Universität Karlsruhe (TH)
Forschungsuniversität • gegründet 1825

Institut für Theoretische Informatik
Lehrstuhl Prof. Dr. D. Wagner

Lösungshinweise 3

Vorlesung Algorithmentechnik im WS 08/09

Problem 1: Kreuzende Schnitte

Zwei Schnitte $(S, V \setminus S)$ und $(T, V \setminus T)$ in einem Graph $G = (V, E)$ *kreuzen sich*, wenn keine der Mengen $A := S \cap T$, $B := S \setminus T$, $C := T \setminus S$ und $D := V \setminus (S \cup T)$ leer ist. Sei $c : E \to \mathbb{R}_0^+$ eine Kantengewichtsfunktion auf G. Zeigen Sie:

(a) Sind $(S, V \setminus S)$ und $(T, V \setminus T)$ zwei sich kreuzende Schnitte minimalen Gewichts λ in G, so gilt $c(A, D) = c(B, C) = 0$ und $c(A, B) = c(B, D) = c(D, C) = c(C, A) = \lambda/2$.

Proof. Zur Veranschaulichung der sich kreuzenden minimalen Schnitte $(S, V \setminus S)$ und $(T, V \setminus T)$ betrachten wir die folgende Abbildung:

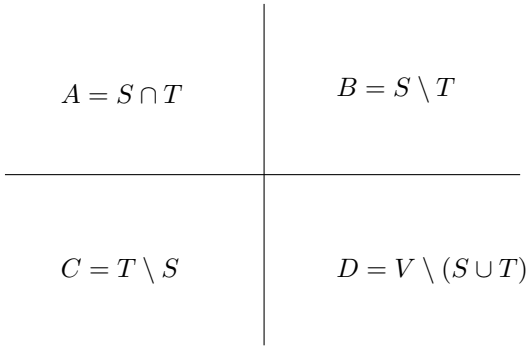


Abbildung 1: Aufteilung von V in disjunkten Mengen $A,\ B,\ C$ und D.

Die folgenden Gleichungen kann man aus der Abbildung (1) entnehmen:

$$\lambda = c(S, V \setminus S) = c(A, C) + c(B, D) + c(A, D) + c(B, C) \tag{1}$$
$$\lambda = c(T, V \setminus T) = c(A, B) + c(C, D) + c(A, D) + c(B, C) \tag{2}$$

- Behauptung 1: $c(A, D) = c(B, C) = 0$.

  Beweis:
  Aus den Gleichungen (1) und (2) folgt:

$$c(A, C) + c(A, B) + c(B, D) + c(C, D) + 2c(A, D) + 2c(B, C) = 2\lambda. \tag{3}$$

  Annahme $c(A, D) > 0$:
  Aus Gleichung (3) und $2c(A, D) > 0$ folgt:

$$c(A, C) + c(A, B) + c(B, D) + c(C, D) + 2c(B, C) < 2\lambda$$
$$\Rightarrow\ c(A, C) + c(C, D) + c(B, C) < \lambda \quad \text{oder} \quad c(A, B) + c(B, D) + c(B, C) < \lambda.$$

Für die Gewichte der Schnitte $(C, V \setminus C)$ und $(B, V \setminus B)$ würde dann gelten:

$$c(C, V \setminus C) = c(A, C) + c(C, D) + c(B, C) < \lambda$$

oder

$$c(B, V \setminus B) = c(A, B) + c(B, D) + c(B, C) < \lambda.$$

Dieses Ergebnis ist aber ein Widerspruch, da sonst die Schnitte $(S, V \setminus S)$ und $(T, V \setminus T)$ nicht minimal wären. Somit ist die Annahme $c(A, D) > 0$ falsch und es muss gelten $c(A, D) = 0$.

Eine analoge Berechnung ergibt $c(B, C) > 0$.

- Behauptung 2: $c(A, B) = c(B, D) = c(D, C) = c(C, A) = \lambda/2$.
  Beweis:
  Aus der bewiesenen Behauptung 1 und den Gleichungen aus 1 und 2 ergibt sich:

$$c(A, B) + c(C, D) + c(A, C) + c(B, D) = 2\lambda.$$

Nun betrachten wir die Schnitte $(A, V \setminus A)$, $(B, V \setminus B)$, $(C, V \setminus C)$ und $(D, V \setminus D)$.
Da ein minimaler Schnitt des Graphen das Gewicht λ hat, können wir die Gewichte dieser Schnitte wie folgt abschätzen:

$$c(A, V \setminus A) = c(A, B) + c(A, C) \geq \lambda \tag{4}$$
$$c(B, V \setminus B) = c(A, B) + c(B, D) \geq \lambda \tag{5}$$
$$c(C, V \setminus C) = c(C, D) + c(A, C) \geq \lambda \tag{6}$$
$$c(D, V \setminus D) = c(C, D) + c(B, D) \geq \lambda. \tag{7}$$

1. Fall: Annahme $c(A, B) < \lambda/2$.
Aus den Ungleichungen (4) und (5) folgt $c(A, C) > \lambda/2$ und $c(B, D) > \lambda/2$. Für den Schnitt $(S, V \setminus S)$ würde dann gelten:

$$c(S, V \setminus S) = c(A, C) + c(B, D) > \lambda$$

Das ist ein Widerspruch da $c(S, V \setminus S) = \lambda$.

