

















































## 5.1. Software Architektur

---

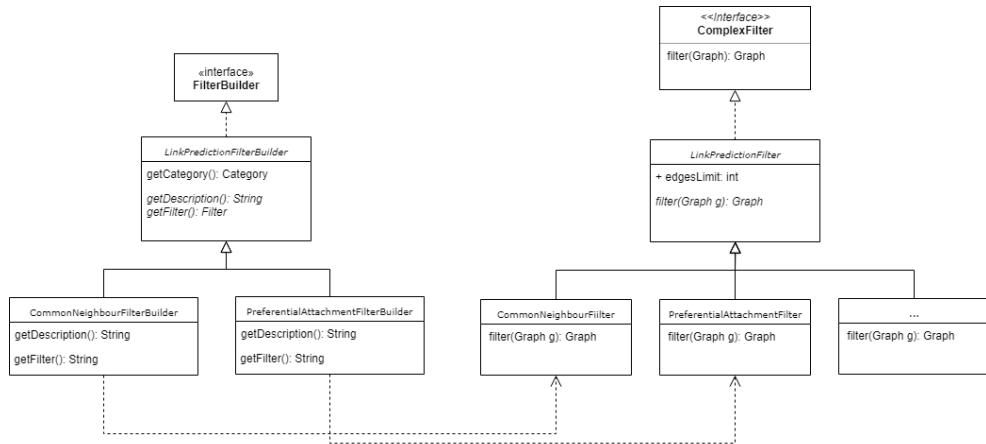


Abbildung 5.3: Klassendiagramm Filter

werden. Für jede Klasse wird ein separater Logger erzeugt. Das Codelisting ?? 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 ?? 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 ?? zeigt zwei solcher Beispiele.

Listing 5.3: Log-Einträge erzeugen

---

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

---

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

```
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

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.PreferentialAttachmentStatistics"
                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 Situationen 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 hingegen 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. Bei der Analyse der bestehenden ui!-Elemente stellten sich die folgenden Punkte heraus:

- Unter dem Menü-Punkt `Statistics` werden diverse Werte des Graphen berechnet und gegebenenfalls im Data Laboratory 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 für gewöhnlich verkleinert dargestellt, also mit weniger Kanten oder Knoten.
- Die Darstellung der verschiedenen Werte wird über den Menü-Punkt `Appearance` vorgenommen. Hier können Farben, und Grösse der Knoten und Kanten eingestellt werden. Beim Filtern werden keine neuen Werte berechnet oder generiert.
- 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.

Das gui!-Konzept des Link Prediction Plugins basiert auf diesen Grundkonzepten.

### 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

## 5.2. GUI Konzept

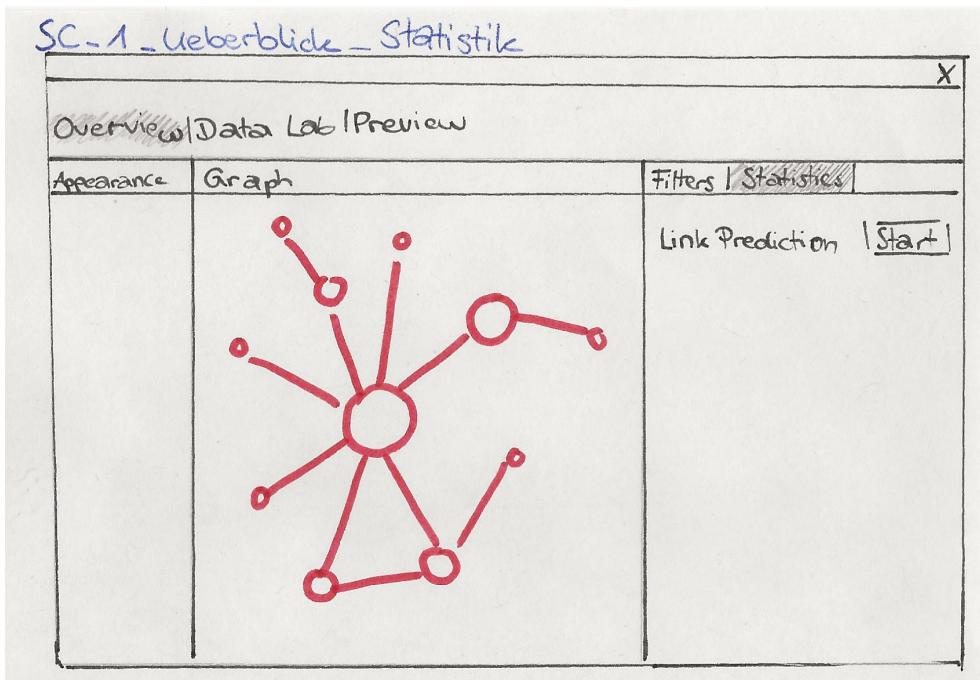


Abbildung 5.4: Überblick der Ansicht Statistik.

