Hamiltonsche Graphen und das Traveling Salesman Problem (TSP)

Albert Kasdorf Andreas Janster Alex Bibanaev Georg Braun

09.05.2018

1 Hamiltonsche Graphen

1.1 Motivation

Das junge Pärchen Anna und Bernd planen ihre Hochzeit. Der Großteil der Aufgaben ist bereits geplant und erledigt. Nur über den Sitzplan für das Hochzeitsessen grübeln beide schon des längeren und kommen zu keinem Ergebnis. Ihr Wunsch ist es, dass während des Essens eine ausgelassen Stimmung herrscht. Daher soll neben jedem Gast, jeweils zu seiner linken und rechten Seite, zwei ihm bekannte Person sitzen.

Eines Abend als Bernd sich mit seinen alten Studienkollegen trifft kommt ihm die zündende Idee. Das Sitzplatz-Problem ließe sich mit einem Graphen modellieren. Die Gäste werden durch Knoten beschrieben und die Bekanntschaft zwischen zwei Gästen durch eine Kante, siehe Abbildung 1a. Wenn es einen Kreis in dem Graphen gibt, in dem jeder Knoten genau einmal vorkommt, wäre das Sitzplatz-Problem gelöst. Nach einigen Versuchen hat er einen Kreis gefunden, der alle Knoten miteinander verbindet, siehe Abbildung 1b.

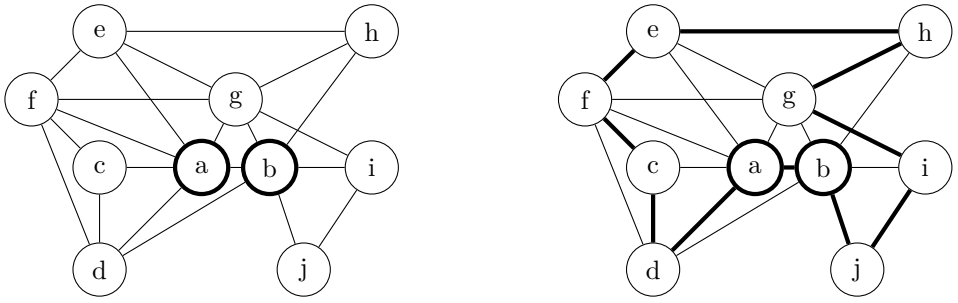


Abbildung 1: Links: Ein Graph der die Bekanntschaft zwischen den Hochzeitsgästen modelliert. Rechts: Ein Kreis in dem Bekanntschaftsgraphen der jeden Gast zwei im bekannte Gäste zuweist.

1.2 Definition

Wie es bereits aus dem einleitenden Beispiel zu erkennen ist, wird in einem Graphen ein Kreis gesucht, der alle Knoten so miteinander verbindet, dass außer dem Start- und Endknoten kein Knoten doppelt vorkommt. Ein Graph mit diesen Eigenschaften wird wie folgt definiert:

Definition 1. *Ein Graph heißt **hamiltonsch** oder **Hamilton-Graph**, wenn in ihm ein Kreis existiert, der jeden Knoten genau einmal enthält. Ein solcher Kreis heißt auch **Hamilton-Kreis**. [1]*

Die Hamilton-Eigenschaft ähnelt der Euler-Eigenschaft, nur das bei dem ersten alle Knoten genau einmal durchlaufen werden und beim zweiten alle Kanten. Ein Graph kann dabei beide, eine oder keine der beiden Eigenschaft in sich vereinen, siehe Abbildung 2.

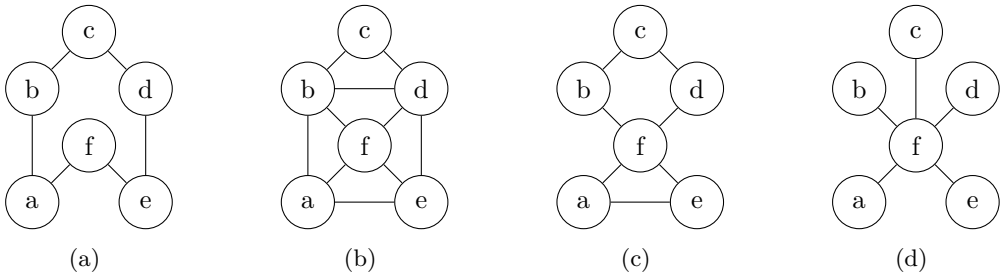


Abbildung 2: Der Graph (a) ist sowohl eulersch als auch hamiltonsch, (b) ist nur hamiltonsch, (c) ist nur eulersch und (d) ist weder noch.

Ähnlich wie mit den Euler-Graphen werden in den folgenden zwei Abschnitten einige Kriterien untersucht, um herauszufinden ob ein Graph einen Hamilton-Kreis besitzt oder nicht. Um es gleich vorweg zu nehmen, bisher konnte kein allgemein gültiges Kriterium gefunden werden, das die Existenz eines Hamilton-Kreises in polynomieller Laufzeit bestätigt.

