Algorithmen und Datenstrukturen

Vorlesung #10 – Approximative Algorithmen und Flussgraphen

Benjamin Blankertz

Lehrstuhl für Neurotechnologie, TU Berlin

benjamin.blankertz@tu-berlin.de

24 · Jun · 2020

Themen der heutigen Vorlesung

- **Approximative Algorithmen**
  - Metrisches TSP: gut, aber nicht beliebig gut approximierbar
  - Allgemeines TSP: nicht approximierbar
  - 0/1-Rucksackproblem: beliebig gut approximierbar
  - Approximationsschema
- **Flussgraphen**
  - Schnitte durch Graphen
  - Herausforderungen: maximaler Fluss (*maxflow*), minimaler Schnitt (*mincut*)
  - Fluss-vergrößernde Pfade
  - Allgemeine *maxflow* Methode: **Ford-Fulkerson**
  - Grenzen der Ford-Fulkerson Methode
  - kürzeste Pfade: **Edmonds-Karp** Algorithmus
  - dicke Pfade: ***Capacity Scaling*** Algorithmus

Motivation von Approximativen Algorithmen

- Heuristische Verfahren erlauben es für bestimmte Problemklassen schnelle Lösungen zu generieren.
- Diese Lösungen sind oft nicht optimal, und man bekommt keine Garantie, wie stark die Abweichung vom Optimum maximal ist.
- Eine positive Ausnahme ist der A* Algorithmus, der zumindest bei Verwendung einer konsistenten Heuristik effizient eine *optimale* Lösung findet.

Approximative Algorithmen

- **Konzept:**
  - Approximative Algorithmen begnügen sich mit suboptimalen Lösungen
- **Ziele:**
  - schnellere Laufzeit (auch im worst-case und bei NP-schweren Problemen)
  - Garantie, wie groß die Abweichung vom Optimum maximal ist

ρ-Approximationsalgorithmus

Ein Algorithmus wird (für $\rho \geq 1$) als **ρ-Approximationsalgorithmus** bezeichnet, wenn für seinen Lösungswert C im Vergleich zum optimalen Wert C^* gilt:

$$C \leq \rho C^* \quad \text{für Minimierungs-, bzw.} \quad C \geq C^*/\rho \quad \text{für Maximierungsprobleme}$$

Approximationen für das metrische TSP

- **Metrisches TSP**: Entfernungen zwischen Knoten erfüllen Dreiecksungleichung: $c(u,w) \leq c(u,v) + c(v,w)$ für jeden Knotenfolge $u - v - w$
- Hierfür gibt es effiziente approximative Algorithmen:
- Christofides Algorithmus: Lösung in $\boldsymbol{O}(V^4)$ mit höchstens 1.5-mal den Kosten des Optimums. Dies ist die beste bekannte Approximation.
- Es kann unter P≠NP keine beliebig guten Approximationen geben. Bislang ist $\frac{123}{122}$ als schärfste Schranke bekannt [Karpinski et al, 2015].
- Hier: einfache Variante mit höchsten 2 mal den Kosten des Optimums (2-Approximationsalgorithmus)

Pseudocode zur 2-Approximation des metrischen TSP

- Dieser Ansatz nimmt einen minimalem Spannbaum als Ausgangsbasis.
- Dieser wird in eine TSP-Tour verwandelt.

Listing 1: **TSP-Tour mit maximal dem doppelten der optimalen Kosten.**

```
1 APPROX-TSP-TOUR(G)
2 bestimme einen minimalen Spannbaum T von G mit bel. Startknoten s
3 sei H die Liste der Knoten von T in Nebenreihenfolge
4 H stellt als Zyklus eine TSP−Tour dar
```

- Zur Erinnerung aus VL #7:
  Nebenreihenfolge = Reihenfolge des Entdecktwerdens in DFS

Korrektheit und Approximation von Approx-TSP-Tour

Approx-TSP-Tour liefert eine korrekte TSP-Tour (jeder Knoten wird genau einmal besucht) in einer Laufzeit in $\boldsymbol{O}(V^2 \log V)$.
Die Tour hat maximal die zweifache Länge einer optimalen Tour.

Beweis.

- Die Laufzeit von Prim für den MST ist in $\boldsymbol{O}(E \log V)$. Die anschließende Tiefensuche zur Sortierung der Knoten ist in $\boldsymbol{O}(V^2)$. Da in der vorliegenden Situation $E \in \boldsymbol{O}(V^2)$ gilt, ist die Laufzeit wie behauptet.
- Eine TSP-Tour wird durch Entfernung einer beliebigen Kante zu einem Spannbaum, also ist die Länge des MST eine untere Schranke für die Länge der optimalen TSP-Tour TSP^*:

$$c(MST) \leq c(TSP^*)$$

2-Approximation für das metrische TSP

- Außerdem kann der MST in einen Zyklus umgewandelt werden, der fast eine (suboptimale) TSP-Tour darstellt:
- Dazu betrachten wir die vollständige Traversierung des MST:
  Bei einer Tiefensuche werden Knoten sowohl bei dem Besuch (Eingangsstempel), als auch beim Verlassen (Ausgangsstempel) in einer Liste gespeichert.
- Diese Traversierung W passiert jede Kante des MST zweimal.
- W bildet einen Zyklus (da DFS zur Wurzel zurückkehrt) und es gilt

$$c(W) \;=\; 2c(MST) \;\leq\; 2c(TSP^*)$$

- Allerdings ist der Zyklus W noch keine zulässige TSP Tour, da die Knoten mehrfach (genauer gesagt doppelt) besucht werden.
- Dies kann durch einen kleinen Umbau des Zyklus behoben werden.

2-Approximation für das metrische TSP

- Bei dem Durchlaufen von W wird jeder Knoten, der vorher schon besucht wurde, ausgelassen. Die resultierende Tour T besucht jeden Knoten nur einmal.
- Durch das Überspringen eines Knotens wird die Länge des Zyklus nicht länger:
- Sei $u - v - w$ eine Knotenfolge des ursprünglichen Zyklus, aus der der Knoten v ausgelassen wird. Wegen der Dreiecks-Ungleichung gilt:

$$c(u,w) \leq c(u,v) + c(v,w)$$

- Daher erhalten wir insgesamt die Abschätzung

$$c(T) \leq c(W) = 2c(MST) \leq 2c(TSP^*)$$

- Da die Tour T genau der Rückgabe von Approx-TSP-Tour entspricht, ist damit die Behauptung bewiesen. □

Nicht-Approximierbarkeit des allgemeinen TSP

- Nun folgt das ernüchternde Resultat, dass es für das allgemeine Handlungsreisendenproblem (also ohne die Voraussetzung der Dreiecksungleichung) gar keine Approximation geben kann.

Das allgemeine Handlungsreisendenproblem ist nicht approximierbar

Unter der Voraussetzung P≠NP gibt es für kein $\rho \geq 1$ einen ρ-Approximationsalgorithmus mit polynomieller Laufzeit für das allgemeine TSP.

Beweis.

- Wir nehmen an, dass es für ein $\rho \geq 1$ einen ρ-Approximationsalgorithmus $\mathcal{X}$ mit polynomieller Laufzeit für das TSP gibt.
- Da die Approximation dann auch für alle größeren Zahlen gilt, können wir ρ als ganzzahlig voraussetzen (z.B. durch Aufrunden).
- Wir zeigen, dass sich mit $\mathcal{X}$ auch das Hamilton-Zyklus Problem lösen lässt.

