Ein Quantenalgorithmus zur Lösung des Maximalen Fluss Problems

Einleitung

Quanten Computing ist ein interdisziplinäres Forschungsgebiet der Mathematik, Physik und Informatik. Im Wesentlichen geht es um die Herausstellung der Überlegenheit von Computern, die mit einem Quantenmodell bestimmte Aufgaben lösen.

Um gleich zu Beginn einen Bezug zur Praxis herzustellen werden geeignete Fragen gestellt, die auf das Maximal Fluss Problem reduziert werden können. Mögliche Fragen sind: 1. Wie viel Autos können von A nach B über ein gewisses Verkehrsnetz geleitet werden? 2. Gibt es Engpässe innerhalb des Netzes? 3. Wie viel Daten können Maximal über das Netzwerk transportiert werden? 4. Wie groß muss die Kapazität der Abwasserleitung eines Hauses oder der Stadt sein, damit ein Abfluss gesichert ist?

Viele weitere Fragestellungen können auf die Frage nach dem Maximalen Fluss reduziert werden. Dies stellt seine Wichtigkeit heraus. Zudem gibt es noch weitere abstraktere Probleme wie etwa das Min-Cut-Problem, welches man auf das Problem des Maximalen Flusses zurückführen kann. Dies stellt gute Gründe dar um sich intensiv mit dieser Problematik und den neuen Möglichkeiten des Quanten Computings in diesem Bereich auseinanderzustezen.

Grundlagen

In diesem Kapitel werden die Theoretischen Grundlagen vermittelt, die sowohl zum Verständnis des Problems als auch zum Verständnis des Quanten Computings benötigt werden.

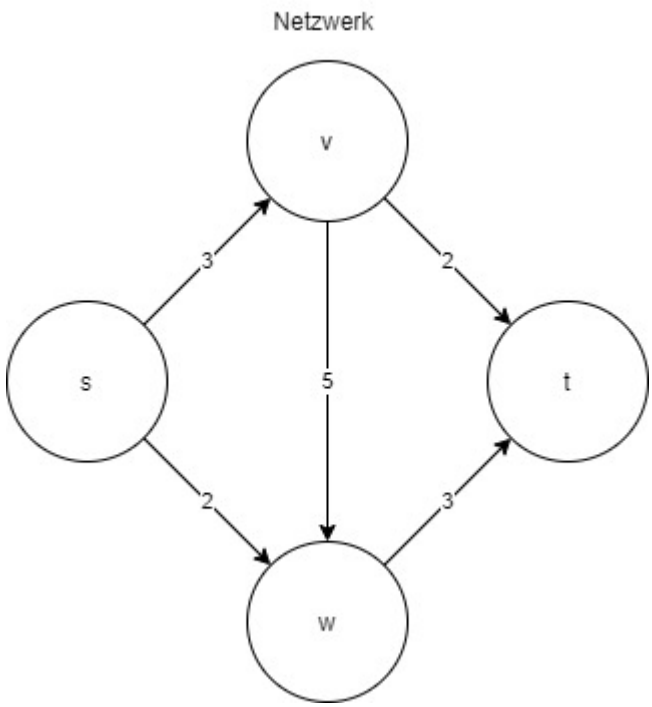
Graphen Theorie

Um das Problem des Maximalen Flusses zu formulieren ist es wichtig einige methematische Begriffe zu definieren. Als erstes soll der Begriff des Graphen genauer betrachtet werden. Ein Graph G ist definiert durch eine Liste, diese besteht aus einer Menge von Knoten und aus einer Menge von Kanten $G = (V, E)$.

Die Menge der Knoten ist wie folgt definiert: $V = \{v_1, v_2, v_3, \ldots, v_n\}$. In gleicher Weise kann man die Menge der Kanten definieren: $E = \{e_1, e_2, e_3, \ldots, e_n\}$

Nimmt man den Graph G als Grundlage und fügt eine Funktion c hinzu, die jeder Kante eine ganze Zahl als Kapazität zuweist und definiert darüber hinaus genau einen Knoten als Quelle (engl. source) sowie genau einen Knoten als Senke (engl. target) dann hat man ein Netzwerk definiert. Formal wird eine Netzwerk N wie folgt definiert $N = (G, c, s, t)$.

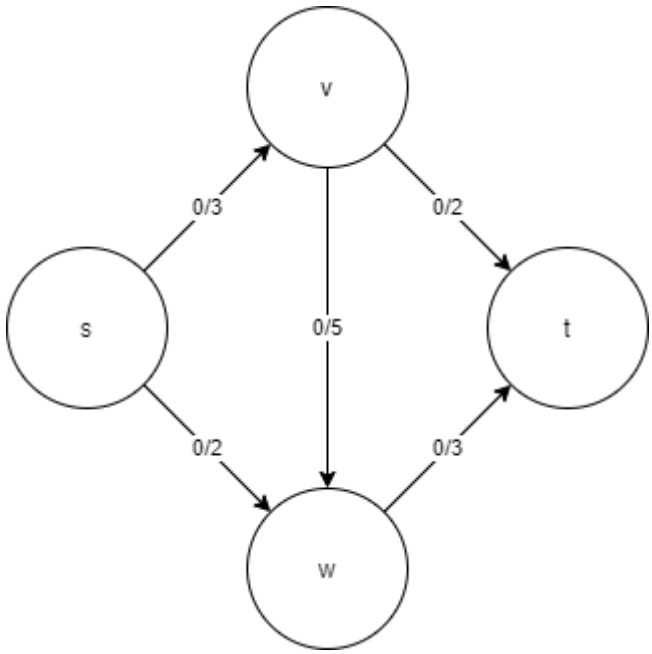
Es muss gelten: $c : E \to R_+$, $s, t \in V$ und $s \neq t$. Im folgenden wird ein Beispiel eines Netzwerks dargestellt.



[Abbildung 1]

Zu letzt bleibt die Frage offen: "Was ist ein Fluss?" Nimmt man das obige Bild als Bezug, so gibt die Kantenbewertung die Kapazität an. Also wie viel Einheiten von etwas über diese Kante von einen zum anderen Knoten fließen kann, in der Praxis sind das etwa die Anzahl der PKW die pro Zeiteinheit eine Straße passieren. Zum Beispiel kann von s nach v maximal 3 Einheiten von etwas fließen.

