Seminarvortrag

Datenstrukturen und Algorithmen

Professor Hagerup, Professur für Komplexität und Algorithmen

11. Mai 2000

Minimale Schnitte in Graphen

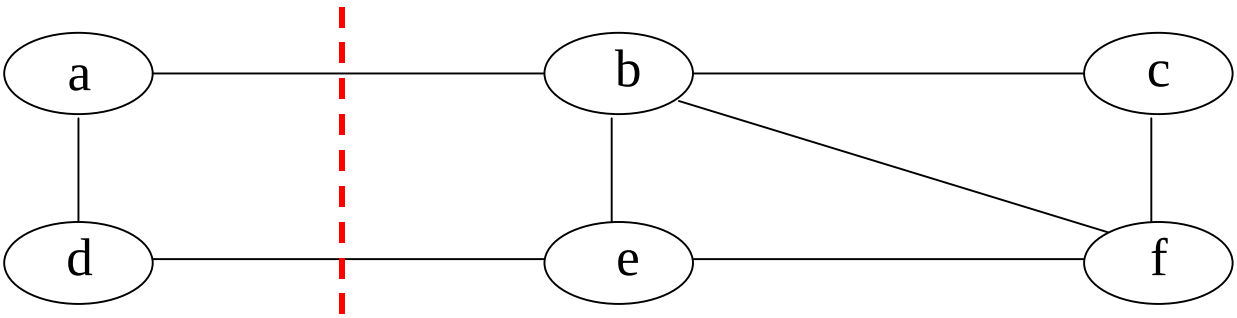
Autoren:

Ernst-Joachim Preussler

Patrick Klein

Was ist ein Schnitt?

- Aufteilung der Knotenmenge des Graphen in *zwei nicht leere* Partitionen.
- Notation:
  1. Menge der Knoten einer Partition.
  2. Durchtrennte Kanten
- Beispiel:



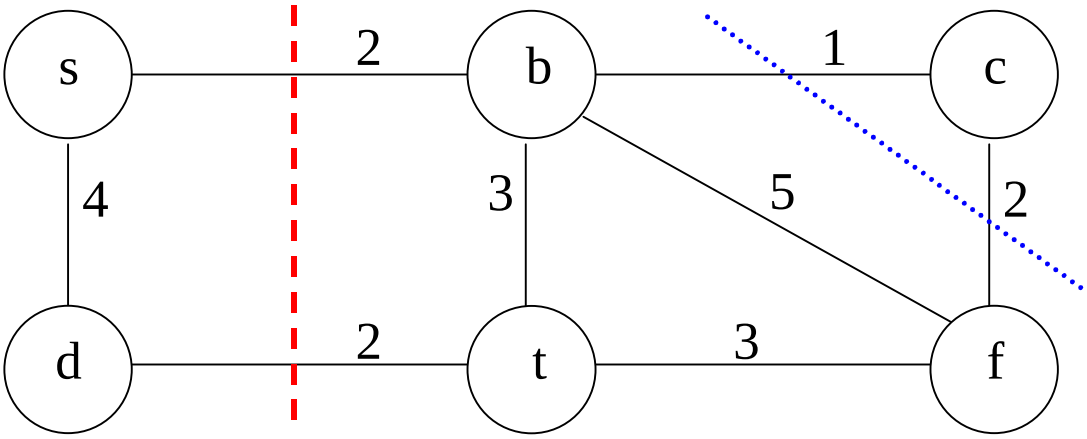
  Mögliche Notationen für obiges Beispiel:
  1. Partition {a, d}
  2. Partition {b, c, e, f}
  3. Schnittkanten (a,b) und (d,e)
- Beliebige Schnitte zu finden ist trivial.
- Minimale Schnitte zu finden ist *nicht* trivial.

Minimaler Schnitt?

- Ungewichtete Graphen:
  - Der Schnitt mit den *wenigsten Kanten* zwischen zwei Partitionen.
- Gewichtete Graphen:
  - Der Schnitt mit dem *geringsten Kantengewicht* zwischen zwei Partitionen
  - Gewichte sind *nichtnegative* reelle Zahlen, sonst Problem *NP*-vollständig.
    - Transformation auf das *NP*-vollständige Problem des maximalen Schnittes.
- Analogie beider Varianten
  - Setze im ungewichteten Fall das Gewicht jeder Kante auf Eins.
- Es werden nur *zusammenhängende Graphen* betrachtet.
  - Minimaler Schnitt sonst trivialerweise Null.

