Technische Universität München

Fakultät für Mathematik
Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

Visualisierung von Flussalgorithmen

`Ford-Fulkerson` Algorithmus
zur Lösung des Maximum Flow Problems
`Cycle Canceling` Algorithmus
zur Lösung des Minimum-Cost Flow Problems

Interdisziplinäres Projekt (IDP) von Quirin Fischer

Themensteller:	Prof. Dr. Peter Gritzmann
Betreuer:	M.Sc. Wolfgang F. Riedl
Abgabedatum:	30.09.2016

Hiermit erkläre ich, dass ich diese Arbeit selbstständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 30.09.2016

Quirin Fischer

Abstract

This interdisciplinary project encompasses the development of interactive applications to visualise two flow algorithms: The Ford-Fulkerson algorithm for solving the maximum-flow problem, and the cycle canceling algorithm for solving the minimum-cost flow problem. This report presents the algorithms, describes the layout of the applications, and explains the technical realisation.

The aim of the applications is to facilitate the comprehension of the algorithms by allowing the step-by-step execution on any desired instance of the problem, displaying the computational phases transparently, and graphically visualising the data structures.

The implementation is done as an application for web browsers by using an existing framework which uses modern web standards like HTML5, SVG, CSS, Javascript and D3.js. In particular, multiple important components of the implementation and their interactions are described in order to give a clear overview of the code base.

Zusammenfassung

Dieses interdisziplinäre Projekt umfasst die Entwicklung von interaktiven Anwendungen für die Visualisierung zweier Flussalgorithmen: Des Ford-Fulkerson Algorithmus zur Lösung des Maximum-Flow Problems, und des Cycle-Canceling Algorithmus zur Lösung des Minimum-Cost Flow Problems. Dieser Bericht stellt die Algorithmen vor, beschreibt den Aufbau der Applikationen, und erläutert die technische Umsetzung.

Das Ziel der Applikationen ist das Verständnis der Algorithmen zu erleichtern, indem eine schrittweise Ausführung an beliebigen Instanzen des Problems ermöglicht, die Berechnungsschritte sichtbar gemacht und Datenstrukturen anschaulich visualisiert werden.

Die Implementierung erfolgt als Webapplikation für den Browser mithilfe eines bestehenden Frameworks, das moderne Webstandards wie HTML5, SVG, CSS, Javascript und D3.js verwendet. Insbesondere werden verschiedene wichtigen Komponenten der Implementierung und deren Zusammenspiel behandelt, um einen klaren Überblick über den Code zu geben.

Inhaltsverzeichnis

Einleitung

Die Graphentheorie ist eine Teilgebiet der (diskreten) Mathematik mit einigen herausstellenden Merkmalen: Sie ist sehr anwendungsbezogen, da viele praktische Entscheidungsprobleme mithilfe von Graphen modelliert werde können. Sie hat weiterhin eine sehr enge Verbindung zur Algorithmik, wobei sowohl algorithmische Probleme mit bestimmten Graphen zusammenhängen, als auch Probleme der Graphentheorie mithilfe von Algorithmen gelöst werden, und daraus folgend auch zur Informatik. Aufgrund ihrer Anschaulichkeit und der Nützlichkeit der Konzepte ist die Graphentheorie einer der ersten Bereiche der Mathematik mit dem Studenten mathematischer, informatischer oder technischer Fachrichtungen in Berührung kommen.

Eine besonders charakteristischer Teil hierbei ist die Beschäftigung mit verschiedenen Graphalgorithmen. Das Verständnis der Abläufe, der Eigenschaften und Strukturen die der einzelne Algorithmus ausnutzt, und den entsprechenden Eigenschaften der resultierenden Lösungen gibt eine vielseitige Perspektive auf die behandelten Problemstellungen. Eine wichtiges Hilfsmittel für die Einarbeitung in neue Algorithmen sind dabei die ausführliche Durchführung des Algorithmus auf verschiedenen Beispielen und Betrachtung der Berechnungsschritte. Insbesondere die Visualisierung des Graphen mit verschiedenen Annotationen ist vielmals hilfreich um ein Gefühl für die Vorgänge zu bekommen.

Eine Klasse von Graphalgorithmen mit vielseitigen Anwendungen sind Flussprobleme. Abgeleitet von praktischen Fragestellungen zum Transport in verschiedenartigen Netzwerken, man denke an Rohrnetze, Straßennetze oder Datennetzwerke, behandeln sie Optimierungsprobleme wie eine Menge an Ressourcen von einem Start- zu einem Zielknoten gelangen kann. Je nach speziellen Anforderungen wird der Graph, der die Netzwerktopologie modelliert, mit verschiedenen Informationen annotiert. Typischerweise sind dies Kapazitäten (verschiedene Teile des Netzwerkes können nur bestimmte Mengen an Ressource transportieren) oder Kosten (die Benutzung mancher Teile des Netzwerks is anderen vorzuziehen).

Dieses Dokument beschreibt die Entwicklung einer interaktiven Anwendung zur Visualisierung zweier Flussalgorithmen - des Ford-Fulkerson Algorithmus für das Maximum-Flow Problem, und des Cycle Canceling Algorithmus für das Min-Cost Flow Problem. Die Anwendung hat das Ziel das Verständnis der Algorithmen zu erleichtern, indem sie einfach Schritt für Schritt auf verschiedene Instanzen des Problems ausgeführt werden können, mit transparenter Anzeige des Berechnungszustandes und anschaulicher Darstellung der Datenstrukturen.

Im erste Kapitel werden die zwei behandelten Probleme und Lösungsalgorithmen behandelt. Im zweiten Kapitel wird der Aufbau der Anwendung und ihre verschiedenen Elemente beschrieben. Das dritte Kapitel beschreibt die technische Umsetzung, mit zunächst einem Überblick über die verwendeten Basistechnologien und schließlich einer Behandlung der Implementierung interessanter Komponenten und deren Zusammenspiel.

Kapitel 1

Behandelte Flussalgorithmen

1.1 Flüsse in Netzwerken

Anschaulich entsprechen Flüsse in Netzwerken der Modellierung von Transportsystemen - Anwendungsfälle sind beispielsweise Wassernetze (wobei Kapazitäten den Rohrdurchmessern entsprechen, und die Menge an geflossenem Wasser dem Fluss), Straßensysteme (Kapazität: Spurbreite, Fluss: beförderte Fahrzeuge) oder Datennetze (Kapazität: Übertragungsgeschwindigkeit, Fluss: übermittelte Datenmenge).

