Universität Augsburg
Mathematisch-Naturwissenschaftlich-
Technische Fakultät

Vorlesung

Kombinatorische Optimierung

gehalten im SS 2017, SS 2019, SS 2021

Prof. Dr. Tobias Harks

Universität Augsburg
Institut für Mathematik
Universitätsstraße 14
86159 Augsburg

Email: tobias.harks@math.uni-augsburg.de

4. Juli 2021

Vorwort

Der erste Teil dieses Skripts basiert auf einem Vorlesungsskript Graphen- und Netzwerkalgorithmen das von Prof. Dr. Rolf Möhring konzipiert wurde. Die Vorlesung habe ich im Sommersemester 2011 an der Technischen Universität Berlin gehalten und das Skript von Prof. Dr. Rolf Möhring ist an einigen Stellen durch eigene Notizen ergänzt bzw. erweitert. Dieses Skript dient primär der begleitenden Darstellung der Inhalte der Vorlesung und ist nicht als eigenständige Veröffentlichung gedacht.

Augsburg, April 2017,
überarbeitet, Juli 2019
Prof. Dr. Tobias Harks

Inhaltsverzeichnis

Kapitel 1

Maximale Flüsse in Netzwerken

Wir führen in diesem Abschnitt den Begriff eines Flusses in einem Transportnetzwerk ein. In vielen Anwendungen möchte man ein Gut durch ein Netzwerk mit beschränkten Kapazitäten transportieren (Wasser, Gas, Autos, etc.).

Definition 1.1. Ein Fluss Netzwerk besteht aus Digraph $G = (V, E)$, Kanten-Kapazitäten $u(e) \geq 0, e \in E(G)$ und 2 ausgezeichneten Knoten s (Quelle) und t (Senke). Bezeichnung: (G, u, s, t).

Definition 1.2 (Fluss, Zirkulation, s,t-Fluss).

1. Ein Fluss bzgl. (G, u) ist eine Kantenbewertung $f(e) \geq 0, e \in E(G)$, mit $f(e) \leq u(e)$ für alle $e \in E(G)$ (Kapazitätseinhaltung). Ein Fluss f erfüllt die Flusserhaltungsgleichung in Knoten v falls Ausfluss aus v = Einfluss in v ist, also
$$\sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)} f(e).$$
2. Eine Zirkulation oder Strömung ist eine Kantenbewertung , die in jedem Knoten die Flusserhaltungsgleichung erfüllt (negative Werte zugelassen).
3. Ein s,t-Fluss im Netzwerk (G, u, s, t) ist ein Fluss f, der in allen Knoten $v \neq s, t$ die Flusserhaltungsgleichungen erfüllt. Der Wert $\nu(f)$ von f ist der Nettoausfluss aus der Quelle, d.h.
$$\nu(f) = \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e).$$

Anschauung:

- Kanten e = Röhren, Kapazität $u(e)$ = "Dicke" der Röhren
- $f(e)$ = Fluss einer Flüssigkeit pro Zeiteinheit durch Querschnitt der Röhre, durch $u(e)$ beschränkt
- Flusserhaltung = unterwegs geht nichts verloren und kommt nichts hinzu.

Kantenbewertung $f(e), e \in E(G)$, ist Vektor $f \in \mathbb{R}^{E(G)}$.

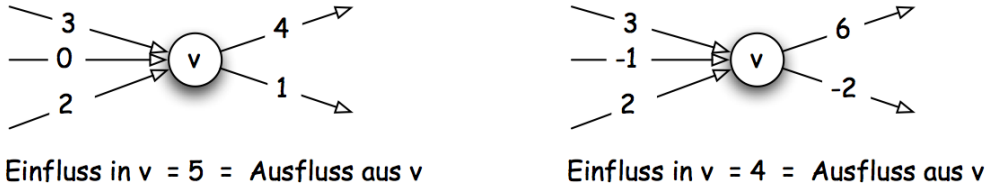


Abbildung 1.1: Illustration der Flusserhaltung eines Flusses und einer Zirkulation im Knoten v.

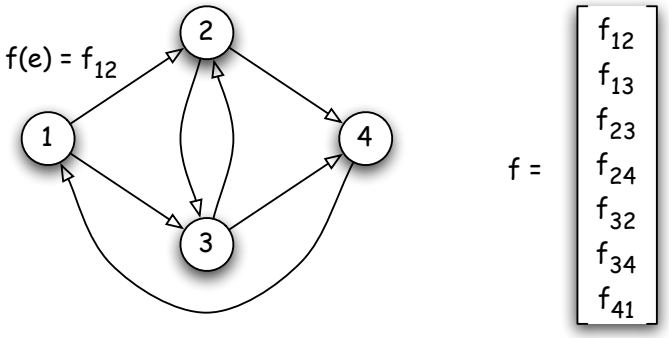


Abbildung 1.2: Illustration eines Flussvektors.

1.1 Zykelraum

Wir betrachten einen ungerichteten elementaren Kreis C eines Digraphen $G = (V, E)$. Wir bezeichnen mit $\zeta(C) \in \{-1, 0, 1\}^{E(G)}$ den Inzidenzvektor von C so dass gilt (bzgl. fester Orientierung von C)

$$\zeta_e(C) = \begin{cases} 1, \text{ falls } e \text{ eine Vorwärtskante (VK) von } C, \\ -1, \text{ falls } e \text{ eine Rückwärtskante (RK) von } C, \\ 0, \text{ wenn } e \notin C. \end{cases} \tag{1.1}$$

Wir geben ein Beispiel.

Beispiel 1.3. Betrachte den Digraph in Abb. 1.3. Der Kreis C mit Orientierung ist rechts abgebildet.

Der zugehörige Inzidenzvektor ergibt sich mittels Definition (1.1) zu:

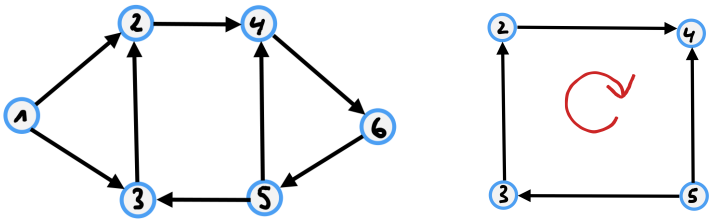


Abbildung 1.3: Kreis C mit Orientierung in G.

$$\zeta(C) = \begin{pmatrix} \zeta_{12} \\ \zeta_{13} \\ \zeta_{24} \\ \zeta_{32} \\ \zeta_{46} \\ \zeta_{53} \\ \zeta_{54} \\ \zeta_{65} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ -1 \\ 0 \end{pmatrix}.$$

Definition 1.4 (Zykelraum (cycle space)). Der Zykelraum von G ist der von den Inzidenzvektoren von elementaren Kreisen erzeugte Untervektorraum von $\mathbb{R}^{E(G)}$.

1.2 Strömungen, Zykelraum und Dekompositionssatz

Lemma 1.5. f ist Strömung in G $\Leftrightarrow$ f ist im Zykelraum von G.

Beweis. Strömungen bilden Untervektorraum von $\mathbb{R}^{E(G)}$:

- f, g Strömungen $\Rightarrow$ f + g ist Strömung.
- f Strömung, $\alpha \in \mathbb{R} \Rightarrow \alpha f$ ist Strömung.

Wir zeigen zunächst: Der Zykelraum ist ein Untervektorraum der Strömungen. Jeder Inzidenzvektor eines Kreises ist eine Strömung. Somit sind die Linearkombinationen von Inzidenzvektoren von Kreisen Strömungen.
Als nächstes zeigen wir: Strömungen bilden einen Untervektorraum des Zykelraums. Wir zeigen: jede Strömung ist Linearkombinationen von Inzidenzvektoren von Kreisen.
Beweis durch Induktion nach der Anzahl der Kanten e mit $f(e) \neq 0$. Induktionsanfang:

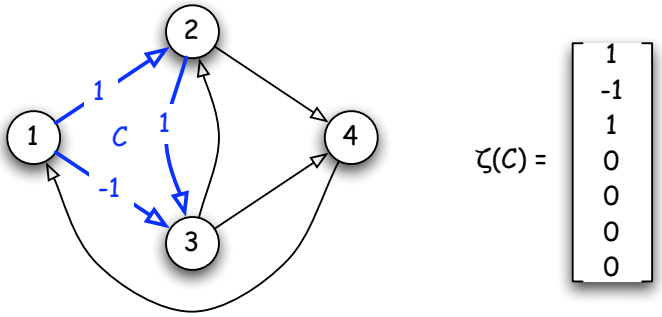


Abbildung 1.4: Illustration eines Zykels.

$f(e) = 0$ für alle $e \Rightarrow f = 0 \Rightarrow f$ ist Linearkombinationen von Inzidenzvektoren von Kreisen.
Induktionsschluss: Sei $f \neq 0$ eine Strömung. Betrachte eine Kante $e_0 = (u, v)$ mit $f(e_0) \neq 0$, o.B.d.A. $f(e_0) > 0$ (sonst betrachte $-f$). Flusserhaltung in v impliziert dass es eine zu v inzidente Kante $e_1 = (v, w)$ gibt mit $f(e_1) > 0$ oder $e_1 = (w, v)$ mit $f(e_1) < 0$.

Abbildung 1.5: Illustration des obigen Arguments.

Fortsetzung des Arguments mit w liefert nach endlich vielen Schritten (ungerichteten) Kreis C mit $f(e) > 0$ auf Vorwärtskanten und $f(e) < 0$ auf Rückwärtskanten. Sei

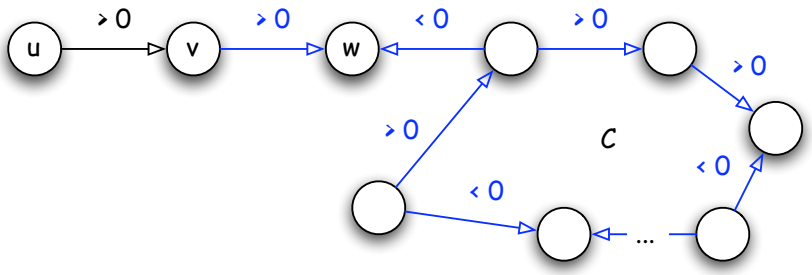


Abbildung 1.6: Illustration des enstehenden Kreises.

$$\epsilon := \min\{|f(e)| : e \in E(C)\}.$$

Nach Konstruktion ist $g := f - \epsilon\zeta(C)$ eine Strömung mit weniger Kanten mit $g(e) \neq 0$ als f. Auf g trifft die Induktionsvoraussetzung zu und somit ist g Linearkombination von Inzidenzvektoren von Kreisen. Daher ist $f = g + \epsilon\zeta(C)$ auch eine Linearkombinationen von Inzidenzvektoren von Kreisen. □

Satz 1.6 (Dekompositionssatz für s, t-Flüsse, Ford-Fulkerson 1962). Sei $f \neq 0$ ein s,t-Fluss in (G,u,s,t) und es gelte $\nu(f) \geq 0$. Dann gilt:

1. f ist positive Linearkombination von (Inzidenzvektoren von) gerichteten s,t-Wegen und gerichteten Kreisen
2. die Anzahl der Wege und Kreise kann auf höchstens m beschränkt werden
3. ist f ganzzahlig, so können die Koeffizienten in der Linearkombination als ganzzahlig gewählt werden

Beweis. Füge zu G die Kante (t,s) hinzu und setze $f(t,s) := \nu(f)$. (Es könnte sein, dass G die Kante (t,s) schon enthält, dann setzt man $f(t,s) := f(t,s) + \nu(f)$ und argumentiert ansonsten analog.)

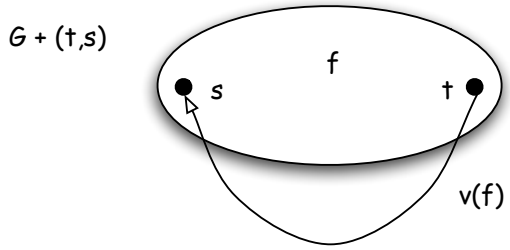


Abbildung 1.7: Illustration des Beweises des Dekompositionssatzes.

Somit ist f eine Strömung in $G+(t,s)$ und es gilt $f \geq 0$. Wie in der Konstruktion der Linearkombination im Beweis von Lemma 1.5 gibt es nur Vorwärtskanten und $\epsilon > 0$ (ganzzahlig bei ganzzahligen Flusswerten $f(e)$). In jedem Induktionsschritt können wir $\epsilon > 0$ abziehen so dass ein Kantenwert in G zu 0 wird. Die resultierende Strömung f in $G+(t,s)$ ist somit eine positive Linearkombination von maximal m gerichteten Kreisen in $G+(t,s)$. Solche Kreise, die die Kante (t,s) enthalten, sind in G gerichtete s,t-Wege. □

Beispiel 1.7. Wir geben ein Beispiel eines Ausgangsgraph mit s,t.

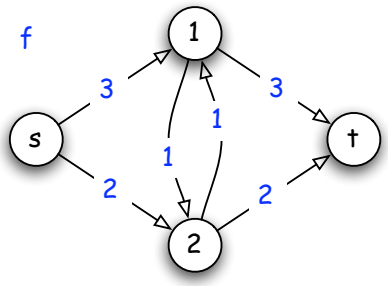


Abbildung 1.8: Anwendung des Dekompositionssatzes.

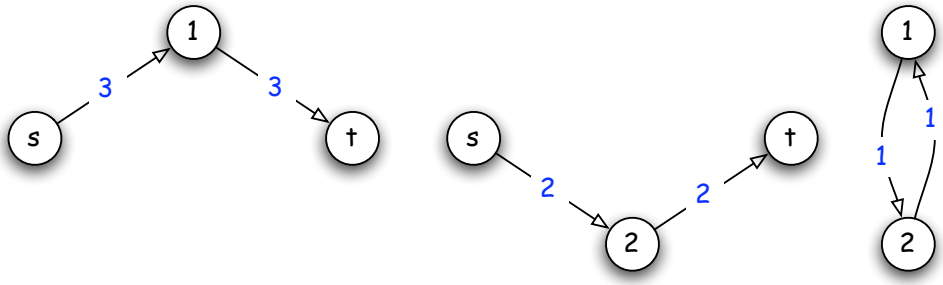


Abbildung 1.9: Zerlegung in Wege und Kreise.

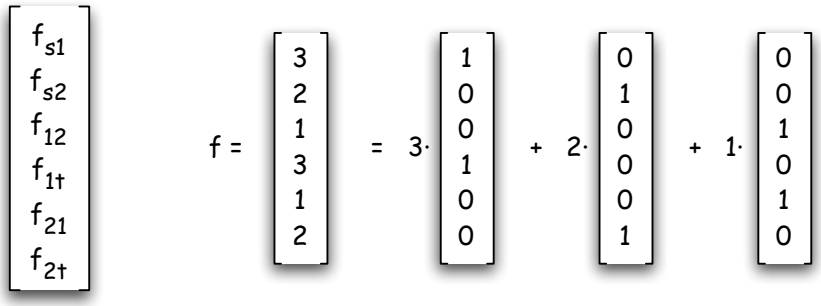


Abbildung 1.10: Linearkombination des Dekompositionssatzes.

Der Begriff Fluss ist kantenweise definiert. Der Dekompositionssatz gibt eine fluss- oder wegorientierte Sicht: wie fliesst der Fluss von s nach t, welche Wege und Kreise benutzt er.

1.3 Lösbarkeit des Max-Fluss Problems

Maximum Flow Problem (MFP)
Instanz: Netzwerk (G, u, s, t)
Aufgabe: Finde s,t-Fluss mit maximalem Wert.

Proposition 1.8. Das MFP hat eine optimale Lösung.

Beweis. Betrachte s,t-Flüsse als Vektoren und sei $X \subseteq \mathbb{R}^{E(G)}$ die Menge dieser Vektoren. Es gilt $X \neq \emptyset$, da $0 \in X$. Weiterhin ist X beschränkt, da

$$X \subseteq \prod_{e \in \mathbf{E}(G)} [0, u(e)].$$

Nebenbedingungen an f sind über $=$ und $\leq$ definiert somit ist X abgeschlossen. Also: X ist nichtleere kompakte Teilmenge von $\mathbb{R}^{E(G)}$. Die Zielfunktion $\nu(f)$ ist linear, also stetig. $\nu(f)$ nimmt das Maximum auf X an. □

1.4 Der Max-Flow Min-Cut Satz

Wir werden nun den (evtl. aus Optimierung II bekannten) Max-Flow Min-Cut Satz konstruktiv beweisen.

1.4.1 Kapazität von Schnitten

Zur Erinnerung: ein s,t-Schnitt ist eine Knotenmenge A mit induzierter Kantenmenge $\delta^+(A) \subseteq E(G)$ mit $s \in A$ und $t \notin A$. Die Menge $\delta^+(A)$ sind die Vorwärtskanten des Schnittes (raus aus A).

Definition 1.9. Die Kapazität des s,t-Schnittes A ist

$$c(A) = \sum_{e \in \delta^+(A)} u(e),$$

also die Summe der Kapazitäten der Vorwärtskanten des Schnittes. Ein minimaler s,t-Schnitt in (G,u,s,t) ist ein s,t-Schnitt mit minimaler Kapazität bzgl. u.

Lemma 1.10. Für jeden s,t-Schnitt A und jeden s,t-Fluss f gilt

$$\nu(f) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e)$$
$$\nu(f) \leq c(A).$$

Beweis. Berechnung von $\nu(f)$ ergibt die Gleichung

$$\begin{aligned}\nu(f) &:= \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) \\ &= \sum_{v \in A} \left(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right),\end{aligned}$$

da Flusserhaltung in allen Knoten $v \neq s,t$ und $t \notin A$ gilt.
Der letzte Term ist gleich

$$\nu(f) = \sum_{e \in \delta^+(A)} f(e) - \sum_{e \in \delta^-(A)} f(e),$$

da jede Kante (x,y) mit $x,y \in A$ einmal positiv und einmal negativ in der vorigen Summe gezählt wird. Die Ungleichung $\nu(f) \leq c(A)$ folgt nun aus

$$0 \leq f(e) \leq u(e) \text{ für alle } e \in E(G).$$

□

Obiges Lemma zeigt, dass der Flusswert nie die Kapazität eines Schnittes überschreiten kann. Tatsächlich wird sogar Gleichheit angenommen, wenn f ein maximaler Fluss und A ein minimaler Schnitt ist. Das ist die wesentliche Aussage des Max-Flow Min-Cut Satzes. Um den Satz algorithmisch konstruktiv zu beweisen benötigen wir noch weitere Vorbereitungen.

1.4.2 Residualgraph und augmentierende Wege

Definition 1.11 (Residualgraph). Aus einem Digraph G entsteht der Digraph $G^{\leftrightarrow}$ durch Hinzufügen aller Kanten in umgekehrter Richtung (parallele Kanten möglich).

1. die ursprüngliche Kante $e \in E(G)$ heisst Vorwärtskante (VK)
2. die hinzugefügte Kante $e^{\leftarrow}$ in umgekehrter Richtung heisst Rückwärtskante (RK) zu e

Sei f ein s,t-Fluss in (G,u,s,t). Die Residualkapazität bzgl. u und f ist die Kantenbewertung u_f auf $G^{\leftrightarrow}$ mit

$$u_f(e) := u(e) - f(e) \text{ für alle } e \in E(G)$$
$$u_f(e^{\leftarrow}) := f(e) \text{ für alle } e \in E(G)$$

Der Residualgraph zu G,u,f ist der Teilgraph G_f von $G^{\leftrightarrow}$ mit

$$V(G_f) := V(G)$$
$$E(G_f) := \{e \in E(G^{\leftrightarrow}) |\ u_f(e) > 0\}$$

Augmentieren eines Flusses f entlang eines s,t-Weges oder Zykels W in G_f um den Wert γ ist die Veränderung von f in G gemäß

$$f(e) := f(e) + \gamma, \text{ falls } e \in E(W) \text{ VK.}$$
$$f(e) := f(e) - \gamma, \text{ falls } e^{\leftarrow} \in E(W) \text{ RK.}$$

Ein f-augmentierender Weg (Fluss erhöhender Weg bzgl. f) ist ein elementarer gerichteter s,t-Weg in G_f.
Wir geben in Abbildung 1.11 ein Beispiel an.

Bemerkung 1.12. In dem Beispiel von Abbildung 1.11 enthält G keine gerichteten Wege, entlang derer man Fluss erhöhen kann. Es ist also sehr wesentlich, gerichtete Wege im Residualgraphen zu betrachten (oder, äquivalent dazu, Wege mit VK und RK in G). Rückwärtskanten sind also essentiell.

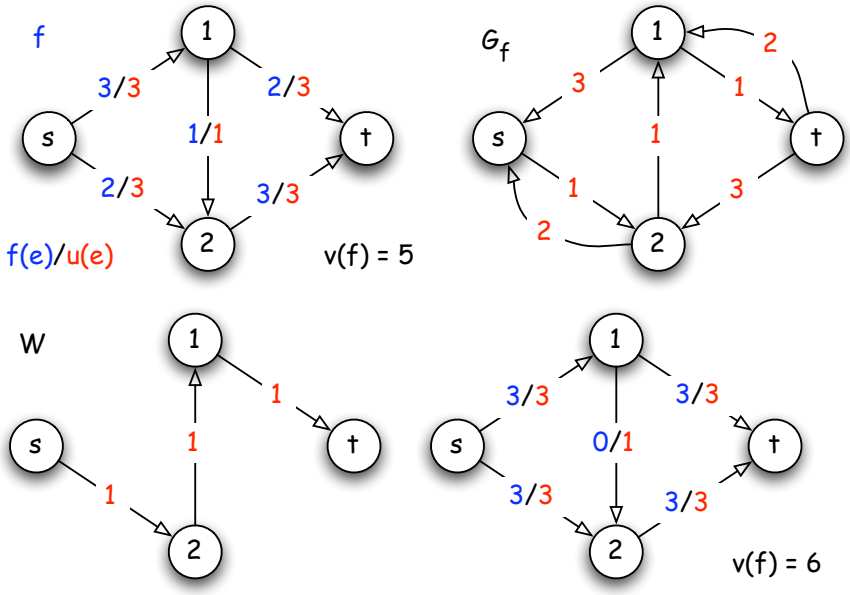


Abbildung 1.11: Augmentierung von f entlang W um $\gamma = 1$.

1.4.3 Der Ford-Fulkerson Algorithmus

Eingabe: Netzwerk (G, u, s, t).
Ausgabe: Maximaler s, t-Fluss f.
Methode:
$f(e) := 0$ für alle Kanten $e \in E(G)$
while es gibt f-augmentierenden s, t-Weg W **do**
  ermittle minimale Residualkapazität γ von W, $\gamma := \min\{u_f(e) | e \in E(W)\}$
  augmentiere f entlang W um γ.
end
return: f

Algorithm 1.4.1: Ford-Fulkerson Algorithmus

Wir wenden den Ford-Fulkerson Algorithmus auf der Beispielinstanz in Abbildung 1.12 an.

Wir halten einige Beobachtungen zum Algorithmus fest.

1. Nach jeder Augmentierung entsteht wieder ein s, t-Fluss mit einem um γ höheren Flusswert.

   - ein gerichteter Weg W in G_f entspricht einem ungerichteten Weg in G mit VK und RK
   - Augmentierung entlang W = Erhöhung auf VK, Erniedrigung auf RK, jeweils um γ und dies erhält die Flusserhaltungsgleichungen in den Knoten
   - Fluss $f(e)$ bleibt wegen Wahl von γ zwischen 0 und $u(e)$
   - Augmentierung erhöht $v(f)$ um γ

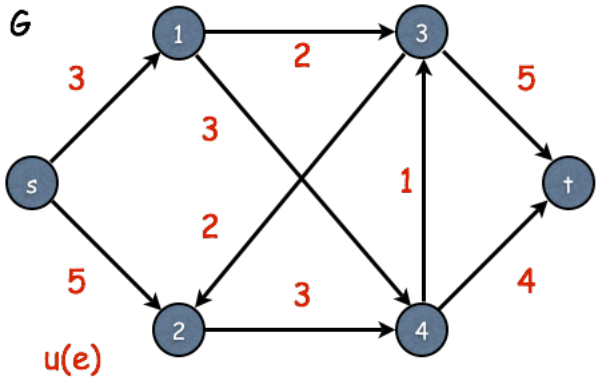


Abbildung 1.12: Beispielinstanz – Volle Rechnung in der Vorlesung.

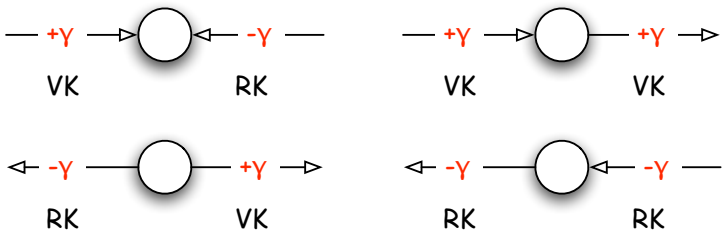


Abbildung 1.13: Einhaltung der Flusserhaltung.

Abbildung 1.14: Augmentierung um $\gamma > 0$.

2. Die Existenz bzw. Ermittlung f-augmentierender s,t-Wege kann mit Breitensuche/Tiefensuche in G_f in $O(m)$ Zeit erfolgen. Mn startet in s und stoppt, wenn t erreicht bzw. wenn es nicht mehr weiter geht.

3. Der Erhöhungswert γ kann klein sein und viele Iterationen erfordern.

4. Bei irrationalen Kapazitäten gibt es Beispiele, in denen der Algorithmus nicht einmal terminiert und die Folge der Flusswerte nicht gegen den optimalen Flusswert konvergiert (Übung).

5. Frage: Ist der gefundene Fluss bei Abbruch maximal?

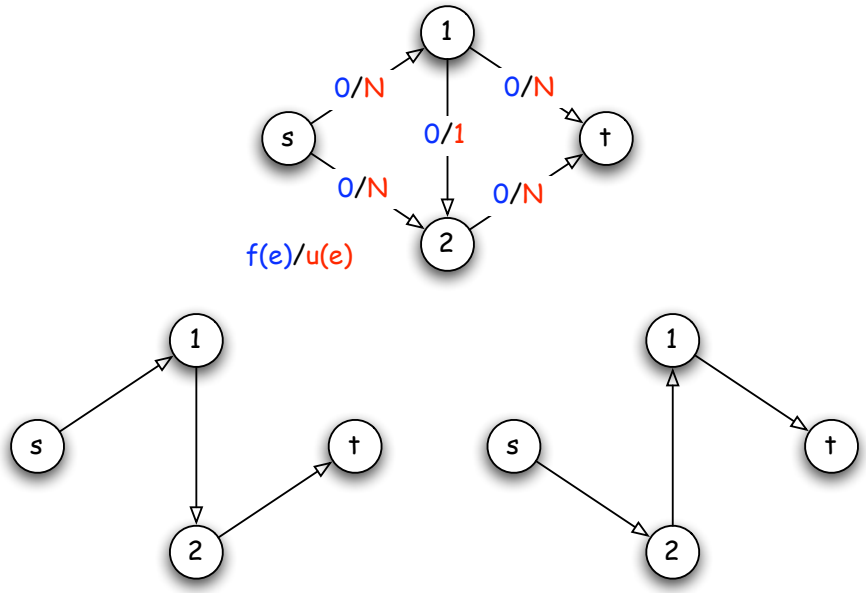


Abbildung 1.15: Beispiel von 2N augmentierenden Wegen mit Wert $\gamma = 1$.

1.4.4 Optimalitätskriterium für maximale Flüsse und der Max-Flow Min-Cut Satz

Satz 1.13 (Optimalitätskriterium für maximale s, t-Flüsse). f ist maximal $\Leftrightarrow$ es gibt keinen f-augmentierenden s,t-Weg.

Beweis. $\Rightarrow$: klar, da ein f-augmentierender s,t-Weg den Flusswert erhöht.
$\Leftarrow$: Definiere

$$R := \{v \in V(G) | \text{ es gibt gerichteten } s, v - \text{Weg in } G_f\}.$$

Es folgt direkt $s \in R, t \notin R$, das heisst das R ein s,t-Schnitt ist.

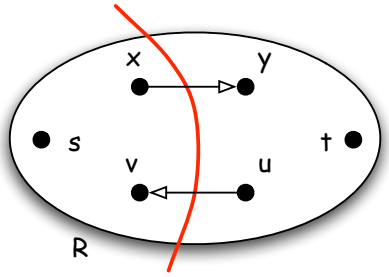


Abbildung 1.16: s,t-Schnitt R.

Für Kante $e = (x,y) \in \delta^+(R)$ in G gilt $f(e) = u(e)$, da sonst $y \in R$ gilt. Für Kante $e = (u,v) \in \delta^-(R)$ in G gilt $f(e) = 0$, da sonst $u \in R$ gilt.

Somit folgt

$$\begin{aligned} \nu(f) &= \sum_{e\in\delta^+(R)} f(e) - \sum_{e\in\delta^-(R)} f(e) \\ &= \sum_{e\in\delta^+(R)} f(e) \\ &= \sum_{e\in\delta^+(R)} u(e) \\ &= c(R), \end{aligned}$$

und mit Lemma 1.10 ($\nu(f) \leq c(R)$ für alle Flüsse f) folgt dass f optimal ist. □

Satz 1.14 (Max-Flow Min-Cut Satz, Ford-Fulkerson 1956, Elias, Feinstein, Shannon 1956). In (G, u, s, t) ist der maximale Flusswert $\nu(f)$ eines s,t-Flusses gleich der minimalen Kapazität eines s,t- Schnittes, d.h.

$$\max_{f \text{ s,t-Fluss}} \nu(f) = \min_{A \text{ s-t Schnitt}} c(A).$$

Wir erhalten eine wichtige Folgerung.

Korollar 1.15. Sind alle Kapazitäten ganzzahlig, so terminiert der Ford-Fulkerson Algorithmus nach $O(U \cdot m^2)$ Zeit mit einem ganzzahligen maximalen Fluss mit Flusswert $\nu \leq m \cdot U$, wobei $U := \max_{e\in E} u(e)$.

Beweis. In jedem Schritt wird der Flusswert um mindestens 1 erhöht. Die Kapazität eines minimalen Schnittes ist höchstens $m \cdot U$. □

1.5 Sätze vom Menger Typ

Satz 1.16 (Menger 1927, Kantenversion). Sei G gerichtet oder ungerichtet, seien s, t zwei verschiedene Knoten aus $V(G)$ und $k \in \mathbb{N}$. Dann sind äquivalent:

1. Es gibt k paarweise kantendisjunkte (gerichtete, falls G gerichtet) s,t-Wege.
2. t ist auch nach Weglassen von beliebigen $k-1$ vielen Kanten von s erreichbar.

Beweis. 1. $\Rightarrow$ 2.: klar.
2. $\Rightarrow$ 1.: 1. Fall: G sei gerichtet.
Betrachte das Fluss Netzwerk (G, u, s, t) mit $u(e) = 1$ für alle Kanten $e \in E(G)$. Mit der Vorraussetzung 2. folgt, dass jeder s,t-Schnitt A die Bedingung $c(A) \geq k$ erfüllt. Somit gibt es einen ganzzahligen s,t-Fluss mit Flusswert $\nu \geq k$. Es folgt, dass der (benutze $u(e) = 1$, Dekompositionssatz) Fluss in mindestens k paarweise kantendisjunkte gerichtete s,t-Wege (und ggf. Zykel) zerlegt werden kann; also gibt es k paarweise kantendisjunkte gerichtete s,t-Wege.

2. Fall: G sei ungerichtet.
Ersetze jede Kante e durch folgendes Konstrukt um die gerichtete Version anwenden zu können:

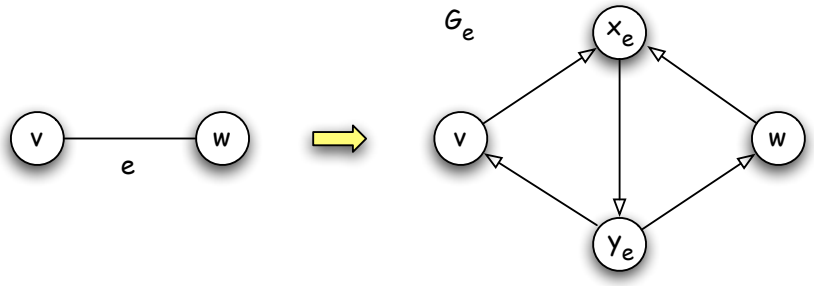


Abbildung 1.17: Illustration der Konstruktion.

Dies ergibt Digraph G'. Wir halten folgende Beobachtung fest:

es gibt in G_e jeweils genau einen Weg der von v nach w führt und umgekehrt. $(\star)$

Mit Voraussetzung 2. gilt, dass s und t in G nach Weglassen von $k-1$ Kanten weiterhin verbunden sind.
Behauptung: es gibt gerichteten s,t-Weg in G' nach Weglassen von beliebigen $k-1$ gerichteten Kanten aus G'.
Zum Beweis der Behauptung:

Betrachte die Konstrukte in G', in denen die weggelassenen Kanten liegen und entferne in G die zugrunde liegenden (ungerichteten) Kanten. Dies sind höchstens $k-1$ Stück.

$\Rightarrow$ in G existiert immer noch ein s,t-Weg W

$\Rightarrow$ in G' existieren für die Kanten aus W die zugehörigen Konstrukte

$\Rightarrow$ es gibt gerichteten s,t-Weg in G'

Behauptung mit Beweis von Fall 1 ergibt, dass in G' k paarweise kantendisjunkte gerichtete s,t-Wege existieren. Diese müssen wegen $(\star)$ in einem Konstrukt genau einen v,w-Weg belegen. Die zugehörigen Kanten e in G bilden ein System von k paarweise kantendisjunkten ungerichteten s,t-Wegen. □

Bemerkung 1.17. Umgekehrt folgt aus dem Satz von Menger der Max-Flow Min-Cut Satz für rationale Kapazitäten (wir haben ihn bereits für reellwertige Kapazitäten bewiesen).

Satz 1.18 (Menger 1927, Knotenversion). Sei G gerichtet oder ungerichtet, seien s,t zwei verschiedene Knoten aus $V(G)$ und $k \in \mathbb{N}$. Weiter sei die Kante $s \to t$ (bzw. $s - t$ falls G ungerichtet) nicht in G. Dann sind äquivalent:

1. Es gibt k paarweise knotendisjunkte (gerichtete, falls G gerichtet) s,t-Wege.

2. t ist auch nach Weglassen von beliebigen (von s und t verschiedenen) $k-1$ vielen Knoten von s erreichbar.

Beweis. 1. $\Rightarrow$ 2. ist wieder klar; 2. $\Rightarrow$ 1.: 1. Fall: G ist gerichtet.
Transformiere G wie folgt (für $v \neq s, t$) um die Kantenversion von Menger anwenden zu können – dies ergibt Graph G'.

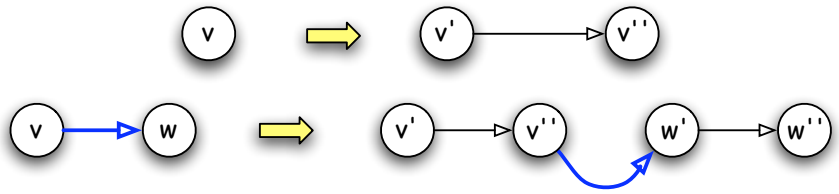


Abbildung 1.18: Illustration der Konstruktion.

Behauptung: G erfüllt 2. bzgl. Knotenversion $\Rightarrow$ G' erfüllt 2. bzgl. Kantenversion.

Der Rest des Beweises findet in einer Übungsaufgabe statt.
2. Fall: G ist ungerichtet: Rückführung auf gerichtete Kantenversion durch ähnliche Transformation, aber ersetze Kanten durch das Konstrukt aus dem Beweis von Satz 1.16. □

Bemerkung 1.19. Interpretation der Menger-Sätze in Straßennetzen: um s und t zu trennen, muss man soviele Straßen/Kreuzungen sperren wie es paarweise kantendisjunkte/knotendisjunkte s, t-Wege gibt.

1.6 Algorithmus von Edmonds und Karp

Der Edmonds-Karp Algorithmus spezifiziert die Wahl der f-augmentierenden Wege im Ford-Fulkerson Algorithmus:

- Wähle kürzesten f-augmentierenden Weg bzgl. Anzahl der Kanten
- erreichbar durch BFS im Residualgraphen G_f.

Satz 1.20 (Edmonds-Karp 1972). Der Edmonds-Karp Algorithmus benötigt maximal $m \cdot n/2$ augmentierende Wege (bei beliebigen Kapazitäten $u(e)$), ermittelt also einen maximalen Fluss in $O(m^2 n)$ Zeit.

Beweisidee. Engpasskante in einem f-augmentierenden Weg entspricht Kante e mit kleinster Residualkapazität, d.h. $\gamma = u_f(e)$.
Behauptung: eine Kante $e \in E(G^{\leftrightarrow})$ kann höchstens $n/4$-mal als Engpasskante auftreten. Daraus folgt: maximal $|E(G^{\leftrightarrow})| \cdot n/4 = 2m \cdot n/4 = m \cdot n/2$ augmentierende Wege. Die Suche eines Weges mit BFS in G_f geht in $O(m)$ Zeit. Somit erhalten wir eine Laufzeit von $O(m^2 n)$ insgesamt. □

Der Beweis der Behauptung ist nichttrivial.

Lemma 1.21. Folgende Voraussetzungen seien erfüllt:

- Sei $f_1, f_2, \ldots$ eine Folge von s,t-Flüssen in G und sei G_{f_k} der zu f_k gehörige Residualgraph.
- W_k sei kürzester s,t-Weg bzgl. Kantenzahl in G_{f_k}.
- f_{k+1} entstehe aus f_k durch Augmentierung entlang Weg W_k.

