Maximaler Fluss in Flussnetzwerken

Proseminar Theoretische Informatik 2020

Maximilian Moeller

Fakultät Informatik, TU Dresden

17. Februar 2021

Inhaltsverzeichnis

In dieser Ausarbeitung soll eine Zusammenfassung des Kapitels 26 „Maximaler Fluss“ aus dem Buch „Introduction to Algorithms“ [Cor+09] gegeben werden. Dabei soll insbesondere auf den Push/Relabel-Algorithmus eingegangen und ein Vergleich zum Ford-Fulkerson-Algorithmus gezogen werden. Im Vordergrund wird nicht das Nachvollziehen von Beweisen, sondern vielmehr der Überblick über das Themengebiet stehen.

Die Fragestellung nach dem maximalen Fluss ist ein Problem aus der Graphentheorie, das in vielen praktischen Fällen Anwendung findet. Überall dort, wo die Menge einer zwischen verschiedenen Punkten transportierten Ware maximiert werden soll, kann auf die hier entwickelten Lösungen zurückgegriffen werden. Als Beispiele seien die Informationsübertragung in Computernetzwerken oder der Flüssigkeitstransport in Rohrleitungssystemen genannt.

Sämtliche Definitionen und Code-Beispiele entstammen [Cor+09]. Sie wurden lediglich in ihrer Notation angepasst. Die Idee zur Visualisierung des Push/Relabel-Algorithmus entstammt [Haa20].

1 Grundlagen

In diesem Abschnitt werden die grundlegenden Voraussetzungen geschaffen, um das Problem des maximalen Flusses sinnvoll formulieren zu können. Darüber hinaus werden einige Konzepte erarbeitet, die für das Verständnis der Algorithmen unerlässlich sind.

1.1 Flussnetzwerke

Ein Flussnetzwerk ist ein gerichteter Graph $G = (V, E)$, in dem jeder Kante $(u, v) \in E$ durch eine Kapazitätsfunktion $c : V \times V \to \mathfrak{B}$ eine Kapazität $c(u, v) \geq 0$ zugeordnet ist. Dabei ist häufig $\mathfrak{B} = \mathbb{N}$, seltener auch $\mathfrak{B} = \mathbb{Z}$ oder $\mathfrak{B} = \mathbb{R}$. Im Folgenden soll deswegen $\mathfrak{B} = \mathbb{N}$ betrachtet werden. Für $(u, v) \notin E$ sei $c(u, v) = 0$. Außerdem fordern wir:

$$\forall (u, v) \in V \times V : (u, v) \in E \Rightarrow (v, u) \notin E \tag{1}$$

Wir verbieten also entgegengerichtete Kantenpaare. Darüber hinaus darf es in einem Flussnetzwerk auch keine reflexiven Kanten – also Kanten mit gleichem Start- und Zielknoten – geben. Statt zu zeigen, dass dies keine echten Einschränkungen sind, sei hier auf [Cor+09, S. 724 f.] verwiesen.
Jedes Flussnetzwerk enthält zwei besondere Knoten: Eine Quelle s und eine Senke t. Der Einfachheit halber liege jeder Knoten $v \in V$ auf einem Pfad von s nach t.

1.2 Fluss in Flussnetzwerken

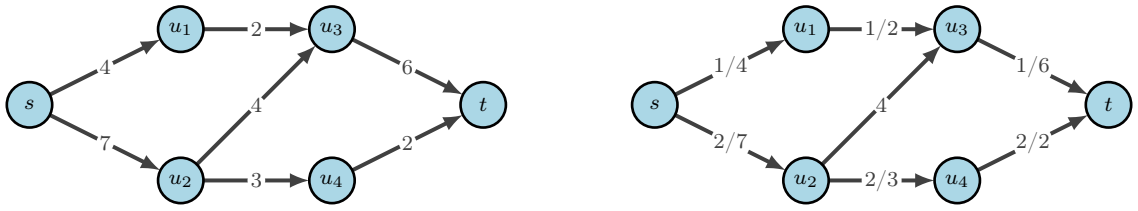
Ein Fluss f in einem Flussnetzwerk $G = (V, E)$ ist eine Funktion $f : V \times V \to \mathfrak{B}$. Dabei muss ein Fluss drei besondere Bedingungen erfüllen:

$$\forall u, v \in V : (u, v) \notin E \Rightarrow f(u, v) = 0 \tag{2}$$

$$\forall u, v \in V : 0 \leq f(u, v) \leq c(u, v) \tag{3}$$

$$\forall u \in V \setminus \{s, t\} : \sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v) \tag{4}$$

Ein Fluss tritt aus der Quelle s aus und fließt über die Kanten und Knoten bis zur Senke t. Die Eigenschaft (4) trägt den Namen Flusserhaltung. Im Beispiel des Stromflusses entspricht sie der Kirchhoffschen Knotenregel.



(a) Ein Flussnetzwerk G (b) Ein Fluss f in G mit $|f| = 3$

Abbildung 1: In der hier verwendeten Darstellung werden Kanten (u, v) mit $f(u, v)/c(u, v)$ beschrifet. Ist $f(u, v) = 0$, so wird nur die Kapazität angegeben.

1.3 Wert eines Flusses

Der Wert $|f|$ eines Flusses f ist definiert als:

$$|f| := \sum_{u \in V} f(s,u) - \sum_{u \in V} f(u,s) \tag{5}$$

Er entspricht also der Menge an Fluss, die aus der Quelle s herausfließt. Wegen (4) also auch der Menge an Fluss, die in die Senke t hineinfließt. Häufig enthält ein Flussnetzwerk keine eingehenden Kanten zur Quelle, womit sich die Gleichung zu $|f| = \sum_{u \in V} f(s,u)$ vereinfacht. Wir können nun das maximaler-Fluss-Problem formal definieren:

Definition. Im maximaler-Fluss-Problem ist ein Flussnetzwerk G mit Quelle s und Senke t gegeben. Gesucht wird ein Fluss f, dessen Wert $|f|$ maximal für dieses Flussnetzwerk ist.