1.3 Knotengrad

Zuerst werden die vollständigen Graphen mit einer Knotenanzahl $n \geq 3$ betrachtet.[1]

Definition 2. *Ein Graph $G = (V, E)$ heißt **vollständig**, wenn jeder Knoten zu jedem anderen Knoten benachbart ist. Da der Graph also nur von der Anzahl der Knoten n abhängt, bezeichnet man ihn auch als K_n. [1]*

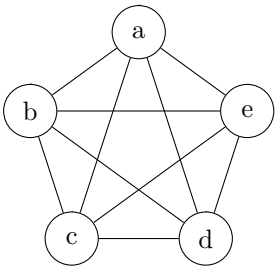
Um einen Kreis in einem Graphen "einzeichnen" zu können, werden pro Knoten mindestens zwei Kanten benötigt, eine Kante um in den Knoten "hineinzugehen" und eine um "herauszugehen". Da jeder Knoten des vollständigen Graphen zu jedem anderen Knoten adjazent ist, kann bei der Kreisbildung von jedem Knoten noch eine Kante zu einem unbesucht Knoten gefunden werden. Da auch der Start- und Endknoten adjazet sind, kann am Ende der Hamilton-Kreis geschlossen werden, siehe Abbildung 3a. Daraus folgt, dass jeder vollständige Graph über einen Hamilton-Kreis verfügt.

Ein vollständiger Graph kann auch über den Minimalgrad beschrieben werden, siehe Definition 3. Dabei gilt der Zusammenhang $\delta(K_n) = n - 1$.

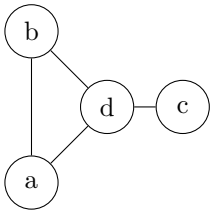
Definition 3. *Sei $G = (V, E)$ ein Graph. Der Wert $\delta(G)$ gibt den **Minimalgrad** in G an, d.h. die kleinste in G vorkommende Gradzahl, und $\Delta(G)$ den **Maximalgrad** eines Knoten in G, d.h. die größte in G vorkommende Gradzahl. [1]*

Neben den vollständigen Graphen existieren auch Graphen, bei dem die Kantenanzahl nicht ausreicht um einen Hamilton-Kreis zu bilden. In der Abbildung 3b bleibt die Bildung eines Kreises in dem Knoten d stecken. Hieraus kann das Lemma 1 abgeleitet werden.

[1] Es werden die vollständigen Graphen mit einer Knotenanzahl von eins und zwei ausgeschlossen, da es sich im ersten Fall um einen Punkt und im zweiten um eine Linie handelt.



(a)



(b)

Abbildung 3: Links: Ein vollständiger Graph K_5 der immer einen Hamilton-Kreis enthält. Rechts: Ein Graph mit einem Minimalgrad von eins kann keinen Hamilton-Kreis enthalten.

Lemma 1. *Ein hamiltonscher Graph enthält keinen Knoten mit Grad 1. [1]*

Im Jahre 1952 hat der britischer Physiker Dirac ein Kriterium aufgestellt mit der die Existenz eines Hamilton-Kreises in einem Graphen nachgewiesen werden kann, siehe Satz 1.

Satz 1. *Sei G ein einfacher Graph mit mindestens drei Knoten, für den außerdem $\delta(G) \geq 0.5 \cdot n$ gilt. Dann enthält der Graph einen Hamilton-Kreis. [1]*

Die Kernaussage ist dabei, das jeder Knoten des Graphen eine gewisse Mindestanzahl an Kanten zu benachbarten Knoten besitzen soll. Hier durch werden, zum einem isolierte Knoten und zum anderen Bäume ausgeschlossen.

Es sei zu erwähnen, dass der Satz von Dirac eine hinreichende jedoch keine notwendige Bedingung für die Existenz eines Hamilton-Kreises ist. Es gibt Graphen die das Kriterium von Dirac nicht erfüllen und trotzdem einen Hamilton-Kreis besitzen, siehe Abbildung 4.

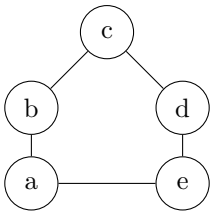


Abbildung 4: Ein Graph der einen Hamilton-Kreis enthält, jedoch das Kriterium von Dirac nicht erfüllt.

1.4 Zusammenhang

Neben den Knotengrad ist der Zusammenhang des Graphens eine weitere wichtige Eigenschaft, siehe Lemma 2.

Lemma 2. *Falls ein Graph hamiltonsch ist, dann ist er auch zusammenhängend. [1]*

Nicht zusammenhängende Graphen können laut der Definition 1 keinen Hamilton-Kreis enthalten. Die Rückrichtung gilt jedoch nicht, ein Baum ist zwar zusammenhängend, aber nicht hamiltonsch.

Satz 2. *Für jeden hamiltonschen Graphen gilt: Wenn k Knoten aus dem Graphen gelöscht werden, zerfällt der Graph in höchstens k Zusammenhangskomponenten. [1]*

Den Satz 2 kann man sich dabei wie folgt vorstellen. Man nehme einen Faden und verbinde diesen zu einem Kreis, der den Hamilton-Kreis in einem Graphen darstellt. Wenn der Faden jetzt an einer Stelle zerschnitten wird, entsteht maximal ein Fadenstück. Mit jedem weiteren Schnitt zerfällt ein Fadenstück zu maximal zwei weiteren Fadenstücken, es können niemals mehr Fadenstücke entstehen als Schnitte getätigt wurden. Zerfällt hingegen der Graph in mehr Zusammenhangskomponenten als Schnitte getätigt wurden, so kann in dem Graphen kein Hamilton-Kreis existiert haben, siehe Abbildung 5.