Dann gilt:

1. Die Länge der Wege steigt schwach monoton, d.h. $|E(W_k)| \leq |E(W_{k+1})|$ (schwache Monotonie).
2. $|E(W_k)| + 2 \leq |E(W_r)|$ für alle $k < r$ so dass $W_k \cup W_r$ ein Paar entgegengesetzter Kanten $e, e^{\leftarrow}$ enthält (starke Monotonie).

Beweis. Zu 1.:
Betrachte

$$G_k := (W_k + W_{k+1}) - \text{ entgegengesetzte Kanten}$$

($+$ im Sinne einer Linearkombination, d.h. Kanten können doppelt vorkommen). Wir geben ein Beispiel.

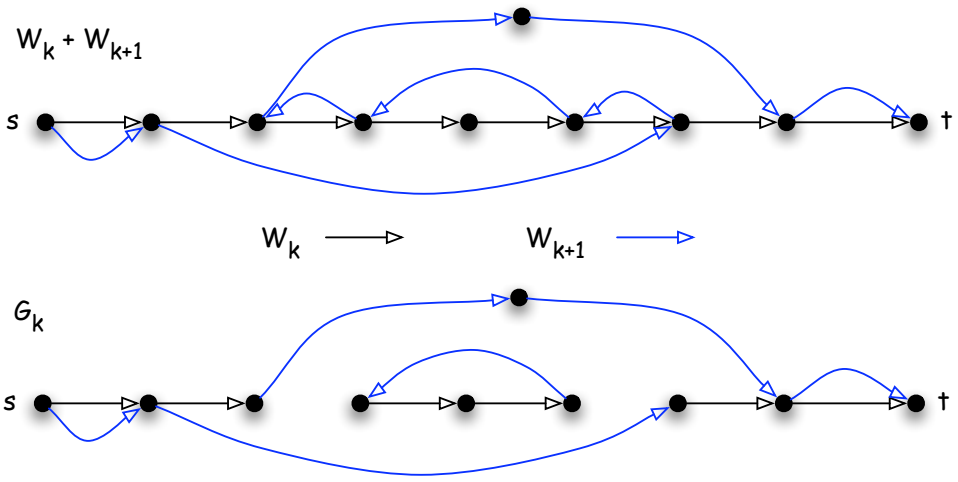


Abbildung 1.19: Beispielhafte Situation für $W_k + W_{k+1}$.

Wir zeigen nun mehrere Eigenschaften.
(a) $E(G_k) \subseteq E(G_{f_k})$ (hier ist nur der einfache Graph ohne "doppelte Kanten" gemeint):
Es gilt, dass W_k Teilgraph von G_{f_k} ist, und W_{k+1} ist Teilgraph von $G_{f_{k+1}}$. Angenommen, es gibt eine Kante $e \in E(G_k) \setminus E(G_{f_k})$. Wegen der Definition von G_k folgt, dass $e \in$

$E(W_{k+1}) \setminus E(G_{f_k})$. Es folgt, dass e neu dazu kommt, also ist e entgegengesetzte Kante einer Kante $e' \in W_k \subseteq E(G_{f_k})$. Solche Paare entgegengesetzter Kanten wurden aber in G_k gelöscht, Widerspruch.
(b) G_k enthält 2 kantendisjunkte s,t-Wege P_1, P_2, denn $G'_k := G_k + 2$ mal Kante (t,s) ist Eulersch im gerichteten Sinne.
Wdh. aus Optimierung II (Kapitel 10):

Definition 1.22. Ein Digraph G ist Eulersch :$\Leftrightarrow$ für jeden Knoten v ist $|\delta^+(v)| = |\delta^-(v)|$.

Korollar 1.23 (Korollar aus Satz bzgl. Euler-Algorithmus). Ein Eulerscher (gerichteter) Graph ist kantendisjunkte Vereinigung von (gerichteten) elementaren Kreisen.

Mit dem Korollar folgt, dass G'_k kantendisjunkte Vereinigung von (gerichteten) elementaren Kreisen ist. Somit enthält für jede der beiden zugefügten Kanten (t,s) der Graph G_k einen s,t-Weg $P_i, i = 1,2$, und diese Wege sind kantendisjunkt in G_k.
Mit (a) folgt, dass P_1, P_2 f_k-augmentierend sind. Die Wahl von W_k als kürzester f_k-augmentierender Weg impliziert

$$\begin{aligned} & |E(W_k)| \leq |E(P_1)|, |E(W_k)| \leq |E(P_2)| \\ \Rightarrow\ & 2|E(W_k)| \leq |E(P_1)| + |E(P_2)| \leq |E(G_k)| \leq |E(W_k)| + |E(W_{k+1})| \\ \Rightarrow\ & |E(W_k)| \leq |E(W_{k+1})| \end{aligned}$$

Zu 2.:

Betrachte k, r so, dass $W_k + W_r$ ein Paar entgegengesetzter Kanten enthält, aber $W_i + W_r$ für $k < i < r$ kein Paar entgegengesetzter Kanten enthält (wegen 1., also Monotonie der Zwischenwerte, reicht es 2 für solche Paare zu zeigen). Betrachte wieder $G_k := (W_k + W_r) -$ entgegengesetzte Kanten.
(c) $E(G_k) \subseteq E(G_{f_k})$.
Angenommen, es gibt eine Kante $e \in E(G_k) \setminus E(G_{f_k})$. Da W_k ein Teilgraph von G_k ist, muss wegen der Definition von G_k gelten: $e \in E(W_r) \setminus E(G_{f_k})$. Somit kommt e neu dazu, also ist e entgegengesetzte Kante einer Kante e' in W_j für ein $j = k, k+1, \ldots, r-1$. Mit der Wahl von k, r folgt dass $e' \in E(W_k)$, somit fallen e', e in G_k raus, Widerspruch.
Entsprechend zu (b): G_k enthält 2 kantendisjunkte s,t-Wege P_1, P_2.
(c) impliziert, dass P_1, P_2 jeweils f_k-augmentierend sind. Somit folgt:

$$\begin{aligned} 2|E(W_k)| \leq |E(P_1)| + |E(P_2)| &\leq |E(G_k)| \\ &\leq |E(W_k)| + |E(W_r)| - 2, \end{aligned}$$

da mindestens ein Paar Kanten entfernt wurde. Es folgt: $|E(W_k)| + 2 \leq |E(W_r)|$. □

Beweis der Behauptung mit Lemma 1.21. Betrachte Kante e. Sei $W_{i_1}, W_{i_2}, \ldots$ die Sequenz der Wege im Algorithmus, die e als Engpasskante enthalten. Betrachte zwei aufeinanderfolgende Wege $W_{i_j}, W_{i_{j+1}}$ der Sequenz. Weil e eine Engpasskante ist, gilt dass

die umgekehrte Kante e' zu e in $G_{f_{i_{j+1}}}$ enthalten ist, e jedoch nicht. Es folgt mit der Eigenschaft, dass e wieder in $W_{i_{j+1}}$ vorkommt, dass es einen augmentierenden Weg W_k mit $i_j < k < i_{j+1}$ gibt, der e' enthält, so dass e wieder in $G_{f_{i_{j+1}}}$ auftritt. Aus Lemma 1.21 folgt:

$$|E(W_{i_j})| + 4 \leq |E(W_k)| + 2 \leq |E(W_{i_{j+1}})| \leq n - 1,$$

da alle Wege elementar sind. Somit gibt es höchstens $n/4$ Pfade in der Sequenz, die e als Engpasskante enthalten. □

1.7 Algorithmus von Dinic

Edmonds-Karp zeigt: kürzeste augmentierende Wege sind günstig. Konsequente Ausnutzung: betrachte alle kürzesten augmentierenden Wege gleichzeitig.

Definition 1.24. Ein Schichtgraph (Levelgraph) LG_f ist definiert als der Graph aller kürzesten (bzgl. Kantenzahl) gerichteten s,t-Wege im Residualgraph G_f. Ein Blockierender s,t-Fluss in LG_f ist ein s,t-Fluss f', der auf jedem s,t-Weg in LG_f eine Kante sättigt, d.h. $f'(e) = u_f(e)$.

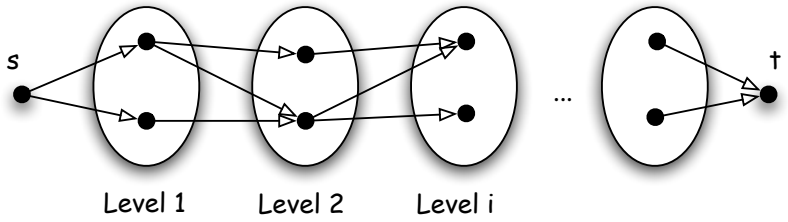


Abbildung 1.20: Level i = Menge aller Knoten mit Abstand i von s. LG_f ist azyklisch und Kanten existieren nur zwischen benachbarten Leveln.

Bemerkung 1.25. Blockierende Flüsse sind i.a. nicht maximal, siehe folgendes Beispiel. Jedoch sind sie im Schichtgraphen einfach zu ermitteln in Zeit $O(n \cdot m)$ (siehe Alg. 1.7.2).

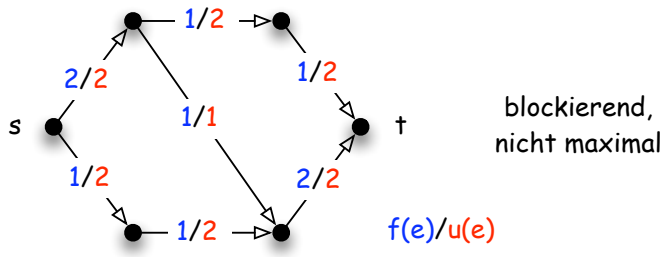


Abbildung 1.21: Blockierender Fluss der nicht maximal ist.

Blockierende Flüsse sind die Grundlage des folgenden Algorithmus von Dinic.

Eingabe: Flussnetzwerk (G, u, s, t).
Ausgabe: Maximaler s,t-Fluss f.
Methode:
$f(e) := 0$ für alle Kanten $e \in E(G)$
while es gibt f-augmentierenden s,t-Weg W **do**
  konstruiere blockierenden Fluss f' in LG_f.
  augmentiere f um f'.
end
return: f

Algorithm 1.7.1: Dinic Algorithmus

Bei obigem Algorithmus meint natürlich "augmentiere f um f'", dass wir folgende Werte setzen:

$$f(e) := f(e) + f'(e) \text{ für alle } e \in E(LG_f) \cap E(G)$$
$$f(e) := f(e) - f'(e^{\leftarrow}) \text{ für alle } e \in E(G) \text{ mit } e^{\leftarrow} \in E(LG_f).$$

Satz 1.26. Der Algorithmus von Dinic terminiert nach maximal $n-1$ Iterationen und gibt einen maximalen Fluss aus.

Beweis. Wenn der Algorithmus terminiert, existiert kein f-augmentierender Weg mehr. Mit Satz 1.13 ist der Algorithmus also korrekt.
Nun zeigen wir die Laufzeitschranke. Sei f der Fluss vor einer Iteration der Hauptschleife und $\bar{f}$ der augmentierte Fluss nach der Iteration. Sei $\mathrm{dist}_{G_f}(s,t)$ die Länge eines kürzesten Weges von s zu t in LG_f bzgl. Gewichtsfunktion $c(e) = 1$ für alle $e \in E(G_f)$ (also Anzahl der Kanten).
Sei W ein kürzester Weg in $G_{\bar{f}}$. Wenn W ein zulässiger gerichteter s,t-Weg in G_f ist, folgt (unter Verwendung des blockierenden Flusses), dass $|E(W)| \geq \mathrm{dist}_{G_f}(s,t) + 1$. Somit können wir annehmen, dass W kein gerichteter s,t-Weg in G_f ist. Da in $G_{\bar{f}}$ aber nur Rückwärtskanten (für Kanten in LG_f) eingeführt werden, folgt, dass W solche Rückwärtskanten enthalten muss. Somit folgt ebenfalls $|E(W)| \geq \mathrm{dist}_{G_f}(s,t) + 1$.
Insgesamt sind die auftretenden Weglängen für einfache s,t-Wege durch $n-1$ beschränkt. □

Nun zeigen wir noch, wie man einen blockierenden Fluss berechnet:

Eingabe: Levelgraph (LG_f, u_f, s, t).
Ausgabe: Blockierender s,t-Fluss f'.
Methode:
$f'(e) := 0$ für alle Kanten $e \in E(LG_f)$.
while es existiert ein Weg von s nach t **do**
  finde einen gerichteten Weg W von s nach t durch Tiefensuche (DFS) – Lösche dabei solche Kanten (u,v), bei denen $v \neq t$ und v hat keine ausgehenden Kanten.
  erhöhe den Fluss f' entlang W um soviel wie möglich
  entferne saturierte Kanten
end
return: f'

Algorithm 1.7.2: Blockierender Fluss

Lemma 1.27. Der Algorithmus konstruiert einen blockierenden Fluss in $O(n \cdot m)$ Zeit.

Beweis. Bei der Ausführung der Tiefensuchen besuchen wir iterativ Knoten $v \in V(LG_f)$ bis $v = t$ gilt und wir einen s,t-Weg W erhalten (entlang dessen wir maximalen Fluss schicken und dann saturierte Kanten löschen). Dabei können auch Kanten (v,w) durchlaufen werden, die in der Tiefensuche wieder "zurückgelaufen" werden, weil t nicht über v erreichbar ist. Wir zählen nun folgende Events bei der Ausführung des Algorithmus:

- $v = t$ und Flusserhöhung (increase)
- $v \neq t$ aber es gibt unbesuchte Kante (v,w) (explore)
- $v \neq t$ aber es gibt keine ausgehende Kante von v und wir löschen die letzte durchgelaufene Kante (u,v) (retreat)

Wir zählen nun die Anzahl der increase-, explore- und retreat-Schritte.

- # increase: maximal m, weil pro increase eine Kante gelöscht wird.
- # retreat: maximal m, weil pro retreat eine Kante gelöscht wird.
- # explore $\leq$ # retreat$+n \cdot$ # increase, weil jedes explore entweder durch ein retreat rückgängig gemacht wird, oder zu einem s-t Weg W führt und pro Weg höchstens n mal vorkommt.

Wir erhalten also $O(m + n \cdot m) = O(n \cdot m)$. □

Insgesamt erhalten wir:

Satz 1.28. Der Algorithmus von Dinic kann in Zeit $O(n^2 \cdot m)$ implementiert werden.

Es gibt sogar bessere Algorithmen zur Berechnung von blockierenden Flüssen in Zeit $O(n^2)$, so dass Dinic in $O(n^3)$ läuft.

1.8 Push-Relabel Algorithmen

Erinnerung: eine Kantenbewertung $f(e), e \in E(G)$, ist ein maximaler s,t-Fluss bzgl. Kapazitäten $u(e)$, wenn gilt

1. Kapazitätseinhaltung: $0 \leq f(e) \leq u(e)$ für alle Kanten e
2. Flusserhaltung: $\sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)} f(e)$ für alle $v \neq s,t$
3. Maximalität: es gibt keinen f-augmentierenden Weg.

Augmentierende-Wege-Algorithmen erfüllen (1) und (2), und stellen (3) am Ende her. Push-Relabel Algorithmen verwenden nicht augmentierende Wege (macht sie flexibler) erfüllen (1) und (3) und stellen (2) am Ende her. Dazu benötigen wir eine Abschwächung der s,t-Fluss Definition in Punkt (2).

Definition 1.29 (Präfluss). Ein s,t-Präfluss in (G,u,s,t) ist eine Kantenbewertung $f(e), e \in E(G)$, mit

$$0 \leq f(e) \leq u(e) \text{ für alle Kanten } e$$
$$ex_f(v) := \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e) \geq 0 \text{ für alle } v \neq s,t.$$

Der Wert $ex_f(v)$ heisst Exzess in v. Ein Knoten v heisst aktiv, falls $ex_f(v) > 0$ (mehr Einfluss als Ausfluss) und $v \neq s,t$. Ein s,t-Präfluss ist ein s,t-Fluss $\Leftrightarrow$ es gibt keine aktiven Knoten.

Als nächstes führen wir Distanzbewertungen ein, die zusätzlich zu s,t-Präflüssen in Push-Relabel Algorithmen genutzt werden.

Definition 1.30 (Distanzbewertung bzgl. s,t-Präfluss). Sei f ein s,t-Präfluss in (G,u,s,t). Eine Distanzbewertung ist eine Knotenbewertung $\Psi(v), v \in V(G)$, mit

$$\Psi(s) = n, \quad \Psi(t) = 0,$$
$$\Psi(v) \leq \Psi(w) + 1 \text{ für alle Kanten } (v,w) \text{ im Residualgraphen zu } f, \text{ d.h. } (v,w) \in E(G_f).$$

Dabei ist der Residualgraph zu einem s,t-Präfluss genauso definiert wie für einen s,t-Fluss (alle Kanten von $G^{\leftrightarrow}$ mit positiver Residualkapazität).

Wir sagen, dass eine Kante $e = (v,w) \in E(G^{\leftrightarrow})$ zulässig ist, falls $e \in E(G_f)$ und $\Psi(v) = \Psi(w) + 1$ gilt.
Unter obigen Voraussetzungen folgt, dass $\Psi(v)$ eine untere Schranke an die kürzeste Weglänge von v nach t bzgl. Anzahl der Kanten darstellt.

Lemma 1.31. Sei Ψ eine Distanzbewertung und $v \neq s$. Dann gilt, dass $\Psi(v) \leq$ Länge kürzester Weg (bzgl. Kantenzahl) von v nach t in G_f.

Beweis. Sei $dist(v,t) :=$ Länge kürzester Weg (bzgl. Kantenzahl) von v nach t in G_f. Wir führen eine Induktion bzgl. $dist(v,t)$ durch. Behauptung ist richtig für $v = t$ und für Kanten (v,t) nach Definition von Ψ (Kantenzahl 0 und 1 auf kürzestem Weg). Schluss von Kantenzahl $k-1$ auf k auf kürzestem Weg: sei (v,w) erste Kante auf kürzestem v,t-Weg in G_f, und dieser habe k Kanten.

$$\begin{aligned} \Rightarrow \Psi(v) &\leq \Psi(w) + 1 \text{ (Definition von } \Psi) \\ &\leq dist(w,t) + 1 \text{ (nach Induktionsvoraussetzung)} \\ &= dist(v,t) \end{aligned}$$

Also: Distanzbewertungen (für $v \neq s$) sind untere Schranken für den Abstand von v zu t in G_f. □

Push-Relabel Algorithmen arbeiten mit einem s,t-Präfluss f und Distanzbewertung Ψ und führen in beliebiger Reihenfolge Push Operationen oder Relabel Operationen aus.

- bei Push Operation wird der Präfluss f aktualisiert
- bei Relabel Operation wird die Distanzbewertung Ψ aktualisiert

Genauer: Sei v aktiver Knoten.
push(e) :
Sei $\gamma := \min\{ex_f(v), u_f(v,w)\}$ mit $e = (v,w)$. Augmentiere f auf e um γ (zur Erinnerung: erhöhen auf VK, erniedrigen auf RK).
relabel(v) :
setze $\Psi(v) := \min\{\Psi(w) + 1 | (v,w) \in E(G_f)\}$.

Bemerkung 1.32.
- push(e) erhält Präfluss-Bedingungen, erniedrigt Exzess in v um γ, erhöht ihn in w um γ.
- relabel(v) erhält die Bedingungen an Distanzbewertungen, $\Psi(v)$ wird nicht erniedrigt, aber möglicherweise erhöht.

Eingabe: Netzwerk (G,u,s,t).
Ausgabe: Maximaler s,t-Fluss f.
Methode:
Initialisierung des s,t-Präfluss f
$f(e) := u(e)$ für alle von s ausgehenden Kanten $e = (s,v) \in E(G)$; $f(e) := 0$, sonst
Initialisierung der Distanzbewertung Ψ
$\Psi(s) := n; \Psi(v) := 0$, sonst
while es gibt aktive Knoten **do**
  wähle aktiven Knoten v
  if keine Kante $e = (v,w)$ ist zulässig **then**
    relabel(v)
  end
  else
    wähle zulässige Kante $e = (v,w)$
    push(e)
  end
end
return: f

Algorithm 1.8.1: Generischer Push-Relabel Algorithmus

Proposition 1.33. Die im Push-Relabel Algorithmus konstruierten Bewertungen f und Ψ sind jeweils s,t-Präflüsse bzw. Distanzbewertungen.

Beweis. Einziger offener Punkt: bleibt Ψ Distanzbewertung nach Push Operation? push(e) mit $e = (v,w) \Rightarrow \Psi(v) = \Psi(w) + 1$. Nach push$(e)$ kommt Kante (w,v) evtl. zu G_f hinzu, aber $\Psi(w) = \Psi(v) - 1$, also gilt für Kante (w,v), dass $\Psi(w) \leq \Psi(v) + 1$. □

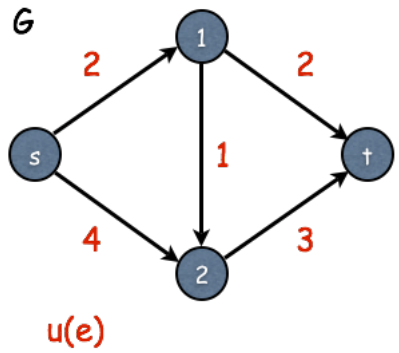


Abbildung 1.22: Wir führen den generischen Push-Relabel Algorithmus an folgendem Beispiel aus – Details dazu in der Vorlesung.

Lemma 1.34. Sei f ein s,t-Präfluss und Ψ eine Distanzbewertung bzgl. f. Dann gilt:

(a) s ist in G_f von jedem aktiven Knoten v erreichbar (auf gerichtetem Weg).

(b) t ist in G_f nicht von s erreichbar (auf gerichtetem Weg).

Beweis. Zu (a):
Sei v aktiv, R die Menge der von v aus erreichbaren Knoten in G_f. Dann gilt $f(e) = 0$ für alle Kanten e, die in G (nicht in G_f !) in R reingehen. Die Summe der Exzesse der Knoten in R ist ≤ 0, denn es gilt

$$\begin{aligned}\sum_{w\in R} ex_f(w) &= \sum_{w\in R}\left(\sum_{e\in\delta_G^-(w)} f(e) - \sum_{e\in\delta_G^+(w)} f(e)\right)\\ &= \sum_{e\in\delta_G^-(R)} f(e) - \sum_{e\in\delta_G^+(R)} f(e) \qquad (1.2)\\ &= 0 - \sum \text{Summe von nicht-negativen (Präflusswerten) } f(e)\\ &\leq 0,\end{aligned}$$

wobei wir für (1.2) ausgenutzt haben, dass sich die innere Differenz weghebt für Kanten e mit beiden Endpunkten in R. Nach Annahme ist v aktiv und $v \in R$, also $ex_f(v) > 0$. Da wir jedoch $\sum_{w\in R} ex_f(w) \leq 0$ gezeigt haben, muss es einen Knoten $w \in R$ mit $ex_f(w) < 0$ geben. Das ist nur möglich bei $w = s$, da für alle $u \neq s,t$ (Definition Präfluss) $ex_f(u) \geq 0$ gilt. Somit folgt $s \in R$.

Zu (b):
Angenommen es gibt s,t-Weg $s = v_0, v_1, \ldots, v_k = t$ (o.E. elementar). Mit Distanzbewertung Ψ gilt: $\Psi(v_i) \leq \Psi(v_{i+1}) + 1 \Rightarrow \Psi(s) \leq \Psi(t) + k = k$ (da $\Psi(t) = 0$). Also folgt $\Psi(s) \leq k \leq n-1$, da der Weg elementar ist. Das ergibt Widerspruch zu $\Psi(s) = n$. □

Satz 1.35. Wenn der Push-Relabel Algorithmus terminiert, so ist f ein maximaler s,t-Fluss.

Beweis. f ist Fluss, da bei Termination kein Knoten mehr aktiv ist. f ist maximal, da es nach Lemma 1.34(b) keinen Weg von s nach t in G_f, also keinen f-augmentierenden Weg gibt. □

Frage: Terminiert der Push-Relabel Algorithmus?
Antwort: ja, bei beliebiger Wahl der Schritte in $O(n^2m)$.
Bei spezieller Wahl (Goldberg-Tarjan Algorithmus) in $O(n^3)$ bzw. $O(n^2m^{1/2})$.

Lemma 1.36. (a) relabel(v) erhöht $\Psi(v)$ um mindestens 1, $\Psi(v)$ wird nie erniedrigt.

(b) $\Psi(v) \leq 2n-1$ im gesamten Algorithmus.

(c) Es werden maximal $2n^2$ Relabel Operationen durchgeführt.

Beweis. Zu (a):
Nur relabel(v) ändert $\Psi(v)$. Ein solcher Schritt wird nur durchgeführt, wenn es keine zulässige Kante (v,w) gibt. Mit $\Psi(v) < \Psi(w)+1$ wird $\Psi(v)$ echt erhöht.
Zu (b):
$\Psi(v)$ wird nur geändert wenn v aktiv ist. Mit Lemma 1.34 folgt, dass es einen gerichteten (o.B.d.A. elementaren) Weg von v nach s in G_f gibt, etwa $v = v_0, v_1, \ldots, v_k = s$. Mit Distanzbewertung Ψ gilt

$$\begin{aligned} & \Psi(v_i) \leq \Psi(v_{i+1}) + 1 \\ \Rightarrow\ & \Psi(v) \leq \Psi(s) + k = n + k, \ (\text{da } \Psi(s) = n) \\ & \leq n + n - 1 = 2n-1, \ \text{da der Weg elementar.} \end{aligned}$$

Zu (c):
Folgt direkt aus (a) und (b) und der Beobachtung, dass insgesamt $|\{0,1,\ldots,2n-1\}| = 2n$ Potentialwerte pro Knoten auftreten können. □

Um die Anzahl der Push Schritte abschätzen zu können, unterscheiden wir zwischen saturierenden Push Schritten (die Kante wird saturiert, d.h. $\gamma = u_f(e)$ bei dem Schritt) und nicht-saturierenden Push Schritten ($\gamma < u_f(e)$ bei dem Schritt, d.h. $\gamma = ex_f(v)$ für $e = (v,w)$).

Lemma 1.37. Die Anzahl der saturierenden Push Schritte ist maximal $2 \cdot m \cdot n$.

Beweis. push(e) saturiere die Kante $e = (v,w)$. Somit wird danach e aus G_f gelöscht und ggf. wird die Kante (w,v) eingeführt. Bei push(e) war e zulässig, d.h. $\Psi(v) = \Psi(w)+1$. Somit muss vor einem weiteren saturierenden Push Schritt mit Kante e folgendes geschehen:

1. $\Psi(w)$ muss um (mind.) 2 erhöht werden (um (w,v) zulässig zu machen)

2. push(w, v) muss durchgeführt werden (um (v, w) wieder in G_f einzuführen)

3. $\Psi(v)$ muss um (mind.) 2 erhöht werden (um (v, w) zulässig zu machen)

Dies ergibt, dass für 2 saturierende push(e)-Schritte $\Psi(v)$ und $\Psi(w)$ um 2 wachsen müssen. Da außerdem $\Psi(v) \geq 1$ beim ersten saturierenden push(e)-Schritt gilt, sowie $\Psi(v) \leq 2n-1$ nach Lemma 1.36, kann es insgesamt maximal n saturierende push(e)-Schritte geben. Somit erhalten wir maximal $2 \cdot m \cdot n$ saturierende Push Schritte ($|E(G^{\leftrightarrow})| = 2m$). □

Nun schätzen wir die Anzahl der nicht-saturierenden Push Schritte ab.

Lemma 1.38. Die Anzahl der nicht-saturierenden Push Schritte ist beschränkt durch $O(n^2 \cdot m)$.

Beweis. Wir benutzen eine Potentialfunktion für den Beweis. Sei $A \subseteq V \setminus \{s, t\}$ die Menge der aktiven Koten zu einem beliebigen Zeitpunkt während der Ausführung des Algorithmus.
Wir definieren das Potential zu diesem Zeitpunkt als

$$\Phi(A) = \sum_{v \in A} \Psi(v).$$

Wir erhalten als Invariante, dass $\Phi \geq 0$ und vor der ersten Hauptiteration $\Phi = 0$.
Ein nicht-saturierender Push entlang (v, w) erniedrigt Φ um 1: es gilt $\Psi(v) = \Psi(w) + 1$ und v wird inaktiv (also $-\Psi(v)$), aber w wird ggf. aktiv (also $+\Psi(w) = \Psi(v) - 1$).
Jeglicher Zuwachs von Φ kann also nur von saturierenden Push-Schritten oder relabel Schritten verursacht sein. Ein saturierender Push kann das Potential um maximal $2n - 1$ erhöhen (Lemma 1.36) und es gibt maximal $2 \cdot n \cdot m$ viele solcher sat. Push Schritte (Lemma 1.37). Also ist die Summe der Zuwächse $O((2n - 1) \cdot 2 \cdot n \cdot m) = O(n^2 \cdot m)$.
Relabel Operationen können nur bei aktiven Knoten durchgeführt werden. Es gibt maximal $n - 2$ aktive Knoten und jedes Label kann maximal zu $2n - 1$ ansteigen. Insgesamt ist die Summe der Zuwächse also $O(n^2)$.
Insgesamt erhalten wir, dass die Summe aller Erhöhungen des Potentials Φ durch $O(n^2 \cdot m)$ beschränkt ist und jeder nicht-saturierende Push Φ um eins erniedrigt. Somit gibt es nur $O(n^2 \cdot m)$ viele solcher Schritte.

□

Zusammengefasst erhalten wir:

Satz 1.39. Der generische Push-Relabel Algorithmus terminiert nach $O(n^2 \cdot m)$ Push und Relabel Operationen.

Beweis.
- # saturierende Push-Schritte: $O(n \cdot m)$
- # nicht-saturierende Push-Schritte: $O(n^2 \cdot m)$

- # Labeling-Schritte: $O(n^2)$

□

Die genaue Laufzeit hängt von der Implementierung der Push und Relabel Operationen ab. Man kann zeigen, dass die Laufzeit $O(n^2 \cdot m)$ ist.
Bei spezieller Wahl der Push und Relabel Operationen kann man sogar eine bessere Laufzeit erreichen. Die beiden folgenden Varianten gehen auf Goldberg und Tarjan zurück, weshalb sie auch als Goldberg-Tarjan Algorithmus bezeichnet werden. Bei beiden Varianten wird der aktive Knoten v nur dann gewechselt, wenn er entweder nicht mehr aktiv ist, oder kein push mehr möglich und somit relabel(v) durchgeführt wurde. Die Varianten unterscheiden sich darin, wie bei einem Wechsel des aktiven Knotens der nächste aktive Knoten ausgewählt wird. Bei der *FIFO*-Variante wird immer gemäß der *first in first out* Regel ausgewählt; bei der *Highest-Label* Variante immer ein aktiver Knoten mit der größten Distanzbewertung.

Satz 1.40 (Goldberg & Tarjan 1988, Cheriyan & Maheshwari 1989). Der Goldberg-Tarjan Algorithmus bestimmt einen maximalen s,t-Fluss in $O(n^3)$ (FIFO) bzw. $O(n^2 m^{1/2})$ (Highest Label).

Beweis. Ein vollständiger Beweis findet sich z.B. in D. Jungnickel (Graphs, Networks and Algorithms, ab Seite 198). □

Kapitel 2

Kostenminimale Flüsse

2.1 Problemformulierung und Spezialfälle

Bei kostenminimalen Flüssen betrachten wir nun Kantenkosten $c(e)$ für die Benutzung einer Kante e.

- Fluss $f(e)$ auf Kante e kostet $f(e) \cdot c(e)$
- Gesamtfluss f kostet $c(f) := \sum_{e \in E(G)} f(e) \cdot c(e)$

Wir werden Flüsse in diesem Abschnitt etwas allgemeiner fassen, indem wir beliebig viele Quellen und Senken erlauben. Desweiteren werden wir den Flussbegriff verallgemeinern:

Definition 2.1 (b-Flüsse). Sei G ein Digraph mit Kantenkapazitäten $u(e)$. Sei $b(v), v \in V(G)$, eine Knotenbewertung mit $\sum_{v \in V(G)} b(v) = 0$ ($b(v)$ heisst Balance von v). Ein b-Fluss in (G, u) ist eine Kantenbewertung $f(e), e \in E(G)$, mit

- $0 \leq f(e) \leq u(e)$ für alle Kanten e
- Ausfluss aus v - Einfluss in $v = b(v)$ für alle Knoten v

$$\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v) \text{ für alle } v \in V(G).$$

- Knoten v heisst Quelle oder Angebotsknoten, falls $b(v) > 0$, $b(v)$ heisst dann das Angebot (supply) in v.
- Senke oder Nachfrageknoten, falls $b(v) < 0, |b(v)|$ heisst dann die Nachfrage (demand) in v.
- Durchflussknoten, falls $b(v) = 0$.

b-Flüsse modellieren Transportaufgaben für ein Gut. Das Angebot soll zur Nachfrage transportiert werden. Daher auch $\sum_{v \in V(G)} b(v) = 0$.
Frage: Existiert ein b-Fluss in (G, u) und können wir dies effizient entscheiden?
Überprüfung mit Max-Fluss Algorithmus mittels folgender Transformation: $(G, u, b) \rightarrow (G', u, s, t)$

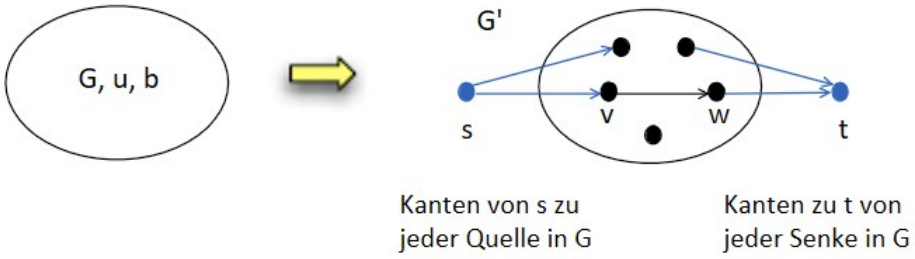


Abbildung 2.1: Transformation auf Max-Flow.

Wir definieren: $u(s,v) := b(v),\ u(v,t) := -b(v)$.

Lemma 2.2. Es gibt genau dann einen b-Fluss in (G,u), wenn gilt:

$$\text{maximaler Flusswert in } (G',u,s,t) = \sum_{v\in V(G)|\ b(v)>0} b(v) \quad \text{(Gesamtangebot)}\ .$$

Beweis. $\Rightarrow$:
Sei f ein b-Fluss in (G,u). Definiere Fluss f' in G' durch

$$\begin{aligned} f'(s,v) &:= b(v) \\ f'(v,t) &:= -b(v) \\ f'(e) &:= f(e),\ \text{sonst} \end{aligned}$$

Nach Konstruktion ist f' s,t-Fluss mit Flusswert $\nu(f') = \sum_{v\in V(G)|\ b(v)>0} b(v)$.
$\Leftarrow$:
Beweis geht analog. □

Flüsse mit minimalen Kosten:

> Minimum Cost Flow Problem (MCFP)
> **Instanz:** Digraph G, Kapazitäten u, Balancen b, Kosten c
> **Aufgabe:** Finde b-Fluss f mit minimalen Kosten (oder entscheide, dass keiner existiert).

Transportanwendungen mit Gesamtnachfrage $\neq$ Gesamtangebot:

- Führe Zusatzknoten ein: fiktiver Anbieter v_0 falls $\sum b(v) < 0$.
- von v_0 Kante (v_0,v) zu allen Knoten $v \in V(G)$ mit großer Kapazität und hohen Kosten sowie Balance $b(v_0) := -\sum b(v)$
- fiktiver Abnehmer v_0, falls $\sum b(v) > 0$

- zu v_0 Kante (v, v_0) von allen Knoten $v \in V(G)$ mit großer Kapazität und hohen Kosten sowie Balance $b(v_0) := -\sum b(v)$
- alles was v_0 versendet bzw. bekommt wird nicht abgenommen bzw. nicht geliefert

Wir betrachten nun wichtige Spezialfälle.

Transshipment Problem
alle $u(e) = \infty$, sonst wie bei MCFP

Hitchcock Problem oder Transportproblem (Hitchcock 1941)
Spezialfall des Transshipment Problems mit G bipartit, $V(G) = A \dot{\cup} B$ und $b(v) \geq 0$ für alle $v \in A$, $b(v) \leq 0$ für alle $v \in B$, und $E(G) \subseteq A \times B$.

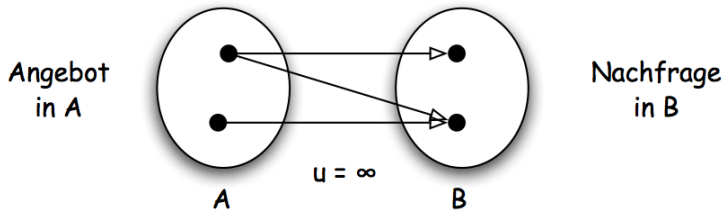


Abbildung 2.2: Hitchcock Problem.

Überraschenderweise gilt: Jedes zulässige MCFP lässt sich in ein (äquivalentes) Hitchcock-Problem transformieren!

Lemma 2.3. Eine zulässige Instanz des MCFP mit n Knoten und m Kanten kann in eine äquivalente Instanz des Hitchcock Problems mit $n + m$ Knoten und $2m$ Kanten transformiert werden.

Beweis. Sei (G, u, b, c) eine zulässige Instanz von MCFP. Definiere dazu Instanz (G', A', B', b', c') von Hitchcock wie folgt:

- Knoten in $A' \leftrightarrow$ Kanten in G, $b'(e) := u(e)$
- Knoten in $B' \leftrightarrow$ Knoten in G, $b'(v) := b(v) - \sum_{e \in \delta^+(v)} u(e)$ (≤ 0 da Instanz zulässig)
- Kanten in G' verlaufen
  - von Knoten $e = (v, w)$ zu Knoten v mit Kosten $c'(e, v) := 0$
  - von Knoten $e = (v, w)$ zu Knoten w mit Kosten $c'(e, w) := c(v, w)$

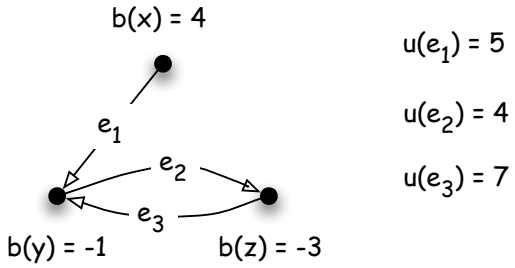


Abbildung 2.3: Beispielinstanz (G, u, b, c) von MCFP.