Bisherige Verfahren

- Berechnung eines maximalen Flusses (*Maxflow*) in einem ***s*-*t*-Netzwerk**.
  - *s*-*t*-Netzwerk ist kantengewichteter Graph.
  - Knoten *s* Quelle, Knoten *t* Senke eines Netzwerkflusses.
- Wert eines maximalen Flusses entspricht dem Wert eines minimalen *s*-*t*-Schnittes.
  - Max-Flow-Min-Cut-Theorem von Ford und Fulkerson.
- Beispiel (minimaler *s*-*t*-Schnitt nicht minimaler Schnitt):



- Maxflow-Algorithmus auf jede *s*-*t*-Kombination: Laufzeit *quadratisch* zur Anzahl der Knoten.
- Knoten *s* fest, für jedes *t* Durchlauf: Laufzeit *linear* zur Anzahl der Knoten.

Laufzeiten

- Mit Maxflow-Algorithmen werden minimale Schnitte in $O(mn \log(n^2/m))$ Zeit berechnet (Hao und Orlin).
- Betrachtung des Min-Cut-Problems *ohne* Maxflow-Algorithmen verspricht *effektivere Ansätze*.
  - Aber: Schnellere Lösung des Min-Cut-Problems liefert *keine* Verbesserung der Maxflow-Algorithmen.
- Algorithmus MINIMUMCUT (Deterministisch)
  - Laufzeit von $O(nm + n^2 \log n)$
  - Findet sicher *einen* minimalen Schnitt.
- Algorithmus RECURSIVE-CONTRACT (Randomisiert)
  - Laufzeit von $O(n^2 \log^3 n)$
  - Findet *alle* minimalen Schnitte mit *hoher* Wahrscheinlichkeit.

Anwendungsgebiete

- Ermittlung minimaler Schnitte in Netzwerken.
  - Schwachstellen für *zufällige* Netzzusammenbrüche.
  - Randomisierter Algorithmus besser.
- Relationen zwischen Hypertextdokumentgruppen finden.
  - Links zwischen den Texten sind Kanten.
  - Kleine Schnitte $\Rightarrow$ Geringe Themenverwandtschaft
  - Randomisierter Algorithmus besser.
- Compilierung paralleler Sprachen
  - Kanten symbolisieren *Prozessorkommunikation*.
  - Beide Algorithmen geeignet.
- Exakte Lösungen für *Traveling*-*Salesman*-Probleme.
  - Technik der *cutting planes.*
  - Beide Algorithmen geeignet.
- Verallgemeinerung: Minimierung submodularer Funktionen.
  - Submodulare Funktion f auf V hat folgende Eigenschaft: $f(Y) + f(Z) \geq f(Y \cap Z) + f(Y \cup Z)$

Der Algorithmus MINIMUMCUT

Definitionen:

- Sei $G = (V, E)$ ungerichteter Graph mit der Knotenmenge V und der Kantenmenge E.
- Jede Kante $e \in E$ hat *nichtnegatives* Gewicht w(e).
- Sei a beliebiger aber fester Startknoten aus V.
- MINIMUMCUTPHASE liefert *cut_of_the_phase*.
- *current_minimum_cut* ist aktueller minimaler Schnitt.

Hauptprogramm:

```
1  PROCEDURE MINIMUMCUT (G, w, a)
2  BEGIN
3  WHILE |V| > 1 DO
4      BEGIN
5      MINIMUMCUTPHASE (G, w, a)
6      IF cut_of_the_phase < current_minimum_cut THEN
       current_minimum_cut = cut_of_the_phase
7      END
8  PRINT current_minimum_cut
9  END
```

Prozedur MINIMUMCUTPHASE

Definition:

- Menge A ist Untermenge von V.
  - Initialisierung mit Startknoten a.
  - Wiederholung der Schleife, solange $A \neq V$

Unterprogramm:

PROCEDURE MINIMUMCUTPHASE (G, w, a)

