Datenstrukturen und effiziente Algorithmen

Markus Vieth David Klopp

5. Februar 2016

Vorwort

Dieses Skript basiert auf unserer Mitschrift der Vorlesung Datenstrukturen und effiziente Algorithmen (DSeA) im WS 2015/16 an der JGU Mainz (Dozent: Prof. Dr. E. Schömer).
Es handelt sich nicht um eine offizielle Veröffentlichung der Universität.
Wir übernehmen keine Gewähr für die Fehlerfreiheit und Vollständigkeit des Skripts.
Fehler können unter Github gemeldet werden.

Inhaltsverzeichnis

1 Vorlesung

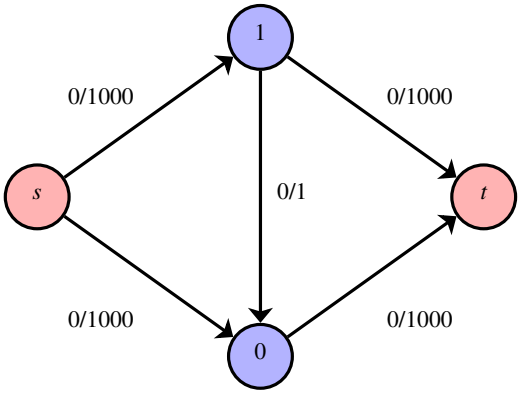


Abbildung 1.1: Wiederholung

1.1 Edmonds-Karp Algorithmus

$$G = (V, E) \quad , c : R \to \mathbb{R}_0^+, G_F = (V, E_f), G_f^L = (V, E_f^L)$$

```
1 f = 0;
2 while(∃ p ⇝ t ∈ G_f^L = (V, E_f^L)) {
3     sei c_min(p) die kleinste Restkapazität auf p
4     f(u,v) = { f(u,v) + c_min(p)   falls (u,v) ∈ p
               { f(u,v) − c_min(p)   falls (v,u) ∈ p
5 }
```

$\delta_f(s, v)$ die kleinste Zahl von Kanten, die in G_f^L benötigt werden, um von s nach v zu gelangen.

1.1.1 Lemma:

Im Verlauf des Edmonds-Karp Algorithmus gilt:

$$\delta_{f'}(s, v) \geq \delta_f(s, v)$$

wobei der Fluss f' durch eine Flussverbesserung aus f hervorgegangen ist.

1.1.2 Beweis durch Widerspruch

Annamhe

$$\exists v \in V : \delta_{f'}(s, v) < \delta_f(s, v) \qquad (**)$$

sei v so gewählt, dass $\delta_{f'}(s, v)$ minimal.
Sei $s \rightsquigarrow u \to v$ ein kürzester Weg in $G_{f'}^L$.

$$\delta_f(s, u) \leq \delta_{f'}(s, u) = \delta_{f'}(s, v) - 1 \qquad (*)$$

Behauptung

$$(u, v) \notin E_f^L$$

Beweis durch Widerspruch

Annahme $(u, v) \in E_f^L$

$$\delta_f(s, v) \leq {}^{\text{I}}\delta_f(s, u) + 1 \leq {}^{\text{II}}\delta_{f'}(s, v) \quad \lightning \text{ zu } (**)$$

$$\Rightarrow (u, v) \notin E_f^L \text{ aber } (u, v) \in E_{f'}^L$$

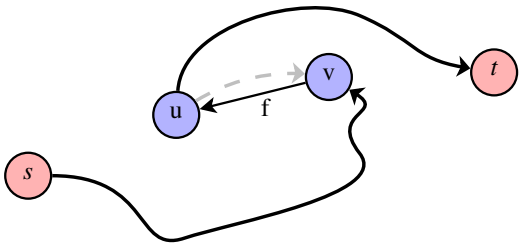


Abbildung 1.2

d.h. Bei der Flussverbesserung von f zu f' wurde die Kante (v, u) benutzt in G_f^L.

$$\begin{aligned} \delta_f(s, u) &= \delta_f(s, v) + 1 \\ &\overset{(**)}{>} \delta_{f'}(s, v) + 1 \\ &\overset{(*)}{\geq} \delta_f(s, u) + 2 \lightning \end{aligned}$$

q.e.d.

1.1.3 Lemma

Eine kante (u, v) kann un den Level-Rest-Netzwerken höchstens $\frac{|V|}{2}$ mal saturiert werden und damit temporär aus dem jeweiligen Rest-Netzwerk verschwinden

1.1.4 Beweis

$$\delta_f(s, v) = \delta_f(s, u) + 1$$

$$\delta_{f'}(s, v) = \delta_{f'}(s, u) - 1$$

$$\delta_f(s, u) = \delta_f(s, v) - 1 \leq \delta_{f'}(s, v) - 1 = \delta_{f'}(s, u) - 2$$

$$\Rightarrow \delta_{f'}(s, u) \geq \delta_f(s, u) + 2$$

q.e.d.