Als Netzwerk bezeichnet man einen gerichteten Graphen $G = (V, E)$ annotiert mit einer Kapazitätsfunktion für Kanten $u : E \to \mathbb{R}^+$ (dem *upper limit*), Startknoten s (der *source*) und Zielknoten t (*target*). Insgesamt notiert man folglich ein Netzwerk als 4-Tupel $N = (G, u, s, t)$. [Wik16]

Ein Fluss in einem Netz ist definiert als eine Funktion $f : E \to \mathbb{R}^+$ (auch *flow*), die für jede Kante des Graphen einen Menge an Fluss bestimmt. Ein gültiger s-t-Fluss f in einem Netzwerk muss weiterhin zwei Bedingungen erfüllen:

Kapazitätskonformität Der Fluss muss auf allen Kanten unter deren Kapazitätsgrenze liegen:

$$\forall e \in E : f(e) \leq u(e)$$

Flusserhalt Der Fluss ist in allen Knoten abgesehen von s und t Konstant, es gelangt gleich viel Fluss zum Knoten, wie ihn verlässt. Bezeichnet man die Menge an eingehenden Kanten eines Knotens als $\delta^-(v) \stackrel{\text{def}}{=} \{e = (x, v) \in E \mid x \in V\}$ und die Menge an ausgehenden Kanten als $\delta^+(v) \stackrel{\text{def}}{=} \{e = (v, x) \in E \mid x \in V\}$, so entspricht dies der Bedingung

$$\forall v \in V \setminus \{s, t\} : \sum_{e \in \delta^-(v)} f(e) = \sum_{e \in \delta^+(v)} f(e)$$

Ein wichtiges Konzept für Flussalgorithmen ist die Konstruktion des zu einem Fluss gehörigen Residualnetzwerkes. Dieses verwendet die gleiche Knotenmenge, enthält allerdings eine neu konstruierte Kantenmenge und andere Kapazitäten. Intuitiv modelliert

das Residualnetzwerk welche Änderungen des Flusses möglich sind - es enthält jede Kanten der ursprünglichen Netzwerks, falls zusätzlicher Fluss auf der Kante möglich ist, mit angepasster Kapazität. Für Kanten, die bereits Fluss tragen, werden zusätzlich Rückkanten mit Kapazität entsprechend des Flusses auf der Kante eingefügt, da es möglich wäre, diesen Fluss wieder zurückzuleiten.

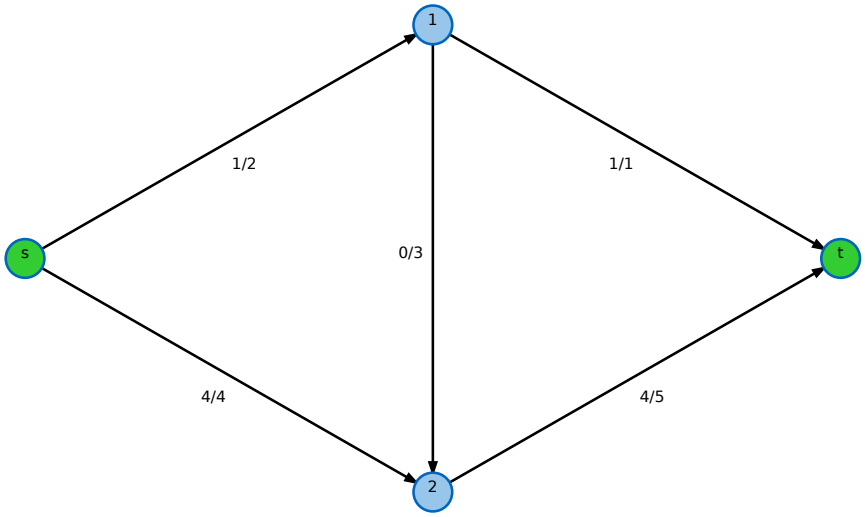

Abbildung 1.1: Ein Netzwerk mit Fluss.

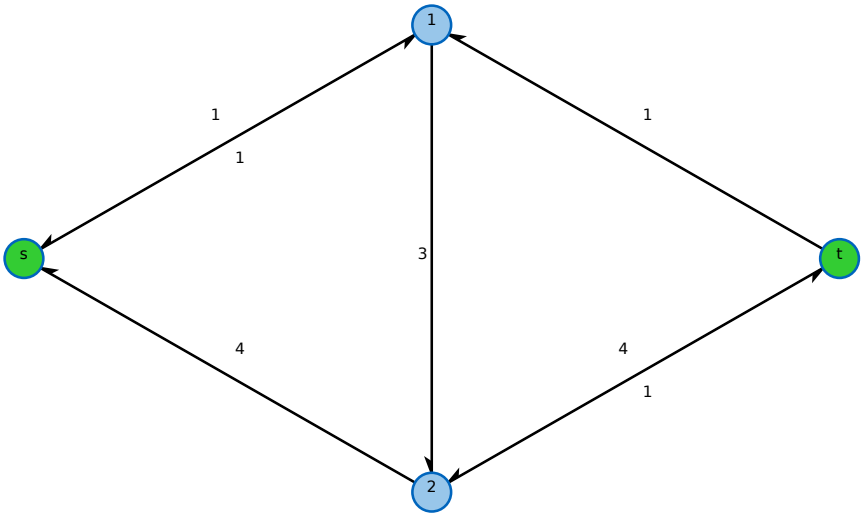

Abbildung 1.2: Das zugehörige Residualnetzwerk.

1.2 Ford-Fulkerson Algorithmus

Eine der naheliegenden Fragestellungen für ein gegebenes Netzwerk ist, wie viel Fluss gleichzeitig transportiert werden kann. Dies entspricht der Suche nach einem maximalen Fluss, das *maximum flow problem*. Zur Berechnung maximaler Flüsse gibt es eine Reihe verschiedener Methoden, die sich in der Komplexität der Berechnung und Effizienz unterscheiden. Ein einfacher Algorithmus, der gleichzeitig Ausgangspunkt für eine Reihe an fortgeschritteneren Methoden bietet, ist der*Ford-Fulkerson Algorithmus*. [FF56]

Der Ford-Fulkerson Algorithmus berechnet den maximal möglichen Fluss, indem ausgehend von einem gültigen Fluss (dem Null-Fluss) wiederholt zusätzlicher Fluss durch das Netzwerk geschickt wird, ohne die Bedingungen zu verletzen. Sobald keine weiteren Änderungen mehr möglich sind ist der maximale Fluss gefunden.

Um eine gültige Änderung zu finden wird zunächst das Residualnetz des aktuellen Flusses konstruiert. In diesem wird dann ein Pfad gesucht, der Start- und Zielknoten verbindet. Entlang dieses *augmentierenden Pfades* wird dann der Fluss um so viel erhöht, bis eine Kante saturiert und aus dem Residualnetz entfernt wird. Dies ist immer die Kante des Pfades mit der minimalen Residual-Kapazität. Diese Prozedur wird dann mit dem neuen Fluss wiederholt, bis kein Pfad mehr gefunden werden kann.

Die genaue Vorgehensweise für die Suche nach dem augmentierenden Pfad ist für den Ford-Fulkerson Algorithmus nicht exakt spezifiziert - sie hat allerdings starke Auswirkungen auf die Effizienz des Algorithmus. Naive Tiefensuche führt zu nicht-polynomieller Laufzeit $\mathcal{O}\left(|V| \cdot |E| \cdot u_{\max}\right)$, und auch das nur falls alle Kapazitäten ganzzahlig sind. Ein Breitensuche-Verfahren läuft dagegen schon in $\mathcal{O}(|V| \cdot |E|^2)$, diese spezielle Anwendung wird als *Edmonds-Karp-Algorithmus* bezeichnet. [EK72] Eine weitere Verbesserung stellt mit einer Laufzeit von $\mathcal{O}(|V|^2 \cdot |E|)$ der *Algorithmus von Dinic* dar, der nur kürzeste augmentierende Pfade verwendet. [Din70][Din06]

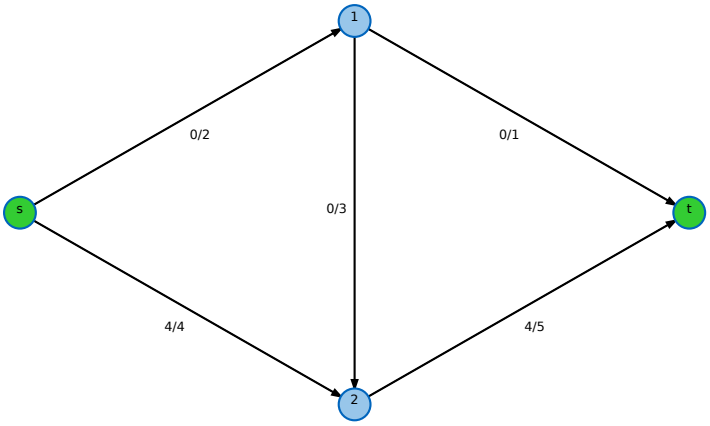

Abbildung 1.3: Ein initialer Fluss.