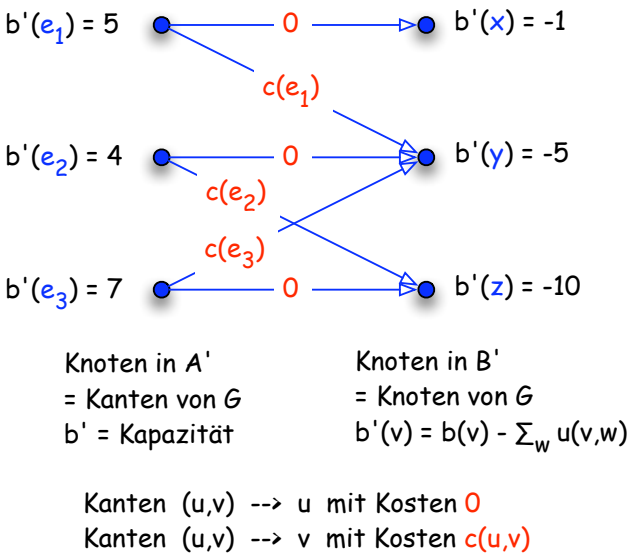


Abbildung 2.4: Transformation auf ein Hitchcock Problem.

Zeige die Äquivalenz:

es ex. b-Fluss f in (G, u, b, c) $\Leftrightarrow$ es ex. b'-Fluss f' in (G', A', B', b', c') mit denselben Kosten.
$\Rightarrow$: Sei f ein b-Fluss in (G, u) und betrachte Kante $e = (v, w)$. Setze $f'(e, w) := f(e)$ und $f'(e, v) := u(e) - f(e)$.
Zeige: f' ist b'-Fluss in (G', A', B', b', c'). Ausfluss - Einfluss für einen e-Knoten in A' mit $e = (v, w)$ ist

$$f'(e, w) + f'(e, v) - 0 = f(e) + u(e) - f(e) = u(e) = b'(e).$$

Ausfluss - Einfluss für einen Knoten v in B' ist

$$\begin{aligned} 0 - \left(\sum_{x:e=(x,v)} f(e) + \sum_{x:e=(v,x)} (u(e) - f(e)) \right) \\ = - \sum_{e \in \delta_G^-(v)} f(e) - \sum_{e \in \delta_G^+(v)} u(e) + \sum_{e \in \delta_G^+(v)} f(e) \\ = b(v) - \sum_{e \in \delta_G^+(v)} u(e) = b'(v). \end{aligned}$$

$c(f) = c'(f')$ klar, da Kanten (e, w) in G' mit Kosten $\neq 0$ gerade den Flusswert $f(e)$ und die Kosten $c(e)$ zugeordnet bekommen.
$\Leftarrow$: Nun zeigen wir die umgekehrte Richtung: b'-Fluss f' in $(G', A', B', b', c') \Rightarrow$ b-Fluss f in (G, u, b, c) mit denselben Kosten.
Sei f' ein b'-Fluss in (G', A', B', b', c'). Setze für Kante $e = (v, w)$ den Fluss $f(e) := f'(e, w)$. Analoge Rechnung ergibt f ist b-Fluss in (G, u, b, c) und $c(f) = c'(f')$. Für die Zahl der Knoten und Kanten in G' gilt

$$n' = m + n, \quad m' = 2m.$$

□

2.2 Optimalitätskriterien für kostenminimale Flüsse

Wir entwickeln nun Optimalitätskriterien für kostenminimale Flüsse, welche auf dem Residualgraph (wie bei maximalen Flüssen) basieren. Dazu werden Kosten $c(e)$ auch für Kanten $e \in E(G^{\leftrightarrow})$ folgendermassen definiert:

- $c(e) := c(e)$, falls $e \in E(G)$
- $c(e^{\leftarrow}) := -c(e)$, falls $e^{\leftarrow}$ Rückwärtskante zu $e \in E(G)$

Beachte: Kosten im Residualgraph G_f sind unabhängig vom b-Fluss f.

Definition 2.4. Ein f-augmentierender Zykel ist ein elementarer Zykel (also gerichteter Kreis) in G_f. Für $0 \leq \gamma \leq \min\{u_f(e) : e \in E(Z)\}$ ergibt die Augmentierung eines b-Flusses f entlang eines f-augmentierenden Zykels Z um den Wert γ wieder einen b-Fluss, denn: Augmentierung erhält Ausfluss-Einfluss in jedem Knoten.

Proposition 2.5. f und f' seien b-Flüsse in (G, u). Dann gelten folgende Eigenschaften:

a) $g : E(G^{\leftrightarrow}) \to \mathbb{R}_+$ mit

- $g(e) := \max\{0, f'(e) - f(e)\}$, falls $e \in E(G)$ Vorwärtskante (VK)
- $g(e^{\leftarrow}) := \max\{0, f(e) - f'(e)\}$, falls $e^{\leftarrow}$ Rückwärtskante (RK) zu $e \in E(G)$

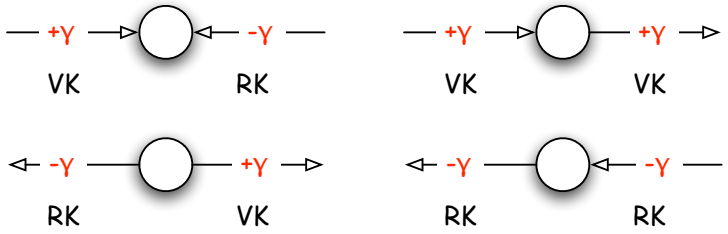


Abbildung 2.5: Flusserhaltung.

ist eine Strömung in $G^{\leftrightarrow}$.

b)
$$g(e) = 0 \text{ für alle Kanten } e \notin E(G_f)$$

c)
$$c(g) = c(f') - c(f).$$

Bezeichnung: $g = f' \Delta f$.

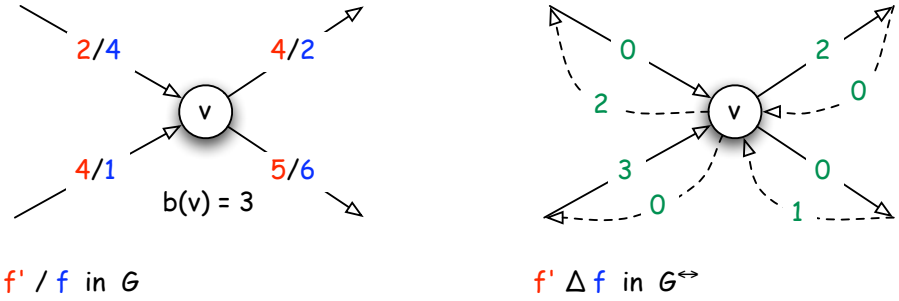


Abbildung 2.6: Beispiel zur Illustration der Aussage von Proposition 2.5.

Beweis. a) g ist eine Strömung:

$S_1(v) :=$ Einfluss bzgl. f' - Einfluss bzgl. f in G
$= (\text{Einfluss - Ausfluss bzgl. g über alle Kanten } e, e^{\leftarrow} | e \in \delta^-(v) \text{ in G}) =: R_1(v).$

Im Beispiel: $6 - 5 = 3 - 2$.

$S_2(v) :=$ Ausfluss bzgl. $f' -$ Ausfluss bzgl. f
$= (\text{Ausfluss} - \text{Einfluss bzgl. g über alle Kanten } e, e^{\leftarrow} | e \in \delta^+(v) \text{ in G}) =: R_2(v).$

Im Beispiel: $9 - 8 = 2 - 1$.

Übung 2.6. Zeige $S_1(v) = R_1(v)$ und $S_2(v) = R_2(v)$.

Also gilt für jeden Knoten v:

$$\begin{aligned}
&\text{Ausfluss in } G^{\leftrightarrow} - \text{Einfluss in } G^{\leftrightarrow} \text{ bzgl. } g \\
&= \sum_{e \in \delta^+_{G^{\leftrightarrow}}(v)} g(e) - \sum_{e \in \delta^-_{G^{\leftrightarrow}}(v)} g(e) \\
&= R_2(v) - R_1(v) \\
&= S_2(v) - S_1(v) \\
&= \sum_{e \in \delta^+_G(v)} (f'(e) - f(e)) - \sum_{e \in \delta^-_G(v)} (f'(e) - f(e)) \\
&= \sum_{e \in \delta^+_G(v)} f'(e) - \sum_{e \in \delta^-_G(v)} f'(e) - \left(\sum_{e \in \delta^+_G(v)} f(e) - \sum_{e \in \delta^-_G(v)} f(e) \right) \\
&= b(v) - b(v) = 0
\end{aligned}$$

b): $g(e) = 0$ für alle Kanten $e \notin E(G_f)$.
Sei nun $e \in E(G^{\leftrightarrow}) - E(G_f)$. Wir führen eine Fallunterscheidung durch:
(1) e ist VK:

$$\Rightarrow f(e) = u(e) \Rightarrow f'(e) \leq f(e) \Rightarrow g(e) = 0.$$

(2) $e = \bar{e}^{\leftarrow}$ ist RK zu $\bar{e}$:

$$\Rightarrow f(\bar{e}) = 0 \Rightarrow f'(\bar{e}) \geq f(\bar{e}) \Rightarrow g(e) = 0.$$

Als letztes zeigen wir:
c): $c(g) = c(f') - c(f)$:

$$c(g) = \sum_{e \in E(G^{\leftrightarrow})} c(e) \cdot g(e) = \sum_{e \in E(G)} c(e) \cdot g(e) + c(e^{\leftarrow}) \cdot g(e^{\leftarrow}).$$

Für die Summanden der letzten Summe gilt:

$$\begin{aligned}
&f'(e) > f(e) \Rightarrow c(e) \cdot g(e) + c(e^{\leftarrow}) \cdot g(e^{\leftarrow}) = c(e) \cdot (f'(e) - f(e)) \\
&f'(e) < f(e) \Rightarrow c(e) \cdot g(e) + c(e^{\leftarrow}) \cdot g(e^{\leftarrow}) = c(e^{\leftarrow}) \cdot (f(e) - f'(e)) = c(e) \cdot (f'(e) - f(e)) \\
&f'(e) = f(e) \Rightarrow c(e) \cdot g(e) + c(e^{\leftarrow}) \cdot g(e^{\leftarrow}) = 0.
\end{aligned}$$

Also folgt

$$c(g) = \sum_{e \in E(G)} c(e) \cdot (f'(e) - f(e)) = c(f') - c(f).$$

□

Nun können wir auf die Strömung $g = f' \Delta f$ die uns bekannten Dekompositionssätze anwenden.

Proposition 2.7. $g = f'\Delta f$ ist positive Linearkombination von Inzidenzvektoren von höchstens m Zykeln in G_f.

Wir erhalten nun den zentralen Satz dieses Abschnitts:

Satz 2.8 (Optimalitätskriterium für MCFP, Klein 1967). Ein b-Fluss f in (G, u, b, c) hat minimale Kosten $\Leftrightarrow$ es gibt keinen f-augmentierenden Zykel mit negativen Kosten (d.h. es gibt keinen Zykel Z in G_f mit $c(Z) = \sum_{e \in E(Z)} c(e) < 0$).

Beweis. $\Rightarrow$:
Annahme f ist optimal und es gibt einen f-augmentierenden Zykel Z mit $c(Z) < 0$. Augmentiere f entlang Z um $\gamma > 0$, d.h.,

$$f'(e) \leftarrow \begin{cases} f(e) + \gamma, & \text{falls } e \text{ VK in } Z \\ f(e) - \gamma, & \text{falls } e^{\leftarrow} \text{ RK in } Z \\ f(e), & e, e^{\leftarrow} \notin Z. \end{cases}$$

Dadurch ergibt sich eine Kostenänderung um

$$c(f') - c(f) = \sum_{e \text{ VK in } Z} \gamma \cdot c(e) - \sum_{e^{\leftarrow} \text{ RK in } Z} \gamma \cdot c(e) = \gamma \cdot c(Z) < 0.$$

Dies ergibt einen Widerspruch zur Optimalität von f.

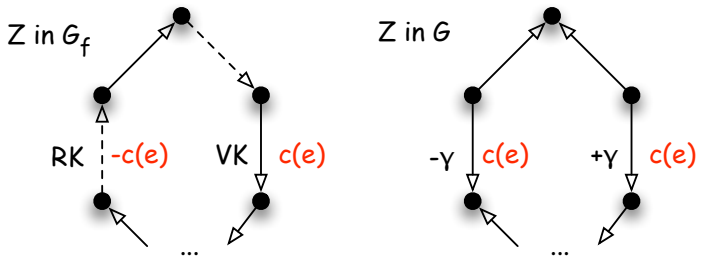


Abbildung 2.7: Beispiel zur Illustration eines negativen Zykels.

$\Leftarrow$:
Angenommen es gibt keinen f-augmentierenden Zykel mit negativen Kosten, aber f ist nicht optimal $\Rightarrow$ es gibt einen b-Fluss f' mit $c(f') < c(f)$
$\Rightarrow$ (Proposition 2.5) $g := f'\Delta f$ ist eine Strömung in G_f und $c(g) = c(f') - c(f) < 0 \Rightarrow$ (Proposition 2.7) g ist positive Linearkombination von Inzidenzvektoren von Zykeln von G_f, etwa

$$g = \lambda_1 \cdot \zeta(Z_1) + \lambda_2 \cdot \zeta(Z_2) + \cdots + \lambda_k \cdot \zeta(Z_k) \text{ mit } \lambda_i > 0.$$

$\Rightarrow$ (c linear):

$$0 > c(g) = \lambda_1 \cdot c(Z_1) + \lambda_2 \cdot c(Z_2) + \cdots + \lambda_k \cdot c(Z_k) \text{ mit } \lambda_i > 0.$$

$\Rightarrow$ ein Zykel Z hat negative Kosten

$\Rightarrow$ es gibt f-augmentierenden Zykel mit negativen Kosten, Widerspruch. □

In Abb. 2.8-2.11 geben wir nun ein Beispiel.

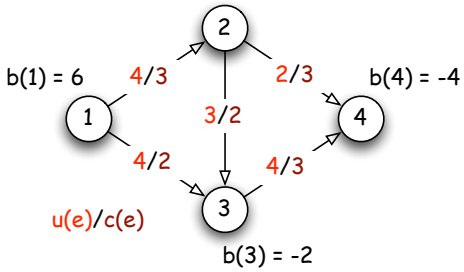


Abbildung 2.8: Beispiel (G, u, c).

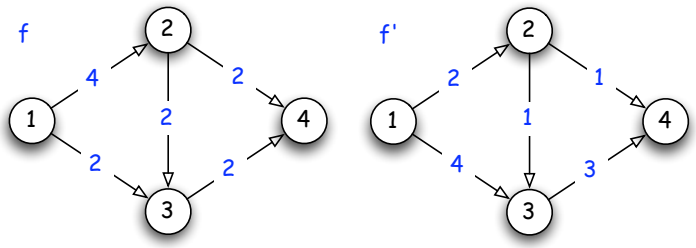


Abbildung 2.9: b-Fluss f mit $c(f) = 32$ und b-Fluss f' mit $c(f') = 28$.

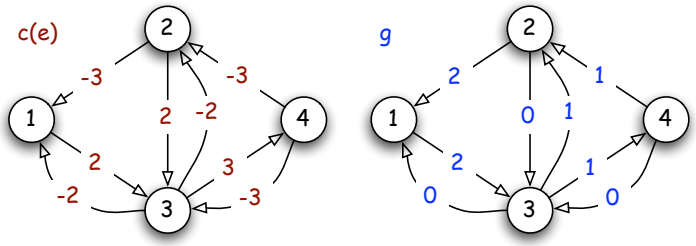


Abbildung 2.10: Residualgraph G_f mit Kosten $c(e)$ und Strömung g in G_f.

Wir werden nun abschließend eine Reformulierung des Optimalitätskriteriums vorstellen.

Definition 2.9. Sei $G = (V, E)$ ein Digraph und $c : E(G) \to \mathbb{R}$ seien Kantenkosten. Ein Potenzial ist eine reellwertige Knotenbewertung $\pi(v)$ für jeden Knoten $v \in V(G)$. Die reduzierten Kosten einer Kante $e = (x, y)$ sind die um die Potenzialdifferenz $\pi(y) - \pi(x)$ bzgl. π reduzierten Kosten

$$c_\pi(x, y) := c(x, y) - \pi(y) + \pi(x).$$

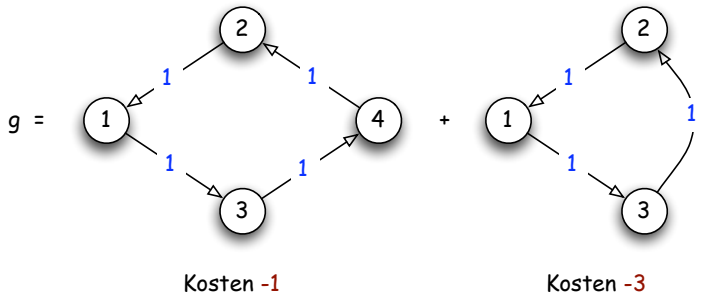


Abbildung 2.11: Zerlegung von g als positive Linearkombination von Zykeln in G_f.

Ein Potential π heisst zulässig bzgl. c, wenn gilt

$$c_\pi(x,y) \geq 0 \text{ für alle Kanten } (x,y) \in E(G).$$

Korollar 2.10 (Ford-Fulkerson 1962). Ein b-Fluss f in (G,u,b,c) ist optimal $\Leftrightarrow$ es gibt ein zulässiges Potenzial in (G_f,c).

Ein solches Potential (bzw. einen negativen Zykel) können wir mit Kürzeste-Wege-Algorithmen ermitteln.

2.3 Cycle-Canceling Algorithmen

Das Optimalitätskriterium für das MCFP suggeriert folgenden Algorithmus:

Eingabe: (G,u,b,c).
Ausgabe: Kostenminimaler b-Fluss f (oder Aussage, dass kein b-Fluss existiert).
Methode:
Berechne b-Fluss f in (G,u) (oder entscheide, dass kein b-Fluss existiert).
while G_f enthält einen Zykel mit negativen Kosten **do**
  augmentiere f entlang eines solchen Zykels um die minimale Residualkapazität
end
return: f

Algorithm 2.3.1: Cycle Canceling Algorithmus

Wir sehen sofort ein, dass wir effizient die Hauptmethode implementieren können:

Satz 2.11. Sei G ein Digraph mit beliebiger Kantenbewertung c. Dann findet man in $O(n \cdot m)$ Zeit ein zulässiges Potenzial oder einen negativen Zykel.

Beweis. Wende Bellmann-Ford-Moore an. □

Probleme: Terminierung klar, falls u,b,c ganzzahlig sind. Aber: die Kosten ändern sich mit jedem gefundenen Zykel evtl. nur um 1 $\rightarrow$ evtl. viele Iterationen nötig.

Bemerkung 2.12. Ermittlung eines (elementaren) Zykels mit minimalem Gewicht ist algorithmisch schwer (NP-schwer) denn: $c(e) = -1$ für alle $e \Rightarrow$ minimales Gewicht eines elementaren Zykel $= -n \Leftrightarrow G$ hat einen Hamilton Kreis.

Wie sich herausstellen wird, sind Zykel mit minimalem mittlerem Gewicht sehr gut geeignet.

2.4 Ein Algorithmus zur Berechnung eines Minimum Mean Cycles

Input: (G, c) wobei G Digraph und $c(e), e \in E(G)$ beliebige reelle Kantengewichte.
Aufgabe: Finde einen elementaren Zykel Z dessen mittleres Gewicht $c(Z)/|E(Z)|$ minimal ist oder stelle fest, dass G azyklisch ist.

Grundlage des Algorithmus von Karp zur Berechnung eines minimum mean cycles ist folgende Struktureinsicht:

Satz 2.13 (Karp 1978). Sei G Digraph, $c(e)$ Kantengewichte und alle Knoten seien von $s \in V(G)$ erreichbar. Für $v \in V(G)$ sei

$F_k(v) :=$ minimale Länge eines s-v Weges mit genau k Kanten, falls existent, und ∞ sonst.

Sei

$\mu(G) :=$ minimales mittleres Gewicht eines elementaren Zykels, falls existent, und ∞ sonst.

Dann gilt

$$\mu(G) = \min_{v \in V(G)} \max_{0 \le k \le n-1} \frac{F_n(v) - F_k(v)}{n - k}.$$

Beweis. Falls G azyklisch ist, folgt, dass $F_n(v) = \infty$, da kein Weg mit genau n Kanten existiert. Es folgt Gleichheit, da dann auch per Def. $\mu(G) = \infty$ ist.
G habe also einen Zykel, d.h. $\mu(G) < \infty$.
Fall 1: $\mu(G) = 0$.
Wir zeigen $\min_{v \in V(G)} \max_{0 \le k \le n-1} \frac{F_n(v) - F_k(v)}{n-k} = 0$.
$\mu(G) = 0 \Rightarrow G$ hat keinen negativen Zykel, d.h., c ist konservativ. Sei $\mathrm{dist}(v)$ die Länge eines kürzesten elementaren Weges von s nach v. Mit

$$F_n(v) \ge \mathrm{dist}(v) = \min\{F_k(v) | 0 \le k \le n-1\},$$

folgt

$$\max_{0 \le k \le n-1} \frac{F_n(v) - F_k(v)}{n - k} \ge 0.$$

Behauptung: es gibt Knoten v mit $F_n(v) = \text{dist}(v)$.
Beweis der Behauptung:

- sei Z ein elementarer Zykel in G mit Länge 0 (existiert wegen $\mu(G) = 0$)
- sei w Knoten in Z
- sei W kürzester s, w-Weg $+n$ mal $Z[w, w]$ (Weg bis w und n mal um den Zykel Z)
- sei W' das Anfangsstück von W der ersten n Kanten
- sei v der Endknoten von W'

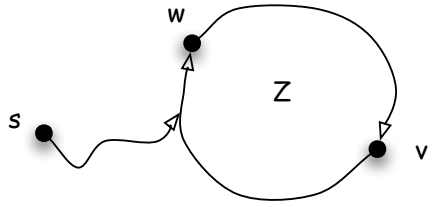


Abbildung 2.12: Konstruktion von W'.

Konstruktion $\Rightarrow$ W ist (nicht-elementarer) s, w-Weg der Länge $\text{dist}(w)$, da $c(Z) = 0$
$\Rightarrow$ W ist kürzester (nicht-elementarer) s, w-Weg
$\Rightarrow$ Anfangsstück W' von W ist kürzester (nicht-elementarer) s, v-Weg
W' hat n Kanten $\Rightarrow$ $c(W') = \text{dist}(v) = F_n(v)$.

Behauptung $\Rightarrow$

$$\max_{0 \le k \le n-1} \frac{F_n(v) - F_k(v)}{n - k} = 0.$$

Fall 2: $\mu(G)$ beliebig.

Addition einer Konstanten K zu allen Kantenkosten ändert $\mu(G)$ und min max ... nur um K. Mit $c'(e) := c(e) + K$ erhalten wir:

$$c'(Z)/|E(Z)| = (c(Z) + |E(Z)| \cdot K)/|E(Z)| = c(Z)/|E(Z)| + K$$

$$\begin{aligned} &(F'_n(v) - F'_k(v))/(n - k) \text{ (bzgl. } c') \\ &= (F_n(v) - F_k(v) + (n - k) \cdot K)/(n - k) \text{ (bzgl. } c) \\ &= (F_n(v) - F_k(v))/(n - k) + K \text{ (bzgl. } c) \end{aligned}$$

Wähle nun $K := -\mu(G)$ und wir können den ersten Fall anwenden.

□

Wir erhalten folgenden Algorithmus:

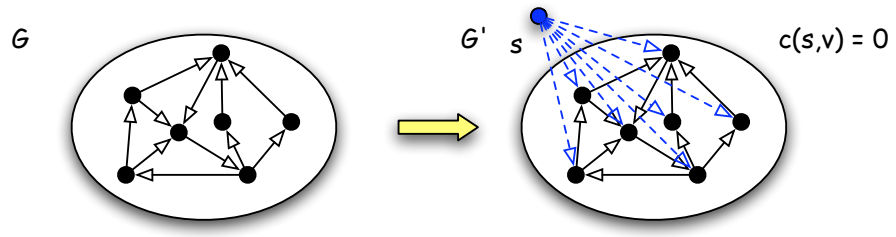


Abbildung 2.13: Transformation von G auf G′.

Eingabe: Gewichteter Digraph (G, c).
Ausgabe: elementarer Zykel Z mit minimalem mittlerem Gewicht oder Information, dass G azyklisch ist.
Methode:
1. Transformiere G mit dem Ziel, alle Knoten von einem Knoten s aus erreichbar zu machen (siehe Abb. 2.13)
(falls es so einen Knoten bereits gibt, kann dieser Schritt entfallen)
Setze $G := G'$
2. Initialisierung:
$F_0(s) := 0; F_0(v) := \infty$, sonst.
3. Hauptschleife, analog zu Bellman-Ford-Moore, nur "rückwärts":

```
for k := 1 to n do
    forall Knoten v aus V(G) do
        F_k(v) := ∞
        forall Kanten (u,v) in δ⁻(v) do
            if F_{k-1}(u) + c(u,v) < F_k(v) then
                F_k(v) := F_{k-1}(u) + c(u,v)
                pred_k(v) := u // Vater von u in Phase k
            end
        end
    end
end
```

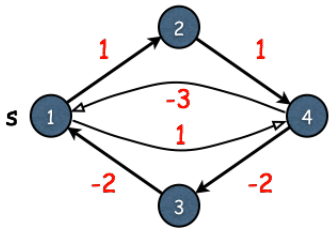
4. Test auf azyklisch

```
if F_n(v) = ∞ für alle v aus V(G) then
    return // G ist azyklisch
end
```

5. Identifikation eines elementaren Zykels mit minimalem mittleren Gewicht: sei v Knoten mit minimalem Wert $\min_{v \in V(G)} \max_{0 \le k \le n-1} \frac{F_n(v) - F_k(v)}{n-k}$
sei Z elementarer Zykel in
$v, \mathrm{pred}_n(v), \mathrm{pred}_{n-1}(\mathrm{pred}_n(v)), \mathrm{pred}_{n-2}(\mathrm{pred}_{n-1}(\mathrm{pred}_n(v))), \ldots$ (muss nicht v enthalten)
return: Z

Algorithm 2.4.1: Minimum Mean Cycle Algorithmus

Wir rechnen nun ein Beispiel durch (siehe Abb. 2.14).



$\text{pred}_k(v)$

$$\text{blau} = \max_{\substack{0<k<n-1 \\ F_n(v)<\infty}} \frac{F_n(v)-F_k(v)}{n-k}$$

alle Knoten v
nehmen Minimum an

k	0	1	2	3	4=n
$F_k(1)$	0	∞	-2	-3	-4
$F_k(2)$	∞	1	∞	-1	-2
$F_k(3)$	∞	∞	-1	0	-3
$F_k(4)$	∞	1	2	-1	-2
$\frac{F_n(1)-F_k(1)}{n-k}$	$\frac{-4-0}{4-0}=-1$	-	$\frac{-4+2}{4-2}=-1$	$\frac{-4+3}{4-3}=-1$	
$\frac{F_n(2)-F_k(2)}{n-k}$	-	$\frac{-2-1}{4-1}=-1$	-	$\frac{-2+1}{4-3}=-1$	
$\frac{F_n(3)-F_k(3)}{n-k}$	-	-	$\frac{-3+1}{4-2}=-1$	$\frac{-3-0}{4-3}=-3$	
$\frac{F_n(4)-F_k(4)}{n-k}$	-	$\frac{-2-1}{4-1}=-1$	$\frac{-2-2}{4-2}=-2$	$\frac{-2+1}{4-3}=-1$	

Abbildung 2.14: Beispiel des Minimum Mean Cycle Algorithmus.

Satz 2.14. Der Minimum Mean Cycle Algorithmus arbeitet korrekt und läuft in $O(n \cdot m)$ Zeit.

Beweis.

- Schritt 1 erzeugt keine neuen Zykel, macht Satz 2.13 anwendbar.
- Schritte 2 und 3 berechnen die $F_k(v)$ korrekt denn sie implementieren die Rekursionsformel
$$F_k(v) = \min\{F_{k-1}(u) + c(u,v) | (u,v) \in \delta^-(v)\}.$$
- Schritt 4 testet korrekt auf azyklisch, denn $F_n(v) = \infty$ für alle $v \Rightarrow$ es gibt keine s,v-Wege mit n Kanten $\Rightarrow$ es gibt keine Zykel.
- Schritt 5 ist korrekt.
Betrachte Instanz (G,c') mit $c'(e) := c(e) - \mu(G)$.
Auf dieser Instanz läuft der Algorithmus genauso, nur die F-Werte ergeben sich zu $F'_k(v) = F_k(v) - k \cdot \mu(G)$. Wahl von v in Schritt 5, siehe Beweis von Satz 2.13 $\Rightarrow F'_n(v) = \min\{F'_k(v) | k = 0,1,\ldots,n-1\}$.
$\mu(G) = 0$ bzgl. $c' \Rightarrow$ (es gibt keine negativen Zykel bzgl. c')
$\Rightarrow$ Wege von s nach v mit n Kanten und Länge $F'_n(v)$ in (G,c') bestehen aus einem kürzesten elementaren s,v-Weg + Zykel der Länge 0 (ein oder mehrere) diese Null-Zykel haben mittleres Gewicht $\mu(G)$ in (G,c), denn Zykel in (G,c) und (G,c') unterscheiden sich bzgl. mittlerer Länge gerade um $\mu(G)$ (siehe Beweis Satz 2.13).

Schritt 5 wählt einen solchen elementaren Zykel.

Laufzeitanalyse:

- Schritt 1 : $O(n)$
- Schritt 2 : $O(n)$
- Schritt 3 : $O(n \cdot m)$ (zu jedem Knoten v alle (u, v) betrachten)
- Schritt 4 : $O(n)$
- Schritt 5 : $O(n^2)$ zur Bestimmung von v
- $O(n)$ zur Suche des Zykels

Summe $O(n \cdot (m + n)) = O(nm)$.

□

2.5 Minimum Mean Cycle Canceling Algorithmus

Eingabe: (G, u, b, c).
Ausgabe: Kostenminimaler b-Fluss f (oder Aussage, dass kein b-Fluss existiert).
Methode:
Berechne b-Fluss f in (G, u) (oder entscheide, dass kein b-Fluss existiert; benutze max flow).
while G_f enthält einen Zykel mit negativen Kosten **do**
  berechne einen Zykel mit minimalem mittleren Gewicht
  augmentiere f entlang dieses Zykels um die minimale Residualkapazität
end
return: f

Algorithm 2.5.1: Minimum Mean Cycle Canceling Algorithmus

Der Korrektheitsbeweis benötigt ähnliche Ideen wie beim Algorithmus von Edmonds-Karp.

Sei

$$\mu(f) := \text{ minimale mittlere Zykellänge in } G_f.$$

Zeige dass $|\mu(f)|$ schnell genug abnimmt im Algorithmus (sofern $\mu(f)$ negativ ist).

Lemma 2.15. Folgende Voraussetzungen seien erfüllt:

- Sei $f_1, f_2, \ldots$ eine Folge von b-Flüssen in (G, u, b, c) und sei G_{f_k} der zu f_k gehörige Residualgraph.
- f_{k+1} entstehe aus f_k durch Augmentierung entlang des negativen Zykels Z_k.
- Z_k sei neg. Zykel minimalen mittleren Gewichts in G_{f_k}, d.h. $c(Z_k)/|E(Z_k)| = \mu(f_k)$.

Dann gilt:

(a) $\mu(f_k) \leq \mu(f_{k+1})$ für alle k (schwache Monotonie).

(b) $\mu(f_k) \leq (n/(n-1)) \cdot \mu(f_r)$ für alle $k < r$ so dass $Z_k \cup Z_r$ ein Paar entgegengesetzter Kanten $e, e^{\leftarrow}$ enthält (starke Monotonie).

Beweis. Zu (a): betrachte f_k und f_{k+1}. Sei

$$H := Z_k + Z_{k+1} - \text{ Paare entgegengesetzter Kanten } e, e^{\leftarrow} \text{ (Mehrfachkanten möglich).}$$

Wir zeigen:
(1) $E(H) \subseteq E(G_{f_k})$ (analog zum Beweis bei Edmonds-Karp)
(2) $c(E(H)) \geq \mu(f_k) \cdot |E(H)|$, denn H ist Eulersch im gerichteten Sinne und wegen (1) Teilgraph von $G_{f_k} \Rightarrow H$ ist kantendisjunkte Vereinigung von (gerichteten) elementaren Zykeln in G_{f_k}, etwa $C_1, C_2, \ldots, C_s$
$\Rightarrow c(E(C_i)) \geq \mu(f_k) \cdot |E(C_i)|$ für $i = 1, 2, \ldots, s$
$\Rightarrow c(E(H)) = c(C_1) + \cdots + c(C_s) \geq \mu(f_k) \cdot (|E(C_1)| + \cdots + |E(C_s)|) = \mu(f_k) \cdot |E(H)|$.
(3) $|E(H)| \leq |E(Z_k)| + |E(Z_{k+1})|$
klar, da bei H Paare entgegengesetzter Kanten weggelassen sind
(4) $c(E(H)) = \mu(f_k) \cdot |E(Z_k)| + \mu(f_{k+1}) \cdot |E(Z_{k+1})|$, denn $c(E(H)) = c(E(Z_k)) + c(E(Z_{k+1}))$, da sich die Kosten für e und $e^{\leftarrow}$ wegheben
$= \mu(f_k) \cdot |E(Z_k)| + \mu(f_{k+1}) \cdot |E(Z_{k+1})|$ wegen der Wahl von Z_k und Z_{k+1}.
Kombination von (2) - (4) ergibt

$$\begin{aligned}
\mu(f_k) \cdot (|E(Z_k)| + |E(Z_{k+1})|) &\leq \mu(f_k) \cdot |E(H)| \text{ wegen (3) und } \mu(f_k) < 0 \\
&\leq c(E(H)) \text{ wegen (2)} \\
&= \mu(f_k) \cdot |E(Z_k)| + \mu(f_{k+1}) \cdot |E(Z_{k+1})| \text{ wegen (4)} \\
\Rightarrow \mu(f_k) \cdot |E(Z_{k+1})| &\leq \mu(f_{k+1}) \cdot |E(Z_{k+1})| \\
\Rightarrow \mu(f_k) &\leq \mu(f_{k+1})
\end{aligned}$$

Zu (b):
Die Aussage muss wegen (a) nur gezeigt werden für k, r sodass für $k < i < r$, $Z_i \cup Z_r$ kein Paar entgegengesetzter Kanten $e, e^{\leftarrow}$ enthält. Wie bei Edmonds-Karp betrachte $H := Z_k + Z_r$ - Paare entgegengesetzter Kanten $e, e^{\leftarrow}$ (Mehrfachkanten möglich). Wie bei Edmonds-Karp folgt $E(H) \subseteq E(G_{f_k})$.
Wie in (a), folgt: (2) $c(E(H)) \geq \mu(f_k) \cdot |E(H)|$
(4) $c(E(H)) = \mu(f_k) \cdot |E(Z_k)| + \mu(f_r) \cdot |E(Z_r)|$.
Unterschied in (3):
(3') $|E(H)| \leq ((n-1)/n) \cdot (|E(Z_k)| + |E(Z_r)|)$, denn in H werden mindestens 2 Kanten von maximal $2n$ weggenommen:

$$|E(H)| \leq \frac{2n-2}{2n} (|E(Z_k)| + |E(Z_r)|) = \frac{n-1}{n} (|E(Z_k)| + |E(Z_r)|) .$$

Wie in (a):

$$\begin{aligned}\mu(f_k)\cdot((n-1)/n)\cdot(|E(Z_k)|+|E(Z_r)|) &\leq \cdots \leq \mu(f_k)\cdot|E(Z_k)|+\mu(f_r)\cdot|E(Z_r)|\\ &\leq \mu(f_r)\cdot(|E(Z_k)|+|E(Z_r)|) \text{ (wegen (a))}\\ \Rightarrow \mu(f_k) &\leq (n/(n-1))\cdot\mu(f_r).\end{aligned}$$

□

Wir erhalten eine wichtige Folgerung:

Korollar 2.16. Im Minimum Mean Cycle Canceling Algorithmus nimmt $|\mu(f)|$ in $m\cdot n$ Iterationen mindestens um den Faktor $1/2$ ab. Daher terminiert der Algorithmus bei ganzzahligen Kosten $c(e)$ nach spätestens $n\cdot m\cdot(\lceil\log_2(n\cdot c_{max})\rceil+1)$ Iterationen mit einer Gesamtlaufzeit von $O(n^2m^2\log(n\cdot c_{max}))$ mit $c_{max}=\max\{|c(e)|:e\in E(G)\}$.

Beweis. Betrachte Folge $Z_k, Z_{k+1}, ..., Z_{k+m}$ sukzessiver Zykel im Algorithmus.

- Jeder Zykel enthält Engpasskante bzgl. Augmentierung $\Rightarrow$ Kante wird entgegengesetzt.
- Es sind $m+1$ Zykel aber nur m Kanten in G $\Rightarrow$ zwei Zykel in der Folge müssen zueinander entgegengesetzte Kanten $e, e^{\leftarrow}$ enthalten, etwa Z_i und Z_j $(i<j)$

  $\Rightarrow$ (mit Lemma 2.15) $\mu(f_k)\leq\mu(f_i)\leq(n/(n-1))\cdot\mu(f_j)\leq(n/(n-1))\cdot\mu(f_{k+m})$.

