

Minimale Schnitte in Graphen

Semesterarbeit von Thomas Kausch
betreut durch Prof. Emo Welzl und Joachim Giesen

Zürich, 3. Februar 1998

Inhaltsverzeichnis

1	Komplexität	4
1.1	$\mathcal{NP}$ -vollständige Schnittprobleme	4
2	Ein einfacher <i>min-Cut</i> Algorithmus	9
2.1	Algorithmus	10
2.2	Korrektheit	11
2.3	Laufzeit	12
3	Verbesserter <i>min-Cut</i> Algorithmus	14
3.1	Algorithmus	14
3.2	Korrektheit	15
3.3	Laufzeit	17
4	Schranken für den minimalen Schnitt	19
4.1	Erste obere Schranke	20
4.2	Zweite obere Schranke	20
4.3	Dritte obere Schranke	23
4.4	Approximationsalgorithmus für <i>max-Cut</i>	25
5	Minimale <i>s-t</i> Schnitte	27
5.1	Das Gewicht minimaler <i>s-t</i> Schnitte	28
5.2	Schnittäquivalente Bäume	30
5.3	Kleinste minimale <i>s-t</i> Schnitte	31
6	Maximale Flüsse und minimale Schnitte	33
6.1	Das Problem des maximalen Flusses	35
6.2	Ein Algorithmus für das <i>max-Flow</i> Problem	38

Einleitung

Das *min-Cut Problem* ist eines der klassischen algorithmischen Probleme der Graphentheorie und tritt in vielen praktischen Anwendungen auf, z.B. in der *Netzwerktechnik*, dem *Circuit-Design* oder der *Transportplanung*. Es besteht darin, die Knotenmenge V in zwei nichtleere Mengen aufzuteilen, so dass die Summe der Gewichte der Kanten, welche die beiden Mengen verbinden, minimal wird. Um einen minimalen Schnitt¹ zu finden, kann man die Dualität zwischen dem *max-Flow* und *min-Cut* Problem ausnutzen. Das berühmte *max-Flow min-Cut* Theorem von Ford und Fulkerson [FF56] sagt, dass die Grösse eines maximalen s - t Flusses gleich der Grösse eines minimalen s - t Schnittes ist. Man findet also einen minimalen Schnitt, indem man alle $\binom{|V|}{2}$ minimalen s - t Schnitte mit Hilfe eines Flussalgorithmus bestimmt und dann den leichtesten auswählt. Bis vor kurzem beruhten alle minimalen Schnittelgorithmen auf Flussberechnungen. Im zweiten Kapitel stellen wir einen minimalen Schnittelgorithmus von Stoer und Wagner [SW94] vor, der keine Flussberechnungen braucht und Zeit $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ benötigt, d.h. eine leicht bessere Laufzeit als die bisher bekannten Schnittelgorithmen besitzt. Der Algorithmus hat eine einfache Beschreibung, ist einfach zu implementieren und besitzt einen einfachen Beweis der Korrektheit. Kurz zusammengefasst, besteht der Algorithmus aus $|V| - 1$ Phasen, in denen jeweils ein minimaler Schnitt zwischen zwei Knoten berechnet wird und diese Knoten dann verschmolzen werden. In Kapitel drei zeigen wir, wie man den Algorithmus von Stoer und Wagner durch eine kleine Veränderung ver-

¹Ein Schnitt in einem ungerichteten, gewichteten Graphen $G = (V, E)$ ist eine Partition der Knoten in zwei nicht leere Mengen S und $V \setminus S$. Das Gewicht des Schnitts $(S, V \setminus S)$ ist die Summe der Gewichte der Kanten, die einen Endknoten in S und den anderen in $V \setminus S$ haben. Ein interessantes Problem ist es, den Schnitt kleinster Grösse zu finden. Dieses Problem wird als *min-Cut* Problem bezeichnet. Wir nehmen im folgenden an, dass der Graph G zusammenhängend ist. Für nicht zusammenhängende Graphen $G = (V, E)$ findet man mit Breitensuche in Zeit $\mathcal{O}(|E|)$ eine Zusammenhangskomponente und damit einen minimalen Schnitt.

bessern kann. Asymptotisch besitzt der neue Algorithmus jedoch dieselbe Laufzeit. Dem neuen Algorithmus kann man einen beliebigen Schnitt übergeben, der zur Berechnung des minimalen Schnittes verwendet wird. Ist das Gewicht dieses Schnittes eine gute obere Schranke für das Gewicht des minimalen Schnittes, so verkürzt sich die Laufzeit des Algorithmus. Wir sind also an der schnellen Berechnung von fast minimalen Schnitten interessiert. Im vierten Kapitel zeigen wir, wie man solche Schnitte findet.

Im fünften Kapitel betrachten wir minimale s - t Schnitte². Wir geben einen einfachen Beweis, dass es in einem Graphen mit $|V|$ Knoten höchstens $|V| - 1$ im Gewicht verschiedenen minimale s - t Schnitte gibt. Dann stellen wir uns die Aufgabe, einen minimalen Schnitt nur mit Hilfe von minimalen s - t Schnittberechnungen zu bestimmen. Wir zeigen, dass man nicht mit weniger als $|V| - 1$ minimalen s - t Schnittberechnungen auskommt und zeigen gleichzeitig, dass $|V| - 1$ s - t Schnittberechnungen genügen, um einen minimalen Schnitt zu berechnen. Zudem beweisen wir, dass jeder Graph *schnittäquivalent* zu einem Baum ist. Diesen Baum kann man mit dem Algorithmus von Gromory und Hu [GH61] in Zeit $\mathcal{O}(|V||E|\log(|V|^2/|E|))$ berechnen. Im letzten Kapitel geben wir einen Beweis für das *max-Flow min-Cut* Theorem von Ford und Fulkerson.

Im ersten Kapitel über Komplexität geht es darum, das *min-Cut* Problem einzuordnen und ein Gefühl für die Schwierigkeit von Schnittproblemen zu bekommen. Wir zeigen, dass viele Schnittprobleme $\mathcal{NP}$ -vollständig sind. Insbesondere ist das *min-Cut* Problem für Graphen mit ganzzahligen Kantengewichten $\mathcal{NP}$ -vollständig. Deshalb ist es ziemlich erstaunlich, dass das *min-Cut* Problem für Graphen mit positiven Kantengewichten in $\mathcal{P}$ liegt.

²Ein Schnitte $(S, V \setminus S)$ ist ein s - t Schnitt, falls Knoten s in S und t in $V \setminus S$ liegt.

Kapitel 1

Komplexität

Wir haben uns das Problem gestellt, in einem ungerichteten, gewichteten Graphen $G = (V, E)$ einen Schnitt $(S, \bar{S})$, mit kleinstem Gewicht zu finden. Wir suchen also eine Partition $(S, \bar{S})$ der Knoten von V , so dass die Summe der Gewichte der Kanten, die einen Endknoten in S und den anderen in $\bar{S}$ haben, minimal wird. Ebenso gut könnte man sich das Problem stellen, einen Schnitt mit grösstem Gewicht zu berechnen oder jenen Schnitt, der die Knotenmenge V in zwei Mengen gleicher Kardinalität aufteilt und das grösste bzw. kleinste Gewicht besitzt. Letztere Probleme werden als *max-Cut*, *min-Bisection* bzw. *max-Bisection* Problem bezeichnet. Wir zeigen, dass all diese Probleme für ungerichtete und mit ganzen Zahlen gewichtete Graphen $\mathcal{NP}$ -vollständig sind. Es gilt sogar, dass das *max-Cut* und *max-Bisection* Problem für Graphen mit ganzzahligen, positiven Kantengewichten $\mathcal{NP}$ -vollständig ist. Deshalb ist es doch ziemlich erstaunlich, dass das *min-Cut* Problem für Graphen mit positiven Kantengewichten in $\mathcal{P}$ liegt und seine Laufzeit durch ein Polynom 3-ten Grades in der Eingabegrösse beschränkt ist; wie wir im nächsten Kapitel zeigen werden.

Eine gute Einführung in die Komplexität findet man im Buch *Computational Complexity* von Papadimitriou [PA94], aus dem auch der Beweis von Theorem 1.1 stammt.

1.1 $\mathcal{NP}$ -vollständige Schnittprobleme

Das *max-Cut* und *min-Cut* Problem sind Optimierungsprobleme. In diesem Abschnitt sind wir nur an ihren Entscheidungsvarianten interessiert. *Min-*

Cut (D) fragt, ob es für einen gegebenen Graphen G und eine Schranke B einen Schnitt mit Gewicht höchstens B gibt. *Max-Cut* (D) fragt, ob es für einen gegebenen Graphen G und eine Schranke B einen Schnitt mit Gewicht mindestens B gibt. Wie man leicht einsieht, sind die Optimierungsprobleme mindestens so schwer, da man mit ihnen die zugehörigen Entscheidungsprobleme lösen kann. In diesem Abschnitt steht *min-Cut* und *max-Cut* für das entsprechende Entscheidungsproblem.

Theorem 1.1 *Das max-Cut Problem ist für Graphen mit positiven, ganzzahligen Kantengewichten $\mathcal{NP}$ -vollständig.*

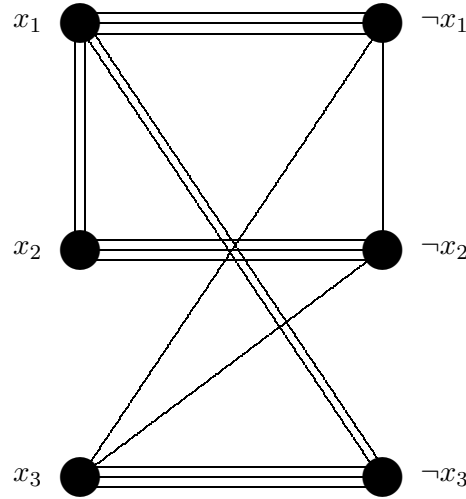
Beweis: Wir zeigen, dass das *max-Cut* Problem für Multigraphen $\mathcal{NP}$ -vollständig ist. Multigraphen sind ungerichtete, ungewichtete Graphen, in denen mehrere Kanten zwischen zwei Knoten existieren dürfen. Eine Kante trägt dabei zum Gewicht eines Schnittes $(S, \bar{S})$ bei, falls sie einen Endknoten in S und den anderen in $\bar{S}$ hat.

Wir reduzieren *NAESAT*¹ auf *max-Cut*. Gegeben sind also m Klauseln mit jeweils 3 Literalen. Wir müssen einen Graphen $G = (V, E)$ und eine Schranke K konstruieren, so dass die Knotenmenge V in zwei gleichgrosse Mengen S und $\bar{S}$ aufgeteilt werden kann, mit mindestens K Kanten im Schnitt $(S, \bar{S})$, genau dann wenn eine Wahrheitsbelegung T existiert, die mindestens ein Literal jeder Klausel *wahr* und eines *falsch* macht. O.B.d.A. können wir annehmen, dass höchstens zwei Literale in einer Klausel gleich sind, da sonst keine Wahrheitsbelegung für *NAESAT* existiert.

Angenommen, es seien m Klauseln $C_1, \dots, C_m$ in denen die n Variablen $x_1, \dots, x_n$ gegeben. G hat $2n$ Knoten, die wir mit $x_1, \dots, x_n$ und $\neg x_1, \dots, \neg x_n$ bezeichnen. Wir brauchen ein Dreieck als Grundbaustein für unsere Reduktion. Dabei benutzen wir die Eigenschaft, dass das Gewicht jedes Schnittes im Dreieck zwei ist. Insbesondere ist das Gewicht eines maximalen Schnittes zwei. Für jede Klausel $C_i = (\alpha \vee \beta \vee \gamma)$ (α, β und γ sind auch Knoten von G) fügen wir zu E die Kanten des Dreiecks $[\alpha, \beta, \gamma]$ hinzu. Falls zwei der Literale identisch sind, so lassen wir die dritte Kante weg und das Dreieck degeneriert so zu zwei parallelen Kanten zwischen den beiden verschiedenen Literalen. Schliesslich fügen wir für jede Variable x_i n_i parallele Kanten von

¹Eine Instanz von *NAESAT* besteht aus einer Menge von m Klauseln mit jeweils drei Literalen. Gesucht ist eine Wahrheitsbelegung T der Variablen, so dass nicht alle Literale einer Klausel *wahr* oder *falsch* sind. *NAESAT* steht für not-all-equal 3SAT und ist $\mathcal{NP}$ -vollständig. Ein Beweis dafür steht in [PA94].

x_i nach $\neg x_i$ ein, wobei n_i die Anzahl der Literale x_i oder $\neg x_i$ in den Klauseln ist. Damit ist die Konstruktion von G beendet. Betrachte Abbildung 1.1), die die Reduktion von $(x_1 \vee x_2 \vee x_2) \wedge (x_1 \vee \neg x_3 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$ darstellt. K wird auf $5m$ gesetzt.

