Aufgabe: Müllabfuhr

1.1 Lösungsidee

Wir betrachten das Straßennetz als Graphen. Sei $G = (V, E)$ der Graph, der das Straßennetz darstellt. Dabei ist V die Menge der Knoten (Kreuzungen) und E die Menge der Kanten (Straßen).

Das gestellte Problem ist sehr schwer. Genauer gesagt ist es NP-schwer, selbst wenn man nur zwei statt fünf Tage betrachtet.[1] Gleichzeitig enthalten die vorgegebenen Beispieleingaben bis zu 1000 Knoten. Wir können also nicht erwarten, eine optimale Lösung zu finden. Stattdessen setzen wir auf Heuristiken.

Dazu werden wir in zwei Schritten vorgehen:

1. Finde einen Weg, der am Startknoten anfängt und aufhört und der alle Kanten mindestens einmal besucht.
2. Teile den gefundenen Weg in fünf möglichst gleich große Teile auf, wobei Anfang und Ende jedes Abschnitts mit dem Start verbunden werden müssen.

Diese beiden Teile können wir optimal lösen. Es ist jedoch wichtig zu erkennen, dass die Gesamtlösung trotzdem nicht immer optimal ist: Eventuell gäbe es einen etwas längeren (oder einen ebenso langen) Gesamtweg, der sich aber besser in fünf Teile teilen lässt und dadurch eine bessere Lösung erzielen würde.

Finden eines Weges, der alle Kanten abdeckt

Die Aufgabe, einen möglichst kurzen Weg zu finden, der alle Kanten abdeckt, ist ähnlich zu dem Eulerkreisproblem. Dabei wird ein Weg gesucht, der jede Kante *genau* einmal besucht.

Sei der *Grad eines Knotens v* die Anzahl Kanten, die an v angrenzen. Ein Eulerkreis existiert genau dann, wenn jeder Knoten im Graphen einen geraden Grad hat. Das lässt sich dadurch begründen, dass jedes Mal, wenn der Weg einen Knoten besucht, eine angrenzende Kante *verbraucht* wird, und man eine weitere Kante braucht, um den Knoten wieder zu verlassen. Wenn ein Knoten einen ungeraden Grad hat, würde man unweigerlich an diesem Knoten *stecken bleiben*. Ein Eulerkreis lässt sich mit dem Algorithmus von Hierholzer effizient finden, wenn er existiert. Dabei geht man zuerst einen beliebigen Pfad durch den Graphen, bis man wieder beim Ausgangsknoten ankommt. Es ist garantiert, dass man wieder beim Ausgangsknoten ankommt, da alle Knoten einen geraden Grad haben. Dann betrachtet man einen Knoten, bei dem noch angrenzende Kanten unbesucht sind und sucht einen weiteren Pfad von dort aus, und fügt diesen in den bisherigen Kreis ein. Dies wird so lange wiederholt, bis alle Kanten besucht wurden.

Falls in dem gegebenen Graphen also nun alle Knoten einen geraden Grad haben, können wir durch Bestimmung des Eulerkreises verhältnismäßig einfach den kürzesten Weg finden, der alle Kanten abdeckt. Was machen wir nun, wenn ein solcher Eulerkreis nicht existiert?

[1] Wenn man das gestellte Problem effizient lösen könnte, dann könnte man auch eine Variante von Subset Sum effizient lösen, indem man für jedes Element der gegebenen Zahlenmenge eine Kante vom Startknoten zum Startknoten mit dieser Länge hinzufügt. Eine effiziente Lösung des Müllabfuhrproblems fände nun eine Aufteilung der Kanten in zwei Teilmengen mit gleich großer Summe, wenn vorhanden. Es ist aber bekannt, dass dies NP-schwer ist, daher muss auch das Müllabfuhrproblem NP-schwer sein.

