Technische Universität München

Fakultät für Mathematik
Lehrstuhl für Angewandte Geometrie und Diskrete Mathematik

Adaption eines didaktischen Konzepts zur Darstellung weiterführender Graphalgorithmen in einer Web-Applikation

Algorithmus von Floyd-Warshall
Algorithmus von Hierholzer
Algorithmus von Hopcroft und Karp
Ungarische Methode
Chinese Postman Problem

Abschlussbericht für ein interdisziplinäres Projekt von

Mark-Johannes Becker
Aleksejs Voroncovs
Ruslan Zabrodin

Themensteller:	Prof. Dr. Peter Gritzmann
Betreuer:	M.Sc. Wolfgang Ferdinand Riedl
Abgabedatum:	15. Juni 2015

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

München, den 15. Juni 2015

Becker, Voroncovs, Zabrodin

Zusammenfassung

Das interdisziplinäre Projekt beschäftigt sich mit der anschaulichen Darstellung von weiterführenden Graphalgorithmen. Betrachtet werden realitätsnahe Probleme, wie das All-Pairs Shortest Path Problem, Matchingprobleme in bipartiten Graphen, das Eulertour Problem und das Chinese Postman Problem. Alle Problemstellungen haben gemeinsam, dass sich die zur Lösung verwendeten Algorithmen anschaulich darstellen lassen.

Das Ziel des Projekts ist, die erwähnten Problemstellungen mit einfachen Worten zu vermitteln, sowie die verwendeten Lösungsverfahren interaktiv zu veranschaulichen. Die Darstellung erfolgt in Form mehrerer Web-Applikationen, welche aus Kontinuitätsgründen auf ein gemeinsames Framework aufbauen, welches bereits bei früheren Projekten zum Einsatz kam. In den Applikationen wird der Benutzer zunächst an die jeweilige Problemstellung herangeführt. Anschließend hat er die Möglichkeit die Algorithmen auf selbst erstellen Graphen schrittweise ausführen zu lassen. Die speziellen Eigenheiten der Algorithmen werden weiterhin in gesonderten Forschungsaufgaben behandelt. ♠

Inhaltsverzeichnis

Kapitel 1

Einleitung

Graphen sind elementare Bestandteile Algorithmen zur Bestimmung von Lösungen vieler alltäglicher Probleme. Navigationsgeräte benutzen sie zur Berechnung der Entfernung zwischen zwei Orten und Logistikunternehmen benötigen sie zur Bestimmung optimaler Auslieferungsrouten. Viele komplexe Fragestellungen können mittels Mengen von Knoten und Kanten modelliert und so als allgemeine Probleme beschrieben werden. Durch diese Abstraktion lassen sich Probleme deutlich einfacher visualisieren und erläutern. Dies gilt auch für die zur Lösung benötigten teils komplexen und hochoptimierten Algorithmen. Mit Graphen ist es möglich, jeden Teilschritt eines Algorithmus gesondert darzustellen und im Detail zu beschreiben, so dass die Funktionsweise von Algorithmen einfacher nachzuvollziehen ist.

Dies ist auch Ziel dieses interdisziplinären Projekts: Wir behandeln mehrere komplexe Probleme, sowie die dafür benötigten Lösungsalgorithmen und stellen diese in Form von Webanwendungen anschaulich dar.

Zu den betrachteten Problemstellungen gehört das *All-Pairs Shortest Path Problem*, welches die Bestimmung der kürzesten Wege aller Knotenpaare in einem Graph verlangt. Der zu Lösung benötigte Algorithmus wurde von den namensgebenden Mathematikern Robert Floyd und Stephen Warshall entwickelt.

Der Algorithmus von Hierholzer löst das *Eulertourproblem*, welches das Finden einer Kantenfolge über alle Kanten des Graphs verlangt, ohne dass eine Kante mehrfach betrachtet wird.

Außerdem betrachten wir mehrere *Matchingprobleme*. So findet der Algorithmus von Hopcroft und Karp kardinalitätsmaximale Matchings und mittels der Ungarischen Methode lassen sich perfekte Matchings optimalen Gewichts bestimmen.

Abschließend betrachten wir das *Chinese Postman Problem*, welches zur Lösung eine Kombination der zuvor genannten Algorithmen benötigt. Es beschreibt das Problem eines Briefträgers, der alle Briefe auf einer möglichst kurzen Route austragen möchte.

Die zu implementierenden Anwendungen richten sich an Schüler und Studenten, die diese unterrichts- bzw. vorlesungsbegleitend verwenden können. Wir führen damit ein didaktisches Konzept aus früheren Projekten fort [Sto13; Vel14; Sef15]. Die Probleme und die Lösungsalgorithmen werden beispielhaft und möglichst anschaulich erklärt. Im Folgenden können die Nutzer die Algorithmen ausprobieren und den Ablauf dynamisch

regeln. Zum Abschluss der Übung werden besondere Aspekte der Probleme durch Forschungsaufgaben vertieft.

Grundlage unserer Implementierung ist ein Framework, welches im Zuge mehrerer Projekte entwickelt wurde. Es handelt sich um ein rein clientseitiges Web Framework zur Darstellung von Graphalgorithmen in modernen Webbrowsern. Es basiert auf HTML5 und modernen JavaScript Bibliotheken, sowie Cascading Stylesheets (CSS). Neben dem Framework selbst konnten wir für unsere Implementierung auch auf den Programmcode verschiedener bereits implementierter Algorithmen zurückgreifen.

In der vorliegenden Dokumentation beschreiben wir zunächst allgemein den Aufbau der Webanwendungen und stellen dann die Merkmale jedes implementierten Algorithmus vor. Der zweite Teil dient als technische Dokumentation der von uns implementierten Neuerungen für zukünftige Projekte. Die Disposition der Abschnitte erfolgte parallel zur Implementierung (Mark J. Becker ♢, Aleksejs Voroncovs ♡, Ruslan Zabrodin ♠). ♢

Kapitel 2

Aufbau der Webanwendungen

Jede Applikation ist in mehrere Teile aufgeteilt. Die Struktur jeder Applikation ist gleich. Alle Anwendungen bestehen aus mehreren Tabs, von denen jeder eine bestimmte Funktion hat (vgl. Abbildung 2.1).

Abbildung 2.1: Jede Anwendung ist in Tabs unterteilt. Um zwischen den Teilen der Anwendung zu navigieren, muss man auf den entsprechenden Tab im Bereich oben klicken.

