Zusammenfassung der Einführung in die diskrete Mathematik

Dozent
Professor Dr. Jens Vygen

Mitschrift
Maximilian Kessler

Zusammenfassung

Dies ist eine grobe Zusammenfassung aller behandelten Inhalte der Vorlesung 'Einführung in die diskeret Mathematik'. Beweise und Details sind (bewusst) nicht Teil dieser Zusammenfassung, sie dient eher einem Überblick, um z.B. für eine Klausur zu lernen und sich die wesentlichen Resultate, Algorithmen und Beweisideen ins Gedächtnis zu rufen.
Jeder Abschnitt entspricht dem Inhalt einer Vorlesung.
Paralleles Nachschlagen in der Fachliteratur (die Vorlesung folgt dem Standardwerk [**Korte**; **Vygen**] oder den Vorlesungsnotizen ist unabdingbar für ein solides Verständnis der Inhalte.

Inhaltsverzeichnis

1 Euler

- (starker) Zusammenhang von Graphen
- prüfen durch 2 x bfs (mit invertierten kanten) -> lineare Zeit

- Eulerkreisproblem:

Bemerkung. Ein Graph ist eulersch, wenn er Eulerbedingung erfüllt, muss also nicht zusammenhängend sein.

Satz 1.1. Graph hat Eulerkreis gdw $\forall v \in V(G)\colon \left|\delta^+(v)\right| = \left|\delta^-(v)\right|$

Beweis. Offensichtllich notwendig. Wenn Bedingung erfüllt, wähle längsten Weg. □

Algorithmus 1 : Eulers Algorithmus

Eingabe : zshg. Eulerscher Graph G
Ausgabe : Eulerweg in G
Laufzeit : $\mathcal{O}(m)$

2 SCC

- Hamiltonkreise
- Graphendurchmusterung: Tiefen und Breitensuche

- topologische Ordnung
- k-Zusammenhang (auch: k-Knotenzusammenhang) von Graphen (Knoten entfernen lässt Zusammenhang)
- k-facher Kantenzusammenhang (Kanten entfernen lässt Zusammenhang)

Satz 2.1 (Dirac, 1952)**.** Jeder einfache ungerichtite graph mit $n \geqslant 3$ Knoten und minimalem Grad $\geqslant \frac{n}{2}$ hat einen Hamiltonkreis

Beweis. Graph ist zusammenhängend. Längster Weg muss schon alle Knoten enthalten, weil wir überall 'weg' kommen. □

Algorithmus 2 : Strongly connected component algorithm

Eingabe : geritchtere Graph G
Ausgabe : Funktion comp : $V(G) \to \mathbb{N}$, die starke zshg. komponenten angibt
Laufzeit : $\mathcal{O}(m + n)$

Beweis. Machen tiefensuche, beim 'zürückgehene' vergeben wir eine nummerierung. □

Korollar 2.2. Ein Graph ist genau dann azyklisch, wenn er eine topologische Ordnung besitzt. Obiger Algorithmus berechnet eine solche Ordnung.

3 MST

- Ohrenzerlegungen
- Minimum Spanning Tree
- Maximum Weight Forest
- Minimale Spannbäume

Satz 3.1 (Whitney, 1932)**.** Ein ungerichteter Graph ist genau dann 2-zusammenhängend, wenn er eine **echte** Ohrenzerlegung besitzt.

Satz 3.2. MST und MWF sind linear aufeinander reduzierbar, d.h. äquivalent.

Satz 3.3 (Sylvester 1857, Cayley 1889)**.** Für $n \in \mathbb{N}$ gibt es auf n Knoten genau n^{n-2} aufspannende Bäume.

Beweis. Zähle

$$\mathfrak{B}_{n,k} = \{(B, f) : B \text{ Branching auf } n \text{ Knoten}, |E(B)| = k, f : E(B) \to \{1, \ldots, k\} \text{ Bijektion}\}.$$

d.h. die Anzahl der **nummerierten** Branchings mit k Kanten auf n Knoten. □

Satz 3.4. Viele äquivalenten Bedingungen, wann ein Baum ein MST ist:

a) T ist optimal
b) Jede Kante $e \in E(G) \backslash E(T)$ ist immer teureste Kante auf Fundamentalkreis
c) Jede Kante $e \in E(T)$ ist billigste Kante im Fundamentalschnitt
d) Nummerierungsbedingung: Wir können $E(T) = \{e_1, \ldots, e_{n-1}\}$ so ordnen, dass für jedes $i \in \{1, \ldots, n-1\}$ eien Menge $X \subseteq\subseteq V(G)$ existiert, so dass e_i enie billigste Kante in $\delta_G(X)$ ist und $e_j \notin \delta_G(X)$ für $j = 1, \ldots, i-1$
ähnlich zu c: nimme Fundamentalschnitt

4 MST2

- Kruskal
- Prim
- Heaps

Algorithmus 3 : Kruskals Algorithmus

Eingabe : ein zshgh. ungerichteter Graph G, Kosten $c : E(G) \to R$
Ausgabe : optimaler Spannbaum
Laufzeit : $\mathcal{O}(m \log n)$

Beweis. füge nacheinander die billigste Kante hinzu, die keinen Wald erzeugt, dann ist Bedingung (b) von obigem Satz trivial erfüllt. □

Algorithmus 4 : Prims Algorithmus

Eingabe : zshg. Graph G, Kosten $c : E(G) \to \mathbb{R}$
Ausgabe : minimaler Spannbaum
Laufzeit : $\mathcal{O}(m + n \log n)$

Beweis. Baue den Baum nacheinander (aus einer zshgkomponente) auf, indem man die distanzen angrenzender Knoten in heap speichert, und immer billigsten knoten hinzufügt. Verwende Fibonacci-Heaps. □

5 F-Heaps

- Fibonacci-Heaps
- Maximum weight branching
- minimum weight arborescence
- minimum weight rooted arborescence

Satz 5.1 (Fib-Heaps)**.** Ein Heap mit

- p Insert
- n deletmin
- m decreasekey

Operationen kann implementiert werden in $\mathcal{O}(m+p+n \log p)$ Laufzeit (amortisiert)

Beweis. Speichere Elemente in einem Branching, das die Heap-Ordnung erfüllt, und 'Knoten haben viele Kinder' -> 'Kinder haben viele Kinder',

damit logarithmische tiefe des Branchings erhalten bleibt. Manche Knoten können markiert werden, wenn sie 'zu wenig Kinder' haben. Genauer:

(i) Heapordnung

(ii) Jeder Knoten u kann Kinder mit $1, \dots, \left|\delta^+(u)\right|$ nummerieren, dass für das i-te Kind
$$\left|\delta^+(v)\right| + \varphi(v) \geqslant i.$$

(iii) Verschiedene Wurzeln haben verschieden Anzahl Kindern. Dadurch gibt es 'wenig' (logarithmisch viele) Wurzeln.