2. Fall: Annahme $c(A, B) > \lambda/2$.
Aus $c(T, V \setminus T) = c(A, B) + c(C, D) = \lambda$ folgt dann $c(C, D) < \lambda/2$. Weiter ergeben sich $c(A, C) > \lambda/2$ und $c(B, D) > \lambda/2$ aus den Ungleichungen (6) und (7) und somit $c(S, V \setminus S) = c(A, C) + c(B, D) > \lambda$ im Widerspruch zu $c(S, V \setminus S) = \lambda$.
Also gilt: $c(A, B) = \lambda/2$.
Analog kann man für c(B,D), c(D,C) und c(C,A) beweisen:

$$c(B, D) = c(D, C) = c(C, A) = \lambda/2. \qquad \square$$

(b) Sind s und t zwei adjazente Knoten mit $c(\{s, t\}) > 0$, so enthält die Menge der minimalen Schnitte von G, die s und t trennen, keine zwei sich kreuzenden Schnitte.

Proof. Annahme $(S, V \setminus S)$ und $(T, V \setminus T)$ wären zwei sich kreuzende minimale Schnitte die s und t trennen. Ohne Einschränkung soll gelten $s \in S \cap T$.

Es gilt:

$$(s \in S \cap T) \Rightarrow (s \in S) \wedge (s \in T) \Rightarrow (t \in V \setminus T) \wedge (t \in V \setminus S) \Rightarrow (t \in V \setminus (S \cup T)).$$

Aus $c(\{s, t\}) > 0$ ergibt sich:

$$c(S \cap T, V \setminus (S \cup T)) > 0.$$

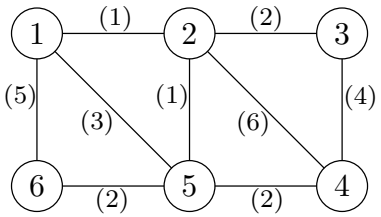
In der Teilaufgabe (a) haben wir aber bereits folgendes für zwei sich kreuzende minimale Schnitte bewiesen:

$$c(A, D) = c(S \cap T, V \setminus (S \cup T)) = 0.$$

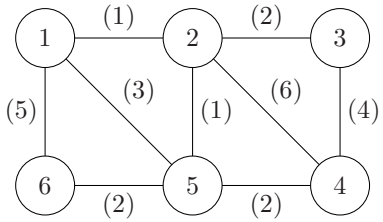
Wir erhalten also einen Widerspruch. Es existieren also keine zwei sich kreuzende minimale Schnitte, die s und t trennen. □

Problem 2: Stoer-Wagner

(a) Wenden Sie auf den unten abgebildeten Graphen (Kantengewichte in Klammern) den Algorithmus von Stoer & Wagner an. Geben Sie nach jeder Phase die Knoten s und t, den Schnitt der Phase und dessen Gewicht an und zeichnen Sie den nach dem Verschmelzen resultierenden Graphen (mit Kantengewichten). **Verwenden Sie in Phase i den Knoten als Startknoten, der Knoten i des Originalgraphen enthält.** Geben Sie zum Schluss den minimalen Schnitt S_{min} an.



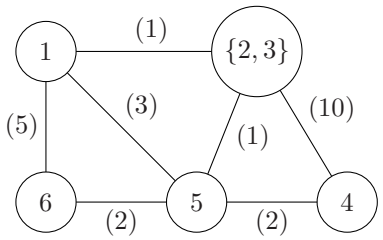
Es sei jeweils $G_i = (V_i, E_i)$ der Graph in Phase i. Anwendung des Algorithmus von Stoer & Wagner:



Phase 1: Anwenden von MinSchnittPhase mit Startknoten 1 auf den rechts abgebildeten Graphen liefert

$$\begin{aligned}
S_1 &= \{1\} \\
S_1 &= \{1, 6\} \\
S_1 &= \{1, 6, 5\} \\
S_1 &= \{1, 6, 5, 4\} \\
S_1 &= \{1, 6, 5, 4, 2\} \\
S_1 &= V_1
\end{aligned}$$

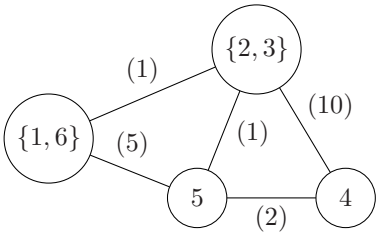
Somit ist $s = 2$ und $t = 3$, der Schnitt der Phase ist $(V_1 \setminus \{3\}, \{3\})$ mit Gewicht 6. Verschmelze Knoten 3 und 2.



Phase 2: Anwenden von MinSchnittPhase mit Startknoten 2, in unserem Fall also $\{2,3\}$ auf den rechts abgebildeten Graphen liefert

$$S_2 = \{\{2,3\}\}$$
$$S_2 = \{\{2,3\},4\}$$
$$S_2 = \{\{2,3\},4,5\}$$
$$S_2 = \{\{2,3\},4,5,1\}$$
$$S_2 = V_2$$

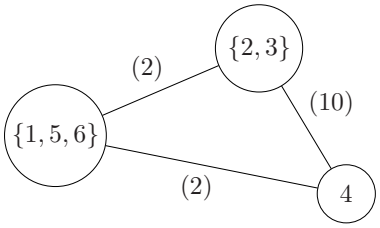
Somit ist $s = 1$ und $t = 6$, der Schnitt der Phase ist $(V_2 \setminus \{6\}, \{6\})$ mit Gewicht 7. Verschmelze Knoten 1 und 6.



