6 Graph-Algorithmen

6.1 Definition Ein *Multigraph* ist eine Struktur (V, E, ν), die aus einer Menge V von Knoten und einer Menge E von Kanten besteht und einer Funktion ν, die jeder Kante die Menge ihrer zwei (nicht notwendig verschiedenen) Endpunkte zuweist.

Bei Multigraphen können zwei Knoten also durch mehrere, wohl unterschiedene Kanten miteinander verbunden sein. Man spricht auch von *Mehrfachkanten*. ◇

In diesem Kapitel sind alle Graphen Multigraphen ohne Schlingen mit *ungerichteten* Kanten.

6.2 Im Folgenden bezeichnen wir stets mit n die Anzahl der Knoten und mit m die Anzahl der Kanten eines Multigraphen. Im Gegensatz zu „normalen“ Graphen ist also m im allgemeinen *nicht* durch $O(n^2)$ nach oben beschränkt.

6.3 Multigraphen kann man mit Datenstrukturen repräsentieren, die leichte Erweiterungen von Adjazenzmatrizen bzw. Adjazenzlisten sind, wie man sie für normale Graphen kennt.

In der Adjazenzmatrix A speichert der Eintrag $A[x, y]$, wie viele Kanten es zwischen x und y gibt. Analog kann man bei Adjazenzlisten vorgehen. Weil es weiter unten von Vorteil ist, soll zusätzlich in einer Variablen n die Anzahl Knoten, in einer Variablen m die Gesamtzahl Kanten und in einem Vektor M für jeden Knoten die Zahl der von ihm ausgehenden Kanten gespeichert sein. Für alle x gelte also $M[x] = \sum_y A[x, y]$ und es sei $\sum_x M[x] = 2m$.

Zeilen und Spalten seien jeweils von 1 bis n durchnummeriert.

6.1 Minimale Schnitte

In diesem Abschnitt beschäftigen wir uns mit dem Problem, minimale Schnitte in Multigraphen zu berechnen.

6.4 Definition Ein *Schnitt* in einem Multigraphen $G(V, E)$ ist durch eine Partitionierung $V = C \cup \bar{C}$ der Knotenmenge in zwei diskunkte Teilmengen gegeben. Die *Größe* eines Schnittes ist die Anzahl der Kanten $\{x, y\}$ mit $x \in C$ und $y \in \bar{C}$.

Ein minimaler Schnitt ist ein Schnitt mininaler Größe. ◇

6.5 Lemma. Wenn in einem Multigraphen die minimalen Schnitte Größe k haben, dann ist auch der Grad jedes Knotens mindestens k. Für die Anzahl der Kanten gilt folglich: $m \geqslant nk/2$.

6.6 Beweis. Andernfalls würde die Partitionierung, die in C nur einen Knoten x mit Grad kleiner als k enthält, und in $\bar{C} = V \setminus \{x\}$ alle anderen Knoten, einen Schnitt mit Größe kleiner als k liefern. ■

6.7 Definition Es sei G ein Multigraph und x und y zwei Knoten von G, die durch mindestens eine Kante e miteinander verbunden sind. Der durch die *Kontraktion* von e entstehende Multigraph G/e wie folgt definiert:

- Seine Knotenmenge ist $V' = (V \setminus \{x, y\}) \cup \{z\}$, wobei z ein neuer Knoten ist.
- Seine Kantenmenge ergibt sich aus E, indem
  - alle Kanten zwischen x und y entfernt werden,
  - jede Kante, die in G einen Knoten $v \in V \setminus \{x, y\}$ mit einem Knoten aus $\{x, y\}$ verband, durch eine Kante zwischen v und z ersetzt wird und
  - alle anderen Kanten von G übernommen werden.

Falls man nacheinander alle Kanten einer Kantenteilmenge $X \subset E$ kontrahiert, so ist die Reihenfolge, in der die Kanten kontrahiert für das Ergebnis irrelevant. Daher schreibt man in diesem Fall dann einfach G/X für den kontrahierten Graphen. ◇

Bei der Kontraktion einer Kante verringert sich die Knotenzahl um genau 1.

