

Teil 8

Graphen: Maximale Flüsse

Minimale Schnitte messen, wie stark ein Graph zusammenhängt, d.h. wieviele und welche Kanten entfernt werden müssen, damit zwei voneinander getrennte Teile entstehen.

Diese Größe beantwortet dabei zwei Fragen:

- Wie zuverlässig ist das Netzwerk (wieviel Kanten müssen ausfallen, bevor es seine Funktion verliert)?
- Wieviel Einheiten kann ich maximal zwischen Knoten der entstehenden Teile transportieren?

Dabei ist (uns) noch nicht klar, wie die zweite Frage mit Hilfe von minimalen Schnitten beantwortet werden kann. Die Betrachtung von **Flüssen** vertieft dies jedoch weiter.

Teil 8: Graphen: Maximale Flüsse

Flüsse

Grundlegendes

Ein **Fluss** in einem gerichteten Graphen soll beschreiben, auf welchen Wegen sich irgendwelche Objekte (z.B. Waren oder Informationen) in einem Netzwerk bewegen.

Dabei interessiert uns insbesondere, wieviel Einheiten maximal zwischen zwei bestimmten Knoten des Graphen transferiert werden können.

Definition (Flussnetzwerk)

Ein **Flussnetzwerk** $N = (V, E; c; s, t)$ ist ein gerichteter Graph $G = (V, E)$ mit nicht-negativen Kanten-**Kapazitäten** $c(e) \geq 0$ und zwei ausgezeichneten Knoten, der **Quelle** s und der **Senke** t .

Bemerkung

Wir nehmen an, dass $c(u, v) = 0 \Leftrightarrow (u, v) \notin E$.

Definition (Fluss)

Ein **Fluss** f ist eine Abbildung $f: V \times V \rightarrow \mathbb{R}$, so dass

- 1 Für alle $u, v \in V$ gilt $f(u, v) \leq c(u, v)$
- 2 Für alle $u, v \in V$ gilt $f(u, v) = -f(v, u)$
- 3 Für alle $u \in V \setminus \{s, t\}$ gilt $\sum_{v \in V} f(u, v) = 0$

$f(u, v)$ wird der **Fluss von u nach v** genannt.

Der **Wert** $|f|$ des Flusses f ist definiert als

$$|f| := \sum_{v \in V} f(s, v).$$

Flüsse

- $f(u, v) \leq c(u, v)$ (Kapazitätsbeschränkung)

Diese Eigenschaft beschränkt den Fluss über eine Kante auf ihre Kapazität. D.h. durch die Kante (u, v) kann nicht mehr fließen, als ihre Kapazität zulässt.

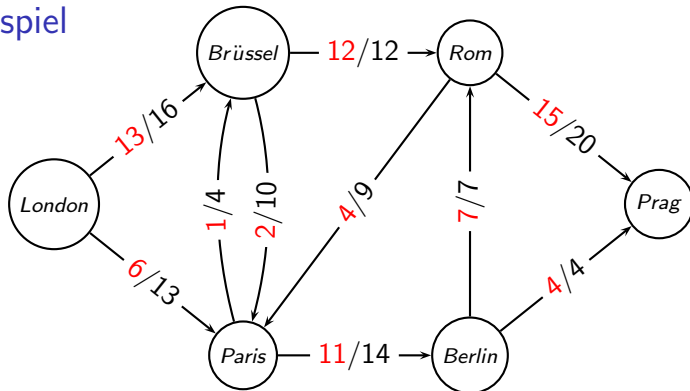
- $f(u, v) = -f(v, u)$ (Anti-Symmetrie)

Diese Eigenschaft besagt, dass der Fluss in einer Richtung dem negativen Fluss in der anderen Richtung entspricht.

- $\sum_{v \in V} f(u, v) = 0$ (Erhaltung)

Diese Eigenschaft entspricht einem Erhaltungssatz. In keinem Knoten außer der Quelle und dem Ziel, wird Fluss „vernichtet“ oder „erzeugt“. Anders formuliert: Alles was in einen Knoten hineinfließt, fließt auch wieder heraus.