Beim Einfügen wollen wir diese Bedingungen immer erhalten, wenn die Heap-Ordnung verletzte wird, entfernen wir das entsprechende Kind und markieren es. Wenn es markiert ist, müssen wir auch den Vater entfernen (iterieren). Beim Einfügen müssen evtl. zwei Wurzeln (entsprechend der Ordnung) zusammengebastelt werden.
Laufzeitanalyse ergibt sich aus einfachen abschätzungen, wie viele Kanten wir höchstens entfernt haben, um plant-laufzeit zu beschränken. □

Proposition 5.2. MWB, MWA, MWRA sind zueinander äquivalent.

6 Branching-Algorithmus

- MWB-algo
- kürzeste Wege

wollen nun MWB lösen:

Lemma 6.1 (Karp, 1972)**.** ein maximaler gewichteter Teilgraph von G mit Eingangsgrad $\leqslant 1$ für jeden Knoten kann in ein maximales gewichtetes Branching B in G umgewandelt werden, indem wir bei jedem Kreis genau eine Kante entfernen.

Beweis. Wähle optimales Branching, sodass maximal viele Kanten von B_0 enthält.
Warnung. wichtig: beweis anschauen □

Algorithmus 5 : Edmond's Branching-Algorithmus

Eingabe : ungerichtete Graph G mit Kosten $c : E(G) \to \mathbb{R}^+$
Ausgabe : maximales Branching
Laufzeit : $\mathcal{O}(nm)$, kann mit F-Heaps auf $\mathcal{O}(m + n \log n)$ verbessert werden

Beweis. Suche jeweils maximale gerichtet Teilgraphen, kontrahiere evtl. vorhandene Kreise und gehe wieder zurück. □

Proposition 6.2 (Kürzeste Wege auf Graphen mit konservativen Gewichten)**.** Sei G gerichteter Graph und $c : E(G) \to \mathbb{R}$ konservativ. Sei $k \in \mathbb{N}$ und seien $s, w \in V(G)$ mit $s \neq w$. Sei P ein kürzester $s-w$-Weg mit höchstens k Kanten. Ist $e = (v, w)$ die letzte Kante von P, so ist $P_{[s,v]}$ ein kürzester $s-v$-Weg mit höchstens $k-1$ Kanten.

7 Kürzeste Wege

- Dijkstras Algorithmus
- Moore-Bellman-Ford Algorithums
- Potenziale

Algorithmus 6 : Dijkstras algorithmus

Eingabe : gerichteter Graph g, nichtnegative Kosten c, Startknoten s
Ausgabe : kürzeste Wege von s zu alle anderen Knoten t
Laufzeit : $\mathcal{O}(m + n \log n)$

Beweis. Speichere distanzen von anderen knoten zu s in heap. Füge jeweils den knoten mit kürzester distanz hinzu, und update entlang aller adjazenten Kanten. Entferne den Knoten aus dem Heap. Mit Fibonacci-Heaps ergibt sich die Laufzeit, da wir jede Kante auch $\leqslant 2$ mal besuchen. □

Algorithmus 7 : Moore-Bellman-Ford-Algo

Eingabe : Graph g, Kosten $c : E(G) \to R$, Startknoten s
Ausgabe : Distanzen zu allen anderen Knoten oder ein Kreis mit negativem Gewicht
Laufzeit : $\mathcal{O}(nm)$

Beweis. Gehe $n - 1$ mal über alle Kanten und update sie evtl. Damit erhalten wir auch alle kürzesten Wege. Für die (evtl auftretenden) negativen Kreise prüfen wir am Ende noch einmal, ob es eine weiter Kante gibt, die wir updaten könnten - falls ja, so tracken wir alle ihre n Vorgänger und finden unter all diesen Kanten einen Kreis. Dieser muss negativ gewesen sein. □

Satz 7.1. Ein gerichteter Graph G mit Kosten $c : E(G) \to \mathbb{R}$ hat genaud ann ein zulässiges Potenzial, wenn c konservativ ist. Wir können in $\mathcal{O}(nm)$ ein solches Potenzial bestimmen oder einen negativen Kreis finden.

8 Min-Mean-Cycle

- allpairs shortest path: wollen kurzeste wege zwischen allen Knoten aus dem Graphen finden
- Metrischer Abschluss
- minimum mean cycle - problem
- shallow-light-tree

Satz 8.1. Das kann man in $\mathcal{O}(mn + n^2 \log n)$ Zeit lösen.

Beweis. Berechne dazu einmal ein Potenzial und wende danach nur noch Dijkstra an (für jeden Startknoten). Das brauch dann $mn + n(m + \log n)$ Zeit. □

Korollar 8.2. Der metrische Abschluss eines Graphen kann in $\mathcal{O}(mn + n^2 \log n)$ gefunden werden.

Setup: Graph G und Kosten c, wollen Kreis mit minimalen durchschnittskosten finden Setze

$$F_k(x) := \min \left\{ \sum_{i=1}^{k} c(e_i) \mid e_i = (v_{i-1}, v_i) \in E(G) \forall i, v_0 = s, v_k = x \right\}.$$

Satz 8.3 (Karp 1978)**.** Seien G, c und sei $s \in V(G)$ so dass alle Knoten von s aus erreichbar sind. Sei

$$\mu(G, c) := \min \left\{ \frac{c(E(C))}{E(C)} \mid C \text{ Kreis in } G \right\}.$$

(das suchen wir). Dann ist

$$\mu(G, c) = \min_{x \in V(G)} \max_{\substack{0 \leqslant k \leqslant n-1 \\ F_k(x) < \infty}} \frac{F_n(x) - F_k(x)}{n - k}.$$