Phase 3: Anwenden von MinSchnittPhase mit Startknoten 3, in unserem Fall also wieder $\{2,3\}$ auf den rechts abgebildeten Graphen liefert

$$S_3 = \{\{2,3\}\}$$
$$S_3 = \{\{2,3\},4\}$$
$$S_3 = \{\{2,3\},4,5\}$$
$$S_3 = V_3$$

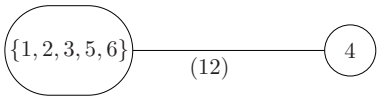
Somit ist $s = 5$ und $t = \{1,6\}$, der Schnitt der Phase ist $(V_3 \setminus \{\{1,6\}\}, \{\{1,6\}\})$ mit Gewicht 6. Verschmelze Knoten 5 und $\{1,6\}$.



Phase 4: Anwenden von MinSchnittPhase mit Startknoten 4 auf den rechts abgebildeten Graphen liefert

$$S_4 = \{4\}$$
$$S_4 = \{4,\{2,3\}\}$$
$$S_4 = V_4$$

Somit ist $s = \{2,3\}$ und $t = \{1,5,6\}$, der Schnitt der Phase ist $(V_4 \setminus \{\{1,5,6\}\}, \{\{1,5,6\}\})$ mit Gewicht 4. Verschmelze Knoten $\{2,3\}$ und $\{1,5,6\}$.



Phase 5: Anwenden von MinSchnittPhase mit Startknoten 5, also $\{1, 2, 3, 5, 6\}$ auf den rechts abgebildeten Graphen liefert

$$S_5 = \{\{1, 2, 3, 5, 6\}\}$$
$$S_5 = V_5$$

Somit ist $s = \{1, 2, 3, 5, 6\}$ und $t = \{4\}$, der Schnitt der Phase ist $(V_5 \setminus \{4\}, \{4\})$ mit Gewicht 12.

Unter allen Schnitten ist der minimale Schnitt der Schnitt aus Phase 4 mit Gewicht 4. Dieser Schnitt $(V_4 \setminus \{\{1, 5, 6\}\}, \{\{1, 5, 6\}\})$ induziert auf G den Min–Schnitt $(\{\{2, 3, 4\}, \{1, 5, 6\})$, siehe auch Abbildung 2.

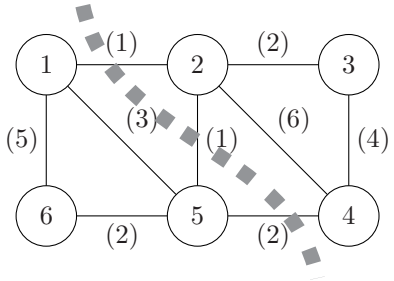


Abbildung 2: Graph G mit minimalem Schnitt

(b) An welcher Stelle im Korrektheitsbeweis zum Algorithmus von Stoer und Wagner wurde verwendet, dass die Kantengewichte nicht negativ sind? Finden Sie ein Beispiel eines Graphen mit negativen Kantengewichten und einen Startknoten a, so dass der Algorithmus von Stoer und Wagner keinen minimalen Schnitt liefert.

Proof. Das nichtnegative Kantengewicht wird im Induktionsschritt zum Beweis der Korrektheit von MinSchnittphase benutzt. Nachdem argumentiert wird dass :

$$\begin{aligned} (S_u \setminus S_v, u) &\subseteq (S'_u, (V \setminus S') \cap (S_u \cup \{u\})) && \text{und} \\ (S'_v, (V \setminus S') \cap (S_v \cup \{v\})) &\subseteq (S'_u, (V \setminus S') \cap (S_u \cup \{u\})) && \text{zudem gilt noch} \\ (S_u \setminus S_v, u) \cap (S'_v, (V \setminus S') \cap (S_v \cup \{v\})) &= \emptyset && \text{(Disjunktheit),} \end{aligned}$$

wird geschlossen dass aus dem Addieren zweier Schnittkosten die Summe nicht kleiner sein wird, als die Summanden:

$$\begin{aligned} c(S_u, u) &\leq c(S'_v, (V \setminus S') \cap (S_v \cup \{v\})) + c(S_u \setminus S_v, u) \\ &\leq c(S'_u, (V \setminus S') \cap (S_u \cup \{u\})) \qquad \text{(Nur bei nichtnegativen Summanden!).} \end{aligned}$$

Die letzte Zeile benutzt die Nichtnegativität der Kantengewichte. Ein Gegenbeispiel mit Startknoten a ist in Abbildung 3 zu sehen ist. Nach der ersten Iteration des Algorithmus werden die Knoten c und d kontrahiert. Danach kann die optimale Lösung nicht mehr gefunden werden. □

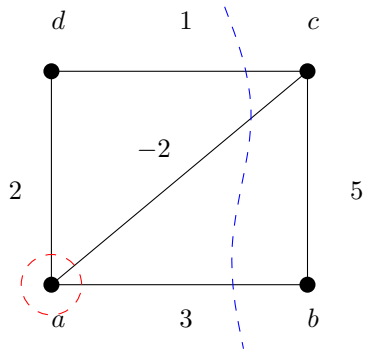


Abbildung 3: Der minimale Schnitt ist blau gestrichelt gezeichnet (Gewicht 2). Der Algorithmus von Stoer und Wagner findet jedoch nur einen Schnitt mit Gewicht 3.