werden. Im Folgenden wird dieser Vorgang beschrieben:

Abbildung ?? zeigt einen Überblick über die Benutzeroberfläche von Gephi. Grau 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 ?? zeigt das Pop-up für die Auswahl der Berechnung. 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  $\geq 1$  eingeben. Dieser besagt, wie viele Durchgänge bei der Link Prediction gemacht 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.

**Filter:** Die hinzugefügten Kanten können via Filter angezeigt werden. Dabei können die anzuzeigenden Kanten einerseits auf einen Algorithmus, anderer-

## 5.2. GUI Konzept

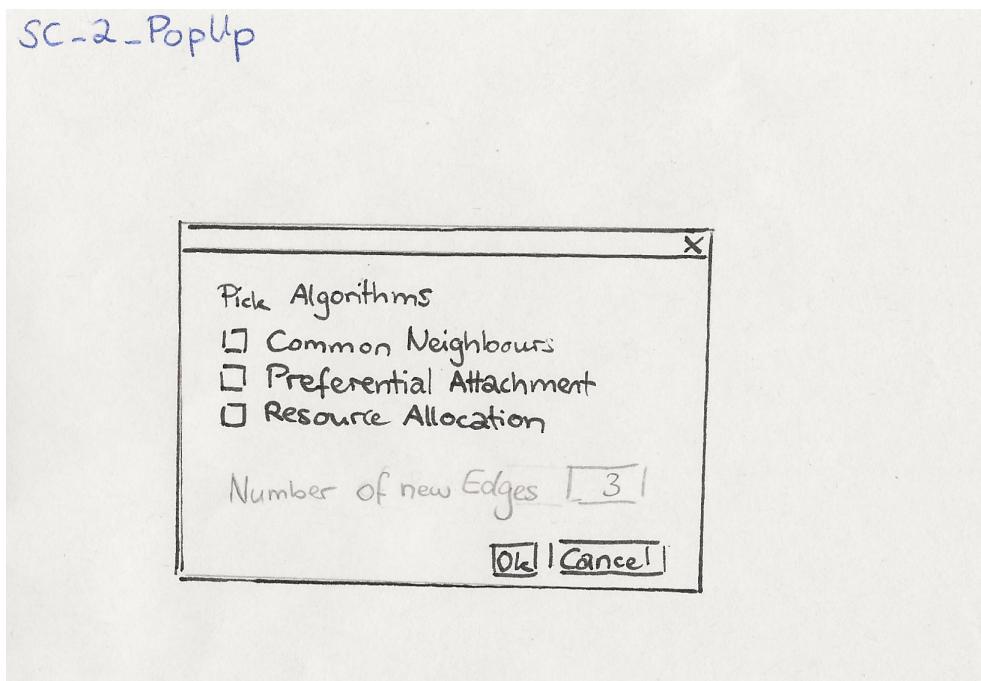


Abbildung 5.5: Pop-Up für die Auswahl der Berechnung.

seits 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 soll eine Warnung an den User ausgegeben werden. Wird diese bestätigt, so wird der Algorithmus trotzdem berechnet. Beim Abbruch kann der User seine gewünschten Parameter nochmals verändern.

