

# Datenstrukturen und effiziente Algorithmen

Markus Vieth      David Klopp

2. 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

<b>Vorwort</b>	<b>iii</b>
<b>1 Vorlesung</b>	<b>1</b>
1.1 Edmonds-Karp Algorithmus . . . . .	1
1.1.1 Lemma: . . . . .	1
1.1.2 Beweis durch Widerspruch . . . . .	1
1.1.3 Lemma . . . . .	2
1.1.4 Beweis . . . . .	2
1.1.5 Laufzeitanalyse von Edmonds-Karp Algorithmus . . . . .	3
1.2 Algorithmus von Dinic . . . . .	3
1.2.1 Sperrfluss (blocking flow) . . . . .	3
<b>2 Vorlesung</b>	<b>4</b>
2.1 Definition: Sperrfluss . . . . .	4
2.1.1 Pseudo-Code . . . . .	4
2.1.2 Begründung zur Laufzeit . . . . .	5
2.2 Maximum Matching als Flussproblem . . . . .	5
2.2.1 Flussnetzwerke mit Einheitskapazität . . . . .	5



# 1 Vorlesung

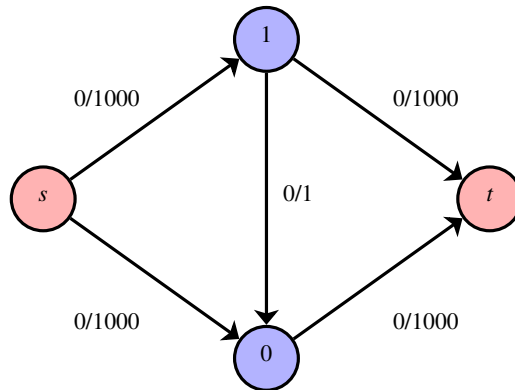


Abbildung 1.1: Wiederholung

## 1.1 Edmonds-Karp Algorithmus

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

```

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**

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

sei  $v$  so gewählt, dass  $\delta_{f'}(s, v)$  minimal.

Sei  $s \rightsquigarrow u \rightarrow v$  ein kürzester Weg in  $G_{f'}^L$ .

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

## Behauptung

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

**Beweis** durch Widerspruch

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

$$\delta_f(s, v) \leq \overset{\text{I}}{\delta_f(s, u) + 1} \leq \overset{\text{II}}{\delta_{f'}(s, v)} \not\leq \text{zu } (**)$$

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

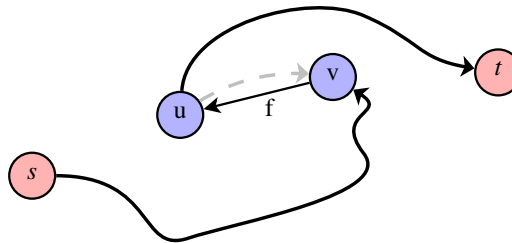


Abbildung 1.2

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

$$\begin{aligned} \delta_f(s, u) &= \delta_f(s, v) + 1 \\ &\overset{(**)}{>} \delta_{f'}(s, v) + 1 \\ &\overset{(*)}{\geq} \delta_f(s, u) + 2 \not\leq \end{aligned}$$

q.e.d.

### 1.1.3 Lemma

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

### 1.1.4 Beweis

$$\begin{aligned} \delta_f(s, v) &= \delta_f(s, u) + 1 \\ \delta_{f'}(s, v) &= \delta_{f'}(s, u) - 1 \\ \delta_f(s, u) &= \delta_f(s, v) - 1 \leq \delta_{f'}(s, v) - 1 = \delta_{f'}(s, u) - 2 \\ &\Rightarrow \delta_{f'}(s, u) \geq \delta_f(s, u) + 2 \end{aligned}$$

q.e.d.

<sup>I</sup>Dreiecksungleichung

<sup>II</sup>wegen (\*)



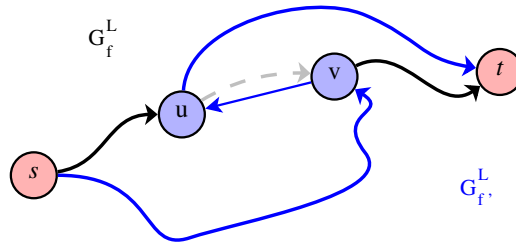


Abbildung 1.3

### 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.

⇒ Es gibt höchstens  $\mathcal{O}(|E| \cdot |V|)$  viele Flussverbesserungen, Jede Flussverbesserung kann in  $\mathcal{O}(|E|)$  ausgeführt werden.

⇒ 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

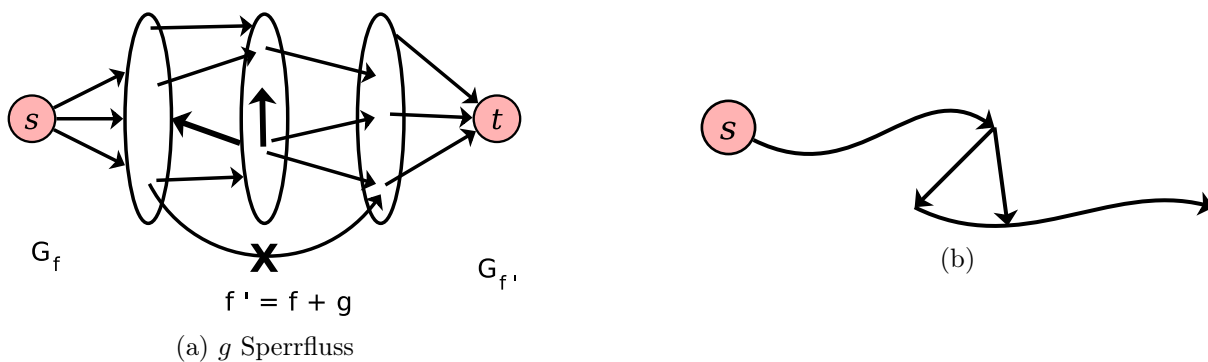
$G = (V, E)$   $g : E \rightarrow \mathbb{R}^+$  ein Fluss

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-Algorithmus 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.



#### 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 ( $\exists (u, v) \in H$ ) {
8          P.push(v);
9          if ( $v \neq t$ ) continue;

