

Randomisierte Algorithmen

6. Graph-Algorithmen

Thomas Worsch

Fakultät für Informatik
Karlsruher Institut für Technologie

Wintersemester 2017/2018

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

6.1 Definition (Multigraph)

- ▶ *ungerichteter Multigraph* (V, E, v) mit
 - ▶ Knotenmenge V der Größe $n = |V|$
 - ▶ Kantenmenge E der Größe $m = |E|$
 - ▶ Abbildung $v : E \rightarrow \{\{x, y\} \mid x, y \in V\}$
- ▶ Es sind *Mehrfachkanten* möglich.
- ▶ In diesem Kapitel:
keine Schlingen, also $\forall e \in E : |v(e)| = 2$.

6.3 Repräsentation von Multigraphen

- ▶ «Adjazenzmatrix» A der Größe $n \times n$
Eintrag $A[x, y]$: Anzahl Kanten zwischen x und y :
$$A[x, y] = |v^{-1}(\{x, y\})|$$
- ▶ außerdem nützlich: Vektor M der Größe n :
Eintrag $M[x]$ gibt an, wieviele Kanten von x wegführen:
$$M[x] = \sum_y A[x, y]$$

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

6.4 Definition Schnitte

- ▶ Ein *Schnitt* in einem Multigraphen ist eine Partitionierung $V = C \cup \bar{C}$.
- ▶ *Größe des Schnittes* ist die Anzahl der Kanten $\{x, y\}$ mit $x \in C$ und $y \in \bar{C}$.

MINCUT

- ▶ Problem Instanz: ein Graph G
- ▶ Frage: Wie groß sind die minimalen Schnitte von G ?
- ▶ beste bekannte deterministische Algorithmen brauchen Laufzeit $O(mn + n^2 \log n) \subseteq O(n^3)$
 - ▶ für planare Graphen reicht $O(n(\log n)^2)$
- ▶ randomisiert?

6.5 Lemma

Wenn die Größe der minimalen Schnitte k ist, dann

- ▶ hat jeder Knoten mindestens Grad k
- ▶ ist die Anzahl Kanten $m \geq nk/2$

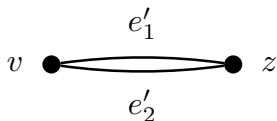
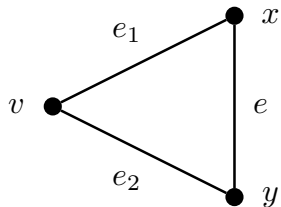
6.6 Beweis

- ▶ Wäre anderenfalls $\deg(x) < k$,
- ▶ dann hätte der Schnitt mit $C = \{x\}$ eine Größe echt kleiner k .

6.7 Definition Kontraktion

- ▶ In Multigraph G seien Knoten x und y durch eine Kante e mit $v(e) = \{x, y\}$ verbunden.
- ▶ G/e entsteht durch *Kontraktion der Kante e* :
 - ▶ Knotenmenge $V' = (V \setminus \{x, y\}) \cup \{z\}$, wobei z neu ist.
 - ▶ Kantenmenge E' ergibt sich aus E , indem
 - ▶ alle Kanten zwischen x und y entfernt werden,
 - ▶ jede Kante zwischen $v \in V \setminus \{x, y\}$ und x oder y ersetzen durch Kante zwischen v und z
 - ▶ alle anderen Kanten von G übernehmen

6.8 Beispiel



Es können Mehrfachkanten entstehen.

6.9 Lemma

Minimale Schnitte von G/e sind

$\left\{ \begin{array}{l} \text{mindestens?} \\ \text{genauso?} \\ \text{höchstens?} \end{array} \right\}$ so groß

wie minimale Schnitte von G .

6.9 Lemma

Minimale Schnitte von G/e sind

$\left\{ \begin{array}{l} \text{mindestens!} \\ \text{genauso?} \\ \text{höchstens?} \end{array} \right\}$ so groß

wie minimale Schnitte von G .

6.10 Beweis

- ▶ Betrachte minimalen Schnitt (K, \bar{K}) von G/e .
- ▶ Seine Größe sei k .
- ▶ O. B. d. A. seien die Endknoten x und y von e in K .
- ▶ Indem man diese beiden Knoten «aus der Kontraktion auspackt» und in der gleichen Partition belässt, erhält man einen Schnitt von G , dessen Größe ebenfalls k ist.
- ▶ Also haben die minimalen Schnitte von G *höchstens* Größe k .