6.8 Durch die Kontraktion einer Kante können (zusätzliche) Mehrfachkanten entstehen. Abbildung 6.1 zeigt ein einfaches Beispiel.

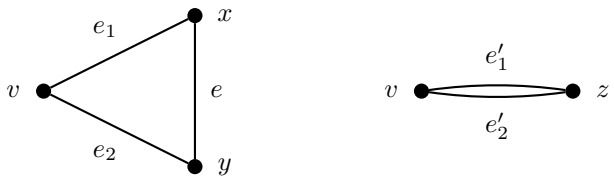

Abbildung 6.1: Durch Kontraktion der Kante e entstehen Mehrfachkanten.

6.9 Lemma. Ist G ein Multigraph mit einer Kante e, so sind die minimalen Schnitte von G/e mindestens so groß wie die von G.

6.10 Beweis. Man betrachte einen 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 Teilmenge 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. (Kontraktion) Es sei A die Adjazenzmatrix eines Multigraphen G und e eine Kante zwischen x und $y > x$. Die Datenstrukturen für G/e lassen sich wie folgt berechnen:

proc graph ← Kontraktion(**graph** G, **edge** e)
⟨*Idee: benutze Zeile/Spalte* x *für den kontrahierten Knoten*⟩
⟨ *und Zeile/Spalte* y *für das, was bislang in Zeile* n *stand*⟩
⟨*Aktualisierung der Kantenzahlen*⟩
$m \leftarrow m - A[x, y]$
$M[x] \leftarrow M[x] + M[y] - 2 \cdot A[x, y]$
$M[y] \leftarrow M[n]$
⟨*Aktualisierung von Zeile/Spalte* x*:*⟩
$A[x, \cdot] \leftarrow A[x, \cdot] + A[y, \cdot]$
$A[\cdot, x] \leftarrow A[\cdot, x] + A[\cdot, y]$
$A[x, x] \leftarrow 0$
⟨*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*⟩
$n \leftarrow n - 1$
return ⟨*Graph, der zu den neuen Datenstrukturen gehört*⟩

Dieser Algorithmus benötigt offensichtlich Laufzeit $O(n)$.

6.12 Algorithmus. (Iterierte Kontraktion)
⟨*Eingabe: ein Multigraph* $G(V, E)$⟩
⟨*Ausgabe: ein Schnitt* C⟩
$H \leftarrow G$
while (H hat mehr als 2 Knoten und mindestens 1 Kante) **do**
$e \leftarrow$⟨*zufällig gleichverteilt gewählte Kante von* H⟩
$H \leftarrow$ Kontraktion(H, e)
od
$(C, \bar{C}) \leftarrow$ ⟨*die beiden Mengen (von Knoten von* G*), die den beiden Knoten von* H *entsprechen*⟩

6.13 Satz. *Algorithmus 6.12 kann so implementiert werden, dass seine Laufzeit in* $O(n^2)$ *ist.*

6.14 Beweis. Jeder Aufruf der Funktion 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 zeigen, dass der Algorithmus so implementiert werden kann,

1. dass man gleichverteilt zufällig eine Kante des Multigraphen auswählen kann, und
2. dass die Mengen C und $\bar{C}$ hinreichend schnell beschafft werden können.

Das geht zum Beispiel so:

1.
$i \leftarrow$ **random**$(1, 2m)$ ⟨*Beachte: jede Kante wird gleich 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⟩

2. Man führt in der Prozedur Kontraktion zusätzlich eine weitere Datenstruktur mit. Es handelt sich um eine boolesche Matrix Q mit so vielen Zeilen und Spalten wie der ursprüngliche Graph G Knoten hat. Eine 1 als Eintrag in $Q[x, y]$ bedeutet, dass die Knoten, die ursprünglich die Nummern x und y hatten, durch Kontraktionen zusammengefasst wurden. Initialisiert werden muss die Matrix also mit der Einheitsmatrix. Algorithmus 6.11 sollte also unmittelbar vor der letzten Zuweisung $n \leftarrow n - 1$ um die folgenden Zeilen erweitert werden:

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

Da am Ende sicher nur noch zwei zusammengefasste Knoten existieren, ist die Bestimmung von C und $\bar{C}$ leicht: C ist z. B. durch die 1-Einträge in der ersten Zeile von Q gegeben und $\bar{C}$ durch die 0-Einträge dieser Zeile.

