

Ausgewählte Graphenalgorithmen

12. Februar 2008

Gehalten im

Wintersemester 2007 / 2008

von

PROF. DR. MICHAEL KAUFMANN
und
DR. KATHARINA ZWEIG

Mitschrift von

BASTIAN BRODBECK,
TILL HELGE HELWIG
und
STEFAN HUSTER

Inhaltsverzeichnis

1 Informationen und Grundlagen	1
1.1 Themenüberblick	1
1.2 Übung	1
1.3 Grundlagen	1
1.4 Bäume	2
2 Isomorphie	3
2.1 Isomorphie geordneter Bäume	3
2.2 Isomorphie ungeordneter Bäume	4
3 Teilbaumisomorphie (Motivsuche)	5
3.1 Top-Down geordnete Teilbaumisomorphie	5
3.2 Top-Down ungeordnete Teilbaumisomorphie	5
3.3 Bottom-Up geordnete Teilbaumisomorphie	6
3.4 Größtmögliche gemeinsame Top-Down-Teilbäume	6
3.5 Teilgraphisomorphie	7
3.6 Isomorphieproblem für allgemeine Graphen	8
4 Teilgraphen mit besonderen Eigenschaften	8
4.1 Aufgabe des Algorithmikers	9
4.2 Clustern von Graphen	9
4.3 Spanner	12
4.4 Spann bäume mit niedrigem, durchschnittlichem Stretch	13
4.5 Teilgraphen, die Zusammenhänge enthalten	17
5 Minimale Kreisbasen	19
5.1 Kreisbasen	19
5.2 Kreisbasen minimalen Gewichts	20
5.2.1 Der Algorithmus von Horton	21
5.3 Obere Schranke für minimale Kreisbasen	23
5.4 Obere Grenze für die Länge einer MCB (Horton '87)	23
6 Max Flow & Min Cost Flow	25
6.1 Algorithmus (Ford/Fulkerson)	27
6.2 Algorithmus 2	28
6.3 Algorithmus 3	29
6.4 Preflow-Push-Algorithmen	30
6.4.1 Verbesserung:	31
6.4.2 Regeln zur Wahl aktiver Knoten:	31
6.4.3 FIFO Preflow Push	32

6.4.4	Highest-Label	32
6.5	MinCut	34
6.5.1	Heuristik	36
6.6	Min Cost Flow	36
6.6.1	Cycle Canceling	37
6.6.2	Vermeide negative Kantenkosten	38
6.6.3	Algorithmen	38
6.6.4	Bellman/Ford:	39

1 Informationen und Grundlagen

1.1 Themenüberblick

- Isomorphie (Prof. Kaufmann, etwa 3 Wochen)
- Zyklenbasen und Spanner (Nina Zweig, bis Weihnachten)
- MinCostFlow oder Planarität (Prof. Kaufmann)

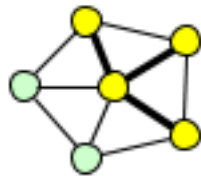
1.2 Übung

Dienstag 16 bis 17 Uhr

1.3 Grundlagen

- Graph $G = (V, E)$
- azyklische Graphen (haben immer eine Topologische Sortierung)
- zusammenhängende (ungerichtete) Graphen
- stark zusammenhängende Graphen (es gibt jeweils Pfade von v nach w und umgekehrt)
- k -fach zusammenhängende Graphen (es gibt k disjunkte Pfade von v nach w)
- Spannbäume
- t -Spanner ($\frac{\sum \text{Kantenlängen aller Kanten auf dem Pfad von } v \text{ nach } w}{\text{Länge der Kante von } v \text{ nach } w} \leq t$)
- NP-vollständige Probleme:
 - TSP [Travelling-Salesman-Problem] (Rundtour in vollständigem Graph)
 - Vertex-Cover [Knotenüberdeckung] (Markierung von Knoten, so dass für jede Kante mindestens End- oder Startknoten markiert ist)
 - Independent Set [unabhängige Menge] (Knoten markieren, so dass keine benachbarten Knoten markiert sind)
 - k -Clique (vollständiger Teilgraph mit k Knoten)
 - Hamilton-Kreis (Rundtour in nicht vollständigem Graph)
- Abkürzungen:
 - V ... Knoten, $n = |V|$ (Knotenanzahl)
 - E ... Kanten, $m = |E|$ (Kantenanzahl)
 - $\text{indeg}(v)$... Anzahl eingehende Kanten an Knoten v
 - $\text{outdeg}(v)$... Anzahl ausgehender Kanten an Knoten v
 - $\text{deg}(v)$... Anzahl ein- und ausgehender Kanten an Knoten v
 - Es gilt: $\sum_v \text{indeg}(v) = \sum_v \text{outdeg}(v) = m$
 - $\sum_v \text{deg}(v) = 2m$
 - $\sum_v (\text{deg}(v))^2 \leq \sum_v (n \cdot \text{deg}(v)) = n \cdot (\sum_v \text{deg}(v)) = \mathcal{O}(n \cdot m)$

- Begriff:
 - Pfad von v nach w in G (Verbindung von v nach w über beliebig viele Knoten)
 - einfacher Pfad (Verbindung von v nach w über beliebig viele Knoten, in denen keiner doppelt vorkommt)
 - Teilgraph (W, S) von (V, E) , falls $W \subseteq V$ und $S \subseteq E$
 - induzierter Teilgraph: $(W, E \cap (W \times W))$



Teilgraph



induzierter Teilgraph

1.4 Bäume

- $T = (V, E)$ ist ein gerichteter Baum mit Wurzel r , falls er zyklfrei ist, Wurzel $r \in V$ existiert und für alle $v \in V$ ein Pfad von r nach v existiert.
- $Tiefe(v)$: Abstand von v zur Wurzel r
- $Hoehe(v)$: $\max_w \{l(v, w) : w \text{ ist Blatt im Unterbaum von } v\}$
- Für Kante (v, w) heißt v Elternknoten von w und w Kindknoten von v
- Blatt: kinderloser Knoten
- geordnete Bäume:
 - Reihenfolge der Kinder steht fest
 - $first(v)$... erstes Kind von v
 - $next(v)$... nächstes Kind des Elternknotens von v (nach v)
 - $last(v)$... letztes Kind von v
 - $pre(v)$... vorheriges Kind des Elternknotens von v (vor v)

- preorder-Durchmusterung:

Idee: Erst Elternknoten, dann Kinder von links nach rechts.

$$preorder : V \rightarrow \{1, 2, \dots, n\}$$

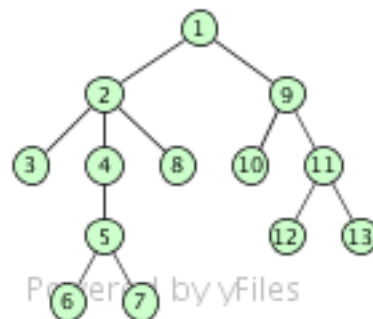
$$\text{mit } preorder(v) = 1$$

$$preorder(first(v)) = 1 + preorder(v)$$

$$preorder(next(v)) = preorder(v) + size(v)$$

$size(v)$... # Knoten im Unterbaum von v

Die Durchmusterung entspricht der DFS [Depth First Search, Tiefensuche] (bzw. der $dfsnum$).



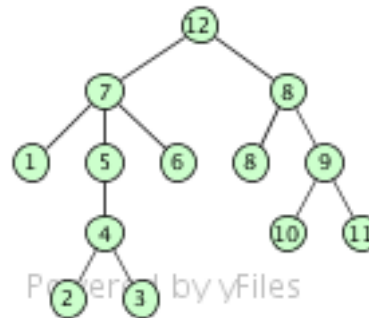
- postorder-Durchmusterung:

$$postorder : V \rightarrow \{1, 2, \dots, n\}$$

$$postord(last(v)) = postord(v) - 1$$

$$postord(next(v)) = postord(v) + size(next(v))$$

Entspricht der DFS (bzw. den *compnum*)



- Top-Down-Levelweise:

$$tdl : V \rightarrow \{1, 2, \dots, n\}, \text{ falls für alle } v, w \in V:$$

$$- \text{Tiefe}(v) < \text{Tiefe}(w) \Rightarrow tdl(v) < tdl(w)$$

$$- \text{Tiefe}(v) = \text{Tiefe}(w) \text{ und } preorder(v) < preorder(w) \Rightarrow tdl(v) < tdl(w)$$

Entspricht der BFS [Breadth First Search, Breitensuche].

- Bottom-Up-Levelweise:

$$bul : V \rightarrow \{1, 2, \dots, n\}, \text{ falls für alle } v, w \in V:$$

$$- \text{Hoehe}(v) < \text{Hoehe}(w) \Rightarrow bul(v) < bul(w)$$

$$- \text{Hoehe}(v) = \text{Hoehe}(w) \text{ und } \text{Tiefe}(v) < \text{Tiefe}(w) \Rightarrow bul(v) < bul(w)$$

$$- \text{Hoehe}(v) = \text{Hoehe}(w) \text{ und } \text{Tiefe}(v) = \text{Tiefe}(w) \text{ und } preorder(v) < preorder(w) \Rightarrow bul(v) < bul(w)$$

2 Isomorphie

2.1 Isomorphie geordneter Bäume

Seien $T_1 = (V_1, E_1)$ und $T_2 = (V_2, E_2)$ zwei geordnete Bäume, dann heißen T_1 und T_2 *isomorph*, falls es eine bijektive Abbildung $M \subseteq V_1 \times V_2$ gibt mit

- $(r_1, r_2) \in M$
- $(first(v), first(w)) \in M$ für Nicht-Blatt-Knoten $v \in V_1$ und $w \in V_2$ mit $(v, w) \in M$
- $(next(v), next(w)) \in M$ für nicht letzte Kinder $v \in V_1$ und $w \in V_2$ mit $(v, w) \in M$

Algorithmus:

1. Nummeriere Knoten beider Bäume mit preorder.
2. Bilde Paare mit Knoten gleicher Nummer. $\Rightarrow M$
3. Teste, ob M die Isomorphiebedingungen erfüllt.

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

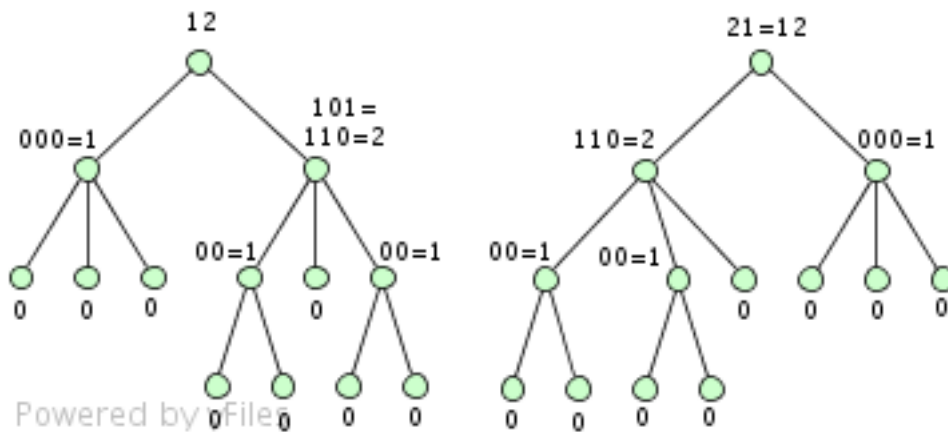
2.2 Isomorphie ungeordneter Bäume

Ungeordnete Bäume $T_1 = (V_1, E_1)$ und $T_2 = (V_2, E_2)$ sind isomorph, falls es eine bijektive Abbildung $M \subseteq V_1 \times V_2$ gibt mit

- $(r_1, r_2) \in M$
- $\forall v \neq r_1 \in V_1, w \neq r_2 \in V_2$ mit $(v, w) \in M: (\text{elter}(v), \text{elter}(w)) \in M$

Algorithmus: funktioniert levelweise

1. Bestimme für jeden Knoten seinen Level (Tiefe).
2. Jeder Knoten erhält als Markierung k -Tupel (wobei $k = \#\text{Kinder}$) wie folgt:
 - Blätter erhalten 0
 - Induktiv:
Seien k -Tupel auf Level i konstruiert. Sortiere Knoten auf Level i gemäß ihren Tupeln. \Rightarrow Listen L_1, L_2
Konstruiere k -Tupel auf Level $i - 1$ wie folgt:
 - Gehe L_1 von links nach rechts durch und übergebe die Kinderzahl nach oben zu den Eltern \Rightarrow Tupel S_1
 - Analog erzeuge S_2
 - Sortiere S_1, S_2 zu S'_1, S'_2
 - Falls $S'_1 \neq S'_2$ sind T_1 und T_2 nicht isomorph. Andernfalls ersetze erstes Tupel in S'_1 durch 1, das nächste verschiedene durch 2 usw. $\Rightarrow L_1$ neu
 - Füge Blätter von Level $i - 1$ vorne an L_1 an
 - Iteriere, falls $i \neq 1$
 - Falls Tupel für die Wurzeln identisch sind, sind T_1 und T_2 isomorph.



Korrektheit: Induktion über Level

Invariante: ...

Ordnen der Teilbäume, so dass tiefere Teilbäume weiter nach rechts kommen. \Rightarrow kanonische Ordnung

Laufzeit: $O(n)$ (Sortieren mit BucketSort)

3 Teilbaumisomorphie (Motivsuche)

Sei $T = (V, E)$ ein ungeordneter Baum. Ein Baum (W, S) heißt Teilbaum, wenn $W \subseteq V$, $S \subseteq E$.

Top-Down-Teilbaum: $eltern(v) \in W$ für alle $v \in W$, $v \neq r$

Bottom-Up-Teilbaum: $kind(v) \in W$ für alle $v \in W$, v Nichtblatt

3.1 Top-Down geordnete Teilbaumisomorphie

Definition: Ein geordneter Baum $T_1 = (V_1, E_1)$ ist isomorph zu einem geordneten Top-Down-Teilbaum von $T_2 = (V_2, E_2)$, falls es eine injektive Abbildung $M \subseteq V_1 \times V_2$ gibt mit:

- $(r(T_1), r(T_2)) \in M$
- $(first(v), first(w)) \in M$ für innere Knoten v, w mit $(v, w) \in M$
- $(next(v), next(w)) \in M$ für nicht letzte Kinder v, w mit $(v, w) \in M$

Algorithmus: Geht wie bei "normaler" Isomorphie von geordneten Bäumen.

3.2 Top-Down ungeordnete Teilbaumisomorphie

Definition:

Ein ungeordneter Baum $T_1 = (V_1, E_1)$ ist isomorph zu einem ungeordneten Top-Down-Teilbaum von $T_2 = (V_2, E_2)$, falls es eine injektive Abbildung $M \subseteq V_1 \times V_2$ gibt mit

- $(r(T_1), r(T_2)) \in M$
- $(eltern(v), eltern(w)) \in M$ für Nichtwurzeln v, w mit $(v, w) \in M$

Problem: Wo anfangen? Nicht bottom-up!

Idee:

Falls Unterbäume zueinander isomorph sind, können innere Knoten aufeinander abgebildet werden.

$T_1(v)$ bezeichne den Teilbaum von T_1 mit Wurzel v .

$T_2(v)$ bezeichne den Teilbaum von T_2 mit Wurzel v .

$Iso(T_1, T_2)$ entscheidet, ob T_1 top-down isomorph zum Teilbaum von T_2 ist:

```

if |T1| = 1 und |T2| ≥ 1 then ok
else
  v ← r(T1), w ← r(T2)
  baue Graph G = (V, E) aus Kindern {v1, ..., vp}, {w1, ..., wq}, E = ∅
  for all vi ∈ {v1, ..., vp}, wj ∈ {w1, ..., wq} do
    if Iso(T1(vi), T2(wj)) then E ← E ∪ {vi, wj}
  endfor
  berechne maximales Matching X für G (G bipartit)
  if |X| = p then ok
  else nicht ok
endif

```

Korrektheit:

Klar, da das Matching jedem Teilbaum mit Wurzel v_i einen eindeutigen teilbaumisomorphen Partner zuordnet. Somit ist $T_1(v)$ auch in $T_2(w)$ enthalten.