A BEGIN

B $A \leftarrow \{a\}$

C WHILE $A \neq V$ DO

D BEGIN

E ADD TO A THE MOST TIGHTLY CONNECTED VERTEX

F END

G STORE THE *cut_of_the_phase*

H SHRINK G BY MERGING THE TWO VERTICES ADDED LAST

I END

Erläuterungen

Zeile E: ADD TO *A* THE MOST TIGHTLY CONNECTED VERTEX

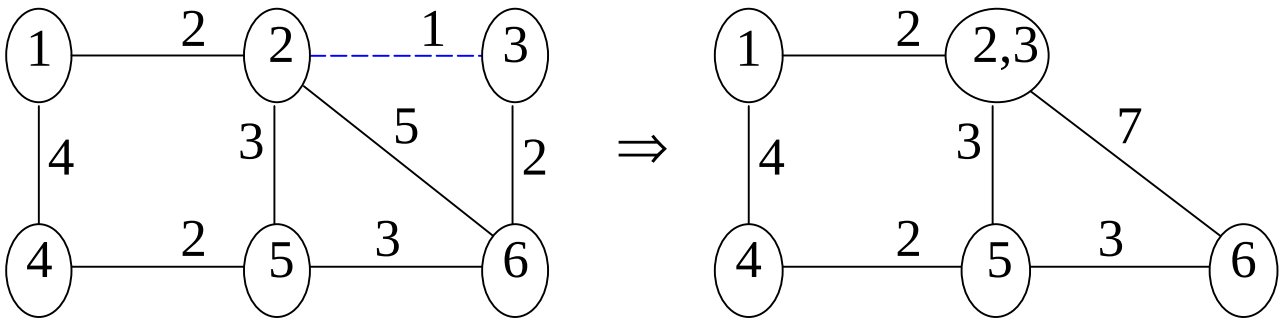
- Der Knoten *v* aus *V* \ *A*, der am *engsten* mit *A* verbunden ist, wird zu *A* hinzugefügt. Bedeutung: Summe der Gewichte der direkten Kanten von *A* nach *v* sind maximal.

Zeile G: STORE THE *cut_of_the_phase*

- *cut_of_the_phase* ist der letzte zu *A* hinzugefügte (Super)-Knoten, welcher mit *t* markiert wird.

Zeile H: SHRINK *G* BY MERGING THE TWO VERTICES ADDED LAST

- Der vorletzte zu *A* hinzugefügten Knoten (mit *s* markiert) und *t* werden zusammengefaßt (**Kontraktion**).
  - Die Kante (*s*,*t*), falls vorhanden, wird entfernt.
  - Entstehende Doppelkanten werden mit der Summe der Kantengewichte als neue Kante zusammengefaßt.
- Beispiel für eine Kontraktion an der Kante (2,3):



Korrektheitsbeweis

Theorem 1:

Seien *s* und *t* zwei Knoten aus *G*. Sei *G / {s,t}* der Graph, der entsteht, wenn *s* und *t* zusammengefaßt werden.

Dann ist ein minimaler Schnitt von *G* das Minimum eines minimalen *s*-*t*-Schnittes von *G* und eines minimalen Schnittes von *G / {s,t}*.

Beweis:

1. Wenn es einen minimalen Schnitt in *G* gibt, der *s* und *t* trennt (*s* und *t* liegen also in verschiedenen Partitionen), dann ist dies notwendigerweise auch ein minimaler *s*-*t*-Schnitt in *G*.
2. Wenn dies nicht der Fall ist, muß sich der minimale Schnitt in *G / {s,t}* befinden, da *s* und *t* in der selben Partition (bezüglich *G*) liegen und die Kante (*s*,*t*), welche als einzige Kante in *G / {s,t}* verschwindet, somit nicht Schnittkante sein kann.

Theorem 2:

Seien *s* und *t* die beiden letzten zu *A* hinzugefügten Knoten. Dann ist der *cut_of_the_phase* ein minimaler *s*-*t*-Schnitt.

Beweis:

Zu zeigen: Jeder beliebige *s*-*t*-Schnitt *C* ist mindestens so groß wie der *cut_of_the_phase*.

Definitionen:

- Knoten $v \neq a$ ist **aktiv** (zu *C*), wenn *v* und der *unmittelbar* vor *v* zu *A* hinzugefügte Knoten in unterschiedlichen Partitionen von *C* liegen („Partitionswechsel“).
- Sei w(*C*) Gewicht des beliebigen, aber festen Schnittes *C*.
- Sei A_v die Menge aller Knoten in *A* vor Hinzunahme von *v*.
- Sei C_v der Schnitt, der sich (durch *C)* auf $A_v \cup \{v\}$ ergibt.
  - Sei w(C_v) das Gewicht dieses Schnittes.

