Eigenwertzentralitätsbestimmung

Computational Humanities

Implementierung des Algorithmus zur Eigenwertzentralität

Timo Homburg

23. August 2018

Inhaltsverzeichnis

1 Einleitung

Die Eigenwertzentralität ist ein wichtige Vorgehensweise zur Analyse von sozialen Netzwerken und zur Modellierung von Beziehungen zwischen wichtigen und unwichtigen Knoten eines Netzwerkes. So ist die Bestimmung der Eigenwertzentralität ist ein essentieller Bestandteil u.a. des Google Pagerank Algorithmus, über die die Analyse der Wichtigkeit von Webseiten von Google vorgenommen wird. Webseiten dienen hier als Knoten in Graphen und werden nach Wichtigkeit gestaffelt im Suchergebnis der Anfrage festgehalten. ZentralitätsmaSSe allgemein stellen bei der Analyse von Netzwerken aller Art eine groSSe Rolle und sind neben anderen Verfahren etablierter Bestandteil des Repertoires um Aussagen über Graphen zu treffen.

1.1 Problemstellung

Die Problemstellung dieses Projekt soll es sein das Konzept der Eigenwertzentralität zu implementieren und die Ergebnisse des Programmes für verschiedene Netzwerke zu testen sowie zu bewerten. Hierbei soll eine Bewertung in Form von Qualität und Güte, sowie im Vergleich zu anderen ZentralitätsmaSSen vorgenommen werden. Das Programm soll eine Übersicht über die Eigenwertszentralität des jeweiligen Graphen anhand der in ihm vorkommenden Knoten und dem Zentralitätswert geben. Zur Übersichtlichkeit sollen diese Werte vom Knoten mit der höchsten Zentralität zum Knoten mit der niedrigsten Zentralität absteigend sortiert sein. Zuletzt soll das Programm eine Ausgabe der Ergebnisse und die Eingabe der Ausgangsdaten in einem standardisierten Format gewährleisten können.

1.2 Motivation

In der Graphenanalyse existieren verschiedene MessgröSSen zur Bestimmung von Zentralitätswerten. Genannt werden sollen hier:

- Knotengradzentralität (Degree Centrality), die Ermittlung der Zentralität eines Knotens über dessen Grad, d.h. die Anzahl seiner Nachbarn
- Betweennesscentrality - Ein Aktor der zwischen vielen Akteurpaaren im Netzwerk auf dessen kürzesten Verbindungen zwischeneinander existiert
- Closenesscentrality - Ein Aktor der über viele kurze (kurz gewichtete) Verbindungen zu allen Akteuren im Netzwerk verfügt ist am Wichtigsten, da er keine Umwege über weitere Knoten nehmen muss um zu anderen Knoten zu gelangen
- Katz Zentralität - Eine Erweiterung der Eigenwertszentralität mit initialen Zentralitätswerten
- Pagerank

Die Eigenwertszentralität beruht im Gegensatz zu den o.g. Verfahren auf 2 einfachen Grundsätzen:

1. Knoten mit einem hohen Knotengrad sind wichtig (Degree Centrality)
2. Knoten die mit wichtigen Knoten verbunden sind, sind ebenfalls wichtig

Während ersterer Punkt bereits durch die Degree Centrality ausgedrückt wird, wurde der zweite Punkt jedoch von noch keinem o.g. Verfahren betrachtet. Die Kombination der beiden Punkte erlaubt uns also Aussagen darüber zu treffen welche Knoten sowohl in Abhängigkeit ihres Grades als auch durch die Verbundenheit zu wichtigen Knoten wichtig sind.

2 Lösungsweg

2.1 Definition

Die Eigenwertzentralität beschreibt die Wichtigkeit eines Knotens in einem Netzwerk anhand der Wichtigkeit seiner Nachbarn. Netzwerke werden durch Graphen und diese wiederum durch Adjazenzmatrizen definiert.

2.2 Ausgangslage

Grundlage des Projektes bildete Kapitel 7.2 in dem Buch Networks: An Introduction von Mark Newman **Newman:2010**, welches die Eigenwertzentralitätsberechnung beschreibt. Aufbauend auf dieser Information musste eine Implementierung geschaffen werden, welche die beschriebenen Schritte des Algorithmus umsetzt.