Abbildung ?? zeigt den Graph nach der Berechnung 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.

Abbildung ?? zeigt die neuen Kanten im **Data Laboratory**. Wie bereits erwähnt, wurden den Kanten neue Attribute hinzugefügt. Neu gibt es ein Attribut "Link Prediction Algorithm", in welchem ersichtlich ist, mit welchem Algorithmus die Kante hinzugefügt wurde. Falls das Attribut keinen Wert enthält, war die Spalte bereits im initialen Graphen vorhanden.

Eine weiteres Attribut ist "Added in Run". Gibt der User bei der Berechnung den Wert 3 mit, so werden hier 3 Kanten pro ausgewähltem Algorithmus nacheinander hinzugefügt. Damit nachträglich erstlichich ist, in welchem

## 5.2. GUI Konzept

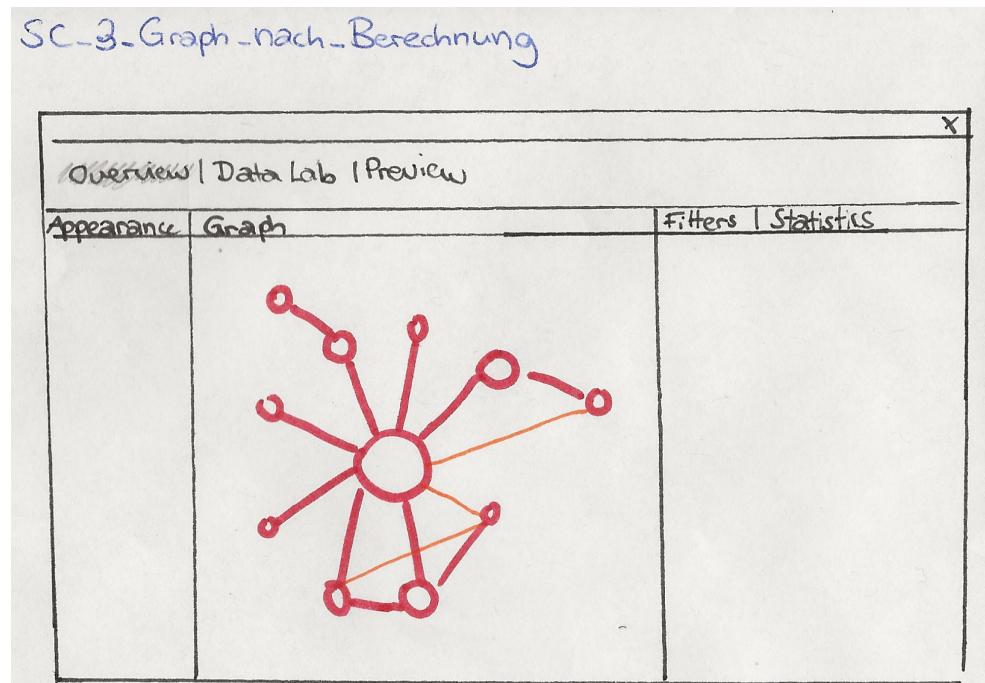


Abbildung 5.6: Modifizierter Graph mit vorhergesagtem Edge.

SC-4-Ueberblick-Data-Lab

Overview   Data Lab   Preview					
Source	Target	LPA Algorithm	Added in Run	Last	Value
1	2	Common Neighbours	1	25	
5	4	Preferential Attachment	3	18	
7	8	Resource Allocation	2	0.7	
3	2	Common Neighbours	0	2	

Abbildung 5.7: Data Laboratory mit neuen Edges.