Beweis. Wir betrachten zunächst den Spezialfall $\mu(G, c) = 0$. Hier ist die rechte Seite offenbar $\geqslant 0$, wir wollen nur noch Gleichheit zeigen. Finde hierzu einen negativen Kreis, wähle Knoten w auf ihm, betrachte kürzesten $s - w$-Weg und nun laufe von w aus so lange im Kreis bis der Weg n lang ist und lande so bei x. Für disese x ist der kürzeste $s - x$-Weg gleich teuer wie der kürzeste $s - x$-Weg mit n Kanten und für dieses x gilt dann Gleichheit.
Den allgemeinen Fall erhalten wir, indem wir eine Konstante auf alle Kanten addieren, das ändert nichts am Ablauf des Algorithmus. □

Algorithmus 8 : Minimum mean cycle algorithmus

Eingabe : Graph mit Kosten
Ausgabe : Kreis mit minimalen Kosten oder die Meldung, dass der Graph azyklisch ist
Laufzeit : $\mathcal{O}(mn)$

Beweis. Im wesentlichen obigen beweis ausführen. Den Kreis finden wir, indem wir von unserem errechneten x den n-langen Weg zu s zurückgehen und unter ihm einen beliebigen Kreis wählen. □

Satz 8.4. Für G, c und beliebiges $\varepsilon > 0$ gibt es enien shallow-light-tree, der $(1 + \varepsilon)$ mal größere Distanzen als die minimalen hat, und auch höchstens $(1 + \frac{2}{\varepsilon})$ mal so teuer ist.

Beweis. Bestimme minimalen Spannbaum, und laufe ein Euelertour ab. Wenn wir einen Knoten erreichen, der zu weit weg ist (mehr als $1+\varepsilon$ der minimalen Distanz), so füge einen kürzesten Weg von s zu diesem Knoten ein. Damit erhalten wir wieder 'mehr' Kanten. Dann bestimme wieder einen kürzeste-Wege-Baum, dieser ist nur billiger und hat auch keine langeren distanzen, also reicht es, die kosten nach dem hinzufügen der kürzesten wege zu beschränken. Hierzu muss man ein paar Ungleichungen aufaddieren. □

9 Max-Flow-Min-Cut

- Algorithmus zu shallow-light-tree: Details (überprüfen der schwächeren Bedingung)
- max-flow-min-cut
- Ford-Fulkerson

Algorithmus 9 : Shallow-Light-Tree

Eingabe : Graph mit Kosten
Ausgabe : ein Shallow-Light-Tree
Laufzeit : $\mathcal{O}()$

Satz 9.1. Max-Flow-Min-Cut: der maximale Fluss nimmt auch den Wert des minimalen cuts an.

Beweis. verwende residualgraphen, fluss ist genau dann maximale, wenn es dort kenie kreise mehr gibt, dann folgt die optimalität leicht, indem man sich die von s aus erreichbaren Knoten ansieht (darunter ist nicht t), diese bilden einen minimalen Schnitt. □

Algorithmus 10 : Ford-Fulkerson-Algorithmus

Eingabe : Netzwerk (G, u, s, t) mit ganzzahligen Kosten $u : E(G) \to \mathbb{Z}_+$
Ausgabe : maxmaler Fluss
Laufzeit : $\mathcal{O}(m \cdot f_{\max})$

Beweis. Augmentiere nacheinander entlang der Wege im Residualgraphen. □

Warnung. dieser algorithmus terminiert nicht enimal zwingend, wenn wir ihn für nicht-ganzzahlige Gewichte aufrufen, und der flusswert konvergiert auch nicht zwingend gegen den optimalen flusswert.

10 Menger

- Flusszerlegung
- Menger
- k-Knoten / Kantenzusammenhang charakterisieren.

Satz 10.1 (Gallai 1958, Ford Fulkerson 1962)**.** Sei (G, u, s, t) ein Netzwerk und f ein $s-t$-Fluss in (G, u). Dann existiert eine Menge $\mathcal{P}$ vos $s-t$-Wegen und eine Menge $\mathcal{C}$ von Kreisen im G mit Gewichten $w : \mathcal{P} \cup \mathcal{C} \to \mathbb{R}_{\geqslant 0}$, so dass ($w$ ganzzahlig falls f ganzzahlig)

$$f(e) = \sum_{P \in \mathcal{P} \cup \mathcal{C} : e \in E(C)} w(P) \qquad \forall e \in E(G).$$

und

$$\text{value}(f) = \sum_{P \in \mathcal{P}} w(P).$$

und $|\mathcal{P}| + |\mathcal{C}| \leqslant |E(G)|$.
Solche $\mathcal{P}, \mathcal{C}, w$ können in $O(nm)$ Zeit gefunden werden.

Beweis. Sehr einfach mit Induktion nach $|\{e \in E(G) \mid f(e) > 0\}|$ □

Satz 10.2 (Menger 1927)**.** Sei G ein Graph (gerichtet oder ungerichtet) und s, t zwei Knoten sowie $k \in \mathbb{N}$. Dann existieren k paarweise Kantendisjunkte $s-t-$ Wege in G genau dann, wenn t nach Entfernen beliebiger $k-1$ Kanten stets t von s aus erreichbar ist.

Satz 10.3 (Menger 1927)**.** Sei G ein Graph (gerichtet oder ungerichtet). Seien s, t zwei nicht benachbarte Knoten. Dann existieren k paarweise **intern disjunkte** $s-t$-Wege (die also paarweise keine Knoten außer s und t gemeinsam haben) gdw nach Entfernen von beliebigen $k-1$ Knoten (außer s und t) t stets von s aus erreichbar bleibt.

Korollar 10.4 (Whitney 1932)**.** (a) Ein ungerichteter Graph G mit $\geqslant 2$ Knoten ist k-Fach Kantenzusammenhängend, gdw für alle $s \neq t \in V(g)$ k paarweise Kantendisjunkte $s-t$-Wege existieren.

(b) Ein ungerichteter Graph G mit mehr als k Knoten ist k-fach (Knoten) zusammenhängend gdw für alle $s, t \in V(G)$ mit $s \neq t$ k paarweise intern disjunkte $s-t$-Wege existieren.

Bemerkung. Ford-Fulkerson funktioniert bekanntlich nur für Kapazitäten aus $\mathbb{Z}$. Aber selbst dann hat er nicht polynomielle Laufzeit.

11 Dinic

- Ziel: polynomielle Algorithmen für Max-Flow-Problem
- Edmonds-Karp - Algorithmus
- Dinics Algorithmus
- Blockierende Flüsse, Präflüsse
- Wie finde ich blockierenden Fluss in azyklischem Graphen?

