Max-Flow-Min-Cut Algorithmen

Projektunterricht

Verfasser: Nandor Kovacs

https://github.com/NandorKovacs/flow

MNG Rämibühl

May 19, 2023

1 Problemstellung

Das Max-Flow-Min-Cut Theorem besagt, das der maximale Fluss auf einem Flussnetzwerk, und der minimale Schnitt des Netzwerks equivalent ist. Ein Flussnetzwerk ist ein Graph mit einer Quelle, und einer Senke. Das kann man sich so vorstellen, das aus der Quelle unendlich viel Wasser fliessen kann, und die Senke unendlich viel Wasser schlucken kann.

Diese Werte sind aus vielen Gründen sehr interessant für einen Graph. Sie finden in der realen Welt viele anwendungen. Hat man Beispielsweise ein Netz aus Röhren und Schläuchen, und man will wissen wie schnell man Wasser durchpumpen kann, ist der maximale Fluss dieses Flussnetzwerks die Lösung. Erweiterungen dieser Algorithmen werden benutzt um die Wasserzufuhr von Gebieten optimal zu konstruieren. Will man neue Wohngebiete oder Industrieanlagen bauen, brauchen die genug Wasser. Wo es optimal ist die bestehende Wasserinfrastruktur auszubauen, kann man mit solchen Algorithmen gut bestimmen.

Eines der ersten Anwendungen, die zur Forschung dieser Algorithmen führten, war eine taktische Kriegsfrage. Wenn man ein Strassennetzwerk hat, auf den Soldaten losgeschickt werden, will man die möglichst effizient aufhalten. Wo es optimal ist Barrikaden aufzubauen, ist die gleiche Frage, wie wo der minimale Schnitt des Netzwerks ist. Um dieses Ziel zu lösen wurde der Ford-Fulkerson Algorithmus entwickelt, einer der ersten, und heute noch einer der meistbekannten Lösungen für das Max-Flow-Min-Cut Problem.

Ungefähr Zeitgleich versuchte man auch mit computergestützten Rechnungen die Logistik von der Nahrungsverteilung zwischen Städten und Dörfern zu verbessern. Es war überraschend, das der gleiche Algorithmus funktionierte. Dies schuf das Max-Flow-Min-Cut Theorem, das heute bewiesen ist.

Noch wichtiger ist die Version des Problems, wo das Flussnetzwerk auch noch Kantengewichte hat die die jeweiligen Kosten für einen bestimmten Weg darstellen. Diese Version wird für die oben beschriebene Wasserinfrastruktur verwendet, da es in so einem Fall wichtig ist wie viel Geld es kostet bestimmte Stellen auszubauen. So ein Problem kann man beispielsweise mithilfe von lineares Programmieren lösen.

In diesem Bericht werde ich zwei Algorithmen vorstellen, die ich gelernt und selber implementiert habe.

2 Lösung mit naiven Ansätzen

2.1 Was wäre der naheliegendste Ansatz an das Problem?

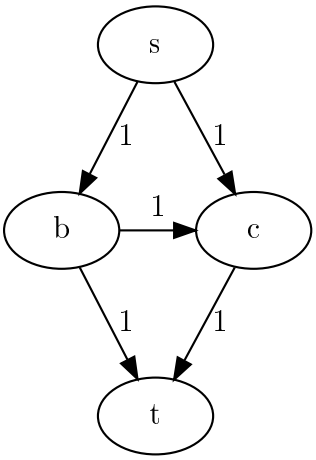
Meine erste Idee wäre, einfach einen beliebigen Pfad zu finden von der Quelle zur Senke, beispielsweise mit dem bekannten Dijkstra Algorithmus. Jetzt würde ich schauen, wie viel Wasser ich entlang dieses Pfades schicken kann. Das würde dem minimum der Kantengewichte entlang des Pfades entsprechen. Jetzt würde ich alle Kantengewichte entlang des Pfades aktualisieren, indem ich so viel abziehe wie ich Wasser entlanggeschickt habe. Alle Kanten die dabei 0 erreichen würde ich entfernen.

```
g ← Flussnetzwerk;
result ← 0;
while pfad ← dijkstra(g) do
    wassermenge ← ∞;
    foreach kante ∈ pfad do
        wassermenge ← min(wassermenge, kante.kapazitaet)
    end
    result ← result + wassermenge;
    foreach kante ∈ pfad do
        kante.gewicht ← kante.gewicht − wassermenge;
        if kante.gewicht = 0 then
            g.loesche(kante);
        end
    end
end
```

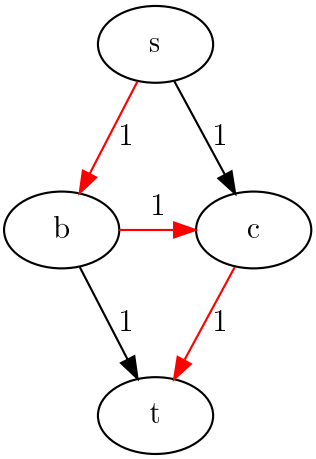
Algorithm 1: Naiver algorithmus mit Dijkstra

