Masterarbeit

Extraktion und Evaluierung von Hierarchien aus Informationsnetzwerken

Jürgen Zernig, BSc.

Institut für Wissensmanagement
Technische Universität Graz

Betreuer: Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic

Graz, im Mai 2013

Deutsche Fassung:

Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008

Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am 16. Mai 2013 _____

(Unterschrift)

Englische Fassung:

STATUARY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

May 16, 2013 _____

(signature)

Danksagung

An dieser Stelle möchte ich allen Personen danken, die mir während meines Studiums mit Rat und Tat zur Seite standen, speziell auch meinem Studienkollegen Jakob Pöllitsch, mit welchem ich jedes Motivationstief erfolgreich bekämpfen konnte.

Für die Betreuung von universitärer Seite bedanke ich mich bei Herrn Assoc.Prof. Dipl.-Ing. Dr.techn. Denis Helic.

Besonderer Dank gebührt meiner Familie, die mich die gesamte Ausbildungszeit hindurch unterstützte.

Graz, im Mai 2013 Jürgen Zernig

Kurzfassung

Das Thema dieser Arbeit beschäftigt sich mit der Erstellung und Evaluierung von Hierarchien, deren Zweck es ist, als Hintergrundinformation für die Navigation in einem Informationsnetzwerk zur Verfügung zu stehen um die Navigation effizienter zu bewerkstelligen. Speziell wird darauf eingegangen wie und ob die Qualität einer Hierarchie von der Anzahl der eingehenden und ausgehenden Kanten des, bei der Extraktion der Hierarchie, gewählten Wurzelknotens abhängt. Die Extraktion wurde mit Hilfe der Stanford Network Analysis Platform (SNAP) und einer modifizierten Breiten- (BFS) und Tiefensuche (DFS) durchgeführt. Als Datensatz wurde ein englischer Wikipedia-Dump verwendet aus dem ein Graph mit mehr als 9 Millionen Kanten und 255 Millionen Knoten entstanden ist. Die Evaluierung betrachtet speziell die Verteilung der Grade der entstandenen Hierarchien, die durchschnittlichen Pfadlängen, die durchschnittlichen Navigationspfadlängen (Stretch) und die Erfolgsraten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

Tabellenverzeichnis

0 Einleitung

Das Thema dieser Arbeit beschäftigt sich mit der Erstellung und Evaluierung von Hierarchien, die einem Simulator, der sich durch Informationsnetzwerke navigiert, als Input zur Verfügung stehen sollen. Diese Hierarchien sind notwendig für eine effiziente Navigation durch einen Informationsgraphen. Sie geben Aufschluss über Zusammenhänge eines Start- und Zielknoten im Graphen.

Eine Hierarchie ist ein hierarchischer Baum, der von einem willkürlichen Knoten aus gebildet wird. Der Simulator kann mit Hilfe der Hierarchie, sehr effizient den Abstand aller Knoten zum Zielknoten berechnen, wenn es möglich ist. Zur Navigation nimmt er denjenigen erreichbaren Knoten mit dem kürzesten Abstand zum Zielknoten. Das wiederholt er solange bis der Abstand 0 ist und er sich im Zielknoten befindet.

Eine Hierarchie kann einerseits im Durchschnitt sehr kurze Wege zu den Zielknoten beinhalten - bei kurzen Wegen ist die Hierarchie jedoch sehr breit gefächert. Die Möglichkeit, dass dann ein Weg zum Zielknoten nicht gefunden wird, weil sich der Zielknoten nicht in einer darunterliegenden Ebene des Startknotens befindet, ist bei breiter Hierarchie statistisch gesehen größer. In diesem Fall könnte der Simulator keinen effizienten Weg zum Zielknoten finden und benötigt solange andere Methoden zur Navigationshilfe, bis er sich wieder in einer Ebene, in der er die Hierarchie zur Navigation verwenden kann, befindet. Die Möglichkeit, dass er die Ebene nicht findet, ist natürlich gegeben.

Die Problematik besteht nun darin, Hierarchien so zu erstellen, dass sie im Durchschnitt kurze Wege beinhalten, und gleichzeitig die Wahrscheinlichkeit den Zielknoten nicht zu finden, minimal ist.

In dieser Arbeit wird die englischsprachige Wikipedia als Informationsnetzwerk untersucht. Um den Zusammenhang in diesem Informationsnetzwerk zu erfassen, wird ein Graph mit Knoten und Kanten erstellt. Ein Knoten im Graphen entspricht einem Artikel der Wikipedia und alle Verweise zu anderen Artikel entsprechen den Verbindungen bzw. Kanten zu den anderen Knoten.

Der Navigator benötigt für eine effiziente Navigation Hintergrundinformationen, die er über die erzeugten Hierarchien bekommen soll.

Die Hierarchie werde ich mit Hilfe der „Stanford Network Analysis Platform“ (SNAP) erstellen.

1 Einführung

1.1 Graphentheorie

Um einen guten Einblick in meine Arbeit zu ermöglichen, werde ich mich in diesem Kapitel mit grundlegenden Themen der Graphentheorie auseinandersetzen.

1.1.1 Definition Graphen

Im Buch „*Networks, Crowds, and Markets*“ von David Easley und Jon Kleinberg [1] wird ein Graph folgend beschrieben: Ein Graph ist eine Möglichkeit um Beziehungen von verschiedensten Objekten darzustellen. Ein Graph besteht aus einem Set von Objekten (**Knoten**) mit einigen Paaren dieser Knoten, die miteinander verbunden sind. Diese Verbindungen zueinander stellt man mit **Kanten** dar. Man betrachte die Graphen in Abbildung 1.1a und 1.1b. Der Graph in dieser Abbildung besteht aus sieben Knoten, hier mit 1 bis 7 beschriftet. Laut „Networks, Crowds, and Markets“ [1] sind nun zwei Knoten die miteinander eine Kante teilen **benachbart**. In 1.1a wären zum Beispiel 6 und 8, 6 und 3 oder 2 und 1 benachbart.

Die häufigste Darstellung von Graphen, ist anlehnend an Abbildung 1.1a: Knoten werden meistens als Kreise dargestellt und optional mit Zahlen und/oder Buchstaben identifiziert. Kanten sind Linien oder Pfeile wenn es sich um einen gerichteten Graphen handelt.

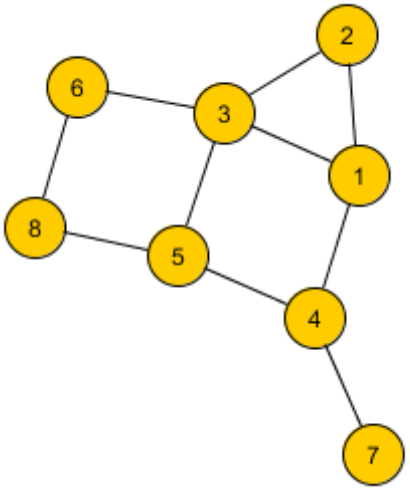


Abbildung 1.1a: Ein ungerichteter Graph

In „*Networks, Crowds, and Markets*“ [1] wird des Weiteren erwähnt, dass es symmetrische und unsymmetrische Verbindungen gibt. In Abbildung 1.1a handelt es sich beispielsweise um eine symmetrische Verbindung,

da keine Pfeile eingezeichnet sind. In Abbildung 1.1b sieht man, dass es eine unsymmetrische Verbindung sein muss, da sie nur in Pfeilrichtung existiert. In den meisten Fällen, werden wir von gerichteten Graphen sprechen, dass heißt ein Knoten A zeigt zu einem Knoten B aber nicht umgekehrt.

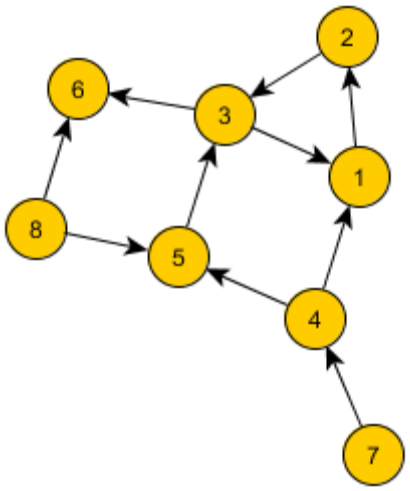


Abbildung 1.1b: Ein gerichteter Graph

Gerichtete Graphen sind Graphen mit unsymmetrischen Verbindungen. Es wird in der Regel von ungerichteten Graphen gesprochen [1], außer es wird explizit darauf hingewiesen, dass es sich um einen gerichteten handelt.

1.1.2 Blätter, Bäume und Wälder

Diestel [2] schreibt, dass ein Graph, der keinen Kreis enthält, ein Baum ist und somit ein Wald ein Graph, dessen Komponenten Bäume sind. Alle Knoten mit dem Grad 1 werden Blätter genannt. Des Weiteren definiert Diestel [2], dass ein Baum als minimale Anzahl mindestens 2 Blätter haben muss und wenn man ein Blatt eines Baumes entfernt, so ist der Rest noch immer ein Baum, sofern er 2 Blätter besitzt.

