



Fachhochschule  
Nordwestschweiz

# Link-Prediction Plugin für Gephi

Semesterarbeit IP5

M. Romanutti, S. Schüler

August 16, 2019

Betreuer: Michael Henninger  
Hochschule für Technik, FHNW Brugg

---

## Summary

Gephi ist eine Open-Source Software zur explorativen Analyse von Graphen. Die Software wird unter anderem im Modul Social Network Analysis (SNA), welches an der Fachhochschule Nordwestschweiz unterrichtet wird, eingesetzt. Gephi ist modular aufgebaut und kann um eigene Plugins erweitert werden. Um neue Kanten in einem bestehenden Netzwerk vorhersagen zu können, soll nun ein solches Plugin entwickelt werden. Die Vorhersage neuer Kanten (sog. "Link Prediction") kann mittels verschiedener Algorithmen berechnet werden. In einer ersten Version wurden die Algorithmen "Common Neighbours" und "Preferential Attachment" implementiert. Der Aufbau des Plugins soll es erlauben, einfach neue Algorithmen hinzuzufügen. Nebst der Vorhersage neuer Kanten soll die Qualität der eingesetzten Link-Prediction-Algorithmen beurteilt werden können. Als Messgröße wird hierfür die Accuracy verwendet, welche den prozentualen Anteil richtig vorausgesagter Kanten berechnet.

Um die Verhaltensweise und Qualität der Algorithmen vergleichen zu können, wurden diese auf unterschiedliche Datensets angewandt. Die Gegenüberstellung zeigt, dass sich die beiden Algorithmen mehrheitlich ähnlich verhalten: Bei den von uns verwendeten Datensets sind die Vorhersagen da am genauesten, wo die Knoten im Graphen untereinander am stärksten verknüpft sind. Ob effektiv eine Korrelation vorliegt, müsste mit weiteren Netzwerken und grösseren Datenmengen überprüft werden.

Die Durchführbarkeit und Korrektheit der Konzepte für das Plugin wurde mittels einem Proof of Concept geprüft. Die daraus gewonnenen Erkenntnisse wurden in die anschliessende Umsetzung übernommen. Die erforderlichen Funktionalitäten des Plugins konnten umgesetzt werden. Das Plugin kann weiter verbessert werden, indem gerichtete und auch gewichtete Graphen unterstützt werden. Aktuell kann ausserdem nur für Netzwerke mit bis zu 1000 Knoten eine schnelle Laufzeit garantiert werden. Um auch bei grösseren Netzwerken eine schnellere Performance zu erreichen, müssten weitere Optimierungsmassnahmen in Betracht gezogen werden.

---

# Inhalt

---

<b>Inhalt</b>	<b>ii</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Ausgangslage . . . . .	1
1.2 Zielsetzung . . . . .	1
1.3 Verwendete Software und Lizenzen . . . . .	2
<b>2 Theoretische Grundlagen</b>	<b>3</b>
2.1 Begriffsdefinitionen . . . . .	3
2.1.1 Social Network Analysis . . . . .	3
2.1.2 Graph . . . . .	3
2.1.3 Link Prediction . . . . .	4
2.1.4 Social Forces . . . . .	5
<b>3 Daten</b>	<b>8</b>
3.1 Anforderungen . . . . .	8
3.2 Datenaufbereitung . . . . .	8
3.3 Charakteristiken der Netzwerke . . . . .	9
<b>4 Link Prediction</b>	<b>12</b>
4.1 Algorithmen . . . . .	12
4.1.1 Common Neighbours . . . . .	12
4.1.2 Preferential Attachment . . . . .	14
4.2 Evaluation . . . . .	16
4.3 Gegenüberstellung der verschiedenen Algorithmen und Netzwerke . . . . .	17
<b>5 Plugin</b>	<b>20</b>
5.1 Software Architektur . . . . .	20
5.1.1 Klassenstruktur . . . . .	21
5.1.2 Loggingkonzept . . . . .	23

5.1.3	Exception-Handling . . . . .	26
5.2	GUI Konzept . . . . .	26
5.2.1	Vorhersagen neuer Kanten . . . . .	27
5.2.2	Filtern vorhergesagter Kanten . . . . .	30
5.2.3	Beurteilung der Qualität verschiedener Algorithmen .	31
5.3	Anleitung Software . . . . .	32
<b>6</b>	<b>Diskussion und Ausblick</b>	<b>34</b>
6.1	Erreichtes . . . . .	34
6.2	Ausblick . . . . .	35
<b>Abbildungsverzeichnis</b>		<b>36</b>
<b>Literaturverzeichnis</b>		<b>37</b>
<b>Abkürzungsverzeichnis</b>		<b>38</b>
<b>Glossar</b>		<b>39</b>
<b>A</b>	<b>Anhang Software</b>	<b>41</b>
A.1	Proof of Concept . . . . .	41
A.2	Wireframes . . . . .	43
A.3	README.md . . . . .	46
<b>B</b>	<b>Weiteres</b>	<b>49</b>
B.1	Projektanforderungen . . . . .	49

## Kapitel 1

---

# Einleitung

---

### 1.1 Ausgangslage

Dieses Dokument dient der Beschreibung des Projekts *Link-Prediction Plugin für Gephi (I4DS01)*. Gephi ist eine Open-Source Software zur explorativen Analyse von Graphen. Die Software ist in der Programmiersprache Java implementiert und modular aufgebaut. Durch den modularen Aufbau können individuelle Plugins entwickelt werden. Gephi bietet eine Architektur, welche es erlaubt, die bestehende Funktionalität einfach um solche eigenen Plugins zu erweitern.

Eine Liste der aktuell verfügbaren Plugins für Gephi ist einsehbar unter [gephi.org/plugins](http://gephi.org/plugins).

### 1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, ein Link-Prediction-Plugin für Gephi zu entwickeln. Unter dem Begriff "Link-Prediction" wird die Vorhersage der nächsten Kanten in einem bestehenden Netzwerk aus mehreren Knoten verstanden. Das Plugin soll folgende Funktionalitäten bieten:

- Auf einem bestehenden Netzwerk können mittels verschiedener Link-Prediction-Algorithmen die Knoten vorausgesagt werden, welche sich als nächstes verbinden. Kanten, welche aufgrund der Link-Prediction zum Netzwerk hinzugefügt werden, müssen entsprechend gekennzeichnet werden.
- Die Qualität der eingesetzten Link-Prediction-Algorithmen für durchgeführte Vorhersagen kann bewertet werden. Dadurch ist es möglich, für ein gegebenes Netzwerk den bestgeeigneten Algorithmus zu evaluieren.

Bei der Architektur des Plugins ist insbesondere wichtig, dass es einfach um weitere Link-Prediction-Algorithmen erweitert werden kann. Die genaue Aufgabenstellung ist unter Anhang B.1 zu finden.

## 1.3 Verwendete Software und Lizenzen

Nachfolgend werden die wichtigsten Komponenten beschrieben, welche für das Entwickeln und Builden des Gephi-Plugins eingesetzt wurden.

- **Gephi:** Grundlage des Link-Prediction-Plugins bildet die Software Gephi, welche zur Visualisierung und Analyse von Graphen und Netzwerken eingesetzt wird. Das Plugin wurde für Gephi Version 0.9.2 entwickelt. Bei Gephi handelt es sich um eine kostenlose Open-Source-Software.
- **Netbeans:** Für die Implementation in Java wird grösstenteils die Entwicklungsumgebung NetBeans eingesetzt. Gegenüber anderen Entwicklungsumgebungen bietet NetBeans beim Einsatz mit Gephi Vorteile beim Debugging und beim Erstellen von grafischen Benutzeroberflächen.
- **Github:** Als Code-Repository wird GitHub<sup>1</sup> eingesetzt.
- **Travis CI:** Das Plugin wird als Maven-Projekt aufgesetzt. Die Software wird mittels Travis CI<sup>2</sup> gebaut und getestet. Weil es sich bei dem Plugin ebenfalls um ein Open-Source-Projekt handelt, kann die Cloud von Travis kostenlos genutzt werden.
- **Sonarcloud:** Neben der statischen Code-Analyse wird der entwickelte Code mittels Sonarcloud<sup>3</sup> analysiert und geprüft. Die Einbindung des Projekts in Sonarcloud ist für Open-Source-Projekte ebenfalls kostenlos.
- **Lizenz:** Das Plugin wird unter der Apache License 2.0 veröffentlicht. Diese erlaubt einen freigebigen Umgang bei der Weiterentwicklung und Verwendung des Plugins.

---

<sup>1</sup><https://github.com/romanutti/gephi-plugins>

<sup>2</sup><https://travis-ci.com/romanutti/gephi-plugins>

<sup>3</sup><https://sonarcloud.io/dashboard?id=org.gephi%3Agephi-plugins>

## Kapitel 2

---

# Theoretische Grundlagen

---

## 2.1 Begriffsdefinitionen

In den folgenden Kapiteln werden die Begriffe "Social Network Analysis", "Social Forces", "Graph" und "Link Prediction" einzeln erläutert. Es handelt sich in Bezug auf die Arbeit um grundlegende Begriffe. Weitere Fachbegriffe, die in der Arbeit verwendet werden, sind im Glossar beschrieben.

### 2.1.1 Social Network Analysis

Soziale Netzwerke bestehen aus Akteuren wie beispielsweise Individuen, Organisationen oder ganze Nationen und deren Beziehungen (Ulrike Baumöl 2019, online). Unter Social Network Analysis (SNA, dt. Soziale Netzwerk-analyse) wird die Methode zur Erfassung und Analyse dieser Netzwerke und der Beziehungen darin verstanden (Wikipedia 2019c, online). Die soziale Netzwerkanalyse ist ein interdisziplinäres Forschungsfeld und wird nach Ulrike Baumöl (2019) in den Sozial- und Verhaltenswissenschaften aber auch in der Betriebswirtschaftslehre oder den Politikwissenschaften angewandt. Gegenwärtig sind sogenannte "soziale Netzwerke" allgegenwärtig und werden insbesondere im Kontext von Internet-Plattformen wie *Facebook* oder *Twitter* häufig betrachtet.

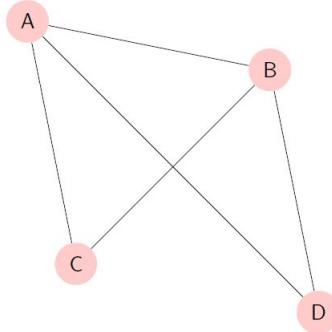
### 2.1.2 Graph

Soziale Netzwerke können einfach als Graph modelliert werden. Ein Graph  $G = (V, E)$  besteht aus einer Menge  $V = \{1, 2, \dots, |V|\}$  von Knoten und Kanten  $E \subseteq V \times V$  (vgl. Ottmann and Widmayer 2017, S. 590). Die Knoten repräsentieren dabei Akteure, während die Kanten die Beziehungen zwischen den Akteuren repräsentieren. Abbildung 2.1 zeigt einen ungerichteten Graphen. Die Kanten enthalten dabei keine Pfeile, was verdeutlicht, dass die

## 2.1. Begriffsdefinitionen

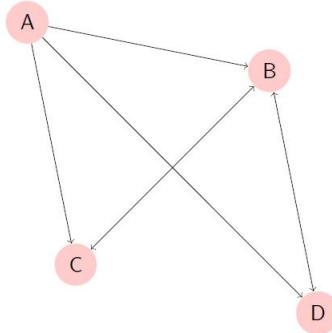
---

Interaktion in beide Richtungen erfolgen kann. Ein Beispiel für einen ungerichteten Graphen ist das Freunde-Netzwerk von *Facebook*.



**Abbildung 2.1:** Ungerichteter Graph

Im Gegensatz dazu gibt es bei einem gerichteten Graphen eine Quelle und ein Ziel. Die Richtung der Interaktion wird durch einen Pfeil visualisiert. Abbildung 2.2 zeigt einen solchen Graphen. Als Beispiel für einen gerichteten Graphen dient ein E-Mail-Netzwerk. Jede Nachricht wird dabei von einem Sender an einen Empfänger versendet.



**Abbildung 2.2:** Gerichteter Graph

### 2.1.3 Link Prediction

Link Prediction baut auf einem bestehenden Netzwerk auf und versucht vorherzusagen, welche Kanten sich als nächstes bilden werden: Der Graph  $G = (V, E)$ , bestehend aus Knoten ( $V$ , engl. Vertices) und Kanten ( $E$ , engl. Edges), hat sich in einem Zeitintervall  $G[t, t_1]$  gebildet. In einem nächsten

Zeitintervall  $G[t_1, t_2]$  könnten neue Kanten zum Netzwerk hinzukommen. Mittels Link Prediction wird nun versucht vorherzusagen, welche Kanten in welcher Reihenfolge zum Graphen hinzugefügt werden (Gao et al. 2015, S. 4). Für solche Vorhersagen gibt es verschiedene Algorithmen mit unterschiedlichen Charakteristiken. Je nach Eigenschaften des vorliegenden Netzwerks und allfälliger Kontextinformationen liefert ein Algorithmus bessere oder schlechtere Resultate. Pro Anwendungsfall muss deshalb evaluiert werden, welcher Algorithmus sich am besten eignet. Der Aufbau der dazu benötigten Messwerte und die daraus folgende Evaluation wird im Kapitel 3 beschrieben.

Welcher Algorithmus für die Link Prediction am sinnvollsten ist, kann von der Art des Netzwerks und den Social Forces im Netzwerk beeinflusst werden.

### 2.1.4 Social Forces

In sozialen Netzwerken wird die Wahrscheinlichkeit für das Entstehen und Erhalten von Kanten durch unterschiedliche Faktoren begünstigt. Zu diesen Faktoren gehören die Social Forces, die man nach Henninger (2018, S. 12) in sechs Hauptkategorien unterteilen kann: Homophilie, Reziprozität, Nähe, Ansehen, soziale Anpassung und Transitivität. Im folgenden Abschnitt werden diese Kategorien kurz beschrieben. Für die in dieser Arbeit implementierten Algorithmen sind dabei vor allem Transitivität und Ansehen relevant.