Wenn man jetzt beispielsweise eine $0/$, vor jede Kantenkapazität schreibt dann beschreibt diese Belegung jeder Kante den Leerenfluss. Der leere Fluss ist im folgenden zu sehen.



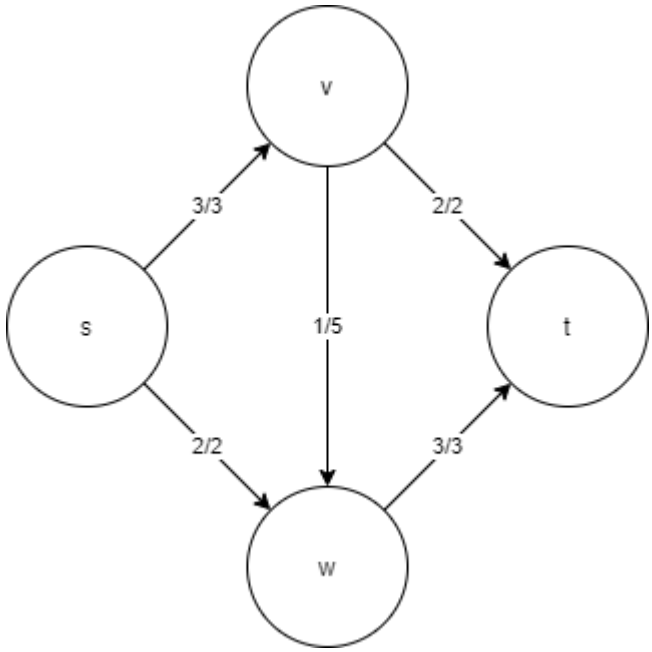
[Abbildung 2]

Ein Fluss ist ein Netzwerk, welches eine zusäzliche Funktion f besitzt. Diese Funktion wird Flussfunktion genannt und weist jeder Kante einen Flusswert zu, formal: $f = E \rightarrow R_+$. Damit der Fluss valide ist müssen zwei Bedingungen erfüllt sein: Erstens die Kapazitätskonformität - formal: $0 \leq f(e) \leq c(e)$, d.h.: Der Fluss an einer Kante ist nicht kleiner als Null und höchstens so groß wie ihre Kapazität. Zweitens der Flusserhalt - formal
$\forall v \in V \setminus \{s,t\} : \sum_{e \in E_{out}(v)} f(e) = \sum_{e \in E_{in}(v)} f(e)$, d.h.: In einen Knoten fließen genau der Wert hinaus, der hinein fließen.

Man stelle sich vor man hat einen beliebigen Knoten v_n dieser besitzt mehrere zuführende Kanten und über diese fließen 12 Einheiten von etwas hinein, dann müssen ebenfalls 12 Einheiten wieder abfließen, dies ist unabhängig von der Zahl der Kanten.

Der **Flusswert** wird formnal wie folgt definiert:
$\Phi(f) := \sum_{e \in E_{out}(s)} f(e) - \sum_{e \in E_{in}(s)} f(e) = \sum_{e \in E_{in}(t)} f(e) - \sum_{e \in E_{out}(t)} f(e)$. Der Wert eines Flusses ergibt sich aus Differenz dessen was aus der Quelle heraus fließt und was in die Quelle hinein fließt. Diese Differenz muss gleich der Differenz dessen was in das Ziel hinein fließt abzüglich dem was heraus fließt sein. In den meisten Fällen fließt nichts aus dem Ziel ab und ebenso nicht in die Quelle hinein.

Der **Maximale Fluss** ist der Fluss mit einem bestimmten Wert n, welcher für keine weitere Belegungen der Kanten größer wird. Der Maximale Fluss für das obige Beispiel des $Netzwerkes\ N$ ist im folgenden zu sehen



[Abbildung 3]

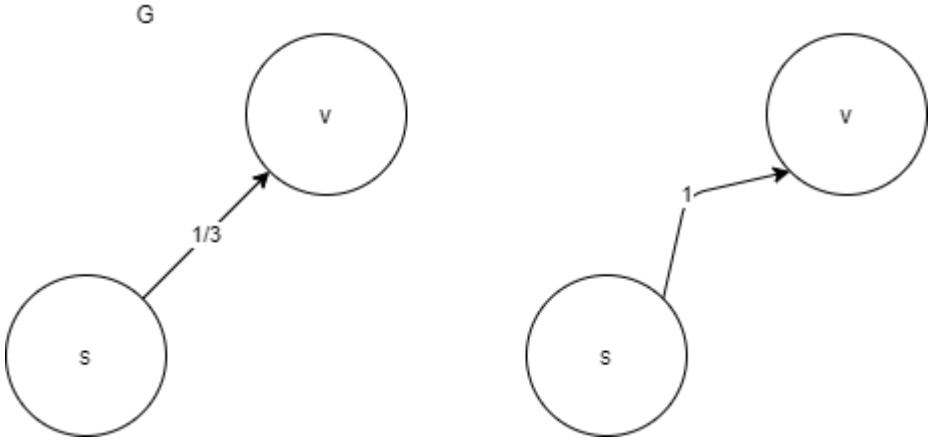
Der Flusswert beträgt fünf und in diesem Beispiel wird ersichtlich, dass keine weitere Änderung an den Kanten zu einer Erhöhung des Flusses führen würde. Die Suche nach einem **Maximalen Fluss** in einem beliebigen Graphen wird als **Maximaleflussproblem** bezeichnet. Das Beispiel in diesem Abschnitt war sehr überschaubar und auch ein Brute-Force-Ansatz führt zur Lösung. Mit zunehmender Komplexität des Graphens benötigt man zur Lösung effiziente Algorithmen. Die Kömplexität eines Graphen kann mit zunehmender Zahl der Knoten und oder Kanten sehr schnell steigen.

Algorithmen

Das Maximaleflussproblem ist ein altbekanntes Problem und es gibt viele Algorithmen um es zu lösen. Der aktuellste Algorithmus zum Lösen des Maximalen Fluss wurde 2012 von Orlin et al. (https://de.wikipedia.org/wiki/Fl%C3%BCsse_und_Schnitte_in_Netzwerken#Algorithmen) [1] veröffentlicht und hat eine Laufzeit von $\mathcal{O}(n \cdot m)$.