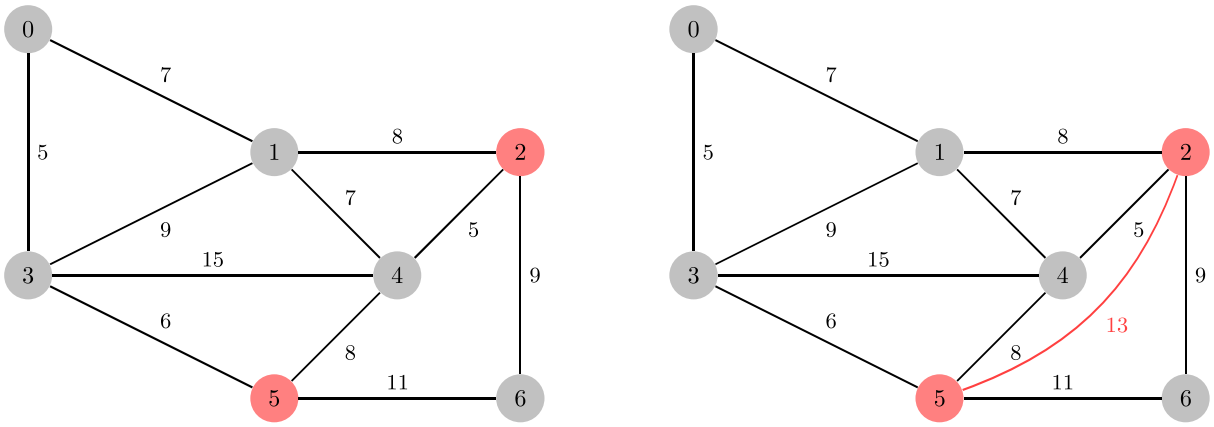

Abbildung 1.1: Beispiel für einen Straßenplan, wo jede Kreuzung durch einen Knoten und jede Straße als eine Kante repräsentiert ist. Die Länge der Straße ist das Gewicht der Kante. Knoten mit geradem Grad sind grau und Knoten mit ungeradem Grad rot markiert. Rechts sind die beiden roten Knoten mit einer Metakante verbunden, welche als Gewicht die kürzeste Verbindung zwischen den beiden Knoten hat. In dem rechten Straßenplan gibt es damit einen Eulerzyklus.

In dem Fall werden wir einige Kanten mehrfach besuchen müssen. Unsere Aufgabe ist es, die Gesamtlänge der mehrfach besuchten Kanten zu minimieren.

Betrachten wir einmal alle Knoten in dem gegebenen Graphen mit ungeradem Grad. Wenn wir jeweils zwei dieser Knoten mit einer neuen Kante verbinden, dann hätte der resultierende Graph einen Eulerzyklus. Praktischerweise wird es auch immer eine gerade Anzahl solcher Knoten geben, da jede Kante zwei Enden hat und daher die Gesamtzahl der Start- und Endpunkte der Kanten gerade ist. Wir fügen nun also *Metakanten* in den Graphen ein. Diese haben jeweils die Länge des kürzesten Weges zwischen zwei Knoten. Wir verbinden je zwei Knoten mit ungeradem Grad mit einer Metakante, sodass nachher jeder Knoten einen geraden Grad hat. Auf dem entstehenden Graphen können wir nun einen Eulerzyklus finden. Dieser deckt alle ursprünglichen Kanten ab. Die Länge entspricht der Länge aller ursprünglichen Kanten plus der Länge der eingefügten Kanten. Wir müssen also nur die Gesamtlänge der eingefügten Kanten minimieren.[2] Das Vorgehen ist exemplarisch in Abbildung 1.1 dargestellt.

Wir müssen nun also alle Knoten mit ungeradem Grad so in Paare aufteilen, dass die Summe der kürzesten Wege zwischen den Partnern minimal ist. Die kürzesten Wege zwischen allen relevanten Knoten können wir zum Beispiel mit dem Dijkstra-Algorithmus oder auch mit dem Algorithmus von Floyd und Warshall finden. Die Aufteilung in Paare ist bekannt als *Matching*. Konkret handelt es sich um *Min Cost Perfect Matching*. Im Gegensatz zu Matching auf bipartiten Graphen ist Matching auf allgemeinen Graphen leider umständlich. Trotzdem gibt es mit dem Blossom-Algorithmus auch für dieses Problem eine bekannte Lösung. Falls man diesen jedoch nicht implementieren möchte, kann man auch schon gute Ergebnisse erreichen, indem man die Knotenpaare mit einem gierigen Algorithmus zuweist – also zum Beispiel über alle Knoten iteriert, und jedem nicht zugewiesenen Knoten den ebenfalls nicht zugewiesenen Knoten mit der kleinsten Distanz zuweist.

[2] Es ist übrigens auch garantiert, dass wir einen Weg minimaler Länge mit dieser Methode finden können, also in mindestens einem optimalen Weg alle mehrfach besuchten Kanten zu solchen *Metakanten* zusammengefügt werden können. Den Beweis bleiben wir hier jedoch schuldig.

Aufteilung des Weges in Tagesabschnitte

Nun haben wir also einen Weg, der alle Kanten abdeckt und so kurz wie möglich ist. Die nächste Aufgabe ist, diesen in fünf Teile aufzuteilen, sodass die Länge des längsten Teils minimal ist. Komplikationen dabei sind:

- Der Weg kann jeweils nur an den Knoten, nicht in der Mitte einer Kante geteilt werden.
- Die Enden jedes Abschnittes müssen mit dem Startknoten verbunden werden.

Betrachten wir zunächst einmal eine etwas einfachere Aufgabenstellung: Gegeben eine Länge x, können wir den Zyklus in 5 Tagesabschnitte der Länge maximal x aufteilen? Um das zu beantworten, starten wir beim Startknoten und wählen von dort den längsten Tagesabschnitt, dessen Länge höchstens x ist. Für den nächsten Tag starten wir bei der ersten Kante, die noch nicht verwendet wurde, und wählen von dort wieder den längsten Abschnitt mit Länge höchstens x, und führen das für die übrigen Tage fort. Wenn wir nach fünf Tagen alle Kanten abgedeckt haben, ist die Länge x ausreichend.