Laufzeit: Matching geht in $\mathcal{O}(m \cdot \sqrt{n})$ bei n Knoten und m Kanten. Insgesamt:

$$\begin{aligned} & \mathcal{O}\left(\sum_{v \in V_1} \sum_{w \in V_2} |\text{kinder}(v)| \cdot |\text{kinder}(w)| \cdot \sqrt{|\text{kinder}(w)|}\right) \\ &= \mathcal{O}\left(\sum_{v \in V_1} |\text{kinder}(v)| \cdot \sum_{w \in V_2} |\text{kinder}(w)|^{\frac{3}{2}}\right) \\ &= \mathcal{O}\left(\sum_{v \in V_1} |\text{kinder}(v)| \cdot \left(\sum_{w \in V_2} |\text{kinder}(w)|\right)^{\frac{3}{2}}\right) \\ &= \mathcal{O}\left(|T_1| \cdot |T_2|^{\frac{3}{2}}\right) \end{aligned}$$

3.3 Bottom-Up geordnete Teilbaumisomorphie**Definition:**

T_1 ist isomorph zum bottom-up Teilbaum eines geordneten Baums T_2 , falls es eine injektive Abbildung $M \subseteq V_1 \times V_2$ gibt mit

- für alle Nichtblattknoten v, w mit $(v, w) \in M$ ist $(\text{first}(v), \text{first}(w)) \in M$
- für alle nicht letzten Knoten v, w mit $(v, w) \in M$ ist $(\text{next}(v), \text{next}(w)) \in M$
- für alle Blätter $v \in V_1$ mit $(v, w) \in M$ ist v ein Blatt in T_2

Idee:

$\text{Iso}(T_1, T_2(w))$ für alle $w \in V_2 \rightarrow$ Laufzeit $\mathcal{O}(n^2)$

Verfeinerter Test: Führe $\text{Iso}(T_1, T_2(w))$ nur dann aus, wenn $\text{Hoehe}(T_1) = \text{Hoehe}(T_2(w))$ und $\text{Groesse}(T_1) = \text{Groesse}(T_2(w))$. \rightarrow Laufzeit $\mathcal{O}(|\mathcal{T}_\in(\sqsupset)|)$

Beobachtung:

$T_2(w)$ und $T_2(w')$ sind disjunkt, falls w und w' betrachtet werden.

\Rightarrow Laufzeit: $\mathcal{O}\left(\sum_{w \text{ betr.}} |T_2(w)|\right) = \mathcal{O}(|T_2|) = \mathcal{O}(n)$

3.4 Größtmögliche gemeinsame Top-Down-Teilbäume

Idee: Ähnlich wie bei ungeordneter Teilbaumisomorphie

Top-Down: Sind zwei Knoten isomorph in der Abbildung, dann auch ihre Elterknoten.

Rekursiv: $TBGroesse(T_1, T_2)$ (berechne größtmögliche siehe entsprechende Teilbäume)

```

if  $|V_1| = 1$  und  $|V_2| \geq 1$  oder umgekehrt then
    return 1
else
     $v \leftarrow r(T_1)$ ,  $w \leftarrow r(T_2)$  mit Kinder  $v_1, v_2, \dots, v_p$  von  $v$  und  $w_1, \dots, w_q$  von  $w$ 

```

```

betrachte bipartiten Graphen aus Kindern mit folgender Kantenmenge:
for all  $1 \leq i \leq p, 1 \leq j \leq q$  do  $c(v_i, w_j) = TBGroesse(v_i, w_j)$ 
berechne maximales gewichtetes (Summe der Kantengewichte im Matching maximal)
    Matching
endif

```

Laufzeit:

Für gewichtetes bipartites Matching ist $\mathcal{O}(n(m + n \log n))$ für ganzzahlige Gewichte.
 $\Rightarrow \mathcal{O}((n_1 + n_2)(n_1 \cdot n_2 + (n_1 + n_2) \cdot \log(n_1 + n_2)))$

Korrektheit:

induktiv

Damit T_1, T_2 bestmöglich übereinstimmen, muss die bestmögliche Übereinstimmung der Kinderbäume gefunden werden.

3.5 Teilgraphisomorphie

Teilgraphisomorphie von $G_1 = (V_1, E_1)$ in $G_2 = (V_2, E_2)$ ist eine injektive Abbildung $M \subseteq V_1 \times V_2$ mit $\forall v_i, v_j \in V_1$
 $\forall w_i, w_j \in V_2$:

$$(v_i, w_i) \in M \wedge (v_j, w_j) \in M \Rightarrow (v_i, v_j) \in E_1 \Rightarrow (w_i, w_j) \in E_2$$

Wir wissen schon: NP-vollständig (z.B. Clique, Hamiltonkreis)

Induzierte Teilgraphisomorphie:

$$(v_i, v_j) \in E_1 \Leftrightarrow (w_i, w_j) \in E_2$$

Clique ist induzierte Teilgraphisomorphie.

Hamiltonkreis ist keine induzierte Teilgraphisomorphie.

Isomorphie maximaler gemeinsamer (induzierter) Teilgraphen:

Gegeben: G_1, G_2

(S_1, S_2, M) mit S_1 (induzierter) Teilgraph von G_1 , S_2 (induzierter) Teilgraph von G_2 und M Isomorphie von S_1, S_2 , so dass es keine S'_1, S'_2 gibt, dass S_1 echter Teilgraph von S'_1 und S_2 echter Teilgraph von S'_2 ist.

Finde alle maximalen induzierten gemeinsamen Teilgraphen von G_1, G_2 :

- Bilde Produkt $G_1 \times G_2$ und suche darin alle maximalen Cliques
- Produkt aus G_1, G_2 ist $G = (V, E)$ mit $V \leftarrow V_1 \times V_2$ und

$$E \leftarrow \{((v_i, w_i), (v_j, w_j)) \in V \times V \mid v_i \neq v_j, w_i \neq w_j, (v_i, v_j) \in E_1, (w_i, w_j) \in E_2\} \cup \\ \{((v_i, w_i), (v_j, w_j)) \in V \times V \mid v_i \neq v_j, w_i \neq w_j, (v_i, v_j) \notin E_1, (w_i, w_j) \notin E_2\}$$

Definition:

Sei $C = (V, E)$ ein vollständiger Teilgraph von $G_1 \times G_2$, dann ist die Projektion von C auf G_1 derjenige Teilgraph von G_1 , der durch diejenigen Knoten v von G_1 induziert wird, für die $(v, w) \in V$ für einen Knoten w von G_2 .

Satz:

Seien $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ zwei Graphen.

1. Sei (S_1, S_2, M) Isomorphie maximaler gemeinsamer induzierter Teilgraphen von G_1 und G_2 , dann gibt es eine maximale Clique in $G_1 \times G_2$, deren Projektionen auf G_1 und G_2 die Teilgraphen S_1 und S_2 ergibt.
2. Sei C maximale Clique auf $G_1 \times G_2$ mit Projektionen S_1 bzw. S_2 von C auf G_1 und G_2 , dann gibt es Isomorphie maximaler gemeinsamer induzierter Teilgraphen von G_1 und G_2 der Form (S_1, S_2, M) .

Beweis:

Sei $C = (X, E)$ Teilgraph von $S_1 \times S_2$ mit $X = \{(v, w) \in V_1 \times V_2 | (v, w) \in M\}$.

Behauptung: C ist eine Clique.

Beweis: M ist bijektiv \Rightarrow für alle $(v_i, w_i), (v_j, w_j) \in X$ gilt für $(v_i, w_i), (v_j, w_j)$ mit $(v_i, w_i) \neq (v_j, w_j)$ auch $v_i \neq v_j$ und $w_i \neq w_j$.

Die M Isomorphie, ist $(v_i, v_j) \in S_1 \Leftrightarrow (w_i, w_j) \in S_2$, falls $(v_i, w_i), (v_j, w_j) \in M$.

Für alle Knotenpaare $(v_i, w_i), (v_j, w_j) \in X$ gibt es eine Kante in C wegen Produktbildung.

Damit ist C eine Clique.

Da (S_1, S_2, M) Isomorphie, liefert Projektion von C auf G_1 den induzierten Teilgraph S_1 und die Projektion von C auf G_2 den induzierten Teilgraph S_2 .

Weil M maximale Teilgraphisomorphie ist, folgt auch Maximalität von C .

3.6 Isomorphieproblem für allgemeine Graphen

Backtracking

Erschöpfende Suche im Raum aller Lösungen.

Suche nach Menge von Konfigurationen, Ordnung der Menge.

Schrittweise Aufbau der Lösung:

$(a_1, \dots, a_k) \rightarrow (a_1, \dots, a_{k+1})$ (probiere alle Lösungen für a_{k+1})

Backtracking, falls keine Erweiterung der partiellen Lösung möglich ist.

Bei Isomorphie: Starte mit leerer Abbildung M und erweitere sie schrittweise.

Beobachtung:

Sei $M \subseteq V_1 \times W_1$ eine Isomorphieabbildung auf den durch V und W induzierten Teilgraphen G_1 bzw. G_2 , dann kann M folgendermaßen erweitert werden:

$\forall v \in V \setminus V_1, w \in W \setminus W_1: M \cup \{(v, w)\}$ ist Isomorphieabbildung zwischen $G(V_1 \cup \{v\})$ und $G'(W_1 \cup \{w\}) \Leftrightarrow$

$\forall x \in V_1, y \in V_2, (x, y) \in M:$

$(x, v) \in E \Leftrightarrow (y, w) \in E'$ und

$(v, x) \in E \Leftrightarrow (w, y) \in E'$

4 Teilgraphen mit besonderen Eigenschaften

Sei $G = (V, E)$ ein Graph

Sei $V' \subseteq V$, dann bezeichne $G(V')$ den durch V' induzierten Teilgraphen $G(V') = (V', E')$ mit $E' = \{(v_1, v_2) | v_1, v_2 \in V' \wedge (v_1, v_2) \in E\}$

Wenn G dicht ist, dh. viele der n^2 mögliche Kanten existieren, sind dünne Teilgraphen (mit nur $\mathcal{O}(n)$ oder $\mathcal{O}(\log n)$ Kanten) gesucht, die gewisse Eigenschaften haben.

1. Soziale Netzwerke
Suche nach dichten Teilbereichen sogenannten Cluster.
2. Minimale Spannbäume (MST)
3. Kommunikationsnetzwerk
Kommunikation erfolgt über kürzeste Wege; Qualität sinkt mit Zahl der Zwischenstationen; aber alle Teilnehmer direkt miteinander verbinden \Rightarrow "zu teuer"
Gesucht: Teilgraph, so dass die Umwege nicht zu groß sind
Extremfall: Teilgraph soll Spannbaum sein
4. Zusätzlich Annahme:
Kommunikation exklusiv auf einer Kante, keine Mehrfachnutzung
Gesucht: Teilgraph, der möglichst viele kantendisjunkte Pfade erhält

4.1 Aufgabe des Algorithmikers

1. Formalisierung der Aufgabe
2. Lösen!
 - (a) Gibt es *immer* einen solchen Teilgraphen (Existenz)
 - (b) Komplexität
 - i. Reduktion von einem NP harten Problem auf ein neues Problem \Rightarrow NP hart
 - ii. Angabe eines *effizienten* Algo (polynomiell)

Beispiel Formalisierung (3.)

1. $\sum_{v \in V} \sum_{w \in V} d_G(v, w)$ soll klein sein
2. $\max \sum_{v \in V} \sum_{w \in V} d_G(v, w)$ soll klein sein
3. $\text{stretch}(v, w) = \frac{d_{G'}(v, w)}{d_G(v, w)}$

Formalisierungen sind nicht immer eindeutig zu finden. Das ist insbesondere bei Problem 1. der Fall

4.2 Clustern von Graphen

Ein Clustering eines Graphen ist eine Partitionierung von V .

Definition: Partitionierung

Eine Partitionierung ist die Untergliederung von V in disjunkte Teilmengen V_1, \dots, V_z , so dass $\bigcup V_i = V$. V ist eine Partition/Cluster.

Wir suchen Cluster, in denen die Knoten innerhalb des Clusters nahe beieinander sind, und die Cluster nur dünn vernetzt sind.

Sei nun G_c ein *Metagraph*, in dem jeder Cluster V_i durch einen Knoten repräsentiert wird, und zwei Knoten $v_i, v_j, i \neq j$ miteinander verbunden werden, wenn es mindestens eine Kante $e = (x, y)$ mit $x \in V_i, y \in V_j$ gibt. Wir fordern nun, dass die Anzahl dieser Kanten $|E(G_c)|$ klein ist.

"Kurzer Abstand" wird wie folgt formalisiert:

Definition: Exzentrizität

Die Exzentrizität $\text{ecc}(v)$ eines Knotens ist definiert als:

$$\text{ecc}(v) := \max_{w \in V} \{d_G(v, w)\}$$

Definition: Radius

Der Radius eines Graphen ist

$$Rad(G) := \min_{v \in V} \{ecc(v)\}$$

Bemerkung: Durchschnitt eines Graphen ist die maximale Exzentrizität.

Theorem: Partitionierung

Gegeben ein ungewichteter Graph $G = (V, E)$ mit n Knoten und eine Zahl $k \geq 1$. Es existiert eine Partitionierung $C = \{V_1, \dots, V_z\}$, so dass

1. $Rad(V_i) \leq k - 1 \quad \forall V_i \in C$
2. $|E(G_C)| \leq n^{1+\frac{1}{k}}$

Definiere $N(V')$ für $V' \subseteq V$ als die Menge der Nachbarn von Knoten aus V' , die selber nicht in V' enthalten sind.

Der folgende Algorithmus leistet das Gewünschte:

```

Setze  $C \leftarrow \emptyset$ 
while  $V \neq \emptyset$  do
  Wähle einen beliebigen Knoten  $v$ 
  Setze  $V' \leftarrow \{v\}$ 
  while  $|N(V')| > n^{\frac{1}{k}} |V'|$  do
    setze  $V' \leftarrow V' \cup N(V')$ 
  endwhile
  Setze  $C \leftarrow C \cup V'$ ,
       $V \leftarrow V \setminus V'$ 
endwhile

```

Zeigen Sie, dass der G_C Radius eines Clusters höchstens $k - 1$ ist.

Nach der ersten while-Schleife: $|V'| > n^{\frac{1}{k}}$

Nach der i -ten Schleife: $|V'| > n^{\frac{i}{k}}$ (oder wir haben gestoppt)

Nach der k -ten Schleife: $|V'| > n^{\frac{k}{k}} = n$ Widerspruch!

Anzahl Interclusteredges $|E(G_c)|$

Wenn die while-Schleife in Zeile 3 (der while-Schleife) stoppt, dann hat V' höchstens $n^{\frac{1}{k}} |V'|$ viele Nachbarn. Selbst wenn dieser Nachbar einen Cluster bildet, ist dadurch die Zahl der von V' gebildeten Kanten in G_c limitiert.

$$\begin{aligned}
 |E(G_c)| &\leq \sum_{V_i \in C} n^{\frac{1}{k}} |V_i| \\
 &= n^{\frac{1}{k}} \underbrace{\sum_{V_i \in C} |V_i|}_{=n \text{ (Part.)}} \\
 &= nn^{\frac{1}{k}} = n^{1+\frac{1}{k}}
 \end{aligned}$$

In dieser Partitionierung ist $|E(G_c)|$ klein, oft würde man lieber die Menge aller Interclusterkanten beschränken.

$$\{(x, y) | x \in V_i, y \in V_j, i \neq j, (x, y) \in E(G)\}$$

Wie nennen diese Menge $E_{out}(G, C)$.

