Kombinatorische Optimierung 1

Pflichtfach
Technische Mathematik
5. Semester

Erstellt von Sara Kropf nach der Vorlesung von Johannes Hatzl im Wintersemester 2009

Institut für Optimierung und Diskrete Mathematik
Technische Universität Graz

Inhaltsverzeichnis

Kapitel 0

Einführung und Motivation

In der kombinatorischen Optimierung beschäftigt man sich mit Optimierungsproblemen bei denen die Menge der zulässigen Lösungen endlich ist. Viele Probleme lassen sich in folgender Form schreiben:

Gegeben:	$E = \{e_1, \ldots, e_2\}$, $\mathcal{F} \subseteq 2^E = \mathcal{P}(E)$ (zulässige Lösungen), $c : E \to \mathbb{R}$ (Kostenfunktion)
Gesucht:	$F \in \mathcal{F}$ mit $c(F) := \sum_{e \in F} c(e) \to \min / \max$, oder $c'(F) = \max_{e \in F} c(e) \to \min$

KÜRZESTE-WEGE-PROBLEM

Gegeben:	(Un-) gerichteter Graph $G = (V, E)$, $c : E \to \mathbb{R}$ (Kantenlänge), s,$t \in V$, $\mathcal{F}$ ist die Menge der einfachen Wege von s nach t
Gesucht:	kürzester Weg P von s nach t, d.h. $\sum_{e \in P} c(e)$ ist minimal

Bemerkung 0.0.1. Wenn negative Kantenlängen erlaubt sind, dann ist das Problem $\mathcal{NP}$-schwer.

LINEARE ZUORDNUNGSPROBLEM

Gegeben:	$G = (A \cup B, E)$ vollständig bipartiter Graph mit $\lvert A\rvert = \lvert B\rvert$, $c : E \to \mathbb{R}_+$, $\mathcal{F}$ ist die Menge der perfekten Matchings
Gesucht:	perfektes Matching $M \subseteq E$ mit $c(M)$ minimal

RUCKSACKPROBLEM

Gegeben:	n Gegenstände, jeder Gegenstand hat einen Wert c_i und ein Gewicht w_i, $i = 1 \ldots, n$, $B \in \mathbb{N}$ ist die Kapazität vom Rucksack
Gesucht:	$F \subseteq \{1, \ldots, n\}$ mit $w(F) = \sum_{e \in F} w(e) \leq B$ und $c(F)$ maximal

Das Rucksackproblem kann als ganzzahliges lineares Program geschrieben werden:

$$x_j = \begin{cases} 0 & \text{falls } j \notin F \\ 1 & \text{sonst} \end{cases}$$

$$\max \sum_{i=1}^{n} c_i x_i$$

$$\text{s.t. } \sum_{i=1}^{n} w_i x_i \leq B$$

$$x_i \in \{0, 1\}$$

Abbildung 1: links: zwei Subtouren, rechts: eine Tour

TSP - Traveling-Salesman-Problem

Gegeben: $K_n = (V, E)$ vollständiger Graph mit $|V| = n$
$D = (d_{ij})_{i=1\ldots n, j=1\ldots n}$ Distanzmatrix
Gesucht: Eine Tour, die jeden Knoten genau einmal besucht und am Ende wieder zum Ausgangsknoten zurückkehrt

IP-Formulierung für das TSP:

$$x_{ij} = \begin{cases} 1 & \text{falls Knoten } j \text{ direkt nach Knoten } i \text{ besucht wird} \\ 0 & \text{sonst} \end{cases}$$

$$\min \sum_{j=1}^{n} \sum_{i=1}^{n} d_{ij} x_{ij}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1 \quad \forall j = 1, \ldots, n$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i = 1, \ldots, n$$

Subtourungleichungen:

$$\sum_{i,j \in U} x_{ij} \leq |U| - 1 \quad \forall \varnothing \neq U \subsetneq V$$

$$x_{ij} \in \{0, 1\}$$

Chinese-Postman-Problem

Gegeben: $G = (V, E)$, $l : E \to \mathbb{R}_+$ Kantenlänge
Gesucht: Eine Wanderung, die jede Kante mindestens einmal besucht (Kanten, die zweimal besucht werden, zählen doppelt) und am Ende wieder zum Postamt zurückkehrt.

Maximales-Clique-Problem

Gegeben: $G = (V, E)$
Gesucht: $C \subseteq V$ mit C einer Clique, d.h. $G(C)$ ist ein vollständiger Graph mit $|C|$ maximal

Dieses Problem ist äquivalent zu folgendem Problem:

Gegeben: $G = (V, E)$
Gesucht: $U \subseteq V$ mit U einer unabhängigen Menge mit $|U|$ maximal.

Definition 0.0.2. Eine Menge von Knoten eines Graphen G heißt unabhängig (oder auch stabil), wenn keine zwei Knoten der Menge verbunden sind.

Definition 0.0.3. Eine Problemklasse P lässt sich linear auf eine Problemklasse Q reduzieren, falls es zwei Funktionen f und g gibt mit

1. f und g können in linearer Zeit berechnet werden.
2. f bildet Instanzen aus P auf Instanzen aus Q ab.
3. g bildet eine Lösung s von der Instanz $f(x)$ auf eine Lösung der Instanz x ab.
4. s ist optimal in $f(x) \Leftrightarrow g(s)$ ist optimal in x

$$\begin{array}{ccc} P & & Q \\ x & \xrightarrow{f} & f(x) \\ & & \Big\downarrow \text{OL} \\ g(s) & \xleftarrow{g} & s \end{array}$$

P und Q heißen äquivalent, wenn sie sich gegenseitig in linearer Zeit reduzieren lassen.

Beispiel 0.0.4. $f(G) = G^c$, $g(V) = V$

Kapitel 1

Spannbaumprobleme und Arboreszenzen

1.1 Problemstellung und Optimalitätskriterien für MST

Definition 1.1.1. Sei $G = (V, E)$ ein ungerichteter Graph und $T = (V, E')$ ein Baum mit $E' \subseteq E$, dann heißt T Spannbaum von G.

Minimales-Spannbaum-Problem (MST)	
Gegeben:	$G = (V, E)$ ein zusammenhängender Graph, $c : E \to \mathbb{R}$
Gesucht:	Spannbaum $T = (V, E')$ von $G = (V, E)$ mit minimalem Gewicht $c(T) := \sum_{e \in E'} c(e)$

Satz 1.1 (Cayley, 1889)**.** *Der vollständige Graph K_n hat n^{n-2} verschiedene Spannbäume.*

Wir können diesen Satz mit dem Matrix-Baum-Satz 1.2 beweisen und bezeichnen die Anzahl der Spannbäume eines Graphen G mit $\tau(G)$. Außerdem bezeichnen wir für eine $n \times n$ Matrix K die Teilmatrix, die man erhält, wenn man die u-te Zeile und v-te Spalte streicht, mit K_{uv}.

Satz 1.2 (Matrix-Baum-Satz ohne Beweis)**.** *Sei $G = (V, E)$ ein Graph mit $|V| = n$, A die Adjazenzmatrix von G, $D = diag(d(1), d(2), \ldots, d(n))$ und $K = D - A$ die sogenannte Kirchhoff-Matrix (oder Laplace-Matrix) von G. Dann gilt*

$$\tau(G) = (-1)^{u+v} \det(K_{uv}) \quad \forall u, v = 1, \ldots, n$$

Beweis. (des Satzes von Cayley 1.1 mit dem Matrix-Baum-Satz)
Für den vollständigen Graphen gilt:

$$K = D - A = \begin{pmatrix} n-1 & -1 & \cdots & -1 \\ -1 & n-1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ -1 & \ldots & -1 & n-1 \end{pmatrix}_{n \times n}$$

Nun setzen wir $u = v = 1$. Wenden wir den Matrix-Baum-Satz an, erhalten wir:

$$\tau(K_n) = \det(K_{11}) = \det \begin{pmatrix} n-1 & -1 & \cdots & -1 \\ -1 & n-1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ -1 & \ldots & -1 & n-1 \end{pmatrix}_{(n-1) \times (n-1)}.$$

Um die gesuchte Determinante zu berechnen, addieren wir die Zeilen 2 bis $n-1$ zur ersten Zeile und danach die erste Zeile zu den restlichen Zeilen, d.h.

$$\det(K_{11}) = \det \begin{pmatrix} 1 & 1 & \cdots & \cdots & 1 \\ -1 & n-1 & -1 & \cdots & -1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -1 \\ -1 & \ldots & \cdots & -1 & n-1 \end{pmatrix}_{(n-1)\times(n-1)}$$

$$= \det \begin{pmatrix} 1 & 1 & \cdots & \cdots & 1 \\ 0 & n & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \ldots & \cdots & 0 & n \end{pmatrix}_{(n-1)\times(n-1)}$$

Da die Determinante einer oberen Dreiecksmatrix das Produkt der Diagonalelemete ist, erhält man $\tau(K_n) = n^{n-2}$. □

Wir werden den Satz von Cayley noch anders beweisen. Dieser Beweis wurde erst kürzlich gefunden und ist im Moment wohl der einfachste Beweis.

Beweis. (Pitman, 1999) Der Beweis beruht auf dem Prinzip des doppelten Abzählens. Wir zählen labeled rooted trees (LRT, gelabelte Wurzelbäume), d.h. die Kanten eines Wurzelbaums bekommen paarweise verschiedene Labels zwischen 1 und $n-1$ und sind per Definition von den Blättern zur Wurzel gerichtet. Wir zählen diese Objekte auf zwei verschiedene Arten.

1. Es gibt $n(n-1)!$ Möglichkeiten einen Spannbaum in einen LRT umzuwandeln (n mögliche Wurzeln, $(n-1)!$ unterschiedliche Labels) $\Rightarrow \#(\text{LRT}) = n(n-1)!\tau(K_n)$.

2. Nun berechnen wir auf wie viele Arten man durch schrittweises Hinzufügen gerichteter Kanten aus einem leeren Graphen einen gelabelten Wurzelbaum erzeugen können. Für die erste gerichtete Kante gibt es $n(n-1)$ Möglichkeiten.
   Nach k gezeichneten Kanten ergibt sich folgende Situation: Es gibt $n-k$ Zusammenhangskomponenten und jede dieser Zusammenhangskomponenten ist ein LRT (siehe Abbildung 1.1). Die Anzahl der Knoten in der j-ten Zusammenhangskomponente sei n_j ($j = 1, \ldots, n-k$). Falls die $k+1$-te neue gerichtete Kante in der ersten Zusammenhangskomponente startet, muss sie in der Wurzel starten und kann als Endknoten jeden beliebigen Knoten außerhalb dieser Zusammenhangskomponente haben (da der Ausgangsgrad ≤ 1 sein muss). Das sind $n - n_1$ Möglichkeiten. Wir können aber auch in jeder anderen Zusammenhangskomponente starten und erhalten insgesamt

   $$\begin{aligned}(n-n_1) + (n-n_2) + \ldots + (n-n_{n-k}) &= (n-k)n - (n_1 + n_2 + \ldots + n_{n-k}) \\ &= (n-k)n - n = (n-k-1)n\end{aligned}$$

   Möglichkeiten in der $(k+1)$-ten Iteration. Insgesamt ergeben sich also

   $$\prod_{k=0}^{n-2} n(n-k-1) = n^{n-1}(n-1)!$$

   gelabelte Wurzelbäume.

Vergleichen wir nun die beiden Ausdrücke ergibt sich

$$n(n-1)!\,\tau(K_n) = n^{n-1}(n-1)!$$

woraus sofort $\tau(K_n) = n^{n-2}$ folgt. □

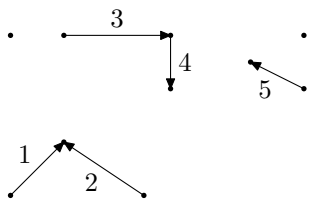

Abbildung 1.1: Konstruktion eines LRT nach 5 eingefügten Kanten, es gibt $10 - 5 = 5$ Zusammenhangskomponenten

Bemerkung 1.1.2. Es gibt im K_n mehr Spannbäume als Hamiltonsche Kreise. Trotzdem ist MST in $\mathcal{P}$ und das TSP $\mathcal{NP}$-schwer.

Bemerkung 1.1.3. Der Satz von Cayley zeigt, dass es zu aufwändig ist, alle spannenden Bäume zu enumerieren. Nimmt man an, dass 10^6 Bäume pro Sekunde enumeriert werden können, erfordert dies bei $n = 30$ genau $30^{28}/10^6$ Sekunden was ca. $7,25 \cdot 10^{27}$ Jahren entspricht.

Das folgende Problem ist eng mit dem minimalen Spannbaumproblem verwandt:

Maximales-Wald-Problem (MWF)	
Gegeben:	$G = (V, E)$ ein zusammenhängender Graph, $c : E \to \mathbb{R}$
Gesucht:	kreisfreier Graph (Wald) $W = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$ mit maximalem Gewicht $c(W) := \sum_{e \in E'} c(e)$ hat

Bemerkung 1.1.4. Dieses Problem ist äquivalent zum MST. Hat man eine Instanz vom MST-Problem, kann man neue Kantengewichte $c'(e) := K - c(e)$ für alle $e \in E$ mit $K = 1 + \max_{e \in E} c(e)$ definieren. Da alle spannenden Bäume von G gleiche Kardinalität haben und $c'(e) > 0$ für alle $e \in E$ gilt, ist ein maximaler Wald bzgl. c' auch ein minimaler Spannbaum bzgl. c. Umgekehrt kann man in einer Instanz vom MWF-Problem alle negativ gewichteten Kanten entfernen und $c'(e) := -c(e)$ für die verbleibenden Kanten setzen. Nun fügt man eine minimale den Graphen zusammenhängend machende Kantenmenge F mit beliebigen Kantengewichten hinzu. Löst man für diesen Graphen das MST-Problem und entfernt aus dem optimalen Spannbaum alle Kanten aus F, dann resultiert daraus ein Wald mit maximalem Gewicht bzgl. der ursprünglichen Gewichtsfunktion.

Wir geben nun ein Optimalitätskriterium für das MST-Problem an, mit dem wir später einen Algorithmus entwickeln werden.

Satz 1.3 (Optimalitätskriterium für Spannbäume)**.** *Sei $G = (V, E)$ ein zusammenhängender Graph, $c : E \to \mathbb{R}$ eine Gewichtsfunktion auf den Kanten und $T = (V, E')$ ein Spannbaum von G. Dann sind folgende Aussagen äquivalent:*

1. *T ist ein minimaler Spannbaum.*
2. *Für jede Kante $e = (x, y) \in E \setminus E'$ gilt: Keine Kante des x-y-Weges in T hat ein höheres Gewicht als e.*
3. *Für jede Kante $e \in E'$ gilt: Ist C eine der beiden Zusammenhangskomponenten von $T - e$, so ist die Kante e eine Kante mit minimalem Gewicht im Schnitt $\delta(V(C))$.*

Beweis. $1 \Rightarrow 2$**:** Angenommen 2 gilt nicht, dann $\exists e = (x, y) \in E \setminus E'$, $\exists f \in E'$ und f liegt in T am Weg von x nach y mit $c(f) > c(e) \Rightarrow T^* := (T - f) + e$ ist wieder ein Spannbaum mit $c(T^*) < c(T)$. Widerspruch.

$2 \Rightarrow 3$**:** Angenommen 3 gilt nicht, dann gibt es eine Kante $e' \in E'$, eine Zusammenhangskomponente C von $T - e'$ und eine Kante $\exists f' = (u, v) \in \delta(V(C))$ mit $c(f') < c(e')$. Es gilt $f' \notin E'$. Dann werden u und v im Baum T eindeutig miteinander verbunden. Dieser eindeutige Weg muss e' enthalten. Widerspruch zu 2.

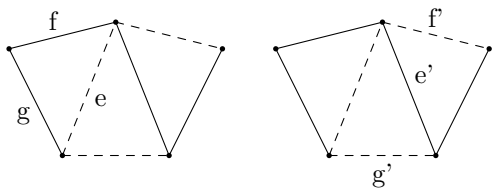

Abbildung 1.2: Optimalitätskrierium für minimale Spannbäume. links: Im Kreis muss $c(e) \geq \max(c(g), c(f))$ sein. rechts: Im Schnitt muss $c(e') \leq \min(c(g'), c(f'))$ sein.

$3 \Rightarrow 1$**:** Angenommen T erfüllt die Bedingung 3, T^* sei jene Optimallösung des MST-Problems, die mit T eine maximale Anzahl an Kanten gemeinsam hat. Zu zeigen ist: $T = T^*$.

Angenommen $T \neq T^* \Rightarrow \exists e = (x, y) \in E \setminus E^*$. Sei C eine Zusamenhangskomponente von $T - e$ und D der Kreis in $T^* + e$, dann $\exists f \neq e \in D \cap \delta(C) \Rightarrow T^* + e - f$ ist wieder ein Spannbaum. T^* ist optimal$\Rightarrow c(e) \geq c(f)$, und T erfüllt die Bedingung $3 \Rightarrow c(f) \geq c(e)$. D.h. $c(e) = c(f) \Rightarrow T^* + e - f$ ist optimal und hat eine Kante mehr mit T gemeinsam. Widerspruch.

□

1.2 Ein allgemeiner Algorithmus für das MST-Problem

Idee: Betrachte die Kanten einzeln und entscheide, ob sie in den Baum aufgenommen werden oder nicht.

- Grün färben heißt, die Kante kommt hinzu
- Rot färben heißt, die Kante kommt nicht hinzu

Algorithmus 1.1 MST-Problem

while Es gibt eine ungefärbte Kante in G **do**
Wende eine der beiden folgenden Färbungsregeln an:
Grüne Regel: anwendbar, wenn ein Schnitt $\delta(X)$ für $\emptyset \neq X \subset V$ von G existiert, für den gilt:

- $\delta(X)$ enthält keine grüne Kante und
- $\delta(X)$ enthält mindestens eine ungefärbte Kante

Aktion: Wähle die billigste ungefärbte Kante im Schnitt und färbe sie grün.
Rote Regel: anwendbar, wenn ein Kreis C in G existiert, für den gilt:

- C enthält keine rote Kante und
- C enthält mindestens eine ungefärbte Kante

Aktion: Wähle die teuerste ungefärbte Kante im Schnitt und färbe sie rot.
end while

Beispiel 1.2.1.
1. Rote Regel: $C = \{v_1, v_3, v_4, v_5\} \rightarrow$ Färbe (v_1, v_3) rot
2. Grüne Regel: $\delta(X) = \{(v_1, v_3), (v_1, v_4), (v_1, v_5), (v_2, v_3)\} \rightarrow$ Färbe (v_1, v_4) grün
3. Grüne Regel: $\delta(X) = \{(v_2, v_3), (v_1, v_3), (v_3, v_4)\} \rightarrow$ Färbe (v_2, v_3) grün
4. Grüne Regel: $\delta(X) = \{(v_1, v_5), (v_4, v_5)\} \rightarrow$ Färbe (v_1, v_5) grün

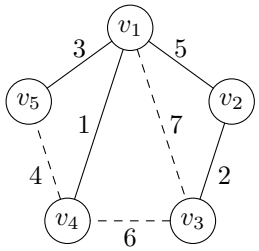

Abbildung 1.3: Endergebnis des Algorithmus 1.1

5. Rote Regel: $C = \{v_1, v_4, v_5\} \rightarrow$ Färbe (v_4, v_5) rot
6. Grüne Regel: $\delta(X) = \{(v_1, v_2), (v_1, v_3), (v_3, v_4)\} \rightarrow$ Färbe (v_1, v_2) grün
7. Rote Regel: $C = \{v_1, v_2, v_3, v_4\} \rightarrow$ Färbe (v_3, v_4) rot

Lemma 1.2.2. (Färbungslemma von Minty) *Sei $G = (V, E)$ ein gerichteter Graph, dessen Kanten grün, rot oder blau gefärbt sind. Sei $e \in E$ eine blaue Kante, dann gilt genau eine der beiden Aussagen:*

1. *Es gibt einen ungerichteten Kreis, der e enthält, mit nur grünen und blauen Kanten, wobei alle blauen Kanten im Kreis die selbe Orientierung haben.*
2. *Es gibt einen ungerichteten Schnitt, der e enthält, mit nur roten und blauen Kanten, wobei alle blauen Kanten im Schnitt gleich gerichtet sind.*

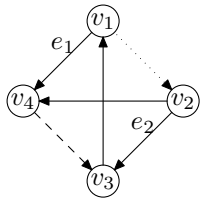

Abbildung 1.4: Graph zum Beispiel 1.2.3

Beispiel 1.2.3. In der Abbildung 1.4 sind die beiden Fälle aus dem Lemma 1.2.2 veranschaulicht (durchgezogen ist die Farbe Blau, strichliert die Farbe Rot und punktiert die Farbe Grün):

- Fall 1: $e_2 \in E$: $C = \{v_2, v_3, v_1\}$
- Fall 2: e_1: $\delta(X) = \{(v_1, v_4), (v_2, v_4), (v_4, v_3)\}$

Beweis. **1 oder 2 treffen zu:** Sei $e = (x, y)$ eine blaue Kante. Wir markieren die Knoten von G nach folgender Regel:

- Markiere y
- Sei v bereits markiert und w unmarkiert. Falls $v \rightarrow w$ eine blaue Kante ist oder $v \rightarrow w$ bzw. $v \leftarrow w$ eine grüne Kante ist, dann markieren wir w und setzen $\text{pred}(w) = v$.

Am Ende dieses Markierungsprozesses gibt es zwei Möglichkeiten:

1. x ist markiert:
   $y, x, \text{pred}(x), \text{pred}(\text{pred}(x)), \ldots$ ergibt einen Kreis, der die erste Aussagen des Lemmas erfüllt.

2. x ist nicht markiert:
   Sei $R = \{v \in V : v \text{ ist markiert}\}$, dann gilt $x \notin R$ und $y \in R$. Betrachte $F := \delta^+(R) \cup \delta^-(R)$. F enthält keine grünen Kanten (Markierungsprozess, jede Kante im Schnitt verbindet einen markierten mit einem nicht markierten Knoten). Außerdem gehen alle blauen Kanten von F in den Schnitt hinein. $\Rightarrow$ F ist ein Schnitt der die Bedingung 2 erfüllt.

1 und 2 treffen nicht gemeinsam zu: Sei $e = (x, y)$ eine blaue Kante, die in einem Schnitt gemäß 2 und einem Kreis gemäß 1 liegt. Der Kreis hat eine Kante $f \neq e$ die ebenfalls im Schnitt liegt. $\Rightarrow$ f ist grün oder blau, da f im Kreis liegt, und f ist rot oder blau, da f im Schnitt liegt $\Rightarrow$ f ist blau. f ist gleichgerichtet wie e im Kreis und f ist nicht gleichgerichtet wie e im Schnitt. Widerspruch.

□

Färbungsinvariante des Algorithmus 1.1: Es gibt zu jedem Zeitpunkt des Algorithmus einen optimalen Spannbaum, der alle grünen und keine roten Kanten enthält.

Lemma 1.2.4. *Für den Algorithmus 1.1 gelten die folgenden beiden Aussagen:*

1. *Solange es ungefärbte Kanten gibt ist entweder die rote oder die grüne Regel anwendbar.*
2. *Jede Anwendung der roten und grünen Regel erhält die Färbungsinvariante.*

Beweis. 1. folgt direkt aus dem Lemma von Minty 1.2.2, wobei ungefärbte Kanten blau sind: Orientiere die Kanten beliebig, entweder es gibt einen Kreis durch eine ungefärbte Kante e, in der alle Kanten ungefärbt oder grün sind, oder es gibt einen Schnitt durch e, in dem alle Kanten ungefärbt oder rot sind.

2. Induktion nach der Anzahl der gefärbten Kanten:
   Induktionsanfang: keine Kante ist gefärbt, Aussage gilt trivialerweise
   Induktionsschluss: Invariante bleibt gültig nach Anwendung der grünen Regel:
   Sei T^* ein optimaler Spannbaum laut Invariante vor der Anwendung der grünen Regel. Sei e die jetzt grün gefärbte Kante. Falls $e \in T^*$, sind wir fertig. Sei also $e \notin T^*$, dann erzeugt e einen Kreis C bzgl. T^*.
   grüne Regel: Es gibt einen Schnitt $\delta(X)$ ohne grüne Kanten und e ist die billigste Kante im Schnitt.
   Sei $f \in \delta(X) \cap C$ und $f \neq e$, dann gilt f ist im Baum und im Schnitt, also ist f ungefärbt. Wegen der grünen Regel gilt $c(e) \leq c(f)$. Das Optimalitätskriterium 2 in Satz 1.3 für T^* garantiert $c(f) \leq c(e)$. Daher gilt $c(e) = c(f)$ und $(T^* - f) + e$ ist Optimallösung, die die Invariante erhält.
   Induktionsschluss: Invariante bleibt gültig nach Anwendung der roten Regel:
   Sei T^* ein optimaler Spannbaum laut Invariante vor der Anwendung der roten Regel. Sei e die jetzt rot gefärbte Kante. Falls $e \notin T^*$, sind wir fertig. Sei also $e \in T^*$, dann erzeugt e einen Schnitt $\delta(X)$ bzgl. T^*.
   rote Regel: Es gibt einen Kreis C ohne rote Kanten und e ist die teuerste Kante im Kreis. Sei $f \in \delta(X) \cap C$ und $f \neq e$, dann ist f nicht im Baum aber im Kreis, also ungefärbt. Wegen der roten Regel gilt $c(e) \geq c(f)$. Das Optimalitätskriterium 3 in Satz 1.3 für T^* garantiert $c(e) \leq c(f)$ und daher $c(e) = c(f)$. Damit ist $(T^* - e) + f$ wieder ein optimale Spannbaum, der die Invariante erfüllt.

□

Insgesamt haben wir damit folgenden Satz gezeigt:

Satz 1.4. *Algorithmus 1.1 liefert einen optimalen Spannbaum.*

1.3 Die Spezialisierungen von Kruskal und Prim

Wir betrachten nun zwei Spezialfälle des Algorithmus 1.1. Der erste Algorithmus 1.2 geht auf Kruskal (1956) zurück.

Algorithmus 1.2 Algorithmus von Kruskal

Gegeben: Ungerichteter, zusammenhängender Graph $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{R}$

Gesucht: Ein minimaler Spannbaum T.

```
Sortiere die Kanten: c(e_1) ≤ c(e_2) ≤ ... ≤ c(e_m)
Setze T := (V(G), ∅)
for i = 1 to m do
  if T + e_i enthält keinen Kreis then
    Setze T := T + e_i
  end if
end for
```

Sei $T = (V, E)$ der momentane Wald (am Anfang gilt $E(T) = \emptyset$) im Algorithmus, dann tritt in jeder Iteration einer der beiden Fälle auf:

1. Kante e_i erzeugt Kreis: e_i ist die teuerste Kante auf diesem Kreis wegen der Sortierung (rote Regel).

2. Kante e_i erzeugt keinen Kreis: e_i verbindet zwei Zusammenhangskomponenten X und Y von T. Weiters ist e_i die einzige Kante in $\delta(X) \cap E(T + e_i)$ und wegen der Sortierung ist e_i die billigste Kante in $\delta(X)$ (grüne Regel).

