

Teil 7

Graphen: Minimale Schnitte

Der Kantenzusammenhang

Ein grundlegendes Problem der Graphentheorie und Netzwerkanalyse, ist die des so genannten **Kantenzusammenhangs**.

Definition (Kantenzusammenhang)

$G = (V, E; w)$ sei ein gewichteter, ungerichteter Graph. Der **Kantenzusammenhang** $\lambda(G)$ von G ist das minimale Gesamtgewicht $w(E')$:= $\sum_{e' \in E'} w(e')$ einer Menge $E' \subseteq E$ von Kanten, so dass $G' = (V, E \setminus E')$ unzusammenhängend ist.

Anders Formuliert

$\lambda(G)$ ist das minimale Gewicht von Kanten, die entfernt werden müssen, damit G nicht mehr zusammenhängend ist.

Schnitte

Ein wichtiges Hilfsmittel zur Berechnung des Kantenzusammenhangs sind **Schnitte**.

Definition (Schnitt)

$G = (V, E; w)$ sei ein gewichteter, ungerichteter Graph. Ein **Schnitt** C von G ist eine echte, nicht-leere Teilmenge $C \subsetneq V$ von Knoten.

Das **Gewicht** $w(C)$ eines Schnittes C ist die Summe der Gewichte aller Kanten zwischen Knoten in C und \bar{C} , d.h.

$$w(C) := \sum_{u \in C, v \notin C} w(u, v).$$

Minimale s - t -Schnitte

Definition (s - t -Schnitt)

$G = (V, E; w)$ sei ein ungerichteter, gewichteter Graph und $s, t \in V$ zwei unterschiedliche Knoten. Ein Schnitt C ist ein **s - t -Schnitt**, wenn $s \in C$ und $t \in \overline{C}$ (oder umgekehrt). Man sagt auch: **C trennt s und t** .

Ein **minimaler s - t -Schnitt** ist ein s - t -Schnitt mit minimalem Gewicht $\lambda_G(s, t)$, d.h.

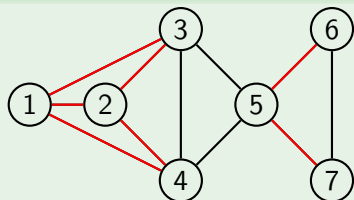
$$\lambda_G(s, t) := \min\{w(C) \mid C \subset V, s \in C, t \notin C\}.$$

Schnitte

Beobachtungen

- $w(C) = w(\overline{C})$
- $\lambda(G) = \min \{w(C) \mid C \subsetneq V, C \neq \emptyset\}$
- $\lambda(G) = \min \{\lambda_G(s, t) \mid s, t \in V\}$
- Wenn G nicht zusammenhängend ist, dann gilt $\lambda(G) = 0$.

Beispiel



- $w(\{1, 2\}) = 4$
- $w(\{6, 7\}) = 2$
- $\lambda(G) = 2$
- $\lambda_G(1, 2) = 3$

Teil 7: Graphen: Minimale Schnitte

Der Algorithmus von Stoer/Wagner

Bezeichnungen

Bezeichnungen

$G = (V, E; w)$ sei ein ungerichteter, gewichteter Graph.

- Für zwei disjunkte Mengen V_1, V_2 von Knoten, sei

$$w(V_1, V_2) := \sum_{u \in V_1, v \in V_2} w(u, v)$$

Knotenkontraktionen

$G = (V, E; w)$ sei ein ungerichteter, gewichteter Graph. Wir sagen, dass G' durch die **Kontraktion** zweier Knoten s und t aus G entsteht, wenn $G' = (V', E'; w')$ mit

- $V' = V \cup \{r\} \setminus \{s, t\}, r \notin V$
- $w'(u, v) = w(u, v)$ falls $u, v \neq r$
- $w'(r, v) = w(s, v) + w(t, v)$ für $v \neq r$
- $E' = \{(u, v) \mid w(u, v) \neq 0\}$

D.h. die Knoten s und t werden durch einen neuen Knoten r ersetzt. Alle Kanten der Form (s, v) und (t, v) werden durch Kanten (r, v) ersetzt, wobei $w'(r, v) = w(s, v) + w(t, v)$.