6.11 Algorithmus

- ▶ Sei G Multigraph mit Kante e zwischen x und $y > x$.
- ▶ Wie in 6.3 vereinbart, sei
 - ▶ A «Adjazenzmatrix»
 - ▶ M Vektor mit $M[x] = \sum_y A[x, y]$
 - ▶ n, m Variablen für Knoten- bzw. Kantenzahl
- ▶ Die Datenstrukturen für G/e lassen sich wie folgt in Linearzeit berechnen:

6.11 Algorithmus (2a)

proc graph ← Kontraktion(**graph** G , **edge** e)

$x \leftarrow \min(v(e))$

$y \leftarrow \max(v(e))$ $\langle y \neq x$ da G schlingenfrei \rangle

\langle Idee: benutze Zeile/Spalte « x für z » \rangle

\langle und Zeile/Spalte « y für n » \rangle

$z \leftarrow x$

\langle Aktualisierung der Kantenzahlen \rangle

$m \leftarrow m - A[x, y]$

$M[z] \leftarrow M[x] + M[y] - 2 \cdot A[x, y]$

$M[y] \leftarrow M[n]$

6.11 Algorithmus (2b)

⟨Aktualisierung von Zeile/Spalte z:⟩

$$A[z, \cdot] \leftarrow A[x, \cdot] + A[y, \cdot]$$

$$A[\cdot, z] \leftarrow A[\cdot, x] + A[\cdot, y]$$

$$A[z, z] \leftarrow 0$$

⟨Aktualisierung von Zeile/Spalte y:⟩

$$A[y, \cdot] \leftarrow A[n, \cdot]$$

$$A[\cdot, y] \leftarrow A[\cdot, n]$$

⟨die bisherige Zeile n ist nun bedeutungslos⟩

⟨Aktualisierung der Knotenzahl⟩

$$n \leftarrow n - 1$$

return *⟨Graph, der zu den neuen Datenstrukturen gehört⟩*

6.12 Erster randomisierter Algorithmus

6.12 Erster randomisierter Algorithmus

Iterierte Kontraktion

6.12 Erster randomisierter Algorithmus

Iterierte Kontraktion

⟨Eingabe: ein Multigraph $G(V, E)$ ⟩

⟨Ausgabe: ein Schnitt (C, \bar{C}) ⟩

$H \leftarrow G$

while (H hat mehr als 2 Knoten) **do**

$e \leftarrow$ ⟨zufällig gleichverteilt gewählte Kante von H ⟩

$H \leftarrow$ Kontraktion(H, e)

od

$(C, \bar{C}) \leftarrow$ ⟨die Knotenmengen von G ,⟩

 ⟨die den Knoten von H entsprechen⟩

6.13 Satz

Algorithmus 6.12 kann so implementiert werden, dass die Laufzeit in $O(n^2)$ ist.

6.14 Beweis

- ▶ Jeder Aufruf von Kontraktion benötigt Laufzeit $O(n)$.
- ▶ Bei jedem Schleifendurchlauf wird die Anzahl der Knoten von H um 1 erniedrigt, d. h. es gibt $n - 2$ solche Durchläufe.
- ▶ Es bleibt zu implementieren:
 1. zufällige gleichverteilte Auswahl einer Kante des Multigraphen
 2. Beschaffung der Mengen C und \bar{C}

6.14 Beweis (2)

1. Zufällige Kantenwahl:

6.14 Beweis (2)

1. Zufällige Kantenwahl:

$i \leftarrow \text{random}(1, 2m)$ *(NB: jede Kante unten zweimal gezählt!)*

$x \leftarrow 0; s \leftarrow 0$

while $s < i$ **do**

$x \leftarrow x + 1$

$s \leftarrow s + M[x]$

od

$i \leftarrow i - (s - M[x])$

$y \leftarrow 0; s \leftarrow 0$

while $s < i$ **do**

$y \leftarrow y + 1$

$s \leftarrow s + A[x, y]$

od

(Wähle Kante zwischen x und y)

6.14 Beweis (3)

2. Beschaffung von C und \bar{C} :

- ▶ weitere Datenstruktur in der Prozedur Kontraktion
- ▶ boolesche Matrix Q mit so vielen Zeilen und Spalten wie der ursprüngliche Graph G Knoten hat.
- ▶ $Q[x, y] = 1$: Knoten mit Ursprungsnummern x und y wurden kontrahiert.
- ▶ Initialisierung: Einheitsmatrix.
- ▶

$$Q[x, \cdot] \leftarrow Q[x, \cdot] \vee Q[y, \cdot]$$
$$Q[y, \cdot] \leftarrow Q[n, \cdot]$$

- ▶ Am Ende nur noch zwei Knoten:
 - ▶ C z. B. durch 1-Einträge in Zeile 1 von Q gegeben
 - ▶ \bar{C} z. B. durch 0-Einträge in Zeile 1 von Q gegeben

6.15 Satz

Algorithmus 6.12 findet einen minimalen Schnitt mit einer

Wahrscheinlichkeit, die $\left\{ \begin{array}{c} \text{ziemlich groß?} \\ \text{mittel?} \\ \text{ziemlich klein?} \end{array} \right\}$ ist.

6.15 Satz

Algorithmus 6.12 findet einen minimalen Schnitt mit einer

Wahrscheinlichkeit, die $\left\{ \begin{array}{l} \text{ziemlich groß?} \\ \text{mittel?} \\ \text{in } \Omega(n^{-2}) \end{array} \right\}$ ist.

6.16 Beweis

in 3 Schritten

1. Es sei (C, \bar{C}) irgendein Schnitt. Behauptung:

Algorithmus 6.12 liefert diesen Schnitt als Ergebnis gdw keine seiner Kanten kontrahiert wird.