Abbildung 1.1: Reduktion auf *max-Cut*

Angenommen, es existiert ein Schnitt $(S, \bar{S})$ mit Gewicht $5m$ oder mehr. O.B.d.A können wir annehmen, dass alle Knoten x_i von ihren Negationen $\neg x_i$ getrennt sind. Denn falls sowohl x_i als auch $\neg x_i$ auf der selben Seite des Schnittes liegen, können sie zusammen mit höchstens $2n_i$ Kanten inzident sein, die im Schnitt liegen. D.h. eines der Literale x_i oder $\neg x_i$ ist mit n_i oder weniger Kanten inzident. Da zwischen x_i und $\neg x_i$ mindestens n_i Kanten liegen, kann dieses Literal die Seite wechseln, ohne die Grösse des Schnittes zu verkleinern. Wir definieren nun eine Wahrheitsbelegung der Variablen so, dass die Literale in S *wahr* und jene in $\bar{S}$ *falsch* werden.

Die gesamte Anzahl der Kanten im Schnitt, die gegenüberliegende Literale verbinden, ist $3m$ (also soviel wie es Literale gibt). Die restlichen $2m$ Kanten müssen durch die Dreiecke verursacht werden, die zu den Klauseln gehören. Da jedes Dreieck höchstens zwei zur Grösse eines Schnittes beitragen kann und es nur m Dreiecke gibt, muss jedes Dreieck im Schnitt liegen. Dass ein Dreieck im Schnitt liegt, bedeutet jedoch, dass mindestens ein Literal seiner Klausel *wahr* und mindestens eines seiner Klausel *falsch* ist.

Deshalb erfüllt die Wahrheitszuweisung die Anforderungen von *NAESAT*.

Sei umgekehrt T eine Wahrheitszuweisung, die alle Klauseln erfüllt. Dann ist es einfach einen Schnitt $(S, \bar{S})$ der Grösse $5m$ zu definieren. Alle Literale mit dem Wert *falsch* gehören zu S und die übrigen zu $\bar{S}$. Da die Literale x_i und $\neg x_i$ nie denselben Wahrheitswert annehmen, liegen sicher $3m$ Kanten im Schnitt. Da jede Klausel mindestens ein Literal mit dem Wert *falsch* und eines mit dem Wert *wahr* besitzt, liegt jedes Dreieck im Schnitt. Also gibt es m Dreiecke mit $2m$ Schnittkanten. Insgesamt liegen also $5m$ Kanten im Schnitt. $\square$

Theorem 1.2 *Das min-Cut Problem ist für Graphen mit ganzzahligen Kantengewichten $\mathcal{NP}$ -vollständig.*

Beweis: Offensichtlich liegt *min-Cut* in $\mathcal{NP}$. Es bleibt zu zeigen, wie eine Instanz von *max-Cut* auf eine Instanz von *min-Cut* abgebildet wird, sodass die neue Instanz genau dann eine Ja-Instanz ist, falls die ursprüngliche Instanz eine Ja-Instanz war. Gegeben eine Instanz ϕ von *max-Cut*. Wir konstruieren eine Instanz $R(\phi)$ von *min-Cut* und zeigen, dass ϕ eine Ja-Instanz von *max-Cut* genau dann wenn $R(\phi)$ eine Ja-Instanz von *min-Cut* ist.

ϕ	$R(\phi)$
$G = (V, E, w)$	$G' = (V, E, w')$
$w : e \in E \mapsto w(e)$	$w' : e \in E \mapsto -w(e)$
K	$-K$

Ist ϕ eine Ja-Instanz von *max-Cut* dann existiert ein Schnitt mit grösserem Gewicht als K und damit ist $R(\phi)$ eine Ja-Instanz von *min-Cut*. Ist umgekehrt $R(\phi)$ eine Ja-Instanz von *min-Cut*, dann ist offensichtlich auch ϕ eine Ja-Instanz von *max-Cut*. Zudem ist die Reduktion in polynomieller Zeit berechenbar. $\square$

In manchen interessanten Anwendungen sind wir an einer Partition $S, \bar{S}$ des Graphen interessiert, in der die zwei Mengen S und $\bar{S}$ nicht beliebig gross sein dürfen. Angenommen wir interessieren uns für Schnitte $S, \bar{S}$ der Grösse mindestens K , für die $|S| = |\bar{S}|$ gilt. Wir nennen dieses Problem *max-Bisection*. Ist *max-Bisection* einfacher oder schwieriger als *max-Cut*? Die zusätzliche Bedingung $|S| = |\bar{S}|$, scheint das Problem schwieriger zu machen. Gleichzeitig wird durch die zusätzlich Bedingung auch die Anzahl möglicher Lösungen verkleinert und damit die Komplexität verringert.

Wir sind wiederum nur an der Entscheidungsvariante von *max-Bisection* interessiert. Diese lautet: Existiert ein Schnitt $(S, \bar{S})$, so dass $|S| = |\bar{S}|$ und das Gewicht des Schnittes mindestens B ist.

Theorem 1.3 *Max-Bisection ist für Graphen mit ganzzahligen, positiven Kantengewichten $\mathcal{NP}$ -vollständig.*

Beweis: Wir reduzieren *max-Cut* auf *max-Bisection*. Sei G der Eingabegraph des *max-Cut* Problems. Wir fügen $|V|$ nicht verbundene Knoten zu G hinzu und erhalten so den Eingabegraphen G' für *max-Bisection*. Nun besitzt G genau dann einen Schnitt mit Gewicht mindestens B , falls G' eine Bisektion mit Gewicht mindestens B besitzt. Da die $|V|$ Knoten in Zeit $\mathcal{O}(|V|)$ zu G hinzugefügt werden können, folgt das Theorem. $\square$

Theorem 1.4 *Min-Bisection ist für Graphen mit ganzzahligen Kantengewichten $\mathcal{NP}$ -vollständig.*

Beweis: Wir reduzieren *max-Bisection* auf *min-Bisection*. Sei G der Eingabegraph des *max-Bisection* Problems. Wir multiplizieren alle Kantengewichte des Graphen mit -1 und erhalten so den Eingabegraphen G' für das *min-Bisection* Problem. Nun besitzt G genau dann eine Bisektion mit Gewicht mindestens B , falls G' eine Bisektion mit Gewicht höchstens $-B$ besitzt. Zudem kann die Reduktion in polynomieller Zeit berechnet werden. $\square$

Kapitel 2

Ein einfacher *min-Cut* Algorithmus

Erstaunlicherweise gibt es einen polynomiellen Algorithmus zur Lösung des *min-Cut* Problems mit positiven Kantengewichten. Man kann z.B. die Dualität zwischen dem *max-Flow* und *min-Cut* Problem ausnutzen, um einen minimalen Schnitt zu finden. Das berühmte *max-Flow min-Cut* Theorem von Ford und Fulkerson [FF56] sagt, dass die Grösse eines maximalen s - t Flusses gleich der Grösse eines minimalen s - t Schnittes ist. Man findet also einen minimalen Schnitt, indem man alle $\binom{n}{2}$ minimalen s - t Schnitte mit Hilfe eines Flussalgorithmus bestimmt und dann den leichtesten auswählt. Bis vor kurzem beruhten alle minimalen Schnittalgorithmen auf Flussberechnungen. Hao und Orlin [HO92] zeigten, wie man mit dem Flussalgorithmus von Goldberg und Tarjan [GT88], dem schnellsten bis heute bekannten Flussalgorithmus, einen minimalen Schnitt in Zeit $\mathcal{O}(|V||E| \log(|V|^2/|E|))$ berechnet. Im gleichen Jahr veröffentlichten Nagamochi und Ibaraki [NI92] den ersten minimalen Schnittalgorithmus, der nicht auf Flussberechnungen beruhte und eine leicht bessere Laufzeit von $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ hat, dafür aber viel komplizierter ist. In diesem Kapitel stellen wir den minimalen Schnittalgorithmus von Stör und Wagner [SW94] vor, der ebenfalls kein Flussalgorithmus ist. Dieser Algorithmus, der aus $|V| - 1$ fast identischen Phasen besteht und sehr einfach zu implementieren ist, besitzt die Laufzeit $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ und einen erstaunlich einfachen Beweis der Korrektheit.

Sei $G = (V, E)$ ein ungerichteter gewichteter Graph mit Knotenmenge V und Kantenmenge E . Jede Kante e hat ein positives reelles Gewicht $w(e)$.

Ein minimaler s - t Schnitt $(S, \bar{S})$ ist eine Partition der Knoten in eine Menge S und $\bar{S}$, sodass Knoten s in S und Knoten t in $\bar{S}$ liegt.

2.1 Algorithmus

Der Algorithmus von Stoer und Wagner [SW94] besteht aus $|V| - 1$ Phasen, in denen jeweils ein minimaler s - t Schnitt in einem Graphen berechnet und das Knotenpaar s, t verschmolzen wird.

MINIMUMCUTPHASE(G, a)

$A \leftarrow a$

solange $A \neq V$

Füge den Knoten $z \in V_G \setminus A$ zu A hinzu, der insgesamt die schwersten Kanten zu A hat.

Sei t und s der letzte und vorletzte Knoten, der zu A hinzugefügt worden ist, so merke $(\{t\}, V_G \setminus \{t\})$ als Phasenschnitt und kontrahiere s und t .

A ist eine Teilmenge der Knoten des Graphen. In jedem Schritt wird ein Knoten $z \notin A$ zu A hinzugefügt, der insgesamt die schwersten Kanten zu A hat, d.h. $w(A, z) = \max\{w(A, y) \mid y \notin A\}$. Dabei ist $w(A, y)$ die Summe der Gewichte der Kanten zwischen A und y . Am Ende jeder Phase werden jene zwei Knoten s und t verschmolzen, die zuletzt zu A hinzugekommen sind, d.h. die beiden Knoten s und t werden durch einen neuen Knoten u ersetzt und alle Kanten von s und t zu einem Knoten $x \in V \setminus \{s, t\}$ werden durch eine Kante (u, x) mit Gewicht $w(u, x) = w(s, x) + w(t, x)$ ersetzt. Kanten mit Endknoten s oder t werden gelöscht. Der Schnitt zwischen dem Knoten t und den übrigen Knoten von V heisst Phasenschnitt. Der leichteste Phasenschnitt ist der gesuchte minimale Schnitt.

MINIMUMCUT(G, a)

solange $|V| > 1$

MINIMUMCUTPHASE(G, a)

Ausgabe des kleinsten Phasenschnittes

2.2 Korrektheit

Der Kern des Beweises ist das folgende Lemma:

Lemma 2.1 *Jeder Phasenschnitt ist ein minimaler s - t Schnitt im momentanen Graphen, wobei s und t die letzten in der Phase hinzugefügten Knoten sind.*

Angenommen das Lemma stimmt. Wir zeigen mit einer einfachen Fallunterscheidung, dass der kleinste Phasenschnitt tatsächlich der gesuchte minimale Schnitt ist. Sei G der Eingabegraph einer Phase und s und t die letzten in der Phase hinzugefügten Knoten. Sei G' der Graph, der durch verschmelzen der Knoten s und t in G entsteht. Wir zeigen, dass entweder der Phasenschnitt ein minimaler Schnitt von G ist oder der minimale Schnitt in G' ein minimaler Schnitt von G ist.

Falls der Graph G einen minimalen Schnitt C_{min} besitzt, der gleichzeitig ein s - t Schnitt ist, dann gilt nach Lemma 2.1 $w(C_{min}) \geq w(t, V \setminus \{t\})$. Nach Definition des minimalen Schnittes ist $w(C_{min}) \leq w(t, V \setminus \{t\})$. D.h. $w(C_{min}) = w(t, V \setminus \{t\})$. Der Phasenschnitt $\{t\}, V \setminus \{t\}$ ist damit ein minimaler Schnitt in G .

Falls der Graph G keinen minimalen Schnitt C_{min} besitzt, der gleichzeitig s - t Schnitt ist, dann besitzt G einen minimalen Schnitt mit s und t auf der selben Seite. Deshalb ist ein minimaler Schnitt in G' , dem Eingabegraphen der nächsten Phase, der sich von G durch verschmelzen von s und t unterscheidet, ein minimaler Schnitt in G .

