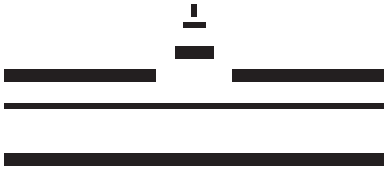


Westfälische
Wilhelms-Universität
Münster

Zentralitätskonzepte in Graphen

Bachelorarbeit
zur Erlangung des akademischen Grades
Bachelor of Science

Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik
Institut für Informatik

Betreuung:

Prof. Dr. Jan Vahrenhold

Eingereicht von:

Armin Wolf

Matrikelnummer:

398214

Münster, September 2015

Inhaltsverzeichnis

Zusammenfassung

Es gibt diverse Konzepte in der Graphentheorie um Knoten und Kanten zu bewerten. In dieser Arbeit werden nach subjektiver Einschätzung die wichtigsten Zentralitätskonzepte erläutert. Ein besonderer Schwerpunkt liegt bei der sogenannten *Betweenness Centrality*, die sowohl für Knoten als auch für Kanten definiert ist. Es wird auf verschiedene Algorithmen zur Berechnung eingegangen. Insbesondere wird ein dazu entwickeltes Programm vorgestellt und analysiert.

1. Einleitung

Sehr viele Problemstellungen lassen sich mithilfe von Graphen leichter lösen. Graphen erleichtern die Auswertung, geben eine Übersicht und zeigen Zusammenhänge zwischen den Datensätzen an. Um diese Vorteile von Graphen nutzen zu können, müssen sie anhand bestimmter Kriterien bewertet werden. Diese Bewertung geschieht neben den klassischen Konstrukten wie Durchmesser, Erreichbarkeit und Dichte auch anhand der sogenannten Graph-Zentralitäten.

Im Rahmen dieser Arbeit sollen die Zentralitätskonzepte in Graphen aufgearbeitet werden. Insbesondere wird auf die Algorithmen von Brandes [Bra01] sowie Girvan und Newman [GN02] eingegangen. Dabei sollen möglichst viele Beispiele und Illustrationen verwendet werden, um das Verständnis der verschiedenen Konzepte darzustellen. Im praktischen Teil der Bachelorarbeit soll ein Programm entwickelt werden, das die *betweenness*-Zentralität sowohl für Knoten als auch für Kanten bestimmt und das Endergebnis visualisiert.

In den folgenden Kapiteln werden die verschiedenen Zentralitäten ausführlich anhand von Beispielen erläutert. Insbesondere wird auf die Algorithmen und deren Eigenschaften eingegangen, die diese Zentralitäten realisieren. Des Weiteren wird das entwickelte Programm betrachtet. Die Struktur wird ausführlich besprochen und die implementierten Algorithmen werden einzeln veranschaulicht. Es wird auf Vor- und Nachteile von Algorithmen eingegangen. Zum Schluss werden die verwendeten Daten anhand des Programms ausgewertet und evaluiert.

2. Knoten-Zentralität

In diesem Kapitel wird auf die verschiedenen Maße eingegangen, die sich anhand der Knoten berechnen lassen. Die Theorie hinter den Zentralitätsmaßen wird der Literatur [ZAL14], [WF94], [Flo62] und [Bra01] entwendet.

2.1. Knotengrad

Unter allen Zentralitätsmaßen ist die Knoten-Zentralität wohl die einfachste Variante und am leichtesten nachzuvollziehen. Der Knotengrad beschreibt im allgemeinen die Anzahl der ein- und ausgehenden Kanten. Bei gerichteten Graphen kann man also von zwei Arten von Knotengraden sprechen. Im Englischen wird dies als *indegree* beziehungsweise *outdegree* bezeichnet. Der Bezeichnung entsprechend wird bei dem *indegree* die Anzahl der eingehenden Kanten berechnet und beim *outdegree* die der ausgehenden Kanten. Da man aber in einem ungerichteten Graphen nicht zwischen ein- und ausgehenden Kanten unterscheidet, ist wohl folgende Definition (für ungerichtete Graphen) eher zutreffend: Der Knotengrad von dem Knoten v_i ist die Anzahl der Kanten, die den Knoten v_i mit einem anderen Knoten v_j $i, j \in (1, .., n)$ verbinden. Der Knotengrad wird auch als $deg(v)$ bezeichnet (nach dem englischen Wort *Degree*).

Definition 1. *Knotengrad-Zentralität*
Sei $v_i \in V$ dann ist die Knotengrad-Zentralität des Knotens v_i definiert als:

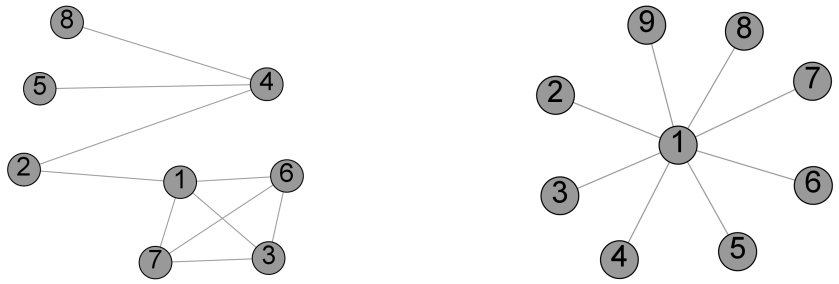
$$C_D(v_i) = \frac{deg(v_i)}{\max_j deg(v_j)}, i, j \in (1, .., n), \tag{2.1}$$

wobei $deg(v_i)$ $i \in (1, .., n)$ für den Knotengrad von dem Knoten v_i und $\max_j deg(v_j)$ für den maximalen Knotengrad im Graphen steht.

Die Normalisierung des Wertes muss nicht zwangsläufig über den maximalen Knotengrad geschehen. Man kann über die Anzahl der Knoten im Graphen die Normalisierung ausführen (siehe [ZAL14]). In einem Graphen $G = (V, E)$ mit $|V| = n$ gilt dann für Knoten v_i $i \in (1, .., n)$

$$C_D(v_i) = \frac{deg(v_i)}{n - 1}. \tag{2.2}$$

Um genauer zu verstehen was die Knotengrad-Zentralität aussagt, betrachte man folgendes Beispiel: Soziale Netzwerke kann man als Graph interpretieren, indem jeder Benutzer einem Knoten entspricht und Bekanntschaften unter den einzelnen Benutzern den Kanten entsprechen. Somit würde ein hohe Knotengrad-Zentralität eines Benutzers bedeuten, dass er in dem Netzwerk eine zentrale Rolle spielt.[WF94]



a) Beispielgraph A **b)** Beispielgraph B

Abbildung 2.1.: Ungerichtete Beispielgraphen für die Berechnung der Knotengrad-Zentralität

Beispiel

Man betrachte den Graphen in der Abbildung 2.1a. Da Knoten 1 vier Kanten besitzt, die ihn mit einem anderen Knoten verbinden, ist somit $deg(1) = 4$. Der Beispielgraph hat insgesamt 8 Knoten, also ist der maximale Knotengrad $n - 1 = 8 - 1 = 7$. Aus den Werten lässt sich nun $C_D(1)$ bestimmen.

$$C_D(1) = \frac{deg(1)}{n-1} = \frac{4}{8-1} = \frac{4}{7} = 0,5714 \tag{2.3}$$

Zum Vergleich $C_D(1)$ vom Graphen der Abbildung 2.1b ist genau 1, da der Knoten mit allen anderen Knoten verbunden ist. Bei einem vollständigen Graphen ist die Knotengrad-Zentralität wiederum für jeden Knoten gleich 1. Abhängig vom Graphen ist ersichtlich, dass sich die Zentralitäten immens voneinander unterscheiden.

2.2. Eigenvektor

Ähnlich wie die Knotengrad-Zentralität ist die Eigenvektor-Zentralität sowohl für gerichtete als auch ungerichtete Graphen definiert. Da in der Mathematik das Thema Eigenwerte und Eigenvektoren mit Matrizen verbunden wird,

ist es nicht außergewöhnlich, dass bei der Berechnung der Zentralität auf die Matrixrepräsentation des Graphen zurück gegriffen wird.

Definition 2. *Eigenvektor-Zentralität*
Sei $G = (V, E)$ ein Graph, A die Adjazenzmatrix und λ ein konstanter Faktor dann lässt sich die Eigenvektor-Zentralität $C_E(v_i)$ $\forall v_i \in V$, $i \in (1, .., n)$ wie folgt bestimmen[ZAL14].

$$c_E(v_i) = \frac{1}{\lambda} \sum_{j=1}^{n} A_{j,i} c_E(v_j) \tag{2.4}$$

Die Gleichung 2.4 lässt sich umformen, indem man die Zentralität für alle Knoten berechnet statt nur für einen. Dann gilt folgendes

$$C_E = \frac{1}{\lambda} A^T C_E \tag{2.5}$$

$$\lambda C_E = A^T C_E \tag{2.6}$$

Mit der Annahme, dass die Eigenvektor-Zentralitäten einen Spaltenvektor bilden, muss die Adjazenzmatrix transponiert werden, um die Matrixmultiplikation korrekt ausführen zu können (siehe [Fis13]). Da λ ein konstanter Faktor ist, kann man den Umformungsschritt von der Gleichung 2.5 zur nächsten Gleichung 2.6 problemlos machen. Durch diese Umformung ist Sichtbar geworden, dass C_E ein Eigenvektor der Adjazenzmatrix A ist.

$$A^T C_E - \lambda C_E = 0 \tag{2.7}$$

$$(A^T - \lambda I) C_E = 0 \tag{2.8}$$

Um die Eigenvektoren zu bestimmen, muss zuerst die Gleichung 2.8 gelöst werden. Da C_E Eigenvektoren sind, gilt ohne Beschränkung der Allgemeinheit $C_E \neq 0$. Somit muss der Term $A^T - \lambda I = 0$ sein. Als nächstes berechnet man die Nullstellen mit Hilfe des charakteristischen Polynoms. Wenn die Eigenwerte bestimmt wurden, muss man sich für einen λ-Wert entscheiden. Hierzu betrachte man folgendes Theorem.

Theorem 1. *Perron-Frobenius aus dem Dokument [ZAL14]*
Sei A die Adjazenzmatrix eines n-dimensionalen Graphen mit einer hohen Dichte, dann existiert eine positiv reelle Zahl λ_{max} auch Perron-Frobenius-Eigenwert genannt. Für die gilt alle Eigenwerte der Matrix A sind kleiner als λ_{max}. Insbesondere existiert ein Eigenvektor $\vec{v}_{max}$ passend zu λ_{max}.

Daher wählt man den maximalen Eigenwert als λ. Die Eigenvektor-Zentralitäten lassen sich anschließend anhand der Gleichung 2.8 berechnen.

a) Beispielgraph A **b)** Beispielgraph B

Abbildung 2.2.: Ungerichtete Beispielgraphen für die Berechnung der Eigenvektor-Zentralität

Beispiel

Um das Vorgehen zu verdeutlichen, wird nun die Eigenvektor-Zentralität der beiden Beispielgraphen aus den Abbildungen 2.2a und 2.2b berechnet. Sei A die Adjazenzmatrix des Graphen 2.2a.

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \tag{2.9}$$

Man berechne nun das charakteristische Polynom. Dafür muss zu erst die Determinante von $(A - \lambda I)$ berechnet werden. Da hier die Adjazenzmatrix eine 3×3 Matrix ist, kann man die Regel von Sarrus verwenden. Bei Matrizen höherer Dimension kann man den Laplaceschen Entwicklungssatz oder ähnliche Verfahren in Anspruch nehmen (siehe [Fis13]).

$$\det \begin{pmatrix} 0-\lambda & 1 & 1 \\ 1 & 0-\lambda & 1 \\ 1 & 1 & 0-\lambda \end{pmatrix} = 0 \tag{2.10}$$

Als Ergebnis der Determinanten-Rechnung erhält man das charakteristische Polynom:

$$\lambda^3 + 3\lambda + 2. \tag{2.11}$$

Durch das bestimmen der ersten Nullstelle kann das Polynom durch Polynomdivision vereinfacht werden. Die erste Nullstelle ist 2. Somit ist Folgendes zu berechnen:

$$(-\lambda^3 + 3\lambda + 2)/(\lambda - 2). \tag{2.12}$$

Dadurch ergibt sich letztendlich als vereinfachtes Polynom:

$$-(\lambda - 2)(\lambda + 1)^2. \tag{2.13}$$

Um jetzt die Nullstellen zu bestimmen, betrachtet man die einzelnen Terme der Gleichung 2.13. Die Gleichung ist erfüllt falls $-(\lambda-2) = 0$ oder $(\lambda+1) = 0$. Nun sieht man, dass es zwei Nullstellen gibt, die schon bekannte $\lambda = 2$ und eine weitere $\lambda = -1$. Diese Nullstellen sind insbesondere die Eigenwerte der Adjazenzmatrix 2.9. Als nächstes müssen die Eigenvektoren bestimmt werden. Nach dem Theorem 1 wählt man den maximalen Eigenwert, um die Eigenvektoren zu bestimmen. Nach dem Einsetzen des Eigenwerts in die Matrix 2.10 erhält man:

$$\begin{pmatrix} 0-\lambda & 1 & 1 \\ 1 & 0-\lambda & 1 \\ 1 & 1 & 0-\lambda \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2.14}$$

Führt man nun die Matrix-Vektor Multiplikation aus, erhält man ein lineares Gleichungssystem. Das Gleichungssystem kann durch ein Verfahren nach eigener Wahl berechnet werden, um folgenden Eigenvektor zu erhalten:

$$v_\lambda = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{2.15}$$

Für die Eigenvektor Zentralität müssen die Werte zunächst normiert werden. Für den Eigenvektor 2.15 gilt:

$$\sqrt{(1^2 + 1^2 + 1^2)} = \sqrt{3} \tag{2.16}$$

Somit erhält man folgende Werte für C_E:

$$C_E = \begin{pmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{pmatrix}. \tag{2.17}$$

Als zweites Beispiel soll die Eigenvektor-Zentralität des Graphen in der Abbildung 2.2b berechnet werden. Dementsprechend sei B die Adjazenzmatrix des betrachteten Graphen.

$$B = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.18}$$

Ähnlich wie im vorherigen Beispiel muss man zuerst mit Hilfe des charakteristischen Polynoms die Eigenwerte bestimmen, aus denen im nächsten Schritt

die Eigenvektoren berechnet werden. Im Anschluss werden die Eigenvektoren normiert.

$$\det \begin{pmatrix} -\lambda & 1 & 1 & 0 \\ 1 & -\lambda & 1 & 1 \\ 1 & 1 & -\lambda & 1 \\ 0 & 1 & 1 & -\lambda \end{pmatrix} = 0 \tag{2.19}$$

Aus der Rechnung ergibt sich das charakteristische Polynom der Form:

$$-\lambda^4 + 5\lambda^2 + 4\lambda = -\lambda(\lambda + 1)(\lambda^2 - \lambda - 4). \tag{2.20}$$

Das Polynom wird durch Polynomdivision in die oben sichtbare Form vereinfacht. Die Eigenwerte können abgelesen werden, indem man die einzelnen Terme gleich null setzt und betrachtet für welchen λ Wert der Term null ergibt. Die Eigenwerte der Matrix B sind:-1.5615, -1, 0, 2.5615. Somit ist $\lambda_{max} = 2.5615$.

$$\det \begin{pmatrix} -2.6855 & 1 & 1 & 0 \\ 1 & -2.6855 & 1 & 1 \\ 1 & 1 & -2.6855 & 1 \\ 0 & 1 & 1 & -2.6855 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2.21}$$

Wird das entstehende lineare Gleichungssystem gelöst, erhält man die Eigenvektoren, die dann noch zu normieren sind. Die Eigenvektor-Zentralitäten des Graphen der Abbildung 2.2b sind dann:

$$C_E = \begin{pmatrix} 0.4351 \\ 0.5573 \\ 0.5573 \\ 0.4351 \end{pmatrix}. \tag{2.22}$$

Werden die zwei Graphen und die dazugehörigen Zentralitäten verglichen, erkennt der Betrachter deutliche Unterschiede. Der Graph A aus Beispiel 1 ist ein vollständiger Graph. Somit hatten alle Knoten dieselbe Eigenwert-Zentralität. Graph B hingegen hatte vier Knoten von denen sich die Knoten 1 und 4 sowie 2 und 3 ähneln. Dementsprechend hatten diese Knotenpaare einen identischen Zentralitätswert. In den nächsten Unterkapiteln wird man sehen, dass sich einige Zentralitäten an der Eigenvektor-Zentralität orientieren.

