir für ein Straßennetz von Mannheim eine optimale Routenplanung für die Müllabfuhr konstruieren. Hierfür nehmen wir den Stadtteil „Gartenstadt“ in Mannheim, da dieser nur eine Straße enthält, die unser Modell nicht berücksichtigt, nämlich eine Straße, die man zweimal befahren muss. Diese können wir ignorieren und das Straßennetz ist somit geeignet für unser vereinfachtes Modell.

Als erstes Grenzen wir den Stadtteil ein und wählen einen Startknoten, für den wir eine optimale Route finden wollen durch das Anwenden unseres erarbeiteten Schemas „Lösung: Finden einer optimaler Route bei der Müllabfuhr“.

[5]https://www.openstreetmap.de/ Letzter Zugriff: 23.12.18

Quelle: [5]

Abbildung 12: Straßennetz von Gartenstadt

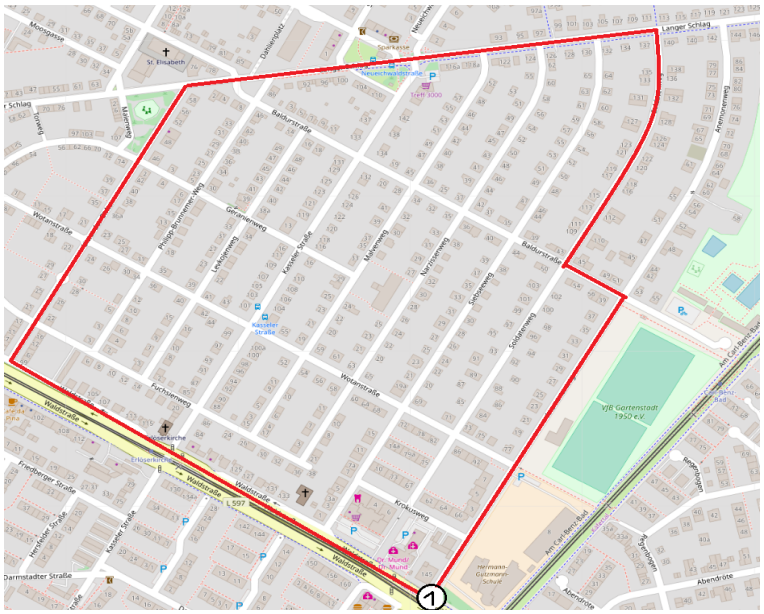
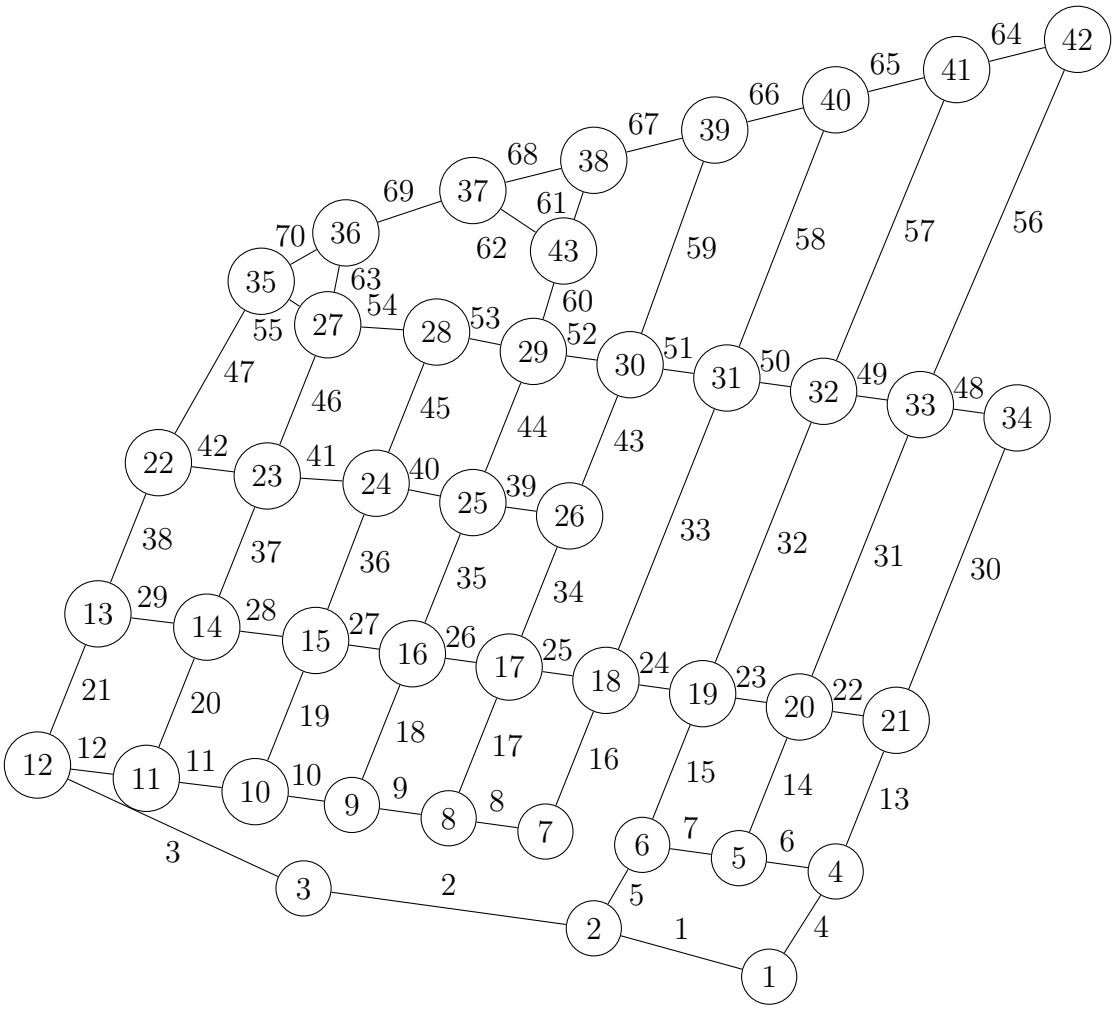
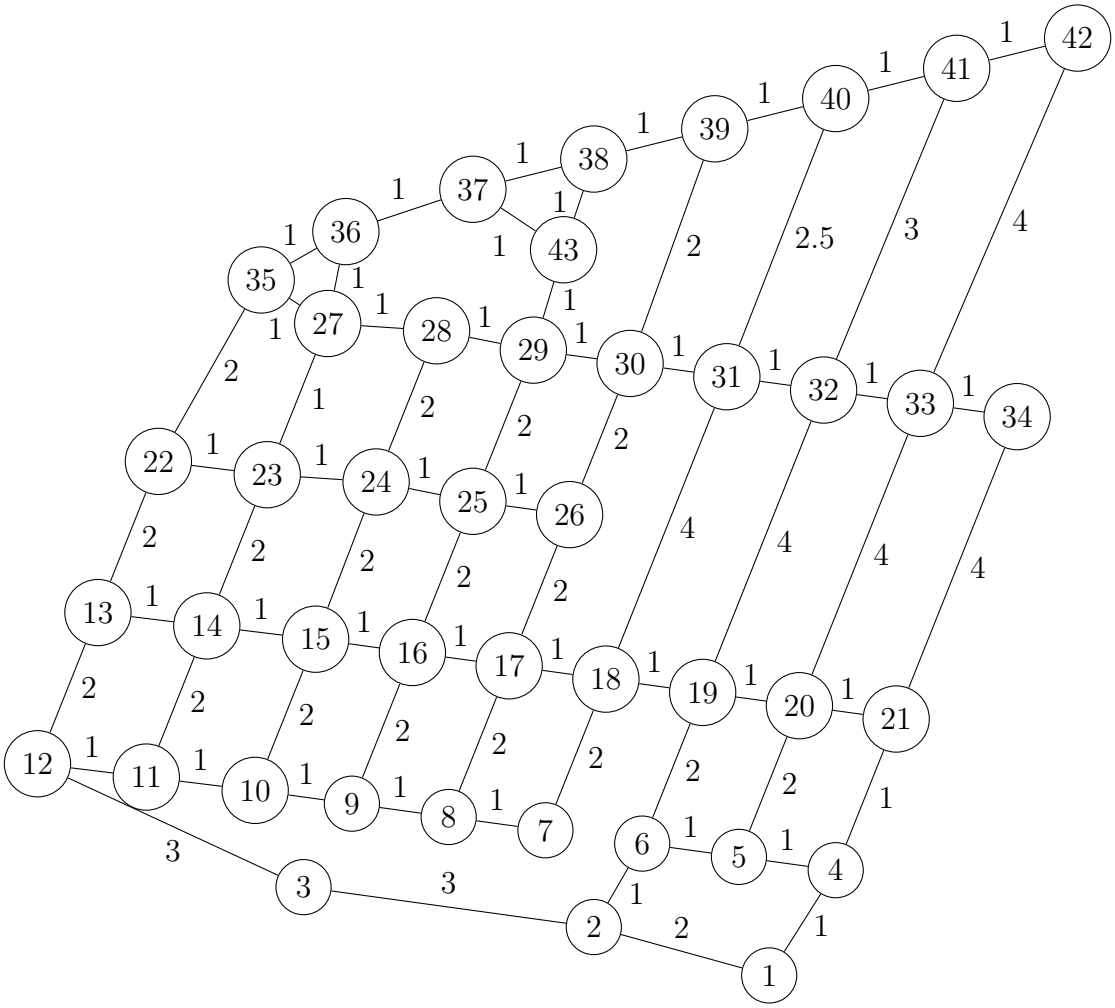


