

Der Preflow-push-Algorithmus

Bea Schumann

26. Juni 2009

Inhaltsverzeichnis

1	Einleitung	1
2	Der generische Algorithmus	2
2.1	Push und Relabel	3
2.1.1	Push	4
2.1.2	Relabel	4
2.1.3	Der Algorithmus	4
3	Korrektheit und Terminierung	5
3.1	Korrektheit	5
3.2	Terminierung	6

1 Einleitung

Um die Problemstellung zu beschreiben ist es notwendig ein paar grundlegende Begriffe einzuführen. Sei $G = (V, E)$ ein gerichteter Graph mit Knotenmenge V und Kantenmenge E mit Anzahl von $V = n$ und Anzahl von $E = m$. Der Einfachheit halber nehmen wir an $m \geq n - 1$.

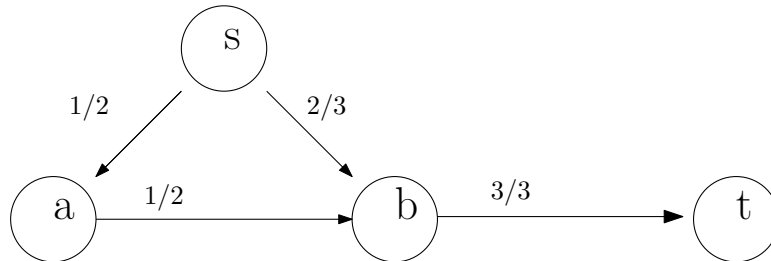
Solch einen Graph nennen wir **Netzwerk** falls er zwei verschiedene ausgezeichnete Knoten hat, genannt Quelle s und Senke t , und eine positive reellwertige **Kapazitätenfunktion** $c(v, w)$ für alle $(v, w) \in E$. Wir erweitern diese Funktion auf alle Knotenpaare, indem wir $c(v, w) = 0$ setzen, falls $(v, w) \notin E$.

Ein **Fluss** f auf G ist nun ebenso eine reellwertige Funktion auf den Knotenpaaren, die jedoch folgende Bedingungen erfüllt:

1. Kapazitätenbedingung: $f(v, w) \leq c(v, w) \forall (v, w) \in V \times V$
2. Antisymmetriebedingung: $f(v, w) = -f(w, v) \forall (v, w) \in V \times V$
3. Flusserhaltungsbedingung: $\sum_{u \in V} f(u, v) = 0 \forall v \in V - \{s, t\}$

Die Antisymmetriebedingung dient hier der Vereinfachung. Die Flusserhaltungsbedingung bedeutet anschaulich (wenn man nur an den positiven Teil des Flusses denkt), dass der gesamte Fluss in einem Knoten $v \notin \{s, t\}$ genauso groß ist, wie der gesamte Fluss, der aus v heraus fließt. Der **Wert** $|f|$ eines Flusses f ist der

gesamte Netzwerkfluss, der in die Senke hinein fließt: $|f| = \sum_v f(v, t)$



Bsp:

Im Bild steht auf der linken Seite an den Kanten der Fluss und auf der rechten Seite die Kapazität.

Das Ziel des Preflow-push Algorithmus ist es, einen maximalen Fluss in einem gegebenem Netzwerk zu finden. Er ist eine Verbesserung des klassischen Ford Fulkerson Algorithmus. Dabei arbeitet der Preflow-push Algorithmus mindestens so schnell, wie jeder andere bekannte Algorithmus für das Problem (in $O(n^3)$). Wenn man ihn geschickt implementiert arbeitet er sogar schneller: benutzt man die dynamic tree Datenstruktur, dann läuft der Algorithmus in $O(nm \log(n^2/m))$ bei einem Graph mit n Knoten und m Kanten. Das ist mindestens so schnell wie alle anderen Algorithmen und bei bestimmten Graphen sogar schneller.

2 Der generische Algorithmus

Der Algorithmus arbeitet mit einem so genannten Präfluss, den er dann sukzessive in einen maximalen Fluss umwandelt. Ein **Präfluss** erfüllt die Bedingungen 1 und 2 eines Flusses, nur Bedingung 3 wird abgeändert in $\sum_{u \in V} f(u, v) \geq 0 \forall v \in V - \{s\}$, also der Fluss in einen Knoten $\neq s$ hinein soll mindestens so groß sein, wie der Fluss hinaus. Jetzt macht es Sinn, einen weiteren Begriff einzuführen. Der **Überschuss** $e(v)$ eines Knoten $v \in V$ ist definiert als $e(v) = \sum_{u \in V} f(u, v)$.