■

6.15 Satz. *Algorithmus 6.12 findet mit Wahrscheinlichkeit $\Omega(n^{-2})$ einen minimalen Schnitt.*

6.16 Beweis. Wir führen den Beweis in drei Schritten:

1. Es sei $(C, \bar{C})$ irgendein Schnitt des ursprünglichen Graphen (der noch keine Mehrfachkanten hat). Wir behaupten, dass Algorithmus 6.12 genau dann diesen Schnitt als Ergebnis liefert, wenn keine der Kanten, die über ihn läuft, während des Algorithmus kontrahiert wird: Es sei e eine am Ende noch vorhandene Kante, „die“ ursprünglich zwei Knoten x und y verband. Dann ist nun etwa $x \in C$ und $y \in \bar{C}$. Wäre die Kante e kontrahiert worden, müssten aber x und y am Ende zum gleichen Knoten von H gehören.

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 also $(K, \bar{K})$ auch ein Schnitt des dann erhaltenen Graphen H_{i-1} und wegen Lemma 6.9 muss es auch ein minimaler Schnitt sein. Also enthält H_{i-1} noch mindestens $(n-i+1)k/2$ Kanten. Folglich ist die Wahrscheinlichkeit, dass als nächstes eine der k Kanten, die zum Schnitt K gehören, kontrahiert wird, höchstens $2/(n-i+1)$ und die Wahrscheinlichkeit, dass keine dieser Kanten kontrahiert wird, mindestens $1-2/(n-i+1)$.

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

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

■

Die Erfolgs-Wahrscheinlichkeit von $2/n^2$ in Satz 6.15 ist natürlich sein klein. Um sie zu vergrößern, kann man den mehrfach ausführen und einen kleinsten der dabei auftretenden Schnitte als Ergebnis wählen.

Wählt man $k = n^2/2$ Wiederholungen, ist die Wahrscheinlichkeit, keinen minimalen Schnitt zu finden, noch

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

Weitere Rechnungen zeigen, dass $k \in \Omega(n^2)$ Wiederholungen auch notwendig sind, um konstante Fehlerwahrscheinlichkeit zu erreichen. Die resultierende Gesamtlaufzeit ist damit in $\Theta(n^4)$ im Gegensatz zu $O(n^3)$ für den schnellsten bekannten deterministischen Algorithmus.

Das Problem besteht im Grunde darin, dass die Wahrscheinlichkeit, „aus Versehen“ eine Kante des minimalen Schnittes zu kontrahieren zu groß ist, wenn der Graph schon weit kontrahiert wurde. Einen besseren Algorithmus erhält man, wenn randomisiert nur bis zu einem Graphen mit t Knoten kontrahiert, und dann deterministisch weiter arbeitet.

6.17 Algorithmus. (Beschränkte iterierte Kontraktion)
proc graph ← IterContract(**graph** G, **int** t)
⟨*Eingabe: ein Multigraph* $G(V, E, \nu)$ *und Anzahl* t *von Knoten am Ende*⟩
⟨*Ausgabe: ein kontrahierter Graph* H⟩
H ← G
while (H hat mehr als t Knoten) **do**
e ←⟨*zufällig gleichverteilt gewählte Kante von* H⟩
H ← Kontraktion(H, e)
od
return H

6.18 Lemma. Dei Wahrscheinlichkeit, dass bei Algorithmus 6.17 im Ergebnisgraphen H noch ein minimaler Schnitt des ursprünglichen Graphen „vorhanden“ ist, ist mindestens