Problem 3: Flüsse

Gegeben sei ein Flussproblem in einem Netzwerk $D = (V, E)$, in dem es zu einigen Kanten (u, v) auch Kanten (v, u) gibt. Zeigen Sie, dass man dieses Flussproblem auf ein Flussproblem auf dem Netzwerk $D' = (V, E')$ überführen kann, wobei gilt: E' ist maximale Teilmenge von E mit $(u, v) \in E' \Rightarrow (v, u) \notin E'$, so dass der Wert des Maximalflusses nicht verändert wird.

Proof. Sei f ein maximaler Fluss auf D und $(u, v), (v, u) \in E$ mit $f(u, v) \geq f(v, u)$. Wir zeigen zunächst, dass das Flussnetzwerk $D' = (V, E')$ mit $E' := E \setminus \{(v, u)\}$ den Wert des Maximalflusses erhält. Damit ist klar, dass die Aussage der Aufgabe gilt. Dazu definieren wir den Fluss f' durch

$$f'(i,j) := \begin{cases} f(i,j) & \text{falls } (i,j) \neq (u,v) \\ f(i,j) - f(j,i) & \text{falls } (i,j) = (u,v) \; . \end{cases}$$

Der Fluss f' erfüllt die Flussbedingung wegen $f(u, v) \geq f(v, u)$, d.h.

$$0 \leq f'(i,j) \leq c(i,j) \; .$$

Außerdem bleibt die Kapazitätsbedingung erhalten für alle Knoten $i \neq u, v$ trivialerweise erhalten. Für Knoten u gilt:

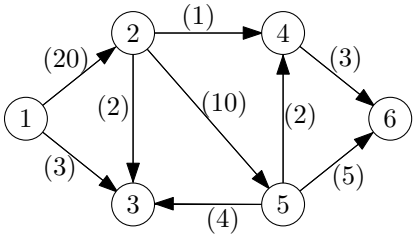
$$\begin{aligned} w'(u) &= \sum_{(w,u)\in E'} f'(w,u) - \sum_{(u,w)\in E'} f'(u,w) \\ &= \sum_{(w,u)\in E'} f'(w,u) - \sum_{(u,w)\in E'\setminus\{(u,v)\}} f'(u,w) - f'(u,v) \\ &= \sum_{(w,u)\in E'} f(w,u) - \sum_{(u,w)\in E'\setminus\{(u,v)\}} f(u,w) - (f(u,v) - f(v,u)) \\ &= \sum_{(w,u)\in E} f(w,u) - \sum_{(u,w)\in E} f(u,w) \\ &= w(u) \end{aligned}$$

Für $u \neq s, t$ gilt $w(u) = 0$. Folglich gilt auch hier die Flusserhaltung. Die Gleichung zeigt darüber hinaus, dass $w'(f') = w(f)$ gilt, da für $u = s$ die Gleichung $w'(f') = w'(s) = w(s) = w(f)$ gilt. Sei $\tilde{f}$ ein Maximalfluss in D' dann gilt also $w'(\tilde{f}) \geq w'(f') = w(f)$. Da sich jeder Fluss in D' trivial zu einem Fluss in D fortsetzen lässt, gilt auch $w'(\tilde{f}) \leq w(f)$ und damit $w'(\tilde{f}) = w(f)$.

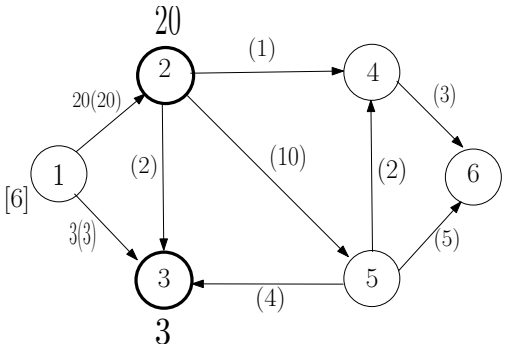
□

Problem 4: Goldberg–Tarjan

Bestimmen Sie den maximalen Fluss in dem unten angegebenen Netzwerk mit $s = 1$ und $t = 6$ (Kantenkapazitäten in Klammern). Benutzen Sie hierzu den Algorithmus von Goldberg–Tarjan. Geben Sie **sämtliche** Zwischenschritte an. Zeichnen Sie das aktuelle Netzwerk **vor** jeder (bis auf die erste) RELABEL-Operation und nachdem der Algorithmus terminiert. Halten Sie sich an die Notation aus dem Skript.