Lemma 11.1. Ist G gerichteter Graph mit $s \neq t \in V(G)$ und ist F die Vereinigung der Kantenmengen aller kürzesten $s-t$-Wege in G, so können wir für $e \in F$ auch die Kante $\overleftarrow{e}$ zu $E(G)$ hinzufügen, und die Vereinigung der Kanten der kürzesten $s-t$-Wege ändert sich nicht.
Mit anderen Worten: Das Hinzufügen von $\overleftarrow{e}$ bringt keine neuen kürzesten Wege mit sich.

Algorithmus 11 : Edmonds-Karp-Algorithmus

Eingabe : Netzwerk (G, u, s, t) mit $u : E(G) \to \mathbb{R}_+$
Ausgabe : ein maximaler $s-t$-Fluss
Laufzeit : $\mathcal{O}(m^2 n)$

Beweis. Zunächst verringert sich $\text{dist}_{G_f}(s,t)$ nie nach obigem Lemma (jeder kürzere Weg müsste ja $\overleftarrow{e}$ benutzen). Solange die Distanz gleich bleibt, kommen jedoch keine Kanten zu obigem F hinzu (auch lemma) und in jeder Iteration verschwindet mindestens eine. Also muss sich die Distanz nach $\leqslant 2m$ Schritten erhöhen (Anzahl Kanten in G_f) und es kann höchstens n mal die Distanz steigen, also brauchen wir $2mn$ Iterationen, und jede Iteration kann mit BFS in $\mathcal{O}(n)$ bewältigt werden. □

Algorithmus 12 : Dinics Algorithmus

Eingabe : ein Netzwerk (G,u,s,t)
Ausgabe : ein maximaler Fluss f in (G,u)
Laufzeit : $\mathcal{O}(n \cdot \mathcal{O}(Iteration))$, mit Karzanov z.B. $\mathcal{O}(n^3)$

Beweis. Wichtig ist, dass wir nur im Levelgrahp G_f^L arbeiten. Dann erhöht sich $\text{dist}_{G_f}(s,t)$ in jeder Iteration nach obigem Lemma. □

Satz 11.2 (Karzanov 1974)**.** In einem azyklischen Netzwerk (G,u,s,t) kann ein blockierender Fluss in $\mathcal{O}(n^2)$ Zeit gefunden werden.

Beweis. Führe hierzu abwechselnd push und balance aus. Wichtig ist hierbei, dass ein knoten, der gebalanced wurde, nie wieder aktiv wird, und dass wir stets die Eigenschaft erhalten, dass es keinen $s-t$-Weg gibt, der nicht-saturierte Kanten benutzt. Da wir in jeder Iteration einen Knoten schließen, sind das n Stück und jede Iteration braucht linear viel Zeit. □

12 Goldberg-Tarjan

- Distanzmarkierungen
- Goldberg-Tarjan

Algorithmus 13 : Push-Relabel-Algorithm

Eingabe : Netzwerk (G,u,s,t)
Ausgabe : maximaler Fluss f in (G,u)
Laufzeit : $\mathcal{O}(n^2\sqrt{m})$

Beweis. Wir arbeiten mit Distanzmarkierung ψ und pushen immer nur abwärts. Dabei erhalten wir, dass ψ Distanzmarkierung ist, und dass f ein Präfluss ist. Wenn Alg terminiert, ist er also automatisch korrekt.
Nun stellen wir zunächst fest, dass $\psi \leqslant 2n-1$, also kann ψ auch nur $\mathcal{O}(n^2)$ mal erhöht werden.
Die Anzahl der saturierenden Pushs ist zudem beschränkt durch $2mn$, weil dann die Kante verschwindet, und wir erst $\psi(v)$ um 2 erhöhen müssen, bevor wir wieder entlang der Kante pushen können.
Es verbleiben die nicht-saturierenden Pushes, hier unterteilen wir in phasen, in denen sich ψ nicht ändert. Wir erhalten durch die Relabel-Abschätzung, dass es $\mathcal{O}(n^2)$ Phasen gibt, und wollen nun die pushes in diesen abschätzen durch $\sqrt{m}$. Die billigen phasen vergessen wir und interessieren uns für die teuren, in denen mehr als $\sqrt{m}$ nicht-saturierende pushes gemacht werden. Dazu betrachten wir die Variable

$$\Phi := \sum_{\substack{v \in V(G) \\ v \text{ aktiv}}} |\{w \in V(G) \mid \psi(w) \leqslant \psi(v)\}|.$$

die sich nur bei relabel oder saturierenden pushes um $\leqslant n$ erhöht, also um höchstens $\mathcal{O}(n(n^2 + mn)) = \mathcal{O}(mn^2)$.
In teuren phasen verringert jeder nicht-saturierende push nun Φ um mindestens $\sqrt{m}$, und damit gibt es höchstens $\mathcal{O}(\frac{mn^2}{\sqrt{m}}) = \mathcal{O}(n^2\sqrt{m})$ viele nicht-saturierende pushes in solchen phasen und wir haben auch die nicht-saturierenden pushes durch $\mathcal{O}(n^2\sqrt{m})$ beschränkt. Also haben wir sowohl die anzahl der pushes als auch anzahl der relabels so beschränkt und damit folgt korrektheit und die laufzeit. □

13 Min-Cut

- Min-Cut Problem auf gesamten Graphen
- MA-Ordnung

Lemma 13.1. Seien $i, j, k \in V(G)$. Dann ist $\lambda_{ik} \geqslant \min\{\lambda_{ij}, \lambda_{jk}\}$

Beweis. Verwende max-flow-min-cut □

Lemma 13.2. Sei (G, u) Graph mit Kapazitäten und $n = |V(G)| \geqslant 2$. Sei $v_1, \ldots, v_n$ eine MA-Ordnung von (G, u). Dann ist

$$\lambda_{v_{n-1}v_n} = u(\delta(v_n)).$$

Beweis. Induziere, indem wir

$$\lambda_{v_{n-1},v_n} \geqslant \min\left\{\lambda^G_{v_{n-2},v_{n-1}}, \lambda^G_{v_{n-2},v_n}\right\}.$$

ausnutzen und die MA-Ordnung benutzen. □

Satz 13.3 (Nagamochi, Iberaki 1992)**.** Für Graphen G mit $u : E(G) \to \mathbb{R}_+$ können wir ein $\varnothing \neq X \subseteq V(G)$ mit $u(\delta(X))$ minimal in $\mathcal{O}(mn + n^2 \log n)$ finden.