Anmerkung. Dieser Fluss muss nicht eindeutig bestimmt sein.

1.4 Restnetzwerke

Restnetzwerke sind ein Konstrukt, das es uns erlauben wird effizient Kanten zu finden, auf denen der Fluss noch erhöht werden kann. Restnetzwerke sind ebenfalls gerichtete Graphen. Sie existieren jedoch nicht für sich, sondern werden durch den Fluss in einem Flussnetzwerk induziert.

Sei $G = (V, E)$ ein Flussnetzwerk und f ein Fluss in G. Die Restkapazität einer Kante ist definiert als:

$$c_f(u,v) := \begin{cases} c(u,v) - f(u,v) & \text{falls } (u,v) \in E \\ f(v,u) & \text{falls } (v,u) \in E \\ 0 & \text{sonst} \end{cases} \tag{6}$$

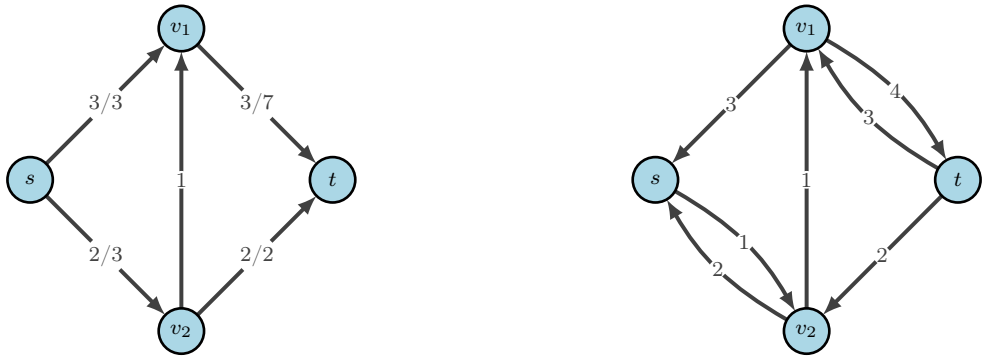
Wegen (1) tritt immer nur genau einer dieser Fälle ein. Daraus können wir die Menge der Restkanten definieren:

$$E_f := \{(u,v) \in V \times V \mid c_f(u,v) > 0\} \tag{7}$$

Dann ist das durch f induzierte Restnetzwerk $G_f = (V, E_f)$. Abbildung 2 zeigt ein Beispiel dafür.

Es ist wichtig zu verstehen, dass Restnetzwerke keine Flussnetzwerke sind, denn sie dürfen entgegengerichtete Kantenpaare enthalten. Das hatten wir für Flussnetzwerke verboten. Fließt entlang einer Kante (u,v) in einem Flussnetzwerk G ein Fluss f mit $0 < f(u,v) < c(u,v)$, so gibt es im Restnetzwerk G_f wegen (6) zwei Kanten zwischen diesen Knoten. Dabei stellt $c_f(u,v)$ diejenige Menge an Fluss dar, die noch über die Kante geschickt werden kann, ohne die Kapazitätsbeschränkung zu verletzen. Dahingegen ist $c_f(v,u)$ die Menge, um die der Fluss auf (u,v) reduziert werden kann. Solche Reduzierungen des Flusses können lokal nötig sein, um den Fluss global zu erhöhen.

Ganz analog zu Fluss in Flussnetzwerken können wir auch Fluss in Restnetzwerken definieren. Ein Fluss in einem Restnetzwerk muss ebenfalls den Anforderungen (2) - (4) genügen. Auch der Wert $|f'|$ eines Flusses f' in einem Restnetzwerk ist analog zu (5) definiert.



(a) Fluss f in einem Flussnetzwerk $G = (V, E)$

(b) durch f induziertes Restnetzwerk $G_f = (V, E_f)$

Abbildung 2: Induktion eines Restnetzwerkes

1.5 Erhöhung eines Flusses

Sei f ein Fluss im Flussnetzwerk $G = (V, E)$ und f' ein Fluss in G_f. Für die Erhöhung von f um f' schreiben wir $f \uparrow f'$ und definieren:

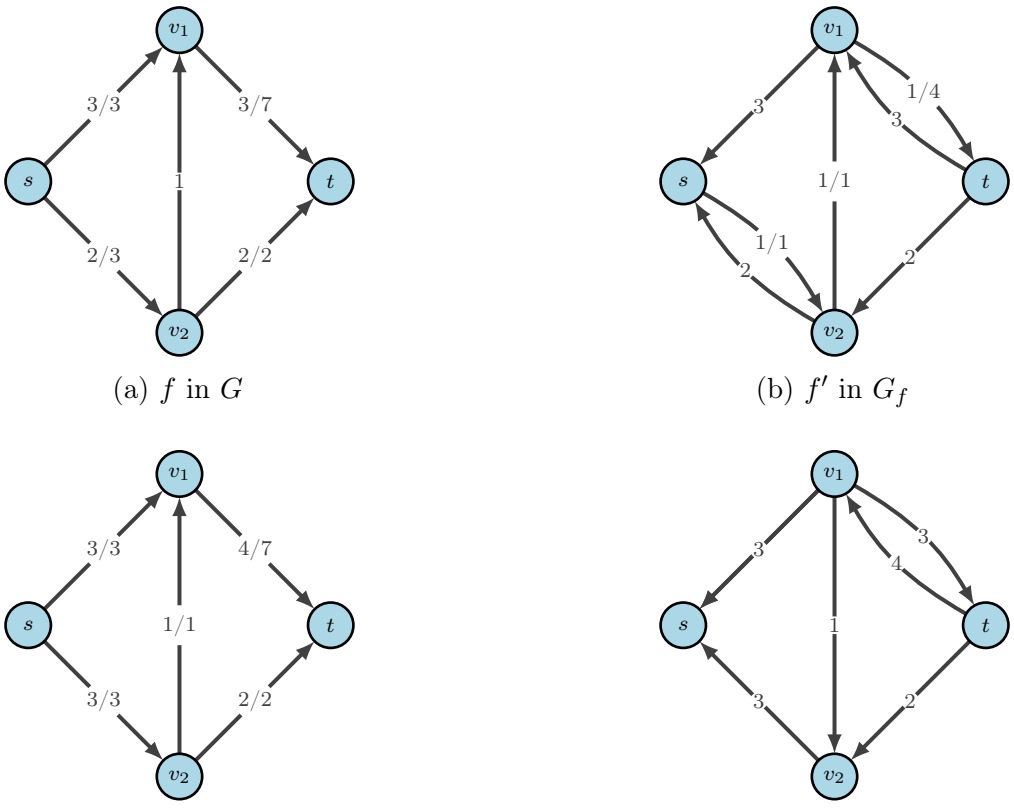
$$(f \uparrow f')(u, v) := \begin{cases} f(u, v) + f'(u, v) - f'(v, u) & \text{falls } (u, v) \in E \\ 0 & \text{sonst} \end{cases} \tag{8}$$