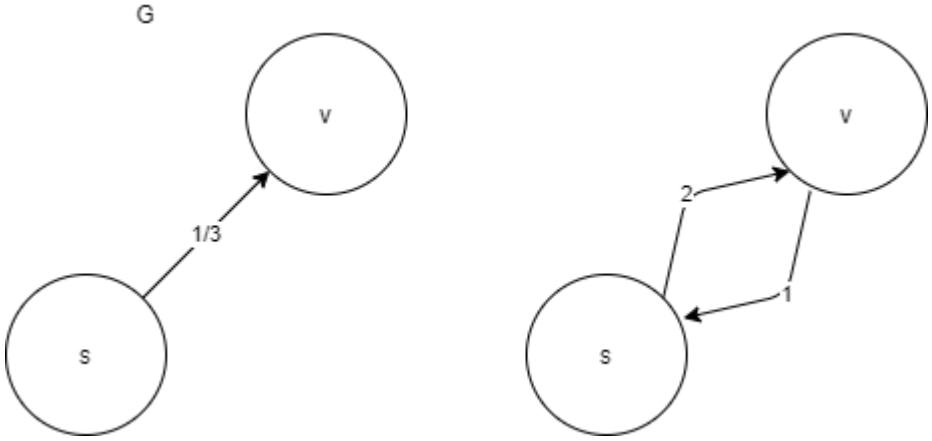
Einen vergleichsweise einfachen Algorithmus, stellt der Edmonds-Karp-Algorithmus (http://theory.stanford.edu/~tim/w16/l/l2.pdf) [2] dar. Dieser wird jedoch durch die Erweiterung von Quanten Computing Prinzipien beschleunigt. Um den Algorithmus zu verstehen werden einige Erläuterungen und Definitionen gegeben.

Am Anfang eines Flussproblems steht ein $Graph\ G$ von diesem wird der $Residualgraph\ G_f$ abgeleitet. Ein **Residualgraph** ist ein exakter Klon von G. Die alten von G übernommenen Kanten heißen **Vorwärtskanten**. Zusätzlich werden an allen Vorwärtskanten, Kanten die in die entgegengesetzte Richtung verlaufen erzeugt. Diese neu erzeugten Kanten werden als **Rückwärtskanten** bezeichnet. Ein Residualgraph kann auch als Restwertgraph bezeichnet werden. Die Kapazität der Vorwärtskanten im Restgraph wird wie folgt berechnet: $c_{neu}(e) = c_{alt}(e) - f(e)$ d.h. man nimmt die Kapazität der Kante e und mindert sie um den aktuellen Fluss über diese Kante, dann erhält man den Restwert der Kante in G_f. Ein Beispiel für die das Berechnen des Wertes an der Vorwärtskante ist im Folgenden zu sehen.



[Abbildung 4] Anmerkung an der Kante von Abbildung 4, muss eine 2 stehen.

Zusätzlich wird die Rückwärtskante mit dem Wert dessen, des Fluss entrpechend von G über diese kante definiert. Zu sehen ist das hier.



[Abbildung 5]

An dieser Stelle sei angemerkt das die Suche nach Wegen bzw. Pfaden nur in G_f stattfindet und nicht in G. Mit Hilfe des Residualgraph und seiner Rückwärtskanten ist es möglich Pfade über die man einen Fluss geschickt hat Rückgängig zu machen, dies ist in dem normalen Graph G nicht möglich.

Edmonds-Karp-Algorithmus

Um den Algorithmus von Edmonds-Karp besser zu verstehen wird als ertes der Pseudocode gezeigt.

Edmonds-Karp Algorithm

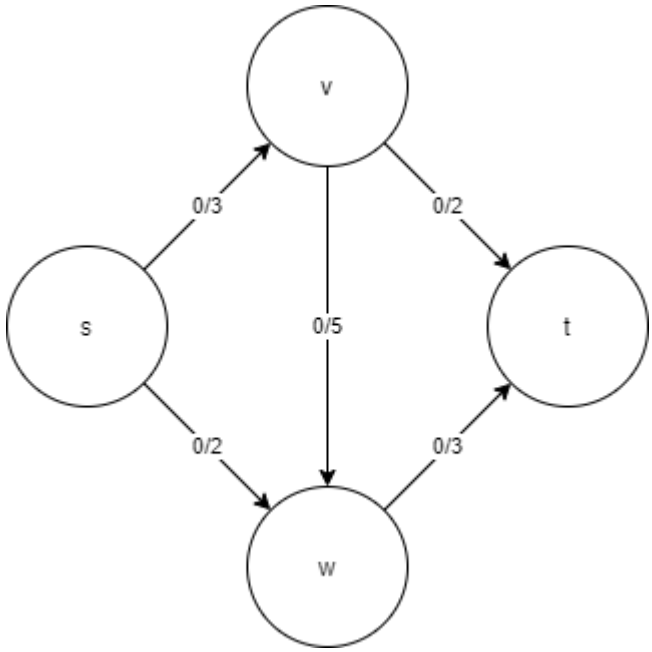
initialize $f_e = 0$ for all $e \in E$
repeat
  compute an s-t path P (with positive residual capacity) in the current residual graph G_f with the fewest number of edges
  `// takes` $O(|E|)$ `time using BFS`
  if no such path **then**
    halt with current flow $\{f_e\}_{e \in E}$
  else
    let $\Delta = \min_{e \in P}$ (e's residual capacity in G_f)
    `// augment the flow` f `using the path` P
    for all edges e of G whose corresponding forward edge is in P **do**
      increase f_e by Δ
    for all edges e of G whose corresponding reverse edge is in P **do**
      decrease f_e by Δ

[Abbildung 6: Pseudoalgorithmus]

Der Algorithmus beginnt mit der Initialisierung des Leeren Fluss. Als nächstes wird eine Schleife aufgerufen. Innerhalb dieser Schleife wird der Algorithmus ein weiteres mal durchlaufen. Es wird als erstes in G_f mittels der Breitensuche (BFS) nach einem Pfad P von s nach t gesucht. **WENN** das nicht geht, **DANN** gib den Leeren Fluss zurück. **SONST** nimm die Kante mit der geringsten Kapazität aus P und für den Fall das es eine Vorwärtskante ist erhöhe den Fluss entlang allen Kanten an G um diesen Wert. Ist es eine Rückwärtskante dann mindere den Fluss entlang der Kante in G.