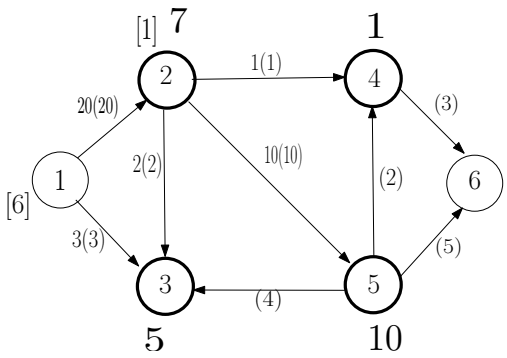
Die erste Darstellung zeigt das Netzwerk zu Beginn des Algorithmus



Wir führen jetzt die folgenden Operationen durch:

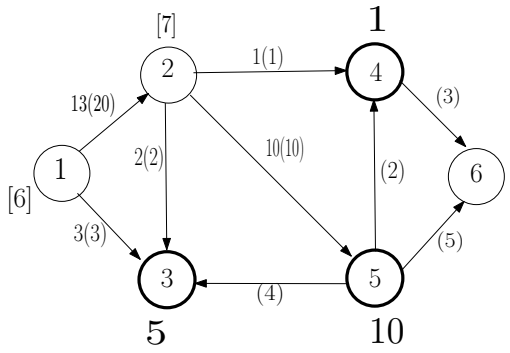
- RELABEL(2). Dann ist $dist(2) = 1$.
- PUSH(2,3) mit $\Delta = 2$ und PUSH(2,4) mit $\Delta = 1$ und PUSH(2,5) mit $\Delta = 10$. Dann ist $e(2) = 7$, $e(3) = 5$, $e(4) = 1$,$e(5) = 10$

  Der neue Zwischenstand:



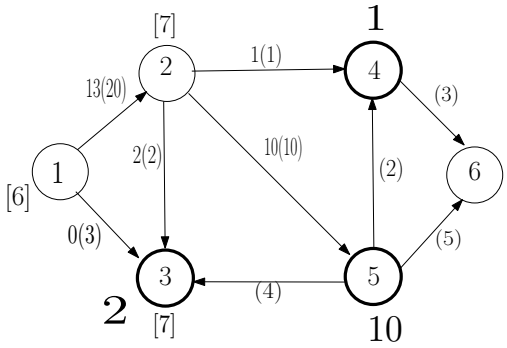
- RELABEL(2). Da $r_f(2, 1) > 0$ und $dist(2) < dist(1)$ ist, folgt $dist(2) := dist(1) + 1 := 7$
- PUSH(2,1) mit $\Delta = 7$ Dann $f(1, 2) = 13$ und $e(2) = 0$

Jetzt haben wir den folgenden Zwischenstand im Netzwerk:



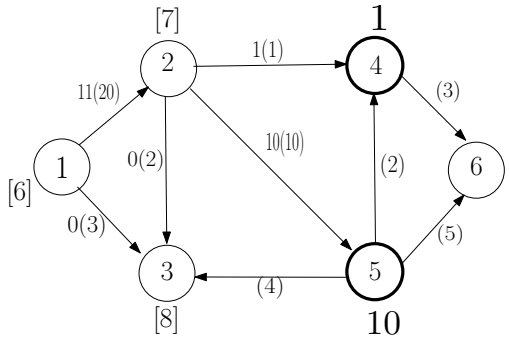
Nun folgen diese Operationen

- RELABEL(3). Da $r_f(3,1) > 0$ und $dist(3) < dist(1)$ ist, folgt $dist(3) := dist(1) + 1 := 7$
- PUSH(3,1) mit $\Delta = 3$. Dann ist $e(3) = 2$
  Der neue Zwischenstand:



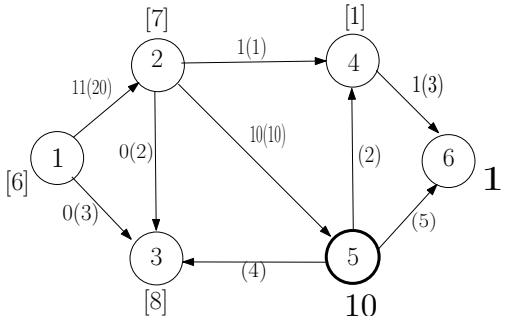
- RELABEL(3). Da $r_f(3,2) > 0$ und $dist(3) < dist(2)$ ist, folgt $dist(3) := dist(2) + 1 := 8$
- PUSH(3,2) mit $\Delta = 2$. Dann $e(2) = 2$ und $e(3) = 0$
- PUSH(2,1) mit $\Delta = 2$. Dann $e(2) = 0$

Der neue Zwischenstand sieht so aus:

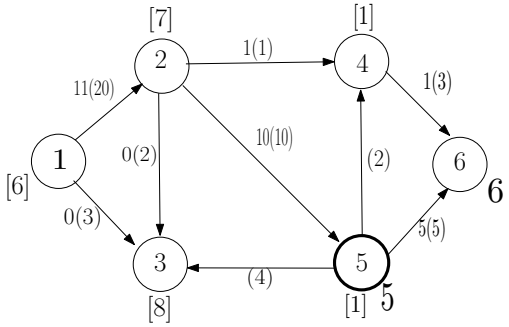


Weiter:

- RELABEL(4), folgt $dist(4) := 1$. PUSH(4,6) mit $\Delta = 1$.Dann ist $e(4) = 0$ und $e(6) = 1$
  Der neue Zwischenstand:

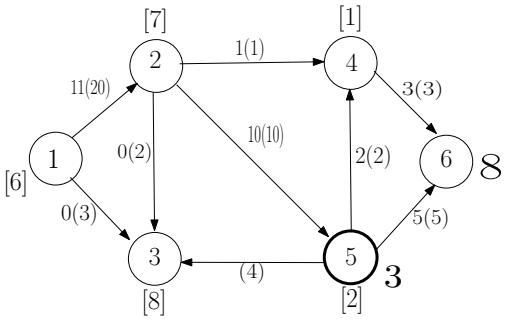


- RELABEL(5) folgt $dist(5) := 1$. PUSH(5,6) mit $\Delta = 5$.Dann $e(5) = 5$ und $e(6) = 6$
  Der neue Zwischenstand:



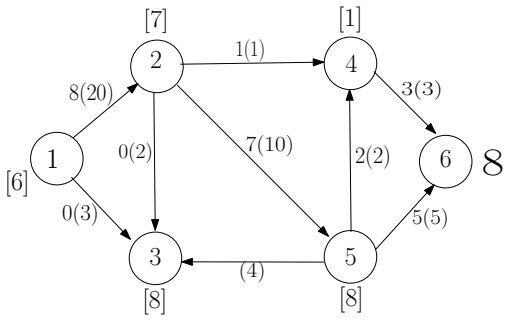
- RELABEL(5),folgt $dist(5) := dist(4) + 1 = 2$. PUSH(5,4) mit $\Delta = 2$.Dann ist $e(5) = 3$ und $e(4) = 2$
- PUSH(4,6) mit $\Delta = 2$. Dann ist $e(4) = 0$ und $e(6) = 8$

Das Netwerk sieht dann so aus:



Zum Schluss noch:

- RELABEL(5). Da $r_f(5, 2) > 0$ und $dist(5) < dist(2)$ ist, folgt $dist(5) := dist(2) + 1 := 8$.
  PUSH(5,2) mit $\Delta = 3$. Dann ist $e(5) = 0$ und $e(2) = 3$
- PUSH(2,1) mit $\Delta = 3$. Dann ist $e(2) = 0$



Nun ist kein Knoten mehr aktiv, der Algorithmus terminiert. Und maximal Fluss ist 8.

Problem 5: Das Escape Problem

Gegeben seien $n \times n$ Gitterpunkte.

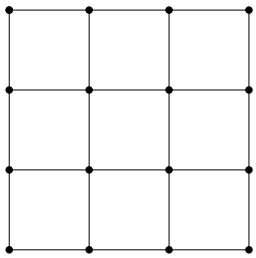


Abbildung 4: Gitternetzwerk für $n = 4$

Sei S eine Teilmenge der Gitterpunkte mit $|S| < n^2$. Das *Escape Problem* besteht darin zu entscheiden, ob es $|S|$ viele knotendisjunkte Wege gibt, so dass jeder Weg von einem Knoten aus S startet und an einem Randpunkt des Gitters endet.

(a) Geben Sie für $n = 4$ eine Ja-Instanz mit möglichst vielen Startknoten an, sowie eine Nein-Instanz mit möglichst wenig Startknoten.

Proof. Abbildung 5 zeit eine Ja-Instanz mit möglichst vielen Startknoten. Intuition: Es gibt 16 Knoten im Gitter aber nur 12 Randknoten, daher muss gelten $|S| \leq 12$. Abbildung 6 zeit eine Nein-Instanz mit möglichst wenigen Startknoten. Intuition: Ein innerer Gitterknoten hat immer mindestens 4 Wege zum Rand (geradeaus nach links/rechts/oben/unten), daher muss ein Knoten von mindestens 4 Knoten umbaut werden um jeden Weg zu einem Randpunkt zu blockieren. □

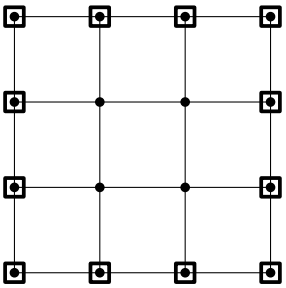


Abbildung 5: Große Ja-Instanz

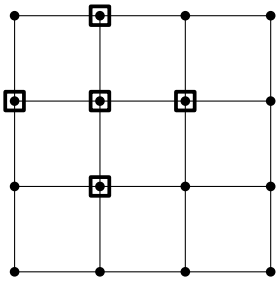


Abbildung 6: Kleine Nein-Instanz

Das maximale Flussproblem aus der Vorlesung kann folgendermaßen erweitert werden: Neben den Kantenkapazitäten c seien noch Knotenkapazitäten $\gamma : V \to \mathbb{R}_0^+$ gegeben. In einem solchen Netzwerk $(D; s; t; c; \gamma)$ heißt eine Abbildung f *Fluss*, wenn sie neben den bekannten Eigenschaften (Flusserhaltungsbedingung und (Kanten)-Kapazitätsbedingung) auch die folgende erfüllt:

- Für alle $i \in V$ ist die *Knotenkapazitätsbedingung*

$$\sum_{\{j|(i,j)\in E\}} f(i,j) \leq \gamma(i) \quad \text{wenn} \quad i \in V \setminus \{t\}$$
$$\sum_{\{j|(j,i)\in E\}} f(j,i) \leq \gamma(i) \quad \text{wenn} \quad i = t$$

  erfüllt.