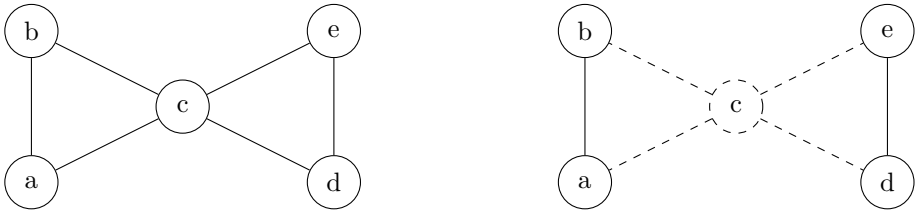


Abbildung 5: Durch das Herauslösen des Knoten c entstehen zwei Zusammenhangskomponenten. Ein Hamilton-Kreis kann in diesem Graphen nicht vorhanden sein.

An dem Petersen-Graph kann man zeigen, dass nie mehr Komponenten entstehen, als Knoten aus dem Graphen herausgelöst werden, und trotzdem enthält er keinen Hamilton-Kreis, siehe Abbildung 6.

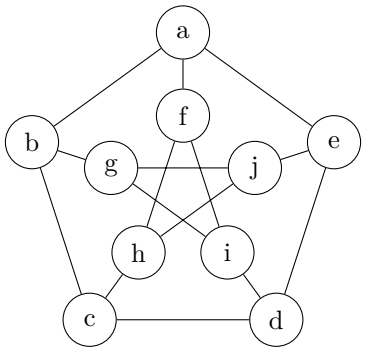


Abbildung 6: Der Petersen-Graph enthält keinen Hamilton-Kreis.

2 Traveling Salesman Problem

Das *Traveling Salesman Problem* oder *Problem des Handlungsreisenden*, wie es auf deutsch heißt, ist ein kombinatorisches Optimierungsproblem und ein Anwendungsfall der Hamilton Kreise. Die klassische Herausforderung besteht darin, eine optimale Reihenfolge für den Besuch einer vorgegebenen Menge von Orten so zu planen, dass die zurückgelegte Strecke des Handlungsreisenden möglichst kostengünstig. Zusätzlich soll kein Ort doppelt besucht werden und die erste Station ist gleich der letzten Station.

2.1 Definition

Das allgemeine *Traveling Salesman Problem* ist wie folgt definiert:

Definition 4. *Traveling Salesman Problem (TSP): Gegeben sei ein vollständiger Graph K_n mit positiven Gewichten c(e) auf den einzelnen Kanten e. Gesucht ist ein Hamilton-Kreis C in K_n mit minimalem Gewicht c(C), wobei gilt*

$$c(C) = \sum_{e \in C} c(e)$$

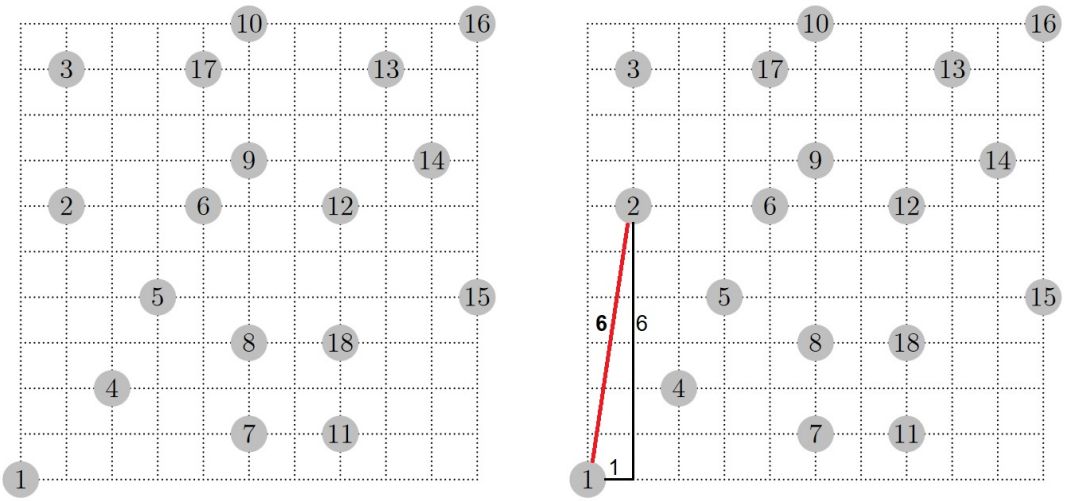
In der Regel wird beim TSP von einem vollständigen Graphen ausgegangen, ansonsten ist im vorliegenden Problem eine Rundreise nicht möglich. Beim allgemeinen *asymmetrischen TSP* können die Kanten in Hin- und Rückrichtung unterschiedliche Längen haben, so dass dieses Problem mit Hilfe eines gerichteten Graphen modelliert werden muss. Falls für alle Knotenpaare $c(u, w) = c(w, u)$ gilt, dann spricht man von einem *symmetrischen TSP*. Dieses kann durch ungerichteten Graphen repräsentiert werden.

2.2 Modellierung eines Problems

Das TSP-Problem tritt in der Praxis in vielen Anwendungen auf. Hierzu gehören zum Beispiel Optimierungsprobleme in der Verkehrsplanung, in der Logistik oder bei Entwürfen von integrierter Schaltungen und Mikrochips. Damit mathematische Verfahren zur Lösung verwendet werden können, muss eine reale Situation durch ein einfaches Modell repräsentiert werden. Orte oder spezielle Punkte werden dabei durch Knoten abgebildet. Die Kantengewichte beschreiben dabei beispielsweise die Entfernung zwischen zwei Orten.