```

```

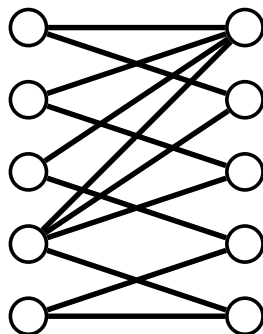
10      //v = 7, d.h. flussverbessernder s-t-Pfad gefunden
11       $c_{min} = \min_{(u,v) \in P} \{c_f(u, v) - g(u, v)\}$ 
12      forall  $(u, v) \in P$  {
13           $g(u, v) = g(u, v) + c_{min}$ ;
14          if  $(g(u, v) == c_f(u, v))$ 
15              lösche  $(u, v) \in H$ ;
16      }
17      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

```

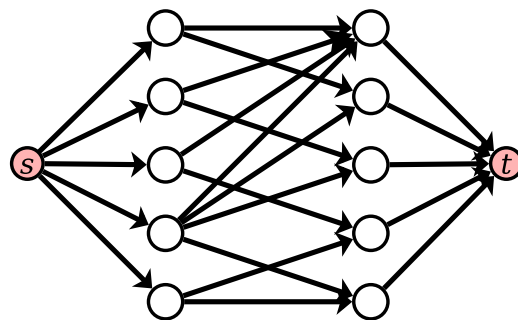
### 2.1.2 Begründung zur Laufzeit

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 Maximum 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)

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

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

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

## 2 Vorlesung

### 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.

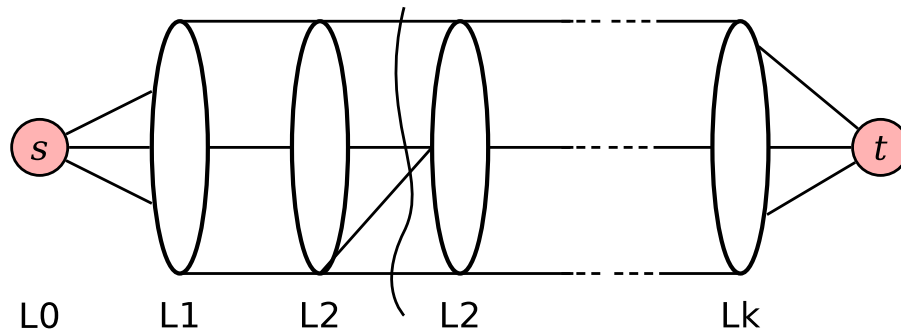


Abbildung 2.3:  $k \geq \sqrt{|E|}$

$|f^*| \leq |f| + \sqrt{|E|}$ , 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.

# Abbildungsverzeichnis

1.1	Wiederholung . . . . .	1
1.2	. . . . .	2
1.3	. . . . .	3