(b) Zeigen Sie, dass die Bestimmung eines maximalen Flusses in einem Netzwerk mit Kanten- **und** Knotenkapazitäten auf ein maximales Flussproblem in einem normalen (d.h. ohne Knotenkapazitäten) Netzwerk mit vergleichbarer Größe zurückgeführt werden kann. Abbildung 7 zeigt wie ein Knoten mit Knotenkapazität umgeformt werden muss, um ein solches Netzwerk auf ein normales Flussnetzwerk zurückzufhren. Die Umformung ist flussäquivalent, da durch

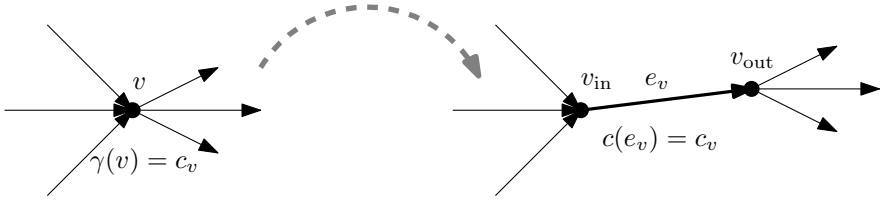


Abbildung 7: Knotenkapaziät wird flussäquivalent in Kantenkapazität umgeformt

die Kapazitätsbedingung der Kante e_v die Knotenkapazitätsbedingung $\gamma(v)$ realisiert wird und alle sonstigen Nachbarschaften und Kantenkapazitätsbedingungen beibehalten werden. Die Knotenkapazitätsbedingung beschränkt den Ausfluss eines Knotens v ebenso wie die Kapazität der Kante e_v. Für t_{out} ist analog der Einfluss beschränkt. Der Fluss im Rest des Netzwerkes bleibt durch diese Umwandlung somit unverändert. Es ist leicht zu sehen, dass jeder Fluss im Netzwerk vor der Umwandlung auch im Netzwerk nach der Umwandlung gültig ist, und umgekehrt.

Formal gilt für ein Netzwerk also: Konstruiere Netzwerk $N' = (D' = (V', E'); s'; t'; c')$ aus einem Netzwerk $N = (D = (V, E); s; t; c; \gamma)$, das Knotenkapazitäten enthält, wie folgt:

- Knoten: $V' = \bigcup_{v\in V} \{v_{\text{in}}, v_{\text{out}}\}$
- Quelle und Senke: $s' = s_{\text{in}}$, und $t = t_{\text{out}}$
- Kanten $E' = \{(v_{\text{out}}, w_{\text{in}}) \mid (v, w) \in E\} \quad \cup \quad \{(v_{\text{in}}, v_{\text{out}}) \mid v \in V\}$
- Kapazitäten: $c' : E' \to \mathbb{R}_0^+$ mit $c'(v_{\text{out}}, w_{\text{in}}) = c(v, w)$ und $c'(v_{\text{in}}, v_{\text{out}}) = \gamma(v)$

Korrektheit: Offenbar ist N' ein normales Flussnetzwerk von vergleichbarer Größe wie N ($|E'| = |E| + |V|$, $|V'| = 2|V|$). Da die Umwandlung eines jeden Knoten den Fluss im Rest des Netzwerkes unverändert lässt und die Knotenbedingungen erfüllt bleiben, gilt zudem, dass der maximale Fluss in N und in N' den gleichen Wert haben. Eine Lösung in N' kann kanonisch auf eine Lösung in N übersetzt werden.

(c) Wie würden Sie das Escape-Problem algorithmisch lösen? Geben Sie die Worst-case Laufzeit Ihres Ansatzes an. *Hinweis:* Pseudocode ist nicht gefordert! Zunächst konstruieren wir aus dem Gitter ein Flussnetzwerk mit Knotenkapazitäten $\gamma \equiv 1$, wie in Abbildung 8 und wandeln

dieses dann entsprechend Teilaufgabe (b) in ein normales Flussnetzwerk um. Die Intuition ist wie folgt:

Von der Quelle s verlaufen Kanten zu jedem möglichen Startknoten (Gefangene starten in Zellen), und nur von den Randknoten Kanten zur Senke t (Gefangene flüchten von Randknoten aus). Diese Kanten haben alle die Kapazität 1 da an jedem Gitterpunkt nur ein Startknoten liegt und an jedem Randknoten nur ein Weg enden darf.