2.2 Wieso funktioniert das nicht?

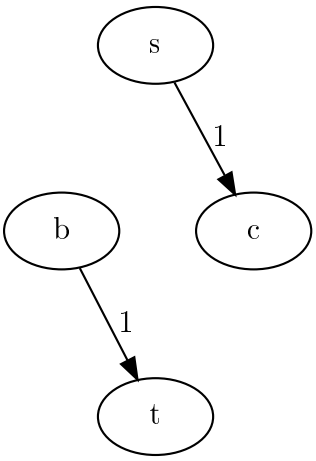
Nehmen wir folgendes Flussdiagramm als Beispiel:



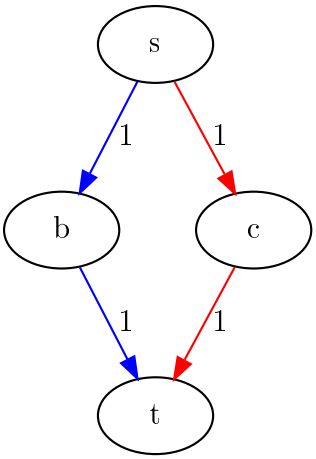
Zuerst wählen wir einen Zufälligen Pfad von s nach t aus:



Jetzt nehmen wir von jeder Kante 1 Gewicht Pfad, da wir 1 Wasser entlang des Pfades schicken. Somit verfallen die Kanten auf dem roten Pfad, da alle nur ein Kantengewicht von 1 haben.

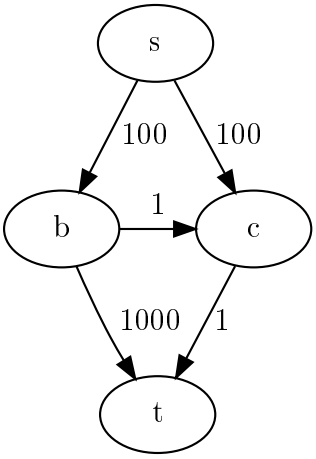


Jetzt hat es keinen Pfad mer von s nach t. Somit kämen wir zu der Lösung, das der maximale Fluss auf diesem Netzwerk 1 ist. Wir haben hier aber einen Beispiel, wo wir 2 Wasser über das Netzwerk schicken können:



2.3 Wieso hat das nicht funktioniert?

Nur weil wir einen Pfad gefunden haben, heisst das noch lange nicht das wir den auch benutzen wollen für die optimale Lösung. Bei folgendem Beispiel ist es ziemlich klar, das es keinen Sinn macht die kannte (b, c) zu benutzen. Wir haben eine riesige kapazität auf (b, t), und nur ganz wenig auf (c, t). Wir möchten alles Wasser wo bei b ankommt sofort durch (b, t) schicken, damit so viel kapazität wie möglich auf (c, t) übrig bleibt für Wasser, das bei c ankommt.



Natürlich könnten wir Glück haben, so dass wir zufälligerweise genau die richtige Pfade in der richtigen Reihenfolge auswählen. Aber wenn wir das nicht garantieren können, ist unser Algorithmus nicht korrekt.

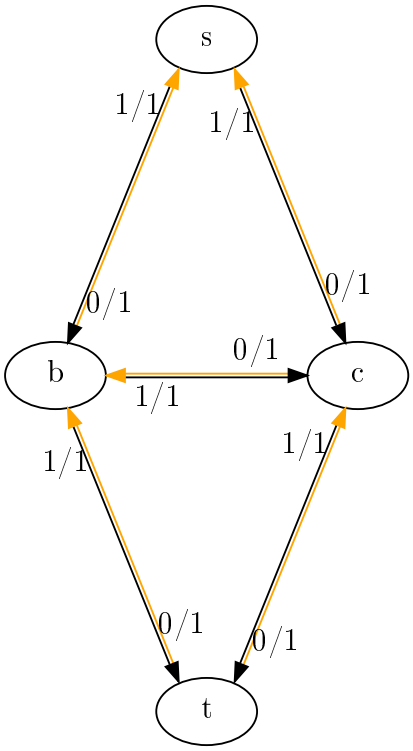
3 Die Ford-Fulkerson Methode

3.1 Die Idee

Das Prinzip der Ford-Fulkerson Methode ist das gleiche wie das Prinzip vom beschriebenen naiven Algorithmus. Wir suchen einen Pfad von s nach t, und lassen so viel Wasser wie es geht durchfliessen.

Der einzige Unterschied ist, das wir beim Ford-Fulkerson für jede Kante eine Kante einfügen, die in die entgegengesetzte Richtung geht, die gleiche Kapazität hat, und immer umgekehrt so viel Wasser darauf fliesst, wie auf der Partnerkante. Anstatt Kanten allmählich zu entfernen, verändern wir unseren Dijkstra so, dass es Kanten auf denen gleich viel Wasser ist wie ihre Kapazität nicht in betracht zieht.