Daher ist dies ein Spezialfall des Algorithmus im vorigen Abschnitt.

Beispiel 1.3.1. Wir wenden den Algorithmus von Kruskal auf den Graphen in Abbildung 1.5 an. Am Beginn sortieren wir die Kanten nach den Gewichten: $c(v_1, v_4) \leq c(v_2, v_3) \leq c(v_1, v_5) \leq c(v_4, v_5) \leq c(v_1, v_2) \leq c(v_3, v_4) \leq c(v_1, v_3)$
Dann werden der Reihe nach die Kanten $(v_1, v_4), (v_2, v_3), (v_1, v_5)$ zum Baum hinzugefügt, da sie keinen Kreis bilden. Die Kante (v_4, v_5) würde einen Kreis bilden und wird deshalb nicht hinzugefügt. Die Kante (v_1, v_2) bildet keinen Kreis, und wird wieder hinzugefügt.

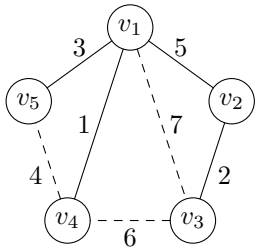

Abbildung 1.5: Endergebnis des Algorithmus von Kruskal 1.2

Laufzeit einfache Implementierung: $\mathcal{O}(m \log n + n^2)$

- $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$ zum Sortieren der Kanten

- Eine Union-Find-Datenstruktur verwaltet die Zusammenhangskomponenten während des Algorithmus. Man verwaltet also ein Array ZK von Knoten mit $ZK[v] = k$ falls v in Zusammenhangskompontente mit Nummer k liegt. Am Beginn wenn es noch keine Kanten gibt gilt $ZK = [1, 2, \ldots, n]$. Dadurch kann man in $\mathcal{O}(1)$ testen, ob eine neue Kante $e = (u, v)$

einen Kreis schließt. Dies ist nämlich genau dann der Fall, wenn $ZK[u] = ZK[v]$. Gilt $ZK[u] \neq ZK[v]$ kann die Kante eingefügt werden und alle Knoten w mit $ZK[w] = ZK[v]$ werden auf $ZK[w] := ZK[u]$ geändert. Dies kann in $\mathcal{O}(n)$ geschehen. Da insgesamt $n-1$ Kanten eingefügt werden ergibt das eine Laufzeit von $\mathcal{O}(n^2)$.

effiziente Implementierung: Verwendet man bessere Datenstrukturen für das Union-Find Problem, dann erhält man eine Laufzeit von $\mathcal{O}(m \log n)$. Falls die Sortierung der Kanten schon gegeben ist, kann man sogar $\mathcal{O}(m\alpha(m,n))$ erreichen, wobei $\alpha(m,n)$ die inverse Ackermannfunktion ist.

Der zweite Spezialfall von 1.1 ist der Algorithmus 1.2 von Prim (1957).

Algorithmus 1.3 Algorithmus von Prim

Gegeben: Ungerichteter, zusammenhängender Graph $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{R}$

Gesucht: Ein minimaler Spannbaum T.

Wähle einen beliebigen Knoten $v \in V(G)$ und setze $T := (\{v\}, \emptyset)$.
while $V(T) \neq V(G)$ **do**
Wähle die billigste Kante $e = (x, y) \in E(G)$ mit $x \in V(T)$, $y \notin V(T)$ und setze $T := T + e$.
end while

Hier wird immer nur die grüne Regel angewendet, da der Baum Schritt für Schritt erweitert wird.

Beispiel 1.3.2. *Prim's Algorithmus 1.3* Wir wenden nun den Algorithmus von Prim wieder auf den Graphen in Abbildung 1.5 an und beginnen mit dem Knoten v_2. Der billigste Nachbar ist der Knoten v_3, also ist (v_2, v_3) grün. Der billigste Nachbar von v_2 oder v_3 ist v_1, also ist (v_1, v_2) grün, usw...

Laufzeit: $O(n)$ Schleifendurchläufe, $O(n)$ für billigsten Nachbarn suchen. Daher hat der Algorithmus eine Laufzeit von $O(n^2)$. Man kann zeigen, dass dies für dichte Graphen (das sind Graphen mit $m = \Omega(n^2)$) bestmöglich ist.

1.4 Das Spannbaumpolytop

Satz 1.5 (Edmonds, 1970)**.** *Sei $G = (V, E)$ ein zusammenhängender ungerichteter Graph mit $|V| = n$. Dann ist das Polytop*

$$P := \left\{ x \in [0,1]^{E(G)} : \sum_{e \in E(G)} x(e) = n-1, \sum_{e \in E(G[X])} x(e) \leq |X| - 1 \quad \forall \emptyset \neq X \subset V \right\}$$

ganzzahlig, d.h. alle Ecken von P sind ganzzahlig. Die Ecken von P sind die Inzidenzvektoren der Spannbäume von G. (P heißt das Spannbaumpolytop von G.)

Beweis.

- Sei T ein Spannbaum von G und x der Inzidenzvektor von T. Dann ist $x \in P$ wegen der Charakterisierung von Bäumen (ein Spannbaum hat genau $n-1$ Kanten und ist kreisfrei). Außerdem ist x ist Ecke von P da $x(e) \in \{0, 1\}$.

- Sei $x \in P$ eine ganzzahlige Ecke. Dann ist x ein Inzidenzvektor einer Kantenmenge eines kreisfreien Teilgraphens mit $n-1$ Kanten, daher ist x wiederum Inzidenzvektor eines Spannbaumes. Zu zeigen bleibt, dass es keine nicht ganzzahligen Ecken von P gibt.
  Sei $c : E(G) \to \mathbb{R}$ eine beliebige Kostenfunktion und sei T ein Spannbaum, der mit Hilfe des Algorithmus von Kruskal 1.2 berechnet wurde. Sei $E(T) = \{f_1, \ldots, f_{n-1}\}$ wobei gilt $c(f_1) \leq \ldots \leq c(f_{n-1})$ und weiters sei $X_k \subseteq V(G)$ die Zusammenhangskomponente von $(V(G), \{f_1, \ldots f_k\})$, die f_k enthält $(k = 1, \ldots, n-1)$.

Sei x^* der Inzidenzvektor von T. Zu zeigen ist, dass x^* ist Optimallösung von

$$\begin{array}{llll} \text{(LP)} & \min & \sum_{e\in E(G)} c(e)x(e) & \\ & \text{st} & -\sum_{e\in E(G)} x(e) & = 1-n \\ & & -\sum_{e\in E(G[X])} x(e) & \geq 1-|X| \quad \text{für } \forall\, \emptyset \neq X \subset V \\ & & x(e) & \geq 0 \end{array}$$

ist. Wir definieren uns für jede Menge $\emptyset \neq X \subseteq V$ eine Dualvariable z_X und fordern $z_X \geq 0 \Leftrightarrow X \neq V(G)$:

$$\begin{array}{llll} \text{(DP)} & \max & \sum_{\{X:\emptyset\neq X\subseteq V(G)\}} (1-|X|)z_X & \\ & \text{st} & -\sum_{\{X:e\subseteq X\subseteq V(G)\}} z_X & \leq c(e) \quad \forall\, e \in E(G) \\ & & z_X & \geq 0 \quad \forall\, \emptyset \neq X \subset V(G) \\ & & z_{V(G)} & \text{frei} \end{array}$$

Wir setzen $z^*_{X_k} := c(f_l) - c(f_k)$ für $k = 1, 2, \ldots, n-2$, wobei l der erste Index, größer als k ist, für den $f_l \cap X_k \neq \emptyset$ gilt, $z^*_{V(G)} := -c(f_{n-1})$ und $z^*_X = 0$ für alle $X \notin \{X_1, X_2, \ldots, X_{n-1}\}$.

Wir zeigen, dass die oben definierte Lösung für das duale Problem tatsächlich zulässig ist. Sei $e = (v, w) \in E(G)$, dann gilt:

$$\sum_{\{X:e\subseteq X\subseteq V(G)\}} z_X = c(f_i)$$

wobei i der kleinste Index ist, für den $v, w \subseteq X_i$ gilt. Es gilt $c(f_i) \leq c(e)$ da v, w in verschiedenen Zusammenhangskomponenten von $(V, \{f_1, \ldots, f_{l-1}\})$ liegen. Da auch $z^*_{X_k} = c(f_l) - c(f_k) \geq 0$ mit $l > k$ gilt, können wir schließen, dass z^* tatsächlich eine zulässige Lösung für das duale Problem ist.
Wir haben eine zulässige Lösung x^* vom primalen Problem und eine zulässige Lösung z^* vom dualen Problem. Wir zeigen, dass x^* und z^* Optimallösungen sind, indem wir den Satz vom komplementären Schlupf anwenden.

Sei $x^*(e) > 0$ und damit $e \in E(T)$. Dann gilt

$$-\sum_{\{X:\emptyset\neq X\subseteq V(G)\}} z_X = c(e).$$

Also wird die entsprechende Nebenbedingung im dualen Problem mit Gleichheit erfüllt. Ist schließlich $z^*_X > 0$ so ist $T[X]$ zusammenhängend und die entsprechende primale Nebenbedingung ist mit Gleichheit erfüllt. Daraus folgt mit dem Satz vom komplementären Schlupf, dass x^* und z^* Optimallösungen sind.

□

Bemerkung 1.4.1. Der Beweis des Satzes 1.5 gibt einen alternativen Beweis zur Korrektheit des Algorithmus von Kruskal 1.2.

1.5 Arboreszenzen mit minimalem Gewicht

Definition 1.5.1. Sei $G = (V, E)$ ein gerichteter Graph. Dann heißt G Branching, wenn folgende Bedingungen erfüllt sind:

1. der zugrundeliegende ungerichtete Graph ist kreisfrei
2. der Eingangsgrad jedes Knoten ist kleiner gleich als 1, dh. $\delta^-(v) \leq 1 \; \forall v \in V$

Ein zusammenhängendes Branching heißt Arboreszenz.

Bemerkung 1.5.2. In einer Arboreszenz gibt es genau einen Knoten $r \in V$ mit $\delta^-(r) = 0$. r heißt die Wurzel der Arboreszenz.

Definition 1.5.3. Sei $G = (V, E)$ ein gerichteter Graph. Dann heißt die Arboreszenz $A = (V, E')$ spannende Arboreszenz falls $E' \subseteq E$.

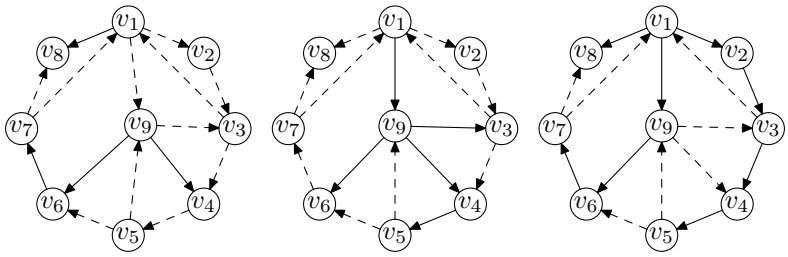

Abbildung 1.6: links ein Branching, in der Mitte eine Arboreszenz und rechts eine spannende Arboreszenz (mit Wurzel v_1)

In der Abbildung 1.6 sind ein Branching, eine Arboreszenz und eine spannende Arboreszenz dargestellt. Ähnlich wie im ungerichteten Fall gibt es auch hier eine Charakterisierung von Arboreszenzen:

Satz 1.6. *Sei $G = (V, E)$ ein Digraph mit n Knoten. Dann sind die folgenden acht Aussagen äquivalent:*

1. *G ist eine Arboreszenz mit Wurzel r.*
2. *G ist ein Branching mit $n - 1$ Kanten und $\delta^-(r) = \emptyset$.*
3. *G hat $n - 1$ Kanten und jeder Knoten ist von r aus erreichbar.*
4. *Jeder Knoten ist von r aus erreichbar, aber das Entfernen einer beliebigen Kante zerstört diese Eigenschaft.*
5. *G erfüllt $\delta^+(X) \neq \emptyset$ für alle $X \subset V(G)$ mit $r \in X$, die Entfernung einer beliebigen Kante von G zerstört jedoch diese Eigenschaft.*
6. *$\delta^-(r) = \emptyset$ und für jedes $v \in V(G) \setminus \{r\}$ gibt es einen eindeutig bestimmten r-v Weg.*
7. *$\delta^-(r) = \emptyset$ und $|\delta^-(v)| = 1$ für alle $v \in V(G) \setminus \{r\}$, und G ist kreisfrei.*
8. *$\delta^-(r) = \emptyset$ und $|\delta^-(v)| \leq 1$ für alle $v \in V(G) \setminus \{r\}$, und jeder Knoten $v \in V$ ist von r aus erreichbar.*

Bemerkung 1.5.4. Es gibt allerdings auch Unterschiede zum ungerichteten Fall:

1. Es gibt Graphen die keine spannende Arboreszenz besitzen (Abbildung 1.7).

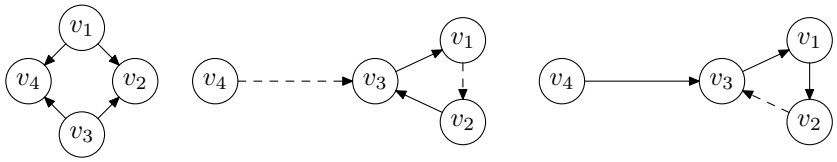

Abbildung 1.7: links ein Graph ohne spannende Arboreszenz, in der Mitte eine maximale Arboreszenz bzgl. Inklusion, rechts eine maximale Arboreszenz bzgl. Kardinalität

2. Im ungerichteten Fall hat jeder Spannbaum gleiche Kardinalität. Im gerichteten Fall gibt es Arboreszenzen, die maximal bzgl. Inklusion sind, aber nicht maximal bzgl. der Kardinalität (Abbildung 1.7).

In diesem Abschnitt wollen wir Algorithmen für folgende drei Problemstellungen angeben.

MINIMALES-GEWICHTETES-ARBORESZENZEN-PROBLEM (MINWA)

Gegeben: $G = (V, E)$ ein gerichteter Graph, $c : E \to \mathbb{R}$
Gesucht: $A = (V, E')$ spannende Arboreszenz mit minimalem Gewicht $c(A) := \sum_{e \in E'} c(e)$, oder entscheide, dass es keine solche gibt.

MAXIMALES-BRANCHING-PROBLEM (MAXB)

Gegeben: $G = (V, E)$ ein gerichteter Graph, $c : E \to \mathbb{R}$
Gesucht: $B = (V', E')$ mit $V' \subseteq V$, $E' \subseteq E$ Branching mit maximalem Gewicht $c(B) := \sum_{e \in E'} c(e)$

MINIMALES-GEWICHTETES-WURZEL-ARBORESZENZ-PROBLEM (MINGWA)

Gegeben: $G = (V, E)$ ein gerichteter Graph, $c : E \to \mathbb{R}$, $r \in V$
Gesucht: $A = (V, E')$ spannende Arboreszenz mit minimalem Gewicht und r als Wurzel von A, oder entschiede, dass es keine solche gibt.

Lemma 1.5.5. *Die obigen drei Probleme sind äquivalent.*

Beweis. Hat man eine Instanz des MinWA-Problems kann man neue Kantenkosten $c'(e) := K - c(e)$ definieren, wobei $K = \max\{|c(e)| : e \in E\} + 1$. Berechnet man nun ein Branching mit maximalem Gewicht, so ist dieses eine spannende Arboreszenz (da $c'(e) > 0$ für alle $e \in E$ gilt) mit minimalem Gewicht bzgl. c.
Für einen Graphen $G = (V, E)$ des MaxB-Problems kann man eine Wurzel r hinzufügen und den Graphen $G' = (V \cup \{r\}, E \cup \{(r, v) : v \in V\})$ mit der Gewichtsfunktion $c'(e) = c(e)$ für alle $e \in E$ und $c'(e) = 0$ für alle $e \in E' \setminus E$ betrachten. Nun sieht man sofort die Äquivalenz der beiden Probleme.
Hat man schließlich eine Instanz des MinGWA-Problems mit der Wurzel r gegeben, fügt man wiederum einen Knoten s und die Kante (s, r) mit Gewicht 0 hinzu. Damit erzwingt man, dass r, nach dem Löschen der Kante (s, r), die Wurzel ist. Damit sind die Probleme äquivalent. □

Da die Probleme äquivalent sind, genügt es einen Algorithmus für das Max-Branching-Problem zu finden. Wir können o.B.d.A. annehmen, dass $c(e) > 0$ $\forall e \in E$, da nicht positive Kanten nie in eine Optimallösung aufgenommen werden.

Eine erste Idee: Relaxiere das Problem und suche eine Lösung $B' \subseteq G$, die die Bedingung $|\delta^-(v)| \leq 1$ und $\sum_{e \in B'} c(e) \to \max$ erfüllt mit der Greedy-Methode, d.h. füge für jeden Knoten die teuerste eingehende Kante hinzu. Ist die Greedy-Lösung kreisfrei, haben wir eine Optimallösung für das ursprüngliche Problem gefunden und wir können aufhören. Ist die Greedy-Lösung nicht kreisfrei, dann müssen wir mindestens eine Kante aus jedem Kreis löschen, um zulässig zu werden. Das nächste Lemma zeigt, dass es ausreicht genau eine Kante aus jedem Kreis zu löschen.

Lemma 1.5.6. *Sei B_0 ein Teilgraph von G mit maximalem Gewicht und $|\delta^-(v)| \leq 1\ \forall v \in V(B_0)$. Dann gibt es ein optimales Branching B von G mit $|E(C) \setminus E(B)| = 1$ für jeden Kreis C in B_0.*

Beweis. Sei B ein optimales Branching mit so vielen Kanten von B_0 wie möglich. Sei C ein Kreis von B_0 mit $E(C) \setminus E(B) = \{(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)\}$ mit $k \geq 2$ und $a_1, b_1, a_2, b_2, \ldots, a_k, b_k$ liegen in dieser Reihenfolge am Kreis. Wir behaupten, dass B einen Weg von b_i nach b_{i-1} hat $\forall i = 1, \ldots k$ ($b_0 = b_k$). Das ist ein Widerspruch, da diese Wege einen Kreis in B bilden.

Betrachte B' mit $V(B') = V(G)$ und $E(B') = \{(x, y) \in E(B) : y \neq b_i\} \cup \{(a_i, b_i)\}$. Es gilt $c(B') \geq c(B)$ (Greedy-Algorithmus) und da B optimal ist folgt $c(B') = c(B)$. Wegen der Definition von B ist B' kein Branching, d.h. B' hat einen Kreis. Dann muss aber B' einen b_i-a_i Weg enthalten und damit auch B einen b_{i-1}-b_i Weg. □

Dieses Ergebnis motiviert den folgenden Algorithmus, der von Edmonds (1967) entwickelt wurde. In der Abbildung 1.8 ist der Ablauf des Algorithmus schematisch dargestellt.

Algorithmus 1.4 Edmond's Branching Algorithmus

Gegeben: Gerichteter Graph $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{R}_+$
Gesucht: Ein maximales Branching B in G

1: Setze $i := 0$, $G_0 = G$ und $c_0 := c$.
2: Konstruiere einen Untergraphen B_i von G_i mit maximalem Gewicht unter der Nebenbedingung, dass jeder Knoten Eingangsgrad ≤ 1 hat.
3: **if** B_i enthält keine Kreise **then**
4: Setze $B := B_i$ und **goto** 14
5: **else**
6: {*Schrumpfung*}
7: Konstruiere (G_{i+1}, c_{i+1}) aus (G_i, c_i) durch Schrumpfung aller Kreise in B_i.
8: Kontrahiere jeden Kreis C zu einem Knoten v_C.
9: **for** $e = (z, y) \in E(G_i)$ mit $z \notin V(C)$, $y \in V(C)$ **do**
10: Setze $c_{i+1}(e') := c_i(e) - c_i(\overline{e}) + c_i(e_C)$ und $\Phi(e') := e$, wobei $e' := (z, v_C)$, $\overline{e} = (x, y) \in E(C)$ und e_C ein Bogen in C mit minimalem Gewicht.
11: **end for**
12: i := i+1 und **goto** 2
13: **end if**
14: **while** $i \neq 0$ **do**
15: {*Expansion*}
16: **for** Kreise C in B_{i-1} **do**
17: **if** Es gibt eine Kante $e' = (z, v_C) \in E(B)$ **then**
18: Setze $E(B) := (E(B) \setminus e') \cup \Phi(e') \cup (E(C) \setminus \overline{\Phi(e')})$
19: **else**
20: Setze $E(B) := E(B) \cup (E(C) \setminus e_C)$
21: **end if**
22: **end for**
23: Setze $V(B) := V(G_{i-1})$, $i := i - 1$ und **goto** 14.
24: **end while**

Der Algorithmus konstruiert also eine Folge von Graphen $G = G_0, G_1, \ldots, G_k$ und Kantengewichte $c = c_0, c_1, \ldots, c_k$. Diese Folge wird fortgesetzt bis die Greedymethode im Graphen G_k für das relaxierte Problem eine kreisfreie Lösung und damit auch eine Optimallösung liefert. Wir müssen also nur noch zeigen, dass der Algorithmus ein optimales Branching B_i in G_i, in ein optimales Branching B_{i-1} in G_{i-1} verwandelt. Mit Induktion folgt dann die Korrektheit des Algorithmus. Man sieht leicht, dass bei der Expansion eines Kreises C für jedes Branching B_i in G_i

$$c_{i-1}(B_{i-1}) = c_i(B_i) + \underbrace{c_{i-1}(C) - c_{i-1}(e_C)}_{\text{Konstante}}$$

folgt. Mit Lemma 1.5.6 folgt dann, dass der Algorithmus 1.4 korrekt ist. Da der Algorithmus höchstens n Iterationen hat und jede Iteration in $\mathcal{O}(m)$ Zeit implementiert werden kann, hat der Algorithmus eine Laufzeit von $\mathcal{O}(mn)$.

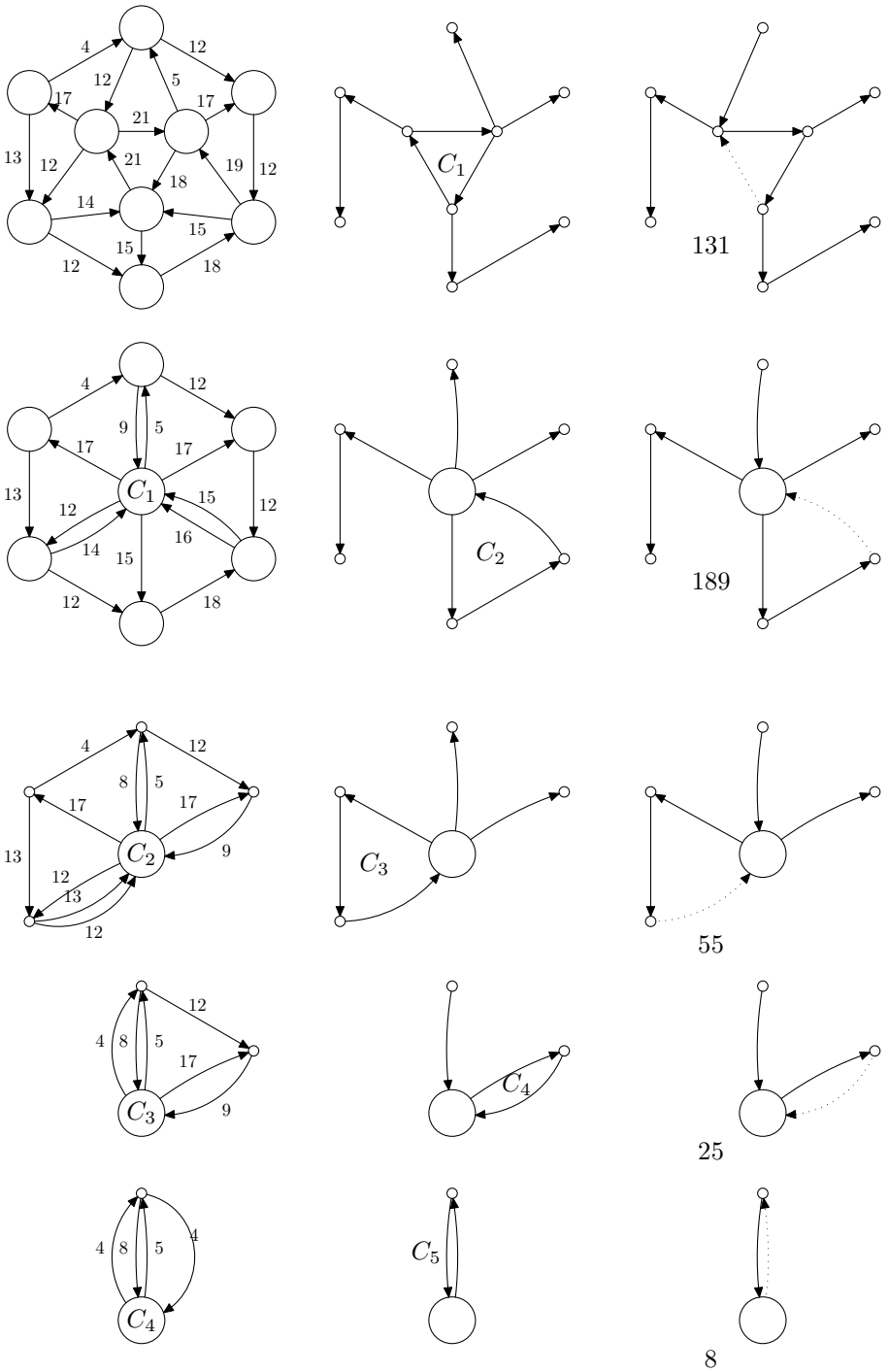

Abbildung 1.8: Beispiel für den Algorithmus 1.4: links der jeweilige Graph, in der Mitte das Ergebnis der Greedy-Methode und rechts das aktuelle Branching mit Zielfunktionswert

Kapitel 2

Kürzeste Wege

2.1 Grundlegende Definitionen und Problemstellung

Definition 2.1.1. Sei $G = (V, E)$ ein (eventuell gerichteter) Graph. Dann heißt eine Folge von Knoten $v_1, \ldots, v_k$ Wanderung, falls $(v_i, v_{i+1}) \in E$ für alle $i = 1, \ldots, k-1$. Sind die Knoten paarweise verschieden heißt die Folge Weg und wir schreiben $P = [v_1, \ldots, v_k]$.
Eine Folge von Knoten $v_0, \ldots, v_k$ heißt Kreis, falls $v_0 = v_k$ gilt, die Knoten $v_0, \ldots, v_k$ paarweise verschieden sind und $(v_i, v_{i+1}) \in E$ für $i = 0, \ldots, k-1$. Wir schreiben $C = [v_0, v_1, \ldots, v_k]$.