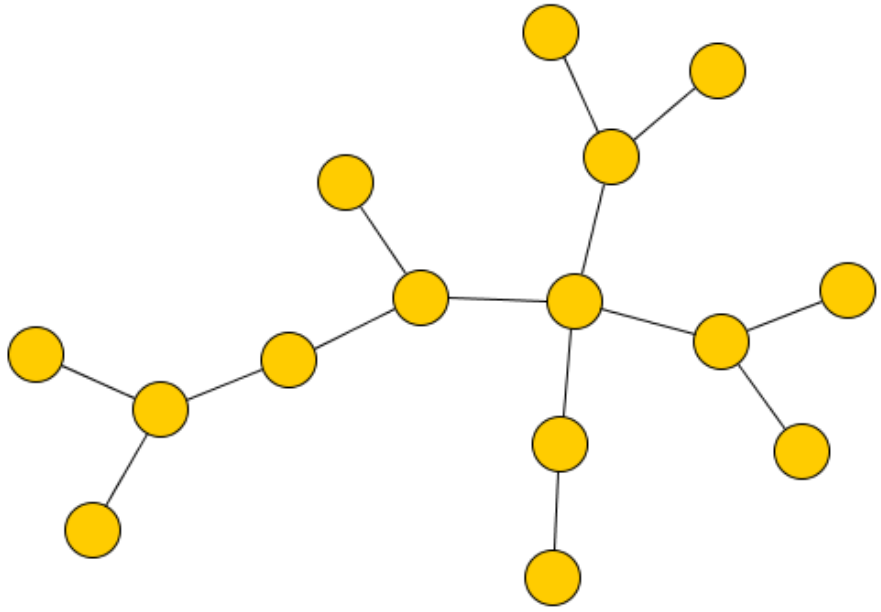


Abbildung 1.1c: Ein Baum

Diestel [2] beschreibt folgende Eigenschaften eines Baumes:

Ein Graph T ist ein Baum wenn folgendes gilt:

- Zwischen je zwei Ecken enthält T genau einen Weg
- T ist minimal zusammenhängend: Das heißt, sobald man irgendeine Kante entfernt, der Graph nicht mehr zusammenhängend ist.
- T ist ohne Zyklus: Das heißt, es entsteht ein Kreis sobald man für zwei nicht benachbarte Knoten eine neue Verbindung einfügt.

1.1.3 Spannbäume

Diestel [2] definiert einen Spannbaum wie folgt:

„*Ein Baum $T \subseteq G$ heißt Spannbaum von G, wenn er ganz G aufspannt, d.h. wenn $V(T) = V(G)$ ist. Ein Spannbaum ist […] also nichts anderes als ein Gerüst eines zusammenhängenden Graphen; insbesondere besitzt jeder zusammenhängende Graph einen Spannbaum.*“

Wenn man einem Knoten im Graphen die Eigenschaft einer Wurzel zuweist, definiert Diestel [2] diesen Graphen als Wurzelbaum.

1.1.4 Minimale Spannbäume

Weißt man jeder Kante des Graphen ein Gewicht (=Kantengewicht) zu, dann ist der minimale Spannbaum derjenige Spannbaum dessen Summe seiner Kantengewichte minimal ist. [21]

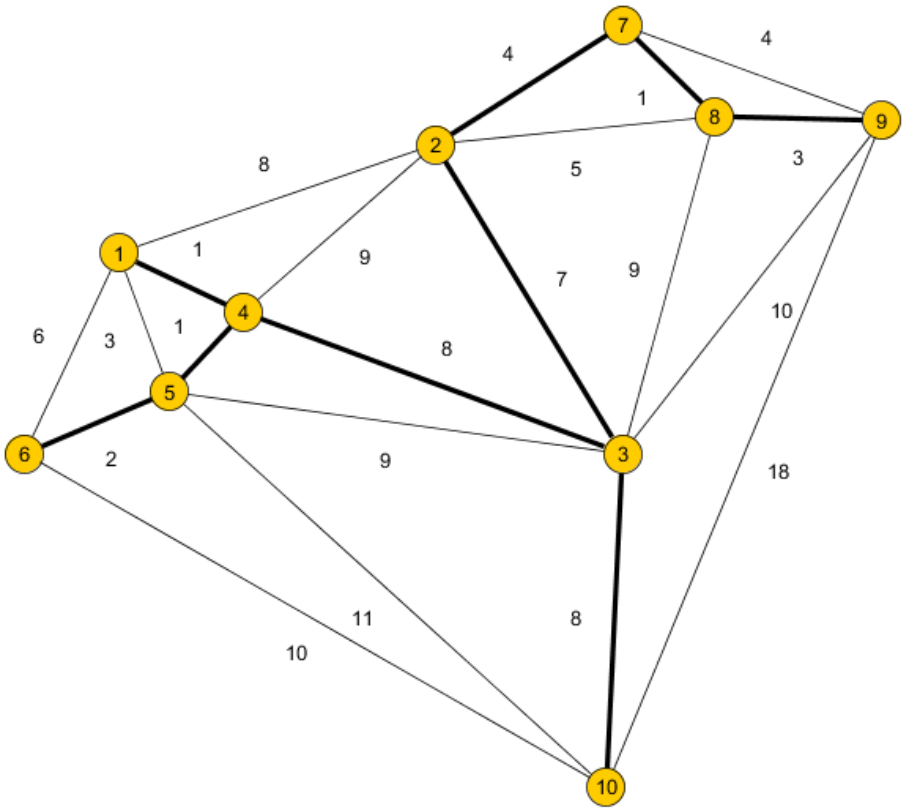


Abbildung 1.1d: Ein minimaler Spannbaum eines gewichteten, ungerichteten Graphen

Die Kanten im Graphen der Abbildung 1.1d haben jeweils eine Nummer. Sie entspricht der Gewichtung der jeweiligen Kante. Der minimale Spannbaum wurde mit visuell breiterer Kantenstärke dargestellt. Die Gesamtsumme des minimalen Spannbaumes dieses Graphen kann durch keine andere Kantenkombination verringert werden.

Gängige Methoden um den minimalen Spannbaum zu berechnen sind zum Beispiel *der Algorithmus von Kruskal* oder *der Algorithmus von Prim.*

1.1.4.1 Algorithmus von Kruskal

Der Algorithmus von Kruskal [14] beinhaltet folgende Grundidee. Es werden die Kanten in aufsteigender Folge der Gewichte durchlaufen und zur Lösung hinzugefügt, wenn sie mit keiner zuvor gewählten Kante einen Kreis bilden. Ansonsten wird sie markiert oder gelöscht und für den weiteren Verlauf nicht mehr berücksichtigt.

Es ist also ein zusammenhängender, kantengewichteter Graph gegeben. Im Falle das Graphen von Abbildung 1.1d würde der Algorithmus folgend beginnen.

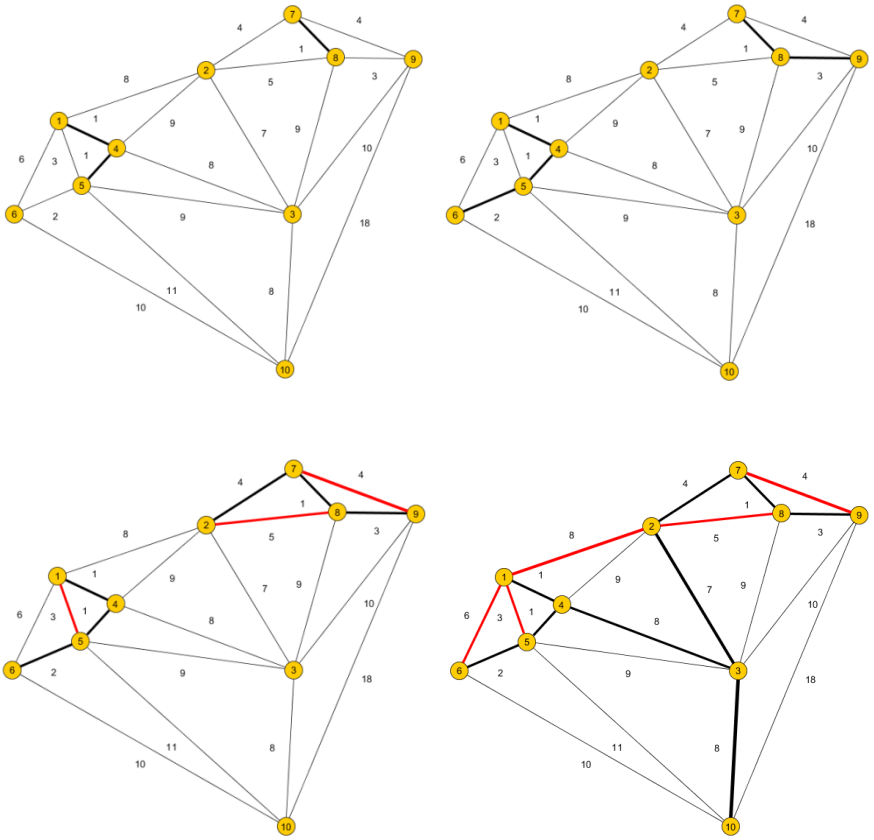


Abbildung 1.1e: Ablauf des Algorithmus von Kruskal angewendet auf den Graphen von Abbildung 1.1d.

Zuerst werden die Kanten 1-4, 4-5 und 7-8 in die Lösung hinzugefügt. Ebenso mit dem nächsthöheren gewichteten Kanten 5-6 und 8-9. Es gibt noch keine Konflikte. Mit 1-5 würde nun der erste Kreis entstehen, deswegen wird diese Kante verworfen. Ebenso mit 7-9. Danach wird 2-7 hinzugefügt. 2-8 und 1-6 ergäben wieder einen Kreis und werden ignoriert. 2-3 wird erfolgreich hinzugefügt. Jetzt gibt es 3 Kanten mit Kantengewicht 8. Je nachdem wer als erstes gewählt wird, wird nun Teil einer Lösung. Zum Beispiel könnte 3-4 aufgenommen werden, 1-2 würde dann einen Kreis ergeben und wird ignoriert und 3-10 wird wieder aufgenommen.

Die erkennbare Alternative wäre statt der Kante 3-4 die Kante 1-2 in den Spannbaum zu geben. Man sieht, dass es vorkommen kann, dass mehrere Möglichkeiten zu einem minimalen Spannbaum führen. Die Summe der Kantengewichte ist bei jeder möglichen Alternative gleich – sie ist immer minimal.

Der Algorithmus von Kruskal ist also ein Greedy-Algorithmus, weil er immer die aktuelle Situation bewertet und das erste, bestmögliche Teilergebnis in die Lösung einfließen lässt.

1.1.4.2 Algorithmus von Prim

Der Algorithmus von Prim, baut den minimalen Spannbaum nun immer zusammenhängend auf. Das bedeutete, dass während der Konstruktion nie ein nicht zusammenhängender Graph in der Ergebnismenge existiert.