Wir können nun eine binäre Suche ausführen, um das kleinste x zu finden, für das wir den Zyklus noch in entsprechend lange Tagesabschnitte teilen können. Wir starten mit einem Minimalwert, für den bekannt ist, dass er als Länge der Abschnitte nicht ausreicht (zum Beispiel 0) und einem Maximalwert, der auf jeden Fall ausreichen würde (zum Beispiel die Gesamtlänge des Zyklus). Dann prüfen wir für den Mittelwert zwischen Minimalwert und Maximalwert, ob er ausreichend ist. Wenn ja, ist der gesuchte Wert höchstens so groß wie der Mittelwert, ansonsten ist er größer. Dementsprechend werden die Maximal- und Minimalwerte angepasst und das Verfahren wiederholt. So finden wir sehr schnell das gesuchte x. Dieses wird die Länge des längsten Tagesabschnitts in unserer Lösung sein.

Laufzeit

Um die Laufzeit des Algorithmus betrachten zu können, definieren wir erst einmal $N = |V|$ und $M = |E|$, i. e. N ist die Anzahl Knoten in dem vorgegeben Graphen und M ist die Anzahl Kanten. Sei außerdem L die Gesamtlänge aller Kanten in der Eingabe.

Zuerst findet der Algorithmus Paare von Knoten mit ungeradem Grad. Dazu müssen wir zuerst die kürzesten Wege zwischen allen Paaren solcher Knoten finden. Der Algorithmus von Floyd und Warshall braucht hierfür eine Laufzeit in $\mathcal{O}(N^3)$. Mit Anwendung des Dijkstra-Algorithmus für jeden Knoten erhalten wir stattdessen eine Laufzeit von $\mathcal{O}(N(N+M)\log N)$, was besser ist, sofern M deutlich kleiner als N^2 ist.

Das Zuteilen der Paare selbst kann dann durchaus in linearer Laufzeit ($\mathcal{O}(N+M)$) geschehen, falls hier ein Greedy-Algorithmus gewählt wird. Falls man jedoch den Blossom-Algorithmus von Edmonds verwendet, um optimale Ergebnisse zu erhalten, wird es länger dauern. Der Blossom-Algorithmus kann mit einer Laufzeit von $\mathcal{O}(n^2m)$ für n Knoten und m Kanten umgesetzt werden. Relevant sind hier jedoch nicht die Kanten in dem Ausgangsgraphen, sondern die Knoten mit ungeradem Grad (bis zu N) und alle $\mathcal{O}(N^2)$ kürzesten Wege zwischen ihnen als Kanten. Die Laufzeit ist also bis zu $\mathcal{O}(N^4)$. Jedoch haben häufig deutlich weniger Knoten einen ungeraden Grad. In den Beispielen vom BWINF sind es nur maximal 240. Außerdem wird die angegebene Worst-Case-Laufzeit des Blossom-Algorithmus nur selten erreicht. Daher kann dieser Schritt auf allen vorgegebenen Beispielen in unter einer Sekunde durchlaufen, wenn

eine effiziente Implementierung dieses Algorithmus verwendet wird.[3].

Danach muss auf dem Graphen ein Eulerkreis gefunden werden. Der Algorithmus von Hierholzer benötigt hierfür nur lineare Laufzeit, also $\mathcal{O}(N+M)$.

Zum Aufteilen des Pfades werden logarithmisch viele Iterationen der binären Suche benötigt, das sich die Größe des betrachteten Intervalls in jedem Schritt halbiert. Jede Iteration betrachtet dabei den gesamten Zyklus der Länge $\mathcal{O}(M)$. Dieser Schritt hat also eine Laufzeit von $\mathcal{O}(M \log L)$.

Die Gesamtlaufzeit ist also (bei Verwendung des Floyd-Warshall- und des Blossom-Algorithmus) $\mathcal{O}(N^3+N^4+(N+M)+M\log L)$. Dies lässt sich grob zu $\mathcal{O}(N^4)$ vereinfachen, da $M < N^2$ und auch L zumindest in den vorgegebenen Eingaben nicht übermäßig groß ist. Auch in der Praxis stellt sich heraus, dass der Blossom-Algorithmus die Laufzeit dominiert.

Weitere Verbesserungen

Da es sich bei dem vorgestellten Algorithmus nur um eine Heuristik handelt, sind die Ergebnisse nicht optimal. Bei einer guten Implementierung liegt die Laufzeit außerdem erst bei ca. einer Sekunde auf den größten vorgegebenen Beispielen. Daher kann man sich Gedanken machen, wie die Ergebnisse noch weiter verbessert werden können. Ein paar Ideen sind hier grob skizziert:

- Ein Graph, der einen Eulerzyklus besitzt, hat typischerweise nicht nur einen, sondern viele verschiedene Eulerzyklen. Die Auswahl des konkreten Eulerzyklus kann außerdem einen Einfluss auf die Qualität des Ergebnisses haben. Es ist jedoch nicht direkt klar, welcher Eulerzyklus der beste ist. Bisher finden wir einen beliebigen. Da jedoch die Laufzeit für das Finden des Eulerzyklus und die darauf folgende Aufteilung des Zyklus bisher nur einen kleinen Teil der Gesamtlaufzeit ausmacht, können wir auch einfach mehrere, z. B. 100 verschiedene Eulerzyklen ausprobieren, um mit gewisser Wahrscheinlichkeit ein etwas besseres Ergebnis zu erreichen. Verschiedene Eulerzyklen können zum Beispiel durch zufälliges durchmischen der Kanten jedes Knotens vor der Eulerzyklussuche gefunden werden.
- Der Eulerzyklus, der gefunden wurde, enthält einige Kanten, die wir nachträglich eingefügt haben, um einen Eulerzyklus zu ermöglichen. Diese Kanten müssen nicht unbedingt befahren werden. Falls die erste oder letzte Kante eines Tagesabschnitts eine solche Kante ist, können wir sie entfernen und den Tagesabschnitt dadurch kürzer machen. Für möglichst gute Ergebnisse sollte dies auch bereits bei der Aufteilung in Tagesabschnitte berücksichtigt werden.
- Bisher gehen wir davon aus, dass unser Eulerzyklus einen definierten Start- beziehungsweise Endpunkt hat, und dass der erste Tagesabschnitt auch an diesem Punkt startet. Es kann sich jedoch lohnen, zu erlauben, dass ein Tagesabschnitt diesen Startpunkt *überlappt*. Dies lässt sich erreichen, ohne die Laufzeit wesentlich zu verschlechtern: Falls die Aufteilung des Zyklus in Tagesabschnitte in einer Iteration der binären Suche nicht geklappt hat, versucht man, den ersten Abschnitt um eine Kante nach vorne zu *verschieben* und passt auch die Enden der übrigen Abschnitte entsprechend an. Falls dadurch der letzte Abschnitt lang genug werden kann, um auch die ersten, nun frei gewordenen Kanten

[3] Getestet haben wir das mit der Implementierung „Blossom V“, die hier beschrieben wird: `https://pub.ist.ac.at/~vnk/papers/blossom5.pdf`

des Zyklus abzudecken, ist man fertig. Ansonsten probiert man, das noch so lange zu wiederholen, bis der erste Abschnitt mindestens die ursprüngliche Position des zweiten Abschnitts erreicht hat.

Qualität der Ergebnisse

Da wir – wie oben festgestellt – mit einer Heuristik *keine optimalen* Ergebnisse erreichen, kann analysiert werden, wie gut die berechneten Ergebnisse tatsächlich sind. Dazu wollen wir eine *untere Schranke* verwenden, um abzuschätzen, welche Länge ein optimales Ergebnis immer haben muss.

Zuerst kann man feststellen, dass die gefundene Länge des längsten Tagesabschnittes maximal fünf mal so lang ist wie der längste Tagesabschnitt in der optimalen Lösung. Wenn wir nämlich alle Tagesabschnitte aus der optimalen Lösung zusammenfügen, hätten wir eine Lösung für nur einen Tag, und diese können wir mit unserem Algorithmus optimal berechnen. Durch das Aufteilen dieses Weges in fünf Tagesabschnitte wird die Länge nur kleiner.

Wir können jedoch noch genauere Aussagen treffen: Wenn man eine optimale Lösung hätte und bei dieser alle Tagestouren zusammenfügen würde, erhält man einen Zyklus, der alle Kanten abdeckt und einige Kanten möglicherweise mehrfach abdeckt. Die Länge der längsten Tagestour ist zumindest ein Fünftel der Gesamtlänge dieses Zyklus. Der Eulerzyklus auf dem erweiterten Graphen, den wir berechnen, ist ebenfalls ein solcher Zyklus auf dem Originalgraphen. Und wir wissen, dass unser Eulerzyklus minimale Länge hat (sofern wir den Blossom-Algorithmus für das Matching verwenden). Daher hat auch der Zyklus aus der optimalen Lösung mindestens diese Länge. Teilen wir nun also die Länge unseres Zyklus durch fünf, erhalten wir eine untere Schranke für die Länge der längsten Tagestour in der optimalen Lösung.

Die untere Schranke erlaubt es, unsere Lösungen einzuordnen. Sollte beispielsweise unsere Lösung sehr nah an der unteren Schranke sein, wissen wir, dass hier sicher nur noch wenig Optimierungsbedarf besteht. Aber Achtung: In die andere Richtung lässt sich keine Aussage treffen. Nur weil unsere Lösung weit von der unteren Schranke entfernt ist, lässt sich nicht direkt sagen, dass wir noch viel optimieren können. Es könnte schließlich auch sein, dass die untere Schranke weit von der optimalen Lösung entfernt ist. Jedoch gilt immer: Lösungen, welche kürzere Tagesstrecken als die angegebene untere Schranke berechnen, müssen falsch sein.