Theorem:

In jedem ungewichteten Graphen G und für einen Parameter $x \geq 1$ gibt es eine Partitionierung C , so dass

1. $Rad(V_i) \leq x \log m \forall V_i \in C$
2. $|E_{out}(G, C)| \leq \frac{m}{x}$

Der PARTITON-Algorithmus liefert die Partitionierung nach genau diesem Theorem.

```

Setze  $C \rightarrow \emptyset$ 
while  $V \neq \emptyset$  do
  1. Wähle einen beliebigen Knoten  $v \in V$ 
  2. Setze  $V' \rightarrow \{v\}$ 
  3. while  $\frac{|E_{out}(V')|}{|E_{in}(V')|} > \frac{1}{x}$  do
    Setze  $V' \rightarrow V' \cup N(V')$ 
  endwhile
  4. Setze  $C \rightarrow C \cup V'$  und  $V \rightarrow V \setminus V'$ 
endwhile

```

Beweis

Wir beweisen zuerst, dass $Rad(V_i) \leq x \cdot \log m \forall V_i \in C$.

Sei $E_{in}(j)$ die Anzahl der Intraclusterkanten nach der j -ten Schleife in Zeile 5 (falls der Algo nicht vorher stoppt)

1. $E_{in}(0) = 1 \geq \frac{1}{x}$
2. $E_{in}(1) \geq 1 + \frac{1}{x}$
3. $E_{in}(2) \geq E_{in}(1) + \frac{E_{in}(1)}{x} = 1 + \frac{1}{x} + \frac{1 + \frac{1}{x}}{x} = 1 + 2 \cdot \frac{1}{x} + (\frac{1}{x})^2$
- j. $E_{in}(j) \geq E_{in}(j-1)(1 + \frac{1}{x}) = (1 + \frac{1}{x})^j$

$$(1 + \frac{1}{x})^j \geq m ?$$

$$j \cdot \ln(1 + \frac{1}{x}) \geq \ln m$$

$$j \geq \frac{\ln m}{\ln(1 + \frac{1}{x})} \geq x \cdot \ln m$$

$$\ln(1 + \frac{1}{x}) \geq \frac{1}{x}$$

$$\ln(1 + \frac{1}{x}) \leq \frac{1}{x}$$

Zwischenspiel:

$$\begin{aligned} \ln(1 + \frac{1}{x}) &\leq \frac{1}{x} \\ \lim_{n \rightarrow \infty} (1 + \frac{1}{x})^n &= e \\ (1 + \frac{1}{2})^n &< e \end{aligned}$$

$$\ln(1 + \frac{1}{x}) < \frac{1}{x}$$

Wir beweisen, dass $|E_{out}(G, C)| \leq \frac{m}{x}$

Für jeden Cluster $V_i \in C$ gilt:

$$E_{out}(V_i) \leq \frac{1}{x} \cdot E_{in}(V_i)$$

$$E_{out}(G, C) = \sum_{V_i \in C} E_{out}(V_i)$$

$$E_{out}(G, C) = \frac{1}{x} \underbrace{\sum_{V_i \in C} E_{in}(V_i)}_m = \frac{m}{x}$$

□

In einem anderen Szenario suchen wir vielleicht eine Partitionierung in nicht zu viele, nicht zu große Cluster.

⇒ Formalisierung

Definition: k -dominating Set

Eine Teilmenge D von Knoten ist ein k -dominating Set eines (ungewichteten) Graphen G , falls für alle $v \in V$ gilt: $\exists d \in D$ mit $d_G(v, d) \leq k$.

Wenn eine solche Menge gefunden wurde, wird C wie folgend definiert:

1. Berechne für jedes Paar $v \in V - D$ und $d \in D$: $d_G(v, d)$
2. $V_i = \{d_i\} \cup \left\{ v_i \in V - D \mid d_G(d_j, v_j) = \min_{d_k \in D} \{d_G(d_k, v_j)\} \right\}$

Hier fehlt eine Tie-Breaking Rule.

Theorem

In jedem zusammenhängenden Graphen und für jedes $k \geq 1$ gibt es ein k -dominating Set mit weniger als $\max \left\{ 1, \frac{n}{k+1} \right\}$ Knoten.

Beweis Sei T ein beliebiger Spannbaum von G mit Wurzel w und sei h seine Tiefe bzgl. w .

Fall 1: Wenn $k \geq h$, bildet w ein einelementiges k -dominating Set.

Fall 2: $k < h$

Wir bilden Mengen T_i :

$$T_i = \{v_j \in V \mid d_T(w, v) = i\}$$

Jede Menge T_i enthält also alle Knoten, in T denselben Abstand zu w haben.

Nun verschmelzen wir diese Teilmengen wie folgend

$$D_j = \bigcup_{i \geq 0} T_{i+j(k+1)}$$

also T_i und jedes $(k+1)$ ste darauffolgende Level. Dadurch entstehen $k+1$ paarweise disjunkte Mengen
⇒ Partitionierung

Da jeder Knoten nur in einer Menge D_j liegt, kann mind. eine Menge D_k höchstens $\left\lfloor \frac{n}{k+1} \right\rfloor$ Knoten enthalten (Schubladenprinzip). Diese Menge D_k bildet ein k -dominating Set.

Zusammenfassung: Partitionierungen

1. $\text{Rad}(V_i) \leq k - 1$
 $|E(G_C)| \leq n^{1+\frac{1}{k}}$ BASIC_PART
2. $\text{Rad}(V_i) \leq x \cdot \log m$
 $|E_{out}(G, C)| \leq \frac{m}{x}$ PARTITION
3. $\text{Rad}(V_i) \leq k$
 $|C| \leq \frac{n}{k+1}$ Spannbaummethode

4.3 Spanner

Gesucht sind möglichst dünne Teilgraphen die für alle urspr. Kanten in G nur kurze Umwege erzeugen

Formalisierung**Definition:** k-Spannerk-Teilgraph $G' = (V, E')$ eines Graphen $G = (V, E)$ ist ein k-Spanner genau dann, wenn $\forall e = (x, y) \in E$ gilt:

$$\frac{d_{G'}(x, y)}{d_G(x, y)} \leq k$$

$$\text{Stretch}_{G'}(x, y) := \frac{d_{G'}(x, y)}{d_G(x, y)}$$

TheoremFür jeden ungewichteten Graphen G und für jedes $k \geq 1$ existiert ein $\mathcal{O}(k)$ -Spanner mit höchstens $n^{1+\frac{1}{k}}$ Kanten.**Beweis**Sei C eine Partitionierung, die mit Algorithmus BASIC_PART und Parameter k berechnet wurde. Sei v_i jeweils der Startknoten von $V_i \in C$. Wir legen jetzt in Cluster V_i einen in v_i gewurzelten, Distanz erhaltenden Spannbaum. G' besteht aus diesen Spannbäumen, vereinigt mit der folgenden Menge:Für je zwei Partitionen $V_i, V_j, i \neq j$, die mindestens eine Kante $e = (x, y)$ mit $x \in V_i \wedge y \in V_j$, teilen, wählen wir eine dieser Kanten und fügen sie zu E' hinzu.**4.4 Spannbäume mit niedrigem, durchschnittlichem Stretch**Ein gewurzelter, Distanz erhaltender Spannbaum erhält zwar für w optimale Routen, ist aber nicht unbedingt optimal für alle Knoten. Wir definieren daher für jeden Spannbaum T in G :

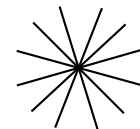
$$Q(T) := \sum_{e=(x,y) \in E} d_T(x, y) \quad (\text{Tree-Distance-Sum})$$

Dann ist der durchschnittliche Stretch $S_{avg}(G, T) = \frac{Q(T)}{m}$. Wir definieren

$$S_{opt}(G) := \min_{T \in \mathcal{T}} \{S_{avg}(G, T)\}$$

wobei \mathcal{T} die Menge aller Spannbäume in G beschreibt. Der einfachste Spannbaum ist ein Stern:

$$Q(T) = \underbrace{\sum_{e \in T} 1}_{n-1} + \underbrace{\sum_{e \in E \setminus T} 2}_{m-(n-1)}$$

**Ring**

$$\frac{Q(T)}{m} = 2 - \frac{2}{n}$$

$$S_{avg}(T^*) \leq \frac{1}{n} \left(\underbrace{(n-1) \cdot 1}_{e \in T} + \frac{\omega(C_n) - \omega(e)}{\omega(e)} \right)$$

$$= \frac{2n-2}{n} = 2 - \frac{2}{n}$$

Denn es gilt:

$$\frac{\omega(C_n) - \omega(e)}{\omega(e)} \leq \frac{(n-1) \cdot \omega(e)}{\omega(e)} = (n-1)$$

Gitter

Der folgende Baum hat in einem $\sqrt{n} \times \sqrt{n}$ Gitter einen S_{avg} von $\mathcal{O}(\log n)$.

Anmerkung: Man kann zeigen, dass für diese Gitter alle Spannbäume einen S_{avg} in $\Omega(\log n)$ haben. $\Rightarrow S_{opt} \in \Theta(\log n)$

Problem P_{QT}

Gegeben: ungewichteter graph G

Gesucht: Spannb Baum T , der $Q(T)$ minimiert.

$$Q(T) = \sum_{e=(x,y) \in E} d_T(x,y) \text{ mit } d_T = \text{tree distance}$$

P_{QT} ist NP-hart, dh, alle NP-harten Probleme können darauf (polynomiell) reduziert werden.

Dazu reicht es, ein einziges bekanntermaßen NP-hartes Problem auf P_{QT} zu reduzieren.

Das heißt, wir geben einen Algorithmus in P an (polynomielle Laufzeit), der aus jeder Instanz des NP-harten Problems P' eine Instanz von P_{QT} herstellt, so dass die Lösung der Instanz I von P' in Polynomzeit aus der Lösung der Instanz von P_{QT} berechnet werden kann:

$$\forall I(P') \xrightarrow[\text{in } P]{\text{Algo}} I(QT) \xrightarrow{\text{? Lösungszeit}} \text{Lösung von } I(QT) \xrightarrow[\text{in } P]{\text{Algo}'}$$

Theorem:

Das Lösen von P_{QT} ist nNP-hart.

Beweis: Reduktion Exact-3-Cover (P_{QT})

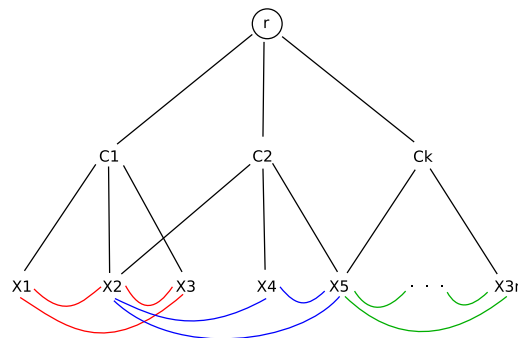
Exact-3-Cover

Gegeben eine Menge von $3n$ Elementen $x = \{x_1, x_2, \dots, x_{3n}\}$ und eine Menge C von Tripeln $C_j \subset X \times X \times X$ (ohne Mehrfachelemente)

Gesucht: Eine Teilmenge $C' \subset C$ mit $|C'| = n$ und $\bigcup_{C_i \in C'} C_i = X$

Annahme: Jedes x_i ist in mindestens einem C_j enthalten (sonst Lösung trivial).

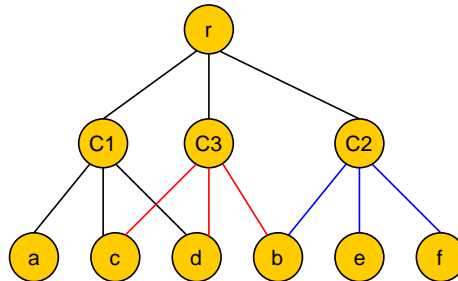
Wir bauen nun aus ansonsten beliebiger Instanz $I(P')$ den folgenden Graphen:



$$V = \{r\} \cup \{c_i, 1 \leq i \leq |C|\} \cup \{x_i, 1 \leq j \leq 3n\}$$

Kanten:

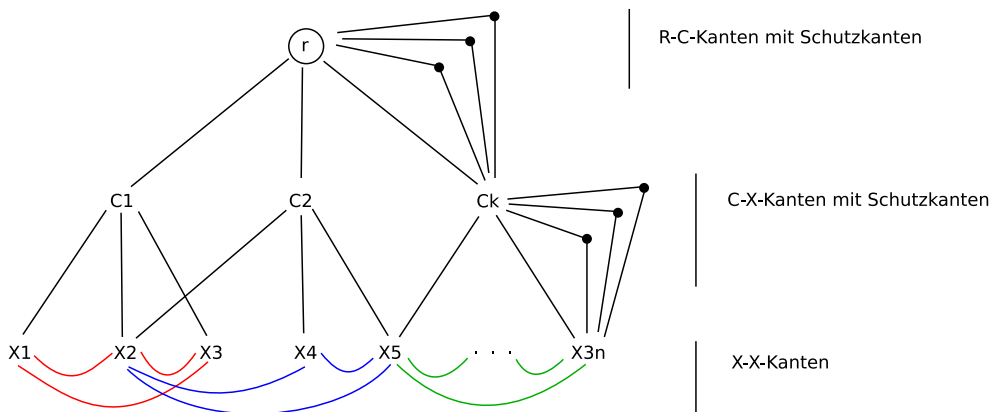
1. r ist mit allen C -Knoten verbunden
2. Jeder C -Knoten mit "seinen" X -Knoten
3. Jeder X -Knoten mit allen X -Knoten mit denen er mindestens einmal zusammen in einem Triple c_i enthalten ist.



Idee:

Ein bezüglich $Q(T)$ minimaler Spannbaum T^* nur R-C und C-X-Kanten benutzen sollte. Und zwar so, dass, falls $I(P')$ ein Exact-3-Cover hat, von jedem C -Knoten entweder alle oder keine X-C-Kante in T^* enthalten ist. Wenn das so ist, besteht die Lösung von $I(P')$ aus der Teilmenge an Tripeln C_i , deren C-X-Kanten in T^* enthalten sind.

Problem: Ohne Schutzkanten könnten auch $X - X$ -Kanten in T^* enthalten sein

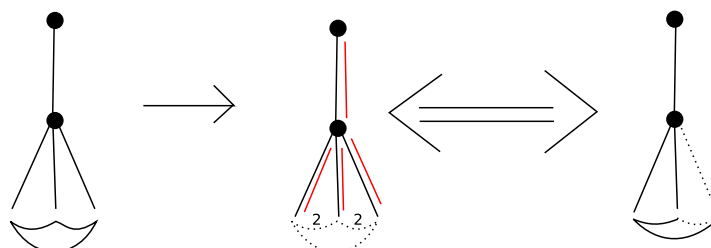


Sei j die Anzahl von Schutzkanten auf jeweils einer R-C- oder C-X-Kante

Frage: Wie hoch muss j sein, um X-X-Kanten in T^* auszuschließen.

Sei also $e = (x_i, x_j)$ eine Kante in einem optimalen Baum T^*

Fall 1: x_i, x_j sind nur in einem Tripel C_k enthalten.

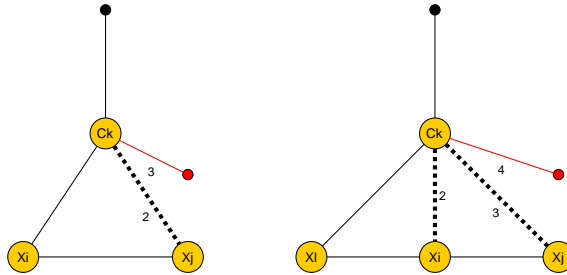


Die von c_k ausgehenden C-X-Kanten haben mindestens $d_T \geq 2/3$.