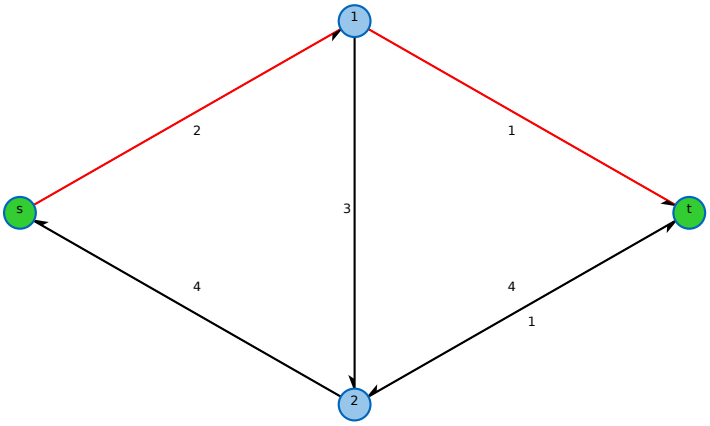

Abbildung 1.4: Ein augmentierender Pfad im Residualnetz.

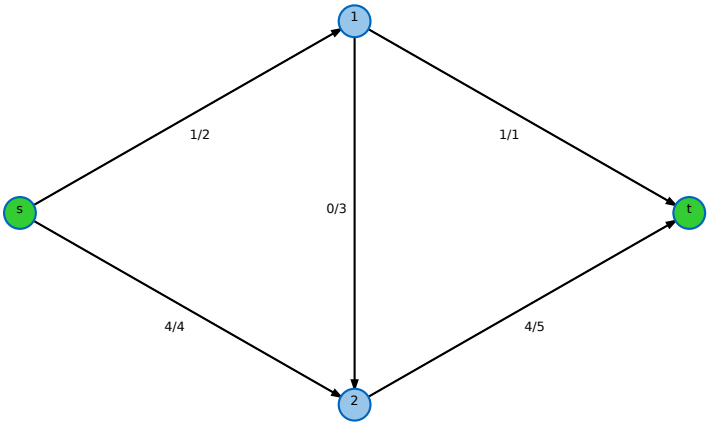

Abbildung 1.5: Der neue Fluss nach Modifikation.

1.3 Cycle Canceling Algorithmus

Eine neue Klasse von Problemen stellt sich, sobald das Netzwerk zusätzlich mit einer Kostenfunktion auf den Kanten annotiert wird. Dies ist eine Funktion $c : E \to \mathbb{R}$ die jeder Kante zuordnet, wie teuer es ist, eine Einheit Fluss über die Kante zu transportieren. Für einen Fluss ergeben sich dann die Gesamtkosten als

$$C = \sum_{e \in E} c(e) \cdot f(e)$$

Es ergibt sich die Frage nach der günstigsten Möglichkeit, eine bestimme Menge an Fluss durch das Netzwerk zu leiten - das *min-cost flow problem*. Eine übliche Variation ist zusätzlich den größtmöglichen Fluss (nach wie vor zu niedrigst möglichen Kosten) zu suchen, was man als *min-cost max flow problem* bezeichnet.

Ein relativ einfacher Algorithmus zur Berechnung von Flüssen mit minimalen Kosten ist der *Cycle Canceling Algorithmus.*[Kle67] Der Ansatz ist dem Ford-Fulkerson Algorithmus ähnlich, da auch hier eine eine vorläufige Lösung, ein gültiger Fluss der gesuchten Gesamtmenge, iterativ verbessert wird. Dazu werden Kreise im Residualnetz gesucht, die negative Gesamtkosten besitzen. Die Umleitung von Fluss entlang eines solchen Kreises verändert den Gesamtfluss nicht, senk allerdings die Kosten. Es kann so viel Fluss umgeleitet werden, bis die Kante des Kreises mit geringster Residualkapazität aus dem Residualnetz entfernt werden muss. Diese Ablauf kann wiederholt werden, bis keine negativen Kreise mehr im Residualnetz gefunden werden.

Zur Suche nach negativen Kreisen eignet sich der *Bellman-Ford Algorithmus*, ausgeführt auf dem Residualgraphen mit den Kantenkosten als Gewichtungen und ausgehend vom Zielknoten.

Der allgemeine Cycle Canceling Algorithmus führt zu einer nichtpolynomiellen Laufzeit in $\mathcal{O}(|V| \cdot |E| \cdot C_{max})$, da jede Iteration eine Ausführung des Bellman-Ford Algorithmus mit Laufzeit $\mathcal{O}(|V| \cdot |E|)$ erfordert und die Kosten um mindestens 1 verringert. Eine Abwandlung mit polynomieller Laufzeit stellt das *Minimum Mean Cycle Canceling Verfahren* dar.[GT89]

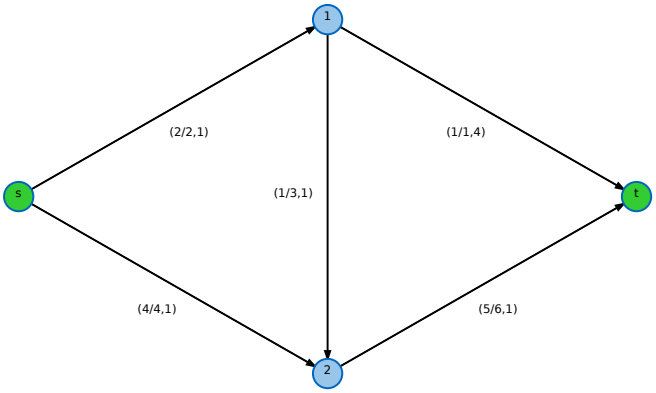

Abbildung 1.6: Ein initialer maximaler Fluss.

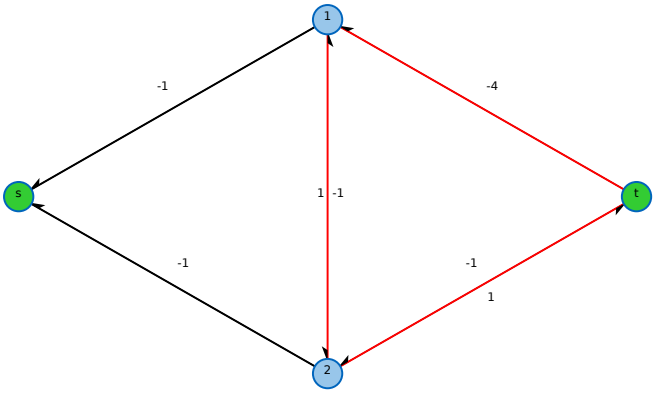

Abbildung 1.7: Das Residualnetz enthält einen negativen Kreis.