- ▶ sei e eine am Ende noch vorhandene Kante ursprünglich zwischen x und y
- ▶ also etwa $x \in C$ und $y \in \bar{C}$
- ▶ wäre e kontrahiert worden, müssten am Ende x und y zum gleichen Knoten von H gehören

6.16 Beweis (2)

2. Es sei (K, \bar{K}) ein minimaler Schnitt von G der Größe k .
 - ▶ Nach $i - 1$ Schleifendurchläufen sei noch keine Kante dieses Schnittes kontrahiert worden.
 - ▶ dann ist (K, \bar{K}) ein Schnitt des dann erhaltenen Graphen H_{i-1} und wegen Lemma 6.9 auch ein minimaler
 - ▶ Also enthält H_{i-1} noch mindestens $(n - i + 1)k/2$ Kanten.

Folglich:

- ▶ Wahrscheinlichkeit, dass als nächstes eine der k Kanten, die zum Schnitt K gehören, kontrahiert wird, höchstens $2/(n - i + 1)$;
- ▶ Wahrscheinlichkeit, dass keine dieser Kanten kontrahiert wird, mindestens $1 - 2/(n - i + 1)$.

6.16 Beweis (3)

3. Die Wahrscheinlichkeit, dass in keinem der Schritte eine der Kanten des Schnittes K kontrahiert wird, ist daher mindestens

$$\begin{aligned} \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1} \right) &= \frac{\prod_{i=1}^{n-2} (n-i-1)}{\prod_{i=1}^{n-2} (n-i+1)} \\ &= \frac{(n-2)!}{n!/2!} \\ &= \frac{(n-2)!2!}{n!} \\ &> \frac{2}{n^2} \in \Omega\left(\frac{1}{n^2}\right) \end{aligned}$$

Bemerkung

- ▶ Die Nicht-Fehler-Wahrscheinlichkeit von $1/n^2$ in Satz 6.15 ist klein.
- ▶ Der kleinste von $k = n^2/2$ Schnitten ist mit W.keit $1 - 1/e$ minimal:

$$\left(1 - \frac{2}{n^2}\right)^{n^2/2} < \frac{1}{e}$$

Weitere Rechnung zeigt: $k \in \Omega(n^2)$ erzwungen.

- ▶ Gesamtlaufzeit: $\Theta(n^4)$
im Gegensatz zu $O(n^3)$ im Deterministischen

Bemerkung

- ▶ Die Nicht-Fehler-Wahrscheinlichkeit von $1/n^2$ in Satz 6.15 ist klein.
- ▶ Der kleinste von $k = n^2/2$ Schnitten ist mit W.keit $1 - 1/e$ minimal:

$$\left(1 - \frac{2}{n^2}\right)^{n^2/2} < \frac{1}{e}$$

Weitere Rechnung zeigt: $k \in \Omega(n^2)$ erzwungen.

- ▶ Gesamtlaufzeit: $\Theta(n^4)$
im Gegensatz zu $O(n^3)$ im Deterministischen
- ▶ Problem: Bei kleinen Graphen werden zu oft «aus Versehen»
Kanten eines minimalen Schnittes kontrahiert.
- ▶ Besser: randomisiert auf t Knoten kontrahieren,
Rest deterministisch

6.17 Algorithmus

```
proc graph ← IterContract(graph  $G$ , int  $t$ )  
  ⟨Eingabe: ein Multigraph  $G(V, E, v)$  und⟩  
  ⟨      Endzahl  $t$  von Knoten⟩  
  ⟨Ausgabe: ein kontrahierter Graph  $H$  mit  $t$  Knoten⟩  
   $H \leftarrow G$   
  while ( $H$  hat mehr als  $t$  Knoten) do  
     $e \leftarrow \langle \text{zufällig gleichverteilt gewählte Kante von } H \rangle$   
     $H \leftarrow \text{Kontraktion}(H, e)$   
  od  
  return  $H$ 
```

6.18 Lemma

Die Wahrscheinlichkeit, dass bei Algorithmus 6.17 im Ergebnis H noch ein minimaler Schnitt des ursprünglichen Graphen «noch vorhanden» ist, ist mindestens

$$\binom{t}{2} / \binom{n}{2} .$$

6.19 Beweis

Abschätzung analog wie in Beweis 6.16:

$$\begin{aligned} \prod_{i=1}^{n-t} \left(1 - \frac{2}{n-i+1} \right) &= \frac{\prod_{i=1}^{n-t} (n-i-1)}{\prod_{i=1}^{n-t} (n-i+1)} \\ &= \frac{\prod_{i=1}^{n-t-2} (n-i-1)t(t-1)}{n(n-1) \prod_{i=3}^{n-t} (n-i+1)} \\ &= \frac{t(t-1)}{n(n-1)} \\ &= \binom{t}{2} / \binom{n}{2} \in \Omega((t/n)^2) \end{aligned}$$