Nicht-Approximierbarkeit des allgemeinen TSP

- Sei ein Graph $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ gegeben, für den die Existenz eines Hamilton-Kreis festgestellt werden soll.

- Wir definieren den vollständigen und gewichteten Graphen $\boldsymbol{G'} = (\boldsymbol{V}, \boldsymbol{E'}, c)$ durch

$$\boldsymbol{E'} = \{(v, w) \in \boldsymbol{V} \times \boldsymbol{V} \mid v \neq w\}$$

$$c(v, w) = \begin{cases} 1 & \text{falls } (v, w) \in \boldsymbol{E} \\ \rho V + 1 & \text{sonst} \end{cases}$$

- Offensichtlich kann $\boldsymbol{G'}$ in polynomieller Zeit in V und E aus $\boldsymbol{G}$ erstellt werden.

- $\boldsymbol{G'}$ ist so definiert, dass der ρ-Approximatiosnalgorithmus $\mathcal{X}$ für das TSP in $\boldsymbol{G'}$ die Frage nach dem Hamilton-Zyklus in $\boldsymbol{G}$ beantwortet!

Nicht-Approximierbarkeit des allgemeinen TSP

- Die Kosten für die optimale TSP-Tour unterscheiden sich um mehr als den Faktor ρ, abhängig davon, ob ein Hamilton-Zyklus in G exisitert oder nicht.

- Daher könnte der ρ-Approximationsalgorithmus $\mathcal{X}$ die Hamilton-Frage entscheiden:

- Wir wenden $\mathcal{X}$ auf das TSP G' an und unterscheiden nach den Kosten der gefundenen Tour T:

1 $c(T) \leq \rho V$: Die Tour enthält nur Kanten, die zu G gehören, da alle anderen Kanten Kosten $\rho V + 1$ haben. Daher stellt die TSP-Tour T einen Hamilton-Zyklus in G dar.

2 $c(T) > \rho V$: Die optimale Tour kann sich maximal um den Faktor ρ unterscheiden, hat also Kosten $> V$. Da ein Hamilton-Zyklus in G eine TSP Tour mit Kosten V wäre, kann es diesen nicht geben, wenn die optimale TSP Tour Kosten $> V$ hat.

Approximativer Ansatz für das 0/1-Rucksack Problem

- Mittels dynamischer Programmierung konnte ein Algorithmus für das 0/1-Rucksackproblem mit pseudopolynomieller Laufzeit formuliert werden.
- Die Laufzeit in $\boldsymbol{O}(KW)$ kann für große Kapazitätsgrenzen W sehr schlecht sein.
- Mit einem approximativen Algorithmus kann man effizient Lösungen finden, die beliebig nah am Optimum sind.
- Wenn im Rahmen von dynamischer Programmierung die Laufzeit verbessert werden soll, muss die Tabelle verkleinert werden.
- Im Sinne einer approximativen Lösung ist dies im Prinzip durch eine Skalierung einer Variable der OPT Funktion möglich.
- Die Anzahl der Objekte kann natürlich nicht skaliert werden. Aber auch eine Skalierung der Kapazitätsgrenze ist nicht zielführend, da die Kapazitätsgrenze exakt eingehalten werden muss.
- Im Gegensatz dazu können die Werte der Objekte skaliert werden. Daher erstellen wir nun einen neuen dynamischen Programmier-Ansatz, bei dem die OPT Funktion von dem zu erreichenden Wert abhängt.

0/1-Rucksack Problem – Duales Dynamisches Programm

- Wir definieren eine rekursive Funktion $\textsc{Opt}(k, v)$, die angibt wieviel Gewicht mindestens notwendig ist, um mit einer Auswahl aus Objekten $1, \ldots, k$ einen vorgegebenen Wert v (oder mehr) zu erreichen und ∞, falls der Wert v gar nicht mit den Objekten erreicht werden kann.
- V sei die Summe aller Werte: $V = \sum_{k=1}^{K} v_k$.
- Der Definitionsbereich von $\textsc{Opt}$ ist $0 \leq k \leq K$ und $0 \leq v \leq V$.
- Den Wert $v = 0$ erreicht man ohne Objekte, also ist die Gewichtsgrenze $\textsc{Opt}(k, 0) = 0$ für alle k.
- Mit $k = 0$ Objekten, kann man keinen Zielwert $v > 0$ erreichen.

$$\textsc{Opt}(k, v) = \begin{cases} 0 & \text{falls } v = 0 \\ \infty & \text{falls } k = 0 \,\&\, v > 0 \\ \max\Big(\underbrace{w_k + \textsc{Opt}(k-1, \max(0, v - v_k))}_{k \text{ ausgewählt}}, \underbrace{\textsc{Opt}(k-1, v)}_{k \text{ nicht ausgewählt}} \Big) & \text{sonst} \end{cases}$$

Analyse der dualen 0/1-Rucksack Lösung

Korrektheit und Laufzeit der dualen 0/1-Rucksack Lösung

Der Algorithmus basierend auf dynamischer Programmierung mit der $\textsc{Opt}$ Funktion von der vorigen Seite findet die optimale Lösung des 0/1-Rucksack Problems mit ganzzahligen Werten und Gewichten in einer Laufzeit in $\boldsymbol{O}(KV)$, wobei V die Summe der Werte aller Objekte ist: $V = \sum_{k=1}^{K} v_k$.

- Die Tabelle, die für die dynamische Programmierung benötigt wird, hat die Größe $(K+1)(V+1)$.
- Das Berechnen jedes Tabelleneintrags kann gemäß der $\textsc{Opt}$ Funktion in konstanter Zeit ausgeführt werden.
- Insgesamt kann also die Tabelle in einer Laufzeit in $\boldsymbol{O}(KV)$ bestimmt werden. □

Analyse der dualen 0/1-Rucksack Lösung

- Nachtrag: Der Lösungswert steht nicht unbedingt am Ende der Tabelle in dem Eintrag (K, V), sondern er muss aus der Tabelle herausgesucht werden.
- Man sucht den größten Wert v, für den $OPT(K, v) \leq W$ gilt.

- **Vergleich der Lösungsvarianten für das Rucksackproblem:**

- Laufzeit von Variante 1 mit $\textsc{Opt}(k, W)$ ist in $\boldsymbol{O}(KW)$.
- Laufzeit von Variante 2 mit $\textsc{Opt}(k, v)$ ist in $\boldsymbol{O}(KV)$. Für den maximalen Wert der Objekte $\bar{v} = \max_k(v_k)$ erhalten wir die grobere Abschätzung $\boldsymbol{O}(K^2\bar{v})$.
- Welche Variante effizienter ist, hängt von der Kapazitätsgrenze und der Größe der Werte ab.
- Die zweite Variante hat den Vorteil, dass sie die Grundlage für ein Approximationsschema liefert.

Approximationsschema

Approximationsschema

Ein **Approximationsschema** für ein Optimierungsproblem ist ein Algorithmus, der zu jeder Eingabe mit optimalem Wert C^* und jedem $\varepsilon > 0$ eine Lösung mit Wert C liefert, wobei $(1-\varepsilon)C^* \leq C \leq (1+\varepsilon)C^*$ gilt.

