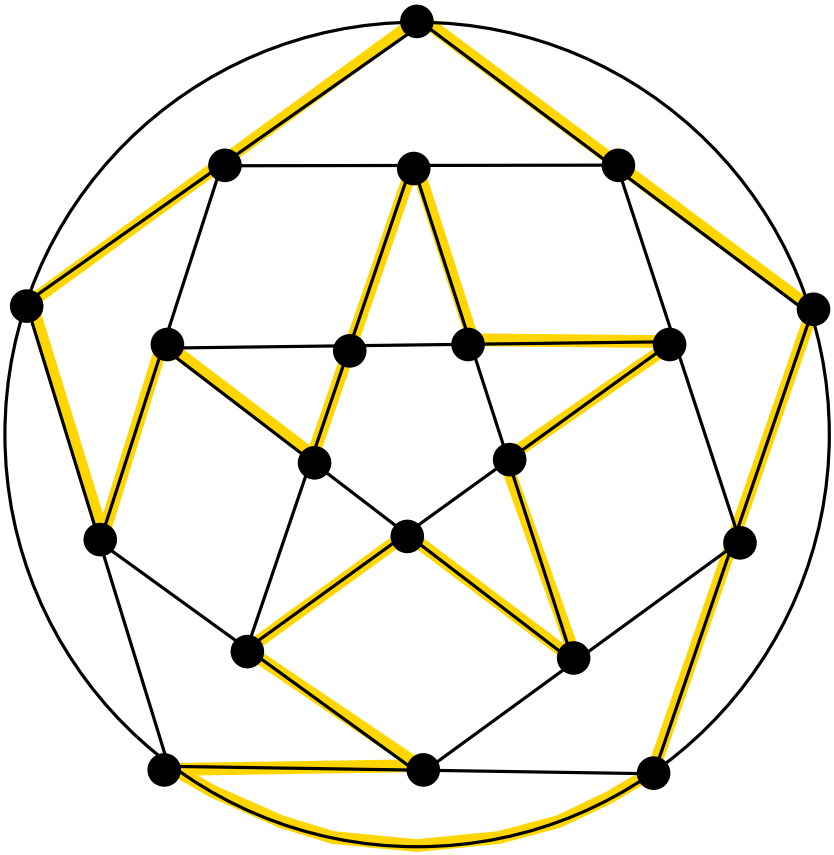
Vorlesung
Diskrete Mathematik

Stefan Felsner

felsner@inf.fu-berlin.de

10. Oktober 1999



Ausgearbeitet für das Schülerseminar 1999 am Fachbereich Mathematik der Freien Unversität Berlin.

Inhaltsverzeichnis

1 Rundreiseprobleme und Graphen

1.1 Einleitung

Diese Vorlesung ist gedacht als Führung durch einige Teilgebiete der Diskreten Mathematik. Den roten Faden für unsere Tour soll ein populäres Problem aus der kombinatorischen Optimierung liefern, das *Traveling Salesman Problem* (Problem des Handlungsreisenden).

Ein Vertreter soll Kunden in n verschiedenen Städten besuchen. Die Fahrzeit zwischen je zwei Städten ist bekannt. Verständlicherweise möchte der Vertreter die Gesamtfahrzeit minimieren, um bald wieder zu Hause zu sein. Sein Problem ist nun, die Städte in der 'richtigen' Reihenfolge anzufahren, also eine optimale Rundreise festzulegen.

Die Eingabe für das Problem ist eine Menge von Zahlen, die die Fahrzeiten zwischen je zwei Städten angeben. Der Einfachheit halber nehmen wir an, daß es n Städte gibt, die mit $1, 2, .., n$ durchnummeriert sind. Dann können die Eingabezahlen in einer $n \times n$ Matrix $C = (c_{i,j})$ organisiert werden, der Eintrag $c_{i,j}$ gibt die Fahrzeit zwischen Stadt i und Stadt j an. Wir nennen C die *Kostenmatrix* für das Problem und werden immer annehmen, daß C eine positive Kostenmatrix ist, d.h. $c_{i,j} \geq 0$ für alle i, j. Gesucht ist nun eine Reihenfolge (*Permutation*) π von $\{1, 2, .., n\}$, die folgenden Ausdruck – die Gesamtfahrzeit – minimiert:

$$\mathrm{cost}(\pi) := \sum_{i=1}^{n-1} c_{\pi(i),\pi(i+1)} + c_{\pi(n),\pi(1)}$$

Beispiel 1. Die Auswertung der Kosten einiger Rundreisen ergibt:

$$C = \begin{pmatrix} 0 & 212 & 171 & 203 \\ 212 & 0 & 186 & 247 \\ 171 & 186 & 0 & 139 \\ 203 & 247 & 139 & 0 \end{pmatrix}$$

$$\mathrm{cost}(1,2,3,4) = 740$$
$$\mathrm{cost}(1,2,4,3) = 769$$
$$\mathrm{cost}(1,3,2,4) = 807$$

Wenn n etwas größer wird, dann läßt sich der naive Ansatz, der im Ausprobieren aller Permutationen besteht, nicht mehr durchführen:

$$22! = 1124000727777607680000.$$

Ein heutiger Rechner kann die Kosten von 100000 Permutationen von 22 Städten in einer Sekunde auswerten (großzügig geschätzt). So ein Rechner wäre 356 Millionen Jahre beschäftigt, bis er alle möglichen Permutationen durchprobiert hat.

In Anwendungen müssen TSPs oder ähnliche Probleme dennoch gelöst werden. Im konkreten Beispiel heißt das, der Vertreter muß sich für eine Rundreise durch die 22 Städte entscheiden, auch wenn er etwas länger fährt, als mit einer optimalen Tour. Wir werden gegen Ende der Vorlesung Algorithmen kennenlernen, die in 'kurzer' Zeit 'gute' Touren berechnen. Aber auch für die Berechnung optimaler Touren hat man erstaunlich wirksame Methoden entwickelt. Letztes Jahr wurde die optimale Tour durch sämtliche 13509 Städte in den USA mit mehr als 500 Einwohnern bestimmt.

1.2 Graphen

Definition 1. *Ein* Graph $G = (V, E)$ *besteht aus einer Menge* $V = \{v_1, v_2, \ldots, v_n\}$ *von* Knoten *und einer Menge* $E \subseteq \binom{V}{2}$ *von* Kanten.*

Es ist oft nützlich, Graphen durch Zeichnungen zu veranschaulichen. Das folgende Beispiel deutet an, daß Zeichnungen auch irreleiten können.

Beispiel 2.

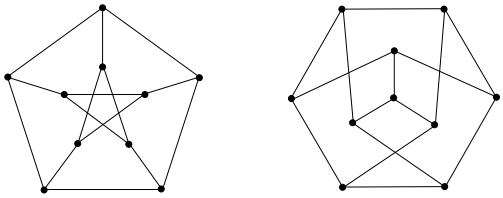


Abbildung 1. Zwei Zeichnungen des berühmten Petersen Graphen.

Einige spezielle Graphen

Ein Graph, in dem jedes Paar von Knoten durch eine Kante verbunden ist, heißt *vollständiger Graph*. Den vollständigen Graphen auf n Knoten bezeichnen wir mit K_n. Der vollständige Graph hat $\binom{n}{2} = \frac{1}{2}n(n-1)$ Kanten.

Der *vollständig bipartite Graph* $K_{m,n}$ hat die Knotenmenge $V = M \cup N$ mit $|M| = m$, $|N| = n$, und alle möglichen Kanten zwischen Knoten in M und N, d.h. $E = M \times N$. Dieser Graph hat $m + n$ Knoten und $m \cdot n$ Kanten.

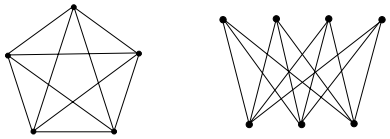


Abbildung 2. Die Graphen K_5, $K_{4,3}$.

Skelettgraphen von Polytopen haben die Ecken eines Polytops als Knoten und die Kanten als Kanten. Besonders interessante Beispiele für Skelettgraphen sind die Graphen der Platonischen Körper: Tetraeder, Würfel, Oktaeder, Dodekaeder und Ikosaeder.