Die Tabs „Einführung“, „Beschreibung des Algorithmus“ und „Weiteres“ enthalten nur den statischen Text mit Bildern und sind deswegen nicht interaktiv. Alle weiteren Tabs sind interaktiv. Sie bieten dem Nutzer Möglichkeiten an, die Funktionen der Algorithmen zu lernen und die Kentnisse des Nutzers zu prüfen. ♡

2.1 Aufbau der Tabs

Einführung: Das Ziel des Tabs ist, dem Nutzer eine allgemeine Vorstellung über die Anwendung zu geben. Der statische Text in diesem Tab enthält einige relevante Informationen zum Problem des jeweiligen Algorithmus, ein diesem Problem entsprechendes Bild sowie zwei Buttons, die zur Erstellung des Graphs und zur Beschreibung des Algorithmus führen. Außer diesen zwei Buttons ist der Tab mit keiner weiteren Interaktivität ausgestattet.
Graph erstellen: In diesem Tab kann der Nutzer einen Graph erstellen, auf dem der Algorithmus später ausgeführt wird. Hier stehen zwei Möglichkeiten zur Verfügung: entweder den Graph selbst mit der Maus zu erstellen oder einen fertigen Graph aus der vorgegebenen Liste zu wählen. Falls der Algorithmus einen gewichteten Graph voraussetzt, kann der Nutzer die Gewichte nach einem Doppelklick auf die Kante selbst eingeben. Wenn der Algorithmus einen bipartiten Graph erfordert, dann wird die Erstellung des Graph beschränkt, damit alle Eigenschaften des bipartiten Graphs erfüllt sind.

Algorithmus ausführen: Hier wird der Ablauf des Algorithmus dargestellt. Der Algorithmus läuft auf dem Graph, der im vorigen Tab erstellt wurde. Man kann sowohl schrittweise den Algorithmus ausführen als auch vorspulen. Außerdem kann der Nutzer den Schnellvorlauf pausieren, in diesem Fall wird der Ablauf im aktuellen Schritt gestoppt. Bei jedem Schritt wird die entsprechende Information zum Status des Algorithmus aufgezeigt und der Stand des Pseudocodes wird im separaten Bereich rechts abgebildet.

Beschreibung des Algorithmus: Dieser Tab stellt die ausführlichen Informationen zum jeweiligen Algorithmus dar. Der Tab enthält statischen Text mit Bildern, der die Logik des Algorithmus erklärt. Im unteren Teil des Tabs stehen die Buttons, die auf andere Tabs verweisen.

Forschungsaufgaben: Bei jeder Anwendung stehen zwei Tabs zur Verfügung, wo der Nutzer ausgewählte Aufgaben lösen muss. Der Zweck der Forschungsaufgaben ist zu prüfen, wie gut der Nutzer die Algorithmen verstanden hat. In den Aufgaben muss man entweder die Logik des entsprechenden Algorithmus reproduzieren oder beantworten, welche Entscheidungen der Algorithmus trifft. Am Ende der Aufgabe sieht man die Statistik, die die Antworten des Nutzers erfasst. Danach wird vorgeschlagen, zu weiteren Tabs der Applikation zu gehen.

Weiteres: Im letzten Tab kann der Nutzer weitere ausführliche Informationen zum Algorithmus lesen. Es handelt sich um den Pseudocode, die Laufzeit der Algorithmen, den Beweis der Korrektheit und die Links zu sonstigen Quellen. ♡

2.2 Algorithmus von Floyd-Warshall

Da die Implementierung des Floyd-Warshall Algorithmus nicht kompliziert ist, war es wichtig zu zeigen, was genau während des Ablaufs schrittweise passiert.

Der Algorithmus wurde so in Schritte unterteilt, dass jeder Schritt eine Verbesserung der Distanz zwischen zwei Knoten darstellt. Um die Verbesserungen anschaulich zu machen, wurde die Abstandsmatrix implementiert. In jeder Zelle der Abstandsmatrix wird der aktuelle Distanzwert zwischen zwei entsprechenden Knoten gespeichert. Wenn man die Maus über eine Zelle in der Matrix bewegt, dann wird der Pfad, der diesem Distanzwert entspricht, im Graphen markiert (vgl. Abbildung 2.2).

Die erste Forschungsaufgabe prüft, ob der Nutzer die Logik des Algorithmus verstanden hat. Während des Ablaufs von Floyd-Warshall auf dem vorgegebenen Graphen werden die Fragen über die vom Algorithmus getroffenen Entscheidungen gestellt.

In der zweiten Forschungsaufgabe soll der Nutzer die fehlende Kantengewichte bestimmen. Ein Teil der Kantenkosten sowie die Abstandsmatrix zum Ende des Algorithmus sind gegeben. Diese Information muss genutzt werden, um die fehlende Gewichte zu bestimmen. ♡

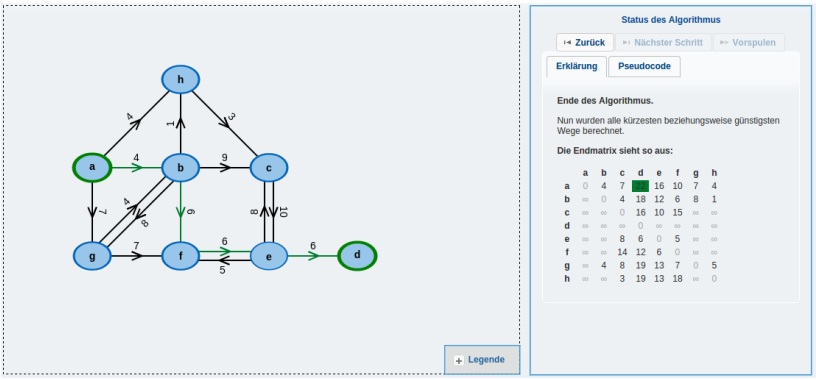

Abbildung 2.2: Nachdem man die Maus über eine Zelle in der Abstandsmatrix bewegt, werden die Kanten, Start- und Endknoten des aktuellen Pfads im Graph grün markiert.

2.3 Algorithmus von Hierholzer

Beim Algorithmus von Hierholzer war es uns wichtig, die resultierende Eulertour verständlich zu visualisieren. Dazu werden zunächst alle Subtouren in unterschiedlichen Farben dargestellt. Einzelne Subtouren können auf der Ergebnisseite einzeln hervorgehoben werden. Der Benutzer erkennt mittels der Farben außerdem, welcher Teil der Tour aus welcher Subtour stammt. Die Tour kann durch eine Animation (vgl. Abbildung 2.3) in der richtigen Reihenfolge abgelaufen werden, sodass der Nutzer verifizieren kann, dass es sich wirklich um eine Eulertour handelt.