Beweis. Berechne hierzu immer eine MA-Ordnung auf dem Graphen und merke $\gamma_: = \lambda^{G_i - 1}_{v_n,v_{n-1}}$, kontrahiere dann v_n und v_{n-1} und iteriere. Das kleinste γ ist dann die Lösung und wir finden den Schnitt X als die Knoten, die kontrahiert wurden um γ_i zu bestimmen. □

Algorithmus 14 : Karger-Stein-Algorithmus (randomisiert)

Eingabe : Graph G mit Kosten $u : E(G) \to \mathbb{R}_+$
Ausgabe : mit einer Wahrscheinlichkeit $\geqslant \frac{1}{\binom{n}{2}}$ ein minimaler Schnitt
Laufzeit : $\mathcal{O}(nm)$ oder zumindest: n Iterationen und in jeder eine kante zufällig wählen.

14 MinCostFlow

- Minimum cost flow problem
- Satz von Gale zum Entscheiden der lösbarkeit von min-cost-flow
- Hitchcock problem

- Residualgraphen
- Optimalitätskriterium

Satz 14.1 (Gale 1957)**.** Sei (G, u) Graph mit Kapazitäten und $b : V(G) \to \mathbb{R}$ mit $b(V(G)) = 0$ ein Angebot. Dann existiert ein b-Fluss in (G, u) genau dann, wenn für alle $X \subseteq V(G) : u(\delta^+(X))) \geqslant b(X)$. Man kann einen b-Fluss finden oder zeigen, dass keiner existiert in $\mathcal{O}(n^2\sqrt{m})$ Zeit.

Satz 14.2. Füge s, t hinzu und verwende max-flow.

Hitchcock-Problem ein gerichteter bipartiter Graph $G = A \sqcup B$ mit $E(G) \subseteq A \times B$. Angebote $b(v) > 0$ für $v \in A$ und Nachfragen $-b(v) \geqslant 0$ für $v \in B$. Kosten $c : E(G) \to \mathbb{R}$.
Finde einen b-Fluss f in (G, ∞) mit minimalen Kosten oder entscheide, dass keiner existiert.

Lemma 14.3 (Orden, Wagner)**.** Eine Instanz von min-cost-flow mit n Knoten und m Kanten kann in eine äquivalente Instan des Hitchcock-Problems mit $n + m$ Knoten und $2m$ Kanten transformiert werden.

Proposition 14.4. Sei (G, u, b, c) eine Instanz von Min-Cost-Flow und seien f, f' zwei b-Flüsse in (G, u).
Definiere $g : E(\overleftrightarrow{G}) \to \mathbb{R}_+$ durch

$$g(e) := \max\left\{0, f'(e) - f(e)\right\} \qquad g(\overleftarrow{e}) := \max\left\{0, f(e) - f'(e)\right\}.$$

Dann ist g eine Zirkulation in $\overleftrightarrow{G}$ und es ist $g(e) = 0\ \forall e \notin E(G_f)$ und es ist $c(g) = c(f') - c(f)$

Proposition 14.5. Für jede Zirkulation f in einem gerichteten Graphen G gibt es eine Menge $\mathcal{C}$ von höchstens $|E(G)|$ Kreisen in G und Zahlen $h(C) > 0$ für $C \in \mathcal{C}$ mit

$$f(e) = \sum_{\substack{C \in \mathcal{C} \\ e \in E(C)}} h(C).$$

Beweis. Flusszerlegung. □

Satz 14.6 (Tolstoi 1930, Klein 1967)**.** sei (g, u, b, c) eine Instanz von Min-Cost-Flow. Ein b-Fluss f in (G, u) ist optimal genau dann, wenn es keinen f-augmentierenden Kreis mit negativem Gesamtkosten gibt.

Beweis. Kreis $\Rightarrow$ nicht optimal ist trivial, für andere Richtung definiere Zirkulation wie g oben und zerlege sie in Kreise, einer davon ist negativ. □

Korollar 14.7 (Ford, Fulkerson 1962)**.** Ein b-Fluss f ist optimal gdw es ein zulässiges Potenzial in (G_f, c) gibt.

15 Min-Mean-Cycle-Cancelling

Obiger Satz legt einen Algorithmus nahe:

Algorithmus 15 : Minimum Mean Cycle Cancelling Algorithm

Eingabe : gerichteter Graph G, Kapazitäten $u : E(G) \to \mathbb{R}_+$, Zahlen $b : V(G) \to \mathbb{R}$ mit $b(V(G)) = 0$, Kosten $c : E(G) \to \mathbb{R}$
Ausgabe : ein kostenminimaler b-Fluss oder die Meldung, dass keiner existiert
Laufzeit : $\mathcal{O}(m^3n^2 \log n)$ oder auch (schwach polynomiell) $\mathcal{O}(m^2n^2 \log(n|c_{min}|))$

Beweis. Eine Iteration können wir mit Karp in $\mathcal{O}(nm)$ Zeit bewältigen, also interessieren wir uns für die anzahl der Iterationen. Hierzu zeigen wir, dass die Durchschnittskosten der Kreise entlang denen wir augmentieren monoton steigend sind und auch alle mn Iterationen um den Faktor 2 größer werden - das erreichen wir durch Abschätzen der Kosten der Vereinigung zweier aufeinanderfolgender Kreise, und dass diese nach m Iterationen zwei gegenläufige Kanten enthalten, da wir bei jedem augmentieren eine Kante vernichten.
Dann erhalten wir zusammen, dass $|\mu(f)|$ zu Beginn ja höchstens $|c_{\min}|$ war und wir somit nach $\mathcal{O}(m^2n^2 \log(n \cdot |c_{min}|))$ vielen Iterationen $|\mu(f)| < \frac{1}{n}$ haben und der Algorithmus ist terminiert. Korrektheit ist trivial. □

Satz 15.1 (Goldberg - Tarjan 1989)**.** Der Algorithmus terminiert nach $\mathcal{O}(m^2n \log n)$ vielen Iterationen, hat alos Laufzeit $\mathcal{O}(m^3n^2 \log n)$.