Ein Beispiel



- Maximale Anzahl der möglichen täglichen Transporte
- **Tägliche Transporte**

Zwischen Brüssel und Paris resultiert ein täglicher Netto-Fluss von 1 von Brüssel nach Paris. Damit sind die **korrigierten Transporte** günstiger.

Ein Beispiel

Die Anti-symmetrie eines Flusses erlaubt die Aufhebung zweier Transporte in entgegengesetzter Richtung.

Um aus einem Fluss einen Transport zu erhalten, wird zwischen zwei Knoten u, v nur in der positiven Richtung transportiert. In der negativen Richtung erfolgt kein Transport.

Als Konsequenz können wir aus einem Fluss nicht die tatsächlichen Transporte rekonstruieren. Allerdings lässt sich aus jedem Fluss eine Möglichkeit für tatsächliche Transporte rekonstruieren.

Die Aufhebung durch die Anti-Symmetrie macht die so gefundenen Lösung unter Umständen sogar effizienter als der ursprüngliche Transport, da weniger Einheiten bewegt werden.

Das Problem

Für ein gegebenes Flussnetzwerk $N = (V, E; c; s, t)$ ist es nun unser Ziel den Fluss von s nach t zu maximieren.

Problem (Max-Flow)

Eingabe: *Ein Flussnetzwerk $N = (V, E; c; s, t)$*

Berechne den maximalen Wert eines Flusses in N .

Das Max-Flow-Min-Cut-Theorem

Satz (Max-Flow-Min-Cut)

Der maximale Wert eines Flusses auf einem Flussnetzwerk $N = (V, E; c; s, t)$ ist gleich dem Gewicht eines minimalen s - t -Schnittes des zugrunde liegenden Graphen $G = (V, E)$.

Teil 8: Graphen: Maximale Flüsse

Die Methode von Ford-Fulkerson

Residualnetzwerke

Ein wichtiges Werkzeug für die Konstruktion maximaler Flüsse sind die so genannten **Residualnetzwerke**.

Definition (Residualnetzwerk)

$N = (V, E; c; s, t)$ sei ein Flussnetzwerk und f ein Fluss auf N . Dann definieren wir das **Residualnetzwerk** N_f als $N_f := (V, E; c_f; s, t)$ mit

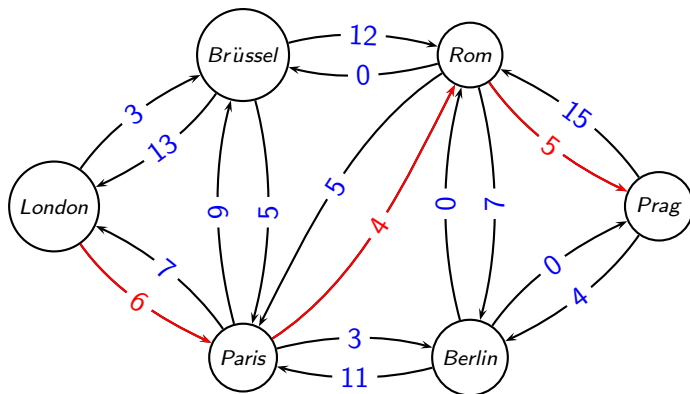
$$c_f(u, v) := c(u, v) - f(u, v)$$

Ein Fluss f kann nun vergrößert werden, in dem ein Weg von s nach t in dem Residualnetzwerk N_f hinzugefügt wird.

Definition (Augmentierende Wege)

Sei $N = (V, E; c; s, t)$ ein Flussnetzwerk und f ein Fluss auf N . Ein **augmentierender Weg** (**augmenting path**) ist ein Weg p in N_f , so dass für alle Kanten (u, v) in p $c_f(u, v) > 0$ gilt.

Ein Beispiel



Entlang dieses Pfades können maximal 4 Einheiten, d.h. die kleinste Kapazität der Kanten, transportiert werden.