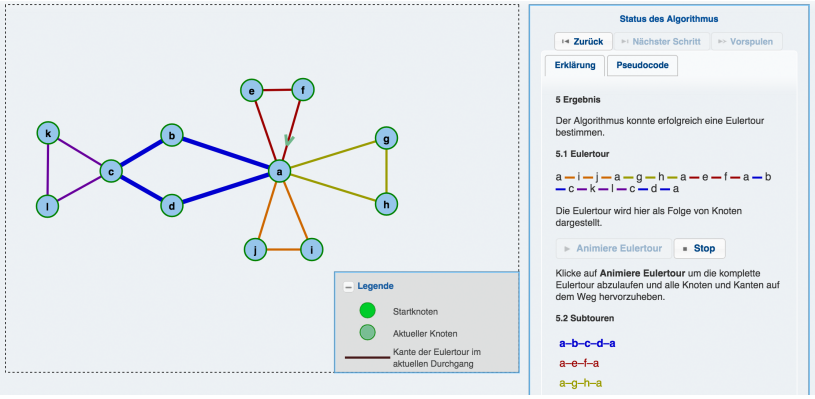

Abbildung 2.3: Visualisierung der vom Hierholzer Algorithmus ermittelten Eulertour. Auf der rechten Seite wird die gesamte Eulertour zusammen mit den Subtouren farbig dargestellt. Im Graph links können einzelne Subtouren hervorgehoben (blau) oder die gesamte Tour mittels Animation abgelaufen (grüner Pfeil) werden.

Die erste Forschungsaufgabe soll sicherstellen, dass der Nutzer die wichtigsten Abläufe des Algorithmus versteht. Sie behandelt daher im Wesentlichen den Ablauf des Algorithmus, das Konzept des Knotengrads und das Verbinden von Touren.

In einer weiteren Forschungsaufgabe werden die Besonderheiten des Algorithmus bei Anwendung auf einen gerichteten Graphen betrachtet. Der Nutzer beantwortet Fragen zu geänderten Voraussetzungen, die der Graph erfüllen muss und zur veränderten Auswahl der Kanten zur Konstruktion von Subtouren. ♢

2.4 Algorithmus von Hopcroft und Karp

Der Algorithmus wurde in Phasen unterteilt. In jeder Phase des Algorithmen wird nach einer inklusions-maximalen Menge von kürzesten disjunkten Augmentationswegen gesucht. Die in einer Phase gefundenen Augmentationswege werden nacheinander hervorgehoben und das Matching augmentiert. Das Aufzeigen der Augmentationswege stand im Fokus bei der Visualisierung des Algorithmus. Der Benutzer sollte diese leicht erkennen und nachvollziehen können.

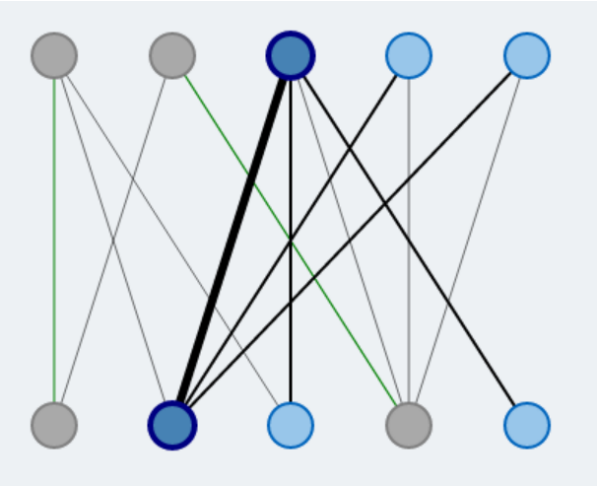

Abbildung 2.4: Aktueller Augmentationsweg im Hopcroft-Karp-Algorithmus. Die grauen Knoten wurden bereits in der aktuellen Iteration verwendet und dürfen deshalb nicht in anderen Augmentationswegen der aktuellen Iteration vorkommen. Die zwei hervorgehobenen Knoten und die hervorgehobene Kante dazwischen stellen den gefundenen knotendisjunkten kürzesten Augmentationsweg dar.

Um das Konzept von disjunkten Augmentationswegen zu verdeutlichen, werden die Knoten der bereits verarbeiteten Augmentationswegen grau markiert. Dadurch wird dem Benutzer verdeutlicht, dass ein Knoten in einer Phase nur auf einem Augmentationsweg vorkommt. Die graue Markierung der benutzten Knoten sollte außerdem

hervorheben, dass die Menge der disjunkten Augmentationswege inklusions-maximal ist. Der Benutzer kann am Ende einer Phase leicht nachvollziehen, dass es keinen weiteren disjunkten kürzesten Augmentationsweg existiert.

In der ersten Forschungsaufgabe soll der Nutzer sein Verständnis über den Ablauf des Algorithmus prüfen. Der Benutzer sollte das Konzept von kürzesten Augmentationswegen verstehen und wissen, welche Kanten nach einem Augmentationsschritt im Matching sein werden.

In der zweiten Forschungsaufgabe bekommt der Nutzer die Gelegenheit, selbst mit Augmentationswegen zu experimentieren. Der Algorithmus stoppt zufällig an einigen Stellen, wo der Benutzer einen Augmentationsweg selbst einzeichnen soll. Es müssen nicht die kürzesten Augmentationswege eingezeichnet werden. Der Benutzer sollte sehen, wie seine Wahl die weitere Ausführung des Algorithmus beeinflusst. ♠

2.5 Ungarische Methode

Obwohl es viele Implementierungen der Ungarischen Methode gibt, bleibt die Idee bei allen gleich. Die vorgestellte Applikation geht nicht tief in die Einzelheiten der konkreten Implementierung, sondern erklärt das generelle Prinzip der Ungarischen Methode. Dafür war der Einsatz des bipartiten gewichteten Graphs notwendig. Da die Ungarische Methode ein perfektes Matching im bipartiten Graphen sucht, wird ein gegebener Graph in der Anwendung bis zum kompletten Graphen vervollständigt, falls der Nutzer das nicht gemacht hat. Um die genutzten Begriffe (wie z. B. Matching oder Augmentationsweg) verständlich zu machen, werden während des Ablaufs unterschiedliche Farben für Knoten und Kanten verwendet (vgl. Abbildung 2.5).