In der letzten Phase wird immer ein minimaler Schnitt berechnet. Deshalb folgt durch Induktion über die Phasen die Behauptung. Es bleibt zu zeigen, dass ein Phasenschnitt ein minimaler s - t Schnitt ist, wobei s und t die letzten in der Phase hinzugefügten Knoten sind.

Beweis(Lemma 2.1): Ein Durchlauf von MINIMUMCUTPHASE ordnet die Knoten des Eingabegraphen G , beginnend mit a und endend mit s und t , nach der Reihenfolge in der sie zu A hinzugekommen sind. Wir betrachten nun einen beliebigen s - t Schnitt C des Graphen G und zeigen, dass dieser mindestens so gross ist wie der Phasenschnitt, d.h. $w(t, V \setminus \{t\}) \leq w(C)$.

Wir nennen einen Knoten $v \neq a$ aktiv bezüglich C , falls v nicht auf der gleichen Seite von C wie sein Vorgänger liegt. Sei $w(C)$ das Gewicht von C ,

A_v die Menge der Vorgänger von v (ohne v), C_v die Partition von $A_v \cup \{v\}$ definiert durch C und $w(C_v)$ das Gewicht der Partition, d.h. der Summe der Gewichte der Kanten, die von der einen Seite der Partion zur anderen Seite gehen. Wir zeigen durch Induktion über die Menge der aktiven Knoten, dass für jeden aktiven Knoten v

$$w(A_v, v) \leq w(C_v). \quad (2.1)$$

Für den ersten aktiven Knoten ist die Ungleichung 2.1 mit Gleichheit erfüllt. Sei nun die Ungleichung 2.1 für alle aktiven Knoten bis zu einem Knoten v erfüllt und sei w der nächste aktive Knoten. Dann gilt

$$w(A_w, w) = w(A_v, w) + w(A_w \setminus A_v, w). \quad (2.2)$$

Nun gilt jedoch $w(A_v, w) \leq w(A_v, v)$, da v als derjenige Knoten gewählt wurde, der mit A_v am stärksten verbunden war. Wegen der Induktionsannahme gilt $w(A_v, v) \leq w(C_v)$. Alle Kanten zwischen $A_w \setminus A_v$ und w liegen im Schnitt von C . Jede Kante die im Schnitt C_v oder $(A_w \setminus A_v, w)$ liegt, liegt auch im Schnitt C_w . Zudem liegt keine Kante im Schnitt C_v und $(A_w \setminus A_v, w)$. Deshalb gilt die Ungleichung (*) und zusammenfassend:

$$\begin{aligned} w(A_w, w) &= w(A_v, w) + w(A_w \setminus A_v, w) \\ &\stackrel{Algo}{\leq} w(A_v, v) + w(A_w \setminus A_v, w) \\ &\stackrel{I.A.}{\leq} w(C_v) + w(A_w \setminus A_v, w) \\ &\stackrel{(*)}{\leq} w(C_w). \end{aligned}$$

Da t immer ein aktiver Knoten ist bezüglich C , können wir schliessen

$$w(V \setminus \{t\}, \{t\}) = w(A_t, t) \leq w(C_t) = w(C).$$

□

2.3 Laufzeit

Die Laufzeit des MINIMUMCUT Algorithmus wird bestimmt durch die Laufzeit der $|V| - 1$ Aufrufe von MINIMUMCUTPHASE. Es genügt zu zeigen, dass ein einzelner Aufruf von MINIMUMCUTPHASE höchstens $\mathcal{O}(|E| + |V| \log |V|)$ Zeit benötigt, um eine Gesamtlaufzeit von $\mathcal{O}(|V||E| + |V|^2 \log |V|)$ zu garantieren.

Um eine schnelle Laufzeit für MINIMUMCUTPHASE zu erhalten, muss der Knoten mit grösstem Gewicht zu A schnell bestimmt werden können. Während einer Phase befinden sich alle Knoten, die noch nicht zu A gehören, in einer Prioritätswarteschlange. Die Priorität eines Knotens v ist die Summe der Gewichte der Kanten die v mit A verbinden, d.h. $w(A, v)$. Falls ein Knoten v zu A hinzugefügt wird, muss v aus der Schlange gelöscht und die Priorität aller Knoten $u \notin A$, die mit v verbunden sind, um $w(u, v)$ erhöht werden. Während einer Phase wird für jede Kante genau eine INCREASEKEY Operation und für jeden Knoten eine EXTRACTMAX Operation ausgeführt; insgesamt also $|E|$ INCREASEKEY- und $|V|$ EXTRACTMAX-Operationen. Benutzt man einen Fibonacci Heap [FT87] als Prioritätswarteschlange, benötigt man für alle Operationen $\mathcal{O}(|E| + |V| \log |V|)$ Zeit.

Kapitel 3

Verbesserter *min-Cut* Algorithmus

Die Laufzeit des einfachen *min-Cut* Algorithmus wird durch die Laufzeit einer Phase bestimmt. Wir beschreiben einen Algorithmus, der den Aufwand für eine Phase verkleinert. Die Kernidee des verbesserten Algorithmus ist es, zwei Knoten in einer Phase vorzeitig zu verschmelzen, falls dadurch der minimale Schnitt nicht verloren geht. Damit werden die Phasen des *min-Cut* Algorithmus kürzer und die Gesamtlaufzeit schneller. Für zyklische Graphen hat der verbesserte Algorithmus jedoch die gleiche Laufzeit wie der einfache Algorithmus.

3.1 Algorithmus

Der Algorithmus besteht aus $|V|-1$ Phasen, in denen entweder ein minimaler s - t Schnitt gefunden oder ein Knotenpaar s, t in einem Graphen G bestimmt wird, so dass der minimale s - t Schnitt in G mindestens so gross, wie der kleinste Phasenschnitt ist. Am Ende jeder Phase werden die Knoten s und t verschmolzen und der Phasenschnitt gespeichert.

```
MINIMUMCUTPHASE( $G, a, \alpha$ )  
( $\alpha$  ist das Gewicht des bislang leichtesten Phasenschnittes)  
 $A \leftarrow a$   
repeat
```

Füge den Knoten $z \in V_G \setminus A$ zu A hinzu, der insgesamt die schwersten Kanten zu A hat.

until $A = V$ oder $w(z, A_z) \geq \alpha$

Sei t und s der letzte und vorletzte Knoten, der zu A hinzugefügt worden ist, so merke $(\{t\}, V_G \setminus \{t\})$ als Phasenschnitt und kontrahiere s und t

A ist eine Teilmenge der Knoten des Graphen. In jedem Schritt wird ein Knoten $z \in A$ zu A hinzugefügt, der insgesamt die schwersten Kanten zu A hat. Dies geht solange bis $A = V$ oder $w(z, A_z)$ mindestens so gross ist wie das kleinste Gewicht eines Phasenschnittes. Am Anfang der Phase ist das kleinste Gewicht eines Phasenschnittes unendlich, da noch kein Phasenschnitt existiert und das Minimum einer leeren Menge unendlich ist.

3.2 Korrektheit

Sei G ein Graph mit Knotenmenge V und Kantenmenge E ein Untergraph G_U ist ein Graph mit Knotenmenge $U \subseteq V$ und Kantenmenge $E_U = \{(u, v) \mid (u, v) \in E \text{ und } u, v \in U\}$. In MINIMUMCUTPHASE werden die Knoten vom Eingabegraph G in einer bestimmten Reihenfolge in A aufgenommen. Sei A_w die Menge der Knoten, die vor w zu A hinzugefügt wurden und Knoten v der Vorfahre von w . Nach Lemma 2.1 ist der Schnitt (w, A_w) ein minimaler v - w Schnitt im Untergraphen $G_{A_w \cup \{w\}}$. Da ein minimaler v - w Schnitt in einem Graphen G mindestens so gross ist wie ein minimaler v - w Schnitt in einem Untergraphen von G der v und w enthält, gilt das folgende Lemma.

Lemma 3.1 *Sei v der Vorfahre von w in der Reihenfolge, in der die Knoten während MINIMUMCUTPHASE(G, a) in A aufgenommen wurden. Ein v - w Schnitt in G hat dann mindestens die Grösse $w(w, A_w)$, wobei A_w die Menge der Knoten ist, die vor w zu A hinzugefügt wurden.*

Beweis: Wir zeigen, dass der kleinste Phasenschnitt der gesuchte minimale Schnitt in $G = (V, E)$ ist. Sei G_i der Eingabegraph und C_i der Phasenschnitt der Phase i . Wir zeigen durch Induktion, dass für jedes $k \in \{1, \dots, |V| - 1\}$

$$\mu(G) = \min(\mu(G_k), w(C_1), \dots, w(C_{k-1})) \quad (3.1)$$

gilt. Dabei bezeichnet $\mu(G)$ das Gewicht eines minimalen Schnittes in G . In der ersten Phase ist $\mu(G) = \mu(G_1)$ und die Behauptung ist richtig. Angenommen die Induktionsannahme 3.1 stimmt für k . Existiert ein Phasenschnitt C_l , $l \in \{1, \dots, k-1\}$, so dass $w(C_l) = \mu(G)$, dann ist 3.1 offensichtlich auch für $k+1$ erfüllt. Sei also

$$\begin{aligned}\mu(G_k) &= \mu(G) \text{ und} \\ \mu(G) &< w(C_1), \dots, w(C_{k-1}).\end{aligned}$$

Wir müssen zeigen, dass dann entweder $w(C_k) = \mu(G)$ oder $\mu(G_{k+1}) = \mu(G)$ gilt. Seien s und t die letzten in der Phase k zu A hinzugefügten Knoten. Terminiert die Phase wegen der Bedingung $A = V$, dann ist der Phasenschnitt C_k nach Lemma 2.1 ein minimaler s - t Schnitt in G_k . Falls der Graph G_k einen minimalen Schnitt besitzt, der gleichzeitig s - t Schnitt ist, dann gilt $w(C_k) = \mu(G_k) = \mu(G)$. Falls der Graph G_k keinen minimalen Schnitt besitzt, der gleichzeitig s - t Schnitt ist, dann besitzt G_k einen minimalen Schnitt mit s und t auf derselben Seite und es gilt $\mu(G_{k+1}) = \mu(G_k) = \mu(G)$. Terminiert die Phase jedoch wegen der Bedingung $w(t, A_t) \geq \alpha$, dann gilt nach Lemma 3.1, dass der s - t Schnitt in G_k mindestens Gewicht $w(t, A_t)$ besitzt. Sei C_{st} ein s - t Schnitt in G_k und α das kleinste Gewicht eines Phasenschnittes $C_1, \dots, C_{k-1}$. Somit ist $\alpha > \mu(G)$ und es gilt

$$w(C_{st}) \geq w(t, A_t) \geq \alpha > \mu(G).$$

Das heisst der minimale Schnitt in G_k ist kein s - t Schnitt. Deshalb ist ein minimaler Schnitt im Graphen G_{k+1} , der aus G_k durch verschmelzen von s und t entsteht, ein minimaler Schnitt in G_k . Somit ist $\mu(G_{k+1}) = \mu(G_k) = \mu(G)$. In der letzten Phase wird immer ein minimaler Schnitt berechnet und es gilt $\mu(G_{n-1}) = w(C_{n-1})$, wobei $n = |V|$. Also

$$\mu(G) = \min(\mu(G_{n-1}), w(C_1), \dots, w(C_{n-2})) \quad (3.2)$$

$$= \min(w(C_1), \dots, w(C_{n-1})) \quad (3.3)$$

□

Da $\alpha = \infty$ können während der ersten Phase nie zwei Knoten vorzeitig verschmolzen werden. Um dies zu vermeiden, kann man einen Schnitt C_0 bestimmen und sich diesen als nullten Phasenschnitt merken. α entspricht dann dem Gewicht des nullten Phasenschnittes. Zum Beispiel könnte man im Graphen $G = (V, E)$ den Knoten v mit kleinstem Grad $\deg(v) := \sum_{w:(v,w) \in E} w(v, w)$ bestimmen und sich den Schnitt $(v, V \setminus v)$ als nullten

Phasenschnitt merken. Den Knoten v mit kleinstem Grad erhält man in Zeit $\mathcal{O}(|V| + |E|)$, indem man alle Kanten durchläuft und für jede Kante $(u, v) \in E$ den Zähler der Endknoten von u und v um $w(u, v)$ erhöht. Eine Minimumsuche über den Zählern der Knoten ergibt den Knoten mit kleinstem Knotengrad. Im nächsten Kapitel zeigen wir, dass der Knotengrad eine gute obere Schranke für den minimalen Schnitt ist. Deshalb wird man in vielen Phasen frühzeitig verschmelzen können.