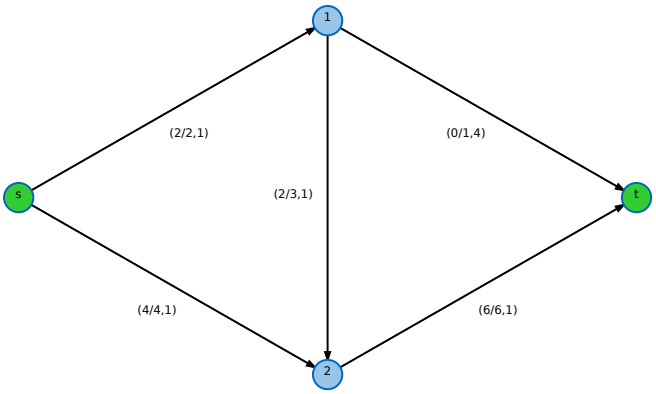

Abbildung 1.8: Fluss nach Verschiebung entlang des negativen Kreises.

Kapitel 2

Aufbau der Webapplikationen

2.1 Framework für Graphalgorithmen

Die Webapplikationen wurden in Analogie zu anderen, bereits bestehenden Applets für die Visualisierung verschiedener Graphalgorithmen konzeptioniert. Diese sind frei zugänglich auf der Webpräsenz des Lehrstuhls M5 für Mathematik an der TUM, erreichbar unter https://www-m9.ma.tum.de/Allgemeines/GraphAlgorithmen.

Die bestehende Sammlung an Applets enthält bereits verschiedene andere Algorithmen der Graphentheorie, von Lösungsmethoden für des kürzeste Wege, Matchings, Spannbäume und mehr. Alle Anwendungen sind mit den gleichen Zielen entwickelt, die Algorithmen möglichst anschaulich und verständlich zu präsentieren, und verwenden das gleiche Seitenlayout. Auch die hier vorgestellten Applets zur Darstellung des Ford-Fulkerson Algorithmus und des Cycle Canceling Algorithmus stellen die Informationen in diesem bewährten Layout dar. Die Inhalte sind auf mehrere Tabs verteilt, die verschiedene Materialien bieten.

Tab: Einführung Die erste Einführung für den Benutzer gibt einen kurzen Überblick über die Problemstellung, die mithilfe des Algorithmus gelöst wird.

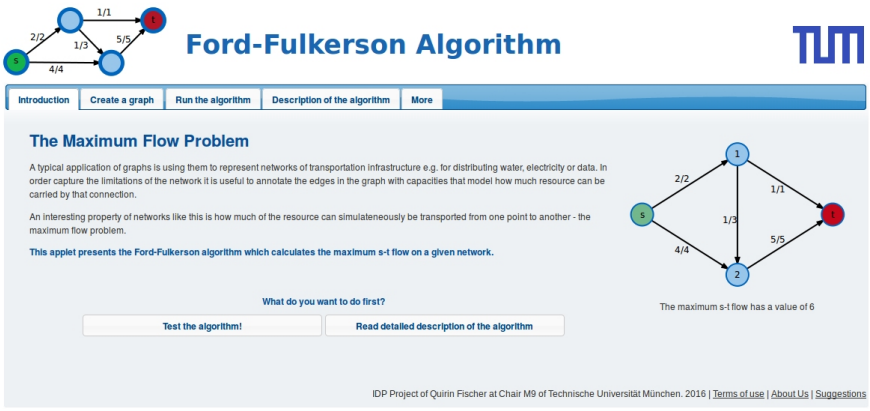

Abbildung 2.1: Der Einführungs-Tab.

Tab: Grapherstellung Ein Grapheditor, um die Instanz des Problems auszuwählen, auf der der Algorithmus ausgeführt werden soll. Der Editor bietet sowohl die Möglichkeit, vorgefertigte Beispiele zu laden, als auch diese zu verändern oder eigene Instanzen anzulegen. Knoten und Kanten des Graphen können einfach angelegt und gelöscht werden, und auf Klick ist auch die Modifikation zusätzlicher Eigenschaften wie Kantengewichte oder Kapazitäten möglich.

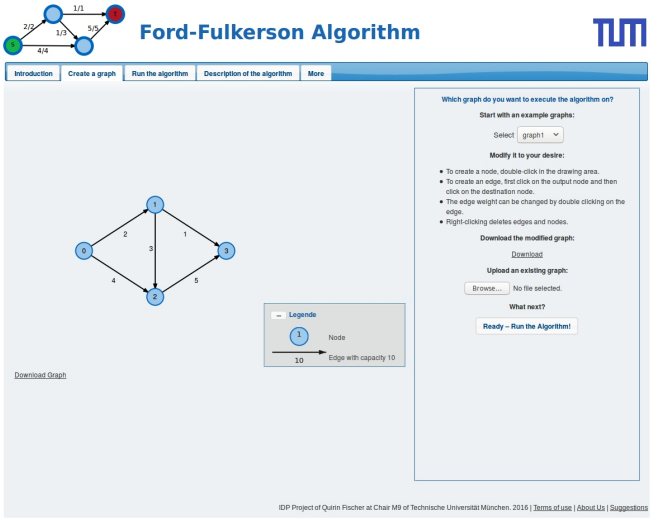

Abbildung 2.2: Der Tab des Grapheditors.

Tab: Ausführung Algorithmus Der wichtigste Tab erlaubt die schrittweise Ausführung des Algorithmus und zeigt die internen Datenstrukturen. Einige Informationen werden grafisch in die Darstellung des Graphen integriert, indem beispielsweise Knoten oder Kanten farblich hervorgehoben werden. Eine seitliche Anzeige liefert wahlweise eine textuelle Beschreibung des aktuellen Arbeitsschrittes, die aktuelle Ausführungsposition in einer Pseudocode-Diagramm, oder die Werte von Variablen die der Algorithmus verwendet.

Die darzustellenden Daten unterscheiden sich zwischen den Algorithmen, so dass die Applikationen sich in diesem Tab am deutlichsten unterscheiden.

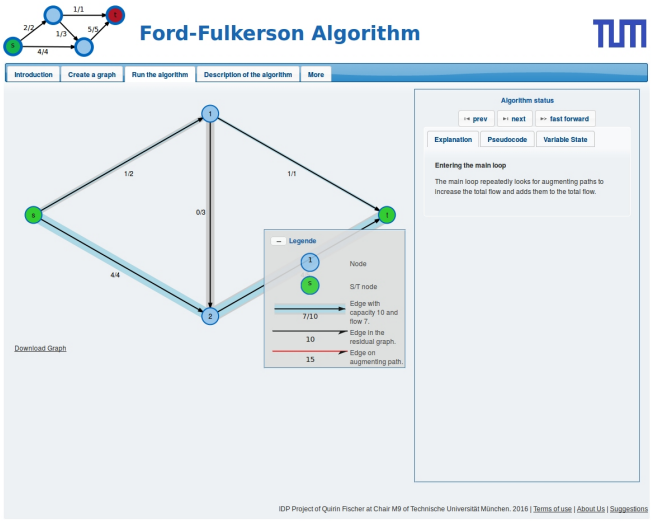

Abbildung 2.3: Der Algorithmus-Tab erlaubt schrittweise Ausführung und visualisiert den Status.

Tab: Beschreibung Algorithmus Eine ausführlichere Beschreibung erläutert die Problemstellung und den genauen Ablauf des Algorithmus.