#### Homophilie

Homophilie beruht darauf, dass zwei Knoten mit Gemeinsamkeiten eine höhere Wahrscheinlichkeit dafür aufweisen, eine Verbindung aufzubauen. Hier sind beispielsweise Hobbys oder die politische Orientierung relevant. Will man für diese Art der sozialen Kräfte einen Algorithmus zur Link Prediction entwerfen, müsste dieser vorhandene Attribute miteinander vergleichen - beispielsweise das Attribut "Lieblingsband". Stimmt die Band in diesem Attribut überein, wäre laut dem Prinzip der Homophilie die Wahrscheinlichkeit höher, dass eine neue Kante entsteht.

#### Reziprozität

Reziprozität oder einfacher formuliert "Gegenseitigkeit" bedeutet, dass man eher mit einer Person Kontakt hat, wenn diese zuerst mit einem Kontakt aufnimmt. Wenn Person  $A$  also  $B$  eine Nachricht schreibt, erhöht dies die Wahrscheinlichkeit, dass  $B$  auf die Nachricht reagiert - eine Ausnahme stellen hier zum Beispiel prominente Personen dar. Diese Art der Social Forces ist vor allem in gerichteten Graphen relevant, da in ungerichteten Graphen die Richtung der Kanten nicht bestimmt ist. Eine Vorhersage zur Entstehung

## 2.1. Begriffsdefinitionen

neuer Kanten müsste dabei aber auch über andere Social Forces geschehen, um den initialen Kontakt zweier Personen vorherzusagen.

### **Nähe**

Der Begriff Nähe kann in zwei Unterkategorien unterteilt werden. Hierbei geht es um "organisatorische" und "physische" Nähe.

Die organisatorische Nähe kommt beispielsweise durch Zusammenarbeit zu stande, auch wenn man sich eventuell nicht am selben Ort befindet. Durch die Angehörigkeit derselben Abteilung in einem Unternehmen steigt die Wahrscheinlichkeit für eine Interaktion und somit für die Bildung einer Kante. Diese Art der sozialen Kraft könnte also zum Beispiel über Attribute wie "Abteilung" oder "Arbeitgeber" überprüft werden. In der Umsetzung eines Algorithmus ist diese Art der Nähe sehr ähnlich wie die der Homophilie.

Die physische Nähe bezieht sich hingegen direkt darauf, wie oft man eine Person sieht. Ist man häufiger in unmittelbarer Nähe einer Person, bevorzugt man diese tendenziell. Für die Bestimmung dieses Wertes muss also ein Attribut gefunden werden, mit welchem sich die Distanz bestimmen lässt. Als Beispiel könnte man hier zwei Studenten nehmen, welche die gleiche Klasse besuchen und beide häufig anwesend sind. Die Wahrscheinlichkeit ist höher, dass sich die physische Nähe positiver auf ihre Beziehung zueinander auswirkt, als bei zwei Studenten, die selten den Unterricht besuchen.

### **Ansehen**

Das Ansehen wird hauptsächlich über die Vernetzung einer Person definiert. Bei der Vernetzung ist es von Vorteil, wenn man die Normen und Werte einer Gruppe teilt und fördert. Dies erleichtert es, Zugang zu mehr Personen zu erhalten. Es kann außerdem auch sein, dass man sich gruppenübergreifend vernetzt. In diesem Fall kann es zu Konflikten kommen, wenn verschiedene Werte vertreten werden. Als Beispiel kann man Musiker nennen, die ihre politische Meinung äußern. Sie haben zuvor einen Fankreis aufgebaut, der aus verschiedenen Gruppen besteht, die unterschiedliche Werte und Normen haben können. Äussert der Musiker nun zum ersten Mal eine politische Meinung, kann dies in einigen der Gruppen auf Zuneigung, in anderen auf Ablehnung stoßen. Aus diesem Grund wird es auch solche Personen geben, die Meinungen geheim halten, die nicht den Gruppennormen und -werten entsprechen, um besser in die Gruppe zu passen und ihr Ansehen zu wahren.

Es ist außerdem wahrscheinlicher, dass die Personen mit einem grossen Bekanntheitskreis im Netzwerk neue Kanten bilden, während eher unpopuläre Personen (welche wenige Kanten aufweisen) das Nachsehen haben.

### **Soziale Anpassung**

Die soziale Anpassung überschneidet sich teilweise mit dem Ansehen. Auch hier stehen Gruppennormen und -werte im Fokus. Hierbei geht es aber explizit um die Anpassung der Meinung in Bezug auf die Zugehörigkeit einer Gruppe. Es handelt sich also weniger um die Vernetzung untereinander, sondern um die Auswirkungen eben jener Vernetzung.

Analysieren könnte man eine solche soziale Anpassung zum Beispiel, indem man Netzwerke über längere Zeiträume beobachtet. Wird hier beispielsweise Person A in eine Gruppe integriert und ändert daraufhin eines ihrer Merkmale (z.B. die politische Ausrichtung oder ihre Hobbys) im Einklang mit der Gruppe, ist es wahrscheinlich, dass dies aufgrund des sozialen Drucks geschehen ist. Dies wird umso wahrscheinlicher, wenn Person A mit Person B, welche ein hohes Ansehen in der Gruppe geniesst, direkt vernetzt ist<sup>1</sup>.

### **Transitivität**

Transitivität ist sehr ähnlich zur Homophilie. Hier stehen anstelle der gemeinsamen Eigenschaften allerdings gemeinsame Beziehungen im Vordergrund. Als Beispiel dienen drei Personen A, B und C - wobei B mit A und mit C befreundet ist. Die Wahrscheinlichkeit, dass A irgendwann auf C trifft, und dass die beiden sich verstehen, ist höher als sich mit einer zufälligen Person auf der Strasse anzufreunden. Hier spielen sicherlich auch andere Kräfte wie das Ansehen, die Homophilie und die Nähe eine Rolle (vgl. vorherige Abschnitte).

---

<sup>1</sup>Da Person B durch das hohe Ansehen einen sehr grossen Einfluss hat und dasselbe gruppenrelevante Merkmal mit Person A teilen dürfte

## Kapitel 3

---

# Daten

---

Um die Korrektheit von Link Predictions testen zu können, und um die Güte der verschiedenen Algorithmen bestimmen zu können, werden bestehende Netzwerke verwendet. Es wurden dafür temporale Datensets, welche in der Literatur und auf dem GitHub-Repository von Gephi verfügbar sind, verwendet (vgl. Moll 2018, online).

### 3.1 Anforderungen

Für den Aufbau der verschiedenen Tests wird ein Netzwerk zu einem bestimmten Zeitpunkt  $t$  benötigt. Auf diesem werden anschliessend Link Predictions durchgeführt. Der veränderte Graph wird anschliessend mit dem effektiven Graphen zum Zeitpunkt  $t + n$  verglichen. Bei einem Netzwerk kann es sich dabei beispielsweise um die Kanten und Knoten eines sozialen Netzwerks wie Twitter handeln: Während einer Person  $A$  zum Zeitpunkt  $t$  drei Personen folgen, sind es zum Zeitpunkt  $t + n$  fünf Personen. Daraus resultieren zwei verschiedene Graphen für dasselbe Netzwerk - allerdings zu unterschiedlichen Zeitpunkten. Voraussetzung für solche Graphen (sog. temporale oder dynamische Graphen) ist, dass die Kanten mit einer fixen Nummerierung oder einem Zeitstempel versehen sind. Diese Anforderung besteht nur für die Entwicklung und Charakterisierung der Evaluationskomponente. Im Plugin werden Graphen unabhängig von der Reihenfolge hinzugefügter Kanten bewertet.

Bei allen aufgeführten Netzwerken handelt es sich um ungerichtete Graphen.

### 3.2 Datenaufbereitung

Um die Korrektheit einzelner und die Güte mehrerer Vorhersagen am effektiven Graphen zu überprüfen, müssen die Datensets in Initial- und Vali-

### 3.3. Charakteristiken der Netzwerke

---

dierungsdaten unterteilt werden. Die Unterteilung wird anhand der Kanten gemacht. Dazu werden die Kanten aufsteigend sortiert (neueste Kanten zuerst). Anschliessend werden alle Daten aus dem Initialset dem Validierungsset zugeordnet. Auf dem Initialset werden danach für die verschiedenen Durchläufe die neuesten 1 %, 5 %, 10 % und 50 % der Kanten gelöscht. Das Initialset bildet das Basisnetzwerk zum Zeitpunkt  $t$ . Das Validierungsset entspricht dem effektiven Netzwerk zum Zeitpunkt  $t + n$ .

In der Tabelle 3.1 sind die Informationen zu den einzelnen Datensets ersichtlich.

**Tabelle 3.1:** Informationen der verwendeten Datensets

Datenset	Anzahl Knoten	Anzahl Kanten	Grösse
Hypertext Conference Network <sup>1</sup>	113	2163	2.5 MB
College Messages Network <sup>2</sup>	845	3166	1.2 MB
Sparrow Social Network <sup>3</sup>	52	446	7 KB
Enron Employees Network <sup>4</sup>	151	1583	1.4 MB
Tortoise Network <sup>5</sup>	135	374	19 KB

<sup>1</sup> Contact network during the Hypertext 2009 conference. Source: <http://ociopatterns.org>.

<sup>2</sup> Private messages sent on an online social network at the University of California, Irvine. Users could search the network for others and then initiate conversation based on profile information. An edge  $(u, v, t)$  means that user  $u$  sent a private message to user  $v$  at time  $t$ . Source: <https://snap.stanford.edu/data/CollegeMsg.html>

<sup>3</sup> Interaction network of sparrows between two points of time. Interaction is defined as birds within an approximately 5-metre radius. Source: <https://bansallab.github.io/asnr/data.html>

<sup>4</sup> Social network that represents 151 Enron employees. In this network each exchange of at least 5 emails between any pair of agents across the entire time range (1998 to 2002) is considered as a link. Source: <https://pdfs.semanticscholar.org/4d7d/8fe43f391a966e0e15b68e6509fe6b533540.pdf>

<sup>5</sup> Social networks of desert tortoises captured at several points of time. Source: <https://bansallab.github.io/asnr/data.html>

## 3.3 Charakteristiken der Netzwerke

Um die Netzwerke beschreiben zu können, werden verschiedene Metriken berechnet. Nach Gao et al. (2015, S. 5) weisen die nachfolgenden vier Messgrössen eine besonders starke Korrelation zum gewählten Link-Prediction-Algorithmus auf.

### Global Cluster Coefficient (GCC)

Der Cluster Coefficient zeigt das Mass der Cliquenbildung in einem Graphen (Henninger 2018, S. 58). Unter Cliquen wird ein Subgraph innerhalb eines Graphen, in welchem die Knoten untereinander stark verbunden sind, verstanden. In Cliquen stammt ein Grossteil der Informationen von denselben

Informationslieferanten - dank der starken Verbundenheit kann sich die Gruppe schnell austauschen.

Der lokale Cluster Koeffizient wird aus dem Quotienten der Anzahl direkten Kanten  $d$  zwischen den Nachbarn eines Knoten und der maximal möglichen Anzahl direkter Kanten  $k_i(k_i - 1)$  berechnet:

$$C_i = \frac{2 * d}{k_i(k_i - 1)} \quad (3.1)$$

Bei ungerichteten Graphen wird die Anzahl direkter Kanten mit dem Faktor 2 multipliziert.  $k_i$  steht für die Anzahl benachbarter Knoten. Beim globalen Cluster Koeffizienten handelt es sich um den Mittelwert der lokalen Cluster Koeffizienten. In einem Graphen mit  $|V|$  Knoten wird er folgendermassen berechnet:

$$GCC = \frac{1}{|V|} \sum_{i=1}^{|V|} C_i \quad (3.2)$$

### Density

Die Density (dt. Dichte) eines Graphen zeigt an, wie komplett die Knoten des Netzwerkes miteinander verbunden sind. Falls es in einem Netzwerk keine Kanten gibt, liegt der Density-Wert bei 0. Falls alle Knoten mit allen anderen Knoten verbunden sind, entspricht der Density-Wert 1. Die Dichte ist in der Regel ein Indikator, wie schnell sich Informationen in einem Netzwerk verbreiten. Die Density für ungerichtete Graphen berechnet sich mit unterstehender Gleichung, wobei  $|E|$  für die Anzahl Kanten und  $|V|$  für die Anzahl Knoten steht:

$$Density = \frac{2|E|}{|V| * (|V| - 1)} \quad (3.3)$$

Bei grösseren Netzwerken steigt mit der Anzahl Knoten auch die Anzahl möglicher Kanten. Häufig ist zu beobachten, dass die Anzahl direkter Verbindungen dabei konstant bleibt - die Density nimmt bei grösseren Netzwerken folglich häufig ab (vgl. Henninger 2018, S. 56).

### Diameter

Unter dem Graph Diameter (dt. Durchmesser) versteht man nach Henninger (2018, S. 57) den *längsten kürzesten Pfad* zwischen allen Knotenpaaren in

### 3.3. Charakteristiken der Netzwerke

---

einem Graphen. Die Formel für den längsten kürzesten Pfad lautet folgendermassen:

$$\text{Diameter} = \max_{i,j} d(i,j) \quad (3.4)$$

Wobei  $d(i,j)$  der kürzeste Pfad zwischen den Knoten  $i$  und  $j$  ist.

#### Average Shortest Path (ASP)

Mit dem Average Shortest Path (ASP) wird die durchschnittliche Länge des kürzesten Pfads zwischen zwei Knoten  $i$  und  $j$  berechnet (Gao et al. 2015, S. 6). Die Formel dafür lautet folgendermassen:

$$ASP = \frac{1}{|V| * (|V| - 1)} * \sum_{i \neq j} d(i,j) \quad (3.5)$$

Mithilfe dieser Metriken lässt sich die gesamte Netzwerkstruktur einordnen. Tabelle 3.2 zeigt die berechneten Werte der verschiedenen Netzwerke.