2.3 Vorgehensweise

Die Vorgehensweise wurde in die folgenden Abschnitte unterteilt:

- Klärung und Implementierung des Ausgangsformats und des Einlesevorgangs
- Implementierung des gegebenen Algorithmus
  - Implementierung der gegebenen Schritte
  - Export und Ausgabe der Ergebnisse
- Testen des Algorithmus mit gegebenen Testdaten
- Prüfung auf Plausibilität
- Bewertung

3 Implementierung

3.1 Ausgangsformat

Das Ausgangsformat des Projektes sollte laut Aufgabenstellung ein matrixabbildendes Format sein. Hierbei wurde auf das bewährte Konzept des Eingabeformates für Matrizen des Programms Matlab zurückgegriffen.
Das Format hat die folgende Struktur:

```
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

Die Elemente der Adjazenzmatrix, die die Eingabe definieren, werden durch Leerzeichen getrennt und durch einen eigens geschriebenen Parser eingelesen. Für die Eingabe musste darauf geachtet werden, dass das Eingabeformat, wie jede Adjazenzmatrix, eine quadratische Matrix darstellt. Im Parser wurde dahingehend ein Prüfungsmechanismus implementiert, der die Verarbeitung bei nichtquadratischen Matrizen beendet und dieses anhand einer Exception im Programm dokumentiert.

3.1.1 GraphML Import

Zum Testen von reellen Graphen bot es sich an ein Graphformat wie GraphML zu importieren und mit diesem Laufzeittests durchzuführen. GraphML wurde gewählt, da die Testdaten in GraphML vorlagen und GraphML als verbreitetes Graphformat in der Lehrveranstaltung genutzt wurde. Die Verwendung von GraphML bot zudem folgende Vorteile:

- Verwendung von Labels - Die Knoten konnten in der Ausgabe mit ihren korrekten Bezeichnern benannt werden
- Aufbau der Adjazenzmatrix nach den Bedürfnissen des Programmierers inklusive Wiederverwendungsmöglichkeit
- Testen von verschiedenen Fällen (Gerichtete/Ungerichtete/Gemischte Graphen)

Welches Dateiformat vorliegt entscheidet das Programm der Einfachheit halber anhand der Dateiendung. Im Folgenden sei ein Aussschnitt aus dem GraphML Eingabeformat gegeben:

```
<graph edgedefault="undirected" id="G" parse.edges="5820" parse.nodes="945" parse.order="free">
        <node id="n0">
          <data key="d0">
            <y:ShapeNode>
              <y:Geometry height="30.0" width="30.0" x="150" y="190"/>
              <y:Fill color="#CCCCFF" transparent="false"/>
              <y:BorderStyle hasColor="false" type="line" width="1.0"/>
              <y:NodeLabel alignment="center" autoSizePolicy="content" fontFamily="Dialog"
```