Anwendung auf den Graphen aus Abbildung 1.1d:

Man wählt einen beliebigen Startknoten zum Beispiel den Knoten 10 und speichert ihn in die Ergebnismenge. Jetzt werden alle Kanten betrachtet und in eine Warteschlange gespeichert (siehe Abbildung 1.1f oben links). Die Kante mit dem kleinsten Gewicht wird nun in die Lösungsmenge eingetragen und aus der Warteschlange entfernt. In diesem Fall wäre das die Kante 3-10. In der Warteschlange befinden sich dann die Kanten 5-10, 6-10 und 9-10 und in der Ergebnismenge die Kante 3-10 und die Knoten 3 und 10. Durch Wählen der Kante 3-10 als Teillösung wird nun der Knoten 3 erreicht, deshalb werden nun alle Kanten dieses Knotens, die noch nicht in der Warteschlange sind, in die Warteschlange eingetragen, sofern sie zu einem Knoten verlaufen der **nicht** in der Ergebnisknotenmenge aufscheint (siehe Abbildung 1.1f oben mitte). Das wären dann die Kanten 2-3, 3-4, 3-5, 3-8 und 3-9. Es wird die Kante 2-3 gewählt, da sie nun das geringste Gewicht, aller gespeicherten Kanten aufweist. Der Knoten 2 wird ebenfalls in die Ergebnismenge eingetragen. Nun befinden sich die Knoten 2,3 und 10 in der Ergebnismenge und die Kanten 3-4, 3-5, 3-8, 3-9, 5-10, 6-10 und 9-10 in der Warteschlange (siehe Abbildung 1.1f oben rechts). Als nächstes werden die Kanten zu Knoten, die sich noch nicht in der Ergebnismenge befinden von Knoten 2 aus eingetragen. Das wären die Kanten 1-2, 2-4, 2-7 und 2-8 (siehe Abbildung 1.1f mitte links). 2-7 ist die Kante mit der kleinsten Gewichtung und sie wird wieder in die Lösungsmenge eingetragen, ihre Kanten in die Kantenmenge und der Knoten 7 in die Knotenmenge. 7-8 ist die nächste Kante die hinzugefügt wird. Nun fällt automatisch die Kante 2-8 und 3-8 aus der Kantenmenge, da es Kanten mit Start- und Endknoten aus der Knotenmenge sind (siehe Abbildung 1.1f mitte mitte). Im nächsten Schritt wird die Kante 8-9 in die Lösungsmenge und der Knoten 9 in die Knotenmenge aufgenommen. Es fallen automatisch die Kanten 3-9, 7-9 und 9-10 aus der Kantenmenge (siehe Abbildung 1.1f mitte rechts). Im nächsten Schritt kann, wie auch beim Beispiel mit dem Algorithmus von Kruskal, nun die Kante 3-4 oder 1-2 als Lösungskante aufgenommen werden. Wir nehmen 3-4 wieder in die Lösung auf und auch den neuen Knoten und die Kante 2-4 wird wieder automatisch von der Kantenliste gelöscht (siehe Abbildung 1.1f unten links). In den nächsten Schritten

werden die Kanten mit dem Kantengewicht 1 und zwar 1-4 und 1-5 hinzugefügt und die Kanten 1-2, 1-5, 3-5 und 5-10 fallen wieder per Definition weg (siehe Abbildung 1.1f unten mitte). Im letzen Schritt wird noch die Kante 5-6 in die Kantenliste eingefügt und der Knoten 6 kommt als letzter Knoten in die Knotenliste. Da sich nun alle Knoten in der Knotenliste befinden, ist auch die Kantenliste leer (siehe Abbildung 1.1f unten rechts).

Die Konstruktion eines minimalen Spannbaumes ist nun abgeschlossen. Durch das Löschen der Kanten, mit Start- und Endknoten der Knotenliste, wurde ein Zyklus verhindert.

Der Unterschied zum Algorithmus von Kruskal ist, dass der Algorithmus von Prim den Baum zusammenhängend aufbaut. Des Weiteren unterscheidet er sich im Verhindern von Zyklen.

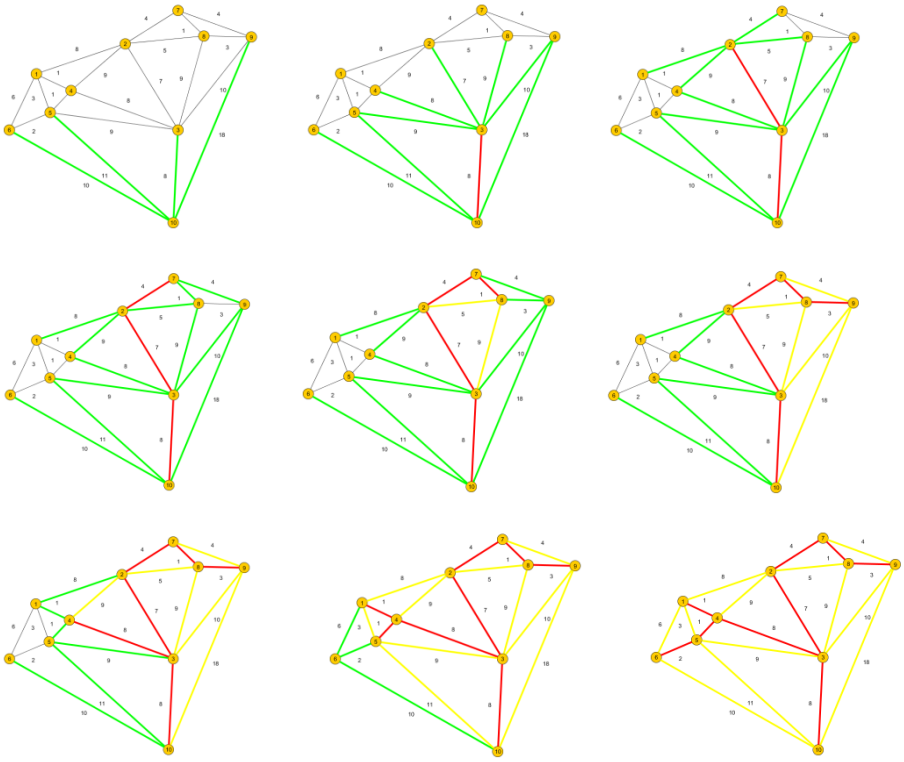


Abbildung 1.1f: Ablauf des Algorithmus von Prim zur Konstruktion eines minimalen Spannbaumes.

1.1.5 Bipartite Graphen

Definition laut Diestel [2]:

„Es sei $r \geq 2$ eine natürliche Zahl. Ein Graph $G = (V, E)$ heißt r-partit wenn eine Partition von V in r Teile existiert, so dass die Endecken einer

jeden Kante von G in verschiendenen Partitionsklassen liegen: Ecken aus der gleichen Klasse dürfen nicht benachbart sein. Ein 2-partiter Graph heißt auch ***bipartit*** *(oder paar).*“ [2]

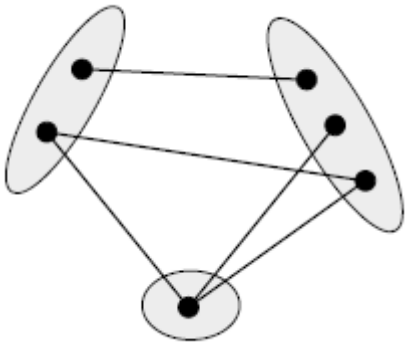


Abbildung 1.1g: Beispiel für einen 3-partiten Graphen. Aus R. Diestel, Graphentheorie, Berlin: Springer Verlag, 2010. S. 16. [2]

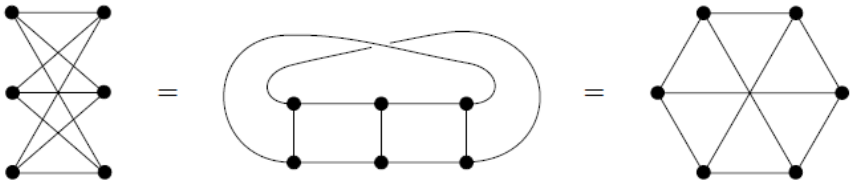


Abbildung 1.1h: Standard Beispiel für eines bipartiten Graphen $K_{3,3}$. (Aus R. Diestel, Graphentheorie, Berlin: Springer Verlag, 2010. S. 16)

1.1.6 Nicht zusammenhängende Graphen

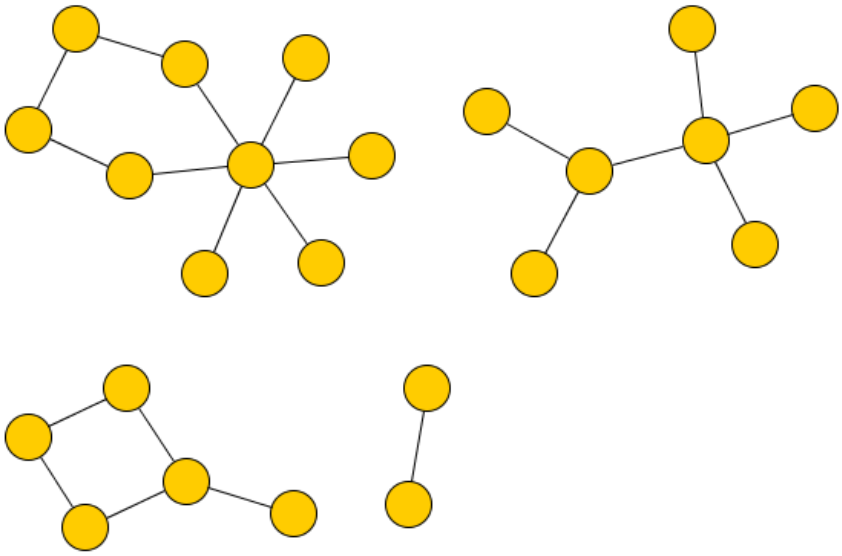


Abbildung 1.1i: Ein nicht zusammenhängender Graph

Ein großer Graph kann auch aus mehreren oder vielen kleinen Graphen bestehen, die nicht miteinander verbunden sind. Im Falle eines Graphen, der ein soziales Netzwerk abbildet, wäre ein derartiger Fall, wenn zwei Gruppen die jeweils untereinander befreundet sind mit keiner Person der anderen Gruppe befreundet und auch nicht über Beziehungen wie Freunde der Freunde zu einem Gruppenmitglied der anderen Gruppe eine Verbindung herstellen könnten. Bei Informationsnetzwerken, wie Wikipedia, wäre ein derartiger Fall, wenn man von einem Artikel nicht zu einem anderen Wikipedia Artikel über Hyperlinks navigieren könnte [22].