6.20 Algorithmus

```
proc cut ← IterContractDetMinCut(graph  $G$ , int  $t$ )  
  ⟨Eingabe: ein Multigraph  $G(V, E, v)$  und⟩  
  ⟨      eine Knotenzahl  $t$ ⟩  
  ⟨Ausgabe: ein Schnitt von  $G$ ⟩  
   $C$  ← ⟨ein trivialer Schnitt⟩  
  for  $i$  ← 1 to  $n^2/t^2$  do  
     $H$  ← IterContract( $G, t$ )  
     $D$  ← DetMinCut( $H$ )  
     $C$  ← min( $C, D$ )  
  od  
  return  $C$ 
```

6.21 Lemma

Wählt man in Algorithmus 6.20 $t = n^{2/3}$, dann ist die Laufzeit in $O(n^{8/3})$ und die Wahrscheinlichkeit, einen minimalen Schnitt von G zu erhalten mindestens $1 - 1/e$.

6.22 Beweis

- ▶ Zeitbedarf ist kleiner gleich

$$\frac{n^2}{t^2} \cdot (n^2 + t^3) = \frac{n^4}{t^2} + n^2 t .$$

- ▶ Für gewähltes t beide Summanden größenordnungsmäßig in $O(n^{8/3})$.
- ▶ Fehlerwahrscheinlichkeit höchstens

$$\left(1 - \frac{t^2}{n^2}\right)^{n^2/t^2} < \frac{1}{e} .$$

6.23 Algorithmus

```
proc cut ← FastCut(graph  $G$ )  
  ⟨Eingabe: ein Multigraph  $G(V, E)⟩$   
  ⟨Ausgabe: ein Schnitt  $C$ ⟩  
  if ( $|V| \leq 6$ ) then  
     $C \leftarrow \langle \text{minimaler Schnitt, deterministisch ermittelt} \rangle$   
  else  
  
  fi
```


6.23 Algorithmus

```
proc cut  $\leftarrow$  FastCut(graph  $G$ )  
   $\langle$ Eingabe: ein Multigraph  $G(V, E)\rangle$   
   $\langle$ Ausgabe: ein Schnitt  $C$  $\rangle$   
  if ( $|V| \leq 6$ ) then  
     $C \leftarrow \langle$ minimaler Schnitt, deterministisch ermittelt $\rangle$   
  else  
     $t \leftarrow \lceil 1 + n/\sqrt{2} \rceil$   
     $H_1 \leftarrow$  IterContract( $G, t$ )  
     $H_2 \leftarrow$  IterContract( $G, t$ )  
     $C_1 \leftarrow$  FastCut( $H_1$ )  
     $C_2 \leftarrow$  FastCut( $H_2$ )  
     $C \leftarrow \min(C_1, C_2)$   
  fi
```

6.24 Satz

Alg. 6.23 hat Laufzeit $O(n^2 \log n)$.

6.25 Beweis

- ▶ maximale Rekursionstiefe: $\Theta(\log n)$
- ▶ beide Aufrufe von **IterContract** benötigen Zeit $O(n^2)$
- ▶ Gesamtzeitbedarf $T(n)$ für Eingabegraphen mit n Knoten

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

- ▶ daher $T(n) \in O(n^2 \log n)$ (Mastertheorem)

6.26 Satz

Alg. 6.23 liefert mit Wahrscheinlichkeit in $\Omega(1/\log n)$ minimalen Schnitt.

6.27 Beweis

- ▶ G Eingabegraph mit minimalen Schnitten der Größe k
- ▶ solcher Schnitt habe eine Reihe rekursiver Aufrufen von FastCut bis zu einer Stelle «überlebt».
- ▶ dann erreichter Graph heiße H
- ▶ Aufrufe von IterContract liefern Graphen H_1 und H_2
- ▶ Aufruf für H wird minimalen Schnitt für G liefern, falls für ein H_i gilt:
 1. Schnitt überlebt die Kontraktionen zur Konstruktion von H_i
 2. FastCut(H_i) findet einen minimalen Schnitt in H_i

6.27 Beweis (2)

Die Wahrscheinlichkeit für Punkt 1 ist nach Lemma 6.17 mindestens

$$\frac{\lceil 1 + t/\sqrt{2} \rceil (\lceil 1 + t/\sqrt{2} \rceil - 1)}{t(t-1)} \geq \frac{t/\sqrt{2} \cdot t/\sqrt{2}}{t(t-1)} \geq \frac{1}{2} \frac{t^2}{t(t-1)} \geq \frac{1}{2}.$$

6.27 Beweis (3)

Zu Punkt 2.:

- ▶ interessant: *untere Schranke* $p(r)$ der W.keit, dass FastCut r Niveaus über dem Rekursionsabbruch minimalen Schnitt findet
- ▶ $p(0) = 1$ und

$$p(r + 1) = 1 - \left(1 - \frac{1}{2} \cdot p(r)\right)^2 = p(r) - \frac{p(r)^2}{4}$$

- ▶ Substitution $q(r) = 4/p(r) - 1$ bzw. $p(r) = 4/(q(r) + 1)$:

$$\frac{4}{q(r + 1) + 1} = \frac{4}{q(r) + 1} - \frac{4}{(q(r) + 1)^2} = \frac{4q(r)}{(q(r) + 1)^2}$$

- ▶ und weiter $q(r + 1) = q(r) + 1 + \frac{1}{q(r)}$.