```
                fontSize="16" fontStyle="plain" hasBackgroundColor="false" hasLineColor="false"
                height="18" modelName="internal" modelPosition="c"
                 textColor="#000000" visible="true" width="48" x="−9.0" y="5">Traum</y:NodeLabel>
                <y:Shape type="ellipse"/>
              </y:ShapeNode>
            </data>
            <data key="d1"/>
          </node>
.....
<edge id="e0" source="n0" target="n1">
     <data key="d2">
       <y:PolyLineEdge>
         <y:Path sx="0.0" sy="0.0" tx="0.0" ty="0.0"/>
         <y:LineStyle color="#000000" type="line" width="2.33935"/>
         <y:Arrows source="none" target="none"/>
         <y:BendStyle smoothed="false"/>
       </y:PolyLineEdge>
     </data>
     <data key="d3"/>
   </edge>
.....
```

Man erkennt die im ersten Schritt einzulesenden Knoten, sowie die im zweiten Schritt einzulesenden Kanten. Hierbei bilden die Anzahl der Knoten die Dimension der Adjazenzmatrix, welche mit 0 vorinitialisiert wird. Über die Kanten kann anschlieSSend die Adjazenzmatrix an den entsprechenden Stellen mit 1 ausgestattet werden. Die Implementierung berücksichtigt sowohl ungerichtete und gerichtete als auch gemischte Graphen. Zur Unterscheidung wurden das Flag edgeDefault=undirected/directed/mixed im Graphknoten und das Flag directed=true/false in den Edgeknoten mit in die Betrachtung einbezogen und die Adjazenzmatrix dementsprechend berücksichtigt.

Beispiel eines Undirected Graphen:
<graph edgedefault="undirected" id="G" parse.edges="5820" parse.nodes="945" parse.order="free">
Beispiel einer beschrifteten Kante:
<edge id="e0" directed="true" source="n0" target="n1">

Als Ergebnis erhält das Programm eine der Graphbeschreibung entsprechenden Graphen, der für die weitere Berechnung der Eigenwertszentralität genutzt werden kann.

3.2 Architektur

Die Architektur des Programms lässt sich in verschiedene Funktionseinheiten aufteilen:

- Einlesen der Adjazenzmatrix inklusive Prüfung auf Wohlgeformtheit
- Anwendung des Eigenwertzentralitätsalgorithmus
- Ausgabe des Ergebnisvektors und Auflistung der Knoten nach Wichtigkeit

Das folgende Klassendiagramm soll eine Übersicht über die Beziehungen der nachfolgend beschriebenen Klassen bieten:

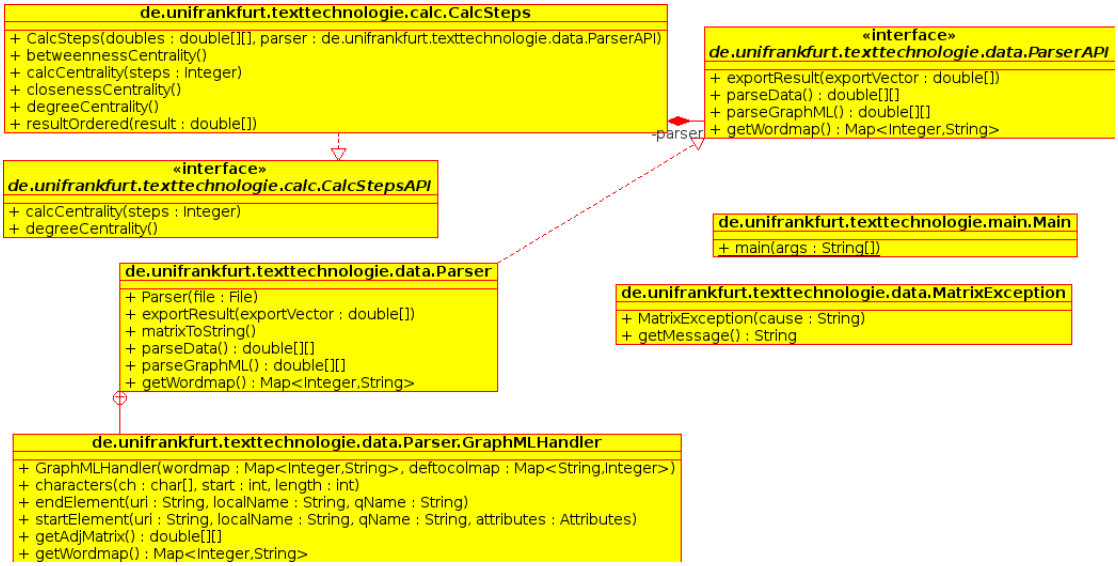

Abbildung 1: Klassendiagramm

- CalcStepsAPI: Interface für die Implementierung des Eigenwertzentralitätsalgorithmus
- CalcSteps: Implementierung des Eigenwertzentralitätsalgorithmus
- GraphMLHandler: Parsen eines GraphML Files für die Generierung der Adjazenzmatrix
- Main: Die Hauptklasse der Anwendung. Sie startet diese und führt die Prüfung der Parameter durch
- MatrixException: Eine Exception für den Fall, das die Matrix keine quadratische Matrix ist
- ParserAPI: Interface für den Import und Export
- Parser: Import und Exportklasse für das Einlesen von GraphML bzw. der Adjazenzmatrix

3.3 Implementierung des Eigenwertzentralitätsalgorithmus

Der Eigenwertzentralitätsalgorithmus lieSS sich in verschiedene Phasen unterteilen.

1. Zunächst wurde die gegebene Adjazenzmatrix durch den im Abschnitt Ausgangsformat beschriebenen Parser eingelesen und als Array gespeichert. Desweiteren wurde ein Ausgangsvektor von der Breite der Matrix erstellt. Dieser wurde mit dem Initialwert 1 vorinitialisiert und sollte die Ausgangsbasis für den iterativ zu ermittelnden Eigenwertsvektor darstellen.
   Beispiel:

```
Eingabematrix:

0 1 0 1 0 0
1 0 0 0 1 0
0 0 0 1 1 0
1 0 1 0 1 0
0 1 1 1 0 1
0 0 0 0 1 0

Startvektor:

1 1 1 1 1 1
```

2. Die Iterationsschritte, angegeben beim Start des Programms wurden im nächsten Schritt ausgeführt. Als Defaultwert wurde 30 angenommen. Bei der Iteration wurden die folgenden Operationen ausgeführt:
   a) Pro Iterationsschritt, nehme jede Zeile der Matrix mit der des aktuell errechneten Vektors mal
   b) Addiere die Ergebnisse auf und speichere sie im Resultvektor
   c) Normalisiere den Resultvektor durch Bilden des ABS-Wertes
   d) Setze den Resultvektor als neuen Startvektor für eine mögliche nächste Iteration
   e) Die Ergebnisse der Iterationsschritte wurden im Log festgehalten:

```
Step1:  [ 0,3244 0,3244 0,3244 0,4867 0,6489 0,1622  ]
Step2:  [ 0,3137 0,3765 0,4392 0,502 0,502 0,251  ]
Step3:  [ 0,3386 0,3144 0,3869 0,4837 0,6046 0,1935  ]
Step4:  [ 0,3073 0,3631 0,419 0,5121 0,5307 0,2328  ]
Step5:  [ 0,3369 0,3226 0,4014 0,4838 0,5878 0,2043  ]
Step6:  [ 0,3103 0,3559 0,4124 0,5103 0,5434 0,2262  ]
Step7:  [ 0,3333 0,3286 0,4055 0,4873 0,5791 0,2091  ]
Step8:  [ 0,3139 0,3511 0,4103 0,5071 0,5504 0,2228  ]
Step9:  [ 0,3302 0,3326 0,4069 0,4905 0,5739 0,2118  ]
Step10: [ 0,3167 0,3479 0,4095 0,5044 0,5548 0,2208  ]
Step11: [ 0,3279 0,3353 0,4075 0,4929 0,5705 0,2135  ]
Step12: [ 0,3187 0,3457 0,4091 0,5025 0,5576 0,2195  ]
Step13: [ 0,3263 0,3371 0,4079 0,4946 0,5682 0,2145  ]
Step14: [ 0,32 0,3442 0,4089 0,5011 0,5595 0,2186  ]
```

```
Step15:  [ 0,3252 0,3384 0,4081 0,4957 0,5667 0,2153  ]
Step16:  [ 0,3209 0,3432 0,4088 0,5002 0,5608 0,218  ]
Step17:  [ 0,3245 0,3392 0,4082 0,4965 0,5656 0,2158  ]
Step18:  [ 0,3216 0,3425 0,4087 0,4995 0,5616 0,2176  ]
Step19:  [ 0,324 0,3398 0,4083 0,497 0,5649 0,2161  ]
Step20:  [ 0,322 0,342 0,4086 0,4991 0,5622 0,2174  ]
Step21:  [ 0,3236 0,3402 0,4083 0,4974 0,5645 0,2163  ]
Step22:  [ 0,3223 0,3417 0,4086 0,4988 0,5626 0,2172  ]
Step23:  [ 0,3234 0,3405 0,4084 0,4976 0,5641 0,2165  ]
Step24:  [ 0,3225 0,3415 0,4085 0,4986 0,5629 0,2171  ]
Step25:  [ 0,3232 0,3406 0,4084 0,4978 0,5639 0,2166  ]
Step26:  [ 0,3226 0,3413 0,4085 0,4985 0,5631 0,217  ]
Step27:  [ 0,3231 0,3408 0,4084 0,4979 0,5638 0,2166  ]
Step28:  [ 0,3227 0,3412 0,4085 0,4984 0,5632 0,2169  ]
Step29:  [ 0,323 0,3408 0,4084 0,498 0,5637 0,2167  ]
```

3. Ausgabe: Die Ausgabe der Ergebnisse wurden sowohl als Ergebnisvektor, als auch als die Knoten absteigend nach Wichtigkeit sortiert implementiert.
   Beispiel mit dem Matrixeingabeformat:

```
Result:  [ 0,3227 0,3412 0,4085 0,4983 0,5633 0,2169 ]

Nodes in descending order:
Node 4 : 0.5632596998766394
Node 3 : 0.4982999893194831
Node 2 : 0.4084754335106465
Node 1 : 0.341155502213457
Node 0 : 0.3227457594285096
Node 5 : 0.21686758503236309
```

   Beispiel mit einem GraphML File:

```
Result:  [ 0,0175 0,0817 0,002 0,0901 0,1254 0,0075.....]
Nodes in descending order:
127: Schwester : 0.24969655010664715
110: Mutter : 0.24610051056775312
84: Vater : 0.24250026224618243
13: Zimmer : 0.22878675345257646
108: Tuer : 0.17637407730593413
141: Hand : 0.17170159005581054
206: Familie : 0.13726794389945596
68: Eltern : 0.12641402867421273
4: Kopf : 0.1254678961695582
65: Herr : 0.12394523090562015
302: Boden : 0.12205407634944478
16: Tisch : 0.11951616411829613
227: Frau : 0.11824348882475601
199: Prokurist : 0.11258621970483979
793: Zimmerherr : 0.10727008986240417
318: Wohnzimmer : 0.10486561381506192
292: Sessel : 0.10364684564288364
30: Fenster : 0.10004809594965769
62: in : 0.0996587959979052
```

```
221: Abend : 0.09445176973081698
34: Seite : 0.09385139515278958
.....
```

Die Ausgabe wird zudem zur möglichen weiteren Bearbeitung in ein Ausgabefile "output.txt" geschrieben.

3.4 Dokumentation

Für die Dokumentation der Quelltexte wurde JavaDoc verwendet. Das generierte Javadoc finden Sie in den Projektdateien[1]. Es wurden sämtliche Methoden, Klassen, Interfaces und Attribute, sowie Konstanten der Klassen dokumentiert und sofern als notwendig erachtet weitere Kommentare an ausgewählten Stellen der Methoden ergänzt.

3.5 Anwenderdokumentation

Die Anwendung ist in Java geschrieben und erfordert zur Ausführung eine Java Virtual Machine 1.7 oder höher. Zur Installation muss lediglich das JAR-File kopiert und mit dem Befehl

```
java -jar Eigenwert.jar filepath 30 //Dateipfad + Iterationsanzahl
```

ausgeführt werden.
Der Dateipfad muss hierbei einem Pfad zu einer Textdatei mit einer quadratischen (Adjazenz-)Matrix oder einer GraphML Datei wie in Sektion "Ausgangsformat" beschrieben enthalten.

3.5.1 Ergebnisse

Nach Ende der Ausführung des Programmes finden sich die Ergebnisdatei "output.txt" im Unterverzeichnis "out" des Programmes. Sie beinhaltet:

- Den Ergebnisvektor in einem matlabkompatiblen Format
- Die Knoten absteigend nach Zentralitätswert geordnet, wenn möglich mit Label

Hierbei kann eine Visualisierung des Ergebnisses beispielsweise durch Matlab erfolgen.
Ergebnisgraph mit Matlab erzeugen.... Knoten(X) und Zentralitätswert (Y)

[1] doc/index.html

4 Validierung des erstellten Programms

Das Programm musste nach Erstellung auf plausible Ergebnisse geprüft werden. Um eine Überprüfung der Richtigkeit vorzunehmen wurden die folgenden Schritte unternommen:

4.1 Analyse von Beispielgraphen

Um das Programm auf Korrektheit zu testen wurden einige Beispielgraphen angeschaut und die Ergebnisse des Programms anhand dieser auf Korrektheit geprüft.

4.1.1 Minimalbeispiel

Der von mir verwendete Graph als Minimalbeispiel war der Folgende:

```
0 1 0 1 0 0
1 0 0 0 1 0
0 0 0 1 1 0
1 0 1 0 1 0
0 1 1 1 0 1
0 0 0 0 1 0
```

Das Programm ermittelt für die Verteilung der Eigenwertzentralität folgende Ausgabe:

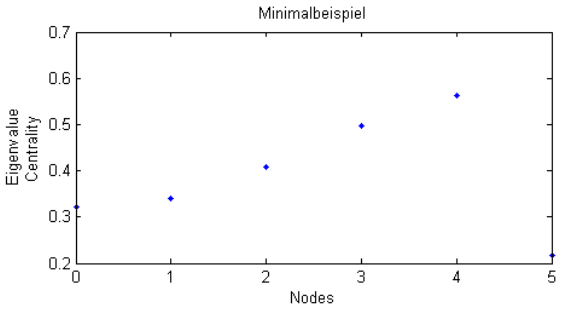

Abbildung 2: Verteilung der Eigenwertzentralität beim gewählten Minimalbeispiel