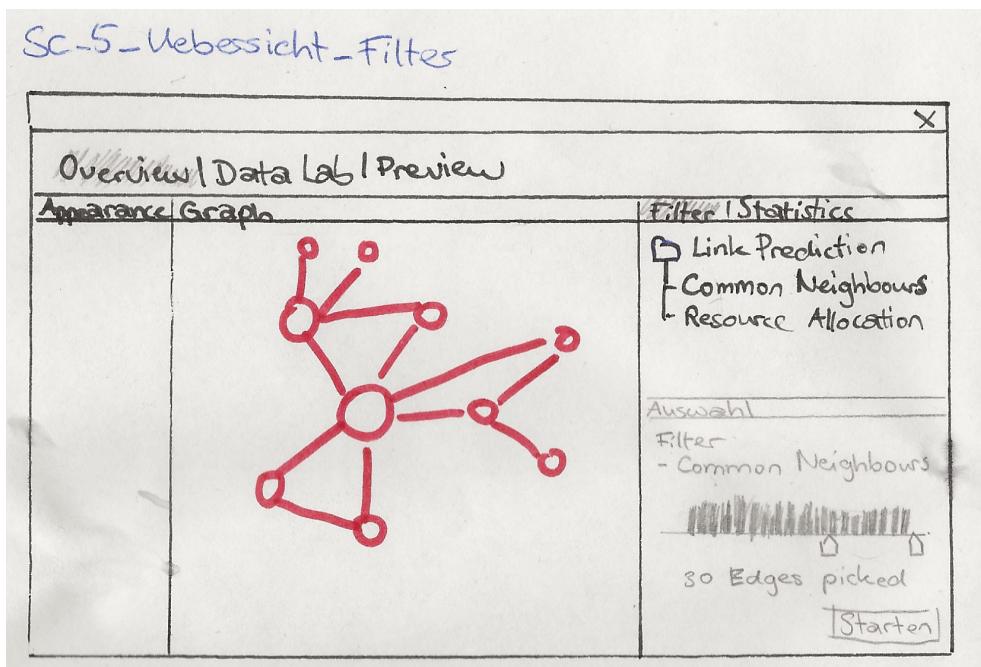


Abbildung 5.8: Überblick der Ansicht Filter.

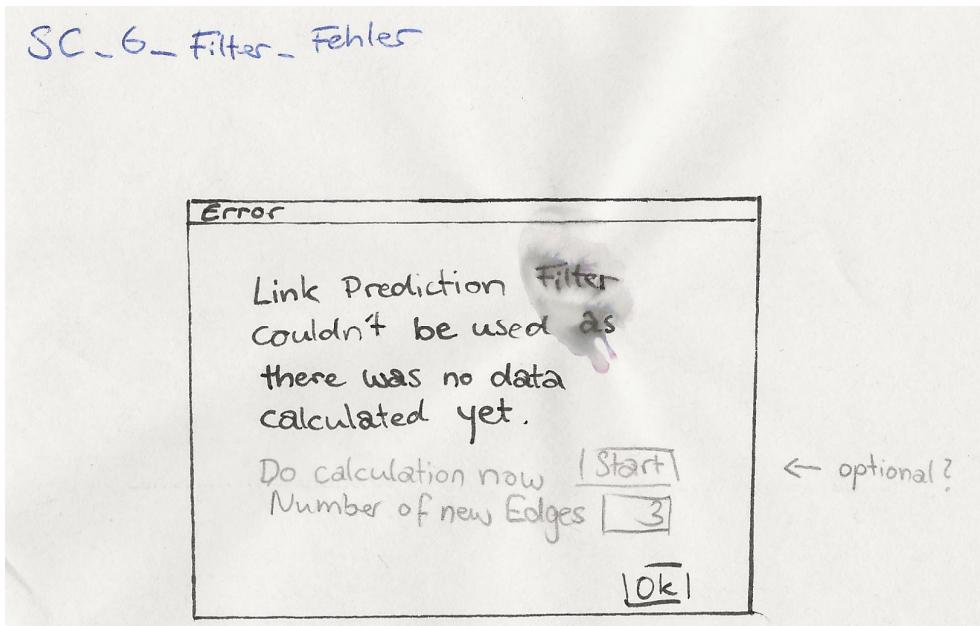
Durchlauf eine Kante hinzugefügt wurde, wird der Iterationsdurchgang in diesem Attribut persisiert. Ein Wert von 1 bedeutet demnach, dass die Kante in der ersten Iteration hinzugefügt wurde, während ein Wert von 3 bedeutet, dass die Kante im dritten Durchlauf zum Graphen hinzugefügt wurde. Kanten, die 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 Spalte 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 ?? 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 wer-