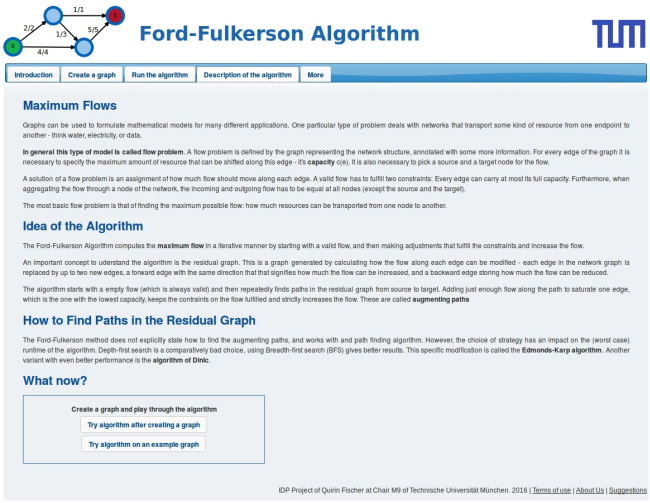

Abbildung 2.4: Eine längere Beschreibung des Algorithmus befindet sich im vierten Tab.

Tab: Weitere Informationen Eine Sammlung nützlicher Informationen wie Beispielsweise einer Pseudocode-Darstellung des Algorithmus, Informationen zur Effizienz des Algorithmus, Hinweise auf abgewandelte Varianten oder eine Auswahl an Literatur und Referenzen.

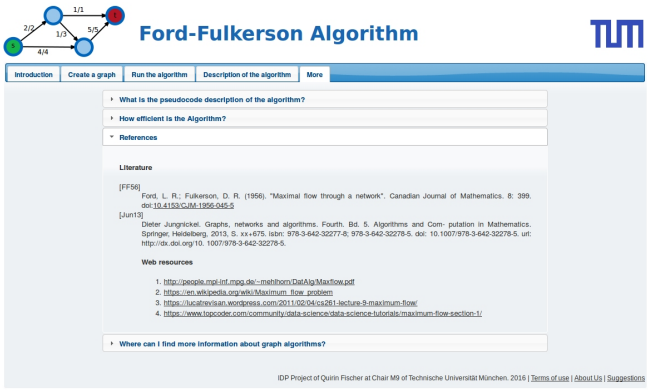

Abbildung 2.5: Der letzte Tab enthält weiteres Hintergrundmaterial.

2.2 Visualisierung des Ford-Fulkerson Algorithmus

Während der Ausführung des Ford-Fulkerson Algorithmus verändert sind vor allem der Inhalt dreier Datenstrukturen: der aktuelle Fluss, das zugehörige Residualnetz, und sofern vorhanden der aktuelle augmentierende Pfad. Der Algorithmus alterniert Änderungen an diesen Daten - zunächst ist ein Fluss vorhanden, ausgehend von diesem wird das Residualnetz bestimmt, die Pfadsuche ergibt einen augmentierenden Pfad, woraufhin bei einem neuen Fluss begonnen wird. Die Visualisierung reflektiert diesen Ablauf und wechselt zwischen drei verschiedenen Ansichten.

In der ersten Ansicht wird ein Fluss visualisiert, indem die Kapazitäten und Flüsse der Kanten durch farbige Umrandung in verschiedenen Stärken dargestellt werden. Ein grauer Farbton zeigt freie Kapazität an, und eine helleres Blau zeigt vorhandenen Fluss. Diese Farbkodierung signalisiert klar, über welche Kanten der Fluss verläuft und wo sich freie Kapazität befindet.

Die zweite Ansicht zeigt das Residualnetzwerk des aktuellen Flusses. Die Richtungen der verfügbaren Kanten (oder der zugehörigen Rückkanten) werden durch Pfeilspitzen gekennzeichnet, mit Angabe der verfügbaren Kapazität auf der entsprechenden Seite der Kante. In der dritten Ansicht wird in diesem Netz der augmentierende Pfad farblich stark hervorgehoben, indem die Kanten des Pfades rot eingefärbt werden.

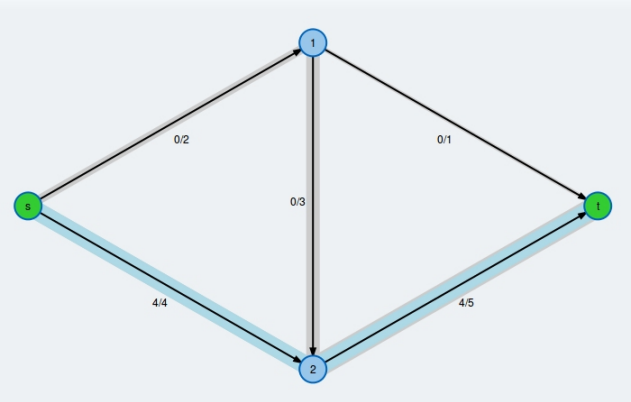

Abbildung 2.6: Flüsse und freie Kapazitäten werden farblich signalisiert.

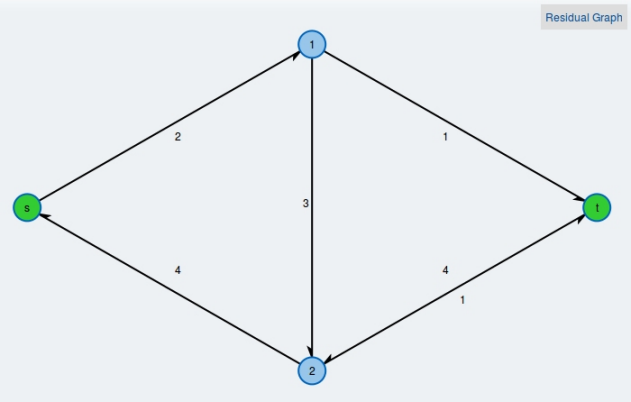

Abbildung 2.7: Das entsprechende Residualnetz.

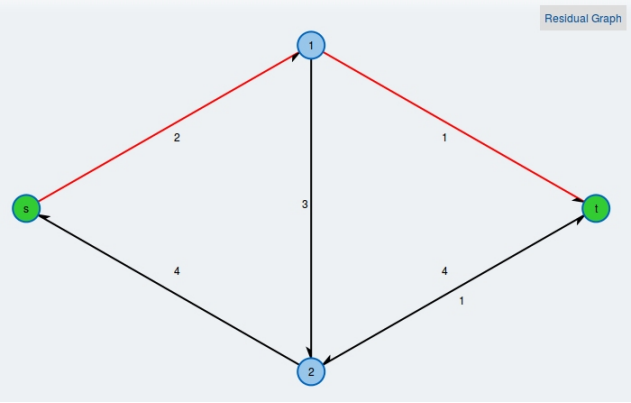

Abbildung 2.8: Der augmentierende Pfad wird rot gekennzeichnet.

2.3 Visualisierung des Cycle Canceling Algorithmus

Auch der Cycle Canceling Algorithmus alterniert zwischen der Anzeige des aktuellen Flusses, des Residualnetzwerks, und des Residualnetzes mit hervorgehobenem negativen Kreis. Im Unterschied zum Ford-Fulkerson Algorithmus werden allerdings bei der Anzeige des Flusses zusätzlich die Kosten der Kanten angezeigt. Im Residualnetz sind die Kanten mit ihren Kosten beschriftet und nicht mit den Kapazitäten, da diese bei der Kreiserkennung verwendet werden. Die Visualisierung verwendet das gleiche Farbschema, negative Kreise werden also ebenfall in rot hervorgehoben.