In diesem Kapitel möchten wir für einen gerichteten Graphen $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{R}$ einen kürzesten Weg P zwischen zwei Knoten s und t finden, d.h. einen Weg mit minimalem Gewicht

$$c(P) := \sum_{e \in E(P)} c(e)$$

oder entscheiden, dass t von s aus nicht erreichbar ist. Von nun an schreiben wir für $s, t \in V$

$$d^c(s,t) = \begin{cases} \text{Länge eines kürzesten Weges von } s \text{ nach } t \\ \infty, \text{ falls es keinen Weg von } s \text{ nach } t \text{ gibt} \end{cases}$$

Sollte keine Verwechslung möglich sein, schreiben wir auch $d(s,t)$ statt $d^c(s,t)$.

Kürzeste-Wege-Problem (SPP)	
Gegeben:	$G = (V, E)$ ein gerichteter Graph, $c : E \to \mathbb{R}$, $s \in V$
Gesucht:	kürzester Weg von s nach t für alle $t \in V$.

Bemerkung 2.1.2. Wenn wir keine Einschränkung an die Kantenkosten treffen, ist das Problem $\mathcal{NP}$-schwer. Setzt man in einem Graphen alle Kantenlängen auf -1, so ist der kürzeste Weg von s nach t ein Hamiltonscher Pfad (falls ein solcher existiert). Da das Hamiltonsche Pfadproblem auf gerichteten Graphen ein $\mathcal{NP}$-schweres Problem ist, ist auch das Finden eines kürzesten Weges ein $\mathcal{NP}$-schweres Problem. Schränken wir uns auf Kantengewichte ein für die es keine Kreise mit negativem Gewicht gibt, ist das Problem in polynomieller Zeit lösbar.

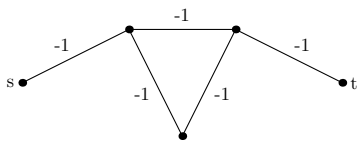

Abbildung 2.1: Zusammenhang Kürzeste-Wege-Problem und Hamiltonscher-Pfad-Problem

Definition 2.1.3. Die Funktion $c : E \to \mathbb{R}$ heißt konservativ, falls jeder Kreis $C = [v_0, \ldots, v_k]$ in G eine nicht negative Länge hat, d.h.

$$c(C) := \sum_{e \in E(C)} c(e) \geq 0$$

gilt.

Bemerkung 2.1.4. Für konservative Kantengewichte gilt, dass die Länge der kürzesten Wanderung von s nach t gleich der Länge des kürzesten Weges von s nach t ist.

Bemerkung 2.1.5. Möchte man kürzeste Wege in ungerichteten Graphen finden, kann man falls alle Kantengewichte nicht-negativ sind, jede ungerichtete Kante durch ein Paar entgegengesetzt gerichteter und gleichgewichteter Kanten ersetzen. Für negative Kanten funktioniert dieser Trick nicht mehr, da der modifizierte Graph dann negative Kreise hat. Für ungerichtete Graphen ist das Problem dennoch lösbar, wird aber in dieser Vorlesung nicht behandelt.

Bemerkung 2.1.6. Die Algorithmen für das Kürzeste-Wege-Problem sollen auch erkennen, ob es einen negativen Kreis in $G = (V, E)$ gibt und diesen gegebenfalls ausgeben. Andernfalls sollen die kürzesten Wege bestimmt werden.

Lemma 2.1.7. *Sei $G = (V, E)$ ein gerichteter Graph und $c : E \to \mathbb{R}$ konservativ. Ist $P = [s, \ldots, u, v]$ ein kürzester Weg von s nach v, dann ist $P - (u, v)$ ein kürzester Weg von s nach u.*

Beweis. Angenommen W ist ein kürzerer Weg von s nach u als $P - (u, v)$. Dann unterscheiden wir zwei Fälle:

1. W hat v nicht als Zwischenknoten, dann ist $W + (u, v)$ ein kürzerer s-v-Weg als P und wir haben einen Widerspruch.
2. $W = [s, \ldots, v, \ldots, u]$ hat v als Zwischenknoten, dann gilt laut Annahme $c(W) < c(P-(u, v))$. Daraus folgt
$$c(W + (u, v)) < c(P) \leq c(W').$$
wobei W' dem Teilweg von W zwischen s und v entspricht. Damit ist aber $W \setminus W' + (u, v)$ ein negativer Kreis und wir erhalten einen Widerspruch.

□

Lemma 2.1.8. *Sei $G = (V, E)$ ein gerichteter Graph und $c : E \to \mathbb{R}$ konservativ, dann gilt*

$$d(s, v) \leq d(s, u) + c(u, v) \quad \forall (u, v) \in E.$$

Beweis. Folgt direkt aus Lemma 2.1.7 □

Definition 2.1.9. Sei $\pi : V \to \mathbb{R}$, dann heißen

$$c^\pi(u, v) := c(u, v) + \pi(u) - \pi(v) \quad \forall (u, v) \in E$$

reduzierte Kosten. Falls $c^\pi(u, v) \geq 0$ für alle $(u, v) \in E$, dann heißt π (zulässiges) Knotenpotential.

Beispiel 2.1.10. Man betrachte für den Graphen in Abbildung 2.2 folgende Funktion: $\pi(s) = 0$, $\pi(1) = -3$, $\pi(2) = 2$, $\pi(3) = 4$, $\pi(4) = 7$. Dann gilt z.B. $c^\pi(2, 3) = 1 + 2 - 4 = -1$ und für den Pfad $P = [s, 1, 3, 4]$ ist $c(P) = 4$ und $c^\pi(P) = c(P) + \pi(s) - \pi(4) = 4 + 0 - 7 = -3$

Sei $P = [v_1, \ldots, v_k]$ ein Weg in G, dann ergibt eine einfach Rechnung

$$\begin{aligned} c^\pi(P) := \sum_{i=1}^{k-1} c^\pi(v_i, v_{i+1}) &= \sum_{i=1}^{k-1} (c(v_i, v_{i+1}) + \pi(v_i) - \pi(v_{i+1})) = \\ &= \left(\sum_{i=1}^{k-1} c(v_i, v_{i+1}) \right) + \pi(v_1) - \pi(v_k) = c(P) + \pi(v_1) - \pi(v_k). \end{aligned}$$

Falls $C = [v_0, v_1, \ldots, v_k]$ ein Kreis ist, folgt analog zu oben $c^\pi(C) = c(C)$.

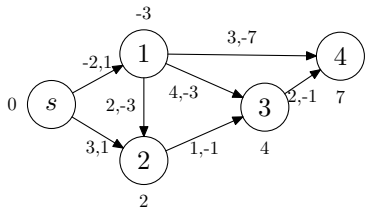

Abbildung 2.2: Beispiel 2.1.10. Die Gewichte c und c^π sind auf den Kanten dargestellt.

Bemerkung 2.1.11. Sei $\pi : V \to \mathbb{R}$, dann gilt

1. P ist genau dann ein kürzester Weg von s nach t bzgl. c, wenn P ein kürzester Weg von s nach t bzgl. c^π ist.
2. Für jeden s-t-Weg P gilt $c^\pi(P) = c(P) + \pi(s) - \pi(t)$.
3. Für jeden Kreis C in G gilt $c^\pi(C) = c(C)$.

Satz 2.1. *Sei $G = (V, E)$ ein gerichteter Graph mit $c : E \to \mathbb{R}$. Dann existiert genau dann ein zulässiges Knotenpotential, wenn c konservativ ist.*

Beweis. Sei $\pi : V \to \mathbb{R}$ zulässig, dann ist $c^\pi(u, v) \geq 0$ für alle $(u, v) \in E$. Daraus folgt, dass $c^\pi(C) \geq 0$ für alle Kreise C in G gilt und daher ist $c(C) = c^\pi(C) \geq 0$ für alle Kreise C.

Sei umgekehrt c konservativ, dann führen wir einen neuen Knoten s' und Kanten (s', v) für alle $v \in V$ ein. Weiters setzen wir $c(s', v) = 0$. Dann hat dieser neue Graph keine negativen Kreise und $\pi(v) = d(s', v) < \infty$. Wegen Lemma 2.1.8 gilt dann $\pi(v) \leq \pi(u) + c(u, v)$ und daher auch $0 \leq c(u, v) + \pi(u) - \pi(v)$ für alle $(u, v) \in E$. Deswegen ist $\pi(v) = d(s', v)$ ein zulässiges Knotenpotential. □

Definition 2.1.12. Sei $G = (V, E)$ ein gerichteter Graph mit $c : E \to \mathbb{R}$ konservativ. Eine Arboreszenz $A = (V', E')$ mit Wurzel s und allen von s aus erreichbaren Knoten heißt Kürzeste-Wege-Arboreszenz (oder Kürzeste-Wege-Baum), falls der eindeutige Weg von s nach v in A ein kürzester Weg von s nach v in G ist.

Bemerkung 2.1.13. Eine Kürzeste-Wege-Arboreszenz $A = (V', E')$ ist eine kompakte Repräsentation der kürzesten Wege von s zu allen von s aus erreichbaren Knoten. (Abbildung 2.3)

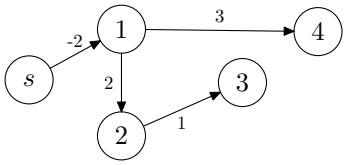

Abbildung 2.3: Kürzeste-Wege-Arboreszenz für den Graphen im Beispiel 2.1.10. Hier sind alle Knoten von s aus erreichbar, daher ist A eine spannende Arboreszenz.

Bemerkung 2.1.14. Man kann leicht zeigen, dass für konservative Kantengewichte eine Kürzeste-Wege-Arboreszenz existiert.

2.2 Ein allgemeiner Algorithmus zur Berechnung von kürzesten Wegen

Wir geben nun ein Grundgerüst zur Berechnung von kürzesten Wegen an. Im Algorithmus 2.1 tritt zur Bestimmung von kürzesten Wegen ein sogenannter Vorgängergraph $A = (V', E')$ auf, der

am Ende des Algorithmus den Kürzesten-Wege-Baum darstellt (falls die Kantenkosten tatsächlich konservativ sind). Dieser Graph sei durch $V' = \{v \in V : \text{pred}(v) \neq \text{null}\} \cup \{s\}$ und $E' = \{(\text{pred}(v), v) : v \in V' \setminus \{s\}\}$ definiert.

Algorithmus 2.1 Kürzeste-Wege-Problem

Gegeben: Gerichteter, zusammenhängender Graph $G = (V, E)$ mit beliebigen Kantengewichten $c : E \to \mathbb{R}$

Gesucht: Eine Kürzeste-Wege-Arboreszenz $A = (V', E')$

```
INIT(G, s)
while π kein zulässiges Knotenpotential do
  TEST(u, v)
end while

INIT(G, s){Unterroutine}
for all v ∈ V do
  π(v) = ∞
  pred(v) = null
end for
π(s) = 0

TEST(u, v){Unteroutine}
if c^π(u, v) < 0 dh. π(v) > c(u, v) + π(u) then
  π(v) = c(u, v) + π(u)
  pred(v) = u
end if
```

Lemma 2.2.1. *Im Algorithmus 2.1 gelten während der Ausführung folgende Bedingungen:*

1. $c^\pi(u, v) \leq 0 \ \forall (u, v) \in E'$
2. *Jeder Kreis in $A = (V', E')$ hat negatives Gewicht bzgl. c^π und c.*
3. *Falls in G von s aus kein Kreis mit negativer Länge erreicht werden kann, so ist A eine Arboreszenz mit Wurzel s und A enthält für alle $v \in V' \setminus \{s\}$ einen Weg von s nach v der Länge höchstens $\pi(v)$, der durch $P = [s, \text{pred}^{(k)}(v), \ldots, \text{pred}(\text{pred}(v)), \text{pred}(v), v,]$ gegeben ist.*

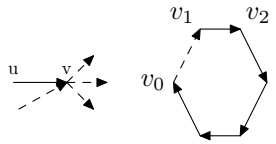

Abbildung 2.4: links: zur Bedingung 1, rechts: zur Bedingung 2

Beweis. Wir führen den Beweis durch Induktion nach Anzahl der TEST(u, v)-Aufrufe. Nach der Initialisierung sind die Aussagen 1 bis 3 des Satzes offenbar erfüllt. Wenn vor einem TEST(u, v)-Aufruf $\pi(v) \leq \pi(u) + c(u, v)$ gilt, wird nichts geändert und die Aussagen bleiben richtig. In diesem Fall ist also nichts zu zeigen.

1. Wenn vor einem TEST(u, v)-Aufruf $\pi(v) > \pi(u) + c(u, v)$ ist, dann gilt nach TEST(u, v): $c^\pi(u, v) = 0$. Weiters gilt $c^\pi(v, w) \leq 0$ für alle Kanten $(v, w) \in E'$ vor TEST(u, v) nach Induktionsvoraussetzung. Da $\pi(v)$ in TEST(u, v) erniedrigt wird, kann $c^\pi(v, w) = c(v, w) +$

$\pi(v) - \pi(w)$ nur sinken und die Aussage bleibt nachher ebenfalls gültig. Für Kanten, die nicht mit v inzidieren bleibt alles unverändert und damit ist die erste Aussage bewiesen.

2. Sei $C = [v_0, \ldots, v_k]$ ein Kreis in A. Dann gilt nach Aussage 1 $c^\pi(C) \leq 0$. Sei o.B.d.A. (v_0, v_1) die letzte Kante, deren Hinzunahme den Kreis C in einer Iteration geschlossen hat. Dann galt vor dieser Iteration, dass $\pi(v_1) > c(v_0, v_1) + \pi(v_0)$, also $c^\pi(v_0, v_1) < 0$ und damit auch $c^\pi(C) = c(C) < 0$.

3. Für alle $u \in V' \setminus \{s\}$ gilt $\text{pred}\, v \neq \text{null}$. Sei $v \in V$ beliebig und betrachte die Folge $v, \text{pred}(v)$, $\text{pred}(\text{pred}(v)), \ldots$, dh. wir erhalten eine Folge $v = v_0, v_1, \ldots$. Zwei Fälle können dabei auftreten:

   (a) Die Folge erreicht s und bricht mit $v_k = s$ ab. Dann ist $P = [s = v_k, v_{k-1}, \ldots, v_2, v_1, v_0 = v]$ ein Weg von s nach v, für den wegen Aussage 1 $c^\pi(P) \leq 0$ gilt. Daher folgt mit der Bemerkung 2.1.11

   $$c(P) \leq \pi(v) - \pi(s) \leq \pi(v) - d^c(s, s) = \pi(v).$$

   (b) Irgendwann gilt $v_j = v_{j+l}$ und $C = [v_j, v_{j+1}, \ldots, v_{j+l-1}, v_j]$ ist ein Kreis (wenn j und l minimal gewählt werden). Dieser Kreis hat wegen Aussage 2 negative Länge. Da $\pi(u) < \infty$ für alle Knoten $u \in C$ ist der Kreis C von s aus erreichbar, was im Widerspruch zur Annahme steht.

   Wir zeigen noch, dass A eine Arboreszenz mit Wurzel s ist und verwenden dabei die Charakterisierung 8 von Satz 1.6. Da jeder Knoten $v \in V' \setminus \{s\}$ einen eindeutigen Vorgänger $\text{pred}(v)$ hat, gilt $|\delta^-(v)| = 1$ für alle $v \in V' \setminus \{s\}$. Wäre $|\delta^-(s)| \geq 1$, dann wäre der Eingangsgrad jedes Knotens in A größer gleich 1 und es gibt einen Kreis C, der nach Aussage 2 eine negative Länge hat und von s aus erreichbar ist. Dies ist ein Widerspruch zur Annahme.

□

Bemerkung 2.2.2. Falls der Algorithmus 2.1 terminiert, liefert $A = (V', E')$ eine Kürzeste-Wege-Arboreszenz, da

$$\begin{aligned} c^\pi(u, v) &\geq 0 \quad \forall (u, v) \in E' \quad \text{(Algorithmus terminiert)} \\ c^\pi(u, v) &\leq 0 \quad \forall (u, v) \in E' \quad \text{(Bedingung 1)} \end{aligned}$$

Daraus folgt $d^{c^\pi}(s, v) = 0$ für alle in G von s aus erreichbaren Knoten v und damit ist $A = (V', E')$ Optimallösung weil $c^\pi(u, v) \geq 0$ für alle $(u, v) \in E$ gilt. Die Länge eines kürzesten Weges von s nach t entspricht dann dem Wert $\pi(t)$.

In den nächsten Abschnitten geben wir eine geeignete Folge von Test(u, v)-Schritten an, die garantiert dass der Algorithmus nach polynomiell vielen Schritten terminiert.

2.3 Der Algorithmus von Dijkstra

In diesem Abschnitt betrachten wir den Spezialfall $c(e) \geq 0$ für alle $e \in E$. Die Idee des Algorithmus von Dijkstra ist es, eine Menge $Q \subseteq V$ von permanent markierten Knoten (das sind Knoten für die $\pi(v) = d(s, v)$ gilt) zu finden, deren $\pi(v)$ dann im weiteren Verlauf des Algorithmus nicht mehr geändert wird. Am Beginn des Algorithmus ist $Q = \{s\}$ und in jeder Iteration fügen wir einen Knoten $v \in V \setminus Q$ zu Q hinzu. Dieser Knoten v hat den kleinsten $\pi(v)$-Wert unter allen Knoten in $V \setminus Q$.

Die Korrektheit des Algorithmus von Dijkstra liefert der folgende Satz:

Satz 2.2. *Der Algorithmus 2.2 löst das Kürzeste-Wege-Problem.*

Algorithmus 2.2 Algorithmus von Dijkstra

Gegeben: Gerichteter, zusammenhängender Graph $G = (V, E)$ mit nicht-negativen Kantengewichten $c : E \to \mathbb{R}_{\geq 0}$

Gesucht: Eine Kürzsete-Wege-Arboreszenz $A = (V', E')$

```
INIT(G, s)
Q = ∅
while Q ≠ V do
  Wähle u ∈ V \ Q mit minimalem π(u)
  Q = Q ∪ {u}
  for all v in der Nachbarschaft von u do
    TEST(u, v)
  end for
end while
```

Beweis. Wir müssen zeigen, dass $\pi(v) = d(s, v)$ für alle $v \in Q$ am Ende des Algorithmus gilt. Dazu verwenden wir Induktion nach der Anzahl der Schleifendurchläufe Am Anfang gilt $Q = \{s\}$ und $\pi(s) = 0 = d(s, s)$.
Für den Induktionsschritt sei $u \in V \setminus Q$ der zu Q hinzugefügte Knoten. Angenommen: $\pi(u) > d(s, u)$. Sei $P = [s = v_0, v_1, \ldots, v_k = u]$ ein Weg der Länge $d(s, u)$. Sei v_i der erste Knoten in P mit $v_i \notin Q$. Dann ist $i \geq 1$, da $v_0 = s$ und $s \in Q$ und $v_{i-1} \in Q$. Nach Induktionsvoraussetzung gilt $\pi(v_{i-1}) = d(s, v_{i-1})$. Nach dem TEST-Schritt gilt am Ende der Iteration:

$$\pi(v_i) \leq \pi(v_{i-1}) + c(v_{i-1}, v_i) = d(s, v_{i-1}) + c(v_{i-1}, v_i) = d(s, v_i) \leq d(s, u) < \pi(u)$$

was im Widerspruch zur Wahl von u steht. □

Laufzeit einfache Implementierung: $\mathcal{O}(n^2)$
Laufzeit mit Fibonacci-Heaps: $\mathcal{O}(m + n \log n)$

Beispiel 2.3.1. Wir lösen das Kürzeste-Wege-Problem für den Graphen in Abbildung 2.5.und erhalten folgende Tabelle (jede Zeile entspricht einer Iteration).

	$1 = s$	2	3	4	5
INIT	0	∞	∞	∞	∞
1		4, Vorgänger: 1	∞	∞	1, Vorgänger: 1
2		3, Vorgänger: 5	7, Vorgänger: 5	4, Vorgänger: 5	
3			5, Vorgänger: 2	4, Vorgänger: 5	
4			5, Vorgänger: 2		

Die ersten beiden TEST-Aufrufe noch einmal detailliert:

1. TEST$(1, 2)$: $\pi(2) > c(1, 2) + \pi(1)$? ja, dh. $\pi(2) = 4 + 0$ und pred$(2) = 1$
2. TEST$(1, 5)$: $\pi(5) > c(1, 5) + \pi(1)$? ja, dh. $\pi(5) = 1 + 0$ und pred$(5) = 1$

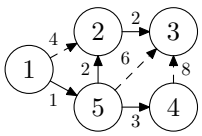

Abbildung 2.5: Beispiel 2.3.1

2.4 Der Algorithmus von Bellmann und Ford

In diesem Abschnitt betrachten wir positive und negative Kantenlängen. Für diesen allgemeinen Fall liefert der Algorithmus von Dijkstra 2.2 nicht mehr die Optimallösung.

Beispiel 2.4.1. Versuche wir das Kürzeste-Wege-Problem für den Graphen in Abbildung 2.6 mit dem Algorithmus von Dijkstra zu lösen, erhalten wir folgende Tabelle:

	1	2	3
1	0	3	4
2		3	4
3			4

Diese Ergebnis ist jedoch falsch, da $d(1,2) = 0$ gilt.

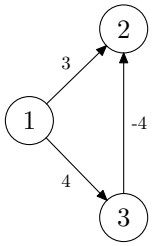

Abbildung 2.6: Beispiel 2.4.1

Der folgende Algorithmus von Bellmann und Ford löst auch dieses allgemeinere Problem. Wiederum geben wir nur eine genaue Abfolge der Test(u, v)-Schritte vor.

Algorithmus 2.3 Algorithmus von Bellmann und Ford

$\pi(v) = \infty \; \forall v \in V$
$\pi(s) = 0$
$\text{pred}(v) = \text{null} \; \forall v \in V$
for $k = 1$ **to** $n - 1$ **do**
  for all $(u, v) \in E$ **do**
    TEST(u, v)
  end for
end for

Lemma 2.4.2. *Am Ende von Phase $k = 1, \ldots, n-1$ im Algorithmus 2.3 gilt $\pi(v) \leq \min\{c(P) : P$ ist ein Wege von s nach v mit höchstens k Kanten$\}$.*

Beweis. Wir zeigen dieses Lemma mit Induktion nach der Anzahl der Phasen: Für $k = 1$ ist die Aussage trivial. Induktionsschritt: Wir zeigen, dass für alle $v \in V$ nach der Phase k der Wert $\pi(v)$ höchstens so groß ist wie jeder Weg von s nach v mit genau k Kanten. Da die Werte $\pi(v)$ im Laufe des Algorithmus nicht erhöht werden, folgt die Aussage mit der Induktionsvoraussetzung. Sei $P = [s = v_0, \ldots, v_k = v]$ ein Weg von s nach v, der genau k Kanten enthält. Dann ist der Weg $W = [s = v_0, \ldots, v_{k-1} = u]$ ein Weg von s nach v_{k-1} mit $k - 1$ Kanten und nach Induktionsvoraussetzung gilt $\pi(v_{k-1}) \leq c(W)$. Wir betrachten die Kante (v_{k-1}, v_k) in der k-ten Phase. Nach TEST(v_{k-1}, v_k) gilt:

$$\pi(v_k) \leq \pi(v_{k-1}) + c(v_{k-1}, v_k) \leq c(W) + c(v_{k-1}, v_k) = c(P).$$

□

Bemerkung 2.4.3. Dieses Lemma gilt für beliebige Kantengewichte, also auch falls der Graph negativen Kreise hat.

Satz 2.3. *Sei G ein Graph mit konservativen Kantenlängen, dann liefert der Algorithmus von Bellmann und Ford 2.3 eine spannende Kürzeste-Wege-Arboreszenz.*

Beweis. Für konservative Kantengewichte hat jeder kürzeste Weg höchstens $n-1$ Kanten und mit dem Lemma 2.4.2 ist alles gezeigt. □

Laufzeit: $\mathcal{O}(nm)$

Beispiel 2.4.4. Wir lösen das Kürzeste-Wege-Problem für den Graphen in Abbildung 2.7 und erhalten folgende Tabelle:

	s	1	2	3
0	0	∞	∞	∞
1	0	3, Vorgänger: s	2, Vorgänger: s	∞
2	0	-1, Vorgänger: v_2	2, Vorgänger: s	4, Vorgänger: v_2

In der ersten Phase wurden die Kanten in dieser Reihenfolge getestet: (v_3, v_2), (v_2, v_3), (v_2, v_1), (v_3, s), (s, v_2), (s, v_1), und in der zweiten Phase: (v_3, v_2), (v_3, s), (s, v_2), (s, v_1), (v_2, v_1), (v_2, v_3). Dannach ändert sich nichts mehr.

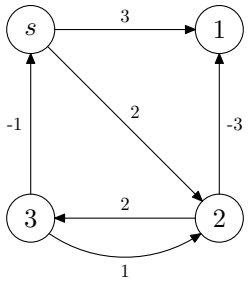

Abbildung 2.7: Beispiel 2.4.4

Wir möchten mit dem Algorithmus von Bellmann und Ford auch feststellen können, ob es negative Kreise gibt. Dazu führen wir am Ende des Algorithmus führen wir für jede Kante $(u, v) \in E$ ein TEST(u, v) durch. Dabei können zwei Möglichkeiten auftreten:

1. $\pi(v) \leq \pi(u) + c(u, v)\ \forall (u, v) \in E$ Dann ist π ein zulässiges Knotenpotential und es existieren laut Satz 2.1 keine negativen Kreise.

   Wenn kein negativer Kreis existiert, dann ist $\pi(v) = d(s, v)$ am Ende des Algorithmus von Bellmann und Ford und

   $$d(s, v) \leq d(s, u) + c(u, v) \Rightarrow \pi(v) \leq \pi(u) + c(u, v).$$

2. Sei $\pi(v) > \pi(u) + c(u, v)$ am Ende des Algorithmus von Bellmann und Ford für eine Kante (u, v). Wie finden wir den negativen Kreis? Betrachten wir von $v_0 = u$ solange die Vorgänger, dh. $v_i = \text{pred}(v_{i-1})$, bis einer der beiden Fälle eintritt:

   (a) Wir wiederholen zum ersten Mal einen Knoten, dh. $v_i = v_{i-j}$, also haben wir einen Kreis, der nach Lemma 2.2.1, Bedingung 2 negative Länge hat.