Diese Definition entspricht zunächst nur einer einfachen Intuition. „Wir erhöhen den Fluss auf (u, v) um $f'(u, v)$ und verringern ihn um $f'(v, u)$, da das Zurückfließenlassen eines Flusses auf einer entgegengerichteten Kante äquivalent dazu ist, den Fluss in dem ursprünglichen Netzwerk zu verringern.“ ([Cor+09, S. 730])

Die so erhaltene Funktion $f \uparrow f'$ erfüllt die Eigenschaften (2) - (4) – ist also wieder ein Fluss – und hat den Wert:

$$|f \uparrow f'| = |f| + |f'| \tag{9}$$

Die Beweise für diese wichtigen Erkenntnisse würden den Rahmen dieser Zusammenfassung sprengen, sie sind jedoch sehr sehenswert. Es sei daher auf [Cor+09, S. 730-732] verwiesen. An Abbildung 3 lässt sich die Erhöhung eines Flusses gut nachvollziehen. Man sieht gut, wie im dort entstandenen Flussnetzwerk $G_{f \uparrow f'}$ kein Fluss mehr fließen kann, da es keine ausgehende Kante von s mehr gibt. Diese Beobachtung ist für das maxflow-mincut-Theorem interessant, das wir noch besprechen werden.



(a) f in G (b) f' in G_f

(c) $f \uparrow f'$ in G (d) $G_{f \uparrow f'}$

Abbildung 3: Erhöhung eines Flusses

2 Ford-Fulkerson-Algorithmen

Zunächst soll die Klasse der Ford-Fulkerson-Algorithmen erläutert werden. Der Basisalgorithmus wurde von L. R. Ford Jr. und D. R. Fulkerson im Jahr 1956 vorgestellt. Wir sprechen hier von einer Algorithmenklasse, weil die hier beschriebenen Ideen in vielen leicht abgewandelten Algorithmen Verwendung finden. Diese unterschiedlichen Implementationsmöglichkeiten ergeben unterschiedliche Laufzeiten.

2.1 Erweiterungspfade

Die Ford-Fulkerson-Algorithmen verwenden Pfade von s nach t im Restnetzwerk – sogenannte Erweiterungspfade. Entlang dieser Pfade kann dann ein Fluss definiert werden, um den der ursprüngliche Fluss im Flussnetzwerk erhöht wird.

Sei also f ein Fluss im Flussnetzwerk G. Mittels eines geeigneten Algorithmus wird zunächst ein Pfad p von s nach t im Restnetzwerk G_f bestimmt. Dann lässt sich die Pfadkapazität wie folgt definieren:

$$c_f(p) := min\{c_f(u,v) \mid (u,v) \text{ liegt auf } p\} \tag{10}$$

Und damit auch ein Fluss:

$$f_p(u,v) := \begin{cases} c_f(p) & \text{falls } (u,v) \text{ auf } p \text{ liegt} \\ 0 & \text{sonst} \end{cases} \tag{11}$$

Wegen (7) gilt offensichtlich $|f_p| > 0$. Somit erhalten wir einen neuen Fluss $f \uparrow f_p$ mit $|f \uparrow f_p| = |f| + |f_p| > |f|$.

2.2 Ford-Fulkerson-Basisalgorithmus

Mit den bisher gewonnenen Kenntnissen ist es uns jetzt möglich den FORD-FULKERSON-Basisalgorithmus nachzuvollziehen. Als Hinweis zur Schreibweise sei gesagt, dass wir den Fluss auf einer Kante als Eigenschaft dieser Kante auffassen und über den Punktoperator darauf zugreifen.

Algorithm 1 FORD-FULKERSON(G,s,t)

```
1: for jede Kante (u,v) ∈ G.E
2:     (u,v).f = 0
3: while es existiert ein Pfad p von s nach t in G_f
4:     c_f(p) = min{c_f(u,v) | (u,v) liegt auf p}
5:     for jede Kante (u,v) von p
6:         if (u,v) ∈ G.E
7:             (u,v).f = (u,v).f + c_f(p)
8:         else
9:             (v,u).f = (v,u).f − c_f(p)
```

Zunächst initialisieren wir allen Fluss mit dem Wert 0. Anschließend erhöhen wir den Fluss entlang eines Erweiterungspfades um die Pfadkapazität und fahren so lange fort, wie es noch Erweiterungspfade gibt.