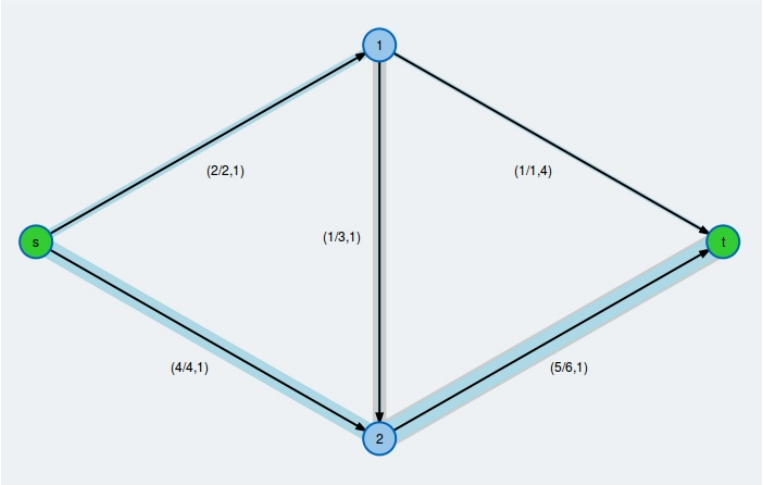

Abbildung 2.9: Flüsse und freie Kapazitäten werden farblich signalisiert, mit zusätzlicher anzeige der Kosten.

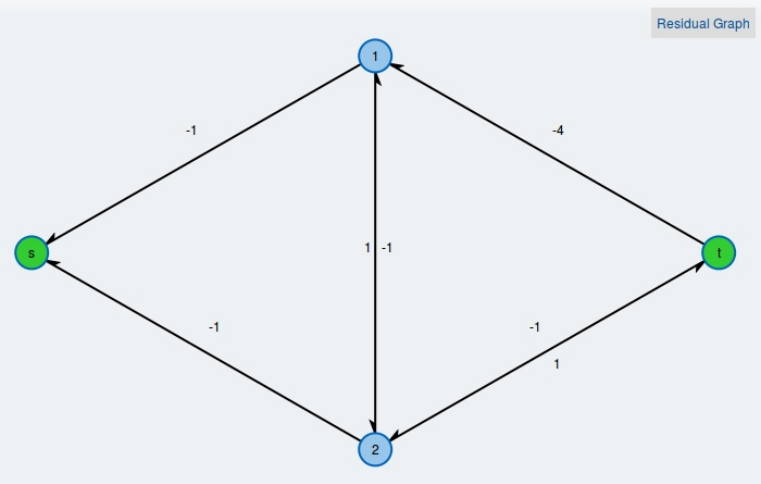

Abbildung 2.10: Im Residualnetz sind die Kanten mit ihren Kosten beschriftet.

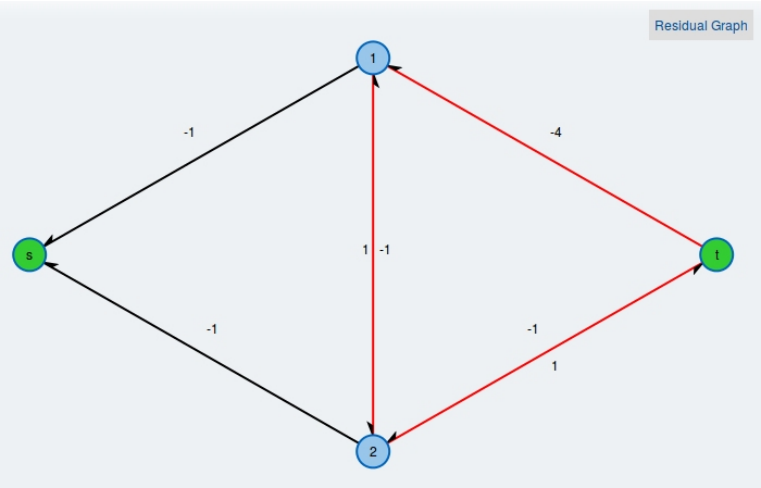

Abbildung 2.11: Ein erkannter negativer Kreis wird in rot markiert.

Kapitel 3

Implementierung

3.1 Technische Konzepte

Die technische Umsetzung der Webapplikationen erfolgt ausgehend von der bestehenden Code-Struktur ähnlicher Andwendungen für andere Algorithmen. Das Framework umfasst die generelle Seitenstruktur, die Tabverwaltung, ein Grapheditor und eine Grundgerüst für die schrittweise Ausführung eines Algorithmus. Implementiert als interaktive Website besteht die Codebasis aus einer Mischung von HTML, CSS und Javascript.

Im Folgenden werden zunächst die verwendeten Technologien und verwendete Muster vorgestellt. Da der bestehende Quellcode eine relative enge Vermengung von Framework-code mit Algorithmus-spezifischen Teilen erforderte, so dass im Anschluss einige funktionale Einheiten des Quellcodes vorgestellt werden.

HTML5 / CSS Das HTML Dokument stellt das statische Gerüst der Website dar. Es bindet die verschiedenen Skripte zur Kontrolle der interaktiven Elemente ein, enthält die statischen Textinhalte, und definiert Platzhalter für die Anzeige dynamischer Elemente. Der HTML Code befindet sich in den Dateien `max-flow/index_en.html` und `min-cost/index_en.html`. Einstellungen für die Darstellung verschiedener Webseiten-Elemente befinden sich in den CSS Dateien in `library/css` und `library-d3-svg/css`.

Javascript Die Interaktiven Elemente des Graph-Editors und des Algorithmus-Tabs sind mittels Javascript implementiert. Dies umfasst die folgenden Bereiche:

Verwaltung der dynamischen Tabs Bei Öffnung und Schließen der Tabs wird die Anzeige initialisiert, und optional wird der Benutzer drauf hingewiesen, dass bei Verlassen des Tabs der Zustand zurückgesetzt wird.

Entgegennahme von Benutzereingaben Für die Bearbeitung des Graphen und während der Auswahl von Start/Zielknoten müssen Einagbe-Events entgegengenommen werden.

Ausführung von Algorithmus-Schritten Berechnung der einzelnen Schritte des Algorithmus.

Graphdarstellung Für die Visualisierung muss der angezeigte Graph an die internen Daten angepasst werden.

SVG Die Darstellung des Graphen geschieht als Vektorgrafik. Die Javascript-Routinen modifizieren dafür zwei `<svg>` Elemente die als Anzeige für Editor und Algorithmus dienen, fügen die nötigen geometrischen Primitiven ein und setzen deren Rendering-Einstellungen. So wird beispielsweise für jeden Knoten des Graphen ein farbiger Kreis und ein Textelement mit der Beschriftung erzeugt; falls der Knoten als Start oder Ziel ausgewählt wird, so wird die Füllfarbe des Kreises geändert.

D3.js Für die dynamische Anpassung des Dokuments ist es nötig, das sogenannte *Document Object Model(DOM)* zu ändern. Diese Synchronisierung von komplexeren internen Daten (dem Graphen) mit einer Menge an Elementen des Dokuments erfolgt mithilfe der Javascript-Bibliothek *D3.js*[BOH11].