- Von den angegebenen Grenzen ist die untere für Maximierungs- und die obere für Minimierungsprobleme relevant.
- Nun ist nicht nur interessant, wie sich die Laufzeit in Abhängigkeit von der Eingabe verhält, sondern auch, wie die Laufzeit von ε abhängt.
- Man nennt ein Approximationsschema ein **Approximationsschema mit vollständig polynomieller Laufzeit**, falls seine Laufzeit polynomiell in der Größe der Eingabe und in $\frac{1}{\varepsilon}$ ist.

Approximationsschema für das 0/1-Rucksack Problem

- Seien K Objekte mit Gewichten w_k und Werten v_k sowie eine Kapazitätsgrenze W gegeben. Sei $\bar{v} = \max_k(v_k)$ der maximale Wert eines Objektes.
- Zu der gegebenen Approximationsgüte $\varepsilon > 0$ definieren wir $E = \varepsilon \frac{\bar{v}}{K}$.
- Wir gehen davon aus, dass es keine Objekte mit einem Gewicht $w_k > W$ gibt. Dies ist keine Einschränkung, weil solche Objekt sowieso nicht ausgewählt werden könnten.
- Wir benutzen E als Skalierungsfaktor und berechnen neue Werte $v'_k = \lfloor \frac{v_k}{E} \rfloor = \lfloor \frac{K}{\varepsilon} \frac{v_k}{\bar{v}} \rfloor$. Die skalierten Werte liegen also zwischen 0 und $\frac{K}{\varepsilon}$.
- Nun wird der zuvor skizzierte Algorithmus auf das Problem mit den skalierten Werten v'_k angewendet.

Approximationsschema für den 0/1-Rucksack

Der oben beschriebene Algorithmus ist ein Approximationsschema mit vollständig polynomieller Laufzeit.

Approximationsgüte

- Sei S eine optimale Lösung für das Originalproblem die Wert $V^* = \sum_{k \in S} v_k$ erzielt und S' eine optimale Lösung für die skalierten Werte v'_k (vom Approx.-Alg.).
- Da S' optimal für die skalierten Werte v'_k ist, erhalten wir:

$$\sum_{k \in S'} v'_k \;\geq\; \sum_{k \in S} v'_k \;=\; \sum_{k \in S} \lfloor \frac{v_k}{E} \rfloor \;\geq\; \sum_{k \in S} (\frac{v_k}{E} - 1) \;\geq\; \frac{V^*}{E} - K$$

- Nun können wir die Werte der Lösung des Approximations-Algorithmus bezogen auf die Originalwerte abschätzen:

$$\sum_{k \in S'} v_k \;\geq\; \sum_{k \in S'} v'_k E \;\geq\; (\frac{V^*}{E} - K)E \;=\; V^* - KE \;\geq\; V^*(1 - \varepsilon)$$

  wobei die letzte Ungleichung mit $\bar{v} \leq V^*$ aus $E \;=\; \varepsilon \frac{\bar{v}}{K} \;\leq\; \varepsilon \frac{V^*}{K}$ folgt.

- Laufzeit: Die Originaltabelle hat die Größe $K \times V \;\leq\; K \times K\bar{v}$, also ist skalierte Tabelle kleiner oder gleich $K \times \frac{K\bar{v}}{E} \;=\; K \times \frac{K\bar{v}K}{\varepsilon\bar{v}} \;=\; K \times \frac{K^2}{\varepsilon}$.
- Somit ist die Laufzeit in $\boldsymbol{O}(\frac{K^3}{\varepsilon})$.

Resümee zu Approximativen Algorithmen

- Bei der Approximation von NP-vollständigen Optimierungsproblemen kann folgendes passieren:

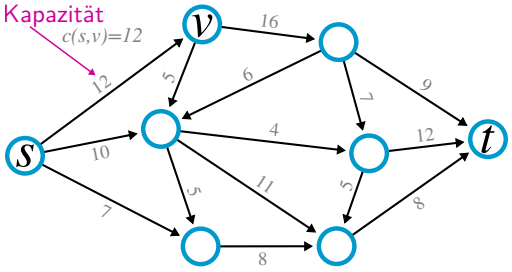
1. Das Problem lässt sich überhaupt nicht approximieren, egal wie lax die Approximationsgüte angesetzt wird. Beispiel: TSP ohne Dreiecksungleichung
2. Das Problem lässt sich approximieren, aber nur bis zu einer gewissen Grenze. Beispiel: TSP mit Dreiecksungleichung
3. Das Problem kann beliebig gut approximiert werden. Beispiel: 0/1-Rucksack

- Bei dem letzten Punkt unterscheidet man noch danach, wie stark die Approximationsgüte die Laufzeit beeinträchtigt.
- Im guten Fall ist die Laufzeit polynomiell in Eingabegröße und $\frac{1}{\varepsilon}$.

Flussgraphen

- Ein **Flussgraph** ist ein gewichteter Digraph $G = (V, E)$. Die Gewichte werden als **Kapazitäten** bezeichnet und sind positiv.

- Wir schreiben $c(v, w)$ für die Kapazität der Kante $v \to w$ und definieren $c(v, w) = 0$ für $v \to w \notin E$.



- Weitere Voraussetzung: Es gibt eine ausgezeichnete **Quelle** s (Knoten mit Eingangsgrad 0) und eine ausgezeichnete **Senke** t (Knoten mit Ausgangsgrad 0).

- Man kann sich die Kanten als Leitungen vorstellen, durch die eine Flüssigkeit fließt.

- Ebenso kann z.B. der Fluss von Informationen durch Netzwerke modelliert werden.

Definition Fluss

- Ein **Fluss** (*flow*) ordnet jeder Kante des Digraphen einen Fluss zu, mittels einer Funktion

  $$f : \boldsymbol{V} \times \boldsymbol{V} \to \mathbb{R} \quad \text{(wobei } f(v,w) = 0 \text{ für } v \to w \notin \boldsymbol{E} \text{ gesetzt wird)},$$

  die die folgenden beiden Bedingungen erfüllt:
- **Kapazitätsbeschränkung:** (*capacity constraint*) Der Fluss jeder Kante ist positiv und höchstens gleich der Kapazität der Kante:

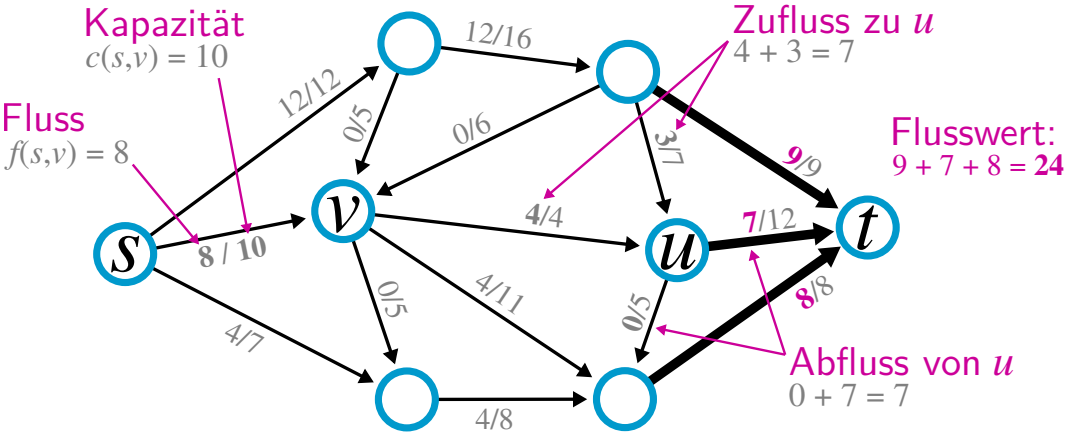
  $$\forall v, w \in \boldsymbol{V} : \quad 0 \leq f(v,w) \leq c(v,w)$$