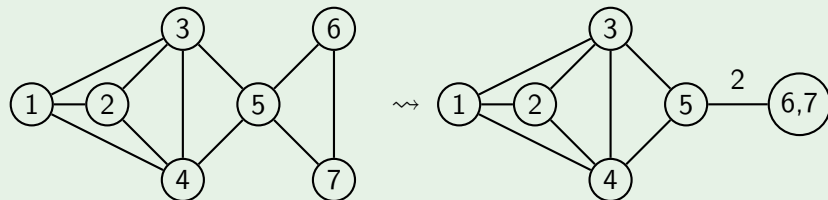
Bemerkung

Durch die Kontraktionen können Eigenschleifen entstehen, die wir ignorieren.

Knotenkontraktionen

Beispiel

Die Kontraktion der Knoten 6 und 7 führt zu dem folgenden Bild:



G' sei ein Graph, der aus $G = (V, E; w)$ durch mehrere aufeinander folgende Kontraktionen entstanden ist. Dann repräsentiert jeder Knoten von G' eine Menge von Knoten von G .

Ist $C' \subseteq V'$ eine Menge von Knoten in G' , dann bezeichnen wir die repräsentierte Menge mit $V[C']$.

Satz

Sei $G = (V, E; w)$ ein ungerichteter, gewichteter Graph mit $w(e) > 0$. Seien $s, t \in V$ zwei Knoten von G und $G/s \sim t$ der Graph der durch Kontraktion von s und t entsteht. Dann gilt

$$\lambda(G) = \min \{ \lambda_G(s, t), \lambda(G/s \sim t) \}.$$

Beweis

Zuerst beobachten wir, dass jede Knotenmenge C' von $G' := G/s \sim t$ eine Knotenmenge C von G induziert, mit $w'(C') = w(C)$. Damit gilt insbesondere $\lambda(G') \geq \lambda(G)$.

Offensichtlich gilt $\lambda_G(s, t) \geq \lambda(G)$.

Es existiert ein minimaler Schnitt C , der s und t nicht trennt. In diesem Fall induziert C ebenfalls einen Schnitt gleichen Gewichtes in $G/s \sim t$ und somit gilt $\lambda(G) = \lambda(G') = \min \{ \lambda_G(s, t), \lambda(G/s \sim t) \}$.

Im zweiten Fall *trennt jeder minimale Schnitt s und t .* In diesem Fall gilt $\lambda_G(s, t) = \lambda(G) = \min \{ \lambda_G(s, t), \lambda(G/s \sim t) \}$. □

Der Meta-Algorithmus

Dieser Satz impliziert einen rekursiven Meta-Algorithmus zur Bestimmung eines minimalen Schnittes.

Algorithmus (META-MINCUT)

Eingabe: *Ein Graph $G = (V, E; w)$*

Ausgabe: *Ein minimaler Schnitt C von G*

wenn $|V| = 1$ **dann** *Gebe \emptyset zurück*

wenn $|V| = 2$ **dann** *Gebe $\{v\}$ zurück*

Wähle zwei Knoten $s, t \in V$.

Berechne einen minimalen s - t -Schnitt C' .

$C := V$ [META-MINCUT($G/s \sim t$)]

wenn $w(C) < w(C')$ **dann** *Gebe C zurück.*

sonst *Gebe C' zurück.*

Um den Algorithmus konkretisieren zu können, muss man einen minimalen s - t -Schnitt für zwei gewählte Knoten s und t berechnen.

Maximale Adjazenzordnungen

Tatsächlich geht man aber einen anderen Weg. Man wählt die beiden Knoten s und t nicht vorher, sondern „gleichzeitig“ mit der Berechnung des minimalen s - t -Schnittes.

Dazu bedient man sich einer **Maximalen Adjazenz Ordnung**.

Definition (MA-Ordnung)

$G = (V, E; w)$ sei ein Graph mit positiven Kantengewichten. Eine Ordnung v_1, \dots, v_n auf den Knoten von G heißt **Maximale Adjazenz Ordnung (MA-Ordnung)**, wenn folgendes gilt:

- $v_1 \in E$
- Für jedes $i \geq 2$ und $j \geq i$ gilt

$$w(\{v_1, \dots, v_{i-1}\}, \{v_i\}) \geq w(\{v_1, \dots, v_{i-1}\}, \{v_j\})$$

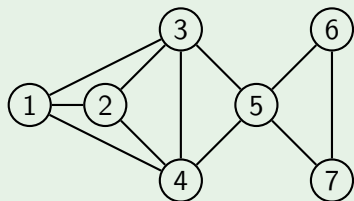
Dabei ist $w(\{v_1, \dots, v_{i-1}\}, \{v_i\})$ die **Adjazenz** von v_i zu den vorhergehenden Knoten.