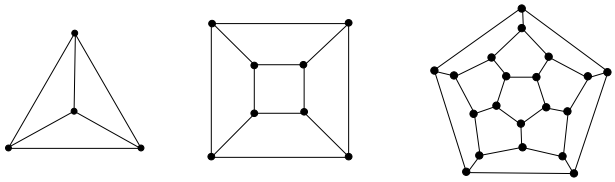


Abbildung 3. Die Skelettgraphen von Tetraeder, Würfel und Dodekaeder.

*Die Wahl der Buchstaben V und E erklärt sich durch die englischen Begriffe *vertex* und *edge*.

Der n-dimensionale *Hyperwürfel* Q_n hat als Knoten alle möglichen 0-1 Vektoren der Länge n, das sind 2^n Stück. Zwei Knoten sind durch ein Kante verbunden, wenn sie sich in genau einer Koordinate unterscheiden. Der gewöhnliche Würfel ist Q_3, siehe Abbildung 4.

Definition 2. *Der* Grad $d(v)$ *eines Knotens* $v \in V$ *in* $G = (V, E)$ *ist die Anzahl der Kanten, die* v *enthalten (man sagt auch: die zu* v inzident *sind).*

Satz 1. *Für jeden Graphen* $G = (V, E)$ *gilt:* $\sum_{v \in V} d(v) = 2\,|E|$.

Beweis. Man wendet das *Prinzip des doppelten Abzählens* auf die Paare (v, e) mit $v \in e$ an. Zählt man nach Knoten geordet, so erhält man $\sum d(v)$. Zählt man nach Kanten geordnet, erhält man $2\,|E|$. □

Folgerung 2. *Jeder Graph hat eine gerade Anzahl Knoten ungeraden Grades.*

Die *Inzidenzmatrix* $M(G)$ macht das Prinzip des doppelten Abzählens anschaulich. Man zählt die Einsen der Matrix einmal Zeilen- und einmal Spaltenweise.

Beispiel 3. $G = (V, E)$ mit $V = \{v_1, v_2, v_3, v_4, v_5\}$ und Kanten $e_1 = \{v_1, v_2\}$, $e_2 = \{v_1, v_5\}$, $e_3 = \{v_2, v_3\}$, $e_4 = \{v_2, v_4\}$, $e_5 = \{v_3, v_4\}$, $e_6 = \{v_3, v_5\}$, $e_7 = \{v_4, v_5\}$.

$$M(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Ein Graph heißt r-*regulär*, wenn alle Knoten den gleichen Grad r haben. Vollständige Graphen, Hyperwürfel und die Skelettgraphen der Platonischen Körper sind regulär.

Übung 1.

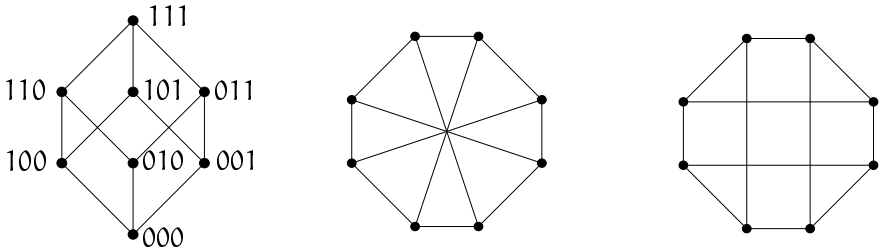


Abbildung 4. Drei 3-reguläre Graphen mit 8 Knoten.

Bestimme vier verschiedene 3-reguläre Graphen mit 8 Knoten.

Übung 2. Verwende doppeltes Abzählen um zu zeigen: Es ist unmöglich, die Zahlen 1 bis 12 so auf die Würfelkanten zu verteilen, daß die Summe der Zahlen, die bei mit v inzidenten Kanten stehen, für jeden Knoten gleich ist.

1.3 Wege und Bäume

Definition 3. *Ein* Weg *in* G *ist eine Folge* $v_0v_1 \dots v_l$ *von Knoten mit* $v_{i-1}v_i \in E$ *für* $i = 1, .., l$. *Der Weg führt von* v_0 *nach* v_l, *seine* Länge *ist* l. *Sind alle* v_i *verschieden, so sprechen wir von einem* einfachen Weg *oder auch einem* Pfad. *Wenn* $v_0 = v_l$, *dann ist der Weg ein* geschlossener Weg. *Sind alle übrigen Knoten verschieden, dann ist er ein* Kreis.

Definition 4. G *heißt* zusammenhängend, *wenn je zwei Knoten durch einen Weg verbunden sind. Eine Kante* e *in einem zusammenhängenden Graphen* $G = (V, E)$ *heißt* Brücke, *wenn* $G' = (V, E \backslash e)$ *nicht zusammenhängend ist.*

Beispiel 4.

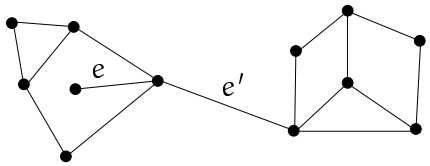


Abbildung 5. Ein Graph mit Brücken e und e'.

In Lemma 3 bis 7 sind einige Aussagen über Pfade, Wege, Brücken und Bäume zusammengestellt. Beweise sind im Anhang ausgearbeitet.

Lemma 3. *In einem zusammenhängenden Graphen sind je zwei Knoten durch einen Pfad verbunden.*

Lemma 4. *Jeder geschlossene Weg ungerader Länge enthält einen ungeraden Kreis.*

Lemma 5. *Sei* G *zusammenhängend. Eine Kante* e *ist eine Brücke von* G *genau dann wenn* e *in keinem Kreis von* G *vorkommt.*

Definition 5. *Ein Graph* G, *der keine Kreise besitzt, heißt* Wald. *Ein zusammenhängender Wald ist ein* Baum.

Beispiel 5.

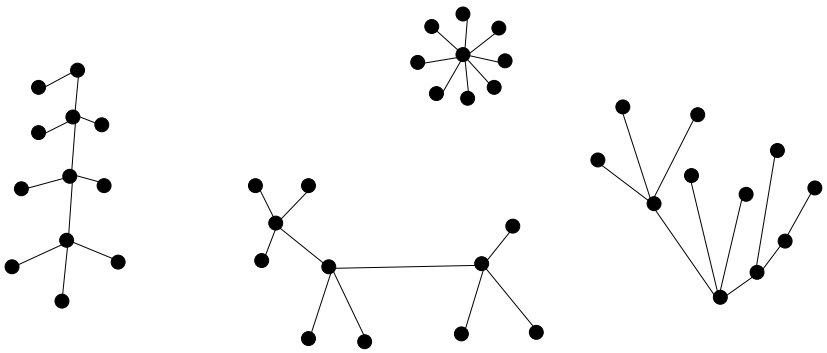


Abbildung 6. Ein aus vier Bäumen bestehender Wald.

Lemma 6. *Jeder Baum mit zumindest zwei Knoten besitzt einen Knoten vom Grad* 1.

Lemma 7. *Sei* G *zusammenhängend.* G *ist ein Baum genau dann wenn jede Kante von* G *eine Brücke ist.*

Satz 8. *Für* $G = (V, E)$ *sind folgende Aussagen äquivalent:*

(1) G *ist kreisfrei und zusammenhängend (d.h.* G *ist ein Baum).*

(2) G *ist kreisfrei und* $|E| = |V| - 1$.

(3) G *ist zusammenhängend und* $|E| = |V| - 1$.

Beweis. Wir zeigen, daß je zwei der drei Eigenschaften kreisfrei, zusammenhängend und $|E| = |V|-1$ die dritte zur Folge haben. Daraus folgt die behauptete Äquivalenz unmittelbar.

(Zusammenhängend und kreisfrei $\Longrightarrow |V| - 1$ Kanten)

Sei v ein Knoten von Grad 1 in G (existiert wegen Lemma 6). Entfernen wir v aus G, so bleibt ein Baum $G' = (V', E')$. Offenbar gilt $|V'| - |E'| = |V| - |E|$. Wir entfernen weiter Knoten vom Grad 1, bis nur noch eine Kante übrig ist, also ist $|V| - |E| = 1$.

(Zusammenhängend und $|V| - 1$ Kanten $\Longrightarrow$ kreisfrei)

Entferne Kanten aus G, die in Kreisen vorkommen, bis der erzeugte Graph G' kreisfrei ist. Da wir keine Brücke entfernt haben (Lemma 5), ist G' zusammenhängend. Wegen Teil 1 hat G' genau $|V| - 1$ Kanten. Da G und G' also gleich viele Kanten haben, gilt $G = G'$. Da G' kreisfrei ist, gilt dasselbe natürlich für G.