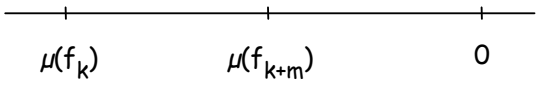


Abbildung 2.15: Situation auf der Zahlengeraden.

$\Rightarrow |\mu(f_k)|\geq(n/(n-1))\cdot|\mu(f_{k+m})|$

$\Rightarrow |\mu(f_{k+m})|\leq((n-1)/n)|\mu(f_k)|$

$\Rightarrow |\mu(f)|$ verkleinert sich nach m Iterationen um mindestens den Faktor $(n-1)/n$

$\Rightarrow |\mu(f)|$ verkleinert sich nach $m\cdot n$ Iterationen um mindestens den Faktor

$$\left(\frac{n-1}{n}\right)^n=\left(1-\frac{1}{n}\right)^n\to e^{-1}\approx 0.37<1/2.$$

Behauptung: $\mu(f)\geq 0$ nach spätestens $t:=n\cdot m\cdot(\lceil\log_2(n\cdot c_{max})\rceil+1)$ Iterationen.
Angenommen nicht.
$|\mu(f)|$ zu Beginn maximal c_{max}.
Wir zeigen, dass nach t Iterationen $|\mu(f)|<1/n$ gilt.

$$|\mu(f)|<1/n$$

$$\Leftrightarrow n \cdot |\mu(f)| < 1$$
$$\Leftrightarrow n \cdot |\mu(f)| \text{ muss } \lceil \log_2(n \cdot |\mu(f)|) \rceil + 1 \text{ mal halbiert werden.}$$

Dies ist spätestens nach $n \cdot m \cdot (\lceil \log_2(n \cdot |\mu(f)|) \rceil + 1) \leq n \cdot m \cdot (\lceil \log_2(n \cdot c_{max}) \rceil + 1)$ Iterationen erreicht.
Als nächstes zeigen wir

$$|\mu(f)| < 1/n \Rightarrow \mu(f) \geq 0, \text{ d.h. es gibt keinen Zykel negativer Länge mehr.}$$

$$|\mu(f)| < 1/n \Rightarrow \mu(f) > -1/n.$$

$c(e)$ ganzzahlig $\Rightarrow$

$$\mu(f) = \frac{\text{Kosten Zykel}}{\text{Anzahl Kanten Zykel}} = \frac{\text{ganze Zahl}}{\text{Zahl in } \{2, 3, \ldots, n\}} > -1/n.$$

Obige Ungleichung ist nur möglich, wenn die ganze Zahl nicht-negativ ist $\Rightarrow \mu(f) \geq 0$.

Also Abbruch nach spätestens $n \cdot m \cdot (\lceil \log_2(n \cdot c_{max}) \rceil + 1)$ Iterationen. Pro Iteration müssen wir ein Minimum Mean Cycle Problem lösen. Das geht in $O(n \cdot m)$ Zeit, also ist der Aufwand $O(n^2 m^2 \log(n \cdot c_{max}))$ insgesamt.

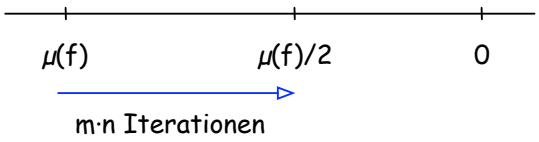


Abbildung 2.16: Situation auf der Zahlengeraden.

□

Bemerkung 2.17. Die Laufzeit hängt noch von c_{max} ab und ist daher nicht streng polynomial, d.h. polynomial in n und m.

Tatsächlich hat der Algorithmus jedoch streng polynomiale Laufzeit (für einen Beweis siehe Korte & Vygen, Combinatorial Optimization, 5th edition, Seite 218):

Satz 2.18 (Goldberg & Tarjan 1989). Der Minimum Mean Cycle Canceling Algorithmus läuft in $O(m^3 n^2 \log n)$ Zeit.

2.6 Der Successive Shortest Path Algorithmus

Idee wie bei Push-Relabel:

- sorge dafür, dass das Optimalitätskriterium immer erfüllt ist
- relaxiere die Balance-Bedingung in den Knoten
- stelle Balance Bedingung am Ende her

Satz 2.19 (Jewell 1958, Iri 1969, Busacker & Gowen 1961). Sei f ein kostenminimaler b-Fluss in (G, u, b, c). Seien s, t zwei Knoten und W ein bzgl. c kürzester s,t-Weg in G_f. Sei f' der Fluss, der durch Augmentation von f entlang W mit Erhöhungswert γ entsteht. Dann ist f' ein kostenminimaler b'-Fluss für die um γ veränderten b-Werte für s und t, d.h. $b'(s) := b(s) + \gamma, b'(t) := b(t) - \gamma, b'(v) := b(v)$, sonst.

Beweis. f' ist b'-Fluss nach Konstruktion. Angenommen f' ist nicht kostenminimal. Dann gibt es einen negativen Zykel Z in $G_{f'}$. Sei $H := Z + W$ - entgegengesetzte Paare von Kanten (wie in Lemma 2.15).

$$\Rightarrow E(H) \subseteq E(G_f) \text{ (wie in Lemma 2.15)}$$
$$\Rightarrow c(E(H)) = c(E(Z)) + c(E(W))$$

da sich die Kosten entgegengesetzter Paare von Kanten zu 0 ergeben

$$\Rightarrow c(E(H)) < c(E(W), \text{ da } c(E(Z)) < 0.$$

H ist kantendisjunkte Vereinigung eines s,t-Weges W' und eines oder mehrerer Zykel, alle in G_f (wie in Lemma 2.15). f kostenminimal $\Rightarrow$ diese Zykel haben Kosten $\geq 0 \Rightarrow c(E(W') \leq c(E(H)) < c(E(W))$, im Widerspruch zur Wahl von W.

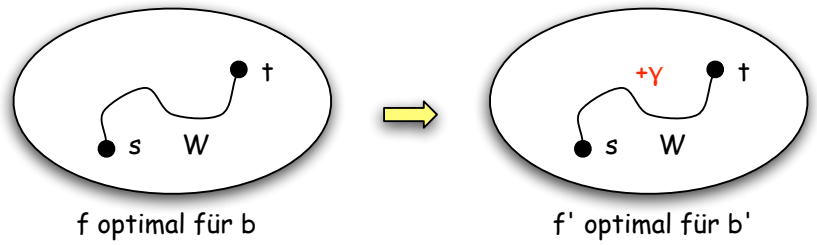


Abbildung 2.17: Augmentierung entlang W.

□

Satz 2.19 ergibt folgende Idee für einen Algorithmus:

- Starte mit b'-Fluss f, für den c konservativ in G_f ist
- Augmentiere entlang kürzester s,t-Wege bis vorgegebene Balancen $b(v)$ erreicht sind
- Frage: wie erreicht man konservatives c?

- Fall 1: c ist konservativ in G
  $\Rightarrow$ starte mit $f = 0$
- Fall 2: c ist nicht konservativ in G
  $\Rightarrow$ sättige Kanten e mit $c(e) < 0$ (d.h. $f(e) := u(e)$) und ändere Balancen entsprechend $\Rightarrow$ diese Kanten e sind nicht mehr in G_f $\Rightarrow$ c ist konservativ in G_f

Eingabe: (G, u, b, c).
Ausgabe: Kostenminimaler b-Fluss f (oder Aussage, dass kein b-Fluss existiert).
Methode:
Mache c konservativ und bestimme anfängliche Balancen b'
setze $b'(v) := 0$ für alle Knoten v, $f(e) := 0$ für alle Kanten e
if c nicht konservativ **then**
  // mache c konservativ durch Sättigung von Kanten mit negativen Kosten
  forall Kanten $e = (v, w)$ mit $c(e) < 0$ **do**
    $f(e) := u(e)$ // sättige e
    $b'(v) := b'(v) + u(e); b'(w) := b'(w) - u(e)$ // passe Balancen an
  end
end
Hauptschleife:
while es gibt Knoten v mit $b'(v) \neq b(v)$ **do**
  wähle Knoten s mit $b(s) - b'(s) > 0$
  if es gibt einen Knoten t mit $b(t) - b'(t) < 0$ der von s in G_f erreichbar ist **then**
    wähle einen solchen Knoten t
  end
  else
    return // es gibt keinen b-Fluss (Übung)
  end
  berechne in G_f einen s,t-Weg W mit minimalen Kosten
  setze $\gamma := \min\{\min_{e \in E(W)} u_f(e), b(s) - b'(s), |b(t) - b'(t)|\}$
  augmentiere f entlang W um γ
  $b'(s) := b'(s) + \gamma, b'(t) := b'(t) - \gamma$.
end
return: f

Algorithm 2.6.1: Successive Shortest Path Algorithmus

Wir geben nun ein Beispiel (siehe Abb. 2.18).

Zum Aufwand des Algorithmus:

Bei beliebigen Kapazitäten ähnliche Probleme wie beim Ford-Fulkerson Algorithmus.

- Falls u und b ganzzahlig sind $\Rightarrow$ Abbruch nach höchstens $B := (1/2) \sum |b(v) - b'(v)|$ Iterationen mit den anfänglichen b-Werten und den im Algorithmus gesetzten b'-Werten
- pro Iteration eine Kürzeste-Wege Berechnung mit Moore-Bellman-Ford (da negative Kantengewichte) $\Rightarrow$ Aufwand $O(n \cdot m + B \cdot n \cdot m)$
- Verbesserung möglich durch Verwendung von zulässigen Potenzialen

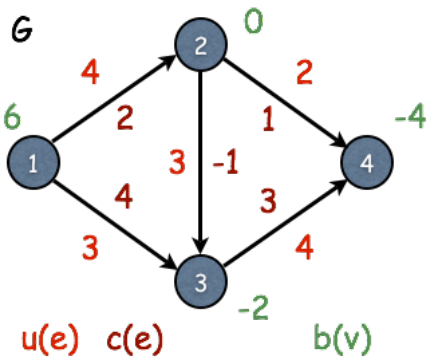


Abbildung 2.18: Beispielinstanz für den Algorithmus.

- Verwendung von Moore-Bellman-Ford einmal am Anfang, Transformation der Kantengewichte durch zulässiges Potenzial, danach pro Iteration nur Dijkstra und Anpassung des zulässigen Potenzials
  $\Rightarrow$ Verbesserung auf $O(nm + B(n^2 + m))$ (Edmonds & Karp 1972) ist für $B = O(n)$ schnellster bekannter Algorithmus

Bemerkung 2.20. Sei Z ein elementarer Zykel in (G, c) und π ein zulässiges Potenzial. Dann gilt:

$$c(Z) = c_\pi(Z).$$

Desweiteren gilt:

$$W \text{ ist ein kürzester } s,t\text{-Weg bzgl. } c \Leftrightarrow W \text{ ist ein kürzester } s,t\text{-Weg bzgl. } c_\pi.$$

Allgemein:

$$\text{Es gibt ein zulässiges Potenzial } \Leftrightarrow c \text{ ist konservativ (keine neg. Zykel).}$$

Satz 2.21 (Edmonds und Karp 1972). Für ganzzahlige Kapazitäten und Balancen kann der Successive Shortest Path Algorithmus durch Verwendung zulässiger Potenziale so implementiert werden, dass er in $O(nm + B(n^2 + m))$ Zeit läuft.

Beweis. 1. Mache c konservativ.

2. Vereinfachung des Problems auf eine einzige Quelle s: Falls die Instanz mehrere Knoten v mit $b(v) - b'(v) > 0$ hat, so transformiere sie wie in Abb. 2.19 dargestellt.

   Lösche zusätzlich alle Knoten, die nicht von s aus erreichbar sind (in G_f) und berechne zulässiges Potenzial π_0.

3. Berechne zulässiges Potenzial π nach Iteration i des Successive Shortest Path Algorithmus für die nächste Iteration $i + 1$. Dazu: sei f_{i-1} der Fluss vor der Iteration i

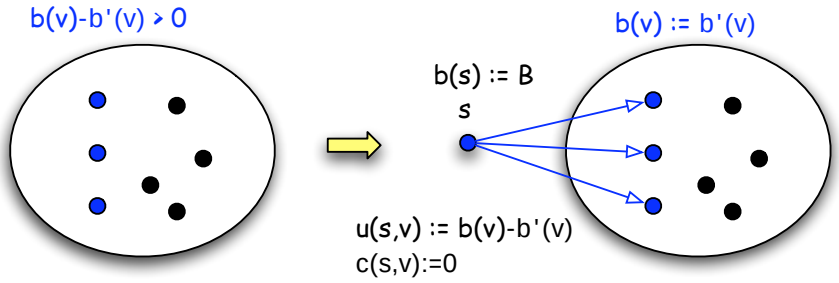


Abbildung 2.19: Transformation mit $B = (1/2)\sum |b(v) - b'(v)|$.

dann wird die Berechnung des kürzesten Weges in Iteration i bzgl. der transformierten Kosten $c_{\pi_{i-1}}(v,w) := c(v,w) - \pi_{i-1}(w) + \pi_{i-1}(v)$ ausgeführt, und zwar jedesmal von Knoten s aus, und π_{i-1} ist ein zulässiges Potenzial für $(G_{f_{i-1}}, c)$. Sei dann $d_i(v)$ die kürzeste Weglänge von s nach v (Knoten v mit $d_i(v) = \infty$ werden gelöscht).
Behauptung: $\pi_i(v) := \pi_{i-1}(v) + d_i(v)$ ist ein zulässiges Potenzial für (G_{f_i}, c).
Beweis durch Induktion nach $i \geq 1$ (beachte, dass π_0 zul. Potenzial für (G_{f_0}, c)):
Betrachte Kante $e = (v,w) \in G_{f_i}$.
Zeige: $c_{\pi_i}(e) \geq 0$.
Fall 1: $e \in E(G_{f_{i-1}})$:
Nach Dreiecksungleichung und Induktionsvoraussetzung ist

$$\begin{aligned} d_i(w) &\leq d_i(v) + c_{\pi_{i-1}}(v,w) \\ &= d_i(v) + c(v,w) - \pi_{i-1}(w) + \pi_{i-1}(v) \\ \Rightarrow &- \pi_{i-1}(w) - d_i(w) + \pi_{i-1}(v) + d_i(v) + c(v,w) \geq 0. \end{aligned}$$

Also ist

$$c_{\pi_i}(e) = c(e) - \pi_i(w) + \pi_i(v) = c(e) - (\pi_{i-1}(w) + d_i(w)) + (\pi_{i-1}(v) + d_i(v)) \geq 0.$$

Fall 2: $e \notin E(G_{f_{i-1}})$:
$\Rightarrow$ umgekehrte Kante (w,v) ist in $E(G_{f_{i-1}})$ und liegt auf dem kürzesten Weg W_i entlang dessen augmentiert wurde

$$\begin{aligned} \Rightarrow d_i(v) &= d_i(w) + c_{\pi_{i-1}}(w,v) = d_i(w) + c(w,v) - \pi_{i-1}(v) + \pi_{i-1}(w) \\ \Rightarrow c_{\pi_i}(v,w) &= -c(w,v) - (\pi_{i-1}(w) + d_i(w)) + \pi_{i-1}(v) + d_i(v) = 0. \end{aligned}$$

4. Dies erlaubt Berechnung des kürzesten Weges mit Dijkstra.

5. Dijkstra erfordert (da parallele Kanten möglich) $O(n^2 + m)$ $\Rightarrow$ insgesamt $O(nm + B(n^2 + m))$.

□

Wir geben ein Beispiel.

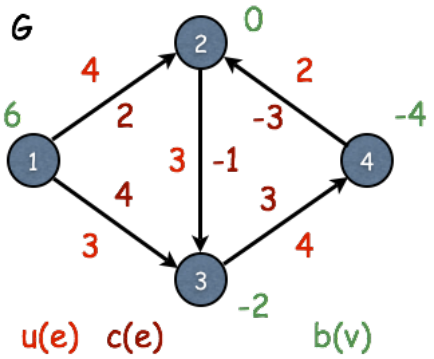


Abbildung 2.20: Beispiel mit negativem Zykel.

2.7 Capacity Scaling Algorithmus

Für große B erhält man bessere Laufzeiten durch Skalieren (Scaling). Dies ist eine wichtige Technik zur Beschleunigung von Algorithmen.
Idee des Skalierens:
Transformiere das Problem so, dass nur Kapazitäten $u(e) = \infty$ auftreten (dies ist möglich, siehe die Transformation in Lemma 2.3). Betrachte dann zunächst nur Wege mit großem Augmentierungswert $\gamma \approx B$ und halbiere iterativ γ. Bei ganzzahligen Daten ist dann $\gamma = 1$

nach $O(\log B)$ Halbierungen und man ist fertig.

Eingabe: (G, u, b, c), c konservativ, $u = \infty$ (allgemeine u über Transformation mit Lemma 2.3).
Ausgabe: Kostenminimaler b-Fluss f (oder Aussage, dass kein b-Fluss existiert).
Methode:
setze $f(e) := 0$ für alle Kanten e
setze $\tilde{b}(v) := b(v)$ für alle Knoten v // müssen dann $\tilde{b}(v) = 0$ erreichen, $\tilde{b}(v)$ drückt Imbalance aus
setze $\gamma := 2^{\lfloor \log B \rfloor}$ mit $B := \max\{1, (1/2)\sum |b(v)|\}$.
Hauptschleife:
while es gibt Knoten v mit $\tilde{b}(v) \neq 0$ **do**
  while es gibt keine Knoten s, t mit $\tilde{b}(s) \geq \gamma$ und $\tilde{b}(t) \leq -\gamma$ und t von s in G_f erreichbar **do**
    if $\gamma = 1$ **then**
      return // es gibt keinen b-Fluss
    end
    else
      setze $\gamma := \gamma/2$
    end
  end
  wähle Knoten s, t mit $\tilde{b}(s) \geq \gamma$ und $\tilde{b}(t) \leq -\gamma$ und t von s in G_f erreichbar
  berechne s, t-Weg W in G_f mit minimalen Kosten
  augmentiere f entlang W um γ // geht, da Kapazitäten $u(e) = \infty$
  setze $\tilde{b}(s) := \tilde{b}(s) - \gamma$, $\tilde{b}(t) := \tilde{b}(t) + \gamma$
end
return: f

Algorithm 2.7.1: Capacity Scaling Algorithmus

Satz 2.22 (Edmonds & Karp 1972). Der Capacity Scaling Algorithmus löst das MCFP mit ganzzahligen Balancen b, konservativem c und $u = \infty$ in $O(n \cdot (n^2 + m) \cdot \log B)$ Zeit.

Beweis. Die Korrektheit folgt sofort aus Satz 2.19.
Zur Laufzeit:
Wir halten folgende Beobachtung fest:

Bemerkung 2.23. Residualkapazitäten sind ∞ auf VK (da $u(e) = \infty$) und Vielfache von γ auf RK (da immer um γ augmentiert und γ sukzessive halbiert wird)

γ-Phase := Schritte im Algorithmus mit gleichem γ. Wir zeigen nun 3 wichtige Eigenschaften:

1. Die Anzahl der Augmentierungen in einer γ—Phase ist höchstens $4n$.
   Beweis: Angenommen, die Aussage gilt nicht. Betrachte eine γ—Phase mit mehr als $4n$ Augmentierungen. Sei f der Fluss zu Beginn dieser γ—Phase und g der Fluss am Ende dieser γ—Phase.
   Beh.: $g - f$ ist ein $\hat{b}$-Fluss (negative Werte zugelassen) in G_f mit $\sum_v |\hat{b}(v)| \geq 8n\gamma$.

Beweis der Behauptung. Beachte, dass $E(G) \subseteq E(G_f)$ (folgt aus Bemerkung 2.23, i.e., $u(e) = \infty$ auf VK). Nun zeigen wir:

$$\sum_v |\hat{b}(v)| \geq 8n\gamma.$$

In jeder Augmentierung in der γ−Phase wird γ entlang eines Weges geschickt.
$\Rightarrow \sum\{\tilde{b}(v)|\tilde{b}(v) > 0\}$ nimmt pro Augmentierung um γ ab und $\sum\{\tilde{b}(v)|\tilde{b}(v) < 0\}$ nimmt pro Augmentierung um γ zu. Betrachte $g - f$ und $\sum |\hat{b}(v)|$ über die Augmentierungen in der γ−Phase. Hier gilt anfangs $g - f = 0$ und $\sum |\hat{b}(v)| = 0$. Am Ende gleich dem jetzigen $g - f$, über das wir argumentieren
$\Rightarrow 2\gamma$ Änderung in $\sum |\hat{b}(v)|$ bzgl. $g - f$ pro Augmentierung $\Rightarrow$ (mehr als $4n$ Augmentierungen) $g - f$ ist ein $\hat{b}$-Fluss in G_f mit $\sum_v |\hat{b}(v)| \geq 8n\gamma$. □

Sei

$$S := \{v \in V(G)|\hat{b}(v) > 0\} \text{ und } S^+ := \{v \in V(G)|\hat{b}(v) \geq 2\gamma\}$$

und

$$T := \{v \in V(G)|\hat{b}(v) < 0\} \text{ und } T^+ := \{v \in V(G)|\hat{b}(v) \leq -2\gamma\}.$$

In der γ−Phase gibt es in G_f keinen Weg von S^+ nach T^+ (sonst gäbe es ihn wegen Bemerkung 2.23 schon zu Beginn der γ−Phase und damit würde die 2γ−Phase länger dauern) $\Rightarrow$ alle $t \in T$, die in G_f erreichbar sind von einem $s \in S^+$, erfüllen $t \in T \setminus T^+$ und daher $\hat{b}(t) > -2\gamma \Rightarrow$ für die Summe $\sum_t \hat{b}(t)$ dieser t gilt

$$\sum_t \hat{b}(t) > \sum_t (-2\gamma) \geq n(-2\gamma).$$

Weil nun $g - f$ ein $\hat{b}$-Fluss in G_f und somit die Balancebedingung gilt, folgt

$$\sum\{\hat{b}(s)|s \in S^+\} \leq \sum_{t \in T \setminus T^+} |\hat{b}(t)| < 2n\gamma.$$

Wir erhalten

$$\begin{aligned}\sum_{v \in V(G)} |\hat{b}(v)| &= 2\sum_{v \in S} |\hat{b}(v)| = 2\sum_{v \in S^+} |\hat{b}(v)| + 2\sum_{v \in S \setminus S^+} |\hat{b}(v)| \\ &< 2(2n\gamma + 2n\gamma) = 8n\gamma, \text{ im Widerspruch zu } \sum_v |\hat{b}(v)| \geq 8n\gamma.\end{aligned}$$

2. wegen sukzessiver Halbierung gibt es höchstens $\lfloor \log B \rfloor + 1$ viele γ−Phasen.

3. eine Moore-Bellman-Ford Berechnung, sonst nur Dijkstra

(1) - (3) $\Rightarrow$ Aufwand = O(Moore-Bellman-Ford) + (#γ-Phasen)·O(# Augmentierungen in γ−Phase)·O(Dijkstra) $= O(nm + (\log B) \cdot n \cdot (n^2 + m)) = O(n(n^2 + m) \log B)$. □

Bemerkung 2.24. Weitere Verbesserung möglich auf streng polynomial (Polynom in n und m) durch geschicktere Wahl der Knoten s, t und Skalierung (Orlin 1993): $O(n \log m)$ Augmentierungen reichen $\Rightarrow O(n \log m(n^2 + m))$.

Kapitel 3
Matchings

3.1 Grundlegende Definitionen

In diesem Kapitel befassen wir uns mit den sogenannten Matchings in einem ungerichteten Graphen $G = (V, E)$. Dazu benötigen wir die folgenden Definitionen:

Definition 3.1. 1. Ein Matching $M \subseteq E(G)$ ist eine Teilmenge der Kantenmenge von G, sodass je zwei Kanten aus M keinen Endpunkt gemeinsam haben.

2. Ein Knoten $v \in V(G)$ heißt dabei überdeckt, falls es eine Kante $e = \{v, w\} \in M$ gibt; w wird dann Partner von v in M genannt. Gibt es so eine Kante e nicht, so bezeichnen wir v als exponierten Knoten (bzgl. M).

3. Ein Matching M heißt nicht erweiterbar, falls für alle Kanten $e \in E(G) - M$ gilt: $M + e$ ist kein Matching. Es lässt sich also keine Kante zu M hinzufügen, sodass danach immer noch ein Matching vorliegt.

4. Ein Matching M heißt maximal, falls für alle Matchings M' von G gilt: $|M| \geq |M'|$. Es existiert also kein Matching, welches mehr Kanten besitzt. Die Kardinalität eines maximalen Matchings bezeichnen wir mit $\nu(G)$.

5. Schließlich heißt ein Matching M perfekt, falls jeder Knoten in G mit einer Kante aus M inzident ist.

Abbildung 3.1 soll die definierten Begriffe veranschaulichen. Dabei sind die Matchingkanten rot eingezeichnet. Im linken Bild ist M ein nicht erweiterbares Matching, jedoch nicht maximal und nicht perfekt. Dies sieht man leicht daran, dass nur die Knoten a und f exponiert sind und keine Kante zwischen den beiden Knoten existiert. Dagegen ist die Aufteilung $M' = \{ab, ce, df\}$ (wie im rechten Teil der Abbildung) ein perfektes Matching, welches eine Kante mehr als M besitzt.

Wir halten fest, dass die folgenden Implikationen gelten (die Rückrichtungen gelten im Allgemeinen nicht):

$$M \text{ ist perfekt } \Rightarrow M \text{ ist maximal } \Rightarrow M \text{ ist nicht erweiterbar}$$

Neben dem Auffinden von maximalen Matchings ist man manchmal auch an Matchings

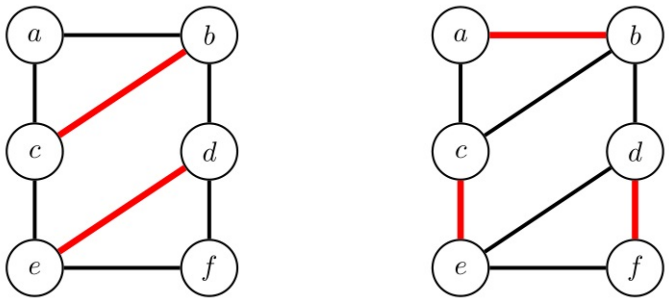


Abbildung 3.1: Veranschaulichung der grundlegenden Begriffe zu Matchings (links: nicht erweiterbares Matching, rechts: perfektes Matching)

mit maximalem Gewicht interessiert. Dazu betrachtet man einen ungerichteten Graphen, der zusätzlich Kantengewichte $c(e) \geq 0$ für alle $e \in E(G)$ besitzt. Dies will man zum Beispiel beim Chinese Postman Problem anwenden. Dabei ist man an einer Tour, die jede Kante eines zusammenhängenden Graphen mindestens einmal durchläuft und minimale Kosten unter allen solchen Touren hat, interessiert (z.B. Briefträger, Müllabfuhr, ...).
Ist G eulersch, so ist natürlich jede Euler Tour eine optimale Lösung. Ist G dagegen nicht eulersch, so müssen manche Kanten mehrfach durchlaufen werden. Diese zusätzlichen Kanten bilden dabei Wege zwischen Knoten mit ungeradem Grad. Wir wollen also die Länge dieser Zusatzwege minimieren. Dies ist mit folgendem Algorithmus möglich:

Eingabe: Ungerichteter zsh. Graph G, Kantengewichte $c(e) \geq 0$.
Ausgabe: Eine Tour minimaler Länge, die jede Kante mindestens einmal durchläuft.
Methode:
if es gibt keine Knoten mit ungeradem Grad **then**
  return Euler Tour von G.
end

1. Ermittle alle Knoten mit ungeradem Grad (davon gibt es eine gerade Anzahl)
2. Betrachte vollständigen Graphen H auf diesen Knoten
3. Kante $e = \{u, v\} \in E(H)$ erhält $c'(e) :=$Länge eines kürzesten Weges in G zwischen u und v (kann wegen $c(e) \geq 0$ berechnet werden)
4. Berechne ein gewichtsmaximales perfektes Matching M in H bzgl. Kantengewichten $-c'(e) + N$ (dabei ist N eine große Konstante)
5. Für jede Kante $e \in M$ nehmen wir einen zugehörigen Weg P mit Länge $c'(e)$ und fügen die Kanten aus P zu G hinzu (Mehrfachkanten möglich; der resultierende Graph G' ist Eulersch mit minimalen zusätzlichen Kosten)

return Euler Tour von G'.

Algorithm 3.1.1: Chinese Postman Algorithmus

Dabei ist allerdings nicht klar, wie wir Punkt 4. algorithmisch bestimmen können. Wir wollen trotzdem ein kleines Beispiel zur Veranschaulichung machen und dabei das gewichtsmaximale Matching per Hinsehen bestimmen.

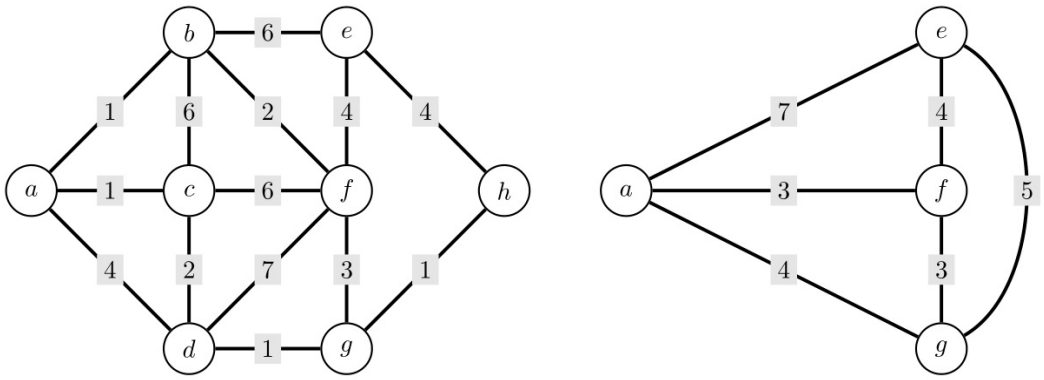


Abbildung 3.2: Veranschaulichung des Chinese Postman Algorithmus

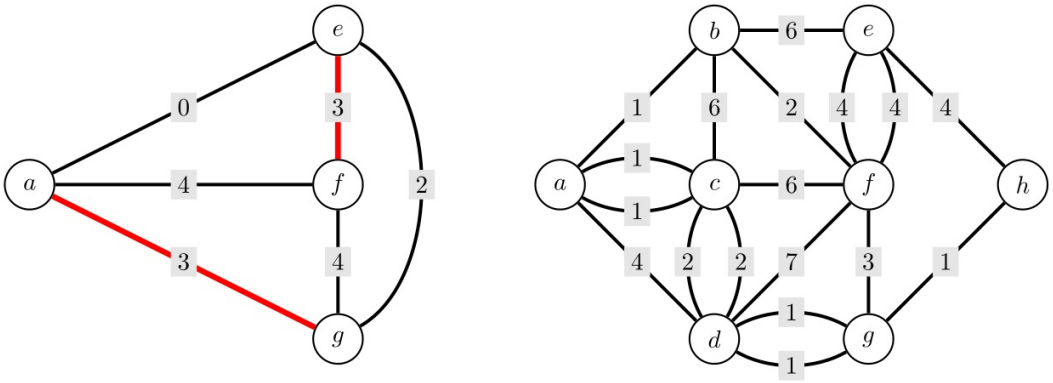


Abbildung 3.3: Gewichtsmaximales perfektes Matching in $(H, -c' + 7)$ und der Graph G'

Gegeben sei dazu der Graph aus Abbildung 3.2 (links). Im rechten Teil der Abbildung ist der Graph H mit zugehöriger Gewichtsfunktion c' abgebildet (die Knoten a, e, f und g haben ungeraden Grad). Die Kosten c' der Kante ag erhalten wir beispielsweise durch den Weg $a-c-d-g$. Die Kosten der anderen kürzesten Wege haben wir auch durch Hinsehen bestimmt. Wählen wir z.B. $N = 7$, so erhalten wir in H ein gewichtsmaximales Matching bzgl. $-c' + N$ wie in Abbildung 3.3 (links); dies entspricht einem gewichtsminimalen perfekten Matching in (H, c'). Daraus resultiert dann der im rechten Teil der Abbildung dargestellte Graph G'; wir haben dazu die Matchingkante ag in den Weg $a - c - d - g$ rücktransformiert, die Kante ef ist gleichzeitig schon der kürzeste Weg von e nach f. Wir durchlaufen insgesamt also vier Kanten mit einem Gesamtgewicht von 8 doppelt.

Eine weitere Kennzahl eines Graphen ist folgendermaßen definiert:

Definition 3.2. 1. Eine Teilmenge $D \subseteq V$ heißt überdeckende Knotenmenge bzw. Knotenüberdeckung von G (vertex cover), falls gilt: $e \cap D \neq \emptyset$ für jede Kante $e \in E(G)$. Das heißt, dass mindestens ein Endknoten jeder Kante in D enthalten ist.

2. Ähnlich zu Definition 3.1 heißt D nicht verkleinerbar, falls $D - v$ keine Knotenüber-

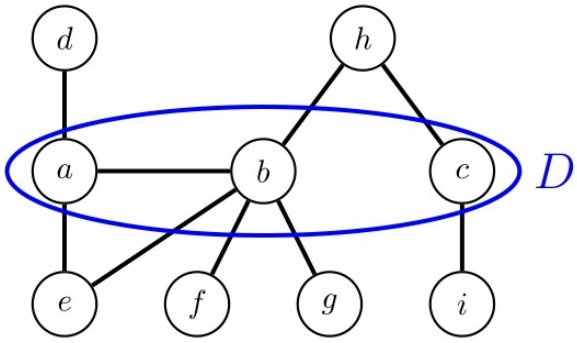


Abbildung 3.4: Beispiel einer minimalen Knotenüberdeckung D eines Graphen

deckung von G für jede Wahl von $v \in D$ ist.

3. Ist $|D| \leq |C|$ für jede überdeckende Knotenmenge C von G, so heißt D minimale Knotenüberdeckung. Die Knotenüberdeckungszahl eines Graphen G ist die Mächtigkeit einer minimalen Knotenüberdeckung. Wir bezeichnen diese mit $\tau(G)$.

Ein Beispiel einer minimalen Knotenüberdeckung findet sich in Abbildung 3.4. Wir erhalten sofort folgendes Resultat.

Satz 3.3. Ist G ein Graph, so gilt $\nu(G) \leq \tau(G)$.

Beweis. Sei $D \subseteq V$ eine Knotenüberdeckung und $M \subseteq E$ eine Menge von Kanten mit $|M| > |D|$. Nach dem Taubenschlagprinzip gibt es dann mindestens einen Knoten $v \in D$, der mit mindestens zwei Kanten aus M inzident ist. Daher kann M kein Matching sein. Damit folgt für jedes Matching von G

$$\nu(G) = \max\{|M| \mid M \text{ ist Matching von } G\} \leq |D|.$$

Dies gilt für alle Knotenüberdeckungen D von G, also gilt wie behauptet

$$\nu(G) \leq \min\{|D| \mid D \text{ ist Knotenüberdeckung von } G\} = \tau(G).$$

□

Übung 3.4. Ist G ein Graph, so gilt $\tau(G) \leq 2 \cdot \nu(G)$.

Wir zeigen zuerst, dass die Endknoten eines jeden nicht erweiterbaren Matchings M eine Knotenüberdeckung sind. Nehmen wir an, dies gilt nicht. Dann gibt es eine Kante, deren Endknoten beide nicht von M überdeckt sind. Damit könnten wir diese Kante zu M hinzufügen, um ein größeres Matching zu erhalten, ein Widerspruch zu unserer Annahme, dass M nicht erweiterbar ist.
Nun gilt also

$$\tau(G) \leq 2 \cdot |M| \leq 2 \cdot \nu(G),$$

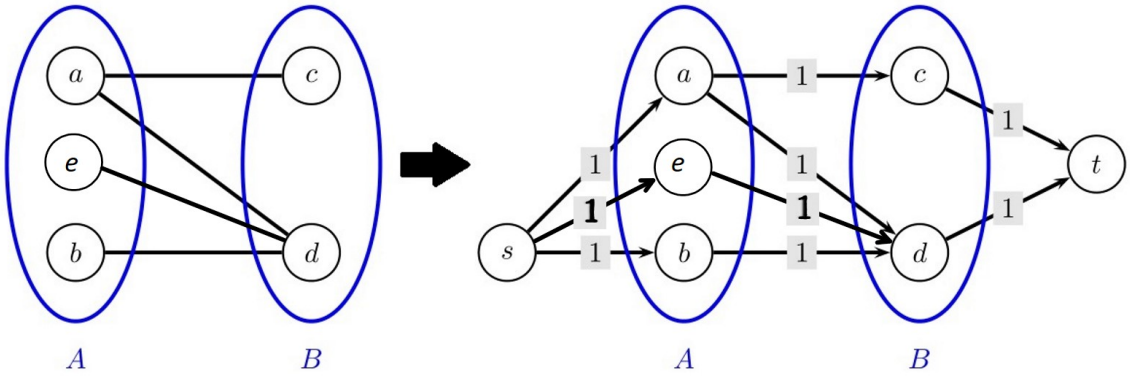


Abbildung 3.5: Transformation im Beweis zum Satz von König

was zu zeigen war. Die erste Ungleichung erhalten wir, da $2 \cdot |M|$ genau die Anzahl der Knoten des nicht erweiterbaren Matchings angibt, welche nach obiger Argumentation eine (eventuell nicht minimale) Knotenüberdeckung bilden. Außerdem gilt sicherlich, dass jedes maximale Matching mindestens so viele Kanten enthält wie das vorhandene nicht erweiterbare Matching.

3.2 Matchings in bipartiten Graphen

Wie wir (maximale) Matchings bestimmen, ist bisher immer noch nicht klar. Wir werden zu Beginn Aussagen über Matchings in bipartiten Graphen $G = (A \cup B, E)$ betrachten und danach schauen, mit welchen Anpassungen sich diese Aussagen auf allgemeine Graphen übertragen lassen.