- **Flusserhaltung:** (*local equilibrium*) Für jeden Knoten außer Quelle und Senke ist der **Zufluss** (Summe vom Fluss der Kanten nach v) gleich dem **Abfluss**:

  $$\forall v \in \boldsymbol{V} - \{s,t\} : \quad \sum_{w \in \boldsymbol{V}} f(w,v) = \sum_{w \in \boldsymbol{V}} f(v,w)$$

- Der **Wert des Flusses** ist definiert als der Zufluss zur Senke: $|f| = \sum_{v \in \boldsymbol{V}} f(v,t)$.

Darstellung eines Flussgraphen mit einem Fluss



- **Herausforderung:**
  Finde einen Fluss mit maximalem Wert (maximaler Fluss, *maxflow*)!

Strategie zur Bestimmung des maximalen Flusses

1. Startet man mit einem 0-Fluss ($f(v, w) = 0$ für alle v, w).

2. Suche iterativ Pfade von s nach t, entlang derer der Fluss erhöht werden kann.

- Dazu führen wir die augmentierenden, bzw. Fluss vergrößernden Pfade ein.

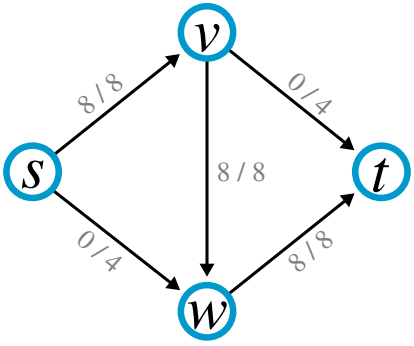
Fluss vergrößernde Pfade

- Ein **(Fluss) vergrößernder Pfad** (*augmenting path*) ist ein ungerichteter Pfad von s nach t, durch den der Flusswert vergrößert werden kann.
- Dabei bedeutet *ungerichtet*, dass eine gerichtete Flusskante auch in Gegenrichtung benutzt werden kann. Trotzdem denkt man den Pfad in Richtung von s nach t.
- Um den Flusswert vergrößern zu können, müssen zwei Bedingungen erfüllt sein:

1. Bei Kanten in Pfadrichtung ist die Kapazität nicht ausgeschöpft. Hier kann der Fluss vergrößert werden.
2. Bei Kanten gegen Pfadrichtung ist der Fluss größer als 0. Hier kann der Fluss reduziert werden.

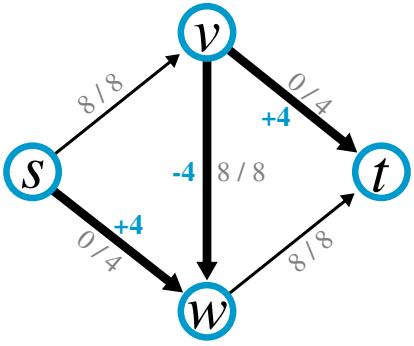
- Der kritische Wert ist der kleinste Wert, um den der Fluss entlang des Pfades
  - auf Kanten in Pfadrichtung erhöht und
  - auf Kanten gegen Pfadrichtung reduziert werden kann.
- Da die letzte Kante zu t in Pfadrichtung geht (da t eine Senke ist), wird der Flusswert um den kritischen Wert des Pfades erhöht.

Kleines Beispiel zur Flussumplanung



- Es wird ein vergrößernder Pfad gewählt
- und der Fluss entsprechend um 8 Einheiten erhöht.

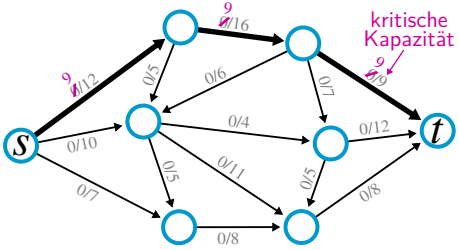
Kleines Beispiel zur Flussumplanung



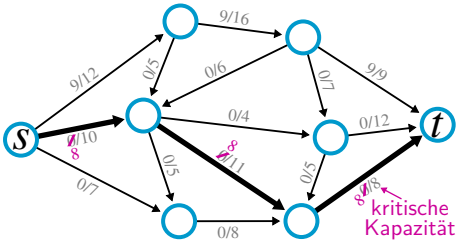
- Zur weiteren Erhöhung um 4 Einheiten, muss der Fluss von v nach w umgeplant werden.
- Der Fluss wird hier von 8 auf 4 vermindert (Kante $v \to w$ gegen Pfadrichtung)
- Auf den Kanten in Pfadrichtung ($s \to w$ und $v \to t$) wird der Fluss um 4 erhöht.
- Insgesamt entspricht dies einer Erhöhung des Flusses entlang des Pfades s–w–v–t um 4.

Vergrößernde Pfade

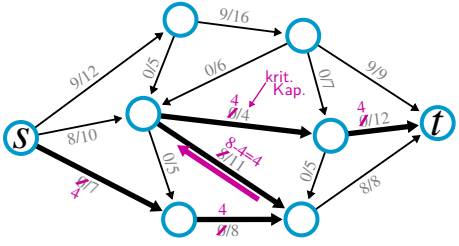
1. Vergrößernder Pfad



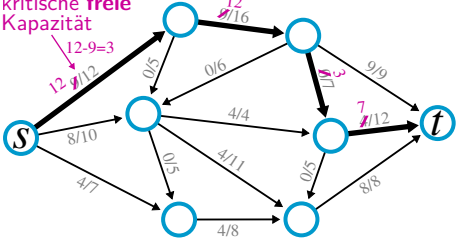
2. Vergrößernder Pfad



3. Vergrößernder Pfad



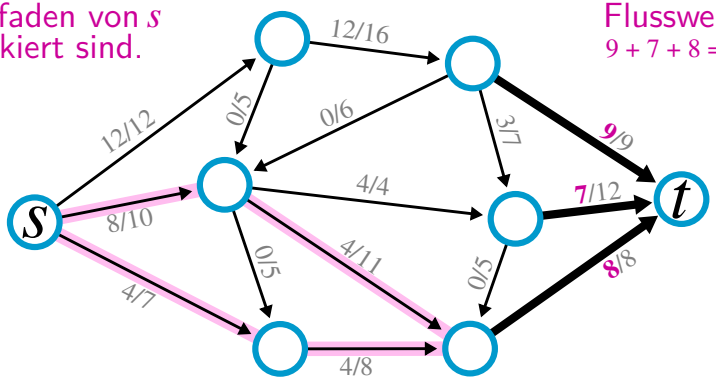
4. Vergrößernder Pfad



Maximaler Fluss

- Wenn kein vergrößernder Pfad mehr existiert, ist der maximale Fluss gefunden.
  Beweis siehe Skript.
- Dies ist der Fall, wenn jeder Pfad von s nach t blockiert ist
  - durch eine Kante in Pfadrichtung ohne freie Kapazität $(f(v,w) = c(v,w))$ oder
  - durch eine Kante gegen Pfadrichtung ohne Fluss $(f(v,w) = 0)$.
- Dann ist Zufluss zu der Senke der maximale Fluss.