3.2 Code-Komponenten und deren Interaktion

3.2.1 Struktur der statischen Seite

Die verschiedenen Tabs sind innerhalb des Hauptelements mit der ID `tabs` definiert. Dieses befindet sich in `max-flow/index_en.html` (Z.145) bzw. `min-cost/index_en.html` (Z.144). Die Struktur sieht wie folgt aus:

Listing 3.1: Tabstruktur

```
<div id="tabs">
  <ul>
    <li><a href="#tab_te"><span>Introduction</span></a></li>
    <li><a href="#tab_tg"><span>Create a graph</span></a></li>
    ...
  </ul>
  <div id="tab_te">
    (Introduction content: ...)
  </div>
  <div id="tab_tg"> ... </div>
  ...
</div>
```

Innerhalb des Tab-Sektionen befinden sich die einzelnen Inhalte und Texte. Die dynamischen Tabs enthalten je einen Platzhalter für die erzeugten Vektorgraphiken, in `max-flow/index_en.html` (Z.184, Z.261) bzw. `min-cost/index_en.html` (Z.191,

Z.262). Für dynamische Tabs definiert das Framework noch einige Init/Exit Routinen, diese befinden sich u.A. in `library-d3-svg/js/Tab.js` und `max-flow/js/siteLayout.js` bzw. `min-cost/js/siteLayout.js`.

3.2.2 Verwaltung der Graphdaten

Die Graphdaten werden mithilfe des Modells definiert in `library-d3-svg/js/Graph.js` verwaltet. Das Datenmodell sorgt dafür, dass die Struktur sowohl für den Editor als auch den Algorithmus zugänglich ist. Knoten und Kanten sind per Index auswählbar und enthalten jeweils Adjazenzlisten. Jedes Element des Graphen kann eine Menge an Ressourcen (z.B. Kapazitäten) und Statusinformationen (z.B. Vorgänger-Informationen während der Pfadsuche) mitführen. Der Grapheditor ist in `library-d3-svg/js/GraphEditor.js` und `library-d3-svg/js/GraphEditorTab.js` implementiert. .

3.2.3 Schrittweise Ausführung des Algorithmus

Die Ausführung des Algorithmus mit den Möglichkeiten der schrittweisen Navigation erfordert eine klare Verwaltung der Zustandsänderungen im Algorithmus. Dies erfolgt mit der folgenden Struktur:

Der aktuelle Zustand ist in der Variable `state` `max-flow/js/FordFulkersonAlgorithm.js` (Z.326) bzw. `min-cost/js/CycleCancelingAlgorithm.js` (Z.305) gespeichert, mit zusätzlicher Statusinformation eingebettet in den Graphen. Beim Durchlaufen des Algorithmus werden diese Statusinformationen in eine Historie kopiert, um eine Rückwärtsnavigation zu erlauben.

Listing 3.2: Historie der Berechnungsschritte

```
this.addReplayStep = function(){
  replayHistory.push({
    "graphState": Graph.instance.getState(),
    "state": JSON.stringify(state),
    ...
  });
};

this.previousStepChoice = function() {
  var oldState = replayHistory.pop();
  Graph.instance.setState(oldState.graphState);
  state = JSON.parse(oldState.state);
  ...
  this.update();
};
```

Die Ausführungslogik des Algorithmus muss in einzelnen Teilfunktionen gegliedert werden, die je nach aktuellem Berechnungsfortschritt (gespeichert in `state.current_step`) die Berechnungen auf den Status anwendet.

Listing 3.3: Schrittweise Ausführung des Algorithmus

```
this.nextStepChoice = function(d) {
  // Speichere aktuellen Schritt im Stack
  this.addReplayStep();
  switch (state.current_step) {
    case STEP_SELECTSOURCE:
      this.selectSource(d);
      break;
    ...
    case STEP_FINDNEGATIVECYCLE:
      findNegativeCycle();
      break;
    case STEP_ADJUSTCYCLE:
      adjustCycle();
      break;
    ...
  }
  this.update();
};

function adjustCycle() {
  for(var i = 0; i < state.cycle.length; ++i)
  {
    var edge = Graph.instance.edges.get(state.cycle[i]["edge"]);

    edge.state.flow += state.cycle_min_flow * ↷
       ↪ state.cycle[i]["direction"];
  }
  state.cycle_min_flow = 0;
  state.cycle = [];
  state.show_residual_graph = false;
  state.current_step = STEP_MAINLOOP;
}
```

3.2.4 Darstellung der Graph-Visualisierung

Die Generierung der Vektorgrafik zur Anzeige des Graphen erfolgt größtenteils in `library-d3-svg/js/GraphDrawer.js`. Zusätzlich können die einzelnen Algorithmen noch Modifikationen durchführen, um zusätzliche Informationen anzuzeigen. So werden beispielsweise die Beschriftung und Farbe von Start- und Zielknoten verändert, oder die Kanten verschieden gestaltet um Fluss zu signalisieren.

Listing 3.4: Algorithmus-spezifische Visualisierung

```
this.onNodesUpdated = function(selection) {
  selection
    .selectAll("circle")
      .style("fill", function(d) {
        if (d.id == state.sourceId)
          return const_Colors.StartNodeColor; //green
        else if (d.id == state.targetId)
          return const_Colors.StartNodeColor; //green
        else
          return global_NodeLayout['fillStyle'];
      });
}

this.onEdgesEntered = function(selection) {
  selection.append("line")
    .attr("class", "cap")
    .style("stroke-width",
        function(d) {
          return algo.flowWidth(d.resources[0]);
        });

  selection.append("line")
    .attr("class", "flow");
}

this.onEdgesUpdated = function(selection) {
  selection.selectAll("line.flow")
    .style("stroke-width",
      function(d) {
      return ↷
        ↪ algo.flowWidth(Graph.instance.edges.get(d.id).state.flow); ↷
        ↪ });
  ...
}
```

3.2.5 Synchronisierung von Anzeige und Berechnungszustand

Um den aktuellen Status des Algorithmus sichtbar zu machen, befinden sich innerhalb des Algorithmus-Tabs drei weitere Tabs um eine Beschreibung des aktuellen Schritts, den aktuellen Schritt in Pseudocode, und die aktuellen Variablenwerte anzuzeigen. Die Beschreibungen und Pseudocode-Informationen sind im HTML Dokument gelistet und werden mithilfe von D3.js ein/ausgeblendet oder farbig markiert. Für die Variablenwerte wurden Platzhalter im HTML platziert, in die Werte eingetragen werden.