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

6.19 Beweis. Eine Abschätzung analog wie in Beweis 6.16 ergibt in diesem Fall:

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

■

6.20 Algorithmus.
proc cut ← IterContractDetMinCut(**graph** G, **int** t)
⟨*Eingabe: ein Multigraph* $G(V, E, \nu)$ *und Anzahl* t *von Knoten für Zwischengraphen*⟩
⟨*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. Der Zeitbedarf ist offensichtlich

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

Für den gewählten Wert von t sind beide Summanden größenordungsmäßig in $O(n^{8/3})$.
Die Fehlerwahrscheinlichkeit ist höchstens

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

■

Man kann das Problem aber noch schneller lösen. Die Idee besteht darin, für im Laufe der Berechnung erhaltene kontrahierte Zwischengraphen umso häufiger (randomisiert) nach einem minimalen Schnitt zu suchen, je kleiner diese Graphen schon sind.

6.23 Algorithmus. (Minimaler Schnitt schnell)
proc cut ← FastCut(**graph** G):
⟨*Eingabe: ein Multigraph* $G(V, E, \nu)$⟩
⟨*Ausgabe: ein Schnitt* C⟩
if $(|V| \leqslant 6)$ **then**
  $C \leftarrow$ ⟨*minimaler Schnitt, mittels vollständiger Suche ermittelt*⟩
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)
  $C \leftarrow$ ⟨*der kleinere der beiden Schnitte* C_1 *und* C_2⟩
fi

6.24 Satz. *Algorithmus 6.23 hat Laufzeit* $O(n^2 \log n)$.

6.25 Beweis. Die maximale Rekursionstiefe ist $\Theta(\log n)$. Die beiden Aufrufe von **IterContract** benötigen eine Zeit in $O(n^2)$. (In dieser Zeit könnten sogar Kontraktionen auf Graphen mit 2 Knoten durchgeführt werden.) Also ist der Gesamtzeitbedarf $T(n)$ für Eingabegraphen mit n Knoten

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

Hieraus ergibt sich $T(n) \in O(n^2 \log n)$. ∎

6.26 Satz. *Algorithmus 6.23 liefert mit einer Wahrscheinlichkeit in* $\Omega(1/\log n)$ *einen minimalen Schnitt.*

6.27 Beweis. Es sei G ein Eingabegraph für den Algorithmus, dessen minimale Schnitte die Größe k haben. Ein solcher Schnitt möge eine Reihe von rekursiven Aufrufen von FastCut bis zu einer Stelle „überlebt" haben; der dann erreichte Graph heiße H. Die durch die Aufrufe von **IterContract** daraus resultierenden Graphen seien H_1 und H_2. Der Aufruf für H wird als Ergebnis einen minimalen Schnitt für G liefern, falls für ein H_i gilt:

1. Der Schnitt überlebt die Kontraktionen zur Konstruktion von H_i.
2. FastCut(H_i) findet einen minimalen Schnitt in H_i.

Die Wahrscheinlichkeit für Punkt 1 ist nach Lemma 6.18 mindestens

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

Zu Punkt 2.: Wir interessieren uns nun für eine *untere Schranke* $p(r)$ der Wahrscheinlichkeit, dass FastCut r Niveaus über dem Rekursionsabbruch einen minimalen Schnitt findet. Man mache sich klar, dass die wie folgt definierten Werte solche untere Schranken darstellen: $p(0) = 1$ und

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

Setzt man $q(r) = 4/p(r) - 1$, bzw. $p(r) = 4/(q(r)+1)$, so ergibt sich hieraus

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

und weiter

$$q(r+1) = q(r) + 1 + \frac{1}{q(r)}$$

Per Induktion kann man leicht zeigen, dass für alle r gilt: $r < q(r) < r+3+H_{r-1}$. (Zur Erinnerung: Die r-te harmonische Zahl ist $H_r = \sum_{i=1}^{r} 1/i$.) Denn es gilt:

$$r+1 < q(r)+1 < q(r+1) = q(r) + 1 + \frac{1}{q(r)} < r+3+H_{r-1}+1+\frac{1}{r} = (r+1)+3+H_r\ .$$