2.3. Katz

Wie schon im letzten Unterkapitel erwähnt, gibt es einige Zentralitätsmaße, die der Eigenvektor-Zentralität ähneln. Dazu gehört unter anderem die Katz-Zentralität. Der Nachteil der Eigenvektor-Zentralität ist bei gerichteten Graphen ohne ausgehenden Kanten bemerkbar, denn dann ist die Zentralität für

den Knoten null. In diesem Fall ist es von Vorteil auf die Katz-Zentralität zurück zu greifen. Zur Vermeidung von null-Zentralitäten wird bei der Katz-Zentralität ein Korrekturterm β addiert.

Definition 3. *Katz-Zentralität*
Sei $G = (V, E)$ ein Graph, A die Adjazenzmatrix und α eine Konstante, die als Schranke dient. Dann lässt sich die Katz-Zentralität für den Knoten $v_i \in V$ $\forall i \in (1, .., n)$ wie folgt bestimmen (siehe [ZAL14]).

$$C_K(v_i) = \alpha \sum_{j=1}^{n} A_{j,i} c_K(v_j) + \beta \tag{2.23}$$

Die Konstante α ist abhängig von dem maximalen Eigenwert der Matrix. Die Vorgehensweise ähnelt der in Kapitel 2.2 erläuterten Eigenvektor-Zentralität. Die Gleichung 2.23 lässt sich als Produkt von Matrizen und Vektoren auffassen. Sei C_K der Vektor mit den Katz-Zentralität Werten, A eine Adjazenzmatrix und $\mathbf{1}$ ein Einservektor (alle Einträge des Vektors sind 1). Dann gilt:

$$C_K = \alpha A^T C_K + \beta \mathbf{1}. \tag{2.24}$$

Im nächsten Umformungsschritt wird $\alpha A^T C_K$ auf die linke Seite der Gleichung gebracht, um dann C_K auszuklammern. Danach werden beide Seiten mit der Inversen-Matrix $(A^T C_K)^{-1}$ multipliziert.

$$C_K - \alpha A^T C_K = \beta \mathbf{1} \tag{2.25}$$
$$C_K(I - \alpha A^T) = \beta \mathbf{1} \tag{2.26}$$
$$C_K(I - \alpha A^T)(I - \alpha A^T)^{-1} = \beta \mathbf{1}(I - \alpha A^T)^{-1} \tag{2.27}$$
$$C_K = \beta(I - \alpha A^T)^{-1}\mathbf{1} \tag{2.28}$$

Die inverse Matrix lässt sich mit Hilfe des Gauß-Jordan-Algorithmus bestimmen. Zum Einfluss des Faktors α ist zu erwähnen, falls $\alpha = 0$ ist, ist $C_K = \beta$. Sobald aber α wächst, sinkt der Einfluss des Korrekturterms β (siehe [ZAL14]). Der α Wert wächst genau dann, wenn $\det(I - \alpha A^T) = 0$ ist. Dies wiederum bedeutet, dass $\alpha = \frac{1}{\lambda}$ ist (folgt aus [ZAL14]). Wenn

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = A^T$$

und $\alpha = \frac{1}{5}$, dann gilt:

$$(A^T - \frac{1}{\alpha} I) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} - \alpha \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad = \begin{pmatrix} -4 & 1 \\ 1 & -4 \end{pmatrix} \tag{2.29}$$
$$(I - \alpha A^T) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} - \frac{1}{\alpha} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad = \begin{pmatrix} \frac{4}{5} & \frac{-1}{5} \\ \frac{-1}{5} & \frac{4}{5} \end{pmatrix} = -5 \begin{pmatrix} \frac{4}{5} & \frac{-1}{5} \\ \frac{-1}{5} & \frac{4}{5} \end{pmatrix} \tag{2.30}$$
$$= (A^T - \frac{1}{\alpha} I) \tag{2.31}$$

Der Wert α sollte in Abhängigkeit von λ_{max} gewählt werden. In der Praxis sollte $\alpha < \frac{1}{\lambda_{max}}$ sein.

Beispiel

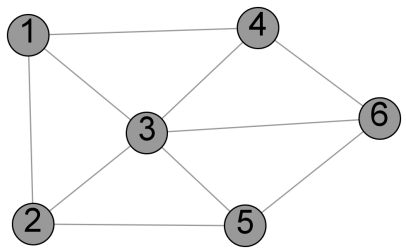


Abbildung 2.3.: Beispielgraph zur Berechnung der Katz-Zentralität

Sei nun A die Adjazenzmatrix des Graphen in der Abbildung 2.3 und $\beta = 0.2$ wie auch im Beispiel von [ZAL14].

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.32}$$

Wenn man nun die Eigenwerte der Matrix bestimmt erhält man $\lambda_{max} = 1.3953$ also ist unser $\alpha = 0.5$, $\alpha < \frac{1}{\lambda_{max}}$ ist somit erfüllt.

$$C_K = \beta(I - \alpha A^T)^{-1}\mathbf{1} \tag{2.33}$$

$$= \begin{pmatrix} 28/55 \\ 42/55 \\ 46/55 \\ 34/55 \\ 34/55 \\ 68/55 \end{pmatrix} = \begin{pmatrix} 0.5090 \\ 0.7636 \\ 0.8363 \\ 0.6181 \\ 0.6181 \\ 1.2363 \end{pmatrix} \tag{2.34}$$

Obwohl Knoten 6 keine ausgehenden Kanten besitzt, hat er doch die größte Zentralität, da die eingehenden Kanten mit betrachtet werden. Als Vergleich soll nun die Eigenvektor-Zentralität dieses Graphen berechnet werden.

$$C_E = \begin{pmatrix} 0.6099 \\ 0.3553 \\ 0.4957 \\ 0.4371 \\ 0.2546 \\ 0 \end{pmatrix} \tag{2.35}$$

Man sieht, dass die Ergebnisse der zwei Verfahren sich sehr unterscheiden. Knoten 6 hat eine Zentralität von 0 obwohl 3 eingehende Kanten vorhanden sind, die aber in der Eigenvektor-Zentralität nicht betrachtet werden. Bei solchen Ausnahmefällen liefert die Katz-Zentralität verlässlichere Werte, als die vorherigen Zentralitäten.

2.4. PageRank

Im Unterkapitel 2.3 der Katz-Zentralität wurde erklärt, dass das Verfahren im Falle von Knoten ohne ausgehenden Kanten zuverlässigere Werte liefert. Die Katz-Zentralität hat aber auch Nachteile. Wenn einem Knoten eine hohe Zentralität zugeordnet wurde, so übergibt er diese an all seine Nachbarknoten. Dies führt zu falschen Ergebnissen beziehungsweise zu Fehlinterpretation von Daten. In diesem Sinne kommt man zum nächsten Zentralitätskonzept, dem PageRanking. Die Grundidee der Zentralität um das Problem zu umgehen ist, dass man die erhaltene Zentralität des Knotens durch die ausgehenden Kanten teilt. So erhält jeder Knoten eine angemessene Zentralität. Die Suchmaschine *Google* verwendet unter anderem ein PageRank Verfahren, um die Menge der Webseiten prioritätsgerecht zu ordnen. Dabei werden die Webseiten als Knoten in einem großen Netzwerk(Graphen) betrachtet. Die Herleitung der Zentralität ähnelt stark der Eigenvektor- beziehungsweise Katz-Zentralität.

Definition 4. *PageRank-Zentralität*
Sei $G = (V, E)$ ein Graph, dann ist die PageRank-Zentralität eines Knotens $v_i \in V$ definiert als

$$C_P(v_i) = \alpha \sum_{j=1}^{n} A_{j,i} \frac{c_P(v_j)}{deg_{out}(v_j)} + \beta, \forall i \in (1, .., n). \tag{2.36}$$

wobei $deg_{out}(v_j)$ der outdegree - *die Anzahl der ausgehenden Kanten - des Knotens v_j sei, α ein konstanter Faktor der als Schranke dient und β ein Korrekturterm um null-Zentralitäten (wenn die Zentralität eines Knotens null ist) zu vermeiden.*

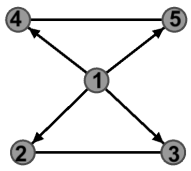
Dies wurde beibehalten, jedoch ist die Gleichung 2.36 nur im Falle von $deg_{out}(v_j) > 0$ definiert (siehe [ZAL14]). Da es leichter ist mit Matrizen zu rechnen als mit den einzelnen Werten, ist es empfehlenswert die rekursive Definition umzuschreiben. Sei A eine Adjazenzmatrix, C_P ein Vektor mit den Zentralitätswerten und D eine Diagonalmatrix mit den *outdegree* Werten der jeweiligen Knoten. Es gilt:

$$C_P = \alpha A^T C_P D^{-1} + \beta \mathbf{1}. \tag{2.37}$$

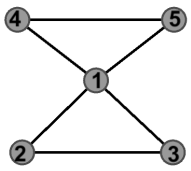
In der Gleichung 2.37 wird mit der Inversen Matrix D^{-1} multipliziert, da die Division bei Matrizen nicht definiert ist, stattdessen multipliziert man mit der Inversen. Es werden dieselben Umformungen durchgeführt wie bei der Gleichung 2.24. Somit erhält man folgende Gleichung:

$$C_P = \beta(I - \alpha A^T D^{-1})^{-1}\mathbf{1}. \tag{2.38}$$

Bei der Wahl des α-Werts hält man sich weiterhin an die Annahme aus dem Kapitel 2.3, $\alpha < \frac{1}{\lambda_{max}}$ wobei λ_{max} der größte Eigenwert sei. In den nächsten zwei Beispielen kann man gut erkennen, wie sich PageRank-Zentralität bei einer Veränderung des Graphen verhält.



a) Beispielgraph A



b) Beispielgraph B

Abbildung 2.4.: Graphen zur Berechnung der PageRank-Zentralität

Beispiel

Sei A die Adjazenzmatrix des Graphen in der Abbildung 2.4a.

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.39}$$

Da der Graph in der Abbildung gerichtete als auch ungerichtete Kanten enthält, muss die Matrix transponiert werden. Danach können die Werte der Matrix mit α multipliziert werden und die Matrizenmultiplikation zwischen A und $D^{-}1$ kann ausgeführt werden. Das Resultat ist weiterhin eine 6×6 Matrix mit Nullen auf der Diagonalen. Im nächsten Schritt wird die Matrix A von der Einheitsmatrix I subtrahiert. Das Ergebnis der bisherigen Rechenschritte sieht man in 2.40.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{-9}{40} & 1 & \frac{-9}{10} & 0 & 0 \\ \frac{-9}{40} & \frac{-9}{10} & 1 & 0 & 0 \\ \frac{-9}{40} & 0 & 0 & 1 & \frac{-9}{10} \\ \frac{-9}{40} & 0 & 0 & \frac{-9}{10} & 1 \end{pmatrix} \tag{2.40}$$

Zuletzt muss die Matrix invertiert und mit β und $\mathbf{1}$ multipliziert werden. Dann erhält man einen Vektor mit folgenden Zentralitätswerten.

$$C_P = \begin{pmatrix} 0.2 \\ 2.45 \\ 2.45 \\ 2.45 \\ 2.45 \end{pmatrix} \tag{2.41}$$

Wenn man jetzt die Werte näher betrachtet, sieht man das Knoten 1 die geringste Zentralität hat, obwohl er mit allen weiteren Knoten im Graphen verbunden ist. Der ihm zugewiesene Wert lässt sich durch seine fehlenden eingehenden Knoten begründen. Somit ist Knoten 1 im ganzen Graphen isoliert und deswegen hat er eine sehr geringe Zentralität.
Zum Vergleich wird nun der zweite Graph aus der Abbildung 2.4b berechnet der nur ungerichtete Kanten besitzt. Sei B die Adjazenzmatrix des Graphen der Abbildung 2.4b und D die dazugehörige Diagonalmatrix.

$$B = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}, D = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix} \tag{2.42}$$

Führt man nun die selben Rechenschritte aus wie im Beispiel 1, dann erhält man folgende Werte:

$$C_P = \begin{pmatrix} 16.20 \\ 8.448 \\ 8.448 \\ 8.448 \\ 8.448 \end{pmatrix}. \tag{2.43}$$

Die Zentralitäten haben sich signifikant verändert. Da Knoten 1 nun auch eingehende Kanten besitzt und die zwei Fraktionen im Graphen somit miteinander verbunden sind, hat Knoten 1 eine wesentlich höhere Zentralität als die restlichen Knoten. Dies zeigt, dass auch kleine Modifikationen der Graph-Struktur zu wesentlichen Veränderungen der Zentralitäten führen.

2.5. Closeness Centrality

Die zuletzt erläuterten Zentralitäten lehnen sich größtenteils an die Eigenvektor-Zentralität an. Die *closseness*-Zentralität hingegen weicht davon ab, hier wird anhand der kürzesten Pfade die Relevanz der Knoten bestimmt. Der Grundgedanke des Verfahrens ist, dass zentrale Knoten mit kleinerem Aufwand die übrigen Knoten im Graphen erreichen können. Unter diesem Aufwand versteht man die durchschnittliche Länge des kürzesten Pfades vom betrachteten zu allen weiteren Knoten. Ein kürzester Pfad zwischen den Knoten $s \to t$ sei eine Folge von Kanten deren Startpunkt der Knoten s sei und der Endpunkt Knoten t. Insbesondere gilt, dass die Summe der Kantengewichte entlang des kürzesten Pfades minimal ist.

Definition 5. *closseness-Zentralität*
Sei $G = (V, E)$ ein Graph und $v_i \in V$, dann ist die closseness-Zentralität des Knotens $v_i, i \in (1, .., n)$ definiert als:

$$C_C(v_i) = \frac{1}{l_{avg}} \tag{2.44}$$

wobei $l_{avg} = \frac{1}{(n-1)} \sum_{v_i \neq v_j} l_{i,j}$ die durchschnittliche Länge der kürzesten Pfade bezeichnet, deren Startpunkt der Knoten v_i ist.

Zur Berechnung der kürzesten Pfade können unterschiedliche Verfahren verwendet werden. Der wohl bekannteste Algorithmus zur Bestimmung von kürzesten Pfaden ist der von Dijkstra (siehe [Lei+01]). Jedoch ist davon abzuraten, da die kürzesten Pfade von allen Knoten aus benötigt werden und der Algorithmus nur die kürzesten Pfade von einem Knoten aus berechnet. Der Algorithmus von Dijkstra ist ein gieriger Algorithmus.

Definition 6. *Gieriger Algorithmus nach [KT06] Ein Algorithmus, der eine Entscheidung bezüglich seiner Rückgabe lokal optimal trifft und eine solche Entscheidung nicht mehr revidiert, wird als gieriger Algorithmus bezeichnet.*

Es werden alle Kanten, die zu benachbarten Knoten führen, betrachtet und die Kante mit dem geringsten Gewicht wird ausgewählt. Dieser Vorgang wird solange wiederholt bis der Zielknoten erreicht ist. Somit müsste man den Algorithmus für jeden Knoten einmal ausführen, was wiederum sehr ineffizient ist. Eine wesentlich leichtere und auch etwas bessere Lösung ist es den

Algorithmus von Floyd zu verwenden (siehe [Flo62]). Der Algorithmus berechnet die kürzesten Pfade für alle Knoten. Der Algorithmus verwendet intern folgendes Lemma:

Lemma 1. *(Bellman Kriterium) nach [Bra01]*
Ein Knoten v ist ein Bestandteil des kürzesten Pfades von $s \to t$, wenn $l_{st} = l_{sv} + l_{vt}$, l sei die Länge der Pfade.