   (b) Wir gelangen zu $v_i = s$ und pred(s) = null (sonst liegt s in einem Kreis). Dann behaupten wir, dass $v_j = v$ für $j \geq 0$.

Angenommen es wäre $P = [s = v_i, \ldots, v_0 = u, v]$ ein elementarer Weg von s nach v, dann hat dieser Weg höchstens $n-1$ Kanten. Außerdem gilt:

$$\begin{aligned}
c(P) &= c^\pi(P) + \pi(v) - \pi(s) \\
&= c^\pi(P) + \pi(v) && \text{weil } \mathrm{pred}(s) = \mathrm{null} \Rightarrow \pi(s) = 0 \\
&\leq c^\pi(u,v) + \pi(v) && \text{weil für Kanten } (u,v) \text{ gilt } c^\pi(u,v) \leq 0 \\
&= c(u,v) + \pi(u) - \pi(v) + \pi(v) && \text{Definition von } c^\pi(u,v) \\
&= c(u,v) + \pi(u) \\
&< \pi(v) && \text{negativer Kreis}
\end{aligned}$$

Widerspruch zum Lemma 2.4.2.

In beiden Fällen haben wir also einen Kreis negativer Länge gefunden. Außerdem muss einer der beiden Fälle auftreten, da $\mathrm{pred}(v) \neq \mathrm{null}$ für alle $v \in V' \setminus \{s\}$ gilt. Damit kann man für ein gegebenes s einen Kreis negativer Länge finden, der von s aus erreichbar ist oder feststellen, dass kein solcher existiert. Sucht man allgemein einen negativen Kreis, kann man einen neuen Knoten s' und neue Kanten (s', v) für alle $v \in V$ einführen. Da von s' ausgehend nun jeder Knoten erreicht werden kann, kann s' nun als gegebener Startknoten verwendet werden um negative Kreise zu finden.

2.5 Kürzeste Wege für alle Knotenpaare

Kürzeste Wege zwischen allen Knotenpaaren	
Gegeben:	$G = (V, E)$, konservative Kantenlängen $c : E \to \mathbb{R}$
Gesucht:	$D = (d_{ij})_{1 \leq i \leq n, 1 \leq j \leq n}$ mit $d_{ij} = d(v_i, v_j)$ für $\forall v_i, v_j \in V$

erste Idee: Wir lösen für jeden Knoten ein Kürzestes-Wege-Problem, zB. mit dem Algorithmus von Bellmann und Ford 2.3. Laufzeit: $\mathcal{O}(n^2 m)$

zweite Idee: Wir berechnen in $\mathcal{O}(mn)$ ein zulässiges Knotenpotential $\pi : V \to \mathbb{R}$ mit dem Algorithmus von Bellmann und Ford. Danach bestimmen wir in $\mathcal{O}(m + n \log n)$ für jeden Knoten den kürzesten-Wegbaum bzgl. c^π (≥ 0) mit dem Algorithmus von Dijkstra 2.2. Laufzeit: $\mathcal{O}(mn + n(m + n \log n)) = \mathcal{O}(mn + n^2 \log n)$

dritte Idee: Algorithmus von Floyd und Warshall 2.4

Sei $V = \{v_1, \ldots v_k\}$. Wir definieren $d^k(v_i, v_j)$ als die Länge des kürzesten Weges von v_i nach v_j wobei dieser Weg nur die Knoten $v_1, \ldots, v_k$ enthält. Es gilt

$$d^0(v_i, v_j) = \begin{cases} c(v_i, v_j) & \text{falls } (v_i, v_j) \in E \\ \infty & \text{sonst} \end{cases}$$

$$d^{k+1}(v_i, v_j) = \min\{d^k(v_i, v_j), d^k(v_i, v_{k+1}) + d^k(v_{k+1}, v_j)\}$$

Dynamisch Programmierung liefert einen $\mathcal{O}(n^3)$-Algorithmus. Dabei können auch die kürzesten Wege leicht mitgespeichert werden (sh. Algorithmus 2.4)

Algorithmus 2.4 Algorithmus von Floyd und Warshall

Gegeben: Gerichteter Graph $G = (V, E)$ mit konservativen Kantengewichten $c : E \to \mathbb{R}$

Gesucht: Länge der kürzesten Wege $d(i, j)$ für alle Kanten (v_i, v_j) und eine Matrix $P = (p_(i, j))_{1 \leq i,j \leq n}$ wobei $(p(i, j), v_j)$ die letzte Kante eines solchen Weges ist.

$d(i, j) := \infty$ für alle $i, j = 1, \ldots, n$
$d(i, j) := c(v_i, v_j)$ falls $(v_i, v_j) \in E$
$d(i, i) := 0$ für alle $i = 1, \ldots, n$
$p(i, j) := i$ für alle $i, j \in V$
for $k = 1$ **to** n **do**
  for $i = 1$**to** n, $i \neq k$ **do**
    for $j = 1$**to** n, $j \neq k$ **do**
      if $d(i, j) > d(i, k) + d(k, j)$ **then**
        $d(i, j) := d(i, k) + d(k, j)$
        $p(i, j) := p(k, j)$
      end if
    end for
  end for
end for

Kapitel 3

Maximale Flüsse in Netzwerken

3.1 Max-Flow-Min-Cut-Satz

In diesem Kapitel werden wir Algorithmen für das maximale Flussproblem entwickeln. Dieses Problem kann als lineares Programm formuliert werden. Da die Restriktionen jedoch eine spezielle Gestalt haben, kann ein dem Problem angepasste Lösungsverfahren entwickelt werden. Von nun an bezeichnen wir einen gerichteten Graphen $G = (V, E)$ mit zwei ausgezeichneten Knoten $s, t \in V$ und Kantenkapazitäten $u : E \to \mathbb{R}_+$ als Netzwerk und schreiben dafür (G, u, s, t). Der Knoten s heißt Quelle und der Knoten t Senke.

Definition 3.1.1. Sei (G, u, s, t) ein Netzwerk und $f : E(G) \to \mathbb{R}_{\geq 0}$ dann definieren wir mit

$$\mathrm{ex}_f(v) := \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e)$$

den Überschuss im Knoten v bzgl. f.
f heißt Fluss wenn folgende Bedingungen erfüllt sind

1. $f(e) \leq u(e)\ \forall e \in E$
2. $\mathrm{ex}_f(v) = 0\ \forall v \in V \setminus \{s, t\}$

Der Wert eines Flusses f ist definiert als $v(f) := -\mathrm{ex}_f(s)$. Gilt $\mathrm{ex}_f(v) = 0$ für alle Knoten $v \in V$, dann heißt f Zirkulation oder Strömung.

Bemerkung 3.1.2. Die Gleichung $\mathrm{ex}_f(v) = 0$ nennt man Flusserhaltungsgleichungen für den Knoten v. Auf Grund der Flusserhaltungsgleichungen gilt $v(f) = -\mathrm{ex}_f(s) = \mathrm{ex}_f(t)$ für jeden Fluss f in (G, u, s, t).

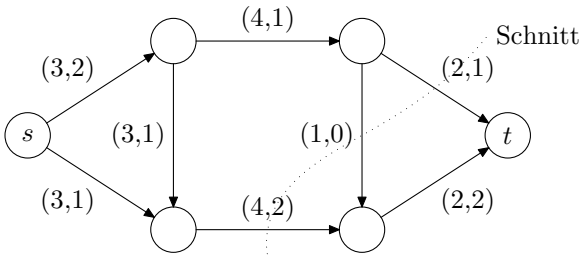

Abbildung 3.1: Ein Netzwerk mit einem Fluss: auf den Kanten steht $(u(e), f(e))$ und der Schnitt hat eine Kapazität von 7

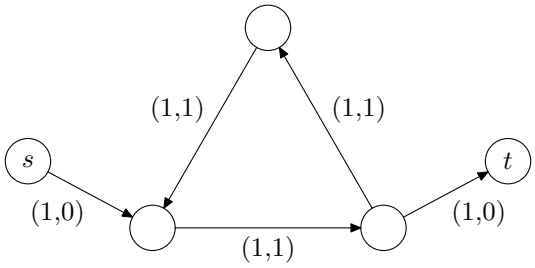

Abbildung 3.2: Auch das ist ein Fluss.

Max-Flow-Problem	
Gegeben:	Netzwerk (G, u, s, t)
Gesucht:	Fluss f mit maximalem Flusswert $v(f)$

Bemerkung 3.1.3. Der Nullfluss, d.h. der Fluss mit $f(e) = 0$ für alle $e \in E$, ist in jedem Netzwerk zulässig. Da der maximale Flusswert nach oben durch $\sum_{i \in V} u(s, i)$ beschränkt ist und das Problem als lineares Programm geschrieben werden kann, hat das maximale Flussproblem eine endliche Optimallösung.

Wir werden später sehen, dass das maximale Flussproblem sehr eng mit dem sogenannten Min-Cut-Problem verwandt ist. Ein Schnitt in einem Netzwerk (G, u, s, t) ist eine Partition $(X, \bar{X})$ der Knotenmenge $V(G)$ mit $s \in X$ und $t \in \bar{X}$. Die Kapazität eines Schnittes ist als die Summe aller Kantenkapazitäten der Kanten im Schnitt definiert, d.h.

$$c(X, \bar{X}) := \sum_{e \in \delta^+(X)} u(e).$$

Damit können wir das Min-Cut-Problem formal definieren.

Min-Cut-Problem	
Gegeben:	Netzwerk (G, u, s, t)
Gesucht:	Schnitt $(X, \bar{X})$ mit minimaler Kapazität $c(X, \bar{X})$

Das nächste Lemma entspricht dem schwachen Dualitätssatz der linearen Optimierung.

Lemma 3.1.4. *Sei (G, u, s, t) ein Netzwerk und $(X, \bar{X})$ ein beliebiger Schnitt. Dann gilt für jeden Fluss f*

$$v(f) = \sum_{e \in \delta^+(X)} f(e) - \sum_{e \in \delta^-(X)} f(e) \tag{3.1}$$

und

$$v(f) \leq c(X, \bar{X}).$$

Beweis. Es gilt

$$\begin{aligned} v(f) &= \sum_{e \in \delta^+(s)} f(e) - \sum_{e \in \delta^-(s)} f(e) \\ &= \sum_{v \in X} \underbrace{\left(\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) \right)}_{=0 \quad \forall v \in X \setminus \{s\}} \\ &= \sum_{e \in \delta^+(X)} f(e) - \sum_{e \in \delta^-(X)} f(e). \end{aligned}$$

Daraus schließt man sofort

$$v(f) \overset{(3.1)}{=} \sum_{e\in\delta^+(X)} f(e) - \underbrace{\sum_{e\in\delta^-(X)} f(e)}_{\geq 0} \leq \sum_{e\in\delta^+(X)} f(e) \leq \sum_{e\in\delta^+(X)} u(e) = c(X\bar{X}).$$

□

Bemerkung 3.1.5. Gelingt es also einen Fluss f und einen Schnitt $(X, \bar{X})$ zu finden für den $v(f) = c(X, \bar{X})$ gilt, dann ist f ein maximaler Fluss und $(X, \bar{X})$ ein minimaler Schnitt. Wir werden zeigen, dass es für jedes Netzwerk tatsächlich eine Fluss f und einen Schnitt $(X, \bar{X})$ mit $v(f) = c(X, \bar{X})$ gibt.

Definition 3.1.6. Sei (G, u, s, t) ein Netzwerk und f ein Fluss. Dann ist das Inkrementnetzwerk (G_f, u_f, s, t) folgendermaßen definiert: $G_f = (V(G), E^+ \cup E^-)$ mit

$$E^+ := \{(v, w) : (v, w) \in E(G) \quad \text{und} \quad f(v, w) < u(v, w)\}$$

und

$$E^- := \{(v, w) : (w, v) \in E(G) \quad \text{und} \quad f(w, v) > 0\}$$

und den Kapazitäten $u_f : E^+ \cup E^- \to \mathbb{R}_+$ mit

$$u_f(v, w) = \begin{cases} u(v, w) - f(v, w) & \text{falls } (v, w) \in E^+ \\ f(w, v) & \text{falls } (v, w) \in E^- \end{cases}$$

Die Kanten in E^+ heißen Vorwärtskanten von G_f, die Kanten in E^- nennt man Rückwärtskanten. Ein Weg P von s nach t in G_f heißt f-augmentierender Weg. Den Fluss f entlang P um γ zu augmentieren, bedeutet einen neuen Fluss $f' := f \oplus P$ wie folgt zu konstruieren:

$$f'(v, w) = \begin{cases} f(v, w) + \gamma & \text{falls } (v, w) \in E^+ \cap E(P) \\ f(v, w) - \gamma & \text{falls } (w, v) \in E^- \cap E(P) \\ f(v, w) & \text{sonst} \end{cases}$$

Bemerkung 3.1.7. Ist P ein augmentierender Weg und $\gamma = \min\{u_f(e) : e \in E(P)\} > 0$, dann kann man sofort nachrechnen, dass $f' = f \oplus P$ wieder ein zulässiger Fluss ist und $v(f') = v(f) + \gamma$ gilt. Daraus folgt, dass es für einen maximalen Fluss f in G_f keinen augmentierenden Weg gibt.

Lemma 3.1.8. *Sei (G, u, s, t) ein Netzwerk und f ein Fluss. Gibt es in G_f keinen augmentierenden Weg, dann ist f ein maximaler Fluss.*

Beweis. Sei $X = \{v \in V : v \text{ ist in } G_f \text{ von } s \text{ aus erreichbar}\}$. Dann gilt nach Definition $s \in X$ und $t \notin X$ weil es keinen augmentierenden Weg in G_f gibt. Auf Grund der Definition von G_f folgen auch die beiden Aussagen:

1. $f(v, w) = u(v, w)$ für alle $(v, w) \in \delta^+(X)$
2. $f(v, w) = 0$ für alle $(v, w) \in \delta^-(X)$

Damit gilt mit der Gleichung in (3.1)

$$v(f) = \sum_{e\in\delta^+(X)} f(e) - \sum_{e\in\delta^-(X)} f(e) = \sum_{e\in\delta^+(X)} u(e) - 0 = c(X, \bar{X}).$$

□

Insgesamt haben wir also den folgenden Satz bewiesen.

Algorithmus 3.1 Algorithmus von Ford und Fulkerson

Gegeben: Netzwerk (G, u, s, t)
Gesucht: Ein maximaler Fluss f in G
  Setze $f(e) = 0$ alle $e \in E(G)$
  while Es gibt einen f-augmentierenden Weg P in G_f **do**
    Berechne $\gamma = \min_{e \in E(P)} u_f(e)$
    Augmentiere f entlang von P um γ
  end while

Satz 3.1 (Ford und Fulkerson, 1956)**.** *Sei (G, u, s, t) ein Netzwerk, dann existiert ein Fluss f und ein Schnitt $(X, \bar{X})$ mit $v(f) = c(X, \bar{X})$. Insbesonders sind der Wert eines maximalen Flusses und die Kapazität eines minimalen Schnittes gleich.*

Der Beweis dieses Satzes motiviert den folgenden Algorithmus 3.1.

Wenn wir annehmen, dass alle Kapazitäten ganzzahlig sind, dann sieht man leicht ein, dass der aktuelle Fluss zu jedem Zeitpunkt des Algorithmus ganzzahlig ist und auch die Kapazitäten im Inkrementnetzwerk als Differenz von zwei ganzen Zahlen wieder ganzzahlig sind. Außerdem erhöht sich der Wert des Flusses in jeder Iteration um mindestens 1. Ist $U := \max_{e \in E} u(e)$ dann hat der minimale Schnitt eine Kapazität von höchstens $(n-1)U$. Daraus kann man folgern, dass der Algorithmus 3.1 nach maximal $(n-1)U$ Iterationen abbricht. Da ein augmentierender Weg in $\mathcal{O}(m)$ Zeit gefunden werden kann, terminiert der Algorithmus nach $\mathcal{O}(Unm)$ Zeit mit einem ganzzahligen maximalen Fluss. Damit ist dieses Verfahren jedoch nur pseudopolynomiell. Wir werden später die Wahl des augemntierenden Weges vorgeben und damit einen polynomiellen Algorithmus erhalten.
Sind alle Kapazitäten rationale Zahlen, erhält man durch Multiplikation mit dem Hauptnenner wiederum ein Netzwerk mit ganzzahligen Kapazitäten und der Algorithmus 3.1 terminiert wieder nach endlich vielen Schritten.
Für irrationale Kapazitäten gibt es allerdings Beispiele, in denen der Algorithmus nicht terminiert und die Folge der Flusswerte nicht gegen den optimalen Flusswert konvergiert.

Zum Abschluss dieses Abschnitts zeigen wir noch, dass Satz 3.1 auch mit Hilfe der Dualitätstheorie aus der linearen Optimierung bewiesen werden kann.

Beweis. (von Satz 3.1 mit Dualitätstheorie)
Sei (G, u, s, t) ein Netzwerk, dann führen wir eine neue Kante (t, s) mit $u(t, s) = \infty$ ein und suchen eine Zirkulation, die den Fluss auf der neuen Kanten (t, s) maximiert. Dieses Problem kann man in folgender Weise als LP schreiben:

$$\begin{array}{llrcll} \text{(LP)} & \max & x(t,s) & & & \\ & \text{s.t.} & Ax & = & 0 & \\ & & 0 \leq & x(e) & \leq u(e) & \forall e \in E \end{array}$$

wobei A die Knoten-Kanten-Inzidenzmatrix von $G + (t, s)$ ist. Für das duale Problem verwenden wir die Variablen $y(e) \geq 0$ für alle $e \in E$ und $z(v)$ für alle $v \in V$. Das duale Problem lautet damit

$$\begin{array}{llrcll} \text{(DP)} & \min & \sum_{e \in E(G)} u(e)y(e) & & & \\ & \text{s.t.} & z(i) - z(j) + y(i,j) & \geq & 0 & \forall (i,j) \in E(G) \\ & & z(t) - z(s) & \geq & 1 & \\ & & y(i,j) & \geq & 0 & \forall (i,j) \in E(G) \end{array}$$

Die Koeffizientenmatrix, die die Nebenbedingungen des dualen Problems angibt, hat die Form $(A^T I)$, wobei I die Einheitsmatrix ist. Da A vollständig unimodular ist, ist auch $(A^T I)$ vollständig unimodular und das duale Problem hat eine ganzzahlige Optimallösung. Sei (y^*, z^*) eine Optimallösung vom dualen Problem, dann ist auch $(y^*, z^* - \alpha)$ für alle $\alpha \in \mathbb{R}$ eine Optimallösung. Sei also

(y^*, z^*) eine ganzzahlige Optimallösung mit $z^*(s) = 0$. Dann definieren wir die beiden Mengen

$$S = \{v \in V : z^*(v) \leq 0\} \quad \text{und} \quad T = \{v \in V : z^*(v) \geq 1\}.$$

(S, T) definiert nun einen Schnitt in (G, u, s, t) und

$$\begin{aligned} v(f^*) &\overset{\text{Dualität}}{=} \sum_{e \in E(G)} u(e) y^*(e) \\ &\geq \sum_{e \in \delta^+(S)} u(e) y^*(e) \\ &\geq \sum_{(i,j) \in \delta^+(S)} u(e) \left(\underbrace{z^*(j)}_{\geq 1} - \underbrace{z^*(i)}_{\leq 0} \right) \\ &\geq \sum_{(i,j) \in \delta^+(S)} u(e) = c(S, T). \end{aligned}$$

Mit dem Lemma 3.1.4 folgt nun der Satz. □

3.2 Der Algorithmus von Edmonds-Karp

Wie schnell ist der Algorithmus 3.1?

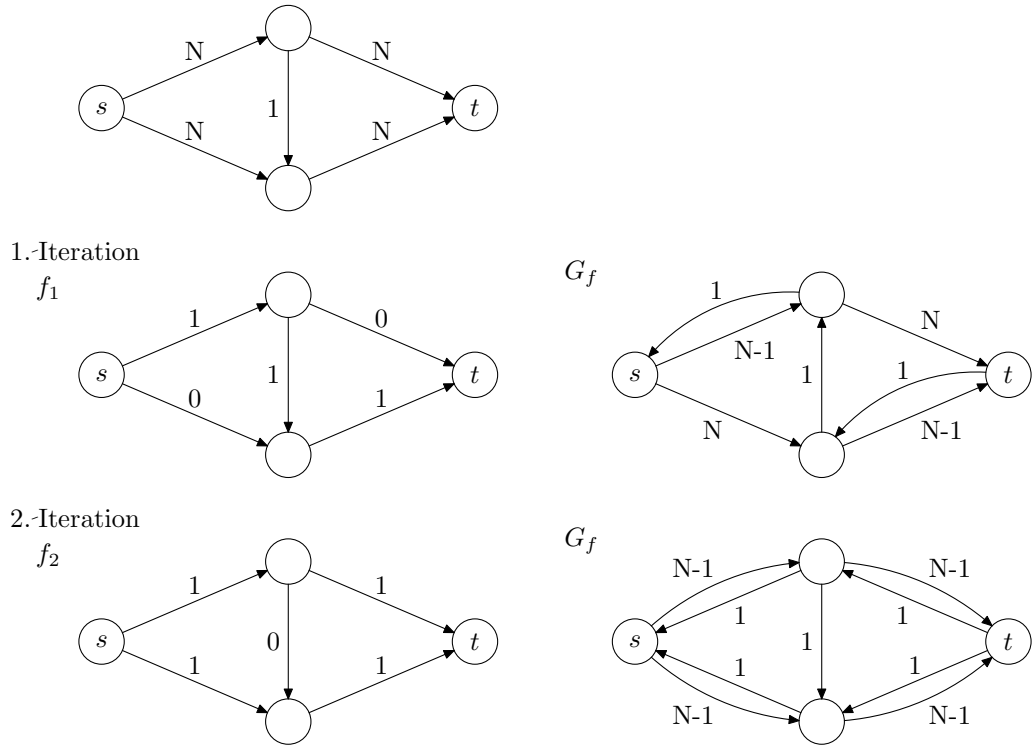

Abbildung 3.3: Der Algorithmus 3.1 ist nur pseudopolynomiell.

Beispiel 3.2.1. Bei der Wahl der augmentierenden Wege in Abbildung 3.3 benötigt man $2N$ Iterationen. Der Algorithmus 3.1 ist also nur pseudopolynomiell.

Der Algorithmus von Edmonds und Karp wurde 1972 veröffentlicht und schreibt die Wahl des augmentierenden Weges vor. Wir müssen immer einen kürzesten augmentierenden Weg im

Inkrementnetzwerk wählen, dh. einen Weg von s nach t mit einer minimalen Anzahl an Kanten. In diesem Abschnitt beweisen wir, dass diese Vorschrift einen polynomiellen Algorithmus liefert.

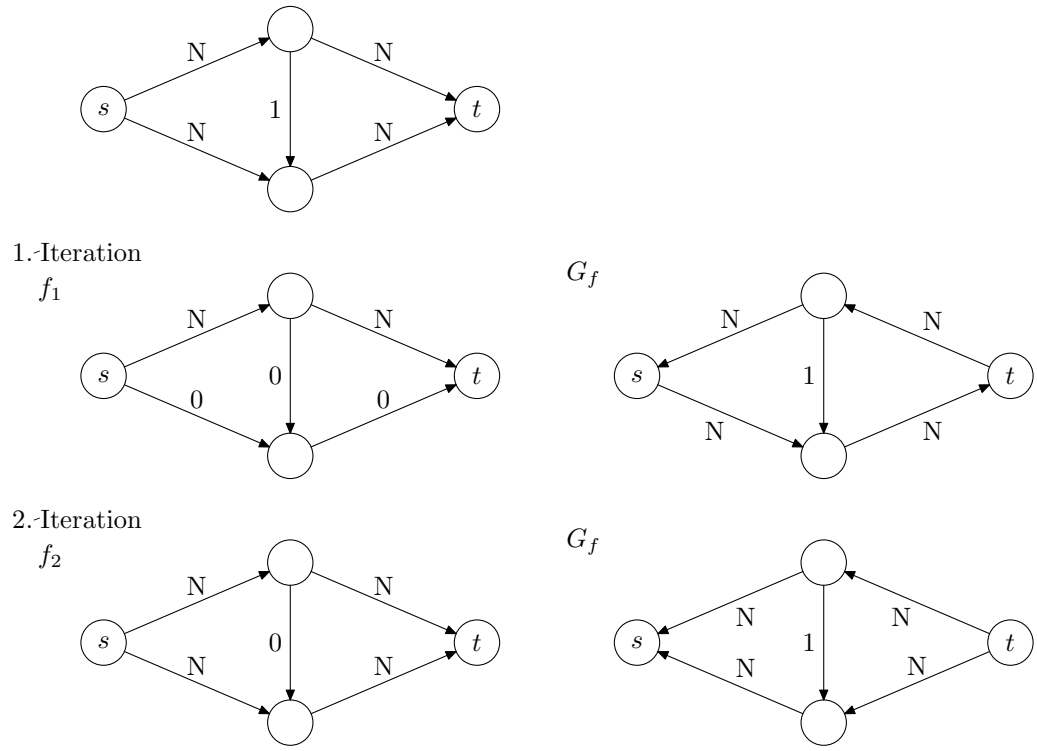

Abbildung 3.4: Hier benötigt man nur zwei Iterationen.

Lemma 3.2.2. *Sei $f_1, f_2, \ldots$ eine Folge von Flüssen, wobei $f_{i+1} = f_i \oplus P_i$ und P_i ein kürzester f_i-augmentierender Weg in G_{f_i} ist. Dann gilt*

1. *$|E(P_k)| \leq |E(P_{k+1})|$ für alle k.*
2. *$|E(P_k)| + 2 \leq |E(P_r)|$ für alle $k < r$, so dass $P_k \cup P_r$ ein Paar gegenläufiger Kanten enthält.*

Beweis. Wir zeigen die beiden Aussagen getrennt:

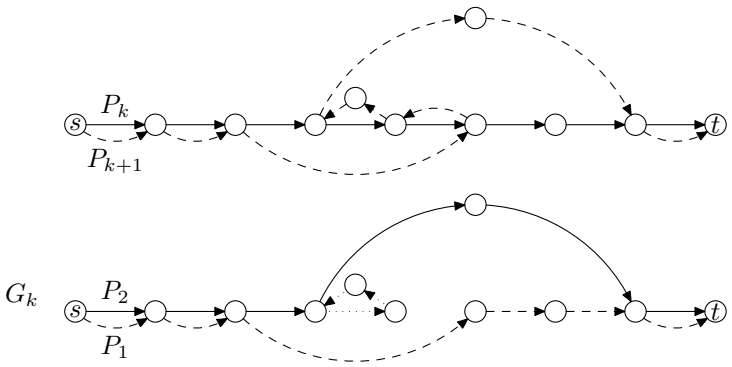

Abbildung 3.5: Oben die zwei Pfade P_k und P_{k+1} und unten der Graph G_k mit den Pfaden P_1 und P_2.