1.2 Alternative Lösung: ILP

Unser letzter vorgestellter Algorithmus benutzte eine *Heuristik*, um gute Ergebnisse zu finden. Allerdings sind diese Ergebnisse natürlich nicht in allen Fällen optimal. In diesem Abschnitt wird kurz eine Möglichkeit vorgestellt, welche optimale Ergebnisse auf allen Instanzen liefert. Allerdings: Da unser Problem NP-schwer ist, wird die folgende Möglichkeit also eine hohe Laufzeit haben – sogar so hoch, dass mit dieser Methode die großen Beispiele des BWINF nicht gelöst werden können. Der folgende Lösungsansatz komplementiert also die Heuristik und kann beispielsweise auf kleinen Instanzen eingesetzt werden, um dort garantiert das beste Ergebnis zu erreichen.

Der Lösungsansatz ergibt sich mit einer Reduktion auf das „Integer Linear Programming“-Problem (kurz *ILP* genannt). Dieses Problem besteht aus eine Menge an *ganzzahligen* Variablen, welche durch *Ungleichungen* eingeschränkt werden. Eine Lösung für das ILP-Problem

besteht aus einer Belegung der Variablen, mit welcher alle Ungleichungen erfüllt sind. Weiterhin kann eine Kostenfunktion angegeben werden. Mit dieser ist es möglich, den Wert einer Variablen zu minimieren. Eine ganzzahlige Lösung, welche alle Ungleichungen erfüllt sowie die Kosten minimiert, nennen wir *optimal*.

Beispielsweise könnte ein ILP-Problem wie folgt aussehen:

$$\begin{array}{lll} \text{minimiere} & x & \\ \text{gegeben} & x+y & \geq 4 \\ & 2x-3y & \geq 19 \\ & x,y \in \mathbb{Z} & \end{array}$$

Eine optimale Lösung für dieses Problem wäre nun $x = 7$, $y = -2$.

ILP ist ebenfalls NP-schwer[4]. Daher gibt es ebenfalls keinen effizienten Algorithmus, um eine optimale Lösung für eine ILP-Instanz zu finden. Jedoch wird sehr intensiv an Algorithmen geforscht, welche zwar theoretisch eine hohe Laufzeit haben, in der Praxis aber doch sehr viele Instanzen schnell lösen können. Solche Programme, wie z. B. GLPK[5], werden „ILP-Solver“ genannt.

Unsere Reduktion

Unser Ziel ist es, eine Instanz des Müllabfuhr-Problems mit der Hilfe eines ILP-Solvers zu lösen. Dazu müssen wir für eine gegebene Müllabfuhr-Instanz eine Instanz des ILP-Problems konstruieren, dieses lösen, und schlussendlich aus der Lösung des ILP-Problems unseren Tagesplan rekonstruieren. Diese Vorgehensweise heißt in der Informatik eine *Reduktion*: Wir *reduzieren* das Müllabfuhr-Problem auf ILP.

Dies bedeutet, dass wir für einen Müllabfuhr-Eingabegraph $G = (V,E)$ eine Menge an Ungleichungen definieren müssen, sodass diese einen Tagesplan ergeben. Zu diesem Zweck führt unsere Reduktion für alle Kanten in $(i,j) \in E$ fünf Variablen x_{ij1} bis x_{ij5} ein. Jede dieser Variablen soll später für die Anzahl stehen, wie oft die Kante an einem bestimmten Tag durchfahren werden soll. Nun können wir mit Hilfe dieser Variablen unser Ziel beschreiben. Beispielsweise soll jede Kante an mindestens einem Tag durchfahren werden. Dies ergibt die folgenden Ungleichungen:

$$\sum_{k=1}^{5} x_{ijk} \geq 1 \quad \forall (i,j) \in E$$

Natürlich müssen noch deutlich mehr Ungleichungen hinzugefügt werden. So muss natürlich sichergestellt werden, dass jede Tour an der Zentrale beginnt und endet. Ebenfalls dürfen natürlich nur Kanten befahren werden, wenn wir diese auf dem Weg erreichen können. Aus Platzgründen werden hier nicht alle Ungleichungen explizit beschrieben. Diese können jedoch zum Beispiel in der Arbeit „On the balanced K-chinese Postmen Problems“[6] nachgelesen werden.

Weiterhin kann mit diesem Ansatz eine Kostenfunktion definiert werden. Dazu wählen wir die maximale Tageslänge aus, welche minimiert werden soll. Insgesamt erhalten wir somit eine Menge an Ungleichungen und eine Kostenfunktion. Für dieses Problem können wir nun einen ILP-Solver befragen, welcher uns die Belegung der Variablen für eine optimale Lösung ausgibt.

[4] Sonst könnten wir unser originales Problem nicht effizient auf ILP reduzieren.

[5] `https://www.gnu.org/software/glpk/`

[6] `https://etd.lib.metu.edu.tr/upload/12618933/index.pdf`, Seiten 13 – 14

Da die Variablen die Informationen enthalten, an welchen Tag welche Straße befahren werden soll, können wir nun sehr einfach einen Tagesplan erstellen[7].