Die Vorgehensweise des Algorithmus ist also das Vergleichen der Pfade, die einen Zwischenknoten enthalten, mit der direkten Kante und das auswählen des günstigeren Weges. Nach erfolgreichem Durchlauf des Algorithmus, enthält die übergebene Adjazenzmatrix die minimalen Distanzen von dem betrachteten Knoten zu jedem anderen Knoten. Daraus können dann die durchschnittlichen Längen berechnet werden, um daraus die Zentralitäten zu bestimmen. Folgendes Beispiel soll die Vorgehensweise erläutern.

Algorithmus 1 Algorithm 97 Shortest Path, Robert W. Floyd siehe [Flo62]

```
Require: integer i, j, k; real inf; inf := 10^10
// Schleife der Zwischenknoten.
for i:= 1 to n do
    // Schleife der Startknoten.
    for j:=1 to n do
        // Die innere Schleife wird nur dann ausgeführt wenn,
        // es eine Kante gibt von Startknoten bis zum Zwischenknoten.
        if m[j][i] < inf then
            // Schleife der Endknoten
            for k:=1 to n do
                // Wenn eine Kante vom Zwischenknoten bis zum Endknoten
                // existiert wird geprüft, ob der Weg über den Zwischenknoten
                // günstiger ist als die direkte Kante.
                if m[i][k] < inf then
                    if m[j][i] + m[i][k] < m[j][k] then
                        m[j][k] ← min(m[j][k], m[j][i] + m[i][k]);
                    end if
                end if
            end for
        end if
    end for
end for
return shortest Path
```

Beispiel

Sei A die Adjazenzmatrix des Graphen in der Abbildung 2.5. Um den Algorithmus von Floyd anwenden zu können, verändere man die Werte. Alle nicht existie-

renden Kanten müssen *unendlich* gesetzt werden.

$$\begin{pmatrix} 0 & 2 & \infty & 2 & \infty & 5 \\ 2 & 0 & 3 & \infty & \infty & \infty \\ \infty & 3 & 0 & 4 & \infty & \infty \\ 2 & \infty & 4 & 0 & 3 & \infty \\ \infty & \infty & \infty & 3 & 0 & 1 \\ 5 & \infty & \infty & \infty & 1 & 0 \end{pmatrix} \tag{2.45}$$

Man geht alle Zeilen und Spalten durch und vergleicht, ob die direkte Kante günstiger ist als ein Pfad durch einen Zwischenknoten. So ergeben sich folgende kürzesten Pfade mit dem Zwischenknoten 1 das heißt bei Betrachtung der ersten Zeile, Spalte:

$$(2 \to 4) = (2 \to 1) + (1 \to 4); \tag{2.46}$$
$$(2 \to 6) = (2 \to 1) + (1 \to 6); \tag{2.47}$$
$$(4 \to 2) = (4 \to 1) + (1 \to 2); \tag{2.48}$$
$$(4 \to 6) = (4 \to 1) + (1 \to 6); \tag{2.49}$$
$$(6 \to 2) = (6 \to 1) + (1 \to 2); \tag{2.50}$$
$$(6 \to 4) = (6 \to 1) + (1 \to 4); \tag{2.51}$$

Nun muss dieses Vorgehen für alle Zeilen und Spalten ausgeführt werden. Als

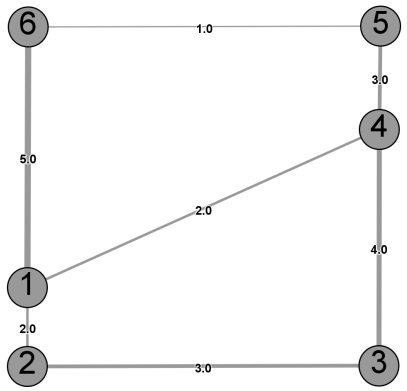


Abbildung 2.5.: Graph zur Berechnung der Closseness-Zentralität

Endergebnis erhält man die Matrix mit der Länge der kürzesten Pfade.

$$\begin{pmatrix} 0 & 2 & 5 & 2 & 5 & 5 \\ 2 & 0 & 3 & 4 & 7 & 7 \\ 5 & 3 & 0 & 4 & 7 & 8 \\ 2 & 4 & 4 & 0 & 3 & 4 \\ 5 & 7 & 7 & 3 & 0 & 1 \\ 5 & 7 & 8 & 4 & 1 & 0 \end{pmatrix} \tag{2.52}$$

Um daraus die Zentralitäten zu bestimmen, muss zuerst die durchschnittliche Länge der kürzesten Pfade l_{avg} berechnet werden. Dafür addiert man die Zeilenwerte der erhaltenen Matrix und dividiert durch $n-1$. Als letzten Schritt berechnet man $\frac{1}{l_{avg}}$. Für den Knoten 1 heißt das:

$$l_{avg} = (2+5+2+5+5)/5 = 19/5 \tag{2.53}$$

$$C_C(1) = \frac{1}{l_{avg}} = \frac{1}{\frac{19}{5}} = \frac{5}{19} = 0.263 \tag{2.54}$$

Dies muss für die restlichen Knoten wiederholt werden und man erhält folgenden Vektor mit den Zentralitäten.

$$C_C = \begin{pmatrix} 0.263 \\ 0.217 \\ 0.185 \\ 0.294 \\ 0.217 \\ 0.2 \end{pmatrix} \tag{2.55}$$

Knoten 1 und 4 haben die höchste Zentralität im Graphen. In einem Sozialen Netzwerk bedeutet dies, dass die Benutzer ein höheres Ansehen haben beziehungsweise bekannter sind.

2.6. Betweenness Centrality

Ein weiteres Zentralitätsmaß ist die sogenannte *betweenness*-Zentralität. Der Name *betweenness centrality* wurde von Freeman eingeführt (siehe [Fre77]). Ähnlich zur Closeness-Zentralität werden für die Berechnung die kürzesten Pfade des Graphen benötigt. Die *betweenness*-Zentralität lässt sich unabhängig vom Kantengewicht bei gerichteten und ungerichteten Graphen bestimmen. Diese Zentralität liefert eine Aussage über die Knoten des Graphen anhand der Anzahl der kürzesten Pfade, die den betrachteten Knoten passieren. Die *betweenness*-Zentralität hat viele Verwendungsstellen: Social Media Analyse oder auch als Ranking von Webseiten [Bra01]. Somit kommt man zur Definition dieses Zentralitätsmaßes.

Definition 7. *Sei $G = (V, E)$ ein Graph und $v \in V$ ein Knoten. Dann gilt:*

$$C_B(v) = \sum_{\substack{s,t \in V \\ s \neq t \neq v}} \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{2.56}$$

wobei σ_{st} für die Anzahl der kürzesten Pfade zwischen Knoten s und t steht und $\sigma_{st}(v)$ für die Anzahl der Pfade, die noch zusätzlich den Knoten v passieren.

Insbesondere dürfen der Start- und Endpunkt nicht derselbe Knoten sein.

Definition 8. *Seien s,t, v ∈ V dann ist die Paar-Abhängigkeit $\delta_{st}(v)$ definiert als Anteil der kürzesten Pfade von s nach t in denen Knoten v enthalten ist, von der Gesamtanzahl der kürzesten Pfade von s nach t. Es gilt:*

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}. \tag{2.57}$$

Hat man die Zentralitäten berechnet, sollte man in der Regel normalisieren, da es sonst keinen guten Vergleich mit anderen Graphen gibt. Es ist empfehlenswert über die Anzahl der möglichen Kanten zu normalisieren. Für einen Graphen mit n Knoten bedeutet das, dass für jeden Knoten der berechnete Wert durch $(n-1)(n-2)$ dividiert werden muss. So erhält man dann Werte aus dem Intervall $[0, 1]$. Wie schon erwähnt, gibt es mehrere Möglichkeiten die kürzeste Pfade zu berechnen. Dementsprechend gibt es auch mehrere Verfahren zur Berechnung der *betweenness*-Zentralität. Zum Zählen der kürzesten Pfade gibt es ebenfalls verschiedene Möglichkeiten. In der Literatur [Bra01] werden zwei Ansätze erwähnt: Der Algebraische und der Kombinatorische Weg.

Lemma 2. *Algebraisches Zählen der Pfade einer bestimmten Länge nach [Bra01] Sei A^k die k-te Potenz der Adjazenzmatrix eines ungewichteten Graphen, dann enthält $A^k(s,t)$ die Anzahl der Pfade von Knoten s nach t der Länge k.*

Beweis. des Lemmas 2 nach [Juk07]
Beweis durch vollständige Induktion über k.
Induktionsanfang: $k = 1$ Die erste Potenz der Matrix ist die Originalmatrix. Sie enthält die direkten Verbindungen zwischen den Knoten i und j. Das ist immer erfüllt.
Induktionsvoraussetzung: Die Behauptung gelte für ein beliebiges jedoch festes $k \in \mathbb{N}$.
Induktionsschritt: $k \to k+1$
Die Potenzierung der Matrix ist mit anderen Worten die Multiplikation der Matrix mit sich selbst. Nach der Definition der Matrixmultiplikation, wird jede Zeile mit jeder Spalte multipliziert. Das heißt, für $A^{k+1}(i,j)$ wird die i-te Zeile mit der j-ten Spalte multipliziert. Es gilt:

$$A^{k+1}(i,j) = \sum_{t=1}^{n} A(i,t) \cdot A^k(t,j). \tag{2.58}$$

Da es sich um eine Adjazenzmatrix handelt, ist $A^{k+1}(i,j)$ genau dann ungleich null, wenn $A(i,t) = 1$ ist $\forall\ t \in (1,...,n)$. Dann ist $A(i,t) \cdot A^k(t,j) = A^k(t,j)$. Nach der Induktionsvoraussetzung gilt, dass $A^k(t,j)$ Pfade der Länge k enthält (von t nach j). Da $A(i,t) = 1$ ist also die Kante existiert, enthält $A^{k+1}(i,j)$ Pfade - von i nach j - der Länge $k+1$. Es folgt, dass die Gleichung 2.58 die Anzahl aller Pfade - von i nach j - der Länge $k+1$ angibt. □

Mit diesem Vorgehen kann man in ungewichteten Graphen alle kürzesten Pfade bestimmen, da alle Kanten das Gewicht $\omega = 1$ haben. Jedoch ist dieses Verfahren sehr kostenintensiv und daher nicht empfehlenswert. Eine einfache Lösung ist es, den Algorithmus von Floyd (siehe [Flo62]) zu modifizieren. Man kann die Zeile 12 im Algorithmus 1 erweitern und die Vorgänger speichern. Dies funktioniert folgendermaßen: In dem Schritt bei dem die Länge des Pfades $(s \to v) + (v \to t)$ geringer ist als die Länge des Pfades $s \to t$, wird v zum Vorgänger des Knotens t. Mithilfe dieser Vorgänger-Matrix *pred* können alle kürzesten Pfade rekonstruiert und abgespeichert werden. Wenn alle kürzesten Pfade wiederhergestellt sind, lässt sich die *betweenness*-Zentralität für jeden Knoten durch das Berechnen und Aufaddieren der Paar-Abhängigkeiten bestimmen. Alternativ kann auf das Wiederherstellen der kürzesten Pfade verzichtet werden, stattdessen kann die Anzahl der kürzesten Pfade zwischen zwei Knoten auf kombinatorischem Weg bestimmt werden.

Lemma 3. *Kombinatorisches Zählen der kürzesten Pfade nach [Bra01]*
Seien $s, t \in V$ und $s \neq v$ dann gilt:

$$\sigma_{sv} = \sum_{u \in \mathrm{pred(s,v)}} \sigma_{su}.$$

Dieses Lemma lässt sich folgendermaßen interpretieren: Da s ungleich v ist, enthält der kürzeste Pfad zwischen s und v mindestens einen Zwischenknoten u. Dieser Knoten ist aber dann in der Vorgänger-Matrix *pred* enthalten. Falls der kürzeste Pfad mehrere Zwischenknoten passiert, so sind alle Vorgänger in der Matrix *pred* enthalten und man erkennt, dass es sich um eine rekursive Formel handelt. Die Korrektheit des Lemmas folgt aus dem Beweis von [Bra01].

Beweis. von Lemma 3 nach [Bra01]
Da alle Kantengewichte positiv sind, gilt für jeden kürzesten Pfad $s \to v$, wenn $u \to v$ die letzte Kante ist:
Die Distanz von $s \to u$ ist kleiner als die Distanz von $s \to v$. Nun folgt die Gleichung aus dem Bellman-Kriterium (Lemma 1), da diese Feststellung für jeden Vorgänger von u gilt.

$$s \to u = s \to pred(u) + \omega_{pred(u),u} \tag{2.59}$$

Somit ist die Anzahl der kürzesten Pfade von $s \to v$ gleich der Summe der kürzesten Pfade von $s \to u$ für alle Vorgänger u des Knotens v.

□

Die meisten Verfahren zur Berechnung der *betweenness*-Zentralität tendieren zu einer Laufzeit $\mathcal{O}(n^3)$. Ulrik Brandes veröffentlichte in seinem Artikel „A Faster Algorithm for Betweenness Centrality“ einen Algorithmus für ungerichtete

und ungewichtete Graphen, der mit einer Laufzeit von $\mathcal{O}(|V||E|)$ die Zentralität bestimmt (siehe Algorithmus 2). Der Algorithmus basiert auf der Berechnungsvorschrift 2.61 . Um die Summen-schreibweise der Paar-Abhängigkeit nicht immer explizit angeben zu müssen, wird die Abhängigkeit definiert.

Definition 9. *Die Abhängigkeit nach [Bra01]*
Für die Knoten $s, v \in V$ ist die Abhängigkeit definiert als:

$$\delta_s(v) = \sum_{t \in V} \delta_{st}(v) \tag{2.60}$$

Zur Berechnung der Abhängigkeit des Knotens v auf allen kürzesten Pfaden, deren Startpunkt der Knoten s ist, kann folgende Berechnungsvorschrift verwendet werden:

$$\delta_s(v) = \sum_{w:v \in pred(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_s(w)). \tag{2.61}$$

Um die Korrektheit zu beweisen, ist noch eine Vorüberlegung erforderlich. Dafür wird ein weiteres Lemma eingeführt mit der Annahme, dass es zwischen zwei Knoten genau einen kürzesten Pfad gibt.

Lemma 4. *nach [Bra01]*
Wenn es genau einen kürzesten Pfad zwischen den Knoten s und t gibt, gilt folgendes für die Abhängigkeit des dazwischen liegenden Knotens v:

$$\delta_s(v) = \sum_{w:v \in \mathrm{pred(s,w)}} (1 + \delta_s(w)).$$

Beweis. des Lemma 4 nach [Bra01]
Da angenommen wurde, dass es genau einen kürzesten Pfad zwischen zwei beliebigen Knoten s und t gibt, bilden die Knoten und Kanten der kürzesten Pfade eine Baumstruktur. Wenn Knoten v ein Teil dieser Baumstruktur ist, dann ist v für alle seine Kinderknoten, ein Teil des kürzesten Pfades von s bis zu dem Kind von v. Knoten v ist für alle seine Kinderknoten der Vorgänger. Das heißt, für alle Knoten s, t gilt $\delta_{st}(v)$ ist 1, wenn v der Vorgänger von t ist, sonst 0. Somit ist $\delta_s(v)$ ist die Summe vom dem Pfad $s \to v$ und alle kürzesten Pfade $s \to t$, für die gilt v ist ein Vorgänger für Knoten t. So ergibt sich die zu zeigende Formel.