6.27 Beweis (4)

- ▶ Induktion: Für alle r gilt:

$$r < q(r) < r + 3 + H_{r-1}$$

- ▶ Offensichtlich ist $q(0) = 3$ und $r < q(r)$ für alle r .
- ▶ Weiter:

$$\begin{aligned} r + 1 < q(r) + 1 < q(r + 1) &= q(r) + 1 + \frac{1}{q(r)} \\ &< r + 3 + H_{r-1} + 1 + \frac{1}{r} \\ &= (r + 1) + 3 + H_r \end{aligned}$$

- ▶ Daher ist $q(r) \in r + O(\log r)$ und folglich $p(r) \in \Omega(1/r)$.
- ▶ Für Ausgangsgraphen mit n Knoten ist die Rekursionstiefe $\Theta(\log n)$
- ▶ folglich gesuchte Wahrscheinlichkeit $\Omega(1/\log n)$.

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

Problemstellung

Ein deterministischer Algorithmus für MST

F -leichte und F -schwere Kanten

Ein randomisierter MSF-Algorithmus

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

Problemstellung

Ein deterministischer Algorithmus für MST

F -leichte und F -schwere Kanten

Ein randomisierter MSF-Algorithmus

Problemstellung

Gegeben: Graph $G = (V, E)$

- ▶ zusammenhängend, ungerichtet
- ▶ Kanten e mit reellen Zahlen $w(e)$ gewichtet

Gesucht:

minimaler aufspannender Baum (minimum spanning tree, MST), d. h.

- ▶ Teilgraph von G , der
- ▶ Baum ist,
- ▶ G aufspannt und
- ▶ unter allen solchen Bäumen minimales Gewicht hat.

Falls G nicht zusammenhängend, existiert nur ein minimaler aufspannender Wald (minimum spanning forest, MSF).

Allgemeine Annahme

O. B. d. A. seien alle Kantengewichte paarweise verschieden.

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

Problemstellung

Ein deterministischer Algorithmus für MST

F-leichte und *F*-schwere Kanten

Ein randomisierter MSF-Algorithmus

Definition

Eine Kante e in einem gewichteten Graphen G heißt *lokal minimal*, wenn für einen Endpunkt x von e gilt:

Von allen Kanten, die mit x inzidieren, hat e das kleinste Gewicht.

Lemma

Jede lokal minimale Kante gehört zu einem MST von G .

Beweis

- ▶ $e = \{x, y\}$ lokal minimale Kante eines Knotens x .
- ▶ T' ein aufspannender Baum von G , der *nicht* e enthalte.

Zeige: T' hat nicht minimales Gewicht.

lautes Nachdenken an der Tafel

Beweis

- ▶ $e = \{x, y\}$ lokal minimale Kante eines Knotens x .
- ▶ T' ein aufspannender Baum von G , der *nicht* e enthalte.

Zeige: T' hat nicht minimales Gewicht.

- ▶ Sei e' die Kante von T' ,
 - ▶ die von x wegführt und
 - ▶ in T' auf dem kürzesten Weg von x nach y liegt.
- ▶ betrachte $T = T' - e' + e$
- ▶ Behauptungen
 1. $w(T) < w(T')$.
 2. T spannt G auf.
 3. T ist ein Baum.

Korollar

Die lokal minimalen Kanten eines Graphen bilden ein Wald.

Borůvka-Phase

Algorithmus, der folgendes leistet:

1. berechne Menge $L(G)$ aller lokal minimalen Kanten von G
2. berechne den kontrahierten Graphen $B(G) = G/L(G)$
 - ▶ entstünden Mehrfachkanten,
wird nur die leichteste Einzelkante behalten

Lemma

Eine Borůvka-Phase kann man in Zeit $O(m + n)$ implementieren.

Beweis: Übung.

Lemma

Durch eine Borůvka-Phase wird die Anzahl der Knoten um mindestens die Hälfte reduziert.

Beweis

- ▶ jede Kante für höchstens zwei Knoten lokal minimal
- ▶ also pro Borůvka-Phase mindestens $n/2$ Kanten entfernt
- ▶ jede Kantenkontraktion entfernt auch ein Knoten

Lemma

Für jeden Graphen G gilt:

Die Kanten in $L(G)$ und die Kanten eines MST von $B(G)$ bilden zusammen einen MST von G .

Beweis

- ▶ Jeder aufspannende Baum T von G , der $L(G)$ «enthält», induziert einen aufspannenden Baum T' von $B(G)$ mit $w(T') = w(T) - w(L(G))$.
- ▶ Jeder aufspannende Baum T' von $B(G)$ induziert einen aufspannenden Baum \bar{T}' von G , der $L(G)$ «enthält», mit $w(T) = w(T') + w(L(G))$.
- ▶ Sei T *minimaler* aufspannender Baum von G und $B(T)$ der zugehörige aufspannende Baum von $B(G)$.
- ▶ Wäre $B(T)$ nicht minimal für $B(G)$, sondern etwa T' ,
- ▶ so hätte der dadurch induzierte aufspannende Baum \bar{T}' von G nur Gewicht $w(\bar{T}') = w(T') + w(L(G)) < w(B(T)) + w(L(G)) = w(T)$ im Widerspruch zur Minimalität von T .