Beweis. Zegie, dass alle $mn(\lceil \log n \rceil + 1)$ Iterationen eine Kante fixiert wird, d.h. ihr Flusswert ändert sich nicht mehr. Dazu stellen wir fest, das alle solche vielen Iterationen $\mu(f) \leqslant 2n\mu(f')$ nach gleicher Abschätzung wie bei letztem Algorithmus, und wir somit eine 'stark negative Kante' finden. Jetzt können wir für diese Kante zeigen, dass $\mu(f'') < \mu(f')$ ist (nachrechnen) für jeden Fluss f'', der diese Kante im Residualgraphen enthält. Damit ist diese Kante fixiert, weil die μ ja monoton wachsen. Nun gibt es m Kanten und eine iteration braucht $\mathcal{O}(mn)$ Zeit, also erhalten wir insgesamt die gewünschten $\mathcal{O}(m^3n^2 \log n)$ Laufzeit. □

Korollar 15.2. Für jede Min-Cost-Flow Instanz (G, u, b, c), für die u und b ganzzahlig sind und es einen optimalen b-Fluss gibt, gibt es auch einen optimalen ganzzahligen b-Fluss.

16 SuccessiveShortestPath

- Sukzessive Kürzeste Wege Algorithmus
- Kapazitäts-Skalierungs-Algorithmus

Satz 16.1 (Jewell 1958)**.** Sei (G, u, b, c) eine Instanz von Min-Cost-Flow. Sei f ein optimaler b-Fluss und sei P ein kürzester s-t-Weg in (G_f, c) für irgendwelche s, t. Sei f' der b'-Fluss, der aus f entsteht durch Augmentation entlang von P um $\gamma \leqslant \min\{u_f(e) \mid e \in E(P)\}$ für geeignetes b'. Dann ist f' ein optimaler b'-Fluss.

Algorithmus 16 : Sukzessive Kürzeste Wege Algorithmus

Eingabe : gerichteter Graph G, Kapazitäten $u : E(G) \to \mathbb{Z}_+$, Zahlen $b : V(G) \to \mathbb{Z}$ mit $b(V(G)) = 0$, konservative Gewichte $c : E(G) \to \mathbb{R}$
Ausgabe : ein optimaler b-Fluss (oder Meldung, dass keiner existiert)
Laufzeit : $\mathcal{O}(nm + B(m + n \log n)))$ mit $B := \frac{1}{2}\sum_{v \in V(G)} |b(v)|$

Beweis. Verwende obigen Satz, dann ist Korrektheit klar. Wenn wir behaupten, dass kein Kreis existiert, dann ist das auch klar nach Satz von Gale. Laufzeit ergibt sich durch die einfache Abschätzung, dass wir $\sum_{v \in V(G)} |b(v)|$ in jedem Schritt um $2\gamma \geqslant 2$ verringern. Um nicht stets mit MBF zu arbeiten, arbeiten wir mit einem Potenzial in jeder Iteration, das wir updaten, also brauchen wir nur einmal MBF und danach in jeder Iteration Dijkstra, das führt zu obiger Laufzeit. □

Bemerkung. Falls c nicht konservativ, könnten wir anfangs alle e mit $c(e) < 0$ satuireren und mit dem entsprechenden b' weitermachen.

Algorithmus 17 : Kapazitäts-Skalierungs-Algorithmus

Eingabe : gerichteter Graphe G mit unendlichen Kapazitäten $u(e) = \infty$, Zahlen $b : V(G) \to \mathbb{Z}$ mit $b(V(G)) = 0$, konservative Gewichte $c : E(G) \to \mathbb{R}$
Ausgabe : optimaler b-Fluss in (G, ∞)
Laufzeit : $\mathcal{O}(n(m + n \log n) \log(2 + b_{\max}))$

Beweis. Korrektheit wie vorhin auch, also schätzen wir die Dauer einer Phase ab (in denen γ Konstant bleibt). Dies sind höchstens n Stück, also können wir gleich vorgehen wie bei sukzessive-kürzeste-wege algo, indem wir in jeder iteration dijkstra anwenden, und wir brauchen nur $\log(2 + b_{\max})$ viele Phasen. Das $\mathcal{O}(mn)$ von einmaligem MBF wird durch den Rest dominiert.

□

17 BipartiteMatching

- Bipartite Graphen
- Zuordnungsproblem
- Matchings
- Satz von König
- Heiratssatz
- Doppelt-Stochastische-Matrizen

Satz 17.1 (König)**.** Ein Graph ist bipartit, gdw es keinen ungeraden Kreis gibt.

Satz 17.2. Wir können das Zuordnugsproblem in $\mathcal{O}(n^3)$ Zeit lösen.

Beweis. Auf Min-cost-flow zurückführen mit Nachfragen $1, -1$ und entsprechenden Kosten. □

Satz 17.3 (König 1931)**.** In einem Bipartiten Graphen ist $\tau(G) =$ kardinalität einer minimalen Knotenüberdeckung gleich $\nu(G) =$ Kardinalität eines maximalen Matchings.

Beweis. Die Ungleichung $\nu(G) \leqslant \tau(G)$ ist trivial, für Rückrichtung verwende Satz von Menger. □

Satz 17.4 (Hall 1935)**.** bipartiter Graph G hat maximales Matching, das A überdeckt, genau dann, wenn

$$\forall X \subseteq A\colon \quad |\Gamma(X)| \geqslant |X| \qquad \text{(Hall-Bedingung)}.$$

Beweis. Notwendigkeit klar. Für Rückrichtung wende Satz von König an. □

Satz 17.5. In einem bipartiten Graphen G können wir ein maximales Matching und eine minimale Knotenüberdeckung in $\mathcal{O}(nm)$ Zeit finden.

Beweis. Zurückführen auf max-flow und dann mit ford-fulkerson lösen. □

Satz 17.6 (Petersen 1891, Berge 1957)**.** Sei G ein Graph und M ein Matching in G. Dann ist M maximal gdw es keinen M-augmentierenden Weg gibt.

Satz 17.7. Sei U endliche Menge und $\mathcal{J}$ eine Familie von Teilmengen von U. dann hat $\mathcal{J}$ ein Repräsentatensystem, d.h. eine injektiive Abbildung $f : \mathcal{J} \to U$ mit $f(S) \in S$ für $S \in \mathcal{J}$, genau dann, wenn $\forall J \subseteq \mathcal{J} : |\bigcup J| \geqslant |J|$

Satz 17.8 (Birkhoff 1946, von Neumann 1953)**.** Jede doppelt stochastische Matrix ist Konvexkombination von Permutationsmatrizen.