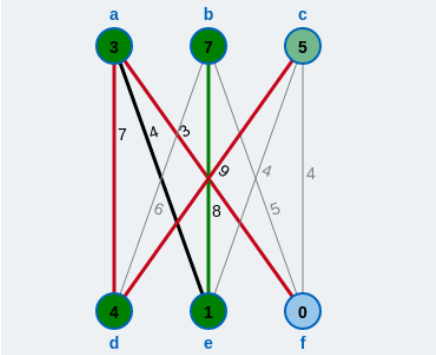

Abbildung 2.5: Um das Matching zu vergrößern, sucht die Ungarische Methode einen Augmentationsweg und markiert ihn rot. Der Augmentationsweg beginnt und endet in Knoten außerhalb des aktuellen Matchings.

Die erste Forschungsaufgabe prüft, wie gut der Nutzer den Aufbau des Algorithmus verstanden hat. Dazu werden während des Ablaufs auf einem beliebigen Graphen

Fragen gestellt. Der Nutzer muss beantworten, wie der Algorithmus die Markierungen zuweist und welche Schritte der Algorithmus als nächstes macht.

Bei der zweiten Forschungsaufgabe werden ebenfalls Fragen während des Ablaufs der Ungarischen Methode gestellt. Der Graph ist in diesem Fall vorgegeben und die Fragen befassen sich mit dem Aufbau des Augmentationswegs und Gleichheitsgraphen. ♡

2.6 Chinese-Postman-Algorithmus

Der implementierte Chinese-Postman-Algorithmus löst das gerichtete Chinese-Postman-Problem. Das Besondere bei dem Chinese-Postman-Algorithmus ist, dass er für die Ausführung weitere Algorithmen benötigt. Diese sind

- Floyd-Warshall-Algorithmus
- Ungarische Methode
- Hierholzer-Algorithmus

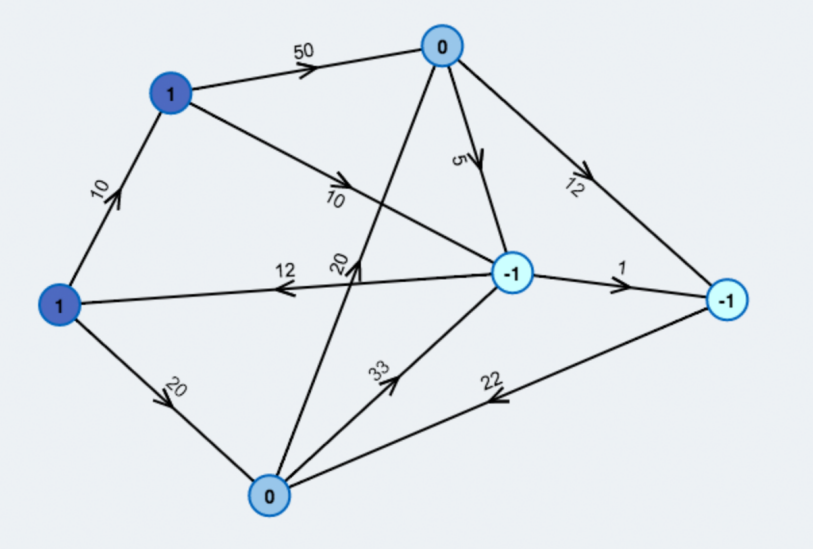

Abbildung 2.6: Hervorheben von Knoten mit ungleichem Ausgangs- und Eingangsgrad im Chinese-Postman-Algorithmus. Die Differenz zwischen dem Ausgangs- und dem Eingangsgrad eines Knotens wird in den Knoten eingetragen. In den Knoten mit negativer Differenz müssen zusätzliche Pfade starten und in den Knoten mit positiver Differenz enden, damit der Graph eulersch wird.

Der Algorithmus ist in verschiedene Schritte eingeteilt. Zuerst wird überprüft, ob das Problem auf dem Graphen lösbar ist. Anschließend wird eine Menge von Pfaden

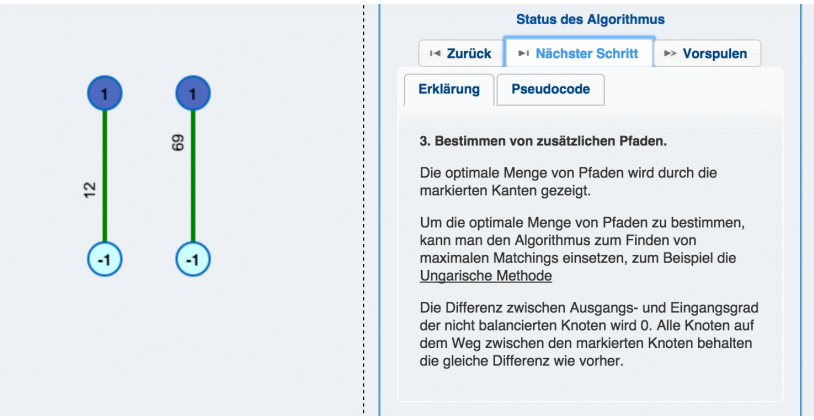

Abbildung 2.7: Matching-Graph im Chinese-Postman-Algorithmus. Hier wird bestimmt, zwischen welchen Knoten neue Wege eingefügt werden. Das Problem wird mit Ungarischer Methode gelöst. Im Matching-Graphen wird durch die grünen Kanten die optimale Zuordnung angezeigt.

gesucht, sodass nach dem Einfügen dieser Pfade in den Graphen der Graph eulersch wird. Die Summe der Längen dieser Pfade sollte minimal sein. Zum Schluss wird ein Eulerkreis im Graphen gefunden, welcher die Lösung des Problems darstellt.

Um die optimale Menge von zusätzlichen Pfaden zu bestimmen, werden zuerst Knoten bestimmt, bei denen die Anzahl der Eingangs- und Ausgangskanten nicht übereinstimmt. Abhängig davon, ob es zu viele Eingangs- oder Ausgangsknoten gibt, werden diese Knoten hell- bzw. dunkelfarbig markiert (vgl. Abbildung 2.6). Außerdem wird die Differenz zwischen Ausgangs- und Eingangsgrad eines Knoten im Knoten gezeigt. Dadurch entstehen zwei Knotenmengen (Knoten mit negativer und Knoten mit positiver Differenz).