Maximale Adjazenzordnungen

Anders formuliert

Eine MA-Ordnung beginnt mit einem beliebigen Knoten v_1 . Sind v_1, \dots, v_{i-1} bereits eingeordnet, so wird v_i unter den Knoten mit maximaler Adjazenz zu den Knoten $\{v_1, \dots, v_{i-1}\}$ gewählt.

Beispiel (MA-Ordnung)



Folgende Ordnungen sind MA-Ordnungen (Liste ist nicht vollständig):

- 1, 2, 3, 4, 5, 6, 7
- 1, 3, 4, 2, 5, 7, 6
- 5, 7, 6, 3, 4, 1, 2

MA-Ordnungen

Lemma (Stoer/Wagner 1994)

v_1, \dots, v_n sei eine MA-Ordnung eines ungerichteten, gewichteten Graphen $G = (V, E; w)$ mit $w(e) > 0$. Dann ist $\{v_n\}$ ein minimaler v_n - v_{n-1} -Schnitt.

Beweis

C sei ein beliebiger v_{n-1} - v_n -Schnitt. Wir verwenden die folgenden Bezeichnungen:

- $V_i := \{v_1, \dots, v_i\}$
- $w(X, u) := \sum_{v \in X} w(v, u)$ für $X \subseteq V$ und $u \notin X$
- $C_i := C \cap V_i$.

Im Folgenden werden wir zeigen, dass $w(C) \geq \deg(v_n)$ gilt.

Beweis (Fortsetzung)

Ein Knoten v_i in der MA-Ordnung heißt **aktiv**, wenn er und v_{i-1} von C getrennt werden, d.h. auf verschiedenen Seiten des Schnittes liegen.

Wir zeigen jetzt induktiv, dass für jeden aktiven Knoten v_i die folgende Ungleichung gilt:

$$w(V_{i-1}, v_i) \leq w(C_i).$$

v_i sei der erste aktive Knoten. Da v_1, \dots, v_{i-1} auf der „anderen“ Seite von C liegen, gilt offensichtlich die Gleichheit:

$$w(V_{i-1}, v_i) \leq w(C_i).$$

Sei nun v_j ein aktiver Knoten für den $w(V_{j-1}, v_j) \leq w(C_j)$ gilt und v_i der nächste aktive Knoten in der MA-Ordnung. D.h. insbesondere $j < i$.

Beweis (Fortsetzung)

O.B.d.A. können wir $v_i \in C$ und $v_j, \dots, v_{i-1} \in \overline{C}$ annehmen.

$$\underbrace{v_1 \dots v_{j-1}}_{V_{j-1}} \quad \underbrace{v_j \dots v_{i-1}}_{V_{i-1} \setminus V_{j-1} \subseteq \overline{C}} \quad \underbrace{v_i}_{\in C} \quad \dots$$

Es ergibt sich

$$\begin{aligned} w(V_{i-1}, v_i) &= w(V_{j-1}, v_i) + w(V_{i-1} \setminus V_{j-1}, v_i) \\ &\leq w(V_{j-1}, v_j) + w(V_{i-1} \setminus V_{j-1}, v_i) && \text{(MA-Ordnung)} \\ &\leq w(C_j) + w(V_{i-1} \setminus V_{j-1}, v_i) && \text{(Indvor.)} \\ &\leq w(C_i) && (V_{i-1} \setminus V_{j-1} \subseteq \overline{C}) \end{aligned}$$

Da v_n aktiv ist, gilt somit

$$\deg(v_n) = w(\{v_n\}) = w(V_{n-1}, v_n) \leq w(C).$$



Die MA-Ordnung

Die MA-Ordnung gibt uns eine einfache Möglichkeit ein paar von Knoten s und t sowie einen minimalen s - t -Schnitt zu konstruieren.

Statt zwei Knoten auszuwählen und einen minimalen s - t -Schnitt zwischen ihnen zu berechnen, kann eine MA-Ordnung dazu genutzt werden, die zwei Knoten und den dazugehörigen Schnitt zu **konstruieren**.

Dazu muss aber die Frage geklärt sein, wie die MA-Ordnung berechnet werden kann.

Da wir in jedem Schritt den Knoten suchen, der maximale Adjazenz zu den bereits eingeordneten Knoten hat, bietet sich eine Priority-Queue an.

Die Einträge sind Knoten und der Schlüssel ist die Adjazenz zu den bereits extrahierten Knoten.