1.2 Graphen als Modelle von Netzwerken

Mit Hilfe der unter 1.1 erläuterten Begriffe, ist es uns nun möglich Modelle von Netzwerkstrukturen zu erstellen. Als Beispiel können wir den Graphen, wie in Kapitel 1.1 beschrieben, dafür nutzen um ein reales Beispiel abzubilden. Es gibt natürlich unzählige Möglichkeiten Graphen für Informationsmodellierung einzusetzen. Ich werde jedenfalls drei große Anwendungsgebiete der Modellierung mit Hilfe von Graphen erläutern.

1.2.1 Kommunikationsnetzwerke

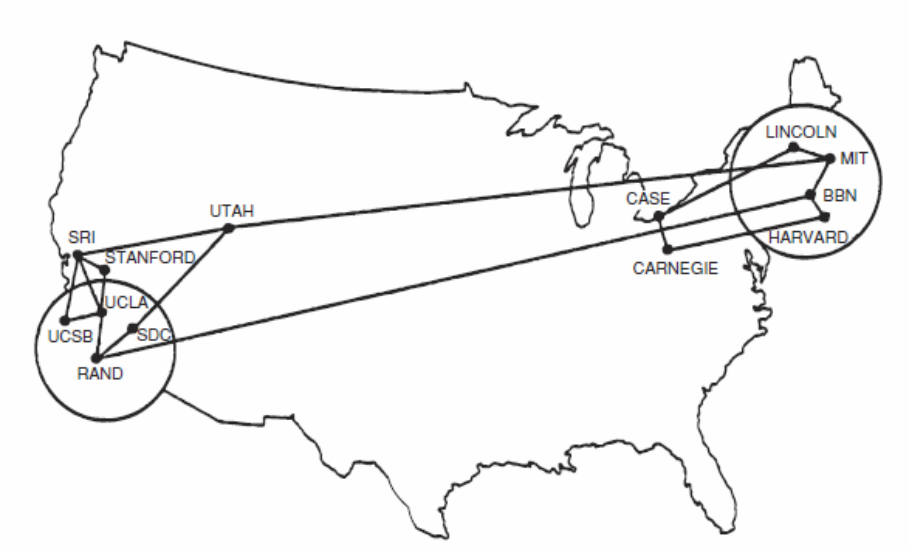


Abbildung 1.2a: ARPANET aus „*ARPANET Completion Report*“ [4]

Abbildung 1.2a zeigt die Netzwerkstruktur des Internets von 1970 [3] als es nur 13 Seiten hatte. Früher wurde es auch „*Advanced Research Projects Agency Network*“ (ARPANET) genannt. Die Knoten in der Abbildung 1.2a stellen die Großrechner dar und die Kanten des Graphen zeigen, zwischen welchen Großrechnern eine direkte

Kommunikationsleitung bestanden hat. Blendet man nun die geografische Karte aus, bleibt ein einfacher Graph mit 13 Knoten und 17 Kanten übrig, der im gleichen Stil wie in Abbildung 1.1a dargestellt werden kann.

Es geht hier nur um die Information, welcher Knoten mit welchem verbunden ist. Graphen werden häufig dazu eingesetzt um logische oder physikalische Verbindungen eines Netzwerkes darzustellen. Diese Information ist nicht von der Darstellung des Graphen abhängig. Von D. Easley [1] wird der Graph nun folgend alternativ dargestellt:

Laut D. Easley [1] sind die Abbildungen 1.2a und 1.2b ein Beispiel für ein **Kommunikationsnetzwerk** in welchem die Knoten die Großrechner oder andere Geräte zeigen und die Kanten stellen dar, mit welchen anderen Rechnern die einzelnen direkt kommunizieren können.

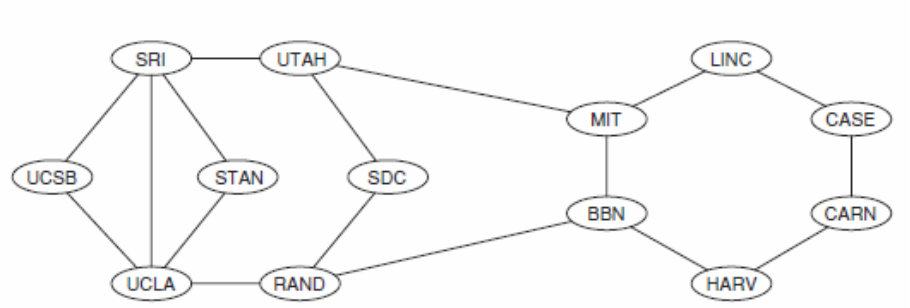


Abbildung 1.2b: Alternative Abbildung des Graphen in Abbildung 1.2a aus 1970 [1]

Zwei weitere Beispiele für Graphen als Modelle von Netzwerken will ich nicht vorenthalten.

1.2.2 Soziale Netzwerke

Graphen eignen sich auch ausgezeichnet dazu, um soziale Netzwerke und die Interaktionen der einzelnen Mitglieder zu zeigen [23]. Dazu möchte ich auf Abbildung 1.2c, einer Zeichnung aus dem „*Journal of Anthropological Research*“ verweisen.

Hierbei wurden die Mitglieder als Knoten und die Freundschaftsbeziehungen als Kanten dargestellt. Man kann schnell erkennen, welche Mitglieder am populärsten sind und welche nicht.

In dem Graphen aus 1.2c kann man ablesen, dass es anscheinend zwei Hauptpersonen gibt, die über ein hohes Bekanntheitsmaß verfügen. Es gibt auch Außenseiter, wie z.B. der Knoten 12. Dieser hat nur eine Verbindung zu einem der Hauptpersonen aller Mitglieder.

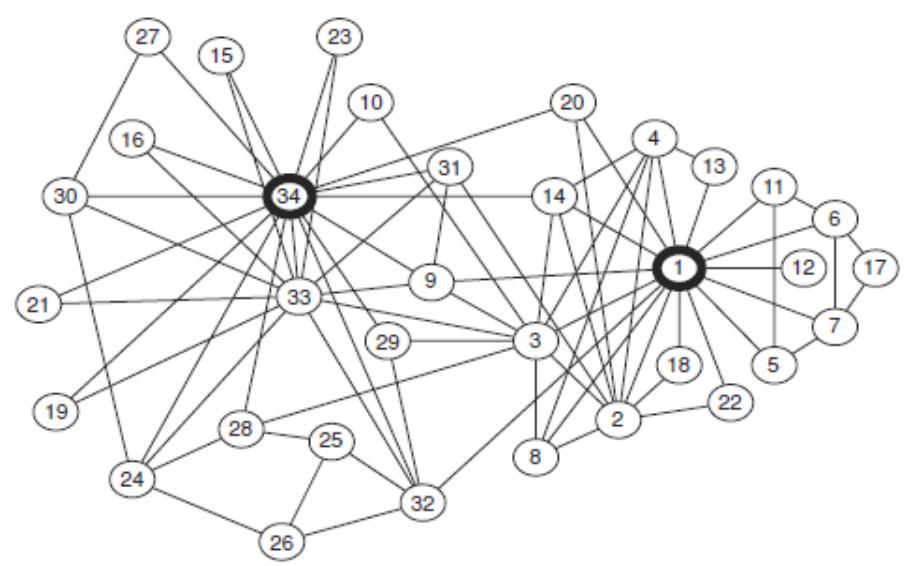


Abbildung 1.2c: Das soziale Netzwerk von Freundschaften eines Karate Clubs mit 34 Mitglieder aus „*An information flow model for conflict and fission in small groups.*“ [5]

1.2.3 Informationsnetzwerke

Nun kommen wir zu den Modellen, die ich später in den Experimenten verwenden werde – **Informationsnetzwerke**.

Wie auch in „*Networks, Crowds, and Markets: Reasoning About a Highly Connected World“* [1] erwähnt, sind die Knoten des Graphen eines Informationsnetzwerkes dessen Informationsressourcen, wie beispielsweise Webseiten, Dokumente oder in unserem Fall Wikipedia Artikel und die Kanten repräsentieren die logischen Verbindungen untereinander, wie zum Beispiel Hyperlinks, Zitate oder Querverweise.

Hyperlinks von Webseiten geben Aufschluss darüber wie die Webseite zusammengehörend ist, wie sie gruppiert ist oder ob verschiedene Bereiche existieren. Man kann auch über die Beliebtheit von Inhalten auf der entsprechenden Webseite Aussagen treffen. Inhalte, auf denen andere Seiten sehr oft verweisen, werden im Durchschnitt auch beliebtere Inhalte und gleichzeitig auch öfters gelesene Inhalte sein.

Die Informationen die man dadurch bekommt, benutzt auch die Suchmaschine Google. Google hat nun sehr viele Webseiten als Input und vergleicht die Verlinkungen untereinander. Sie bildet ein Rating für jede Webseite. Das Ranking eines Inhaltes bzw. einer Webseite wird im Groben über die Anzahl der auf den jeweiligen Inhalt zeigenden Webseiten gebildet, unter Berücksichtigung des bereits bestehenden Rankings. Somit haben öfter vorkommende Seiten ein höheres Ranking und werden im Suchindex weiter oben eingereiht. [28]

Um solche Daten generieren zu können, werden ebenfalls Abläufe angewendet, die viele Gemeinsamkeiten mit den Experimenten dieser Diplomarbeit haben.

Bildet man Informationsnetzwerke als Graphen ab, so wird einem schnell bewusst, dass Informationen meistens stark vernetzt sind. Die Vernetzung kann man direkt an der gewaltigen Anzahl von Kanten im Graphen ablesen. Bei etwas größeren Netzwerken, kommt man jedoch optisch schnell an die Grenzen.