Ziel:

- Z.Z.: Für *jeden aktiven* Knoten *v* gilt: w(A_v, *v*) ≤ w(C_v).
- ⇒ Da *t* aktiv, folgt **w(A_t, *t*) ≤ w(C_t)**.
- Beachte: w(A_t, *t*) = *cut_of_the_phase*
- Beachte: w(C_t) = w(*C)*

Beweis per Induktion

Annahme:

Für *jeden aktiven* Knoten v gilt: $w(A_v, v) \leq w(C_v)$.

Verankerung:

Für den ersten aktiven Knoten x gilt: $w(A_x, x) = w(C_x)$.

Induktionsschritt:

Induktionsannahme sei bis zum aktiven Knoten v bewiesen.

Sei u der nächste aktive Knoten, der hinzugefügt wird.

$\Rightarrow$(1) $w(A_u, u) = w(A_v, u) + w(A_u \setminus A_v, u)$

Es gilt $w(A_v, u) \leq w(A_v, v)$, da v am engsten mit A_v verbunden ist.

$\Rightarrow$(2) $w(A_u, u) \leq w(A_v, v) + w(A_u \setminus A_v, u)$

Nach Induktionsannahme gilt $w(A_v, v) \leq w(C_v)$.

$\Rightarrow$(3) $w(A_u, u) \leq w(C_v) + w(A_u \setminus A_v, u)$

Es gilt: $w(C_v) + w(A_u \setminus A_v, u) = w(C_u)$.

$\Rightarrow$(4) $\mathbf{w(A_u, u) \leq w(C_u)}$

Damit ist die Induktionsannahme bewiesen.

Laufzeitanalyse

- MINIMUMCUT hat n-1 Aufrufe von MINIMUMCUTPHASE.
- MINIMUMCUTPHASE: Auswahl des nächsten Knotens.
  - Prioritätswarteschlange: Knoten aus $V \setminus A$.
  - Sortiert nach Kantengewichten von v nach A, also w(A,v) für alle v aus $V \setminus A$.
  - Ein Knoten v wird in A eingefügt:
    - Der Knoten v muß entfernt werden (EXTRACTMAX).
    - Kantengewicht w(A,w) jedes Knotens w aus $V \setminus A$ wird um Kantengewicht der Kante (v,w), falls existent, erhöht (INCREASEKEY).
- INCREASEKEY wird für jede Kante exakt einmal *in konstanter Zeit* durchgeführt, wird also m-mal benutzt.
- EXTRACTMAX hat (mit Fibonacci-Heaps) eine *amortisierte Laufzeit* von O(log n) und wird n-mal benutzt.
- $\Rightarrow$ Laufzeit MINIMUMCUTPHASE: O($m + n \log n$)
- $\Rightarrow$ Laufzeit MINIMUMCUT:

  O($nm + n^2 \log n$) für n Knoten und m Kanten.

Randomisierte Algorithmen

- Motivation: Erhoffte schnellere Laufzeit
- Typ: Monte Carlo Algorithmus, d.h. keine Erfolgsgarantie

***Algorithmus 1**:*

Procedure Contract (**G**)

*Solange **G** noch mehr als zwei Knoten enthält:*

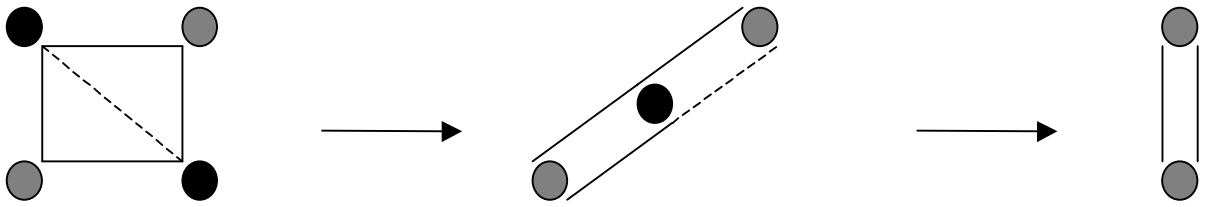
*Wähle zufällig eine Kante (**u**,**v**) aus **E**;*

*Verschmelze die Knoten **u** und **v** zu einem neuen Knoten **u**;*

*Alle Kanten der Form (**w**,**u**) oder (**w**,**v**) werden zu Kanten (**w**,**u**), für alle **w** aus **V**;*

*Alle Kanten der Form (**u**,**v**) werden gelöscht;*

*Return **G**.*



Zwei Fragen:

- Ist der Schnitt minimal?
- Wie groß ist die Wahrscheinlichkeit, einen minimalen Schnitt zu finden?