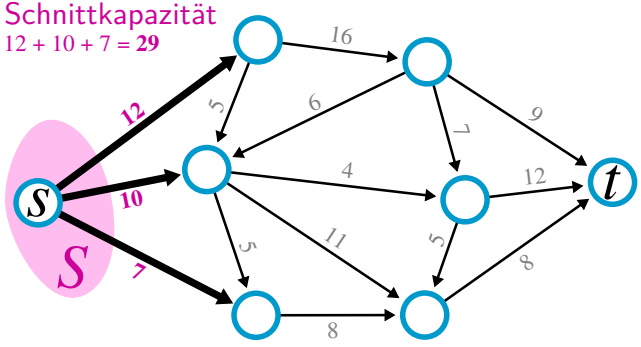
Kanten auf Pfaden von s die nicht blockiert sind.



Flusswert:
9 + 7 + 8 = **24**

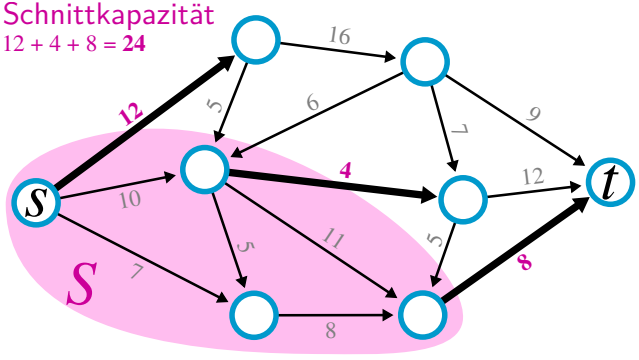
Schnitte und minimaler Schnitt

- Ein s von t trennender **Schnitt** (*cut*) teilt die Knoten eines Flussgraphen in zwei zusammenhängende, nicht-leere Teilmengen $\boldsymbol{S}$ und $\boldsymbol{T} = \boldsymbol{V} - \boldsymbol{S}$, wobei die Quelle in $\boldsymbol{S}$ und die Senke in $\boldsymbol{T}$ ist: $s \in \boldsymbol{S}$, $t \in \boldsymbol{T}$.
- Die **Kapazität eines Schnittes** ist die Summe der Kapazitäten der kreuzenden Kanten die $\boldsymbol{S}$ verlassen. Kanten, die nach $\boldsymbol{S}$ hereinführen werden **nicht** gezählt.



Schnitte und minimaler Schnitt

- Ein s von t trennender **Schnitt** (*cut*) teilt die Knoten eines Flussgraphen in zwei zusammenhängende, nicht-leere Teilmengen S und $T = V - S$, wobei die Quelle in S und die Senke in T ist: $s \in S$, $t \in T$.
- Die **Kapazität eines Schnittes** ist die Summe der Kapazitäten der kreuzenden Kanten die S verlassen. Kanten, die nach S hereinführen werden **nicht** gezählt.



Herausforderung:
Finde einen Schnitt mit minimaler Kapazität (minimaler Schnitt, *mincut*)!

Bemerkung:
(Minimale) Schnitte werden für beliebige gewichtete Graphen betrachtet, nicht nur für Flussgraphen.

Fluss über einen Schnitt

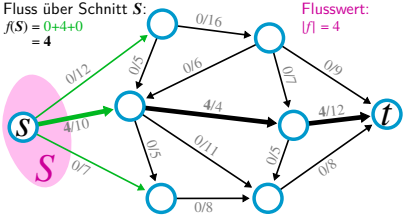
- Wir definieren als **Fluss über einen Schnitt** $f(S)$ zu gegebenem Fluss f und Schnitt S die Summe über den Fluss aller kreuzenden Kanten. Dabei werden Kanten aus S positiv und Kanten nach S negativ gerechnet.

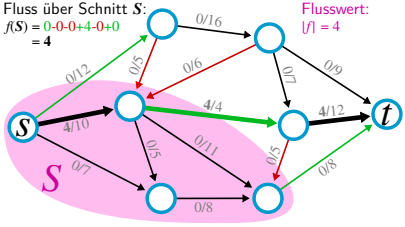
Fluss über einen Schnitt

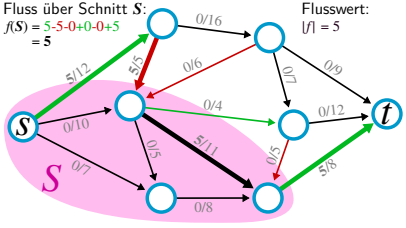
Schnitttheorem: Zusammenhang von Flüssen und Schnitten

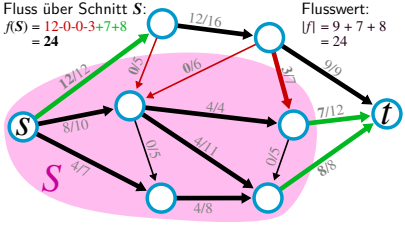
Der Fluss ist über alle Schnitte derselbe. Er entspricht immer dem Flusswert.

Zunächst betrachten wir ein paar Beispiele:

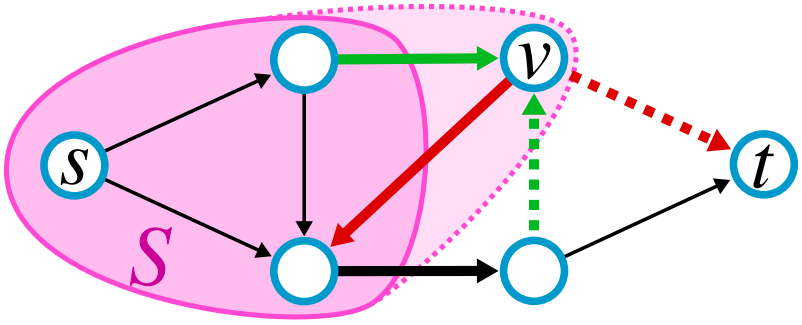








Induktiver Beweis der Gleichheit des Flusses über alle Schnitte



- Es fällt weg (rot/grün gestrichelt): Fluss der Kanten von v nach $\boldsymbol{T}$ und der negativ gewichtete Fluss der Kanten von $\boldsymbol{T}$ nach v.
- Es kommt hinzu (rot/grün durchgezogen): Fluss der Kanten von $\boldsymbol{S}$ nach v und der negativ gewichtete Fluss der Kanten von v nach $\boldsymbol{S}$.
- In Summe kommt also der Zufluss nach v dazu (grün), und es wird der Abfluss von v abgezogen (rot). Nach der Flusserhaltungsbedingung (S. 22) ergibt dies 0. □