Im Fall von bipartiten Graphen können wir Satz 3.3 noch konkretisieren.

Satz 3.5 (Satz von König). In jedem bipartiten Graphen G gilt $\nu(G) = \tau(G)$.

Beweis. Wir wollen den Max-Flow-Min-Cut Satz anwenden und transformieren dazu G in ein Flussnetzwerk G' (vgl. Abbildung 3.5), indem wir jede Kante $e = \{a, b\}$ durch die gerichtete Version $e = (a, b)$ ersetzen sowie eine Quelle s mit Kanten (s, v) für alle Knoten $v \in A$ und eine Senke t mit Kanten (w, t) für alle $w \in B$ einführen. Dabei erhalten alle Kanten eine Kapazität von 1.
Da alle Kapazitäten ganzzahlig sind, betrachten wir einen ganzzahligen maximalen s,t-Fluss f in G', d.h. $f(e) \in \{0, 1\}$ für alle $e \in E(G)$. Offensichtlich bilden Kanten von A nach B mit $f(e) = 1$ ein Matching M in G (kein Knoten kann mehrere ein- oder ausgehende Kanten haben wegen der Wahl der Kantenkapazitäten).
Sei nun $S := \{s\} \cup \{v \in V(G) \mid v \text{ von } s \text{ in } G_f \text{ erreichbar}\}$ (vgl. Abbildung 3.6). Wie auch schon im Kapitel zu maximalen Flüssen gesehen, bildet S einen Schnitt minimaler Kapazität. Wir behaupten nun, dass $D := (A - S) \cup (B \cap S)$ eine Knotenüberdeckung in G ist.

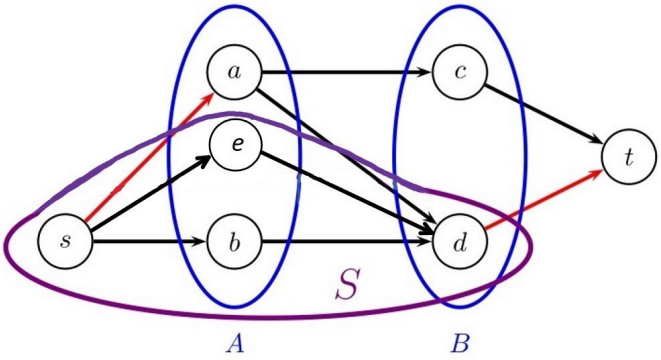


Abbildung 3.6: Minimaler Schnitt S im Beweis zum Satz von König

Dazu halten wir fest, dass Kanten (s, v) mit $v \in A - S$ gesättigt sind, da diese Knoten nach Definition von S im Residualgraph nicht mehr von s aus erreichbar sind. Desweiteren sind auch alle Kanten (w, t) mit $w \in B \cap S$ gesättigt, da sonst t im Residualgraph noch von s aus erreichbar wäre, ein Widerspruch dazu, dass f ein maximaler Fluss ist (die gesättigten Kanten wurden dabei rot eingezeichnet). Nehmen wir per Widerspruch an, dass eine Kante $e = \{a, b\}$ mit $a \in A$ und $b \in B$ keinen Endpunkt in D hat. Das heißt nichts anderes als $a \in S$ und $b \notin S$. Wir unterscheiden nun, ob die Kante e gesättigt ist oder nicht.

Fall 1: $f(a, b) = 1$. Da $a \in S$ gilt, ist a von s aus in G_f erreichbar. Dies ist auf zwei Weisen möglich, die wir nochmals unterscheiden müssen.

Fall 1a: $f(a, b) = 1$ und a ist direkt über die Kante (s, a) erreichbar. Das heißt, dass $f(s, a) = 0$ ist. Dies ist ein Widerspruch zur Flusserhaltung in a.

Fall 1b: $f(a, b) = 1$ und a ist über eine Rückwärtskante (b', a) in G_f erreichbar. Das heißt, dass $f(a, b') = 1$ ist. Da nach Annahme auch $f(a, b) = 1$ ist, fließen aus a mindestens zwei Flusseinheiten raus, es geht aber in G' nur eine Kante mit Kapazität 1 in a ein. Deshalb erhalten wir wieder einen Widerspruch zur Flusserhaltung in a.

Fall 2: $f(a, b) = 0$. Damit ist die Kante $e = (a, b)$ in G_f enthalten. Da $a \in S$ ist, ist nun auch b von s aus in G_f erreichbar, was $b \in S$ implizieren würde, ein Widerspruch.

Die Kapazität des Schnitts S ist $c(S) = |A - S| + |B \cap S|$, da alle Kanten die Kapazität 1 besitzen (dies entspricht also der Anzahl der gesättigten Kanten, die in Abbildung 3.6 rot eingezeichnet sind).
Da D also tatsächlich eine Knotenüberdeckung ist, gilt

$$|D| = c(S) = \nu(f) = |M| = \nu(G).$$

Damit erhalten wir $\nu(G) \geq \tau(G)$, da D nicht notwendigerweise eine minimale Knotenüberdeckung ist. Die geforderte Gleichheit erhalten wir aber mit Satz 3.3, da zusätzlich $\nu(G) \leq \tau(G)$ die Aussage unseres Satzes impliziert. □

Der Satz von König lässt sich auch noch in einer äquivalenten Version formulieren.

Satz 3.6 (Satz von Hall (1935)). Sei $G = (A \cup B, E)$ ein bipartiter Graph. Dann sind folgende Aussagen äquivalent:

1. G hat ein Matching M das A überdeckt (d.h. jeder Knoten aus A ist mit einer Kante aus M inzident).
2. Zu jeder Teilmenge $X \subseteq A$ gibt es genügend viele „Partner“ in B, d.h. $|\Gamma(X)| \geq |X|$ für alle $X \subseteq A$ mit $\Gamma(X) := \{v \in B \mid \exists u \in X : \{u, v\} \in E(G)\}$.

Der Spezialfall $|A| = |B|$ ist auch als Heiratssatz von Hall bekannt.

Beweis. Die Richtung 1. $\Rightarrow$ 2. ist klar (Widerspruchsbeweis).
Für 2. $\Rightarrow$ 1. nehmen wir an, dass es kein Matching gibt, das A überdeckt, d.h. $\nu(G) < |A|$. Aus Satz 3.5 folgt damit, dass auch $\tau(G) < |A|$ ist. Seien nun $A' \subseteq A$ und $B' \subseteq B$ so, dass $D := A' \cup B'$ eine minimale Knotenüberdeckung ist, d.h.

$$\left|A'\right| + \left|B'\right| = \left|A' \cup B'\right| = \tau(G) < |A|\,. \qquad (*)$$

Da D eine Knotenüberdeckung ist, kann es keine Kanten von $A - A'$ nach $B - B'$ geben und impliziert damit $\Gamma(A - A') \subseteq B'$. Damit erhalten wir insgesamt

$$\left|\Gamma(A - A')\right| \leq \left|B'\right| \overset{(*)}{<} |A| - \left|A'\right| = \left|A - A'\right|,$$

also einen Widerspruch zu $|\Gamma(A - A')| \geq |A - A'|$. □

Wir können außerdem festhalten, dass offensichtlich

$$\nu(G) \leq \min\{|A|, |B|\}$$

gilt, da einerseits gelten muss, dass jeder Knoten mit maximal einer Matchingkante inzident ist und andererseits nur Kanten von A nach B verlaufen.
Aus der Transformation im Beweis zum Satz von König erhalten wir zusätzlich folgendes Resultat.

Satz 3.7. Ein maximales Matching in einem bipartiten Graphen kann in $O(nm)$ Zeit berechnet werden.

Beweis. Wir verwenden z.B. Ford-Fulkerson im transformierten Graph. Der maximale Flusswert entspricht $\nu(G)$. Da wir jede Augmentierung in $O(m)$ durchführen können und maximal $|A| \leq n$ oft augmentieren, erhalten wir wie gewünscht eine Laufzeit von $O(nm)$. □

Bemerkung 3.8. Durch Augmentierung entlang kürzester Wege (Hopcroft & Karp) lässt sich die Laufzeit auf $O(\sqrt{n}m)$ verbessern (ohne Beweis).

Als Spezialfall des Heiratssatzes erhalten wir das folgende Korollar, welches eine hinreichende Bedingung an die Existenz eines perfekten Matchings angibt.

Korollar 3.9. Sei $G = (A \cup B, E)$ ein regulärer bipartiter Graph. Dann besitzt G ein perfektes Matching.

Beweis. Da G regulär ist, muss $|A| = |B|$ gelten. Es reicht also aus, Bedingung 2. aus Satz 3.6 nachzuprüfen (A-überdeckend impliziert dann auch B-überdeckend). Sei dazu r der Grad aller Knoten von G. Für $X \subseteq A$ mit $|X| = k$ gibt es genau $k \cdot r$ Kanten von X nach B. Da auch jeder Knoten in B mit genau r Kanten inzident ist, müssen diese $k \cdot r$ Kanten mit mindestens k verschiedenen Knoten aus B inzident sein, d.h. $|X| \leq |\Gamma(X)|$. □

Abschließend wollen wir für bipartite Graphen folgenden interessanten Zusammenhang festhalten.

Satz 3.10. Sei $G = (A \cup B, E)$ ein bipartiter Graph mit $X \subseteq A$ und $Y \subseteq B$. Gibt es nun in G ein X-überdeckendes Matching M_X und ein Y-überdeckendes Matching M_Y, so gibt es auch ein Matching M, welches $X \cup Y$ überdeckt.

Beweis. Wir betrachten den (bipartiten) Teilgraph $G' = (W, E')$ von G, der durch die Kanten in $M_X \cup M_Y$ festgelegt ist. Dabei ist offensichtlich $X \cup Y \subseteq W$ und jeder Knoten in G' hat entweder Grad 1 oder 2 (je nachdem, ob ein Knoten in genau einem der beiden Matchings oder beiden enthalten ist). Hat ein Knoten Grad 2, so ist dieser mit einer Matchingkante aus M_X und einer aus M_Y inzident. Daher sind die Zusammenhangskomponenten von G' entweder Kreise gerader Länge (mit abwechselnd Kanten aus M_X bzw. M_Y), oder alternierende Pfade. Aus diesen Einsichten werden wir nun ein Matching M konstruieren, welches $X \cup Y$ überdeckt.
Betrachten wir eine beliebige Zusammenhangskomponente C von G', so unterscheiden wir drei Fälle, vgl. Abbildung 3.7:

Fall 1: C ist ein (alternierender) Kreis. Dann fügen wir alle Kanten aus $C \cap M_X$ zu M hinzu, also genau die Kanten im Kreis, die in M_X enthalten sind. Damit sind alle Knoten in C überdeckt.

Fall 2: C ist ein (alternierender) Pfad ungerader Länge (P_o). Dann gehören die erste und die letzte Kante des Pfads zum selben Matching $M' \in \{M_X, M_Y\}$. Auch hier fügen wir alle Kanten zu M hinzu, die in M' enthalten sind.

Fall 3: C ist ein (alternierender) Pfad gerader Länge (P_e). Da G bipartit ist, müssen die beiden Endknoten zur gleichen Partition gehören, o.B.d.A. zu A. Da C ein alternierender Pfad ist, kann einer der beiden Endknoten nicht von M_X überdeckt werden (d.h. dieser Knoten ist nicht in X enthalten), da genau eine der beiden Endkanten von C zu M_X gehört. Das bedeutet, dass wir diesen Knoten mit M gar nicht überdecken müssen. Daher reicht es auch hier aus, die Kanten aus M_X zu M hinzuzufügen.

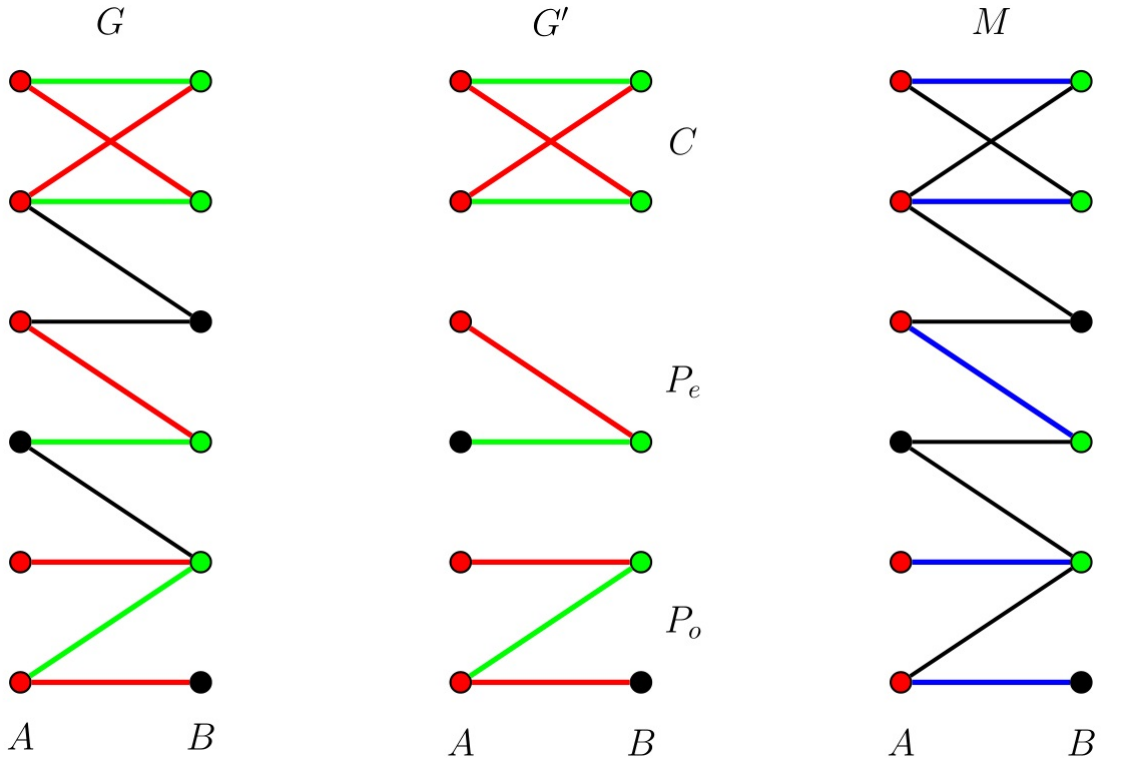


Abbildung 3.7: Situation im Beweis zu Satz 3.10

Offensichtlich ist M nun ein Matching, welches alle Knoten aus $X \cup Y$ überdeckt (kein Knoten ist mit mind. zwei Kanten inzident, da wir die Zusammenhangskomponenten betrachtet haben und diese disjunkt sind).

Im Beispiel (Abbildung 3.7 (links)) haben wir in rot eine Teilmenge $X \subseteq A$ und in grün eine Teilmenge $Y \subseteq B$ ausgewählt und in der jeweiligen Farbe ein X- bzw. Y-überdeckendes Matching eingezeichnet. In der Mitte ist der resultierende Graph G' abgebildet, der aus einem Kreis gerader Länge (C), einem Pfad ungerader Länge (P_o) und einem Pfad gerader Länge (P_e) besteht. Wir wählen in C die grünen Kanten; in P_e und P_o müssen wir jeweils die roten Kanten wählen. Daraus erhalten wir das blaue Matching, welches in der Tat alle roten und grünen Knoten überdeckt. □

3.3 Matchings in beliebigen Graphen

Wir haben im letzten Abschnitt gesehen, dass sich ein maximales Matching in bipartiten Graphen effizient bestimmen lässt. Bevor wir zum Algorithmus von Edmonds kommen, der ein maximales Matching in beliebigen Graphen bestimmt, wollen wir eine strukturelle Aussage über maximale Matchings treffen. Dazu benötigen wir die folgenden Begriffe.

Definition 3.11. 1. Eine ungerade Komponente eines Graphen G ist eine Zusammenhangskomponente von G, die eine ungerade Anzahl an Knoten besitzt.

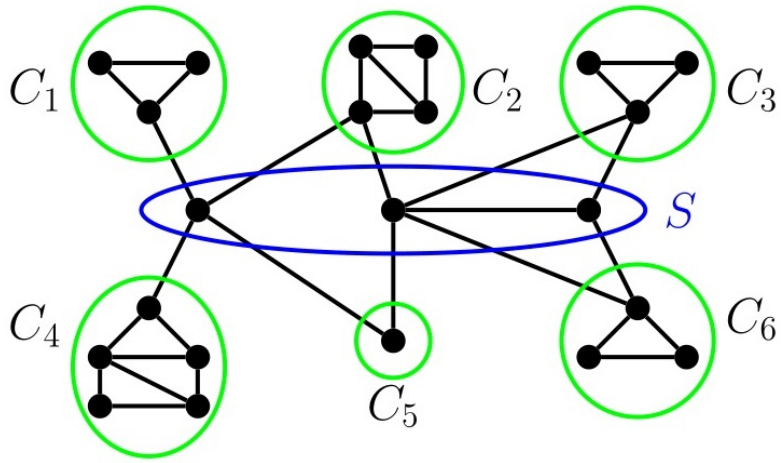


Abbildung 3.8: Beispiel zum Defizit einer Menge S

2. Für eine Menge $S \subseteq V(G)$ bezeichne $o(S)$ die Anzahl ungerader Komponenten von $G - S$.

3. Das Defizit von S ist $d(S) := o(S) - |S|$.

In Abbildung 3.8 besteht S aus drei Knoten. Die Zusammenhangskomponenten C_1, C_3, C_4, C_5 und C_6 sind jeweils ungerade, C_2 ist gerade. Deshalb gilt hier $d(S) = 5 - 3 = 2$.
Wir erhalten folgendes Resultat.

Satz 3.12 (Berge-Tutte Formel). Sei $G = (V, E)$ ein Graph mit n Knoten und sei M ein maximales Matching von G. Dann gibt es genau

$$d = \max\{d(S) \mid S \subseteq V(G)\}$$

exponierte Knoten, bzw. anders ausgedrückt gilt

$$\nu(G) = \frac{1}{2}(n - d) = \frac{1}{2}(n - \max\{o(S) - |S| \mid S \subseteq V(G)\}).$$

Beweis. Sei M ein Matching von G, welches genau e Knoten exponiert lässt und sei $S \subseteq V(G)$ beliebig. Offensichtlich muss jede ungerade Komponente von $G - S$ mindestens einen exponierten Knoten enthalten, außer einer der enthaltenen Knoten ist Partner eines Knotens in S. Deshalb kann die Anzahl ungerader Komponenten in $G - S$ nicht größer sein als die Summe aus der Anzahl aller Knoten in S und der Zahl der exponierten Knoten, also gilt $d(S) = o(S) - |S| \leq e$. Da dies für alle Wahlen von S gilt, muss es schon mal mindestens $\max\{d(S) \mid S \subseteq V(G)\}$ exponierte Knoten geben.
Per Induktion nach n zeigen wir nun, dass es tatsächlich ein Matching gibt, welches nur

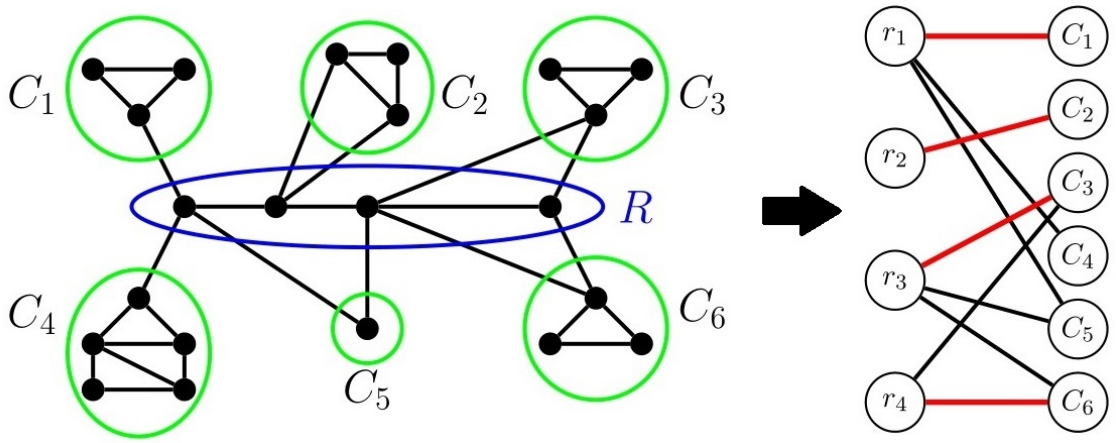


Abbildung 3.9: Inklusionsmaximale Menge R sowie bipartiter Graph B

d Knoten exponiert lässt. Für $n = 0$ ist dies trivialerweise erfüllt. Wir stellen fest, dass

$$d(S) = o(S) - |S| \equiv n \pmod 2 \quad \text{für alle } S \subseteq V(G) \tag{3.1}$$

gilt. Dies liegt daran, dass n genau die Summe von $|S|$ und der Zahl aller Knoten in den Zusammenhangskomponenten von $G-S$ ist, also ist $o(S)+|S| \equiv n \pmod 2$ (die Anzahl der Knoten in geraden Zusammenhangskomponenten ist kongruent zu $0 \pmod 2$; außerdem können wir Plus durch Minus ersetzen).

Wir wählen nun eine inklusionsmaximale Menge R unter allen Mengen S, die das größte Defizit besitzen, d.h. die $d(S) = d$ erfüllen, und behaupten, dass dann jede Zusammenhangskomponente von $G - R$ eine ungerade Zahl an Knoten enthält (vgl. Abbildung 3.9, links). Nehmen wir per Widerspruch an, dass C eine gerade Komponente in $G - R$ ist. Dann wählen wir ein Blatt b eines beliebigen Spannbaums von C und fügen dieses zu R hinzu. Dann ist das Defizit immer noch d, jedoch ist die Inklusionsmaximalität von R verletzt.

Sei nun R^* die Menge aller (notwendigerweise ungeraden) Komponenten von $H := G - R$. Wir betrachten den bipartiten Graphen B mit Knotenmenge $R \cup R^*$ (jeder Knoten von R^* repräsentiert eine ungerade Komponente, vgl. Abbildung 3.9, rechts). Ein Knoten $r \in R$ und eine Komponente $C \in R^*$ sind genau dann adjazent, wenn es eine Kante $\{r, c\}$ in G gibt mit $c \in C$. Jetzt zeigen wir, dass in B ein R-überdeckendes Matching existiert indem wir Bedingung 2. aus dem Satz von Hall nachweisen. Sei dazu also $X \subseteq R$ und sei $T := \Gamma(X)$ die Menge der Nachbarn von X in B, d.h. die Menge aller Komponenten von H, die zu (mindestens) einem Knoten in X benachbart sind. Dann ergeben die Knoten in $R^* - T$ wieder ungerade Komponenten von $G - (R - X)$, da keine Komponente in $R^* - T$ einen Nachbar in X besitzt. Zusätzlich könnten noch mehr ungerade Komponenten existieren, sei diese Zahl gleich $k \geq 0$ (z.B. wenn in Abbildung 3.9 die beiden rechten Knoten von R als X gewählt werden, bilden diese beiden Knoten zusammen mit C_3, C_5 und C_6 eine

weitere ungerade Komponente in $G - (R - X)$ mit 9 Knoten). Also ist

$$d(R - X) \overset{\text{Def.}}{=} |R^* - T| + k - |R - X| \overset{\text{R hat maximales d}}{\leq} d = d(R) \overset{\text{Def.}}{=} |R^*| - |R|,$$

anders ausgedrückt also insbesondere

$$|R| - |R - X| + k \leq |R^*| - |R^* - T|.$$

Mit $T = \Gamma(X)$ folgt wie gewünscht $|X| \leq |X| + k \leq |\Gamma(X)|$, sodass tatsächlich ein R-überdeckendes Matching in B existiert.
Wir können daher jeden Knoten $y \in R$ mit einem Knoten x_y in $G - R$ assoziieren, sodass yx_y eine Kante in G ist und die Knoten x_y paarweise zu verschiedenen Zusammenhangskomponenten von $G - R$ gehören. Daraus erhalten wir ein Matching M_R von G, welches R überdeckt und genau $o(R) - |R| = d$ (ungerade) Komponenten von $G - R$ nicht erreicht. Nun reicht es zu zeigen, dass es für jede Komponente C in $G - R$ und jeden Knoten $x \in C$ ein perfektes Matching M_C im Graphen $G[C - x]$ gibt. Denn dann bildet die Vereinigung von M_R und all den Matchings M_C ein Matching von G, welches genau d Knoten exponiert lässt, für den Fall, dass wir $x = x_y$ immer dann wählen, wenn eine der Komponenten C einen dieser speziellen Knoten enthält, siehe Abbildung 3.10.
Nehmen wir per Widerspruch an, dass es eine Komponente C gibt, in der $G[C-x]$ kein perfektes Matching enthält. Dann gibt es nach Induktionsvoraussetzung eine Menge $Y \subseteq C-x$ mit $o_{C-x}(Y) > |Y|$ (da $d > 0$ ist). Dabei bezeichnet $o_{C-x}(Y)$ die Zahl der ungeraden Komponenten in $G[C - x] - Y$. Da C eine ungerade Komponente ist (wegen der Inklusionsmaximalität von R, was oben schon gezeigt wurde), ist $|C - x|$ gerade. Damit gilt dann nicht nur $o_{C-x}(Y) > |Y|$, sondern wegen (3.1) sofort

$$o_{C-x}(Y) \geq |Y| + 2. \tag{3.2}$$

Da R, Y und x paarweise disjunkt sind, gilt

$$\begin{aligned} o(R \cup Y \cup x) &\overset{(1)}{=} o(R) - 1 + o_C(Y \cup x) = \\ &\overset{(2)}{=} |R| + d - 1 + o_{C-x}(Y) \geq \\ &\overset{(3)}{\geq} |R| + d - 1 + |Y| + 2 = \\ &\overset{(4)}{=} |R \cup Y \cup x| + d. \end{aligned}$$

Dabei haben wir in (1) verwendet, dass wir die Komponente C gesondert betrachten (deshalb haben wir sicherlich noch $o(R) - 1$ viele ungerade Komponenten und eben noch diejenigen, die durch das Weglassen von $Y \cup x$ entstehen). Nach Annahme ist in (2) $o(R) = |R| + d$; in (3) setzen wir obige Abschätzung (3.2) ein. Schließlich verwenden wir in (4), dass $R \cup Y \cup x$ genau $|R| + |Y| + 1$ viele Knoten enthält.
Damit ist aber $R \cup Y \cup x$ auch eine Menge mit Defizit d, ein Widerspruch zur Maximalität von R, was unseren Beweis komplettiert. □

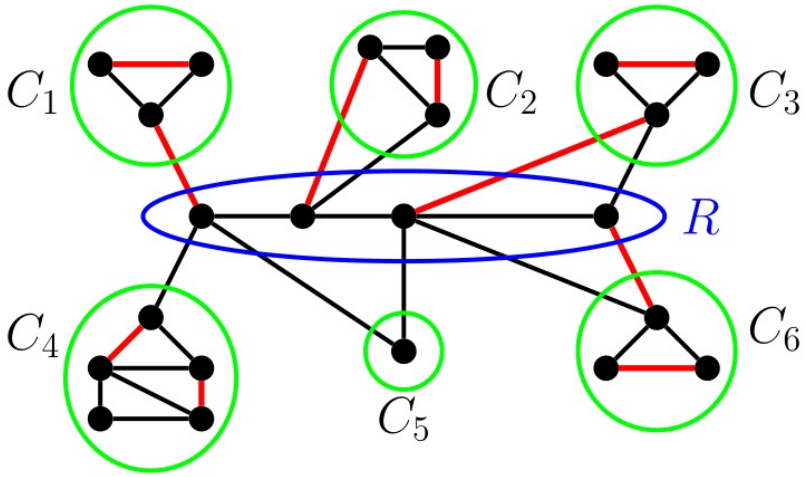


Abbildung 3.10: Maximales Matching in G mit zwei exponierten Knoten

Korollar 3.13. Ein Graph G hat genau dann ein perfektes Matching, wenn $o(S) \leq |S|$ für alle $S \subseteq V(G)$ gilt.

Beweis. Dies folgt direkt aus Satz 3.12 mit $d = 0$. □

Wir geben hier noch einen kleinen Ausblick auf eine interessante Eigenschaft: In einem Graphen gibt es im Allgemeinen eine Teilmenge von Knoten, die in jedem maximalen Matching enthalten ist, aber auch eine Teilmenge, deren Knoten nur in manchen maximalen Matchings enthalten sind. Ist ein Knoten in jedem maximalen Matching enthalten, so nennen wir ihn stets überdeckt.

Definition 3.14. Sei $G = (V, E)$ ein Graph. Wir bezeichnen mit $B(V)$ die Menge aller Knoten, die stets überdeckt sind (d.h. diese Knoten sind in jedem maximalen Matching überdeckt) und mit $D(V)$ die Menge aller Knoten von G, die nicht stets überdeckt sind, d.h. $D(V) = V - B(V)$. Sei dann $A(V)$ die Menge der Nachbarknoten von $D(V)$, die in $V - D(V)$ liegen. Schließlich bezeichnen wir mit $C(V) := V - (D(V) \cup A(V))$ die Menge der restlichen Knoten von G. Dann heißt das Tripel $(A(V), C(V), D(V))$ die Gallai-Edmonds-Zerlegung von G.

Der folgende Satz liefert nun eine Aussage, für welche Menge in der Berge-Tutte-Formel tatsächlich das Maximum angenommen wird:

Satz 3.15. Sei $G = (V, E)$ ein Graph und $(A(V), C(V), D(V))$ dessen Gallai-Edmonds-Zerlegung. Dann gilt

$$\nu(G) = \frac{1}{2}(n - o(A(V)) + |A(V)|),$$

das heißt $A(V)$ bildet eine Menge mit maximalem Defizit.

Für den Beweis dieses Satzes und auch der beiden nachfolgenden Sätze verweisen wir auf das Buch Graphs, Networks and Algorithms von Dieter Jungnickel.

Beachte, dass es im Allgemeinen mehrere Mengen mit maximalem Defizit gibt, die Gallai-Edmonds-Zerlegung ist hingegen eindeutig.
Es bleibt die Frage, ob bzw. wie man die Gallai-Edmonds-Zerlegung überhaupt effizient bestimmen kann – es kann ja sehr viele maximale Matchings eines Graphen geben. Wir geben folgenden Satz an:

Satz 3.16. Sei $G = (V, E)$ ein Graph, $(A(V), C(V), D(V))$ dessen Gallai-Edmonds-Zerlegung und M ein beliebiges maximales Matching von G. Sei dann X die Menge aller M-exponierten Knoten. Dann gilt:

1. $D(V)$ ist die Menge aller Knoten, die von einem Knoten in X durch einen alternierenden Pfad gerader Länge erreicht werden können (insb. gilt $X \subseteq D(V)$).
2. $A(V)$ ist die Menge aller Knoten, die von einem Knoten in X durch einen alternierenden Pfad ungerader Länge erreicht werden können, aber nicht durch einen alternierenden Pfad gerader Länge.
3. $C(V)$ ist die Menge aller restlichen Knoten.

Satz 3.17. Bei der Ausführung von Edmonds Matchingalgorithmus (kommt später noch) kann man die Gallai-Edmonds-Zerlegung eines Graphen erhalten.

Aus diesem Satz folgt nun, dass die Gallai-Edmonds-Zerlegung eines Graphen effizient bestimmt werden kann.

Für das Beispiel in Abbildung 3.10 erhalten wir (ausschließlich durch überlegen)

$$\begin{aligned} D(V) &= \{v \in V \mid v \in \{C_1, C_3, C_4, C_5, C_6\}\}, \\ A(V) &= \{v \in R \mid \nexists \{v, w\} \text{ mit } w \in C_2\}, \\ C(V) &= \{v \in V \mid v \in C_2 \text{ oder } v \text{ ist adjazent zu einem Knoten in } C_2\}. \end{aligned}$$

Wir werden nun analog zu augmentierenden Wegen für Flüsse augmentierende Wege für Matchings einführen (vgl. Abbildung 3.11).

Definition 3.18. Sei M ein Matching in einem Graphen G.

1. Ein M-alternierender Weg in G ist ein elementarer Weg P in G für den $E(P) - M$ ein Matching ist, d.h. P enthält alternierend Matching und Nicht-Matching Kanten.
2. Ein M-alternierender Weg P heißt M-augmentierend, falls beide Endpunkte von P exponiert sind. Dann ist notwendigerweise $|E(P)|$ ungerade.

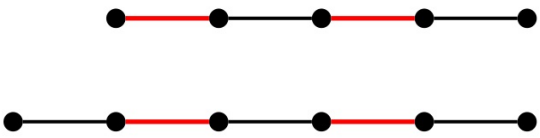


Abbildung 3.11: M-alternierender Weg (oben) und M-augmentierender Weg (unten)

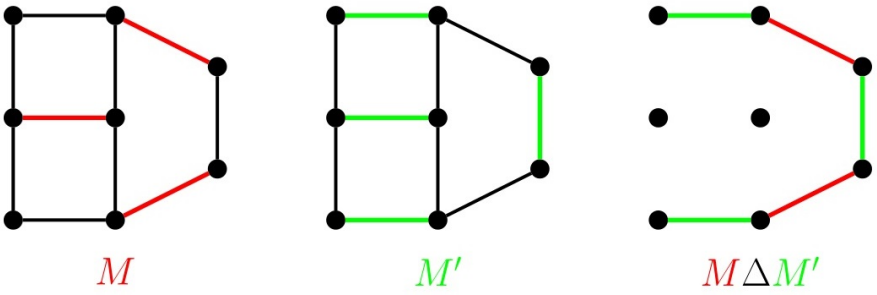


Abbildung 3.12: Symmetrische Differenz $M\Delta M'$ zweier Matchings M und M'

Wir erhalten damit eine Charakterisierung, wann ein Matching maximal ist.

Satz 3.19 (Berge (1957)). Ein Matching M ist genau dann maximal, wenn es keinen M-augmentierenden Weg gibt.

Beweis. Für die Hinrichtung nehmen wir per Widerspruch an, dass es noch einen M-augmentierenden Weg P gibt. Dann ersetzen wir die Kanten aus P in M durch die Kanten aus $E(P) - M$ und vergrößern damit das Matching, ein Widerspruch.
Nun zur Rückrichtung: Wir nehmen an, dass M nicht maximal ist, d.h. es gibt ein Matching M' mit $|M'| > |M|$. Wir betrachten sodann die symmetrische Differenz $M'\Delta M := (M' - M) \cup (M - M')$ und den Graphen $G' = (V, M'\Delta M)$, vgl. Abbildung 3.12. Nun haben alle Knoten in G' einen Grad ≤ 2, d.h. die Zusammenhangskomponenten bestehen entweder aus isolierten Knoten, elementaren Kreisen (jeweils gleiche Anzahl von Kanten bzgl. M' und M) oder aus elementaren Wegen. Davon gibt es wieder drei mögliche Typen. Entweder

- beginnt und endet ein elementarer Weg mit Kanten aus M, d.h. $|M| > |M'|$ auf diesem Weg,
- beginnt ein elementarer Weg mit einer Kante aus M und endet mit einer Kante aus M', d.h. $|M| = |M'|$ auf diesem Weg,
- beginnt und endet dieser Weg mit Kanten aus M', d.h. $|M'| > |M|$ auf diesem Weg.

Im letzten Fall hätten wir damit aber einen M-augmentierenden Weg gefunden, was nicht sein kann. In allen übrigen Fällen gilt jedoch $|M| \geq |M'|$, ein Widerspruch zur Annahme. □

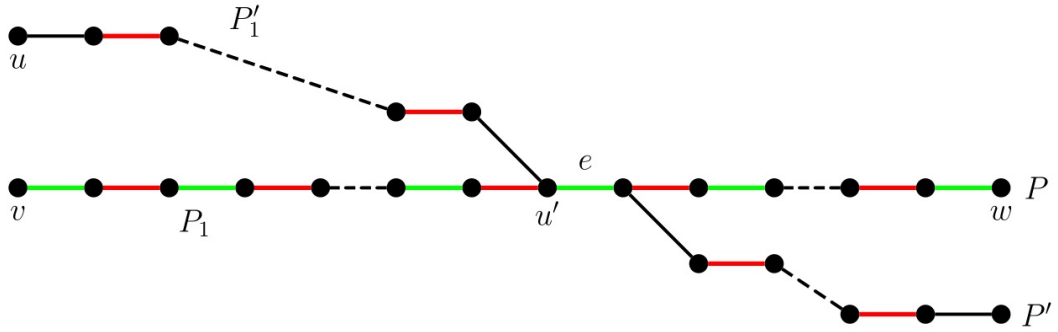


Abbildung 3.13: Situation im Beweis, Matchingkanten aus M sind rot, aus M' grün.

Weiterhin erhalten wir folgenden nützlichen Satz, wenn wir die Laufzeit von Matching-Algorithmen betrachten:

Satz 3.20. Sei G ein Graph, M ein Matching in G und u ein exponierter Knoten (bzgl. M). Sei weiterhin P ein augmentierender Weg und sei $M' = M \Delta P$. Dann gilt: Gibt es keinen augmentierenden Weg bzgl. M, der in u startet, so gibt es auch keinen augmentierenden Weg bzgl. M', der in u startet.

Beweis. Seien v und w die Endknoten des augmentierenden Weges P. Da u nicht Endknoten eines augmentierenden Weges bzgl. M ist, gilt $u \neq v, w$. Nehmen wir per Widerspruch an, dass es einen augmentierenden Weg P' bzgl. M' gibt, der in u beginnt. Wenn nun P und P' knotendisjunkt sind, dann ist offensichtlich P' auch ein M-augmentierender Weg, ein Widerspruch zur Voraussetzung.
Sei nun u' der erste Knoten in P', der auch in P enthalten ist und sei e die eindeutige Matchingkante in M', die zu u' inzident ist. Dann teilt u' den Pfad P in zwei Teile auf. Denjenigen Teil, der e nicht enthält, bezeichnen wir mit P_1. Außerdem sei P_1' der Teilpfad von u nach u' in P'. Dann ist aber $P_1 P_1'$ ein M-augmentierender Weg, der in u beginnt (siehe Abbildung 3.13), ein Widerspruch. □

Das Ziel ist nun, so lange nach augmentierenden Wegen zu suchen, bis keine mehr existieren. Einen Augmentierungsschritt nennen wir dabei Phase. Aus Satz 3.20 folgt direkt:

Korollar 3.21. Sei u ein exponierter Knoten, von dem aus in einer Phase kein augmentierender Weg gefunden wird. Dann gibt es in allen nachfolgenden Phasen auch keinen augmentierenden Weg, der in u startet.