2.3 Analyse des Ford-Fulkerson-Basisalgorithmus

Man sieht leicht, dass die so beschriebene Funktion f bei Terminierung tatsächlich ein Fluss ist. Schwieriger hingegen ist die Frage, ob f auch ein maximaler Fluss ist. Sie lässt sich mithilfe des **maxflow-mincut-Theorems** beantworten, dessen Beweis wir hier auslassen. Kurz zusammengefasst stellt dieses Theorem den Zusammenhang zwischen Flüssen in Flussnetzwerken und Schnitten in Flussnetzwerken her. Ein Schnitt ist eine besondere Partitionierung des Flussnetzwerks in zwei Teilmengen, von denen eine die Quelle s und die andere die Senke t enthält. Unter anderem besagt das **maxflow-mincut-Theorem**:

$$|f| \text{ ist maximal} \Leftrightarrow \text{es existiert kein Erweiterungspfad in } G_f \tag{12}$$

Da wir für die Kantenkapazitäten im Flussnetzwerk natürliche Zahlen annehmen, wächst der Fluss in jeder Iteration des Algorithmus um mindestens eine Einheit. Offensichtlich muss ein maximaler Fluss auch einen Wert aus $\mathbb{N}$ haben und somit terminiert der Algorithmus nach endlich vielen Schritten mit einem korrekten Ergebnis. Auch für Kantenkapazitäten aus $\mathbb{Q}$ sind diese Beobachtungen interessant. Für ein solches Flussnetzwerk kann man alle Kapazitäten mit dem Hauptnenner multiplizieren und so das Netzwerk „skalieren“. Dann führt man den Algorithmus auf dem skalierten Flussnetzwerk aus und skaliert das Ergebnis zurück, indem man es durch den Hauptnenner dividiert. Dieses Vorgehen lässt sich jedoch nicht auf jeden Zahlenbereich ausweiten. Ford und Fulkerson konnten ein Flussnetzwerk mit Kantenkapazitäten aus $\mathbb{R}$ finden, für welches ihr Algorithmus nicht nur nicht terminiert, sondern der Fluss noch nicht einmal gegen einen maximalen Fluss konvergiert.

In Hinblick auf die Laufzeit des Algorithmus ist vor allem relevant, wie wir den Pfad p bestimmen. Mit einer geeigneten Datenstruktur und Breitensuche geben [Cor+09] dafür $\mathcal{O}(E)$ an. Wir hatten bereits beobachtet, dass wir für ein Flussnetzwerk mit einem maximalen Fluss f' höchstens $|f'|$ Iterationen berechnen müssen. Als Gesamtlaufzeit ergibt sich somit $\mathcal{O}(E \cdot |f'|)$ Dass in diesem Spezialfall die Laufzeit nicht nur von der Größe der Eingabe, sondern auch von der Größe der Ausgabe abhängt, ist ein Nachteil des Basisalgorithmus. Er entsteht dadurch, dass die Reihenfolge, in der wir die Erweiterungspfade wählen entscheident ist. Besonders verständlich wird dies, wenn man Abbildung 4 betrachtet. Hier könnten wir den Erweiterungspfad immer abwechselnd als $p = (s, v_1, v_2, t)$ und $p = (s, v_2, v_1, t)$ wählen, da sich die Kante zwischen v_1 und v_2 im Restnetzwerk durch die jeweilige Erhöhung umdreht. Wenn wir den Erweiterungspfad nicht beliebig, sondern als einen kürzesten Pfad zwischen s und t wählen, erhalten wir den Edmonds-Karp-Algorithmus. Benutzen wir Breitensuche für die Bestimmung eines solchen kürzesten Pfades, so verbessert sich die Laufzeit auf $\mathcal{O}(V \cdot E^2)$.

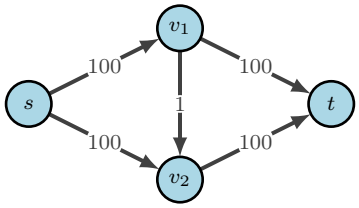


Abbildung 4

3 Push/Relabel-Algorithmen

Während die FORD-FULKERSON-Algorithmen global nach einem Erweiterungspfad suchen, arbeiten Push/Relabel-Algorithmen wesentlich lokaler. Sie betrachten stets nur einen Knoten und seine direkte Umgebung gleichzeitig. Auch bei den Push/Relabel-Algorithmen gibt es mehrere Implementationsmöglichkeiten, die verschiedene Laufzeiten mit sich bringen. Wir werden hier ausschließlich den Basisalgorithmus betrachten. Zunächst wird dabei eine Datenstruktur initialisiert. Dann werden so lange Push- oder Relabel-Operationen ausgeführt, bis keine ausführbare Operation verbleibt.

3.1 Grundlagen

Push/Relabel-Algorithmen verletzen zeitweise die Flusserhaltung (4). Statt in jedem Iterationsschritt einen gültigen Fluss zu erzeugen, verwenden sie Vorflüsse. Ein Vorfluss ist ein Fluss, der lediglich eine schwächere Form der Flusserhaltung erfüllen muss:

$$\forall u \in V \setminus \{s\} : \sum_{v \in V} f(v,u) \geq \sum_{v \in V} f(u,v) \tag{13}$$

In einen Knoten u kann also mehr Fluss hineinfließen, als aus ihm hinaus fließt. Die Differenz dieser Mengen wird der Flussüberschuss $e(u)$ genannt:

$$e(u) := \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v) \tag{14}$$

In den Knoten kann also ein beliebig großer Flussüberschuss gespeichert werden. Ist $e(u) > 0$, so sagen wir u ist überflutet. Wir werden sehen, wie der Flussüberschuss im Laufe des Algorithmus immer weiter in Richtung Senke geschoben (gepusht) wird. Sobald die maximale Menge an Fluss von s nach t fließt, muss der Algorithmus dann nur noch den verbliebenen Flussüberschuss in allen Knoten wieder zurück zur Quelle schicken.