Datenset	GCC	Density	Diameter	ASP
Hypertext Conference Network	0.535	0.342	3.000	1.661
College Messages Network	0.122	0.001	7.000	3.108
Sparrow Social Network	0.695	0.336	3.000	1.756
Enron Employees Network	0.519	0.133	4.000	2.136
Tortoise Network	0.391	0.041	10.000	3.735

**Tabelle 3.2:** Metrikwerte der Netzwerke

Auffallend sind dabei insbesondere die tiefen Werte für GCC und Density beim College Messages Network. Daraus kann geschlossen werden, dass Knoten in diesem Netzwerk nur lose zusammenhängen und die Verknüpfung nicht sehr stark ausgeprägt ist. Beim Tortoise Network weichen ausserdem die Werte für Density und Diameter von den Werten der anderen Datensets ab. Die Werte legen die Vermutung nahe, dass dieses Netzwerk ebenfalls nicht sehr stark verknüpft ist. Diesen stark abweichenden Werten gilt bei der Interpretation der erzielten Resultate der verschiedenen Algorithmen (vgl. Kapitel 4.3) besonderes Augenmerk.

## Kapitel 4

---

# Link Prediction

---

### 4.1 Algorithmen

Link Predictions basieren auf einem bestehenden Netzwerk und versuchen, neue Kanten zu prognostizieren. Es gibt verschiedene Algorithmen, um diese Kanten vorherzusagen. Initial werden im Plugin die Algorithmen "Common Neighbours" und "Preferential Attachment" implementiert. Diese werden nachfolgend genauer beschrieben.

Die Funktionsweise der besagten Algorithmen wird anhand von konkreten Beispielen veranschaulicht. Als Grundlage für die Vorhersagen wird der Beispielgraph aus Abbildung 4.1 verwendet.

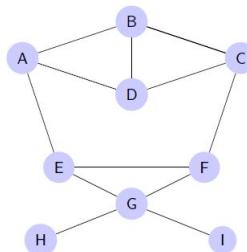


Abbildung 4.1: Beispielgraph (in Anlehnung an Henninger 2018, S.97)

#### 4.1.1 Common Neighbours

Der Common Neighbours Algorithmus berechnet für zwei nicht verbundene Knoten, wie viele gemeinsame Nachbarn existieren. Die Wahrscheinlichkeit für das Entstehen einer neuen Kante ist umso höher, desto mehr gemeinsame Nachbarn zwei Knoten haben. Common Neighbours beruht auf dem

Social Forces Prinzip der Transitivität: Freunde, welche gemeinsame Freunde haben, werden sich wahrscheinlicher miteinander anfreunden. Je mehr gemeinsame Freunde zwei Knoten aufweisen können, desto höher ist also die Wahrscheinlichkeit, dass diese Knoten Kontakt knüpfen und eine neue Kante gebildet wird (vgl. Kapitel 2.1.4).

Die folgende Formel zeigt, wie die Anzahl der gemeinsamen Nachbarn von zwei Knoten X und Y berechnet werden kann:

$$\text{common\_neighbours}(X, Y) = |N(X) \cap N(Y)| \quad (4.1)$$

Der Wert, der aus der Berechnung für diesen Algorithmus resultiert, ist die Summe der Anzahl aller gemeinsamen Nachbarn der beiden Knoten X und Y. Dieser Wert muss für alle Knoten-Kombinationen berechnet und anschliessend der höchste Wert ausgewählt werden.

Angewandt auf den eingangs eingeführten Beispielgraphen würde der Common Neighbour Algorithmus eine neue Kante zwischen den Knoten A und C vorhersagen. Die folgenden Beispiele zeigen weitere Werte, welche durch Anwenden des Algorithmus resultieren.

$$\text{common\_neighbours}(A, C) = |B, D| = 2$$

$$\text{common\_neighbours}(A, F) = |E| = 1$$

$$\text{common\_neighbours}(A, I) = |\{\}| = 0$$

Das Code-Listing 4.1 zeigt die vereinfachte Implementierung des Algorithmus. Weil initial für jeden Knoten durch sämtliche Knoten iteriert werden muss, besitzt die Implementierung die Komplexitätsklasse  $O(n^2)$ .

---

**Listing 4.1:** Common neighbour implementation

---

```
// CommonNeighboursStatistics.java
public void calculateAll(Graph graph) {

    //Iterate over all nodes
    for (Node a : graph.getNodes()) {

        // Remove self from nodes,
        // of which neighbourhood will be
        // verified
        List<Node> others = graph.getNodes().remove(a);
```

---

```

// Get neighbours of a
List<Node> aNeighbours = getNeighbours(graph, a);

//Calculate common neighbours
for (Node b : others) {

    // Get neighbours of b
    List<Node> bNeighbours = getNeighbours(graph, b);
    // Count number of common neighbours
    int commonNeighboursCount =
        getCommonNeighboursCount(aNeighbours, bNeighbours);

    // Temporary save calculated
    // value if edge does not exist
    if (isNewEdge()) {
        saveCalculatedValue(a, b, commonNeighboursCount)
    }
}

// Add edge with highest calculation to graph
Edge max = getHighestPrediction();
graph.addEdge(max, "Common Neighbours");
}

```

---

### 4.1.2 Preferential Attachment

Mit dem Preferential Attachment Algorithmus werden ebenfalls die Nachbarn eines Knotens berücksichtigt. Diese müssen jedoch nicht mit den Nachbarn des anderen Knoten übereinstimmen - es geht lediglich um die Anzahl der eigenen Nachbarn. Bei Preferential Attachment wird von der Annahme ausgegangen, dass die Wahrscheinlichkeit einen neuen Kontakt zu knüpfen grösser wird, je grösser das eigene Netzwerk bereits ist. Die Entstehung einer neuen Kante wird in diesem Fall nicht über Transitivität bestimmt, sondern über Ansehen (vgl. Kapitel 2.1.4).

Die Formel für den Preferential Attachment Algorithmus lautet wie folgt:

$$preferential\_attachment(X, Y) = |N(X)| \cdot |N(Y)| \quad (4.2)$$

Um den resultierenden Wert zu erhalten, wird die Anzahl Nachbarn zweier Knoten  $X$  und  $Y$  miteinander multipliziert. Dieser Wert muss für alle Knoten-Kombinationen berechnet und anschliessend der höchste Wert ausgewählt werden.

Angewandt auf den Beispielgraphen würde mit Preferential Attachment eine Kante zwischen den Knoten  $A$  und  $G$  vorausgesagt und folgende weiteren Werte berechnet:

$$\text{preferential\_attachment}(A, G) = 3 * 4 = 12$$

$$\text{preferential\_attachment}(A, C) = 3 * 3 = 9$$

Das Code-Listing 4.2 zeigt die vereinfachte Implementierung des Algorithmus. Der Algorithmus besitzt die Komplexitätsklasse  $O(n^2)$ .

**Listing 4.2:** Preferential attachment implementation

---

```
// PreferentialAttachmentStatistics.java
public void calculateAll(Graph graph) {

    //Iterate over all nodes
    for (Node a : graph.getNodes()) {

        // Remove self from nodes,
        // of which neighbourhood will be
        // verified
        List<Node> others = graph.getNodes().remove(a);

        // Get neighbours of a
        List<Node> aNeighbours = getNeighbours(graph, a);

        //Calculate preferential attachment
        for (Node b : others) {

            // Get neighbours of b
            List<Node> bNeighbours = getNeighbours(graph, b);
            // Multiply number of neighbours
            int totalNeighboursCount = aNeighbours.size() *
                bNeighbours.size();

            // Temporary save calculated
            // value if edge does not exist
            if (isNewEdge()) {
                saveCalculatedValue(a, b, totalNeighboursCount)
            }
        }
    }

    // Add edge with highest calculation to graph
    Edge max = getHighestPrediction();
}
```

---

```

graph.addEdge(max, "Preferential Attachment");

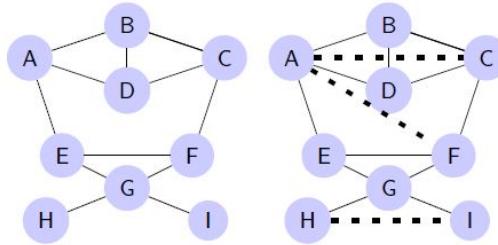
}

```

---

## 4.2 Evaluation

Basierend auf zwei Graphen wird die Genauigkeit der Vorhersagen unter Verwendung verschiedener Algorithmen bewertet. Ausgehend von einem initialen Graphen  $G_{i,t}$  zum Zeitpunkt  $t$  prognostizieren wir  $n$  Kanten  $E_i$ , was zu einem Graphen  $G_{i,t+n}$  zum Zeitpunkt  $t+n$  führt. Dieser neue Graph  $G_{i,t+n}$  wird zur Validierung der vorhergesagten Kanten verwendet. Abbildung 4.2 zeigt die Graphen  $G_{i,t}$  (links) und  $G_{i,t+n}$  (rechts) anhand des vorhin eingeführten Beispiels. Die hinzugefügten Kanten  $E_i$  sind dabei mit einer gestrichelten Linie eingezeichnet.



**Abbildung 4.2:** Initialer Graph zum Zeitpunkt  $t$ , resp.  $t+n$

Die Qualität des Algorithmus wird anhand der Genauigkeit der hinzugefügten Kanten bestimmt. Dazu wird ein Validierungsgraph  $G_{v,t+n}$  verwendet, bei welchem die zusätzlichen Kanten  $E_v$  zum Zeitpunkt  $t+n$  bereits vorhanden sind. In unserem Beispielnetzwerk enthält dieser Graph gegenüber dem Graphen  $G_{i,t}$  zusätzlich die Kanten  $(A,C)$ ,  $(E,H)$  und  $(H,I)$ . Abbildung 4.3 auf der nächsten Seite zeigt den Graphen  $G_{v,t+n}$  zum Zeitpunkt  $t+n$ .

Als Qualitätsmaß wird die Accuracy verwendet. Diese berechnet sich als prozentualer Anteil der korrekt vorhergesagten Kanten im Verhältnis zu allen hinzugefügten Kanten.

$$Acc = |E_i \cap E_v| / |E_v| * 100 \quad (4.3)$$

Im obigen Beispiel mit drei zusätzlichen Kanten wurden die Kanten  $(A,C)$  und  $(H,I)$  korrekt vorhergesagt. Daraus folgt eine Accuracy von 66.67 %.

### 4.3. Gegenüberstellung der verschiedenen Algorithmen und Netzwerke

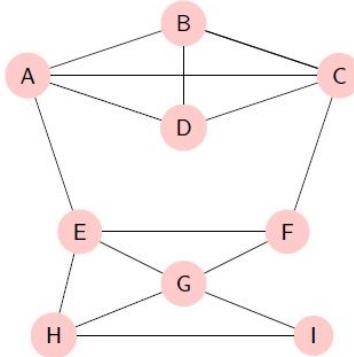


Abbildung 4.3: Validierungsgraph zum Zeitpunkt  $t + n$

Die Herleitung dazu sieht folgendermassen aus:

$$E_i = \{(A, C), (A, F), (H, I)\}$$

$$E_v = \{(A, C), (E, H), (H, I)\}$$

$$|E_i \cap E_v| = |\{(A, C), (H, I)\}| = 2$$

$$Acc = |E_i \cap E_v| / |E_v| * 100 = 2 / 3 * 100 = 66.67$$

## 4.3 Gegenüberstellung der verschiedenen Algorithmen und Netzwerke

Die Qualität der verschiedenen Algorithmen kann beurteilt werden, indem die eben eingeführte Accuracy auf die Daten aus Kapitel 3 angewendet wird. Um die Qualität der Algorithmen besser beurteilen zu können, wird die Entwicklung mit zunehmender Anzahl Iterationen betrachtet. In jedem Iterationsschritt wird eine neue Kante zu den Graphen hinzugefügt und die Accuracy zu diesem Zeitpunkt berechnet.

Die Tabelle 4.1 zeigt die Accuracy für die verschiedenen Datensets nach Hinzufügen von 1 %, 5 %, 10 % und 50 % neuen Kanten. Die Accuracy wird jeweils für die Algorithmen Common Neighbour (CN) und Preferential Attachment (PA) ausgewiesen. Für Einträge mit dem Symbol “-“ sind keine temporalen Daten zur Verifizierung der Accuracy verfügbar. Diese Einträge werden bei der Interpretation in den folgenden Abschnitten nicht weiter berücksichtigt.

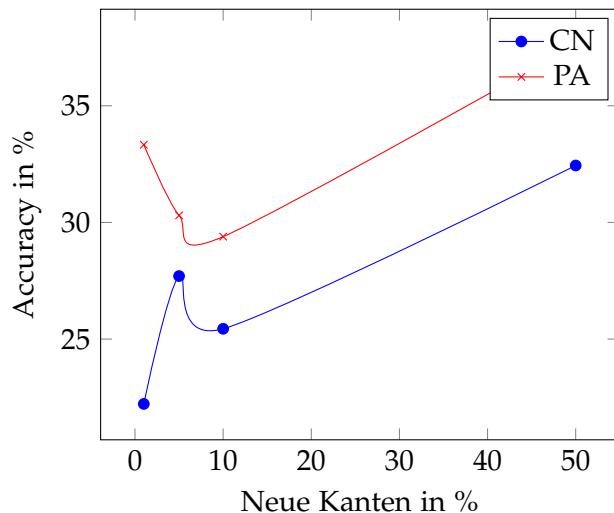
### 4.3. Gegenüberstellung der verschiedenen Algorithmen und Netzwerke

**Tabelle 4.1:** Erreichte Accuracy in %

Datenset	Accuracy							
	1% neue Kanten		5% neue Kanten		10% neue Kanten		50% neue Kanten	
	CN	PA	CN	PA	CN	PA	CN	PA
Hypertext Network	22.22	33.33	27.70	30.30	25.44	29.39	32.44	37.60
College Messages Network	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Sparrow Social Network	-	-	-	-	-	-	26.60	26.60
Enron Employees Network	0.00	0.00	0.00	0.00	9.40	0.00	12.26	9.94
Tortoise Network	-	-	-	-	11.54	0.00	9.73	8.85

Bei der Analyse der erreichten Werte fällt auf, dass im College Message Network keine richtigen Vorhersagen getroffen werden konnten. Auch bei den anderen Netzwerken sind die Vorhersagen da am genauesten, wo ein hoher Clustering Koeffizient und eine hohe Density vorliegen. Dies legt die Vermutung nahe, dass für die Algorithmen eine Korrelation zwischen der Qualität der Vorhersagen und der Clustering Koeffizient resp. der Density vorliegt. Um die Vermutung empirisch zu überprüfen, müssten jedoch noch weitere Netzwerke untersucht werden.