Alternativer, rechnerischer Beweis

- Der Sachverhalt kann auch direkt, ohne Induktion, bewiesen werden.
- Wir nehmen die Definition des Fluss eines Schnittes und addieren den folgenden Term, der 0 ergibt, da beide Summen über alle Kanten innerhalb $\boldsymbol{T}$ laufen:

$$\sum_{\substack{v\to w\in \boldsymbol{E}\\ v\in \boldsymbol{T},\, w\in \boldsymbol{T}}} f(v,w) - \sum_{\substack{w\to v\in \boldsymbol{E}\\ v\in \boldsymbol{T},\, w\in \boldsymbol{T}}} f(w,v)$$

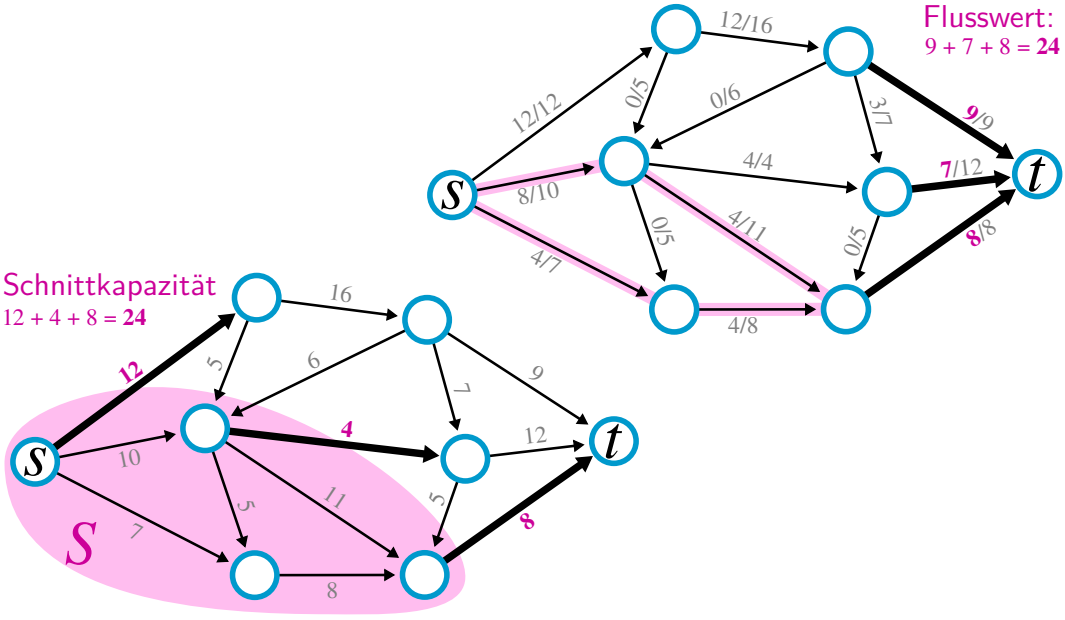
- Auf diese Weise erhalten wir nach Umsortieren der Summanden:

$$f(\boldsymbol{S},\boldsymbol{T}) = \sum_{\substack{v\to w\in \boldsymbol{E}\\ v\in \boldsymbol{S},\, w\in \boldsymbol{T}}} f(v,w) - \sum_{\substack{w\to v\in \boldsymbol{E}\\ v\in \boldsymbol{S},\, w\in \boldsymbol{T}}} f(w,v) \quad \text{Definition Fluss über Schnitt}$$

$$= \sum_{\substack{v\to w\in \boldsymbol{E}\\ v\in \boldsymbol{V},\, w\in \boldsymbol{T}}} f(v,w) - \sum_{\substack{w\to v\in \boldsymbol{E}\\ v\in \boldsymbol{V},\, w\in \boldsymbol{T}}} f(w,v) \quad \text{Addition von obigem Term, } \boldsymbol{V} = \boldsymbol{S} \cup \boldsymbol{T} \text{ und Umsortieren}$$

$$= \sum_{w\in \boldsymbol{T}} (\text{Zufluss zu } w - \text{Abfluss von } w) = |f| \quad \text{Wegen Flusserhaltung S. 22 bleibt nur der Term für } t.$$

Zusammenhang: Maximaler Fluss und minimaler Schnitt



Maximaler Fluss und minimaler Schnitt durch vergrößernde Pfade

Vergrößernde Pfade und maximaler Fluss

Ein Fluss f ist genau dann maximal, wenn es keine vergrößernden Pfade gibt.

- Zum Beweis (siehe Skript) wird die Äquivalenz der folgenden drei Aussagen für einen Fluss f gezeigt:

1 Es gibt einen Schnitt, dessen Kapazität mit dem Wert von f übereinstimmt.

2 f ist ein maximaler Fluss.

3 Es gibt keinen vergrößernden Pfad für f.

- Die Äquivalenz von **1** und **2** ergibt auch folgenden Sachverhalt:

Maximaler Fluss und minimaler Schnitt

Der Wert des maximalen Flusses entspricht der Kapazität des minimalen Schnittes.

Allgemeine Methode zum Identifizieren des maximalen Flusses

Von **Ford-Fulkerson** wurde die Technik der vergrößernden Pfade entwickelt, um eine allgemeine Methode zum Identifizeren des maximalen Flusses in Flussgraphen anzugeben:

```
1 for each e in E
2   f(e) ←0
3 end
4 while es gibt einen vergrößernden Pfad p in (G, f) do
5   cv ←kritischer Wert von f
6   vergrößere f entlang p um cv
7 end
```

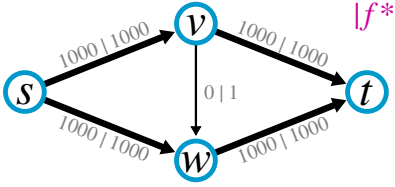
- Mit "*vergrößere f entlang p um cv*" ist das auf Seite 25 beschriebene Verfahren der vergrößernden Pfade gemeint:
  - Auf Kanten in Richtung des Pfades p wird der Fluss um cv erhöht und
  - auf Kanten gegen Richtung des Pfades p wird der Fluss um cv reduziert.

Korrektheit und Laufzeit der Ford-Fulkerson Methode

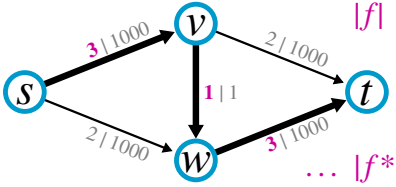
- **Korrektheit:** Wenn die allgemeine Ford-Fulkerson Methode terminiert, wissen wir nach dem Satz über vergrößernde Pfade (Seite 37), dass das Ergebnis ein maximaler Fluss ist.
- Bevor wir die Laufzeit diskutieren, die die Terminierung impliziert, besprechen wir Beispiele, die der Methode Schwierigkeiten bereiten.
- Dabei ist zu beachten, dass bisher keine Strategie zur Auswahl der vergrößernden Pfade spezifiziert wurde.
- Die Beispiele beruhen auf einer 'unglücklichen' Reihenfolge.