Neben dem Flussüberschuss gibt es noch eine weitere Größe. Die Höhe $h(u)$ eines Knotens u bestimmt, welche Operationen wir wie ausführen können. Wir fixieren folgende Höhen: $h(s) = |V|$ und $h(t) = 0$. Alle anderen Knoten beginnen mit Höhe 0, steigen jedoch im Laufe des Algorithmus an. Flussüberschuss wird ausschließlich von Knoten größerer Höhe zu Knoten geringerer Höhe gepusht. Das schließt jedoch nicht aus, dass nicht auch von einem Knoten geringerer Höhe Fluss zu einem Knoten größerer Höhe fließt. Sei $G = (V, E)$ ein Flussnetzwerk und f ein Vorfluss in G. Wir fordern weiterhin von der Höhenfunktion $h : V \to \mathbb{N}$:

$$\forall (u,v) \in E_f : h(u) \leq h(v) + 1 \tag{15}$$

Wir beobachten für später auch die Kontraposition:

$$\forall u, v \in V : h(u) > h(v) + 1 \Rightarrow (u,v) \notin E_f \tag{16}$$

3.2 Die Push-Operation

Wir werden sowohl für die Push-, als auch später für die Relabel-Operation Voraussetzungen festlegen, die erfüllt sein müssen um diese Operationen anwenden zu können. Die Push-Operation wird verwendet, um Flussüberschuss von einem Knoten zu einem anderen zu drücken. Wir können PUSH(u, v) anwenden, wenn u überflutet ist, die Restkapazität $c_f(u, v) > 0$ ist und $h(u) = h(v) + 1$ gilt. Die ersten beiden Bedingungen sind trivial. Wie sollten wir Fluss von u nach v drücken, wenn u nicht überflutet wäre oder die Kante (u, v) bereits gesättigt wäre. Mit der dritten Bedingung legen wir fest, dass wir Fluss stets nur über eine Kante pushen, die zu einem Knoten mit einer um eine Einheit niedrigeren Höhe führt. Dies ergibt sich direkt aus (16), wo wir gesehen haben, dass es keine Kanten im Restnetzwerk zu einem Knoten geben kann, dessen Höhen um mehr als eine Einheit niedriger ist. Solange es sich bei h also tatsächlich um eine Höhenfunktion handelt, wäre ein Push über mehr als eine Höheneinheit also unzulässig.

Algorithm 2 PUSH(u, v)

```
1: // Anwendbar wenn: u.e > 0, c_f(u,v) > 0, und u.h = v.h + 1.
2: // Aktion: Drücke Δ_f(u,v) = min(u.e, c_f(u,v)) Flusseinheiten von u nach v.
3:
4: Δ_f(u,v) = min(u.e, c_f(u,v))
5: if (u,v) ∈ E
6:     (u,v).f = (u,v).f + Δ_f(u,v)
7: else
8:     (v,u).f = (v,u).f − Δ_f(u,v)
9: u.e = u.e − Δ_f(u,v)
10: v.e = v.e + Δ_f(u,v)
```

Wir wählen $\Delta_f(u, v)$ als das Minimum aus Flussüberschuss in u und verbliebener Restkapazität $c_f(u, v)$. Danach verändern wir den Vorfluss auf der gewählten Kante und passen die Überschüsse an.

3.3 Die Relabel-Operation

Wir haben bereits gesehen, wie wir Flussüberschuss von einem Knoten zum nächsten bewegen können. Es kann allerdings vorkommen, dass ein Knoten überflutet ist und keine gültige Push-Operation für ihn existiert. In diesem Fall wenden wir die Relabel-Operation auf den Knoten an, die seine Höhe wachsen lässt – wir markieren u um. Bedingungen für die Ausführung von RELABEL(u) sind also, dass $e(u) > 0$ ist und $\forall(u, v) \in E_f : h(u) < h(v)$ gilt. Ist nämlich die Höhe aller Knoten, zu denen Restkanten führen höher als die von u, so können wir den Flussüberschuss in u durch keine Push-Operation verringern.

Algorithm 3 RELABEL(u)

1: // **Anwendbar wenn:** $u.e > 0$, und $u.h \leq v.h$ für alle $v \in V$ mit $(u, v) \in E_f$.
2: // **Aktion:** setze u.h auf einen höheren Wert.
3:
4: $u.h = 1+\min\{v.h \mid (u, v) \in E_f\}$

Wegen $e(u) = \sum_{v \in V} f(v, u) - \sum_{v \in V} f(u, v) > 0$ muss es mindestens einen Knoten v geben mit $f(v, u) > 0$. Daraus folgt direkt, dass $c_f(u, v) > 0$ und damit $(u, v) \in E_f$. „Die Operation RELABEL(u) weist u also die größte Höhe zu, die nach den Beschränkungen der Höhenfunktionen erlaubt ist.“ ([Cor+09, S. 753]) Da wir die Höhenfunktion nur in der Relabel-Operation verändern, können wir uns also sicher sein, niemals (15) zu verletzen.

3.4 Der generische Push/Relabel-Algorithmus

Für den generischen Push/Relabel-Algorithmus benötigen wir nun nur noch einen initialen Vorfluss. Diesen erzeugen wir wie folgt:

Algorithm 4 Initialize-Preflow(G,s)

1: **for** jeden Knoten $v \in G.V$
2: $\quad v.h = 0$
3: $\quad v.e = 0$
4: **for** jede Kante $(u, v) \in G.E$
5: $\quad (u, v).f = 0$
6: $s.h = |G.V|$
7: **for** jeden Knoten v mit $(s, v) \in G.E$
8: $\quad (s, v).f = c(s, v)$
9: $\quad v.e = c(s, v)$
10: $\quad s.e = s.e - c(s, v)$