Weiter fällt auf, dass die erreichten Accuracy-Werte mit zunehmender Anzahl hinzugefügter Kanten steigen. Dies ist auf die aktuelle Implementierung zurückzuführen: Für jede hinzugefügte Kante wird überprüft, ob diese im Graphen vorhanden ist. Je grösser die Anzahl hinzugefügter Kanten, desto grösser ist auch die Wahrscheinlichkeit, dass eine hinzugefügte Kante im Graphen existiert. Abbildung 4.4 zeigt dies am Beispiel vom Hypertext Conference Network.



**Abbildung 4.4:** Accuracy im Hypertext Network

### 4.3. Gegenüberstellung der verschiedenen Algorithmen und Netzwerke

---

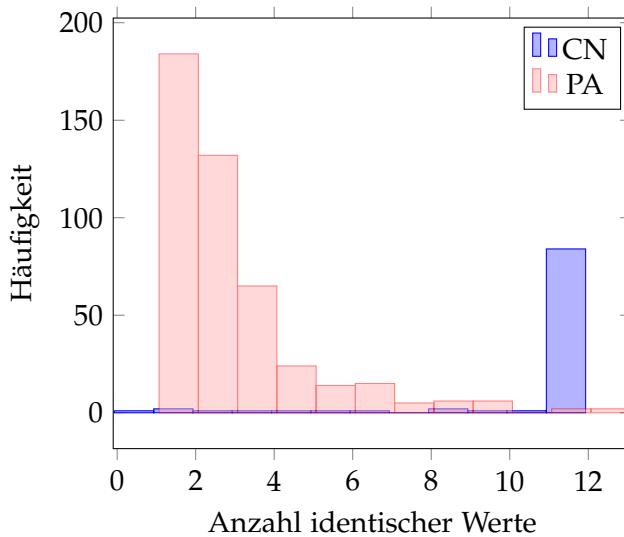


Abbildung 4.5: Histogramm der errechneten Werte im Hypertext Network

Bei den berechneten Werten kann festgestellt werden, dass die berechneten Prediction-Werte beim Common Neighbour Algorithmus weniger breit gestreut sind als beim Preferential Attachment Algorithmus. In der aktuellen Implementierung wird für die Wahl des nächsten Prediction Links eine Priority Queue verwendet. Falls es mehrere potentielle Kanten mit demselben Prediction-Wert gibt, wird willkürlich eine der Kanten ausgewählt. Da beim Preferential Attachment Algorithmus die Werte besser streuen, kann eindeutiger gesagt werden, welche Kante als nächstes zum Netzwerk hinzugefügt werden soll. Abbildung 4.5 zeigt die Verteilung der berechneten Werte bei 50% hinzugefügter Kanten für das Hypertext Conference Network.

## Kapitel 5

---

# Plugin

---

Basierend auf den beschriebenen Erkenntnissen wurde ein Plugin für Gephi entwickelt, mit welchem Kanten mit verschiedenen Link-Prediction-Algorithmen vorhergesagt werden können. Kanten, welche zum Netzwerk hinzugefügt worden sind, werden entsprechend gekennzeichnet. Die Anzahl vorhergesagter Kanten kann von den Benutzern angegeben werden. In der Literatur werden hauptsächlich Algorithmen beschrieben, welche auf ungerichtete Graphen anwendbar sind. Algorithmen für gerichtete Graphen weisen eine wesentlich höhere Komplexität auf. Die Arbeit beschränkt sich deshalb auf Link Predictions für ungerichtete Graphen - ein entsprechender Hinweis wurde im Gephi Plugin implementiert. Das Plugin enthält zudem eine Evaluationskomponente, mit welcher die Qualität der verschiedenen Algorithmen verglichen werden kann.

Das Plugin wird unter der Apache 2.0 Lizenz veröffentlicht.

### 5.1 Software Architektur

Das Plugin besteht aus folgenden Hauptkomponenten:

- **Statistik:** Neue Kanten können via Statistiken zu einem ungerichteten Graphen hinzugefügt werden. Für die ausgewählten Algorithmen werden die Link-Prediction-Werte berechnet und jeweils die Kante mit dem höchsten Wert zum Graphen hinzugefügt. Die Anzahl hinzuzufügender Kanten kann angegeben werden. Die einzelnen Kanten werden dabei iterativ hinzugefügt. Die Berechnung des nächsten vorhergesagten Wertes basiert immer auf dem Graphen des vorherigen Iterationsschrittes.
- **Filter:** Die hinzugefügten Kanten können via Filter angezeigt werden. Dabei können die anzuzeigenden Kanten einerseits auf einen Algorith-

mus, andererseits auf eine gewünschte Anzahl hinzugefügter Kanten beschränkt werden.

- **Evaluation:** Die Güte der vorhergesagten Kanten kann berechnet werden. Via Statistiken wird die Accuracy von verschiedenen Algorithmen verglichen und beurteilt.

### 5.1.1 Klassenstruktur

Die Klassenstruktur ist grösstenteils bereits durch das bestehende Grundgerüst von Gephi vorgegeben. Für eine möglichst einfache Erweiterbarkeit, werden entsprechende Design Patterns eingesetzt.

Abbildung 5.1 zeigt das Klassendiagramm der Statistik-Klassen. Es basiert auf dem Composite Command Pattern<sup>1</sup>, um die Link Prediction Wahrscheinlichkeiten der verschiedenen Algorithmen zu berechnen. Das Composite Pattern beruht darauf, in einer abstrakten Klasse sowohl konkrete Objekte als auch ihre Behälter zu repräsentieren. Damit können sowohl einzelne Objekte als auch ihre Kompositionen einheitlich behandelt werden (Wikipedia (2019a), online). Das Makro Command Objekt LinkPredictionMacro wird eingesetzt, um die Link-Prediction-Werte von mehreren ausgewählten Algorithmen zu berechnen.

<sup>1</sup>[https://en.wikipedia.org/wiki/Composite\\_pattern](https://en.wikipedia.org/wiki/Composite_pattern)

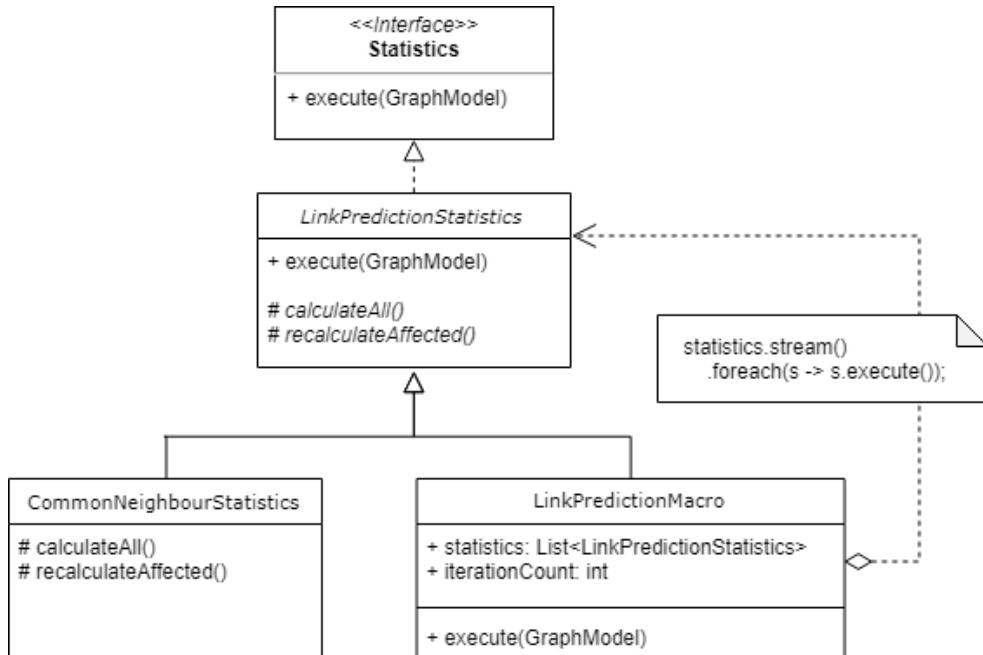


Abbildung 5.1: Klassendiagramm Statistiken

Bei der `execute`-Methode wird auf das Template Pattern<sup>2</sup> zurückgegriffen. Mit dem Template Pattern kann eine Grundstruktur eines Algorithmus vorgegeben werden, wobei durch Unterklassen ermöglicht wird, bestimmte Schritte des Algorithmus zu überschreiben, ohne dessen grundlegende Struktur zu verändern. Das Pattern wird hier verwendet, um die unveränderlichen Teile eines Algorithmus einmalig zu implementieren, doppelten Code zu reduzieren und die Erweiterungen der Unterklassen zu regulieren. In der `execute`-Methode wird der grobe Ablauf umschrieben. Dieser sieht vor, dass bei der initialen Berechnung der Link-Prediction-Werte für alle Knoten durch sämtliche Knoten iteriert werden muss.

Beim Ausführen der Link Prediction kann angegeben werden, wieviele Kanten hinzugefügt werden sollen. Aus Performancegründen erwies es sich als sinnvoll, für die nachfolgenden Berechnungen nicht mehr sämtliche Werte nochmals zu berechnen. Stattdessen werden nach dem Hinzufügen einer Kante im nächsten Iterationsschritt nur die davon betroffenen Kanten neu berechnet. Falls beispielsweise eine Kante ( $A, C$ ) zum Graphen hinzugefügt wird, müssen anschliessend nur die Werte der Kanten mit den Knoten  $A$  und  $C$  neu berechnet werden. Die `execute`-Methode enthält deshalb den groben Ablauf, welcher die Unterscheidung zwischen initialem Durchlauf und nachfolgender Neuberechnung vorsieht. Die Methoden `calculateAll` und `recalculateAffected`, die von den konkreten Klassen implementiert werden und die effektiven Link-Prediction-Werte berechnen, werden als abstrakt definiert. Das Listing 5.1 zeigt die vereinfachte Struktur der Methode.

---

**Listing 5.1:** Link prediction implementation

---

```
// LinkPredictionStatistics.java
@Override public void execute(GraphModel graphModel) {

    if (isInitialExecution()) {
        // Iterate over all nodes for first iteration
        calculateAll(Graph graph)
    } else {
        // Only change affected node for subsequent iterations
        recalculateAffectedNodes(factory);
    }

    // Add highest predicted edge to graph
    addHighestPredictedEdgeToGraph(factory, getAlgorithmName());
}


```

---

<sup>2</sup><https://de.wikipedia.org/wiki/Schablonenmethode>

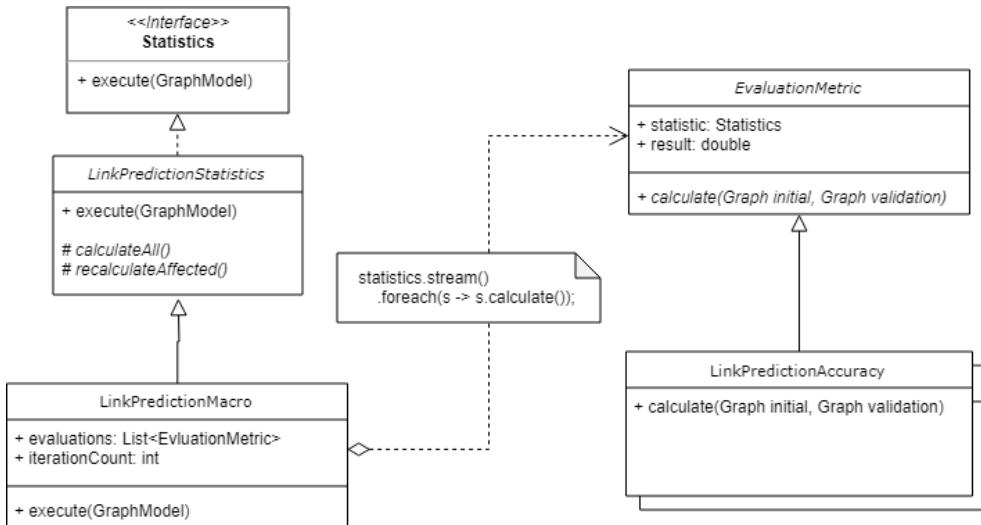


Abbildung 5.2: Klassendiagramm Evaluationskomponente

Eine ähnliche Struktur wird bei der Evaluationskomponente verwendet. Die Qualität der verschiedenen Algorithmen wird mit der bereits eingeführten Accuracy (vgl. Formel 4.3) bestimmt. Damit das Plugin später einfach um andere Evaluationsmetriken erweitert werden kann, wurde eine neue Basisklasse `EvaluationMetric` eingeführt. Diese enthält den zu evaluierenden Algorithmus, die Berechnung der Kennzahl und die erreichte Güte der Vorhersage. Konkrete Evaluationsmetriken, wie `LinkPredictionAccuracy` erweitern die abstrakte Basisklasse mit einer konkreten Implementierung der `calculate`-Methode. Abbildung 5.2 zeigt das vereinfachte Klassendiagramm der Evaluation-Klassen.

Die Filter-Klassen wurden mithilfe des Abstract Factory Pattern<sup>3</sup> umgesetzt. Die Klassenstruktur besteht aus dem Factory Pattern mit verschiedenen Builder Klassen und dem Strategy Pattern, welches eine spezifische Umsetzung der Filter pro Algorithmus erlaubt. Abbildung 5.3 auf der nächsten Seite zeigt das resultierende Klassendiagramm.

### 5.1.2 Loggingkonzept

Im Plugin wird das Logging-Framework log4j<sup>4</sup> eingesetzt. Das Logging wird eingesetzt um technische Probleme rückverfolgen zu können: Die Logs zeichnen Ereignisse auf, die für die Entwicklung und den Einsatz des Plugin von Nutzen sein können. Insbesondere geht es darum ein Fehlverhalten der Applikation nachzuvollziehen und dokumentieren zu können. Das Logging kann in Gephi direkt in einer Konsole ausgegeben werden.