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

6.2 Minimale spannende Bäume

Ausgangspunkt für die in diesem Abschnitt betrachtete Problemstellung sind ungerichtete Graphen, deren Kanten e mit reellen Zahlen $w(e)$ gewichtet sind. Das *Gewicht* $w(T)$ eines Teilgraphen T ist die Summe der Gewichte der Kanten, die zu T gehören.

Mit n wird wieder die Anzahl der Knoten und mit m die Anzahl der Kanten von G bezeichnet.

6.28 Problem. (Minimum spanning trees (MST)) Gegeben: ein zusammenhängender ungerichteter Graph $G = (V, E)$, dessen Kanten e mit rellen Zahlen $w(e)$ gewichtet sind.
Gesucht: ein Teilgraph von G, der ein Baum ist, der G aufspannt und unter allen solchen Bäumen minimales Gewicht hat.

Ist der Ausgangsgraph nicht zusammenhängend, dann existiert nur ein minimaler aufspannender Wald (MSF), der aus den MST für die einzelne Zusammenhangskomponenten besteht.

6.29 Man kann ohne Beschränkung der Allgemeinheit davon ausgehen, dass die Gewichte aller Kanten eines Graphen paarweise verschieden sind. Das erreicht man nötigenfalls, indem man alle Kanten $e_1, \ldots, e_k$ mit gleichem $w(e_1) = \cdots = w(e_k)$ willkürlich total ordnet und festlegt, dass eine in der Ordnung frühere Kante auch „leichter“ sei.

Die nachfolgend betrachteten Algorithmen haben übrigens die Eigenschaft, dass die tatsächlichen Gewichte irrelevant sind. Es werden immer nur Gewichte zweier Kanten miteinander verglichen, um herauszufinden, welche leichter ist.

6.2.1 Ein deterministischer Algorithmus für MST

Wir bezeichnen im folgenden eine Kante als *lokal minimal*, wenn es einen Knoten gibt, mit dem sie inzidiert und unter allen diesen das kleinste Gewicht hat.

6.30 Lemma. Jede lokal minimale Kante gehört zu einem MST von G.

6.31 Beweis. Es sei e die lokal minimale Kante eines Knotens x; der andere Endpunkt von e sei y. Es sei T' ein aufspannender Baum von G, der *nicht* e enthalte. Wir zeigen: T' hat nicht minimales Gewicht.

Dazu sei e' die Kante von T', die von x wegführt und in T' auf dem kürzesten Weg von x nach y liegt. Man betrachte nun den Graphen T, der aus T' entsteht, indem man e' entfernt und e hinzunimmt. Wir behaupten:

1. $w(T) < w(T')$.
2. T spannt G auf.
3. T ist ein Baum.

Der Reihe nach:

1. Das ist klar, denn $w(e) < w(e')$ und $w(T) = w(T') - w(e') + w(e)$.
2. Es sei z der andere Endpunkt von e' und es seien v_1, v_2 zwei Punkte von G. Wenn der kürzeste Weg in T' zwischen ihnen e' *nicht* beinhaltet, dann sind die beiden in $T' - e'$ immer noch miteinander verbunden. Falls der Pfad e' beinhaltete, dann kann man in T diese Kante ersetzen durch den Pfad von x nach y und von dort zu z. Die beiden Punkte v_1 und v_2 sind also immer noch miteinander verbunden.

3. Angenommen, T enthielte einen Zyklus. Da T' Baum war, muss der Zyklus die Kante e enthalten. Also gibt es in $T' - e'$ einen nicht verkürzbaren Pfad von x nach y. Dieser Pfad ist offensichtlich verschieden vom kürzesten Pfad in T' von x nach y, denn dieser beinhaltete Kante e'. Also gäbe es in T' zwei verschiedene (einer mit e', einer ohne e'), nicht verkürzbare Pfade von x nach y. Also gäbe es in T' auch schon einen Zyklus im Widerspruch dazu, dass T' Baum ist.