(Kreisfrei und $|V| - 1$ Kanten $\Longrightarrow$ zusammenhängend)

Seien $G_1, \ldots, G_k$ die Zusammenhangskomponenten von G. Wegen Teil 1 hat jedes G_i genau $|V(G_i)| - 1$ Kanten, zusammen haben sie $|V| - k$ Kanten. Also ist $k = 1$, und G ist zusammenhängend. □

Übung 3. Seien $d_1 \ldots d_n$ natürliche Zahlen. Zeige, daß ein Baum mit Graden $d_1 \ldots d_n$ genau dann existiert, wenn $\sum d_i = 2n - 2$.
Folgerung: Ein Baum mit zumindest zwei Knoten besitzt mindestens zwei Knoten vom Grad 1.

Übung 4. Sei S eine n elementige Menge, $A_1, A_2, \ldots, A_n$ seien verschiedene Teilmengen von S. Zeige, daß es ein Element $x \in S$ gibt, so daß $A_1 \cup \{x\}, A_2 \cup \{x\}, \ldots, A_n \cup \{x\}$ verschieden sind.

Hinweis: Betrachte den Graphen mit Knoten $a_1, a_2, \ldots, a_n$ und einer Kante $a_i \leftrightarrow a_j$, wenn sich A_i und A_j nur in einem Element unterscheiden. Markiere die Kante $a_i \leftrightarrow a_j$ mit $x \in S$, wenn sich A_i und A_j nur in x unterscheiden. Zeige, daß nicht alle Elemente von S als Markierung auftreten können. Untersuche dazu eine Kantenmenge, in der zu jeder auftretenden Markierung genau eine Kante vorkommt.

2 Euler–Wege und Euler–Kreise

In diesem Abschnitt beschäftigen wir uns mit Wegen, die jede Kante eines gegebenen Graphen genau einmal benutzen. Solche Wege heißen *Euler–Wege*. Der Name geht auf Eulers Lösung des Königsberger Brückenproblems zurück. In einem populären Spielchen "Haus vom Nikolaus" ist ein Euler–Weg in einem speziellen Graphen gesucht.

Beispiel 6.

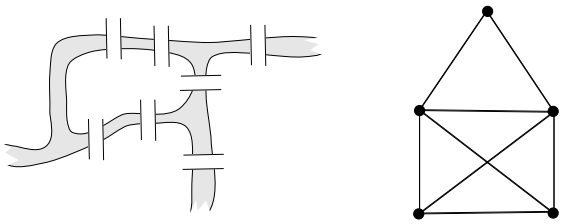


Abbildung 7. Zwei berühmte Euler–Probleme.

Beobachtung. Wenn G einen geschlossenen Euler–Weg (kurz *Euler–Kreis** genannt) besitzt, dann sind alle Knotengrade gerade. Hat G einen offenen Euler–Weg, dann besitzt G genau zwei Knoten ungeraden Grades.

Folgerung. Das Königsberger Brückenproblem ist nicht lösbar.

Satz 9. *Ein Graph* G *besitzt einen Euler–Kreis* $\Longleftrightarrow$ G *ist zusammenhängend, und alle Grade sind gerade.*

Beweisidee: Es bleibt nur "$\Leftarrow$" zu zeigen. Da alle Grade ≥ 2 sind, können wir bei Start in v_0 immer auf neuen Kanten weitergehen, bis wir wieder in v_0 ankommen. Sei C der so erzeugte geschlossene Weg. Sei $G' = G \backslash C$ und seien $G_1 \dots G_z$ die zusammenhängenden Komponenten von G'. Jedes G_i ist zusammenhängend, und alle Grade sind gerade. Durch Induktion nach der Kantenzahl schließen wir, daß jedes G_i einen Euler–Kreis besitzt. Alle diese Kreise können mit C verknüpft werden. So erhält man einen Euler–Kreis für G. □

Folgerung. Ein Graph G enthält einen Euler–Weg $\Longleftrightarrow$ G ist zusammenhängend und hat genau 2 Knoten ungeraden Grades.

Beweis. Wenn G einen Euler–Weg enthält, dann muß G sicherlich zusammenhängend sein und genau zwei Knoten ungeraden Grades besitzen. Seien u, v die beiden Knoten ungeraden Grades. Betrachte $G' = G \cup \{uv\}$. Wegen Satz 9 besitzt G' einen Euler–Kreis C. Nach Entfernen von uv aus C wird daraus ein Euler–Weg in G. □

*Dieser Name stimmt zwar nicht mit unserer Definition 3 überein, aber er ist Tradition!

2.1 Ein Algorithmus für Euler–Kreise

Zwar ist es möglich, den Beweis von Satz 9 zu einem Algorithmus für Euler–Kreise umzuformulieren. Der folgende Algorithmus von Fleury ist jedoch interessanter:

- Wähle beliebigen Startknoten v_0.
- Sind $v_0, v_1, \ldots, v_{i-1}$ schon gegeben, so wähle als v_i einen Nachbarn von v_{i-1} in $G_i = G \backslash \{e_1 \ldots e_{i-1}\}$, so daß (wenn möglich) $e_i = v_{i-1}v_i$ keine Brücke in G_i ist.

Satz 10. *Ist* G *eulersch, dann berechnet der Algorithmus von Fleury einen Euler–Kreis.*

2.2 Chinese Postman Problem: Eine Anwendung von Euler–Wegen

Ein Postbote muß, ausgehend vom Postamt, alle Straßen seines Bezirkes durchlaufen und wieder zum Postamt kommen. Falls der Graph (Plan des Bezirks) keinen Euler–Kreis besitzt, muß er Kanten mehrfach durchlaufen. Das Ziel des Postboten ist es, eine möglichst kurze Strecke zu laufen.

Wenn alle Knoten gerade sind, dann ist G eulersch, und die Länge des Weges ist die Summe der Kantenlängen. Gesucht ist nun eine längenminimale Kantenmenge $F \subset E$, so daß die Vereinigung der Kantenmengen $G+F$ eulersch ist. (Zwar hat $G+F$ doppelte Kanten – ist ein *Multigraph* –, aber Satz 9 gilt auch für Multigraphen).

Das folgende Verfahren löst das Problem

(1) Bestimme die Menge U der Knoten ungeraden Grades, $|U| = 2m$.

(2) Für je zwei Knoten $u, v \in U$ berechne die Länge $\delta(u, v)$ eines kürzesten Weges von u nach v (*kürzeste Wege Problem*).

(3) Ein *Matching* auf U ist eine Menge M von m Paaren aus U, die jedes $u \in U$ genau einmal erfaßt. Bestimme ein längenminimales Matching, d.h. ein M mit $\delta(M)$ minimal, wobei
$$\delta(M) = \sum_{\{u,v\} \in M} \delta(u, v)$$

(4) Ist M_0 ein minimales Matching, so verdoppeln wir für alle $\{u, v\} \in M$ einen kürzesten Weg von u nach v.

(5) Berechne einen Euler–Kreis in dem so entstandenen eulerschen Multigraphen.

Drei der fünf Teilprobleme dieser Lösung sind algorithmisch interessant. Wichtig ist, daß sich alle Teilprobleme effizient (d.h. auch für große Eingaben, z.B. $|V| = 100000$, in vertretbarer Zeit) lösen lassen.

Beispiel 7. Wendet man den Algorithmus für das Chinese Postman Problem auf den in der Abbildung dargestellten Graphen an, so erhält man eine Tour der Länge $\delta(G)+10 = 62$.

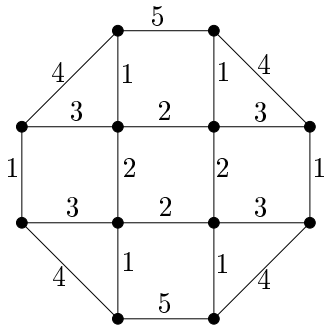


Abbildung 8. Ein Graph mit längengewichteten Kanten.

Übung 5. Sei G ein eulerscher Graph mit maximalem Grad $\Delta > 2$. Zeige: Die Anzahl der Euler–Kreise von G ist gerade.

Übung 6. Beweise oder widerlege: Jeder Graph mit einer geraden Anzahl von Knoten und einem Euler–Kreis hat eine gerade Anzahl von Kanten.