```
Nodes in descending order:
Node 4 : 0.5632596998766394
Node 3 : 0.4982999893194831
Node 2 : 0.4084754335106465
Node 1 : 0.341155502213457
Node 0 : 0.3227457594285096
Node 5 : 0.21686758503236309
```

Dieses entspricht den Erwartungen, da Knoten 4, die meisten Verbindungen, sowie Verbindungen zu den ebenfalls wichtigen Knoten 3 und 2 besitzt.

4.1.2 Franz Kafka - Die Verwandlung

Beispielgraph 1 bestand aus dem Wortnetzwerk des Textes Die Verwandlung von Franz Kafka:

Ausgehend von dem Kontext des Graphen wäre die Erwartungshaltung, dass die Eigenwertzentralität sich auf zentrale Worte der Geschichte wie

- Die Protagonisten
- Wesentliche Begriffe rund um die Verwandlung des Hauptprotagonisten Gregor Samsa
- Alltägliche Begriffe die in einen Haushalt gehören (die Geschichte spielt in einem Haus)

gruppiert.
Das Programm ermittelt für die Verteilung der Eigenwertzentralität folgende Ausgabe:

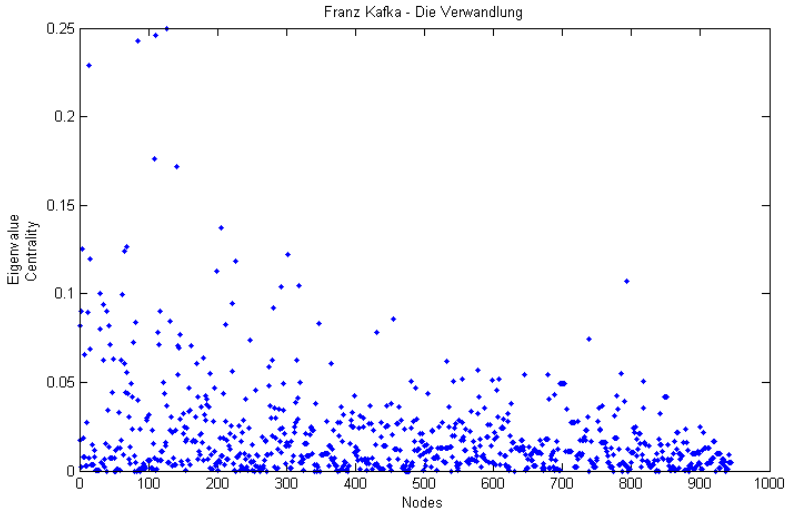

Abbildung 3: Verteilung der Eigenwertzentralität bei Kafkas Die Verwandlung

Die 10 Lemmata mit der höchsten Eigenwertzentralität waren:

127: Schwester : 0.24969655010664715
110: Mutter : 0.24610051056775312
84: Vater : 0.24250026224618243
13: Zimmer : 0.22878675345257646
108: Tuer : 0.17637407730593413
141: Hand : 0.17170159005581054
206: Familie : 0.13726794389945596
68: Eltern : 0.12641402867421273
4: Kopf : 0.1254678961695582
65: Herr : 0.12394523090562015

4.1.3 Artikelgraph Computer Science

Beispielgraph 2 bestand aus einem Artikelgraph über Informatik:
Hier wäre meine Erwartung gewesen, dass sich Begriffe als Knoten etablieren, die mit vielen anderen Begriffen der IT Welt in Beziehung stehne.
Das Programm ermittelt für die Verteilung der Eigenwertzentralität folgende Ausgabe:

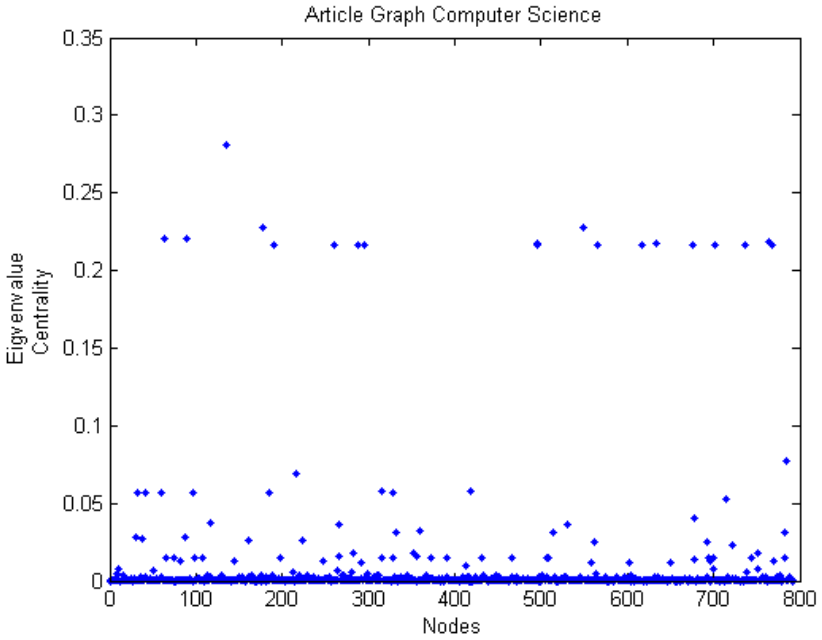

Abbildung 4: Verteilung der Eigenwertzentralität im Article Graph Computer Science

Die 10 Lemmata mit der höchsten Zentralität waren:

136: Internet Relay Chat : 0.28068735949035245