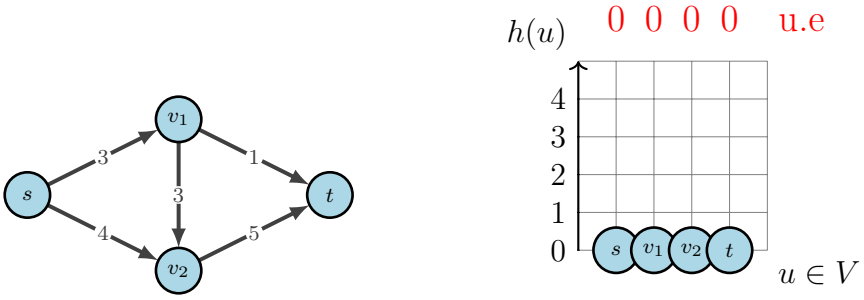
Wir schicken also von der Quelle aus einen Fluss zu allen angrenzenden Knoten. Dabei wählen wir die Menge an Fluss genau gleich zur jeweiligen Kapazität der Kante. Man sieht leicht, dass f dann ein Vorfluss ist. Diejenigen Kanten (u, v), für die $h(u) > h(v) + 1$ ist, sind genau die Kanten mit $u = s$. Da diese Kanten bereits bis zu ihrer Kapazität mit Fluss gefüllt sind, kommen sie im Restnetzwerk nicht vor. Die Eigenschaft h beschreibt also eine Höhenfunktion.

Der generische Push/Relabel-Algorithmus setzt sich wie folgt zusammen:

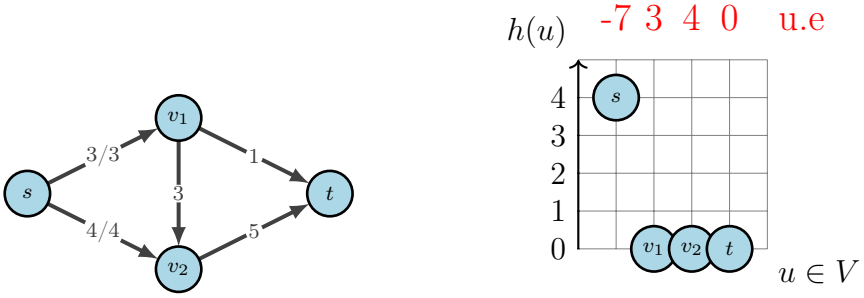
Algorithm 5 Generic-Push/Relabel

1: Initialize-Preflow(G,s)
2: **while** es existiert eine ausführbare Push- oder Relabel-Operation
3: $\quad$ wähle eine ausführbare Push- oder Relabel-Operation und führe sie aus

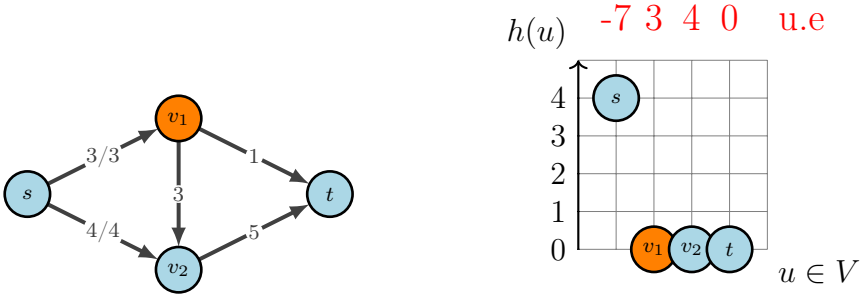
Auf den nachfolgenden Seiten soll ein Beispiel für die Ausführung des generischen Push/Relabel-Algorithmus gegeben werden.



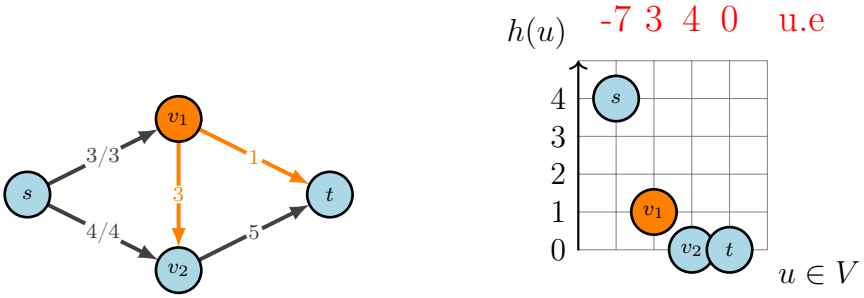
(a) Initiale Konfiguration. Links sind Flussnetzwerk und Fluss zu sehen, rechts werden die Höhe der Knoten im Diagramm, sowie deren Flussüberschuss in rot aufgeführt.



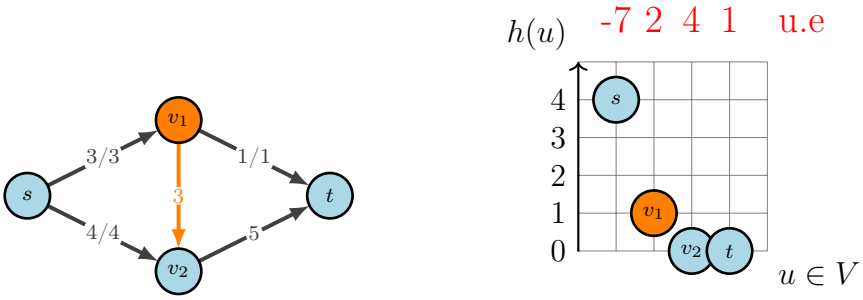
(b) Nach Initialize-Preflow



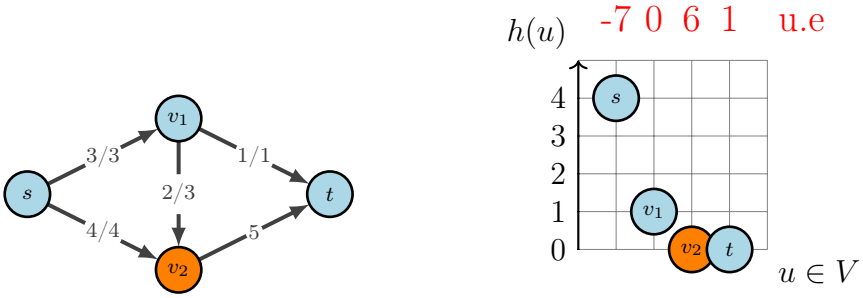
(c) Der aktuell betrachtete Knoten wird orange dargestellt. RELABEL(v_1)