3 Hamilton–Kreise

In diesem Abschnitt wollen wir Wege und Kreise in Graphen betrachten, die jeden Knoten genau einmal besuchen. Solche Pfade und Kreise heißen *hamiltonsch*. Ein Graph heißt *hamiltonscher Graph*, wenn er einen Hamilton–Kreis besitzt.

- K_n ist hamiltonsch für $n \geq 3$.
- $K_{m,n}$ ist nur für $n = m \geq 2$ hamiltonsch.

Es ist keine vollständige Charakterisierung von hamiltonschen Graphen bekannt. Daher sucht man nach notwendigen und hinreichenden Bedingungen.

Beispiel 8. Sir Hamilton beschrieb 1856 ein Spiel, bei dem es darum ging, einen Weg $v_1 \ldots v_5$ im Dodekaedergraphen zu einer Rundreise zu ergänzen.

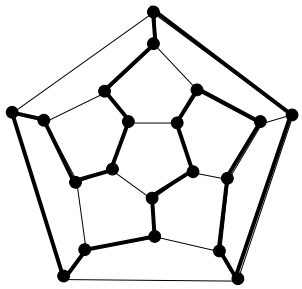


Abbildung 9. Ein Hamilton–Kreis im Dodekaedergraphen.

Satz 11 (Notwendige Bedingung). *Ein hamiltonscher Graph* $G = (V, E)$ *erfüllt für alle nichtleeren Knotenmengen* $A \subseteq V$*:*

$$\textit{Anzahl Komponenten von } G - A \leq |A|.$$

Dabei ist $G - A$ *der Graph, der mach dem Entfernen der Knoten aus* A *und aller zu ihren inzidenten Kanten verbleibt.*

Beweis. Wir geben dem Kreis eine Orientierung und betrachten Kanten $x \to y$ mit $x \notin A$ und $y \in A$. Jede Komponente von $G - A$ wird über so eine Kante verlassen, und die Zielknoten dieser Kanten in A sind alle verschieden. □

Beispiel 9. Die Abbildung zeigt einen Graphen (den kleinsten polyedrischen), der keinen Hamilton–Kreis, aber einen Hamilton–Weg besitzt. (Die Menge der grauen Knoten verletzt die Bedingung aus dem Satz).

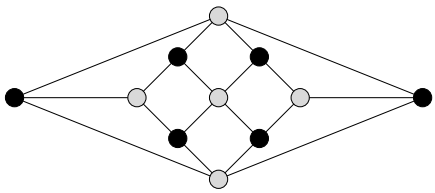


Abbildung 10. Ein nicht hamiltonscher Graph.

Der Petersen Graph (Abbildung 1) zeigt, daß die in Satz 11 gegebene Bedingung nicht hinreichend ist. Jede Teilmenge A seiner Knoten erfüllt die Bedingung, aber der Graph ist dennoch nicht hamiltonsch.

Satz 12 (Hinreichende Bedingung). *Ist* $G = (V, E)$ *ein Graph mit*

$$d(u) + d(v) \geq |V| \quad \textit{für alle Paare } u, v \textit{ mit } uv \notin E$$

dann ist G *hamiltonsch.*

Beweis. (indirekt) Unter allen Gegenbeispielen mit n Knoten sei G eines mit einer maximalen Anzahl von Kanten. Wir wissen $G \neq K_n$, da K_n hamiltonsch ist. Wegen der Wahl von G ist $G + uv$ hamiltonsch für alle Paare u, v mit $uv \notin E$. Also besitzt G einen hamiltonschen Weg $u = v_1 \ldots v_n = v$, der in u beginnt und in v endet. Wenn ein Nachbar von v unmittelbar vor einem Nachbarn von u liegt, d.h. uv_{i+1} und $v_i v$ sind beides Kanten, dann ist $v_1 \ldots v_i v_n v_{n-1} \ldots v_{i+1} v_1$ ein Hamilton–Kreis für G.

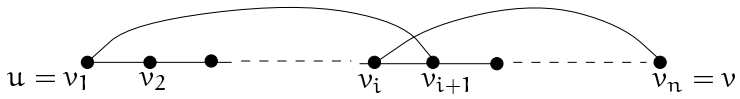


Abbildung 11. Der Austauschschritt.

Wir zeigen nun, daß so ein Paar (v_i, v_{i+1}) existiert. Sei $S = \{i : uv_{i+1} \in E\}$ und $T = \{i : v_i v \in E\}$. Es gilt

$$|S \cup T| + |S \cap T| = |S| + |T| = d(u) + d(v) \geq n$$

Da weder S noch T den Index n enthalten, muß $|S \cup T| < n$ und also $|S \cap T| \geq 1$ sein. Damit findet man so ein Paar konstruiert einen Hamilton–Kreis in G. Das ist im Widerspruch zu den Annahmen über G. Also kann es kein Gegenbeispiel geben. □

Übung 7. Betrachte einen $3 \times 3 \times 3$ Würfel aus 27 Käsestückchen. Eine Maus sucht einen Weg von einem Eckstück über alle anderen Stücke, der im Mittelstück endet. Ist dies möglich? Gleiche Frage für $5 \times 5 \times 5$ und allgemeines ungerades n.

Übung 8. Ein Graph heißt gerichtet oder orientiert, wenn jede Kante uv mit einer der beiden Orientierungen $u \to v$ oder $v \to u$ versehen ist. Eine Orientierung eines vollständigen Graphen heißt *Turnier**. Zeige: Jedes Turnier besitzt einen gerichteten Hamilton–Pfad, das ist ein Hamilton–Pfad, in dem alle Kanten in Pfeilrichtung durchlaufen werden.

4 Kombinatorische Optimierung

4.1 Optimierung und Algorithmen

Ein Optimierungsproblem ist gegeben durch eine Menge X und eine Funktion $f : X \to \mathbb{R}$. Gesucht ist eine Lösung $x \in X$, die $f(x)$, maximiert oder minimiert. In kombinatorischen Optimierungsproblemen ist X eine kombinatorische Menge.
Beispiele für kombinatorische X:

- Menge der Hamilton Kreise in G (TSP).
- Menge der Pfade von u nach v in G (kürzeste Wege Problem).
- Gegeben n Pakete und ein Rucksack. $X = \{$ Menge von Paketen, die im Rucksack Platz haben $\}$ (Rucksackproblem).

In der kombinatorischen Optimierung beschäftigt man sich mit Methoden (Algorithmen) zur Lösung solcher Probleme. Weitere Beispiele kombinatorischer Optimierungs–Probleme haben wir schon im Kapitel über Euler–Wege gesehen: Das Chinese Postman Problem oder das Matching–Problem. Ein 'guter' Algorithmus für ein kombinatorisches Optimierungsproblem hat einige Forderungen zu erfüllen:

(1) Er *muß korrekt* sein (d.h. die Ausgabe x muß in X liegen)

(2) Er *soll optimieren* (d.h. $f(x)$ soll maximal oder minimal sein)

(3) Er *muß effizient* sein (d.h. man muß den Algorithmus bei vernünftigen Problemgrößen auch tatsächlich einsetzen können).

---

*Ein Turnier-Graph stellt ein mögliches Ergebnis von einem Wettkampf dar, in dem je zwei Teams einmal gegeneinander spielen und jedes Spiel einen Sieger hat.

Eine mathematische Präzisierung der Forderung nach Effizienz sieht so aus: Wir sagen, ein Problem besitzt einen effizienten Lösungsalgorithmus, wenn die Laufzeit als Funktion der Eingabegröße n durch ein Polynom beschränkt ist, also z.B. $n, n^2, 8n^3 - n$, aber *nicht* $2^n, n!$.

Wenn sich die berühmte $P \neq NP$ Vermutung der Komplexitätstheorie als richtig erweist, dann kann es für viele wichtige Probleme (wie z.B. das TSP) keinen Algorithmus geben, der alle drei Forderungen erfüllt. Da auf Effizienz nicht verzichtet werden kann, schwächt man Forderung (2) ab, und verlangt nur noch eine *Approximation* des Optimums. Wir sagen, x ist eine α-*approximierende Lösung* eines Minimierungssproblems, wenn

$$f(x) \leq \alpha \cdot f(x_{opt}).$$

Der Faktor α beschreibt, um wieviel die Kosten $f(x)$ der gefundenen Lösung x die minimalen Kosten $f(x_{opt})$ übersteigen können. Gesucht werden effiziente Algorithmen, die einen möglichst kleinen Approximationsfaktor $\alpha > 1$ gewährleisten.