3.3 Laufzeit

Die Laufzeit des verbesserten *min-Cut Algorithmus* ist mindestens so gut wie die des einfachen, da jede Phase höchstens dieselbe Laufzeit hat. Das folgende Lemma zeigt jedoch, dass beide Algorithmen die gleiche obere Schranke für die Laufzeit besitzen.

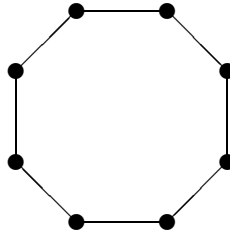
Lemma 3.2 *Die Phasen des verbesserten Algorithmus entsprechen den Phasen des einfachen Algorithmus, falls man einen Zyklus mit gleichen Kantengewichten als Eingabegraph wählt.*

Beweis: Wir zeigen die Behauptung durch Induktion über die Anzahl Knoten des Eingabegraphen. Die Behauptung ist richtig für Zyklen mit zwei Knoten. Angenommen die Behauptung gilt für Zyklen mit $n - 1$ Knoten. Sei G ein Zyklus mit n Knoten. Für den Eingabegraphen G ist die erste Phase des einfachen und verbesserten Algorithmus identisch. Der Eingabegraph G' für die zweite Phase ist für beide Algorithmen derselbe und G' ist wieder zyklisch. Nach Induktionsannahme sind deshalb die weiteren Phasen für beide Algorithmen gleich. $\square$

Der verbesserte *min-Cut Algorithmus* sucht minimale s - t Schnitte, die mindestens so gross sind wie das Gewicht des kleinsten Phasenschnitts. Findet er einen solchen s - t Schnitt werden s und t verschmolzen. Bei zyklischen Graphen wird schon in der ersten Phase ein minimaler Schnitt gefunden. Da jedoch die minimalen s - t Schnitte alle dasselbe Gewicht haben, können in keiner Phase Knoten vorzeitig verschmolzen werden!

In jeder Phase wird vom *min-Cut Algorithmus* ein (s, t) Knotenpaar ausgewählt und der minimale s - t Schnitt berechnet. Falls der *min-Cut Algorithmus* jedes (s, t) Knotenpaar mit Wahrscheinlichkeit $\binom{|V|}{2}$ auswählt, dann

Abbildung 3.1: Für Zyklen degeneriert der verbesserte Algorithmus



werden allgemein jene Graphen hohe Laufzeiten haben, bei denen alle minimalen s - t Schnitte fast gleiches Gewicht besitzen. Denn für diese Graphen ist die Wahrscheinlichkeit klein, dass in einer Phase s und t früh verschmolzen werden kann.

Kapitel 4

Obere Schranken für den minimalen Schnitt

Im vorangehenden Kapitel haben wir einen *min-Cut* Algorithmus beschrieben, der für einen Graphen einen minimalen Schnitt berechnet. Dem Algorithmus kann man einen beliebigen Schnitt übergeben, der zur Berechnung des minimalen Schnittes verwendet wird. Ist das Gewicht dieses Schnittes eine gute obere Schranke für das Gewicht des minimalen Schnittes, so verkürzt sich die Laufzeit des Algorithmus. Deshalb macht es Sinn, dem Algorithmus einen Schnitt mit möglichst kleinem Gewicht zu übergeben. Allerdings sollte die Berechnung dieses Schnittes effizient möglich sein, zum Beispiel in Zeit $\mathcal{O}(|V|)$. Zur Bestimmung des Schnittes kann man auch einen Monte Carlo Algorithmus¹ benutzen, der einen minimalen Schnitt berechnet. Andererseits können wir uns fragen, wie gross ein minimaler Schnitt in einem Graphen höchstens sein kann. Es gibt drei Methoden, wie man solche obere Schranken beweisen kann. Am einfachsten ist es, direkt zu beschreiben, wie man einen Schnitt mit Gewicht kleiner gleich $\alpha(G)$ in einem Graphen G konstruiert. Dabei ist α eine Funktion, die jedem Graphen eine positive, ganze Zahl zuordnet. Eine andere Methode ist es, einen Monte Carlo Algorithmus, der einen minimalen Schnitt in G berechnet, zu analysieren. Das Gewicht des von diesem Algorithmus berechneten Schnittes ist eine Zufallsvariable W , von der man den Erwartungswert $E(W)$ bestimmen kann. Damit gibt es

¹Ein Monte Carlo Algorithmus für das minimale Schnitt Problem ist ein randomisierter Algorithmus, der meistens minimale Schnitte, manchmal aber auch einen nicht minimalen Schnitt berechnet. Die Wahrscheinlichkeit, dass dies geschieht, ist jedoch begrenzt. Die Laufzeit eines Monte Carlo Algorithmus wird nicht durch den Zufall bestimmt.

im Graphen einen Schnitt mit Gewicht höchstens $E(W)$. $E(W)$ ist also eine obere Schranke für den minimalen Schnitt. Eine dritte Methode definiert mit Hilfe des Graphen G eine Matrix L , die sogenannte Laplacematrix von G und beschränkt das Gewicht eines minimalen Schnittes in G durch eine Funktion der Eigenwerte von L . Eine Schranke, die auf der letzten Methode beruht, findet man in [FC96].

Wir beschränken uns in diesem Kapitel auf Multigraphen. Multigraphen sind ungerichtete, ungewichtete Graphen $G = (V, E)$, in denen mehrere Kanten dieselben Endknoten besitzen dürfen. Ein Schnitt $(S, \bar{S})$ in einem ungerichteten Multigraphen G ist eine Partition der Knotenmenge in zwei nichtleere Mengen S und $\bar{S}$. Das Gewicht des Schnittes $(S, \bar{S})$ ist die Anzahl der Kanten, die einen Endknoten in S und den anderen in $\bar{S}$ haben. Der Knotengrad $\deg(v)$ eines Knotens v , ist die Anzahl Kanten im Graph, die v als Endknoten besitzen.

4.1 Erste obere Schranke

In einem Graphen $G = (V, E)$ gibt es immer einen Schnitt mit Gewicht $\min_{v \in V} \deg(v)$, wobei $\deg(v)$ die Anzahl der mit v inzidenten Kanten bezeichnet. Damit gilt das folgende Lemma.

Lemma 4.1 *Sei $G = (V, E)$ ein Multigraph. Falls μ das Gewicht eines minimalen Schnittes in G bezeichnet und $\deg(v)$ die Menge der mit Knoten v inzidenten Kanten ist gilt $\mu \leq \min_{v \in V} \deg(v)$.*

Der kleinste Knotengrad ist für viele Graphen eine ziemlich gute obere Schranke für den minimalen Schnitt. Es gibt aber auch Graphenklassen, für die dies nicht gilt. Für Graphen, die aus zwei vollständigen Komponenten mit jeweils n Knoten bestehen und über eine Kante verbunden sind, ist das Gewicht des minimalen Schnittes 1 aber der minimale Knotengrad n .

4.2 Zweite obere Schranke

Lemma 4.2 *Sei $G = (V, E)$ ein Multigraph. Falls μ das Gewicht eines minimalen Schnittes in G bezeichnet, so gilt für alle $p \in (0, 1)$*

$$\mu \leq \frac{2|E|(p - p^2)}{1 - p^{|V|} - (1 - p)^{|V|}} \quad (:= \alpha).$$

Beweis: Wir zeigen, dass für jedes $p \in (0, 1)$ ein Schnitt in $G = (V, E)$ mit Gewicht kleiner gleich α existiert.

Man konstruiert für $G = (V, E)$ einen zufälligen Schnitt $(S, V \setminus S)$ indem man jeden Knoten von V mit Wahrscheinlichkeit p in S aufnimmt. Das Gewicht des zufälligen Schnittes $(S, V \setminus S)$ ist eine Zufallsvariable Z von der man leicht den Erwartungswert berechnen kann. Sei X_e die Indikatorvariable, dass die Kante e im zufälligen Schnitt $(S, V \setminus S)$ liegt.

$$\begin{aligned} E(Z) &= E\left(\sum_{e \in E} X_e\right) \\ &= \sum_{e \in E} E(X_e) \\ &= \sum_{e \in E} \text{Prob}(\text{Kante } e \text{ im liegt Schnitt } (S, V \setminus S)) \\ &= \sum_{e \in E} 1 - p^2 - (1 - p)^2 \\ &= 2|E|(p - p^2). \end{aligned}$$

Der Erwartungswert $E(Z)$ kann auch direkt berechnet werden. Sind $S_1, \dots, S_l$ alle möglichen Teilmengen von V und $p_1, \dots, p_l$ die zugehörigen Wahrscheinlichkeiten, mit denen die Teilmengen erzeugt werden. Bezeichne $w(S)$ für beliebiges $S \subset V$ das Gewicht des Schnittes $(S, V \setminus S)$. Falls $S_1 = V$ und $S_l = \emptyset$, so gilt

$$E(Z) = p_1 \cdot 0 + p_2 \cdot w(S_2) + \dots + p_{l-1} \cdot w(S_{l-1}) + p_l \cdot 0.$$

Wie man sich leicht überzeugt ist

$$\sum_{i=2}^{l-1} p_i = 1 - p_1 - p_l = 1 - p^{|V|} - (1 - p)^{|V|}.$$

Angenommen es existiert kein $i \in \{2, \dots, l-1\}$, so dass

$$w(S_i) \leq \frac{2|E|(p - p^2)}{1 - p^{|V|} - (1 - p)^{|V|}} (= \alpha)$$

Wir zeigen, dass diese Annahme zum Widerspruch führt. Das heisst es existiert ein Schnitt $(S_i, V \setminus S_i)$ mit $i \in \{2, \dots, l-1\}$ und Gewicht kleiner gleich α .

$$2|E|(p - p^2) = \sum_{i=2}^{l-1} p_i w(S_i)$$

$$\begin{aligned}
&> \sum_{i=2}^{l-1} p_i \frac{2|E|(p-p^2)}{1-p^{|V|}-(1-p)^{|V|}} \\
&= \frac{2|E|(p-p^2)}{1-p^{|V|}-(1-p)^{|V|}} \sum_{i=2}^{l-1} p_i \\
&= 2|E|(p-p^2)
\end{aligned}$$

□

Sei μ das Gewicht eines minimalen Schnittes in $G = (V, E)$. Dann ist $\mu \leq \lfloor \alpha \rfloor$, weil das Gewicht eines minimalen Schnittes in einem Multigraphen immer ganzzahlig ist. Falls man p gegen 0 gehen lässt, wird α minimal und es gilt

$$\begin{aligned}
\mu &\leq \lim_{p \rightarrow 0} \frac{2|E|(p-p^2)}{1-p^{|V|}-(1-p)^{|V|}} \\
&= \lim_{p \rightarrow 0} \frac{\frac{d}{dp} 2|E|(p-p^2)}{\frac{d}{dp} (1-p^{|V|}-(1-p)^{|V|})} \\
&= \lim_{p \rightarrow 0} \frac{2|E|(1-2p)}{|V|(1-p)^{|V|-1} - |V|p^{|V|-1}} \\
&= \frac{2|E|}{|V|}.
\end{aligned}$$

Ein minimaler Schnitt in einem Graphen hat also höchstens Gewicht $\frac{2|E|}{|V|}$. Wir zeigen nun, dass es einen deterministischen Algorithmus gibt, der immer einen solchen Schnitt findet und geben gleichzeitig einen einfacheren Beweis, dass ein minimaler Schnitt höchstens Gewicht $\frac{2|E|}{|V|}$ besitzt.

Lemma 4.3 *Sei $G = (V, E)$ ein Multigraph und μ das Gewicht eines minimalen Schnittes in G , so gilt $\mu \leq \frac{2|E|}{|V|}$.*

Beweis: Sei $d(v)$ die Menge der Kanten, die einen Endknoten v besitzen. Für alle $v \in V$ ist $|d(v)| \geq \mu$, da man sonst jenen Knoten mit $|d(v)| < \mu$ als besseren Schnitt wählen könnte. Damit folgt

$$2|E| = \sum_{v \in V} |d(v)| \geq \sum_{v \in V} \mu = |V|\mu.$$

□

Analog kann man für gewichtete Graphen eine ähnliche Schranke für den minimalen Schnitt beweisen². Offensichtlich gilt $2|E| = \sum_{v \in V} |d(v)| \geq |V| \min_{v \in V} |d(v)|$. Deshalb ist $\frac{2|E|}{|V|} \geq \min_{v \in V} |d(v)|$. Also findet der deterministische Algorithmus, der den Knoten v mit kleinstem Grad im Graphen bestimmt und den Schnitt $(v, V \setminus \{v\})$ zurückgibt, immer einen Schnitt mit Gewicht kleiner gleich $\frac{2|E|}{|V|}$. Insbesondere ist die einfache obere Schranke aus Lemma 4.1 immer besser als jene aus Lemma 4.3!