2.2.1 Das Bohren von Leiterplatten

Ein solches Beispiel aus der Praxis ist das Herstellen einer elektronischen Leiterplatine. Dabei geht es darum, dass möglichst schnell n Löcher in eine Platine gebohrt werden müssen. Je schneller eine Platine fertig ist, desto mehr können produziert werden. Das Bohren eines einzelnen Lochs ist ohne Qualitätsverlust nicht zu beschleunigen. Um jedoch Zeit zu sparen, darf der Bohrer keine unnötigen Wege zurücklegen. Die meisten Fräs- oder Bohrmaschinen in der Leiterplattenindustrie sind lediglich mit zwei Motoren bzw. Schienen ausgestattet um die gesamte Platine zu erreichen. Deshalb kann man sich das System als *2D-Koordinatensystem* vorstellen, siehe Abbildung 7a. Der eine Motor steuert den Bohrer auf der X-Achse, also die rechts-links-Bewegung. Der andere fährt den Bohrer entlang der Y-Achse bzw. nach vorne oder hinten. Damit nur die minimale Wegstrecke zurückgelegt wird, also keine Zeit verschwendet wird, muss der optimale Zyklus für den Bohrer errechnet werden. Wenn man bedenkt, dass der Bohrer sich über Bohrloch *1* befindet und zu Loch *2* kommen möchte, wird klar, dass der eine Stellmotor seine Bewegung schneller



(a) (b)

Abbildung 7: (a) Beispielbohrungen einer Leiterplatine. (b) Bohrerbewegung zwischen Loch 1 und Loch 2

beendet hat, als der andere und deshalb eine Ruhephase hat bzw. auf den zweiten Stellmotor warten muss. Es ist also wichtig, wie der Abstand zwischen zwei Löchern bestimmt wird.

Es reicht nicht den einfachen Abstand zwischen den beiden Punkten auszumessen oder auszurechnen (euklidischer Abstand). Der maximale Abstand muss genutzt werden:

$$d_{max} = max\left\{|x_2 - x_1|, |y_2 - y_1|\right\}$$

In Abbildung 7b sind beispielhaft die Kantengewichte der Kante *1-2* in Millisekunden sowie die Bewegung der zwei Stellmotoren zu sehen. Während der Stellmotor auf der X-Achse lediglich eine Millisekunden benötigt um den Bohrer auf die korrekte Position zu manövrieren, benötigt der Stellmotor auf der Y-Achse ganze 6 Millisekunden. Das Gewicht der Kante ist also mit *6* zu bemessen.

Bedenkt man nun, das man eine Zeit-optimierte Rundreise für den Bohrer finden möchte, bildet man aus dem Graphen einen vollständigen Graphen. Es soll also jeder Knoten mit allen anderen verbunden sein. In dem Beispiel geht man davon aus, dass der Bohrer sich über die ganze Platine ohne Hindernisse bewegen kann. An dem vollständigen Graphen können danach die Lösungsmethoden des TSP angewendet werden.

2.2.2 Straßenkarte eines Autobahnnetz

Bedenkt man nun, das eine Zeit optimierte Rundreise für den Bohrer gefunden werden soll, bildet man aus dem Graphen einen vollständigen Graphen. Es soll also jeder Knoten mit allen anderen verbunden sein. In dem Beispiel geht man davon aus, dass der Bohrer sich über die ganze Platine ohne Hindernisse bewegen kann. An dem vollständigen Graphen können danach die Lösungsmethoden des TSP angewendet werden.

2.2.3 Straßenkarte eines Autobahnnetzes

Sollen die Lösungsmethoden des TSP's an einer Straßenkarte, zum Beispiel dem Autobahnnetz angewendet werden, kann man ebenfalls einen vollständigen Graphen bilden obwohl nicht alle Orte mit einer direkten Autobahn verbunden sind. Die imaginären Strecken führen dabei nicht in die Innenstadt bzw. zum Knoten selbst sondern führen direkt daran vorbei, siehe Abbildung 8. Die Streckenkosten der jeweiligen Teilstrecken werden addiert als würden diese durch den Knoten führen.

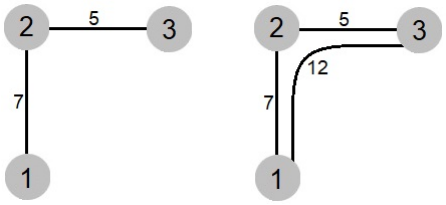


Abbildung 8: Bildung eines vollständigen Graphen

Eine andere Lösung des Problems ist ein zusätzliches Kantengewicht zu erstellen. Da der existierende Graph vollständig sein muss, werden die noch nicht existierenden Kanten hinzugefügt und mit dem zusätzlichen Gewicht *1* abgespeichert. Die existierenden Straßen bzw. Kanten werden mit einer *0* versehen.

An diesem vollständigen Graphen kann man nun die Lösungsmethoden des TSP's anwenden. Bei jeder genutzten Straße bzw. Kante wird der Wert des zusätzlichen Kantengewichtes auf einen Zähler addiert. Beträgt der Wert dieses Zählers bei gefundener günstiger Route immer noch *0*, ist alles in Ordnung, es wurden nur existierende Straßen genutzt. Andernfalls muss eine neue Route gesucht werden.