Die MA-Ordnung

Algorithmus (MA-ORDER)

Eingabe: Ein Graph $G = (V, E; w)$

Ausgabe: Die beiden letzten Knoten s, t einer MA-Ordnung auf G

Daten: Eine Prioritätswarteschlange Q

$s := \text{nil}, t := \text{nil}$

für alle $v \in V$ **tue** $Q.\text{insert}(v, 0)$

solange Q nicht leer ist **tue**

$s := t$

$t := Q.\text{delete_max}()$ // Extrahiere den nächsten Knoten

für alle $v \in N(t)$ **tue** // Aktualisiere die Nachbarn

wenn $v \in Q$ **dann** $Q.\text{increase_key}(v, Q.\text{key}(v) + w(t, v))$

Ende

Ende

Gebe s und t zurück.

Algorithmus (MINCUT)

Eingabe: Ein ungerichteter, gewichteter Graph $G = (V, E; w)$ mit $w(e) > 0$.

Ausgabe: Ein minimaler Schnitt C von G .

$C := \emptyset, w := \infty$

$G' := G$

solange $|V(G')| \geq 2$ **tue**

$(s, t) := \text{MA-ORDER}(G')$

wenn $\deg_{G'}(t) < w$ **dann**

$w := \deg_{G'}(t), C := V[t]$

// Speichere den Schnitt

Ende

$G' := G'/s \sim t$

// Kontrahiere s und t

Ende

Die Laufzeit von MINCUT

Zuerst ermitteln wir die Laufzeit von MA-ORDER.

Insgesamt führen wir die folgenden Operationen durch:

- $|V|$ insert- und delete_max-Operationen
- $|E|$ increase_key-Operationen (eine pro Kante)

Unter Verwendung von Fibonacci-Heaps ergeben sich somit amortisierte Kosten von

$$O(|V| + |V| \log |V| + |E|) = O(|V| \log |V| + |E|)$$

zur Berechnung einer MA-Ordnung.

Da wir mit jedem Durchlauf zwei Knoten kontrahieren, berechnen wir insgesamt $|V| - 1$ MA-Ordnungen mit maximal $|V|$ Knoten und $|E|$ Kanten. Die Kontraktion zweier Knoten lässt sich in Zeit $O(|V| + |E|)$ realisieren, so dass sich die Gesamtlaufzeit von MINCUT als

$$O(|V|^2 \log |V| + |V||E|)$$

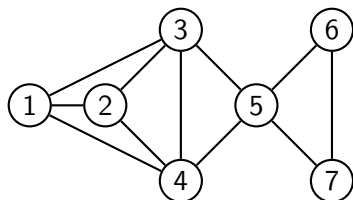
ergibt.

Die Laufzeit von MINCUT

Satz

Ein minimaler Schnitt eines ungerichteten, gewichteten Graphen $G = (V, E; w)$ mit positiven Kantengewichten lässt sich, unter Verwendung von Fibonacci-Heaps, in Zeit $O(|V|^2 \log |V| + |V||E|)$ berechnen.

MINCUT - Ein Beispiel



MA-Ordnungen

- 1 5, 6, 7, 3, 4, 1, 2
 $w = 3$ und $C = \{2\}$.
- 2 5, 6, 7, 4, $\{1,2\}$, 3
- 3 $\{1,2,3\}$, 4, 5, 6, 7
 $w = 2$ und $C = \{7\}$.
- 4 $\{6,7\}$, 5, 4, $\{1,2,3\}$
- 5 $\{1,2,3,4\}$, 5, $\{6,7\}$
- 6 $\{1,2,3,4\}$, $\{5, 6,7\}$

Damit ist $\{7\}$ ein minimaler Schnitt mit Kapazität 2.

Teil 7: Graphen: Minimale Schnitte

Verbesserung des Algorithmus

Eine einfache Beobachtung

Beobachtung

Ist v_1, \dots, v_n eine MA-Ordnung von $G = (V, E; w)$, so ist v_1, \dots, v_i eine MA-Ordnung auf dem von den Knoten v_1, \dots, v_i induzierten Graphen $G_i := G[v_1, \dots, v_i]$.

Die Adjazenz $\text{adj}(v_i) := w(v_{i-1}, v_i)$ von v_i ist somit das Gewicht eines minimalen v_{i-1} - v_i -Schnittes in G_i .

Da aber das Hinzufügen von Knoten und Kanten, das Gewicht eines minimalen v_{i-1} - v_i -Schnittes nur erhöht, gilt somit

$$\text{adj}(v_i) = \lambda_{G_i}(v_{i-1}, v_i) \leq \lambda_G(v_{i-1}, v_i).$$

Diese Beobachtung lässt sich nun ausnutzen, um die Berechnung eines minimalen Schnittes (unter Umständen) zu beschleunigen.