□

Jetzt lässt sich die Korrektheit der Berechnungsvorschrift 2.61 beweisen, da sie eine Verallgemeinerung des Lemma 4 ist.

Beweis. der Berechnungsvorschrift 2.61 nach [Bra01]
Da mehrere kürzeste Pfade existieren, muss die Definition der Paar-Abhängigkeit für einen Knoten v erweitert werden in $\delta_{st}(v, e)$. Sei $e = (v, w)$ die Kante, die Bestandteil des kürzesten Pfades sein muss, damit Knoten v im kürzesten Pfad

enthalten ist. Die Paar-Abhängigkeit $\delta_{st}(v, e)$ drückt damit den Anteil der kürzesten Pfade von s nach t, die über die Kante e und den Knoten v laufen, über die gesamten kürzesten Pfade von s nach t aus. Somit ergibt sich die Gleichung:

$$\delta_{st}(v, e) = \frac{\sigma_{st}(v, e)}{\sigma_{st}}. \tag{2.62}$$

Jetzt kann man die neu definierte Paar-Abhängigkeit in die Gleichung des Lemmas 3 einsetzen und erhält:

$$\begin{aligned}
\delta_s(v) &= \sum_{t \in V} \delta_{st}(v) && (2.63)\\
&= \sum_{t \in V} \sum_{w : v \in pred(s,w)} \delta_{st}(v, (v \to w)) && (2.64)\\
&= \sum_{w : v \in pred(s,w)} \sum_{t \in V} \delta_{st}(v, (v \to w)) && (2.65)
\end{aligned}$$

Sei w ein beliebiger Knoten, für den gilt, dass $v \in$ *pred(s,w)*. Da σ_{sw} die Anzahl der kürzesten Pfade zwischen s und w bezeichnet, ist dies gleich der Anzahl der kürzesten Pfade zwischen s und dem Vorgänger, also v und der Kante $e = (v, w)$. Das heißt, für beliebige Knoten s und $t \neq w$ gilt: $\frac{\sigma_{sv}}{sw}\sigma_{st}(v)$ kürzeste Pfade enthalten den Knoten v und die Kante e. Somit verändert sich die Paar-Abhängigkeit der Knoten folgendermaßen:

$$\delta_{st}(v, e) = \begin{cases} \frac{\sigma_{sv}}{\sigma_{sw}} & \text{für } t = w \\ \frac{\sigma_{sv}}{\sigma_{sw}} \frac{\sigma_{st}(v)}{\sigma_{st}} & \text{für } t \neq w \end{cases}$$

Setzt man das nun in $\delta_s(v)$ ein, erhält man die zu zeigende Gleichung.

$$\begin{aligned}
\delta_s(v) &= \sum_{w : v \in pred(s,w)} \sum_{t \in V} \delta_{st}(v, e)) && (2.66)\\
&= \sum_{w : v \in pred(s,w)} (\frac{\sigma_{sv}}{\sigma_{sw}} + \sum_{t \in V \setminus w} \frac{\sigma_{sv}}{\sigma_{sw}} \frac{\sigma_{st}(v)}{\sigma_{st}}) && (2.67)\\
&= \sum_{w : v \in pred(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_s(w)). && (2.68)
\end{aligned}$$

□

Algorithmus 2 bestimmt für ungewichtete Graphen die *betweenness*-Zentralität. **Zeile 1 - 9:** Die äußere For-Schleife iteriert über alle Knoten, sodass der Inhalt für jeden Knoten einmal ausgeführt wird. Um den betrachteten Knoten und dessen Nachbarknoten abzuspeichern, werden aus Effizienz-Gründen eine Warteschlange (Queue Q) und ein Stapel (Stack S) verwendet. Der aktuell betrachtete Knoten s wird am Anfang der äußeren For-Schleife an das Ende der Warteschlange hinzugefügt.

Zeile 10 - 24: Die While-Schleife arbeitet die Warteschlange ab. Zu Beginn wird ein Knoten vom Anfang der Warteschlange entfernt und auf den Stapel gelegt. Danach werden alle Nachbarn des Knoten s besucht und es wird entschieden ob sie ein Teil des kürzesten Pfades sind. Alle Nachbarn, die im kürzesten Pfad vorkommen, werden zur Warteschlange hinzugefügt. Außerdem wird die Anzahl der kürzesten Pfade für den jeweiligen Nachbarknoten angepasst und die Vorgänger werden gesetzt.
Zeile 25 - 36: Die zweite While-Schleife arbeitet den Stapel ab, berechnet die Abhängigkeiten, die in der Zeile 25 initialisiert werden - mithilfe der Berechnungsvorschrift 2.61 - für jeden Knoten und anhand dessen die *betweenness*-Zentralität des Knotens. Um die *betweenness*-Zentralität weiter zu erläutern, betrachte man folgendes Beispiel.

Beispiel

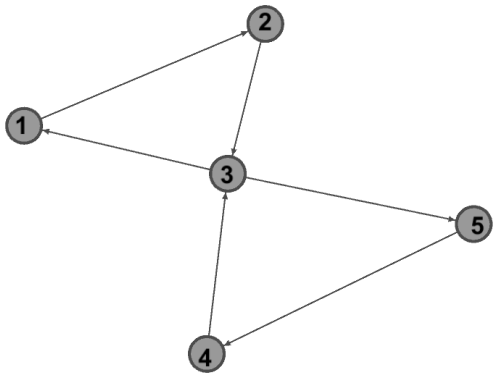


Abbildung 2.6.: Graph zur Berechnung der *betweenness*-Zentralität

Als erstes müssen alle kürzesten Pfade des Graphen der Abbildung 2.6 berechnet werden. Dann erhält man beispielhaft folgende kürzesten Pfade deren Startpunkt der Knoten 1 ist:

$$1 \to 2$$
$$1 \to 2 \to 3$$
$$1 \to 2 \to 3 \to 5$$
$$1 \to 2 \to 3 \to 5 \to 4$$

Wenn alle kürzesten Pfade bestimmt wurden, müssen die Paar-Abhängigkeiten berechnet werden. Werden diese aufaddiert, erhält man die *betweenness*-Zentralität

für den jeweiligen Knoten. Somit wäre $\sum_{i=1}^{5} \delta_i(2)$ die *betweenness*-Zentralität für den Knoten 2.

$$\sum_{i=1}^{5} \delta_i(2) = \sum_{i=1}^{5} \sum_{j=1}^{5} \delta_{ij}(2) = \sum_{i=1}^{5} \sum_{j=1}^{5} \frac{\sigma_{ij}(2)}{\sigma_{ij}} \tag{2.69}$$

Das heißt man erhält $\frac{1}{1} + \frac{1}{1} + \frac{1}{1} = 3$, da Knoten 2 genau in 3 kürzesten Pfaden vorkommt in denen er kein Anfangs- und Endknoten ist (siehe Abbildung 2.69). Wird dies für alle Knoten berechnet, erhält man folgende Werte:

$$\begin{aligned} C_B(1) &= 3 \\ C_B(2) &= 3 \\ C_B(3) &= 10 \\ C_B(4) &= 3 \\ C_B(5) &= 3 \end{aligned}$$

Nun können die Werte - wie schon am Anfang des Kapitels erwähnt - normalisiert werden, indem man jeden Wert durch $(n-1)(n-2)$ dividiert, wobei $n = |V|$ ist. In diesem Fall also $(5-1)(5-2) = 4 \cdot 3 = 12$. Die normierten *betweenness*-Zentralitäten des Graphen sind:

$$\begin{aligned} C_B(1) &= 0.25 \\ C_B(2) &= 0.25 \\ C_B(3) &= 0.83\overline{3} \\ C_B(4) &= 0.25 \\ C_B(5) &= 0.25 \end{aligned}$$

Man erkennt eindeutig das Knoten 3 eine hohe Relevanz im Graphen hat, da er beide Teile des Graphen verbindet.

2.7. Zusammenfassung der Knoten-Zentralitäten

Dieses Unterkapitel dient als Zusammenfassung der bisher erwähnten Zentralitäten. Es werden alle Zentralitäten anhand eines Graphen bestimmt, um einen Vergleich darzustellen. Hierfür betrachte man folgende Abbildung 2.7. Betrachtet man die Tabelle 2.1 so erkennt man, dass die verschiedenen Zentralitäten nahezu dieselben Knoten als wichtig einordnen. Knoten 4 ist bei jedem Verfahren über 50%, da er die Subgraphen miteinander verbindet. Deshalb spielt er eine zentrale Rolle im Graphen. Bei der Knotengrad-Zentralität werden die Knoten $(1, 2, 3, 6, 7, 8)$ als gleich wichtig eingeordnet, im Gegensatz zu den anderen Verfahren. Die weiteren Zentralitäten betrachten $(1, 8)$ und $(2, 3, 6, 7)$ ähnlich. Bei der Bewertung von dem Knoten 5 weicht die *betweenness*-Zentralität auffallend von den anderen Zentralitäten ab. Da Knoten 5 in

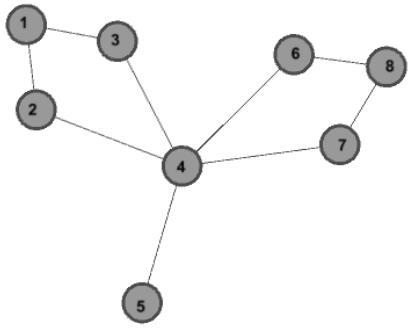


Abbildung 2.7.: Graph zur Berechnung der *betweenness*-Zentralität

Knoten	C_D	C_E	C_K	C_P	C_C	C_B
1	0.2857	0.2577	0.2854	0.3403	0.1750	0.0476
2	0.2857	0.3334	0.3376	0.3250	0.3888	0.2857
3	0.2857	0.3334	0.3376	0.3250	0.3888	0.2857
4	0.7142	0.6059	0.5636	0.5200	0.5384	0.7619
5	0.1428	0.2340	0.2519	0.2740	0.3333	0.0000
6	0.2857	0.3334	0.3376	0.3250	0.3888	0.2857
7	0.2857	0.3334	0.3376	0.3250	0.3888	0.2857
8	0.2857	0.2577	0.2854	0.3403	0.1750	0.0476

Tabelle 2.1.: Verschiedene Zentralitäten des betrachteten Graphen. Als α und β Wert wurden 0.3 und 0.2 gewählt.

keinem kürzesten Pfad als Zwischenknoten vorkommt, besitzt er eine Zentralität von 0, im Gegensatz zu der Closeness-Zentralität die dem Knoten 0.333 zuordnet. Bei den Werten der Eigenvektor-, Katz- und PageRank-Zentralität bemerkt man einen klaren Zusammenhang. Die Abweichung der Werte ist wesentlich kleiner als bei der Closeness- oder Knotengrad-Zentralität. Der Unterschied zwischen Eigenvektor- und Katz-Zentralität würde sich bei einem gerichteten Graphen deutlicher bemerkbar machen, da Knoten ohne ausgehende Kanten eine Zentralität von 0 erhalten würden.

Letztendlich unterscheiden sich die Werte der verschiedenen Verfahren, doch die globale Aussage über die Knoten des Graphen ist gleichartig. Knoten 4 hat die höchste Zentralität gefolgt von Knoten $2, 3, 6$ und 7.

2.8. Anwendungen/weiterführende Literatur

Zentralitäten werden an vielen Stellen verwendet, hauptsächlich beim Analysieren eines Sozialen- oder Biologischen-Netzwerks. Die weiterführende Forschung der *betweenness*-Zentralität ist in dieser Hinsicht sehr interessant, da einige Algorithmen darauf basieren. Es werden natürlich Verfahren gesucht, welche die Zentralität schneller und effizienter auswerten. Aber auch unkonventionelle Methoden bleiben nicht außen vor. M. E. J. Newman veröffentlichte einen Artikel (siehe [New05]), der sich mit der Berechnung der *betweenness*-Zentralität anhand randomisierter Traversierung beschäftigt. Der Gedanke dahinter ist, dass für jeden Knoten gezählt wird wie oft er besucht wurde bei einem zufälligen Durchlaufen des Graphen. Des Weiteren wurde der im Kapitel 2.6 gezeigte Algorithmus von Ulrik Brandes, der für eine schnelle und kostengünstige Berechnung der Zentralitäten parallelisiert. Dies wurde im Artikel [Mad+09] ausführlich veranschaulicht. Durch das Parallelisieren kann ein Speed-Up von nahezu 5 erzielt werden. Eine weitere Verwendung beschreibt der Artikel „Skill characterization based on betweenness“ (siehe [ŞB09]). Der in dem Artikel beschriebene Algorithmus bedient sich der *betweenness*-Zentralität, um sich für bestimmte Aufgaben automatisiert die Fähigkeiten anzueignen. Dieser Algorithmus stammt aus dem Bereich der künstlichen Intelligenz.

Algorithmus 2 Algorithmus zur Berechnung der *betweenness*-Zentralität, Ulrik Brandes (siehe [Bra01]).

Require: $C_B[v] \leftarrow 0, v \in V$;
1: **for** $s \in V$ **do**
2: // *Initialisierung der einzelnen Komponenten.*
3: $S \leftarrow$ empty stack;
4: $P[w] \leftarrow$ empty list, $w \in V$;
5: $\sigma[t] \leftarrow 0, t \in V$; $\sigma[s] \leftarrow 1$;
6: $d[t] \leftarrow -1, t \in V$; $d[s] \leftarrow 0$;
7: $Q \leftarrow$ empty queue; enqueue $s \rightarrow Q$;
8: // *Für jedes Element der Warteschlange werden die Nachbarn besucht,*
9: // *kürzeste Pfade werden aufgebaut.*
10: **while** Q *not empty* **do**
11: dequeue $v \leftarrow Q$; push $v \rightarrow S$;
12: **for all** *neighbor* w *of* v **do**
13: // *Unbetrachtete Knoten werden an die Warteschlange* Q *angehängt.*
14: **if** $d[w] < 0$ **then**
15: enqueue $w \rightarrow Q$; $d[w] \leftarrow d[v] + 1$;
16: **end if**
17: **if** $d[w] = d[v] + 1$ **then**
18: // *Anzahl der kürzesten Pfade werden angepasst,*
19: // v *wird als Vorgänger des Knotens* w *gesetzt.*
20: $\sigma[w] \leftarrow \sigma[w] + \sigma[v]$;
21: append $v \rightarrow P[w]$;
22: **end if**
23: **end for**
24: **end while**
25: $\delta[v] \leftarrow 0, v \in V$;
26: **while** S *not empty* **do**
27: pop $w \leftarrow S$;
28: // *Die Paar-Abhängigkeiten werden berechnet.*
29: **for** $v \in P[w]$ **do**
30: $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]} \cdot (1 + \delta[w])$;
31: // *Die betweenness-Zentralität des Knotens* w *wird gesetzt.*
32: **if** $w \neq s$ **then**
33: $C_B[w] \leftarrow C_B[w] + \delta[w]$;
34: **end if**
35: **end for**
36: **end while**
37: **end for**

3. Kanten-Zentralität

Im letzten Kapitel wurden die verschiedenen auf Knoten basierenden Zentralitäten besprochen. Dieses Kapitel beschränkt sich auf die *edge betweenness*, die *betweenness*-Zentralität für die Kanten des Graphen. Mit diesem Thema beschäftigten sich M. E. J. Newman und M. Girvan als erstes, dies wird aber im Laufe des Kapitels noch ausführlicher erklärt.