2.3 Lösungsmethoden

Grundsätzlich gibt es zwei verschiedene Kategorien von Lösungsansätzen: *exakte Algorithmen* und *Heuristiken*. Unter der Vorraussetzung einer beliebig langen Laufzeit, finden *Exakten Algorithmen* eine Optimallösung. *Heuristiken* dagegen garantieren nicht, dass eine optimale Lösung gefunden wird, sondern versuchen innerhalb eines Zeitrahmens eine zulässige Lösung zu finden. Unter mithilfe von Entscheidungsregeln wird sich dabei möglichst nah an die Optimallösung angenähert.

2.3.1 Exakte Algorithmen

Im Folgenden wird das Problem von *exakten Algorithmen* untersucht. Dazu wird ein einfaches Beispiel eines TSP-Problems mit vier Knoten betrachtet. Ausgehend vom ersten Knoten sollen

alle anderen Knoten einmal besucht werden. Unter der Bedingung, dass am Ende die günstigste Tour (Hamiltonkreis) ausgewählt werden soll.

Eine sehr intuitive Herangehensweise an das Problem ist, wir erzeugen eine Tour nach der anderen. Dabei bestimmen wir die Länge jeder Tour und merken uns stets die bislang günstigste. Die Abbildung 9 zeigt alle erzeugten Touren für unser Szenario.

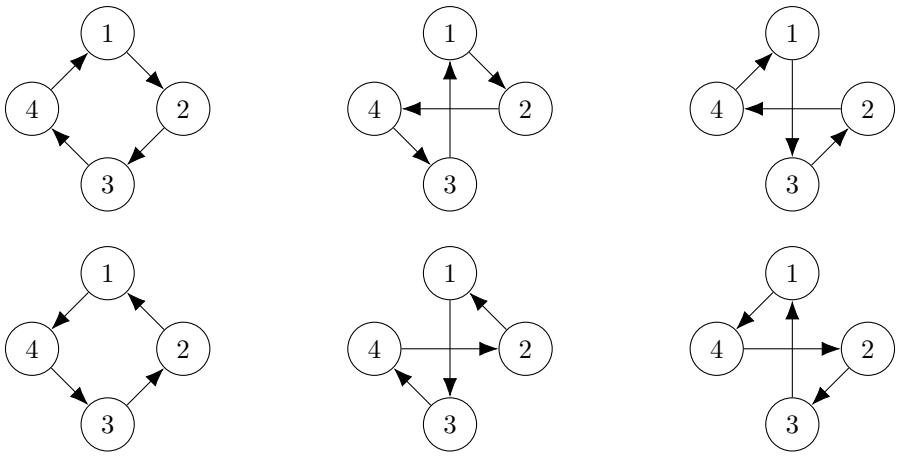


Abbildung 9: Alle Lösungen für einen vollständigen Graphen mit vier Knoten.

Das ermitteln der optimalen Lösung schien recht einfach, doch ein solcher Einsatz ist in der Praxis unbrauchbar. Die Begründung dafür liegt in der benötigten Rechenzeit für die Ermittlung der Lösung. Diese spielt nämlich die entscheidende Rolle, wenn es um das Lösen von TSP-Problemen geht. Die angewendete intuitive Herangehensweise war zwar ein exakter Algorithmus, jedoch ist diese Methode für die meisten Anwendungsfälle in der Praxis unbrauchbar.

Um die Ursache dafür zu verstehen, ist zuerst einmal zu klären, wie viele verschiedene Hamilton-Kreise es in einem vollständigen Graphen mit n Knoten gibt: Wenn man von einem beliebigen Knoten v_0 beginnt, dann kann die Tour als Nächstes einen der $n-1$ anderen Knoten enthalten. Unter der Voraussetzung das dieser Knoten $v_1 \neq v_0$ ist, verbleiben für die Auswahl des nächsten Knotens noch $n-2$ Möglichkeiten. Wird diese Gedankenfolge fortgesetzt, dann erschließt sich demnach, dass ein vollständiger asymmetrischer Graph genau $(n-1)!$ Touren enthält. Beim *symmetrischen TSP* wird nicht zwischen einer Tour und ihrer Rückrichtung unterschieden. Für die Lösungen in Abbildung 9 würde es bedeuten, dass die Lösungen der unteren Reihe wegfallen. Somit gilt:

Satz 3. *Ein vollständiger symmetrischer Graph enthält genau* $0.5 * (n-1)!$ *Hamilton-Kreise.*

Die Fakultätsfunktion wächst sehr. Für das *Bohren von Leiterplatten* Problem aus Abbildung 7, hat es zur Folge, dass $0.5 * 21! = 2.554.547.108.585.472.0000$ Touren generiert und jeweils die Kosten berechnet werden müssen. Für die Beispiel-Herangehensweise, muss man für jede Tourlänge n Additionen vornehmen, das wären bei 18 Löchern rund 3,2 Billiarden Additionen.

Daraus lässt sich die Schlussfolgerung ziehen, dass für praxisrelevante Szenarien es eine unvermeidbare Laufzeitexplosion entsteht. Das liegt daran, dass das TSP-Problem zu einer Klasse von sehr schwierigen Problemen gehört, den sogenannten NP-vollständigen Problemen. Es wird angenommen, dass jeder deterministische Algorithmus zur exakten Lösung dieser Probleme mindestens exponentiell viele Rechenschritte ausführen muss.