■

6.32 Korollar. Die lokal minimalen Kanten eines Graphen bilden ein Wald.

6.33 (Borůvka-Phase) Eine Borůvka-Phase ist ein Algorithmus, der folgendes leistet:

1. Er bestimmt die Menge $L(G)$ aller lokal minimalen Kanten eines Graphen G.
2. Er berechnet den kontrahierten Graphen $G/L(G)$. Falls Mehrfachkanten entstünden, wird nur die mit kleinstem Gewicht behalten und die anderen entfernt.

Der so aus G entstehende Graph werde mit $B(G)$ bezeichnet.

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

6.35 Beweis. Übung. ■

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

6.37 Beweis. Eine Kante kann für höchstens zwei Knoten lokal minimal sein. Also werden in einer Borůvka-Phase mindestens $n/2$ Kanten entfernt. Mit jeder entfernten Kante wird auch ein Knoten entfernt. ■

6.38 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.

6.39 Beweis. Jeder aufspannende Baum T von G, der alle Kanten aus $L(G)$ enthält, induziert einen aufspannenden Baum T' von $B(G)$ mit $w(T') = w(T) - w(L(G))$. Umgekehrt gilt auch, dass jeder aufspannde Baum T' von $B(G)$ einen aufspannenden Baum $\bar{T}'$ von G induziert , der alle Kanten aus $L(G)$ enthält und Gewicht $w(T) = w(T') + w(L(G))$ hat (überlegen Sie sich das).

Sei nun T ein *minimaler* aufspannender Baum von G (er enthält also alle Kanten aus $L(G)$; siehe Lemma 6.30) und $B(T)$ der korrespondierende 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. ■

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

6.41 Algorithmus. (Borůvkas MST-Algorithmus) Man führt solange Borůvka-Phasen durch, bis der resultierende Graph höchstens noch zwei Knoten hat und führt Buch, welche Kanten jeweils kontrahiert werden. Wegen Lemma 6.38 ist dann klar, welche Kanten den MST bilden.

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

6.43 Beweis. Nach Lemma 6.36 benötigt man nur $O(\log n)$ viele Phasen, von denen jede nach Lemma 6.34 nur $O(m+n)$ Zeit benötigt. In zusammenhängenden Graphen ist $n \in O(m)$. ■

6.2.2 F-leichte und F-schwere Kanten

In diesem und im folgenden Abschnitt wird mit F ein Wald in einem Graphen G bezeichnet, d. h. also ein Teilgraph, der aus einem oder mehreren disjunkten Bäumen besteht.

6.44 Definition Es sei F ein Wald in G, 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 (falls er existiert; sonst sei $P = \emptyset$). Wir setzen

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

Eine 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 $w(e) \leqslant W_F(v_1, v_2)$. ◇

6.45 Lemma. Es sei F ein beliebiger(!) Wald von G und $e = \{v_1, v_2\}$ eine Kante in G. Wenn e F-schwer ist, dann gehört e nicht zum MST von G.

Äquivalent ist: Wenn eine Kante zum MST gehört, dann ist sie F-leicht.

6.46 Beweis. Die 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 der Bestimmung des MST etwa 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; siehe Korollar 6.40) MST hinzugenommen. ■

6.47 Lemma. Ein aufspannender Baum T eines Graphen G ist minimal, wenn die einzigen T-leichten Kanten in G die Kanten von T sind.

6.48 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. Nach Lemma 6.45 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. ■

Ohne einen der (nicht ganz einfachen) Algorithmen (siehe Dixon, Rauch und Tarjan 1992; King 1997) anzugeben halten wir noch fest:

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

6.2.3 Ein randomisierter MSF-Algorithmus

Es sei RandomSample(G, p) 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.