4.3 Dritte obere Schranke

Wir führen zuerst den Begriff des lokal minimalen Schnittes ein. Einen Schnitt $(S, \bar{S})$ nennt man lokal minimal, falls sich bei einem Seitenwechsel eines Knotes das Gewicht des Schnittes nicht verkleinert. Das heisst,

$$\begin{aligned} \forall v \in S & : w(S, V \setminus S) \leq w(S \setminus v, V \setminus S \cup v) \\ \forall v \in V \setminus S & : w(S, V \setminus S) \leq w(S \cup v, V \setminus S \setminus v). \end{aligned}$$

Die Menge aller Graphen lässt sich in zwei Klassen unterteilen: Graphen in denen ein lokal minimaler Schnitt existiert und Graphen für die dies nicht gilt. Für die zweite Klasse ist der minimale Schnitt ein trivialer Schnitt, das heisst mindestens eine der Teilmengen des Schnittes hat Grösse eins. Der Grund ist folgender: Angenommen es existiert ein minimaler Schnitt, so dass beide Teilmengen Grösse mindestens zwei haben. Nach Voraussetzung ist der minimale Schnitt nicht lokal minimal. Dann existiert ein Knoten für den ein Seitenwechsel den Schnitt verkleinert und der neue Schnitt ein zulässiger Schnitt ist (keine Teilmenge leer). Dies ist ein Widerspruch. Für die erste Klasse von Graphen, ist der minimale Schnitt entweder lokal minimal oder ein trivialer Schnitt.

²Sei G ein gewichteter Graph mit $|V|$ Knoten und positiven Kantengewichten, dessen minimaler Schnitt Gewicht μ hat. Offensichtlich gilt dann

$$2 \sum_{(u,v) \in E} w(u,v) = \sum_{u \in V} \sum_{v \in d(u)} w(u,v) \geq \sum_{v \in V} \mu = |V|\mu.$$

Die Summe der Gewichte aller Kanten im Graphen ist also mindestens $\mu \frac{|V|}{2}$ gross. Ist diese Abschätzung bestmöglich? Das heisst, gibt es für jedes μ und $|V|$ einen positiven kantengewichteten Graphen mit $|V|$ Knoten, minimalen Schnittkosten μ und Gesamtgewicht $\mu \frac{|V|}{2}$? Wie man sich leicht selbst überzeugt, erfüllt der vollständige Graph mit den Kantengewichten $\frac{\mu}{|V|-1}$ diese Anforderungen.

Lemma 4.4 Sei $G = (V, E)$ ein Multigraph. Ein minimaler Schnitt $(S, \bar{S})$ in G ist entweder lokal minimal oder ein trivialer Schnitt.

Lemma 4.5 Sei $G = (V, E)$ ein Multigraph. Alle lokal minimalen Schnitte $(S, \bar{S})$ in G haben Gewicht $\leq \frac{|E|}{2}$.

Beweis: Sei (V_1, V_2) ein lokal minimaler Schnitt in $G = (V, E)$, d.h. ein Wechsel eines Knotens $v \in V_1$ nach V_2 oder umgekehrt bringt keinen kleineren Schnitt. Wir definieren

$$\begin{aligned} m_c &= |\{(u, v) \in E \mid u \in V_1, v \in V_2\}| \\ m_1 &= |\{(u, v) \in E \mid u, v \in V_1\}| \\ m_2 &= |\{(u, v) \in E \mid u, v \in V_2\}| \end{aligned}$$

Es gilt dann $|E| = m_c + m_1 + m_2$. Weiter definieren wir für alle $u \in V$

$$\begin{aligned} m_u^1 &= |\{v \in V_1 \mid (u, v) \in E\}| \\ m_u^2 &= |\{v \in V_2 \mid (u, v) \in E\}| \end{aligned}$$

Für alle $u \in V_1$ gilt $m_u^1 \geq m_u^2$ und für alle $u \in V_2$ gilt $m_u^2 \geq m_u^1$, da der Schnitt lokal optimal ist. Damit folgt

$$\begin{aligned} 2m_1 &= \sum_{u \in V_1} m_u^1 \geq \sum_{u \in V_1} m_u^2 = m_c \\ 2m_2 &= \sum_{u \in V_2} m_u^2 \geq \sum_{u \in V_2} m_u^1 = m_c \end{aligned}$$

Wir haben also gezeigt $2m_1 \geq m_c$ und $2m_2 \geq m_c$ und somit $m_1 + m_2 \geq m_c$. Addiert man auf beiden Seiten der letzten Gleichung m_c und ist μ das Gewicht des kleinsten Schnittes in G so gilt

$$|E| = m_1 + m_2 + m_c \geq 2m_c \geq 2\mu.$$

Die letzte Ungleichung gilt, da die Kosten μ des minimalen Schnittes kleiner als die Kosten m_c des lokal minimalen Schnittes sind. $\square$

Lemma 4.6 Sei $G = (V, E)$ ein Multigraph mit einem lokalen Minimum und sei μ das Gewicht eines minimalen Schnittes in G , so gilt $\mu \leq \frac{|E|}{2}$.

Offensichtlich folgt aus Lemma 4.5 sofort Lemma 4.6, weil ein minimaler Schnitt immer kleiner als ein lokal optimaler Schnitt ist. Da es in Graphen mit zwei und drei Knoten keine lokalen Minima gibt und $\frac{|E|}{2} \geq \frac{2|E|}{|V|}$ gilt, falls $|V| \geq 4$, ist die neue Schranke immer schlechter als jene in 4.3. Die neue Schranke hat also keinen praktischen Nutzen als obere Schranke für den minimalen Schnitt. Allerdings kann man mit denselben Überlegungen eine sehr gute unteren Schranke für das *max-Cut* Problem herleiten. Man bemerke, dass ein maximaler Schnitt auch ein lokal maximaler Schnitt ist und deshalb in jedem Multigraphen ein lokal maximaler Schnitt existiert.

4.4 Approximationsalgorithmus für *max-Cut*

Lemma 4.7 Sei $G = (V, E)$ ein Multigraph und μ das Gewicht eines lokal maximalen Schnittes in G , so gilt $\mu \geq \frac{|E|}{2}$.

Beweis: Sei (V_1, V_2) ein lokal maximaler Schnitt in $G = (V, E)$, d.h. ein Wechsel eines Knotens $v \in V_1$ nach V_2 oder umgekehrt bringt keinen grösseren Schnitt. Wir definieren wiederum

$$\begin{aligned} m_c &= |\{(u, v) \in E \mid u \in V_1, v \in V_2\}| \\ m_1 &= |\{(u, v) \in E \mid u, v \in V_1\}| \\ m_2 &= |\{(u, v) \in E \mid u, v \in V_2\}| \end{aligned}$$

Es gilt dann $|E| = m_c + m_1 + m_2$. Weiter definieren wir für alle $u \in V$

$$\begin{aligned} m_u^1 &= |\{v \in V_1 \mid (u, v) \in E\}| \\ m_u^2 &= |\{v \in V_2 \mid (u, v) \in E\}| \end{aligned}$$

Für alle $u \in V_1$ gilt $m_u^1 \leq m_u^2$ und für alle $u \in V_2$ gilt $m_u^2 \leq m_u^1$. Damit folgt

$$\begin{aligned} 2m_1 &= \sum_{u \in V_1} m_u^1 \leq \sum_{u \in V_1} m_u^2 = m_c \\ 2m_2 &= \sum_{u \in V_2} m_u^2 \leq \sum_{u \in V_2} m_u^1 = m_c \end{aligned}$$

Wir haben also gezeigt $2m_1 \leq m_c$ und $2m_2 \leq m_c$ und somit $m_1 + m_2 \leq m_c$. Addiert man auf beiden Seiten der letzten Gleichung m_c und ist μ das Gewicht des grössten Schnittes in G , so gilt

$$|E| = m_1 + m_2 + m_c \leq 2m_c.$$

Da der Schnitt mit maximalem Gewicht höchstens alle Kanten enthalten kann und falls μ das Gewicht eines maximalen Schnittes ist, folgt

$$\mu \leq |E| \leq 2m_c.$$

□

Wir beschreiben einen Algorithmus, der einen Schnitt in einem Graphen findet, der mindestens halb so gross ist, wie der maximale Schnitt im Graphen. Ein solcher Algorithmus heisst $\frac{1}{2}$ -Approximationsalgorithmus, falls er polynomielle Laufzeit hat. Der Algorithmus startet mit einem zufälligen Schnitt. In jeder Phase wird in Zeit $\mathcal{O}(|E|)$ der Knoten bestimmt, der das Gewicht des momentanen Schnittes durch einen Seitenwechsel erhöhen kann. Oder man stellt fest, dass es keinen solchen Knoten gibt. Da höchstens $|E|$ Kanten im maximalen Schnitt liegen und der Schnitt in jeder Phase um mindestens 1 wächst, gibt es höchstens $|E|$ Phasen. Wenn der Algorithmus terminiert, ist der berechnete Schnitt lokal maximal. Wegen Lemma 4.7 ist der lokal maximale Schnitt mindestens halb so gross wie $|E|$. Da der maximale Schnitt höchstens Gewicht $|E|$ besitzt, ist der gefundene Schnitt also mindestens halb so gross wie der maximale Schnitt. Die Gesamtlaufzeit des Algorithmus ist $\mathcal{O}(|E|^2)$, also ein Polynom in der Eingabegrösse.

Theorem 4.1 *Max-Cut besitzt einen $\frac{1}{2}$ -Approximationsalgorithmus.*

Kapitel 5

Minimale s - t Schnitte

Im zweiten Kapitel haben wir einen *min-Cut* Algorithmus beschrieben, der in $|V| - 1$ Phasen einen minimalen Schnitt in einem Graphen findet. Dabei wird in jeder Phase ein minimaler s - t Schnitt in einem anderen Graphen berechnet und am Ende der Phase die Knoten s und t verschmolzen. Also findet man einen minimalen Schnitt mit $|V| - 1$ minimalen s - t Schnittberechnungen und $|V| - 1$ Verschmelzoperationen. Angenommen, man will für einen Graphen einen minimalen Schnitt nur mit Hilfe von minimalen s - t Schnitt Berechnungen bestimmen. Wieviele solcher Berechnungen wird man mindestens benötigen? Wir zeigen, dass man nicht mit weniger als $|V| - 1$ minimalen s - t Schnittberechnungen auskommt und zeigen gleichzeitig, dass $|V| - 1$ s - t Schnittberechnungen genügen, um das Gewicht des minimalen Schnittes zu bestimmen.

Wieviele minimale s - t Schnitte verschiedener Grösse gibt es? Wir werden auf zwei Arten beweisen, dass es höchstens $|V| - 1$ im Gewicht verschieden minimale s - t Schnitte gibt. Für den zweiten Beweis benötigen wir dazu den Begriff des *schnittäquivalenten Baumes*. Ein Baum $B = (V, E')$ ist *schnittäquivalent* zu einem Graphen $G = (V, E)$, wenn für alle Knotenpaare $(s, t) \in V \times V$ das Gewicht eines minimalen s - t Schnittes im Baum und im Graphen gleich ist. Wir zeigen, dass jeder Graph *schnittäquivalent* zu einem Baum ist. Der Baum B kann mit Hilfe des Mehrfachflussalgorithmus von Gomory und Hu [GH61] in Zeit $\mathcal{O}(|E||V| \log \frac{|V|^2}{|E|})$ berechnet werden. Dazu benutzt man das *max-Flow min-Cut* Theorem von Ford und Fulkerson ¹.

¹Gomory und Hu haben gezeigt, wie man für ein Netzwerk $N = (V, E, c)$ ein flussäquivalentes Baumnetzwerk B in Zeit $\mathcal{O}(|V||E| \log(|V|^2/|E|))$ konstruiert. Da der Wert eines

5.1 Das Gewicht minimaler s - t Schnitte

Sei G ein ungerichteter, gewichteter Graph mit Kantenmenge E und Knotenmenge V . Für Knoten $s, t \in V$ und $s \neq t$ bezeichne $C(s, t)$ das Gewicht eines minimalen s - t Schnittes. Lemma 5.1 und 5.2 bilden den Kern des Beweises, dass zu jedem Graphen G ein schnittäquivalenter Baum T existiert.