<sup>3</sup><http://www.hsufengko.com/blog/abstract-factory-design-pattern-example>

<sup>4</sup><https://logging.apache.org/log4j/2.x/>

## 5.1. Software Architektur

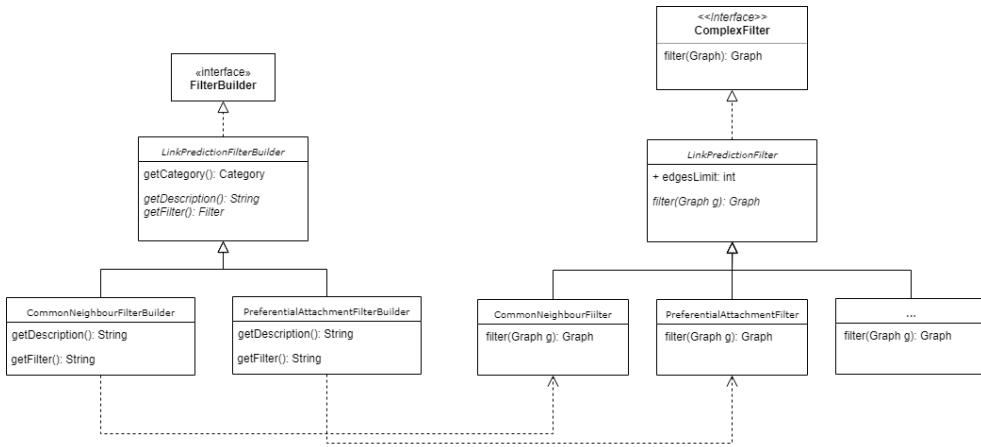


Abbildung 5.3: Klassendiagramm Filter

### Logger erzeugen

Logger sind für das Verfassen von Meldungen verantwortlich. Sie werden direkt aus dem Code aufgerufen und bestimmen mit Hilfe von Appendern, wohin und in welchem Format die Meldungen geschrieben werden. Für jede Klasse wird ein separater Logger erzeugt. Das Codelisting 5.2 zeigt, wie ein solcher Logger in der gewünschten Klasse erzeugt wird.

Listing 5.2: Erzeugen eines Loggers

```
import org.apache.log4j.Logger;

/**
 * ...
 */
public class LinkPredictionStatistics {

    // Console Logger
    private static Logger consoleLogger =
        LogManager.getLogger(LinkPredictionStatistics.class);
```

### Log-Einträge erzeugen

Log4j bietet verschiedene Log-Levels an, mit welchen Log-Einträge geschrieben werden können. Tabelle 5.1 auf der nächsten Seite zeigt die verschiedenen Log-Levels anhand der API-Docs von log4j.

Grundsätzlich kann direkt mit den Funktionen auf dem Logger-Objekt (z.B. trace, debug, etc.) geloggt werden. Falls in der Log-Meldung Variablen ausgewertet werden, empfiehlt sich jedoch die Lambda-Schreibweise mit Lazy-Evaluation. Listing 5.3 zeigt zwei solcher Beispiele.

Level	Beschreibung
TRACE	The TRACE Level designates finer-grained informational events than the DEBUG.
DEBUG	The DEBUG Level designates fine-grained informational events that are most useful to debug an application.
INFO	The INFO level designates informational messages that highlight the progress of the application at coarse-grained level.
WARN	The WARN level designates potentially harmful situations.
ERROR	The ERROR level designates error events that might still allow the application to continue running.
FATAL	The FATAL level designates very severe error events that will presumably lead the application to abort.

Tabelle 5.1: Logging-Levels

Listing 5.3: Log-Einträge erzeugen

---

```

public int getNextIteration(Graph graph, String algorithm) {
    consoleLogger.trace("Lookup next iteration");
    int lastIteration;
    // do some lookups here

    consoleLogger.log(Level.DEBUG, () -> "Number of last iteration: "
        + lastIteration);
}

```

---

## Logger Hierarchie

Alle Logger der verschiedenen Klassen erben vom Root-Logger. Über diesen kann die Konfiguration für alle Logger geändert werden. Einzelne Logger können die Root-Konfiguration bei Bedarf übersteuern.

## Appender

Aktuell wird im Plugin hauptsächlich der ConsoleAppender eingesetzt. Mit dessen Hilfe kann direkt in die Gephi-Konsole geloggt werden.

## Konfiguration

Die Konfiguration kann direkt in der Konfigurationsdatei `log4j.xml` vorgenommen werden. Das Codelisting 5.4 zeigt die wichtigsten Auszüge der Datei.

Listing 5.4: log4j.xml

---

```

<?xml version="1.0" encoding="UTF-8" ?>
<Configuration>
<Appenders>
<Console name="console" target="SYSTEM_OUT">

```

```
<PatternLayout pattern="%d{HH:mm:ss} %-5p %L [%C] - %m%n"/>
</Console>

<Loggers>
<Logger name =
        "org.gephi.plugins.linkprediction.statistics.PreferentialAttach
        mentStatistics" additivity="false">
<AppenderRef ref="console" />
</Logger>

<Root level="INFO" additivity="false">
<AppenderRef ref="file" />
</Root>
</Loggers>

</Configuration>
```

---

### 5.1.3 Exception-Handling

Beim Handling von unerwarteten Situation wird zwischen Exceptions und Errors unterschieden.

Exceptions sind unerwartete Zustände, bei denen das Plugin weiterlaufen kann, allenfalls ist dabei eine Handlung des Benutzers erforderlich. In solchen Situationen wird dem Benutzer ein Pop-up Fenster mit einem entsprechenden Hinweis angezeigt. Im Plugin existiert dafür die Basisklasse `LinkPredictionWarning`, welche durch die konkreten Warning-Klassen erweitert wird. Im Gegensatz zu Exceptions tritt bei Errors ein Fehler auf, der ein korrektes Weiterlaufen des Plugins verhindert. In diesem Fall wird auf das Gephi-interne Exception-Handling zurückgegriffen und der Stacktrace dem Benutzer angezeigt.

## 5.2 GUI Konzept

Die Link-Prediction-Funktionalität soll möglichst gut in die bestehende Gephi-Benutzeroberfläche integriert werden. Das GUI-Konzept des Plugins basiert auf den folgenden zentralen UI-Elementen:

- Unter dem Menü-Punkt `Statistics` werden diverse Werte des Graphen berechnet und gegebenenfalls im Data Laboratory (dt. Datenlabor) hinzugefügt. Die Darstellung der Ergebniswerte erfolgt über einen Report, sofern sinnvoll.
- Unter dem Menü-Punkt `Filter` werden lediglich bereits vorhandene Daten, gefiltert. Hierbei werden Graphen auf Subgraphen reduziert.

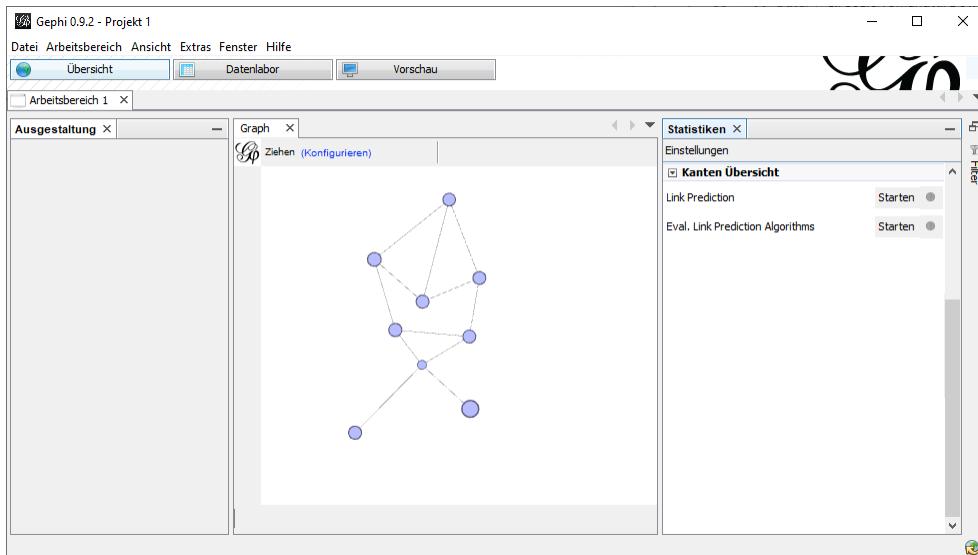
- Über den Menü-Punkt **Appearance** (dt. Ausgestaltung) kann die Darstellung der Knoten und Kanten verändert werden.
- Im **Data Laboratory** sind die aktuellen Daten zu finden. Werden neue Daten über Statistics berechnet oder die Daten im Laboratory verändert, werden nur die neusten Daten angezeigt. Um alte Daten zu erhalten, muss bei jeder Änderung ein neuer Workspace erstellt werden.

In den nachfolgenden Kapiteln wird die Benutzerführung der wichtigsten Funktionalitäten beschrieben. Die vollständigen Wireframes sind im Anhang A.2 ersichtlich.

### 5.2.1 Vorhersagen neuer Kanten

Die Berechnung der neuen Kanten muss im Bereich der **Statistics** passieren, da hier neue Werte berechnet und dem Data Laboratory hinzugefügt werden. Abbildung 5.4 zeigt einen Überblick über die Benutzeroberfläche von Gephi. Hellblau hinterlegt sind die angewählten Menü-Punkte. Unter dem Menü **Statistics** wird hier nun ein Element "Link Prediction" mit einem Start-Button eingefügt. Um die Darstellung auf die wesentlichen Elemente zu reduzieren, wurden bestehende Statistiken nicht eingezeichnet.

Falls die **Start**-Schaltfläche angewählt wird, öffnet sich ein Pop-up Fenster, in welchem man verschiedene Algorithmen auswählen kann. Für alle ausgewählten Algorithmen, werden Link Predictions durchgeführt. Abbildung 5.5 auf der nächsten Seite zeigt das Pop-up für die Auswahl der Berech-



**Abbildung 5.4:** Überblick der Ansicht Statistik.

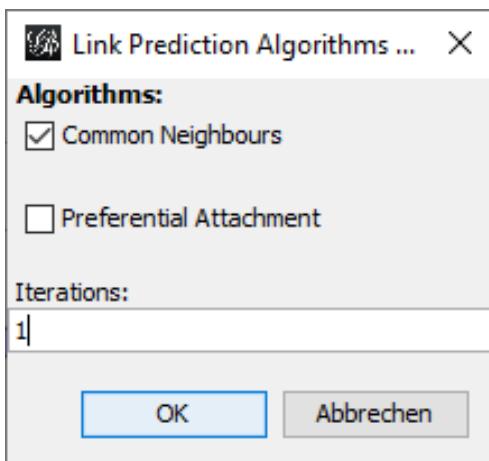


Abbildung 5.5: Pop-up für die Auswahl der Berechnung.

nung. Weitere Informationen zum jeweiligen Algorithmus werden angezeigt, wenn mit der Maus über das entsprechende Element gefahren wird. In einem zusätzlichen Eingabefeld kann man einen Wert  $>= 1$  eingeben. Dieser besagt, wie viele Durchgänge bei der Link Prediction durchlaufen werden sollen, respektive wie viele neue Kanten dem Graphen hinzugefügt werden. Die einzelnen Kanten werden iterativ zum Graphen hinzugefügt. Die Berechnung des nächsten vorhergesagten Wertes basiert immer auf dem Graphen des vorherigen Iterationsschrittes.

Die hinzugefügten Kanten können via Filter angezeigt werden. Dabei können die anzuseigenden Kanten einerseits auf einen Algorithmus, andererseits auf eine gewünschte Anzahl hinzugefügter Kanten beschränkt werden.

Ein grosses Netzwerk oder auch eine hohe Anzahl an Iterationen kann in einer grossen Laufzeit resultieren. In diesem Fall wird eine Warnung an den User ausgegeben. Wird diese bestätigt, so wird der Algorithmus trotzdem berechnet. Andernfalls kann der User seine gewünschten Parameter nochmals verändern.

Abbildung 5.6 zeigt den Graph nach der Berechnung der Link Predictions. Gegenüber dem initialen Graphen wurden neue Kanten hinzugefügt. Diese können mit Hilfe der Appearance-Einstellungen eingefärbt werden, da im Data Laboratory Felder enthalten sind, die aussagen, ob eine Kante mit einem Link-Prediction-Algorithmus hinzugefügt wurde oder ob sie bereits existiert hat.

Ebenfalls auf der nächsten Seite zeigt Abbildung 5.7 die neuen Kanten im **Data Laboratory**. Wie bereits erwähnt, wurden den Kanten neue Attribute hinzugefügt. Neu gibt es ein Attribut "Chosen Link Prediction Algorithm", in welchem ersichtlich ist, mit welchem Algorithmus die Kante hinzugefügt

## 5.2. GUI Konzept

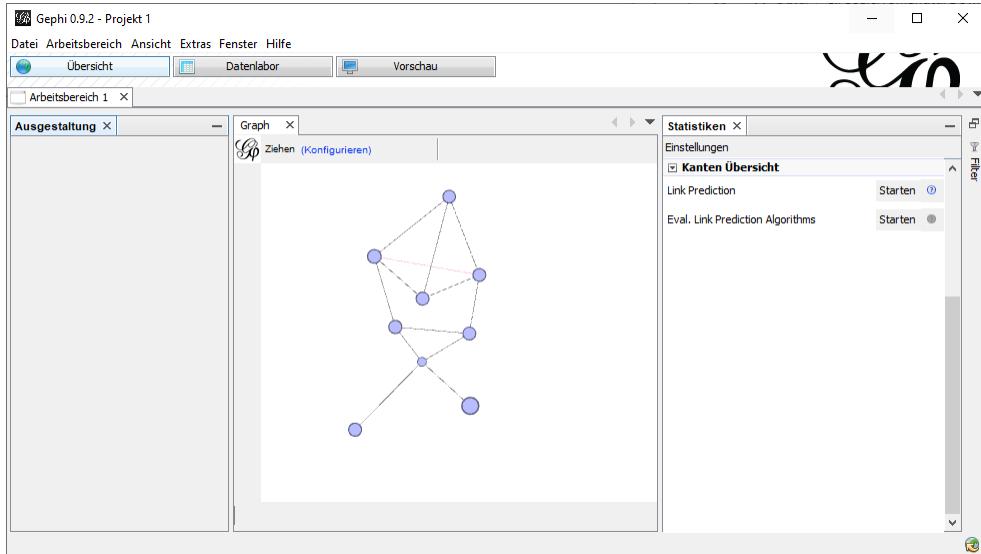


Abbildung 5.6: Modifizierter Graph mit vorhergesagtem Edge.