Residualnetzwerke und augmentierende Pfade

Lemma

Sei $N = (V, E; c; s, t)$ ein Flussnetzwerk und f ein Fluss auf N .
Desweiteren sei f' ein Fluss auf den Residualnetzwerk N_f . Dann ist der
Fluss $f + f'$ mit

$$f + f'(u, v) := f(u, v) + f'(u, v)$$

ein Fluss auf N mit Wert $|f + f'| = |f| + |f'|$.

Beweis

Vorlesung Graphentheorie

Die Methode von Ford-Fulkerson

Aus einem Pfad p in N lässt sich ein Fluss f_p konstruieren:

$$f_p(u, v) := \begin{cases} w & \text{falls } (u, v) \in p \\ -w & \text{falls } (v, u) \in p \\ 0 & \text{sonst} \end{cases}$$

mit $w := \min\{c(u, v) \mid (u, v) \in p\}$.

Algorithmus (Methode von Ford-Fulkerson)

Eingabe: Ein Flussnetzwerk $N = (V, E; c; s, t)$.

Ausgabe: Ein maximaler Fluss f auf N .

Initialisiere f als Nullfluss

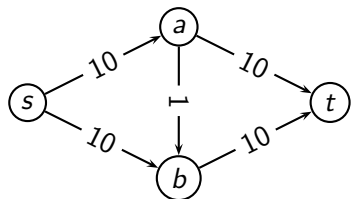
solange ein Pfad p in N_f existiert **tue** $f := f + f_p$

Gebe f zurück

Ein problematisches Beispiel

Da jeder augmentierende Pfad den Wert des gefundenen Flusses erhöht, scheint die Strategie zur Auswahl eines Pfades im Residualnetzwerk unerheblich zu sein.

Doch tatsächlich hat die Konstruktion des Pfades eine wesentliche Auswirkung auf die Laufzeit, und bei irrationalen Gewichten, sogar auf das Ergebnis selbst.



Im nebenstehenden Beispiel benötigt man, wie man leicht sieht, nur zwei Wege, um einen maximalen Fluss zu erhalten.

Durch eine ungünstige Wahl der augmentierenden Wege, kann man aber leicht auf 20 Durchgänge kommen.

Ein problematisches Beispiel

Sind die Kantenkapazitäten irrational, kann es sogar passieren, dass das Verfahren gar nicht terminiert. Außerdem kann es sein, dass der Wert des resultierenden Flusses nicht gegen den Wert eines maximalen Flusses konvergiert.

Aus diesem Grund ist es wichtig, sich Gedanken über die optimale Auswahl von augmentierenden Pfaden zu machen.

Teil 8: Graphen: Maximale Flüsse

Der Algorithmus von Edmonds-Karp

Die Idee von Edmonds-Karp

Edmonds und Karp bewiesen, dass es für die Methode von Ford-Fulkerson genügt, stets einen kürzesten augmentierenden Pfad zu suchen. Dabei wird die Länge lediglich über die Anzahl der Kanten, nicht ihre Kapazität, gemessen.

Ein solcher kürzester Weg lässt sich mittels einer Breitensuche finden.

Algorithmus (Edmonds-Karp)

Eingabe: Ein Flussnetzwerk $N = (V, E; c; s, t)$.

Ausgabe: Ein maximaler Fluss f auf N .

Initialisiere f als Nullfluss

solange ein kürzester Weg p von s nach t in N_f existiert **tue**

$f := f + f_p$

Gebe f zurück

Lemma

Sei $N = (V, E; c; s, t)$ ein Flussnetzwerk und p sei ein kürzester Weg von s nach t in N . Dann gilt für alle Knoten $v \in V \setminus \{s, t\}$, dass die Distanz $\delta_{N_{f_p}}(s, v)$ in N_{f_p} größer oder gleich der Distanz $\delta_N(s, v)$ in N ist.

Beweis

Wir nehmen an, dass ein Knoten $v \in V \setminus \{s, t\}$ existiert mit $\delta_{N_p}(s, v) < \delta_N(s, v)$.

O.B.d.A. können wir annehmen, dass v der Knoten mit kleinster Distanz $\delta_{N_{f_p}}(s, v)$ unter allen Knoten ist, deren Distanz in N_{f_p} sinkt.