6.50 Lemma. Zu gegebenen G und p kann ein MSF F' für ein $G' =$ RandomSample(G, p) konstruiert werden, indem man einmal jede Kante von G betrachtet, sofern die Kanten nach aufsteigendem Gewicht sortiert vorliegen.

6.51 Beweis. Die Kanten $e_1, \ldots, e_m$ von G seien nach aufsteigendem Gewicht geordnet. Die Konstruktion von $G' = \text{RandomSample}(G, p)$ erfolge, indem nacheinander für jedes i die Kante e_i mit Wahrscheinlichkeit p zu G' hinzu genommen werde.

Die Konstruktion des MSF F' kann in diesem Fall wie folgt erledigt werden:

$F'_0 \leftarrow \emptyset$
$k \leftarrow 0$
for $i \leftarrow 1$ **to** m **do**
  if RandomFloat$(0,1) \leqslant p$ **then**
    ⟨*nimm* $e_i = \{u, v\}$ *zu* G' *hinzu*⟩
    if u und v gehören zu verschiedenen Zusammenhangskomponenten von F'_{k-1} **then**
      $k \leftarrow k + 1$
      $F'_k \leftarrow F'_{k-1} + e_i$
    fi
  fi
od
$F' \leftarrow F'_k$ ⟨*ist der MSF*⟩

Offensichtlich wird durch diesen Algorithmus ein Wald konstruiert. Die Bedingung, dass die Enden u und v von e_i zu verschiedenen Zusammenhangskomponenten von F'_k gehören, ist äquivalent zu der Bedingung, dass e_i im betreffenden Schleifendurchlauf gerade F'_k leicht ist.

Nach Lemma 6.47 ist das Endergebnis F' genau dann MSF, falls er jede F'-leichte Kante von G' enthält. Wegen der aufsteigenden Gewichte der Kanten ist ein e von G' genau dann F'-leicht, wenn e zu dem Zeitpunkt, zu dem es auf Zugehörigkeit zu F'_k untersucht wird, F'_k-leicht ist. Und zu diesem Zeitpunkt ist e genau dann F'_k-leicht, wenn es verschiedenen Zusammenhangskomponenten von F'_k verbindet. ∎

6.52 Lemma. Es sei F' ein minimaler aufspannender Wald von $G' = \text{RandomSample}(G, p)$. Der Erwartungswert für die Anzahl der F'-leichten Kanten von G (!) ist kleiner oder gleich n/p.

6.53 Beweis. Da der MST eindeutig ist, ist es für die Aussage gleichgültig, mit welchem Algorithmus er berechnet wird.

Betrachten wir den Algorithmus aus Beweis 6.51. Als *Phase* i ($i \geqslant 1$) bezeichnen wir die Reihe Schleifendurchläufe, zu deren Beginn k immer den Wert $i-1$ hat. Salopp gesprochen beginnt Phase i, nachdem F'_{i-1} gebildet wurde und endet mit der Bildung von F'_i.

Während jeder Phase werden sowohl einige F'_{i-1}-schwere Kanten betrachtet, die für den Beweis offensichtlich irrelevant sind, und einige F'_{i-1}-leichte Kanten.

Jede Kante von G, die F'_{i-1}-leicht ist, hat Wahrscheinlichkeit p, dass sie zu G' und folglich zu F'_{i-1} hinzu genommen wird. Und mit der ersten Wahl einer solchen Kante endet die Phase auch. Folglich ist die Anzahl der in Phase i betrachteten F'_{i-1}-leichten Kanten geometrisch verteilt mit Parameter p. Der Erwartungswert für die Anzahl ist $1/p$.