Ist für einen Algorithmus keine Approximationsgarantie bekannt, aber die Intuition sagt, daß die Lösung brauchbar ist (d.h. nicht zu schlecht sein wird), so spricht man von einer *Heuristik*.

Bevor wir uns mit Heuristiken und approximierenden Algorithmen für das TSP befassen, wollen wir ein paar Beispiele für Algorithmen sammeln.

- **Euklidischer Algorithmus**
  Gegeben: $a_1, a_2 \in \mathbb{Z}$ Gesucht: $ggt(a_1, a_2)$
- **Weitere zahlentheoretische Probleme**
  Polynom-Multiplikation/Division; Faktorisierung.
- **Such- und Sortierprobleme**
  Gegeben: $A = \{a_1, a_2 \ldots, a_n\}$ aus total geordneter Grundmenge.

  Sortierproblem. Gesucht: Eine Umsortierung $a_1^*, a_2^* \ldots a_n^*$ von A, so daß $a_1^* < a_2^* < \ldots < a_n^*$

  Median. Gesucht: Ein Element $x \in A$, so daß (fast) gleichviele Elemente aus A kleiner/größer als x sind.

  Maximum. Gesucht: Ein $x \in A$, so daß alle Elemente aus A kleiner x sind.

Ein Algorithmus für *Maximum*

```
x ← a_1
for i = 2 to n
    x ← max(x, a_i)
return x
```

Dieses Verfahren benötigt $n - 1$ Vergleiche. Kann das Maximum x mit weniger als $n - 1$ Vergleichen ermittelt werden? Nein, um x einwandfrei festzulegen, muß es für jedes $a \neq x$ ein b geben, so daß a und b verglichen wurden und $b > a$.

4.2 Optimierungs–Probleme auf Graphen

Kürzeste Wege.

Gegeben: $G = (V, E)$, eine Längenfunktion $\omega : E \to \mathbb{R}^+$ und zwei Knoten $u, v \in V$.
Gesucht: Ein kürzester Weg von u nach v.

Matching.

Gegeben: K_{2n} und Kostenfunktion $\omega : E \to \mathbb{R}^+$.
Gesucht: Billigstes vollständiges Matching, d.h. billigste Kantenmenge $M \subset E$, in der jeder Knoten Grad 1 hat.

Minimum-Spanning-Tree (MST).

Gegeben: $G = (V, E)$ zusammenhängend mit Kostenfunktion $\omega : E \to \mathbb{R}^+$.
Gesucht: Billigster zusammenhängender Subgraph $T = (V, F)$, d.h. $F \subseteq E$ und

$$\sum_{e \in F} \omega(e) \leq \sum_{e \in F'} \omega(e)$$

für alle zusammenhängenden Subgraphen $T' = (V, F')$ von G.
Beobachtung: T ist ein Baum.

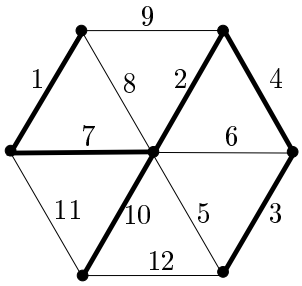


Abbildung 12. Ein Graph mit einem MST.

4.3 MST und der Greedy-Algorithmus

Mit dem Namen *Greedy** *–Algorithmus* bezeichnet man Algorithmen, die Entscheidungen so treffen, daß der augenblickliche Gewinn möglichst groß ist. Greedy-Algorithmen können für gewisse Probleme (z.B. TSP) beliebig schlechte Ergebnisse liefern. In anderen Fällen sind sie, wie wir gleich sehen werden, sogar optimal. Ein Greedy–Algorithmus für MST geht nach dem folgenden Prinzip vor:

Nimm immer die billigste Kante, die möglich ist, d.h. die keinen Kreis erzeugt.

Der Algorithmus läßt sich in einer an Programmiersprachen angelehnten Notation so aufschreiben:

* *greedy* = gierig, gefräßig

Sortiere E so, daß
$\omega(e_1) \leq \omega(e_2) \leq \ldots \leq \omega(e_m)$
$F \leftarrow \emptyset$
for $i = 1$ to m
    if $F \cup e_i$ kreisfrei, then
        $F \leftarrow F \cup e_i$
return F

Satz 13. *Der Algorithmus berechnet einen MST* $T = (V, F)$ *in jedem zusammenhängenden, gewichteten Graphen* $G = (V, E)$.

Beweis. Nach Konstruktion ist T kreisfrei. Da es in G Kanten gibt, die $U \subset V$ mit $V \setminus U$ verbinden, und die billigste dieser Kanten in T aufgenommen wurde, ist T auch zusammenhängend. Also ist T ein aufspannender Baum für G.

Sei T^* ein MST für G, der möglichst viele Kanten mit T gemeinsam hat. Angenommen, $T \neq T^*$. Sei $e = e_i$ die erste Kante, die in T, aber nicht in T^* ist.

Behauptung: Für alle $j < i$ gilt $e_j \in T^* \iff e_j \in T$. Aus der Wahl von e_i folgt $e_j \in T \Rightarrow e_j \in T^*$. Die Implikation $e_j \notin T \Rightarrow e_j \notin T^*$ folgt aus der Konstruktion; jede Kante $e_j \notin T$, $j < i$ bildet einen Kreis mit Kanten die sowohl in T als auch in T^* sind.

Betrachte $T^* + e$, dieser Graph enthält einen Kreis und dieser Kreis enthält eine Kante $e' = e_j$, die nicht in T ist. Aus obiger Behauptung folgt $i < j$, also $\omega(e_i) \leq \omega(e_j)$. Nun ist $T^* + e - e'$ ein Baum, dessen Gewicht um $\omega(e_j) - \omega(e_i)$ kleiner ist als das von T^*. Also ist $T^* + e - e'$ ebenfalls ein MST, hat aber mehr Kanten mit T gemeinsam als T^*. Widerspruch. □

Übung 9. Ein *Minimax Spanning Tree* ist ein aufspannender Baum T in einem zusammenhängenden kantengewichteten Graphen, dessen schwerste Kante so leicht wie möglich ist. Finde einen Algorithmus für das Minimax Spanning Tree Problem.

Übung 10. Sechs Personen wissen jeder eine Neuigkeit. Sie führen eine Reihe von Telefonaten durch und bei jedem Gespräch erzählen sich die Teilnehmer alle Neuigkeiten, die sie bis dahin erfahren haben. Wie viele Telefonate sind nötig, bis alle alles wissen.

5 Heuristiken und Approximationen für TSP

Zur Erinnerung: Ein TSP Problem ist durch eine $n \times n$ Kostenmatrix $C = (c_{ij})$ gegeben. Ist C symmetrisch (d.h. $c_{ij} = c_{ji}$), dann läßt sich das TSP Problem auffassen als Problem, einen gewichtsminimalen Hamilton-Kreis im vollständigen Graphen K_n mit Kantengewichten $c(v_i v_j) = c_{ij}$ zu bestimmen.

5.1 Greedy-Methoden

Nächster Nachbar (NN)
Starte in einem beliebigem Knoten $s \in V$. Verlängere den Weg schrittweise um die billigste Kante zwischen dem Endknoten des Weges und einem noch nicht besuchten Nachbarn.

Beispiel 10. Sei $c_{ii+1} = 0$ für $i = 1, \ldots, n-1$, $c_{1n} = B > 2$ und $c_{ij} = 1$ für alle übrigen Paare i, j. Bei Start in v_1 findet NN eine Tour mit Kosten B, während eine optimale Tour nur 2 kostet.

Billigstes Einfügen (NI Nearest Insertion)
Starte in beliebigem Knoten $s \in V$. Vergrößere den Kreis K schrittweise: Finde $u \notin K$ und $v \in K$ so daß $c(u,v)$ minimal ist. Füge v nach u in den Kreis ein, d.h. der neue Kreis ist $K + vu + uv' - vv'$ (siehe Abbildung).

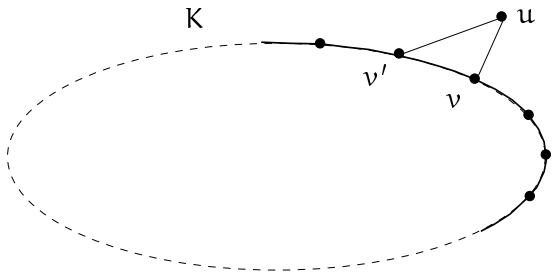


Abbildung 13. Kreis K wird bei v um neuen Knoten u verlängert.