Offensichtlich ist ein Präfluss ein Fluss, wenn $e(v) = 0$ gilt $\forall v \in V$.

Die Idee des Algorithmus ist es nun zu einem gegebenem Netzwerk einen Präfluss zu wählen und dann so viel Fluss wie nur möglich in Richtung Senke zu pumpen. Wenn die Senke von einem Knoten mit positivem Überschuss nicht erreichbar ist, soll Fluss in Richtung Quelle gepumpt werden. Damit wird erreicht, dass der Wert des Flusses größer wird, denn wird Fluss in Richtung Quelle gepumpt, dann kann natürlich wieder mehr Fluss in die Gegenrichtung gepumpt werden ohne die Flusserhaltungsbedingung zu verletzen. Am Ende soll kein Knoten mehr Überschuss haben. Wir werden sehen, dass wir dann einen maximalen Fluss erhalten.

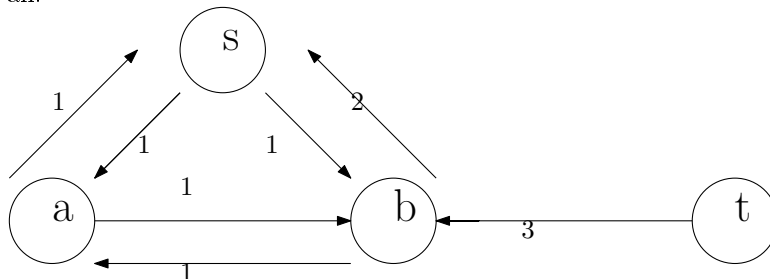
Um das obige Prinzip zu realisieren sind zwei Operationen notwendig: push und relabel

2.1 Push und Relabel

Als erstes müssen wir das oben angesprochene Pumpen von Fluss formalisieren. Dazu definiert man die **Restkapazität** eines Knotenpaaren (v, w) als $r_f(v, w) = c(v, w) - f(v, w)$. Im obigen Beispiel gilt $r_f(s, a) = 1$, $r_f(s, b) = 1$, $r_f(a, b) = 1$ und $r_f(b, t) = 0$. Mit Hilfe der Restkapazität kann nun entschieden werden in welche Richtung Fluss gepumpt werden kann, denn es gilt offensichtlich: Falls für $v \in V$ gilt $e(v) > 0 \wedge r_f(v, w) > 0$ kann ein Flussüberschuss bis zu $\delta = \min(e(v), r_f(v, w))$ von v zu w gepumpt werden, um hinterher immer noch einen Präfluss zu haben.

Wir definieren einen Hilfsgraph, den **Restgraphen**, dessen Knotenmenge V die des ursprünglichen Graphen ist, dessen Kantenmenge jedoch die Menge der Restkanten E_f ist. In E_f sind dabei alle Kanten $(v, w) \in E$ für die gilt: $r_f(v, w) > 0$.

Man beachte, dass es zwei Möglichkeiten für eine Kante (v, w) gibt positiven Überschuss zu haben: entweder (v, w) ist eine Kante mit weniger Fluss als Kapazität oder (w, v) hat positiven Fluss. Im ersten Fall vergrößert das Pumpen von Überschuss von v nach w den Fluss auf (v, w) , im zweiten Fall verkleinert es den Fluss auf (w, v) . Aus diesem Grund kann es im Restgraphen Kanten in zwei Richtungen geben. Schauen wir uns den Restgraphen für unser obiges Beispiel an:



Mit Hilfe des Restgraphen kann also ermittelt werden in welche Richtung Fluss gepumpt werden kann.