3.1. Theoretische Hintergründe

Wie schon erwähnt basiert die *edge betweenness* auf der *betweenness*-Zentralität. Für einen Graphen G ist zur Berechnung der *edge betweenness* jeder Kante die Auswertung aller kürzesten Pfade von G mithilfe der vorgesehenen Algorithmen (Dijkstra, Floyd, o.ä.) notwendig. Danach wird die Anzahl der kürzesten Pfade von s nach t, die Kante e enthalten, berechnet. Die *edge betweenness* einer Kante lässt sich sowohl bei gerichteten als auch bei ungerichteten Graphen bestimmen.

Definition 10. *Sei $G = (V, E)$ ein Graph und $e(i, j) \in E$ eine Kante, dann ist die Kanten betweenness-Zentralität von e definiert als:*

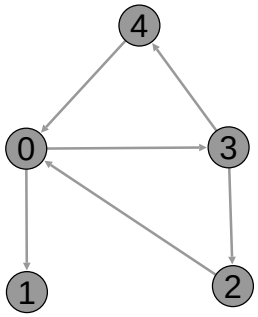
$$C_B^E(e) = \sum_{\substack{s,t \in V \\ s \neq t}} \frac{\sigma_{st}(e)}{\sigma_{st}} \tag{3.1}$$

wobei σ_{st} für die Anzahl der kürzesten Pfade - von Knoten s nach t - steht.

Die Berechnung der Kanten-Zentralität kann mithilfe des angepassten Floyd-Algorithmus(1) aus Kapitel 2.6 durchgeführt werden. In diesem Fall betrachtet man nicht die kürzesten Pfade in denen der Knoten v vorkommt sondern die Start- und Endknoten der Kante $e(i, j)$. Das Normalisieren der Werte kann über den maximalen Zentralitätswert geschehen. Die Laufzeit des modifizierten Floyd-Algorithmus ist $\Theta(|V|^3)$. Falls es erwünscht ist, können beide Zentralitäten gleichzeitig berechnet werden. Betrachte man folgendes Beispiel zum Verständnis des Vorgehens.

Beispiel

Sei $G = (V, E)$ der Graph aus der Abbildung 3.1. Berechnet man nun die kürzesten Pfade mit dem Algorithmus von Dijkstra (siehe [Lei+01]), erhält man die kürzesten Pfade Ergebnisse aus Tabelle 3.1.



Kante	Edge Betweenness
0 -> 3	1.0000
4 -> 0	0.8571
2 -> 0	0.7142
3 -> 4	0.7142
3 -> 2	0.7142
0 -> 1	0.5714

Abbildung 3.1.: Der Beispielgraph und *edge betweenness*-Werte.

Jetzt muss für jede Kante e geprüft werden in wie vielen kürzesten Pfaden sie enthalten ist. Beispielweise ist die Kante $(3, 2)$ in 5 kürzesten Pfaden enthalten. Somit hat Kante $e(3, 2)$ einen *edge betweenness* Wert von $C_B^E(e) = 5$. Hat

0:	**1:**	**2:**	**3:**	**4:**
$0 \to 1$	-	$2 \to 0$	$3 \to 2$	$4 \to 0$
$0 \to 3$	-	$2 \to 0 \to 3$	$3 \to 4$	$4 \to 0 \to 1$
$0 \to 3 \to 2$	-	$2 \to 0 \to 3 \to 4$	$3 \to 2 \to 0$	$4 \to 0 \to 3$
$0 \to 3 \to 4$	-	-	$3 \to 4 \to 0$	$4 \to 0 \to 3 \to 2$
-	-	-	$3 \to 2 \to 0 \to 1$	-
-	-	-	$3 \to 4 \to 0 \to 1$	-

Tabelle 3.1.: Alle kürzesten Pfade des Beispielgraphen.

man die Werte für alle Kanten bestimmt, kann man sie normalisieren. Der maximale *edge betweenness* Zentralitätswert beträgt 7 (siehe Kante $(0, 3)$). Für jede wird nun der erhaltene Wert durch 7 dividiert und man erhält die Werte aus der Abbildung 3.1. Alternativ können die Werte auch anhand der Anzahl möglicher Kanten normalisiert werden. Bei einem Graphen mit n Knoten wäre das $(n - 1) \cdot (n - 2)$.

3.2. Praktische Umsetzung

Die *edge betweenness* wurde von M. E. J. Newman und M. Girvan eingeführt. Ihr Artikel „Community structure in social and biological networks“ wurde 2002 veröffentlicht. Der Algorithmus, der in diesem Artikel beschrieben wird, kann im Rahmen der *community analysis* genutzt werden. Das Konzept der

edge-betweenness wurde verwendet, um Graphen hierarchisch in Teilgraphen aufzuteilen.

Die *community analysis* ist ein Bereich der Informatik, die sich mit der Identifizierung von Gruppen, *Cliquen* oder auch sogenannten *communities* beschäftigt (siehe [ZAL14]). Diese Gruppierungen in zum Beispiel sozialen Netzwerken haben eine Gruppenkohäsion, anhand der sie bestimmt werden können. Es gibt verschiedene *community detection* Algorithmen, die nach verschiedenen Merkmalen *communities* bestimmen. Man teilt die *Community Detection* Algorithmen in zwei Kategorien ein: *Group-Based Community Detection* und *Member-Based Community Detection*. Der Girvan-Newman Algorithmus bestimmt die hierarchischen *communities* und gehört daher zu den *Group-Based community detection* Algorithmen. Das Auffinden der *communities* geschieht in folgenden Schritten(siehe [GN02]):

1. Berechne den *edge-betweenness* Wert zu jeder Kante im Netzwerk.
2. Entferne die Kante mit dem höchsten C_B^E Wert.
3. Berechne den *edge-betweenness* Wert zu den restlichen Kanten.
4. Führe Schritt 2-3 solange aus bis keine Kanten mehr vorhanden sind.

Durch das Entfernen der Kanten deren *edge-betweenness* höher ist, bilden sich unverbundene Gruppierungen, dies sind die *communities*. Die entfernten Kanten sind sogenannte *intergroup edges*, die Gruppierungen im Netzwerk verbinden. Es bleiben die schwachen Verbindungen innerhalb der *communities*. Durch Verwendung des Newman Algorithmus (siehe [New01]) ist die Berechnung der *betweenness*-Werte für einen Graphen $G = (V, E)$ in $\mathcal{O}(|V||E|)$ Zeit möglich. Da nach jeder entfernten Kante die *betweenness* neu berechnet werden muss, benötigt der Girvan-Newman Algorithmus asymptotisch $\mathcal{O}(|E|^2|V|)$ Zeit. Man

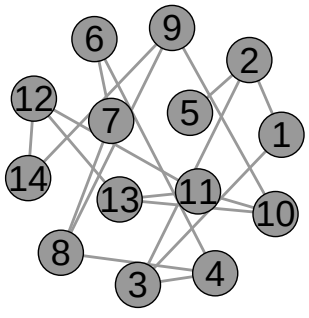


Abbildung 3.2.: Ein Beispielgraph der *communities* enthält.

betrachte folgendes Beispiel:

Beispiel

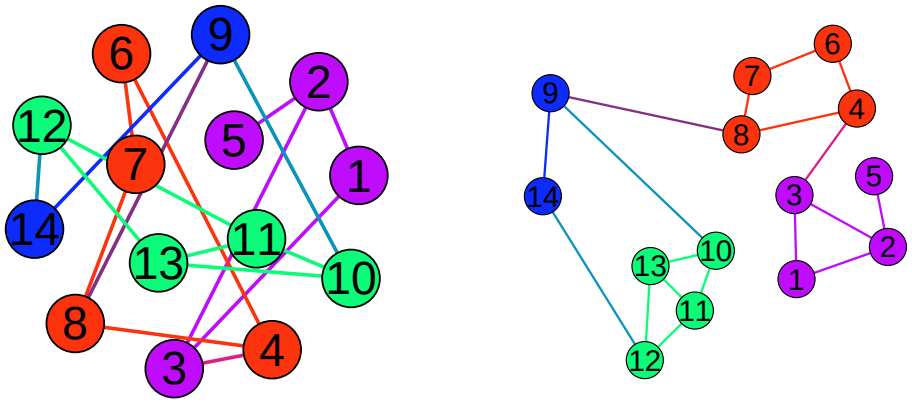
Sei $G = (V, E)$ der ungerichtete Graph aus der Abbildung 3.2. Es sollen nun die *communities* mithilfe des Girvan-Newman Algorithmus bestimmt werden. Der erste Schritt ist es, die *betweenness*-Werte der jeweiligen Kanten zu bestimmen und die Kante(n) mit dem maximalen Wert zu extrahieren. In diesem Beispiel wäre es, wie man in der Abbildung 3.3 erkennt, die Kante von Knoten 8 nach 9. Diese Kante wird nun entfernt, dadurch erhält man zwei

Abbildung 3.3.: Der Beispielgraph nach der ersten Iteration des Girvan-Newman Algorithmus.

Giant Components, die man auch als zwei *communities* auffassen kann. Nach dem Entfernen der Kante wird die *Edge-betweenness* erneut berechnet und die Kanten mit dem höchsten *betweenness* Wert werden wieder entfernt. In der dritten Iteration sieht der Graph wie in Abbildung 3.4 aus und die Kanten haben die dort sichtbaren *edge betweenness*-Werte. Wie schon erwähnt, werden

Abbildung 3.4.: Der Beispielgraph in der dritten Iteration des Girvan-Newman Algorithmus.

diese Schritte solange ausgeführt bis keine Kanten mehr vorhanden sind. In jeder Iteration werden mehr Kanten mit demselben *betweenness* Wert entfernt. Nach dem erfolgreichen Durchlaufen des Algorithmus kann entschieden werden wie viele *communities* man annehmen möchte. Bei diesem Beispiel sollen vier *communities* gefunden werden. Das Endergebnis wird in den Abbildungen 3.5 veranschaulicht.



a) Die vier gesuchten *communities* im Beispielgraph.

b) Umgeformter Graph basierend auf den *communities*.

Abbildung 3.5.: Das Endergebnis des Girvan-Newman *community detection* Algorithmus auf dem Beispielgraph.

3.3. Anwendungen/weiterführende Literatur

Das Konzept der *edge-betweenness* von M. Girvan und M. E. J. Newman wird an vielen Stellen verwendet, viele Algorithmen basieren darauf. Bei der Analyse von sozialen- und biologischen Netzwerken wird sehr häufig auf *edge-betweenness* beziehungsweise auf den Girvan-Newman Algorithmus zurückgegriffen. Im Artikel [Tei+13] wird eine weitere Form der Kanten-Zentralität eingeführt. Die sogenannte *Spanning Edge-Betweenness*, die statt kürzesten Pfaden minimale Spannbäume des Netzwerks betrachtet und anhand dessen die Kanten-Zentralität bestimmt. In dem Artikel [Yan05] wird ein paralleler Algorithmus eingeführt. Gruppierungen in biologischen Netzwerken können so noch schneller erkannt werden. Dafür werden die *edge betweenness*-Werte parallel berechnet. Eine weitere Verwendung der *edge-betweenness* findet man im Artikel [ATV07]. Es werden virtuelle *communities* bestimmt anhand der Musikrichtung, die der Benutzer bevorzugt. Zum Schluss ist noch der Artikel [Has+09] zu erwähnen, in dem unter Verwendung der *edge betweenness* und *betweenness*-Zentralität *(betweenness centrality)* ein *Layout-Algorithmus* konstruiert wird, um Graphen übersichtlicher visualisieren zu können.

4. Implementierung

In diesem Kapitel wird die Struktur und der Ablauf des angefertigten Programms erläutert. Insbesondere wird auf die Implementierung der *betweenness*-Zentralität für Knoten und Kanten eingegangen.

4.1. Architektur

Ablauf des Programms

Das Ziel war es ein Programm zu entwickeln, das aus einem Datensatz einen Graphen erstellt und dessen Knoten und Kanten bewertet. Die Bewertung sollte anhand der in Kapitel 2.6 und 3 eingeführten Zentralitäten geschehen. Der genaue Ablauf ist in der Abbildung 4.1 sichtbar. Das Programm nimmt ausschließlich Dateien des Formats .GEXF an. Mithilfe des Programms *Gephi*(siehe [BHJ09]) können Dateien eingelesen und als .GEXF Datei exportiert werden. Wenn die gewünschte Datei in das passende Format umgewandelt wurde, kann das Programm gestartet werden, um den Import auszuführen. Nach dem erfolgreichen Import wird der Graph visualisiert und die Algorithmen können anschließend ausgeführt werden. Standardmäßig werden alle berechneten Werte angezeigt und der Graph dementsprechend angepasst. Natürlich ist es auch möglich den Graphen samt berechneten Zentralitäten zu exportieren. Der Graph wird wie zum Zeitpunkt des Imports in eine .GEXF Datei exportiert. So können die Werte zu einem späteren Zeitpunkt wieder aufgerufen werden.

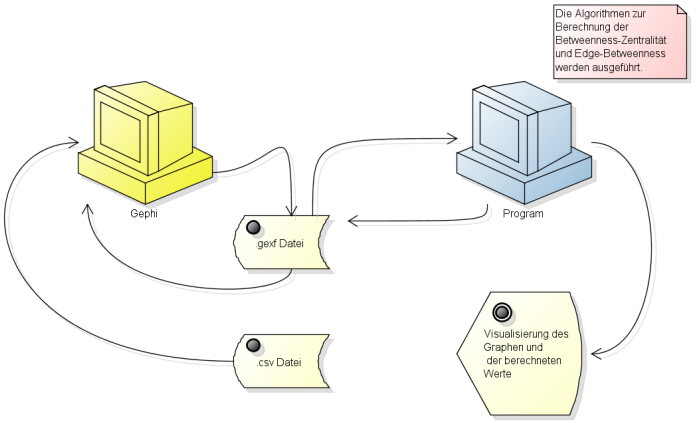


Abbildung 4.1.: Ablauf des Programms. Die visuelle Darstellung des Imports, Exports, der Berechnungen und der Visualisierung.

Externe Bibliotheken

Zum Importieren und Exportieren wurde das Gephi Tool-Kit verwendet. Obwohl das Tool-Kit ein Visualisierungs-Modul enthält, wurde stattdessen eine weitere Bibliothek verwendet. Die GraphStream Library (siehe [Dut+07]) ist eine dynamische Graph Bibliothek, daher kann der Graph zur Laufzeit verändert werden. Somit können zum Beispiel Knoten zur Laufzeit zum Graphen hinzugefügt oder auch entfernt werden. Um die Korrektheit der Algorithmen zu gewährleisten, wurden diese mithilfe von JUnit-Tests getestet. Ansonsten wurden keine externen Bibliotheken verwendet.