Abbildung 13: Rote Linie stellt Rand dar und der Knoten eins, den Start- bzw. Endknoten.

1. Schritt: Wandel das Straßennetz in ein Graphen um. Die Zahlen an der Kante sind nicht das Kantengewicht sondern lediglich nur eine Nummerierung.



Als erstes bilden wir die Inzidenzmatrix mit dem Algorithmus „inzidenzmatrix“, der aus einem String, der in Latex einen Graphen zeichnet, eine Inzidenzmatrix erstellt. Als nächstes ordnen wir jeder Kante Kantengewichte zu. Wir wollen dabei die Zeit als Kantengewicht nehmen, wobei die angegebenen Zeiten aufgerundet und gemessen wurden durch Google Maps.



Nun können wir analog wie im 1. Beispiel die einzelnen Schritte durchgehen. Unser Endergebnis sieht dann so aus:

```
wegfleury =

  Columns 1 through 19

     1     4     5    20    21    34    33    42    41    40    39    38    37    43    37    36    27    28    27

  Columns 20 through 38

    36    35    22    35    27    23    22    13    12    13    14    23    24    28    29    43    38    39    30

  Columns 39 through 57

    29    25    24    15    14    11    10    15    16    25    26    17    18    19    20    33    32    41    40

  Columns 58 through 76

    31    30    26    17    16     9     8    17    18    31    32    19    18     7     8     9    10    11    12

  Columns 77 through 87

     3     2     6    19    20    21     4     5     6     2     1


weghierholzer =

  Columns 1 through 19

     1     2     3    12    13    14    23    24    28    29    43    38    39    38    37    43    37    36    27

  Columns 20 through 38

    23    22    35    27    28    27    36    35    22    13    12    11    10    15    16    25    26    17    26

  Columns 39 through 57

    30    31    40    41    32    33    42    41    40    39    30    29    25    24    15    14    11    10     9

  Columns 58 through 76

     8    17    18    31    32    19    20    33    34    21    20    19    18    17    16     9     8     7    18

  Columns 77 through 87

    19     6     2     6     5     4    21    20     5     4     1
```