Ein weiteres Problem ist zu entscheiden, welche Knoten näher an der Senke und welche näher an der Quelle liegen. Hierbei ist nur der Restgraph interessant, da nur entlang der Kanten im Restgraph Fluss gepumpt werden kann. Man führt eine Höhenfunktion d ein: $d : V \rightarrow \mathbb{N} \cup \infty$, wobei $d(s) = n$, $d(t) = 0$ und $d(v) \leq d(w) + 1 \forall (v, w) \in E_f$. Falls $d(v) < n$, dann ist die Höhe eine untere Schranke für die Entfernung von v zu t im Restgraphen, und falls $d(v) \geq n$, dann ist die $d(v) - n$ eine untere Schranke für die Entfernung von v zu s . Die Einführung von unendlichen Höhen dient nur der Vereinfachung. Es wird sich herausstellen, dass die Höhen während jedes Schritts des Algorithmus endlich bleiben.

Jetzt noch eine letzte Definition und wir können wir Operationen Push und Relabel beschreiben. Ein Knoten v heißt aktiv, falls $v \in V - \{s, t\}$, $d(v) < \infty$ und $e(v) > 0$

2.1.1 Push

- Vorbedingung: v ist aktiv, $r_f(v, w) > 0$ und $d(v) = d(w) + 1$
- Aktion: Sende $\delta = \min(e(v), r_f(v, w))$ Einheiten Fluss von v nach w wie folgt:
 $f(v, w) \leftarrow f(v, w) + \delta$; $f(w, v) \leftarrow f(w, v) - \delta$;
 $e(v) \leftarrow e(v) - \delta$; $e(w) \leftarrow e(w) + \delta$

2.1.2 Relabel

- Vorbedingung: v ist aktiv und $\forall w \in V, r_f(v, w) > 0 \Rightarrow d(v) \leq d(w)$
- Aktion: $d(v) \leftarrow \min\{d(w)+1 \mid (v, w) \in E_f\}$; falls die Menge leer ist, setze $d(v) \leftarrow \infty$

2.1.3 Der Algorithmus

```
procedure MAX-FLOW( $V, E, s, t, c$ )
  <<Initialisiere preflow>>
  for all  $(v, w) \in (V - \{s\}) \times (V - \{s\})$  do
     $f(v, w) \leftarrow 0$ ;
     $f(w, v) \leftarrow 0$ ;
  end for
  for all  $v \in V$  do
     $f(s, v) \leftarrow c(s, v)$ ;
     $f(v, s) \leftarrow -c(s, v)$ ;
  end for

  <<Initialisiere Höhen und Überschüsse>>
   $d(s) \leftarrow n$ ;
  for all  $v \in V - \{s\}$  do
     $d(v) \leftarrow 0$ ;
     $e(v) \leftarrow f(s, v)$ ;
  end for

  <<Schleife>>
  while  $\exists$  mögliche Basisoperation do
    wähle eine Basisoperation und führe sie aus;
  end while

  return( $f$ );
end procedure
```

Der Algorithmus beginnt mit einem Präfluss, der gleich der Kapazität ist bei allen Kanten, die die Quelle verlassen und 0 bei allen anderen Kanten. Wir weisen den Kanten eine Anfangshöhe zu: $d(s) = n$ und $d(v) = 0 \forall v \in V - \{s\}$. Dies ist offensichtlich eine Höhenfunktion und zwar die einfachste, die wir finden können. Eigentlich ist das nicht die geschickteste Wahl, aber wir werden es bei dieser belassen, um die Beweise zu vereinfachen.

In welcher Reihenfolge der Algorithmus die Operationen Push und Relabel durchführt wird nicht festgelegt, weshalb er generischer Algorithmus genannt wird. Die Reihenfolge spielt keine Rolle für die Terminierung, aber beeinflusst natürlich die Laufzeit. Der Algorithmus endet, wenn es keine aktiven Knoten mehr gibt.

Bemerkung 1 Falls f ein Präfluss ist, d eine Höhenfunktion für f und v ein aktiver Knoten, dann ist immer entweder eine Push- oder eine Relabeloperation für v möglich.

3 Korrektheit und Terminierung

Nun werden wir die Korrektheit beweisen und auf die Terminierung Algorithmus eingehen.

3.1 Korrektheit

Wir nehmen an der Algorithmus terminiert und beweisen, dass er in diesem Fall korrekt ist. Damit das erfüllt ist, muss in jedem Schritt des Algorithmus gelten, dass d immer noch eine Höhenfunktion ist.

Lemma 1 Der Algorithmus erhält die Eigenschaft der Höhenfunktion.