1. Wir definieren den Graphen $G_k = (V, E_k)$, der alle Kanten von den Pfaden P_k und P_{k+1} enthält, wobei alle Paare gegenläufiger Kanten entfernt werden.

   Bevor wir die erste Aussage des Satzes beweisen können, halten wir folgende zwei Tatsachen fest:

   - $E_k \subseteq E(G_{f_k})$
     Es gilt $P_k \subseteq E(G_{f_k})$. Angenommen es gibt eine Kante $e \in E(P_{k+1}) \subseteq E(G_{f_{k+1}})$ mit $e \notin E(G_{f_k})$, dann ist e neu hinzugekommen, also ist e eine entgegengesetzte Kante zu einer Kante $e' \in E(P_k) \subseteq E(G_{f_k})$. Solche Paare entgegengesetzter Kanten kommen aber in G_k nicht vor.
   - G_k enthält 2 kantendisjunkte Wege P_1 und P_2 von s nach t
     Fügen wir zu G_k zwei Kanten von t nach s ein, dann ist der neue Graph ein Eulergraph im gerichteten Sinne. Der Graph ist also kantendisjunkte Vereinigung von gerichteten Kreisen. Damit enthält G_k für jede der beiden zugefügten Kanten (t, s) einen Weg P_1 und P_2, die kantendisjunkt sind.

   Aus dem ersten Punkt folgt nun, dass P_1 und P_2 f_k-augmentierende Wege sind. Da P_k ein kürzester solcher Weg ist, folgt

   $$|E(P_k)| \leq |E(P_1)| \qquad \text{und} \qquad |E(P_k)| \leq |E(P_2)|.$$

   Insgesamt ergibt das

   $$2|E(P_k)| \leq |E(P_1)| + |E(P_2)| \leq |E(G_k)| \leq |E(P_k)| + |E(P_{k+1})|$$

   und damit $|E(P_k)| \leq |E(P_{k+1})|$.

2. Für den Beweis der zweiten Aussage betrachten wir k und r so, dass P_k und P_r entgegengesetzte Kanten haben, aber P_i und P_r für $k < i \leq r$ kein solches Paar besitzen. Sei $G_k = (V, E_k)$ nun der Graph, der alle Kanten von den Pfaden P_k und P_r enthält, wobei wiederum alle Paare gegenläufiger Kanten entfernt werden. Dann gilt

   - $E(G_k) \subseteq E(G_{f_k})$
     Angenommen eine Kante e ist in $E(P_r)$ aber nicht in $E(G_{f_k})$. Dann ist e neu hinzugekommen, also ist e entgegengesetzt zu einer Kante e' in $E(P_k)$ oder $E(P_{k+1})$ oder ... oder $E(P_{r-1})$. Auf Grund der Wahl von k und r ist $e' \in E(P_k)$ und damit sind e' und e nicht in G_k.
   - G_k enthält 2 kantendisjunkte Wege P_1 und P_2 von s nach t
     Der Beweis dieser Tatsache erfolgt analog wie oben.

   Insgesamt gilt dann also, dass P_1 und P_2 f_k-augmentierende Wege sind und

   $$2|E(P_k)| \leq |E(P_1)| + |E(P_2)| \leq |E(G_k)| \leq |E(P_k)| + |E(P_r)| - 2$$

   da mindestens ein Paar Kanten entfernt wurde. Daraus folgt $|E(P_k)| + 2 \leq |E(P_r)|$, was zu beweisen war.

□

Satz 3.2. *Der Algorithmus von Edmonds und Karp benötigt höchstens $\frac{nm}{2}$ augmentierende Wege.*

Beweis. Wir zählen, wie oft eine Kante e während des Algorithmus als Bottleneckkante eines augmentierenden Weges höchstens auftreten kann (e heißt Bottleneckkante wenn $u(e) = \min_{e \in P} u(e)$ und P ein im Algorithmus ausgewählter augmentierender Weg ist). Angenommen e ist eine Bottleneckkante bzgl. P_k und P_r mit $k < r$. Dann folgt mit obigem Lemma $|E(P_k)| + 4 \leq |E(P_r)|$, da e dann in P_r liegen kann, falls es einen Weg P_i mit $k < i < r$ gibt, der die zu e entgegengesetzte Kante enthält. Da die Länge jedes augmentierenden Weges kleiner gleich n ist, kann e höchstens $\frac{n}{4}$ mal eine Bottleneckkante sein. Dies gilt für jede Kante und deren entgegengesetzte Kante. Damit gibt es maximal $\frac{mn}{2}$ augmentierende Wege. □

Satz 3.3. *Der Algorithmus von Edmonds und Karp liefert einen maximalen Fluss in $\mathcal{O}(nm^2)$.*

Beweis. Mit Breitensuche kann ein augmentierender Weg in $\mathcal{O}(m)$ Zeit gefunden werden und es gibt $\mathcal{O}(nm)$ augmentierende Wege. □

3.3 Blockierende Flüsse

In diesem Abschnitt möchten wir einen Algorithmus angeben, der noch schneller ist als der Algorithmus von Edmonds und Karp. Die Idee des neuen Algorithmus beruht auf den folgenden zwei Bemerkungen:

- Es ist nicht notwendig das Inkrementnetzwerk nach jedem augmentierenden Weg zu aktualisieren.
- Wir benötigen im Inkrementnetzwerk nur Kanten, die auf kürzesten Wegen liegen. Daher müssen nicht alle Kanten berücksichtigt werden und wir erhalten einen Graphen mit speziellen Eigenschaften.

Bevor der Algorithmus angegeben wird, benötigt man noch folgende Definitionen :

Definition 3.3.1. Sei (G, u, s, t) ein Netzwerk und f ein zulässiger Fluss. Dann heißt f blockierend, wenn es im Graphen $G' = (V, E')$ mit $E' = \{e \in E(G) : f(e) < u(e)\}$ keinen Weg von s nach t gibt.

Definition 3.3.2. Sei (G, u, s, t) ein Netzwerk und f ein zulässiger Fluss, dann definieren wir den Schichtgraph oder Levelgraph G_f^L von G_f als

$$\left(V(G), \{e = (u, v) \in E(G_f) : d_{G_f}(s, u) + 1 = d_{G_f}(s, v)\}\right).$$

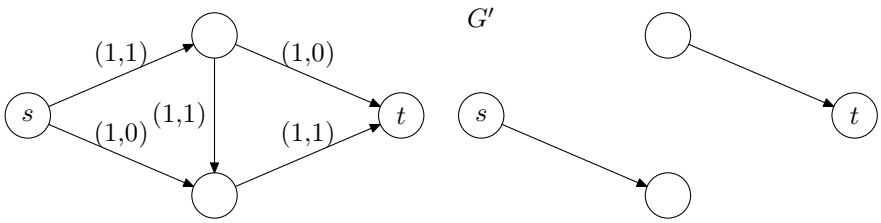

Abbildung 3.6: Ein blockierender aber nicht maximaler Fluss.

Bemerkung 3.3.3. Jeder maximale Fluss ist blockierend, aber nicht umgekehrt.

Bemerkung 3.3.4. G_f^L ist azyklisch und kann in $\mathcal{O}(m)$ Zeit berechnet werden (z.B. mit Breitensuche).

Mit diesen beiden Definitionen können wir nun den Algorithmus 3.2 angeben.

Satz 3.4. *Der Algorithmus 3.2 findet in $\mathcal{O}(n^2m)$ einen maximalen Fluss.*

Beweis. Ein blockierende Fluss kann in $\mathcal{O}(nm)$ Zeit berechnet werden, weil jeder augmentierende Weg von s nach t eine Länge kleiner gleich n hat und für jeden augmentierenden Weg gibt es eine Bottleneckkante, die gelöscht werden kann. Die Anzahl der Iterationen ist mit n beschränkt, da die Länge der kürzesten Wege in jeder Iteration zunimmt. Die Gesamtlaufzeit ist daher $\mathcal{O}(n^2m)$. □

Bemerkung 3.3.5. Die Konstruktion eines Schichtgraphens und das Auffinden eines blockierenden Flusses kann sogar in $\mathcal{O}(n^2)$ Zeit erfolgen. Mit dieser Verbesserung ergibt sich für den Algorithmus 3.2 eine Gesamtlaufzeit von $\mathcal{O}(n^3)$.

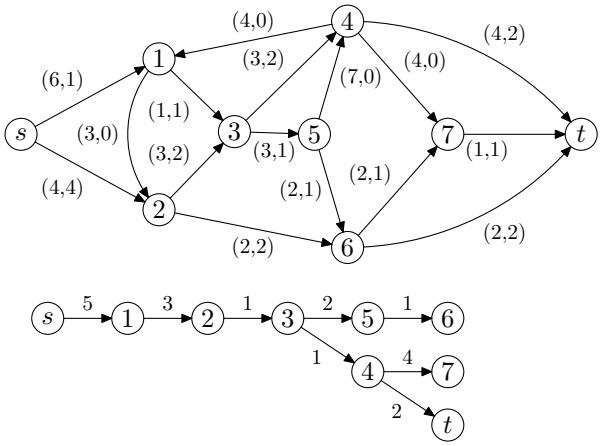

Abbildung 3.7: Ein Beispiel für einen Levelgraphen.

Algorithmus 3.2 Algorithmus von Dinic

Gegeben: Netzwerk (G, u, s, t)
Gesucht: Ein maximaler Fluss f in G

1: Setze $f(e) = 0$ alle $e \in E(G)$
2: Erstelle den Levelgraphen G_f^L von G_f
3: Bestimme einen blockierenden Fluss f' in G_f^L
4: **if** $v(f') = 0$ **then**
5: Optimallösung gefunden
6: **else**
7: Augmentiere f um f' und **goto** 2.
8: **end if**

3.4 Push-Relabel-Algorithmus von Goldberg und Tarjan

Wir haben gezeigt, dass ein Fluss $f : E \to \mathbb{R}_{\geq 0}$ genau dann ein maximaler Fluss in einem Netzwerk (G, u, s, t) ist, wenn die folgenden Bedingungen erfüllt sind:

1. $f(e) \leq u(e)$
2. $\mathrm{ex}_f(v) = 0 \ \forall v \in V \setminus \{s, t\}$
3. es gibt keinen Weg von s nach t in G_f

In den bisherigen Algorithmen war 1 und 2 immer erfüllt und das Ziel war es, das Optimalitätskriterium 3 zu erreichen. In diesem Kapitel wählen wir einen neuen Zugang. Wir halten 1 und 3 fest und terminieren falls 2 erfüllt ist. Das heißt, wir arbeiten während des Algorithmus nicht mit einem Fluss, sondern mit einem sogenannten Präfluss, der in folgender Definition formal eingeführt wird.

Definition 3.4.1. Sei (G, u, s, t) ein Netzwerk, dann heißt $f : E \to \mathbb{R}_{\geq 0}$ Präfluss, falls folgende Bedingungen erfüllt sind:

1. $f(e) \leq u(e)$ für alle $e \in E(G)$
2. $\mathrm{ex}_f(v) \geq 0$ für alle $v \in V \setminus \{s\}$.

Ein Knoten $v \in V \setminus \{s, t\}$ heißt aktiv, falls $\mathrm{ex}_f(v) > 0$ ist.

Bemerkung 3.4.2. Das Konzpt des Inkrementgraphen G_f kann direkt von Flüssen auf Präflüsse erweitert werden. Wir werden später davon sprechen, den Präfluss f entlang einer Kante $e \in E(G_f)$ um γ zu augmentieren. Dies bedeutet, dass wir den Fluss auf e um γ erhöhen falls e eine Vorwärtskante ist und ansonsten den Fluss auf der zu e entgegengesetzen Kante um γ Flusseinheiten zu verringern.

Ein weiteres wichtiges Konzept für den Push-Relabel-Algorithmus ist eine Distanzmarkierung. Diese stellt sicher, dass 3 während des Algorithmus erfüllt bleibt.

Definition 3.4.3. Sei $G = (V, E)$ ein gerichteter Graph und $s, t, \in V$. Dann heißt eine Funktion $d : V \to \mathbb{N}_0$ Distanzmarkierung falls

1. $d(s) = n$
2. $d(t) = 0$
3. $d(u) \leq d(v) + 1$ für alle $e = (u, v) \in E(G)$

Eine Kante (u, v) heißt zulässig, falls $d(u) = d(v) + 1$ ist.

Bemerkung 3.4.4. Sei $P = [v, v_1, \ldots, v_k = t]$ ein Weg von v nach t in G. Dann gilt

$$d(v) \leq d(v_1) + 1 \leq d(v_2) + 2 \leq \ldots \leq d(v_k) + k = d(t) + k = k,$$

dh. $d(v)$ ist höchstens so groß wie die Länge des kürzesten Weges von v nach t.

Bemerkung 3.4.5. Ist f ein Präfluss und d eine Distanzmarkierung in G_f, dann gilt $d(v) \leq |P|$ für jeden Weg P von v nach t in G_f. Da $d(s) = n$ gilt, ist t von s aus nicht erreichbar, weil jeder Weg eine Länge kleiner gleich $n - 1$ hat.
Ist f sogar ein Fluss und d eine Distanzmarkierung in G_f dann ist f optimal.

Algorithmus 3.3 Push-Relabel-Algorithmus

Gegeben: ein Netzwerk (G, u, s, t)
Gesucht: ein maximaler Fluss f
  Setze $f(e) := u(e)$ für jedes $e \in \delta^+(s)$
  Setze $f(e) := 0$ für jedes $e \in E(G) \setminus \delta^+(s)$
  Setze $d(s) := n$ und $d(v) := 0$ für alle $v \in V(G) \setminus \{s\}$
  while es gibt einen aktiven Knoten v **do**
    if kein $e \in \delta^+_{G_f}(v)$ ist zulässige Kante **then**
      RELABEL(v)
    else
      sei $e \in \delta^+_{G_f}(v)$ eine zulässige Kante und PUSH(e)
    end if
  end while

  PUSH$(v, w)\{$*Unterroutine*$\}$
  Setze $\gamma := \min\{\mathrm{ex}_f(v), u_f(v, w)\}$
  Augmentiere f entlang $e = (v, w)$ um γ

  RELABEL$(v)\{$*Unterroutine*$\}$
  Setze $d(v) := \min\{d(w) + 1 : (v, w) \in E(G_f)\}$

Beispiel 3.4.6. In Abbildung 3.4 ist zuerst das Ausgangsnetzwerk zu sehen, und dann die Inkrementnetzwerke mit Distanzmarkierungen, Überschüssen und Kapazitäten. Strichlierte Kanten sind nicht erlaubt, da die Distanzmarkierungen der Endpunkte nicht passen. Als erstes wird der Knoten 2 ausgewählt. Es gibt keine zulässige Kante, also wird gerelabelt zu $1 + \min\{4, 0\} = 1$. Danach ist

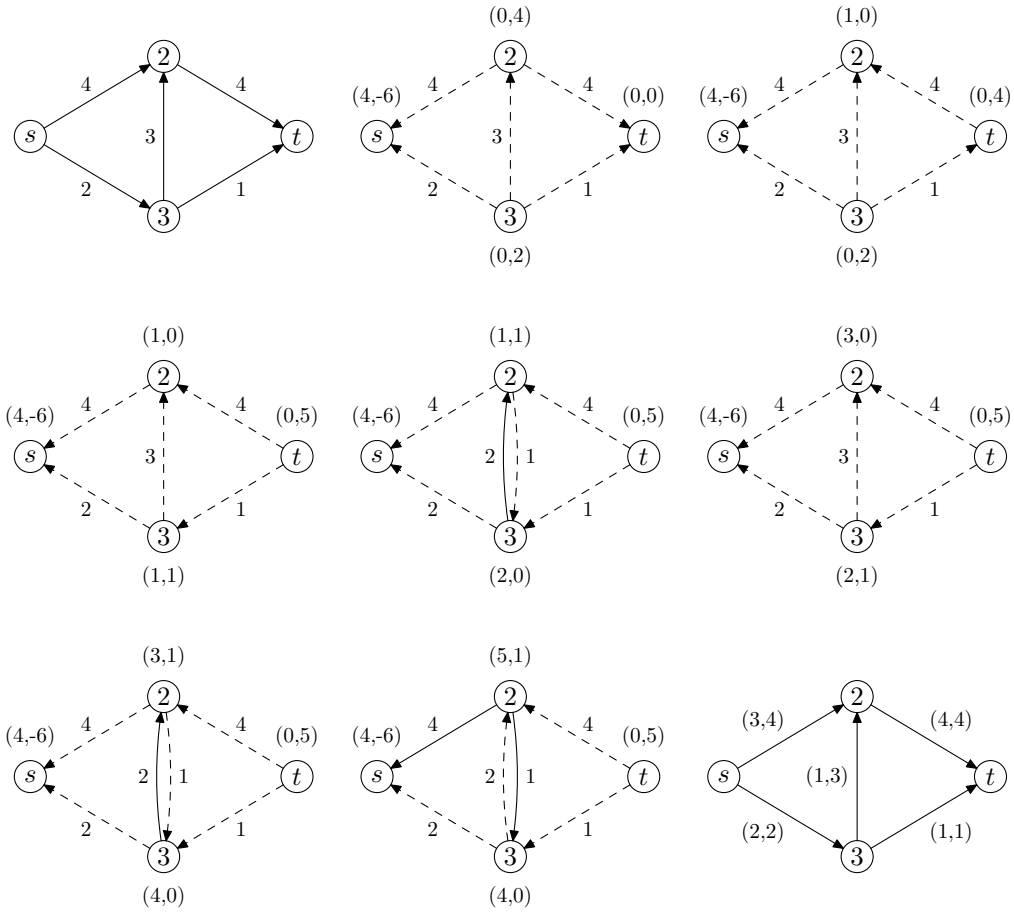

Abbildung 3.8: Graphen zum Beispiel 3.4.6.

die Kante $(2, t)$ zulässig, und es werden 4 gepusht. Jetzt ist der Knoten 3 der einzige aktive Knoten aber es gibt keine zulässigen Kanten, also wird zu $1 + \min\{4, 1, 0\} = 1$ gerelabelt. Dann wird über die Kante $(3, t)$ eine Einheit gepusht. Dann wird der Knoten 3 auf $1 + \min\{4, 1\} = 2$ gerelabelt, und über $(3, 2)$ eine Einheit gepusht. Diese eine Einheit wird dann noch zwischen den Knoten 2 und 3 ein paar mal hin und her gepusht und die beiden Distanzmarkierungen der Knoten immer erhöht, bis einer der Knoten (hier der Knoten 2) die Distanzmarkierung 5 hat. Dann kann die eine Einheit über die Kante $(2, s)$ zum Knoten s gepusht werden und der Algorithmus terminiert. Der letzte Graph stellt nochmal den optimalen Fluss dar.

Lemma 3.4.7. *Während des Push-Relabel-Algorithmus 3.3 bleibt d eine Distanzmarkierung und f ein Präfluss.*

Beweis. Am Begin des Algorithmus ist d eine Distanzmarkierung und f ein Präfluss. Wir müssen noch zeigen, dass jede PUSH- und RELABEL-Operation diese Eigenschaften nicht zerstört.

RELABEL(v)**:** Da sich f nicht ändert, bleibt f ein Präfluss. Weiters setzt man $d(v) = \min\{d(w) + 1 : (v, w) \in E(G_f)\}$ und daher deshalb bleibt d auch eine Distanzmarkierung.

PUSH(v, w)**:** In diesem Fall kann die Kante (w, v) neu ins Inkrementnetzwerk aufgenommen werden. Da Push(v, w) ausgeführt wird, gilt aber $d(v) = d(w) + 1 \Rightarrow d(w) \leq d(v) + 1$. Für alle anderen Kanten ist nichts zu zeigen. Weiters bleibt f ein Präfluss da $\gamma \leq \mathrm{ex}_f(v)$ gilt.

□

Bemerkung 3.4.8. Wenn der Algorithmus terminiert ist f sogar ein Fluss und d eine Distanzmarkierung. Damit ist f maximal.

Wir müssen nur noch zeigen, dass der Algorithmus nach einer endlichen Anzahl von PUSH- und RELABEL-Aufrufen terminiert.

Lemma 3.4.9. *Sei f ein aktueller Präfluss während des Algorithmus und v ein aktiver Knoten. Dann existiert in G_f ein Weg von v nach s.*

Beweis. Sei $R = \{w \in V | w \text{ ist von } v \text{ in } G_f \text{ erreichbar}\}$. Zu zeigen ist also, dass s in R ist. Nach der Definition von R gilt $f(e) = 0$ für alle $e \in \delta^+(R)$. Daher ist

$$\sum_{w \in R} \mathrm{ex}_f(w) = \underbrace{\sum_{e \in \delta^+(R)} f(e)}_{=0} - \sum_{e \in \delta^-(R)} f(e) \leq 0.$$

Da $\mathrm{ex}_f(u) \geq 0$ für alle $u \in V \setminus \{s\}$, $\mathrm{ex}_f(v) > 0$ und $v \in R$, ist $s \in R$. □

Lemma 3.4.10. *Während des Algorithmus gilt $d(v) \leq 2n - 1$ für alle $v \in V$.*

Beweis. Sei v ein aktiver Knoten, dann gibt es einen Weg $P = [v, \ldots, s]$ von v nach s der Länge kleiner gleich $n - 1$, dh. $d(v) \leq d(s) + (n - 1) = 2n - 1$. □

Korollar 3.4.11. *Der Algorithmus benötigt $\mathcal{O}(n^2)$ RELABEL-Operationen.*

Beweis. $d(v)$ nimmt im Laufe des Algorithmus nur zu. Deswegen kann es höchstens $(2n - 1)n$ RELABEL-Operationen geben. □

Definition 3.4.12. Eine PUSH(v, w)-Operation heißt sättigend, falls $\gamma = u_f(v, w) \leq \mathrm{ex}_f(v)$ ist, und nicht sättigend, falls $\gamma = \mathrm{ex}_f(v) \leq u_f(v, w)$ ist.

Lemma 3.4.13. *Die Anzahl der sättigenden PUSH-Operationen beträgt $\mathcal{O}(nm)$.*

Beweis. Wir zeigen, dass auf der Kante (v, w) höchstens n sättigende PUSH Operationen stattfinden können. Bei der ersten sättigenden PUSH(v, w)-Operation gilt $d(v) = d(w) + 1$ und die Kante verschwindet bis zu einer PUSH(w, v)-Operation aus dem Inkrementnetzwerk. Bis zu dieser PUSH(w, v)-Operation muss $d(w)$ allerdings um mindestens zwei erhöht worden sein (denn jetzt ist (w, v) eine zulässige Kante und $d(v)$ wird nicht verringert). Dh. $d(v)$ muss ebenfalls um mindestens zwei steigen um wiederum PUSH(v, w) durchführen zu können. Das ergibt höchstens $\frac{2n-1}{2}$ sättigende PUSH(v, w)-Operationen und damit $\mathcal{O}(mn)$ sättigende PUSH-Operationen insgesamt. □

Lemma 3.4.14. *Der Algorithmus führt $\mathcal{O}(n^2 m)$ nicht sättigende PUSH Operationen aus.*

Beweis. Sei $A \subseteq V \setminus \{s, t\}$ die Menge der aktiven Knoten. Dann definieren wir die Potentialfunktion

$$D = \sum_{v \in A} d(v) \geq 0.$$

Am Ende des Algorithmus gibt es keinen aktiven Knoten mehr und daher gilt dann $D = 0$. Am Beginn des Algorithmus ist $D = 0$, da $d(v) = 0$ für alle aktiven Knoten gilt. Wir untersuchen nun, was mit D passiert wenn wir eine PUSH- oder RELABEL-Operation ausführen.

nicht sättigende PUSH(v, w)-Operation: Nach PUSH(v, w) verringern wir den Überschuss in v auf 0 und daher gilt danach $v \notin A$, dh. D verringert sich um $d(v)$. Eventuell wird D um $d(w)$ erhöht (nämlich dann, wenn w vorher nicht aktiv war). Da (v, w) aber zulässig war, gilt $d(v) = d(w) + 1$ und damit wird D insgesamt verringert.

sättigende PUSH(v, w)-Operation: Hier wir D höchstens um $d(w) \leq 2n - 1$ erhöht. Da es nach Lemma 3.4.13 $\mathcal{O}(nm)$ sättigende PUSH-Operationen gibt, kann der Potentialanstieg auf Grund von sättigende PUSH-Operationen nach oben mit $\mathcal{O}(n^2 m)$ abgeschätzt werden.

RELABEL(v)**:** D wird erhöht, da v aktiv ist. Da allerdings $d(v) \leq 2n-1$ ist, ist $\mathcal{O}(n^2)$ eine Schranke für den Potentialanstieg bei RELABEL-Operationen.

Damit ist über den gesamten Algorithmus der Potentialanstieg durch $\mathcal{O}(n^2m)$ gegeben. Weil jede nicht sättigende PUSH-Operation D um mindestens eins verringert, gibt es höchstens $\mathcal{O}(n^2m)$ davon. □

Schreiben wir die Wahl des aktiven Knoten im Algorithmus 3.3 vor, kann die Anzahl der nicht sättigenden PUSH-Operationen noch verringert werden:

1. HIGHEST-LABEL-PUSH-RELABEL: Wir wählen unter den aktiven Knoten jenen mit maximalem d-Wert.

2. FIFO-PUSH-RELABEL: Die aktiven Knoten werden in einer FIFO-List verwaltet.

Lemma 3.4.15. *Wendet man HIGHEST-LABEL-PUSH-RELABEL an, dann benötigt man $\mathcal{O}(n^3)$ nicht sättigende PUSH-Operationen.*

Beweis. Wählt man einen Knoten v, dann wird dieser solange gewählt, bis er inaktive wird, dh. eine nicht-sättigende Push(v, w)-Operation durchgeführt. Der Knoten v wird erst wieder aktiv, wenn für mindestens einen Knoten w der Wert $d(w)$ erhöht wurde. Falls vor der nächsten Erhöhung n nicht-sättigende Push-Operation ausgeführt, ist keine aktive Ecke mehr vorhanden und der Algorithmus terminiert. Da es $\mathcal{O}(n^2)$ RELABEL-Operationen gibt, kann es nur $\mathcal{O}(n^3)$ nicht sättigende PUSH-Operationen geben. □

Bemerkung 3.4.16. Man kann sogar zeigen, dass es bei HIGHEST-LABEL-PUSH-RELABEL nur $\mathcal{O}(n^2\sqrt{m})$ nicht sättigende PUSH-Operationen gibt. Bei FIFO-PUSH-RELABEL benötigt man ebenfalls nur $\mathcal{O}(n^3)$ nicht sättigende PUSH-Operationen.
Der Push-Relabel-Algorithmus mit HIGHEST-LABEL-PUSH-RELABEL kann tatsächlich so implemenitert werden, dass seine Laufzeit $\mathcal{O}(n^2\sqrt{m})$ beträgt. Mit geeigneten Datenstrukturen wie dynamic trees können sogar noch schnellere Laufzeit erzielt werden.