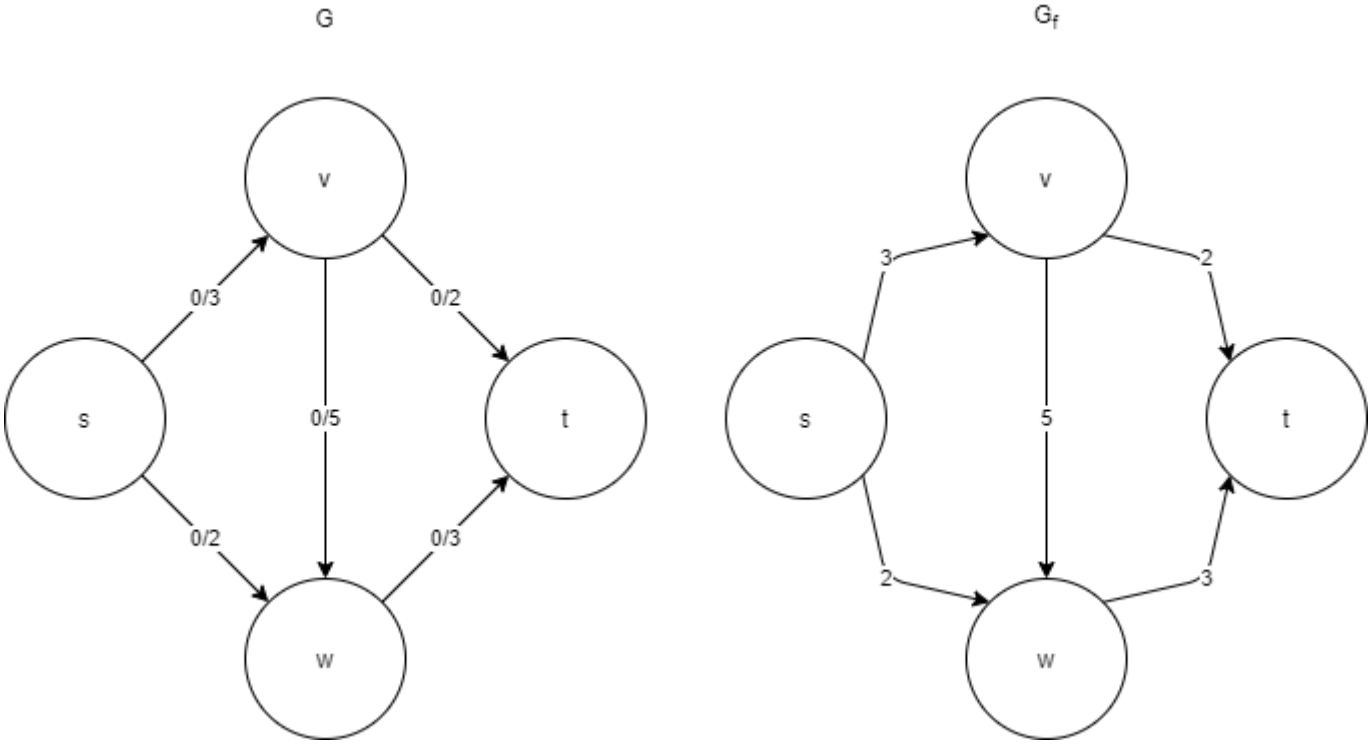
Das mittels der Breitensuche nach einem Pfad von s nach t gesucht wird führt dazu, dass die Nachfolger eines Knotens in Schichten ermittelt werden. Diese Schichten auch Layer genannt beeinhalten die in jeder Iteration die jeweils kürzesten Wege und dies stellt den emminenten Vorteil dieses Algorithmus dar. Ansonsten ist dieser Algorithmus stark an den Ford- Fulkerson-Algorithmus angelehnt Ford-Fulkerson-Algorithmus (http://timroughgarden.org/w16/l/l1.pdf).

Zur Veranschaulichung wird der EK-Algorithmus an dem Graph G erläutert. Ein Graph wird dem Algorithmus übergeben, nun wird dieser Graph mit dem Leeren Fluss initialisiert. Das führt nun zu folgendem Ergebnis.



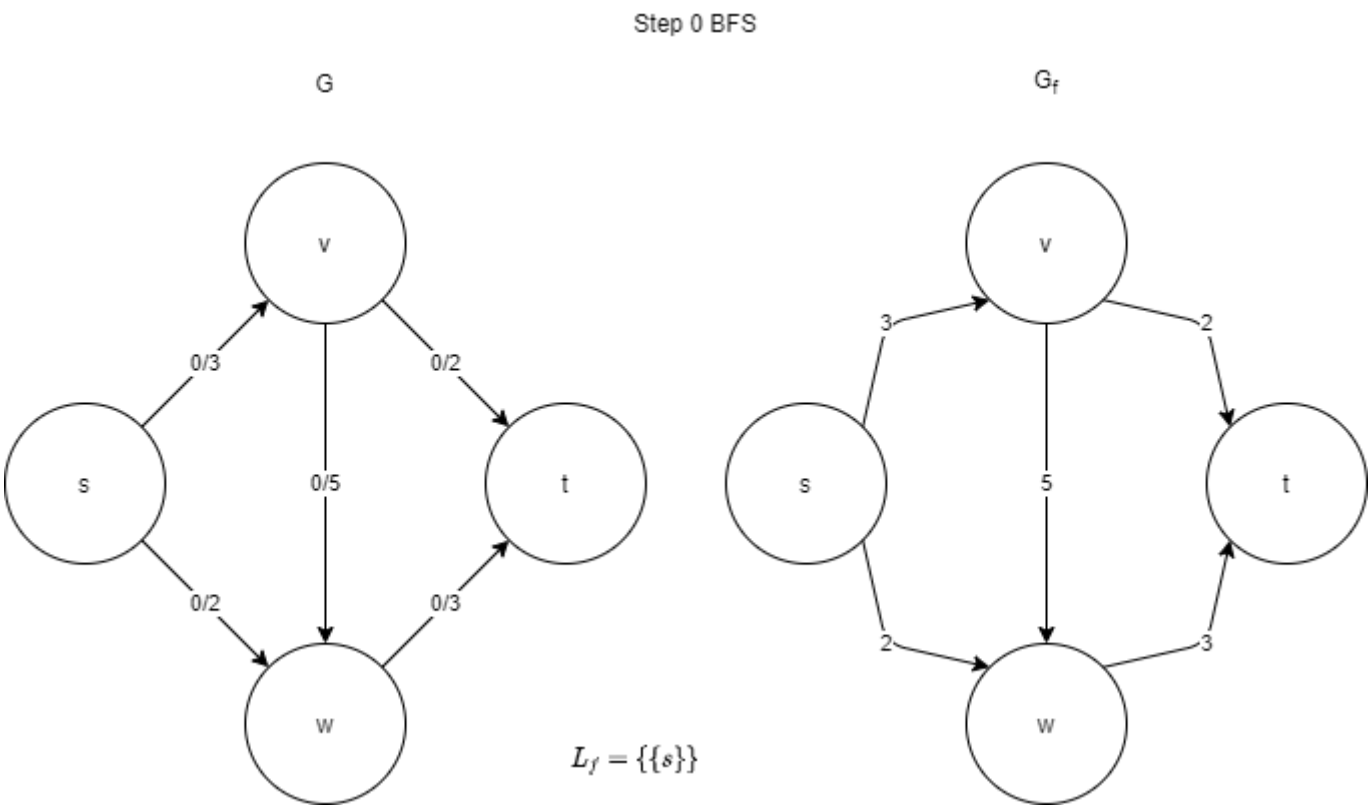
[Abbildung 7]