Das ist die Ursache dafür, dass in der Praxis Heuristiken verwendet werden, aber auch weil man oft gar nicht an der Optimallösung interessiert ist. Aufgrund der Tatsache, dass die verwendeten Daten schon Fehler enthalten oder nicht genau ermittelt werden können. Zum Beispiel eine Städtetour durch Europa mit einem Fahrzeug ist von vielen Faktoren abhängig wie: Fahrzeugtyp, Wetter oder Verkehr. Man kann also zum Schluss kommen, dass in solchen Szenarien die Forderung nach Optimalität überzogen ist.

3 Algorithmen

3.1 Nearest-Neighbor-Heuristik

Die Nearest-Neighbor Heuristik ist ein heuristisches Verfahren, welches genutzt wird um eine Lösung für das Problem des Handlungsreisenden zu ermitteln. Es handelt sich dabei um einen Greedy-Algorithmus der wie folgt funktioniert:

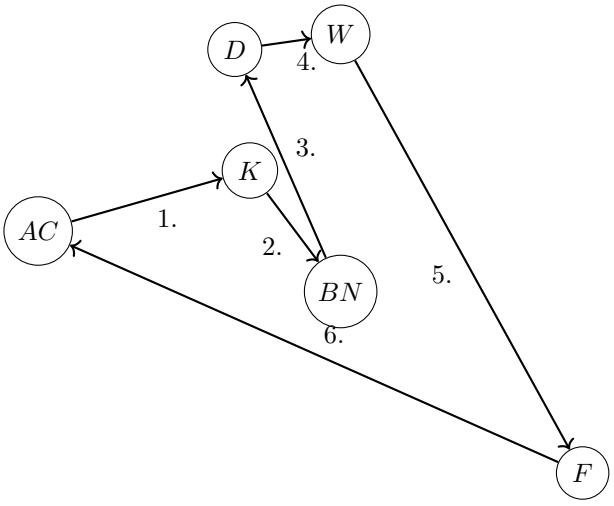
Input : Ein vollständiger Graph T mit Kantengewichten $c(e)$
Output : Ein Hamilton-Kreis
Schritt 1: : Wähle einen beliebigen Knoten als Startknoten v
Schritt 2: : Ermittle die niedrigste Kante, welche den aktuellen Knoten v mit einem unbesuchten Knoten v_u verbindet.
Schritt 3: : Setze $v = v_u$
Schritt 4: : Wenn noch nicht alle Knoten besucht wurden gehe wieder zu Schritt 2.
Schritt 5: : Füge die Kante vom letzten besuchten Knoten zum Startknoten hinzu um den Kreis zu schließen.

Algorithmus 1 : Nearest-Neighbor Algorithmus

Eine beispielhafte Anwendung des Algorithmus wird anhand einer Routenfindung zwischen den Städten Aachen (AC), Bonn (BN), Düsseldorf (D), Frankfurt (F), Köln (K) und Wuppertal (W) demonstriert. Angenommen die Entfernungen entsprechen den Angaben in der Tabelle in Abbildung 10

	AC	BN	D	F	K	W
AC		91	80	259	70	121
BN	91		77	175	27	84
D	80	77		232	47	29
F	259	175	232		189	236
K	70	27	47	189		55
W	121	84	29	236	55	

(a)



(b)

Abbildung 10: Links: Entfernung der Städte in km. Rechts: Tour des Nearest-Neighbor Algorithmus mit Startknoten AC.

Mit Aachen als Startpunkt würde der Algorithmus die Tour in Abbildung 10 konstruieren und ein Ergebnis mit Gesamtkosten von 698 Kilometern liefern. Es wird auch deutlich, dass diese Lösung nicht das beste Ergebnis ist. Tatsächlich lässt sich ein relativ simples Beispiel konstruieren welches zeigt, dass die Nearest-Neighbor Heuristik auch eine beliebig schlechte Lösung für das

Problem des Handlungsreisenden liefern kann. In Abbildung 11 (a) ist dieser Graph mit den jeweiligen Kantengewichten dargestellt. Wählt man den Knoten 1 als Startknoten und fordert, dass $x > 1$ ist, so liefert der Nearest-Neighbor Algorithmus den Hamilton-Kreis in Abbildung 11 (b) als Ergebnis. Das Ergebnis enthält eine Kante mit variablen Kantengewicht. Die Kosten betragen also $1 + 1 + 1 + x = x + 3$. Aufgrund der Variable kann das Ergebnis des Algorithmus beliebig schlecht sein. Ein weiterer Hamilton-Kreis ist in Abbildung 11 (c) dargestellt. In diesem Kreis betragen die Kosten $1+2+1+2 = 6$. Falls $x > 3$ gilt handelt es dabei um einen optimalen Kreis.

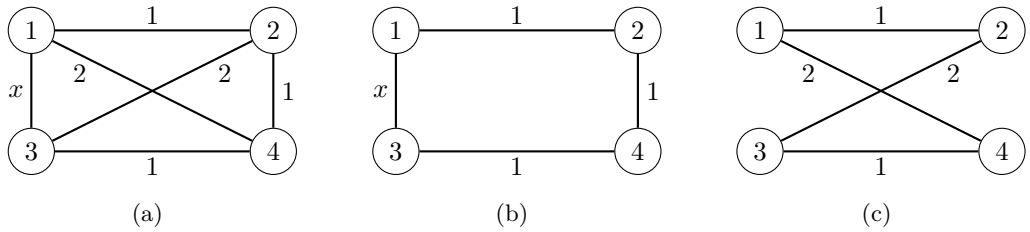


Abbildung 11: Links: Beispiel Graph. Mitte: Ergebnis Nearest-Neighbor. Rechts: Optimale Lösung falls $x > 3$

3.2 Doppelter-Baum-Algorithmus

3.2.1 Funktionsweise

Der Doppelte-Baum Algorithmus von Rosenkranz, Stearns und Lewis aus dem Jahr 1977 berechnet einen Hamilton-Kreis. Zunächst wird der Algorithmus formal beschrieben und anschließend anhand eines Beispiels illustriert.