[I] Dreiecksungleichung
[II] wegen $(*)$

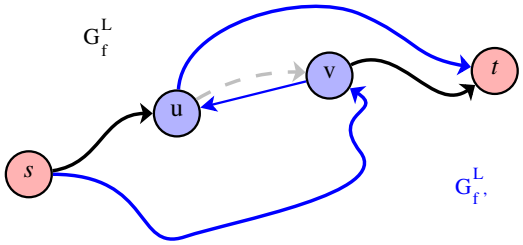


Abbildung 1.3

1.1.5 Laufzeitanalyse von Edmonds-Karp Algorithmus

Bei jeder Flussverbesserung wird mindestens eine Kante saturiert. Jede einzelne Kante kann aber höchstens $\frac{|V|}{2}$ mal saturiert werden.
$\Rightarrow$ Es gibt höchstens $\mathcal{O}(|E| \cdot |V|)$ viele Flussverbesserungen, Jede Flussverbesserung kann in $\mathcal{O}(|E|)$ ausgeführt werden.
$\Rightarrow$ Gesamtlaufzeit: $\mathcal{O}(|E|^2 \cdot |V|)$

1.2 Algorithmus von Dinic

1.2.1 Sperrfluss (blocking flow)

$$G_f^L = (V, E_f^L)$$

Wir konstruieren einen Sperrfluss g für einen Graphen H, indem wir wiederholt entlang von (s,t)-Pfaden Fluss von s nach t transportieren.
Bevor wir diesen Prozess wiederholen, löschen wir saturierte Kanten aus H. Läuft man bei der Wegesuche in eine Sackgasse, so muss diese aus H entfernt werden, damit man zu einem späteren Zeitpunkt nicht wieder in diese Sackgasse gerät.

Ziel Algorithmus zur Sperrflussberechnung in Zeit $\mathcal{O}(|V| \cdot |E|)$

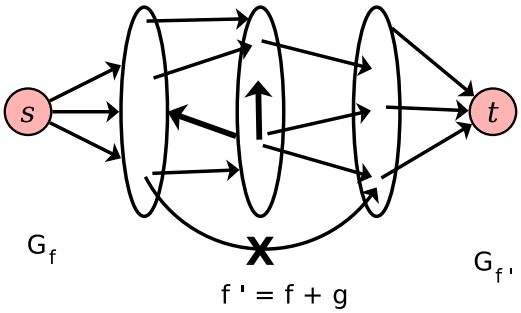
2 Vorlesung

2.1 Definition: Sperrfluss

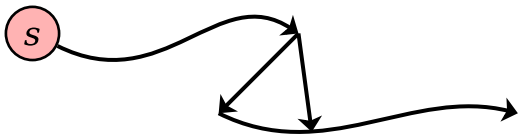
$$G = (V, E) \quad g : E \to \mathbb{R}^+ \text{ ein Fluss}$$

Auf jedem $s-t$-Pfad gibt es eine saturierte Kante $(u, v) \in E$, d.h. $g(u, v) = c(u, v)$

Achtung Wir betrachten hierbei nicht die Kantenmenge E_f des Restnetzwerkes.
Es gilt: $\delta_{f'}(s, v) \geq \delta_f(s, v)$, wobei f' aus f durch eine Flusserhöhung hervorgegangen ist.
Für den Dinic-Algorihtmus gilt darüber hinaus, dass $\delta_{f'}(s, t) \geq \delta_f(s, t) + 1$ wobei f' aus f durch eine Flussverbesserung mittels eines Sperrflusses g hervorgegangen ist.



(a) g Sperrfluss



(b)

Beweisidee

Konsequenz Im Dinic-Algorithmus genügt es, $|V|$ Sperrfluss-Berechnungen durchzuführen.

Ziel Sperrfluss in Zeit $\mathcal{O}(|V| \cdot |E|)$ berechnen.

Gesamtlaufzeit Dinic	$\mathcal{O}(\|V\|^2 \cdot \|E\|)$
vs. Edmonds-Karp	$\mathcal{O}(\|V\| \cdot \|E\|^2)$
(Sleator & Tarjan :	$\mathcal{O}(\|V\| \cdot \|E\| \cdot \log \|V\|))$