$\omega = s \rightsquigarrow u \rightarrow v$ sei ein kürzester Pfad von s nach v in N_{f_p} . Damit gilt $(u, v) \in N_{f_p}$ und $\delta_{N_{f_p}}(s, u) = \delta_{N_{f_p}}(s, v) - 1$.

Da v unter allen Knoten, deren Distanz zu s sank, einer mit kleinster Distanz ist, kann die Distanz von u zu s in N_{f_p} nicht gesunken sein, d.h.

$$\delta_{N_{f_p}}(s, u) \geq \delta_N(s, u).$$

Beweis (Fortsetzung)

Wäre nun (u, v) im ursprünglichen Graphen N enthalten gewesen, so würde Folgendes gelten:

$$\begin{aligned}\delta_N(s, v) &\leq \delta_N(s, u) + 1 && \text{(Dreiecksungleichung)} \\ &\leq \delta_{N_{f_p}}(s, u) + 1 = \delta_{N_{f_p}}(s, v),\end{aligned}$$

was im Widerspruch zu unserer Annahme steht.

Damit gilt $(u, v) \notin N$ und $(u, v) \in N_{f_p}$.

Dies kann nur gelten, wenn die Kante (v, u) in p vorkommt, denn nur dann wird ihre Kapazität in N_{f_p} erhöht.

Damit muss (v, u) auf einem kürzesten Pfad von s nach v in N liegen, und somit gilt

$$\delta_N(s, v) = \delta_N(s, u) - 1 \leq \delta_{N_{f_p}}(s, u) - 1 = \delta_{N_{f_p}}(s, v) - 2,$$

was wiederum unserer Annahme widerspricht. □

Satz

Auf einem Flussnetzwerk $N = (V, E; c; s, t)$ benötigt der Edmonds-Karp Algorithmus $O(|V||E|)$ Suchen nach augmentierenden Wegen.

Beweis

Sei N_f das bislang konstruierte Residualnetzwerk. Eine Kante (u, v) auf einem augmentierenden Weg p heißt *kritisch*, wenn

$$c_f(u, v) = \min\{c(e) \mid e \in p\}.$$

D.h. falls $f' := f + f_p$ ist, haben kritische Kanten in $N_{f'}$ Kapazität 0 und in N_f positive Kapazität.

Im Folgenden zeigen wir, dass jede Kante $(u, v) \in E$ in N höchstens $|V|/2$ mal kritisch sein kann.

Beweis (Fortsetzung)

Sei $(u, v) \in E$ eine beliebige Kante. Ist sie kritisch, so gilt

$$\delta_f(s, v) = \delta_f(s, u) + 1$$

wobei δ_f die Distanz in N_f vorher ist.

Anschließend ist (u, v) nicht mehr im Residualnetzwerk, bis die Kante in anderer Richtung, d.h. als (v, u) , wieder im augmentierenden Pfad liegt. In dieser Situation gilt aber

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$$

Aufgrund des vorhergegangenen Lemmas gilt $\delta_f(s, v) \leq \delta_{f'}(s, v)$ und somit

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1 \geq \delta_f(s, v) + 1 = \delta_f(s, u) + 2.$$

Beweis (Fortsetzung)

D.h. zwischen zwei Runden, in denen die Kante (u, v) kritisch ist, steigt die Distanz $\delta_f(s, v)$ von v zur Quelle s um mindestens 2.

Da die Distanz von v zu s aber höchstens $|V| - 1$ betragen kann, kann die Kante (u, v) höchstens $\frac{|V|-1}{2} \leq \frac{|V|}{2}$ mal kritisch sein.