Der Algorithmus sieht wie folgt aus:

Input : Ein vollständiger Graph K_n mit Kantengewichten $c(e)$
Output : Ein Hamilton-Kreis
Schritt 1: : Konstruiere einen minimal spannenden Baum T von K_n.
Schritt 2: : Verdopple alle Kanten von T (daraus resultiert ein eulerscher Graph T_d).
Schritt 3: : Berechne eine Euler-Tour in T_d.
Schritt 4: : Durchlaufe die Euler-Tour von einem Startknoten aus. Falls dabei ein Knoten schon besucht wurde, nehme die Abkürzung zum nächsten unbesuchten Knoten auf der Tour.

Algorithmus 2 : Doppelter-Baum-Algorithmus

Um ein besseres Verständnis für den Algorithmus zu bekommen wird dieser anhand eines Beispielgraphen erläutert. Der vollständige Graph für den ein Hamilton-Kreis gefunden werden soll ist in Abbildung 12 dargestellt. Der Algorithmus fordert als Eingabe einen vollständigen Graphen mit Kantengewichten. Als Kantengewichte wird der euklidische Abstand zwischen den jeweiligen Knoten genutzt. Für eine übersichtlichere Darstellung wurde in der Abbildung auf die Einzeichnung der Kantengewichte verzichtet.

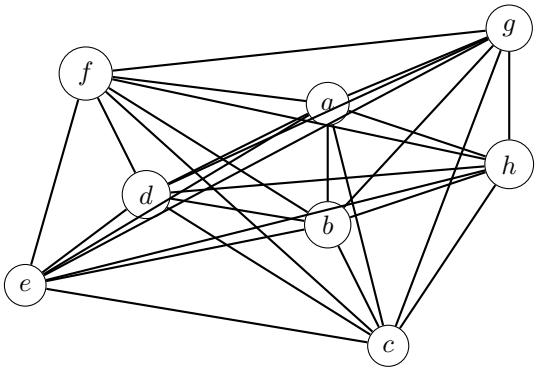


Abbildung 12: Vollständiger Beispielgraph (Kantengewichte nicht eingezeichnet)

Der erste Schritt des Algorithmus erfordert die Konstruktion eines minimal spannenden Baumes T. Dieser ist in Abbildung 13 dargestellt.

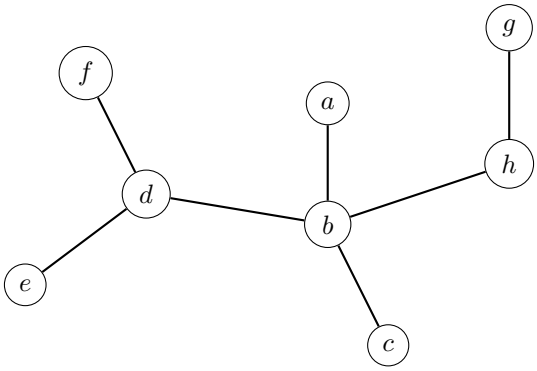


Abbildung 13: Minimal spannender Baum

Im zweiten Schritt werden die Kanten aus dem minimal spannenden Baum T verdoppelt, sodass ein eulerscher Graph T_d entsteht. In diesem Graph wird wie im nächsten Schritt gefordert eine Euler-Tour konstruiert. In Abbildung 14 ist ein Beispiel für eine Euler-Tour eingezeichnet.