Als nächstes wird die **repeat** Anweisung betreten und es wird in G_f von einem Weg von s nach t gesucht. Als erstes wird G_f aus G konstruiert, wegen der Übersicht werden im Residualgraph G_f Kanten mit der Kapazität Null weggelassen.



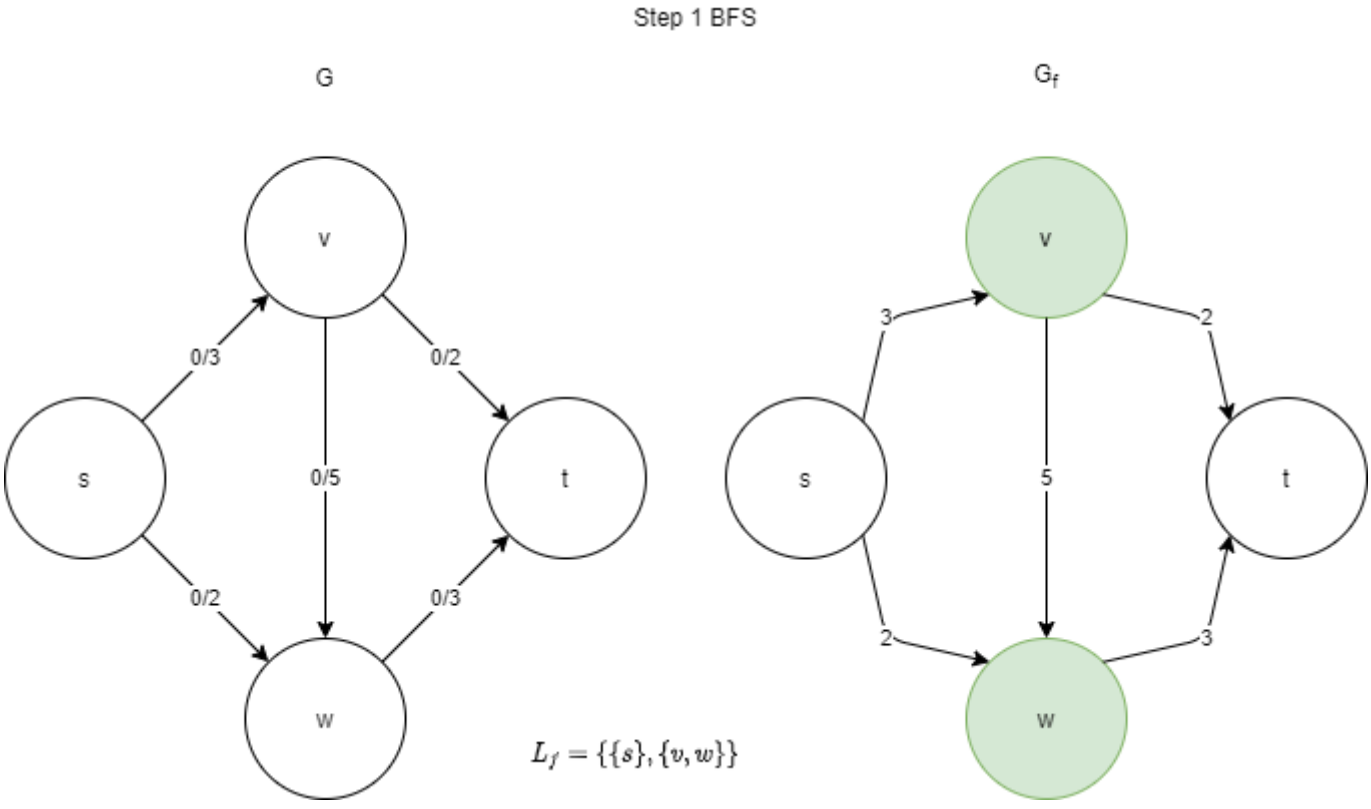
[Abbildung 8]

In dem gewonnenen Residualgraph wird mit BFS nach einem Weg von s nach t gesucht, hierbei wird eine weiterer Graph L_f aufgebaut. Eigentlich eher eine Menge, die geordnet wird nach der Entfernung von s. Dieser geschichtete Untergraph beeinhaltet als erstes Element s, in diesem Szenario startet die Breitensuche dargestellt im Folgenden.