Im nächsten Schritt wird ein bipartiter Graph erstellt, der aus den markierten Knoten besteht. Um den Wechsel von dem Ausgangsgraphen zu dem bipartiten Graphen nachvollziehbar zu gestalten, wurde eine Animation eingefügt. Alle für den Matching-Graphen relevanten Knoten werden in den bipartiten Matching-Graphen übergeführt. Das Aussehen des Matching-Graphen wurde aus der Ungarischen Methode übernommen.

In der Implementierung des Algorithmus bestimmt die Differenz zwischen Ausgangs- und Eingangsgrad eines Knotens, wie oft dieser im Matching-Graphen vorkommt. Es entsteht ein vollständiger bipartiter Graph, sodass mit der Ungarischen Methode ein optimales Matching bestimmt werden kann. Das Gewicht einer Kante ist die Länge des kürzesten Weges von dem Knoten mit negativer Differenz zwischen Ausgangs- und Eingangsgrad zu dem Knoten mit positiver Differenz. Bei der Visualisierung des Algorithmus wurde eine Entscheidung zugunsten Übersichtlichkeit getroffen, sodass

ein Knoten nur einmal im Matching-Graphen vorkommen sollte. Dadurch wird für den Nutzer nachvollziehbar, zwischen welchen Knoten neue Wege eingefügt werden müssen. Die Differenz zwischen Ausgangs- und Eingangsgrad eines Knotens bestimmt die Anzahl von Pfaden, die in diesem Knoten enden beziehungsweise anfangen müssen und somit die Anzahl von Matching-Kanten, die zu diesem Knoten inzident sind.

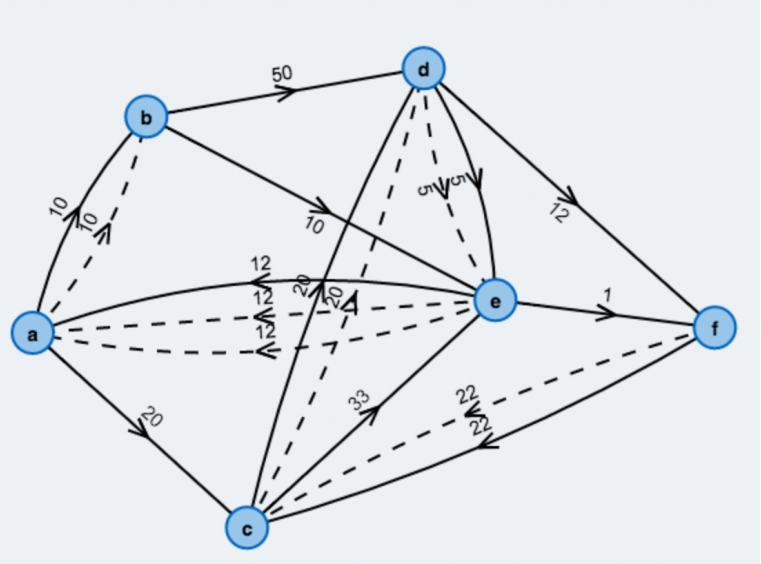

Abbildung 2.8: Graph nach Einfügen von zusätzlichen Pfaden im Chinese-Postman-Algorithmus. Die gestrichelten Kanten stellen neu eingefügte Kanten dar. Nach Einfügen der Pfade wird der Graph eulersch.

Nachdem die optimalen Pfade im Matching-Schritt bestimmt wurden, werden die Knoten des Matching-Graphen an ihren Platz in den normalen Graphen übergeführt. Allerdings werden die Matching-Kanten ebenfalls in den Graphen übernommen. Der Benutzer sieht somit auch im normalen Graphen, zwischen welchen Knoten neue Wege eingefügt werden müssen. Das Einfügen von neuen Pfaden wird ebenfalls mittels einer Animation aufgezeigt. Die aktuelle Matching-Kante wird markiert und die Kanten auf dem neuen Weg werden schrittweise in den Graphen eingefügt. Anschließend wird die Matching-Kante gelöscht.

Am Ende des Algorithmus wird die Eulertour des Graphen bestimmt. Der Nutzer kann die Animation der Tour selbst starten und stoppen. Die Visualisierung der Eulertour ist aus dem Hierholzer-Algorithmus übernommen.

Die erste Forschungsaufgabe sollte das Wissen des Benutzers prüfen. Der Benutzer sollte verstehen, welche Knoten im Matching-Graphen gebraucht werden und wie viele zusätzliche Wege in einem Knoten starten bzw. enden müssen.

Die zweite Forschungsaufgabe lässt dem Benutzer die Möglichkeit, den Rundweg selbst zu bestimmen, der alle Kanten des Graphen enthält. Anschließend wird die Länge dieses Weges mit der optimalen Lösung verglichen. Der Benutzer kann dadurch feststellen, inwiefern seine Lösung von der optimalen Lösung abweicht. ♠

Kapitel 3

Implementierung

Das folgende Kapitel dient als technische Dokumentation unserer Implementierung. Im ersten Teil erläutern wir die Schritte, die für die Installation der Anwendungen nötig sind. Im Weiteren stellen wir Neuheiten im Framework vor, die wir für die Implementierung der Algorithmen implementiert haben.

3.1 Installation

Zur Installation sind keine speziellen Anforderungen zu erfüllen. Die Webapplikationen wurden vollständig mittels HTML5, JavaScript und CSS implementiert, sodass keine zusätzliche serverseitige Software benötigt wird. Zum Aufrufen der Anwendungen wird ein moderner Webbrowser benötigt.

Die Bereitstellung des Codes erfolgt über den Online Dienst GitHub, der das verteile Versionskontrollsystem Git benutzt. Alle Anwendungen liegen in einem gemeinsamen Repository, welches unter `https://github.com/herzog31/adv-graph-algorithms` erreichbar ist. Zur Installation kann man entweder auf der Repository Seite die letzte freigegebene Version (Release) als Zip oder Tar Archiv herunterladen oder die aktuellste Version des Repositories mittels folgendem Befehl in das aktuelle Verzeichnis kopieren.

```
1 git clone -b master https://github.com/herzog31/adv-graph-↷
    ↪ algorithms.git
```

Abbildung 3.1: Befehl zum Kopieren des GitHub Repositories

Um die Anwendungen als Webanwendungen online bereitzustellen wird ein Webserver benötigt. Hier emfiehlt sich die Installation des Apache oder nginx HTTP Servers.