Wir müssen also jeden exponierten Knoten maximal einmal genauer anschauen. Dies verwenden wir im Algorithmus von Edmonds.

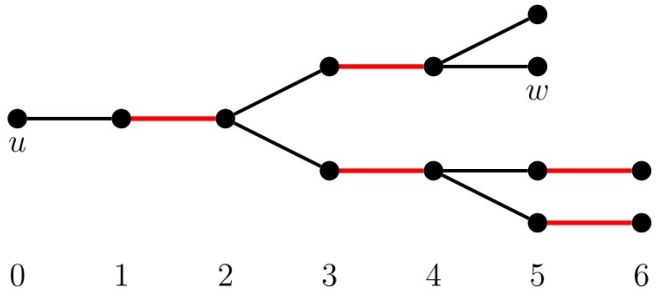


Abbildung 3.14: M-alternierender Baum mit M-augmentierendem u-w-Weg

3.4 Der Algorithmus von Edmonds

In diesem Abschnitt wollen wir den Algorithmus von Edmonds erarbeiten, welcher ein maximales Matching in einem beliebigen Graphen bestimmt.
Für bipartite Graphen haben wir schon gesehen, dass wir den Algorithmus von Ford-Fulkerson zur Bestimmung eines maximalen Matchings benutzen können, indem wir den Graph in ein Flussnetzwerk transformieren. Wir wollen nun einen Algorithmus für bipartite Graphen angeben, der die Einsichten zu M-augmentierenden Wegen aus dem letzten Abschnitt verwendet. Dies ist ein Spezialfall des allgemeinen Algorithmus von Edmonds, welchen wir aufbauend auf die folgenden Erkenntnisse betrachten werden.
Sei nun $G = (V = A \cup B, E)$ ein bipartiter Graph mit $A = \{1, \ldots, a\}$, $B = \{1', \ldots, b'\}$ und $|V(G)| = a + b = n$. Wir nehmen weiterhin o.B.d.A. an, dass $a \leq b$ gilt (um die Zahl der benötigten Schritte zu minimieren).
Das Vorgehen ist nun folgendermaßen: Einen M-augmentierenden Weg suchen wir durch eine Breitensuche von einem exponierten Knoten $u \in A$ aus. Dadurch entstehen wieder Schichten, wobei u in Schicht 0 liegen soll. Damit liegen alle Knoten aus A in einer geraden Schicht und alle Knoten aus B in einer ungeraden Schicht. Wird bei der Breitensuche von einem Knoten v in A aus ein noch nicht im Baum enthaltener Knoten w in B entdeckt, so sind zwei Fälle möglich:

Fall 1: w ist bereits im Matching M enthalten, sei z der Partner von w. Dann fügen wir sofort auch die Kante $\{w, z\}$ zum Breitensuchbaum hinzu und hängen z an die Warteschlange an.

Fall 2: Ist w exponiert, so haben wir einen M-augmentierenden Weg gefunden und können unser Matching vergrößern. Der M-augmentierende Weg ergibt sich aus dem (eindeutigen) u-w-Weg im Breitensuchbaum.

Den entstehenden Breitensuchbaum nennt man auch M-alternierenden Baum (vgl. Abbildung 3.14, Matchingkanten sind wieder rot eingezeichnet; außerdem wurden die einzelnen Schichten nummeriert).
Beachte dabei: durch das sofortige Hinzufügen von Matchingkanten zum Baum wird im Verlauf der Breitensuche immer nur von einem Knoten in A aus gesucht! Außerdem wissen

wir aus dem letzten Abschnitt, dass wenn die Breitensuche terminiert und dabei kein exponierter Knoten gefunden wurde, so existiert kein M-augmentierender Weg von u aus. Damit kann aber auch in den folgenden Phasen kein M-augmentierender Weg existieren, der in u beginnt.

Daher müssen wir von jedem der maximal n exponierten Knoten nur einmal eine Breitensuche durchführen (jeweils Laufzeit $O(m)$), dies ergibt demnach eine Laufzeit von $O(mn)$.

Formal können wir den Algorithmus von Edmonds für bipartite Graphen folgendermaßen beschreiben: Wir speichern unser Matching in einem Array mate ab, wobei $\text{mate}(i) = j'$ bedeuten soll, dass $\{i, j'\} \in M$ liegt. Ist ein Knoten v exponiert, so sei $\text{mate}(v) = 0$. Weiterhin gibt die Funktion $p(j')$ für $j' \in B$ denjenigen Knoten in A an, von dem j'

erreicht wurde und nrex gibt die Anzahl der exponierten Knoten in G an.

Eingabe: Bipartiter Graph $G = (V = A \cup B, E)$.
Ausgabe: Ein maximales Matching M in G, gespeichert im Array mate.

```
for v ∈ V                                                    // Initialisierung
 do
 | mate(v) ← 0
end
r ← 0; nrex ← a + b
while nrex ≥ 2 und r < a                  // M kann evtl. noch vergrößert werden
 do
 | r ← r + 1                                  // gehe zum nächsten Knoten aus A
 | if mate(r) = 0                                        // r ist also exponiert
 |  then
 |  | for i = 1,...,b                  // initialisiere leeren Breitensuchbaum
 |  |  do
 |  |  | p(i') ← 0
 |  | end
 |  | Q ← {r}; augment = false
 |  | while augment = false und Q ≠ ∅ do
 |  |  | lösche ersten Knoten x aus Q
 |  |  | if es gibt y ∈ A_x mit mate(y) = 0      // augmentierender Weg gefunden
 |  |  |  then
 |  |  |  | wähle so ein y
 |  |  |  | while x ≠ r                                  // aktualisiere Matching
 |  |  |  |  do
 |  |  |  |  | mate(y) ← x; next ← mate(x); mate(x) ← y; y ← next; x ← p(y)
 |  |  |  | end
 |  |  |  | mate(y) ← x; mate(x) ← y; nrex ← nrex −2; augment = true
 |  |  | end
 |  |  | else
 |  |  |  | es gibt kein so ein y                 // alle Nachbarn von x gematcht
 |  |  |  | for y ∈ A_x                              // erweitere Breitensuchbaum
 |  |  |  |  do
 |  |  |  |  | if p(y) = 0 then
 |  |  |  |  |  | p(y) = x; hänge mate(y) an Q an
 |  |  |  |  | end
 |  |  |  | end
 |  |  | end
 |  | end
 | end
end
return mate(v) für alle v ∈ V
```

Algorithm 3.4.1: Der Algorithmus von Edmonds (bipartit)

Beispiel 3.22. Hier noch ein kleines Beispiel zum Algorithmus. Gegeben sei der bipartite Graph in Abbildung 3.15 (links) mit $A = \{1, 2, 3\}$ und $B = \{1', 2', 3', 4'\}$. In den ersten beiden Schritten findet man von Knoten 1 sofort den Knoten $1'$ und von Knoten 2 sofort Knoten $2'$, das momentane Matching sieht also so wie in der mittleren Abbildung aus.

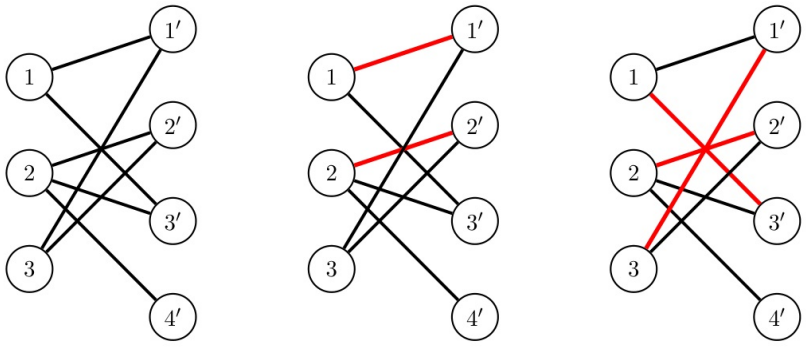


Abbildung 3.15: Schrittweise Entstehung des Matchings im Beispiel zum Algorithmus von Edmonds

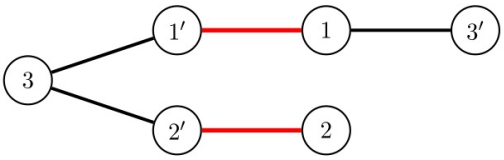


Abbildung 3.16: M-alternierender Baum im Beispiel zum Algorithmus von Edmonds

Im nächsten Schritt geht man zu Knoten 3. Die Nachbarn von 3 sind die Knoten $1'$ und $2'$. Da beide schon gematcht sind, werden zusätzlich die Kanten $\{1', 1\}$ und $\{2', 2\}$ in den M-alternierenden Baum aufgenommen, vgl. Abbildung 3.16. Beim nächsten Suchschritt von 1 aus entdeckt man den exponierten Knoten $3'$. Damit ändert man das Matching zu $M = \{\{1, 3'\}, \{3, 1'\}, \{2, 2'\}\}$. Da nun alle Knoten aus A abgearbeitet sind, terminiert der Algorithmus.

3.5 Edmonds Matching Algorithmus für beliebige Graphen

Satz 3.19 (Berge) gibt die Idee für einen Algorithmus wie bei Flüssen: Finde iterativ augmentierende Wege bis keiner mehr existiert. Bei beliebigen Graphen ist es jedoch unklar, wie man dabei vorgehen soll.
Wir geben ein Beispiel (siehe Abbildungen 3.17, 3.18 und 3.19).
Ungerade Kreise machen im Weiteren die Hauptschwierigkeit, wenn man sie zulässt, ist unklar, wie oft man sie durchlaufen soll (darf) und wie man das kontrolliert.
Lösung durch Edmonds (1965) "Paths, trees, and blossoms". Ein wichtiger Punkt im Algorithmus besteht darin, Entscheidungen über die Richtung des zu ermittelnden augmentierenden Weges aufzuschieben (delayed decisions). Dies wird erreicht durch Schrumpfung von Teilen des Graphen. Auf den geschrumpften Teilen wird die Entscheidung erst nach Vorliegen von mehr Information getroffen. Wir wiederholen die Definition eines M-alternierenden Baumes.

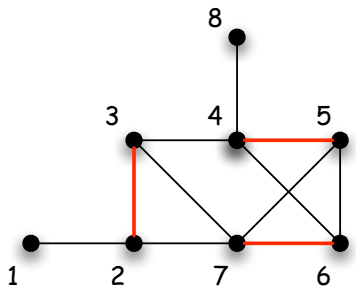


Abbildung 3.17: Betrachte folgenden Graph mit Matching M.

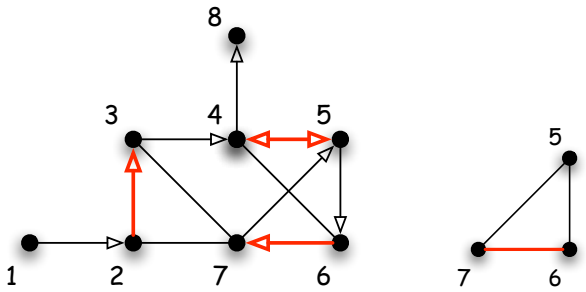


Abbildung 3.18: Erster Versuch eines augmentierenden Weges mit Tiefensuche von 1 aus: $1-2-3-4-5-6-7-5-4-8$. Ist nicht elementar, also nicht M-augmentierend. Er enthält ungeraden Kreis.

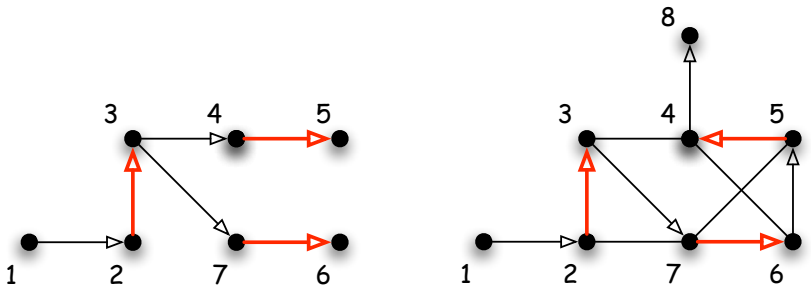


Abbildung 3.19: Zweiter Versuch: Breitensuche von 1 aus. Geht nicht weiter. Es existiert jedoch ein augmentierender Weg (siehe Bild rechts).

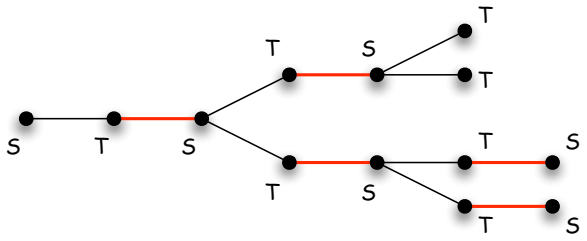


Abbildung 3.20: M-alternierender Baum.

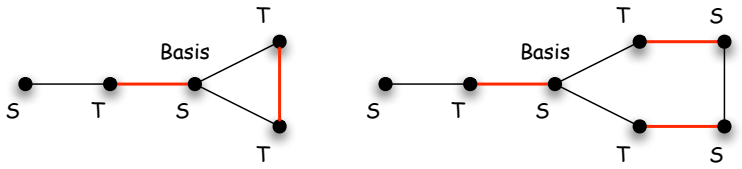


Abbildung 3.21: Blüten.

Definition 3.23.

1. M-alternierender Baum = Baum mit exponierter Wurzel, dessen Wege von der Wurzel bis zu den Blättern M-alternierende Wege sind. Knoten bekommen entlang dieser Wege abwechselnd das Label S oder T, die Wurzel bekommt S.

2. Beobachtung:

   Ist ein Blatt in einem M-alternierenden Baum exponiert, so ist der Weg von der Wurzel bis zu diesem Blatt M-augmentierend. (3.3)

3. Blüte (blossom) bzgl. M-alternierendem Baum = elementarer Kreis B, der entsteht durch eine Kante aus $E(G) - M$ zwischen zwei S-Knoten oder einer Kante aus M zwischen zwei T-Knoten.

4. Basis der Blüte = zugehöriger (einziger) exponierter Knoten von $G[B]$, dieser hat Label S.

5. Stängel der Blüte = Weg von der Wurzel bis zur Basis im Baum.

Wir halten folgende weitere Beobachtung fest:

Für eine Blüte B gilt: (3.4)

- B hat ungerade Knotenzahl $2r + 1$ und r Matching Kanten
- Von der Basis aus gibt es innerhalb von B zu jedem anderen Knoten aus B alternierende Wege in beide "Richtungen". Ein alternierender Weg durch eine Blüte

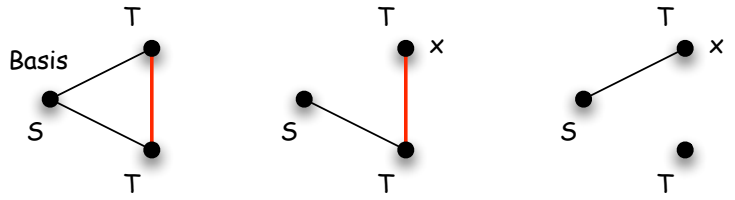


Abbildung 3.22: Alternierende Wege in der Blüte zu x (letzte Kante ist wahlweise Matching- oder Nicht-Matching-Kante).

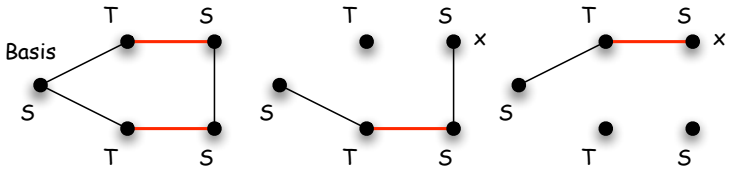


Abbildung 3.23: Alternierende Wege in der Blüte zu x (letzte Kante ist wahlweise Matching- oder Nicht-Matching-Kante).

kann also zwei verschiedene Alternativen wählen. Die Entscheidung darüber wird zunächst aufgeschoben (delayed decision). Blüten kodieren also Alternativen für die Suche nach M-augmentierenden Wegen.

Definition 3.24. Kontraktion (Schrumpfen, shrinking) einer Blüte B in G ergibt den Graphen G^cB. Er entsteht durch Ersetzen von $G[V(B)]$ durch einen Pseudoknoten ν_B und der Einführung von Kanten (ν_B, x) zu allen $x \notin B$, die mit einem Knoten $y \in B$ in G verbunden sind.

Wir erhalten folgende Eigenschaft (vgl. Abbildung 3.24 und 3.25).

Ist T ein alternierender Baum mit Blüte B, so ist T^cB ein alternierender Baum von G^cB, wobei ν_B das Label S erhält. (3.5)

Wir erhalten folgendes wichtige Lemma.

Lemma 3.25. Sei B eine Blüte von G (bzgl. eines M-alternierenden Baumes). Dann sind äquivalent:

1. es gibt einen augmentierenden Weg in G bzgl. M.
2. es gibt einen augmentierenden Weg in G^cB bzgl. $M' := M - E(B)$.

Beispiel für die Richtung "$\Leftarrow$" (Abbildung 3.26 und 3.27):

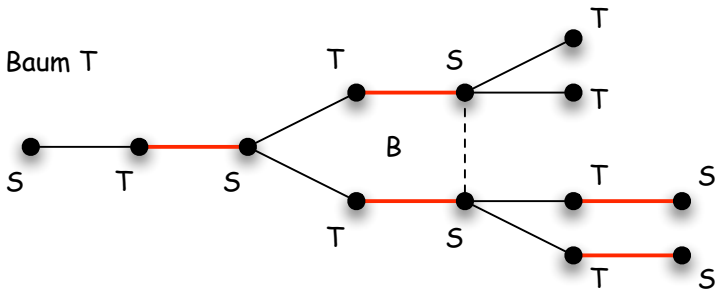


Abbildung 3.24: Baum T.

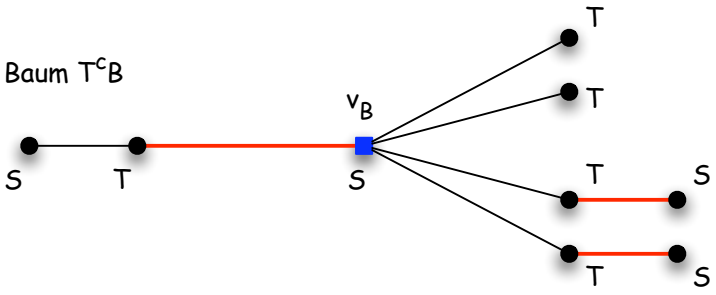


Abbildung 3.25: Baum $T^C B$.

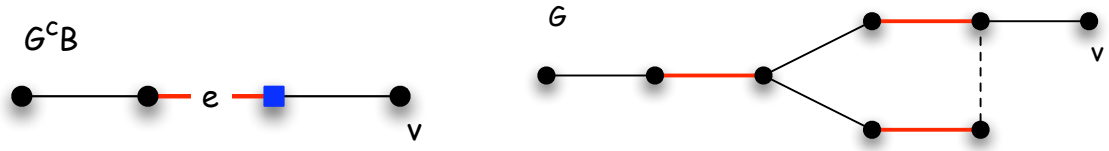


Abbildung 3.26: v_B ist nicht exponiert.

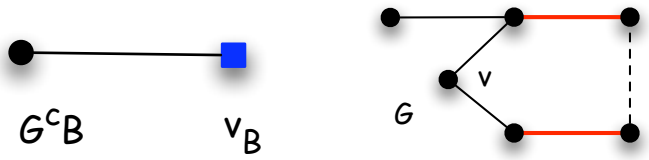


Abbildung 3.27: v_B ist exponiert.

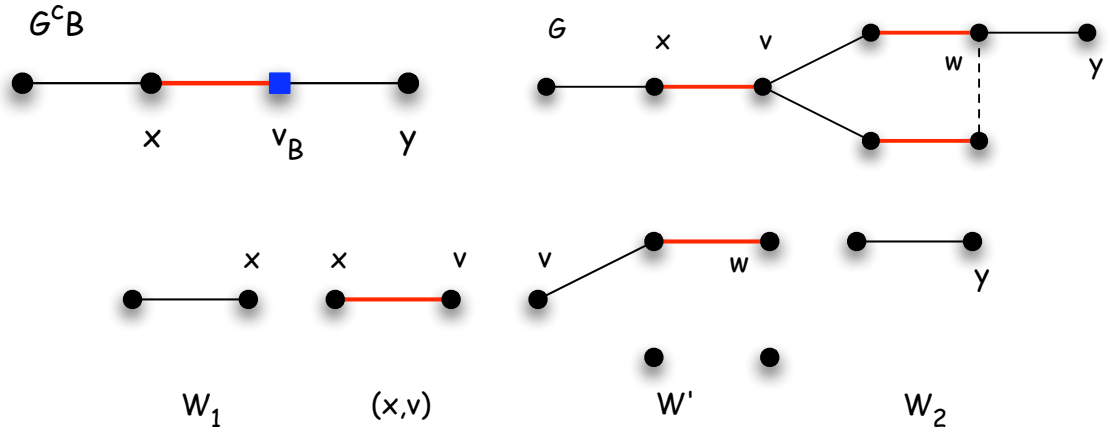


Abbildung 3.28: Situation in Fall (b1).

Beweis. $\Leftarrow$:
Sei W M'-augmentierend in G^cB. Beachte dass dieser Weg beliebig durch G^cB verläuft, nicht notwendigerweise im alternierenden Baum T zur Blüte B enthalten sein muss.
(a) W geht nicht durch den Pseudoknoten v_B. Dann ist W ist M-augmentierend in G, da nur Knoten und Kanten aus G benutzt werden.
(b) W geht durch den Pseudoknoten v_B.

Fall (b1) v_B ist nicht exponiert (vgl. Abbildung 3.28):
Dann hat W die Form $W = W_1 + (x, v_B) + W_2$ mit $(x, v_B) \in M$. Mit (3.4) folgt, dass es einen M-alternierenden Weg W' in B von Basis v bis zum Knoten w in B, der zum zweiten Knoten y aus W_2 adjazent ist, gibt, wobei die letzte Kante in B passend als Matching-Kante gewählt wird. Daraus folgt $W_1 + (x, v) + W' + W_2$ ist M-augmentierend in G.

Fall (b2) v_B ist exponiert (vgl. Abbildung 3.29):
W hat dann die Form $W = W_1 + (x, v_B)$ mit $(x, v_B) \notin M$. Wähle also in B einen alternierenden Weg W', der mit einer Matching Kante (y, z) beginnt (wobei y in G zu x adjazent ist) und mit der Basis v endet. Weil nun v_B exponiert ist, folgt dass v exponiert ist, und somit ist $W_1 + (x, y) + W'$ M-augmentierend in G.

$\Rightarrow$:
analog, folgt aber auch aus einem späteren Satz, der zum Beweis nur die hier bewiesene Richtung des Lemmas braucht.

□

Definition 3.26. Ungarischer Baum bzgl. M = maximaler M-alternierender Baum (d.h. nicht vergrösserbar), ohne Blüten, ohne exponierte Blätter (d.h. es gibt in ihm keinen M-augmentierenden Weg).

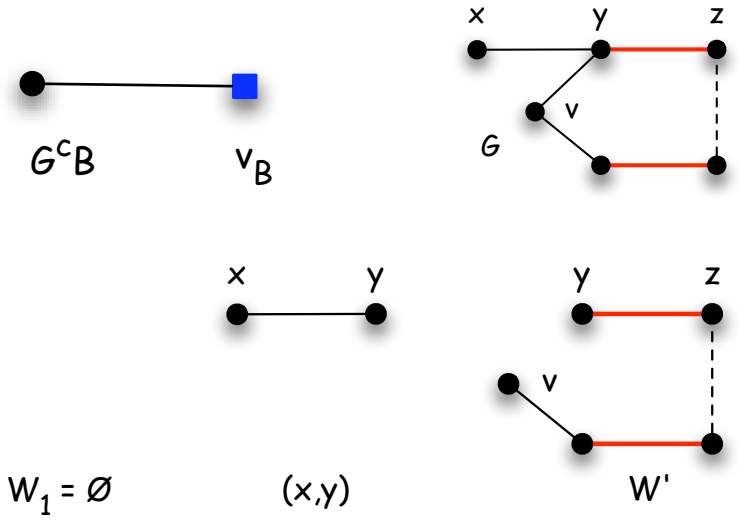


Abbildung 3.29: Situation in Fall (b2).

Eingabe: Ungerichteter Graph G.
Ausgabe: Ein Matching M maximaler Kardinalität.
Methode:
Initialisierung: setze $G' := G$ und $M' := M := M_0$ (M_0 beliebiges Matching, $M_0 = \emptyset$ möglich)
Hauptschleife:
Repeat konstruiere M'-alternierenden Baum T in G' // zum Beispiel mit Breitensuche
breche ab sobald einer der folgenden Fälle während der Konstruktion eintritt.
Fall:

- T enthält M'-augmentierenden Weg
  - konstruiere zugehörigen M-augmentierenden Weg W in G gemäss Lemma 3.25
  - augmentiere M entlang W
  - setze $G' := G$ und $M' := M$ // beginne wieder in G mit grösserem Matching
- T enthält Blüte B
  - kontrahiere B
  - setze $G' := G'^{c}B$ und $M' := M' - E(B)$
- T ist ungarisch
  - setze $G' := G' - V(T)$ und $M' := M' - E(T)$ // $V(T)$ und inzidente Kanten wegnehmen.

End Fall
Until: $G' = \emptyset$ oder es gibt keine exponierten Knoten. // M' ist maximal in G'
return: M

Algorithm 3.5.1: Edmonds Matching Algorithmus

Wir rechnen ein Beispiel in der Vorlesung vor.

Bezüglich der Korrektheit des Algorithmus ist folgendes zu zeigen:

1. Der Algorithmus terminiert: im case statement tritt einer von 3 Fällen auf
   - das Matching wird vergrössert $\Rightarrow$ maximal $n/2$ mal
   - eine Blüte wird kontrahiert $\Rightarrow$ # Knoten nimmt um ≥ 2 ab $\Rightarrow$ maximal $n/2$ mal pro Vergrösserung des Matchings
   - ein ungarischer Baum wird abgespalten $\Rightarrow$ # Knoten nimmt um ≥ 1 ab $\Rightarrow$ maximal n mal pro Vergrösserung des Matchings
2. Das bei Termination konstruierte Matching ist maximal. Wir können die Korrektheit wie bei dem Max-Fluss Problem über Dualitätssatz (Min-Max Satz) zeigen.
   - duale Struktur zu s,t-Fluss $\rightarrow$ s,t-Schnitt
   - duale Struktur zu Matching $\rightarrow$ Odd Set Cover.

Definition 3.27. Odd Set Cover (OSC) eines ungerichteten Graphen G ist ein Mengensystem $\mathcal{V} = \{V_1, \ldots, V_p\}$ mit $V_i \subseteq V(G), |V_i|$ ungerade

- falls $|V_i| = 1$, etwa $V_i = \{v\}$, so hat V_i die Kapazität $c(V_i) := 1$ und überdeckt alle mit v inzidenten Kanten
- falls $|V_i| = 2r + 1$ mit $r \geq 1$, so hat V_i die Kapazität $c(V_i) := r$ und überdeckt alle Kanten in $G[V_i]$, also alle Kanten mit beiden Endpunkten in V_i.
- alle Kanten von G werden überdeckt.
- $c(\mathcal{V}) := \sum_i c(V_i)$ heisst Kapazität des OSC $\mathcal{V}$.
- Sind alle $|V_i| = 1$, so ist $\mathcal{V}$ ein vertex cover (Knotenüberdeckung) und $c(\mathcal{V})$ ist die Kardinalität des vertex cover.

Satz 3.28 (Edmonds 1965). In einem ungerichteten Graphen gilt

$$\max\{|M| : M \text{ ist Matching in } G\} = \min\{c(\mathcal{V}) : \mathcal{V} \text{ ist OSC in } G\}.$$

Ein Spezialfall hiervon ist der Satz von König.

Beweis. "$\leq$":
jedes Matching M kann von den von den $V_i \in \mathcal{V}$ überdeckten Kanten höchstens $c(V_i)$ enthalten $\Rightarrow |M| \leq c(\mathcal{V})$ für jedes Matching M und jedes OSC $\mathcal{V} \Rightarrow$ "$\leq$".
"$\geq$":
Betrachte die Situation bei Terminierung des Algorithmus von Edmonds im momentanen, teilweise kontrahierten Graphen G' (Ungarische Bäume dazu genommen) mit Matching M_0' in G' und zugehörigem Matching M_0 in G.

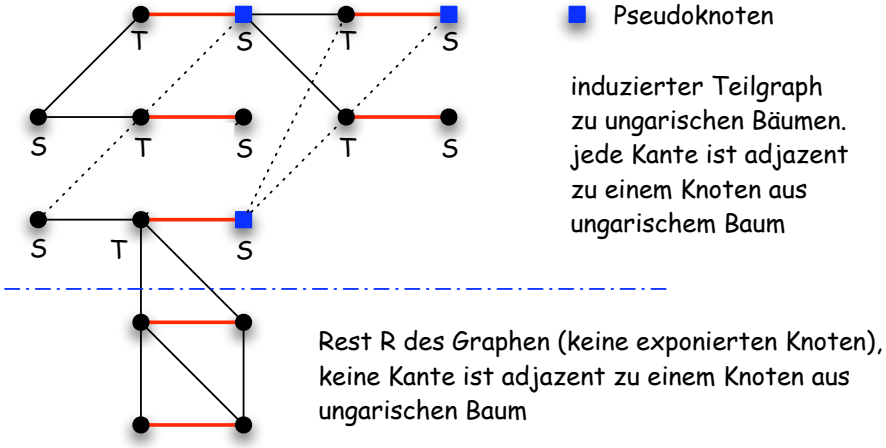


Abbildung 3.30: Pseudoknoten.

Übung 3.29. Für jede Kante $e \in E(G)$ gilt nun einer der folgenden Fälle:

1. e ist inzident zu Knoten mit Label T
2. e ist in (geschrumpfter) Blüte von G' enthalten
3. e ist im Rest R des Graphen

(Vollständige Fallunterscheidung, da ungarische Bäume maximal alternierend ohne exponierte Blätter!)

Konstruiere OSC $\mathcal{V}_0$ wie folgt:

(a) mache alle T-Knoten zu einelementigen Mengen V_i (Pseudoknoten sind nie T-Knoten).

(b) die zu Pseudoknoten (geschachtelt) kontrahierten Blüten bilden die mehrelementigen Mengen V_i.
Es folgt, dass $c(V_i) = \#$ Matching Kanten aus G in V_i.

(c) sei $k := \#$ Matching Kanten im Restgraphen R $\Rightarrow$ R hat $2k$ Knoten.

- $k = 0 \Rightarrow \mathcal{V}_0$ ist gemäss (a) und (b) fertig konstruiert und $|M_0| = c(\mathcal{V}_0)$.
- $k = 1 \Rightarrow$ nehme einelementige Menge V_j zu $\mathcal{V}_0$ hinzu
- $k \geq 2 \Rightarrow$ nehme eine einelementige Menge zu $\mathcal{V}_0$ hinzu, fasse die anderen $2k-1$ Knoten zu einer Menge V_j zusammen.

In allen Fällen ist $|M_0| = c(\mathcal{V}_0)$ und es folgt

$$\max\{|M| : M \text{ ist Matching in } G\} = \min\{c(\mathcal{V}) : \mathcal{V} \text{ ist OSC in } G\}.$$

□

Abschliessend halten wir fest:

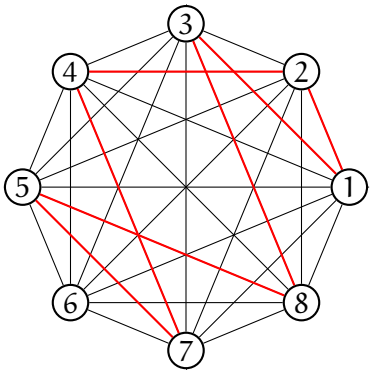
Satz 3.30. Der Algorithmus von Edmonds kann in $O(n^3)$ implementiert werden.

Kapitel 4

Komplexitätsklassen P und NP

Beispiel 4.1 (TSP und Algorithmen). Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit Kantenkosten $c_e, e \in E$.
Aufgabe: Finde eine kürzeste Tour, die alle Knoten V besucht.

- $G = K_n$ vollständiger Graph mit n Knoten
- Tour = Permutation $\pi = \{1, 2, 4, 7, 5, 8, 3\}$
- Es gibt $n!$ viele Permutationen
- $n = 8 : n! = 40320$
- $16! = 20922789888000$



Bisher haben wir Graphenprobleme kennengelernt, für die effiziente Algorithmen existieren. Effizient heisst: Worst Case Laufzeit (gemessen in einer natürlichen Definition von Schritten) ist durch ein Polynom in der Länge des Inputs (etwa $|V(G)|+|E(G)| = n+m$) beschränkt, z.B. Zusammenhang $O(n+m)$, Kürzeste Wege $O(nm)$, Max Fluss $O(n^3)$. Exakte Definition erfordert Formalisierung von Inputlänge, Rechnermodell (was sind Schritte?). Nach geschehener Formalisierung: Kann man Probleme bzgl. ihrer Komplexität klassifizieren? Was kommt nach effizient?

Schwere und einfache Probleme:

Leichte Probleme	Schwere Probleme
• Sortieren von n Zahlen Merge-Sort $\sim n\log(n)$	• TSP $\sim n^2 2^n$
• kürzeste Wege Dijkstra $\sim m + n\log(n)$	• Satisfiability $\sim 2^n$
• minimaler Spannbaum Prim $\sim n^2$	• Hamiltonkreis
• ...	• Vehicle routing
• ...	• Partition
	• ...
	• ...

Wie können wir die Schwierigkeit von Problemen vergleichen?

4.1 Kodierung von Inputdaten

Maß für die Größe des Inputs ist die Länge = # Bits (Zeichen) zur Kodierung der Input Instanz eines Problems. Die Länge hängt ab von der gewählten Kodierung der Daten. Beispiel ganze Zahlen

- Input: Zahl 5127
- Inputlänge: 4 bei Dezimaldarstellung
- 13 bei Dualdarstellung
- 5127 bei unärer Darstellung (Bierdeckel)

Beispiel einfache Graphen $G = (V, E)$: Inputlänge: n^2 bei Adjazenzmatrix, $n + m$ bei Adjazenzlisten.
Allgemein:

- I Instanz einer Problemklasse
- $C(I)$ Kodierung von I
- $\langle C(I)\rangle$ Länge der Kodierung von I mittels C

Definition 4.2. Kodierungen C_1, C_2 heissen polynomial äquivalent (bzgl. einer Problemklasse) $:\Leftrightarrow$ es gibt Polynome $p_1, p_2 : \mathbb{N} \to \mathbb{N}$ so dass für alle Instanzen I der Problemklasse gilt:

$$\langle C_2(I)\rangle \leq p_1(\langle C_1(I)\rangle)$$
$$\langle C_1(I)\rangle \leq p_2(\langle C_2(I)\rangle)$$

Beispiel einfache Graphen: Kodierungen als Adjazenzmatrix und Adjazenzlisten sind polynomial äquivalent.
Beispiel Zahlen: binäre und k-äre (z.B. dezimale) Darstellung von ganzen Zahlen sind polynomial äquivalent.

- binär: $\langle n \rangle = \lceil \log_2(|n|+1) \rceil + 1$.
- k-är: $\langle n \rangle = \lceil \log_k(|n|+1) \rceil + 1 = \lceil (\log_2(|n|+1))/\log_2 k \rceil + 1$

ABER: binäre und unäre Darstellung natürlicher Zahlen sind nicht äquivalent

$$\text{binär: } \langle n \rangle = \lceil \log_2(|n|+1) \rceil + 1$$
$$\text{unär: } \langle n \rangle = n \geq 2^{\lceil \log_2(|n|+1) \rceil + 1}/4 - 1$$

also ist $\langle n \rangle$ exponentiell in $\lceil \log_2(|n|+1) \rceil + 1$.
Standardkodierungen:

- ganze Zahlen: binär, $\langle n \rangle = \lceil \log_2(|n|+1) \rceil + 1$
- Vektor $x = (x_1, ..., x_n)$ als n ganze Zahlen, $\langle x \rangle = \langle x_1 \rangle + \cdots + \langle x_n \rangle$
- Graph als Array von Adjazenzlisten, $\langle G \rangle = n + m$
  ACHTUNG: hier werden Kodierungslängen von Knoten und Kanten als 1 gezählt, obwohl man zur Unterscheidung von n Objekten eigentlich $n \cdot (\lceil \log_2 n \rceil + 1)$ Bits braucht
- Kantenbewertungen als ganze Zahlen, also mit Länge $\langle u(e) \rangle = \lceil \log_2(|u(e)|+1) \rceil + 1$

Beispiel 4.3. Max Fluss Problem (G, u, s, t) hat Inputlänge

$$\langle (G, u, s, t) \rangle = \langle G \rangle + \langle u \rangle + 2 = n + m + \sum_e \langle u(e) \rangle + 2 = O(n + m \log U)$$
$$\text{mit } U := \max_e u(e) \ (u(e) \geq 0 \text{ vorausgesetzt}).$$

Wir gehen nun im Folgenden von einer binären Standardkodierung von Inputdaten aus.

4.2 Rechnermodelle

Es gibt eine Reihe von etablierten Rechnermodellen und wir listen hier die wichtigsten.