Kleiner Flussgraph mit potenziell langer Laufzeit

- Der abgebildete Flussgraph hat den maximalen Fluss von $|f^*| = 2.000$, der durch die Kombination der beiden Pfade $s \to v \to t$ und $s \to w \to t$ mit jeweils 1.000 Einheiten erreicht wird.
- Eine unglückliche Wahl der vergrößernden Pfade ist ein Wechsel von $s \to v \to w \to t$ und $s \to w \to v \to t$.
- Diese Pfade haben beide den kritischen Wert 1, so dass insgesamt 2.000 Iterationen nötig sind, um den maximalen Fluss $|f^*|$ zu erzeugen.
- Dieses Beispiel zeigt auch, dass die Laufzeit der Ford-Fulkerson Methode, sofern kein geeignetes Verfahren zur Pfadauswahl angegeben wird, nicht nur von der Struktur des Graph, sondern auch von seinen Kapazitäten abhängen kann.



$|f^*| = 2000$



$|f| = 5$

$\ldots \; |f^*| = 2000$

Kleiner Graph ohne Terminierung

- Es gibt keine Garantie, dass die Ford-Fulkerson Methode überhaupt terminiert.
- Dies kann (nur) bei irrationalen Kapazitäten passieren, siehe Anhang, Folie 49.
- Wir beschränken uns daher auf rationale Kapazität und oBdA sogar auf Kapazitäten in $\mathbb{N}^{>0}$.
  - Beliebige rationale Zahlen können mit dem KGV aller Nenner multipliziert werden, um einen äquivalenten Flussgraphen mit Kapazitäten in $\mathbb{N}^{>0}$ zu definieren.

Laufzeit der Allgemeinen Ford-Fulkerson Methode

Laufzeit der Ford-Fulkerson Methode

Die Ford-Fulkerson Methode benötigt für einen Flussgraphen, dessen Kapazitäten natürliche Zahlen sind, eine Laufzeit in $\boldsymbol{O}(E|f^*|)$, wobei f^* der maximale Fluss ist.

- Alternativ kann die Laufzeit der allgemeinen Ford-Fulkerson Methode als $\boldsymbol{O}(EVC)$ angegeben werden, wobei C eine obere Schranke für die Kapazitäten ist, da $|f^*| \leq VC$.

- Um eine Laufzeitschranke zu erzielen, die nur von der Größe des Graphen abhängt, muss man eine spezielle Strategie zur Auswahl des vergrößernden Pfades anwenden.

- Folgende Strategien scheinen plausibel:
  - Wähle einen vergrößernden Pfad mit wenigen Kanten
  - Wähle einen vergrößernden Pfad mit großem Fluss (großem kritischen Wert)

Der Edmonds-Karp Algorithmus (Pfad mit wenigen Kanten)

- Der **Edmonds-Karp Algorithmus** wählt als vergrößernden Pfad in der Ford-Fulkerson Methode einen Pfad, der die wenigsten Kanten hat.
- Man fängt mit einem leeren Fluss f an. Der Fluss wird iterativ vergrößert:
- Wähle Pfad von s nach t im sogenannten Restgraphen G_f mit Breitensuche (geringste Kantenanzahl).
- Die Details werden in diesem Jahr nicht besprochen, stehen aber im Skript.

Laufzeit des Edmonds-Karp Algorithmus

Der Edmonds Karp Algorithmus bestimmt den maximalen Fluss eines Flussgraphen in einer Laufzeit von $O(E^2V)$.

Verbesserungen der Laufzeit von Edmonds-Karp

- Bisher: $\boldsymbol{O}(VE)$ viele Flussvergrößerungen, jeweils $\boldsymbol{O}(E)$, insgesamt $\boldsymbol{O}(E^2V)$.
- Es gibt Beispiele für Flussgraphen, bei denen die Anzahl der notwendigen Flussvergrößerungen tatsächlich in $\boldsymbol{\Theta}(VE)$ liegt, wenn immer ein kürzester vergrößernder Pfad gewählt wird. An dieser Schranke ist also in Edmonds-Karp nichts zu verbessern.
- Es kann aber die benötigte Zeit für Flussvergrößerungen reduziert werden.
- Der *blocking-flow* Algorithmus wurde in [Dinic 1970] vorgeschlagen, also *vor* der Veröffentlichung von Edmonds-Karp.
- Dabei werden Pfade in einem 'Niveaugraphen' schrittweise aktualisiert, um jeweils den nächsten vergrößernden Pfad effizienter zu finden.
- Auf diese Weise lässt sich eine Laufzeit in $\boldsymbol{O}(EV^2)$ erreichen.
- Mit dynamischen Bäumen [Sleator & Tarjan 1983] kann sogar eine Laufzeit in $\boldsymbol{O}(EV \log V)$ erzielt werden.

Der Kapazitätskontrolle Algorithmus (Pfad mit großem Fluss)

- Alternative Strategie zur Auswahl der vergrößernden Pfade:
- Wähle einen Pfad, der den Fluss maximal vergrößert.
- Dies wurde auch von Edmonds und Karp vorgeschlagen. Es fehlt aber eine effiziente Implementierung.
- Geben wir uns also mit weniger zufrieden: Der vergrößernde Pfad erhöht den Fluss nicht maximal, aber relativ stark.
- Parameter Δ zum ***Capacity Scaling***: Wähle nur Pfade mit einem Fluss $\geq \Delta$.
- Wenn es keine solchen Pfade mehr gibt, halbiere Δ und iteriere.

Laufzeit des *Capacity Scaling* Algorithmus

Laufzeit des *Capacity Scaling* Algorithmus

Der *Capacity Scaling* Algorithmus bestimmt den maximalen Fluss in einer Laufzeit in $\boldsymbol{O}(E^2 \log C)$. Dabei ist C die maximale Kapazität des Flussgraphen.

- Die Details werden in diesem Jahr nicht besprochen, stehen aber im Skript.

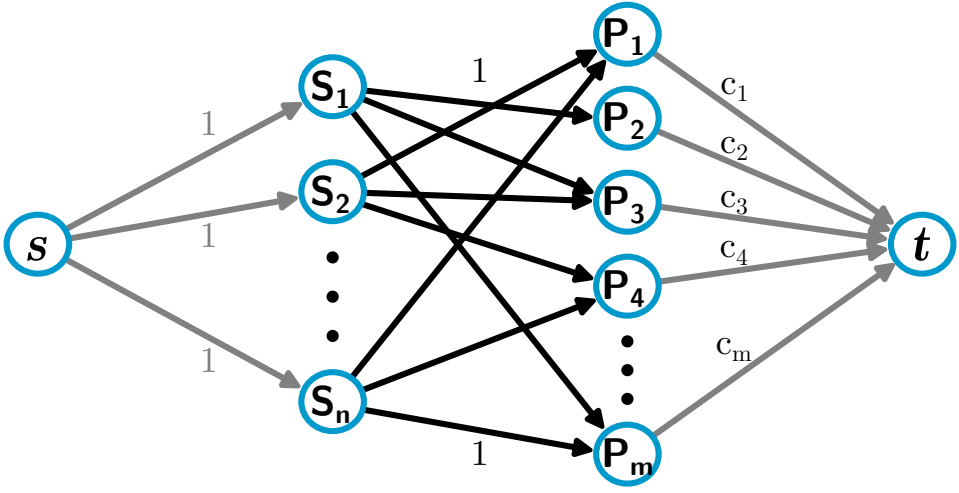
Laufzeiten von *maxflow* Algorithmen

Die folgende Tabelle zeigt eine Übersicht über die Laufzeiten von *maxflow* Algorithmen für Flussgraphen mit ganzzahligen Kapazitäten.