Die lokale Ausführung ist ohne einen Webserver möglich. Verschiedene Webbrowser besitzen allerdings eine Sicherheitsrichtlinie, die das Öffnen von lokalen Dateien über JavaScript verbieten. Diese Sicherheitsrichtline kann jedoch durch spezielle Einstellungen umgangen werden. Für Google Chrome ist dies über den Start Parameter `--allow-file-access-from-files` möglich. ♢

3.2 MathJAX

Unter den Tabs „Beschreibung des Algorithmus“ werden die komplizierten Algorithmen, wie beispielsweise die Ungarische Methode, in möglichst einfachen Worten im Fließtext erklärt. Wir gehen davon aus, dass der Nutzer während der Bearbeitung der Forschungsaufgaben verschiedene Sachverhalte in der Algorithmenbeschreibung nachschlagen muss. Das wiederholte Lesen von vollständigen Absätzen wollten wir allerdings vermeiden.

Dazu haben wir zusätzlich zur textuellen Beschreibung wichtige Punkte der Algorithmen auch als mathematische Formeln dargestellt. Der Nutzer erlangt so durch das erste Lesen der Beschreibung ein Grundverständnis der Algorithmen und kann sich während der Bearbeitung aufkommende Fragen durch die dargestellten Formeln schneller erschließen.

Zur Darstellung der Formeln in den Webapplikationen verwenden wir die JavaScript Bibliothek MathJAX[1]. Diese ist frei unter der Apache-Lizenz erhältlich und wird u.a. auch von Wikipedia zur Darstellung jeglicher mathematischer Ausdrücke genutzt. Mit der Bibliothek ist es möglich, mit LaTeX beschriebene Ausdrücke als SVG oder PNG im Webbrowser zu rendern.

Damit die Bibliothek in den Webanwendungen genutzt werden kann, muss sie wie folgt in den `<head>` Bereich des HTML Dokuments eingebunden werden (vgl. Abbildung 3.2).

```
1 <script type="text/javascript" src="../library/js/mathjax/↷
    ↪ MathJax.js?config=TeX-AMS-MML_SVG.js&locale=de"></script>
2 <script type="text/x-mathjax-config">
3   MathJax.Hub.Config({
4     showMathMenu: false,
5     showMathMenuMSIE: false
6   });
7 </script>
```

Abbildung 3.2: Einbinden der MathJAX Bibliothek

Nach der Einbindung werden sämtliche LaTeX Ausdrücke, welche sich zwischen den Begrenzungszeichen `\(` und `\)` befinden, automatisch übersetzt. In einem Beispiel wird hier der HTML Code in Abbildung 3.3 von MathJAX im Browser gerendert (vgl. Abbildung 3.4.

Werden Ausdrücke zur Laufzeit in das DOM mittels JavaScript eingefügt, so müssen diese separat übersetzt werden (vgl. Abbildung 3.5). ♢

[1] https://www.mathjax.org/

```
<p style="text-align: center;">
  \(\Delta = \min\limits_{s \in S\ \wedge\ y \in Y \setminus T
      ↪ }\{l(s) + l(y) - w(s,y)\}\)
</p>
```

Abbildung 3.3: MathJAX Beispiel: HTML Code

$$\Delta = \min_{s \in S\ \wedge\ y \in Y \setminus T} \{l(s) + l(y) - w(s,y)\}$$

Abbildung 3.4: MathJAX Beispiel: Darstellung im Browser

```
MathJax.Hub.Queue(["Typeset", MathJax.Hub]);
```

Abbildung 3.5: Befehl zum erneuten Rendern von Ausdrücken im HTML Dokument

3.3 Bipartite Graphen

Zwei Algorithmen, die im Rahmen des Projekts implementiert wurden, nämlich Hopcroft und Karp Algorithmus und die Ungarische Methode, müssen auf bipartiten Graphen ausgeführt werden. Dafür haben wir das Konzept des bipartiten Graphen im existierenden Framework entwickelt.

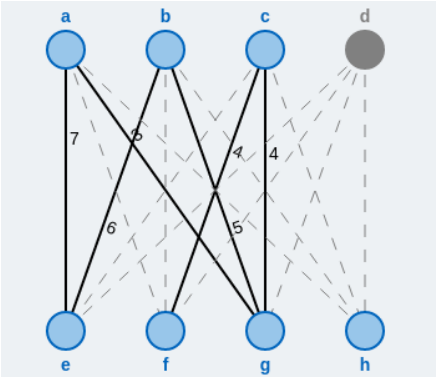

Abbildung 3.6: Ein bipartiter Graph stellt einen Graph mit zwei Partitionen (oben und unten) dar. Wenn der eingegebene Graph bei der Ungarischen Methode nicht vollständig ist dann werden zusätzliche graue Knoten und Kanten hinzugefügt.

Um die Darstellung des Matching-Problems und dessen Lösung möglichst anschaulich zu machen, haben wir die Knoten im Graphen in zwei Gruppen geteilt (vgl. Abbil-

dung 3.6). Die Kanten können nur zwischen den Knotengruppen erstellt werden. Bei der Ungarischen Methode war es wichtig, den Graph mit zusätzlichen Knoten und Kanten zu vervollständigen, wenn der ursprüngliche Graph nicht komplett war. Das ist notwendig, weil die Ungarische Methode ein perfektes Matching sucht.

Falls eine Kante zwischen beliebigen zwei Knoten fehlt, wird eine zusätzliche graue Kante mit dem Gewicht 0 hinzugefügt. Wenn die Anzahl der Knoten in den Partitionen ungleich ist, dann werden zusätzliche Knoten hinzugefügt. Alle Nachbarkanten bei diesen zusätzlichen Knoten haben das Gewicht 0. ♡

3.4 Multigraphen

In bisherigen auf dem Framework basierenden Algorithmen war es nicht möglich, Multikanten zu erstellen, was auch für die Algorithmen nicht notwendig war. Es war möglich, maximal zwei Kanten zwischen zwei Knoten einzufügen. In dem Chinese-Postman-Algorithmus werden jedoch während der Ausführung Kanten in den Graphen eingefügt. Deshalb muss das Graphenmodell für den Chinese-Postman-Algorithmus Multikanten darstellen können. Um dieses Problem zu lösen wurde das Framework erweitert, sodass es jetzt Multikanten darstellen kann. ♠