**Abbildung 5.9:** Fehlermeldung, wenn die Berechnung der Link Prediction noch nicht durchgeführt wurde.

den sollen. Durch Positionieren des Sliders können alle Kanten ausgewählt werden, welche in einem gewünschten Iterationrange zum Graphen hinzugefügt wurden. Der Filter wird durch Anwählen der Schaltfläche **Start** angewendet.

Falls für den spezifischen Algorithmus noch keine Kanten zum Graphen hinzugefügt wurden, erscheint ein Pop-up Fenster mit entsprechendem Hinweis. Abbildung ?? zeigt das Pop-up Fenster.

Der Platzhalter "Link Prediction Filter" wird durch den gewählten Algorithmus, z.B. "Common Neighbour Filter" ersetzt. Ursprünglich war angedacht, dass im Fehlerfall die Berechnung direkt aus dem Pop-up Fenster angestossen werden soll. Um die Trennung zwischen den beiden Konzepten *Filter* und *Statistik* klar zu halten, wurde darauf verzichtet. Die notwendige Berechnung kann weiterhin über **Statistics** vorgenommen werden (vgl. voriger Abschnitt).

Abbildung ?? zeigt den gefilterten Graph. Er enthält alle Kanten, welche mit dem gewählten Algorithmus hinzugefügt wurden und innerhalb der konfigurierten Iterationswerte liegen.

### 5.2.3 Beurteilung der Qualität verschiedener Algorithmen

Dummy text.

### 5.3. Anleitung Software

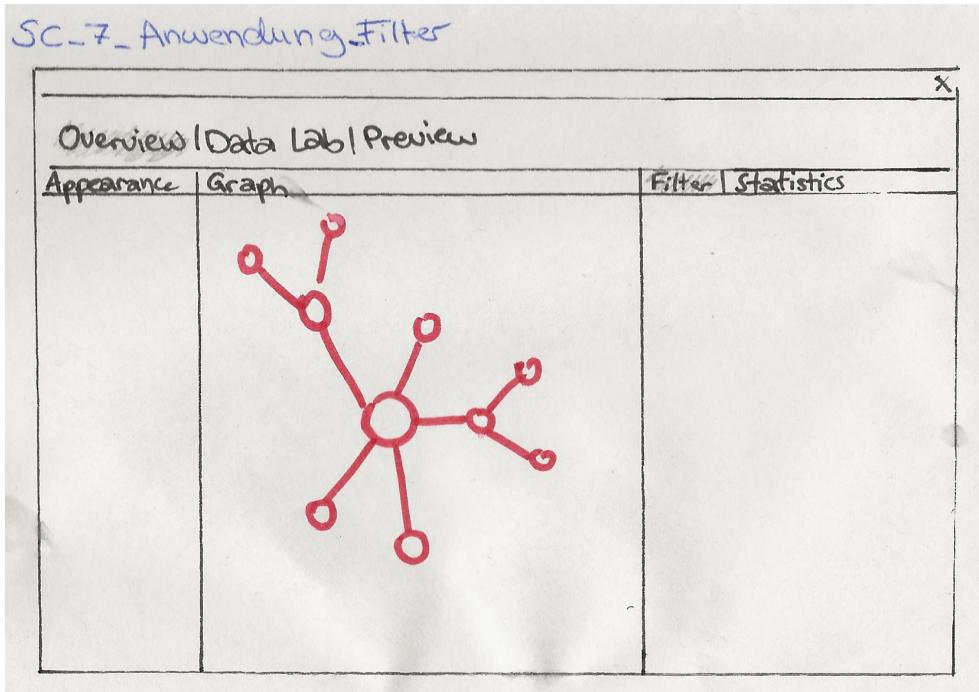


Abbildung 5.10: Gefilterter Graph.