Untere Schranken

Da das ILP-Problem NP-schwer ist, kann unsere Lösung auf den großen gegebenen Beispielen (muellabfuhr5.txt bis muellabfuhr8.txt) in vernünftiger Zeit keine Lösung finden. Trotzdem können wir aus unserem Lösungsansatz Infos über die optimale Lösung finden. Dazu benutzen wir die folgende Beobachtung:

> Wenn wir eine optimale Lösung mit *rationalen Zahlen* für eine ILP-Instanz finden, dann ist jede Lösung mit *ganzen Zahlen* schlechter (oder gleich) zu der rationalen Lösung.

Dieser Satz gilt, da alle Lösungen mit nur ganzen Zahlen auch direkt Lösungen mit rationalen Zahlen sind. Diese Beobachtung ist sehr nützlich, da es möglich ist, rationale Lösungen für ILP-Probleme *effizient* zu berechnen. In der Informatik nennt man diese Methode LP-Relaxation.

Mithilfe der LP-Relaxation können wir zwar keine optimale Lösung finden, jedoch erneut eine *untere Schranke* für alle Lösungen. Da jeder Tagesplan auch immer alle Ungleichungen erfüllen muss, kann ein Tagesplan nicht besser als die optimale (rationale) Lösung zu unseren Ungleichungen sein. Allerdings kann es durchaus sein, dass die untere Schranke ein deutlich kleinerer Wert als die optimale Lösung ist; insbesondere muss keine Lösung mit Kosten gleich der unteren Schranke existieren.

1.3 Einfachere Ansätze

Beide vorgestellten Ansätze benötigen viel algorithmisches Vorwissen. Es sind auch andere Ansätze denkbar.

Ein einfacherer heuristischer Ansatz wäre zum Beispiel, vom Startpunkt aus fünf Wege zu suchen, wobei immer an den bisher kürzesten Weg die am schnellsten erreichbare, noch nicht verwendete Kante angefügt wird.

Eine Alternative für diesen Ansatz wäre auch, die Wege nacheinander zu erstellen. Dazu müsste man zuerst eine Maximallänge festlegen, und dann nacheinander fünf Pfade berechnen, die möglichst viele zuvor noch nicht besuchte Kanten enthalten, und dabei diese Maximallänge nicht überschreiten. Bei der Auswahl der Kanten wird auch immer die am schnellsten erreichbare verwendet. Falls die Maximallänge zu gering gewählt ist, werden so nicht alle Kanten abgedeckt. Falls man es aber doch schafft, alle Kanten abzudecken, hat man eine Lösung mit dieser Maximallänge als längste Länge einer Tagestour. Man kann nun binäre Suche anwenden, um eine kleine Maximallänge zu finden, bei der noch eine Lösung gefunden wird.

Genau genommen stimmt es bei diesem Ansatz nicht ganz, dass, wenn eine Lösung für eine Maximallänge gefunden wird, auch für alle größeren Maximallängen eine Lösung gefunden werden wird. Daher kann es sich lohnen, in der binären Suche die obere beziehungsweise untere Grenze dem geprüften Wert nur anzunähern, statt sie direkt auf diesen Wert zu setzen.

[7] Wir suchen für jeden Tag einen Eulerkreis auf den für diesen Tag ausgewählten Kanten, dieser wird unsere Tour.

1.4 Ergebnisse auf den Beispieleingaben

Die folgende Tabelle fasst die Ergebnisse der verschiedenen Ansätze zusammen. In den Spalten ist jeweils die *Länge der längsten Tagesstrecke*, welche von der jeweiligen Lösung berechnet wurde, eingetragen.

Die betrachteten Ansätze sind die folgenden:

ILP Optimale Lösung

H1 Zu Beginn beschriebene Heuristik, die den Blossom-Algorithmus zum Zuteilen von Knotenpaaren verwendet.

H2 Wie H1, jedoch wurden die Knotenpaare mit einem Greedy-Algorithmus zugeteilt.

G1 Greedy-Algorithmus, der parallel fünf Pfade erstellt.

G2 Greedy-Algorithmus, der mit binärer Suche eine Maximallänge bestimmt und für diese nacheinander fünf Pfade erstellt.

G3 Wie G2, aber mit modifizierter binäre Suche, die sich dem geprüften Wert langsamer annähert.