18 TuringMaschinen

- Turing Maschinen
- Berechnungsprobleme

- Church'sche These

Satz 18.1 (Church'sche These)**.** Die menge der (in poly Zeit) berechenbarer Funktionen ist unabhnängig von der Konkreten Definition des Algorithmus, d.h. alles was mit einem Computer geht, geht auch mit einer Turing-Maschine.

Satz 18.2. Wir können mehr-band-TMs in ein-band-tms übersetzen, und das barucht nur polynomiell viel zeit.

19 PundNP

- P, NP
- Hamiltonkreis
- randomisierte Algorithmen, nicht-deterministisch-polynomielle algorithmen
- Polynomielle Reduktion.

Proposition 19.1. $P \subseteq NP$

Proposition 19.2. Ein entscheidungpsroblem $P = (X, Y)$ liegt in NP genau dann, wenn es einen polynomiellen nicht-deterministischen Algorithmus gibt.

Proposition 19.3. Ist P_1 auf P_2 reduzierbar und gibt es einen polynomiellen Algorithmus, der P_2 löst, dann gibt es auch einen für P_1.

20 Cook

- NP-schwere Probleme, NP-vollständige Probleme
- satz von cook

Proposition 20.1. Falls es einen poly-algo für ein NP-schweres problem gibt, ist $P = NP$.

Bemerkung. NP vollständig $\Rightarrow$ NP -schwer

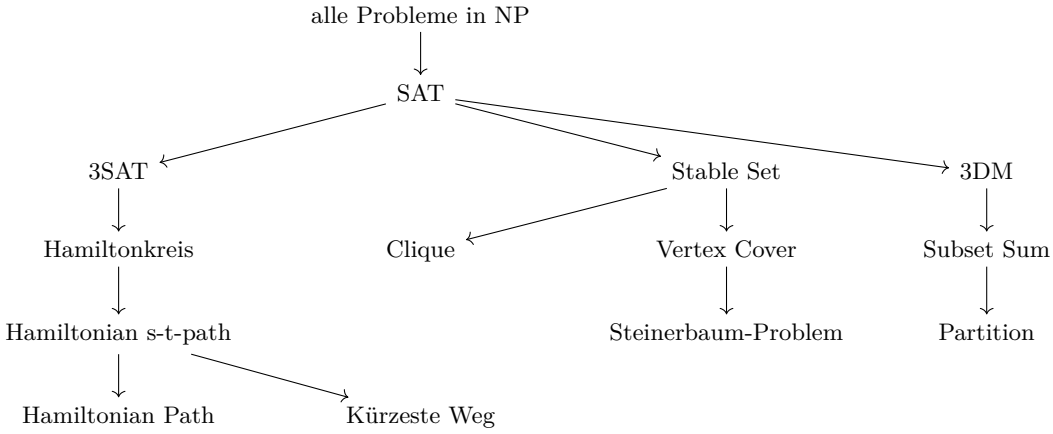
Proposition 20.2. Ein NP-vollständiges Problme hat genau dann einen polynomiellen Algorithmus, wenn $P = NP$.

Satz 20.3 (Cook 1971)**.** SAT ist NP-vollständig.

Satz 20.4. 3SAt ist NP-vollständig.

21 NPcompleteProblems

- weitere NP-vollständige Probleme



22 3DM

- 3-dim-matchings
- subset sum, partition
- coNP

Satz 22.1. Falls $NP \neq coNP$, so liegt kein coNP-vollständiges Problem in NP

23 TSP

- TSP
- Varianten: in Graphen, metrisches TSP
- Steinerbaum-Problem

Proposition 23.1. Das TSP ist NP-schwer, selbst wenn $c(e) \in \{1, 2\}$.

Beweis. Reduziere Hamiltonkreis auf TSP. □

Satz 23.2. Das TSP (mit ganzz. Kantengewichten) hat genau dann einen polynomiellen Algorithmus, wenn P = NP.

Beweis. Zurückführen aus das TSPE (gibt es Tour mit $\leqslant k$ Kosten), dieses hat (nach annahme) einen poly-algorithmus. □

Satz 23.3. es gibt einen (exakten) Algorithmus mit Laufzeit $\mathcal{O}(n^2 2^n)$ für das TSP

Satz 23.4 (Sahni 1976)**.** Falls $P \neq NP$, so gibt es keinen k-Approximationsalgorithmus für das TSP für irgendein $k \geqslant 1$.

Beweis. Transformiere Hamiltonkreis hierauf. □

Satz 23.5. Das metrische TSP ist NP-schwer.

Proposition 23.6. Das metrische TSP und das TSP auf Graphen sind äquivalent. Gibt es einen k-Approximationsalgorithmus für das eine, so auch für das andere.

Algorithmus 18 : Doppelbaum-Algorithmus

Eingabe : Instanz (G, c) des TSP in Graphen
Ausgabe : eine Tour T, die höchstens doppelt so teuer ist wie die optimale Tour.
Laufzeit : $\mathcal{O}(m + n \log n)$

Bemerkung. man kann mit etwas verfeinerung auch einen $\frac{3}{2}$-Approximationsalgorithmus daraus bauen.

Satz 23.7 (Karp 1972)**.** Das Steinerbaum-Problem ist NP-schwer, selbst wenn $c(e) = 1$ für alle $e \in E(G)$

Beweis. Reduktion von Vertex Cover. □

24 Multiflow

- Algorithmus für Steinerbaum-Problem
- Multiflow-Problem
- Kantendisjunkte-Wege-Problem

Algorithmus 19 : 2-Approximationsalgorithmus

Eingabe : ungerichteter Graph G, Kosten $c : E(G) \to \mathbb{R}_+$ und $\varnothing \neq T \subseteq V(G)$
Ausgabe : ein Steinerbaum für T in G, oder die Meldung, dass keiner existiert
Laufzeit : $\mathcal{O}(n^3)$

Proposition 24.1. (a) Sei (G, H, u, b) Instanz des Multiflow-Problems. Existiert eine Lösung, so ist das Schnittkriterium erfüllt.

(b) Sei (G, H) Instanz des KDW-Problems. Dann gilt:
$\exists$ Lösung $\Rightarrow \exists$ gebrochene Lösung $\Rightarrow$ Schnittkriterium erfüllt.