Lemma 1.1:

Ein Schnitt (***A***, ***B***) wird vom *Contract*-Algorithmus nicht zerstört ⇔ Keine Kante des Schnittes wird kontrahiert.

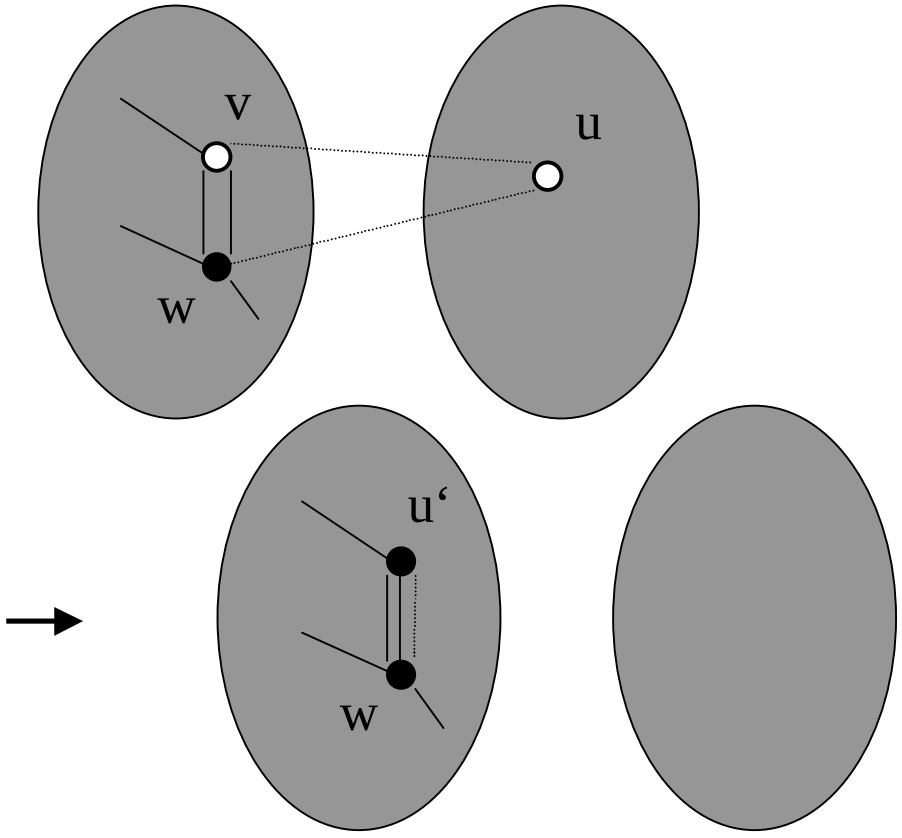
Beweis:

⇒: Trivial

⇐: Behauptung: Zwischen zwei Knoten können keine Kanten unterschiedlichen Typs (Schnittkanten, keine Schnittkanten) verlaufen.

Beweis dieser Behauptung durch Induktion

Beispiel:



Theorem 1.2:

Ein minimaler Schnitt in ***G*** wird durch *Contract* mit einer Wahrscheinlichkeit von $\Omega(n^{-2})$ nicht zerstört.