Abbildung 14: Hierholzer unten, Fleury oben

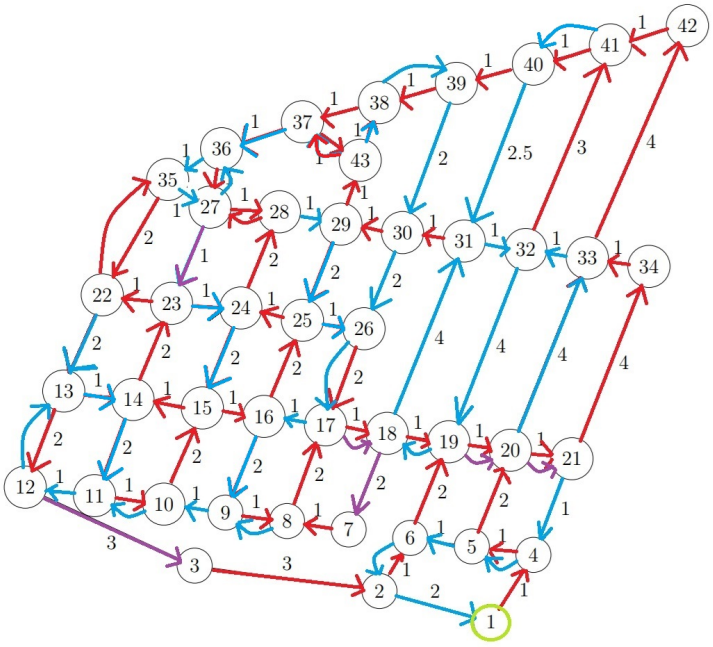


Abbildung 15: Optimale Route in Gartenstadt

5 Ausblick

Kein Modell, dass die Realität widerspiegeln soll, ist perfekt. Wir können noch einiges tun um unser Modell realitätsnäher zu machen. Schon ganz am Anfang hat unser Modell Einschränkungen vorgesehen: Einbahnstraßen und Straßen, die man doppelt befahren muss. Eine Erweiterung des Modells mit diesen beiden Einschränkungen könnte man vornehmen.
Wir haben zusätzlich nicht beachtet, dass Müllfahrzeuge natürlich nicht unendlich viel Müll aufsammeln können, d.h sie müssen ab einer bestimmten Kapazität wieder zurück zur Müllhalde fahren um abzuladen. Das wird unser Problem natürlich komplexer machen.
Als letztes könnte man auch ein Programm schreiben, dass jedes Straßennetz direkt als ein ungerichteten Graphen modelliert und in eine Inzidenzmatrix umwandelt. Hier haben wir „von Hand“die Modellierung vorgenommen.

Literatur

[1] Prof Dr. Simone Göttlich, Prof. Dr. Oliver Kolb: *Optimierung Skriptum.* Universität Mannheim, Lehrstuhl für wissenschaftliches Rechnen.

[2] Christina Büsing: *Graphen- und Netzwerkoptimierung.* Spektrum Akademischer Verlag Heidelberg, 2010.

[3] Claus Peter Ortlieb,Caroline v. Dresky, Ingenuin Gasser, Silke Günzel: *Mathematische Modellierung: Eine Einführung in zwölf Fallstudien.* Springer Fachmedien Wiesbaden 2009, 2013.

[4] Claus Peter Ortlieb,Caroline v. Dresky, Ingenuin Gasser, Silke Günzel: *Mathematische Modellierung: Eine Einführung in zwölf Fallstudien.* Springer Fachmedien Wiesbaden 2009, 2013.

[5] Wladimir Velminski: *Leonhard Euler. Die Geburt der Graphentheorie.* Kulturverlag Kadmos, Berlin 2008.