Ihre Schutzkanten haben dementsprechend mindestens $d_T \geq 3/4$

Widerspruch! Wenn $j > 2$, wäre $Q(T')$ kleiner als $Q(T^*)$, wenn wir in T' $x_i x_j$ über die C-X-Kanten anbinden

Fall 2: Sei nun in der vereinigten Nachbarschaft von x_j, x_i mehr als ein C-Knoten. damit gibt es den Kreis:

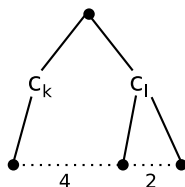
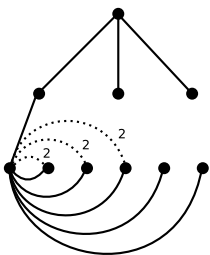
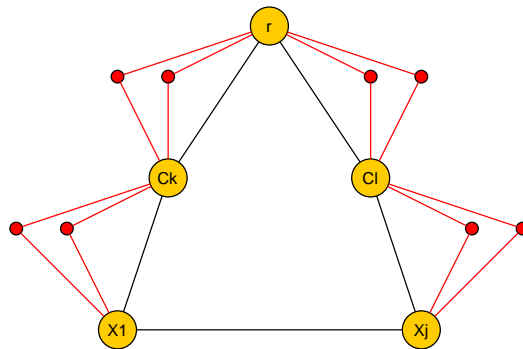


$x_j - c_k - r - c_l - x_j - x_i$
 Mind. eine Kanten davon ist nicht in T
 $\Rightarrow \geq d_T \geq 4$
 \Rightarrow deren Schutzkanten haben $\geq d_T \geq 5$
 können haben d_T von 2!
 $\Rightarrow -3$ pro Schutzkante

Da T^* nach Annahme optimal ist, muss der Beitrag den diese Schutzkanten zu $Q(T^*)$ beitragen, durch die kleinere d_T von anderen Kanten ausgeglichen werden.

\Rightarrow Beobachtung: nur X-X-Kanten profitieren davon, dass X-X-Kanten im Baum sind

\Rightarrow Bester Fall: Es gibt einen X-Knoten der mit allen anderen X-Knoten direkt verbunden ist. Wenn dieser Knoten mit seinen Kanten im Baum ist, haben alle anderen X-X-Knoten $d_T \geq 2$:



Best-Case für X-X

“worst-case” für X-X-Kanten, wenn keine X-X-Kanten in T sind

Anzahl möglicher X-X-Kanten: $\frac{3n(3n-1)}{2}$

Best-Case-Beitrag zu $Q(T) \Rightarrow 2 \cdot \frac{3n(3n-1)}{2}$

Worst-Case-Beitrag zu $Q(T)$ (wenn T keine X-X-Kanten enthält) $\Rightarrow 4 \cdot \frac{3n(3n-1)}{2}$

\Rightarrow Da die Schutzkanten mit -3 von dem Enthalten "ihrer" Kante in T profitieren, reichen $j > \frac{3n(3n-1)}{2}$ viele aus, um X-X-Kanten in einem optimalen Spannbaum auszuschließen.

Wie sieht $Q(T)$ in einer solchen Instanz aus?

Alle R-C-Kanten sind in T^* : $|C| \cdot 1 + \underbrace{j \cdot |C| \cdot 2}_{\text{Schutzkanten}}$

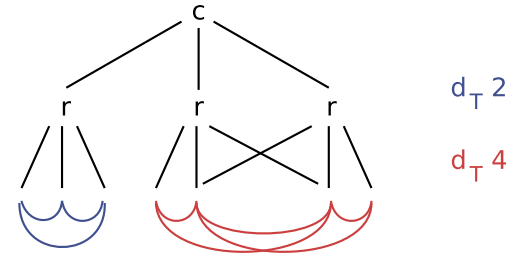
$3n$ C-X-Kanten in T^* : $3n \cdot 1 + \underbrace{j \cdot 3n \cdot 2}_{\text{Schutzkanten}}$

$3|C| - 3n$ C-X-Kanten $\notin T$: $3(|C| - n) \cdot 3$

Nicht variable Anteil hängt nur von der Instanzgröße ($|C|$) ab.

X-X-Kanten: $3n$ X-X-Kanten haben $d_T 2 \Leftrightarrow \exists$ exact-3-cover in $I(P')$

Damit ist die Reduktion gezeigt. □



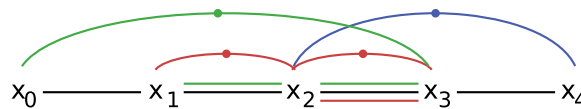
4.5 Teilgraphen, die Zusammenhänge enthalten

Definition: Zusammenhang (Connectivity)

Ein Graph $G = (V, E)$ heißt k -fach (kanten-)zusammenhängend, wenn es *keine* Menge X' von $k - 1$ Knoten (Kanten) gibt, so dass $G \setminus X'$ *nicht* zusammenhängend ist.

Äquivalent:

Ein Graph ist dann k -fach kantenzusammenhängend genau dann, wenn es zwischen allen Knotenpaaren $x, y \in V$ mindestens k kantendisjunkte Pfade gibt.



Beweis:

\Rightarrow Nimm $x, y \Rightarrow \exists \geq k$ kantendisjunkte Pfade, wenn weniger als k Kanten entfernt werden, bleibt mindestens ein Pfad erhalten.

\Leftarrow Wenn es zwischen einem Knotenpaar höchstens k kantendisjunkte Pfade gibt, gibt es eine Kantenmenge $X', |X'| \leq k$, so dass $G \setminus X'$ zerfällt.

Eine solche Kantenmenge heißt Cut.

Beweis: Induktion

$k = 1$: Zwischen einem Knotenpaar x_0, x_l gibt es nur einen kantendisjunkten Pfad $P(x_0, x_l) : x_0 - x_1 - x_2 - \dots - x_l$

Wenn keine der Kanten ein P einen Cut bildet, dann gibt es für je zwei aufeinander folgende Knoten $x_i - x_{i+1}$ jeweils eine alternative Verbindung.

Sei P_i die alternative Verbindung für die Kante (x_{i-1}, x_i) .

Wir beschreiben Pfad P_i als m -dimensionalen Vektor:

0	1	0	...	1
0	1	2	...	$m - 1$

der eine 1 enthält, wenn die Kante mit Index x_j Element von P_i ist.

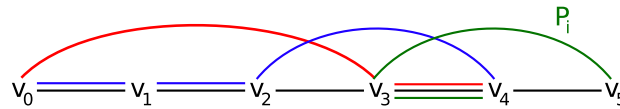
Behauptung: Die XOR-Vereinigung aller P_i enthält einen Pfad von x_0 nach x_l der kantendisjunkt zu $P(x_0, x_l)$ ist, im Widerspruch zur Annahme.

Induktion:

P_1 ist kantendisjunkter Pfad zum Teilpfad $P(x_0, x_1) \in P(x_0, x_l)$

Induktionsvoraussetzung:

Sei $\bar{P}_{i-1} = \text{XOR}_{k=1}^{i-1} P_k$ von v_0 nach v_{i-1} . P_i ist ein Alternativpfad von v_{i-1} bis v_i .

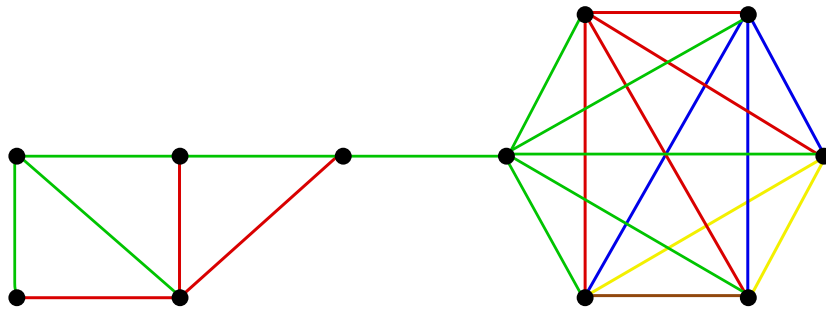


Wir nummerieren die Knoten auf den Pfaden von v_i nach v_0 und von v_i nach v_l ausgehend durch. Sei e die letzte gemeinsame Kante auf diesen Pfaden, dann fällt sie durch die XOR-Vereinigung weg und es bleiben Pfade von v_0 zum Endknoten von e mit dem höheren Index v_z und von v_z nach v_i .

Da der Pfad von v_0 bis v_{i-1} kantendisjunkt zum ursprünglichen Pfad war und P_i auch, ist auch die XOR-Vereinigung kantendisjunkt.

Zu zeigen: Falls es ein Knotenpaar mit höchstens k kantendisjunkten Pfaden gibt, gibt es auch ein Cutset mit C mit $|C| \leq k$.

$k = i$: Nimm einen Pfad P zwischen u, v aus dem Graphen. Bilde nach Induktionsvoraussetzung ein Cutset C' mit $|C'| = i - 1$. Wähle für Pfad P in $G \setminus C'$ eine Cutkante.



Wir definieren als maximal aufspannenden Wald $F = (V, E')$ in einem Graphen G die Vereinigung von Spannbäumen aller Zusammenhangskomponenten. Besteht eine Zusammenhangskomponente aus nur einem Knoten, trägt sie nicht zum Wald bei. Sei nun $F_i = (V, E_i)$ der maximal aufspannende Wald im Graphen $G \setminus \underbrace{E_0 \cup E_1 \cup \dots \cup E_{i-1}}_{E_{i-1} := \bigcup_{k=1}^{i-1} E_k}$.

Theorem:

Der Teilgraph $G_i = (V, \bar{E}_i)$ erfüllt für alle $x, y \in V$: $\lambda(x, y, G_i) \geq \min \{ \lambda(x, y, G), i \}$, wobei $\lambda(x, y, G_i)$ die maximale Anzahl kantendisjunkter Pfade zwischen x, y in G_i angibt.

Beweis per Induktion:

$i = 1$: T_1 ist maximal aufspannender Wald \Rightarrow alle Knoten einer Zusammenhangskomponente sind miteinander verbunden.

Sei nun $x, y \in V$ und i so, dass $\lambda(x, y, G_i) \geq \min \{\lambda(x, y, G), i\}$ und $\lambda(x, y, G_{i+1}) < \min \{\lambda(x, y, G), i + 1\}$. (IV)
 Da G_i ein Teilgraph von G_{i+1} ist, gilt: $\lambda(x, y, G_i) \leq \lambda(x, y, G_{i+1}) \leq \lambda(x, y, G)$

$$\begin{aligned} &\stackrel{\text{IV}}{\Rightarrow} \min \{\lambda(x, y, G), i\} \leq \lambda(x, y, G_i) \leq \lambda(x, y, G_{i+1}) < \min \{\lambda(x, y, G), i + 1\} \\ &\Rightarrow \lambda(x, y, G_i) = \lambda(x, y, G_{i+1}) = i \end{aligned}$$

Sei P^* der Pfad (zwischen x und y) in ganz G , der noch nicht in G_{i+1} enthalten ist. Es fehlt mindestens eine Kante $e = (p, r)$ in \bar{E}_i . Da jede zu P^* fehlende Kante in $G \setminus \bar{E}_i$ enthalten ist und F_{i+1} ein maximaler aufspannender Wald ist, muss F_{i+1} entweder einen zu P^* alternativen Pfad zwischen p, r haben und damit einen Pfad zwischen x, y schließen. Widerspruch.

5 Minimale Kreisbasen

5.1 Kreisbasen

Was ist ein Kreis in einem Graphen $G = (V, E)$?

naiv: Knotenfolge $v_1, v_2, \dots, v_k = v_1, (v_i, v_{i+1}) \in E \forall 1 < i \leq k - 1, v_i \neq v_j \forall i \neq j, i, j \leq k - 1$
 \Rightarrow spezielle Art von Kreisen: *einfache Kreise*

Definition

Ein Teilgraph $C = (V_c, E_c)$ von $G = (V, E)$ heißt *Kreis* genau dann, wenn alle Knoten von C einen geraden Grad haben.

Wir werden den Teilgraphen C durch seine Kantenmenge E_c beschreiben (von E_c induzierter Graph ist wieder C).

Sei die Kantenmenge E nun beliebig aber fest sortiert. Dann kann man jede Teilmenge $E_t \subseteq E$ als m -dimensionalen Vektor über $\{0, 1\}$ darstellen, indem wir den Vektor X^{E_t} definieren als:

$$X_i^{E_t} := \begin{cases} 1 & , \text{ falls } e_i \in E_t \\ 0 & , \text{ sonst} \end{cases}$$

Insbesondere kann man alle Kreise C als Vektoren X^C so darstellen.

Dann erhalten wir den Kreisraum von G , indem wir einfach die Vektoren X^C für $C \in \mathcal{C}$ bilden. Die Addition im Kreisraum lässt sich jetzt wieder als Operation \oplus auf \mathcal{C} verstehen:

$$c_1 \oplus c_2 = (E_{C_1} \cup E_{C_2}) \setminus (E_{C_1} \cap E_{C_2}) \quad (\text{XOR-Operation})$$

Anmerkung:

In einem zusammenhängenden Graphen muss eine Kreisbasis $m - n + 1$ Kreise enthalten.

Definition

Eine linear unabhängige Menge von Kreisen des Graphen G , die den gesamten Kreisraum aufspannt, heißt *Kreisbasis* (engl: Cycle-Base CB) von G .

Spezielle Kreisbasis:

1. Eine spezielle CB erhalten wir, wenn wir zusätzlich fordern, dass es eine Ordnung der Kreise in der CB gibt, so dass für jeden Kreis $C_i \in CB$ eine Kante $e \in C_i$, die in keinem Kreis C_j mit $j < i$ vorkommt. Eine solche Kreisbasis nennen wir **schwach fundamental**.
2. Eine Kreisbasis B mit der Eigenschaft, (dass für jede Ordnung der Kreise in B gilt:) Für $C_i \in B \exists e \in E$ mit $e \notin C_j$ mit $j \neq i$ und $C_j \in B$, nennen wir **stark fundamental**.

Eine einfache Methode, eine (stark fundamentale) Kreisbasis eines Graphen G zu erstellen, ist die folgende:

1. Bestimme ausspannenden Wald/Baum $T = (V_T, E_T)$ von G
2. Für jede Nichtbaumkante $e_i = (v, w)$, $e_i \in E \setminus E_T$ bilde den Kreis $C_i : P_T(v, w) \cup e_i$, wobei $P_T(v, w)$ den Pfad von v nach w im Baum T beschreibt. Diese Kreise nennen wir Fundamentalkreise zu e_i .
3. Die Menge all dieser Kreise C_i bildet eine Kreisbasis von G

Beweis

1. Die Menge B aller so konstruierten Kreise ist linear unabhängig.
2. Alle Kreise in G lassen sich als Linearkombination aus Kreisen $C_i \in B$ darstellen.

Behauptung: $C^* = \sum_{e_i \in C \setminus E_T} C_i$

Dazu 4 Fälle:

- (a) Für alle Kanten $e_i \in C^* \setminus E_T$ ist der Kreis C_i in der Linearkombination enthalten und e_i ist in keinem anderen Kreis in B und damit auch nicht in einem anderen Kreis in der Linearkombination enthalten. Also hat X^{C^*} an der Stelle e_i eine 1.
- (b) Für jede Kante $a = (u, v) \in E_T \setminus C^*$, zerfällt der Baum T ohne a in zwei Teile K_u und K_v , wobei $u \in K_u, v \in K_v$.
 C^* kreuzt diesen Schnitt eine gerade Anzahl von Malen. Für jede dieser Kanten e_j auf denen C^* den Schnitt kreuzt, gibt es einen Kreis $C_j \in B$, der in der Linearkombination vorkommt. Jeder dieser C_j beinhaltet Kante a . Damit wird an der Stelle a in X^{C^*} eine gerade Anzahl von Einsen aufsummiert.
- (c) Für jede Kante $b \in E_T \cap C^*$ zerfällt T ohne b in zwei Teile. Aber C^* kreuzt den Schnitt nur nach einer ungeraden Anzahl von Malen auf Nichtbaumkanten. Mit äquivalenter Argumentation hat X^{C^*} an der Stelle b eine 1.
- (d) Für jede Kante $d \in E \setminus (E_T \cup C^*)$ gibt es keinen Kreis, der d enthält, in der Linearkombination, also hat X^{C^*} an dieser Stelle einen Eintrag 0.

Damit hat X^{C^*} für alle Kanten aus dem Kreis einen Eintrag 1 und sonst einen Eintrag 0. → Damit bildet B ein CB.