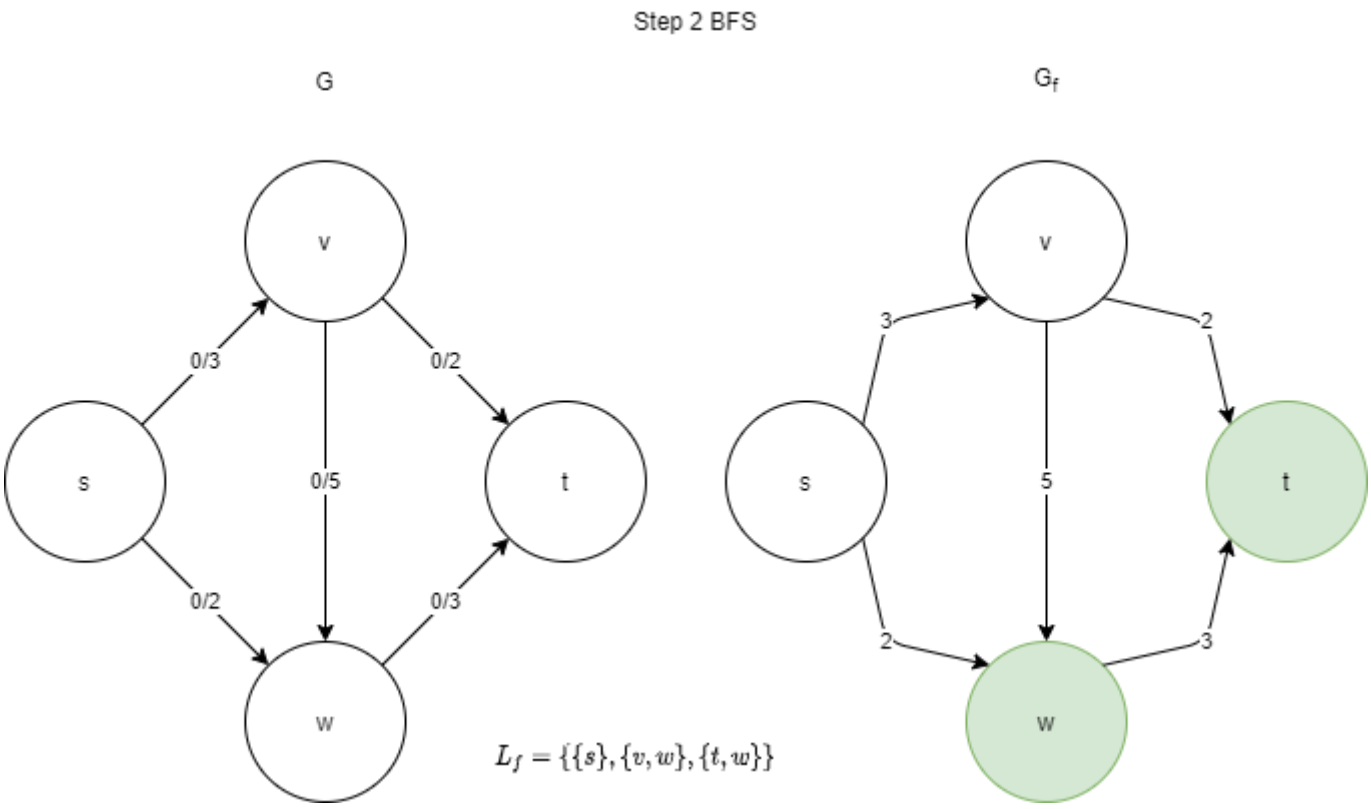
[Abbildung 9]

Die Breitensuche sucht alle Nachbarn des Startknoten und nimmt sie in die Menge L_f auf, das ist der Layered Subgraph. Das Ergebnis ist unten dargestellt.



[Abbildung 10]

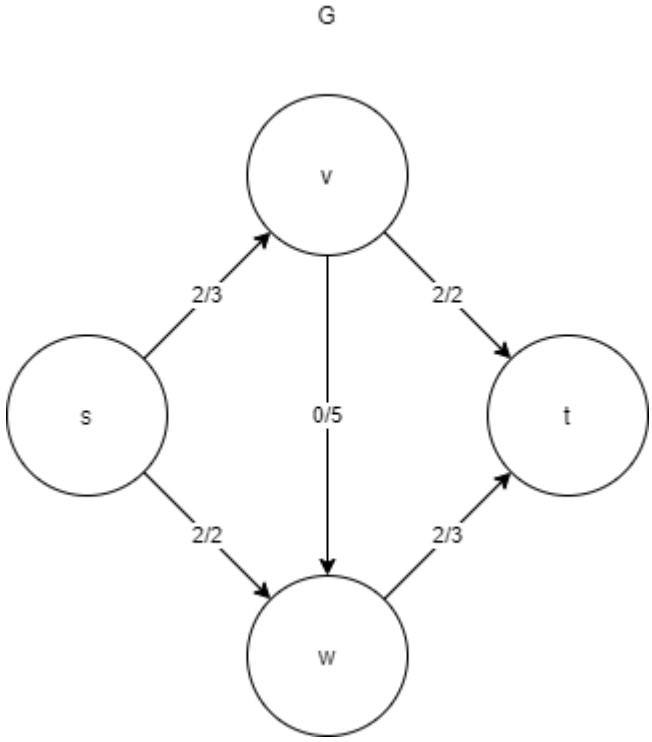
Die Suche wird nun von den Knoten v und w fortgesetzt das Ergebnis der Suche wiederum in L_f aufgenommen. Zu sehen ist dies in der Abbildung.



[Abbildung 11]