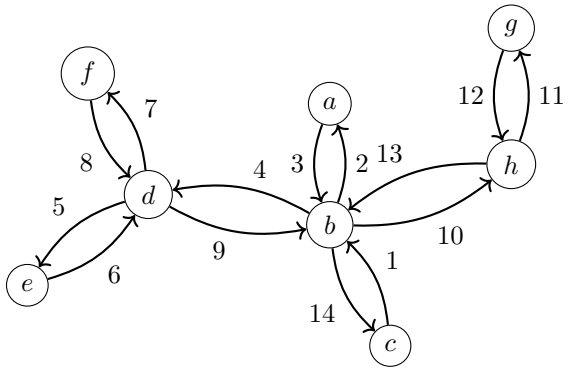


Abbildung 14: Euler-Tour

Anhand der dargestellten Euler-Tour wird nun die Vorgehensweise des vierten Schritts im Algorithmus erläutert. In diesem Schritt wird der Hamilton-Kreis ermittelt. Als Startpunkt wird der Knoten c gewählt. Der resultierende Kreis ist in Abbildung 15 dargestellt. Von c beginnend foglt man der Euler-Tour zu b. Von dort aus geht es weiter zu a. Bei einem Versuch der Euler-Tour weiter zu folgen stellt man fest, dass diese zum bereits besuchten Knoten b führt. Der nächste unbesuchte Knoten in der Euler-Tour ist der Knoten d. Um dort hin zu kommen wird die Kante von a zu d gewählt (Erinnerung: Es handelt sich um einen vollständigen Graphen. Deshalb ist es möglich die direkte Verbindung zu nutzen). Von dort aus wird wieder der Euler-Tour zu e gefolgt. Auch dort ist die Verfolgung der Euler-Tour zu d nicht mehr möglich, da dieser Knoten bereits besucht wurde. Der nächste unbesuchte Knoten auf der Tour ist f. Wie auch zuvor wird nun die direkte Kante von e nach f genutzt. Bei dem Versuch der Euler-Tour wieder zu folgen bemerkt man, dass der nächste Knoten d auch schon besucht wurde. Der darauf folgende Knoten b wurde auch schon besucht. Erst der darauf folgende Knoten h wurde noch nicht besucht, sodass die Kante von f nach h genutzt wird. Nach dem vorgestellten Schema wird weiter verfahren bis letztendlich alle Knoten besucht wurden. Die zum Erreichen der Knoten genutzten Kanten bilden somit den resultierenden Hamilton-Kreis.

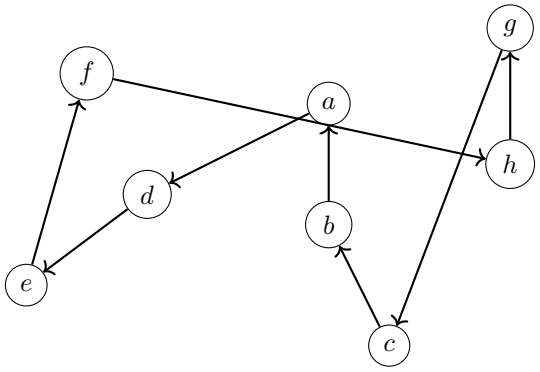


Abbildung 15: Hamilton-Kreis

3.2.2 Dreiecksungleichung

Mit den Kosten c zweier Knoten bezeichnet man das Kantengewicht der Kante, welche diese Knoten verbindet.

Definition 5. *Die Dreiecksungleichung garantiert bei einem vollständigen Graphen, dass für alle Knoten u,v und w gilt:*

$$c(u, v) \leq c(u, w) + c(w, v) \tag{1}$$

Im Kern sagt dies aus, dass der direkte Weg von einem Knoten u nach v kürzer ist, als der Umweg über einen zusätzlichen Knoten w, siehe Abbildung 16.

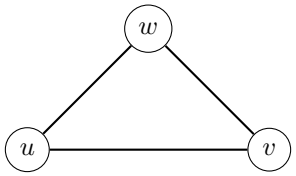


Abbildung 16: Dreiecksungleichung

3.2.3 Abschätzung des Doppelten-Baum-Algorithmus

Der Doppelter-Baum-Algorithmus liefert zwar einen Hamilton-Kreis, jedoch ist dieser nicht zwangsläufig optimal. Erweitert man die Eingabe des Algorithmus dahingehend, dass die Dreiecksungleichung erfüllt ist, so lässt sich das Ergebnis zumindest eingrenzen. Eine obere Schranke wird wie folgt festgelegt. Die durch den Algorithmus bestimmte Tour ist maximal doppelt so lang wie eine optimale Tour. Formal lässt es sich wie folgt ausdrücken:

Satz 4. *K_n sei ein vollständiger Graph mit Kantengewichten, welche die Dreiecksungleichung erfüllen. Ferner sei T' das Ergebnis des Doppelten-Baum-Algorithmus und OPT eine optimale Lösung. Dann gilt*

$$c(T') \leq 2 * c(OPT). \tag{2}$$

Dieser Satz lässt sich wie folgt beweisen. Die Tour T' kann maximal doppelt so lang sein wie die Kosten des minimalen Spannbaums T.

$$c(T') \leq 2 * c(T) \tag{3}$$

Die Begründung dafür ist, dass im Algorithmus der minimal-spannende Baum verdoppelt wird und somit die Euler-Tour mit der Länge $2 * c(T)$ entsteht. Bei der Berechnung der Hamilton-Tour wird die Euler-Tour verfolgt. Ist dabei auf dem Weg zum nächsten Knoten ein bereits besuchter Knoten vorhanden, so wird der direkte Weg zu diesem genommen (siehe Algorithmus Schritt 4). Aufgrund der Dreiecksungleichung ist der direkte Weg kürzer als der zuvor geplante Umweg über den schon bereits besuchten Knoten. Somit wurde eine obere Schranke für das Ergebnis des Algorithmus aufgezeigt. Eine untere Schranke lässt sich durch folgende Annahme finden: Der Graph hat wegen der Dreiecksungleichung nur Kantengewichte ≥ 0. Entfernt man eine Kante aus der optimalen Tour OPT entsteht ein spannender Baum. Dieser ist nicht billiger als der minimal-spannende Baum T. Es gilt also:

$$c(T) \leq c(OPT) \tag{4}$$

Die Kombination aus der oberen und unteren Schranke führt zu folgendem Ergebnis, das der obigen Gleichung 2 entspricht.

$$c(T') \leq 2 * c(T) \leq 2 * c(OPT) \tag{5}$$

4 Literatur

[1] Christina Büsing. *Graphen-und Netzwerkoptimierung.* Springer-Verlag, 2010.

[2] Dieter Jungnickel. *Graphen, Netzwerke und Algorithmen.* BI-Wissenschaftsverlag, 1994.