Korollar

Wenn alle Kantengewichte paarweise verschieden sind, ist der MST eindeutig.

Borůvka's MST-Algorithmus

- ▶ Borůvka-Phasen bis nur noch 1 Knoten im Graph
- ▶ führe Buch, welche Kanten jeweils kontrahiert werden
- ▶ vorangegangene Lemmata: klar, welche Kanten MST bilden

Lemma

Borůvkas MST-Algorithmus benötigt $O(m \log n)$ Zeit.

Beweis

- ▶ $O(\log n)$ viele Phasen,
- ▶ von denen jede $O(m + n)$ Zeit benötigt
- ▶ In zusammenhängenden Graphen ist $n \in O(m)$.

Andere deterministische Algorithmen



Bernard Chazelle

A Minimum Spanning Tree Algorithm with Inverse-Ackermann
Type Complexity

Journal of the ACM, 47:1028–1047, 2000.

- ▶ Laufzeit $O(m\alpha(m, n))$
- ▶ mit $\alpha(m, n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log_2 n\}$

Andere deterministische Algorithmen



Bernard Chazelle

A Minimum Spanning Tree Algorithm with Inverse-Ackermann Type Complexity

Journal of the ACM, 47:1028–1047, 2000.

- ▶ Laufzeit $O(m\alpha(m, n))$
- ▶ mit $\alpha(m, n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log_2 n\}$



Seth Pettie, Vijaya Ramachandran

An optimal minimum spanning tree algorithm

Journal of the ACM, 49:16–34, 2002.

Andere deterministische Algorithmen



Bernard Chazelle

A Minimum Spanning Tree Algorithm with Inverse-Ackermann Type Complexity

Journal of the ACM, 47:1028–1047, 2000.

- ▶ Laufzeit $O(m\alpha(m, n))$
- ▶ mit $\alpha(m, n) = \min\{i \geq 1 : A(i, \lfloor m/n \rfloor) \geq \log_2 n\}$



Seth Pettie, Vijaya Ramachandran

An optimal minimum spanning tree algorithm

Journal of the ACM, 49:16–34, 2002.

*«Although our time bound is optimal,
the exact function describing it is not known at present.»*

Inverse Ackermann-Funktion(en)

siehe auch

- ▶ Folien von Raimund Seidel:

<http://cgi.di.uoa.gr/~ewcg06/invited/Seidel.pdf>

- ▶ WWW-Beitrag von Gabriel Nivasch:

<http://www.gabrielnivasch.org/fun/inverse-ackermann>

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

Problemstellung

Ein deterministischer Algorithmus für MST

F -leichte und F -schwere Kanten

Ein randomisierter MSF-Algorithmus

Definition

- ▶ sei F ein Wald in G (beliebig, nicht notwendig MSF)
- ▶ v_1 und v_2 zwei Knoten von G und
- ▶ $P(v_1, v_2)$ die Menge der Kanten des kürzesten Pfades in F von v_1 nach v_2 (oder $P = \emptyset$)
- ▶ Definiere

$$W_F(v_1, v_2) = \begin{cases} \max\{w(e) \mid e \in P(v_1, v_2)\} & \text{falls } v_1, v_2 \text{ im} \\ & \text{gleichen Baum} \\ \infty & \text{sonst} \end{cases}$$

- ▶ Kante $e = \{v_1, v_2\}$ heißt F -schwer, falls $w(e) > W_F(v_1, v_2)$ ist, und
- ▶ sie heißt F -leicht, falls $w(e) \leq W_F(v_1, v_2)$.

Lemma

- ▶ Es sei
 - ▶ F' ein beliebiger(!) Wald von G und
 - ▶ $e = \{v_1, v_2\}$ eine Kante in G .
- ▶ Wenn e F' -schwer, dann gehört e nicht zum MST von G .
- ▶ äquivalent:
Wenn eine Kante zum MST gehört, dann ist sie F' -leicht.
 - ▶ wohlgemerkt für beliebigen Wald F'
- ▶ äquivalent:
Die schwerste Kante eines Kreises gehört nie zum MST.

Beweis

- ▶ Kante $e = \{v_1, v_2\}$ sei F' -schwer und
- ▶ es bezeichne P den Pfad von v_1 nach v_2 , dessen Kanten alle leichter sind als die Kante e selbst.

- ▶ Bei Bestimmung des MST mit Borůvkas Algorithmus
- ▶ wird e *nie* lokal minimale Kante von v_1 oder v_2 sein, sondern immer eine der Kanten von P .
- ▶ Also wird e auch nie zum (eindeutigen) MST hinzugenommen.

Lemma

Ein aufspannender Baum T eines Graphen G hat minimales Gewicht, wenn die einzigen T -leichten Kanten in G die Kanten von T sind.