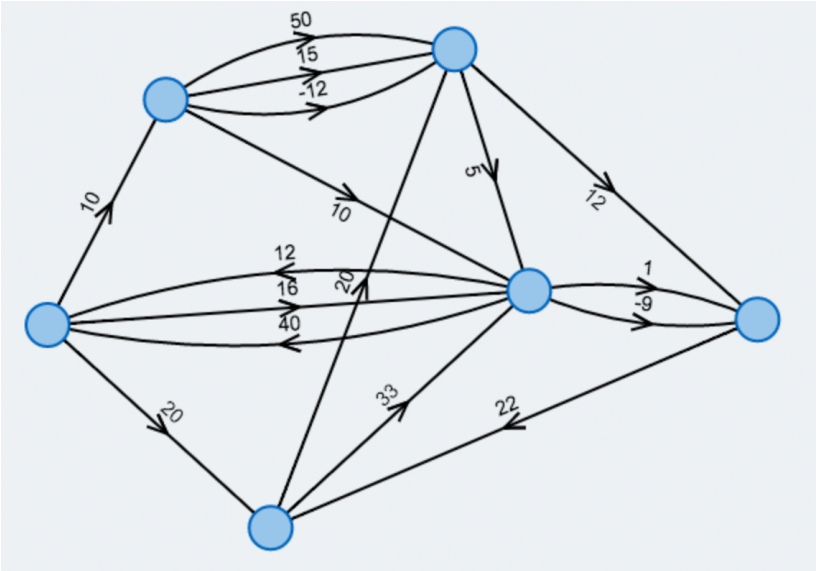

Abbildung 3.7: Multikanten im Graphen.

Es wurde eine Entscheidung getroffen, den Multikanten eine Krümmung zu geben, damit sie für den Nutzer übersichtlich dargestellt werden können. Die Abbildung 3.7 zeigt einen Graphen mit Multikanten. Die größte Herausforderung bestand darin, die

Krümmungen der Kanten zu bestimmen, sodass die Kanten zwischen zwei Knoten sich nicht überlappen. Außerdem sollte sichergestellt werden, dass genug Abstand zwischen zwei Multikanten besteht, damit die Kantengewichte gelesen werden können. Die Krümmung einer Kante sollte sich ändern, falls Kanten zwischen dem Quell- und Endknoten eingefügt beziehungsweise gelöscht werden. Um diese Probleme zu lösen wird jeder Multikante ihre eigene Krümmung zugeordnet, die dynamisch beim Zeichnen der Kante berechnet wird. Dies stellt sicher, dass sich die Kanten zwischen zwei Knoten nicht überlappen, die Kanten symmetrisch angeordnet werden und die Krümmung dynamisch angepasst werden kann.

3.5 Zufällig generierte Fragen

In bisherigen auf diesem Framework basierenden Projekten war die Varianz der Fragen in den Forschungsaufgaben problematisch. Die Forschungsaufgaben behandelten meistens statische Fragen an vorher festgelegten Stellen zu vorgegebenen Graphen. Daraus resultierend bietet sich dem Nutzer ein wenig abwechslungsreiches Erlebnis und die Motivation, eine Forschungsaufgabe mehrfach zu bearbeiten, ist nicht gegeben.

In unserer Implementierung verfolgen wir daher einen anderen Ansatz. Die Forschungsaufgaben sind grundsätzlich so aufgebaut, dass sie die gleiche Struktur und den gleichen Quellcode wie die reguläre Ausführung des Algorithmus benutzen. Das schließt auch den vom Nutzer selbst gewählten Graph aus dem „Graph erstellen“ Tab mit ein. Statt statischen Fragen definiert der Entwickler eine Menge von Fragetypen. Vor jedem Schritt des Algorithmus wird dann anhand einer Wahrscheinlichkeitsmatrix bestimmt, ob zu dem folgenden Schritt eine Frage gestellt wird und wenn ja, welcher Fragetyp gewählt werden soll. Außerdem sollen wenn möglich die Inhalte der generierten Fragen eines Fragetyps variieren. So werden beispielsweise die Knoten, zu denen ein Wert bestimmt werden soll, zufällig gewählt (vgl. Abbildung 3.8).

Mit dieser Implementierung gelingt es, dass in jeder Bearbeitung einer Forschungsaufgabe unterschiedliche Kombinationen von Fragen gestellt werden. Zusätzlich kann der Nutzer durch selbst erstelle Graphen auch Spezialfälle testen. ♢

Implementierung

Die beschriebene Funktionalität zur Generierung von Fragen wurde direkt in die Implementierung der Forschungsaufgaben in den Dateien `aufgabe1.js` und `aufgabe2.js` integriert. In der Funktion `nextStepChoice()` wird vor der Auswahl des nächsten Schritts des Algorithmus mittels der Funktion `askQuestion()` bestimmt, ob und welche Frage gestellt werden soll. Für jeden Fragetyp exisiert eine separate Funktion (bspw. `generateNextStepQuestion()`), die auf den aktuellen Graph zugreift und eine Frage mit zufälligen Elementen generiert und in das DOM Element `#tf1_div_questionModal`

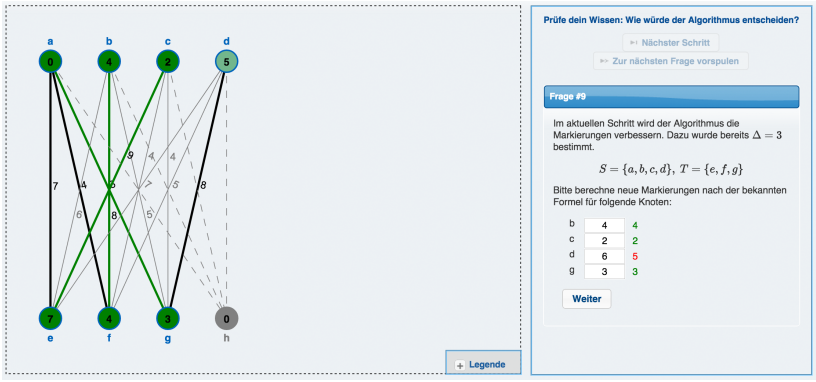

Abbildung 3.8: Beispiel für eine zufällig generierte Frage: Im aktuellen Schritt des Algorithmus wurde der Fragetyp „Markierungen berechnen“ bestimmt. Dazu wurde eine Frage generiert, die den Nutzer auffordert, Markierungen für vier zufällig gewählte Knoten zu bestimmen.

schreibt. Die gegebene Antwort wird durch Aufruf der Funktion `saveAnswer()` gespeichert und mit der richtigen Lösung verglichen. Am Ende der Forschungsaufgabe kann mittels `showQuestionResults()` eine Übersicht angezeigt werden, die zeigt, wieviele Fragen richtig beantwortet wurden. Weitere Informationen sind der Inline-Dokumentation zu entnehmen. ♢

3.6 Gemeinsam genutzte Dateien

Im Rahmen dieses interdisziplinären Projekts wurden Webapplikationen für insgesamt fünf Algorithmen entwickelt. Jede Applikation liegt in einem von den anderen Algorithmen unabhängigen Projektordner, der jeweils auch das Graph Framework aus früheren Projekten enthält. Daher lagen im gesamten Repository viele Dateien mehrfach vor. Dies war insofern problematisch, da Änderungen, die alle Anwendungen betreffen, in mehreren Dateien vorgenommen werden mussten.