Lemma 5.1 *Seien u, v und w paarweise verschiedene Knoten aus V , dann gilt*

$$C(u, v) \geq \min(C(u, w), C(w, v))$$

Beweis: Sei $S, \bar{S}$ ein minimaler u - v Schnitt. Dann gilt für alle $w \in V$ entweder $w \in S$ oder $w \in \bar{S}$. Im ersten Fall ist $S, \bar{S}$ auch ein w - v Schnitt und damit

$$C(w, v) \leq C(u, v) \quad (5.1)$$

Im zweiten Fall ist $S, \bar{S}$ auch ein u - w Schnitt und damit

$$C(u, w) \leq C(u, v). \quad (5.2)$$

Da wenigstens eine der Ungleichungen (5.1) oder (5.2) erfüllt sein muss, folgt $\min(C(u, w), C(w, v)) \leq C(u, v)$. $\square$

Lemma 5.2 *Sei $v_1, \dots, v_k$ eine Folge von paarweise verschiedenen Knoten aus V und $k \geq 3$, dann gilt*

$$C(v_1, v_k) \geq \min(C(v_1, v_2), C(v_2, v_3), \dots, C(v_{k-1}, v_k)) \quad (5.3)$$

Beweis: Wir beweisen (5.3) durch Induktion über die Anzahl Knoten der Folge. Nach Lemma 5.1 gilt die Behauptung für Knotenfolgen der Länge drei. Angenommen die Behauptung stimmt für Knotenfolgen der Länge kleiner k . Wir zeigen, dass die Behauptung auch für Knotenfolge der Länge k gilt.

$$\begin{aligned} C(v_1, v_k) &\stackrel{\text{I.A.}}{\geq} \min(C(v_1, v_2), C(v_2, v_k)) \\ &\stackrel{\text{I.A.}}{\geq} \min(C(v_1, v_2), \min(C(v_2, v_3), C(v_3, v_4), \dots, C(v_{k-1}, v_k))) \\ &\geq \min(C(v_1, v_2), C(v_2, v_3), \dots, C(v_{k-1}, v_k)) \end{aligned}$$

maximalen s - t Flusses in G gleich dem Gewicht eines minimalen s - t Schnittes in N ist, ist B auch schnittäquivalent zu N .

Dabei gelten die ersten beiden Ungleichungen wegen der Induktionsannahme.

□

Angenommen alle k Werte $C(v_1, v_k), C(v_1, v_2), \dots, C(v_{k-1}, v_k)$ wären verschieden. Dann dürfen wir ohne Einschränkung annehmen, dass $C(v_1, v_k)$ der kleinste Wert ist. Folglich gilt $C(v_1, v_k) < \min(C(v_1, v_2), \dots, C(v_{k-1}, v_k))$, was jedoch nach Lemma 5.2 ein Widerspruch ist. Also gilt Korollar 5.1. Mit Hilfe von diesem Korollar können wir nun leicht zeigen, dass es in einem Graphen $G = (V, E)$ höchstens $|V| - 1$ minimale s - t Schnitte verschiedener Grösse gibt.

Korollar 5.1 *Sei $v_1, \dots, v_k$ eine Folge von paarweise verschiedenen Knoten aus V und $k \geq 3$, dann gilt*

$$|\{C(v_1, v_k), C(v_1, v_2), C(v_2, v_3), \dots, C(v_{k-1}, v_k)\}| < k.$$

Theorem 5.1 *Sei $G = (V, E, w)$ ein ungerichteter, gewichteter Graph, dann gibt es höchstens $|V| - 1$ im Wert verschiedene minimale s - t Schnitte.*

Beweis: Sei $G^* = (V, E^*, w^*)$ ein vollständiger Graph, so dass das Gewicht einer Kante $(u, v) \in E^*$ dem Gewicht eines minimalen u - v Schnittes in G entspricht, d.h.

$$\forall (u, v) \in E^* : w^*(u, v) = C(u, v). \quad (5.4)$$

Sei $\hat{G} = (V, \hat{E}, w^*)$ ein Untergraph von G^* , so dass $\hat{G}$ nur Kanten mit unterschiedlichem Kantengewicht enthält. Angenommen $\hat{G}$ enthält mehr als $|V| - 1$ Kanten, das heisst $|\hat{E}| > |V| - 1$. Dann enthält $\hat{G}$ einen Zyklus Z . Seien $v_1, \dots, v_k$ die Knoten dieses Zyklus, wobei $k \leq |V|$ ist. Nach Definition von $\hat{G}$ und G^* gilt

$$|\{C(v_1, v_k), C(v_1, v_2), C(v_2, v_3), \dots, C(v_{k-1}, v_k)\}| = k.$$

Nach Korollar 5.1 muss jedoch gelten

$$|\{C(v_1, v_k), C(v_1, v_2), C(v_2, v_3), \dots, C(v_{k-1}, v_k)\}| < k.$$

Damit war die Annahme falsch und $\hat{G}$ bzw. G^* besitzt höchstens $|V| - 1$ Kanten mit verschiedenem Gewicht. Folglich gibt es in G höchstens $|V| - 1$ minimale s - t Schnitte verschiedener Grösse. □

5.2 Schnittäquivalente Bäume

Theorem 5.2 *Ein ungerichteter, gewichteter Graph $G = (V, E_G, w)$ ist immer schnittäquivalent zu einem Baum $T = (V, E_T, w^*)$.*

Beweis: Sei $G^* = (V, E^*, w^*)$ ein vollständiger Graph, so dass das Gewicht einer Kante $(u, v) \in E^*$ dem Gewicht eines minimalen u - v Schnittes in G entspricht, d.h.

$$\forall (u, v) \in E^* : w^*(u, v) = C(u, v). \quad (5.5)$$

Der Baum T sei ein teuerster aufspannender Baum in G^* . Somit ist $E_T \subseteq E^*$. Es seien u und v zwei verschiedene Knoten aus V . Folglich sind u und v in T durch einen eindeutigen Weg

$$\mathcal{W} : u = v_1, \dots, v_k = v$$

verbunden. Es sei (v_l, v_{l+1}) diejenige Kante auf dem Weg $\mathcal{W}$ mit kleinstem Gewicht, d.h. $w^*(v_l, v_{l+1}) = \min\{w^*(v_j, v_{j+1}) \mid 1 \leq j \leq k-1\}$. Wir müssen zeigen, dass

$$C(v_1, v_k) = w^*(v_l, v_{l+1}).$$

Hat der Weg $\mathcal{W}$ Länge eins so ist die Behauptung offensichtlich richtig. Sei also $\mathcal{W}$ ein Weg der Länge > 1 und damit $k \geq 3$. Dann gilt

$$\begin{aligned} C(v_1, v_k) &\stackrel{\text{Lemma 5.2}}{\geq} \min(C(v_1, v_2), \dots, C(v_{k-1}, v_k)) \\ &\stackrel{(5.5)}{=} \min(w^*(v_1, v_2), \dots, w^*(v_{k-1}, v_k)) \\ &= w^*(v_l, v_{l+1}). \end{aligned}$$

Angenommen $C(v_1, v_k) > w^*(v_l, v_{l+1})$ dann könnte man allerdings die Kante (v_l, v_{l+1}) in T durch die Kante (v_1, v_k) ersetzen und man erhielte einen Baum $\hat{T}$ mit grösseren Kosten als T , im Widerspruch zur Definition von T . Also gilt $C(v_1, v_k) = w^*(v_l, v_{l+1})$. Da $w^*(v_l, v_{l+1})$ das Gewicht eines minimalen u - v Schnittes in T ist, folgt die Behauptung. $\square$

Der Baum $T = (V, E_T, w^*)$ hat $|V| - 1$ Kanten. Weil das Gewicht eines minimalen s - t Schnittes im Baum T immer dem Gewicht einer Kante entspricht, gibt es in T höchstens $|V| - 1$ minimale s - t Schnitte mit unterschiedlichem Gewicht. Da T schnittäquivalent mit G ist, gibt es auch in G höchstens $|V| - 1$ minimale s - t Schnitte unterschiedlicher Grösse.

5.3 Kleinste minimale s - t Schnitte

Die kleinsten minimalen s - t Schnitte sind auch minimale Schnitte. Man kann also einen minimalen Schnitt für einen Graphen $G = (V, E)$ finden, indem man in G alle möglichen $\binom{|V|}{2}$ minimalen s - t Schnitte berechnet. Der kleinste minimale s - t Schnitt ist dann der gesuchte minimale Schnitt. Wir haben in Kapitel zwei gezeigt, dass sogar $|V| - 1$ minimale s - t Schnittberechnungen genügen, um den minimalen Schnitt zu finden. Wir wollen diese Aussage nochmals auf eine andere sehr einfache Art beweisen.

Lemma 5.3 *Ein minimaler Schnittes in $G = (V, E)$ kann durch höchstens $|V| - 1$ minimale s - t Schnittberechnungen in G gefunden werden.*

Beweis: Sei $n = |V|$ und $v_1, \dots, v_n$ die Knotenmenge von G . Bezeichne $C(s, t)$ wiederum das Gewicht eines minimalen s - t Schnittes. Dann gilt

$$\min_{s, t \in V} C(s, t) = \min(C(v_1, v_2), \dots, C(v_{n-1}, v_n)). \quad (5.6)$$

Angenommen die Gleichung 5.6 würde nicht gelten. Dann würden Knoten v_l und v_m in V existieren, so dass

$$C(v_l, v_m) < \min(C(v_1, v_2), \dots, C(v_{n-1}, v_n)) \quad (5.7)$$

$$\leq \min(C(v_l, v_{l+1}), \dots, C(v_{m-1}, v_m)). \quad (5.8)$$

Dies widerspricht jedoch Lemma 5.2 und deshalb gilt 5.6. Da $\min_{s, t \in V} C(s, t)$ das Gewicht eines minimalen Schnittes ist und für die Berechnung der rechten Seite von 5.6 nur $n - 1$ minimale s - t Schnittberechnungen benötigt werden, folgt das Lemma 5.3. $\square$

Eine interessante Frage ist es, ob das *min-Cut* Problem mit weniger als $|V| - 1$ minimalen s - t Schnittberechnungen gelöst werden kann. Wir werden zeigen, dass dies nicht geht. Da der beste bis heute bekannte *min-Cut* Algorithmus [HO92] für $|V| - 1$ minimale Schnittberechnungen Zeit $\mathcal{O}(|E||V| \log(|V|^2/|E|))$ benötigt, brauchen somit alle bekannten Algorithmen $\Omega(|E||V| \log(|V|^2/|E|))$ Zeit.

Lemma 5.4 *Für die Berechnung eines minimalen Schnittes in $G = (V, E)$ benötigt ein s - t Schnitt basierter Algorithmus mindestens $|V| - 1$ minimale s - t Schnitte.*

Beweis: Sei O ein Orakel, das für einen Graphen $G = (V, E)$ und zwei Knoten s und t in G einen minimalen s - t Schnitt findet. Sei A ein Algorithmus, der nur durch Fragen an das Orakel O einen minimalen Schnitt in G berechnet. Eine Frage von Algorithmus A ist also ein s, t Knotenpaar und eine Antwort des Orakels ein minimaler s - t Schnitt. Wir werden zeigen, dass Algorithmus A mindestens $|V| - 1$ Fragen stellen muss. Angenommen es gibt einen Algorithmus A^* , der mit $l < |V| - 1$ Fragen einen minimalen Schnitt mit Gewicht c^* findet. Wir konstruieren einen Graphen $\hat{G}$ mit minimalem Schnitt $\hat{c} - \epsilon < c^*$, für den Algorithmus A^* einen minimalen Schnitt mit Gewicht c^* berechnet. Dies ist ein Widerspruch zur Definition von A^* . O.E. können wir annehmen, dass A^* bei Eingabe von G^* die Fragen

$$(v_1, v_2), \dots, (v_l, v_{l+1})$$

stellt. Aus den Antworten des Orakels O konstruieren wir den Graphen $\hat{G}$, der eine Kette $v_1, \dots, v_n$ ist und die minimalen s - t Schnitte als Kantengewichte hat.

$$\begin{aligned} w(v_i, v_{i+1}) &:= C(v_i, v_{i+1}) \text{ für } i \in \{1, \dots, l\} \\ w(v_j, v_{j+1}) &:= c^* - \epsilon \quad \text{für } j \in \{l+1, \dots, n-1\}. \end{aligned}$$

Algorithmus A^* behandelt $\hat{G}$ wie G^* . □

Kapitel 6

Maximale Flüsse und minimale Schnitte

Da viele Algorithmen zum Berechnen minimaler s - t Schnitte einen maximalen s - t Fluss berechnen, wollen wir der Vollständigkeit halber das *max-Flow* Problem vorstellen und den Zusammenhang zum *min-Cut* Problem diskutieren. Die Eingabe für das *max-Flow* Problem ist ein spezieller Graph, den wir Netzwerk nennen. Ein Netzwerk $N = (V, E, s, t, c)$ ist ein gerichteter, gewichteter Graph (V, E, c) mit zwei ausgezeichneten Knoten s (der Quelle) und t (der Senke). Für jede Kante $e \in E$ ist die Kapazität $c_e > 0$ gegeben. Ein Fluss f in N ist eine Zuweisung von nichtnegativen Zahlen f_e für jede Kante e im Netzwerk, so dass

1. für alle Kanten $e \in E$

$$f_e \leq c_e \tag{6.1}$$

2. und für alle Knoten $v \in V$, ausser s und t ,

$$\sum_{e \in \delta(v)} f_e = \sum_{e \in \delta(\bar{v})} f_e. \tag{6.2}$$

gilt. Dabei bezeichnet $\delta(v)$ die Menge der Kanten, die v als Anfangsknoten besitzen und $\delta(\bar{v})$ die Menge der Kanten, die v als Endknoten besitzen.