- RAM (Random Access Machine): sehr leistungsfähig, hat Registerarithmetik und indirekte Adressierung
- Turing Maschine: sehr grob, nicht viel Ähnlichkeit mit modernem Rechner, 1936 von Alan Turing als Gedankenmodell zur Formalisierung von Berechnungen entwickelt

Dennoch gilt: jeder dieser (und anderer) Maschinentypen kann durch jeden anderen Typ simuliert werden, wobei die Zeiten für Rechenoperationen sich in Abhängigkeit von der

gleichen Inputlänge nur um ein Polynom unterscheiden und der benötigte Speicher nur um einen konstanten Faktor. "Um ein Polynom unterscheiden" bedeutet:

Ist T_i die Laufzeit auf Rechnermodell $i (i = 1, 2)$, so existieren Polynome p_1, p_2 mit

$$T_1(I) \leq p_2(T_2(I)) \text{ für alle Inputs } I$$
$$T_2(I) \leq p_1(T_1(I)) \text{ für alle Inputs } I$$

Äquivalenzthese: Diese Äquivalenz gilt für alle "vernünftigen"Rechnermodelle (für viele bereits nachgewiesen in der Theorie der Berechenbarkeit)

Definition 4.4. Die Laufzeit $T_A(I)$ eines Algorithmus A angewandt auf eine Instanz I bzgl. eines Rechnermodells ist definiert als die Summe der Kosten aller Schritte in A bei Input I.

Kosten eines Schrittes hängen vom Rechnermodell ab, z.B.:

- Addition von 2 k-stelligen Dualzahlen $= O(k)$
- Addition von 2 k-stelligen Dualzahlen $= O(1)$ (unit cost model); dies ist unsere generelle Annahme für alle arithmetischen Operationen

4.3 Optimierungs- vs. Entscheidungsprobleme

Beispiel 4.5 (Optimierungsprobleme).
- Berechne einen kürzesten s, t-Weg W in $G = (V, E)$ bzgl. c.
- Berechne eine kürzeste Tour T in $G = (V, E)$ bzgl. c.

Beispiel 4.6 (Entscheidungsprobleme).
- Existiert ein s, t-Weg W in $G = (V, E)$ mit Länge $\leq L$ bzgl. c?
- Existiert eine Tour T in $G = (V, E)$ mit Länge $\leq L$ bzgl. c?

Übung 4.7. Wenn wir effizient entscheiden können, können wir effizient optimieren (binäre Suche über L)!

4.4 Komplexitätsklasse P

Definition 4.8 (Komplexitätsklasse P (polynomial time)). P ist die Menge aller Entscheidungsprobleme Π, für die ein Algorithmus A existiert, der jede Instanz $I \in \Pi$ in polynomieller Zeit in der Größe der Instanz $\langle I \rangle$ löst.

$$T_A(I) \leq p(\langle I \rangle) \text{ für alle } I \in \Pi.$$

Beispiel 4.9 (Kürzeste Wege Problem). Instanz I: Gibt es einen s,t-Weg in $G = (V, E)$ mit Länge $\leq L$ bzgl. $c \geq 0$? Wende Dijkstra an!

$$\text{Laufzeit} \leq c_1|E| + c_2|V|\log(|V|) \leq p(\langle I \rangle).$$

Beispiel 4.10 (Minimaler Spannbaum MST). Instanz I: Gibt es einen Spannbaum in $G = (V, E)$ mit Länge $\leq L$ bzgl. c?
Wende Prim an!

$$\text{Laufzeit} \leq c_1|V|^2 \leq p(\langle I \rangle).$$

4.5 Die Klasse NP

Definition 4.11. NP (non-deterministic polynomial time): Ein Entscheidungsproblem Π ist in NP, falls es für JA-Instanzen polynomiell grosse Zertifikate gibt, die in polynomieller Zeit verifizierbar sind.

Beispiel 4.12 (TSP ist in NP). Instanz I: Zertifikat: Tour T in $G = (V, E)$ mit Länge $\leq L$. Überprüfung geht in polynomieller Zeit.

Beispiel 4.13 ($P \subseteq NP$). Instanz I: Konstruiere Zertifikat durch Anwendung eines polynomiellen Algorithmus.

Millenniumsproblem: P versus NP (1 Million Dollar, Clay Institute)

4.6 Polynomielle Reduktionen

Wie können wir die Komplexität von Problemen vergleichen?

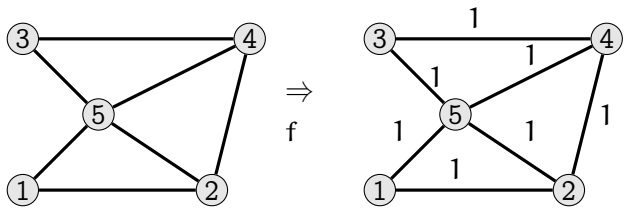
Definition 4.14 (Reduktion). Seien Π_1 und Π_2 Entscheidungsprobleme. Π_1 heisst polynomial transformierbar auf Π_2 (in Zeichen $\Pi_1 \leq \Pi_2$)
$:\Leftrightarrow$ es existiert eine Transformation $f : \Pi_1 \to \Pi_2$ mit:

$\forall I \in \Pi_1$ ist $f(I) \in \Pi_2$ in polynomialer Zeit (polynomial in $\langle I \rangle$) konstruierbar

I ist Ja-Instanz von $\Pi_1 \Leftrightarrow f(I)$ Ja-Instanz von Π_2.

Beispiel 4.15 (Hamilton Circuit). Instanz: Graph $G = (V, E)$.
Frage: Existiert ein Kreis C der jeden Knoten in V genau einmal durchläuft?

Beispiel 4.16 (TSP). Instanz: (G, c, L), Graph $G = (V, E)$, Kosten $c_e, e \in E$, Zahl L.
Frage: Existiert eine Tour T mit Länge $\leq L$ bzgl. c?



Reduktion Ham.Circ. $\leq$ TSP: Polynomiell berechenbare Funktion $f : G \to (G', c, L)$ mit

G hat Ham. Circ. $\Leftrightarrow f(G) = (G', c, |V|)$ besitzt Tour mit Länge $\leq |V|$.

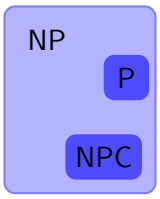
4.7 NP-Vollständigkeit

Falls sich Problem A auf Problem B reduzieren lässt schreiben wir: $A \leq B$.
Z.B. Hamilton Circuit $\leq$ TSP.

Definition 4.17 (NPC (NP-complete)). Sei $\Pi \in$ NP eine Problemklasse (z.B. TSP).

$$\Pi \text{ ist NP-vollständig } \Leftrightarrow A \leq \Pi \text{ für alle } A \in \text{NP}.$$

NPC ist die Menge der NP-vollständigen Probleme.



Nun kommen wir zum zentralen Resultat welches besagt, dass NPC $\neq \emptyset$. Dazu führen wir zunächst die Problemklasse Satisfiability ein.

Eine Instanz von Satisfiability (SAT) ist gegeben durch

- m Klauseln $C_1, \ldots, C_m$ in Booleschen (binär) Variablen $x_1, \ldots, x_n$.
- Klausel C_i ist logischer Ausdruck der Form

$$C_i = y_{i_1} \vee y_{i_2} \vee \cdots \vee y_{i_k} \text{ mit } y_{i_j} \in \{x_{i_j}, \bar{x}_{i_j}\}.$$

- Die y_{i_j} heissen Literale.

Frage: Existiert eine Belegung der Variablen mit TRUE (= 1) und FALSE (= 0), so dass alle Klauseln wahr sind? (Es wird von einer erfüllenden Belegung gesprochen.)

Beispiel 4.18. Betrachte zwei Klauseln:

$$\bar{x}_3, x_1 \vee x_2 \vee x_3$$

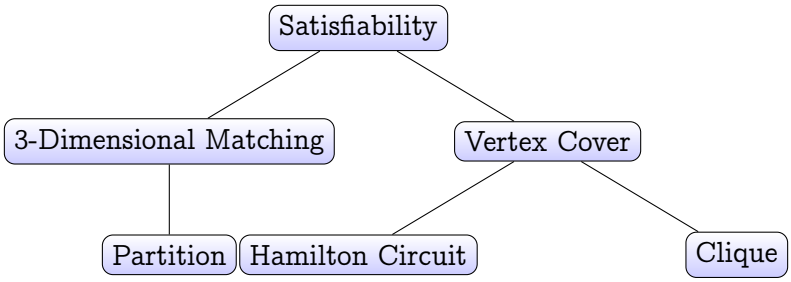
Diese Instanz ist erfüllbar (also eine Ja-Instanz). Nehme z.B. $x_3 = 0, x_1 = 1, x_2 = 0$.
Die folgende Instanz ist nicht erfüllbar:

$$x_1 \vee x_2 \vee x_3, x_1 \vee \bar{x}_2, x_2 \vee \bar{x}_3, x_3 \vee \bar{x}_1, \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$$

Wir überlegen uns folgendes:

- C_1 wahr $\Rightarrow$ ein x_j muss wahr sein
- C_2, C_3, C_4 wahr $\Rightarrow$ alle x_j wahr oder alle x_j falsch
- C_5 wahr $\Rightarrow$ ein x_j muss falsch sein

Satz 4.19 (Cook ('71), Karp ('72)). Satisfiability liegt in NPC!



Wir geben nun eine erste nichttriviale Reduktion von SAT an. Betrachte folgende Problemklasse:

Stable Set
Instanz: Ein ungerichteter Graph G, eine natürliche Zahl k
Frage: Hat G eine stabile Menge mit k Knoten?
(stabile Menge = Menge von Knoten, die paarweise nicht adjazent sind)

Satz 4.20. Stable Set ist NP-vollständig.

Beweis. (a) Stable Set $\in$ NP. Zertifikat ist eine stabile Menge der Grösse k.
(b) Transformation SAT $\leq$ Stable Set.
Betrachte eine Instanz I von SAT, also Klauseln $Z_1, ..., Z_m$ mit

$$Z_i = y_{i_1} \vee y_{i_2} \vee \cdots \vee y_{i_{k_i}} \text{ mit } y_{i_j} \in \{x_{i_j}, \bar{x}_{i_j}\}.$$

Konstruiere zu I eine Instanz $f(I)$ von Stable Set, also einen Graphen G und eine Zahl k mit

1. es gibt erfüllende Belegung für I $\Leftrightarrow$ G hat stabile Menge der Grösse k
2. Konstruktion geht in polynomialer Zeit

Konstruktion:

- Klausel $Z_i \rightarrow$ Clique C_i mit k_i Knoten, die den Literalen von Z_i entsprechen (Cliquen disjunkt für verschiedene Klauseln)
- Knoten aus verschiedenen Cliquen werden durch eine Kante verbunden, wenn sich die entsprechenden Literale "widersprechen", also eines Negation des anderen ist
- $k := m$ (Anzahl der Klauseln)

Beispiel: I:

$$\bar{x}_1 \vee x_2 \vee x_3, x_1 \vee \bar{x}_3, x_2 \vee \bar{x}_3, \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$$

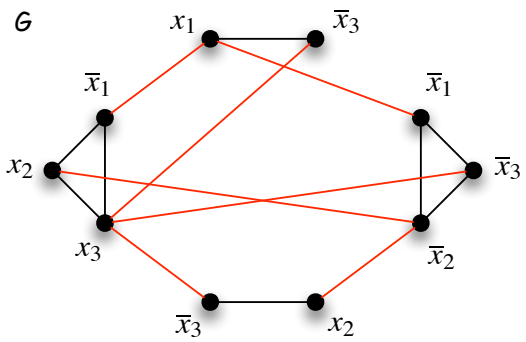


Abbildung 4.1: Konstruktion von G für obiges Beispiel.

Konstruktion geht offenbar in polynomialer Zeit, somit ist (2) erfüllt. Zu (1): es gibt erfüllende Belegung für I $\Leftrightarrow$ G hat stabile Menge der Grösse $k = m$.
$\Leftarrow$:
G habe stabile Menge S mit m Knoten.

- $\Rightarrow$ S enthält aus jeder Clique C genau einen Knoten
- setze die zugehörigen Literale auf TRUE => kein Widerspruch, da in stabiler Menge
- setze andere Literale beliebig, aber konsistent
- $\Rightarrow$ dies liefert erfüllende Belegung

$\Rightarrow$:

- betrachte erfüllende Belegung von $Z_1, ..., Z_m$

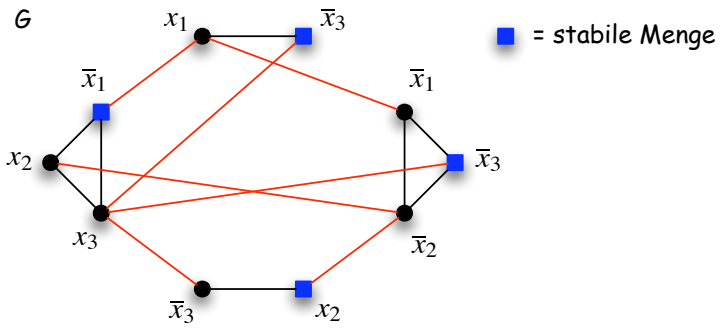


Abbildung 4.2: Konstruktion für $k = m = 4$. Somit werden folgende Werte gesetzt: $x_1 =$ FALSE, $x_2 =$ TRUE, $x_3 =$ FALSE.

- wähle pro Klausel ein Literal mit Wert TRUE
- wähle zugehörige Knoten als Menge S
- $\Rightarrow$ S ist stabil mit m Knoten, da keine Widerspruchskanten in S (wegen erfüllender Belegung)

Im Beispiel ist: $x_1 =$ FALSE, $x_2 =$ FALSE, $x_3 =$ FALSE eine erfüllende Belegung von

$$\bar{x}_1 \vee x_2 \vee x_3, x_1 \vee \bar{x}_3, x_2 \vee \bar{x}_3, \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3.$$

□

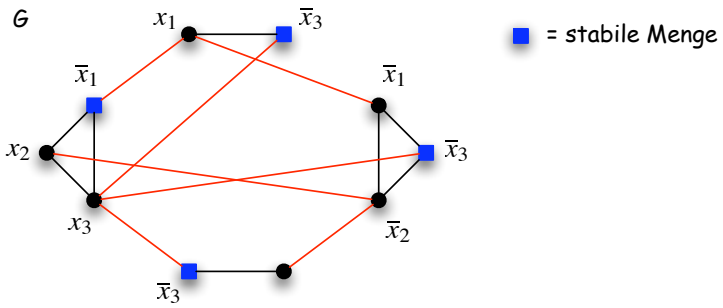


Abbildung 4.3: Konstruktion der stabilen Menge aus einer erfüllenden Belegung.

Wir haben nun also eine NP-vollständige Problemklasse (SAT) auf Stable Set reduziert. Da wir aber jede Problemklasse aus NP auf SAT reduzieren können (Satz 4.19), können wir jetzt auch jede Problemklasse auf Stable Set reduzieren. Damit ist Stable Set selbst ebenfalls NP-vollständig.
Wir haben hierbei den folgenden Satz verwendet:

Satz 4.21. Gelten $\Pi_1 \in \mathsf{NPC}, \Pi_2 \in \mathsf{NP}$ und $\Pi_1 \leq \Pi_2$, so gilt auch $\Pi_2 \in \mathsf{NPC}$.

3SAT
Instanz: Eine Menge von Klauseln aus je genau drei Literalen
Frage: Existiert eine Belegung der Variablen, die alle Klauseln zugleich wahr macht?

Übung 4.22. 3SAT $\in$ NPC.

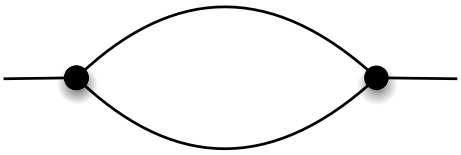
Der Beweis der NP-Vollständigkeit des folgenden Problems zeigt die Verwendung von sogenannten Gadgets für die Reduktion.

Hamilton Circuit (HC)
Instanz: ungerichteter Graph $G = (V, E)$
Frage: Gibt es einen Kreis in G, der jeden Knoten genau einmal besucht?

Satz 4.23. Hamilton Circuit ist NP-vollständig.

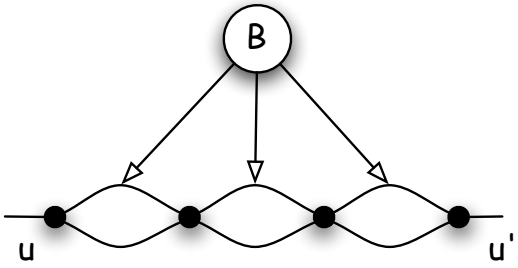
Beweis. (a) HamiltonCircuit $\in$ NP: Zertifikat ist ein Hamiltonkreis.
(b) Transformation 3SAT $\leq$ Hamilton Circuit:
Wir müssen also Instanzen I von 3SAT (Konjunktionen von Klauseln aus drei Literalen) in polynomieller Zeit so in Graphen $G(I)$ kodieren, dass es in $G(I)$ genau dann einen Hamiltonkreis gibt, wenn die Ausgangsinstanz I eine erfüllende Belegung hatte.
Ideen hierzu:

1. Jede Variable kodieren wir durch zwei Knoten, die jeweils nur durch genau eine Kante mit dem übrigen Graphen verbunden sind und untereinander durch zwei parallele Kanten.

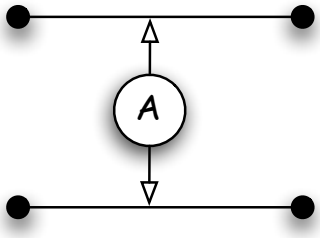
   

   Ein Hamiltonkreis muss nun genau eine der beiden parallelen Kanten durchlaufen. Wir werden das Auswählen der oberen Kante so interpretieren, dass die Variable mit True belegt wurde und das Auswählen der unteren Kante so, dass die Variable mit False belegt wurde.

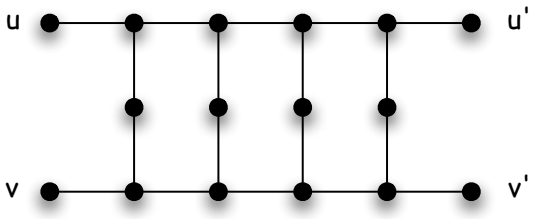
2. Ebenso kodieren wir die drei Literale einer Klausel mit drei solchen Kantenpaaren, wobei erneut die obere Kante eines Parallelenpaares dafür stehen soll, dass das Literal wahr ist und die untere dafür, dass es falsch ist. Hierbei müssen wir sicherstellen, dass der zu einer Klausel gehörende Bereich nur durchlaufen werden kann, wenn wenigstens aus einem Kantenpaar die obere Kante gewählt wird (d.h. mindestens ein Literal wahr ist).



3. Schließlich müssen wir noch dafür sorgen, dass die Variablenbelegungen konsistent mit den Wahrheitswerten der Literale sind, d.h. wenn wir für die Variable x_i die obere Kante (x_i = True) wählen, dann müssen wir in allen Literalen $\overline{x_i}$ die untere Kante wählen (dieses Literal ist falsch) und in allen Literalen x_i die obere Kante.

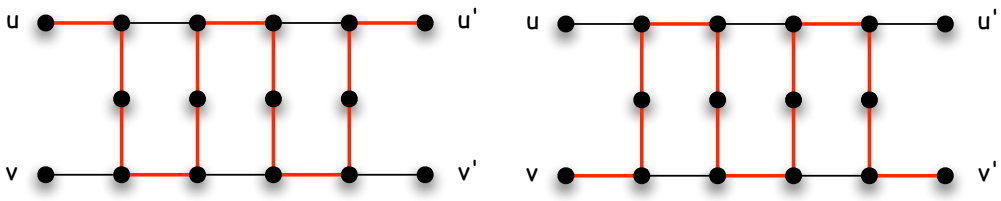


Für 3. verwenden wir das folgende Gadget (das wir ab sofort als „Gadget A“ bezeichnen werden):



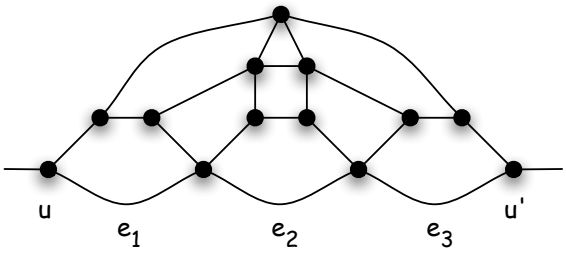
Proposition 4.24. Ist das Gadget A über die Knoten u, u', v und v' in einen größeren Graphen eingebunden, so durchläuft jeder Hamiltonkreis in diesem Graphen das Gadget von u nach u' oder von v nach v'.

Beweis. Die einzigen Möglichkeiten, wie ein Hamiltonkreis das Gadget A ablaufen kann, sind die folgenden beiden Wege:



□

Für 2. verwenden wir ein weiteres Gadget B:

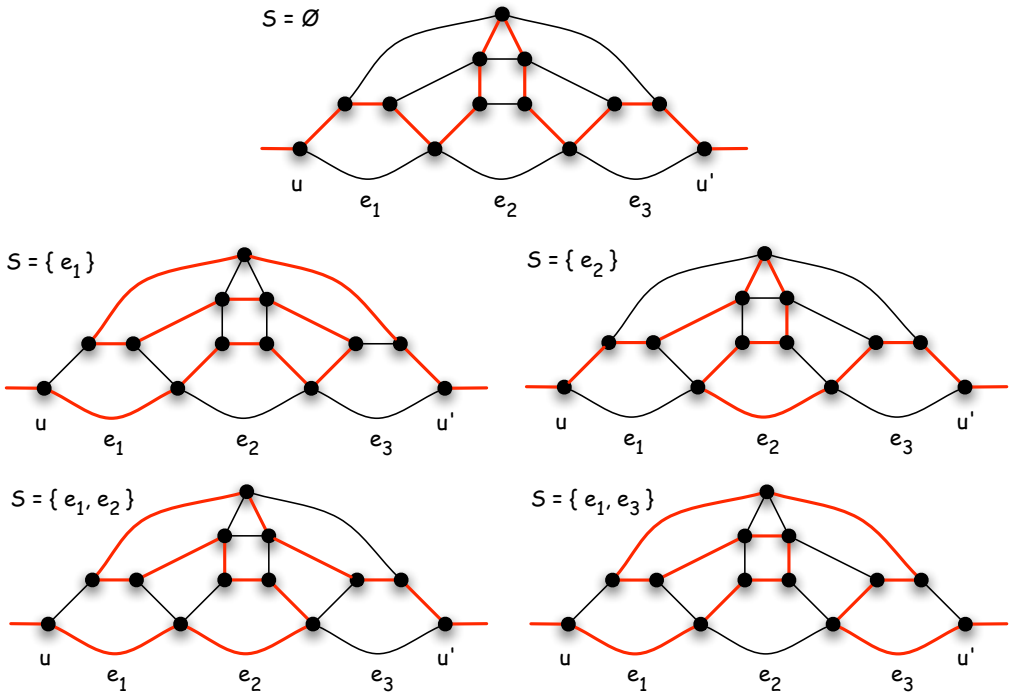


Proposition 4.25. Sei das Gadget B nur über die Knoten u und u' mit einem größeren Graphen verbunden. Gibt es in diesem Graphen einen Hamiltonkreis, so kann dieser das Gadget genau auf eine der folgenden Weisen durchlaufen:

- unter Benutzung keiner der Kanten e_1, e_2, e_3
- unter Benutzung von genau einer der Kanten aus e_1, e_2, e_3
- unter Benutzung von genau zwei der Kanten aus e_1, e_2, e_3

Insbesondere kann es keinen Hamiltonkreis geben, der in einem solchen Gadget alle unteren Kanten verwendet.

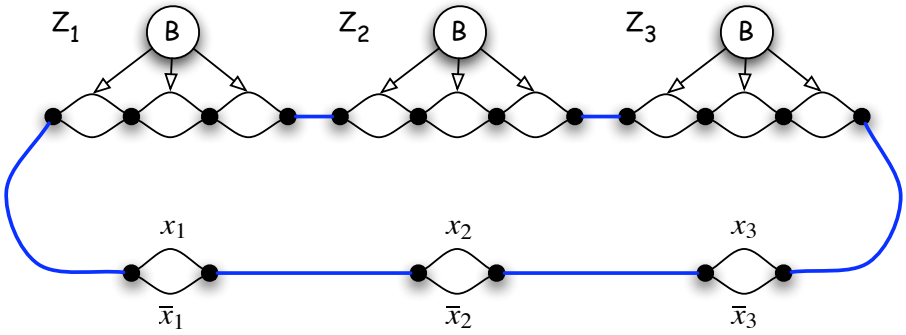
Beweis. Die einzigen möglichen Wege sind



sowie die dazu spiegelsymmetrischen. □

Mit Hilfe dieser beiden Gadgets können wir nun Instanzen von 3SAT (also Konjunktionen von 3-Klauseln) wie folgt in Graphen umwandeln: Für jede auftauchende Variable führen

wir je ein Paar paralleler Kanten ein und verbinden diese Paare (seriell) mit je einer Kante. Für jede Klausel fügen wir eine Kopie des Gadgets B ein und schalten auch diese hintereinander.



Dann verbinden wir noch jede untere Kante aus den Gadgets B mit den entsprechenden Variablenkanten (der oberen, falls das Literal positiv ist, und der unteren, falls es negiert ist).

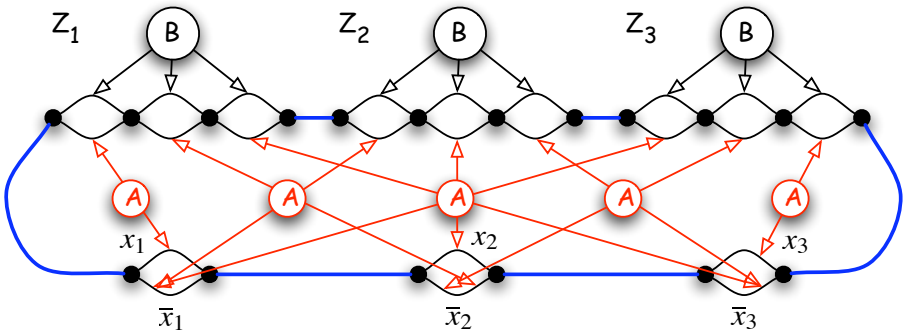


Abbildung 4.4: Der konstruierte Graph zur Instanz $x_1 \vee \overline{x_2} \vee \overline{x_3}, \overline{x_1} \vee x_2 \vee \overline{x_3}, \overline{x_1} \vee \overline{x_2} \vee x_3$.

Klar: Die Reduktion ist in polynomieller Zeit möglich.
Noch zu zeigen: Korrektheit der Reduktion, d.h.

$$I \text{ hat erfüllende Belegung} \iff G(I) \text{ hat Hamiltonkreis}$$

$\Rightarrow$ Sei $\beta : \{\text{Variablen in der Formel}\} \to \{\text{True, False}\}$ eine erfüllende Belegung. Dann gibt es im Graphen $G(I)$ den folgenden Hamiltonkreis: In der unteren Hälfte wird von dem zu einer Variablen x_i gehörenden Kantenpaar genau dann die obere Kante durchlaufen, wenn $\beta(x_i) = \text{True}$ ist (und sonst die untere). Beim Durchlaufen der Gadgets B in der oberen Hälfte muss ebenfalls genau dann die obere Kante eines Kantenpaares durchlaufen werden, wenn dieses Literal durch die Belegung wahr gemacht wird. Da es sich um eine erfüllende Belegung handelt, ist in jeder Klausel mindestens ein Literal wahr, das heißt in jedem Gadget wird bei wenigstens einem der drei Parallelenpaare die obere Kante durchlaufen. Nach Proposition 4.25 kann Gadget B so immer ganz durchlaufen werden. Insgesamt erhalten wir so also einen Hamiltonkreis.

⇐ Ist umgekehrt ein Hamiltonkreis K gegeben, so können wir aus diesem eine I-erfüllende Belegung zurückgewinnen: Dazu setzen wir eine Variable auf True, wenn im entsprechenden Kantenpaar die obere Kante in K enthalten ist, und auf False sonst. Nach Proposition 4.25 durchläuft K in jedem Gadget B mindestens eine obere Kante, d.h. das entsprechende Literal wird durch die gewählte Belegung wahr (wegen Proposition 4.24). Damit ist also jede Klausel erfüllt.

4.8 Die Klasse coNP

In der Klasse NP liegen alle Probleme mit „einfachen“ Ja-Zertifikaten (einfach zu überprüfenden, nicht notwendigerweise einfach zu findenden!). Für Nein-Instanzen hingegen ist es im Allgemeinen überhaupt nicht klar, wie ein entsprechendes Zertifikat aussehen soll (wie etwa können wir belegen, dass es keinen Hamiltonkreis in einen gegebenen Graphen gibt, ohne dazu alle möglichen Kreise aufzulisten?).
Umgekehrt gibt es aber auch Probleme, die zwar vielleicht keine einfachen Ja-Zertifikate besitzen, dafür aber einfache Zertifikate für Nein-Instanzen. Diese Probleme bilden die Komplexitätsklasse coNP:

Definition 4.26 (coNP). Ein Entscheidungsproblem Π ist in coNP, falls es für NEIN-Instanzen polynomiell große Zertifikate gibt, die in polynomieller Zeit verifizierbar sind.

Beispiel 4.27 (Validity (VAL)). Ein Beispiel für ein Problem aus coNP:

VALIDITY (VAL)
Instanz: Eine boolsche Formel F
Frage: Ist F eine Tautologie? (d.h. wahr für jede beliebige Belegung der Variablen)

Ein Nein-Zertifikat für eine Instanz von VAL ist eine Belegung der Variablen, für die die Formel nicht erfüllt ist. Ein derartiges Zertifikat ist offenbar polynomiell und in polynomieller Zeit verifizierbar.

Weitere Beispiele erhalten wir durch das Negieren der Fragestellung in Problemen aus NP - etwa:

coSAT
Instanz: Eine Menge von Klauseln in boolschen Variablen
Frage: Existiert keine Belegung der Variablen, die alle Klauseln zugleich wahr macht?

Analog zur Klasse NP gibt es auch innerhalb von coNP eine Klasse von „schwersten“ Problemen in dieser:

Definition 4.28 (coNPC (coNP-complete)). Sei $\Pi \in \text{coNP}$ eine Problemklasse

$$\Pi \text{ ist coNP-vollständig } \Leftrightarrow A \leq \Pi \text{ für alle } A \in \text{coNP}.$$

coNPC ist die Menge der coNP-vollständigen Probleme.

Ein Beispiel für ein solches Problem ist coSAT, wie direkt aus dem folgenden Satz folgt:

Satz 4.29. Sei Π ein Entscheidungsproblem, dann gilt:

- $\Pi \in \text{NP} \iff \text{co}(\Pi) \in \text{coNP}$
- $\Pi \in \text{NPC} \iff \text{co}(\Pi) \in \text{coNPC}$

Eine interessante Klasse von Problemen ist nun die Schnittmenge von NP und coNP, also Probleme bei denen sowohl Ja- als auch Nein-Instanzen polynomielle Zertifikate haben.

Beispiel 4.30. Das Problem „Gibt es ein Matching der Kardinalität k?“ liegt sowohl in NP als auch in coNP. Ein Ja-Zertifikat ist ein entsprechendes Matching, ein Nein-Zertifikat ein Odd-Set-Cover mit Kapazität $< k$.
Alternative Begründung: Das Problem liegt in P (Edmonds ist polynomiell), also auch in NP und coNP.

Beispiel 4.31. Das folgende Problem liegt ebenfalls in $\text{NP} \cap \text{coNP}$, es ist allerdings nicht bekannt, ob es auch in P liegt (es ist also kein polynomieller Algorithmus dafür bekannt):

IntegerFactorization
Instanz: Eine natürliche Zahl N und eine natürliche Zahl k
Frage: Hat N einen Teiler zwischen 1 und k?

Ein Ja-Zertifikat ist einfach ein entsprechender Faktor. Ein Nein-Zertifikat ist die Angabe der vollständigen Faktorisierung von N in lauter Primfaktoren $> k$ (eine Zahl N hat $O(\log N)$ Primfaktoren und die Primalität kann in Polynomzeit geprüft werden).

4.9 Pseudopolynomielle Algorithmen

Subset Sum
Instanz: natürliche Zahlen $z_1, \ldots, z_N$ und K
Frage: Gibt es eine Teilmenge $S \subseteq \{1, \ldots, N\}$ mit Wert $\sum_{i \in S} z_i = K$?

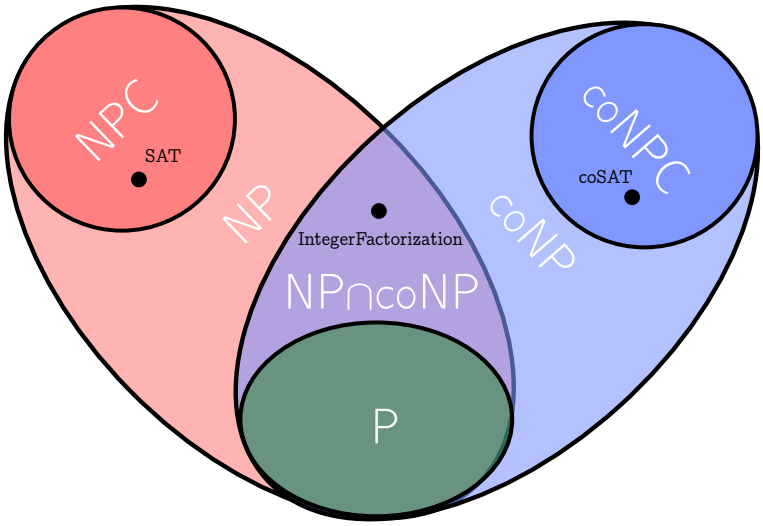


Abbildung 4.5: Eine mögliche Landkarte der Komplexitätsklassen

Fakt 4.32. Subset Sum ist NP-vollständig.

Wir wollen nun einen Algorithmus für Subset Sum finden, der auf dem Prinzip der dynamischen Programmierung basiert: Dies bezeichnet ein „bottom-up"-Verfahren, bei dem zunächst kleine, leicht zu lösende Teilinstanzen („bottom") gelöst werden, mit deren Hilfe man im nächsten Schritt Lösungen für etwas größere Teilinstanzen findet. Dies führt man dann so lange fort, bis man bei einer Lösung der ursprünglichen, großen Instanz („top") angekommen ist.

Im Falle von Subset Sum betrachten wir dazu als Teilinstanzen für (ganzzahlige) $n \leq N$ und $k \leq K$ das Problem $I_{n,k}$: „Gibt es eine Teilmenge $S \subseteq \{1, \ldots, n\}$ mit Wert genau k?"

Die Instanzen $I_{0,k}$ lassen sich leicht lösen: Da wir hierbei keine der Zahlen nutzen dürfen, können wir nur die Summe 0 erreichen. Also ist $I_{0,0}$ eine Ja-Instanz und alle anderen $I_{0,k}$ sind Nein-Instanzen.

Kennen wir nun bereits die Antworten für alle $I_{n,k}$ für ein festes n, so können wir $I_{n+1,k}$ wie folgt beantworten: Die Antwort ist „Ja", wenn $I_{n,k}$ eine Ja-Instanz ist (wir müssen das neue Element also gar nicht nutzen) oder wenn $I_{n,k-z_{n+1}}$ eine Ja-Instanz ist (wir müssen das neue Element nutzen). Andernfalls ist die Antwort „Nein".

Die Berechnung der Antworten aller Teilinstanzen lässt sich gut innerhalb einer Tabelle mit $K+1$ Spalten und $N+1$ Zeilen durchführen, wie das folgende Beispiel verdeutlicht.

Beispiel 4.33. Wir betrachten die Instanz gegeben durch die Zahlen $1, 3, 3, 5, 8, 4$ und den Zielwert $K = 10$. Ordnen wir die Teilinstanzen so in einer Tabelle an, dass die Instanz $I_{n,k}$ in der n-ten Zeile und der k-ten Spalte steht, so können wir die Werte dieser Tabelle mit dem oben beschriebenen Algorithmus Zeile für Zeile berechnen (und müssen dazu immer nur die Einträge der jeweils eins darüber liegenden Zeile betrachten):

$n\downarrow \backslash k\rightarrow$	0	1	2	3	4	5	6	7	8	9	10
0	ja										
1	ja	ja									
2	ja	ja		ja	ja						
3	ja	ja		ja	ja		ja	ja			
4	ja	ja		ja	ja	ja	ja	ja	ja	ja	
5	ja	ja		ja	ja	ja	ja	ja	ja	ja	
6	ja	ja		ja	ja	ja	ja	ja	ja	ja	ja

Die Antwort auf die eigentlich zu lösende Instanz steht dann in der rechten unteren Zelle der Tabelle.

Satz 4.34. Der Algorithmus entscheidet Subset Sum in $O(NK)$.