Kapitel 4

Minimale Kostenflüsse

4.1 Grundlegende Definitionen und Optimalitätskriterium

Definition 4.1.1. Sei G ein gerichteter Graph mit $u : E \to \mathbb{R}_+$ und $b : V \to \mathbb{R}$. Dann heißt $f : E \to \mathbb{R}$ b-Fluss wenn folgende Bedingungen erfüllt sind:

1. $0 \leq f(e) \leq u(e)\ \forall e \in E$
2. $-\operatorname{ex}_f(v) = b(v)\ \forall v \in V$

Min-Cost-Max-Flow-Problem	
Gegeben:	gerichteter Graph G, $u : E \to \mathbb{R}_+$, $b : V \to \mathbb{R}$, $c : E \to \mathbb{R}$
Gesucht:	b-Fluss f mit minimalen Kosten $\sum_{e \in E} c(e)f(e)$

Bemerkung 4.1.2. Die Zahl $b(v)$ heißt Balance des Knoten v. Ein Knoten v mit $b(v) > 0$ nennt man auch Quelle mit Angebot $b(v)$ und ein Knoten v mit $b(v) < 0$ heißt Senke mit Nachfrage $b(v)$.

Bemerkung 4.1.3. Die Bedingung

$$\sum_{v \in V} b(v) = 0$$

ist notwendig für die Existenz eines b-Flusses in G. Das Auffinden eines zulässigen b-Flusses ist zum maximalen Flussproblem äquivalent. Man erweitert den Graphen G um zwei Knoten s und t und definiert einen neuen Graphen $G' = (V(G) \cup \{s,t\}, E')$ mit $E' = E(G) \cup \{(s,v) : b(v) > 0\} \cup \{(v,t) : b(v) < 0\}$. Weiters betrachten wir folgende Kapazitäten

$$u'(e) = \begin{cases} u(e) & \forall e \in E(G) \\ b(v) & \forall e = (s,v) \in E' \\ -b(v) & \forall e = (v,t) \in E' \end{cases}$$

Löst man nun in (G', u', s, t) das maximales Flussproblem und erhält einen Fluss f mit $v(f) = \frac{1}{2}\sum_{v \in V} |b(v)|$, dann ist f eingeschränkt auf $E(G)$ ein b-Fluss.

Hat man in einem Graphen einen b-Fluss gegeben stellt sich die Frage, ob dieser ein kostenminimaler b-Fluss ist. Dies kann wieder mit Hilfe des Inkrementnetzwerks entscheiden werden. Dazu erweitern wir den Inkrementgraphen G_f mit Kapazitäten u_f um die Kostenfunktion n $c_f : E(G_f) \to \mathbb{R}$ mit

$$c_f(v,w) = \begin{cases} c(v,w) & \text{falls } f(v,w) < u(v,w) \\ -c(w,v) & \text{falls } f(w,v) > 0 \end{cases}.$$

Gibt es in G_f nun einen negativen Kreis bzgl. c_f, dann ist f nicht optimal.

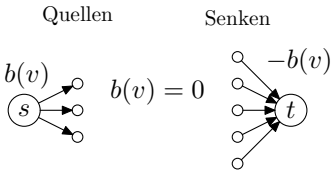

Abbildung 4.1: Berechnung einer zulässigen Lösung

Beispiel 4.1.4. In Abbildung 4.2 ist ein Netzwerk gegeben mit einem b-Fluss. Die Kantenbeschriftung gibt zuerst die Kosten, dann die Kapazität und dann den Flusswert an. Rechts sieht man das Inkrementnetzwerk mit den Kosten c_f und den Kapazitäten. Dabei gilt $c_f(e) = c(e)$ falls e eine Vorwärtskante ist und $c_f(e) = -c(e)$ falls e eine Rückwärtskante ist. Die strichlierten Kanten ergeben einen Kreis C mit negativer Länge $c_f(C) = -1$. Dann ist $\gamma = \min_{e \in C} u_f(e) = 1$ und f wird entlang von C um γ augmentiert.

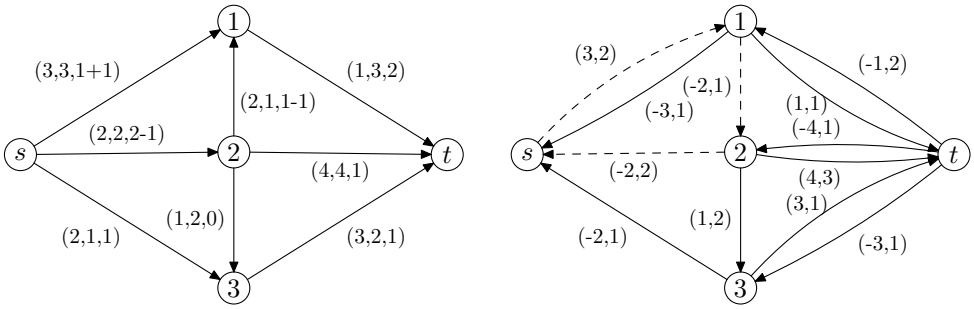

Abbildung 4.2: Netzwerk zu Beispiel 4.1.4

Wir wollen nun zeigen, dass die Existenz eines negativen Kreises in G_f ein notwendiges und hinreichendes Kriterium ist. Dazu betrachten wir eine LP-Formulierung des minimalen Kostenflussproblems:

$$
\begin{array}{lll}
\text{(LP)} & \min & \displaystyle\sum_{e \in E} c(e) f(e) \\
& \text{s.t.} & \displaystyle\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v) \qquad \forall v \in V \\
& & 0 \leq \quad f(e) \quad \leq u(e) \qquad \forall e \in E
\end{array}
$$

Bemerkung 4.1.5. Gilt $b(v) = 0 \; \forall v \in V \setminus \{v_1, v_2\}$, $b(v_1) = 1$ und $b(v_2) = -1$, dann entspricht das MCFP einem kürzesten Wegeproblem von v_1 nach v_2.

Für das duale Problem führen wir freie Variablen $\pi(v)$ für alle $v \in V$ ein. Zusätzlich benötigen wir noch die nicht-negativen Variablen $y(e) \geq 0$ für alle $e \in E$. Das duale Problem ist nun gegeben als

$$
\begin{array}{lll}
\text{(DP)} & \max & \displaystyle\sum_{v \in V} b(v)\pi(v) \quad - \quad \sum_{e \in E} u(e) y(e) \\
& \text{s.t.} & -\pi(v) + \pi(w) - y(e) \leq c(e) \qquad \forall e = (v,w) \in E \\
& & y(e) \geq 0 \qquad \forall e \in E
\end{array}
$$

Setzt man $c^\pi(v,w) = c(v,w) + \pi(v) - \pi(w)$, dann bezeichnet $c^\pi(e)$ den reduzierten Kostenkoeffizienten von e und man kann das duale Problem auf folgende Form umschreiben:

$$
\begin{array}{lll}
\text{(DP)} & \max & \displaystyle\sum_{v \in V} b(v)\pi(v) \quad - \quad \sum_{e \in E} u(e) y(e) \\
& \text{s.t.} & -c^\pi(v,w) \leq y(v,w) \qquad \forall e = (v,w) \in E \\
& & y(e) \geq 0 \qquad \forall e \in E
\end{array}
$$

Für gegebene Knotenpotentiale $\pi(v)$ erhält man den bestmöglichen Wert für $y(v,w)$ durch

$$y(v,w) = \max\{0, -c^\pi(v,w)\} \quad \forall (v,w) \in E.$$

Nun folgt mit dem Satz vom komplementären Schlupf:

1. $f(e) > 0 \Rightarrow y(e) = -c^\pi(e) \Rightarrow c^\pi(e) \leq 0$ weil $y(e) = \max\{0, -c^\pi(e)\}$ ist in einer Optimallösung.
2. $y(e) > 0 \Rightarrow f(e) = u(e)$, und $y(e) > 0 \Leftrightarrow -c^\pi(e) > 0$, dh. $c^\pi(e) < 0 \Rightarrow f(e) = u(e)$.

Damit haben wir den folgenden Satz bewiesen:

Satz 4.1. *Sei f ein b-Fluss für das dazugehörige MCFP. Dann ist f genau dann optimal, wenn es Knotenpotentiale $\pi(v)$ gibt, die die beiden folgenden Bedingungen erfüllen:*

1. $c^\pi(e) > 0 \Rightarrow f(e) = 0$
2. $c^\pi(e) < 0 \Rightarrow f(e) = u(e)$

Dieser Satz ermöglicht es uns nun ein weiteres Optimalitätskriterium zu beweisen:

Satz 4.2. *Sei f ein b-Fluss für das dazugehörige MCFP. Dann ist f genau dann optimal, wenn es in G_f keinen negativen Kreis bezüglich c_f gibt.*

Beweis. Wenn es einen negativen Kreis gibt, haben wir schon gesehen, dass wir eine bessere Lösung erhalten, wenn wir entlang dieses Kreises augmentieren. Es bleibt also nur mehr die andere Richtung zu zeigen. Angenommen es existiert kein negativer Kreis in G_f. Dann gibt es wegen Satz 2.1, ein zulässiges Knotenpotential $\pi(v)$, das heißt

$$c_f(x,y) + \pi(x) - \pi(y) \geq 0$$

gilt für alle $(x,y) \in E(G_f)$.
Für Vorwärtskanten gilt nun aber $f(x,y) < u(x,y)$ und damit auch $c_f(x,y) = c(x,y)$ woraus sofort $c(x,y) + \pi(x) \geq \pi(y)$ folgt. Analog gilt für Rückwärtskanten $c_f(x,y) = -c(y,x)$ und damit $-c(y,x) + \pi(x) \geq \pi(y)$.
Diese beiden Bedingungen sind nun äquivalent zu jenen aus Satz 4.1. □

4.2 Algorithmen für das MCFP

In diesem Abschnitt werden einige Algorithmen zur Auffindung von minimalen Kostenflüssen diskutiert. Der erste Algorithmus sucht für einen gegebenen b-Fluss negative Kreise im Inkrementnetzwerk und verbessert so sukzessive die Lösung. Im Gegensatz dazu startet der sogenannte Augmentierende-Wege-Algoritmus mit dem Fluss $f(e) = 0$ für alle $e \in E$. Dieser konstruiert dann einen optimalen b-Fluss, in dem er auf geschickte Art und Weise Fluss von Quellen zu Senken schickt bis ein b-Fluss entsteht. Beide Algorithmen sind allerdings nicht polynomial und daher werden wir am Ende noch einen anderen Algorithmus betrachten.

4.2.1 Negative-Kreise-Algorithmen

Der Negative-Kreise-Algorithmus startet mit einem b-Fluss und augmentiert schrittweise entlang negativer Kreise.

Beispiel 4.2.1. Ein Netzwerk mit Kosten, Kapazitäten und dem b-Fluss. Am Beginn ist $c(f) = 16$, dannach ist $c(f) = 14$.

Falls dieser Algorithmus terminiert liefert er wegen Satz 4.2 einen optimalen b-Fluss. Allerdings können hier für reelle Eingabedaten die gleichen Schwierigkeiten auftreten, wie beim Algorithmus von Ford-Fulkerson zur Bestimmung von maximalen Flüssen. Sind alle Eingabedaten ganzzahlig liefert der Algorithmus eine ganzzahlige Optimallösung.

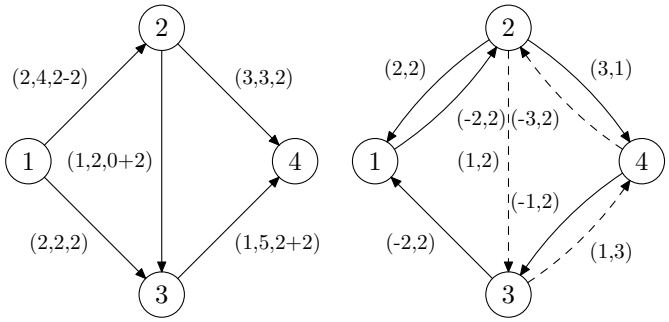

Abbildung 4.3: Negative-Kreise-Algorithmus

Korollar 4.2.2. *Ist $b(v) \in \mathbb{Z}$ für alle $v \in V$ und $u(e) \in \mathbb{N}$ für alle $e \in E$, dann existiert ein optimaler b-Fluss mit $f(e) \in \mathbb{N}_0$.*

Bemerkung 4.2.3. Die Laufzeit des Algorithmus ist nur pseudopolynomiell, dh. die Laufzeit hängt von der Größe der Eingabewerte ab. Findet man immer Kreise, die den Zielfunktionswert maximal verbessern, dann ist der Algorithmus polynomiell, das Auffinden solcher Kreise ist allerdings $\mathcal{NP}$-schwer. Einen polynomialen Algorithmus erhält man auch, wenn jedes Mal ein augmentierende Kreis mit minimalen durchschnittlichem Kosten gesucht wird. das Auffinden eines solchen Kreises ist kann in polynomialer Zeit erfolgen und man kann dadurch eine Laufzeit von $\mathcal{O}(m^3 n^2 \log n)$ erreichen.

4.2.2 Der Augmentierende-Wege-Algorihtmus

Bevor der kürzseste Wegealgorithmus beschrieben wird, benötigen wir noch folgende grundsätzliche Definition.

Definition 4.2.4. Sei (G, u, s, t) ein Netzwerk, dann heißt $f : E \to \mathbb{R}_{\geq 0}$ Pseudofluss, falls $f(e) \leq u(e)$ $\forall e \in E(G)$. Sei $b : V \to \mathbb{R}$ gegeben, dann definieren wir für einen Pseudofluss f die Balance von $v \in V$ als $e_f(v) = \mathrm{ex}_f(v) + b(v)$. Ein Knoten $v \in V$ mit $e(v) > 0$ heißt übersättigter Knoten. Ein Knoten $v \in V$ mit $e(v) < 0$ heißt unterversorgter Knoten.

Bemerkung 4.2.5. Falls f ein Pseudofluss und $e(v) = 0$ $\forall v \in V$, dann ist f ein b-Fluss.

Der nächste Satz stellt die theoretische Grundlage des Algorithmus dar.

Satz 4.3. *Sei f eine zulässige Lösung für ein MCFP mit $b : V \to \mathbb{R}$ mit minimalen Kosten. Sei P ein kürzester Weg in G_f bzgl. c_f von s nach t wobei $b(s) > 0$ und $b(t) < 0$, dann ist $f' = f \oplus P$ optimal für $b' : V \to \mathbb{R}$ mit*

$$b'(v) = \begin{cases} b(v) - \gamma & v = s \\ b(v) + \gamma & v = t \\ b(v) & sonst \end{cases}$$

mit $\gamma = \min\{b(s), -b(t), \min_{e \in P} u_f(e)\}$.

Beweis. Angenommen f' ist nicht optimal bzgl. b'. Dann gibt es einen negativen Kreis C in $G_{f'}$. Betrachte den Graphen H, der aus $((V, E(P) \cup E(C))$ durch Entfernen von Paaren entgegengesetzt orientierter Kanten entsteht. Es gilt $E(H) \subseteq E(G_f)$, da $E(P) \subseteq E(G_f)$ und falls eine Kante $e \in E(C)$ ist, aber nicht in $E(G_f)$, dann muss die entgegengesetzte Kante auf dem Pfad P liegen, diese Kanten wurden allerdings gelöscht.
Weiters gilt

$$\sum_{e \in E(H)} c_f(e) = c_f(P) + \underbrace{c_f(C)}_{<0} < c_f(P).$$

H besteht aus einem s-t-Weg und Kreisen, die nicht negative Länge bzgl. c_f haben (da $E(H) \subseteq E(G_f)$ und G_f keine negativen Kreise besitzt). Daher ist der s-t-Weg in H billiger als P, was im Widerspruch zur Wahl von P steht. □

Damit erhalten wir den Augmentierenden-Wege-Algorithmus 4.1 zur Berechnung eines minimalen Kostenflussproblems. Wiederum stellen sich bei beliebigen (reellen) Kapazitäten dieselben Probleme wie beim Algorithmus von Ford und Fulkerson. Setzt man u und b allerdings als ganzzahlig voraus benötigt der Algorithmus $B = \frac{1}{2}\sum_{v \in V} |b(v)|$ Iterationen und $\mathcal{O}(nm)$ Zeit für die Berechnung eines kürzestens Weges. Damit ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(nmB)$.

Algorithmus 4.1 Augmentierende-Wege-Algorithmus

1: Starte mit einem Pseudofluss, der garantiert, dass es in G_f keine negativen Kreise gibt (z.B. $f(e) = u(e)\ \forall e = (v,w) \in E : c(e) < 0$ und $b(v) = b(v) - u(e)$, $b(w) = b(w) + u(e)$, und $f(e) = 0$ sonst)
2: Wähle $v \in V$ und eine $w \in V$ mit $b(v) > 0$ und $b(w) < 0$
3: Suche kürzesten v-w-Weg P in G_f
4: augmentiere f entlang von P um $\gamma = \min\{b(v), -b(w), \min_{e \in P} u_f(e)\}$
5: $b(v) := b(v) - \gamma$, $b(w) := b(w) + \gamma$
6: Falls $b(v) = 0\ \forall v \in V$, dann **stop**, sonst **goto** Schritt 2

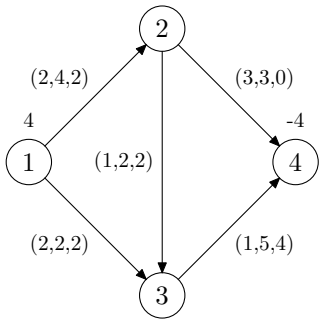

Abbildung 4.4: Augmentierende-Wege-Algorithmus

Beispiel 4.2.6. In Abbildung 4.4 ist ein Netzwerk gegeben. Wir starten mit dem Nullfluss. Der billigste Weg von 1 nach 4 ist $1-3-4$. Es wird also der Fluss um $\gamma = \min\{4, -(-4), 2, 5\} = 2$ erhöht. Danach ist der billigste Weg im Inkrementnetzwerk $1-2-3-4$ und $\gamma = 2$.

Mit Hilfe des nächsten Lemmas kann die Laufzeit des Algorithmus noch etwas verbessert werden. Bei der vorliegenden Version müssen immer wieder kürzeste Wege in Graphen mit positiven und negativen Kantenlängen berechnet werden. Man kann allerdings auch erreichen, dass dies nur in der ersten Iteration notwendig ist und danach nur mehr positive Kantenkosten auftreten. Damit kann dann eine Laufzeit von $\mathcal{O}(B(m + n \log n))$ erzielt werden.

Lemma 4.2.7. *Sei f ein Pseudofluss in einem gerichteten Graphen $G = (V, E)$ und $s \in V$, $\pi : V \to \mathbb{R}$ mit $c^\pi(v,w) \geq 0$ für alle $(v,w) \in E(G_f)$. Sei $d(v) = d_{G_f}(s,v)$ die Länge des kürzesten Weges von s nach v in G_f. Dann gilt*

1. *$c_{\tilde{\pi}}(v,w) \geq 0\ \forall (v,w) \in E(G_f)$ mit $\tilde{\pi}(v) = \pi(v) + d(s,v)$*
2. *Ist die Kante (u,v) am kürzesten Weg von s nach v, dann gilt $c_{\tilde{\pi}}(u,w) = 0$.*

Beweis. Sei $e = (v,w) \in E(G_f)$. Dann gilt $d(w) \leq d(v) + c_\pi(v,w) = d(v) + c(v,w) + \pi(v) - \pi(w)$. Daraus folgt $0 \leq c(v,w) + \pi(v) + d(v) - \pi(w) - d(w) == c(v,w) + \tilde{\pi}(v) - \tilde{\pi}(w) = c_{\tilde{\pi}}(v,w)$.
Um den zweiten Punkt zu beweisen, beobachte man, dass $d(w) = d(v) + c_\pi(v,w) = \ldots$ gilt. Der Rest wird ganz analog bewiesen. □

Lemma 4.2.8. *Sei f ein Pseudofluss und $\pi : V \to \mathbb{R}$ mit $c_\pi(v,w) \geq 0$ für alle $(v,w) \in E(G_f)$. Sei $s \in V$ mit $b(s) > 0$ und P ein kürzester s-t-Weg in G_f bzgl. c_π. Dann erfüllt $f' = f \oplus P$ folgende Aussage:*

$$c_{\tilde{\pi}}(v,w) \geq 0 \quad (v,w) \in E(G_{f'}).$$

Beweis. Nach obigen Lemma gilt $c_{\tilde{\pi}}(v,w) \geq 0$ für alle $e = (v,w) \in E(G_f)$. Sei $(v,w) \in E(G_{f'}) \setminus E(G_f)$ und damit $(w,v) \in E(P)$, dann gilt wegen der zweiten Aussage im Lemma 4.2.7 $c_{\tilde{\pi}}(v,w) = 0$. □

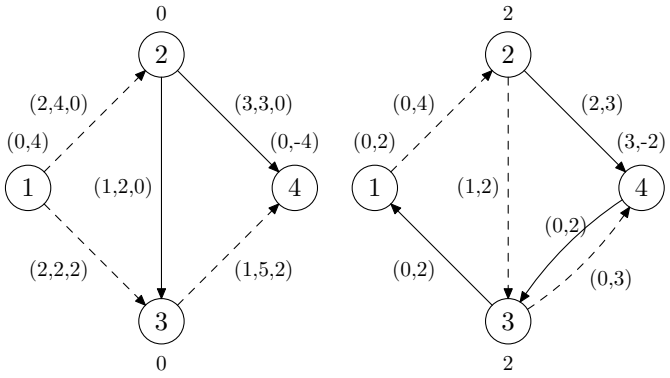

Abbildung 4.5: Augmentierende-Wege-Algorithmus mit Beachtung negativer Kosten: die kürzesten Wegearboreszenzen sind strichliert dargestellt.

4.2.3 Ein Kapazitätsskalierungsalgorithmus

In diesem Kapitel wird der erste polynomielle Algorithmus für das minimale Kostenflussproblem diskutiert. Dabei nehmen wir an, dass alle Eingabedaten natürliche Zahlen sind und dass es im Netzwerk einen Pfad mit unbeschränkter Kapazität zwischen allen Knotenpaaren gibt. Die zweite Bedinugng kann durch zusätzliche Kanten mit hohen Kosten garantiert werden.

Der hier vorgestellte Algorithmus ist eine Modifikation des augmentierenden Wegealgorihtmus. Beim augmentierenden Wegealgorihtmus kann es vorkommen, dass viele Augmentierungsschritte mit sehr wenig Fluss nötig sind. Der Kapazitätsskalierungsalgorithmus bessert genau diese Schwachpunkt aus und schickt in jeder Iteration "viel" Fluss von einem übersättigten zu einem unterversorgten Knoten. Sei $\Delta \geq 1$ ein Skalierungsparameter, der garantieren soll, dass in jeder Iteration mind. Δ Einheiten verschickt werden. Für einen Pseudofluss f definieren wir die beiden Mengen

$$S_f(\Delta) = \{v \in V | e_f(v) \geq \Delta\}$$

und

$$T_f(\Delta) = \{v \in V | e_f(v) \leq -\Delta\}$$

der viel übersättigten bzw. unterversorgten Knoten. Weiters sei $G_f(\Delta)$ sei das Inkrementnetzwerk G_f ohne Kanten e für die $u_f(e) < \Delta$ gilt. $G_f(\Delta)$ enthält also nur Kanten, die viel Fluss transportieen können, da sie genügend Kapazität haben. Es gilt also $u_f(e) \geq \Delta$ für alle Kanten in $G_f(\Delta)$.

Die Struktur des Algorithmus ist einfach: Für ein gegebenes Δ suchen wir einen Knoten $s \in S_f(\Delta)$ und einen Knoten $t \in T_f(\Delta)$ und augmentieren um genau Δ (selbst wenn wir um mehr als um Δ augmentieren können!) entlang eines augmentierenden Weges in $G_f(\Delta)$. Dies wird so lange fortgesetzt bis entweder $S_f(\Delta) = \emptyset$ oder $T_f(\Delta) = \emptyset$. Dann setzen wir $\Delta = \frac{\Delta}{2}$ und beginnen wieder von vorne. Dabei werden die Kanten aus $G_f(\Delta)$ mit jenen Kanten ergänzt, für die $\frac{\Delta}{2} \leq u_f < \Delta$ gilt. Sollten diese Kanten negative reduzierte Kosten haben, wird entlang dieser Kanten augmentiert. Danach werden wiederum augmentierende Wege zwischen Quellen und Senken gesucht und um

genau $\frac{\Delta}{2}$ augmentiert. Am Ende gilt $G_f(1) = G_f$ und da alle Eingabedaten ganzzahlig waren, erhalten wir einen b-Fluss mit minimalen Kosten.