Wie in Abbildung 1.2d abgebildet, ist das Ergebnis einer Visualisierung eines größeren Graphs äußerst unübersichtlich. Trotzdem kann man einige Eigenschaften ablesen, wie im konkreten Fall, dass der Graph ein nicht zusammenhängender Graph ist, da links ein Teilgraph erkennbar ist und dass es zwei große Cluster in diesem Graphen gibt, erkennbar durch die signifikante Anhäufung mehrerer Knoten an diesen Stellen.

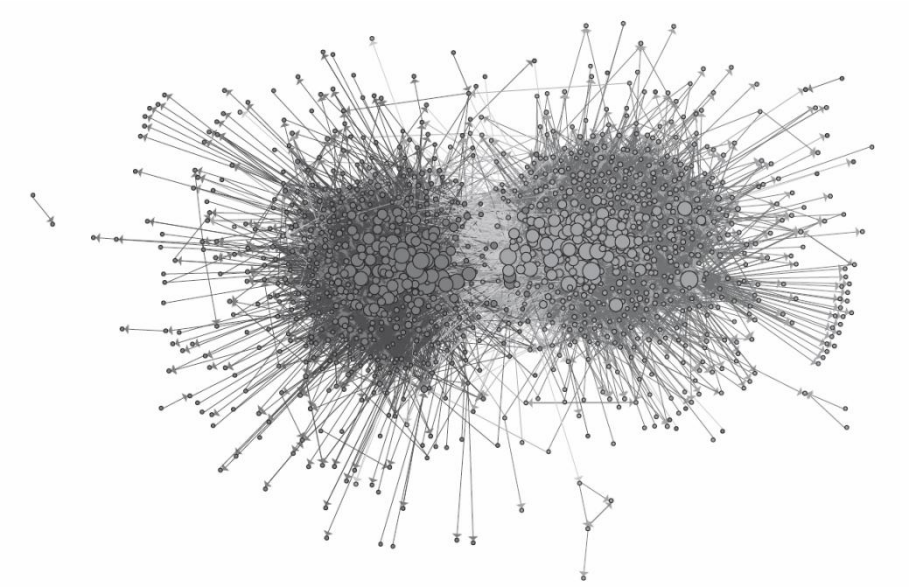


Abbildung 1.2d: Die Netzwerkstruktur eines politischen Blogs aus 2004 aus *Lada Adamic and Natalie Glance. The political blogosphere and the 2004 U.S. election: Divided they blog. In Proceedings of the 3rd International Workshop on Link Discovery, pages 36–43, 2005.* [6]

1.3 Pfade

Nun möchte ich einige fundamentalen Konzepte und Definitionen in der Welt von Graphen näherbringen. Es gibt etliche Anwendungsgebiete von Graphen. Graphen sind einfach zu verwenden und stellen simpel verschiedenste Sachverhalte auch optisch sehr gut dar. [24]

Die Anwendungsgebiete sind sehr weit gestreut. Um nur einige Bereiche zu nennen, findet man Graphen beispielweise bei der Berechnung von Netzwerken [26], bei der Abzählung von Isomeren in der Chemie, zur Berechnung von kürzesten Wegen bei diversen Zustellungsrouten [30], zur Berechnung des Informationsflusses in sozialen Netzwerken [27], zur Darstellung oder Nachbildung einer Verkehrsinfrastruktur, bei Verkehrswegen, bei der Sammlung von Schlagwörtern(*Folksonomien*) [19] und dessen Algorithmen [20] und beispielsweise zur Abbildung von Informationsnetzwerken [31].

Da es so viele Anwendungsgebiete gibt, bei denen mit Graphen hantiert wird, stellt sich die Frage, wie man mit den Graphen arbeiten kann. Dazu möchte ich den Begriff Pfad einführen.

Laut „*Networks, Crowds, and Markets*“ [1] ist ein Pfad eine simple Sequenz von Knoten mit der Eigenschaft, dass jedes aufeinanderfolgendes Knotenpaar in der Sequenz mit einer Kante verbunden ist. Es ist wichtig, dass man bei Pfaden nicht nur die aufeinanderfolgenden Knoten betrachtet, sondern auch die Kanten, die diese Knoten verbinden, untersucht.

In Abbildung 1.2c wäre beispielsweise die Sequenz 12, 1, 9, 34, 16 ein simpler Pfad. Auch die Sequenz STAN, SRI, UCSB, UCLA, RAND in Abbildung 1.2b erfüllt die Voraussetzungen eines Pfades.

Ein Pfad erfüllt auch seine Definition wenn er Knoten mehrmals enthält. Als Beispiel: 33, 34, 16, 33, 9, 3 in Abbildung 1.2c ist ein Pfad. Bei der Analyse von Informationsnetzwerken werden wir jedoch solchen Pfaden aus dem Weg gehen, da es unser Ziel ist kurze Wege zwischen zwei Knoten zu berechnen. Der Fall, dass ein Knoten mehrmals in einem solchen Pfad vorkommt, lässt direkt daraus schließen, dass der Pfad mit Sicherheit nicht der kürzeste war, bzw. der Pfad nicht effizient ist und gekürzt werden kann.

1.3.1 Kreise

Ein spezieller Fall eines nicht simplen Pfades, der eine Ringstruktur aufweist wird Kreis genannt, wenn der Pfad mindestens drei Knoten enthält, der Startknoten dem Endknoten entspricht und die anderen Knoten verschieden sind [1]. In Abbildung 1.2b wäre dies beispielsweise die Sequenz SRI, UCSB, UCLA, RAND, SDC, UTAH, SRI.

Im ARPANET von 1970 (siehe Abbildung 1.2a) gehört jede Kante zu einem Kreis. ARPANET wurde auf dieser Grundlage konstruiert um einen Ausfallschutz zu gewährleisten. Ist eine Verbindung zwischen zwei Knoten fehlerhaft oder nicht mehr verfügbar, ist durch diese Gegebenheit, die Konnektivität noch gegeben, da ein weiterer Weg zu den Knoten, dessen eine Kante (temporär) entfernt wurde, existiert.

Auch im täglichen Leben begegnen uns Kreise. Wenn man beispielsweise jemanden kennenlernt und durch Konversation herausfindet, dass der neue Kontakt mit anderen befreundeten Personen zu tun hat, hat sich gerade eben ein neuer Kreis geschlossen. [18]

1.4 Konnektivität

Ein weiteres sehr wichtiges Kriterium eines Graphen ist seine Konnektivität. Kann in einem gegebenen Graphen jeder Knoten einen anderen Knoten über einen Pfad erreichen, so ist die Konnektivität gegeben, man sagt dann, dass der Graph „*connected*“ ist. Gibt es in einem Graphen jedoch Knoten, zwischen denen kein Pfad existiert, so hat das direkte Auswirkungen auf die Erfolgsrate beim Navigieren mit Hierarchien in dem jeweiligen Graphen. Die Erfolgsrate kann dann beim Navigieren zwischen Random Pairs nicht mehr 100% erreichen, wenn man diese Knoten im Ausgangsgraphen lässt.

Es gibt keinen Grund, zu erwarten dass ein Graph „*connected*“ ist. Gerade bei Abbildungen von sozialen Netzwerken oder von Informationsnetzwerken ist die Chance sehr groß, dass der Ergebnisgraph nicht zusammenhängend ist. Bei einem sozialen Netzwerk kann man sich das relativ einfach vorstellen. Wenn man annimmt, dass der Graph einer Abbildung eines sozialen Netzwerkes „*connected*“ ist, wäre das das Selbe als würde man behaupten, dass jede Person des Netzwerkes über dessen Freunde zu jeder anderen Person eine Verbindung besitzt. Das ist zwar möglich, aber wahrscheinlicher ist es, dass kleine Gruppen existieren, die einzelne kleine Teilgraphen bilden.

Gleiches ist bei Informationsnetzwerken zu erwarten. Deswegen muss man, um die Ergebnisse des Experimentes nicht negativ zu verfälschen, darauf achten, ob bei der Navigation der zufälligen oder willkürlich ausgewählten Start- und Zielknoten im Informationsnetzwerkgraphen überhaupt ein Weg existiert.

Besteht ein Graph aus mehreren Teilgraphen, so werden diese Teilgraphen auch häufig Komponenten genannt. Die Komponenten beinhalten Knoten, die miteinander verbunden sind. Man nennt sie auch „*connected components*“ wobei man meistens nur von „*components*“ spricht. Wie auch im Buch „*Networks, Crowds, and Markets*“ [1] definiert,

ist eine Komponente ein Subset von Knoten für die gilt, dass zwischen allen Knoten der Komponente ein Pfad existiert und dass das Subset nicht ein Teil eines größeren Sets, mit der Eigenschaft, dass jeder Knoten jeden anderen Knoten erreicht, ist.

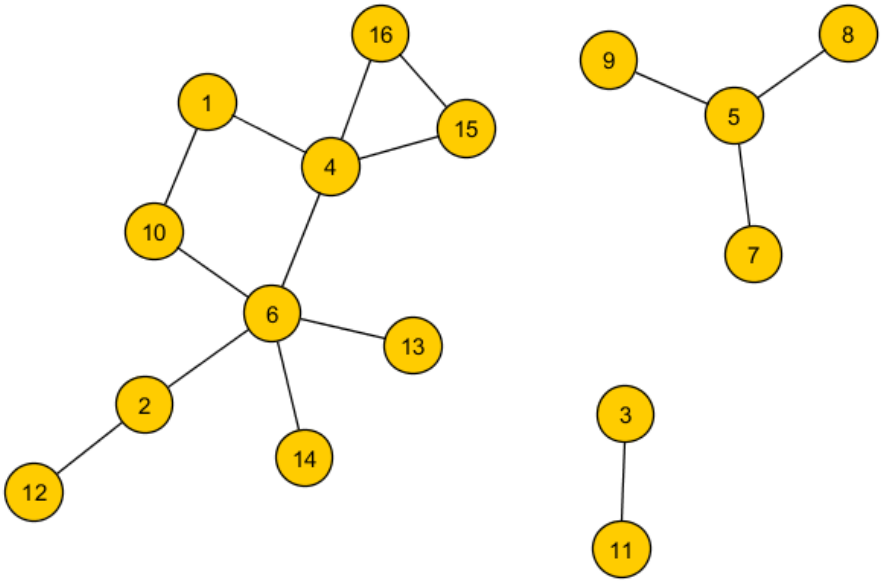


Abbildung 1.4a: Ein Graph mit drei verbundenen Komponenten

1.5 Suche in Graphen

Wie sucht man nun einen Pfad in einem Graphen?