Der Wert $|f|$ eines Flusses f ist die Summe der f 's derjenigen Kanten die s verlassen minus der Summe der f 's jener Kanten, die in s enden, d.h

$$|f| := \sum_{e \in \delta(s)} f_e - \sum_{e \in \delta(\bar{s})} f_e. \tag{6.3}$$

Das *max-Flow Problem* lautet nun: Gegeben ein Netzwerk N , finde einen Fluss f mit grösstmöglichem Wert. So ein f heisst *maximaler Fluss*.

Man kann sich das *max-Flow Problem* für ein Netzwerk $N = (V, E, s, t, c)$ so vorstellen: Knoten s ist eine Wasserquelle und Knoten t ein Brunnen. Das Netzwerk N kann man nun als Leitungsnetz interpretieren, das die Quelle mit dem Brunnen verbindet. Eine Kante e im Netzwerk ist eine Leitung mit maximaler Durchflussmenge c_e pro Zeiteinheit. Die Knoten ausser s und t sind Kreuzungspunkte der Leitungen. Wieviel Wasser kann pro Zeiteinheit maximal von s nach t fließen?

Wir werden zeigen, wie man das *max-Flow Problem* in Zeit $\mathcal{O}(|V|^5)$ lösen kann. Es gibt viel schnellere Algorithmen für dieses Problem. Zum Beispiel der Preflow-Push Algorithmus, der Laufzeit $\mathcal{O}(|V|^2|E|)$ besitzt, was für Netzwerke mit wenigen Kanten viel besser ist. Wir werden den Algorithmus jedoch nur dazu brauchen, um das *max-Flow min-Cut Theorem* zu beweisen und deshalb spielt für uns die Laufzeit keine Rolle.

Ein *Schnitt* $(S, \bar{S})$ in einem Netzwerk $N = (V, E, s, t, c)$ ist eine Partition der Knotenmenge V in eine Teilmenge S von V und die Menge $\bar{S} = V \setminus S$, wobei s in S und t in $\bar{S}$ liegt.

Die Menge der *Schnittkanten* $\delta(S)$ ist die Menge der Kanten mit Anfangsknoten in S und Endknoten in $\bar{S}$, d.h.

$$\delta(S) := \{e \in E \mid \text{tail}(e) \in S \text{ head}(e) \in \bar{S}\}. \quad (6.4)$$

Die *Kapazität* $C(S, \bar{S})$ eines Schnittes $(S, \bar{S})$ ist definiert als die Summe der Kapazitäten jener Kanten, die im Schnitt liegen, d.h.

$$C(S, \bar{S}) := \sum_{e \in \delta(S)} c_e \quad (6.5)$$

Ein Schnitt im Netzwerk mit kleinster Kapazität heisst *minimaler Schnitt*. Wir werden zeigen, dass in einem Netzwerk der Wert des maximalen Flusses gleich der Kapazität eines minimalen Schnittes ist. Zudem werden wir beweisen, dass jeder Algorithmus, der einen maximalen Fluss berechnet, auch einen minimalen Schnitt bestimmt. Das heisst, das *max-Flow Problem* ist mindestens so schwer wie das *min-Cut Problem*.

6.1 Das Problem des maximalen Flusses

Gegeben ein Netzwerk $N = (V, E, s, t, c)$ gesucht ist ein Fluss f mit grösstem Wert. Offensichtlich ist dieses Problem ein Problem der linearen Programmierung mit Variablen f_e für $e \in E$ und σ für den Wert des Flusses. Das lineare Programm sieht also wie folgt aus:

$$\begin{aligned} & \text{maximiere } \sigma \\ & \sum_{e \in \delta(v)} f_e - \sum_{e \in \delta(\bar{v})} f_e = \begin{cases} \sigma & \text{falls } v = s \\ 0 & \text{falls } v \in V \setminus \{s, t\} \\ -\sigma & \text{falls } v = t \end{cases} \\ & 0 \leq f_e \leq c_e \quad \text{für alle } e \in E. \end{aligned}$$

Dieses lineare Programm könnte man mit dem Simplexverfahren lösen. Allerdings ist es effizienter, die Lösung direkt mit einem kombinatorischen Algorithmus zu bestimmen.

Wir zeigen, dass für einen Fluss f und einen Schnitt $(S, \bar{S})$ in einem Netzwerk, $|f| \leq C(S, \bar{S})$ gilt. Findet man also einen Fluss f und einen Schnitt $(S, \bar{S})$ im Netzwerk, mit $|f| = C(S, \bar{S})$, so ist der Fluss f maximal und der Schnitt $(S, \bar{S})$ minimal.

Lemma 6.1 *Sei f ein beliebiger Fluss im Netzwerk $N = (V, E, s, t, c)$ und $(S, \bar{S})$ ein beliebiger Schnitt, dann gilt*

$$\begin{aligned} |f| &= \sum_{e \in \delta(S)} f_e - \sum_{e \in \delta(\bar{S})} f_e \\ |f| &\leq C(S, \bar{S}) \end{aligned}$$

Beweis: Sei f ein beliebiger Fluss im Netzwerk N und $(S, \bar{S})$ ein beliebiger Schnitt. Addiert man alle Gleichungen 6.2 für $v \in S$, dann gilt:

$$\begin{aligned} |f| &\stackrel{6.3}{=} \sum_{v \in S} \left\{ \sum_{e \in \delta(v)} f_e - \sum_{e \in \delta(\bar{v})} f_e \right\} \\ &= \sum_{v \in S} \left\{ \sum_{\substack{t(e)=v \\ h(e) \in S}} f_e + \sum_{\substack{t(e)=v \\ h(e) \in \bar{S}}} f_e - \sum_{\substack{h(e)=v \\ t(e) \in S}} f_e - \sum_{\substack{h(e)=v \\ t(e) \in \bar{S}}} f_e \right\} \end{aligned}$$

$$\begin{aligned}
 &= \sum_{v \in S} \underbrace{\left\{ \sum_{\substack{t(e)=v \\ h(e) \in S}} f_e - \sum_{\substack{h(e)=v \\ t(e) \in S}} f_e \right\}}_{\stackrel{(*)}{=} 0} + \sum_{v \in S} \underbrace{\left\{ \sum_{\substack{t(e)=v \\ h(e) \in \bar{S}}} f_e - \sum_{\substack{h(e)=v \\ t(e) \in \bar{S}}} f_e \right\}}_{\stackrel{6.4}{=} \sum_{e \in \delta(S)} f_e - \sum_{e \in \delta(\bar{S})} f_e}
 \end{aligned}$$

Dabei bezeichnet $t(e)$ den Anfangsknoten und $h(e)$ den Endknoten für jede Kante $e \in E$. Zur Gleichung (*): Es kommen nur solche f_e 's in der Summe vor, für welche die Kante e beide Endknoten in S hat. Jedes f_e tritt genau zweimal auf, einmal positiv und einmal negativ. Es folgt

$$\begin{aligned}
 |f| &= \sum_{e \in \delta(S)} f_e - \sum_{e \in \delta(\bar{S})} f_e \\
 &\stackrel{6.1}{\leq} \sum_{e \in \delta(S)} c_e \\
 &\stackrel{6.5}{=} C(S, \bar{S}).
 \end{aligned}$$

□

Nun sehen wir einen Augenblick von den Richtungen auf den Kanten des Netzwerkes N ab und betrachten einen Weg W von s nach t . Ist das Netzwerk N wiederum gerichtet, so heisst eine Kante e des Weges *Vorwärtskante*, falls die Richtung von e in N gleich der Richtung von e in W ist, wenn man W von s nach t abschreitet, sonst *Rückwärtskante*.

Sei f ein Fluss im Netzwerk N . Ein *flussvergrößernder Weg* W bezüglich f ist ein Weg von s nach t , so dass

$$f_e < c_e \text{ für alle Vorwärtskanten } e \text{ in } W \quad (6.6)$$

$$f_e > 0 \text{ für alle Rückwärtskanten } e \text{ in } W. \quad (6.7)$$

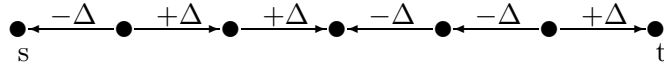
Existiert ein flussvergrößernder Weg W bezüglich f und ist $\Delta := \min(\{c_e - f_e \mid e \text{ ist Vorwärtskante in } W\} \cup \{f_e \mid e \text{ ist Rückwärtskante in } W\})$. Dann ist f' definiert durch

$$f'_e := \begin{cases} f_e + \Delta & , \text{ falls } e \text{ Vorwärtskanten in } W \\ f_e - \Delta & , \text{ falls } e \text{ Rückwärtskanten in } W \\ f_e & , \text{ falls } e \text{ nicht in } W \end{cases}$$

wieder ein Fluss mit Wert $|f| + \Delta$. Wir haben also einen Fluss mit grösserem Wert gefunden, denn $\Delta > 0$. Man sagt f sei mit Hilfe des flussvergrößernden Weges W vergrössert worden.

Vollständigkeitshalber zeigen wir noch formal, dass f' tatsächlich ein Fluss mit Wert $|f| + \Delta$ ist. Wir zeigen zuerst, dass der Fluss f' die Bedingung $0 \leq f'_e \leq c_e$ für alle Kanten $e \in E$ erfüllt. Ist $e \notin W$ dann ist $f'_e = f_e$ und die Behauptung ist offensichtlich richtig. Ist andererseits $e \in W$ so ist e entweder eine Vorwärts- oder Rückwärtskante. Sei e eine Vorwärtskante dann gilt $0 \leq f_e + \Delta \leq f_e + (c_e - f_e) = c_e$. Ist e dagegen eine Rückwärtskante dann ist $c_e \geq f_e - \Delta \geq f_e - f_e = 0$. Damit erfüllt f' die Bedingung 6.1. Als nächstes zeigen wir, dass f' auch die Bedingung 6.2 erfüllt. Die Bedingung 6.2 wird offenbar von Knoten erfüllt die nicht auf W liegen, denn f ist ein Fluss. Es bleibt zu zeigen, dass Knoten die auf dem Weg W liegen die Bedingung 6.2 erfüllen. Sei v ein solcher Knoten. Dann ist v mit genau zwei Kanten des Weges W inzident. Nun gibt es vier Möglichkeiten wie v mit diesen beiden Kanten verbunden ist. Man überzeugt sich leicht selbst, mit Hilfe von Abbildung 6.1, dass die Bedingung 6.2 immer erfüllt ist.

Abbildung 6.1: Flussvergrößernde Wege



Existiert also in einem Netzwerk N ein flussvergrößernder Weg W bezüglich eines Flusses f , so ist f nicht maximal. Wir werden zeigen, dass die Umkehrung auch gilt.

Lemma 6.2 *Sei $N = (V, E, s, t, c)$ ein Netzwerk. Ein Fluss f ist maximal, genau dann wenn für f kein flussvergrößernder Weg in N existiert.*

Beweis: Ist f ein maximaler Fluss, so gibt es offensichtlich keinen flussvergrößernden Weg, denn sonst wäre f nicht maximal. Umgekehrt nehmen wir an, dass es keinen flussvergrößernden Weg W bezüglich f gibt. Wir werden zeigen, dass f maximal ist. Sei S die Menge aller Knoten $v \in V$ für die ein flussvergrößernder Weg von s nach v bezüglich f existiert. Natürlich ist $t \notin S$. Dann gilt für alle $e \in \delta(S)$ $f_e = c_e$. Ansonsten wäre $head(e) \in S$ und dies wäre ein Widerspruch zur Definition von $\delta(S)$. Andererseits gilt für alle $e \in \delta(\bar{S})$ $f_e = 0$, da sonst $tail(e) \in S$. Damit folgt mit Lemma 6.1

$$|f| \stackrel{\text{Lemma 6.1}}{=} \sum_{e \in \delta(S)} f_e - \sum_{e \in \delta(\bar{S})} f_e$$

$$\begin{aligned}
&= \sum_{e \in \delta(S)} c_e - 0 \\
&\stackrel{6.5}{=} C(S, \bar{S}).
\end{aligned}$$

Angenommen es gibt einen Fluss f' mit $|f'| > |f| = C(S, \bar{S})$. Dann würde $C(S, \bar{S}) < |f'|$ gelten, was jedoch ein Widerspruch zu Lemma 6.1 wäre. Also ist f maximal. $\square$

Wir zeigen nun, dass in einem Netzwerk N der Wert eines maximalen Flusses gleich der Kapazität eines minimalen Schnittes ist.