Satz 24.2. Sei (G, H) eine Instanz des (gerichteten oder ungerichteten) KDW-Problems mit $v \in V(G)$, so dass folgendes gilt:
(a) $f \in \delta^+(v) \forall f \in E(H)$
(b) $f \in \delta^-(v) \forall f \in E(H)$
(c) $f \in \delta(v) \forall f \in E(H)$
Dann hat (G, H) eine Lösung gdw das Schnittkriterium erfüllt ist.

Lemma 24.3. Sei (G, H) und v wie oben. (G', H') gehe aus (G, H) hervor durch hinzufügen eines neuen Knoten x und erszeten von jeder mit v inzidenten Nachfragekante f wie folgt
- Falls $f = \{v, w\}$, ersetze f ducrh $\{v, x\}$ und füge Angebotskante $\{w, x\}$ hinzu
- Falls $f = (v, w)$, ersetze f durch (v, x) und füge Angebotskante (x, w) hinzu
- Falls $f = (w, v)$, so ersetze f durch (x, v) und füge Angebotskante (w, x) hinzu

Dann gilt: (G', H') hat genau dann eine Lösung, wenn (G, H) eine Lösung hat und (G', H') erfüllt das Schnittkriterium genau dann, wenn (G, H) das Schnittkriterium erfüllt.

Beweis. Aus dem Lemma folgt der Satz, weil wir jetzt einfach Menger anwenden können. □

25 gerichteteKDW

-

Bemerkung. Aus obigem Vorgehen folgt auch ein polynomieller Algorithmus für das KDW-Problem, wenn $f \in \delta(v)$ für $f \in E(H)$.

Satz 25.1 (1976)**.** Das gerichtete KDW-Problem ist sogar dann NP-schwer, wenn G azyklisch ist und H nur aus zwei Menegn paralleler Kanten besteht.

Satz 25.2. Transformation von SAT durch geschickten Graphen.

Proposition 25.3. Sei (G, H) Instanz des gerichteten KDW-Problems, wobei H nur eine Menge paralleler kanten enthält und $G + H$ eulersch ist. Dann hat (G, H) immer eine Lösung.

Beweis. Zerlege $G + H$ in Kreise, keiner enthält zwei Kanten aus H, und

wähle die Kreise die (genau) eine Kante aus H enthalten. □

Satz 25.4 (Nash-Williams 1969)**.** Sei (G, H) Instazn des gerichteten KDW-Problems, obei H nur aus zwei Mengen paralleler Kanten besteht und $G + H$ eulersch ist. Dann hat (G, H) genau dann eine Lösung, wenn das Schnittkriterium erfüllt ist.

Beweis. Verwende zuerst einmal Menger und dann obige Proposition. □

H hat nur k Mengen paralleler Kanten	$G + H$ beliebig	$G + H$ Eulersch
$k = 1$	Menger (Lösung $\Leftrightarrow$ Schnittkriterium)	$\exists$ immer Lösung
$k = 2$	NP-schwer	$\exists$ Lösung $\Leftrightarrow$ Schnittkriterium
$k = 3$	NP - schwer	NP-schwer

Satz 25.5. Das gerichtete KDW-Problem ist sogar dann NP-schwer, wenn G azyklisch ist und $G + H$ eulersch ist und H nur aus drei Mengen paralleler Kanten besteht.

Beweis. Transformation von $G + H$ beliebig, aber $k = 2$ durch einführen von dritter kantenmenge, die den Graphen eulersch macht. □

Lemma 25.6. Sei (G, H) eine Instanz des gerichteten KDW-Problems, wobei G azyklisch und $G + H$ eulersch ist. Betrachte die Instanz (G', H') des ungerichteten KDW-Problems, die durch Ignorieren aller Orientierung entsteht. Dann ist eine Lösung von (G, H) eine Lösung von (G', H') und umgekehrt.

Satz 25.7. Das ungerichtete KDW-Problem ist sogar dann NP-vollständig, wenn $G+H$ eulersch ist und H nur drei Mengen paralleler Kanten enthält.

Beweis. Folgt aus den vorherigen beiden Resultaten. □

26 Rothschild-Whinston

- Rothschild-Whinston
- Ford-Fulkerson zur Orientierung von Graphen

Satz 26.1 (Rothschiled, Whinston 1966)**.** sei (G, H) eine Instanz des ungerichteten KDW, wobei $G + H$ eulersch ist und H eine Kantenüberdeckung der Kardinalität 2 hat. Dann hat (G, H) genau dann eine Lösung, wenn das Schnittkriterium erfüllt ist.

Beweis. Nutze FF um die Kanten so zu orientieren, dass das Schnittkriterium und die Euler-Eigenschaft erhalten bleiben. Dann verwende Nash-Williams. □

Satz 26.2 (Ford, Fulkerson 1962)**.** Sei G ein ungerichteter Garph und H ein gerichteter Graph mit $V(G) = V(H)$. Dann hat G eine orientierung G', sodass $G' + H$ eulersch ist, gdw:

- $\left|\delta_H^+(v)\right| + \left|\delta_H^-(v)\right| + |\delta_G(v)|$ ist gerade für alle $v \in V(G)$
- $\left|\delta_H^+(X)\right| - \left|\delta_H^-(X)\right| \leqslant |\delta_G(X)| \quad \forall X \subseteq V(G)$

Beweis. Die eine richtung ist trivial. Für die andere nutze induktion nach Kanten. Die kritische Bedingung ist die 2, wir untersuchen sog. kritische Mengen für die Gleichheit gilt. Gibt es keine, können wir irgendwie orientieren, gibt es welche, wissen wir schon wie wir orientieren müssen.
Ein (mehr oder weniger) einfaches Abschätzungsargument zeigt dann, dass eine Kante nicht gezwungen ist, in zwei Richtungen gleichzeitig gerichtet zu werden.
Alternativ: Verwende b-Flüsse. Orientier alle Kanten in bedie Richtungen, setze

$$b(v) = \left|\delta_H^-(v)\right| - \left|\delta_H^+(v)\right|.$$

für $v \in V(G)$ und zeige nun, dass es einen b-Fluss im so erhaltenen Graphen G' gibt. Diesen wählen wir ganzzahlig, und wir orientieren unn in G Kanten von v nach w wenn $f((v,w)) = 1$ und $f((w,v)) = 0$, damit haben wir enien Teil von G orientirt, sodass $G_1 + H$ eulersch ist. Den restlichen (eulerschen) Teil von G orientieren wir dann noch (trivial) und sind fertig. □

Literatur

[1] Bernhard Korte, Jens Vygen, *Kombinatorische Optimierung*, Springer, 2018