Beweis

- ▶ Wenn die einzigen T -leichten Kanten in G die Kanten von T sind,
- ▶ dann sind alle Kanten, die nicht zu T gehören T -schwer.
- ▶ Also gehören sie also sicher nicht zum MST von G .
- ▶ Also besteht der MST nur aus Kanten, die zu T gehören.
- ▶ Von ihnen kann man aber auch keine weglassen, da es dann kein aufspannender Baum von G mehr ist.

Satz (Dixon (1992), King (1997))

Zu einem Graphen G und einem Wald F in G kann man alle F -schweren Kanten von G in Zeit $O(m + n)$ finden.

Beweis: nicht leicht.

Wir werden den Satz einfach verwenden.

Überblick

Einleitung

Minimale Schnitte

Minimale aufspannende Bäume

Problemstellung

Ein deterministischer Algorithmus für MST

F -leichte und F -schwere Kanten

Ein randomisierter MSF-Algorithmus

Zufällige Teilgraphen

$\text{RandomSample}(G, p)$ sei eine Funktion, die

- ▶ einen Graphen G' mit den gleichen Knoten wie in G liefert,
- ▶ bei dem jede Kante von G unabhängig mit Wahrscheinlichkeit p zu G' hinzugenommen wird.

Lemma

- ▶ Wenn die Kanten eines Graphen G nach aufsteigendem Gewicht sortiert vorliegen,
- ▶ dann kann zu gegebenen p ein MSF F' für ein $G' = \text{RandomSample}(G, p)$ konstruiert werden,
- ▶ indem man jede Kante von G nur einmal betrachtet.

Achtung:

- ▶ Wir werden dieses Vorgehen nicht in unserem letztendlichen Algorithmus verwenden,
- ▶ sondern nur, um das Lemma nach diesem zu beweisen.

Beweis

- ▶ Kanten e_1, \dots, e_m nach aufsteigendem Gewicht geordnet
- ▶ konstruiere gleichzeitig
 $G' = \text{RandomSample}(G, p)$ und MSF F' von G'

$F'_0 \leftarrow \emptyset; k \leftarrow 0$

for $i \leftarrow 1$ **to** m **do**

„betrachte“ Kante e_i

if $\text{RandomFloat}(0, 1) \leq p$ **then**

⟨ nimm $e_i = \{u, v\}$ zu G' hinzu ⟩

lautes Nachdenken an der Tafel

fi

fi

od

Beweis

- ▶ Kanten e_1, \dots, e_m nach aufsteigendem Gewicht geordnet
- ▶ konstruiere gleichzeitig
 $G' = \text{RandomSample}(G, p)$ und MSF F' von G'

$F'_0 \leftarrow \emptyset; k \leftarrow 0$

for $i \leftarrow 1$ **to** m **do**

„betrachte“ Kante e_i

if $\text{RandomFloat}(0, 1) \leq p$ **then**

⟨nimm $e_i = \{u, v\}$ zu G' hinzu⟩

if *⟨ u, v in versch. Zshg.komp. von F'_{k-1} ⟩* **then**

$k \leftarrow k + 1; \quad F'_k \leftarrow F'_{k-1} + e_i$

fi

fi

od

$F' \leftarrow F'_k$ *⟨ist der MSF⟩*

Beweis (2)

- ▶ Offensichtlich wird ein Wald F' konstruiert.
- ▶ zeige: F' ist MSF (mit vorigem Lemma)
- ▶ zeige: jede F' -leichte Kante von G' ist in F' .
- ▶ **Annahme:** $e = \{v_1, v_2\}$ ist F' -leichte Kante von G' , die **nicht in F'** ist.
- ▶ Zwei Möglichkeiten:
 1. v_1 und v_2 liegen in verschiedenen Bäumen von F' .
 2. v_1 und v_2 liegen in einem Baum T' von F' , eine Kante e' auf Pfad $v_1 \rightsquigarrow v_2$ in T' hat größeres Gewicht als e

Beweis (2)

- ▶ Offensichtlich wird ein Wald F' konstruiert.
- ▶ zeige: F' ist MSF (mit vorigem Lemma)
- ▶ zeige: jede F' -leichte Kante von G' ist in F' .
- ▶ **Annahme:** $e = \{v_1, v_2\}$ ist F' -leichte Kante von G' , die **nicht in F'** ist.
- ▶ Zwei Möglichkeiten:
 1. v_1 und v_2 liegen in verschiedenen Bäumen von F' .
 2. v_1 und v_2 liegen in einem Baum T' von F' , eine Kante e' auf Pfad $v_1 \rightsquigarrow v_2$ in T' hat größeres Gewicht als e
- ▶ in *beiden* Fällen galt, als e betrachtet wurde und ein F'_k vorlag:
 - ▶ v_1 und v_2 lagen in verschiedenen Bäumen von F'_k .
 - ▶ Dann wäre aber e zu F'_k hinzugenommen worden.