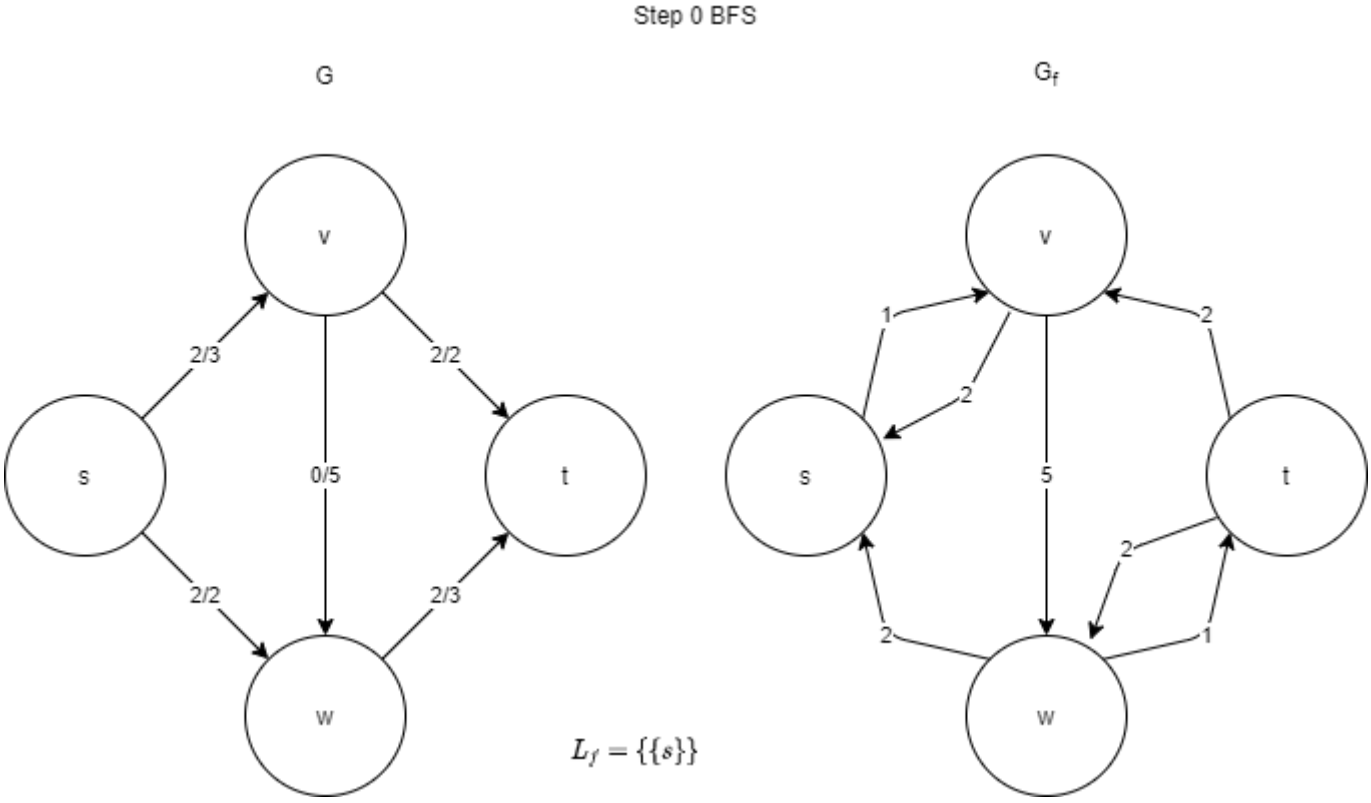
Die Breitensuche endet damit das ein Weg in G_f von s nach t gefunden wurde. Wegen vollständigkeit wurde w in L_f mit aufgenommen, es kann aber auch vernachlässigt werden. Wie schon erwähnt sind die Mengen in L_f bezüglich ihres Abstandes bzw. Distanzen zu s geordnet. Die Distanz beschreibt in diesem Zusammenhang die Anzahl der Kanten die Zwischen den Knoten liegen. So liegt beispielsweise zwischen s und v eine Kante, zwischen s und t liegen zwei Kanten.

Das Ergebnis der Breitensuche hat in der ersten Iteration des EK-Algorithmus zwei mögliche Pfade von s nach t ermittelt nämlich $\{s, v, t\}$ **und** $\{s, w, t\}$. Die weitere Abarbeitung des Algorithmus wird nun im ELSE-Zweig mit der Bestimmung der minimalsten Kapazität entlang des Weges P fortgesetzt, diese beträgt zwei. Anschließend wird der Graph G bzw. Fluss in G angepasst.



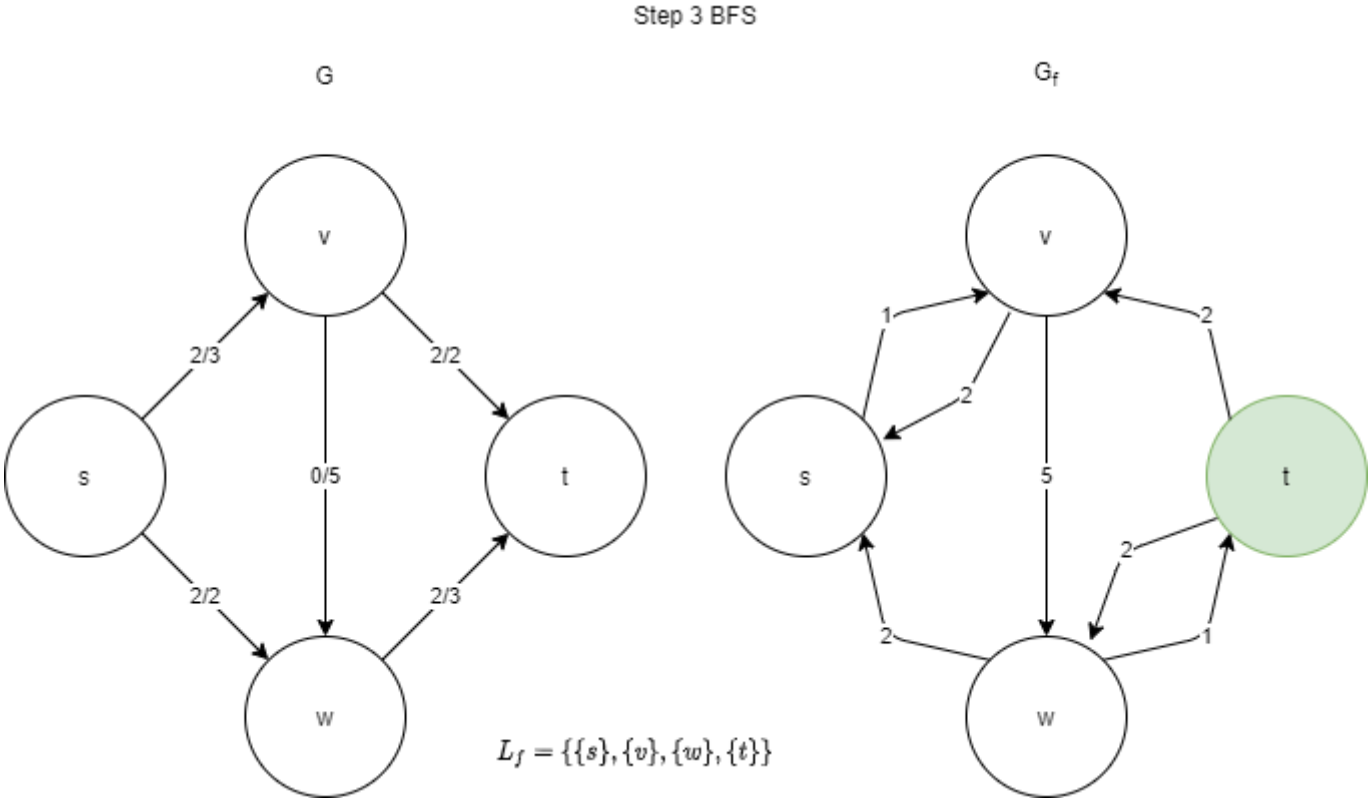
[Abbildung 12]