Da insgesamt nur $|E|$ Kanten vorliegen und jede nur $\frac{|V|}{2}$ mal kritisch sein kann, ergibt sich, dass der Algorithmus von Edmonds-Karp höchstens $O(|E||V|)$ Durchgänge benötigt. □

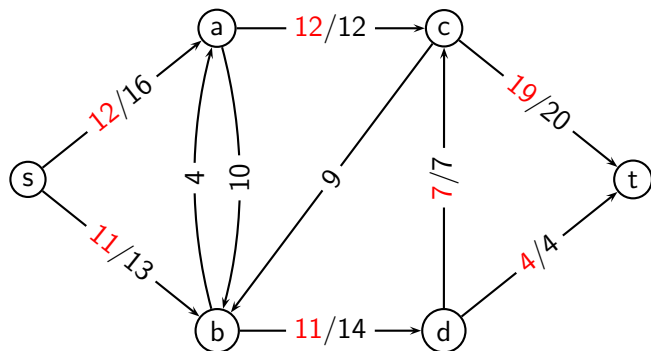
Satz

Der Algorithmus von Edmonds-Karp benötigt zur Berechnung eines maximalen Flusses auf $N = (V, E; c; s, t)$ Zeit $O(|V||E|^2)$.

Beweis.

Jede Breitensuche benötigt $O(|E|)$ Zeit. □

Ein Beispiel



Kapazität: $12 + 4 + 7 = 23$

Teil 8: Graphen: Maximale Flüsse

Shortest Path Augmenting Algorithmus

Distanzmarken

Idee

Statt immer wieder eine vollständige Breitensuche auszuführen, führen wir **Distanzmarken** mit, die uns erlauben den kürzesten Weg schneller zu finden.

Definition (Distanzmarken)

$N = (V, E; c; s, t)$ sei ein Flussnetzwerk. Eine **Distanzmarkierung** auf N ist eine Abbildung $d: V \rightarrow \mathbb{N}$. d heißt **gültig**, wenn

- $d(t) = 0$
- $d(u) \leq d(v) + 1$ für jede Kante $(u, v) \in E$.

Wir werden nun beschreiben, wie solche Distanzmarken genutzt werden können, um den Algorithmus von Edmonds-Karp auf $O(|V|^2|E|)$ zu beschleunigen.

Distanzmarken

Lemma

Seien $d(v)$ gültige Distanzmarken auf $N = (V, E; c; s, t)$. Dann ist $d(v)$ für $v \in V$ eine untere Schranke für die Distanz von v zu t .

Beweis.

Sei $w = (v = v_0, v_1, v_2, \dots, v_k = t)$ ein kürzester Weg der Länge k von v nach t .

Die zweite Bedingung für die Gültigkeit impliziert induktiv:

$$d(v_{k-1}) \leq d(t) + 1 = 1$$

$$d(v_{k-2}) \leq d(v_{k-1}) + 1 \leq 2$$

\vdots

$$d(i) = d(v_0) \leq d(v_1) \leq d(v_2) + 1 \leq k$$



Distanzmarken

Definition

Seien $d(v)$ gültige Distanzmarken auf $N = (V, E; c; s, t)$. Eine Kante $(u, v) \in E$ heißt **zulässig**, wenn $d(u) = d(v) + 1$.

Mit anderen Worten, ist eine Kante zulässig, wenn sie, bezogen auf die Distanzmarken, näher zu t führt.

Lemma

Seien $d(v)$ gültige Distanzmarken auf $N = (V, E; c; s, t)$. Ein **zulässiger Weg** von s nach t , d.h. ein Weg, der nur aus zulässigen Kanten besteht, ist ein kürzester Weg.

Beweis.

Sei $\omega = (s = v_0, \dots, v_k = t)$ ein zulässiger Weg der Länge k von s nach t .
Damit gilt für alle Kanten (v_i, v_{i+1}) auf dem Weg

$$d(v_i) = d(v_{i+1}) + 1.$$

Da $d(v_k) = d(t) = 0$ gilt, gilt somit

$$d(v_i) = k - i,$$

was wiederum $d(v_0) = d(s) = k$ impliziert.

Da $d(s)$ eine untere Schranke für die Distanz von s nach t ist, und ω ein Weg der Länge $d(s) = k$ ist, muss er ein kürzester Weg sein. □

Somit ist das Ziel, einen zulässigen Weg im Residualnetzwerk zu finden, und dabei die Distanzmarken so anzupassen, dass sie nach der Kombination des Flusses mit dem augmentierenden Weg weiterhin gültig sind.