Lemma

- ▶ Sei F' der MSF von $G' = \text{RandomSample}(G, p)$.
- ▶ Dann ist der Erwartungswert für die Anzahl der F' -leichten Kanten von G (!) kleiner oder gleich n/p .

Beweis (1)

- ▶ betrachte vorangegangenen Algorithmus
- ▶ Wenn Kante e von G am Ende
 - ▶ F' -leicht ist, dann war sie auch schon F'_k -leicht zu dem Zeitpunkt, zu dem sie „betrachtet“ wurde
 - ▶ F' -schwer ist, dann war sie auch schon F'_k -schwer zu dem Zeitpunkt, zu dem sie „betrachtet“ wurde

Beweis (2)

- ▶ *Phase* $i \geq 1$: Schleifendurchläufe solange $k = i - 1$
- ▶ Während Phase i hat jede Kante von G , die F'_{i-1} -leicht ist, Wahrscheinlichkeit p , dass sie zu G' und folglich zu F'_{i-1} hinzu genommen wird.
- ▶ mit erster Wahl einer solchen Kante endet die Phase
- ▶ Anzahl der in Phase i betrachteten F'_{i-1} -leichten Kanten geometrisch verteilt mit Parameter p
- ▶ Erwartungswert ist $1/p$

Beweis (3)

- ▶ Am Ende ist $|F'| = k \leq n - 1$, d. h. es ist $k \leq n - 1$ mal die Bedingung $\text{RandomFloat}(0, 1) < p$ erfüllt.
- ▶ Berücksichtigung der restlichen F' -leichten Kanten, die alle *nicht* mehr zu F' hinzugenommen werden:
 - ▶ **for**-Schleife weitermachen, bis n mal „ $\text{RandomFloat}(0, 1) < p$ “ wahr
 - ▶ Erwartungswert für Anzahl Schritte jeweils $1/p$ bis einmal der Fall
 - ▶ ZV Y : *insgesamt* notwendige Schleifendurchläufe
- ▶ Y dominiert interessierende Anzahl Schleifendurchläufe.
- ▶ $E[Y] = n/p$.

Randomisierter Algorithmus für MSF-Berechnung

proc $F \leftarrow \text{MSF}(G)$:

1. $G_1 \leftarrow B(B(B(G)))$
 $C_1 \leftarrow \langle \text{Kanten, die bei Berechnung von } G_1 \text{ kontrahiert} \rangle$
if $\langle G_1 \text{ enthält keine Kanten mehr} \rangle$ **then**
 return C_1
fi
2. $G_2 \leftarrow \text{RandomSample}(G_1, 1/2)$
3. $F_2 \leftarrow \text{MSF}(G_2)$
4. $C_2 \leftarrow \langle \text{die } F_2\text{-schweren Kanten in } G_1 \rangle$
 $G_3 \leftarrow \langle G_1 \text{ ohne die Kanten in } C_2 \rangle$
5. $F_3 \leftarrow \text{MSF}(G_3)$
6. $F \leftarrow C_1 \cup F_3$
return F

Satz

Der obige Algorithmus berechnet den MSF des Eingabegraphen G .

Beweis

- zu 1. Die Kanten aus C_1 gehören zum MSF von G .
Enthält G_1 keine Kanten, dann bildet Kantenmenge C_1 den MSF.
- zu 3. F_2 ist ein Wald in G_2 und folglich auch in G_1 und G .
- zu 4. Die F_2 -schweren Kanten von G_1 gehören nicht zum MSF von G_1 .
Folglich hat G_3 den gleichen MSF wie G_1 ,
- zu 5. der als F_3 berechnet wird.
- zu 6. Also ist $C_1 \cup F_3$ ein MSF und wegen Zusammenhang sogar ein MST von G .

Satz

Der Erwartungswert für die Laufzeit des Algorithmus ist $O(m + n)$.

Beweis

$T(n, m)$: Erwartungswert der Laufzeit des Algorithmus für Graphen mit n Knoten und m Kanten.

1. $G_1 \leftarrow B(B(B(G)))$: Laufzeit ist $O(m + n)$.
 G_1 hat höchstens $n/8$ Knoten und m Kanten.
2. $G_2 \leftarrow \text{RandomSample}(G_1, 1/2)$: Laufzeit ist $O(m + n)$.
 G_2 hat $\leq n/8$ Knoten und erwartete Kantenzahl $m/2$.
3. $F_2 \leftarrow \text{MSF}(G_2)$: erwartete Laufzeit ist $T(n/8, m/2)$
4. $C_2 \leftarrow \langle \text{die } F_2\text{-schweren Kanten in } G_1 \rangle$: Laufzeit ist $O(m + n)$.
 G_3 hat $\leq n/8$ Knoten; erwartete Kantenzahl $(n/8)/(1/2) = n/4$.
5. $F_3 \leftarrow \text{MSF}(G_3)$: erwartete Laufzeit $T(n/8, n/4)$
6. $F \leftarrow C_1 \cup F_3$: Laufzeit ist $O(n)$.

$$T(n, m) \leq T(n/8, m/2) + T(n/8, n/4) + c(n + m) .$$

$$\leadsto T(n, m) \in O(n + m).$$