2.1.1 Pseudo-Code

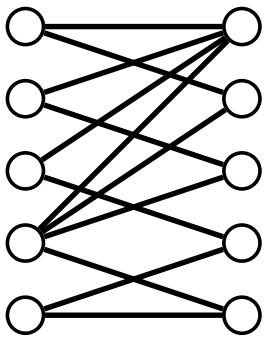
```
1  H = G_f^L;
2  stack P;
3  P.push(s);
4  g = 0;
5  while(true) {
6      u = P.top();
7      if (∃ (u, v) ∈ H) {
8          P.push(v);
9          if (v ≠ t) continue;
```

```
10          //v = 7, d.h. flussverbessernder s-t-Pfad gefunden
11          c_min = min_{(u,v)∈P} {c_f(u, v)-g(u, v)}
12          forall (u, v) ∈ P {
13              g(u, v) = g(u, v)+c_min;
14              if (g(u, v) == c_f(u, v))
15                  lösche (u, v) ∈ H;
16          }
17          P.clear();
18          P.push(s);
19          continue;
20      } //end if
21      //Sackgasse
22      lösche alle zu u inzidenten Kanten aus H;
23      P.pop();
24      if (u == s)
25          break;
26  } //end while
```

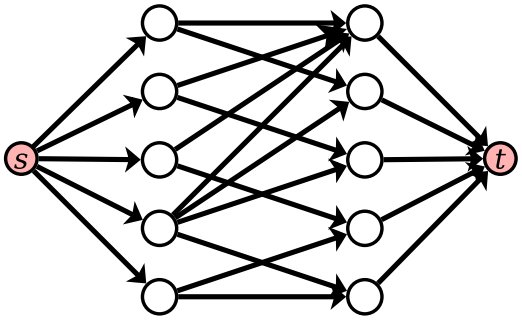
2.1.2 Begründung zur Laufzeit

Jeder $s-t$-Pfad wird in Zeit $\mathcal{O}(|V|)$ gefunden. Danach wird mindestens eine Kante aus H entfernt, weil sie saturiert wird. Jede kante kann höchstens einmal als Sackgasse betreten werden, weil sie anschließend gelöscht wird.

2.2 Maxmimum Matching als Flussproblem



(a) Matchingproblem



(b) Flussproblem (Kantengewichte sind 1)

$$E \supseteq M^* = \{(u,v) \in E \cap V_1 \times V_2 \mid f(u,v) = 1\}$$

$$\begin{aligned} \text{F.F.-Laufzeit} \quad & \mathcal{O}(|f^*| \cdot |E|) \\ & = \mathcal{O}(|V| \cdot |E|) \end{aligned}$$

$|f|$ ist ganzzahlig = Kardinalität des maximum Matchings $|M^*|$

2.2.1 Flussnetzwerke mit Einheitskapazität (unit capacity network flow)

$$G = (V, E) \quad c : E \to \{0, 1\}$$

Sperrfluss-Berechnung läuft in $\mathcal{O}(|E|)$, weil immer alle Kanten auf den flussverbessernden $s-t$-Pfaden gelöscht werden können.

Dinic: $\mathcal{O}(|V| \cdot |E|)$

Satz

Mit Hilfe des Dinic-Algorithmus lässt sich ein Flussproblem mit Einheitskapazität in Zeit

$$\mathcal{O}\left(\min\left(|E|^{\frac{1}{2}}, |V|^{\frac{2}{3}}\right)\right)$$

lösen.

Beweis

1. Fall Zeige, dass $2 \cdot \sqrt{|E|}$ viele Sperrfluss-Berechnungen genügen.

1. Phase Zuerst $\sqrt{|E|}$ Sperrfluss-Berechnungen. $\Rightarrow$ Fluss f

$$\Rightarrow \delta_f(s,t) \geq \sqrt{|E|}$$

$\Rightarrow$ Schnitt zwischen zwei Levels L_i und L_{i+1} mit weniger als $\sqrt{|E|}$ Kanten.

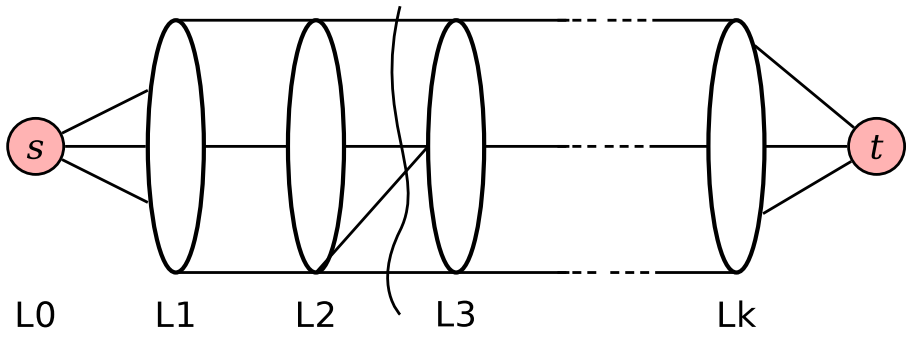


Abbildung 2.3: $k \geq \sqrt{|E|}$

$|f^*| \leq |f| + \sqrt{|E|}$, weil über diesen Schnitt zusätzlich zu f noch ein Fluss der Größe $\sqrt{|E|}$ möglich ist.