Struktur

Zum strukturierten Programmieren sollte man sich an einem Entwurfsmuster der Software-Entwicklung orientieren. Beispielsweise am MVC(*Model-View-Controller*) oder die daraus hervorgegangene Form MVP(*Model-View-Presenter*). Bei dem MVP Entwurfsmuster sind die Datenhaltungsklassen (Model) und die Visualisierungsklassen (View) voneinander getrennt und können über den Presenter kommunizieren. Es gilt dabei, dass für die Komponenten die Schnittstellen definiert werden müssen, welche in der Java Umgebung als *Interfaces* bekannt sind. Die Klassen-Struktur des Programms weicht aber etwas von dem MVP Entwurfsmuster ab, statt Schnittstellen wurden abstrakte Klassen erstellt. Wie man in der Abbildung 4.2 des UML Diagramms[1] erkennt, hat

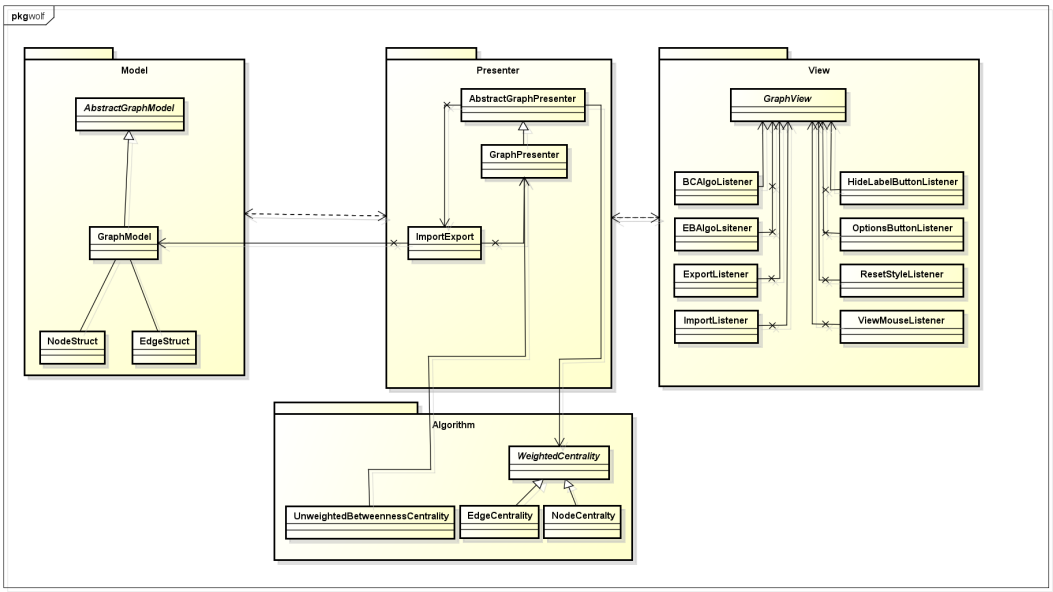


Abbildung 4.2.: UML Digramm zur Übersicht der Klassen-Struktur. Das ausführliche Diagramm ist im Anhang aufzufinden (siehe Abbildung A.1).

die `ImportExport` Klasse assoziative Verbindungen zu den Klassen `GraphModel` und `GraphPresenter`. Beim Starten des Programms wird dem `GraphPresenter` ein leeres `GraphModel` übergeben. Beim Importieren muss ein neues `GraphModel` erstellt und dem `GraphPresenter` übergeben werden, deswegen ist diese Verbindung notwendig. Es wurden noch weitere Konzepte aus der Software-Entwicklung verwendet. Die Synchronisation zwischen den Berechnungen und der Visualisierung wurde mithilfe des Beobachter Musters zustande gebracht. Natürlich wurde beachtet, dass sämtliche Berechnungen auf einem separaten *thread* durchgeführt werden und das der *event dispatching thread (EDT)* sich nur um die Visualisierung kümmert.

4.2. Umsetzung der Algorithmen

Wie schon im letzten Unterkapitel erwähnt wurden insgesamt zwei Algorithmen implementiert. Die *betweenness*-Zentralität für Knoten und Kanten. Bei der Knoten *betweenness*-Zentralität wird zwischen gewichteten und nicht gewichteten Graphen unterschieden. In Kapitel 2.6 wurden mehrere Möglichkeiten erläutert, um die Knoten *betweenness*-Zentralität zu berechnen. Hier wurde auf die Variante des Floyd Algorithmus zurückgegriffen, um alle kürzesten Pfade bei gewichteten Graphen zu berechnen und zu speichern. Bei ungewichteten Graphen ist der Algorithmus von Floyd nicht empfehlenswert. Aus diesem Grund wurde der Algorithmus von Brandes nach dem Algorithmus 2 implementiert.
Bei dem Floyd-Algorithmus kann nach der Abfrage, ob es sich beim aktuellen Pfad um einen Teil eines kürzesten Pfades handelt, der Zwischenknoten als Vorgänger in die Vorgängermatrix eingetragen werden.

```
1  // find the shortest Path and enter the predeccessor value
2  for (int k = 0; k < capacity; k++) {
3    for (int i = 0; i < capacity; i++) {
4      if ((distanceMatrix[i][k] != Double.POSITIVE_INFINITY))
5        for (int j = 0; j < capacity; j++) {
6          if (distanceMatrix[k][j] != Double.POSITIVE_INFINITY) {
7            if (((distanceMatrix[i][k] +
8                distanceMatrix[k][j]) < distanceMatrix[i][j])) {
9                //set the new shortest Path
10               distanceMatrix[i][j] = distanceMatrix[i][k]
11                                    + distanceMatrix[k][j];
12               predMatrix[i][j] = predMatrix[k][j];
13           }
14         }
15       }
16   }
17 }
```

Listing 4.1: Berechnung der kürzesten Pfade und das Vermerken der Vorgängerknoten.

[1]Zum Erstellen der Diagramme wurde das Programm Astah verwendet. Stand: 7.9.2015 URL:http://astah.net/

Die Vorgängermatrix muss zuerst initialisiert werden (siehe Listing A.1) und erst danach kann der Floyd-Algorithmus durchgeführt werden. Im Listing 4.1 sieht man, dass ein Knoten erst dann eingetragen wird, wenn die Kosten über diesen Knoten zu gehen kleiner sind als beim aktuellen kürzesten Pfad(anfangs ist dies die direkte Kante). Wenn dies geschehen ist, müssen anhand der Vorgängermatrix die kürzesten Pfade wiederhergestellt und abgespeichert werden. Dies geschieht in der `reconstructPath()`-Methode für jeweils zwei Knoten (siehe Listing 4.2). Die kürzesten Pfade werden für jeden Anfangs- und Endknoten in einer Liste abgespeichert. Die im Listing 4.2 zu sehende # dient als Trennzeichen. Die eigentliche Berechnung der *Betweenness Centrality* folgt nachdem für alle Knotenpaare die kürzesten Pfade bestimmt wurden.

```
1   private void reconstructPath(int i, int j) {
2     if (i == j) {
3       path +="#" + i + "#";
4     } else {
5       if (predMatrix[i][j] == null) {
6         path += "";
7       } else {
8         reconstructPath(i, predMatrix[i][j]);
9         path += j + "#";
10      }
11    }
12  }
```

Listing 4.2: Die kürzesten Pfade zwischen Knoten i und j werden wiederhergestellt und abgespeichert

Die Methode `countPathThrough(Integer v)` berechnet die Paar-Abhängigkeiten und summiert diese auf. Die Methode wird für jeden Knoten aufgerufen und das Ergebnis, also die *betweenness*-Zentralität, wird für den jeweiligen Knoten gesetzt. Dabei werden alle Bedingungen geprüft. Zum Beispiel darf der Anfangsknoten nicht der Endknoten sein beziehungsweise der betrachtete Knoten v darf weder Anfangs- noch Endknoten sein.
In der Funktion `calculateBetweennesCentrality()` wird dann `countPathThrough(Integer v)`(siehe Anhang Listing A.3) für jeden Knoten im Graph aufgerufen und der Wert wird in einem Vektor gespeichert. Dieser Vektor wird an den `GraphPresenter` weitergegeben, der dann die Zentralitätswerte im Graphen für jeden Knoten setzt. Die Berechnung der *edge betweenness* wurde nur für gewichtete Graphen implementiert. Zuerst wird die Methode im Listing A.1 ausgeführt, danach müssen die kürzesten Pfade wiederhergestellt werden. Um im nächsten Schritt die Kanten *betweenness*-Zentralität zu berechnen, wird die Methode `countPathThrough(Integer v)` erweitert (siehe Listing 4.3 oder im Anhang Listing A.2). Da die kürzesten Wege - zwischen i nach$j$ - gezählt werden müssen, in denen die betrachtete Kante vorkommt, sind die Übergabeparameter der Anfangs- und Endknoten der gewählten Kante. Ähnlich wie bei der Knoten *betweenness*-Zentralität wird die Paar-Abhängigkeit *delta* berechnet. In dieser wird gezählt wie viele kürzeste Pfade es von Knoten i nach j gibt, welche die betrachtete Kante enthalten. Diese Anzahl wird dann durch

die Zahl aller kürzesten Pfade von Knoten i nach j dividiert (siehe Listing 4.3). Diese Methode wird für jede Kante einmal aufgerufen, um die Zentralität zu bestimmen. Der `GraphPresenter` weist anschließend jeder Kante den Zentralitätswert zu.

```
1   if (i != j) {
2     if ((paths[i][j].get(k).startsWith("#" + i + "#"))
3         && (paths[i][j].get(k).endsWith("#" + j + "#"))) {
4       sigma++;
5       if (paths[i][j].get(k).contains("#" + source + "#" + target + "#")) {
6         sigmaE++;
7       }
8     }
9   }
```

Listing 4.3: Ein Teil der veränderten countPathThrough Methode zum Berechnen der Kanten *betweenness*-Zentralität. Die ganze Methode ist im Anhang unter Listing A.2 sichtbar.

5. Evaluation

Dieses Kapitel bezieht sich auf die Auswertung von Datensätzen. Es werden verschiedene Daten verwendet und die erhaltenen Werte werden verglichen. Die Datensätze wurden nach verschiedenen Kriterien ausgewählt. Es wurde auf eine kleine Anzahl der Knoten und Kanten Wert gelegt. Essentiell für die *edge betweenness* war es, dass der Graph *communities* enthält, damit der Girvan-Newman Algorithmus anschauliche Ergebnisse liefert.

5.1. Daten

Insgesamt wurden drei Datensätze verwendet und ausgewertet. Es wurden möglichst kleine Datensätze verwendet, um den Zeitaufwand der Berechnung der Zentralitäten zu minimalisieren.

- „Zachary's karate club“ [Zac77]
- „Political Blogs“ [AG05]
- „Les Miserables“ [KKK93]

„Zachary's karate club“ ist ein bekanntes Netzwerk eines Universität Karate-Klubs, das als ungerichteter und ungewichteter Graph modelliert wird. Dieses Beispiel Netzwerk ist sehr verbreitet in der *community analysis*. Der zweite Datensatz enthält Verbindungen zwischen politischen *Blogs* aus den USA. Dieser Graph ist gerichtet und ungewichtet. Zum Schluss wurde noch der ungerichtete und gewichtete Graph *Les Miserables* untersucht. Der Graph enthält Wörter aus dem Roman von *Victor Hugo*.

Auf den verschiedenen Graphen wurden verschiedene Algorithmen ausgeführt. Die Ergebnisse dazu folgen im nächsten Unterkapitel.

5.2. Ergebnisse

Zachary's karate club: Auf diesen Graphen wurde die Methode zur Berechnung der Betweenness-Zentralität für ungewichtete Graphen angewendet. Da diese Methode basierend auf dem Algorithmus von Brandes implementiert wurde, benötigt die Berechnungen $\mathcal{O}(|E||V|)$ Zeit. Der Graph besitzt 77 Knoten und 254 Kanten, davon sind folgende Knoten am zentralsten: 1, 43, 33,

3, 32. Knoten 1 hat eine Zentralität von 0.43 (siehe Abbildung 5.1) und wird daher auch als roter Knoten visualisiert. Der orange Knoten ist der Knoten 34 und hat einen Zentralitätswert von 0.304. Diese zwei Knoten stehen im Klub für den Trainer und den Präsidenten des Klubs. Des Weiteren sind die Knoten 33, 3, 32 als grüne Knoten gekennzeichnet, das heißt ihr Wert unterscheidet sich nur minimal.

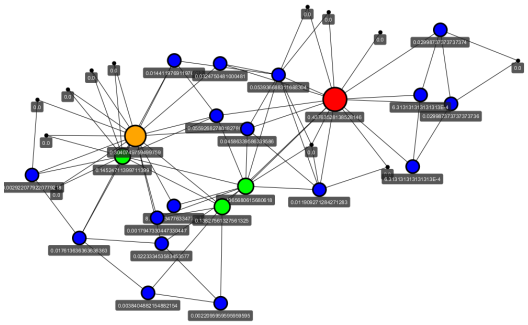


Abbildung 5.1.: Betweenness-Zentralitätswerte im Graphen „Zachary's karate club"

Political Blogs: Der Graph der politischen Blogger mit 224 Knoten und 19025 Kanten, hat wesentlich mehr Verbindungen als der vorherige. Die größte Zentralität unter den Bloggern hat der Knoten mit Namen *blogsforbush.com* mit einem Wert von 0.1461, gefolgt von *atrios.blogspot.com* mit 0.06088 und *instapundit.com* mit 0.05103. Die Abbildung 5.2 zeigt das Ranking der Knoten nach der Betweenness-Zentralität.

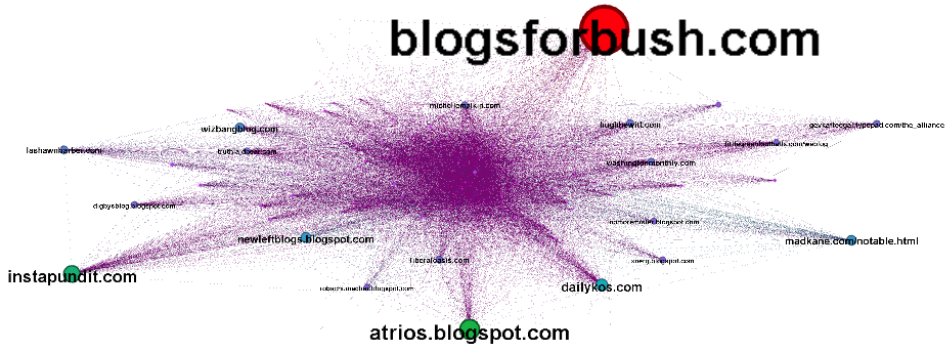


Abbildung 5.2.: *betweenness*-Zentralität Ranking im Graphen „Political Blogs" visualisiert mit Gephi.

Les Miserables: Da es sich hier um einen gewichteten Graphen handelt wurde, die Betweenness-Zentralität mithilfe des Floyd-Algorithmus berechnet.