Korollar:

Die Dimension des Kreisraumes zu einem Graphen $G = (V, E)$ mit $|V| = n$, $|E| = m$, ist $\nu = m - n + K(G)$, wobei $K(G)$ die Anzahl der Zusammenhangskomponenten von G ist.

Beweis: Folgt direkt aus der Konstruktion von stark fundamental Kreisbasen.

5.2 Kreisbasen minimalen Gewichts

Definition:

Sei zu einem Graphen G eine Gewichtsfunktion $w : E \rightarrow \mathbb{R}^T$ gegeben. Dann sei das Gewicht einer Kreisbasis B von G definiert als

$$w(B) = \sum_{C \in B} w(C) = \sum_{C \in B} \sum_{e \in C} w(e)$$

Dann ist eine minimale Kreisbasis (MCB) eine Kreisbasis mit minimalem Gewicht.

Bemerkungen:

1. Jeder Kreis in einer MCB ist einfach.
2. Für jede Kante $e \in E$, die in einem Kreis enthalten ist, gilt: Es gibt in jeder Kreisbasis B mindestens einen Kreis, der diese Kante e enthält. Insbesondere enthält jede MCB von G zu jedem $e \in E$ einen "kürzesten" Kreis, der e enthält. Wobei "kürzester Kreis" einen Kreis minimalen Gewichts beschreibt.
3. Die Menge $K := \{C_{\min}(e_i) : C_{\min}(e_i) \text{ kürzester Kreis der } e \text{ enthält, } e \in E\}$ ist im Allgemeinen keine CB.

Naiver Algorithmus zum Berechnen einer MCBEingabe: Graph $G = (V, E)$ Ausgabe: MCB zu G 1 Berechne Menge C aller Kreise2 Ordne Menge $\mathcal{C} := \{c_1, c_2, \dots, c_k\}$ aller Kreise in G aufsteigend nach Gewicht3 Init. $B \leftarrow \emptyset$ 4 **for** $i = 1$ **to** K **do**5 **if** $B \cup \{c_i\}$ linear unabhängig6 $B \leftarrow B \cup \{c_i\}$ 7 **endif**8 **endfor**Frage: Warum berechnen wir nicht alle *einfachen* Kreise?

Antwort: Der Hamilton Kreis ist ein einfacher Kreis und ist NP-hart, somit bringt uns das keinen Vorteil!

Anmerkung:

$$w(\text{MCB}) \leq w(\text{WFMCB}) \leq w(\text{SFMCB})$$

Sei $G = (V, E)$ und $w : E \rightarrow \mathbb{R}^+ : e \mapsto 1$ und sei B eine SFMCB von G , dann

$$w(B) = \sum_{\substack{e \in E \setminus E_T \\ e = (u,v)}} d_T(u, v) + 1 = \sum_{C \in B} \sum_{e \in C} w(e)$$

Spannbäume mit niedrigem durchschnittlichem Stretch:

$$Q(T) = \sum_{\substack{e \in E \\ e = (u,v)}} d_T(u, v)$$

$$\sum_{e \in E \setminus E_T} d_T(u, v) + 1 = (m - n + 1) + \sum_{e \in E \setminus E_T} d_T(u, v) = (m - n + 1) + \left(\sum_{e \in E} d_T(u, v) \right) - (n - 1)$$

 B ist SFCB, gegeben durch T .

$$w(B) = Q(T) + (m - 2(n - 1))$$

5.2.1 Der Algorithmus von Horton**Satz:** HortonFür jeden Kreis C einer MCB eines Graphen G existiert zu jedem Knoten V auf C eine Kante $\{u, w\}$ aus C , so dass gilt:

$$C = \text{SP}(u, v) + \text{SP}(w, v) + \{u, w\}$$

wobei $\text{SP}(u, v)$ den kürzesten Pfad von u nach v in G beschreibt.**Beweis** später

Daraus lässt sich ein Algorithmus ableiten.

*Algorithmus von Horton*Eingabe: $G = (V, E)$ Ausgabe: MCB von G 1 Initialisiere: Menge $H \leftarrow \emptyset$ 2 **for** $v \in V$ und $\{u, w\} \in E$ **do**3 Berechne $C_V(u, w) = \text{SP}(u, v) + \text{SP}(w, v) + \{u, w\}$ 4 **if** $C_V(u, w)$ einfacher Kreis5 $H \leftarrow H \cup \{C_V(u, w)\}$ 6 **endif**7 **endfor**8 Sortiere H aufsteigend nach Gewicht ($H = \{c_1, c_2, \dots, c_k\}$)9 Wende naiven Algorithmus (Schritt 2-4) auf H an**Korrektheit des Algorithmus':** Folgt direkt aus dem Satz von Horton.**Laufzeit:**

1. $|H| \leq n \cdot m$
2. Berechnung von H :
 $\mathcal{O}(n \cdot m \cdot L(\text{kürzester Weg}))^1$
3. Sortiere H :
 $\mathcal{O}((n \cdot m) \cdot \log(n \cdot m))$
4. Laufzeit naiver Algorithmus:
Test auf lineare Unabhängigkeit:
#Tests: $\mathcal{O}(n \cdot m)$ ($= |H|$)
gegen max. $m - n + K(G)$ viele Elemente: $\mathcal{O}(m)$
Einzelner Test: $\mathcal{O}(m^2)$
 \Rightarrow insgesamt $\mathcal{O}(m^3 \cdot n)$

Für den Beweis brauchen wir noch einige weitere Eigenschaften von Kreisbasen. Besonders interessiert uns das Austauschen von Basiselementen.

Lemma 1

Falls B eine Kreisbasis ist, $c \in B$ und $c = c_1 \oplus c_2$, dann ist entweder $B \setminus \{c\} \cup \{c_1\}$ oder $B \setminus \{c\} \cup \{c_2\}$ wieder eine Kreisbasis

Beweis

1. Wenn Kreis c_1 nicht als Linearkombination von Kreisen aus $B \setminus \{c\}$ darstellbar ist, dann ist offensichtlich $B \setminus \{c\} \cup \{c_1\}$ eine Basis.
2. Wenn c_1 als Linearkombination von Kreisen aus $B \setminus \{c\}$ darstellbar ist, dann muss c_2 linear unabhängig sein zu $B \setminus \{c\}$ und damit $B \setminus \{c\} \cup \{c_2\}$ eine Basis. \square

Lemma 2

Sei B eine Kreisbasis von G . Für zwei Knoten $x, y \in V$ und für einen Weg P von x nach y (in G), kann jeder Kreis $c \in B$, der x und y enthält, ersetzt werden durch einen Kreis c' der P enthält.

¹L() heißt Laufzeit von

Beweis:

c enthält zwei Pfade P_1, P_2 von x nach y . Damit können wir c schreiben als $c = c_1 \oplus c_2$, wobei c_1 gegeben ist durch P und P_1 und c_2 durch P und P_2 . Nach Lemma 1 ist nun entweder $B \setminus \{c\} \cup \{c_1\}$ oder $B \setminus \{c\} \cup \{c_2\}$ eine Basis.

Bemerkung:

Angenommen in der Situation von Lemma 2 ist weder P_1 noch P_2 ein kürzester Pfad (SP) von x nach y . Sei aber P ein kürzester Pfad zwischen x und y . Dann ist offenbar $w(c_1) < w(c)$ und $w(c_2) < w(c)$. Somit kann mit Lemma 2 jede Basis die c enthält in eine Basis B' umwandeln, die statt c entweder c_1 oder c_2 enthält und ein geringeres Gewicht als B besitzt.

Damit haben wir gezeigt, dass in einer MCB B jeder Kreis der Knoten x, y enthält auch einen kürzesten Pfad zwischen x und y enthält.

Beweis (Satz von Horton)

Betrachte einen beliebigen Kreis c der MCB B , sowie einen beliebigen Knoten v auf c . Seien die Knoten auf c indiziert mit v_0, v_1, \dots, v_k mit $v_0 = v_k = v$. Zum Knoten v_i auf c sei Q_i der Weg von v nach v_i auf c , in Richtung der Indizierung. Analog sei P_i der Weg von v_i nach v . Somit teilen Q_i und P_i den Kreis c in zwei Hälften. Nach unserer Anmerkung muss nun entweder Q_i oder P_i ein kürzester Pfad zwischen v und v_i sein.

Sei nun i der größte Index, so dass Q_i der kürzeste Weg zwischen v und v_i ist. Dann ist $c = Q_i \oplus \{v_i, v_{i+1}\} \oplus P_{i+1}$ die gesuchte Darstellung aus dem Satz. \square

5.3 Obere Schranke für minimale Kreisbasen

$$w(\text{MCB}) \leq w(\text{WFCB})^2 \leq w(\text{SFCB})^3$$

Damit gilt für obere Schranken:

Sei $S \geq w(B)$ für alle SFCB B

$\Rightarrow S \geq w(B')$ für WFCB B' , $\Rightarrow S \geq w(B'') \forall$ MCB B''

Definition (Girth)

Sei $G = (V, E)$ ein Graph. Wir bezeichnen mit **girth** $g(G)$ die Länge des kürzesten Kreises in G .

5.4 Obere Grenze für die Länge einer MCB (Horton '87)**Theorem:**

Die Länge einer MCB in einem ungerichteten, ungewichteten und einfachen Graphen G mit n Knoten ist $\leq n^2$.

Beweis: per Induktion

Klar für $n = 1, 2, 3$.

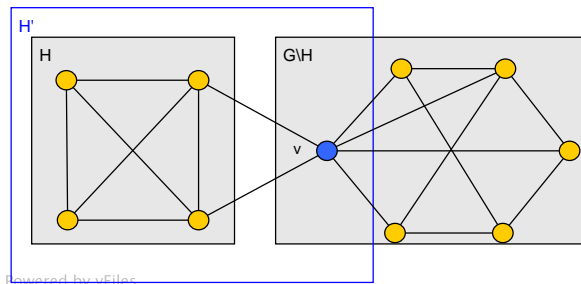
Induktionsschritt:

Sei nun $n \geq 3$ und sei G ein Graph mit $n + 1$ Knoten.

Falls G nicht zweifach knotenzusammenhängend ist: Sei v ein Schnittknoten, so dass $G \setminus \{v\}$ nicht zusammenhängend ist.

²ab 2007 für WFCB: $w(B) \in \mathcal{O}(n)$ nach Rizzi

³ab 2005 (zu SFCB): $w(B) \in \mathcal{O}(m \log^2 n \cdot \log \log n)$ nach Elkin, Emek, Spielman und Teng



Sei H eine Zusammenhangskomponente in $G \setminus \{v\}$ und $G \setminus H$ der Restgraph (inkl. v). Sei k die Anzahl der Knoten in H und $n + 1 - k$ die in $G \setminus H$.

Sei nun H' definiert als H plus v samt Kanten. Aus der Induktionsvoraussetzung folgt, dass es für H' und $G \setminus H$ eine CB mit Länge $\leq (k + 1)^2$ bzw. $\leq (n + 1 - k)^2$ gibt. Eine CB für G kann aus den beiden CBs für H' und $G \setminus H$ durch Vereinigung gebildet werden.

Wir nehmen daher nun an, dass G zweifach zusammenhängend ist. Sei x ein Knoten in G und T ein beliebiger Spannbaum in G . Wir bilden jetzt für Knoten x $\deg(x) - 1$ linear unabhängige Kreise (B') mit einer Gesamtlänge $\leq 2(n + 1)$ und fügen sie zu einer CB von $G \setminus \{x\}$ mit Länge $\leq n^2$ hinzu. Damit entsteht CB für G mit Gesamtlänge $\leq (n + 1)^2$.

ν' für $G \setminus \{x\} = m + 1 - n$
 ν für $G = m + k - n$

Sei nun $T' := T \cup E(x)$, wobei $E(x) = \{e = (x, y) \in E(G)\}$.

Laut IV gibt es eine Kreisbasis für $G \setminus \{x\}$ mit Länge $\in \mathcal{O}(n^2)$.

Aufgabe:

Finde genügend neue Kreise für x (Menge heiße B') mit Länge $\leq 2n$, so dass die Kreisbasis B von $G \setminus \{x\}$ und B' eine Gesamtlänge $\leq (n + 1)^2$ hat.

B' ist die Menge an Kreisen, die die Regionen einer planaren Zeichnung von T' entsprechen (-1 Region). Da in einer solchen Menge von Kreisen jede Kante höchstens zweimal vorkommt und planare Graphen nur $m \in \mathcal{O}(n)$ viele Kanten haben, trägt B' $\mathcal{O}(n)$ Länge bei.

$\Rightarrow \mathcal{O}(n^2)$ für die Gesamtlänge

Theorem (Rizzi, 2007)

$L(\text{wFMCB}) \in \mathcal{O}(W \log n)$

$W \cdot \sum_{e \in E} w(e)$ (Summe über alle Kantengewichte; m im ungewichteten Fall)

Beweis

Wir betrachten das folgende Lemma:

Lemma (Erdős, Posa 1962)

Sei G ein Graph mit minimalem Grad ≥ 3 . Dann ist der Girth (Taillenweite) $\leq 2 \log n$

Der Folgende Algorithmus konstruiert eine WFCB mit Länge $\in \mathcal{O}(W \log n)$.

(Gerichteter Graph):

```

1 while  $E \neq \emptyset$  do
2   while  $\exists v \in V$  mit  $\deg(v) = 1$  do
3      $G \leftarrow G - v$ 
4   endwhile
5   if  $\exists z \in V$  mit  $\deg(z) = 2$  und Nachbarn seien  $u, v$  do
6     Sei  $e_1 = (u, z)$  und  $e_2 = (z, v)$ 
7     Sei  $G'$  gebildet aus  $G - z$  durch hinzufügen einer neuen Kante  $e$ 
        mit Gewicht  $w(e) = w(e_1) + w(e_2)$ 
8   else
9     Sei  $C'$  ein Kreis mit  $L(C') \leq 2 \log n$ 
10    Sei  $e^*$  die Kante mit max. Gewicht in  $C'$ 
11    Setze  $G \leftarrow G - e^*$  und  $C \leftarrow C \cup C'$ 
12  endif
13 endwhile

```

Korrektheit

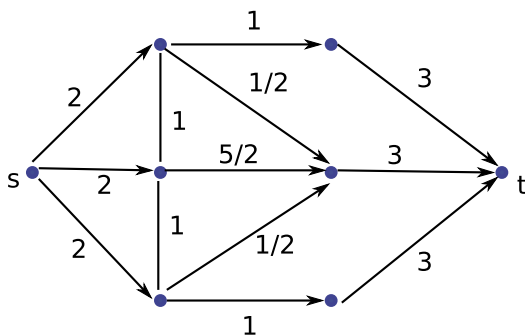
- Algorithmus entfernt in jedem Schritt entweder 1 Knoten / Kante
- Kreise sind für fundamental anordenbar \Rightarrow linear unabhängig
- Am Ende keine Zyklen mehr im Graphen $\Rightarrow m - n + 1$ Kanten entfernt \Rightarrow genügend Kreise

aus b) + c) $\Rightarrow C$ ist CB

- Jede Kante e^* addiert $\leq w(e^*)$ zu $L(C)$ und ist danach in keinem Kreis mehr enthalten.

□

6 Max Flow & Min Cost Flow

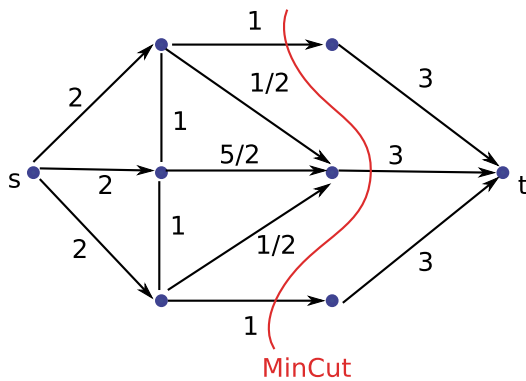


Definition: Netzwerkfluss

Sei $G = (V, E)$ ein gerichteter Graph mit Quelle s , Senke t und für alle Kanten $(v, w) \in E$ gibt es $cap(v, w) \geq 0$. Sei $cap(v, w) = 0$ für $(v, w) \notin E$.