$\Rightarrow$ **2. Phase:** Um von f zu f^* zu kommen, reichen weitere $\sqrt{|E|}$ viele Sperrfluss-Berechnungen aus.

3 Vorlesung

3.0.1 Laufzeit für Dinic-Algorithmus bei Flussnetzwerken mit Einheitskapazität

$$\mathcal{O}\left(\min\left\{|E|^{\frac{1}{2}}, |V|^{\frac{2}{3}}\right\} \cdot |E|\right)$$

2. Fall

1. Phase Führe $2 \cdot |V|^{\frac{2}{3}}$ Sperrflussberechnungen durch. $\Rightarrow$ $\delta_f(s,t) > 2 \cdot |V|^{\frac{2}{3}}$

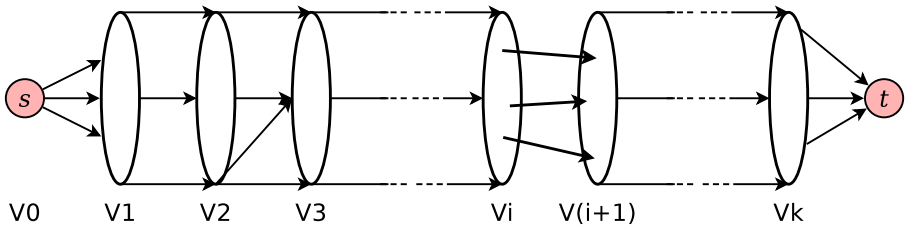


Abbildung 3.1: Schaubild

Behauptung

$$k = 2 \cdot |V|^{\frac{2}{3}}$$

$$\exists 0 < i < k: \quad |V_i| \leq |V|^{\frac{1}{3}}$$
$$\text{und } |V_{i+1}| \leq |V|^{\frac{1}{3}}$$

Falls Behauptung gilt $\Rightarrow$ $c_f(V_i, V_{i+1}) \leq$ #Kanten über diesen Schnitt $\leq |V_i| \cdot |V_{i+1}| \leq |V|^{\frac{2}{3}}$. Maximaler Fluss $|f^*|$ ist vom aktuellen Fluss f, der sich nach der 1. Phase eingestellt hat höchstens noch $|V|^{\frac{2}{3}}$ entfernt. Deshalb genügen in der 2. Phase noch $|V|^{\frac{2}{3}}$ Sperrflussberechnungen um $|f^*|$ zu erreichen.

Beweis der Behauptung $< |V|^{\frac{2}{3}}$ Schichten haben $> |V|^{\frac{1}{3}}$ Knoten. Andernfalls gäbe es mehr als $|V|^{\frac{2}{3}} \cdot |V|^{\frac{1}{3}} = |V|$ viele Knoten.

Beweisidee Wir färben die Schichten weiß, welche weniger als $|V|^{\frac{1}{3}}$ Knoten haben und den Rest schwarz, da es mehr weiße als schwarze gibt, müssen 2 weiße aufeinander folgen.:

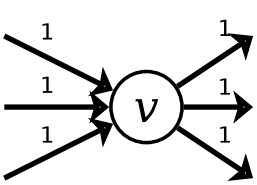
$$\underbrace{w\ s\ w\ s\ w\ s\ \ldots\ w\ s}_{k}$$

$> |V|^{\frac{2}{3}}$ Schichten haben $\leq |V|^{\frac{1}{3}}$ Knoten
$\Rightarrow$ es muss i geben, mit $|V_i| \cdot |V_{i+1}| \leq |V|^{\frac{1}{3}}$

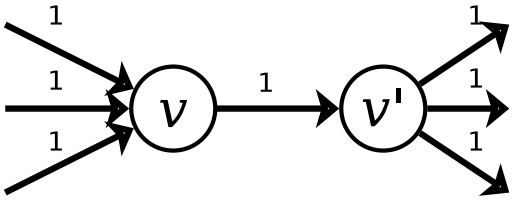
q.e.d.

3.0.2 Finden Knotendisjunkter Wege

Idee Man Forme den Graphen wie folgt um:



(a) Originall Knoten



(b) Umgeformter Knoten

3.0.3 Ergänzung zum Paper

$$\mathrm{pot}_f(v) = \min\left(\sum_{(v,u)\in E} c(v,u) - f(v,u), \sum_{(u,v)\in E} c(u,v) - f(u,v)\right)$$

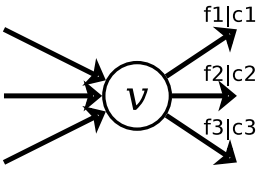


Abbildung 3.3

Abbildungsverzeichnis