Listing 3.5: Tabs zur Anzeige des aktuellen Status des Algorithmus

```
<div id="ta_div_statusTabs">
  <ul>
    <li><a href="#ta_div_statusErklaerung">Explanation</a></li>
    <li><a href="#ta_div_statusPseudocode">Pseudocode</a></li>
    <li><a href="#ta_div_variables">Variable State</a></li>
  </ul>
  <div id="ta_div_statusErklaerung">
    <div id="explanation-select-source">
      <h3>First choose a source node</h3>
      <p>Click on a node to select it as the source/starting
         ↪ node s</p>
    </div>
    <div id="explanation-select-target">
      <h3>Then choose a target node</h3>
      <p>Click on a node to select it as the target/sink node
         ↪ t</p>
    </div>
    ...
   </div>
  <div class="PseudocodeWrapper" id="ta_div_statusPseudocode">
     ↪ ... </div>
  <div id="ta_div_statusVariables">
    <h3>Variable State</h3>
    <table class="algoInformationen">
      <tr>
        <th class="algoInfoTH"><span>cycle</span></th>
        <th class="algoInfoTH"><span>adjustment</span></th>
      </tr>
      <tr>
        <td id="variable-value-cycle" class="algoInfoTD">-</td>
        <td id="variable-value-adjustment"
           ↪ class="algoInfoTD">-</td>
      </tr>
    </table>
  </div>
</div>
```

Listing 3.6: Anzeige des aktuellen Status

```
this.updateDescriptionAndPseudocode = function() {
  var sel =
     ↪ d3.select("#ta_div_statusPseudocode").selectAll("div");
  sel.classed("marked", function(a, pInDivCounter, divCounter) {
    return d3.select(this).attr("id") ===
       ↪ "pseudocode-"+state.current_step;
```

```
  });
  var sel = ↷
    ↪ d3.select("#ta_div_statusErklaerung").selectAll("div");
  sel.style("display", function(a, divCounter) {
    return (d3.select(this).attr("id") === ↷
      ↪ "explanation-"+state.current_step) ? "block" : "none";
  });
  ...
};

this.updateVariables = function() {
  ...
  var path_string = "["+path_edge_strings.join(",")+"]";
  d3.select("#variable-value-path").text(path_string);
  d3.select("#variable-value-augmentation").text(state.augmentation);
}
```

Fazit

Zusammenfassung

In diesem Dokument wurden der Inhalt, die Gestaltung, und die technische Umsetzung zweier Webapplikationen zur Visualisierung von Flussalgorithmen vorgestellt.

Der Ford-Fulkerson Algorithmus ist anwendbar zur Berechnung maximaler Flüsse, und der Cycle Canceling Algorithmus ermöglicht die Bestimmung von Flüssen mit minimalen Kosten. Beide Algorithmen verfolgen eine iterative Vorgehensweise, indem ausgehend von einem gültigen Fluss in dessen Residualnetzwerk nach einem besonderen Kantenzug gesucht wird. Im Fall des Ford-Fulkerson Algorithmus ist dies ein s-t Pfad, im Cycle Canceling Algorithmus ein Kreis mit negativen Gesamtkosten. Anschließend wird der Fluss entlang des Kantenzugs modifiziert bis sich der Residualgraph ändert, und eine neue Iteration durchgeführt wird. Ein Iterationsschritt verbessert bei beiden Algorithmen die Bewertung des Ergebnisses, sobald keine Modifikation mehr gefunden wird, ist ein Optimum und Ergebnis erreicht.

Die Gestaltung der Applikationen erfolgt analog zu bestehenden Webapplets für andere Graphalgorithmen und ist auf mehrere Tabs gegliedert. Neben Informationsmaterial zur Erläuterung des Algorithmus sind ein Grapheditor enthalten, der die Auswahl beliebiger Probleminstanzen erlaubt. Im Haupttab kann der Algorithmus Schritt für Schritt auf dem gewählten Graph ausgeführt werden, wobei der Zustand der Graphen und der Programmvariablen grafisch dargestellt wird. Bei beiden Algorithmen entspricht dies der alternierenden Anzeige des Flusses, des Residualgraphen, und des besonderen Kantenzugs.

Die technische Umsetzung erfolgte ausgehend von einer bestehenden Codebasis als Webanwendung implementiert in HTML5/CSS/Javascript. Zur Ausgabe des Graphen wird eine Vektorgrafik im SVG Format erzeugt, für die Anpassung des Webseiteninhalts wird die Bibliothek D3.js verwendet. Die verschiedenen logischen Komponenten des Programms bestehen typischerweise aus einen bestimmten Bereich des HTML Dokuments, das von einer Javascript-Routine verändert wird.

Möglichkeiten zur Weiterentwicklung

Für weitere Arbeiten an den Webapplets gibt es eine Reihe an Ansatzpunkten:

Verbesserung der Visualisierung von Residualgraphen Diese Version stellt die Kanten des Residualgraphen nur mithilfe von Richtungsmarkierungen an den Ursprungskanten dar, die als gerade Linien gezeichnet werden. Eine mögliche Alternative wäre die Erzeugung von separaten, eventuell bogenförmigen Linien, um eine klarere Unterscheidung zwischen Kanten des Graphen und des Residualgraphen zu erzeugen.

Implementierung von Varianten Bei beiden implementierten Algorithmen existieren sehr geringfügige Modifikationen um bessere Laufzeiten zu erreichen. Eine naheliegende Erweiterung wäre, diese Modifikationen zusätzlich zu implementieren, und dem Nutzer die Auswahl zwischen verschiedenen Methoden zu geben. Eventuell wäre es sogar möglich, mehrere Varianten gleichzeitig auszuführen, und die entstehenden Unterschiede anzuzeigen.

Klarere Abgrenzung des Frameworks In der aktuellen Codebasis sind die Grenzen zwischen Aufgaben des Framework und der algorithmus-spezfischen Implementierung nicht besonders eindeutig. Hier wäre möglicherweise eine Umorganisation des Codes wünschenswert, die eine saubere Schnittstelle für die Inhalte des einzelnen Algorithmus bietet. Ein Designziel wäre beispielsweise, sämtliche spezifischen Inhalte (auch die statischen Texte) aus der HTML-Struktur in das Javascript des Algorithmus auszulagern. Mit einer zusätzlichen Auslagerung von Programmlogik, die für mehrere Algorithmen nützlich ist (wie z.B. die Historienverwaltung), sollte eine deutliche Entflechtung des Codes möglich sein, möglicherweise bis zu dem Punkt, dass ein Algorithmus vollständig in einem Javascript enthalten ist (zuzüglich vielleicht einiger Stildefinitionen in einer CSS Datei). Dies sollte nur die Beschreibungsmaterialien, die logische Ausführung des Algorithmus und die Darstellung des Status enthalten.

Abbildungsverzeichnis

Listings

Literatur

[BOH11] M. Bostock, V. Ogievetsky und J. Heer. „D3: Data-Driven Documents“. In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011).

[Din06] Y. Dinitz. „Theoretical Computer Science“. In: Hrsg. von O. Goldreich, A. L. Rosenberg und A. L. Selman. Berlin, Heidelberg: Springer-Verlag, 2006. Kap. Dinitz' Algorithm: The Original Version and Even's Version, S. 218–240. ISBN: 3-540-32880-7, 978-3-540-32880-3.

[Din70] E. A. Dinic. „Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation“. In: *Soviet Math Doklady* 11 (1970), S. 1277–1280.

[EK72] J. Edmonds und R. M. Karp. „Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems“. In: *J. ACM* 19.2 (Apr. 1972), S. 248–264. ISSN: 0004-5411.

[FF56] L. R. Ford und D. R. Fulkerson. „Maximal Flow through a Network.“ In: *Canadian Journal of Mathematics* 8 (1956), S. 399–404.

[GT89] A. V. Goldberg und R. E. Tarjan. „Finding Minimum-cost Circulations by Canceling Negative Cycles“. In: *J. ACM* 36.4 (Okt. 1989), S. 873–886. ISSN: 0004-5411.

[Kle67] M. Klein. „A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems“. In: *Management Science* 14.3 (1967), S. 205–220. eprint: `http://dx.doi.org/10.1287/mnsc.14.3.205`.

[Wik16] Wikipedia. *Flüsse und Schnitte in Netzwerken*. Online; accessed 29-September-2016. 2016.