Theorem 6.1 *Sei $N = (V, E, s, t, c)$ ein Netzwerk f ein maximaler Fluss und $(S, \bar{S})$ ein minimaler Schnitt in N , dann gilt $|f| = C(S, \bar{S})$.*

Beweis: In dem Beweis von Lemma 6.2 haben wir gezeigt, dass es einen Schnitt $(S', \bar{S}')$ gibt, sodass $|f| = C(S', \bar{S}')$. Da der Schnitt $C(S, \bar{S})$ minimal ist, gilt $C(S, \bar{S}) \leq C(S', \bar{S}')$. Also folgt $|f| \geq C(S, \bar{S})$. Nach Lemma 6.1 gilt jedoch auch $|f| \leq C(S, \bar{S})$ und damit $|f| = C(S, \bar{S})$. $\square$

6.2 Ein Algorithmus für das *max-Flow* Problem

Die Grundidee bei der Konstruktion eines maximalen Flusses für ein Netzwerk $N = (V, E, s, t, c)$ ist es, den Fluss schrittweise zu erhöhen. Ausgehend vom leeren Fluss ($f_e = 0$ für alle Kanten $e \in E$) vergrößert der Algorithmus wiederholt den jeweiligen Fluss, indem er einen flussvergrößernden Weg bezüglich des momentanen Flusses bestimmt und dann den Fluss mit dessen Hilfe erhöht. Einen flussvergrößernden Weg findet man durch Traversierung des Netzwerks. Bei der Traversierung des Netzwerks muss Kante e in Vorwärtsrichtung bzw. Rückwärtsrichtung durchlaufen werden, falls $f_e < c_e$ bzw. $f_e > 0$ gilt. Findet man keinen flussvergrößernden Weg, dann ist der momentane Fluss nach Lemma 6.2 maximal.

Wir beschreiben nun einen Algorithmus, der für ein Netzwerk $N = (V, E, s, t, c)$ einen maximalen Fluss berechnet. Zur Vereinfachung der Notation beschränken wir uns auf Netzwerke N ohne Parallelkanten, d.h. falls $(i, j) \in E$ dann ist $(j, i) \notin E$. Zudem nehmen wir an, dass die Kantenkapazitäten positive ganze Zahlen sind. Ist f ein Fluss in N , so ist $N(f) = (V, E', s, t, c')$ ein Netzwerk mit Kantenmenge

$$E' := \{(i, j) \mid (i, j) \in E, f(i, j) < c(i, j)\} \cup \{(j, i) \mid (i, j) \in E, f(i, j) > 0\}$$

und Kapazitäten

$$c'(i, j) := \begin{cases} c(i, j) - f(i, j) & , \text{ falls } (i, j) \in E \\ f(j, i) & , \text{ falls } (j, i) \in E \end{cases}$$

für $(i, j) \in E'$. Ein flussvergrößernder Weg von s nach t in N bezüglich f ist offensichtlich ein Weg von s nach t in $N(f)$. Umgekehrt ist ein Weg von s nach t in $N(f)$ ein flussvergrößernder Weg in N . Damit ist ein Fluss f genau dann maximal, wenn kein Weg von s nach t in $N(f)$ existiert. Ob t von s aus in $N(f)$ erreichbar ist, kann man durch Breitensuche in Zeit $\mathcal{O}(|V| + |E'|)$ bestimmen. Somit können wir leicht entscheiden, ob ein Fluss maximal ist. Dies führt zu folgendem Algorithmus für das *max-Flow Problem*:

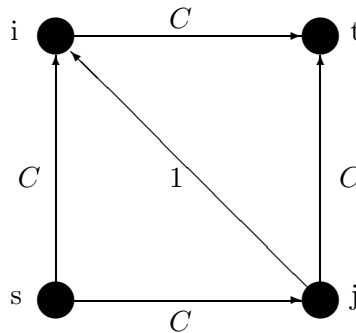
1. Starte mit dem leeren Fluss f in $N = (V, E, s, t, c)$.
2. Konstruiere $N(f)$ und versuche einen Weg, von s nach t in $N(f)$ zu finden.
3. Falls kein Weg von s nach t in $N(f)$ existiert dann ist f maximal und wir sind fertig.
4. Falls ein Weg existiert, finde die Kante mit kleinster Kapazität c' auf diesem Weg. Durchlaufe diesen Weg von s nach t in N und addiere zu jeder Kante e des Weges c' bzw. $-c'$ zu f_e , falls sie vorwärts bzw. rückwärts durchlaufen wird. Das Resultat ist ein grösserer Fluss f' für das Netzwerk N . Setze $f := f'$ und gehe zu 2.

Man bemerke, dass der Wert eines maximalen Flusses immer ganzzahlig ist. Dies sieht man leicht durch Induktion über die Anzahl Iterationen ein. Wir behaupten, dass der Fluss f am Ende einer Iteration immer ganzzahlig ist. Die Behauptung stimmt nach der nullten Iteration, da der leere Fluss immer ganzzahlig ist. Wir nehmen also an, dass der Fluss f nach der k -ten Iteration ganzzahlig ist. Dann besitzt das Netzwerk $N(f)$ nur ganzzahlige Kapazitäten. Falls kein Weg von s nach t in $N(f)$ existiert ist f maximal und die Behauptung bewiesen. Existiert jedoch ein Weg, so ist die kleinste Kapazität c' einer Kante auf diesem Weg ebenfalls ganzzahlig und damit auch der Fluss nach der $(k + 1)$ -ten Iteration.

Wie ist die Laufzeit dieses Algorithmus? In jeder Iteration berechnen wir einen Weg von s nach t und erhöhen den Fluss entlang dieses Weges. Einen Weg finden wir mit einem Besuchsalgorithmus in Zeit $\mathcal{O}(|V| + |E'|)$

und den Fluss können wir in Zeit $\mathcal{O}(|V|)$ erhöhen. Damit ist die Laufzeit einer Iteration beschränkt durch $\mathcal{O}(|V|^2)$. Nun gibt es höchstens $C|V|$ Iterationen, wobei C die maximale Kapazität einer Kante im Netzwerk ist. Der Grund ist dieser: Jede Iteration erhöht den Fluss um mindestens 1, denn die Kapazitäten sind ganze Zahlen und der maximale Fluss ist höchstens $C|V|$, da es weniger als $|V|$ Kanten gibt, die s verlassen. Also gibt es weniger als $C|V|$ Iterationen. Wie Abbildung 6.2 kann ist die Wirklichkeit fast so schlecht: Wenn unser Besuchsalgorithmus immer einen Pfad zurückgibt, der die Kante (i, j) oder Kante (j, i) enthält, werden $2C$ Iterationen gebraucht. Die Laufzeit für diesen Algorithmus ist also $\mathcal{O}(|V|^3 C)$.

Abbildung 6.2: Schlechtes Beispiel



Obwohl diese Laufzeit polynomiell erscheint, ist sie dies nicht. Dies liegt an der linearen Abhängigkeit der Laufzeit von C . Wir nehmen an dass C binär codiert ist, was für Zahlen Sinn macht. Vergrößert sich C um eine Stelle, so verdoppelt sich die Laufzeit aber die Eingabegrösse wächst nur um eine Stelle. Deshalb ist die Laufzeit $\mathcal{O}(|V|^3 C)$ nicht polynomiell sondern wächst exponentiell in der Eingabegrösse. Nehmen wir an, dass wir den Fluss immer entlang des kürzesten Weges von s nach t in $N(f)$ vergrößern, wobei der kürzeste Weg jener mit der kleinsten Anzahl Kanten ist. Eine Kante auf einem Weg von s nach t in $N(f)$ heisst *Flaschenhals*, falls sie die kleinste Kapazität c' besitzt. Man kann zeigen, dass in höchstens $|V|$ Iterationen die Kante (i, j) oder ihre Gegenkante (j, i) ein Flaschenhals ist, falls man den Fluss entlang des kürzesten Weges von s nach t vergrößert. Weil es höchstens $|V|^2$ Kanten gibt und in jeder Iteration mindestens eine Kante zum Flaschenhals wird gibt es höchstens $|V|^3$ Iterationen. Offensichtlich finden wir mit Breitensuche den kürzesten Weg von s nach t in $N(f)$ in Zeit $\mathcal{O}(|V|^2)$. Damit besitzt der Algorithmus Laufzeit $\mathcal{O}(|V|^5)$.

Schlusswort

Wir haben uns nur mit Algorithmen beschäftigt, die immer einen minimalen Schnitt finden und deren Laufzeit nur von der Eingabe abhängt. Es gibt noch zwei weitere Klassen von Schnittalgorithmen. *Monte Carlo Algorithmen* sind randomisierte Algorithmen, die meistens einen minimalen Schnitt, manchmal aber auch einen nicht minimalen Schnitt berechnen. Die Wahrscheinlichkeit, dass dies geschieht, ist jedoch klein. Die Laufzeit ist keine Zufallsvariable. *Monte Carlo Algorithmen* können zum Beispiel verwendet werden, um einen nullten Phasenschnitt, für den verbesserten Algorithmus zu finden. Karger und Stein [KS93] haben einen solchen Algorithmus beschrieben, der Laufzeit $\mathcal{O}(|V|^2 \log |V|)$ besitzt und mit Wahrscheinlichkeit $\Omega(1/\log |V|)$ einen minimalen Schnitt findet. Wiederholt man diesen Algorithmus $\log^c |V|$ mal, so findet man mit sehr hoher Wahrscheinlichkeit einen minimalen Schnitt. Die Gesamtlaufzeit beträgt also $\mathcal{O}(|V|^2 \log^{\mathcal{O}(1)} |V|)$. Man kann sich Schnittalgorithmen vorstellen, die immer einen minimalen Schnitt berechnen, deren Laufzeit jedoch eine Zufallsvariable ist. Randomisierte Algorithmen mit dieser Eigenschaft heissen *Las Vegas Algorithmen*. Eine interessante Frage ist es, ob es einen *Las Vegas Algorithmus* gibt, der eine erwartete Laufzeit von $\mathcal{O}(|V|^2 \log^{\mathcal{O}(1)} |V|)$ besitzt. Bis heute ist kein solcher bekannt. Wegen der Dualität zwischen dem *max-Flow* und *min-Cut* Problem ist jeder *Las Vegas Algorithmus* für das *max-Flow* Problem auch ein solcher für das *min-Cut* Problem.

Literaturverzeichnis

- [FC96] F. R. K. CHUNG *Laplacians of Graphs and Cheeger's inequalities* Combinatorics 2(1996), 157-172.
- [FT87] M. L. FREDMAN AND TARJAN, *Fibonacci heaps and their uses in improved network optimization algorithms*, Journal of the ACM 34(1987), 596-615.
- [FF56] L. R. FORD AND D. R. FULKERSON, *Maximal flow through a network*, Canadian Journal of Mathematics 8(1956), 399-404.
- [GT88] A. V. GOLDBERG AND R. E. TARJAN, *A new approach to the maximum flow problem*, Journal of the ACM 35(1988), 921-940.
- [GH61] R. E. GOMORY, *Multi-terminal network flows*, Journal of the SIAM 9(1961) 551-570.
- [HO92] X. HAO AND J. B. ORLIN, *A faster algorithm for finding the minimum cut in a graph*, 3rd ACM-SIAM Symposium on Discrete Algorithms (1992), 165-174.
- [KS93] D. R. KARGER AND C. STEIN, *An algorithm for minimum cuts*, Proceedings of the 25th Annual ACM Symposium on Theory of Computing (1993), 757-765.
- [NI92] H. NAGAMUCHI AND T. IBARAKI, *Linear time algorithms for finding a sparse k -connected spanning subgraph of a k -connected graph* Algorithmica 7(1992), 583-596.
- [PA94] C. H. PAPADIMITRIOU, *Computational Complexity*, Addison-Wesley 1994.
- [SW94] M. STOER AND F. WAGNER, *A simple min cut algorithm*, preprint.