Eingabedatei	ILP	H1	H2	G1	G2	G3	Untere Schranke
`muellabfuhr0`	4	4	4	4	4	4	4
`muellabfuhr1`	18	18	18	25	18	18	18
`muellabfuhr2`	9	10	10	14	10	10	9
`muellabfuhr3`	21	22	22	24	22	22	21
`muellabfuhr4`	10	10	10	10	10	10	10
`muellabfuhr5`	n/a	1468	1468	1479	1468	1468	1464
`muellabfuhr6`	n/a	525291	558100	647578	631102	618121	477419
`muellabfuhr7`	n/a	794793	799288	1236244	813009	813009	447294
`muellabfuhr8`	n/a	2719313	2719313	3355168	3123746	3122164	2666041

Im Folgenden werden die Ergebnispfade für einige Dateien aufgeführt. Aus Platzgründen werden die konkreten Wege jedoch nur für die Dateien bis `muellabfuhr4.txt` angegeben.

muellabfuhr0.txt

Diese Lösung war bereits in der Aufgabenstellung vorgegeben.

```
Tag 1: 0 -> 8 -> 9 -> 8 -> 0, Gesamtlaenge: 4
Tag 2: 0 -> 4 -> 3 -> 2 -> 0, Gesamtlaenge: 4
Tag 3: 0 -> 8 -> 7 -> 6 -> 0, Gesamtlaenge: 4
Tag 4: 0 -> 8 -> 1 -> 2 -> 0, Gesamtlaenge: 4
Tag 5: 0 -> 6 -> 5 -> 4 -> 0, Gesamtlaenge: 4
Maximale Laenge einer Tagestour: 4
```

muellabfuhr1.txt

```
Tag 1: 0 -> 6 -> 3 -> 5 -> 0, Gesamtlaenge: 11
Tag 2: 0 -> 6 -> 3 -> 2 -> 3 -> 6 -> 0, Gesamtlaenge: 18
Tag 3: 0 -> 6 -> 7 -> 5 -> 4 -> 3 -> 6 -> 0, Gesamtlaenge: 18
Tag 4: 0 -> 6 -> 7 -> 6 -> 3 -> 1 -> 6 -> 0, Gesamtlaenge: 15
Tag 5: 0 -> 6 -> 7 -> 4 -> 0, Gesamtlaenge: 16
Maximale Laenge einer Tagestour: 18
```

muellabfuhr2.txt

```
Tag 1: 0 -> 5 -> 11 -> 8 -> 12 -> 8 -> 7 -> 9 -> 5 -> 0, Gesamtlaenge: 9
Tag 2: 0 -> 9 -> 12 -> 1 -> 7 -> 11 -> 2 -> 10 -> 9 -> 0, Gesamtlaenge: 9
Tag 3: 0 -> 6 -> 4 -> 3 -> 11 -> 3 -> 13 -> 1 -> 6 -> 0, Gesamtlaenge: 9
Tag 4: 0 -> 9 -> 7 -> 14 -> 8 -> 2 -> 14 -> 6 -> 9 -> 0, Gesamtlaenge: 9
Tag 5: 0 -> 9 -> 13 -> 14 -> 13 -> 4 -> 10 -> 14 -> 5 -> 0, Gesamtlaenge: 9
Maximale Laenge einer Tagestour: 9
```

muellabfuhr3.txt

```
Tag 1: 0 -> 14 -> 2 -> 1 -> 12 -> 14 -> 11 -> 13 -> 10 -> 2 -> 11 -> 1 -> 10
  -> 11 -> 12 -> 0 -> 10 -> 9 -> 0 -> 11 -> 6 -> 0, Gesamtlaenge: 21
Tag 2: 0 -> 2 -> 12 -> 13 -> 14 -> 7 -> 11 -> 8 -> 5 -> 14 -> 6 -> 13 -> 5
  -> 2 -> 6 -> 10 -> 7 -> 4 -> 2 -> 13 -> 1 -> 0, Gesamtlaenge: 21
Tag 3: 0 -> 13 -> 4 -> 14 -> 10 -> 5 -> 11 -> 9 -> 7 -> 5 -> 9 -> 1 -> 14
  -> 3 -> 11 -> 4 -> 5 -> 1 -> 4 -> 10 -> 3 -> 0, Gesamtlaenge: 21
Tag 4: 0 -> 7 -> 13 -> 9 -> 12 -> 10 -> 8 -> 13 -> 3 -> 12 -> 5 -> 6 -> 12
  -> 8 -> 4 -> 12 -> 7 -> 1 -> 6 -> 3 -> 4 -> 0, Gesamtlaenge: 21
Tag 5: 0 -> 8 -> 14 -> 9 -> 6 -> 4 -> 9 -> 2 -> 7 -> 3 -> 9 -> 8 -> 7 -> 6
  -> 8 -> 2 -> 3 -> 1 -> 8 -> 3 -> 5 -> 0, Gesamtlaenge: 21
Maximale Laenge einer Tagestour: 21
```

muellabfuhr4.txt

```
Tag 1: 0 -> 9 -> 8 -> 9 -> 0 -> 1 -> 2 -> 1 -> 0, Gesamtlaenge: 8
Tag 2: 0 -> 9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1 -> 0, Gesamtlaenge: 10
Tag 3: 0 -> 9 -> 8 -> 9 -> 0, Gesamtlaenge: 4
Tag 4: 0 -> 1 -> 0, Gesamtlaenge: 2
Tag 5: 0 -> 1 -> 0, Gesamtlaenge: 2
Maximale Laenge einer Tagestour: 10
```

1.5 Bewertungskriterien

Die aufgabenspezifischen Bewertungskriterien werden hier erläutert.