178: Internet Protocol : 0.22812944705612734
549: Gopher (Search Engine) : 0.22805749905572886
63: Secure Shell : 0.22022683379612942
89: Transport Layer Security : 0.22009216821641575
764: IPsec : 0.2182801856512766
495: Extensible Messaging and Presence Protocol : 0.2178440055187665
633: FTP : 0.21700221898092673
767: User Datagram Protocol : 0.21668200361272816
296: Telnet : 0.21658730589239772

Genannt werden überwiegend Protokolle und mit FTP ein Keyword der Informatik. Ohne die Artikel gelesen zu haben, würde ich die Ausgabe als authentisch betrachten.

4.1.4 Diseasome

Beispielgraph 3 bestand aus einem Graphen mit unterschiedlichen Krankheiten und deren Beziehungen:
Zentrale Schlagworte für Krankheiten bzw. Krankheiten die mit anderen Krankheiten in Verbindung stehen würden die Erwartungshaltung an diesen Graphen darstellen.
Das Programm ermittelt für die Verteilung der Eigenwertzentralität folgende Ausgabe:

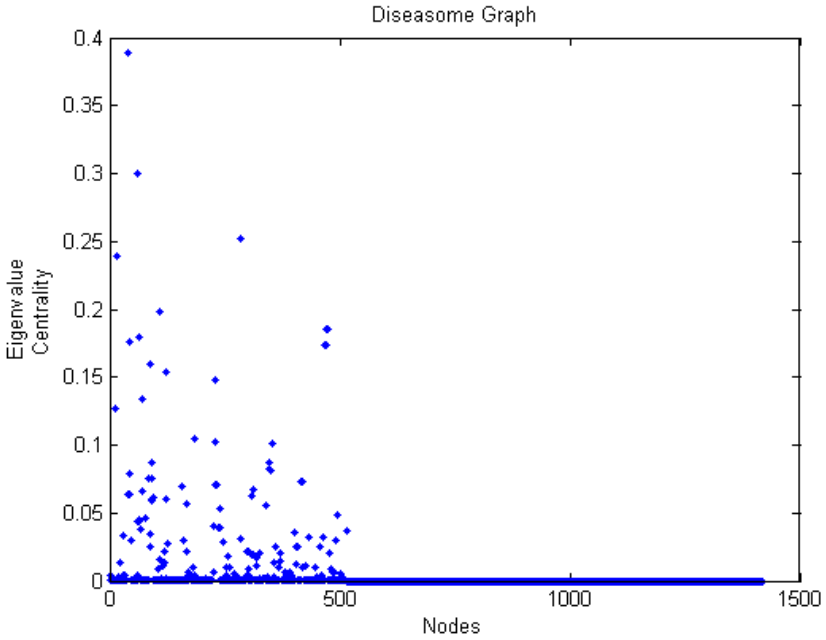

Abbildung 5: Verteilung der Eigenwertzentralität im Diseasome Graphen

Die 10 Lemmata mit der höchsten Zentralität waren:

40: Colon cancer : 0.3888795553195531
62: Breast cancer : 0.2996946928377661
286: Thyroid carcinoma : 0.25169571907866617
16: Pancreatic cancer : 0.2387400391301031
109: Hepatic adenoma : 0.19839357376266906
470: Li−Fraumeni syndrome : 0.18591899966657327
473: Osteosarcoma : 0.18591899966657327
64: Prostate cancer : 0.1790697282308446
43: Gastric cancer : 0.17596752331083734
468: Adrenal cortical carcinoma : 0.1732176870766493

4.1.5 Random Graph

Als letztes Beispiel wurde ein Random Graph verwendet, in dem es erwartungsgemäSS eine relativ niedrige Eigenwertszentralität geben sollte, die in einem begrenzten Umfeld liegt und relativ gleich verteilt ist. Das Ergebnis der Analyse zeigt dies wie erwartet:

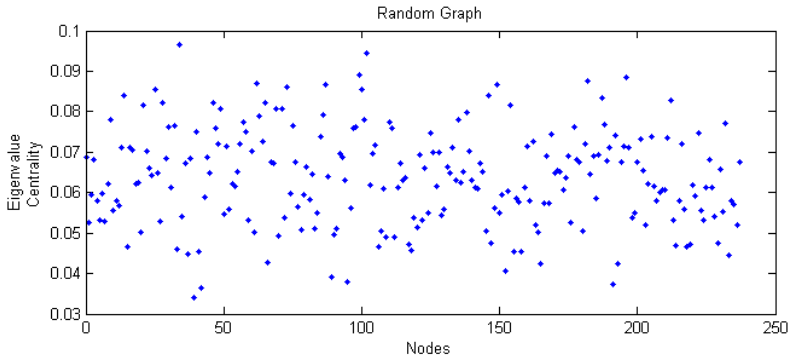

Abbildung 6: Verteilung der Eigenwertzentralität im Random Graphen

Die 10 Knoten mit der höchsten Zentralität waren:

```
34: 34 : 0.09659125062024143
102: 102 : 0.09441297933356685
99: 99 : 0.08902545313479526
196: 196 : 0.08862129729528338
182: 182 : 0.08756454003391524
62: 62 : 0.08686053595989886
149: 149 : 0.08670005945376014
87: 87 : 0.08666290692785925
73: 73 : 0.08617841942457295
100: 100 : 0.08559186759188087
```

4.2 Laufzeit

Die Laufzeit des Programmes hängt von der GröSSe der Matrix n und von der Anzahl der Durchläufe(Steps) s ab.
Bei jedem Durchlauf wird eine Komplexität von

$$O(s * n * n)$$

benötigt.

Liegt die Eingabedatei im GraphML Format vor, so erfolgt zusätzlich das Parsen der Datei mit einem SAXParser. Die Komplexität wird hierdurch jedoch nicht erhöht.

4.3 Kritik

Zum einen ist eine Einschätzung aufgrund der eingebauten Degree Centrality als eine gute Klassifikation einzustufen. Knoten im Netzwerk, die viele Verbindungen haben sollten in jedem Fall als für das Netzwerk wichtig und zentral klassifiziert werden. Allerdings bietet der zweite Ansatz Anlass zur Kritik:

An einem Beispiel deutlich gemacht, würde eine Homepage, die möglicherweise wenige Verbindungen zu anderen Homepages hat, aber in vielen Suchmaschinen (womöglich auf den hinteren Plätzen) verlinkt ist als wichtig gelten, da die Suchmaschine auf möglichst viele Homepages im Internet verlinken will. Evtl. ist diese Homepage jedoch keine wichtige Homepage, sonderen greift nur einige Schlagworte auf die in der Suchmaschine häufig gesucht werden. Hier seien zum Beispiel Crawlingwebseiten zu nennen, die es schaffen, Suchmaschinennutzer mit gezielten Suchworten auf ihre mit Werbung oder möglicherweise Schadsoftware bestückten Webseiten zu leiten. Dies kann natürlich auch bei vielen Verlinkungen auf der Crawlerwebseite der Fall sein, die selbstverständlich ebenfalls automatisiert gewonnen werden können. Weitere Filterungen zur Optimierung in beispielsweise diesem Anwendungsfall wären insofern nötig.

5 Fazit

Die Implementierung des Eigenwertzentralitätsalgorithmus konnte für die gegebenen Graphen die Eigenwertzentralität akkurat berechnen.
Die Laufzeit des Verfahrens ist mit quadratischer bzw. maximal kubischer Laufzeit im effizienten Bereich. Zusammenfassend lässt sich sagen, dass das Verfahren getestet auf verschiedenen Graphen ein, trotz der Kritik guten Anhaltspunkt bietet um Graphen wie z.B. soziale Netzwerke oder Texte zu analysieren. Mithilfe weiterer ZentralitätsmaSSe bzw. Koeffizienten lässt sich ein relativ aussagekräftiges Ergebnis erzielen.