Aus den 77 Knoten und 254 Kanten kristallisieren sich 6 besonders zentrale Knoten heraus. Die Werte der Knoten sind in der Tabelle 5.1 abgebildet. Der Algorithmus zur Bestimmung der *edge betweenness* lieferte folgende Wer-

Name	Zentralität
Valjean	0.4778
Gavroche	0.3259
Myriel	0.1768
Fantine	0.1438
Javert	0.1347
Theanardier	0.1301

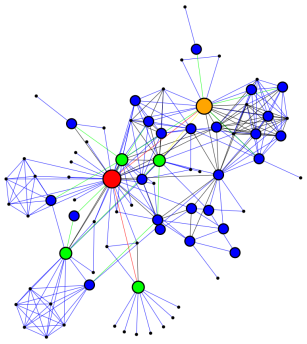


Tabelle 5.1.: Die Knoten Betweenness-Zentralitäten des Graphen Les Miserables.

te: Die Kanten, welche die Knoten mit einer hohen Betweenness-Zentralität verbinden, erhalten sinngemäß einen höheren *edge betweenness* Wert. Somit hat die Kante zwischen *Gavroche* und *Valjean* den höchsten Wert mit 0.1989, hingegen die Kante zwischen *MmeHucheloup* und *Gavroche* nur einen Wert von 0.022. Die zweit zentralste Kante ist die zwischen *Valjean* und *Myriel* mit einer *edge betweenness* von 0.1989. Es folgen noch die Kanten zwischen *Theanardier*→*Gavroche*, *Valjean*→*Marguerite* und *Thenardier*→*Fantine* mit den Werten 0.0940, 0.0680 und 0.0659. Insbesondere gibt es viele Kanten mit Zentralitätswert 0, da sie nicht in den kürzesten Pfaden vorkommen. Im Graph

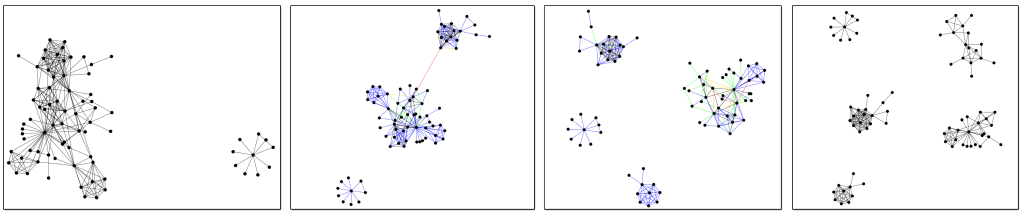


Abbildung 5.3.: Die Visualisierung der einzelnen zwischenschritte des Girvan-Newman Algorithmus[GN02].

lassen sich *communites* finden, daher wurde der Girvan-Newman Algorithmus mit Hilfe des Programms ausgeführt bis zum Erkennen von 5 Gruppierungen. Nach der vierten Iteration kristallisierte sich die erste *community* heraus wie man im ersten Bild der Abbildung 5.3 sehen kann. Nach jeder entfernten Kante wurde die *edge betweenness* neu berechnet und die Kante mit dem höchsten Wert wurde dann wiederum entfernt. Das zweite Bild zeigt die Iteration 23 an. Man sieht, dass sich noch eine Kante zwischen zwei Gruppierungen befindet. Diese Kante wurde von dem Programm mit roter Farbe gekennzeichnet, da

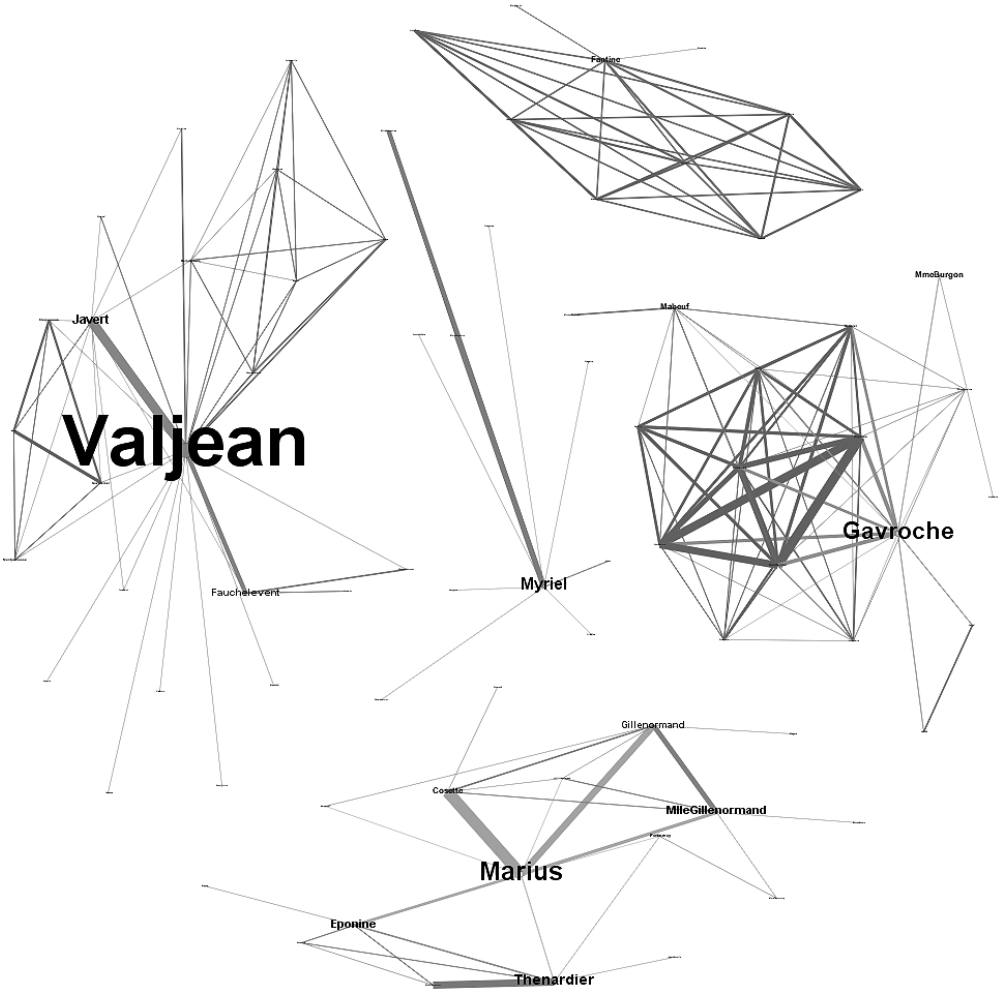


Abbildung 5.4.: Die Visualisierung der *communites*, die der Girvan-Newman Algorithmus[GN02] lieferte.

sie die höchste Zentralität besitzt. Die nächste *community* wurde nach dem Entfernen der Kante 23→11 in der 33sten Iteration sichtbar. Das Verfahren wurde bis zur 64sten Iteration durchgeführt nach der die fünfte *community* gefunden wurde (siehe Abbildung 5.3 viertes Bild).
In den fünf *communites* sind die Mittelpunkte die ersten sechs zentralsten Knoten aus der Abbildung 5.1. Es lässt sich ein klarer Zusammenhang zwischen der Knoten- und Kanten-Zentralität erkennen.

Die zwei Algorithmen zur Berechnung der Knoten-Betweenness-Zentralität unterscheiden sich in der Laufzeit immens. Dies ist schon bei einem Graphen der Größenordnung vergleichbar zu „Political Blogs“ bemerkbar. Für den „Political Blogs“ Graphen hat die Berechnung der Zentralitäten, basierend auf dem Floyd Algorithmus und unter einem Prozessor mit 2.4 GHz Taktfrequenz, 11

Minuten und 41 Sekunden benötigt. Der Algorithmus von Brandes hingegen benötigte bei demselben Graphen und unter selben Bedingungen nur 49 Sekunden.

6. Fazit

Die Zentralitätskonzepte der Graphen spielen eine wichtige Rolle. Mithilfe der Knoten- und Kanten-Zentralitäten können Zusammenhänge beziehungsweise Unterschiede zwischen den Komponenten identifiziert werden. Außerdem werden die Zentralitäten in verschiedenen Algorithmen verwendet. Als Teil der Suchalgorithmen in einer Suchmaschine (siehe Kapitel 2.4) bis zum identifizieren von Gruppierungen in sozialen- und biologischen-Netzwerken (siehe Kapitel 3.1). In den ersten Kapiteln wurden wie gefordert auf die wichtigsten Zentralitätskonzepte anhand von Beispielen eingegangen. Bei den Betweenness-Zentralitäten hat man feststellen können, dass es viele Möglichkeiten zur Realisierung gibt. Der aktuell schnellste und effizienteste Algorithmus ist der von Brandes (siehe [Bra01]). Es gibt dazu verschiedene Implementierungen zu den verschiedenen Typen von Graphen.
Das entwickelte Programm bietet eine gute Vorlage zum Einbinden von neuen Zentralitäten beziehungsweise zum Optimieren der implementierten Algorithmen. Es ist empfehlenswert bei größeren Datensätzen eine parallelisierte Version der Algorithmen zu verwenden. Durch die Implementierung von zwei Knoten basierten *Betweenness Centrality* Algorithmen konnten gute Vergleiche erzielt werden.
Bei der Visualisierung sollte bei größeren Graphen gegebenenfalls auf die *open Source Software Gephi* zurück gegriffen werden, da das Standard Layout von der *GraphStream* Bibliothek die Knoten schnell unübersichtlich positioniert.
Zusammenfassend lässt sich sagen, dass dieser Themenbereich weiterhin aktuell sein wird durch die verbreitete Anwesenheit der sozialen- und biologischen-Netzwerke aber auch durch die Verwendung von Graph-Strukturen in weiteren Bereichen.

A. Anhang

```
1   private void calculateShortestPathes() {
2
3     // initialize the Matrix with the predeccessor value
4     predMatrix = new Integer[capacity][capacity];
5     paths = new LinkedList[capacity][capacity];
6
7     // initialize the value in the predeccessor Matrix
8     for (int i = 0; i < capacity; i++) {
9       for (int j = 0; j < capacity; j++) {
10        paths[i][j] = new LinkedList<String>();
11        predMatrix[i][j] = null;
12        if (!(i == j || distanceMatrix[i][j] == Double.POSITIVE_INFINITY)) {
13          predMatrix[i][j] = i;
14        }
15      }
16    }
17
18    // find the shortest Path and enter the predeccessor value
19    for (int k = 0; k < capacity; k++) {
20      for (int i = 0; i < capacity; i++) {
21        if (distanceMatrix[i][k] < Double.POSITIVE_INFINITY)
22          for (int j = 0; j < capacity; j++) {
23            if (distanceMatrix[k][j] < Double.POSITIVE_INFINITY)
24              if (((distanceMatrix[i][k] + distanceMatrix[k][j]) <
                    distanceMatrix[i][j])) {
25                distanceMatrix[i][j] = distanceMatrix[i][k] + distanceMatrix[k
                      ][j];
26                predMatrix[i][j] = predMatrix[k][j];
27              }
28
29          }
30      }
31    }
32  }
```

Listing A.1: Der veränderte Floyd-Algorithmus.

```
1   protected double countPathThrough(Integer source, Integer target) {
2     double delta = 0;
3
4     for (int i = 0; i < capacity; i++) {
5       for (int j = 0; j < capacity; j++) {
6         double sigmaE = 0;
7         double sigma = 0;
8         for (int k = 0; k < paths[i][j].size(); k++) {
9           if (i != j) {
10            if ((paths[i][j].get(k).startsWith("#" + i + "#"))
11                && (paths[i][j].get(k).endsWith("#" + j + "#"))) {
12              sigma++;
13              if (paths[i][j].get(k).contains("#" + source + "#" + target + "#
                    ")) {
14                sigmaE++;
15              }
16            }
17          }
18        }
19        if (sigma == 0) {
20          delta += 0;
21        } else {
22          delta = delta + (sigmaE / sigma);
23        }
24      }
25    }
26    return delta;
27  }
```

Listing A.2: Die Methode countPathThrough zum Berechnen der Betweenness Centrality Werte für Kanten.

```
1   protected double countPathThrough(Integer v) {
2     double delta = 0;
3     for (int i = 0; i < capacity; i++) {
4       for (int j = 0; j < capacity; j++) {
5         double sigmaV = 0;
6         int sigma = 0;
7         for (int k = 0; k < paths[i][j].size(); k++) {
8           if (!(paths[i][j].get(k).startsWith("#" + v + "#"))
9               && !((paths[i][j].get(k).endsWith("#" + v + "#")))) {
10            if ((paths[i][j].get(k).startsWith("#" + i + "#"))
11                && (paths[i][j].get(k).endsWith("#" + j + "#"))) {
12              sigma++;
13              if (paths[i][j].get(k).contains("#" + v + "#")) {
14                sigmaV++;
15              }
16            }
17          }
18        }
19        if (sigma == 0) {
20          delta += 0;
21        } else
22          delta = delta + (sigmaV / sigma);
23      }
24    }
25    return delta;
26  }
```

Listing A.3: Die Methode countPathThrough zum Berechnen der Betweenness Centrality Werte.

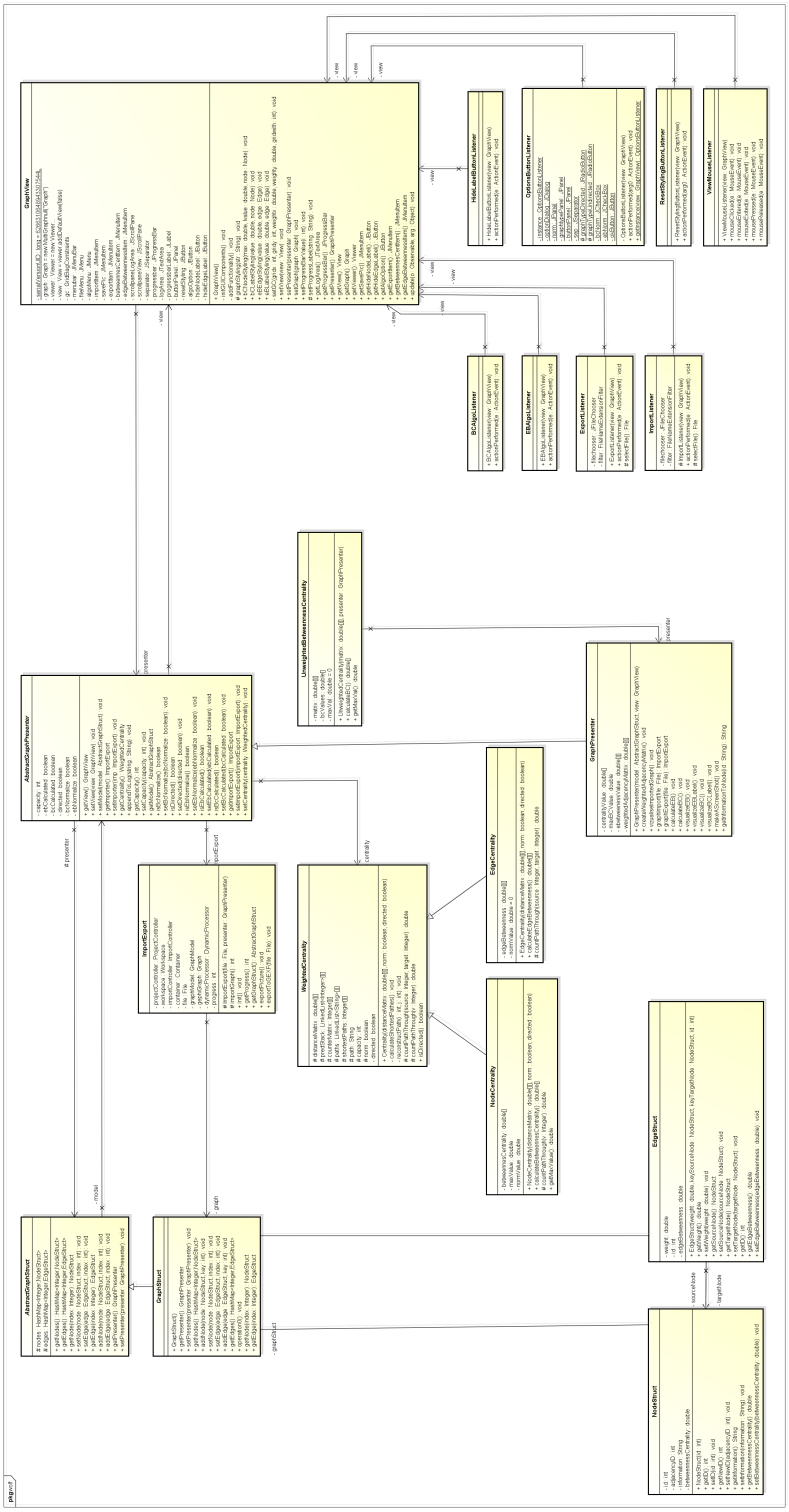


Abbildung A.1.: Detailiertes UML Diagramm.