Fluss $f : V \times V \rightarrow \mathbb{R}$ mit

- $f(v, w) = -f(w, v)$ (Schiefsymmetrie)
- $f(v, w) \leq cap(v, w)$ (Kap-Beschränkung)
- $\forall v \neq s, t: \sum_w f(v, w) = 0$ (Flusserhaltung)



Flusswert $|f| = \sum_v f(s, v)$ soll maximiert werden.

Schnitt $X \subseteq V$ definiert Partition von V in X und $\bar{X} = V \setminus X$ mit $s \in X$ und $t \notin X$. Es gilt:

$$cap(X) = \sum_{\substack{v \in X \\ w \in \bar{X}}} cap(v, w)$$

MinCut ist ein Schnitt minimaler Kapazität.

Fluss über einen Schnitt X ($f(X)$) ist

$$f(X) = \sum_{\substack{v \in X \\ w \notin X}} f(v, w)$$

Es gilt: $\sum_{v \in X} f(v, w) = |f|$ und:

Lemma

Für beliebigen Fluss f ist der Fluss über einen beliebigen Schnitt X gleich dem Flusswert.

Klar ist: $MaxFlow \leq MinCut$

Satz: MaxFlow-MinCut-Theorem (Ford/Fulkerson)

$$MaxFlow = MinCut$$

Konzept (Restgraph)

- Restkapazität $res : V \times V \rightarrow \mathbb{R}$ definiert durch $res(v, w) = cap(v, w) - f(v, w)$. Bedeutet: Wir können $f(v, w)$ um $res(v, w)$ erhöhen bis zur Kapazitätsgrenze.
- Restgraph R für f : Graph mit Knotenmenge V und Kanten (v, w) der Kapazität $res(v, w)$ mit $res(v, w) > 0$.
- Erweiternder Pfad für f : Pfad von s nach t in R .
- Restkapazität des Pfades p : $\min_{(v,w) \in R} res(v, w) =: res(p)$. Bedeutet: Bis zu $res(p)$ Flusseinheiten können wir entlang p schicken und damit f erhöhen.

Lemma:

Sei f ein beliebiger Fluss, f^* ein maximaler Fluss. Für den Restgraphen R in f ist der Wert eines maximalen Flusses auf R gleich $|f^*| - |f|$.

Beweis:

Sei f' ein beliebiger Fluss auf R und sei $(f + f')(v, w) = f(v, w) + f'(v, w)$. Es gilt: $f + f'$ ist Fluss auf G mit Wert $|f| + |f'|$.

$$\Rightarrow |f'| + |f| \leq |f^*|$$

$$\Rightarrow |f'| \leq |f^*| - |f|$$

Ähnlich ist $f^* - f$ definiert durch $(f^* - f)(v, w) = f^*(v, w) - f(v, w)$ ein Fluss auf R mit dem Wert $|f^*| - |f|$ und ist daher maximal.

Satz: MaxFlow-MinCut-Theorem

Die folgenden Behauptungen sind äquivalent:

1. Fluss f ist maximal.
2. Es gibt keinen erweiternden Pfad für f .
3. $|f| = \text{cap}(X)$ für einen Schnitt X .

Beweis: ⁴

1. \Rightarrow 2.: Durch einen erweiternden Pfad könnte f vergrößert werden, was also nicht maximal.

2. \Rightarrow 3.: Angenommen es gäbe keinen erweiternden Pfad für f . Sei X die Menge der von s aus erreichbaren Knoten in R . Dann ist X ein Schnitt und

$$|f| = \sum_{\substack{v \in X \\ w \notin X}} f(v, w) = \sum_{\substack{v \in X \\ w \notin X}} \text{cap}(v, w) = \text{cap}(X)$$

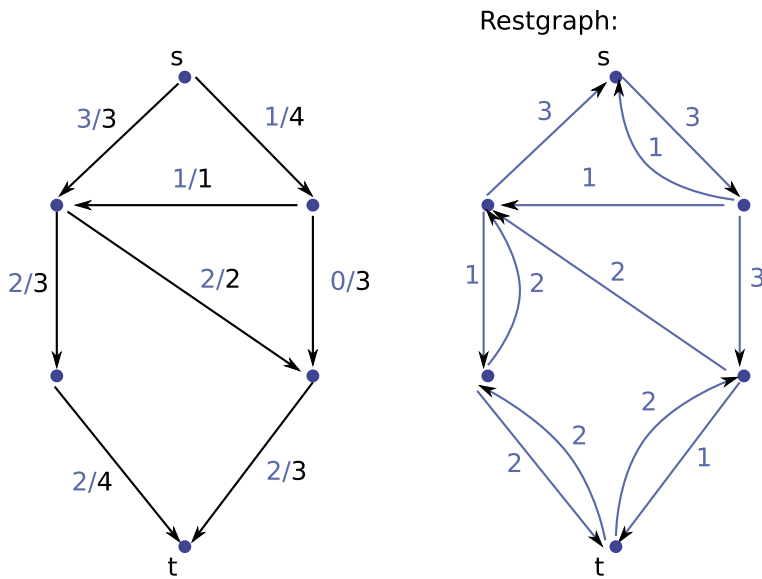
3. \Rightarrow 1.: Da $|f| \leq \text{cap}(X)$ für beliebigen Fluss f und beliebigen Schnitt X , folgt aus $|f| = \text{cap}(X)$, dass f maximal ist.

6.1 Algorithmus (Ford/Fulkerson)

- Starte mit Nullfluss
- Solange es geht wiederhole:
 - Finde erweiternden Pfad p für aktuellen Fluss
 - Schicke $\text{res}(p)$ Einheiten entlang p

⁴Evtl. prüfungsrelevant

Beispiel:



Laufzeit Annahme die Kapazitäten sind ganzzahlig.
 \Rightarrow Maximal $|f^*|$ Iterationen, da Flusserhöhung ganzzahlig auf f^* ganzzahlig.

Aber Falls manche Kapazitäten groß sind, kann es sehr lange dauern.

Lemma (geschickte Wahl von Pfaden hilft)
 Beginnend mit Nullfluss gibt es Wahl von höchstens m einzelnen Pfaden, die zum Maxflow führen.

Beweis

Sei f^* ein max. Fluss und sei G^* der Teilgraph der von f^* benutzten Kanten.
 Iteriere. Suche p_i von s nach t in G^* . Sei Δ_i der min. Fluss über eine Kante e_i in p_i . Erniedrige Fluss über Kante in p_i um Δ_i . e_i fällt aus G^* raus.
 \rightarrow In jeder iteration eine Kante weniger.

Satz

Geht die Erweiterung entlang kürzester Wege, so hält Methode mit erweiterter Pfade noch $(n - 1) \cdot m$ Erweiterungen und läuft dann in $\mathcal{O}(n \cdot m^2)$.

Beweis

1. Distanz zwischen s und t nimmt nie ab.
2. Spätestens nach m Erweiterungen entlang kürzester Wege gleicher Länge wächst Distanz um mindestens 1.
 Höchstens $(n - 1) \cdot m$ Erweiterungen und $\mathcal{O}(m)$ per Erweiterung auf kürzestem Weg mit BFS.

$\Rightarrow \mathcal{O}(n \cdot m^2)$

6.2 Algorithmus 2

“Suche immer den Pfad mit maximaler Restkapazität” (Edmonds / Karp '72)

Satz

Diese Methode führt für Ganzzahlige Kapazitäten und die maximale Kapazität C in $\mathcal{O}(m \cdot \log C)$ Schritten zum Ziel.

Beweis

Sei f beliebiger Fluss und f^* der Maxflow. Dann gibt es f' auf Restgraph und $|f'| = |f^*| - |f|$.

\Rightarrow Es gibt höchstens m erweiterte Pfade mit Gesamtrestkapazität mindestens $|f'|$

\Rightarrow Der größte davon hat Restkapazität $\geq \frac{|f^*| - |f|}{m}$

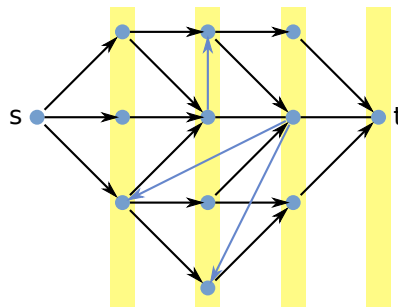
Mache $2m$ Erweiterungen mit maximaler Restkapazität. Spätestens dann wird der Fluss nur noch höchstens $\frac{|f^*| - |f|}{2m}$ erhöht, also hat sich die maximale Restkapazität um Faktor 2 oder mehr verringert.

\Rightarrow nach $\log C$ solcher Runden von je $2m$ Iterationen ist der Fluss maximal.

\Rightarrow Laufzeit : $\mathcal{O}(\underbrace{m \log C}_{\# \text{Iterationen}} \cdot \underbrace{m \log n}_{\text{krz. Wege}})$

6.3 Algorithmus 3

Dinic: Erweitere entlang kürzester Wege und entlang mehrere solcher Wege gleichzeitig.



Level(v): Länge des Kürzesten Weges von s nach v in R

Levelgraph L : Teilgraph von R mit Kanten (v, w) , so dass $\text{Level}(w) - \text{Level}(v) = 1$ mit BFS konstruierbar in $\mathcal{O}(m)$.

blockierender Fluss f : Jeder Pfad von s nach t enthält saturierte⁵ Kante ($f(e) = \text{cap}(e)$)

Algorithmus

```

starte mit Nullfluss  $f$ 
Iteriere
  berechne Levelgraph
  berechne blockierenden Fluss  $f'$  auf Levelgraph
  erhöhe  $f$  um  $f'$ 
bis  $t$  nicht mehr erreichbar im Levelgraph

```

Lemma

Algorithmus hält nach maximal $n - 1$ Iterationen für blockierte Flüsse.

Denn: Level für jeden Knoten wächst höchstens $\text{level}(t) < \text{level}_{\text{schluss}}(t)$

\Rightarrow Höchstens $n - 1$ Iterationen

⁵saturiert heißt gesättigt, also die kante hat keinen Restkapazität mehr

Lemma

Blockierender Fluss kann in Zeit $\mathcal{O}(m \cdot n)$ auf Levelgraph mit m Kanten, n Knoten gefunden werden.

Idee: Iterativ suche mit DFS einen Pfad von s nach t . Schicke dort maximalen Fluss = $\min_{e \in p} \text{cap}(e)$ rüber. Es geht mit $\mathcal{O}(m)$ Iterationen, da jeweils ≥ 1 Kante wegfällt.

→ $\mathcal{O}((n + m) \cdot m)$, aber es geht auch in $\mathcal{O}(n \cdot m)$

⇒ Gesamtlaufzeit: $\mathcal{O}(n \cdot m^2)$

6.4 Preflow-Push-Algorithmen

Schickt eventuell zu viel Fluss durchs Netzwerk. Maximaler Fluss geht zur Senke. Rest zurück zur Quelle.

Preflow $x : E \rightarrow \mathbb{R}$ mit $x(e) \leq \text{cap}(e)$

$\sum_{w, (w,v) \in E} x(w,v) - \sum_{w, (v,w) \in E} x(v,w) \geq 0$ für alle $v \in V - \{s, t\}$

Überfluss $e(v) = \sum x(w,v) - \sum x(v,w) \geq 0$

v ist aktiv, falls $e(v) > 0$

Beachte: s, t sind nie aktiv

Versuche Überfluss Richtung t (d = Abstand zu t) zu schicken.

Kante (v, w) erlaubt, falls $d(v) - d(w) = 1$. Hat v keine erlaubte Kante, so erhöhe Distanz-Wert $d(v)$, so dass mindestens eine erlaubte Kante entsteht.

preprocessing:

$x \leftarrow 0$

berechne Distanzen $d(v)$ für alle v zu t

$x(s,v) \leftarrow \text{cap}(s,v)$ für alle $(s,v) \in E$

$d(s) \leftarrow n$

push-relabel(v)

if \exists erlaubte Kante (v,w) then

 push $\delta = \min(e(v), r(v,w))$ Fluss von v nach w (*Push*)

else

$d(v) \leftarrow \min\{d(w) + 1 \mid (v,w) \in E \text{ und } r(v,w) > 0\}$ (*Relabel*)

preflow-push

preprocessing

while \exists aktive Knoten do

 sei v aktiv

 push-relabel(v)

endwhile

Push ist sättigend falls $\delta = r(v,w)$ und sonst nicht sättigend (danach ist $e(v) = 0$).

Beobachtungen:

- s ist von jedem Knoten mit $e(v) > 0$ erreichbar, denn: Fluss zu v hat die zugehörigen umgedrehten Kanten erzeugt und somit kann potentiell der ganze Fluss wieder zu s zurück.
- Es gilt: $d(v) < 2n$ für alle $v \in V$, denn beim letzten Relabel gibt es einen Pfad von v nach s (der höchstens Länge $n - 2$ hat). Wegen $d(s) = n$ und $d(v) = d(w) + 1$ für alle Kanten auf dem Pfad gilt: $d(v) \leq d(s) + |P| < 2n$
⇒ Es gibt höchstens $2n^2$ Relabel-Operationen.

3. Es gibt maximal $2n \cdot m$ sättigende Pushes.

Vor jedem Relabel von v maximal $\deg(v)$ sättigende Pushes, die von v ausgehen.

Insgesamt gibt es an v maximal $2n \cdot \deg(v)$ sättigende Pushes.

\Rightarrow Insgesamt maximal $2n \cdot \sum_v \deg(v) < 2n \cdot m$ sättigende Pushes.

4. Es gibt maximal $n^2 \cdot m$ nicht sättigende Pushes.

Beweis:

Benutze Potenzialfunktion $\Phi = \sum_{v \text{ aktiv}} d(v)$. Da es maximal n aktive Knoten gibt und $d(v) < 2n$ gilt, ist $\Phi \leq 2n^2$ am Anfang und zum Schluss 0.

1. Relabel-Operation wird durchgeführt.

Φ steigt um x , falls $d(v)$ um x steigt.

Insgesamt steigt $d(v)$ um höchstens $2n$.

$\Rightarrow \Phi$ steigt um höchstens $2n^2$ durch Relabeling.

2. Sättigender Push über (v, w) erzeugt eventuell neuen Überfluss an w .

w wird (eventuell) aktiv und Φ steigt dann um $d(w) < 2n$.

$\Rightarrow \Phi$ steigt durch sättigende Pushes maximal um $4n^2m$.

3. Nicht sättigender Push über (v, w) .

Φ wird um $d(v)$ erniedrigt, da v danach nicht mehr aktiv, eventuell um $d(w)$ erhöht, falls w vorher nicht aktiv war.

Mit $d(v) = d(w) + 1$ sinkt Φ mindestens um 1 bei nicht sättigenden Pushes.

Insgesamt haben wir eine Erhöhung von Φ von $2n^2$ um höchstens $2n^2 + 4n^2m$ und y Erniedrigungen von Φ um mindestens 1.

$\Rightarrow y \leq \mathcal{O}(n^2m)$ (y ist die Anzahl der nicht sättigenden Pushes)

6.4.1 Verbesserung:

Warte nicht ab, bis der Überfluss zurückgeflossen ist.

Halte Menge V' von Knoten, von denen aus t nicht mehr erreichbar ist.

1. Zu V' kommen die Knoten mit d -Wert $\geq n$.

2. Mache ab und zu eine BFS von t ausgehend und entferne nicht mehr erreichbare Knoten aus V' .

3. Von Knoten aus V' mache kein Push-Relabel mehr.

6.4.2 Regeln zur Wahl aktiver Knoten:

1. FIFO: $\mathcal{O}(n^3)$

2. Highest-Label: $\mathcal{O}(n^2 \cdot \sqrt{m})$

3. Excess-Scaling: $\mathcal{O}(n \cdot m + n^2 \cdot \log C)$ (wobei C die größte Kapazität ist)

6.4.3 FIFO Preflow Push

Wende Push-Relabel solange auf Knoten v an, bis entweder $e(v) = 0$ oder Relabel-Operation angewendet wurde. Liste der aktiven Knoten wird als Schlange gehalten ("vorne rein, hinten raus").