Mehr Kontraktionen pro Durchlauf

Sei C ein bekannter Schnitt mit $w(C) = \tau$. Dann gilt $\lambda(G) \leq \tau$.

Wird nun bei der Konstruktion der MA-Ordnung ein Knoten v_i mit $\text{adj}(v_i) \geq \tau$ gefunden, so können v_{i-1} und v_i nur von Schnitten mit Kapazität $\geq \tau$ getrennt werden. D.h. wir können v_{i-1} und v_i kontrahieren, ohne Schnitte mit kleinerem Gewicht zu zerstören.

Dieses Vorgehen ermöglicht unter Umständen mehr als eine Kontraktion pro Runde.

Algorithmus (MA-ORDER2)

Eingabe: Ein Graph $G = (V, E; w)$ eine Schranke τ

Ausgabe: Eine Kontraktion von G .

Daten: Eine Prioritätswarteschlange Q

für alle $v \in V$ **tue** $Q.insert(v, 0)$

$s := \text{nil}, t := \text{nil}$

solange Q nicht leer ist **tue**

$s := t$

$a := Q.key(Q.max())$ // Extrahiere Maximum

$t := Q.delete_max()$

für alle $v \in N(t) \cup Q$ **tue** // Korrigiere Adjazenzen

$Q.increase_key(v, Q.key(v) + w(t, v))$

Ende

wenn $a \geq \tau$ **dann** $G := G/s \sim t;$ // Kontrahiere s und t

Ende

Ein angepasster Algorithmus

Ist die Schranke τ stets kleiner oder gleich dem Minimalgrad, so werden die letzten beiden Knoten auf jeden Fall kontrahiert (die Adjazenz des letzten Knotens ist sein Grad). Damit sinkt die Zahl der Knoten mit jeder Runde um mindestens eins.

Um diese Bedingung stets zu erfüllen, wird nach den Kontraktionen ein Knoten minimalen Grades $\delta(G)$ gesucht. Ist dieser neue Minimalgrad kleiner als τ , wird die Schranke gesenkt.

Gleichzeitig wird die Menge der durch den gefundenen Knoten repräsentierten Originalknoten als Schnitt mit dem aktuell kleinstem Gewicht gespeichert.

Algorithmus (MINCUT2)

Eingabe: Ein Graph $G = (V, E; w)$

Ausgabe: Die beiden letzten Knoten s, t einer MA-Ordnung auf G

Daten: Ein Graph $G' = (V', E'; w')$

$G' := G$ // Kopiere G

Suche einen Knoten $v \in V'$ mit minimalem Grad.

$\tau := \delta(G), C := \{v\}$

solange $|V'| \geq 2$ **tue**

$G' := MA - ORDER2(G', \tau)$ // Baue MA-Ordnung und
kontrahiere

Suche einen Knoten $v \in V'$ mit minimalem Grad.

wenn $\deg(v) < \tau$ **dann** // Ist der Minimalgrad gesunken?

$\tau := \deg(v), C := V[v]$ // Setze neuen Schnitt

Ende

Ende

Beweis der Korrektheit (Skizze)

Offensichtlich kann die Schranke τ im Laufe des Algorithmus nur sinken. Da dies nur geschieht, wenn ein kleinerer Schnitt gefunden wurde, gilt jedoch stets $\tau \geq \lambda(G)$.

Da in jeder Runde mindestens ein Paar von Knoten kontrahiert wird, besteht der Graph nach höchstens $n - 1$ Runden nur noch aus einem Knoten.

Außerdem wurde im Laufe der Konstruktion der MA-Ordnung sichergestellt, dass nur Knotenpaare s, t mit $\lambda_G(s, t) \geq \tau$ kontrahiert wurden.

Damit ist klar, dass am Ende $\tau = \lambda(G)$ gelten muss, da ansonsten mindestens zwei Knoten verbleiben müssten, die die Hälften eines minimalen Schnittes repräsentieren.

Satz (Brinkmeier, 2005)

Der Algorithmus MINCUT2 berechnet einen minimalen Schnitt eines ungerichteten, gewichteten Graphen $G = (V, E; w)$ mit positiven Kantengewichten. Dazu benötigt er im schlechtesten Fall $O(|V|^2 \log |V| + |V||E|)$ Zeit. Im Mittel benötigt er jedoch weniger Zeit.