Im allgemeinen ist NI besser als NN. Aber im obigen Beispiel erzeugt NI bei Start in v_1 dieselbe schlechte Lösung.

Satz 14 (Sahni-Gonzales 1976). *Wenn $P \neq NP$, dann gibt es kein α, zu dem ein effizienter, α-approximierender Algorithmus für das TSP Problem existiert.*

5.2 Euklidisches TSP

Eine Kostenmatrix C heißt d-dimensional realisierbar, wenn es Punkte $p_1, p_2, \ldots, p_n$ im $\mathbb{R}^d$ gibt, so daß $\mathrm{dist}(p_i, p_j) = c_{ij}$, wobei dist der euklidische Abstand $\|p_i - p_j\|$ ist, d.h. die Länge der Strecke mit Endpunkten p_i und p_j. Das Problem, mit dem sich der Handlungsreisende aus der ersten Vorlesung quält, ist 2-dimensional. Können wir solche Probleme leichter lösen?

Satz 15 (Arora 1996). *Für jedes $\epsilon > 0$ gibt es effiziente $(1+\epsilon)$-approximierende Algorithmen für euklidisches TSP in jeder Dimension.*

Wir wollen uns hier mit einer Eigenschaft befassen, die weniger restriktiv ist als "euklidisch". Eine Kostenmatrix erfüllt die Dreiecksungleichung, wenn für alle i, j, k gilt

$$c_{ik} \leq c_{ij} + c_{jk}.$$

Satz 16. *Billigstes Einfügen (NI) ist ein 2-approximierender Algorithmus für TSP Probleme, die die Dreiecksungleichung erfüllen.*

Die wesentliche Beobachtung zum Beweis dieses Satzes ist

> Die Kanten (u,v) mit $u \notin K$, $v \in K$ und $c(u,v)$ minimal, die NI benutzt, sind genau die Kanten, die von einem der Greedy–Algorithmen für MST (Prims Algorithmus) in den Baum aufgenommen werden.

Ein Hamilton–Kreis H, aus dem wir eine Kante löschen, bleibt ein aufspannender Weg, also ein spezieller aufspannender Baum. Daraus folgt für die optimalen Kosten von MST und TSP

$$\mathsf{cost}(\mathsf{MST}) \leq \mathsf{cost}(\mathsf{TSP})$$

Nun beweisen wir $\mathsf{cost}(\text{NI-Tour}) \leq 2 \cdot \mathsf{cost}(\mathsf{MST})$ und damit auch die behauptete 2-Approximation von NI durch Induktion: Der Induktionsanfang fuer $n = 3$ ist klar (Dreiecksungleichung). Nun setzen wir für die Knotenmenge $V_1 = V(K)$ voraus:

$$\mathsf{cost}(K) \leq 2 \cdot \mathsf{cost}(\mathsf{MST}(V_1))$$

Sei K^* der Kreis der durch den Einbau von u in K entsteht und $V_2 = V_1 \cup \{u\}$. Nun gilt

$$\begin{aligned} \mathsf{cost}(K^*) &= \mathsf{cost}(K) + c(v,u) + c(u,v') - c(v,v') \\ &\leq \mathsf{cost}(K) + 2c(v,u) \\ &\leq 2\mathsf{cost}(\mathsf{MST}(V_1)) + 2c(v,u) = 2\mathsf{cost}(\mathsf{MST}(V_2)) \\ &\leq 2\mathsf{cost}(\mathsf{TSP}) \end{aligned}$$

□

Ähnlich wie im Beweis des vorigen Satzes überlegt man sich

Lemma 17. *Ist C eine Kostenmatrix, die die Dreiecksungleichung erfüllt, und H ein eulerscher aufspannender Subgraph von K_n, dann gilt:* $\mathsf{cost}(\mathsf{TSP}) \leq \mathsf{cost}(H)$.

Das folgende zweistufige Verfahren zur Konstruktion eines billigen eulerschen aufspannenden Subgraphen wurde von Christofides vorgeschlagen.

(1) Berechne einen MST T und die Menge U der Knoten ungeraden Grades von T.

(2) Berechne ein kostenminimales Matching M von U.

Betrachten wir nun die Kantenmenge $T + M$. Sie bildet einen aufspannenden eulerschen Subgraphen von G. Für die Kosten wissen wir $\mathsf{cost}(T + M) = \mathsf{cost}(T) + \mathsf{cost}(M) \leq \mathsf{cost}(\mathsf{TSP}) + \mathsf{cost}(M)$. Wir werden nun die Kosten für das Matching abschaetzen. Dazu vergleichen wir die Kosten für das Matching mit den Kosten einer optimalen TSP Tour K. Sei $u_1u_2 \ldots u_{2k}$ die Reihenfolge, in der die Knoten aus U in K auftreten. Wir betrachten zwei Matchings

$$M_1 = \{(u_1u_2), (u_3u_4), \ldots (u_{2k-1}, u_{2k})\} \quad \text{und} \quad M_2 = \{(u_2, u_3), (u_4, u_5) \ldots (u_{2k}, u_1)\}.$$

Natürlich gilt $\mathsf{cost}(M) \leq \mathsf{cost}(M_1)$ und $\mathsf{cost}(M) \leq \mathsf{cost}(M_2)$. Wegen der Dreiecksungleichung gilt außerdem $\mathsf{cost}(M_1) + \mathsf{cost}(M_2) \leq \mathsf{cost}(K) = \mathsf{cost}(\mathsf{TSP})$. Zusammengefaßt folgt also $\mathsf{cost}(M) \leq \frac{1}{2}\mathsf{cost}(\mathsf{TSP})$.

Das Gewicht des eulerschen Graphen $T+M$ ist also höchstens $\mathsf{cost}(\mathsf{TSP}) + \frac{1}{2}\mathsf{cost}(\mathsf{TSP})$. Mit Lemma 17 haben wir damit folgenden Satz bewiesen.

Satz 18 (Christofides 1976). *Es gibt einen effizienten 3/2-approximierenden Algorithmus für TSP mit Dreiecksungleichung.*

Bemerkung. Inzwischen ist bekannt, daß man TSP mit Dreiecksungleichung nicht beliebig genau approximieren kann (andernfalls ist $P = NP$). Die untere Schranke ist aber weit von der noch immer besten bekannten Schranke 3/2 für das Problem entfernt.

Übung 11. Wende Christofides Algorithmus auf die 9 Punkte aus Abbildung 14 an. Finde ein einfaches Argument dafür, daß das Ergebnis nicht optimal ist.

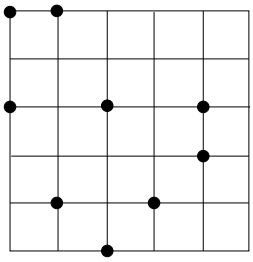


Abbildung 14. 9 ganzzahlige Punkte in $\mathbb{R}^2$.

Übung 12. Zeige: Einen Algorithmus der TSP mit Dreiecksungleichung optimal löst, kann man verwenden ,um beliebiges TSP ebenfalls optimal zu lösen. Hinweis: Verändere die Einträge der Kostenmatrix des allgemeinen Problems.

6 Untere Schranken und optimale Lösungen

Eine untere Schranke für die Kosten einer optimalen TSP Tour (mit Dreiecksgleichung) haben wir im vorigen Abschnitt schon kennengelernt. Die dort benutzte Ungleichung war:

$$\mathsf{cost}(\mathsf{MST}) \leq \mathsf{cost}(\mathsf{TSP}).$$

Eine etwas bessere untere Schranke liefern die sogenannten 1-Bäume: Ein 1-Baum ist ein aufspannender Baum auf den Knoten $\{v_2 \dots v_n\}$ ergänzt um zwei Kanten, die v_1 mit dem Baum verbinden. Da jede Tour ein 1-Baum ist, gilt für die optimalen Kosten M1T eines 1-Baumes:

$$\mathsf{cost}(\mathsf{M1T}) \leq \mathsf{cost}(\mathsf{TSP}).$$

Dies ist ein Beispiel für eine Einbettung von TSP in ein größeres Problem, dessen optimale Lösung effizient berechnet werden kann.

Die erfolgreichste Einbettung für TSP ist die sogenannte LP-Relaxation (LP steht für Lineare Programmierung). Wir beginnen damit, für jede Kante (i, j) eine Variable x_{ij} einzuführen.