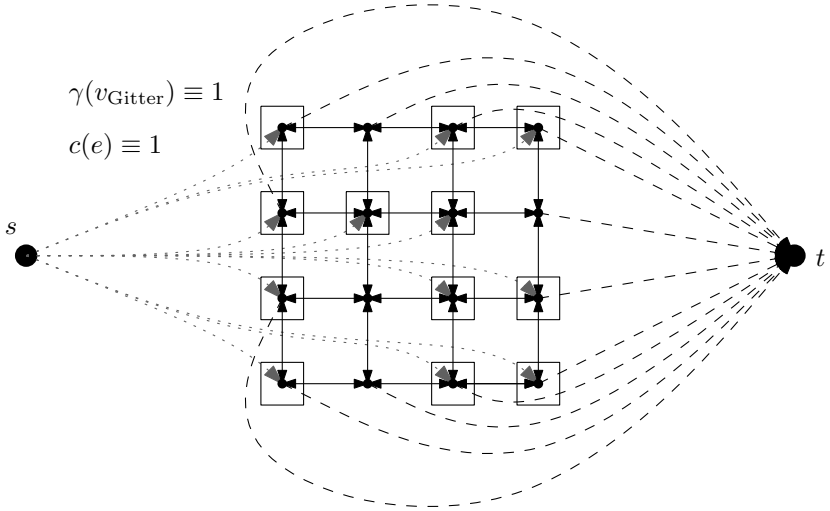


Abbildung 8: Das erweiterte Flussnetzwerk zu einer Instanz des Escape Problems (die Startknoten aus S sitzen in Kisten). Alle Startknoten werden von s gespeist, t ist Fluchtweg.

Die Gitterkanten sind bidirektional. Die Gitterknoten haben die Kapazität 1 da sie nur einmal genutzt werden dürfen. Die Forderung, dass Gitterkanten die Kapazität 1 haben ist zwar redundant (durch Knotenbedingungen schon erfüllt), macht die Situation aber verständlicher.

Ein maximaler Fluss in diesem Netzwerk entspricht dann direkt einer Menge von knotendisjunkten Wegen von Gitterknoten in S zu Randknoten. Hat der maximale Fluss den Wert $|S|$, so existiert genau für jeden Startknoten ein solcher Weg.

Wandeln wir nun dieses erweiterte Flussproblem entsprechend Teilaufgabe (b) in ein normales Flussnetzwerk um, so können wir dies mit bekannten Mitteln (etwa Goldberg-Tarjan) lösen. In Abbildung 9 ist dies illustriert.

Laufzeit:

- gegeben: Instanz der Größe n $\rightarrow n \times n$ Gitterpunkte

Transformation Flussproblem mit Knotenkapazität:

- $|V| = n^2 + 2$ Knoten
- $|E| = |S| < n^2$ Kanten von Masterquelle
- $+4 \cdot (n-1)$ Kanten zur Mastersenke
- $+2n(n-1) \cdot 2$ Kanten zw. Gitterpunkten
- $\rightarrow O(n^2)$ Knoten und $O(n^2)$ Kanten

Transformation Flussproblem ohne Knotenkapazität

- $|E'| = |E| + |V|$ Kanten $\in O(n^2)$
- $|V'| = 2|V|$ Knoten $\in O(n^2)$

Laufzeit Transformationen $\in O(n^2)$. Laufzeit max. Fluss wie folgt:

- Goldberg-Tarjan: $O(|V'|^2|E'|) = O(n^6)$.
- Goldberg-Tarjan-Highest-Label: $O(|V'|^2|E'|^{\frac{1}{2}}) = O(n^5)$.
- Goldberg-Tarjan-Excess-Scaling: $O(|E'||V'| + |V'|^2 \log C) \underset{\text{C konstant}}{=} O(|E'||V'| + |V'|^2) = O(n^4)$.
- Ford-Fulkerson: $O(|E'| \cdot |f^*|) \overset{\text{größte Ja-Instanz}}{\underset{\in O(4(n-1))}{=}} O(|E'| \cdot 4(n-1)) = O(n^3)$.

Insgesamt: Laufzeit $\in O(n^2 + n^3) = O(n^3)$.

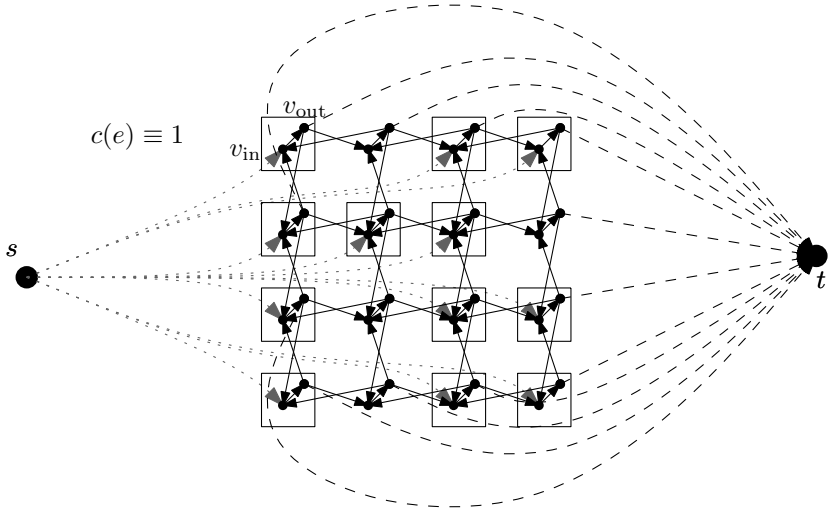


Abbildung 9: Das umgewandelte normale Flussnetzwerk. Aus Gründen der Übersicht wurden Quelle und Senke nicht verdoppelt, es liegen dort auch keine echten Knotenkapazitätsbedingungen vor.