Für die effiziente Suche in Graphen benötigt man mehr Informationen. Unter Anderem sind zu diesem Zweck Informationen über Distanzen zwischen Knoten bzw. die Pfadlängen interessant.

Die Distanz zwischen zwei Knoten A und B entspricht, laut D. Easley und J. Kleinberg [1], der Länge des kürzesten Pfades zwischen A und B.

1.5.1 Distanzen berechnen

Um die kürzesten Distanzen zwischen 2 Knoten in einem kleinen Graphen zu finden, genügt oft ein Blick. Bei großen und komplexeren Graphen ist es nicht mehr so einfach. Man benötigt eine Methode.

Mit Hilfe der Breitensuche kann man hervorragend die Distanzen in einem Graphen berechnen, man muss den *„Breadth First Search“* Algorithmus nur minimal modifizieren.

Die Funktionsweise für die Distanzberechnung zwischen zwei willkürlich festgelegten Knoten mit Hilfe der Breitensuche funktioniert folgend:

Zuerst wird ein Startknoten definiert. Dieser Startknoten wird in eine Warteschlange geschrieben und ist der erste Knoten der Warteschlange mit dem Wert 0. Dieser Wert entspricht gleichzeitig der Ebene, in der sich der Knoten befindet und stellt die Distanz zum Startknoten dar. Nun wird von Beginn der Warteschlange ein Knoten entnommen und markiert. Falls das gesucht Element gefunden wurde, kann abgebrochen und der gespeicherte Wert als Distanz zurückgegeben werden. Wenn nicht, werden alle nicht markierten Nachfolger dieses Knotens, die sich noch nicht in der Warteschlange befinden, ans Ende der Warteschlange gehängt und mit dem Wert des Vaterknotens + 1 markiert. Ist die Warteschlange leer, existiert kein Pfad zu dem Endknoten und die Distanz wäre unendlich. Ansonsten wird wieder ein Knoten der Warteschlange entnommen und die Prozedur wie vorher fortgesetzt, bis der Knoten entdeckt wurde, dann wird der Wert des Knotens retourniert.

Mehr zum Breitensuche werde ich später im Kapitel 3 „Produktion von Hierarchien“ erwähnen, da sich das Hauptexperiment mit der Erstellung von Hierarchien mittels modifizierter Breitensuche befasst.

Zur Veranschaulichung einer Breitensuche betrachte man Abbildung 1.5a. Es entsteht ein hierarchischer Baum, der solange wächst bis der Zielknoten gefunden oder alle erreichbaren Knoten besucht wurden.

Je nach Anwendungsfall, ob man nun einen bestimmten Endknoten sucht, oder aber alle kürzesten Distanzen eines Startknotens berechnen möchte, ändert sich das Abbruchkriterium.

Wie man oben bei der Laufzeit sieht, sind bei großen Graphen maximal $|V| + |E|$ Schritte notwendig um von einem gewählten Startknoten alle Distanzen auszurechnen.

Wenn man nun für alle Knoten des Graphen alle kürzesten Distanzen berechnen möchte, käme der Faktor $|E|$ als Multiplikator bei der Laufzeit hinzu.

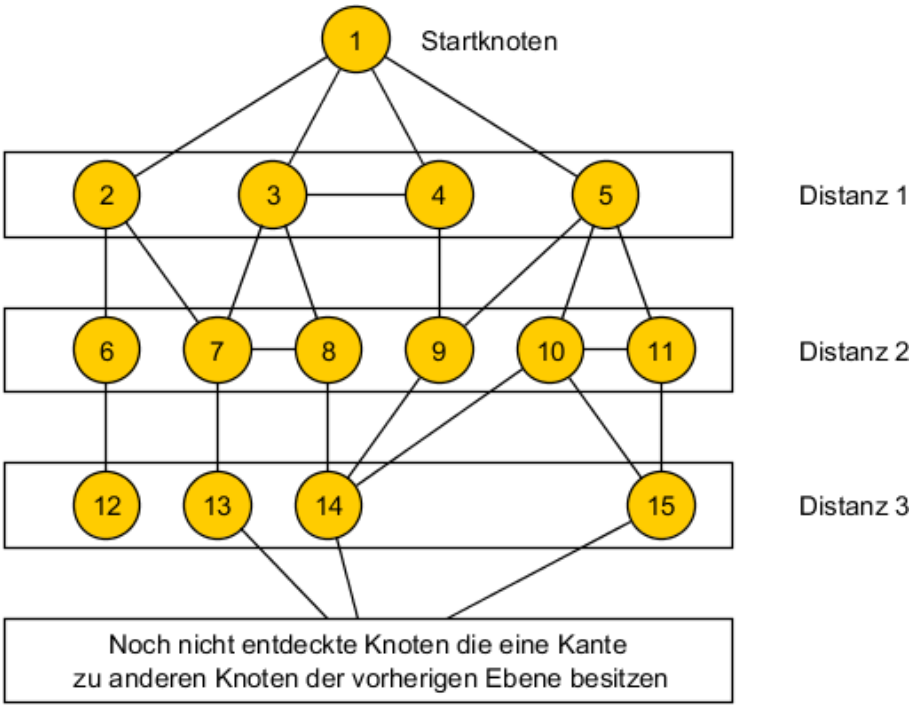


Abbildung 1.5a: Die Breitensuche entdeckt die Distanzen zu Knoten einer Ebene – jede Ebene besteht aus Knoten die mindestens eine Kante zu der vorherigen Ebene besitzen.

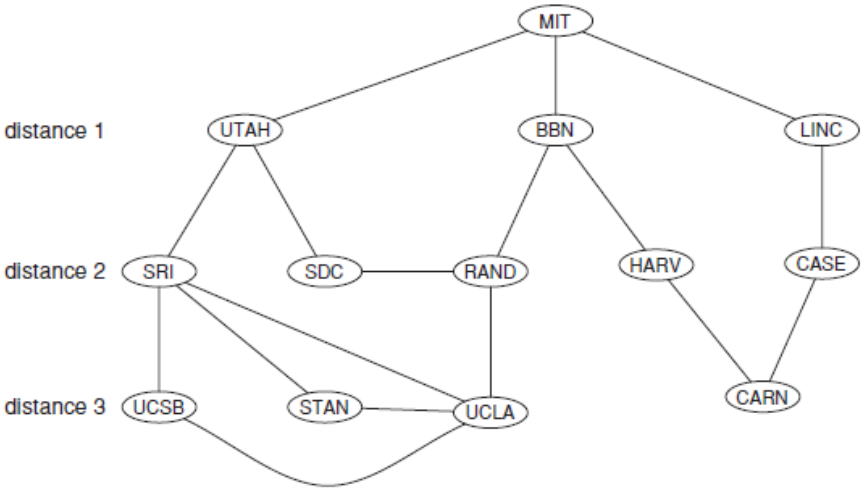


Abbildung 1.5b aus [1]: Die entstandenen Ebenen einer Breitensuche aus dem ARPANET (Abbildung 1.2b) mit Wurzelknoten MIT.

Interessant ist, dass erst durch die veränderte Darstellung des Graphen mit den eingeführten Distanzebenen, bewusst wird, wie gering der kürzest mögliche Pfad von einem Startknoten zu einem frei wählbaren Endknoten ist.

Da man in der Praxis jedoch nicht alle kürzesten Pfade aller Knoten eines großen Graphen als zusätzliche Information, für beispielweise der Navigation, in einem Graphen berechnen und mitgeben kann, weil der Speicherbedarf und auch die Laufzeit zu hoch sind, versucht man das Hintergrundwissen mit Hilfe von Hierarchien zu maximieren. Die Frage welcher Anfangsknoten als Wurzelknoten verwendet werden soll, um allgemein gute Ergebnisse in der Navigation zu erhalten stellt sich bis jetzt noch.

2 Umlegung auf Informationsnetzwerke

2.1 SNAP

SNAP, die „*Stanford Network Analysis Platform*“ [32], ist ein performantes System zur Analyse und Manipulation von großen Netzwerken. Die Bibliothek wurde seit 2004 entwickelt und wächst permanent durch Forschungsprojekte im Bereich der Analyse von großen sozialen als auch Informationsnetzwerken weiter. Beim größten Datensatz, der mit SNAP untersucht wurde, handelt es sich um das Microsoft Instant Messenger Netzwerk mit 240 Millionen Knoten und 1,3 Milliarden Kanten. Die *Stanford Network Analysis Platform* gewährleistet die Skalierbarkeit von Netzwerken, von Graphen mit Millionen Knoten und Milliarden Kanten.

Der Kern der SNAP Library ist in C++ geschrieben und ist entsprechend auf Leistung und kompakte Repräsentation von Graphen optimiert. Die Library eignet sich sehr gut für die Manipulation von großen Graphen und der Berechnung von strukturellen Eigenschaften. SNAP unterstützt auch das Zuweisen von Attributen bei Knoten und Kanten. Der verwendete Graph kann mit SNAP auch während der Berechnung dynamisch verändert werden und es unterstützt gerichtete als auch ungerichteten Graphen.

Im November 2009 wurde die SNAP Bibliothek erstmalig online, unter BSD Lizenz (*Berkeley Software Distribution*), zur Verfügung gestellt. Software unter BSD Lizenz kann frei verwendet, kopiert, verändert und verbreitet werden.