Satz 4.4. *Der Algorithmus 4.2 terminiert nach $\mathcal{O}((n+m)\log U)$ Flusserhöhungen.*

Beweis. Die äußere While-Schleife wird $\mathcal{O}(\log U)$ mal durchlaufen. Zu zeigen bleibt, dass in jeder Phase $\mathcal{O}(n+m)$ Flusserhöhungen stattfinden. Am Ende einer 2Δ-Phase gilt entweder $e(v) < 2\Delta$ für alle $v \in V$ oder $e(v) > -2\Delta$ für alle $v \in V$. Dann gilt

$$D := \sum_{e(v)>0} e(v) = -\sum_{e(v)<0} e(v) \leq 2\Delta n.$$

Im Schritt 6 kann auf jeder Kante höchstens 2Δ geschickt werden und damit kann D höchstens um $2\Delta m$ wachsen. Vor dem ersten Durchlauf der While-Schleife in der Δ-Phase gilt

$$D \leq 2\Delta n + 2\Delta m = 2\Delta(n+m).$$

In jeder Iteration sinkt D um Δ, daher gibt es $\mathcal{O}(n+m)$ Iterationen. □

Korollar 4.2.9. *Der Algorithmus kann in $\mathcal{O}((n+m)\log U(m+n\log n))$ Zeit ausgeführt werden.*

Algorithmus 4.2 Kapazitätsskalierungsalgorithmus

1: $f(e) = 0\ \forall e \in E$, $\pi(v) = 0\ \forall v \in V$
2: $U = \max\{\max_{v\in V} |b(v)|, \max_{e\in E} u(e)\}$
   $\Delta = 2^{\lfloor \log_2 U \rfloor}$
3: **while** $\Delta \geq 1$ **do**
4: **for all** $e \in E(G_f(\Delta))$ **do**
5: **if** $u_f(e) > 0$ und $c_f(e) < 0$ **then**
6: augmentiere den Pseudofluss f entlang e um $u_f(e)$
7: **end if**
8: **end for**
9: Berechne $S_f(\Delta)$ und $T_f(\Delta)$
10: **while** $S_f(\Delta) \neq \emptyset$ und $T_f(\Delta) \neq \emptyset$ **do**
11: Wähle $s \in S_f(\Delta)$ und $t \in T_f(\Delta)$
12: Sei P ein kürzester Weg in $G_f(\Delta)$ bzgl. c_π
13: Setze $\pi := \pi + d$
14: Augmentiere f entlang von P um Δ.
15: Berechne f, $S_f(\Delta)$, $T_f(\Delta)$, $G_f(\Delta)$
16: **end while**
17: $\Delta = \frac{\Delta}{2}$
18: **end while**

Kapitel 5

Matchings

5.1 Problemstellung un Optimalitätskriterium

Definition 5.1.1. Sei $G = (V, E)$ ein Graph, dann heißt $M \subseteq E$ Matching, falls jeder Knoten $v \in V$ mit höchstens einer Kante aus M inzident ist. Ein Matching M heißt perfekt, falls $|M| = \frac{|V|}{2}$. Ein Matching M ist maximal falls $|M| \geq |M'|$ für alle Matchings M' in G. Sei $v \in V$ ein Knoten, der mit einer Kante aus M inzident ist, dann heißt v gematcht.

Maximales-Matching-Problem	
Gegeben:	$G = (V, E)$
Gesucht:	$M \subseteq E$ Matching mit $\|M\|$ maximal

Definition 5.1.2. Sei $G = (V, E)$ ein Graph und $M \subseteq E$ ein Matching. Dann heißt ein Weg $P = [v_1, \ldots, v_k]$ alternierend, falls die Kanten in P abwechselnd in M und nicht in M sind. P heißt augmentierend, falls P alternierend ist und v_1 und v_k nicht gematcht sind.

Bemerkung 5.1.3. Sei $G = (V, E)$ ein Graph, M ein Matching und P ein augmentiereder Weg. Dann ist $M' = M \triangle E(P)$ ein Matching mit $|M'| = |M| + 1$.

Satz 5.1. *Sei M ein Matching in $G = (V, E)$. Dann ist M genau dann maximal, wenn es in G keinen M-augmentierenden Weg gibt.*

Beweis. M ist maximal $\Rightarrow \nexists$ augmentierender Weg (siehe Bemerkung 5.1.3).

Angenommen, es gibt keinen M-augmentierenden Weg und es gibt ein Matching M' mit $|M'| > |M|$. Dann betrachte $M \triangle M'$. Jeder Knoten $v \in V$ ist mit höchstens zwei Kanten aus $M \triangle M'$ inzident. Deshalb kann $M \triangle M'$ in gerade Kreise und Pfade partitioniert werden. Alle Wege sind in M und M' alternierend, und einer dieser Wege ist M-augmentierend. Widerspruch. □

Grundidee: Wir starten mit einem Matching (z.B. $M = \emptyset$) und suchen einen M-augmentierenden Weg.

Laufzeit: $\mathcal{O}(n \cdot A(m, n))$, wobei $A(m, n)$ die Laufzeit für das Finden eines augmentierenden Weges ist.

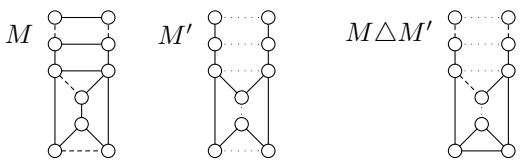

Abbildung 5.1: Zwei Matchings mit der symmetrischen Differenz

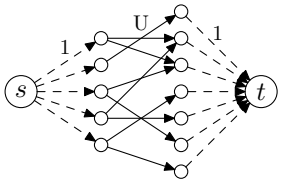

Abbildung 5.2: Ein Matchingproblem mit dem dazugehörigen Flussproblem

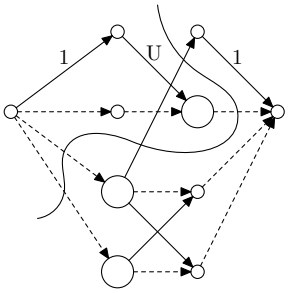

Abbildung 5.3: Ein Matching mit einem maximalen Fluss und einem minimalen Schnitt. Die großen Knoten bilden eine minimale Knotenüberdeckung.

5.2 Das Matchingproblem auf bipartiten Graphen

Wir können das Problem auf eine Flussproblem transformieren, wobei $U > |E(G)|$ ist.

Jeder ganzzahlige Fluss f im erweiterten Netzwerk in Abbildung 5.2 entspricht einem Matching M und umgekehrt. Es gilt $v(f) = |M|$. Wir können also ein maximales Flussproblem lösen und erhalten ein maximales Matching. Das Konzept des augmentierenden Weges ist hier äquivalent.

Bemerkung 5.2.1. Mit dem Algorithmus für blockierende Flüsse in Schichtgraphen kann das maximale Flussproblem und damit das maximale Matchingproblem in bipartiten Graphen in $\mathcal{O}(\sqrt{n}m)$ Zeit gelöst werden.

Falls U groß ist, ist keine ursprüngliche Kante in einem minimalen Schnitt.

$$\begin{aligned} \text{ganzzahliger Fluss} &\equiv \text{Matching} \\ \text{max. Matching} &\equiv \text{max. Fluss} \\ \text{max. Fluss} &\equiv \text{min. Schnitt} \\ \text{min. Schnitt} &\equiv \text{min. Knotenüberdeckung} \end{aligned}$$

Definition 5.2.2. Sei $G = (V, E)$ ein ungerichteter Graph, dann heißt $C \subseteq V$ Knotenüberdeckung (Cover), wenn jede Kante mindindestens einen Endknoten in C hat.

Lemma 5.2.3. *Sei $G = (V, E)$ ein ungerichteter Graph, dann gilt*

$$\min_{C \text{ Cover}} |C| \geq \max_{M \text{ Matching}} |M|.$$

Beweis. Jede Matchingkante muss von einem Knoten überdeckt werden. □

Satz 5.2 (von König)**.** *Sei $G = (V_1 \cup V_2, E)$ ein bipartiter Graph, dann gilt* $\max |M| = \min |C|$.

Beweis. Wir betrachten das Matchingproblem als max. Flussproblem (siehe oben). Sei (S, T) ein minimaler Schnitt. Dann gibt es keine Kanten von $S \cap V_1$ nach $T \cap V_2 \Rightarrow C = (T \cap V_1) \cup (S \cap V_2)$ ist ein Cover mit $|C| = |T \cap V_1| + |S \cap V_2|$. Dies ist genau die Kapazität des Schnittes. □

Satz 5.3 (Hall)**.** *Sei $G = (V_1 \cup V_2, E)$ ein bipartiter Graph. Dann hat G ein Matching M mit $|M| = |V_1|$ genau dann, wenn*

$$|X| \leq |N(X)| \quad \forall X \subseteq V_1.$$

Beweis. Die eine Richtung ist klar.

Nun die andere: Angenommen es gilt $\max |M| < |V_1| \overset{\text{König}}{\Rightarrow} \min |C| < |V_1|$. Sei $C = V_1' \cup V_2'$ mit $V_1' \subseteq V_1$, $V_2' \subseteq V_2$ und $|V_1'| + |V_2'| < |V_1|$. Es gilt $N(V_1 \setminus V_1') \subseteq V_2' \Rightarrow |N(V_1 \setminus V_1')| \leq |V_2'| < |V_1| - |V_1'|$. Widerspruch. □

Anwendungsbeispiel

Gegeben:	n Jobs $J_1, \ldots, J_n$ m Maschinen $M_1, \ldots, M_m$ jeder Job i besteht aus m Operationen, dh. O_{ij} muss auf Maschine j genau p_{ij} Zeiteinheiten bearbeitet werde.
Gesucht:	Maschinenbelegungsplan: Zu jedem Zeitpunkt darf auf einer Maschine nur ein Job bearbeitet werden. Kein Job darf parallel auf 2 Maschinen bearbeitet werden. Die Operationen dürfen unterbrochen werden. Ziel ist es, die Bearbeitungsdauer zu minimieren.

Beispiel 5.2.4.

$$J_1 = O_{11}, O_{12}, O_{13}$$
$$J_2 = O_{21}, O_{22}, O_{23}$$
$$p_{11} = 4, \quad p_{12} = 2, \quad p_{13} = 5$$
$$p_{21} = 3, \quad p_{22} = 6, \quad p_{23} = 4$$

$$T_j := \sum_{i=1}^{n} p_{ij} \qquad \text{Laufzeit der Maschine } j$$

$$L_i := \sum_{j=1}^{m} p_{ij} \qquad \text{Produktionszeit von Job } i$$

$$T := \max\{\max_{j=1,\ldots,m} T_j, \max_{i=1,\ldots n} L_i\} \leq MS$$

$$T = 13 \Rightarrow \text{Optimallösung}$$

Zeit	$1-4$	$5-6$	$7-8$	$9-11$	$12-13$
M_3		J_1	J_2	J_1	J_2
M_2	J_2	J_2			J_1
M_1	J_1			J_2	

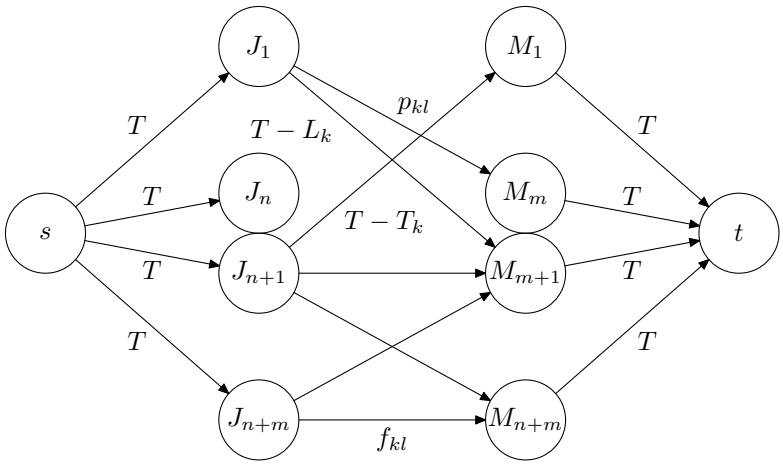

Abbildung 5.4: Das allgemeine Netzwerk (G, u, s, t).

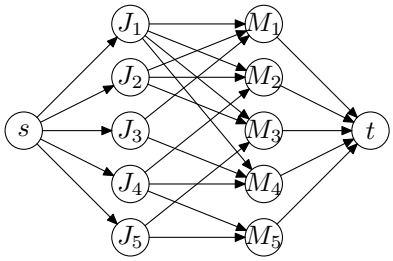

Abbildung 5.5: Das Netzwerk (G, u, s, t) zu Beispiel 5.2.5.

Wir konstruieren ein Netzwerk (G, u, s, t):

$$
\begin{aligned}
V &= \{J_1, \ldots, J_{n+m}, M_1, \ldots, M_{n+m}\} \\
E &= E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5 \cup E_6 \\
\text{mit } E_1 &= \{(s, J_k) : k = 1, \ldots, n+m\} \\
E_2 &= \{(M_k, t) : k = 1, \ldots, n+m\} \\
E_3 &= \{(J_k, M_l) : p_{kl} > 0, k = 1, \ldots, n, l = 1, \ldots, m\} \\
E_4 &= \{(J_{n+k}, M_k) : k = 1, \ldots, m, T_k < T\} \\
E_5 &= \{(J_k, M_{m+k}) : k = 1, \ldots, n, L_k < T\} \\
E_6 &= \{(J_{n+k}, M_{m+l}) : k = 1, \ldots, m, l = 1, \ldots, n\} \\
u(e) &= \begin{cases} T & \forall e \in E_1 \cup E_2 \\ p_{kl} & \forall e \in E_3 \\ T - T_k & \forall e = (J_{n+k}, M_k) \in E_4 \\ T - L_k & \forall e = (J_k, M_{m+k}) \in E_5 \\ f_{kl} & \forall e = (J_{n+k}, M_{m+l}) \in E_6 \end{cases}
\end{aligned}
$$

Mit f_{kl} so, dass $\sum_{e \in \delta^+(J_{n+k})} u(e) = T = \sum_{e \in \delta^-(M_{n+k})} u(e)$ gilt.

Beispiel 5.2.5 (Fortsetzung von Beispiel 5.2.4). Der rechte untere Teil wird z.B. mit Hilfe der Nordwestregel aufgefüllt. Die hervorgehobenen Kanten bilden ein Matching:

	M_1	M_2	M_3	M_4	M_5
J_1	**4**	2	5	2	0
J_2	3	**6**	4	0	0
J_3	6	0	0	**7**	0
J_4	0	5	0	4	**4**
J_5	0	0	**4**	0	9

Betrachten wir alle Kanten $e \in E_3 \cup \ldots \cup E_6$ mit $u(e) > 0$. Sei $X \subseteq M = \{M_1 \ldots, M_{n+m}\}$. Dann gilt

$$|X| \cdot T = \sum_{M_j \in X} \sum_{J_i \in N(M_j)} u(J_i, M_j) \leq |N(X)| \cdot T \quad \forall X \subseteq M.$$

Daher existiert ein perfektes Matching.

Beispiel 5.2.6 (Fortsetzung von Beispiel 5.2.4). perfektes Matching ≡ Belegung für $\min_{e \in \text{ Matching}} u(e)$ Zeiteinheiten

neue Matrix:

	M_1	M_2	M_3	M_4	M_5
J_1	0	2	**5**	2	0
J_2	3	**2**	4	0	0
J_3	**6**	0	0	3	0
J_4	0	5	0	**4**	0
J_5	0	0	0	0	**9**

Definition 5.2.7. Sei $\mathcal{M} = \{M_1, \ldots, M_n\}$ eine Familie von Teilmengen einer endlichen Menge M. Eine Multimenge $\{e_1, \ldots, e_n\} \subseteq M$ mit $e_i \in M_i$ heißt Repräsentantensystem. Gilt außerdem $e_i \neq e_j \ \forall i \neq j$, dann heißt die Multimenge Transversale von $\mathcal{M}$.

Beispiel 5.2.8.

$$\begin{aligned} M &= \{1, \ldots, 10\} \\ M_1 &= \{1, 3, 5, 7, 9\} \\ M_2 &= \{2, 4, 6, 8, 10\} \\ M_3 &= \{3, 6, 9\} \\ M_4 &= \{4, 8\} \\ M_5 &= \{5, 10\} \\ M_6 &= \{10\} \\ \{1, 2, 3, 4, 5, 10\}: &\quad \text{RS und Transversale} \\ \{3, 4, 3, 4, 10, 10\}: &\quad \text{RS} \end{aligned}$$

Satz 5.4. *Eine Familie $\mathcal{M} = \{M_1, \ldots, M_n\}$ hat genau dann eine Transversale wenn für alle $J \subset \{1, \ldots, n\}$ gilt*

$$|\bigcup_{j \in J} M_j| \geq |J|.$$

Beweis. Die eine Richtung ist klar. Nun die andere Richtung: Wir konstruieren einen bipartiten Graphen $G = (A \cup B, E)$ mit $A = \{e_1, \ldots, e_k\} = M$, $B = \{M_1, \ldots, M_n\} = \mathcal{M}$ und $e = (e_j, M_r) \Leftrightarrow e_j \in M_r$. Zu zeigen ist, dass ein Matching $E' \subseteq E$ existiert mit $|E'| = n \Rightarrow$ zu zeigen: jede Knotenüberdeckung $C \subseteq (A \cup B)$ erfüllt $|C| \geq n$.

Sei C eine Knotenüberdeckung und $J = \{j : M_j \notin C\}$. Es gilt $|C| = |C \cap M| + |C \cap \mathcal{M}| = |C \cap M| + n - |J| \geq n$. Zu zeigen: $|C \cap M| \geq |J|$. Sei $U = \bigcup_{j \in J} M_j$. Dann gilt $|U| \geq |J|$. $C \cap M$ enthält mindestens $|U|$ Elemente. □

5.3 Maximale Matchings in allgemeinen Graphen

Definition 5.3.1. Sei $G = (V, E)$ ein Graph. Eine Teilmenge $U \subseteq V$ heißt ungerade Komponente falls

- $|U|$ ungerade ist und
- U eine Zusammenhangskomponente ist.

Wir bezeichnen mit $\mathrm{oc}(G)$ die Anzahl der ungeraden Komponenten in G.

Lemma 5.3.2. *Sei $G = (V, E)$ ein Graph und $S \subseteq V$. Dann gilt*

$$\max |M| \leq \frac{1}{2} \left(|V| - \mathrm{oc}(G - S) + |S| \right).$$

Beweis. Sei U eine ungerade Komponente von $G - S$. Um alle Knoten in U zu überdecken muss es mind. eine Matchingkante zwischen U und S geben. Daher können höchstens $|V| + |S| - \mathrm{oc}(G - S)$ Knoten überdeckt werden. Da jede Matchingkante genau zwei Knoten überdeckt, gilt $\max |M| \leq \frac{1}{2}(|V| - \mathrm{oc}(G - S) + |S|)$. □

Bemerkung 5.3.3. Es gilt $\max |M| \leq \min_{S \subseteq V} \frac{1}{2}(|V| - \mathrm{oc}(G - S) + |S|)$. Wir werden sogar zeigen, dass die Ungleichung in Wahrheit eine Gleichung ist (Tutte-Berge-Formel). Falls es ein $S \subseteq V$ gibt, mit $|S| < \mathrm{oc}(G - S)$, dann hat G kein perfektes Matching.

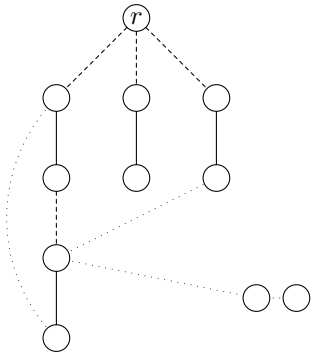

Abbildung 5.6: Ein verkümmerter Baum: durchgezogene Kanten sind im Matching, die strichlierten im Baum, und die punktierten außerhalb des Baumes.

Definition 5.3.4. Sei $G = (V, E)$ ein Graph und M ein Matching in G. Sei $r \in V$, der nicht gematcht ist. Ein alternierender Baum T mit Wurzel r ist ein Baum in G, der folgende Bedingungen erfüllt:

1. Für jeden Knoten in T ist der Weg zur Wurzel r in T ein alternierender Weg.
2. Jeder Knoten $v \in V(T) \setminus \{r\}$ wird von einer Kante in $M \cap E(T)$ überdeckt.

Wir bezeichnen mit $\mathrm{even}(T)$ und $\mathrm{odd}(T)$ die Mengen der Knoten in T, die geraden bzw. ungeraden Abstand zur Wurzel r in T haben.

Bemerkung 5.3.5. Es gilt $|\mathrm{even}(T)| = |\mathrm{odd}(T)| + 1$.

Sei T ein alternierender Baum, $u \in \mathrm{even}(T)$, $(u, v) \in E(G)$ mit $v \notin V(T)$. Dann gibt es zwei Fälle:

1. v ist nicht gematcht. In diesem Fall ist der Weg von r nach u in T und die Kante (u, v) ein augmentierender Weg. Wir können also augmentieren und die Kardinalität des Matchings um eins vergrößern.

2. v ist gematcht und $(v, w) \in M$ (ist eindeutig bestimmt und $(v, w) \notin E(T)$). In diesem Fall können wir die Kanten (u, v) und (v, w) zu T hinzufügen und wir erhalten wieder einen alternierenden Baum.

Wir betrachten nun einen alternierenden Baum T, $u \in \text{even}(T)$, $v \in V(T)$. Dann gibt es wieder zwei Fälle:

3. Jede Kante (u, v) mit $u \in \text{even}(T)$ erfüllt $v \in \text{odd}(T)$.

4. Es gibt eine Kante (u, v) mit $u \in \text{even}(T)$ und $v \in \text{even}(T)$.

Definition 5.3.6. Ein alternierender Baum T heißt verkümmert, wenn für jede Kante $e \in E(G)$, von der ein Endknoten in $\text{even}(T)$ liegt, der andere Endknoten von e in $\text{odd}(T)$ liegt.

Lemma 5.3.7. *Ist T ein verkümmerter Baum in G, dann hat G kein perfektes Matching.*

Beweis. Setze $S = \text{odd}(T)$, dann gilt $1 + |S| = |\,\text{odd}(T)| + 1 = |\,\text{even}(T)| \leq \text{oc}(G - S) \Rightarrow |S| < \text{oc}(G - S)$. □

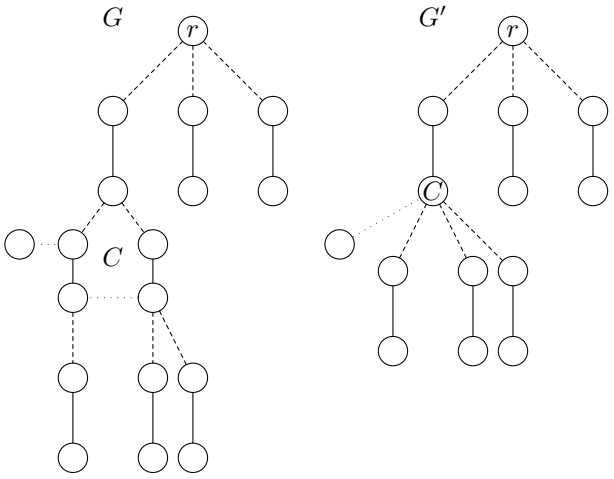

Abbildung 5.7: Ein alternierender Baum mit einem ungeraden Kreis, der kontrahiert wird.

Bemerkung 5.3.8. Der Fall 4 kann für bipartite Graphen nicht auftreten, da die Kante $(u, v) \in E(G)$ mit $u, v \in \text{even}(T)$ einen ungeraden Kreis induziert. In bipartiten Graphen können wir nun algorithmisch erkennen, ob es ein perfektes Matching gibt.

Idee zu Fall 4: Durch das Hinzufügen der Kante (u, v) entsteht ein Kreis C, der zu einem Superknoten kontrahiert wird. Wir schreiben dann $G' = G/C$.

- Wie können wir einen augmentierenden Weg in G' nutzen, um das Matching in G zu vergrößern?
- Welche Information für G liefert ein verkümmerter Baum in G'?

Lemma 5.3.9. *Sei C ein ungerader Kreis in G, $G' = G/C$ und M' ein Matching in G'. Dann existiert ein Matching M in G mit $M \subseteq M' \cup E(C)$, sodass die Anzahl der ungematchten Knoten bzgl. M in G gleich groß ist wie die Anzahl der ungematchten Knoten bzgl. M' in G'.*

Beweis. Wir setzen $M = M' \cup E'$ wobei $E' \subseteq E(C)$ und $|E'| = \frac{|E(C)|-1}{2}$ gilt, für ein geeignetes E'. □

Bemerkung 5.3.10. Entstehen im Algorithmus Superknoten C', die aus mehreren Superknoten bestehen, dann enthält C' immer eine ungerade Anzahl an Knoten des ursprünglichen Graphen.

Lemma 5.3.11. *Sei $G' = G/C$, M' ein Matching in G', T' ein alternierender Baum in G', sodass in* $\text{odd}(T')$ *kein Superknoten ist. Falls T' verkümmert ist, dann hat G kein perfektes Matching.*

Beweis. Setzen wir $S = \text{odd}(T')$, dann bilden die Knoten und Superknoten $\text{even}(T')$ ungerade Komponenten in G. Daher ist $|S| < \text{oc}(G - S)$ und es gibt kein perfektes Matching. □

Bemerkung 5.3.12. Kontrahiert man in einem alternierenden Baum T einen ungeraden Kreis C, dann erhält man wiederrum einen alternierenden Baum T' und der Superknoten $v(C)$ ist in $\text{even}(T')$.

Algorithmus 5.1 Maximales Matching in allgemeinen Graphen

```
1: G' = G, M' = M
2: if |M'| = 1/2|V| then
3:    Stop
4: else
5:    Wähle einen nicht gematchten Knoten r ∈ V und setze T = ({r}, ∅)
6: end if
7: if ∃(u, v) ∈ E(G) mit u ∈ even(T) und v ∉ odd(T) then
8:    if v ist nicht gematcht (Fall 1) then
9:       Augmentiere M' und setze M' zu einem Matching M in G fort.
10:      G' = G, M' = M
11:      goto 2
12:   end if
13:   if v ist von (v, w) ∈ M' gematcht (Fall 2) then
14:      T = T ∪ {(u, v), (v, w)}
15:      goto 7
16:   end if
17:   if v ∈ even(T) (Fall 4) then
18:      Kontrahiere den Kreis C
19:      goto 7
20:   end if
21: else
22:   Es gibt kein perfektes Matching (Fall 3).
23: end if
```

Bemerkung 5.3.13. Im Algorithmus kommt es zu $\mathcal{O}(n)$ Augmentationsschritten sowie $\mathcal{O}(n^2)$ Kontraktionen und Wachstumsschritten. Jede Augmentation verringert die Anzahl der nicht gematchten Knoten. Zwischen zwei Augmentationsschritten kann es zu $\mathcal{O}(n)$ Kontraktionen und Wachstumsschritten kommen. Die Gesamtlaufzeit ist $\mathcal{O}(n^3)$.

Frage: Wie findet man ein maximales Matching, wenn G kein perfektes Matching besitzt?

Finden wir einen verkümmerten Baum T_1 in G, entfernen wir alle Knoten $V(T_1)$ aus G und suchen erneut ein perfektes Matching M'. Dieser Vorgang wird solange wiederholt, bis ein perfektes Matching gefunden wird, oder der leere Graph entsteht.