## 5.3 Anleitung Software

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

## 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 bereitete 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. Neben den erforderlichen Funktionen zur Vorhersage von Links und zur Evaluation der verschiedenen Algorithmen wurde zusätzlich ein Filter zum selektieren der Kanten je 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 sämtliche Knoten initial zweimal durchlaufen werden. Indem beim Hinzufügen einer neuen Kante nur die beiden davon betroffenen Kanten geändert werden, kann der Aufwand für alle weiteren Iterationen drastisch reduziert werden.

Tests mit der aktuellen Implementierung haben gezeigt, das Link-Predictions in Graphen mit bis zu 1000 Knoten problemlos 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 müssen beim initialien Berechnen der Prediction-Werte alle Knoten zweimal 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 ausgehende und eingehende Kanten zwischen denselben zwei Nodes 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 mit in die Berechnungen einbezogen werden. Aktuell werden diese ignoriert und neue Kanten nicht gewichtet.

---

## **Abbildungsverzeichnis**

---

---

## **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 (englisch abstract factory, kit) ist ein Entwurfsmuster aus dem Bereich der Softwareentwicklung, das zur Kategorie der Erzeugungsmuster (englisch creational patterns) gehört. Es definiert eine Schnittstelle zur Erzeugung einer Familie von Objekten, wobei die konkreten Klassen der zu instanzierenden Objekte nicht näher festgelegt werden (vgl. ?, ?, online).

**Average Shortest Path** Messgrösse zur Charakterisierung eines Netzwerkes, entsprich 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 (? ?, 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.

## Abbildungsverzeichnis

---

**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.

**Plugin** Ein Plugin ist eine optionale Software-Komponente, die eine bestehende Software erweitert bzw. verändert (vgl. ?, ?, 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 Schablonenmethoden-Entwurfsmuster 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 (?, ?, online).

## Anhang A

---

# Anhang Software

---

### A.1 Proof of Concept

Für die Einarbeitung in die Gephi-Plugin Entwicklung und 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 **api!** (`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 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 **ui!**

## A.1. Proof of Concept

---

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 `datalaboratory!` 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 Concept 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 `api`s 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.1. Proof of Concept

---

# A.2 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` :

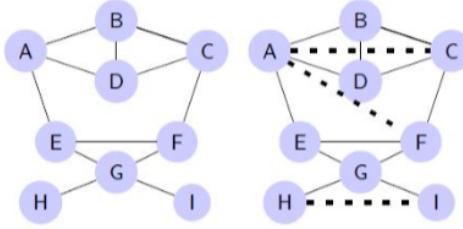
- **Chosen link prediction algorithm:** Algorithm that was used to predict the edge.
- **Added in run:** Iteration, in which the edge was added.
- **Last link prediction value:** Calculated link prediction value according to the used algorithm. The values are not normalized.

### Filter predictions

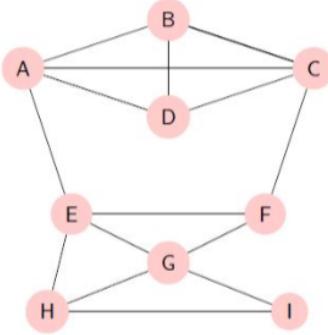
The filters under `Filters > Link Prediction` then allow you to narrow down the corresponding edges. Edges can be filtered according to the algorithms with which they were added. Furthermore, the number of added edges can also be restricted to the first  $n$  added edges.

### 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:

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

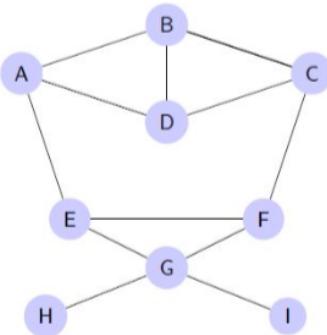
$$E_v = \{(A,C), (E,H), (H,I)\}$$

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

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

## 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):

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

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

Name(s):

---

---

---

---

First name(s):

---

---

---

---

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

---

---

---

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.*