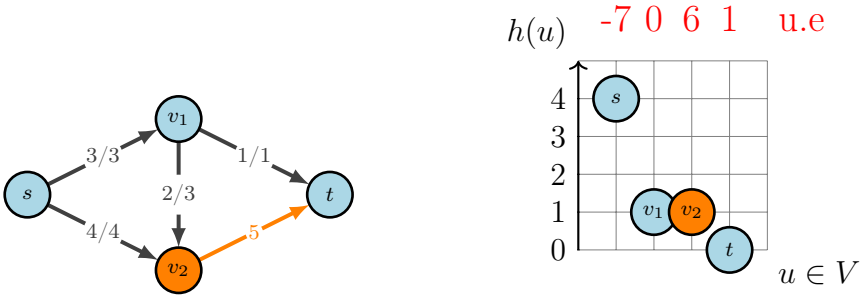
(d) Alle verfügbaren Push-Operationen werden ebenfalls markiert. PUSH(v_1, t)



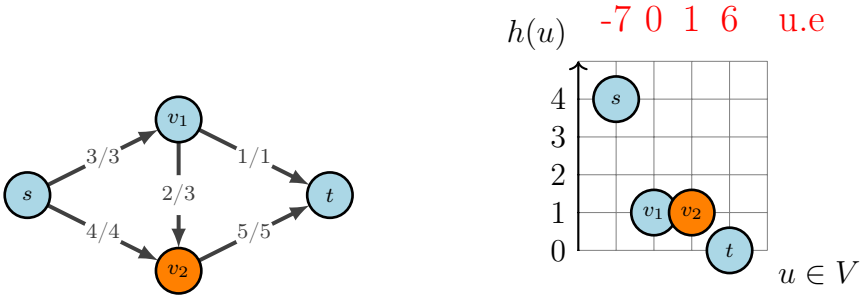
(e) PUSH(v_1, v_2)



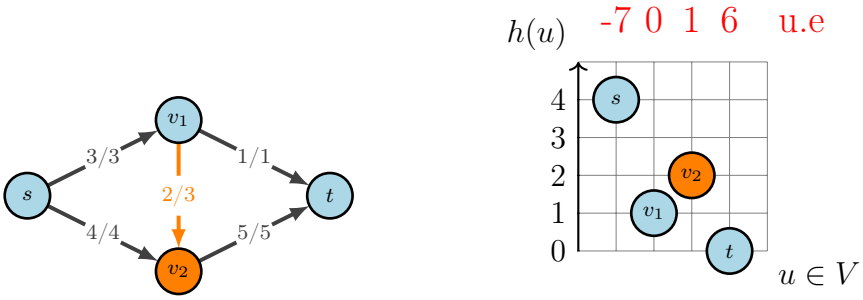
(f) RELABEL(v_2)



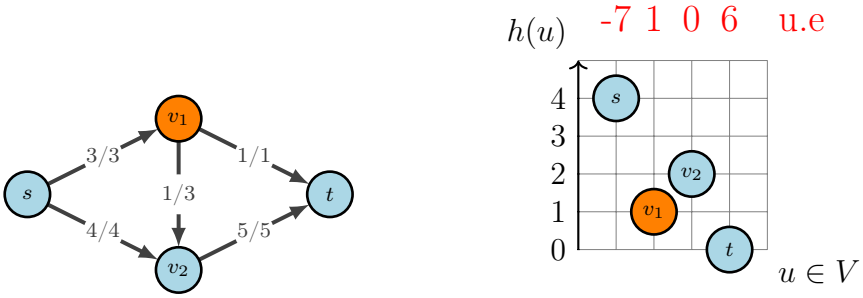
(g) PUSH(v_2, t)



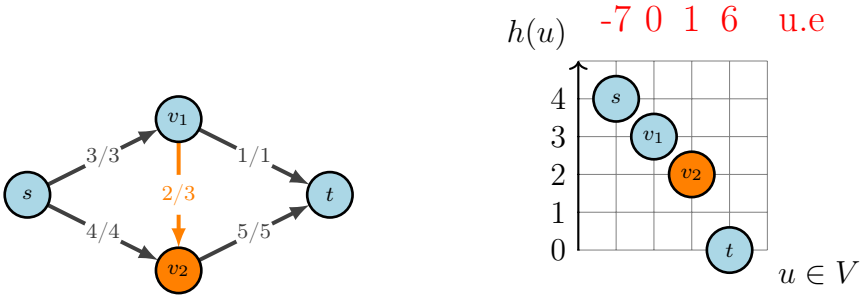
(h) RELABEL(v_2) ist wieder erlaubt, da $(v_2, t) \notin E_f$



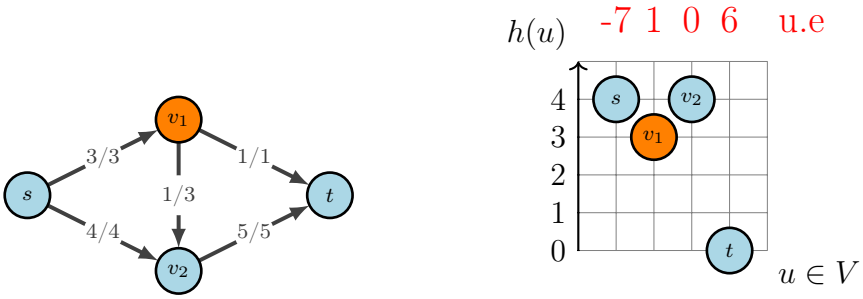
(i) PUSH(v_2, v_1). Die Push-Operation betrachtet Restkapazitäten, hier wird jedoch nur das Flussnetzwerk gezeigt.