Der EK-ALgorithmus beginnt eine weitere Iteration. Im ersten Schritt dieser Iteration wird wieder der Residualgraph abgeleitet von dem aktuellen Fluss in G und L_f wird mit dem ersten Knoten s initialisiert.



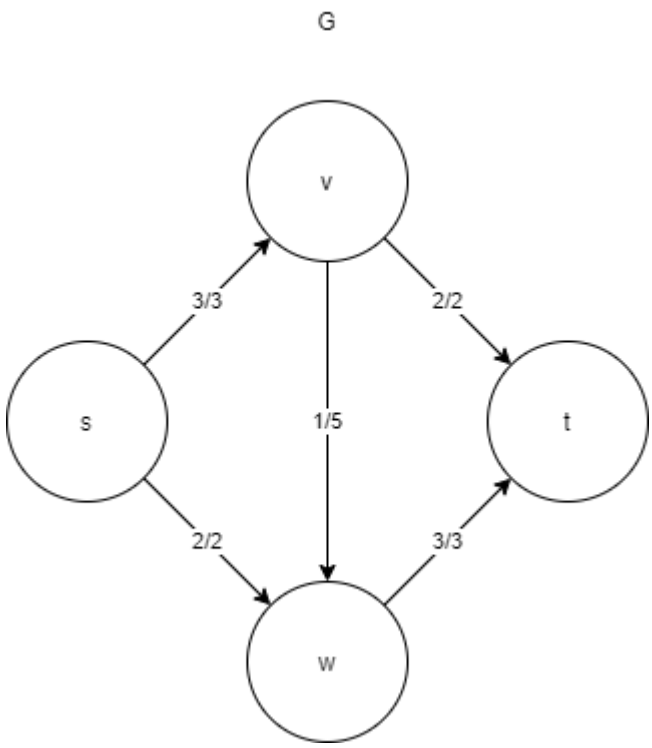
[Abbildung 13]

An dieser Stelle wird mit dem Ergebnis der Breitensuche fortgefahren.



[Abbildung 14]

Es wurde also ein weiterer Pfad P von s nach t gefunden. An dieser Stelle wird die Abarbeitung des Algorithmus im Else-Zweig mit der Bestimmung der minimalsten Kapazität fortgesetzt. Diese beträgt eins und nun wird der Fluss in G entsprechend angepasst. Im folgenden ist das Ergebnis von G dargestellt.



[Abbildung 15]

Jetzt wird ein letztes mal ein Residualgraph aufgebaut aber es kann kein Pfad nach t gefunden werden und an dieser Stelle terminiert der Algorithmus. Wenn keine weiteren Wege gefunden werden können, hat man einen Maximalen Fluss gefunden oder den Leeren. Als Suchalgorithmus dient die Breitensuche, in jeder Iteration müssen hierbei alle Kanten gegangen werden um die entsprechenden Nachfolger zu suchen. Im Beispiel wirkt dies trivial und ist wenig rechenaufwändig. In der Realität kann es aber sehr komplexe Graphen und extrem viele Knoten und oder Kanten geben. Die Komplexität riesiger Graphen stellt hierbei ein praktisches Hinderniss dar. Genau an dieser Stelle, der Pfadsuche, in G_f wird ein beachtlicher **speed up** durch den Einsatz von Quanten Computing erwartet.

Quanten-Computing-Grundlagen

Das Quanten Computing bringt viele neue Möglichkeiten, Ideen und damit verbundene Schwierigkeiten mit sich. An dieser Stelle werden die Prinzipien vorgestellt, die notwendig sind um die Vorteile sowie die Nachteile zu verstehen und gegebenenfalls eine Umsetzung teilweise vorzunehmen.

Wie so oft in der Wissenschaft wird das Rad nicht immer neu erfunden, sondern viel mehr werden neue Technologien und Erkenntnisse miteinander verknüpft um aktuelle Lösungsansätze effizienter zu machen oder sie zu verbessern. Genau so verhält es sich bei der Suche nach dem Maximalen Fluss Problem, dass durch Quantum Computing beschleunigt werden soll. Hierbei wird der schon im vorherigen Abschnitt vorgestellte und ausführlich erläuterte Edmonds-Karp-Algorithmus dienen.

Anstelle der Breitensuche soll der sogenannte Grover-Suchalgorithmus oder kurz Grover verwendet werden. Doch worin liegt hier der Vorteil? Es soll kurz der fundamentale und weniger wissenschaftliche Unterschied zwischen klasischem Computing und Quanten Computing eingegangen werden.

Man kann sich einen beliebeigen Rechner mit x64 Architektur vorstellen mit 64 Bit-Registern und man kann sich einen belibigen n-Bit Quanten-Rechner vorstellen. In dem Register eines Rechner mit 64 Bit Architektur steht genau eine Zahl und auf diese Zahl soll fünf addiert werden. Für eine Addition wird genau ein Rechenzyklus benötigt. In dem Pendant des n-Bit Quanten-Rechner können 2^n Zahlen stehen - alle diese Zahlen stehen gleichzeitig im Quantenregister. Auf jede dieser Zahlen wird fünf addiert, die geschieht in ebenfalls genau einem Rechenzyklus. Je nachdem wie man das Ergebniss ausliest ist das Ergebnis eine dieser 2^n Zahlen. Diese Gleichzeitigkeit nicht zu verwechseln mit Parallelität ist eines der Grundlegenden Prinzipien des Quanten Computing.

Grover-Suche