2.1.1 Graphen in SNAP:

Die folgenden Graphen sind in SNAP umgesetzt:

TUNGraph: Ungerichteter Graph (eine Kante zwischen einem ungeordneten Paar von Knoten)

TNGraph: Gerichteter Graph (eine gerichtete Kante zwischen einem geordneten Paar von Knoten)

TNEGraph: directed multi-graph (beliebige Anzahl von gerichteten Kanten zwischen einem Paar von Knoten)

2.1.2 Netzwerktypen in SNAP:

TNodeNet<TNodeData>: ähnlich dem TNGraph jedoch mit einem TNodeData Objekt für jeden Knoten

TNodeEDatNet<TNodeData, TEdgeData>: ähnlich dem TNGraph jedoch mit TNodeData an jedem Knoten und TEdgeData an jeder Kante

TNodeEdgeNet<TNodeData, TEdgeData>: ähnliche dem TNEGraph jedoch mit TNodeData an jedem Knoten und TEdgeData an jeder Kante

TBigNet<TNodeData>: ist eine speichereffiziente Implementierung von TNodeNet welche eine Speicherfragmentierung verhindert. Dadurch kann sie Milliarden Kanten, in Abhängigkeit wieviel Speicher zur Verfügung steht, verarbeiten.

2.1.3 Nützliche Funktionen in Snap

Eine kurze Einleitung über die vorhandenen Grundfunktionen in SNAP.

Die folgenden Funktionen geben Iteratoren zurück die auf bestimmte Knoten oder Kanten zeigen.

BegNI(): Retourniert einen Iterator der auf den ersten Knoten zeigt.

EndNI(): Retourniert einen Iterator der auf das Ende des letzen Knotens zeigt.

GetNI(u): Retourniert einen Iterator der auf den Knoten mit der ID „u“ zeigt.

BegEI(): Retourniert einen Iterator der auf die erste Kante zeigt.

EndEI(): Retourniert einen Iterator der auf das Ende der letzen Kante zeigt.

GetEI(u,v): Retourniert einen Iterator der auf die Kante zwischen Knoten u und Knoten v zeigt.

GetEI(e): Retourniert einen Iterator der auf die Kante mit der ID „e“ zeigt. (nur bei Multigraphen)

Funktionen die Knoten-ID’s oder Eigenschaften von Knoten zurückgeben.

GetId(): Gibt die aktuelle Knoten-ID zurück.

GetOutDeg(): Gibt die „*out-degree´*s“ des Knoten zurück.

GetInDeg(): Gibt die „*in-degree´*s“ des Knoten zurück.

GetOutNId(e): Gibt den Knoten am Ende der ausgehenden Kante „e“ zurück.

GetInNId(e): Gibt den Knoten der eingehenden Kante „e“ zurück.

IsOutNId(int NId): Gibt an, ob vom aktuellen Knoten eine ausgehende Kante zu einem Zielknoten existiert.

IsInNId(n): Gibt an, ob ein Zielknoten eine ausgehende Kante zum aktuellen Knoten besitzt.

IsNbhNId(n): Gibt an, ob ein Knoten n ein Nachbar ist.

Funktionen für weitere Daten von Knoten und Kanten.

GetDat(): Gibt den Datentyp TNodeData des Knotens zurück.

GetOutNDat(e): Gibt Daten, die mit dem Endknoten des Knotens der e-ten ausgehenden Kante assoziiert sind, zurück.

GetInNDat(e): Gibt Daten, die mit dem Endknoten des Knotens der e-ten eingehenden Kante assoziiert sind, zurück.

GetOutEDat(e): Gibt Daten, die mit der e-ten ausgehenden Kante assoziiert sind, zurück.

GetInEDat(e): Gibt Daten, die mit der e-ten eingehenden Kante assoziiert sind, zurück.

2.2 Navigationssimulator

Für die Navigation in einem Graphen benutzen wir die sogenannte „**Greedy Navigation**“ (siehe 2.2.2). Als Hintergrundwissen wird dem Navigator eine erzeugte Hierarchie zur Verfügung gestellt.

Die Arbeit von M. Eder [7] befasste sich mit der Erstellung eines Navigationsframeworks, mit denen die Auswertungen meiner Arbeit folgen werden.

2.2.1 MUN – Framework

Das von M. Eder [7] entwickelte Framework MUN (*Modeling User Navigation*), ist eine Modellierung von Benutzer-Navigation, mit der Basis von SNAP.

Das MUN - Framework besteht aus zwei großen Teilen. Einerseits ist es der Framework-Kern und andererseits der Navigationsteil.

2.2.1.1 Framework-Kern

Im Ordner „core“ befindet sich der Kern des Frameworks, welcher folgende Klassen beinhaltet:

- *Environment*:
  Behandelt alle Parameter die man setzen kann bzw. muss um die Umgebung des Frameworks zu konfigurieren

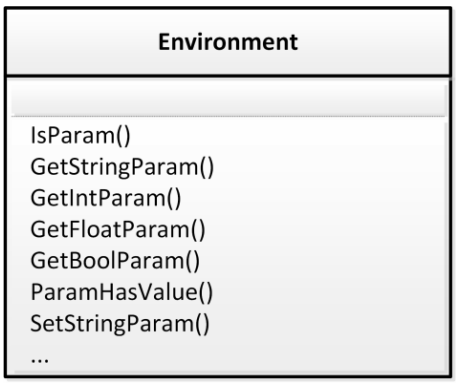


Abbildung 2.2a: Die Klasse Environment aus Eder [7]

- *Graph*:
  Diese abstrakte Klasse beinhaltet allen notwendigen Funktionen zur Manipulation eines Graphen, wie zum Beispiel das Abfragen der Anzahl von Knoten und Kanten, das Abfragen ob die Knoten A und B benachbart sind, das Einfügen von Kanten oder weiteren Knoten im Graphen, als auch das Löschen und auch das Laden und Speichern des Graphen.
- *GraphD*
  GraphD ist gerichteter Graph, implementiert das Interface der abstrakten Klasse *Graph* und ist ein Adapter für die gerichteten Graphenmodell-Klassen TNGraph und PNGraph in SNAP Library.
- *GraphU*
  GraphU stellt einen ungerichteten Graphen dar, implementiert auch das Interface der abstrakten Klasse *Graph* und ist ein Adapter für die ungerichteten SNAP Graphenmodell-Klassen PUNGraph und TUNGraph.

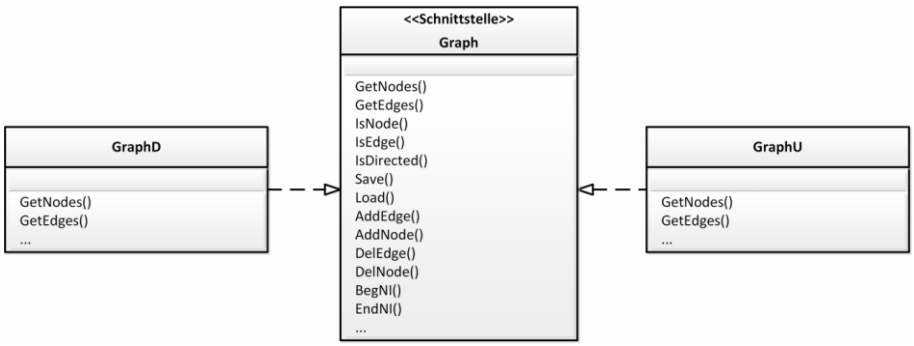


Abbildung 2.2b: Die Klassen *GraphD* und *GraphU*.
Abbildung aus Eder [7]

- *NodeI*
  NodeI ist ein Interface für Knoten-Iteratoren mit Methoden die von Knoten-Iteratoren unterstützt werden sollen. Mit dieser Klasse ist es sehr einfach über alle vorhandenen Knoten im Graphen zu iterieren und in Echtzeit die Eigenschaften des jeweiligen Knotens, wie zum Beispiel Ausgangsknoten, Eingangsknoten, Knotengrad und Nachbarknoten, abzufragen.
- *NodeID / NodeIU*
  NodeID und NodeIU sind die Knoten-Iteratoren für gerichtete und ungerichtete Graphen.

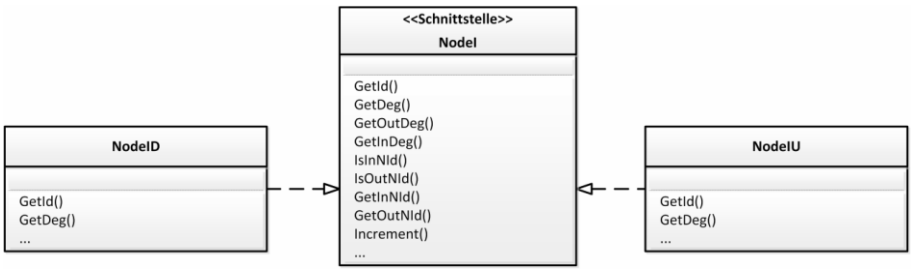


Abbildung 2.2c: Die Klassen *NodeID* ind *NodeIU* aus *Eder [7]*

- *Pairs*
  Diese Klasse beschäftigt sich mit den Suchpaaren welche aus Start- und Zielknoten bestehen und zwischen denen Navigiert werden soll. Die Generierung von Paaren ist in der Klasse auch umgesetzt.
- *Path*
  Die *Path* Klasse im Modeling User Navigation Framework behandelt das Thema der Navigationspfade und schließt auch alle Daten die entstehen in sich ein.
- *PathCollection*
  Die PathCollection ist eine Sammlung von Path Objekten und bietet als Funktionalität das laden und speichern von mehreren Pfaden.

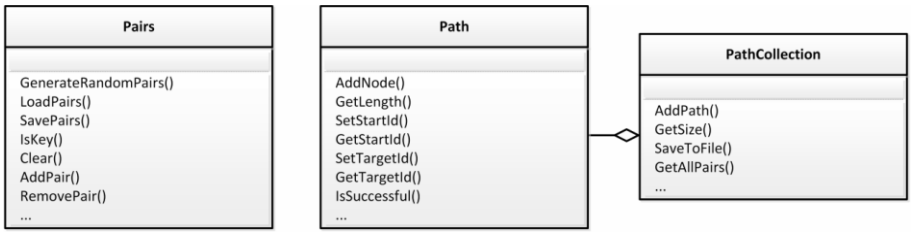


Abbildung 2.2d: Die Klassen Pairs, Path und PathCollection.
Abbildung aus *Eder [7]*

- *ShortestDistance*
  Errechnet und speichert die kürzesten Distanzen von beliebigen Path Objekten.