Eine Tour T kann nun beschrieben werden, indem wir $x_{ij} = 1$ setzen, wenn (ij) in T vorkommt. So haben wir jeder Tour einen Punkt in einem $\binom{n}{2}$-dimensionalen Raum zugeordnet. Aber natürlich entspricht nicht jeder Punkt dieses Raumes einer Tour. Man kann zeigen, daß gültige Touren genau den Punkten entsprechen, die die folgenden drei Bedingungen erfüllen:

*Teile dieses Abschnitts sind aus *Die optimierte Odyssee* von Grötschel und Padberg übernommen, `www.spektrum.de/odyssee.html`

$$x_{ij} \in \{0, 1\} \tag{1}$$

$$\sum_{ij} x_{ij} = n \tag{2}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1 \text{ für alle } S \subset \{1 \ldots n\} \tag{3}$$

Damit kann das TSP formuliert werden als: Finde eine Belegung der x_{ij}, die den Bedingungen (1)-(3) genügt und die *Zielfunktion*

$$\mathsf{cost}(x) = \textstyle\sum_{i,j} c_{ij} x_{ij}$$

minimiert.

Leider ist damit noch nichts gewonnen, denn dieses ganzzahlige lineare Programm ist genauso schwer wie das TSP. Jetzt kommt der entscheidende Schritt. Wir 'relaxieren' Bedingung (1) und erlauben den x_{ij} beliebige Werte zwischen 0 und 1. Für die optimale Lösung x^* dieses Problems gilt

$$\mathsf{cost}(x^*) \leq \mathsf{cost}(\mathsf{TSP}).$$

Für die Suche nach dem Minimum einer linearen Funktion mit linearen Nebenbedingungen gibt es ein effizientes Verfahren, den Simplex-Algorithmus.

Am besten versteht man, was da passiert, wenn man sich ein geometrisches Bild macht, auch wenn unser Vorstellungsvermögen nur drei Dimensionen erfaßt. Nehmen wir alle Punkte (Vektoren) mit Koordinaten zwischen 0 und 1. Sie bilden einen Würfel der Kantenlänge 1 mit einer Ecke im Nullpunkt. Die Ecken dieses Einheitswürfels sind die Vektoren, deren Komponenten sämtlich gleich 0 oder 1 sind. Das gilt auch in d Dimensionen, nur hat der Einheitswürfel jetzt 2^d Ecken. Eine lineare Ungleichungs-Nebenbedingung ist wie ein Messer, das von dem Würfel (denken wir ihn uns aus Käse) ein Stück abschneidet. Es entsteht ein von ebenen Flächen begrenzter Körper.

Im d-dimensionalen Raum ist die Menge der zulässigen Punkte eines linearen Problems, das heißt derjenigen Punkte, die sämtliche Nebenbedingungen erfüllen, die Verallgemeinerung eines solchen Körpers: ein *Polytop*. Das Minimum einer linearen Zielfunktion auf einem Polytop wird stets in einer Ecke angenommen. Der Simplex-Algorithmus sucht, von einer Ecke des Polytops ausgehend, eine benachbarte Ecke mit kleineren Kosten. Er verfährt nach diesem Greedy-Prinzip, bis er nicht mehr weiterkommt. Die Ecke, an der er stehenbleibt, ist dann das gesuchte Minimum (oder eines von mehreren gleichberechtigten).

Leider handelt es sich nur um das Minimum des relaxierten Problems, nicht um das des ursprünglichen. Im allgemeinen hat also der Lösungsvektor, den der Simplex-Algorithmus liefert, keine ganzzahligen Koordinaten. Man muß demnach das Käsemesser nochmals ansetzen, um aus dem Polytop des linearen Problems das "echte" Polytop des ursprünglichen Problems zu machen; das ist das kleinste Polytop, das sämtliche Inzidenzvektoren von Touren enthält.

Das gelingt jedoch nicht mit einem Schnitt. Schlimmer noch: Wir wissen nicht einmal genau, wie wir schneiden müßten. Explizite lineare Programme für Rundreiseprobleme sind bislang für höchstens 9 Städte bekannt. Genauer gesagt: Das mit dem 9-Städte-TSP assoziierte Polytop hat 9 Gleichungen und 42104442 Ungleichungen mit 36 Variablen, entsprechend den 36 möglichen direkten Verbindungen zwischen 9 Punkten.

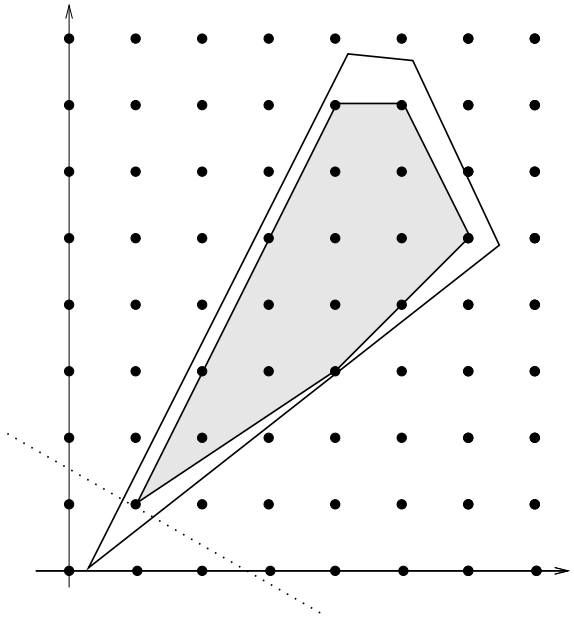
Ein weiteres Problem kommt hinzu. Man kann im allgemeinen noch nicht einmal mit dem Polytop der zulässigen Punkte des linearen Optimierungsproblems anfangen. Dazu müßte man nämlich sämtliche Nebenbedingungen berücksichtigen, und das sind für große Städtezahlen ebenfalls astronomisch viele (es gibt 2^n Ungleichungen vom Typ (3)). Gleichwohl gelingt es, weitaus größere Probleme mit Methoden der linearen Programmierung optimal zu lösen. Wie ist das möglich?

Wir ersetzen abermals das Problem durch ein größeres, einfacheres: Von den astronomisch vielen Nebenbedingungen verwenden wir nur eine sehr kleine Teilmenge (wir beschneiden den Käsewürfel nur sehr grob), finden einen Lösungsvektor, das heißt, eine optimale Ecke des zu großen Polytops, und bessern dann erst nach: Wir führen eine Nebenbedingung ein, die mit der bisher gefundenen Ecke gleich ein großes Stück Käse abschneidet.

Das oben erläuterte iterative Verfahren heißt Schnittebenenalgorithmus (cutting plane algorithm). Sind die Ungleichungen, die das echte (ganzzahlige) Polytop beschreiben, vollständig bekannt, muß der Schnittebenenalgorithmus nach einer endlichen Anzahl von Schritten beendet sein. Wir kennen gegenwärtig all diese Ungleichungen aber weder für das Travelling Salesman Problem noch für die meisten anderen kombinatorischen Optimierungsprobleme von praktischer Bedeutung. Der Schnittebenenalgorithmus kann also scheitern, bevor eine optimale Rundreise gefunden wurde.

Beispiel 11.

Zur Anschauung hier ein Beispiel in zwei Dimensionen: Ein Polytop P das durch vier Ungleichungen bestimmt ist (vier Kanten). Das darin enthaltene Polytop der ganzzahligen Punkte ist grau dargestellt. Die gepunktete Strecke s deutet die Kostenfunktion x_1+2x_2 an. Die Kosten aller Punkte auf jeder Geraden parallel zu s sind konstant. Die Ecke des ganzzahligen Polytops die von s getroffen wird ist die optimale Ecke. Das größere Polytop P enthält Punkte mit einem kleineren Zielfunktionswert.



Was sind unsere Wahlmöglichkeiten in einem solchen Fall? Wir könnten aufhören und uns mit der Kenntnis einer guten unteren Schranke begnügen. Wir könnten auch ein Enumerationsverfahren anwenden: Wenn der Schnittebenenalgorithmus unschlüssig (also ohne eine zulässige Lösung für unser eigentliches Problem) endet, gibt es Kanten, die weder mit 0 noch mit 1 belegt sind. Wir greifen eine solche Kante heraus, setzen ihren Wert willkürlich auf 1 ("die Tour muß diesen Weg verwenden") und versuchen das derart eingeschränkte - und deshalb einfachere - Problem zu lösen. Dasselbe versuchen wir mit

der Festlegung auf 0 statt 1 ("die Tour darf diesen Weg nicht verwenden"). Von den beiden Lösungen, die sich dabei ergeben, nehmen wir die bessere.