Wir erweitern das Matching in T_i durch Dekontraktion der Kreise und erhalten ein Matching M_i.

$$M = \bigcup_{i=1}^{k} M_i \cup M'$$

Dann gilt $|M| = \frac{n-k}{2}$.

$$S = \bigcup_{i=1}^{k} \text{odd}(T_i) \Rightarrow \text{oc}(G - S) = \sum_{i=1}^{k} |\text{even}(T_i)| = \sum_{i=1}^{k} |\text{odd}(T_i)| + 1 = |S| + k$$

Es gilt

$$\max|M| \leq \frac{1}{2}(|V| - \mathrm{oc}(G-S) + |S|) \quad \forall S \subseteq V.$$

Einsetzen ergibt

$$\max|M| \leq \frac{1}{2}(|V| - |S| + k + |S|) = \frac{n-k}{2}.$$

Satz 5.5. *Tutte-Berge-Formel:*

$$\max_{M \ Matching} |M| = \min_{S \subseteq V} \frac{1}{2}(|V| - \mathrm{oc}(G-S) + |S|)$$

5.4 Das Matchingpolytop

Für ein Matching M in einem Graphen $G = (V, E)$ schreiben wir den Vektor $x_M \in \{0,1\}^{|E|}$ wobei $x_M(e) = 1 \Leftrightarrow e \in M$. Wir wollen das folgende Polytop

$$\mathrm{conv}(\mathrm{PM}(G)) = \mathrm{conv}\{x_M \in \{0,1\}^{|E|} : M \text{ ist ein perfektes Matching in } G\}$$

untersuchen.

Satz 5.6. *(von Birkhoff) Sei $G = (V, E)$ ein bipartiter Graph. Dann gilt*

$$P := \{x \in [0,1]^{|E|} : \sum_{e \in \delta(v)} x(e) = 1 \quad \forall v \in V\} = \mathrm{conv}(PM(G)).$$

Beweis. Es gilt $\mathrm{conv}(\mathrm{PM}(G)) \subseteq P$.

Sei x eine Ecke von P und x ist nicht ganzzahlig.

$$\widetilde{E} = \{e \in E | 0 < x(e) < 1\}$$

Jeder Knoten, der mit einer Kante $e \in \widetilde{E}$ inzident ist, muss mit mind. einer weiteren Kanten aus $\widetilde{E}$ inzident sein da $\sum_{e \in \delta(v)} x(e) = 1$.

Daher enthält die Kantenmenge $\widetilde{E}$ einen Kreis C. C ist gerade da G bipartit ist. Wir definieren uns zwei Lösungen, wobei wir zu den Kanten in C alternierend ε dazuzählen bzw. abziehen. x ist nun die Summe der beiden Lösungen durch 2. Widerspruch zur Eigenschaft einer Ecke. □

Bemerkung 5.4.1. Für allgemeine Graphen benötigen wir noch weitere Bedingungen. zB.: $G = C_3$

Hier gilt $\mathrm{PM}(G) = \emptyset \Rightarrow \mathrm{conv}(\mathrm{PM}(G)) = \emptyset$, aber $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}) \in P \Rightarrow \mathrm{conv}(\mathrm{PM}(G)) \subset P$.

Satz 5.7. *Sei $G = (V, E)$ ein Graph und $x \in P$. Dann ist x genau dann eine Ecke von P, wenn $x(e) \in \{0, \frac{1}{2}, 1\}$ und die Kanten, für die $x(e) = \frac{1}{2}$ gilt, knotendisjunkte ungerade Kreise sind.*

Beweis. Sei $\bar{x} \in P$ mit den Eigenschaften des Satzes. Dann definieren wir $w(e) = -1$ falls $\bar{x}(e) = 0$ und $w(e) = 0$ falls $\bar{x}(e) > 0$. Betrachte $F = \{x \in P : w^t x = 0\}$.

Es gilt $\bar{x} \in F$. Zu zeigen bleibt, dass $F = \{\bar{x}\}$ damit $\bar{x}$ eine Ecke ist. Für jedes $x \in F$ gilt

$$0 = w^t x = -\sum_{e \in E : \bar{x}(e) = 0} \underbrace{x(e)}_{\geq 0}.$$

Daher gilt $x(e) = 0$ falls $\bar{x}(e) = 0$.

Wir betrachten die Kanten für die $\bar{x}(e) = \frac{1}{2}$. Die Werte von x auf den Kreisen müssen abwechselnd ε und $1 - \varepsilon$ sein. Da der Kreis ungerade Länge hat gilt $\varepsilon = \frac{1}{2}$ und daher $x = \bar{x}$ und $\bar{x}$ ist eine Ecke.

Angenommen $\bar{x}$ ist eine Ecke von P. Wir zeigen zuerst $x(e) = \{0, \frac{1}{2}, 1\}$ $\forall e \in E$.

Es gibt ein c sodass $\bar{x}$ eindeutige Optimallösung von $\max\{c^t x | x \in P\}$ ist. Wir konstruieren aus G einen bipartiten Graphen $H = (V', E')$ durch Einführung der Knoten v' und v'' (statt v) und der Kanten (u', v'') und (u'', v') statt (u, v). Wir setzen $c(u, v) = c'(u'', v') = c'(u', v'')$. Nun betrachten wir $\max\{c'^t x' | x' \in P(H)\}$.

Sei $x \in P$, dann ist $x'(u', v'') = x'(u'', v') = x(u, v)$ ein Vektor in $P(H)$. Es gilt $2c^t x = c'^t x'$.

Sei $x' \in P(H)$, dann ist $x(u, v) = \frac{1}{2}(x'(u', v'') + x'(u'', v'))$ ein Vektor in P mit halben Zielfunktionswert.

Die Optimallösung von $\max\{c'^t x' | x' \in P(H)\}$ ist ganzzahlig, dh., dass $\bar{x}(e) \in \{0, \frac{1}{2}, 1\}$. Im Beweis der Tatsache, dass die Kanten für $\bar{x}(e) = \frac{1}{2}$ gilt, knotendijunkte Kreise ungerader Länge bilden können die gleichen Argumente wie im Satz von Birkhoff verwendet werden. □

Satz 5.8. *(Edmonds) Es gilt*

$$\text{conv}(PM(G)) = \{x \in \mathbb{R}^{|E|} | x(e) \geq 0 \quad \forall e \in E, \quad \sum_{e \in \delta(v)} x(e) = 1 \quad \forall v \in V,$$
$$\sum_{e \in \delta(U)} x(e) \geq 1 \quad \forall U \subseteq V \text{ mit } |U| \equiv 1 \mod 2\} := Q.$$

Beweis. Die eine Richtung ist klar.

Angenommen es gilt $\text{conv}(\text{PM}(G)) \subsetneq Q$, dann wählen wir den Graphen $G = (V, E)$ mit $|V|+|E|$ minimal (dh. G ist zusammenhängend und $|V|$ ist gerade).

Sei x eine Ecke in Q. Dann gilt $x(e) \in (0, 1)$ $\forall e \in E$ (falls $x(e) = 0$ kann die Kante e gelöscht werden und falls $x(u, v) = 1$ können die Knoten u und v gelöscht werden). Dh. der kleinste Grad in G ist größer gleich 2.

Falls $|E| = |V|$, dann ist G ein Kreis und die Aussage des Satzes stimmt.

Falls $|E| > |V|$ ist, gilt:

x ist eine Ecke von Q. Dann erfüllt x $|E|$ linear unabhängige Nebenbedingungen von Q mit Gleichheit. Da $x(e) > 0$ $\forall e \in E$ und $|E| > |V|$, gibt es ein $U \subseteq V$ mit $|U|$ ungerade und $\sum_{e \in \delta(U)} x(e) = 1$ und $3 \leq |U| \leq |V| - 3$. Wir kontrahieren die Menge U zu einem Knoten und erhalten einen Graphen G^U. Wir kontrahieren $\bar{U} = V \setminus U$ und erhalten $G^{\bar{U}}$.

Seien x^1 und x^2 die Projektion von x auf die Kanten von G^U bzw. $G^{\bar{U}}$. x^1 und x^2 erfüllen alle Ungleichungen von Q. Für die kleineren Graphen ist die Aussage des Satzes erfüllt da $|E| + |V|$ minimal gewählt worden ist. Daher kann x^1 und x^2 als Konvexkombination von perfekten Matchings geschrieben werden:

$$x^1 = \frac{1}{k} \sum_{j=1}^{k} x_{M_j^1} \text{ und } x^2 = \frac{1}{k} \sum_{j=1}^{k} x_{M_j^2}$$

wobei $M_1^i, \ldots, M_k^i$ perfekte Matchings in G^u bzw. $G^{\bar{U}}$ sind. Für jede Kante $e \in \delta(U)$ gilt $x^1(e) = x(e) = x^2(e)$. Daher ist die Anzahl der Indizes $j \in \{1, \ldots, k\}$ mit $e \in M_j^1$ $k \cdot x(e)$. Analoges gilt für die Anzahl der Indizes $j \in \{1, \ldots, k\}$ mit $e \in M_j^2$. Daher ist $M_j := M_j^1 \cup M_j^2$ ein perfektes Matching in G und $x = \frac{1}{k} \sum_{j=1}^{k} x_{M_j}$. Widerspruch. □

Kapitel 6

Matroide

6.1 Grundlegende Definitionen und einfache Eigenschaften

Definition 6.1.1. Sei E eine endliche Menge und $\mathcal{F} \subseteq \mathcal{P}(E)$ eine Familie von Mengen. Dann heißt $(E, \mathcal{F})$ Unabhängigkeitssystem, wenn

(M1) $\emptyset \in \mathcal{F}$

(M2) $X \subseteq Y \in \mathcal{F} \Rightarrow X \in \mathcal{F}$

gilt. Die Mengen in $\mathcal{F}$ heißen unabhängig, die restlichen Mengen $\mathcal{P}(E) \setminus \mathcal{F}$ heißen abhängig. Inklusionsminimale abhängige Mengen heißen Kreise, inklusionsmaximale unabhängige Mengen nennt man Basen. Für eine gegebene Menge $X \subseteq E$ heißen die inklusionsmaximalen unabhängigen Teilmengen von X die Basen von X.

Definition 6.1.2. Sei $(E, \mathcal{F})$ ein Unabhängigkeitssystem, dann definieren wir für $X \subseteq E$ den Rang von X durch $r(X) := \max\{|Y| : Y \subseteq X, Y \in \mathcal{F}\}$. Ferner definieren wir den Abschluss von X als $\sigma(X) := \{Y \in E : r(X \cup \{y\}) = r(X)\}$.

Definition 6.1.3. Ein Unabhängigkeitssystem $(E, \mathcal{F})$ heißt Matroid, wenn gilt:

(M3) Sind $X, Y \in \mathcal{F}$ mit $|X| > |Y|$ dann gibt es ein $x \in X \setminus Y$ mit $Y \cup \{x\} \in \mathcal{F}$.

Beispiel 6.1.4.

1. Stable-Set-Problem: $G = (V', E')$, $E = V'$, $\mathcal{F} = \{F \subseteq E : F \text{ ist eine stabile Menge in } G\}$. M1: $\emptyset$ ist stabil. M2: $X \subseteq Y \in \mathcal{F} \Rightarrow X \in \mathcal{F}$. M3 gilt nicht.
2. Traveling-Sales-Problem: $\mathcal{F} = \{F \subseteq E : F \text{ ist eine Teilmenge eines Hamiltonschen Kreis in } G\}$. M1, M2 gelten, aber M3 gilt nicht.
3. MST-Problem: $E = E(G)$, $\mathcal{F} = \{\text{Wälder}\}$. Sowohl M1, M2 und M3 gelten.
4. Matchings: Gegeben $G = (V(G), E(G))$, $E = E(G)$, $\mathcal{F} = \{M \subseteq E : M \text{ Matching}\}$. M1, M2 gelten, M3 aber nicht.
5. Rucksackproblem: $N = \{1, \ldots, n\}$, $w : N \to \mathbb{R}_+$, $C > 0$. $E = N$, $\mathcal{F} = \{A \subseteq N : \sum_{a \in E} w(a) \leq C\}$. M1, M2 gelten, M3 nicht.

Satz 6.1. *Die folgenden Unabhängigkeitssysteme $(E, \mathcal{F})$ sind Matroide:*

1. *E ist die Teilmenge der Zeilenmenge einer Matrix A über einem Körper und $\mathcal{F} = \{F \subseteq E :$ F ist linear unabhängig im Sinne der linearen Algebra$\}$.*
2. *E ist die Menge der Kanten eines ungerichteten Graphen und $\mathcal{F} = \{F \subseteq E :$ $(V(G), F)$ ist ein Wald$\}$ (graphisches Matroid).*

3. *$E = \{1, 2, 3 \ldots, n\}$ und $\mathcal{F} = \{F \subseteq E : |F| \leq k\}$ für ein $k \in \mathbb{N}$.*

4. *E ist eine Teilmenge der Kantenmenge eines ungerichteten Graphen G, S ist eine stabile Menge in G, $b : S \to \mathbb{N}$, $\mathcal{F} = \{F \subseteq E : d_F(s) \leq b(s) \quad \text{für alle } s \in S\}$.*

5. *E ist eine Teilmenge der Kantenmenge eines gerichteten Graphen G, $S \subseteq V(G)$, $b : S \to \mathbb{N}$, $\mathcal{F} = \{F \subseteq E : d_F^-(s) \leq b(s)\}$.*

Beweis. M1 und M2 sind klar.
Nun zeigen wir M3 für:

1. Basisergänzungssatz

2. Sei $X, Y \in \mathcal{F}$. Angenommen $Y \cup \{x\} \notin \mathcal{F}$ für alle $x \in X \setminus Y$. Dann ist zu zeigen: $|Y| \geq |X|$. Für jede Kante $x = (u, v)$ liegen u und v in der gleichen Zusammenhangskomponente bzgl. Y. Sei $Z \subseteq V(G)$ eine Zusammenhangskomponente bzgl. X, dann ist diese eine Teilmenge einer Zusammenhangskomponente bzgl. Y. Die Anzahl p der Zusammenhangskomponenten bzgl. X ist nicht kleiner als die Anzahl q der Zusammenhangskomponenten bzgl. Y:

$$|V| - |X| = p \geq q = |V| - |Y| \Rightarrow |Y| \geq |X|.$$

3. trivial

4. Sei $X, Y \in \mathcal{F}$. Angenommen $|X| > |Y|$. Dann ist $S' = \{s \in S : d_Y(s) = b(s)\} \subsetneq S$. Daher $\exists e \in X \setminus Y$ mit $e \notin \delta(s)$ und $s \in S'$. Daher ist $Y \cup \{e\} \in \mathcal{F}$.

5. analog

□

Satz 6.2. *Sei $(E, \mathcal{F})$ ein Unabhängigkeitssystem, dann sind folgende Aussagen äquivalent:*

(M3) *$X, Y \in \mathcal{F}$ mit $|X| > |Y|$ dann $\exists x \in X \setminus Y : Y \cup \{x\} \in \mathcal{F}$*

(M3') *$X, Y \in \mathcal{F}$ mit $|X| = |Y| + 1$ dann $\exists x \in X \setminus Y : Y \cup \{x\} \in \mathcal{F}$*

(M3") *Für jede Menge $X \subseteq E$ gilt, alle Basen von X haben die gleiche Kardinalität.*

Beweis. (M3) $\Leftrightarrow$ (M3') und (M3) $\Rightarrow$ (M3") sind klar. Nun zeigen wir (M3") $\Rightarrow$ (M3): Sei $X, Y \in \mathcal{F}$ mit $|X| > |Y|$. Dann ist Y keine Basis von $X \cup Y$ wegen (M3").Daher gibt es ein $x \in (X \cup Y) \setminus Y = X \setminus Y$ mit $Y \cup \{x\} \in \mathcal{F}$. □

Definition 6.1.5. Sei $(E, \mathcal{F})$ ein Unabhängigkeitssystem, dann definieren wir für $X \subseteq E$ den unteren Rang von X als

$$\rho(X) = \min\{|Y| : Y \subseteq X, Y \in \mathcal{F}, Y \cup \{x\} \notin \mathcal{F} \quad \forall x \in X \setminus Y\}.$$

Der Rangquotient von $(E, \mathcal{F})$ ist definiert durch

$$q(E, \mathcal{F}) = \min_{X \subseteq E} \frac{\rho(X)}{r(X)}.$$

Bemerkung 6.1.6. Es gilt $q(E, \mathcal{F}) \leq 1$ da $\rho(X) \leq r(X)$ für alle $X \subseteq E$ gilt.
Aus der Definitione folgt, dass $\rho(X) = r(X)$ für alle $X \subseteq E$ genau dnn wenn $(E, \mathcal{F})$ ist ein Matroid ist. Weiters ist diese bedingung äquiivalent zur Tatsache, dass $q(E, \mathcal{F}) = 1$.
Der Rangquotient gibt an "wie weit ein Unabhängigkeitssystem von einem Matroid entfernt ist."

Im restlichen Teil dieses Kapitels betrachten folgendes Problem:

Gegeben:	Unabhängigkeitssystem $(E, \mathcal{F})$ und $c : E \to \mathbb{R}$
Gesucht:	Bestimme $X \in \mathcal{F}$ mit $c(X) = \sum_{e \in X} c(e)$ maximal

Algorithmus 6.1 Greedy-Algorithmus

```
1: Sortiere c(e_1) ≥ c(e_2) ≥ ... ≥ c(e_n)
2: F = ∅
3: for i = 1 to n do
4:    if F ∪ {e_i} ∈ F then
5:       F := F ∪ {e_i}
6:    end if
7: end for
```

Dieses Problem kann mit dem sogenannten Greedyalgorithmus 6.1 gelöst werden. Wir stellen uns nun die Frage, unter welchen Bedingungen der angegebene Algorithmus das Problem optimal löst bzw. wie gut die Lösung ist, falls der Algorithmus nicht die Optimallösung findet. Die Antwort auf diese Frage gibt der folgende Satz.

Satz 6.3. *Sei $(E, \mathcal{F})$ ein Unabhängigkeitssystem und $c : E \to \mathbb{R}$. Dann gilt*

$$q(E, \mathcal{F}) \leq \frac{G(E, \mathcal{F}, c)}{OPT(E, \mathcal{F}, c)} \leq 1, \tag{6.1}$$

wobei $OPT(E, \mathcal{F}, c)$ bzw. $G(E, \mathcal{F}, c)$ der Zielfunktionswert der Optimallösung bzw. des Greedyalgorithmus ist.

Ferner gibt es für jedes Unabhängigkeitssystem eine Kostenfunktion $c : E \to \mathbb{R}$ mit

$$q(E, \mathcal{F}) = \frac{G(E, \mathcal{F}, c)}{OPT(E, \mathcal{F}, c)}.$$

Beweis. Sei $E = \{e_1, \ldots, e_n\}$ mit $c(e_1) \geq \ldots \geq c(e_n) > 0$, $G_n \subseteq E = \{e_1, \ldots, e_n\}$ die Greedylösung und $O_n \subseteq E$ die Optimallösung. weiters definieren wir $E_j = \{e_1, \ldots, e_j\}$ für $j \leq n$, $O_j = O_n \cap E_j$ und $G_j = G_n \cap E_j$. Wir setzen noch Setze $d_n = c(e_n)$ und $d_j = c(e_j) - c(e_{j+1})$ für $j = 1, \ldots, n$.

Es gilt $O_j \in \mathcal{F}$ und $|O_j| \leq r(E_j)$. Weiters ist G_j ist eine Basis von E_j, daher folgt $|G_j| \geq \rho(E_j)$. Damit erhält man

$$\begin{aligned}
G(E, \mathcal{F}, c) &= \sum_{j=1}^{n} \left(|G_j| - |G_{j-1}|\right) c(e_j) = \\
&= \sum_{j=1}^{n} \left(|G_j| - |G_{j-1}|\right) \left(d_j + c(e_{j+1})\right) = \\
&= \sum_{j=1}^{n} |G_j| d_j + \underbrace{\sum_{j=1}^{n} |G_j| c(e_{j+1}) - \sum_{j=1}^{n} |G_{j-1}| \underbrace{\left(d_j + c(e_{j+1})\right)}_{=c(e_j)}}_{=0} \geq \\
&\geq \sum_{j=1}^{n} \rho(E_j) d_j \geq \\
&\geq q(E, \mathcal{F}) \sum_{j=1}^{n} r(E_j) d_j \geq \\
&\geq q(E, \mathcal{F}) \sum_{j=1}^{n} |O_j| d_j = \\
&= q(E, \mathcal{F}) \sum_{j=1}^{n} \left(|O_j| - |O_{j+1}|\right) c(e_j) = \\
&= q(E, \mathcal{F}) \mathrm{OPT}(E, \mathcal{F}, c)
\end{aligned}$$

und die Ungleichung (6.1) ist gezeigt.
Für den Beweis des zweiten Teiles des Satzes wähle $F \subseteq E$ und Basen B_1 und B_2 von F mit $q(E, \mathcal{F}) = \frac{|B_1|}{|B_2|}$. Definiere

$$c(e) = \begin{cases} 1 & \text{für } e \in F \\ 0 & \text{sonst} \end{cases}.$$

Sortiert man nun so, dass $c(e_1) \geq \ldots \geq c(e_n)$ und $B_1 = e_1, ldots, e_{|B_1|}$, dann gilt $G(E, \mathcal{F}, c) = |B_1|$ und $\mathrm{OPT}(E, \mathcal{F}, c) = |B_2|$. Damit wird die untere Schranke angenommen. □

Damit bekommen wir nun unmittelbar das folgende Resultat.

Korollar 6.1.7. *Der Greedyalgorithmus 6.1 liefert genau dann eine Optimallösung wenn $(E, \mathcal{F})$ ein Matroid ist.*

Satz 6.4. *(Edmonds) Sei $(E, \mathcal{F})$ ein Matroid und $r : \mathcal{P}(E) \to \mathbb{N}$ seine Rangfunktion. Dann ist das Matroidpolytop von $(E, \mathcal{F})$, dh. die konvexe Hülle der Inzidenzvektoren aller Elemente in $\mathcal{F}$ gegeben durch*

$$\left\{ x \in \mathbb{R}^{|E|} : x \geq 0, \sum_{e \in A} x(e) \leq r(A) \quad \textit{für alle} \quad A \subseteq E \right\}.$$

Beweis. Das angegebene Polytop enthält offensichtlich alle Inzidenzvektoren unabhängiger Mengen. Damit genügt es zu zeigen, dass alle Ecken des Polytops ganzzahlig sind. Dh. wir müssen zeigen, dass

$$\max\{c^t x : x \geq 0, \sum_{e \in A} x(e) \leq r(A) \quad \forall A \subseteq E\} \tag{6.2}$$

für jede beliebige Zielfunktion eine ganzzahlige Lösung besitzt. Außerdem können wir annehmen, dass $c(e) \geq 0$ für alle $e \in E$ gilt, da sonst $x(e) = 0$ gilt.
Sei x^* eine Optimallösung von 6.2. Wie vorhin bezeichne $G_n \subseteq E$ die Greedylösung und $O_n \subseteq E$ die Optimallösung des Problems. Weiters setzen wir wieder $E_j = \{e_1, \ldots, e_j\}$ für $j \leq n$, $O_j = O_n \cap E_j$ und $G_j = G_n \cap E_j$. Damit gilt dann

$$G(E, \mathcal{F}, c) = \ldots \geq \underbrace{q(E, \mathcal{F})}_{=1} \sum_{j=1}^{n} (|O_j| - |O_{j-1}|)\, c(e_j) =$$
$$= \sum_{j=1}^{n} x^*(e_j) c(e_j) = c^t x^*$$

Die Greedylösung liefert damit eine weitere ganzzahlige Optimallösung. □

Bemerkung 6.1.8. Falls $(E, \mathcal{F})$ das graphische Matroid ist, folgt aus obigem Satz der Satz 1.5 über das Spannbaumpolytop.

6.2 Schnitte von Matroiden

Gegeben:	$M_i = (E, \mathcal{F})$ sind Matroide $(i = 1, \ldots, n)$, $c : E \to \mathbb{R}$
Gesucht:	$F \subseteq E$ mit $F \in \bigcap_{i=1}^{n} \mathcal{F}_i$ mit $c(F) \to \max$

Beispiel 6.2.1. 1. Matching in bipartiten Graphen $G = (A \cup B, E)$. Dabei ist $n = 2$ mit $M_1 = (E(G), \mathcal{F}_1)$ mit $\mathcal{F}_1 = \{F \subseteq E : d_F(v) \leq 1 \quad \forall v \in A\}$, und $M_2 = (E(G), \mathcal{F}_2)$ mit $\mathcal{F}_2 = \{F \subseteq E : d_F(v) \leq 1 \quad \forall v \in B\}$. $\mathcal{F}_1 \cap \mathcal{F}_2 = \{F \subseteq E : F \text{ ist eine Matching}\}$, dabei ist $(E, \mathcal{F}_1 \cap \mathcal{F}_2)$ kein Matroid.

2. Branching in gerichteten Graphen $G = (V, E)$. Dabei ist $n = 2$ mit $M_1 = (E(G), \mathcal{F}_1)$ mit $\mathcal{F}_1 = \{F \subseteq E : F \text{ ist kreisfrei}\}$, und $M_2 = (E(G), \mathcal{F}_2)$ mit $\mathcal{F}_2 = \{F \subseteq E : d_F^-(v) \leq 1 \quad \forall v \in V\}$. $\mathcal{F}_1 \cap \mathcal{F}_2 = \{F \subseteq E : F \text{ ist Branching}\}$.

3. Traveling-Salesman-Problem. Dabei ist $n = 3$. Gegeben ist ein gerichteter Graph $G = (V, E)$ mit $|V| = k$. Entferne alle Kanten $(i, 1)$ und füge stattdessen $(i, k + 1)$ hinzu. Es gibt einen Hamiltonschen Kreis in G genau dann, wenn es einen Hamiltonschen Pfad von 1 nach $k + 1$ in $G + (k + 1)$ gibt. $E = E(G + (k + 1))$, $M_1 = (E, \mathcal{F}_1)$ mit $\mathcal{F}_1 = \{F \subseteq E : d_F^-(v) \leq 1 \quad \forall v \in V\}$, $M_2 = (E, \mathcal{F}_2)$ mit $\mathcal{F}_2 = \{F \subseteq E : d_F^+(v) \leq 1 \quad \forall v \in V\}$ und $M_3(E, \mathcal{F}_3)$ mit $\mathcal{F}_3 = \{F \subseteq E : F \text{ ist kreisfrei}\}$.

Bemerkung 6.2.2. Das Problem ist für $n = 1$ mit dem Greedyalgorithmus lösbar. Für $n = 2$ kann man eine Verallgemeinerung des Matchingalgorithmus für bipartiten Graphen verwenden, In beiden Fällen gibt es polynomielle Algorithmen. Für $n = 3$ ist das Problem jedoch $\mathcal{NP}$-vollständig.

Index