Unser Ziel war es daher, gemeinsam genutzte Dateien auszulagern. Wir beschränken uns hier auf Dateien, welche das Layout und Design der Anwendungen definieren, sowie universelle Programmbibliotheken (bspw. MathJAX). Diese Beschränkung ist notwendig, um die Flexibilität im Entwicklungsprozess zu gewährleisten. Es existieren weitere Dateien, die sich zum aktuellen Zeitpunkt anwendungsübergreifend nicht signifikant unterscheiden. Diese sind allerdings algorithmenspezifisch und könnten in zukünftigen Entwicklungsphasen weiter verändert werden.

Zur Auslagerung legten wir zunächst einen `library` Ordner an. Das Finden von Dateiduplikaten übernimmt ein Python Skript (vgl. `deduplicate.py`), welches mittels der `difflib` Bibliothek alle Dateien in den Projektordnern paarweise miteinander

vergleicht und einen Ähnlichkeitsquotienten bestimmt. Das Skript zeigt dann alle Dateien, die einen festgelegten Ähnlichkeitsgrenzwert überschreiten (vgl. Abbildungen 3.9 und 3.10). Anhand des Quotienten lässt sich ablesen, ob Dateien im Gesamten oder in Teilen ausgelagert werden können. Das Verschieben in den `library` Ordner und die Anpassung der Pfade in den Anwendungen erfolgt dann manuell.

```
chinese-postman/img/TUMLogo.png
-> 100.00% identical to floyd-warshall/img/TUMLogo.png
-> 100.00% identical to hierholzer/img/TUMLogo.png
-> 100.00% identical to hopcroft-karp/img/TUMLogo.png
-> 100.00% identical to hungarian/img/TUMLogo.png
```

Abbildung 3.9: Vollständige Übereinstimmung bei der Datei `TUMLogo.png`

```
chinese-postman/js/siteAnimation.js
-> 67.01% identical to hierholzer/js/siteAnimation.js
-> 91.49% identical to hopcroft-karp/js/siteAnimation.js
-> 71.13% identical to hungarian/js/siteAnimation.js
```

Abbildung 3.10: Mäßige Übereinstimmungen bei der Datei `siteAnimation.js`

Für zukünftige Projekte können die Parameter des Skripts angepasst werden. Weitere Informationen hierzu sind der Inline-Dokumentation zu entnehmen. ♢

Kapitel 4

Zusammenfassung

Das interdisziplinäre Projekt beschäftigt sich mit der anschaulichen Darstellung von weiterführenden Graphalgorithmen. Es wurden folgende Probleme behandelt: All-Pairs Shortest Path Problem, Matchingprobleme in bipartiten Graphen, das Eulertour Problem und das Chinese-Postman-Problem. Die Problemstellungen wurden mit einfachen Worten erklärt und die verwendeten Lösungsverfahren interaktiv veranschaulicht. Die Darstellung der Algorithmen erfolgt in Form mehrerer Web-Applikationen, die auf ein gemeinsames Framework aufbauen, welches bereits bei früheren Projekten zum Einsatz kam. In den Applikationen wird der Benutzer zunächst an die jeweilige Problemstellung herangeführt. Anschließend erhält er die Möglichkeit, die Algorithmen auf selbst erstellten Graphen schrittweise ausführen zu lassen. Außerdem wurden speziellen Eigenschaften der Algorithmen in gesonderten Forschungsaufgaben behandelt.

Im Rahmen des Projekts wurden mehrere Änderungen und Weiterentwicklungen an dem Framework durchgeführt. Für Matchingprobleme auf bipartiten Graphen wurde ein bipartites Graphenmodell und die zugehörige Visualisierung entwickelt. Für den Chinese-Postman-Algorithmus wurde ein Graphenmodell entwickelt, das Multikanten unterstützt. Die MathJax-Bibliothek ermöglicht es, in der Beschreibung der Algorithmen mathematische Formeln einzusetzen. Dadurch können wichtige Aspekte der Algorithmen als mathematische Formeln dargestellt werden, sodass das Grundverständnis der Algorithmen dem Benutzer besser vermittelt wird. Um die Motivation, eine Forschungsaufgabe mehrfach zu bearbeiten, zu erhöhen, wurden dynamische Fragen erstellt. Die Anzahl der Fragen hängt von dem Graphen ab und der Fragetyp wird während der Abarbeitung zufällig ausgewählt. Da das vorliegende interdisziplinäre Projekt eine Gruppenarbeit darstellt, war es möglich identische und gemeinsam genutzte Dateien auszulagern. Dadurch existieren sie nur einmal, was eventuelle Änderungen an dem Projekt vereinfacht. ♠

Abbildungsverzeichnis

Literatur

[Sef15] S. R. Sefidgar. „Enhancement and Adaption of a didactic Concept to the Presentation of Spanning Tree Algorithms in a Web Application“. Interdisziplinäres Projekt. Technische Universität München, 2015.

[Sto13] R. Stotz. „Entwicklung und Implementierung eines didaktischen Konzepts zur Veranschaulichung verschiedener Graphalgorithmen zum Einsatz in der gymnasialen Oberstufe“. Interdisziplinäres Projekt. Technische Universität München, 2013.

[Vel14] L. Velden. „Entwicklung und Implementierung eines didaktischen Konzepts für die Wissenskontrolle zu verschiedenen Graphenalgorithmen zum Einsatz in der gymnasialen Oberstufe. Dijkstra- und A*-Algorithmus“. Interdisziplinäres Projekt. Technische Universität München, 2014.