Natürlich kann der Lösungsversuch für diese Teilprobleme abermals unschlüssig enden. Wieder verzweigt der Lösungsprozeß in zwei Alternativen (branching), und es entsteht ein ganzer Baum von Lösungsmöglichkeiten. Das entsprechende Baumsuchverfahren ist unter dem Namen Branch and Cut bekannt. Im wesentlichen nach diesem Schema wurde auch das 13509 Städte-Problem gelöst.

Literatur

[1] M. Aigner, *Graphentheorie. Eine Entwicklung aus dem 4-Farben Problem*, Teubner, Stuttgart, 1984.

[2] ——, *Diskrete Mathematik*, Vieweg, Wiesbaden, 1993.

[3] N. L. Biggs, E. Lloyd, and R. J. Wilson, *Graph theory 1736-1936*, Clarendon Press; Oxford University Press, London, 1976.

[4] R. Diestel, *Graphentheorie*, Springer, Berlin, 1996.

[5] A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, Cambridge, 1985.

[6] M. Grötschel and L. Lovász, *Combinatorial optimization.*, in Handbook of Combinatorics, Vol. 2, Graham, Grötschel, and Lovász, eds., Elsevier, 1995, pp. 1541–1597.

[7] D. Jungnickel, *Graphen, Netzwerke und Algorithmen (3. Auflage)*, Spektrum Akademischer Verlag, Heidelberg, 1994.

[8] E. Lawler, J. Lenstra, A. Rinnooy Kan, and D. Shmoys, *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, John Wiley and Sons, Chichester, 1985.

[9] D. B. West, *Introduction to Graph Theory*, Prentice Hall, Upper Saddle River, NJ, 1996.

[10] R. J. Wilson, *Introduction to Graph Theory*, Longman, London, 1975.

Anhang

Beweise von Lemma 3-7

Lemma 3. *In einem zusammenhängenden Graphen sind je zwei Knoten durch einen Pfad verbunden.*

Beweis. Seien x, y beliebige Knoten von G, da G zusammenhängend ist gibt es einen Weg $W = (v_0, v_1, v_2, \ldots, v_k)$ der x und y verbindet, d.h. $v_0 = x$ und $v_k = y$. Wir müssen zeigen, daß es einen einfachen Weg (einen Pfad) gibt. Vorab eine Beobachtung: Wenn W einen Knoten doppelt besucht, also $v_i = v_j$ mit $0 \leq i < j \leq k$, dann ist $W' = (v_0, \ldots, v_i, v_{j+1}, \ldots, v_k)$ ein Weg von x nach y. Die Länge von W' ist kleiner als die von W, daher kann ein kürzester Weg von x nach y keinen Knoten doppelt besuchen und ist also ein Pfad. □

Lemma 4. *Jeder geschlossene Weg ungerader Länge enthält einen ungeraden Kreis.*

Beweis. Sei $W = (v_0, v_1, v_2, \ldots, v_k, v_0)$ ein geschlossener Weg ungerader Länge, d.h. $k+1$ ist ungerade. Wenn W kein Kreis ist, dann gibt es einen doppelt besuchten Knoten; $v_i = v_j$ mit $0 \leq i < j \leq k$. Nun sind auch $(v_i, v_{i+1}, \ldots, v_{j-1}, v_i)$ und $(v_0, v_1, \ldots, v_i, v_{j+1}, \ldots, v_k, v_0)$ geschlossene Wege. Die Längen dieser Wege sind $l_1 = j - i$ und $l_2 = k + 1 - j + i$. Wegen $l_1 + l_2 = k + 1$ muß eine der beiden Längen l_1, l_2 ungerade sein. Man wiederholt diese Überlegung mit dem ungeraden Weg. So kommt man zu einer Folge $W = W_0, W_1, W_2, \ldots$ von geschlossenen Wegen ungerader Länge, so daß W_{i+1} kürzer als W_i und in W_i enthalten ist. Da die Länge von W_0 endlich ist gibt es ein letztes Glied W_r dieser Folge. Dies ist der gesuchte einfache geschlossene Weg ungerader Länge. □

Lemma 5. *Sei G zusammenhängend. Eine Kante e ist keine Brücke von G genau dann, wenn e in einem Kreis von G vorkommt.*

Beweis. Sei $e = (u, v)$ eine Kante und $K = (u, v, v_2, \ldots, v_k, u)$ ein Kreis der e enthält. Seien x und y beliebige Knoten, wir zeigen, daß in $G' = (V, E \setminus e)$ ein Weg zwischen x und y existiert. Sei W ein Weg in G der x und y verbindet. Ersetzt man das jedes Paar v, u in W durch $v, v_2, \ldots, v_k, u$ und jedes Paar u, v durch $u, v_k \ldots, v_2, v$ so erhält man einen Weg von x nach y der e vermeidet. Dies ist ein x, y verbindender Weg in G'. Daraus schliessen wir:
($\Leftarrow$) Liegt e auf einem Kreis, dann ist $G' = (V, E \setminus e)$ zusammenhängend, also ist e keine Brücke.
($\Rightarrow$) Ist $e = (u, v)$ keine Brücke, dann ist G' zusammenhängend, also gibt es in G' einen Pfad $(u, v_1 \ldots, v)$ (Lemma 3). Damit ist $(u, v_1 \ldots, v, u)$ ein Kreis in G der e enthält. □

Lemma 6. *Jeder Baum mit zumindest zwei Knoten besitzt zumindets zwei Knoten vom Grad 1.*

Beweis. Wir starten mit einer beliebigen Kante (v_0, v_1) des Baumes und 'suchen ein Blatt jenseits von v_1'. Wenn $d(v_1) = 1$ dann ist v_1 das Blatt, andernfalls gibt es einen Nachbarn $v_2 \neq v_0$ von v_1. Wenn $d(v_2) = 1$ dann ist v_2 das Blatt, andernfalls gibt es einen Nachbarn $v_3 \neq v_1$ von v_2. Wenn $d(v_3) = 1$ dann ist v_3 das Blatt, andernfalls.... Wir erhalten einen

Weg $v_1, v_2, \ldots$. Da Bäume kreisfrei sind, ist der Weg einfach. Da der Graph endlich ist, ist auch jeder einfache Weg endlich. Für den letzte Knoten v_k des Weges gilt $d(v_k) = 1$. Einen zweiten Knoten vom Grad eins findet man in der anderen Richtung als Schlußknoten eines Weges $v_0, v_{-1}, \ldots$ □

In Übung 3 findet man eine zweite Beweisidee für dieses Lemma. Man benutzt dazu, daß ein Baum auf n Knoten genau $n - 1$ Kanten hat.

Lemma 7. *Sei* G *zusammenhängend.* G *ist ein Baum genau dann, wenn jede Kante von* G *eine Brücke ist.*

Beweis. G Wald $\iff$ G hat keinen Kreis $\iff$ jede Kante ist eine Brücke. Das erste " $\iff$ " war die Definition, das zweite ist Lemma 5.

Da Bäume gerade die zusammenhängenden Wälder sind, folgt die Behauptung. □

Erläuterung zum Titelbild

Der dargestellte Graph ist ein 4-regulärer Graph mit 20 Knoten. Das Bild beweist, daß man die Kanten des Graphen in zwei disjunkte Hamilton–Kreise zerlegen kann. Eine weitere Zerlegung der Kantenmenge in fünf disjunkte Kreise, die sich paarweise je zwei mal kreuzen ist möglich.

In einer kürzlich fertiggestellten Arbeit konnten wir zeigen:

Satz. *Sei* G *ein 4-regulärer Graph, der kreuzungsfrei in die Ebene gezeichnet ist, und dessen Kantenmenge sich in Kreise zerlegen läßt, so daß gilt:*

(1) *Jeder Knoten ist Kreuzungspunkt zweier Kreise.*

(2) *Je zwei Kreise kreuzen sich genau zwei mal.*

(3) *Einer der Kreise hat von jedem Paar der übrigen Kreise genau einen Kreuzungspunkt im Inneren.*

Dann läßt sich die Kantenmenge von G *in zwei disjunkte Hamilton-Kreise zerlegen.*