Die Idee

Falls gültige Distanzmarken bekannt sind, entspricht die Suche nach einem kürzesten, augmentierenden Weg der Suche nach einem **zulässigen** Weg. Diese lässt sich effizienter gestalten als eine Breitensuche.

Im Algorithmus werden wir stets genau einen zulässigen Pfad, beginnend in der Quelle s , mitführen. Dazu speichern wir zu jedem Knoten v den Vorgänger $\text{pred}[v]$ auf dem Pfad und den aktuell letzten Knoten u . damit kann der Weg rückwärts rekonstruiert werden.

Die Suche nach einem zulässigen Weg realisieren wir mit zwei Operationen: **advance** und **retreat**.

advance erweitert den Weg, wenn möglich um eine zulässige Kante, während **retreat** einen Schritt rückwärts macht und die Distanzmarken des „verlassenen“ Knotens verändert.

retreat

Die Funktion `retreat` wird aufgerufen, wenn von dem aktuellen Knoten v keine zulässige Kante ausgeht.

In diesem Fall, gehen wir zu dem vorletzten Knoten des Weges zurück (wenn möglich) und passen die Distanzmarke des letzten Knotens v an. Genauer gesagt setzen wir

$$d(v) := \min \{d(u) + 1 \mid (v, u) \in N\}.$$

Algorithmus (`retreat`)

Eingabe: Ein Flussnetzwerk $N = (V, E; c; s, t)$, gültige Distanzmarken $d(v)$, ein aktueller Knoten v und ein Feld `pred` von Vorgängern.

$$d(v) := \min \{d(u) + 1 \mid (v, u) \in N\}$$

wenn $v \neq s$ **dann** $v := \text{pred}[v]$

Algorithmus (SPA – Shortest Path Augmenting)

Eingabe: Ein Flussnetzwerk $N = (V, E; c; s, t)$.

Ausgabe: Ein maximaler Fluss von s nach t in N .

Daten: Ein Fluss f

Initialisiere f als Nullfluss

Erstelle durch eine rückwärts durchgeführte Breitensuche, die korrekten Distanzmarken $d(v)$

$v := s$

solange $d(s) < n$ **tue** // Bis kein Pfad im Residualnetzwerk existiert

wenn v einen zulässige Kante (v, u) besitzt **dann**

$\text{pred}[u] := v$ // advance

$v := u$

wenn $v = t$ **dann** $\text{augment}(N, f, v, \text{pred})$ und $v := s$

Ende

sonst $\text{retreat}(N, d, v, \text{pred})$

Ende

Algorithmus (augment)

Eingabe: Ein Flussnetzwerk $N = (V, E; c; s, t)$, ein Fluss f , ein Weg p von s nach t in N , beschrieben durch pred .

Ausgabe: Das Residualnetzwerk N_{f_p} und der Fluss $f + f_p$.

$\delta := \infty$

$v := t$

solange $\text{pred}[v] \neq v$ **tue**

$\delta := \min(\delta, c(\text{pred}[v], v))$

$v := \text{pred}[v]$

// Suche kleinste Kante

Ende

$v := t$

solange $\text{pred}[v] \neq v$ **tue**

$f(\text{pred}[v], v) := f(\text{pred}[v], v) + \delta$

$c(\text{pred}[v], v) := c(\text{pred}[v], v) - \delta$

$c(v, \text{pred}[v]) := c(v, \text{pred}[v]) + \delta$

// Augmentiere den Weg

Ende

Die Korrektheit des SPA-Algorithmus

Lemma

Die Distanzmarken sind zu jedem Zeitpunkt des SPA-Algorithmus gültig. Weiterhin erhöht jede retreat-Operation die Distanzmarken.

Beweis

Offensichtlich verändert die advance-Operation weder das Netzwerk noch die Distanzmarken, so dass deren Gültigkeit bewahrt bleibt. Wir müssen lediglich die retreat-Operation und die Augmentierung betrachten.

Anfänglich konstruiert der Algorithmus über eine Breitensuche korrekte (und damit gültige) Distanzmarken. Deshalb können wir im Folgenden annehmen, dass die Markierungen vor der jeweiligen Operation gültig sind.