Definition (Phase)

Abarbeiten der Liste bis zum ersten Mal ein Knoten kommt, der in der vorherigen Phase abgearbeitet wurde.

Behauptung:

Es gibt höchstens $\mathcal{O}(n^2)$ Phasen.

→ Pro Phase gibt es für jeden Knoten nur einen nicht sättigenden Push

⇒ Anzahl der Phasen · Anzahl der Knoten = $\mathcal{O}(n^2) \cdot \mathcal{O}(n) = \mathcal{O}(n^3)$

Beweis:

Betrachte Potenzial $\Phi = \max \{d(v) | v \text{ ist aktiv}\}$.

Fall 1: Während einer Phase gibt es mindestens ein Relabel.
 Φ steigt höchstens so viel, wie der d -Wert maximal ist.
 Insgesamt steigt Φ in diesem Fall also maximal um $2n^2$.

Fall 2: Es gibt kein Relabel in dieser Phase.
 Alle aktiven Kanten mit maximalem d -Wert werden inaktiv.
 D.h. der maximale d -Wert sinkt um mindestens 1.

Insgesamt: d -Wert des Nachbars von s + Fall 1 + Fall 2 = $n - 1 + 2n^2 + (n + n^2) = \mathcal{O}(n^2)$ Phasen

⇒ FIFO Methode läuft in Zeit $\mathcal{O}(n^3)$.

6.4.4 Highest-Label

Nimmt jeweils aktive Knoten mit höchstem d -Wert.

- Es gibt $\mathcal{O}(n^3)$ nicht sättigende Pushes.
 Sei $h^* = \max \{d(v) | v \text{ aktiv}\}$. Wir betrachten zuerst aktive Knoten v mit $d(v) = h^*$, dann $h^* - 1, h^* - 2, \dots$
 Bei Relabel fängt neue Phase an. Es gibt maximal $2n^2$ Relabels. Gibt es n nicht sättigende Pushes hintereinander, sind wir fertig, denn alle Knoten sind inaktiv.
 ⇒ $\mathcal{O}(n^3)$ nichtsättigende Pushes.
- Zu zeigen: Es gibt $\mathcal{O}(n^2 \cdot \sqrt{m})$ nicht sättigende Pushes.

Beweis:

Wähle $K (= \sqrt{m})$. Für Knoten v sei $d'(v) = \frac{|\{w | d(w) \leq d(v)\}|}{K}$ und betrachte Potenzial $\Phi = \sum_{v \text{ aktiv}} d'(v)$.

Phase: Alle Pushes zwischen 2 Veränderungen von $h^* = \max \{d(v) | v \text{ aktiv}\}$.
 Teure Phase: Enthält mehr als K nicht sättigende Kanten.
 Billige Phase: Sonst.

Behauptungen:

- (a) Anzahl der Phasen ist maximal $4n^2$.
- (b) Anzahl nicht sättigender Pushes in billigen Phasen ist höchstens $4n^2K$.
- (c) $\Phi \geq 0$ immer und anfang ist $\Phi \leq \frac{n^2}{K}$.
- (d) Relabels und sättigende Pushes erhöhen Φ um höchstens $\frac{n}{K}$.
- (e) Nicht sättigende Pushes erhöhen Φ nicht.
- (f) Teure Phasen mit $Q \geq K$ nicht sättigenden Pushes erniedrigen Φ um mindestens Q .

Aus diesen Behauptungen folgt: (d) und (e) bedeuten, dass Φ um höchstens $(2n^2 + m \cdot n) \cdot \frac{n}{K}$ steigt, und somit ist der Schwund von Φ $(2n^2 + m \cdot n) \cdot \frac{n}{K} + \frac{n^2}{K}$ nach (c).

\Rightarrow In teuren Phasen gibt es $\frac{(2n^3 + m \cdot n^2 + n^2)}{K}$ nicht sättigende Pusches und mit (b) haben wir insgesamt höchstens $\frac{(2n^3 + m \cdot n^2 + n^2)}{K} + 4n^2K$
 $= \frac{(3m \cdot n^2)}{K} + 4n^2K$ nicht sättigende Pushes.
 $= \mathcal{O}(n^2 \cdot \sqrt{m})$ für $K = \sqrt{m}$

Beweis der Behauptungen:

- (a) $h^* = 0$ am Anfang. Wird durch Relabels erhöht, also maximal $2n^2$ mal. \Rightarrow Es gibt maximal $2n^2$ Erniedrigungen.
- (b) folgt direkt aus (a).
- (c) $\Phi \leq \frac{n^2}{K}$, da $d'(v) \leq \frac{n}{K}$ für alle v .
- (d) $d'(v) = \frac{n}{K}$, somit kann Φ pro Operation maximal um $\frac{n}{K}$ steigen.
- (e) Nicht sättigender Push macht v inaktiv. w wird eventuell inaktiv. Aber $d'(v) \geq d'(w)$, da $d(v) = d(w) + 1$.
- (f) Sei P eine teure Phase mit Q nicht sättigende Pushes. h^* bleibt während P konstant. Pushes passieren nur auf Knoten mit d -Wert h^* . P endet, falls es keine solchen Knoten mehr gibt oder h^* -Wert hat sich erhöht. In beiden Fällen enthält h^* $Q > K$ Knoten. Jeder nicht sättigende Push erniedrigt Φ um mindestens 1 (da $\underbrace{d'(w) \leq d'(v) - 1}_{\star}$ beim Push von (v, w)).

\star) Da es davon mindestens K Stück gibt.

Satz

Der Excess Scaling Algo läuft in Zeit $\mathcal{O}(nm + n^2 \log C)$ mit C maximale Kapazität

Idee

Sei $\Delta \geq e_{max}$ der maximale Überfluss. v hat nen großen Überfluss, falls $e(v) \geq \frac{\Delta}{2}$ ($\geq \frac{e(v)}{2}$), und keinem Überfluss sonst. Wir kümmern uns nur um große Überflüsse.
 e_{max} sollte sich nicht weiter erhöhen.

Regel: Suche unter den Knoten mit großem Überfluss den mit minimaler Distanz. Statt $\delta = \min\{e(v), r(v, w)\}$ schicke nun $\delta = \min\{e(v), r(v, w), \Delta - e(v)\}$ über Kante (v, w) .

Anfangs wird $e(w)$ Δ nicht übersteigen. Anfangs ist $\Delta = 2^{\lceil \log C \rceil}$, also $C \leq \Delta \leq 2C$.

Fällt e_{max} unter $\frac{\Delta}{2}$, so gibt es eine neue Skalierungsphase, das heißt $\Delta \leftarrow \frac{\Delta}{2}$. Nach $\lceil \log C \rceil + 1$ Skalierungsphasen ist $e_{max} = 0$ und damit Maxflow gefunden.

Eigenschaften

1. Jeder nicht sättigende Push schickt $\geq \frac{\Delta}{2}$ Flusseinheiten
2. $e_{max} \leq \Delta$ (Invariante)
3. Pro Phase gibt es $\mathcal{O}(n^2)$ nicht sättigende Pushes $\Rightarrow \mathcal{O}(n^2 \log C)$ nicht sättigende Pushes

6.5 MinCut

MinCut in ungerichteten Graphen: Gegeben $G = (V, E)$, $w : E \rightarrow \mathbb{R}^+$ Gewichte
 Schnitt $C = V$ mit $W(c) = \sum_{\substack{e \in E \\ |e \cap C|=1}} w(e)$ MinCut ist Schnitt mit minimalem Gewicht

Berechnung naiv: n^2 MaxFlow für feste $s, t \Rightarrow$ Laufzeit $\mathcal{O}(n^2 \cdot n^{2m})$
 Besser: n mal MaxFlow mit festem s in $\mathcal{O}(n \cdot n^{2m})$
 Einfacher Algo von Nagamouchi / Ibaraki '92 in $\mathcal{O}(n \cdot m + n^2 \log n)$

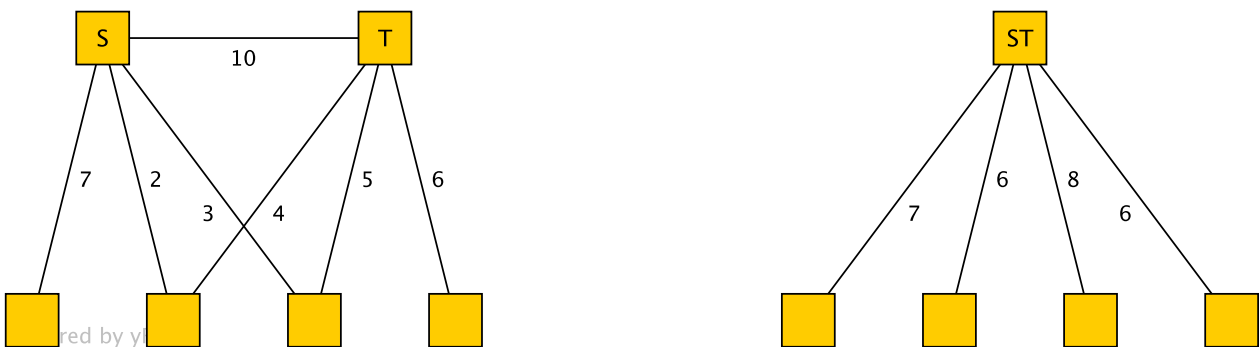
Idee

$s - t$ Schnitt ist Schnitt der genau einen der Knoten s und t enthält.
 $n - 1$ Phasen: Bestimme in jeder Phase Paar s, t und ein $s - t$ Schnitt C .
 Gibt es MinCut der s und t trennt, so ist C ein MinCut. Wenn nicht, liegen s und t auf einer Seite des MinCut.
 Wir vereinigen s, t so dass der neue Graph den selben MinCut hat wie der von G . In jeder Phase schrumpft der Graph um 1, ein neuer Kandidat für den MinCut wird gefunden. Nach $n - 1$ Phasen sind wir fertig.

Phase: sei a irgendein Knoten.
 $A \leftarrow \{a\}$
Iteriere
 bestimme $v \notin A$ mit $w(v, A) = \sum_{e=(v,y), y \in A} w(e)$ maximal
 füge v zu A hinzu
od
 s und t sind die beiden letzten Knoten, die zu A hinzukommen

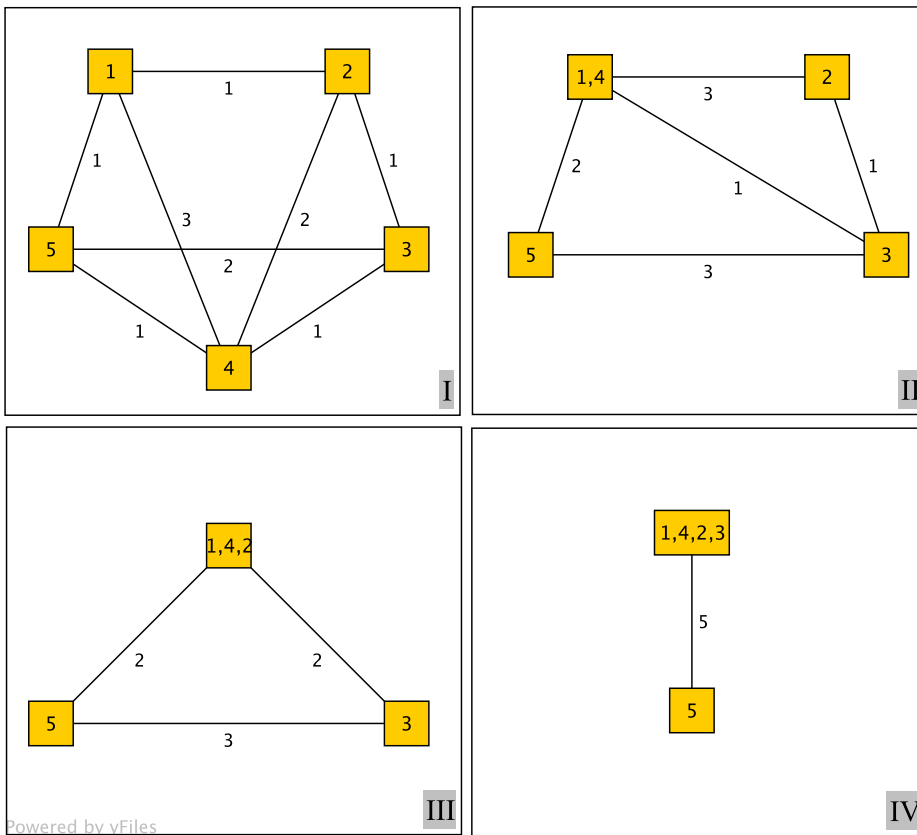
Lemma

Es gilt: $\{t\}$ ist ein minimaler $s - t$ Schnitt.
 Bestimme v mit $w(v, A)$ minimal wie bei Prim's MST-Algorithmus (suche für jeden Baum die minimale Kante, die davon weggeht; hier maximales Gewicht).
 Speichere $w(v, A)$ in PriorityQueue. Gesamtzeit $\mathcal{O}(m + n \log n)$ pro Phase.
 Vereinigen von s und t :
 Lösche t . Hänge alle Kanten zu t nun an s dran. Erhöhe evtl. Gewichte.



Kante (s, t) fällt weg

Beispiel:



- I: $a = 1$
 $A = \{1\}$
- II: $A = \{1, 4\}$
- III: $A = \{1, 4, 2\}$
- IV: $s = 3$
 $t = 5$
 $w(\{t\}) = 5$

Phase 2: Vereinige 3 und 5

$a = 1, A = \{1\}$
 $\rightarrow A = \{1, 4\}$
 $A = \{1, 4, 2\}$
 $s = \{2\}, t = \{3, 5\}, w(\{t\}) = 4$
 Zweiter Kandidat $C = \{3, 5\}$

Phase 3: Vereinige 35 und 2

$a = \{1\}$
 $\rightarrow A = \{1, 4\}$
 $s = \{1, 4\}, t = \{3, 5, 2\}$
 $\Rightarrow \text{MinCut} = \{3, 5\}$

Behauptung:

$\{t\}$ ist nun $s - t$ -Schnitt.

Beweis:

Sei C' ein beliebiger Schnitt. Wir zeigen, dass $w(C') \geq w(\{t\})$.
 v_1, \dots, v_n sei die Reihenfolge, in der Knoten zu A kommen. $s = v_{n-1}, t = v_n$. v_i heißt kritisch, falls $i \geq 2$ und

v_i und v_{i-1} liegen auf verschiedenen Seiten von C' . t ist kritisch. k sei die Zahl kritischer Knoten und i_1, \dots, i_k ihre Indizes.

Es gilt: $w(\{t\}) = w(v_{i_k}, A_{i_k-1})$ für $A_i = \{v_1, v_2, \dots, v_i\}$ und

$$w(C') \geq \sum_{j=1}^k w(v_{i_j}, A_{i_j-1} \setminus A_{i_{j-1}-1})$$

mit $v_n = t = v_{i_k}$.

Behauptung:

$\forall l, 1 \leq l \leq k$ gilt:

$$w(v_{i_l}, A_{i_l-1}) \leq \sum_{j=1}^l w(v_{i_j}, A_{i_j-1} \setminus A_{i_{j-1}-1})$$

Beweis durch Induktion:

Stimmt für $l = 1$.