Beweis. Die Korrektheit zeigen wir durch Induktion über die Zeilen: Sind die Instanzen in der n-ten Zeile bereits richtig entschieden, so entscheidet der Algorithmus auch die Instanzen der $(n+1)$-ten Zeile korrekt. Die Laufzeit von $O(NK)$ ergibt sich daraus, dass die Berechnung eines Tabelleneintrags (d.h. einer Teilinstanz $I_{n,l}$ unter Benutzung der bereits bekannten kleineren Teilinstanzen) in konstanter Zeit erfolgt. □

Auf den ersten Blick mag es nun so scheinen als hätten wir hiermit einen polynomiellen Algorithmus für ein NP-vollständiges Problem gefunden (und damit $P = NP$ gezeigt).
Dies wäre aber nur dann der Fall, wenn die Kodierung einer Instanz von Subset Sum eine Länge von NK hätte (oder zumindest NK polynomiell in dieser Länge wäre). Tatsächlich hat die (binäre) Kodierung einer Instanz aber lediglich eine Länge von etwa

$$\sum_{i=1}^{N} \log_2(z_i) + \log_2(K) \leq N \log_2(\max\{z_i\}) + \log_2(K) \leq (N+1)\log_2(\text{number}(I)),$$

(wobei $\text{number}(I)$ den maximalen Betrag einer in I auftretenden Zahl bezeichnet). Und im Allgemeinen ist NK nicht polynomiell (sondern sogar exponentiell) in dieser Kodierungslänge.
Unser Algorithmus für Subset Sum ist also nicht polynomiell in $\langle I\rangle$, er ist allerdings polynomiell in $\langle I\rangle$ und $\text{number}(I)$. Wir bezeichnen derartige Algorithmen als pseudopolynomiell:

Definition 4.35. Ein Algorithmus A für ein Problem Π heißt pseudopolynomiell, wenn seine Laufzeit polynomiell in der Größe der Eingabeinstanz **und** im größten Betrag einer in der Eingabe auftretenden Zahl ist.

$$\exists \text{Polynom } p : \forall I \in \Pi : T_A(I) \leq p(\langle I\rangle, \text{number}(I))$$

Schränkt man sich also auf Instanzen mit „kleinen Zahlen“ ein, so wird ein pseudopolynomieller Algorithmus zu einem polynomiellen.

Definition 4.36. Sei Π ein Entscheidungsproblem und q ein Polynom. Dann besteht die Restriktion Π_q von Π aus allen Instanzen $I \in \Pi$ mit $\text{number}(I) \leq q(\langle I \rangle)$.

Definition 4.37. Ein Entscheidungsproblem $\Pi \in \text{NPC}$ heißt stark NP-vollständig, wenn es ein Polynom q gibt, sodass Π_q NP-vollständig ist. Π heißt schwach NP-vollständig, falls es kein solches Polynom gibt.

Bemerkung 4.38. TSP ist stark NP-vollständig (es werden in der Reduktion nur Gewichte 1 benutzt).

Satz 4.39. Falls $P \neq NP$, so kann kein stark NP-vollständiges Problem durch einen pseudopolynomialen Algorithmus gelöst werden.

Beweis. Angenommen P_0 wird durch einen pseudopolynomialen Algorithmus A gelöst. A ist polynomial in $\langle I \rangle$ und $\text{number}(I)$, d.h. es gibt ein Polynom p mit $T_A(I) \leq p(\langle I \rangle, \text{number}(I))$. Betrachte nun die Restriktion $P_{0,q}$ von P_0 für ein Polynom q. Für eine Instanz $I \in P_{0,q}$ gilt dann $\text{number}(I) \leq q(\langle I \rangle)$ und damit

$$T_A(I) \leq p(\langle I \rangle, \text{number}(I)) \leq p(\langle I \rangle, q(\langle I \rangle)) = \text{ Polynom in } \langle I \rangle.$$

A löst also $P_{0,q}$ in polynomialer Zeit, also $P = NP$, Widerspruch. □

Für TSP gibt es also keinen pseudopolynomialen Algorithmus (oder $P = NP$).

Kapitel 5

Approximationsalgorithmen

Definition 5.1. Ein Approximationsalgorithmus A für ein Optimierungsproblem P_0 berechnet für jede Instanz $I \in P_0$ eine Lösung $A(I) \in S_I$ in einer Zeit, die polynomial in $\langle I \rangle$ ist. Wir bezeichnen mit $A(I)$ sowohl die Lösung, wie auch den Wert der Lösung, aus dem Kontext ist immer klar, was gemeint ist. Der Approximationsalgorithmus A hat eine Gütegarantie $\rho \geq 1$, falls gilt

$$A(I) \leq \rho \cdot \mathrm{OPT}(I) \text{ für alle } I \in P_0.$$

Hier bezeichnet $\mathrm{OPT}(I)$ den Wert einer Optimallösung von I (Minimierungsprobleme). Für Maximierungsprobleme gilt für einen Approximationsalgorithmus A mit Gütegarantie $0 \leq \rho \leq 1$

$$A(I) \geq \rho \cdot \mathrm{OPT}(I) \text{ für alle } I \in P_0.$$

5.1 Metrisches TSP

Metrisches TSP
Instanz:

- vollständiger Graph $K_n[V]$ mit $V = \{1, \ldots, n\}, n \geq 3$
- Kantenbewertung c_{ij} mit rationalen Zahlen, für die gilt
  - Dreiecksungleichung

    $$c_{ij} \leq c_{ik} + c_{kj} \text{ für alle } i, j, k$$

  - Symmetrie

    $$c_{ij} = c_{ji} \text{ für alle } i, j$$

  - $c_{ii} = 0$ (hieraus folgt $c_{ij} \geq 0$)

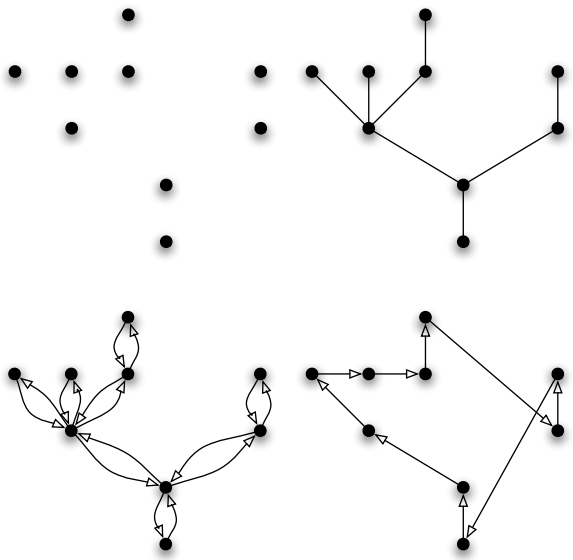
Aufgabe: Bestimme einen Hamilton Kreis C mit minimaler Länge.

Bemerkung 5.2. Metrisches TSP ist stark NP-vollständig.

Eingabe: $K_n[V], c$.
Ausgabe: Ein Hamilton Kreis C mit Approximationsgüte $\rho \leq 2$.
Methode:

1. berechne einen MST T von $K_n[V]$ bzgl. c mit Wurzelknoten r aus V
2. Verdopple Kanten in $T \rightarrow T_d$ (Graph T_d ist Eulersch!)
3. berechne eine Eulertour in T_d
4. gehe entlang der Eulertour ausgehend von r (benutze DFS mit "left before right" als tie-breaking Regel) und nehme Abkürzungen sobald ein Knoten schon besucht wurde. Dies ergibt eine Tour $MST(I)$.
5. Ausgabe: $MST(I)$

Algorithm 5.1.1: Doppelbaum-Algorithmus (Rosenkrantz, Stearns, Lewis 1977)



Satz 5.3. Der Doppelbaum-Algorithmus ist eine 2-Approximation.

Beweis. Sei T^* eine optimale Tour.

$$\begin{aligned} c(MST(I)) &\leq c(T_d) \text{ (Dreiecksungleichung)} \\ &= 2c(T) \text{ (Eulertour)} \\ &\leq 2c(T^*) \text{ ($T^* - \{e\}$ ist ein Baum)} \end{aligned}$$

□

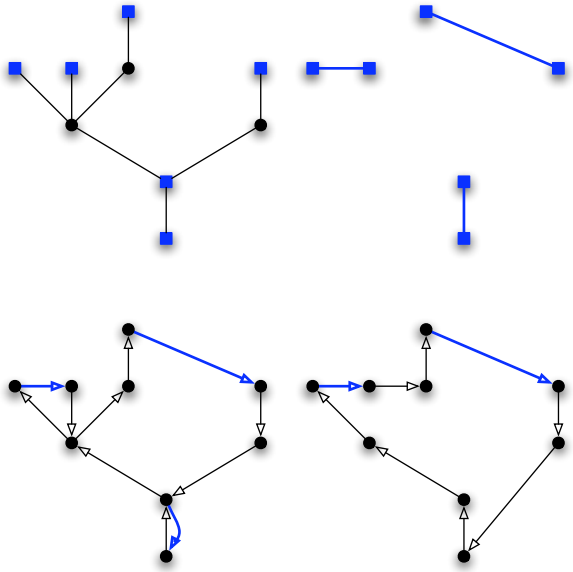
5.2 Christofides' Algorithmus

Wir ersetzen Schritt (2) durch

(2a) berechne in T die Menge U von Knoten mit ungeradem Grad ($\Rightarrow |U|$ ist gerade)

(2b) berechne ein perfektes Matching M mit minimalem Gewicht bzgl. c_{ij} in $K_n[U]$ (geht in polynomialer Zeit, ohne Beweis)

(2c) füge Matchingkanten zu T hinzu (der resultierende Graph ist Eulersch!)



Satz 5.4. Der Algorithmus von Christofides ist ein polynomieller 3/2-Approximationsalgorithmus.

Beweis. Behauptung: $c(M) \leq OPT(I)/2$.

- gehe entlang optimaler Tour T^* und lasse Knoten $v \notin U$ aus
- Wir erhalten eine Tour (ggf. unter Benutzung von Abkürzungen) T_U in $K_n[U]$
- Dreiecksungleichung: $c(T_U) \leq c(T^*)$
- T_U definiert 2 perfekte Matchings M_1 und M_2 in $K_n[U]$
- (benutze jede zweite Kante in T_U; das ist möglich, da $|U|$ gerade ist)
- für das berechnete Matching M gilt:

$$2c(M) \leq c(M_1) + c(M_2) = c(T_U) \leq c(T^*) = OPT(I)$$
$$\Rightarrow c(\text{Christofides}(I)) \leq c(T) + c(M) \leq (3/2) \cdot OPT(I).$$

□

5.3 Approximationsalgorithmen mit absoluter Gütegarantie

Wir betrachten nun Probleme, die sich bis auf einen additiven Fehler approximieren lassen. Der Prototyp eines solchen Problems ist Edge Coloring.

> Edge Coloring
> Instanz: ungerichteter Graph G
> Frage: Finde eine Kantenfärbung von G mit minimaler Anzahl von Farben

Eine Kantenfärbung mit k Farben ist eine Abbildung $f : E(G) \to \{1, \ldots, k\}$ mit e, e' haben gemeinsamen Endpunkt $\Rightarrow f(e) \neq f(e')$. Das kleinste k, für das eine solche Kantenfärbung möglich ist, heißt dann die Kanten-chromatische Zahl von G.

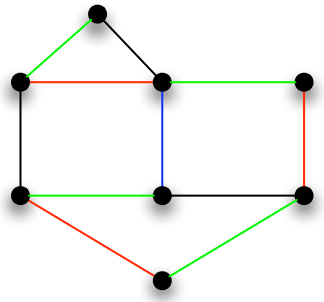


Abbildung 5.1: Beispielgraph mit einer Kantenfärbung mit 4 Farben.

Satz 5.5 (Vizing 1964). Ein einfacher ungerichteter Graph G mit Maximalgrad d hat eine Kantenfärbung mit maximal $d + 1$ Farben, und eine solche kann in polynomialer Zeit konstruiert werden.

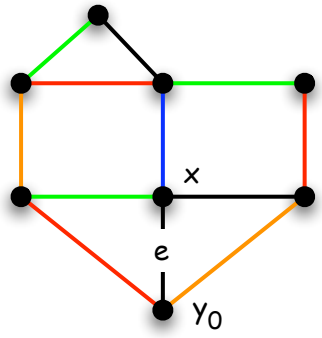
Beweis. Induktion nach $m := |E(G)|$.
Trivial für $m = 0$. Sei also $m > 0$ und e eine Kante von G.

- Falls $G - e$ einen kleineren Maximalgrad als G hat, so existiert nach Induktionsvoraussetzung eine Kantenfärbung mit d Farben für $G - e$ und wir können die Farbe $d + 1$ für die Kante e verwenden.
- Falls $G - e$ denselben Maximalgrad wie G hat, so existiert nach Induktionsvoraussetzung eine Kantenfärbung mit Farben $\{1, 2, \ldots, d + 1\}$ für $G - e$. Wir müssen jetzt $G - e$ geeignet "umfärben" um für e eine freie Farbe zu bekommen.

In $G - e$ ist in jedem Knoten v eine Farbe unbenutzt (da $d + 1$ Farben vorhanden, aber $d(v) \leq d$). Wir konstruieren zu $e = (x, y_0)$ eine maximale Folge von Kanten $(x, y_0), (x, y_1), \ldots$ und eine Folge von Farben $c_0, c_1, \ldots$ wie folgt:

- c_i ist eine fehlende Farbe bei y_i in $G - e$
- (x, y_{i+1}) ist eine Kante mit Farbe c_i in $G - e$

Beim Abbruch ergeben sich folgende Fälle, für die wir jeweils umfärben.
Abbruch mit $k = i$, falls Farbe c_k bei x nicht auftritt oder $c_j = c_k$ für ein $j < k$.



$G - e$ und G haben beide Maximalgrad 4, Färbung mit 5 Farben ist zulässig in $G - e$.
Mögliche Folge (x, y_{i+1}) mit Farben c_i in $G - e$

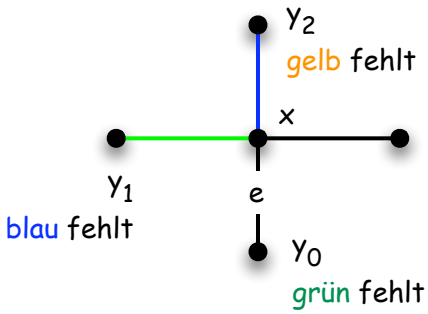


Abbildung 5.2: Gelb fehlt bei x.

Fall 1:
Falls Farbe c_k bei x nicht auftritt und $c_j \neq c_k$ für alle $j < k$, so färbe Kante (x, y_i) mit Farbe $c_i, i = 0, \ldots, k$.
Dies ergibt zulässige Färbung nach Konstruktion der Farben c.

Fall 2:
Falls Farbe $c_j = c_k$ bereits für ein $j < k$, so färbe Kante (x, y_i) mit Farbe $c_i, i = 0, \ldots, j-1$ und lasse (x, y_j) ungefärbt. Dann fehlt Farbe c_k (etwa rot) bei y_j und bei y_k.
Sei blau eine fehlende Farbe bei x (rot existierte bei x, da die Folge sonst bereits bei $k = j$ abgebrochen). Wähle dann den ersten zutreffenden der folgenden Fälle zum Umfärben

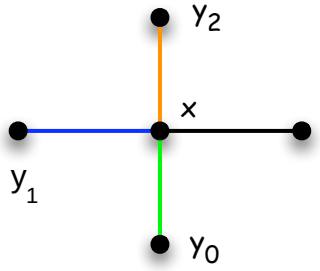


Abbildung 5.3: Gefärbte Kanten nach Tausch im Fall dass die Farbe "Gelb" bei x nicht auftritt.

1. falls blau bei y_j fehlt, so färbe (x, y_j) blau

2. falls blau bei y_k fehlt, so färbe (x, y_i) mit $i = j, \ldots, k-1$ mit Farbe c_i und (x, y_k) blau.
   Beachte, dass die Kanten (x, y_i) mit $i = j, \ldots, k-1$ nicht blau sind wenn man in diesen Fall kommt.

3. betrachte den Teilgraphen aus allen roten und blauen Kanten. In ihm hat (wegen Kantenfärbung) jeder Knoten Grad ≤ 2. Also sind seine Zusammenhangskomponenten elementare Kreise und elementare Wege. Die Knoten x, y_j, und y_k sind Endknoten von Wegen (da bei ihnen blau bzw. rot fehlt, aber nicht beides). Also liegen sie nicht alle 3 in derselben Zusammenhangskomponente. Wähle eine Zusammenhangskomponente, die genau einen dieser 3 Knoten enthält, und tausche rot und blau in dieser Komponente. Somit trifft nun einer der vorigen Fälle zu.

□

Korollar 5.6. Für Edge Coloring existiert ein absoluter Approximationsalgorithmus mit Fehler 1.

Beweis. Der Algorithmus hinter dem Satz von Vizing liefert eine Kantenfärbung mit Wert $\leq d + 1$. Der Maximalgrad d ist eine untere Schranke. □

5.4 Approximationsschemata

Definition 5.7. Ein Approximationsschema für eine Problemklasse P_0 (mit Kosten $c \geq 0$) ist ein Algorithmus A, der als Input eine Instanz $I \in P_0$ und ein $\epsilon > 0$ erhält und für jedes ϵ ein $(1+\epsilon)$-Approximationsalgorithmus ist (äquivalent: ist eine Familie von Algorithmen A_ϵ, so dass jedes A_ϵ ein $(1+\epsilon)$-Approximationsalgorithmus ist).
Bezeichnung: PTAS für Polynomial Time Approximation Scheme.

Definition 5.8. Ein Approximationsschema heisst voll polynomial :$\Leftrightarrow$ seine Laufzeit ist polynomial in $\langle I \rangle$ und $1/\epsilon$.
Bezeichnung: FPTAS für Fully Polynomial Time Approximation Scheme.

Zum Unterschied PTAS und FPTAS: die Approximationsgüte $(1+\epsilon)$ ist in beiden Fällen die gleiche, der Unterschied liegt in der Laufzeit. Ein PTAS ist für jedes feste ϵ polynomial in $\langle I \rangle$, kann aber exponentiell in $1/\epsilon$ sein, wie z.B. $O(n^{2/\epsilon})$. Ein FPTAS ist polynomial in $\langle I \rangle$ und $1/\epsilon$, wie z.B. $O(\frac{n^2}{\epsilon})$.
Wir leiten nun ein FPTAS für Knapsack her.

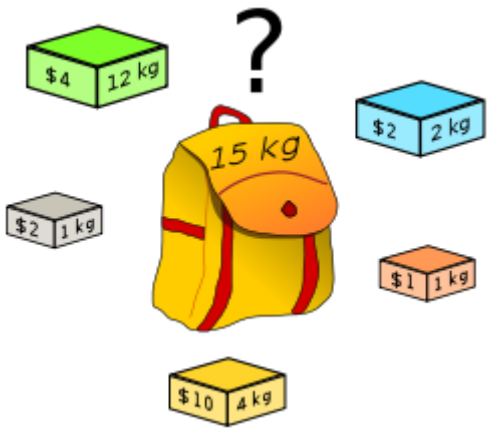


Abbildung 5.4: Illustration von Knapsack. Ziel ist es, den Rucksack mit Items zu füllen so dass der Wert des Rucksacks (Summe der Einzelwerte der gepackten Items) maximal ist. Quelle des Bildes: Wikipedia.

Knapsack
Input: $I = (N, w, p, W)$

- Items $N = \{1, \ldots, n\}$
- Jedes Item $i \in N$:
  - Gewicht $w_i \in \mathbb{Z}_+$
  - Profit $p_i \in \mathbb{R}_+$
- Rucksackkapazität $W \in \mathbb{Z}_+$

Aufgabe: Finde $S^* \subseteq N$ mit

$$S^* \in \arg\max_{S \subseteq N} \sum_{i \in S} p_i$$
$$\text{u.d.N.} : \sum_{i \in S} w_i \leq W.$$

Wir werden zunächst das Prinzip der dynamischen Programmierung benutzen.
Grundprinzip:

1. Struktureinsicht von Optimallösungen (Optimale Substruktur)
2. Rekursive Darstellung des Optimalwertes
3. Auflösung der Rekursion (Bottom-Up)
4. Rekonstruktion der Optimallösung

Bemerkung 5.9. Optimale Substrukturen sind uns schon beim kürzeste Wege Problem begegnet.

- Sei $I = (N, w, p, W)$ Instanz von KNAPSACK
- Für $\{1, \dots, i\} \subseteq N$ und $j \leq W$: $[i, j]$ induzierte Instanz
- Also $I = [n, W]$
- Sei $S^* = \{i_1, \dots, i_k\} \subseteq N, i_1 \leq i_2 \leq \cdots \leq i_k$ optimal

Lemma 5.10.
- Sei $i_k \in S^*$.
- $S^* \setminus \{i_k\}$ ist optimal für $[i_k - 1, W - w_{i_k}]$.

Beweis: Per Widerspruch (Übung).

Datenstruktur: Array $V[i, j], 0 \leq i \leq n, 0 \leq j \leq W$

- Sei $V[i, j]$ der Optimalwert einer induzierten Instanz $[i, j]$, d.h., mit Items $\{1, \dots, i\}$ und Kapazität $0 \leq j \leq W$.

Rekursionsformel:

$$V[i, j] = \max\{V[i-1, j], p_i + V[i-1, j-w_i]\},$$
$$V[0, j] = 0, \quad 1 \leq i \leq n, 0 \leq j \leq W$$

Beweis der Rekursion:

- 1. Fall: i ist nicht in der Optimallösung für $[i, j]$
  - Optimalwert bleibt $V[i-1, j]$ bei Entfernung von i.
- 2. Fall: i ist Teil der Optimallösung für $[i, j]$
  - i verbraucht w_i Kap., daher ist beste Lösung ohne i gegeben durch Optimalwert $V[i-1, j-w_i]$.

Items \ Gewicht W	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2
2	0	0	2	2	5	5	7
3	0	2	2	4	5	7	7
4	0	2	3	5	5	7	8

Auflösung der Rekursion:

- Betrachte Matrix $V[i,j],\ 0 \leq i \leq n, 0 \leq j \leq W$
- Erste Zeile $V[0,j] = 0$ für alle $0 \leq j \leq W$

$$V[i,j] = \max\{V[i-1,j], p_i + V[i-1, j-w_i]\}$$

Wir geben nun ein Beispiel.

Beispiel 5.11. Betrachte folgende Instanz von Knapsack.

$$N = \{1,2,3,4\}, W = 6$$
$$p_1 = 2, p_2 = 5, p_3 = 2, p_4 = 3$$
$$w_1 = 2, w_2 = 4, w_3 = 1, w_4 = 2$$

Es gilt die Einträge der Matrix $V[i,j]$ zu berechnen. Die erste Zeile ergibt sich mit $V[0,j] = 0$ für alle $0 \leq j \leq W$.

$$V[1,1] = 0, V[1,2] = 2, V[1,3] = 2, V[1,4] = 2, V[1,5] = 2, V[1,6] = 2$$
$$V[2,4] = \max\{V[1,4], p_2 + V[1,4-4]\} = 5 + V[1,0] = 5$$
$$V[2,6] = \max\{V[1,6], p_2 + V[1,6-4]\} = 5 + V[1,2] = 7$$
$$V[3,3] = \max\{V[2,3], p_3 + V[2,3-1]\} = 2 + V[2,2] = 4$$
$$V[3,4] = \max\{V[2,4], p_3 + V[2,4-1]\} = V[2,4] = 5$$

Eingabe: Instanz $I = (N, w, p, W)$.
Ausgabe: Optimalwert $V[n, W]$.
Initialisierung: $V[0, j] := 0$ für $j = 0, \ldots, W$
Methode:
for $i = 1, \ldots, n$ **do**
  for $j = 0, \ldots, W$ **do**
    if $w_i \leq j$ **then**
      $V[i, j] \leftarrow \max\{V[i-1, j], p_i + V[i-1, j-w_i]\}$
    end
    else
      $V[i, j] \leftarrow V[i-1, j]$
    end
  end
end
return $V[n, W]$

Algorithm 5.4.1: Dynamisches Programm

Rekonstruktion der Optimallösung: Array von Binärwerten $x[i, j] \in \{0, 1\}$

$$x[i, j] = \begin{cases} 1, & \text{falls item } i \text{ Teil der Optimallösung von } [i, j] \text{ ist,} \\ 0, & \text{sonst.} \end{cases}$$

Implementation:

```
...
if w_i ≤ j then
  if p_i + V[i−1, j−w_i] > V[i−1, j] then
    V[i, j] ← p_i + V[i−1, j−w_i]
    x[i, j] ← 1
  end
  else
    V[i, j] ← V[i−1, j]
    x[i, j] ← 0
  end
end
...
```

Rekonstruktion von S^*:

Initialisiere $j \leftarrow W$, $S^* = \emptyset$.
for $i = n, \ldots, 1$ **do**
  if $x[i, j] == 1$ **then**
    $S^* \leftarrow S^* \cup \{i\}$
    $j \leftarrow j - w_i$
  end
end
return S^*

Satz 5.12 (Pseudopolynomieller Algorithmus). Der Algorithmus hat eine Laufzeit von $O(n \cdot W)$.

Bemerkung 5.13. Mit Binärkodierung von W (mit Grösse $\log_2(W)$) ist die Laufzeit exponentiell in $\log_2(W)$.

Bemerkung 5.14. KNAPSACK ist NP-vollständig, daher ist ein vollständig polynomieller Algorithmus nicht zu erwarten.

Nun stellen wir einen alternativen pseudopolynomiellen Algorithmus für ganzzahlige Profite $p_i \in \mathbb{Z}_+$ vor. Sei

$$P := \max_{i \in N | w_i \leq W} p_i$$

der Profit eines profitabelsten Items. Wir erhalten sofort, dass

$$OPT(I) \leq n \cdot P$$

gelten muss. Für jedes $i \in \{0, 1, \ldots, n\}$ und $p \in \{0, 1, \ldots, nP\}$, bezeichne $A[i, p]$ das kleinstmögliche Gewicht einer Teilmenge von $\{1, \ldots, i\}$ (bzw. von $\emptyset$ für $i = 0$) die genau Profit p hat, falls eine solche Teilmenge existiert; sonst setze $A[i, p] = \infty$. Für $A[i, p] < \infty$ sei $S_{i,p}$ eine zugehörige Menge.
Für $A[i, p]$ haben wir wieder den Basisfall $A[0, p]$, wobei $A[0, 0] = 0$ und alle anderen Werte sind ∞. Wir bekommen folgende Rekurrenz:

$$A[i, p] = \begin{cases} \min\{A[i-1, p], w_i + A[i-1, p-p_i]\}, & \text{falls } p_i \leq p \\ A[i-1, p], & \text{sonst.} \end{cases}$$

Satz 5.15. Es gibt einen pseudopolynomiellen Algorithmus der KNAPSACK in Zeit $O(n^2P)$ löst.

Beweis. Die optimale Menge ist durch die Menge $S_{n,p}$ gegeben, bei der p maximal ist und für die gilt $A[n, p] \leq W$. Da wir $(n+1)(nP+1)$ viele Werte $A[i, p]$ berechnen müssen, bekommen wir eine Gesamtlaufzeit von $O(n^2P)$. □

Nun kommen wir zum angekündigten FPTAS für KNAPSACK. Der Algorithmus funktioniert folgendermassen:

Gegeben: $I = (N, w, p, W)$, $\epsilon > 0$.
Setze $K := \frac{\epsilon P}{n}$
for $i = 1, \ldots, n$ **do**
  $p'_i := \lfloor \frac{p_i}{K} \rfloor$
end
Wende den pseudopolynomiellen Algorithmus aus Satz 5.15 auf Instanz $I' = (N, w, p', W)$ an mit Ausgabe S'
return S'

Lemma 5.16. Der Algorithmus berechnet eine Lösung S', für die gilt:

$$\sum_{i \in S'} p_i \geq (1-\epsilon) \cdot \mathrm{OPT}(I).$$

Die Laufzeit des Algorithmus ist $O(\frac{n^3}{\epsilon})$.

Beweis. Sei S^* eine optimale Lösung für I. Weil alle p_i's durch K geteilt und dann abgerundet wurden um p'_i zu erhalten, gilt für alle $i \in N$:

$$p'_i = \lfloor \frac{p_i}{K} \rfloor \Rightarrow p'_i \leq \frac{p_i}{K} \Leftrightarrow K p'_i \leq p_i, \tag{5.1}$$

und

$$p'_i = \lfloor \frac{p_i}{K} \rfloor \Rightarrow p'_i \geq \frac{p_i}{K} - 1 \Leftrightarrow K p'_i \geq p_i - K \Leftrightarrow p_i - K p'_i \leq K. \tag{5.2}$$

Somit folgt aus (5.2):

$$\sum_{i \in S^*} p_i - K \sum_{i \in S^*} p'_i \leq n \cdot K. \tag{5.3}$$

Aufgrund der Optimalität von S' bzgl. p' gilt

$$\sum_{i \in S^*} p'_i \leq \sum_{i \in S'} p'_i.$$

Insgesamt erhalten wir

$$\begin{aligned}
\sum_{i \in S'} p_i &\underset{(5.1)}{\geq} K \sum_{i \in S'} p'_i \\
&\geq K \sum_{i \in S^*} p'_i \\
&\underset{(5.3)}{\geq} \sum_{i \in S^*} p_i - nK \\
&= \mathrm{OPT}(I) - \epsilon P \\
&\geq (1-\epsilon) \cdot \mathrm{OPT}(I),
\end{aligned}$$

wobei sich die vorletzte Gleichung aus $\mathrm{OPT}(I) \geq P$ ergibt (beachte, dass es eine zulässige Lösung mit Wert P gibt).
Die Laufzeit ergibt sich durch:

$$O(n^2 P') = O\left(n^2 \frac{P}{K}\right) = O\left(n^2 \frac{P}{\frac{\epsilon P}{n}}\right) = O\left(\frac{n^3}{\epsilon}\right).$$

□

5.5 LP-basierte Approximation für Standortprobleme

Wir betrachten folgendes Standortproblem beschrieben durch $I = (\mathcal{C}, \mathcal{F}, c, f)$: Es gibt eine Menge $\mathcal{C}$ von Kunden und eine Menge $\mathcal{F}$ von möglichen Standorten. Die Öffnungskosten von Standort i betragen $f_i \geq 0, i \in \mathcal{F}$. Es gibt eine Distanzmatrix $c_{ij}, i \in \mathcal{F}, j \in \mathcal{C}$ zwi-

schen jedem Paar $(i,j) \in \mathcal{F} \cup \mathcal{C}$ welche die Anbindungskosten von Kunde j zu Standort i beschreibt.

Wir geben nun eine lineare ganzzahlige Formulierung für das Problem an:

$$\min \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}, j\in\mathcal{C}} c_{ij} x_{ij} \tag{IP}$$

$$\sum_{i\in\mathcal{F}} x_{ij} = 1 \qquad \forall j \in \mathcal{C} \tag{5.4}$$

$$y_i - x_{ij} \geq 0 \qquad \forall i \in \mathcal{F}, j \in \mathcal{C} \tag{5.5}$$

$$y_i, x_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{F}, j \in \mathcal{C}$$

Wir bezeichnen das zugehörige relaxierte lineare Programm mit (LP). Das zu (LP) duale Programm ist gegeben durch:

$$\max \sum_{j\in\mathcal{C}} v_j \tag{D}$$

$$w_{ij} \geq v_j - c_{ij} \qquad \forall i \in \mathcal{F}, j \in \mathcal{C} \tag{5.6}$$

$$\sum_{j\in\mathcal{C}} w_{ij} \leq f_i \qquad \forall i \in \mathcal{F} \tag{5.7}$$

$$w_{ij} \geq 0 \qquad \forall i \in \mathcal{F}, j \in \mathcal{C} \tag{5.8}$$

Wir werden im Folgenden annehmen, dass die Anbindungskosten die sogenannte Dreiecksungleichung erfüllen.

Definition 5.17. Anbindungskosten erfüllen die Dreiecksungleichung, falls für alle $j, l \in \mathcal{C}$ und $i, k \in \mathcal{F}$ gilt

$$c_{ij} \leq c_{il} + c_{kl} + c_{kj}.$$

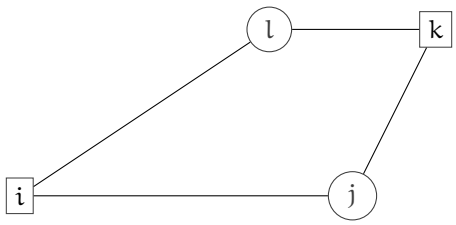


Abbildung 5.5: Illustration der Dreiecksungleichung.

Wir bereiten nun einen LP-basierten Approximationsalgorithmus vor.

Definition 5.18. Sei (x^*, y^*) eine optimale LP-Lösung. Wir bezeichnen i als Nachbarstandort von Kunde j, falls gilt $x^*_{ij} > 0$. Wir bezeichnen mit

$$N(j) = \{i \in \mathcal{F} \mid x^*_{ij} > 0\}$$

die Nachbarn von j bzgl. x^*.

Wir erhalten folgendes Lemma.

Lemma 5.19. Sei (x^*, y^*) und (v^*, w^*) ein Paar primal-dualer Optimallösungen. Dann gilt:

$$x^*_{ij} > 0 \Rightarrow c_{ij} \leq v^*_j.$$

Beweis. Die komplementären Schlupf-Bedingungen implizieren:

$$x^*_{ij} > 0 \Rightarrow w^*_{ij} = v^*_j - c_{ij}.$$

Weil nun $w^*_{ij} \geq 0$ gilt, folgt die Behauptung. □

Um den anschliessenden Approximationsalgorithmus zu beschreiben benötigen wir noch einen weiteren Nachbarschaftsbegriff.

Definition 5.20. Die Menge $N^2(j)$ bezeichne alle benachbarten Kunden der benachbarten Standorte von Kunde j. Formal:

$$N^2(j) := \{k \in \mathcal{C} \mid \text{Kunde } k \text{ ist ein Nachbar eines Standortes } i \in N(j)\}.$$

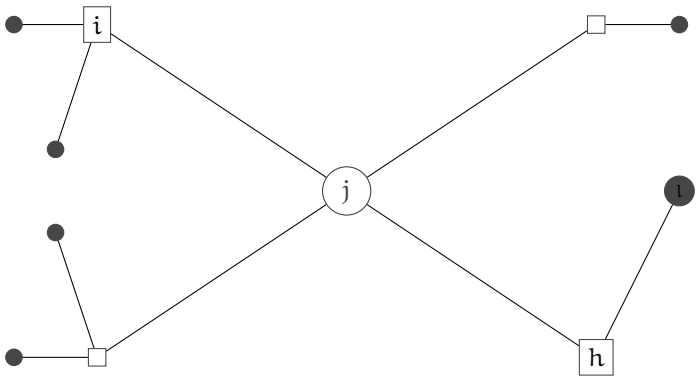


Abbildung 5.6: Illustration der Nachbarschaftsmengen $N(j)$ und $N^2(j)$. Quadrate repräsentieren Standorte und Kreise entsprechen Kunden. Die Menge der grau gefärbten Kunden entspricht der Menge $N^2(j)$.

Gegeben: $I = (\mathcal{C}, \mathcal{F}, c, f)$.
Löse das primale und duale LP mit Lösungen (x^*, y^*) und (v^*, w^*)
$C \leftarrow \mathcal{C}$
$k \leftarrow 0$
while $C \neq \emptyset$ **do**
  $k \leftarrow k + 1$
  Wähle $j_k \in C$ mit minimalem v_j^*
  Wähle $i_k \in N(j_k)$ als günstigsten Standort in $N(j_k)$
  Verbinde j_k und alle unverbundenen Kunden in $N^2(j_k)$ mit i_k
  $C \leftarrow C - \{j_k\} - N^2(j_k)$
end
return $\cup\{i_k\}$ und Assignments von Kunden

Algorithm 5.5.1: LP-basierter Approximationsalgorithmus

Satz 5.21. Der LP-basierte Approximationsalgorithmus berechnet in polynomieller Zeit eine 4-approximative Lösung.

Beweis. Wir schätzen zunächst die Anbindungskosten der gewählten Kunden j_k zu den assoziierten Standorten i_k ab. Mit Lemma 5.19 gilt

$$c_{i_k,j_k} \leq v_{j_k}^*.$$

Wir können nun auch die Öffnungskosten von i_k beschränken:

$$f_{i_k} = f_{i_k} \sum_{i \in N(j_k)} x_{i,j_k}^* \leq \sum_{i \in N(j_k)} f_i x_{i,j_k}^*.$$

Hierbei haben wir für die erste Gleichung die Bedingung (5.4) benutzt und für die zweite Ungleichung die Minimalität von i_k bzgl. der Öffnungskosten f_i. Mit der Bedingung (5.5) $(x_{ij}^* \leq y_i^*)$ folgt weiterhin

$$f_{i_k} \leq \sum_{i \in N(j_k)} f_i x_{i,j_k}^* \leq \sum_{i \in N(j_k)} f_i y_i^*.$$

Summieren wir diese Ungleichung für alle bei der Ausführung des Algorithmus geöffneten Standorte i_k erhalten wir:

$$\sum_k f_{i_k} \leq \sum_k \sum_{i \in N(j_k)} f_i y_i^* \leq \sum_{i \in \mathcal{F}} f_i y_i^* \leq OPT,$$

wobei die vorletzte Ungleichung aus der Tatsache folgt, dass alle $N(j_k)$ einen leeren Schnitt haben, da Kunden aus $N^2(j_k)$ auch direkt mit i_k verbunden werden.

Es bleibt nun noch die Anbindungskosten der Kunden in den Mengen $N^2(j_k)$ zu beschränken. Setze der Einfachheit halber $j = j_k$ und $i = i_k$ für eine Iteration k. Mit Lemma 5.19 gilt wie oben $c_{ij} \leq v_j^*$. Betrachte nun die Anbindungskosten eines Kunden $l \in N^2(j)$ zu Standort i, wobei l ein Nachbar von Standort h ist, welcher wiederum benachbart zu

Kunde j ist (siehe Abbildung 5.6). Mithilfe der Dreiecksungleichung folgt:

$$c_{il} \leq c_{ij} + c_{hj} + c_{hl} \leq v_j^* + v_j^* + v_l^*,$$

wobei wir $c_{hj} \leq v_j^*$ benutzen, da $h \in N(j)$ gilt. Desweiteren erhalten wir

$$v_j^* \leq v_l^*,$$

da Kunde j ja die kleinste Dualvariable v_j^* unter allen noch unverbundenen Kunden hatte und l zu Zeitpunkt k noch unverbunden war. Somit folgt

$$c_{il} \leq 3v_l^*.$$

Insgesamt erhalten wir

$$\sum_k \{f_{i_k} + c_{i_k,j_k} + \sum_{l \in N^2(j_k)} c_{i_k,l}\} \leq \mathrm{OPT} + 3\sum_{j \in \mathcal{C}} v_j^* \leq 4\ \mathrm{OPT},$$

wobei die letzte Ungleichung aufgrund der schwachen Dualität gilt. □