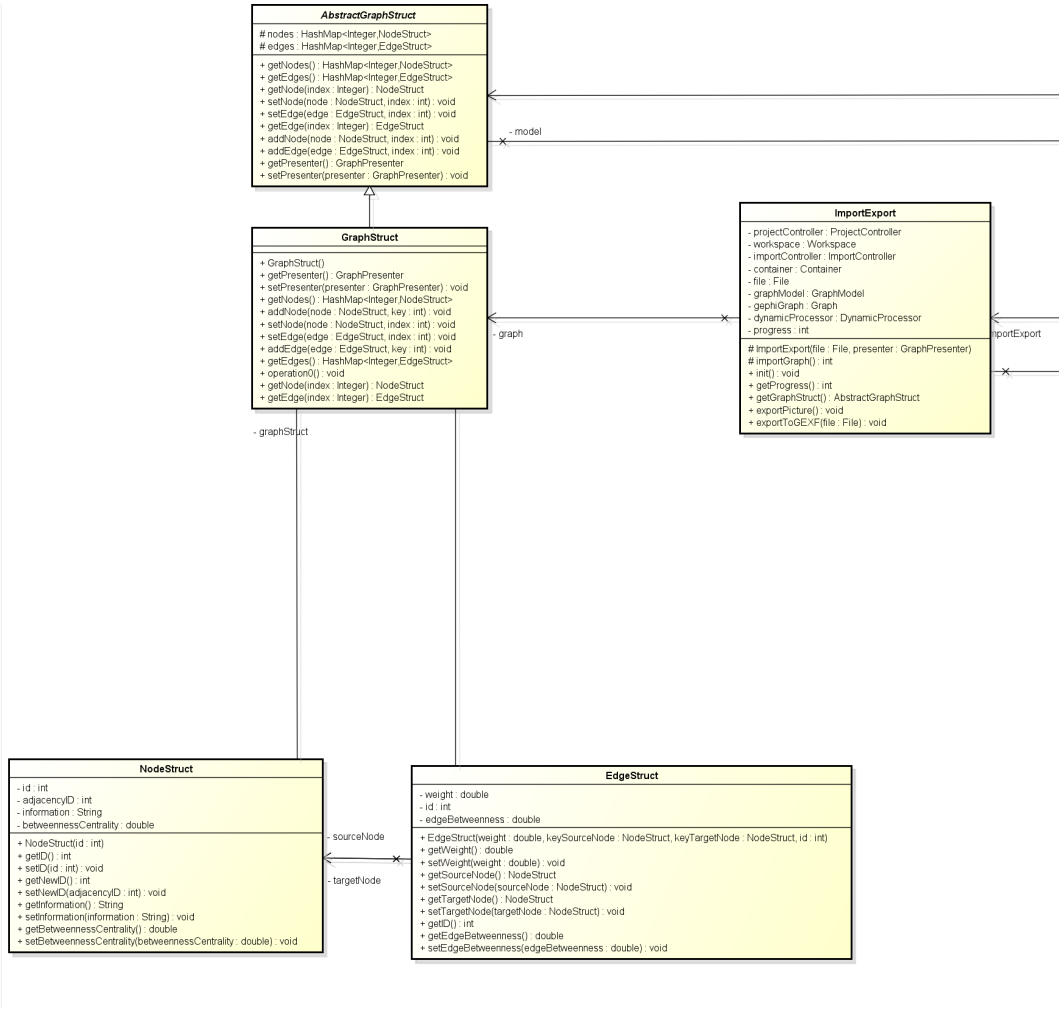


Abbildung A.2.: Detailiertes UML Diagramm. (Model)

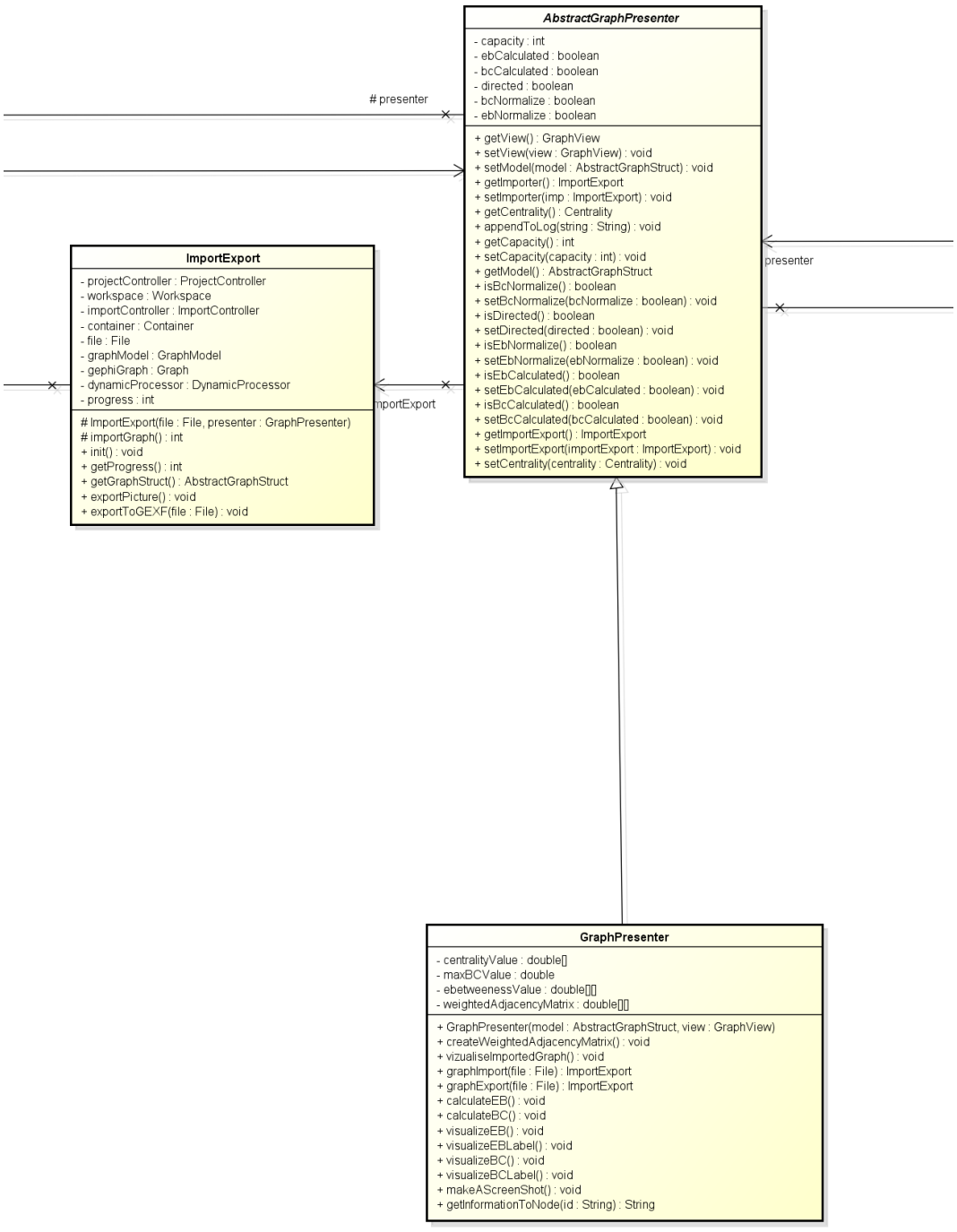


Abbildung A.3.: Detailiertes UML Diagramm. (Presenter)

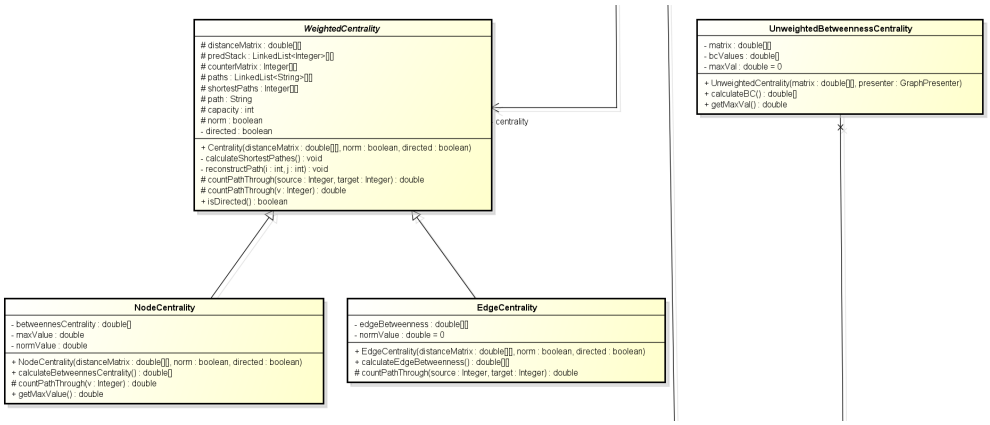


Abbildung A.4.: Detailiertes UML Diagramm. (View)

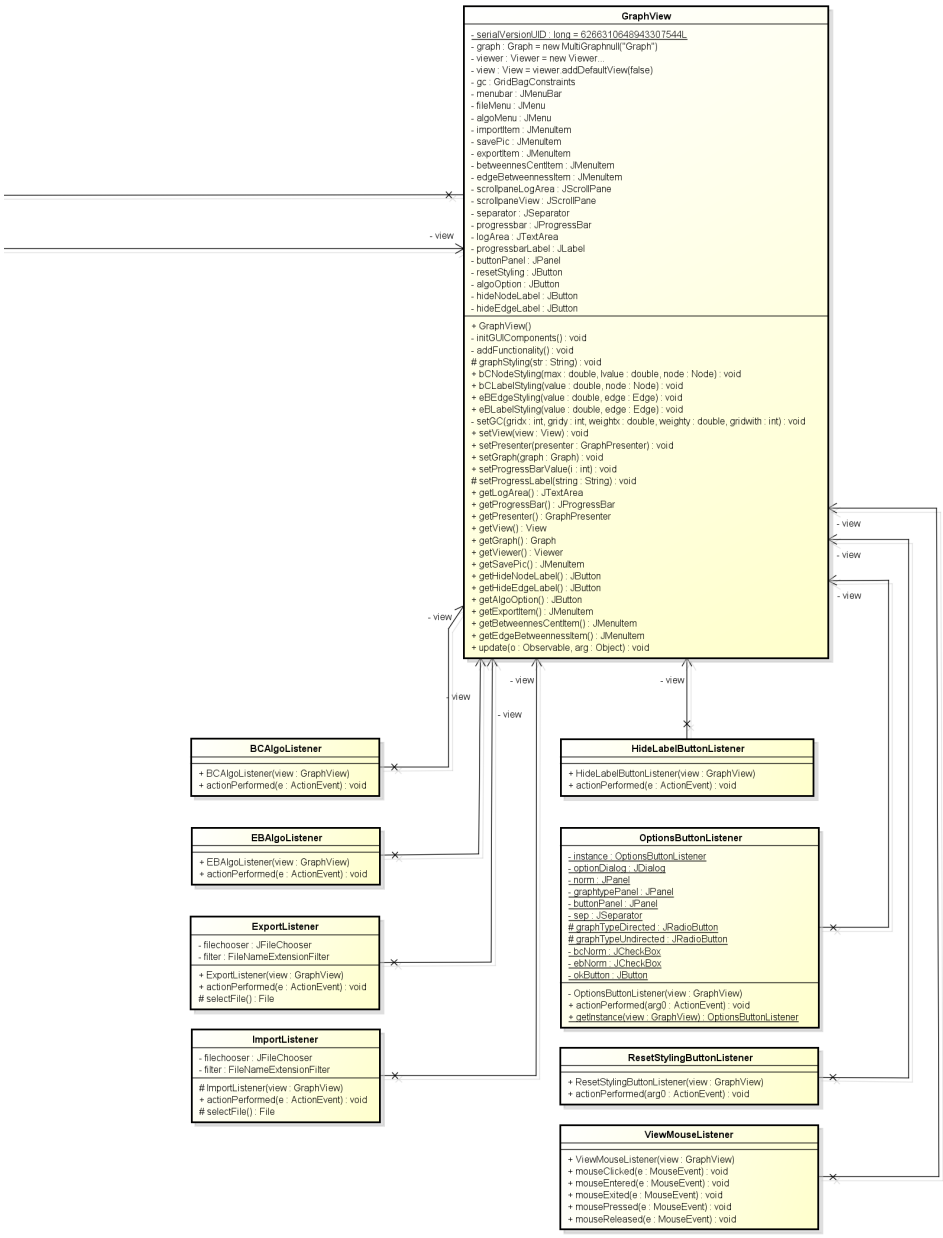


Abbildung A.5.: Detailiertes UML Diagramm. (View)

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[AG05] Lada A Adamic und Natalie Glance. „The political blogosphere and the 2004 US election: divided they blog“. In: *Proceedings of the 3rd international workshop on Link discovery*. ACM. 2005, S. 36–43.

[ATV07] Amélie Anglade, Marco Tiemann und Fabio Vignoli. „Virtual Communities for Creating Shared Music Channels.“ In: *ISMIR*. 2007, S. 95–100.

[BHJ09] Mathieu Bastian, Sebastien Heymann und Mathieu Jacomy. *Gephi: An Open Source Software for Exploring and Manipulating Networks*. 2009. URL: `http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154`.

[Bra01] Ulrik Brandes. „A faster algorithm for betweenness centrality*“. In: *Journal of Mathematical Sociology* 25.2 (2001), S. 163–177.

[Dut+07] Antoine Dutot u. a. „Graphstream: A tool for bridging the gap between complex systems and dynamic graphs“. In: *Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007)*. 2007.

[Fis13] Gerd Fischer. *Lineare Algebra: Eine Einführung für Studienanfänger*. Springer-Verlag, 2013.

[Flo62] Robert W Floyd. „Algorithm 97: shortest path“. In: *Communications of the ACM* 5.6 (1962), S. 345.

[Fre77] Linton C Freeman. „A set of measures of centrality based on betweenness“. In: *Sociometry* (1977), S. 35–41.

[GN02] Michelle Girvan und Mark EJ Newman. „Community structure in social and biological networks“. In: *Proceedings of the national academy of sciences* 99.12 (2002), S. 7821–7826.

[Has+09] Tatsunori B Hashimoto u. a. „BFL: a node and edge betweenness based fast layout algorithm for large scale networks“. In: *BMC bioinformatics* 10.1 (2009), S. 19.

[Juk07] Stasys Jukna. *Crashkurs Mathematik: für Informatiker*. Springer Science & Business Media, 2007.

[KKK93] Donald Ervin Knuth, Donald Ervin Knuth und Donald Ervin Knuth. *The Stanford GraphBase: a platform for combinatorial computing*. Bd. 37. Addison-Wesley Reading, 1993.

[KT06] Jon Kleinberg und Éva Tardos. *Algorithm design*. Pearson Education India, 2006.

[Lei+01] Charles E Leiserson u. a. „Introduction to algorithms“. In: *The MIT press* 5 (2001), S. 595–601.

[Mad+09] Kamesh Madduri u. a. „A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets“. In: *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE. 2009, S. 1–8.

[New01] Mark EJ Newman. „Scientific collaboration networks. I. Network construction and fundamental results“. In: *Physical review E* 64.1 (2001), S. 016131.

[New05] Mark EJ Newman. „A measure of betweenness centrality based on random walks“. In: *Social networks* 27.1 (2005), S. 39–54.

[ŞB09] Özgür Şimşek und Andre S. Barreto. „Skill Characterization Based on Betweenness“. In: *Advances in Neural Information Processing Systems 21*. Hrsg. von D. Koller u. a. Curran Associates, Inc., 2009, S. 1497–1504. URL: `http://papers.nips.cc/paper/3411-skill-characterization-based-on-betweenness.pdf`.

[Tei+13] Andreia Sofia Teixeira u. a. „Spanning edge betweenness“. In: *Workshop on Mining and Learning with Graphs*. Bd. 24. 2013, S. 27–31.

[WF94] Stanley Wasserman und Katherine Faust. *Social network analysis: Methods and applications*. Bd. 8. Cambridge university press, 1994.

[Yan05] Qiaofeng Yang. „A Parallel Algorithm for Clustering Protein-Protein Interaction Networks“. In: *Stanford University, CA*. 2005.

[Zac77] Wayne W Zachary. „An information flow model for conflict and fission in small groups“. In: *Journal of anthropological research* (1977), S. 452–473.

[ZAL14] Reza Zafarani, Mohammad Ali Abbasi und Huan Liu. *Social media mining: an introduction*. Cambridge University Press, 2014.

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über

Zentralitätskonzepte in Graphen

selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind

---

Armin Wolf, Münster, 19. September 2015