Beweis mittels Induktion über die Anzahl der Push- und Relabeloperationen Anfangsanordnung ist offensichtlich eine Höhenfunktion. Angenommen d ist eine beliebige Höhenfunktion. Wird jetzt Relabel ausgeführt, ist d offensichtlich immer noch eine Höhenfunktion, denn es gilt immer noch $d(v) \leq (w) + 1 \forall (v, w) \in E_f$. Wird Push ausgeführt und Fluss von v nach w geschickt, dann können zwei Fälle eintreten: entweder (v, w) wird aus G_f herausgelöscht, was keine Auswirkung auf die Eigenschaft der Höhenfunktion hat, oder (w, v) wird in G_f eingefügt. Im letzteren Fall gilt aber: $d(v) = d(w) + 1$ (sonst wäre Push nicht ausgeführt wurden). Also ist d in jedem Fall eine Höhenfunktion. \square

Jetzt ist noch zu zeigen, dass der Präfluss am Ende des Algorithmus ein maximaler Fluss ist. Dazu benutzen wir folgendes Theorem:

Theorem 1 Ein Fluss f ist maximal, wenn es keinen einfachen (=kreisfreien) Weg von der Quelle zur Senke in G_f gibt.

Anschaulich gesehen ist das klar: wenn es einen Weg von der Senke zur Quelle in G_f gäbe, dann könnte Fluss entlang dieses Weges gepumpt werden, also der Fluss wäre nicht maximal gewesen.

Lemma 2 Sei f ein Präfluss und d eine Höhenfunktion, dann gibt es keinen einfachen Weg von der Quelle s zur Senke t in G_f .

Beweis mittels Widerspruch

Angenommen es gibt einen einfachen Weg von s nach t : $s = v_0, v_1, \dots, v_l = t$, so hat dieser maximal die Länge $l \leq n$. Da d eine Höhenfunktion ist gilt: $d(v_i) \leq d(v_{i+1}) + 1$ für $0 \leq i < l \Rightarrow d(s) \leq d(t) + l < n$, da $d(t) = 0$ (d ist eine Höhenfunktion). Dies ist aber ein Widerspruch zu $d(s) = n$. Also war die Annahme falsch und es gibt keinen einfachen Weg. \square

Theorem 2 *Angenommen der Algorithmus terminiert und alle Höhen sind endlich bei der Terminierung. Dann ist der Präfluss f ein maximaler Fluss, d.h. Algorithmus ist korrekt.*

Beweis *Wenn der Algorithmus terminiert und alle Höhen endlich sind, dann gilt $\forall v \in V - \{s, t\} : \sum_{u \in V} f(u, v) = 0$, da v nicht mehr aktiv ist. Also ist F ein Fluss und nach Theorem 1 und Lemma 2 sogar maximal. \square*

3.2 Terminierung

Um zu zeigen, dass der Algorithmus terminiert wird eine obere Schranke für die Laufzeit angegeben. Wir skizzieren hier nur den Beweis:

- Beweis**
1. *Beweisschritt: $\forall v \in V$ wird $d(v)$ niemals kleiner, eine Anwendung der Relabel-Operation vergrößert $d(v)$.*
 2. *Beweisschritt: Zu jeder Zeit während der Ausführung des Algorithmus gilt für jeden Knoten $v \in V$: $d(v) \leq 2n-1$. Damit ist auch gezeigt, dass die Höhen endlich bleiben.*
 3. *Beweisschritt: Die Anzahl der Relabel-Operationen ist höchstens $2n-1$ pro Knoten und höchstens $(2n-1)(n-2) < 2n^2$ bei allen Knoten. Die Multiplikation mit $2n-2$ kommt daher, dass wir im worst case alle Knoten ausser der Quelle und Senke Relabeln.*

Zum letzten Beweisschritt fehlt uns noch eine Definition: ein Push heißt sättigend, falls $r_f(v, w) = 0$ nach dem Push.

4. *Beweisschritt: Die Anzahl der sättigenden Push-Operationen ist höchstens $O(2nm)$ und die Anzahl der nicht-sättigenden Push-Operationen ist höchstens $O(4n^2m)$ Es folgt: Der Algorithmus terminiert nach höchstens $O(n^2m)$ Basisoperationen. \square*

Literatur

A.V.Goldberg, R.E.Tarjan. A New Approach to the Maximum-Flow-Problem