Beweis (Die Korrektheit des SPA-Algorithmus – Fortsetzung)

Betrachten wir zuerst die Augmentation.

Da die Markierungen gültig sind, wenn für jede Kante (u, v) gilt $d(u) \leq d(v) + 1$, bleibt die Gültigkeit erhalten, wenn eine oder mehrere Kanten entfernt werden.

Bei der Augmentation können aber auch neue Kanten (u, v) entstehen. Damit dies geschieht, muss die Kante (v, u) in dem zulässigen Pfad und damit selbst zulässig sein. D.h. es gilt $d(v) = d(u) + 1$ und somit $d(u) = d(v) - 1 \leq d(v) + 1$, was die Gültigkeit der Markierungen nach der Augmentation zeigt.

Beweis (Die Korrektheit des SPA-Algorithmus – Fortsetzung)

Betrachten wir nun die retreat-Operation.

retreat verändert die Marke eines Knotens v nur, wenn es keine zulässige inzidente Kante gibt. D.h. keine Kante (v, u) erfüllt $d(v) = d(u) + 1$. Zusammen mit der Gültigkeits-Bedingung $d(v) \leq d(u) + 1$, erfüllen alle Kanten (v, u) die Bedingung $d(v) < d(u) + 1$.

Damit gilt

$$d(v) < \min \{d(u) + 1 \mid (v, u) \in E\} = d'(v),$$

welches die neue Distanzmarke von v ist. Damit gilt insbesondere $d'(v) \leq d(u) + 1$ für alle Kanten (v, u) und die neue Markierung ist bezüglich der ausgehenden Kanten gültig.

Für die eingehenden Kanten (u, v) gilt

$$d(u) \leq d(v) + 1 < d'(v) + 1,$$

was die Gültigkeit auch für sie beweist.



Die Korrektheit des SPA-Algorithmus

Satz

Der SPA-Algorithmus berechnet einen maximalen Fluss in einem Flussnetzwerk $N = (V, E; c; s, t)$.

Beweis

Wie bereits gesehen konstruiert der SPA-Algorithmus in jeder Runde einen augmentierenden kürzesten Pfad im Residualnetzwerk. Damit entspricht er dem Algorithmus von Edmonds-Karp.

Um die Korrektheit zu zeigen, müssen wir nur noch zeigen, dass der Algorithmus erst terminiert, wenn kein Pfad von s nach t im Residualnetzwerk existiert.

Der Algorithmus terminiert, wenn die Quelle eine Distanzmarke $d(s) \geq n$ erreicht hat. In diesem Falls ist die tatsächliche Distanz von s zu t auf mindestens n gestiegen.

Beweis (Die Korrektheit des SPA-Algorithmus – Fortsetzung)

Da aber jeder Knoten nur einmal auf einem kürzesten Weg vorkommen kann, muss die Distanz kleiner gleich $n - 1$ sein.

Damit impliziert $d(s) \geq n$, dass kein Weg von s nach t mehr existieren kann.



Suchen von zulässigen Kanten

Bevor wir die vom SPA-Algorithmus benötigte Zeit ermitteln können, müssen wir uns noch damit beschäftigen, wie wir zu einem gegebenen Knoten v eine zulässige Kante (v, u) finden können.

Ein einfacher Ansatz wäre, dass mit jedem Erreichen des Knotens v die Liste der ausgehenden Kanten von Anfang an durchsucht wird, bis eine zulässige Kante gefunden wurde, oder das Ende der Liste erreicht wurde.

Dieses Vorgehen ist aber ungünstig, da bei einem Rückschritt von v die Liste des Vorgängers erneut durchsucht würde, obwohl klar ist, dass alle Kanten bis zur Kante (u, v) weiterhin unzulässig sind.

Daher merkt man sich, welches die Kante hinter der zuletzt betrachteten Kante in der Liste ist, und beginnt die Suche bei ihr.

Suchen von zulässigen Kanten

Zu Beginn jeder Runde setzen wir die **aktuelle** (ausgehende) Kante jedes Knotens auf die erste Kante der List.

Wird nun eine zulässige Kante gesucht, beginnen wir mit der **aktuellen** Kante.