Da in jeder Runde mindestens eine Kontraktion vorgenommen wird, benötigt MINCUT2, wie der Algorithmus von Stoer/Wagner, höchstens $|V| - 1$ Runden. Damit ergibt sich die gleiche asymptotische Laufzeit im schlechtesten Fall.

Das folgende Beispiel zeigt jedoch, dass in vielen Fällen deutlich weniger Runden erforderlich sind.

Beispiel



Technische Anmerkungen

Der Algorithmus MINCUT2 in der dargestellten Form ist technisch ungünstig. Durch einige Veränderungen lässt er sich beschleunigen:

- Die Suche nach einem Knoten minimalen Grades lässt sich mit dem Aufbau der MA-Ordnung verbinden. Da nur durch Kontraktionen Knoten mit kleinerem Grad als bisher entstehen können, genügt es nach jeder Kontraktion den Grad des neuen Knotens zu überprüfen. Dadurch kann die Schranke sogar **während** einer Runde gesenkt werden, was im Folgenden mehr Kontraktionen ermöglichen kann.
- Statt die Kontraktionen direkt auszuführen (jede mit $O(|V| + |E|)$ Zeit), kann man die Kontraktionen bis zum Ende einer Runde verschieben. Dann können alle mit einer Gesamtlaufzeit von $O(|V| + |E|)$ durchgeführt werden.

Weitere Verbesserungen

Weitere Verbesserungen ergeben sich für ganzzahlige Gewichte, indem keine allgemeine Prioritätswarteschlange verwendet wird.

Es kann gezeigt werden, dass es beim Aufbau der MA-Ordnung genügt, bei der Extraktion stets einen Knoten v zu wählen, so dass für alle verbliebenen Knoten u folgendes gilt:

$$\text{adj}(v) \geq \min(\tau, \text{adj}(u)).$$

D.h. der extrahierte Knoten hat entweder eine Adjazenz $\geq \tau$ oder, falls kein solcher Knoten existiert, maximale Adjazenz.

Satz (Brinkmeier, 2005)

Ein minimaler Schnitt eines ungerichteten Graphen $G = (V, E; w)$ mit positiven, ganzzahligen Kantengewichten, lässt sich in Zeit $O(\delta(G)|V|^2 + |E|) = O(\delta(G)|V|^2)$ berechnen, wobei $\delta(G)$ der minimale Grad eines Knotens in G ist.

Weitere Verbesserungen

Bemerkung

Die Laufzeitschranke $O(\delta(G)|V|^2)$ ist mit etwas Vorsicht zu behandeln.

Ist z.B. $\delta(G) \in O(2^n)$, d.h. haben die Kanten exponentielles Gewicht, hat der Algorithmus exponentielle Laufzeit.

Für „gutartige“ Kantengewichte, z.B. alle kleiner gleich einer vorgegebenen Konstante, ist der Algorithmus in der Regel aber schneller als das ursprüngliche Verfahren von Stoer und Wagner.

Weitere Verbesserungen

Lemma (Brinkmeier, 2005)

$G = (V, E; w)$ sei ein ungerichteter Graph mit positiven, ganzzahligen Kantengewichten.

- 1 Für ein gegebenes $\tau > 0$ kann man in Zeit $O(\tau|V|^2)$ prüfen, ob $\lambda(G) < \tau$ gilt. Ist das der Fall, kann ein minimaler Schnitt in der gleichen Zeit berechnet werden.
- 2 Ein minimaler Schnitt von G kann in Zeit $O(\lambda(G)|V|^2)$ berechnet werden.

Beweis

Um die erste Behauptung zu zeigen, verwenden wir lediglich den Algorithmus MINCUT2 mit τ als anfänglicher Schranke.

Die zweite Behauptung ergibt sich dann mittels einer Art binärer Suche.

Beweis (Fortsetzung)

Für wachsendes $i = 1, 2, \dots$, prüfen wir mittels MINCUT2, ob $\lambda(G) < 2^i$ gilt. Dies wird wiederholt, bis das Intervall $2^{i-1} \leq \lambda(G) < 2^i$ erreicht wird. Dann erhalten wir einen minimalen Schnitt von G .

Die Laufzeit ergibt sich als

$$\begin{aligned} O(2|V|^2) + \dots + O(2^{i-1}|V|^2) + O(2^i|V|^2) \\ &= O((2 + \dots + 2^{i-1} + 2^i)|V|^2) \\ &= O(2^{i+1}|V|^2) \\ &= O(\lambda(G)|V|^2). \end{aligned}$$