Beweis:

- Naiver Ansatz: Graph, in dem die Knoten in einem Kreis angeordnet sind, mit n Knoten und n Kanten.
  - Wahrscheinlichkeit einen minimalen Schnitt zu finden ist Eins und es existieren $\binom{n}{2}$ minimale Schnitte.
- Rechnerischer Beweis über Anzahl von Kanten.

$$\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right)$$

$$=\frac{n-2}{n}\frac{n-3}{n-1}\frac{n-4}{n-2}\cdots\frac{2}{4}\frac{1}{3}=\frac{2}{n(n-1)}=\binom{n}{2}^{-1}=\Omega\left(n^{-2}\right)$$

Korollar 1.3:

Ein minimaler Schnitt (***A***, ***B***) übersteht die Kontraktion auf *k* Knoten mit einer Wahrscheinlichkeit von

$$\binom{k}{2}\Big/\binom{n}{2}=\frac{(k-1)k}{(n-1)n}=\Omega\left((k/n)^2\right),\quad k\geq 2$$

Algorithmus 2:

*Procedure Contract‘ (***G***)*

Solange **G** *aus mehr als zwei Knoten besteht*

*Wähle eine Kante (***u**,**v***) mit einer Wahrscheinlichkeit gemäß ihres Gewichtes (*)*

Verschmelze **u** *und* **v**

Return **G**

(*)

- Die Gewichte aller Kanten werden aufsummiert.
- Wahrscheinlichkeit, daß eine Kante gewählt wird, ist Kantengewicht/Gesamtgewicht.
- Hohes Kantengewicht $\Rightarrow$ hohe Wahrscheinlichkeit, ausgewählt zu werden.

- *Contract(**G**,k)* ist eine Modifikation des Contract-Algorithmus, dessen äußere Schleife beim Erreichen von *k* Knoten terminiert.

Korollar 1.4:

Der Algorithmus *Contract* findet einen minimalen Schnitt in **G** mit einer Wahrscheinlichkeit von $\Omega(n^{-2})$.

Implementierung: Auswahl

- Repräsentation durch die Adjazenz-Matrix $\boldsymbol{W}$ und Array $\boldsymbol{D}(\boldsymbol{u})$ mit

$$D(u) = \sum_{v} W_{u,v}$$

- Gewichte der ersten k Kanten ergeben den Gewichtsvektor

$$W_k = \sum_{i=1}^{k} w_i$$

- $\boldsymbol{W}_{\mathrm{m}}$ ist der Vektor über alle m Kantengewichte.

Auswahlverfahren:

- Black-Box *Random-Select* wählt bei Eingabe des Gewichtsvektors aus m Kanten in O(log m) Zeit eine Kante aus.
- Auswahl der Kante ($\boldsymbol{u}$,$\boldsymbol{v}$) in O(n) möglich
  - $\boldsymbol{u}$ wird mit Wahrscheinlichkeit gemäß $\boldsymbol{D}(\boldsymbol{u})$ gewählt
  - $\boldsymbol{v}$ wird mit Wahrscheinlichkeit gemäß $\boldsymbol{W}_{\mathrm{u,v}}$ gewählt

Implementierung: Auswahl

Lemma 2.1:

Wird eine Kante wie beschrieben gewählt, dann ist die Wahrscheinlichkeit, daß diese Kante gewählt wurde, proportional zu ihrem Gewicht $\boldsymbol{W}_{u,v}$.

Beweis:

Sei

$$\sigma = \sum_v D(v)$$

Damit ist $\sigma = 2\boldsymbol{W}_m$. Dann ist die Wahrscheinlichkeit

P[(***u***,***v***) wurde gewählt] =

$$= \frac{D(u)}{\sigma}\frac{W_{u,v}}{D(u)} + \frac{D(v)}{\sigma}\frac{W_{u,v}}{D(v)} = \frac{2W_{u,v}}{\sigma} = \alpha W_{u,v}$$

Mit $\alpha = 1/\boldsymbol{W}_m$ ist die Wahrscheinlichkeit, eine Kante auf diese Art zu wählen, gleich der Wahrscheinlichkeit, eine Kante direkt auszuwählen.

Implementierung: Kontraktion einer Kante

Algorithmus 3:

*Procedure Edge-Contract (***u**,**v***);*
$D(\mathbf{u}) := D(\mathbf{u}) + D(\mathbf{v}) - 2W_{\mathbf{u},\mathbf{v}}$;
$D(\mathbf{v}) := 0$;
$W_{\mathbf{u},\mathbf{v}} := W_{\mathbf{v},\mathbf{u}} := 0$;
For alle Knoten **w** *außer* **u** *und* **v** *do*
$W_{\mathbf{u},\mathbf{w}} := W_{\mathbf{u},\mathbf{w}} + W_{\mathbf{v},\mathbf{w}}$;
$W_{\mathbf{w},\mathbf{u}} := W_{\mathbf{w},\mathbf{u}} + W_{\mathbf{w},\mathbf{v}}$;
$W_{\mathbf{v},\mathbf{w}} := W_{\mathbf{w},\mathbf{v}} := 0$