The screenshot shows the Gephi 0.9.2 software interface with the "Datenlabor" tab selected. The main window is titled "Gephi 0.9.2 - Projekt 1". The menu bar and toolbar are identical to the previous screenshot. The central workspace is titled "Datentabelle" and displays a table of edges. The table has columns: Ursprung, Ziel, Typ, Id, Label, Interval, Weight, Chosen Link Prediction Algorithm, Added in Run, and Last Link Prediction Value. The data in the table is as follows:

Ursprung	Ziel	Typ	Id	Label	Interval	Weight	Chosen Link Prediction Algorithm	Added in Run	Last Link Prediction Value
6	12	Ungerichtet	91			1.0	Prefential Attachment	1	12
7	12	Ungerichtet	108			1.0	Prefential Attachment	2	15
8	12	Ungerichtet	123			1.0	Prefential Attachment	3	18
6	7	Ungerichtet	66			1.0	Common Neighbours	1	2
6	11	Ungerichtet	100			1.0	Common Neighbours	2	2
7	10	Ungerichtet	116			1.0	Common Neighbours	3	2
6	8	Ungerichtet	5			1.0		0	0
8	7	Ungerichtet	6			1.0		0	0
8	9	Ungerichtet	7			1.0		0	0
6	9	Ungerichtet	8			1.0		0	0
9	7	Ungerichtet	9			1.0		0	0
6	10	Ungerichtet	10			1.0		0	0
10	11	Ungerichtet	11			1.0		0	0
11	7	Ungerichtet	12			1.0		0	0

Abbildung 5.7: Data Laboratory mit neuen Edges.

wurde. Falls das Attribut keinen Wert enthält, war die Kante bereits im initialen Graphen vorhanden.

Ein weiteres Attribut ist “Added in Run”. Gibt der User bei der Berechnung den Wert 3 mit, so werden drei Kanten pro ausgewähltem Algorithmus nacheinander hinzugefügt. Damit nachträglich ersichtlich ist, in welchem Durchlauf eine Kante hinzugefügt wurde, wird der Iterationsdurchgang in diesem Attributpersistiert. Ein Wert von 1 bedeutet demnach, dass die Kante in der ersten Iteration hinzugefügt wurde, während ein Wert von 3 bedeutet, dass die Kante im dritten Durchlauf zum Graphen hinzugefügt wurde. Kanten, die bereits im initialen Graphen vorhanden waren, enthalten keinen Wert in dieser Spalte.

Bei der dritten zusätzlichen Spalte “Last Value” wird der letzte berechnete Wert einer neu hinzugefügten Kante eingetragen. Damit kann nachträglich einfach nachvollzogen werden, aufgrund welcher Wahrscheinlichkeit eine Kante zum Graphen hinzugefügt wurde. Bei Kanten, die im initialen Graphen bereits vorhanden waren, ist dieses Attribut wiederum leer.

### 5.2.2 Filtern vorhergesagter Kanten

Die neu hinzugefügten Kanten sollen gefiltert werden können. Der folgende Abschnitt veranschaulicht, wie die Funktionalität im Gephi Plugin visualisiert wird.

In Abbildung 5.8 auf der nächsten Seite ist zu sehen, wie der Link-Prediction-Filter in die bestehende Übersicht eingebunden wird. Unter der Schaltfläche **Filter** gibt es eine neue Kategorie “Link Predictions”. Darin werden die Filter für die verschiedenen Algorithmen aufgeführt. Mithilfe eines grafischen Elements kann konfiguriert werden, welche Kanten des jeweiligen Algorithmus angezeigt werden sollen. Durch Positionieren des Sliders können alle Kanten ausgewählt werden, welche in einem gewünschten Iterations-Range zum Graphen hinzugefügt wurden. Der Filter wird durch Anwählen der Schaltfläche **Start** angewendet. Der gefilterte Graph enthält alle Kanten, welche mit dem gewählten Algorithmus hinzugefügt wurden und innerhalb der konfigurierten Iterationswerte liegen, sowie alle initialen Kanten.

Falls für den spezifischen Algorithmus noch keine Kanten zum Graphen hinzugefügt wurden, erscheint ein Pop-up Fenster mit entsprechendem Hinweis. Der Platzhalter “Link Prediction Filter” wird durch den gewählten Algorithmus, z.B. “Common Neighbour Filter” ersetzt. Ursprünglich war angelehnt, dass im Fehlerfall die Berechnung direkt aus dem Pop-up Fenster angestoßen werden kann. Um die Trennung zwischen den beiden Konzepten *Filter* und *Statistik* klar zu halten, wurde aber schliesslich darauf verzichtet. Die notwendige Berechnung kann weiterhin über **Statistics** vorgenommen werden (vgl. vorheriger Abschnitt).

## 5.2. GUI Konzept

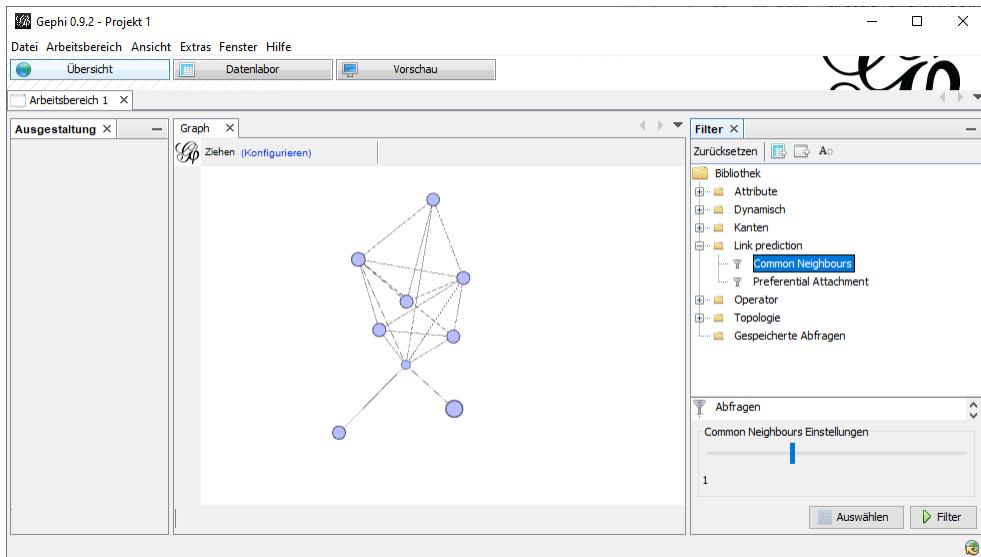


Abbildung 5.8: Überblick der Ansicht Filter.

### 5.2.3 Beurteilung der Qualität verschiedener Algorithmen

Bei der Implementierung verschiedener Algorithmen ist es für den User interessant zu wissen, welche Algorithmen am besten auf verschiedene Graphen angewendet werden können. Eine solche Aussage kann nur für zwei Graphen, die zwei verschiedene Zeitpunkte desselben Netzwerkes abbilden, gemacht werden. Ausgangspunkt für die Evaluation sind daher zwei Graphen, wie in Abbildung 5.9 dargestellt.

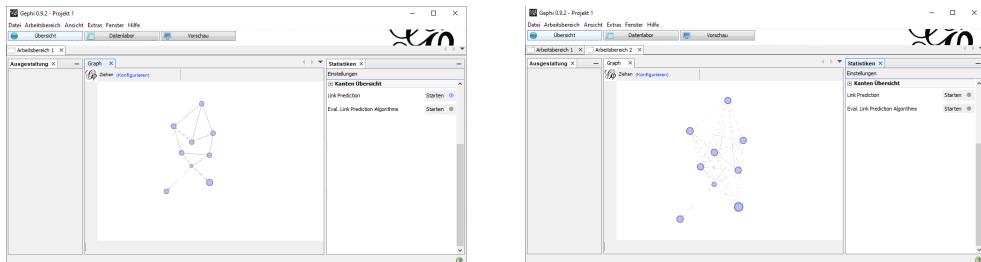
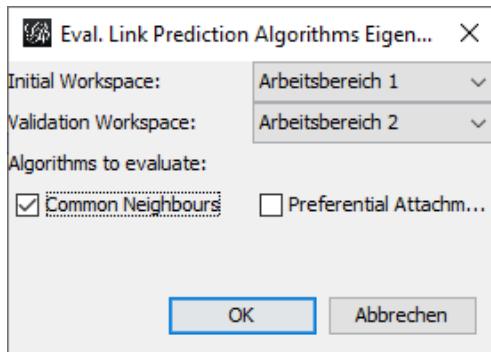


Abbildung 5.9: Auszuwertender Graph zum Zeitpunkt  $t$  und  $t + n$ .

Der Graph auf der linken Seite ist dabei ein Netzwerk zum Zeitpunkt  $t$  und entspricht dem initialen Graphen. Der Graph auf der rechten Seite zeigt das selbe Netzwerk zum Zeitpunkt  $t + n$  und entspricht dem Validierungsgraphen. Beim Anwählen der Schaltfläche "Link Prediction Evaluation" öffnet sich ein Pop-up Fenster, in welchem die beiden Graphen und die zu berechnenden Algorithmen ausgewählt werden können. Der initiale Graph wird dann für jeden Algorithmus dupliziert und die Berechnung auf dem jeweili-



**Abbildung 5.10:** Konfiguration für Evaluation der Algorithmen

gen Duplikat durchgeführt. Währenddessen werden die tatsächlich vorhandenen neuen Kanten aus dem Validierungsgraphen mit den hinzugefügten Kanten auf dem initialen Graphen verglichen. So wird ein Genauigkeitswert für das Ergebnis errechnet. Abbildung 5.10 zeigt das Pop-up für die Konfiguration.

Die berechneten Genauigkeiten werden anschliessend in Form eines Reports ausgewiesen. Dabei werden einerseits die erreichten Accuracy-Werte in absteigender Reihenfolge dargestellt, andererseits ist aber auch die Entwicklung der Accuracy für die einzelnen Iterationen dargestellt. Abbildung 5.11 zeigt den Entwurf eines solchen Reports.

## 5.3 Anleitung Software

Eine Anleitung der Software befindet sich im Anhang A.3 in Form des README.md.

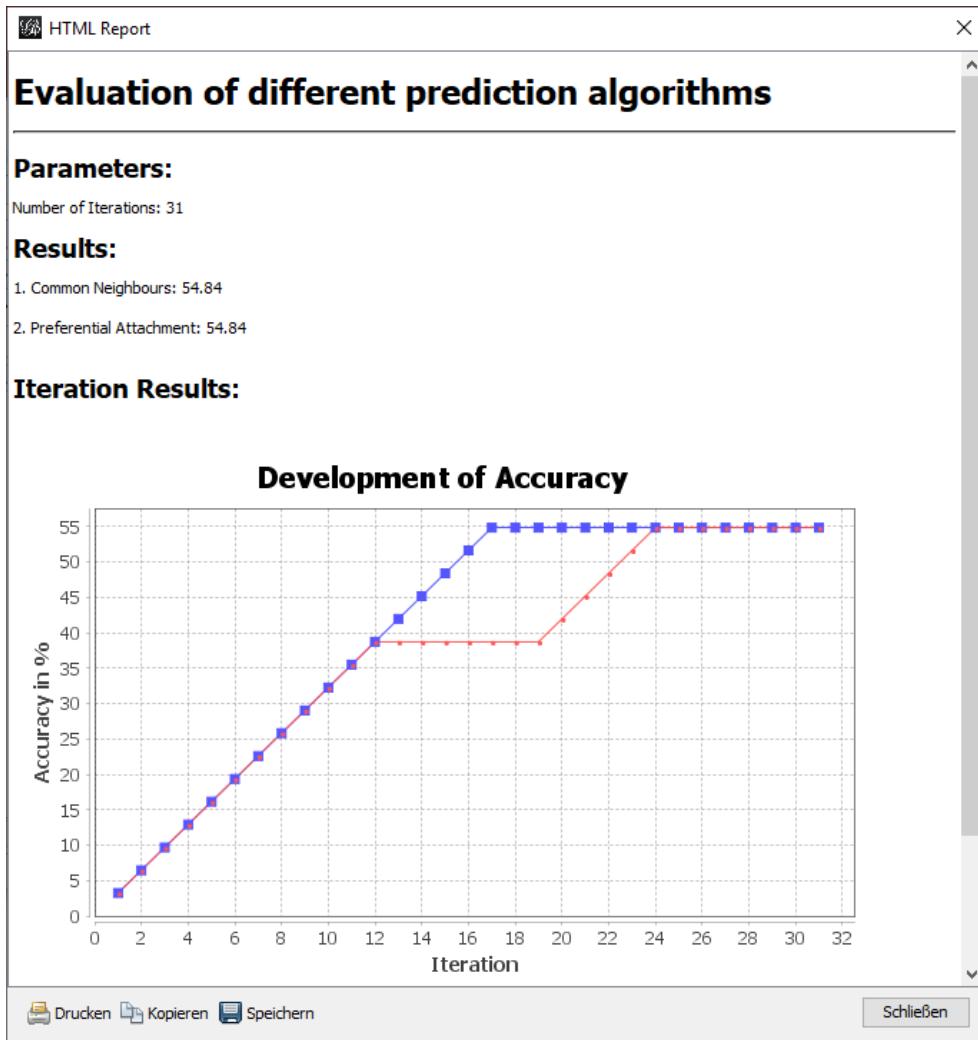


Abbildung 5.11: Report der Evaluationsresultate

## Kapitel 6

---

# Diskussion und Ausblick

---

Das Erarbeiten des Link-Prediction-Plugins war eine herausfordernde und zugleich lehrreiche Arbeit. Mit dem Projekt einher ging die Möglichkeit, im Open-Source-Umfeld zu einer etablierten Software beizutragen. Insbesondere das modulübergreifende Anwenden von Wissen aus dem Studiengang und der Austausch im Team und mit dem Betreuer bereiteten grosse Freude. In den folgenden Kapiteln werden die Resultate reflektiert und ein Ausblick auf nachfolgende Tätigkeiten dargeboten.