Algorithmen zum Finden des Maximalen Flusses		
Algorithmus	***worst-case***	**alternativ**
Ford-Fulkerson	$\boldsymbol{O}(E\|f^*\|)$	$\boldsymbol{O}(EVC)$
Edmonds-Karp	$\boldsymbol{O}(E^2V)$	
blocking-flow	$\boldsymbol{O}(EV^2)$	
blocking-flow mit dynamischen Bäumen	$\boldsymbol{O}(EV \log V)$	
Capacity scaling	$\boldsymbol{O}(E^2 \log C)$	

C ist die maximale Kapazität, $|f^*|$ der maximale Fluss.

Verteilung von Programmierpraktika mit *Maxflow*



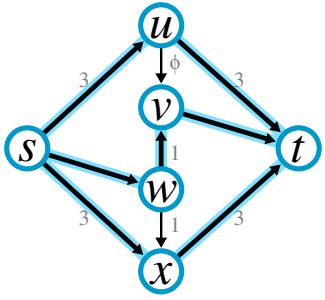
Anhang

Inhalt des Anhangs:

- Beispiel Flussgraph mit irrationalen Kapazitäten, für den die Ford-Fulkerson Methode bei ungünstiger Wahl der vergrößernden Pfade nicht terminiert: S. **??**

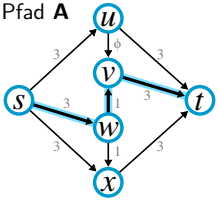
Kleiner Graph ohne Terminierung

- Sei $\phi = (\sqrt{5}-1)/2$, das Verhältnis des goldenen Schnittes.
- Es gilt $\phi^2 = \left(\frac{\sqrt{5}-1}{2}\right)^2 = \frac{5-2\sqrt{5}+1}{4} = \frac{3-\sqrt{5}}{2} = 1-\phi$
- und $\phi - \phi^2 = \phi(1-\phi) = \phi \cdot \phi^2 = \phi^3$.
- Der abgebildete Graph hat einen maximalen Fluss von mindestens 7:
  Die Pfade $s-u-t$ und $s-x-t$ bringen jeweils 3 und $s-w-v-t$ bringt 1.

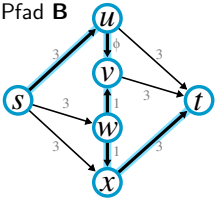


Kleiner Graph ohne Terminierung

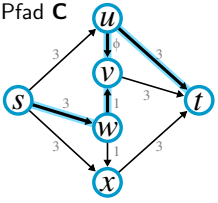
Pfad **A**



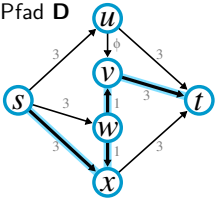
Pfad **B**



Pfad **C**



Pfad **D**



Folgende Wahl vergrößernder Pfade führt zu einer endlosen Sequenz:

Pfad	Fluss	Restkapazitäten $u\to v$	$v\to w$	$w\to x$				
A	1	ϕ	0	1	$\to$	ϕ^{k+1}	0 ϕ^k	(für $k=0$)
B	ϕ	0	ϕ	ϕ^2	da $1-\phi=\phi^2$			
C	ϕ	ϕ	0	ϕ^2				
B	ϕ^2	ϕ^3	ϕ^2	0	da $\phi-\phi^2=\phi^3$			
D	ϕ^2	ϕ^3	0	ϕ^2	$\to$	ϕ^{k+3}	0 ϕ^{k+2}	(für $k=0$)
A + $K\cdot$ BCBD	$1+\sum_{k=1}^{2K} 2\phi^k$	ϕ^{2K+1}	0	ϕ^{2K}				

[Zwick 1995; Erickson, http://jeffe.cs.illinois.edu/teaching/algorithms]

- Die Pfad Sequenz *A und dann immer wiederholend B, C, B, D* konvergiert nicht zum maximalen Fluss:

$$1 + 2\sum_{k=1}^{\infty} \phi^k = 1 + \frac{2}{1-\phi} = 4 + \sqrt{5} < 7$$

- Zum vollständigen Beweis fehlen noch folgende (einfachen) Punkte:
- Zeige die Eigenschaften der Sequenz (siehe vorige Seite) durch Induktion nach k.
- Zeige insbesondere $1 - \phi^k = \phi^{k+1}$ und $\phi^k - \phi^{k+1} = \phi^{k+2}$.
- Prüfe bei der Induktion, dass die Kanten mit Kapazität 3 nicht über ihre Kapazitätsgrenzen gefüllt werden.

Literatur I

Generell:

- Cormen TH, Leiserson CE, Rivest R, Stein C. *Algorithmen - Eine Einführung.* De Gruyter Oldenbourg, 4. Auflage; 2013. ISBN: 978-3486748611
- Dasgupta S, Papadimitriou CH, Vazirani UV. *Algorithms.* McGraw-Hill Higher Education; 2008. ISBN: 978-0073523408
- Ottmann T & Widmayer P. *Algorithmen und Datenstrukturen.* Springer Verlag, 5. Auflage; 2011. ISBN: 978-3827428042
- Kleinberg J, Tardos E. *Algorithm Design.* Pearson Education Limited; Auflage: Pearson New International Edition (30. Juli 2013). ISBN: 978-1292023946

Anderes Vorlesungsmaterial:

- Röglin H. *Skript zur Vorlesung Randomisierte und Approximative Algorithmen*, Universität Bonn, `http://www.roeglin.org/teaching/WS2011/RandomisierteAlgorithmen/RandomisierteAlgorithmen.pdf`

Literatur II

- Wayne K. Vorlesung *Theory of Algorithms* (COS 423), Princeton University 2013. https://www.cs.princeton.edu/courses/archive/spring13/cos423/lectures.php
- Erickson J, *Algorithms lecture notes*, http://algorithms.wtf.

Originalveröffentlichungen:

- Karpinski M, Lampis M, Schmied R. *New inapproximability bounds for TSP*. Journal of Computer and System Sciences. 2015 Dec 1;81(8):1665-77.
- Zwick U. *The smallest networks on which the ford-fulkerson maximum flow procedure may fail to terminate*. Theoretical computer science. 1995 Aug 21;148(1):165-70.
- Dinic EA. *Algorithm for solution of a problem of maximum flow in a network with power estimation*, Soviet Math. Dokl. 11 (5), 1277-1280, 1970.
- Sleator DD, Tarjan RE. *A data structure for dynamic trees*. Journal of computer and system sciences. 1983 Jun 1;26(3):362-91.

Literatur III

- Orlin JB. *Max flows in O(nm) time*. In: Symp. on Theory of Computing 2012 (pp. 765-774).

Danksagung I

Bei der Darstellung der Algorithmen zu Flussgraphen habe ich viele Ideen von den großartigen Folien von Kevin Wayne zu seiner Vorlesung *Theory of Algorithms* (COS 423, Princeton University 2013) aufgenommen. (Seine Vorlesung orientiert sich seinerseits an den Büchern von Kleinberg & Tardos und von Kozen.)

Index