Beispiel:

	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

u	1	2	3	4
D(u)	3	2	2	3

Knoten	1	2	3	4
Verschmolzen	1	2	3	4

	1	2	3	4
1	0	2	2	0
2	2	0	0	0
3	2	0	0	0
4	0	0	0	0

u	1	2	3	4
D(u)	4	2	2	0

Knoten	1	2	3	4
Verschmolzen	1	2	3	1

	1	2	3	4
1	0	2	0	0
2	2	0	0	0
3	0	0	0	0
4	0	0	0	0

u	1	2	3	4
D(u)	2	2	0	0

Knoten	1	2	3	4
Verschmolzen	1	2	1	1

Laufzeit: Contract

Korollar 2.2:

Der *Contract*-Algorithmus kann in $O(n^2)$ Zeit ausgeführt werden.

- Erfolgswahrscheinlichkeit eines *Contract*-Durchlaufes (entspricht *Contract*(***G***,2)) beträgt $\Omega(n^{-2})$.
- Bessere Erfolgsaussichten mit mehreren Durchläufen.
- Bei $dn^2 \ln n$ Durchläufen beträgt die Wahrscheinlichkeit:

$$\left(1-\frac{1}{n^2}\right)^{dn^2 \ln n} \leq e^{-d \ln n} \leq n^{-d}$$

- Aber: Laufzeit $O(n^4 \ln n)$.
- Besser: Berechnungszeit verkürzen durch Verkleinerung der Teilprobleme.

Rekursiver Algorithmus

Algorithmus 4:

*Recursive-Contract(***G**, n*)*

Eingabe: Ein Graph **G** *mit* n *Knoten*

Falls **G** *weniger als 7 Knoten hat*

Dann

G‘*:= Contract(***G**, *2)*

Ausgabe: Das Gewicht der Kanten zwischen den beiden resultierenden Knoten

Sonst Wiederhole zweimal

G‘ *:= Contract(***G**, $\lceil n/(\sqrt{2}) + 1 \rceil$*)*

*Recursive-Contract(***G‘**, $\lceil n/(\sqrt{2}) + 1 \rceil$*)*

Ausgabe: Das Minimum der beiden Resultate

Lemma 3.1:

Der Algorithmus *Recursive-Contract* läuft in $O(n^2 \log n)$ Zeit.

Beweis:

- Zwei unabhängige Kontraktionen des Graphen auf $\lceil n/\sqrt{2}+1 \rceil$ Knoten.
- Jeweils rekursiver Aufruf des Algorithmus.
- Die Kontraktionen benötigen jeweils $O(n^2)$ Zeit
- Das ergibt folgende Rekursionsgleichung:

$$T(n) = 2\left(n^2 + T\left(\left\lceil \frac{n}{\sqrt{2}} + 1 \right\rceil \right) \right)$$

- Abschätzung der Lösung mit $T(n) = O(n^2 \log n)$

Lemma 3.3:

Recursive-Contract findet einen bestimmten minimalen Schnitt mit einer Wahrscheinlichkeit von $\Omega(1/\log n)$.

Beweis:

Zwei Bedingungen, um einen minimalen Schnitt zu finden:

- Der Schnitt übersteht eine der beiden Kontraktionen.
- Der Schnitt wird in dem darauf folgenden rekursiven Aufruf gefunden.

Beweis: Lemma 3.3

- Die Erfolgswahrscheinlichkeit, einen minimalen Schnitt zu finden, ist gleich dem Produkt der Wahrscheinlichkeiten, eine der Kontraktionen zu überstehen, und von einem rekursiven Aufruf gefunden zu werden.
- Wahrscheinlichkeit, die Kontraktion zu überstehen, ist:

$$\left(\frac{\left(\left\lceil n/\sqrt{2}+1 \right\rceil\right)\left(\left\lceil n/\sqrt{2}+1 \right\rceil - 1\right)}{n(n-1)} \right) \geq \frac{1}{2}$$

- Erfolgswahrscheinlichkeit von *Contract* ist bei 6 Knoten mindestens 1/15.
- $\Rightarrow$ Rekursionsgleichung für P(n):

$$P(n) \geq \begin{cases} 1-\left(1-\frac{1}{2}P\left(\left\lceil \frac{n}{\sqrt{2}}+1 \right\rceil\right)^2\right) & \textit{falls } n \geq 7 \\ \frac{1}{15} & \textit{sonst} \end{cases}$$

- Es genügt, eine untere Schranke für P(n) zu berechnen.
- Lösung durch Variablenersetzung.
- $\boldsymbol{p}_k$ = Erfolgswahrscheinlichkeit auf der k-ten Ebene der Rekursion.