### 6.1 Erreichtes

Das Ziel, ein Link-Prediction-Plugin zu entwickeln, ist erreicht worden. Der Pull-Request des Plugins wurde fristgerecht gestellt. Bis zur Fertigstellung der Arbeit wurde der Pull-Request allerdings weder genehmigt, noch kommentiert. Neben den erforderlichen Funktionen zur Vorhersage von Links und zur Evaluation der verschiedenen Algorithmen wurde zusätzlich ein Filter zum Selektieren der Kanten pro Algorithmus entwickelt. Die verwendete Architektur erlaubt es, einfach neue Link-Prediction-Algorithmen und Kennzahlen zur Beurteilung der Qualität der Algorithmen hinzuzufügen. Als Reaktion auf die anfänglich langen Laufzeiten bei grossen Graphen wurde die Berechnung der Link Predictions optimiert: Für die Berechnung der Link-Prediction-Werte müssen initial für sämtliche Knoten alle Knoten durchlaufen werden. Indem beim Hinzufügen einer neuen Kante nur die noch möglichen Kanten der beiden veränderten Knoten neu berechnet werden, kann der Aufwand für alle weiteren Iterationen drastisch reduziert werden.

Tests mit der aktuellen Implementierung haben gezeigt, dass Link Predictions in Graphen mit bis zu 1000 Knoten problemlos und schnell berechnet werden können. Das Berechnen in grösseren Graphen kann in längeren

Laufzeiten resultieren. Ein entsprechender Hinweis wurde im README.md vermerkt.

## 6.2 Ausblick

Trotz diverser Optimierungsbemühungen muss beim initialen Berechnen der Prediction-Werte die Knotenzahl quadratisch durchlaufen werden. Dies erfordert bei grossen Netzwerken seine Zeit. Um auch bei solchen Netzwerken eine schnellere Performance zu erreichen, müssten weitere Optimierungsmassnahmen in Betracht gezogen werden. Folgende Möglichkeiten würden sich dazu anbieten:

- Anstatt, dass die Vorhersagen für alle Knoten berechnet werden, könnte man im UI die Möglichkeit anbieten, ein Subset bestehend aus spezifischen Knoten auszuwählen. Damit würde die Berechnung nur für alle relevanten Knoten durchgeführt.
- Falls ein Clustering zur Verfügung steht (entweder über ein Knoten-Attribut oder über eine berechnete Messgröße), könnte man die Berechnung auf alle Knoten dieses Clusters beschränken.
- Die Berechnung der verschiedenen Algorithmen könnte parallelisiert durchgeführt werden. Aktuell wird der Graph während der Berechnung mit einem Write-Lock geschützt. Dieser müsste für die Parallelisierung unter Berücksichtigung allfälliger Nebeneffekte aufgehoben werden.

Aktuell werden vom Plugin nur ungerichtete und ungewichtete Graphen unterstützt. Falls die Algorithmen auf gerichtete Graphen angewendet werden, werden ausgehende und eingehende Kanten zwischen denselben zwei Knoten als eine Kante betrachtet: Ein Graph mit einer Kante von Knoten A nach Knoten C und einer Kante von Knoten C nach Knoten A würde von den Algorithmen als eine Kante zwischen den beiden Knoten interpretiert. In künftigen Releases könnte zusätzlich die Funktionalität angeboten werden, ungerichtete und gerichtete Graphen unterschiedlich zu behandeln.

Weiter könnten auch Kantengewichte und - sofern vorhanden - andere Attribute mit in die Berechnungen einbezogen werden. Aktuell werden diese ignoriert und neue Kanten nicht gewichtet. Mögliche Ansätze zu Algorithmen mit gerichteten Graphen zeigen Garcia-Gasulla and Cortés (2014, S. 5).

Darüber hinaus beschränkte sich das User-Feedback auf den Auftraggeber. Über die Facebook-Gruppe von Gephi wurde das Plugin zwar verbreitet, doch auf die Bitte um Feedback ist keines der Gruppenmitglieder eingegangen. Falls das Plugin ergänzt oder optimiert werden sollte, könnte weiteres Feedback Hinweise auf Verbesserungsmöglichkeiten liefern.

---

# Abbildungsverzeichnis

---

2.1	Ungerichteter Graph . . . . .	4
2.2	Gerichteter Graph . . . . .	4
4.1	Beispielgraph (in Anlehnung an Henninger 2018, S.97) . . . . .	12
4.2	Initialer Graph zum Zeitpunkt $t$ , resp. $t + n$ . . . . .	16
4.3	Validierungsgraph zum Zeitpunkt $t + n$ . . . . .	17
4.4	Accuracy im Hypertext Network . . . . .	18
4.5	Histogramm der errechneten Werte im Hypertext Network . . .	19
5.1	Klassendiagramm Statistiken . . . . .	21
5.2	Klassendiagramm Evaluationskomponente . . . . .	23
5.3	Klassendiagramm Filter . . . . .	24
5.4	Überblick der Ansicht Statistik. . . . .	27
5.5	Pop-up für die Auswahl der Berechnung. . . . .	28
5.6	Modifizierter Graph mit vorhergesagtem Edge. . . . .	29
5.7	Data Laboratory mit neuen Edges. . . . .	29
5.8	Überblick der Ansicht Filter. . . . .	31
5.9	Auszuwertender Graph zum Zeitpunkt $t$ und $t + n$ . . . . .	31
5.10	Konfiguration für Evaluation der Algorithmen . . . . .	32
5.11	Report der Evaluationsresultate . . . . .	33

---

## Literaturverzeichnis

---

- Gao, F., Musial, K., Cooper, C., and Tsoka, S. (2015). Link Prediction Methods and Their Accuracy for Different Social Networks and Network Metrics. *Scientific Programming*, 2015:1–13.
- Garcia-Gasulla, D. and Cortés, U. (2014). *Link Prediction in Very Large Directed Graphs*. Universitat Politècnica de Catalunya, Barcelona.
- Henninger, M. (2018). *Soziale Netzwerkanalyse*. Fachhochschule Nordwestschweiz, Brugg.
- Moll, M. (2018). *Datasets Gephi Wiki*. <https://github.com/gephi/gephi/wiki/Datasets>. (aufgerufen am 9.6.2019).
- Ottmann, T. and Widmayer, P. (2017). *Algorithmen und Datenstrukturen*. Springer Vieweg, Wiesbaden, 6. auflage edition.
- Ulrike Baumöl, H. I. (2019). *Soziale Netzwerkanalyse*. <http://www.enzyklopädie-der-wirtschaftsinformatik.de/lexikon/datenwissen/Wissensmanagement/Soziales-Netzwerk/Soziale-Netzwerkanalyse>. (aufgerufen am 4.6.2019).
- Wikipedia (2018a). *Abstrakte Fabrik*. [https://de.wikipedia.org/wiki/Abstrakte\\_Fabrik](https://de.wikipedia.org/wiki/Abstrakte_Fabrik). (aufgerufen am 30.7.2019).
- Wikipedia (2018b). *Schablonenmethode*. <https://de.wikipedia.org/wiki/Schablonenmethode>. (aufgerufen am 30.07.2019).
- Wikipedia (2019a). *Kompositum (Entwurfsmuster)*. [https://de.wikipedia.org/wiki/Kompositum\\_\(Entwurfsmuster\)](https://de.wikipedia.org/wiki/Kompositum_(Entwurfsmuster)). (aufgerufen am 30.7.2019).
- Wikipedia (2019b). *Plug-in*. <https://de.wikipedia.org/wiki/Plug-in>. (aufgerufen am 29.7.2019).
- Wikipedia (2019c). *Soziale Netzwerkanalyse*. [https://de.wikipedia.org/wiki/Soziale\\_Netzwerkanalyse](https://de.wikipedia.org/wiki/Soziale_Netzwerkanalyse). (aufgerufen am 4.6.2019).

---

## **Abkürzungsverzeichnis**

---

<b>ASP</b>	Average Shortest Path
<b>ACC</b>	Average Clustering Coefficient
<b>API</b>	Application Programming Interface
<b>CN</b>	Common Neighbours Algorithm
<b>FHNW</b>	Fachhochschule Nordwestschweiz
<b>GCC</b>	Global Clustering Coefficient
<b>GUI</b>	Graphical User Interface
<b>PA</b>	Preferential Attachment Algorithm
<b>POC</b>	Proof of Concept
<b>SNA</b>	Social Network Analysis
<b>UI</b>	User Interface

---

## Glossar

---

**Abstract Factory Pattern** Die abstrakte Fabrik (engl. abstract factory) ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung, das zur Kategorie der Erzeugungsmuster gehört. Es definiert eine Schnittstelle zur Erzeugung einer Familie von Objekten, wobei die konkreten Klassen der zu instanzierenden Objekte nicht näher festgelegt werden (vgl. Wikipedia 2018a, online).

**Average Shortest Path** Messgröße zur Charakterisierung eines Netzwerkes, entspricht der durchschnittlichen Länge des kürzesten Pfads zwischen zwei Knoten.

**Clique** Subgraph innerhalb eines Graphen, in welchem die Knoten untereinander stark verbunden sind.

**Cluster Coefficient** Messgröße zur Charakterisierung von Netzwerken, welche das Mass der Cliquenbildung in einem Graphen aufzeigt.

**Composite Command Pattern** Das Composite Pattern beruht darauf, in einer abstrakten Klasse sowohl konkrete Objekte als auch ihre Behälter zu repräsentieren. Damit können sowohl einzelne Objekte als auch ihre Kompositionen einheitlich behandelt werden (Wikipedia 2019a, online). Beim Composite Command Pattern fungiert die Kompositionsklasse als Makro-Klasse. Sie enthält eine Liste von Objekten, welche einzeln ausgeführt werden können.

**Data Laboratory** Knoten- und Kantentabellen eines Graphen in Gephi. Das Data Laboratory (dt. Datenlabor) kann über den gleichnamigen Button im Gephi-UI geöffnet werden.

**Density** Messgröße, welche verdeutlicht, wie komplett die Knoten des Netzwerks miteinander verbunden sind.

**Diameter** Messgröße, welche dem längsten kürzesten Pfad zwischen allen Knotenpaaren in einem Graphen entspricht.

**Gephi** Gephi ist eine Open-Source-Software zur explorativen Analyse von Graphen. Die Software ist in der Programmiersprache Java implementiert und modular aufgebaut. Für das Projekt wurde die Version 0.9.2 eingesetzt.

**Kante** Verbindung zwischen zwei Knoten, die die Interaktion zwischen den Knoten darstellt.

**Knoten** Akteur in einem Netzwerk.

**Pfad** Ein Pfad bezeichnet die Kanten, welche genutzt werden müssen, um von einem Knoten zu einem anderen zu gelangen.

**Plugin** Ein Plugin ist eine optionale Software-Komponente, die eine bestehende Software erweitert bzw. verändert (vgl. Wikipedia 2019b, online).

**Template Pattern** Das Template Pattern ist ein in der Softwareentwicklung eingesetztes Entwurfsmuster, mit dem Teilschritte eines Algorithmus variabel gehalten werden können während dabei eine Grundstruktur vorgegeben wird. Beim Template Pattern wird in einer abstrakten Klasse das Skelett eines Algorithmus definiert. Die konkrete Ausformung der einzelnen Schritte wird an Unterklassen delegiert. Dadurch besteht die Möglichkeit, einzelne Schritte des Algorithmus zu verändern oder zu überschreiben, ohne dass die zu Grunde liegende Struktur des Algorithmus modifiziert werden muss (Wikipedia 2018b, online).

**Temporales Netzwerk** Netzwerk, bei welchem die Historie der hinzugefügten Kanten über den Verlauf der Zeit ersichtlich ist. Zum Nachvollziehen, zu welchem Zeitpunkt ein Knoten oder eine Kante zum Graphen hinzugefügt wurde, bieten sich Zeitstempel oder eine Nummerierungsfolge an.

## Anhang A

---

# Anhang Software

---

### A.1 Proof of Concept

Für die Einarbeitung in die Gephi-Plugin Entwicklung und um erste Ansätze für die Umsetzung des Projekts zu finden, hat sich das Projektteam entschlossen, ein Proof of Concept durchzuführen.

#### Ziel

Mit dem Proof of Concept, soll sich einerseits mit dem Application Programming Interface (API) von Gephi vertraut gemacht werden. Andererseits soll die Funktionsweise von ersten Ideen und Konzepten überprüft werden.

#### Umsetzungsprozess

- **Filter:** Filter werden in Gephi verwendet, um ein Netzwerk auf Knoten oder Kanten zu reduzieren, welche bestimmte Eigenschaften besitzen. In Bezug auf Link Prediction wurde deshalb eine neue Filterkategorie eingeführt, unter welcher nach verschiedenen Prediction-Algorithmen gefiltert werden kann. Durch Auswahl eines Algorithmus können jene Kanten herausgefiltert werden, welche durch das Anwenden des Algorithmus neu hinzugefügt werden. Der Filter wird dabei um ein UI-Element ergänzt, bei welchem die Anzahl der hinzugefügten Kanten eingegrenzt werden kann.
- **Statistiken:** Die Wahrscheinlichkeit, dass sich eine neue Kante zwischen zwei Knoten bildet, entspricht einem berechneten Wert. Diese Berechnung wird mittels Statistiken angestoßen. Die gesamte Logik und Berechnung der verschiedenen Algorithmen wird anschliessend durch verschiedene Statistik-Objekte gekapselt.

Die Funktionsweise für das Implementieren von neuen Statistik-Objekten ist im Gephi-Plugin Bootcamp gut dokumentiert. Darauf aufbauend konnte ein neues Statistik-Objekt mit entsprechendem Button im

## A.1. Proof of Concept

---

UI implementiert werden. Bei der Berechnung wurde der Algorithmus "Common Neighbours" eingesetzt, bei welchem die Anzahl Nachbarn jedes Knotens gezählt werden.

Für die Implementierung des Algorithmus konnten bestehende Funktionen von Gephi verwendet werden um die Knoten aus dem Graphen, respektive dessen Nachbarn auszulesen. Die berechneten Werte konnten anschliessend bereits im Data Laboratory abgelegt werden.