Wird eine zulässige Kante gefunden, setzen wir die **aktuelle** Kante auf die nächste Kante und machen einen advance-Schritt mit der gefundenen Kante.

Machen wir entlang von (u, v) einen Schritt zurück, so wird die Distanzmarke von v auf mindestens $d(u) + 1$ erhöht, wodurch die Kante (u, v) im weiteren Verlauf nicht mehr zulässig ist.

Wird das Ende erreicht, wissen wir, dass keine der Kanten in der Liste (mehr) zulässig ist. Damit muss die Distanzmarke des aktuellen Knoten erhöht werden.

Die Laufzeit des SPA-Algorithmus

Bemerkung

Wenn der SPA-Algorithmus die Distanzmarke eines Knotens v höchstens k -mal ändert, dann wird jede von v ausgehende Kante höchstens $2k + 1$ -mal überprüft:

- für jede Erhöhung einmal für die Suche nach einer zulässigen Kante,
- für jede Erhöhung einmal für die Suche nach der kleinsten Nachbarmarkierung,
- und einmal für die Suche nach einer zulässigen Kante nach der letzten Erhöhung.

Damit ergibt sich die Gesamtzeit für die Suche nach zulässigen Kanten und dem Anpassen der Marken als $O(k|E|)$.

Die Laufzeit des SAP-Algorithmus

Lemma

Wenn der SAP-Algorithmus die Distanzmarke jedes Knotens höchstens k -mal erhöht, werden höchstens $\frac{k|E|}{2}$ Augmentationen durchgeführt.

Beweis.

Ähnlich wie beim Beweis der Korrektheit des Algorithmus von Edmonds-Karp, kann man sehen, dass zwischen zwei Augmentationen in denen die Kante (u, v) **kritisch** ist, die Distanzmarken von u und v um mindestens zwei steigen müssen.

Damit kann jede Kante höchstens $\frac{k}{2}$ -mal kritisch sein. Da aber bei jeder Augmentation mindestens eine Kante kritisch ist, finden höchstens $\frac{k|E|}{2}$ Augmentationen statt. □

Die Laufzeit des SAP-Algorithmus

Lemma

Der SAP-Algorithmus erhöht die Distanzmarke jedes Knotens höchstens $|V|$ -mal. Damit ist die Anzahl der Augmentationen höchstens $\frac{|V||E|}{2}$.

Beweis.

Jeder Änderung der Distanzmarke, erhöht sie um mindestens 1.

Sobald $d(v) \geq n$ gilt, wird der Knoten v nicht mehr betrachtet, da für die Zulässigkeit einer Kante (u, v) stets $|V| > d(s) \geq d(u) = d(v) + 1$ gelten muss.

D.h. nach spätestens $|V|$ Erhöhungen wird die Marke von v nicht mehr verändert.

Mit dem vorhergehenden Lemma ergibt sich somit die zweite Behauptung. □

Die Laufzeit des SAP-Algorithmus

Satz

Der SAP-Algorithmus berechnet einen maximalen Fluss in einem Flussnetzwerk $N = (V, E; c; s, t)$ in Zeit $O(|V|^2|E|)$.

Beweis.

- $O(|V||E|)$ Zeit für die Suche von zulässigen Kanten und dem Anpassen der Distanzmarken.
- $O(|V|12|E|)$ Zeit für die Augmentationen ($O(|V|)$ pro Augmentation).
- $O(|V|^2)$ retreat-Operationen.
- $O(|V|^2)$ advance-Operationen.



Andere Algorithmen

Name	Laufzeit	Bemerkungen
Ford-Fulkerson-Methode		
Generisch		ganzzahlige Kapazitäten
Edmonds-Karp	$O(V E ^2)$	
SPA	$O(V ^2 E)$	Effizient in der Praxis
Preflow-Push-Methode		
Generisch	$O(V ^2 E)$	Effizient Implementation erfordert Heuristiken
FIFO Preflow-Push	$O(V ^3)$	Sehr effizient in der Praxis
Highest-Label Pref-Push	$O(V ^2\sqrt{ E })$	Wahrscheinlich am effizientesten