$l \geq 2$:

$$\begin{aligned} w(v_{i_l}, A_{i_l-1}) &= w(v_{i_l}, A_{i_{l-1}-1}) + w(v_{i_l}, A_{i_l-1} \setminus A_{i_{l-1}-1}) \\ &= w(v_{i_{l-1}}, A_{i_{l-1}-1}) + w(v_{i_l}, A_{i_l-1} \setminus A_{i_{l-1}-1}) \\ &= \sum_{j=1}^{l-1} w(v_{i_j}, A_{i_j-1} \setminus A_{i_{j-1}-1}) + w(v_{i_l}, A_{i_l-1} \setminus A_{i_{l-1}-1}) \\ &= \sum_{j=1}^l w(v_{i_j}, A_{i_j-1} \setminus A_{i_{j-1}-1}) \end{aligned}$$

□

$$\Rightarrow w(\{t\}) = w(v_{i_k}, A_{i_k-1}) \leq \sum_{j=1}^k w(v_{i_j}, A_{i_j-1} \setminus A_{i_{j-1}-1})$$

6.5.1 Heuristik

Wir merken uns den aktuellen Kandidaten für den MinCut mit Gewicht w .

Idee: Kanten mit Gewicht $\geq w$ kommen in keinem besseren MinCut vor. Verschmelze ihre Endpunkte.

Beachte: Durch das Verschmelzen erhöht sich das Gewicht der anderen Kanten.

Praktisch ist dies eine sehr große Verbesserung.
Theoretisch jedoch keine.

6.6 Min Cost Flow

Gegeben Netzwerk $G = (V, E, \text{cap}, \text{cost})$ mit Kapazität $\text{cap} : E \rightarrow \mathbb{R}^+$ und $\text{cost} : E \rightarrow \mathbb{R}^+$, den Kosten. Nehme für cost Schiefsymmetrie an: $\text{cost}(v, w) = -\text{cost}(w, v)$

Kosten von Fluss f : $\sum_{f(v,w) > 0} \text{cost}(v, w) \cdot f(v, w)$

MinCost Flow is ein *maximaler Fluss* minimaler Kosten.

Lemma

Fluss f hat minimale Kosten **genau dann**, wenn der entsprechende Restgraph keine negativen Zykel hat.

Beweis:

“ \Rightarrow ” Annahme R hat negative Zykel. Schicke Fluss entlang Zykel und vermindere so die Kosten.

“ \Leftarrow ” Annahme f hat nicht minimale Kosten. Sei f^* ein Fluss maximaler Kosten mit $|f| = |f^*|$
 Betrachte $f^* - f$ auf R . Hat Wert 0, aber negative Kosten. $f^* - f$ besteht aus Menge von Flüssen auf Zykel. Also muss mindestens einer der Zykel negative Kosten haben.

Anmerkung zu Prüfungsfragen: Was machen bei negativen Zyklen?

6.6.1 Cycle Canceling**Strategie:** Cycle Canceling

Starte mit Maxflow. Suche negative Zykel in R und schicke Fluss dort entlang, solange bis es keine negativen Zykel mehr gibt.

Alternative: Wenn es auf G keine negativen Zykel gibt:

Erweitere Fluss entlang billigster Wege.

Lemma

Ist f ein Fluss minimaler Kosten, so hat jeder Fluss, der durch Erweiterung von f über einen Pfad minimaler Kosten entsteht, minimale Kosten.

Beweis:

Sei p ein billigster, erweiternder Pfad für f mit Restgraph R und sei f' der Fluss, der durch die Erweiterung entsteht.

Annahme, f' haben nicht minimale Kosten. Dann gibt es im Restgraph R' bezüglich f' einen negativen Zykel c . c besteht aus Kanten aus R und aus Kanten aus $R' - R$, welches umgekehrte Kanten von p sind. Seien $p \oplus c$ die Kanten, die auf p oder c vorkommen, aber **nicht** auf p und **gleichzeitig** auf c .

Kosten von $p \oplus c$ sind $\text{cost}(p) + \text{cost}(c) < \text{cost}(p)$ (Anm: $\text{cost}(c)$ negativ; weggelassene Kanten haben zusammen Kosten 0 (Schiefsymmetrie))

$p \oplus c$ zerfällt in einen Pfad von s nach t und Menge von Zykeln. Nach obigem Lemma gibt es aber keine negativen Zykel in R

\Rightarrow Kosten des neuen s-t-Pfades sind kleiner als $\text{cost}(p)$

Widerspruch!

Nimm an $\text{cap} : E \rightarrow \mathbb{Z}$ ganzzahlig.

Flüsse bei beiden Methoden bleiben also ganzzahlig, das heißt, beide Methoden terminieren. Bei 'billigster Wege'-Methode terminiert Verfahren nach $\leq |f^*|$ Erweiterungen.

Lemma

Die billigsten erweiterten Pfade haben nicht fallende Kosten.

Beweis

Sei f minimaler Fluss mit Restgraph R und sei $\text{cost}(v)$ die Kosten eines billigsten Weges von s nach v in R . Es gilt für Kante $(v, w) \in R$: $\text{cost}(v) + \text{cost}(v, w) > \text{cost}(w)$, und falls (v, w) auf billigstem Weg zu w liegt gilt: $\text{cost}(v) + \text{cost}(v, w) = \text{cost}(w)$. Wir erweitern entlang einem billigsten Pfad, so gilt für die neuen Kanten aus $R' \setminus R$ diese Gleichheit, also $\text{cost}(w, v) = -\text{cost}(v, w) = \text{cost}(v) - \text{cost}(w)$.

Benutze Induktion über Zahl der Kanten auf billigsten Weg von s nach v , um zu zeigen: $\text{cost}'(v) \geq \text{cost}(v)$.

Also: benutze f^* mal 'billigste Wege'.

Dijkstra für nichtnegative Kantenkosten. $\mathcal{O}(n \log n + m)$
haben aber negative Kantenkosten: Bellman/Ford: $\mathcal{O}(n \cdot m)$

\Rightarrow Laufzeit: $\mathcal{O}(n \cdot m \cdot |f^*|)$

6.6.2 Vermeide negative Kantenkosten

Berechne zuerst den Baum der billigsten Wege von s aus (in $\mathcal{O}(nm)$ nach Bellman/Ford). Reduziere cost zu $\text{cost}'(v, w) = \text{cost}(v, w) + \text{cost}(v) - \text{cost}(w)$.

cost' ist nicht negativ, da $\text{cost}(v) + \text{cost}(v, w) \geq \text{cost}(w)$, das heißt, liegt (v, w) auf erweiterndem Pfad, gilt Gleichheit und somit $\text{cost}'(v, w) = 0$ und auch $\text{cost}'(w, v)$.

Es gilt: Billigster Weg bezüglich cost' ist identisch zum billigsten Weg bezüglich cost , denn

$$\sum_{(v,w) \in P} \text{cost}'(v, w) = \sum_{(v,w) \in P} \text{cost}(v, w) + \text{cost}(s) - \text{cost}(t)$$

mit P Pfad von s nach t . Wir wenden also jeweils cost' an und rechnen mit Dijkstra weiter \Rightarrow Laufzeit $\mathcal{O}(m \cdot n + |f^*|(n \log n + m))$

Streng polynomiell: $\mathcal{O}((n \log n)(m + n \log n))$ (nach Orlin)
 $\mathcal{O}(mn \cdot \log(nc) \cdot \log \log U)$ (nach Ahuja Golaberg und Orlin Tarjan '92)

6.6.3 Algorithmen

MinCost Flow: Beginne mit MaxFlow

```
solange  $R$  negative Zykel enthält
  schicke Fluss entlang einem solchen Zykel
od
```

- Idee:** Finde negative Zykel: $\mathcal{O}(n \cdot m)$ mit Bellman/Ford
 \rightarrow Gesamtlaufzeit beträgt $\mathcal{O}(nm \cdot C \cdot U)$ wobei $C =$ Gesamtkosten und $U =$ größte Kapazität
Fall: Kapazität ganzzahlig
#Iterationen von Cycle Canceling $\mathcal{O}(m \cdot C \cdot U)$ denn Gesamtkosten $\leq m \cdot C \cdot U$. In jeder Iteration gehen aktuelle Kosten um ≥ 1 runter
 $\rightarrow m \cdot C \cdot U$ Iterationen
- Idee:** Schicke den Fluss entlang einem Zykel, der maximalen Fortschritt bringt
Es gilt: Eigentlich genügen m -Zykel, um MinCost Flow zu erreichen
 \rightarrow Verbessern entlang dem 'besten' Zykel ergibt optimalen Fluss innerhalb von $\mathcal{O}(m \log(C \cdot U))$ Iterationen
Aber: 'Besten' Zykel zu finden ist NP vollständig!
- Idee:** Schicke Fluss entlang Zykel mit minimalen mittleren Kantenkosten (minimum mean cycle)
 \rightarrow braucht $\mathcal{O}(\min\{n \cdot m \cdot \log(n \cdot C), nm^2 \cdot \log n\})$ Iterationen

Algorithmus: Sei $\mu(Z) = \sum_{(i,j) \in Z} \text{cost}(i,j)/|Z|$ die mittleren Kosten von Zykel Z

Suche $\mu^* = \min_Z \mu(Z)$
 Annahme: $\text{cost}(i, j)$ ganzzahlig, $\max \text{cost}(i, j) =: M$
 Es gilt: $\mu^* \leq \mu$
 Suche linear / binär nach μ^*
 Setze $\text{cost}'(i, j) \leftarrow \text{cost}(i, j) - \mu$
 Erhalte Netzwerk R'
 Wende Bellman/Ford auf R' an

1. R' hat Zykel Z negativer Kosten $\rightarrow \mu > \mu^*$
 denn: $0 > \sum_{i,j \in Z} \text{cost}'(i, j) = \sum \text{cost}(i, j) - \mu \cdot k$ mit $k = \text{länge von } Z$
 $\Rightarrow \mu \cdot k > \sum \text{cost}(i, j) \Rightarrow \mu > \frac{\sum \text{cost}(i, j)}{k}$
2. R' hat keinen Zykel negativer Kosten, aber Zykel der Kosten 0.
 $\Rightarrow \mu = \mu^*$
 0-Zykel hat mittlere Kosten μ^*
3. Alle Zykel haben positive Kosten
 $\Rightarrow \mu < \mu^*$

Im Fall 1 wird μ erniedrigt, im Fall 3 erhöht. Abbruch im Fall 2.

Binäre Suche: Startintervall $[-M, M]$

Beachte: $M \in \mathbb{R}$, Intervalle werden immer kleiner. Abbruch bei Intervallgröße $\leq 1/n^2$

Wie findet man Zykel mit Kosten 0?

Anmerkung: Es gibt keine negativen Zykel.

Berechne Distanzen $d(v)$ für alle v von s aus. Sei $\text{cost}'(i, j) = \text{cost}(i, j) + d(i) - d(j)$ (reduzierte Kosten)

Es gilt: $\text{cost}'(i, j) \geq 0$

$$\sum_{i,j \in Z} \text{cost}(i, j) = \sum \text{cost}'(i, j)$$

In 0-Zykel haben alle Kanten reduzierte Kosten 0.

Also: Wirf alle nicht-0-Kanten weg und prüfe, ob 0-Graph zyklisch ist.

6.6.4 Bellman/Ford:

$$\text{Relax}(v, w): d(w) \leftarrow \min \{d(w), d(v) + c(v, w)\}$$

Algorithmus:

```

d(s) ← d(v) ← ∞ ∀ v ∈ V \ {s}
for i ← 1 to n - 1 do
  for all (v, w) ∈ E do Relax(v, w)
endfor

```

Laufzeit: $\mathcal{O}(m * n)$

Gibt es keinen negativen Zykel, reichen $n - 1$ Phasen aus. $\rightarrow d_1$ -Werte

In weiteren n Phasen werden Kanten entdeckt, die auf negativen Zykel liegen. $\rightarrow d_2$ -Wert

Bei $d_1(v) = d_2(v) \Rightarrow \mu(v) = d_1(v)$, wobei μ die Kosten des billigsten Weges sind.

Wenn $d_2(v) < d_1(v) \Rightarrow \mu(v) = -\infty$

\rightarrow Wie bestimmen wir negative Zykel 'richtig'?

Sei T der kürzeste-Wege-Baum: $\text{Relax}(v, w)$

Normal: $Insert(w, Q)$

Jetzt: Wir wissen: alle Knoten in T_w (Unterbaum an w) haben noch nicht endgültige Distanz
 \rightarrow lösche alle Knoten in T_w aus Q füge w hinzu, w wird Kind von v in T .

Ist w Vorgänger von v , liegt negativer Zykel vor. Alle Knoten, die von v aus erreichbar sind, kommen zu \bar{V} .
 $\bar{V} = \{v | \mu(v) = -\infty\}$

Genauer: T sei Kürzeste-Wege-Baum mit Wurzel s . Für w Kind von v in T , dann $pred(w) = v$.

Ist $pred(w) \neq void$, so wurde w schon mal zu T hinzugefügt. T wird durch eine nach preorder sortierten Liste von Knoten dargestellt, also Wurzel r mit Teilbäumen T_1, \dots, T_k wird dargestellt durch $Liste(T) = r, Liste(T_1), Liste(T_2), \dots$. Außerdem brauchen wir Knotenarray Grad T für Grad und Knotenarray Position T für Positionen von Knoten im Baum.

Für $v \notin T$, $GradT(v) = 0$, $positionT(v) = void$ für $v \in T$ ist $T(PositionT(v)) = v$

Queue Q ist ebenfalls Knotenliste. Position der Knoten in Q durch $PositionQ$ als Knotenarray

Sei $witem$ Variable, die Knoten w in T entspricht. $witem \leftarrow PositionT(w)$

$Deletesubtree(witem)$ löscht Teilbaum T_w aus T raus und gibt nächsten Knoten nach T_w zurück.

Rekursiv:

Falls w hat keine Kinder: Lösche w aus T , eventuell aus Q

Falls w hat Kinder: Lösche Unterbäume rekursiv zuerst den ersten. (direkt nach w in T)

Algorithmus:

Q, T werden mit s initialisiert

Invariante

- $d(v)$ Kosten eines Pfades von s nach v . Falls v in T , ist $d(v)$ die Kosten des Baumpfades von s nach v , $pred(v)$ ist Baumkante zu v .
- War v schon in T , jetzt aber nicht mehr, so ist $\mu(v) < d(v)$
- Q enthält nur Blätter von T . Blätter in k -ter Iteration haben die Tiefe k oder $k + 1$ in T .
- Für negative Zykel gibt es boolesches Kantenarray $InVneg(v)$, mit: $-InVneg(v) = true$ bedeutet $v \in \bar{V}$
 - Enthält Pfad, der $\mu_k(v)$ definiert, einen negativen Zykel, so ist $InVneg(v) = true$ (wobei μ_k kürzester Pfad mit k Kanten)
 - Falls $InVneg(v) = true$ und w von v aus erreichbar, so ist $InVneg(w) = true$.
- Ist $v \in T \setminus Q$, so ist $d(v) + c(v, w) \geq d(w)$ für w mit $InVneg(w) = false$.
- Für v mit $InVneg(v) = false$, gilt: $d(v) \leq \mu_k(v)$

Phase k endet, falls Q keine Knoten der Tiefe k enthält. Ist $Q = \emptyset$, endet der Algorithmus.

Sei v der gerade betrachtete Knoten aus Q mit Tiefe k . v wird aus Q gelöscht. Betrachte Kanten $e = (v, w)$.

Ist $InVneg(w) = true$, ist $\mu(w) = -\infty$. Tue nichts.

- $d(w) \leq d(v) + c(e) \rightarrow$ tue nichts
- $d(w) > d(v) + c(e)$: Lösche w und seine Nachfolger aus T und Q , $d(w) \leftarrow d(v) + c(e)$; $pred(w) = v$.
 - War v nicht in T_w , ist v immernoch in T .
 w wird Kind von v und füge w in Q ein. Füge w direkt hinter v in die Liste T ein, erhöhe Grad von v .
 - War das v in T_w liegt ein negativer Zykel vor (Baumpfad von w nach v plus e)
 Füge alle Knoten, die von v aus erreichbar sind, zu \bar{V} .

\rightarrow Invarianten werden eingehalten. Alle Knoten v mit $\mu(v) = -\infty$ werden gefunden.

Laufzeit: $\mathcal{O}(n * m)$. Es gibt $\leq n$ Phasen und jeder Knoten wird in jeder Phase ≤ 1 mal entfernt.