Die jetzt eingefügte Rückkanten bilden einen sogenannten Restnetzwerk.



Die Intuition dahinter ist, das wir bei einer schlechten Pfadwahl, das unser Resultat wie vorher kaputt machen würde, den schlechten Teil zurückfahren können. Anfangs sind die Kantengewichte des Restnetzwerks gleich gross wie dessen Kapazitäten, also können wir sie nicht benutzen. Sobald wir eine Kante aber benutzt haben, können wir dessen Rückkante benutzen.

3.2 Beweis

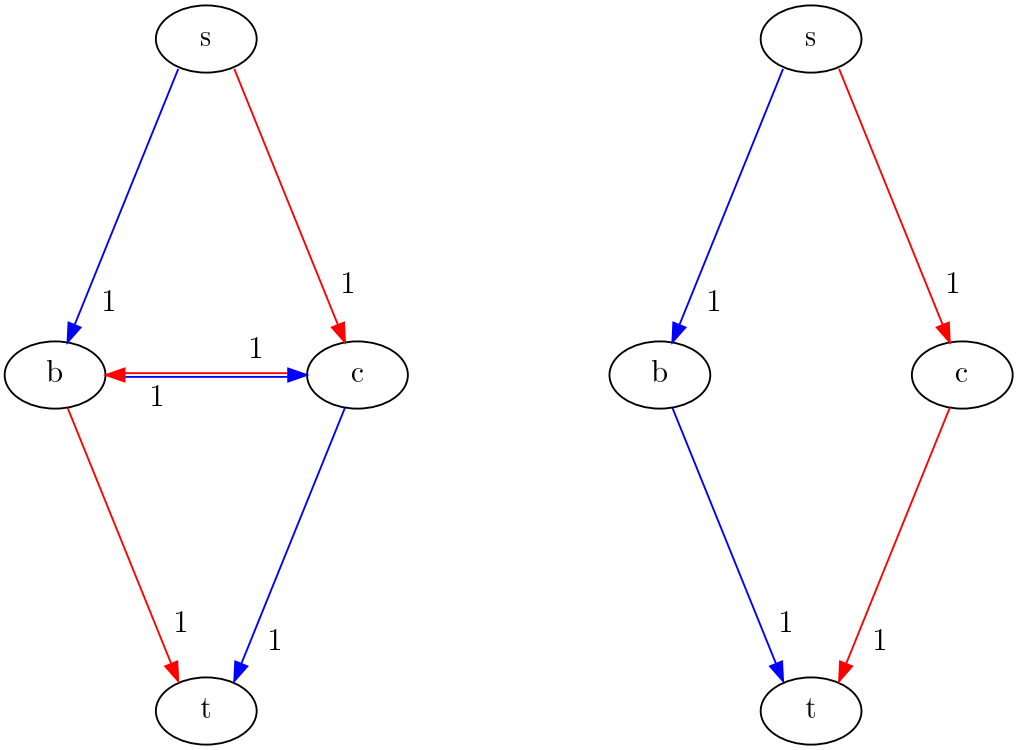
3.2.1 Der Algorithmus findet keine unmögliche Lösungen

Das unser naiver Algorithmus keine unmögliche Lösungen gibt, ist trivial zu sehen. Alle Pfade können nacheinander ausgeführt werden, und gehen von der Quelle bis zur Senke.

Haben wir nun eine Kante, dessen Restkante wir benutzen, können wir das immer so umstellen, das wir dies nichtmehr tun. Benutzen wir die Kante (a, b) mit Gewicht x, und rückwärts mit Gewicht y, wissen wir das $x \geq y$ wegen der Regel das Kanten im Restnetzwerk immer so viel Platz haben wie Platz verbraucht ist auf der Kante im Flussnetzwerk. Wir wissen nun,

das man vom Knoten b mindestens x Wasser zu t befördern können, und vom Knoten a mindestens y Wasser zu t befördern können. Grund dafür ist, das wir Pfade, die in t enden, gefunden haben die durch (a, b) gehen mit der Gewichtssumme x, und Pfade, die durch (b, a) gehen mit der Gewichtssumme y.

Jetzt verändern wir die Pfade so, das wir die Rückkante nicht benutzen. Wir schicken nur noch $x - y$ Wasser durch (a, b), und kein Wasser durch (b, a). Die Menge an Wasser, die von a abfliessen muss, ändert sich nicht, da jetzt y weniger kommt von b, aber auch y weniger weggeschickt wird nach b. Genau gleich muss nicht mehr Wasser von b abfliessen, da jetzt zwar y weniger nach a geschickt wird, aber auch y weniger kommt von a.



3.2.2 Der Algorithmus findet die optimale Lösung