Beweis: Lemma 3.3

- Ein Blatt liegt auf Ebene 0.

$$p_0 = \frac{1}{15}$$

$$p_{k+1} = 1-\left(1-\frac{1}{2}p_k\right)^2 = p_k - \frac{1}{4}p_k^2$$

$$z_0 = 59$$

$$z_{k+1} = z_k + 1 + \frac{1}{z_k}$$

- $\boldsymbol{z}_k = 4/\boldsymbol{p}_k - 1$, also $\boldsymbol{p}_k = 4/(\boldsymbol{z}_k + 1)$.

- $\boldsymbol{z}_k \geq 1 \Rightarrow$ (Induktion) $k < \boldsymbol{z}_k < 59 + 2k \Rightarrow \boldsymbol{z}_k = \Theta(k)$

$\Rightarrow \boldsymbol{p}_k = \Theta(1/k)$

- Daraus folgt (Logarithmusreihe):

$$P(n) \geq p_{2\log_2 n + O(1)} = \Theta\left(\frac{1}{\log n}\right)$$

- Wahrscheinlichkeit, einen bestimmten minimalen Schnitt zu finden, liegt in $\Omega(1/\log n)$.

Erfolgswahrscheinlichkeit Recursive-Contract

Theorem 3.4:

Alle minimalen Schnitte in einem gewichteten Graphen mit n Knoten und m Kanten können mit hoher Wahrscheinlichkeit in $O(n^2 \log^3 n)$ Zeit und $O(m \log (n^2/m))$ Platz gefunden werden.

Beweis:

Es existieren maximal $(n^2-n)/2$ minimale Schnitte in einem Graphen. Wird *Recursive-Contract* $O(\log^2 n)$-mal wiederholt, dann ist die Wahrscheinlichkeit durch $O(1/n^4)$ beschränkt, einen bestimmten minimalen Schnitt nicht zu finden. Also ist die Wahrscheinlichkeit, einen der maximal $(n^2-n)/2$ minimalen Schnitte zu verpassen, durch

$$O\left(\frac{n^2-n}{2n^4}\right) = O\left(\frac{1}{n^2}\right)$$

nach oben beschränkt.

Gesicherte logarithmische Laufzeit von *Random-Select*

- Betrachtung ist nötig, falls das Gesamtgewicht aller Kanten nicht mehr polynomiell in m ist.

Hier nur der Ansatz:

- Setze $\boldsymbol{N}=tm^4$ und wähle eine Zufallszahl s aus $[0,\boldsymbol{N}]$
- Wähle Kante i mit $\boldsymbol{W}_{i-1} \leq \boldsymbol{W}_m\, s/\boldsymbol{N} < \boldsymbol{W}_i$.
- Fehler tritt auf, wenn $\boldsymbol{W}_n\, s/\boldsymbol{N}$ und $\boldsymbol{W}_n\, (s+1)/\boldsymbol{N}$ verschiedene Kanten referieren.
- Die Wahrscheinlichkeit dafür liegt bei $m/\boldsymbol{N} = O(1/tm^3)$.

Ergebnisse:

- Berechnung minimaler Schnitte mittels Probabilismus ist in $O(n^2 \log^3 n)$ mit einer Fehlerwahrscheinlichkeit von $O(1/n^2)$ möglich.
- Verbesserung durch Parallelisierung auf geeigneten Maschinenmodellen.
- Berechnung von Mehrfachschnitten (Graph wird in $r > 2$ Partitionen aufgeteilt) in $O(n^{2(r-1)} \log^2 n)$ Zeit möglich.
- Berechnung von fast-minimalen Schnitte in einer α-Umgebung um den minimalen Schnitt (Wert liegt im Intervall $[c, \alpha c]$, $\alpha > 1$) in $O(n^{2\alpha} \log^2 n)$ Zeit möglich.