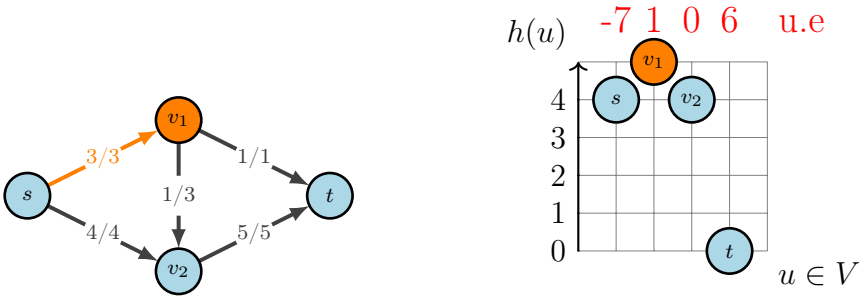
(j) RELABEL(v_1) und PUSH(v_1, v_2)



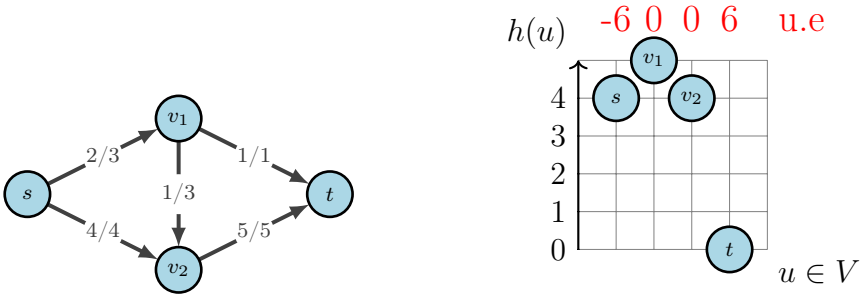
(k) RELABEL(v_2) und PUSH(v_2, v_1)



(l) RELABEL(v_1). Die Höhe von v_1 übersteigt $s.h$ und wir können den Flussüberschuss auflösen.



(m) PUSH(v_1, s)



(n) Finale Konfiguration. Es existiert keine ausführbare Push- oder Relabel-Operation mehr. Der Algorithmus hat einen maximalen Fluss f mit $|f| = 6$ gefunden. Offensichtlich gilt bei Terminierung immer $|f| = -s.e$ für einen maximalen Fluss f.

3.5 Analyse des generischen Push/Relabel-Algorithmus

Die Korrektheit des Push/Relabel-Algorithmus lässt sich über einige Beobachtungen zur Höhenfunktion, sowie eine geeignete Schleifeninvariante beweisen. Die Terminiertheit erhält man aus Abschätzungen zu der Anzahl an Push- und Relabel-Operationen. Daraus ergibt sich auch eine Schranke für die maximale Anzahl an Operationen und wir erhalten eine Laufzeit des generischen Push/Relabel-Algorithmus von insgesamt $\mathcal{O}(V^2 \cdot E)$. Diese Lösungsstrategie ist also schneller als die der FORD-FULKERSON-Algorithmen, lässt sich aber noch weiter verbessern. Auch hier ist die Reihenfolge, in der wir die Operationen ausführen bestimmend für die Laufzeit. Eine Erweiterung des generischen Algorithmus ist der Relabel-to-Front-Algorithmus. Er verwaltet neben Höhenfunktion und Flussüberschüssen auch noch eine Liste der Knoten des Netzwerkes. Beginnend am Anfang der Liste wird nach einem überfluteten Knoten u gesucht. Der Algorithmus versucht dann auf diesen so lange Push- oder Relabel-Operationen anzuwenden, bis u nicht mehr überflutet ist. Sobald ein Knoten ummakiert wird, wird er außerdem an den Anfang der Liste verschoben. Daher kommt auch der Name des Relabel-to-Front-Algorithmus. Er erreicht eine Laufzeit von $\mathcal{O}(V^3)$, was mindestens genau so gut ist, wie die generische Implementierung.

4 Zusammenfassung

Wir haben gesehen, mit welch unterschiedlichen Strategien sich das maximaler-Fluss-Problem lösen lässt. Während die FORD-FULKERSON-Algorithmen durch einen recht globalen Lösungsansatz Erweiterungspfade identifizieren, arbeiten Push/Relabel-Algorithmen wesentlich lokaler. Beide Algorithmen lassen diverse Implementierungen zu, die sich in verschiedenen Laufzeiten wiederspiegeln. Wir haben zu jeder Algorithmenfamilie eine Verbesserung angeschnitten, deren Laufzeiten besser waren, als die ihrer generischen Pendants. Einige der derzeit schnellsten bekannten Algorithmen entstammen der Klasse der Push/Relabel-Algorithmen, es gibt jedoch noch schnellere Algorithmen, die sich keiner der beiden hier besprochenen Klassen zuordnen lassen (z.B. der Algorithmus von Goldberg und Rao)

Besonders über den Zusammenhang mit den Schnitten, den wir nur kurz besprochen haben, erhalten wir interessante Anwendungsfälle. Das automatisierte Unterscheiden zwischen Vordergrund und Hintergrund in einer Bildbearbeitungssoftware lässt sich beispielsweise auf ein maximaler-Fluss-Problem zurückführen. Auch zum effizienten Errechnen maximaler bipartiter Matchings können diese Algorithem benutzt werden, zum Beispiel bei der bestmöglichen Zuteilung von Ressourcen zu Verbrauchern. Die hochinteressante Frage, ob ein Baseball-Team noch die Playoffs erreichen kann, lässt sich ebenfalls als maximaler-Fluss-Problem vormulieren. [Kin16]

5 Literatur

[Cor+09] Thomas H. Cormen u. a. *Introduction to Algorithms*. 3. Aufl. The MIT Press, 2009. ISBN: 978-3-486-74861-1.

[Haa20] Adrian Haarbach. *Goldberg Tarjan Push Relabel Algorithm*. [Online; accessed 20-July-2020]. 2020.

[Kin16] Toby Kingsman. *Baseball Elimination Problem*. [Online; accessed 27-September-2020]. 2016.