Am Ende ist $|F'| = k \leqslant n-1$. Folglich ist die Zufallsvariable X der bis zum Ende von Phase k betrachteten leichten Kanten die Summe von k identisch geometrisch verteilten unabhängigen Zufallsvariablen. Um die danach noch betrachteten, aber nicht hinzugenommenen Kanten mit zu berücksichtigen, stelle man sich vor, dass die **for**-Schleife noch länger durchlaufen werde, bis insgesamt n mal die Bedingung RandomFloat$(0,1) < p$ erfüllt war. Der Erwartungswert für die Anzahl Schritte, bis das jeweils einmal der Fall war, ist ebenfalls $1/p$. Es bezeichne Y die Zufallsvariable für die dafür insgesamt notwendige Anzahl von Schleifendurchläufen.

Dann gilt für alle z: $\mathbf{P}(X > z) \leqslant \mathbf{P}(Y > z)$ (man sagt, dass X von Y stochastisch dominiert werde). Es gilt dann auch $\mathbf{E}[X] \leqslant \mathbf{E}[Y]$. Dieser Erwartungswert ist aber $\mathbf{E}[Y] = n/p$. ∎

Der Algorithmus aus Beweis 6.51 dient nur dazu, den Beweis von Lemma 6.52 zu erleichtern; dessen Aussage ist aber unabhängig davon, wie man den MSF konstruiert hat. Der Algorithmus ist im Weiteren *nicht* von Bedeutung Es ist also z. B. nicht notwendig, dass die Kanten nach Gewicht sortiert vorliegen.

6.54 Algorithmus.

proc $F \leftarrow \mathrm{MSF}(G)$:
⟨*Eingabe: gewichteter ungerichteter Graph* G *mit* n *Knoten und* m *Kanten*⟩
⟨*Ausgabe: der minimale aufspannende Wald* F *von* G⟩

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

return F

6.55 Satz. *Algorithmus 6.54 berechnet den MSF des Eingabegraphen* G.

6.56 Beweis.

zu 1. Nach Lemma 6.38 ist klar, dass die Kanten aus C_1 zum MSF von G gehören. Enthält G_1 keine weiteren Kanten, dann bilden also die Kanten aus C_1 gerade den MSF.

zu 3. F_2 ist ein Wald in G_2 und folglich auch in G_1 und G.

zu 4. Nach Lemma 6.45 gehören die F_2-schweren Kanten von G_1 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. Wieder nach Lemma 6.38 ist daher $C_1 \cup F_3$ ein MSF und also (wegen Zusammenhang) ein MST von G.

∎

6.57 Satz. *Der Erwartungswert für die Laufzeit von Algorithmus 6.54 ist* $O(m + n)$.

6.58 Beweis. Es bezeichne $T(n, m)$ den Erwartungswert der Laufzeit von Algorithmus 6.54 für Eingabegraphen mit n Knoten und m Kanten. Für die einzelnen Schritte gilt:

1. Laufzeit ist $O(m + n)$. Und G_1 hat höchstens $n/8$ Knoten und m Kanten.

2. Laufzeit ist $O(m+n)$. Und G_2 hat höchstens $n/8$ Knoten und der Erwartungswert für die Anzahl Kanten ist $m/2$.
3. Erwartungswert für die Laufzeit ist $T(n/8, m/2)$.
4. Laufzeit ist $O(m+n)$. Und G_3 hat höchstens $n/8$ Knoten und Erwartungswert für die Anzahl Kanten ist $(n/8)/(1/2) = n/4$ (nach Lemma 6.52).
5. Erwartungswert für die Laufzeit ist $T(n/8, n/4)$.
6. Laufzeit ist $O(n)$.

Insgesamt ergibt sich

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

Als Lösung dieser Rekurrenzungleichung ergibt sich $T(n,m) \in O(n+m)$. ∎

Literatur

Dixon, B., M. Rauch und R. E. Tarjan (1992). „Verification and Sensitivity Analysis of Minimum Spanning Trees in Linear Time“. In: *SIAM Journal on Computing* 21.6, S. 1184–1192 (siehe S. 55).

King, Valerie (1997). „A Simpler Minimum Spanning Tree Verification Algorithm“. In: *Algorithmica* 18.2, S. 263–270 (siehe S. 55).