Damit dem Graphen Kanten hinzugefügt werden können, während diese noch gelesen und überarbeitet werden, muss ein Write-Lock verwendet werden. Die Logik, welche für den Algorithmus "Common Neighbours" implementiert wurde, kann für die spätere Umsetzung grösstenteils wiederverwendet werden.

### Erkenntnisse

Es konnten verschiedene Erkenntnisse aus der Umsetzung des Proof Of Concepts gewonnen werden. Zum einen war es sehr hilfreich, sich zum ersten Mal aktiv mit dem API von Gephi zu beschäftigen. Besonders wichtig war dabei die Erkenntnis, dass die Beispiele aus dem Bootcamp teilweise veraltet sind. Um solche Beispiele mit einer neuen Gephi-Version und den dadurch unterschiedlichen APIs zum Laufen zu bringen, muss zusätzlich Aufwand eingerechnet werden.

Weiter konnte mithilfe des Proof Of Concepts sichergestellt werden, dass sämtliche Installationen, Konfigurationen und die Entwicklungsumgebung korrekt funktionieren. Viele konfigurative Probleme konnten hier bereits erkannt und frühzeitig behoben werden.

Vom technischen Aspekt her konnten wichtige Erkenntnisse für die spätere Umsetzung gewonnen werden - beispielsweise wie ein Gephi-Plugin aufgebaut ist. Diese Erkenntnisse fliessen direkt in die Überlegungen beim Entwurf der Software-Architektur ein.

## A.2. Wireframes

### A.2 Wireframes

**SC-1\_Ueberblick\_Statistik**

**SC-2\_PopUp**

Source	Target	LP Algorithm	Added in Run	Last Threshold (LP)
1	2	Common Neighbours	1	15
5	4	Preferential Attachment	3	0.3
7	8	Resource Allocation	2	0.7
3	2	Common Neighbours	0	2

**SC-3\_Graph\_nach\_Berechnung**

**SC-4\_Ueberblick\_Data\_Lab**

**SC-5\_Uebersicht\_Filter**

**SC-6\_Filter\_Fehler**

Error

Link Prediction Filter couldn't be used as there was no data calculated yet.

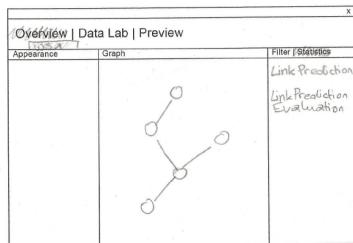
Do calculation now! Start  
Number of new Edges 3  
OK!

← optional

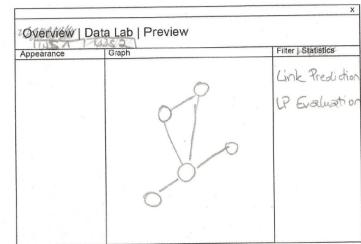
**SC-7\_Anwendung\_Filter**

## A.2. Wireframes

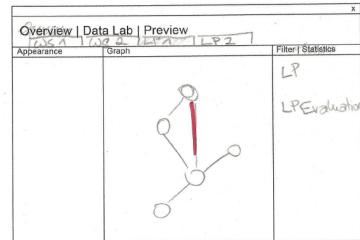
SC\_1\_Overview\_Graph1



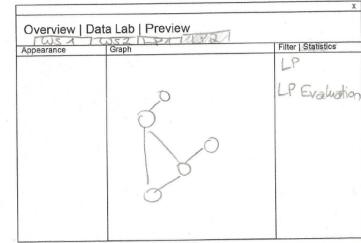
SC\_2\_Overview\_Graph2



SC\_3\_Overview\_LPGraph1



SC\_4\_Overview\_LPGraph2



SC\_5\_LPE\_Settings

Wireframe of SC\_5\_LPE\_Settings. It displays the 'Link Prediction Evaluation' settings. It shows 'Origin Graph' set to 'WSA' and 'End Graph' set to 'WS2'. Below this, it says 'Link Prediction Algorithms' and lists 'LPA 1' (selected), 'LPA 2', and 'LPA 3'. A 'Start' button is at the bottom.

SC\_6\_LPE\_Results

Wireframe of SC\_6\_LPE\_Results. It shows the 'Link Prediction Evaluation' results. It lists 'Ranking Accuracy' with two items: '1) LPA 1: 100% (1/1 Edges)' and '2) LPA 2: 0% (0/1 Edges)'. An 'OK' button is at the bottom.

SC\_7\_LPE\_Error1

Wireframe of SC\_7\_LPE\_Error1. It shows an error message: 'Only one workspace was found. Please open another:' with a 'Search' input field. Below are 'cancel' and 'open' buttons.

SC\_8\_LPE\_Error2

Wireframe of SC\_8\_LPE\_Error2. It shows an error message: 'Origin Graph is bigger than End Graph! Evaluation not possible!' with an arrow pointing to the text. It also lists 'Link Prediction Algorithms' and 'LPA 1' (selected). A note 'Kriterien müssen definiert werden' is on the right. A 'Start' button is at the bottom.

## A.2. Wireframes

---

# A.3 README.md

## Link Prediction

[Build passing](#) [License Apache 2.0](#)

Link-prediction plugin for Gephi, which allows to predict the next edges to be formed using different prediction algorithms. Edges that are added to the network based on the prediction are marked accordingly. Users can limit the number of edges predicted. The plugin contains an evaluation component, which allows to compare the quality of the different algorithms regarding the graph on hand.

The plugin is released under the Apache 2.0 license.

## Features

In release 1.2.0 the plugin contains the following functionality:

- **Statistics:** New edges can be added to an undirected graph using selected [link prediction algorithms](#) in the `statistics` tab. The number of new edges can be specified. In doing so,  $n$  new edges are added to the graph iteratively. The calculation of the next predicted edge is always based on the graph of the preceding iteration step.
- **Filter:** The added edges can be displayed by means of filters. On the one hand, the corresponding algorithm is specified as the filter criterion. On the other hand, the number of added edges can also be restricted.
- **Evaluation:** Based on an initial graph and a validation graph the accuracy of the link predictions using different algorithms are evaluated. Besides the final accuracy, the generated report also shows the accuracy after each iteration step.

## Get started

### Run Gephi with installed plugin

If you checked out the sources via Maven, you can run Gephi with your plugin pre-installed using the following command. Make sure to run `mvn package` beforehand to rebuild.

```
mvn org.gephi:gephi-maven-plugin:run
```

If you downloaded the plugin distribution files (\*.nbm) you can just navigate to `Tools > Plugins > Downloaded` and add the Plugin there.

### Predict new edges

To predict new edges, run a new link prediction using `Statistics > Edge Overview > Link Predictions`. The number of new edges can be specified. In doing so,  $n$  new edges are added to the graph iteratively. The calculation of the next predicted edge is always based on the graph of the preceding iteration step. Information to the newly added edges are visible in the following columns under `Data Laboratory > Edges`:

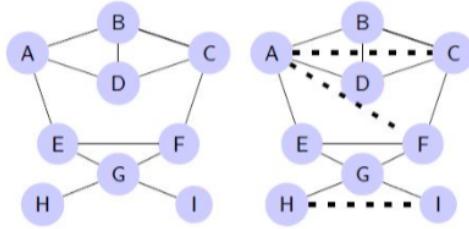
- **Chosen link prediction algorithm:** Algorithm that was used to predict the edge.
- **Added in run:** Iteration, in which the edge was added.
- **Last link prediction value:** Calculated link prediction value according to the used algorithm. The values are not normalized.

### Filter predictions

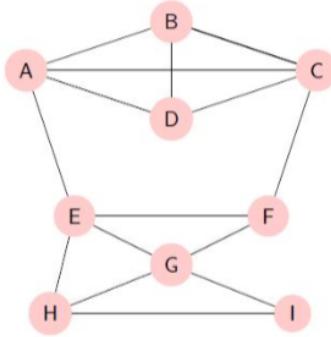
The filters under `Filters > Link Prediction` then allow you to narrow down the corresponding edges. Edges can be filtered according to the algorithms with which they were added. Furthermore, the number of added edges can also be restricted to the first  $n$  added edges.

### Evaluate prediction algorithms

The evaluation can be run using `Statistics > Edge Overview > Eval. Link Predictions Algorithms`. Starting with an initial graph  $G_{i,t}$  at time  $t$  we predict  $n$  edges  $E_i$  which results in a Graph  $G_{i,t+n}$  at time  $t+n$ . For each algorithm a new workspace is created which is used to apply the predictions. The initial and validation graph are therefore not changed. The following figure shows the graphs  $G_{i,t}$  (on the left) and  $G_{i,t+n}$  (on the right):



To evaluate their accuracy using a validation graph  $G_{v,t+n}$  and its additional edges  $E_v$  at time  $t+n$  are used. In comparison to the Graph  $G_{i,t}$  this graph additionally contains the edges  $(A, C)$ ,  $(E, H)$  and  $(H, I)$ :



The accuracy then is calculated as percentage of the correct predicted edges. In the current implementation, the results are rounded to two places.

$$Acc = |E_i \cap E_v| / |E_v| * 100$$

In the above example of three additional edges the edges  $(A, C)$  and  $(H, I)$  were predicted correctly. Therefore an accuracy of 66.67% is achieved:

$$E_i = \{(A, C), (A, F), (H, I)\}$$

$$E_v = \{(A, C), (E, H), (H, I)\}$$

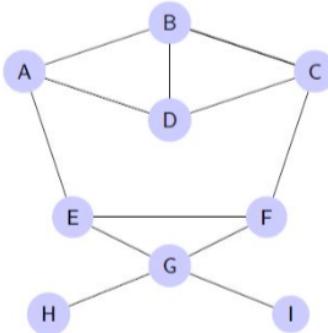
$$|E_i \cap E_v| = |\{(A, C), (H, I)\}| = 2$$

$$Acc = |E_i \cap E_v| / |E_v| * 100 = 2 / 3 * 100 = 66.67$$

## Algorithms

---

Link prediction is based on an existing network and attempts to predict new edges. The most popular application is the suggestion of new friends on social networking platforms. To predict a new edge, different algorithms exist. The plugin allows to easily add new algorithms. Currently, the algorithms [common neighbours](#) and [preferential attachment](#) are implemented. To show the functionality of the algorithms, the following example graph is used:



#### Common neighbours

Common neighbours calculates for two unconnected nodes how many common neighbours exist. The higher the calculated value, the more likely a new edge will be added between the two nodes.

The following formula represents, how the number of common neighbours of two nodes  $X$  and  $Y$  can be calculated. The call to the function  $N(:)$  returns all neighbours of a node in a set, e.g.  $N(A)$  returns all neighbour nodes of node  $A$ .

$$cn(X, Y) = |N(X) \cap N(Y)|$$

Applied to the above example graph, the common neighbour algorithm would predict a new edge between nodes  $A$  and  $C$ . The following examples show some of the calculated values:

$$cn(A, C) = |B, D| = 2$$

$$cn(A, F) = |E| = 1$$

$$cn(A, I) = |\{\}| = 0$$

The current implementation of the algorithm has a complexity class  $O(n^2)$ .

#### Preferential attachment

The basic assumption with Preferential Attachment is that the probability that a node is affected by a newly added edge, is just proportional to the number of neighbours. The more neighbours a node has, the larger the likelihood that it will be affected.

To calculate preferential attachment, the number of neighbours of both nodes are multiplied by each other. The call to the function  $N(:)$  returns again all neighbours of a node in a set, e.g.  $N(A)$  returns all neighbours of node  $A$ .

$$pa(X, Y) = |N(X)| * |N(Y)|$$

Applied to the above example graph, preferential attachment would predict an edge between node  $A$  and  $G$  and calculate the following values:

$$pa(A, G) = 3 * 4 = 12$$

$$pa(A, C) = 3 * 3 = 9$$

The implementation of the preferential attachment algorithm is of complexity class  $O(n^2)$ .

#### Limitations

---

With the limitations of the implemented algorithms, only undirected, unweighted graphs are supported currently.

The plugin was tested using graphs with less than a thousand nodes. Link predictions in larger networks can lead to long runtimes.

## Anhang B

---

# Weiteres

---

### B.1 Projektanforderungen

Das Projekt wird im Rahmen des IP5 an der FHNW durchgeführt. Sein Hauptziel ist es, ein Plugin für das Datenanalyse-Tool Gephi zu entwickeln, mithilfe dessen Vorhersagen möglicher neuer Kanten zwischen Knoten in einem Netzwerk durchgeführt werden können. Solche Vorhersagen neuer Verbindungen zwischen zwei Knoten werden Link Predictions genannt. Das Projekt wird von Michael Henninger betreut und wurde auch von ihm in Auftrag gegeben. Die genauen Vorgaben für die Umsetzung lauten, wie folgt:

- Von den verschiedenen, existierenden Link-Prediction-Algorithmen soll mehr als einer ausgewählt und in das Plugin eingebaut werden.
- Da das Plugin am Schluss als Open-Source-Projekt zur Verfügung gestellt werden soll, muss es umfassend dokumentiert, gut getestet und bezüglich der Algorithmen leicht erweiterbar gestaltet werden.
- Es muss eine geeignete Visualisierung und Datenstruktur gefunden werden, um die vorausgesagten Kanten in einem Netzwerk darzustellen. Dazu müssen mindestens simple Entwürfe skizziert und abgenommen werden.
- Darüber hinaus soll es die Möglichkeit geben, zwei Netzwerke zu verschiedenen Zeitpunkten zu vergleichen und zu berechnen, mit welchem der implementierten Link-Prediction-Algorithmen die beste Vorhersage gemacht worden wäre. Auch hierfür ist es essentiell, eine geeignete Visualisierung und Datenstruktur zu bestimmen.
- Die vorhergesagten Kanten sollen speziell markiert werden. So können diese beispielsweise eine neue Spalte mit Schwellwerten/Wahrscheinlichkeiten erhalten. Kanten, die es dabei schon von Anfang an gegeben hat, bekommen dabei einen Default-Wert wie 0 oder NULL.

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

Link-Prediction Plugin für Gephi

**Authored by** (in block letters):

*For papers written by groups the names of all authors are required.*

Name(s):

Romanutti

First name(s):

Marco

Schüler

Saskia

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Leitfaden Berichte 4.01' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Brugg, 16.8.2019

Signature(s)

---

*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*