1. Lösungsweg

(1) *Problem adäquat modelliert*: Das Straßennetz kann sinnvoll als Graph dargestellt werden, dessen Kanten z. B. in einer Adjazenzliste gespeichert werden.

(2) *Laufzeit des Verfahrens in Ordnung*: Alle vorgegebenen Testfälle sollten in wenigen Minuten bearbeitet werden können. Falls ein optimales Verfahren gewählt wurde, reicht es, wenn damit alle Testfälle bis einschließlich `muellabfuhr4.txt` in angemessener Zeit bearbeitet werden können. Solch ein Verfahren muss aber mit einer Heuristik ergänzt werden, sodass auch auf größeren Eingaben Ergebnisse geliefert werden.

(3) *Speicherbedarf in Ordnung*: Das Programm sollte nicht unnötig viel (mehr als einige GB) Speicher auf den gegebenen Beispielen brauchen.

(4) *Verfahren mit korrekten Ergebnissen*: Die Ergebnisse müssen in dem Sinne korrekt sein, dass alle Straßen abgedeckt werden, nur an Kreuzungen oder Enden von Sackgassen gewendet wird, höchstens 5 Tage benutzt werden, und alle Tagesstrecken an der Zentrale beginnen und enden. (Hier wird nicht verlangt, dass die Länge der Tour minimal ist.) Sofern Einsendungen kürzere Tagesstrecken als die angegebene untere Schranke vorschlagen, ist eines dieser Kriterien sicher verletzt.

(5) *Verfahren mit guter Ergebnisqualität*: Die Länge der längsten Tagestour soll möglichst kurz sein. Die Ergebnisse der vorgestellten Musterlösungen sind bereits sehr gut und ähnliche Ergebnisse können Pluspunkte geben. Falls die Heuristik zu einfach ist und dadurch die Ergebnisse deutlich schlechter als der zweite Greedy-Algorithmus (G2 in der Tabelle) sind, gibt es Punktabzug; starke Abzüge gibt es bei Ergebnissen, die schlechter sind als G1.
Dabei darf aber durchaus eine Abwägung zwischen Laufzeit und Ergebnisqualität getroffen werden: Wenn die Laufzeit des Verfahrens besonders gering ist, dürfen die Ergebnisse etwas schlechter sein.

(6) *Mehrere Lösungsansätze*: Da Heuristiken offensichtlich nicht zu garantiert optimalen Ergebnissen führen, ist es sinnvoll, sich über unterschiedliche Lösungsansätze zumindest Gedanken zu machen. Wer wirklich mehrere und deutlich unterschiedliche Ansätze ausführlich beschrieben oder sogar realisiert und miteinander verglichen hat, hat Pluspunkte verdient.

2. Theoretische Analyse

(1) *Verfahren / Qualität insgesamt gut begründet*: Es muss erkannt werden, wenn das verwendete Verfahren keine optimalen Ergebnisse liefert. Soweit es nicht offensichtlich ist, sollte außerdem darauf eingegangen werden, warum das Verfahren korrekte Ergebnisse liefert, also alle Straßen tatsächlich abgedeckt werden.

(2) *Gute Überlegungen zur Laufzeit des Verfahrens*: Typischerweise sollte die asymptotische Laufzeit des Verfahrens betrachtet werden. Es ist ideal, aber nicht notwendig, hierfür die $\mathcal{O}$-Notation zu verwenden. Falls die Laufzeit im schlechtesten Fall exponentiell ist, sollte dies erkannt werden. Praktische Laufzeitmessungen können die Angabe der asymptotischen Laufzeit ersetzen. In Kombination – insbesondere, wenn die tatsächliche Laufzeit

auf den Beispielen stark von der Worst-Case-Laufzeit abweicht – kann ein Vergleich der asymptotischen und tatsächlichen Laufzeit auch Pluspunkte geben.

(3) *NP-Schwere des Problems erkannt*: Falls die NP-Schwere erkannt und korrekt begründet wird, etwa durch eine (informelle) Reduktion auf ein bekanntes NP-schweres Problem, können Pluspunkte vergeben werden.

3. Dokumentation

(3) *Vorgegebene Beispiele dokumentiert*: Es muss die berechnete maximale Länge der Tagesstrecken zu allen Beispielen vorhanden sein. Eine Lösung für `muellabfuhr0.txt` ist nicht nötig, da diese von BWINF bereits angegeben wurde.
Für alle Beispiele bis `muellabfuhr4.txt` sollten außerdem die konkreten Tagestouren dokumentiert werden. Für die übrigen Beispiele wird dies nicht gefordert.

(5) *Ergebnisse nachvollziehbar dargestellt*: Entweder die *Länge jeder Tagestour* oder aber die *Länge der längsten Tagestour* muss explizit dokumentiert sein. Es ist nicht ausreichend, nur die Touren selbst ohne Länge anzugeben.
Eine Tagestour selbst kann zum Beispiel als Liste von besuchten Knoten ausgegeben werden.