2.2.1.2 Framework – Navigation

Der zweite Hauptteil, des von Eder [7] entwickelten Frameworks ist der Navigationsteil. Er befindet sich im Ordner „navigation“. In diesem Ordner befindet sich die Navigator Klasse und die NavigationStrategyFactory. Weiters gibt es noch vier Unterordner „attrition“, „linkfilter“, „nodeselector“ und „strategy“.

Der Navigator benutzt das Environment Objekt, um die Navigationssimulation initialisieren zu können. Das Environment Objekt enthält wiederrum die Konfiguration, mit der der Navigator eine Instanz einer Navigations-Strategie-Implementierung erzeugen kann. Sobald der Navigationssimulator gestartet wird, kann er mit dem Generieren von Pfaden beginnen, in dem die entsprechende Methode der jeweiligen Navigationsstrategie aufgerufen wird. Zurückgeliefert wird ein PathCollection Objekt, welches die generierten Pfade beinhaltet.

NavigationsStrategie-Objekte werden mit Hilfe der NavigationsStrategyFactory erzeugt, die den nächsten wichtigen Teil der Navigation ausmacht. Die Factory erstellt gemäß der Konfiguration das passende NavigationsStrategie-Objekt. Die Implementierung lässt es zu, weitere Navigationsstrategien ohne größeren Aufwand zu erstellen und zu benutzen.

Im Verzeichnis „attrition“ hat M. Eder [7] folgende Klassen entwickelt.

- *IAttrition*
  Ein Interface, um weitere Abbruchkriterien einzuführen.
- *IAttritionSupport*
  Ein Interface, das von der jeweiligen Strategie implementiert werden muss, wenn sie zusätzliche Abbruchkriterien einbaue möchte, bzw. um IAttrition verwenden zu können.

Im Ordner „linkfilter“ wurden folgende Klassen entwickelt:

- *ILinkFilter*
  Diese Klasse erlaubt es Knoten zu filtern über die ein Schritt erfolgen kann.
- *ILinkFilterSupport*
  Dieses Interface muss von der jeweiligen Strategie implementiert werden um den ILinkFilter verwenden zu können.

Der „nodeselector“ Ordner beinhaltet folgende von Eder entwickelte Klassen:

- INodeSelector
  Die Zuständigkeit dieser Node-Selector Klasse ist es, jeweils den zu der ausgewählten Strategie passenden nächsten Knoten, aus der Liste der möglichen nächsten Knoten, auszuwählen.
- NSBaseNodeSelector
  Die abstrakte Klasse „NSBaseNodeSelector“ deckt zur Vereinfachung die benötigte Grundfunktionalität für Selectoren ab.
- NSRandom
  Ist ein Node-Selector, der zufällig einen nächsten Knoten, aus der Liste von möglichen Knoten, auswählt.

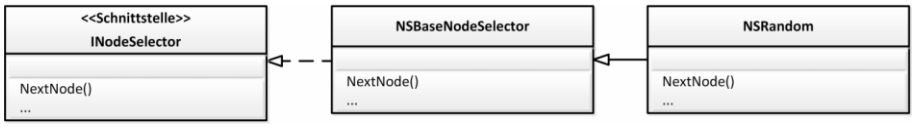


Abbildung 2.2e: Der INodeSelector aus *Eder [7].*

- INavigationStrategy
  Definiert eine Navigationsstrategie, die als austauschbarere Teil des Navigators anzusehen ist.
- SBaseStrategy
  Implementiert partiell das NavigationsStrategie Interface um Codeteile für speziellere Strategie-Implementierungen bereitzustellen.
- SCombiNavigator
  Diese Strategie kann mit mehreren Node-Selectoren umgehen. Man gibt in der Konfiguration das Wahrscheinlichkeitsmodell an, anhand dessen die aktuell zu verwendende Strategie ausgewählt wird.
- SRandomNavigator
  Diese einfache Strategie ist auf zufällige Navigation ausgelegt und benutzt den Node-Selector „*NSRandom*“.

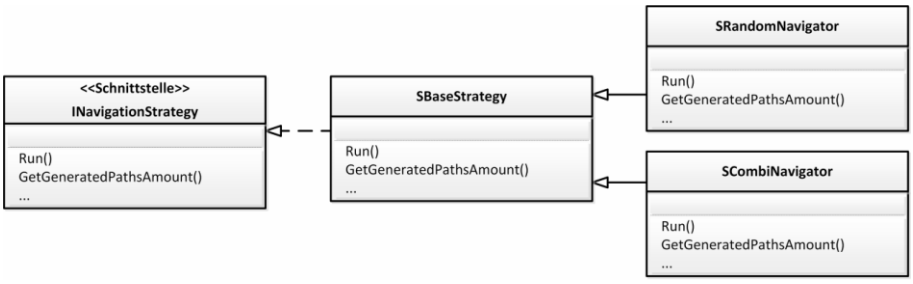


Abbildung 2.2f: Die INavigationsStrategy aus *Eder [7]*

Das *Modeling User Navigation Framework* bietet also alle grundlegenden Möglichkeiten um es so zu konfigurieren, wie es in dieser Masterarbeit notwendig ist. Das *Modeling User Navigation Framework* ist darauf ausgelegt um menschliche Navigationspfade [34] nach zu modellieren. In dieser Arbeit, geht es jedoch um die kürzest möglichen Pfade und Distanzen und nicht um menschliche Navigation. Mit der richtigen Konfiguration kann jedoch genau diese Forderung eingestellt werden. Es wird eine „Greedy Navigation“ benötigt um die erstellten Hierarchien auf die gewünschten Eigenschaften abprüfen zu können. Auf Greedy Navigation komme ich im Punkt 2.2.2 zurück.

2.2.1.3 Die Konfiguration des Navigators

Das Konfigurationsfile des Navigators besteht aus folgenden Parametern:

- *global.amount*

  Über diesen Parameter kann man die maximale Anzahl der Suchpaare bestimmen. Es sollen alle Suchpaare, in diesem Fall, die 100.000 zufälligen Paare verwendet werden, daher ist der Wert bei der Evaluierung der Hierarchien *global.amount:0*.

- *global.output*

  Der Ausgabeordner wird mit diesem Parameter bestimmt. Sinnvollerweise pro Hierarchie in einem eigenen Ordner.

- *graph.file*

  Gibt den Pfad zu dem jeweiligen Graphen an. Der Navigator kann auch mit binären Graphen umgehen und erkennt automatisch ob es sich um einen binären Graphen handelt.

- *graph.type*

  Diese Einstellung beschreibt den Typen des Graphen. Der Navigator muss wissen, ob der Graph gerichtet oder ungerichtet ist. Im Fall von Wikipedia handelt es sich um einen gerichteten (directed) Graphen. Daher wird *graph.type:d* verwendet.

- *hier.file*

  Gibt den Pfad zu der jeweiligen Hierarchie, die als Hintergrundwissen angewendet werden soll, an.

- *hier.type*

  Diese Einstellung sagt dem Navigator, ob die Hierarchie als gerichteter oder ungerichteter Baum gesehen werden soll. Es handelt sich wieder um eine gerichtete Hierarchie: *hier.type:d*.

- *pairs.inputfile*

  Gibt den Pfad zu den, in unserem Fall zufälligen, Start- und Endknoten Paaren an.

- *shortestd.inputfile*

  Gibt den Pfad zu der Datei an, die die kürzesten Wege der mit *pairs.inputfile* angegebenen Paare an.

- *navigation.strategy*

  Gibt die zu verwendende Strategie an. Bei der Evaluierung ist nur die Greedy Navigation sinnvoll. *navigation.strategy:greedy*.

- *navigation.maxhops*

  Gibt die maximale Distanz, die der Navigator im schlechtesten Fall navigieren soll, an. Wird der Wert überschreitet bricht der Navigator ab und wertet den Versuch als nicht erfolgreich. Bei der Evaluierung wurde hier *navigation.maxhops:100* gewählt.

- *navigation.revisits*

  Gibt an wie oft ein Knoten wiederholt besucht werden darf. Im Falle der Greedy Navigation ist ein wiederholter Besuch nicht sinnvoll und wird daher ausgeschlossen.

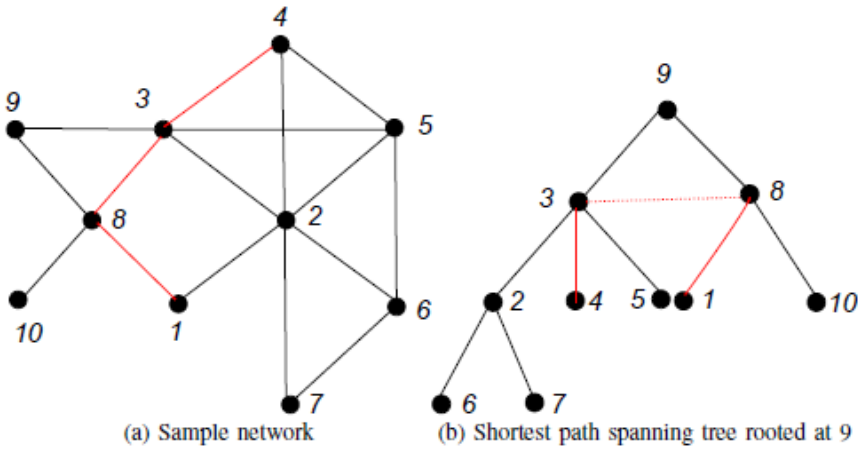
2.2.2 Greedy Navigation

Wie auch im Paper „*Navigational Evaluation of Breadth First Search Spanning Trees*“ [8] erwähnt wird, handelt sich hierbei um folgende Vorgehensweise.

Ein Greedy-Navigationspfad ist jener Pfad, mit der Eigenschaft, dass jeder seiner gewählten Knoten, in dem Zeitpunkt der Auswahl, die kürzest mögliche Distanz zu dem Zielknoten besitzt. Die Knoten des Greedy-Navigationspfades können also nicht durch andere Knoten ersetzt werden, sodass die Pfadlänge kürzer wird, da bereits die besten Knoten gewählt wurden.

Das heißt also, dass der Navigator alle möglichen Kandidaten betrachtet und sich für den Kandidaten mit der minimalen Distanz entscheidet. Er navigiert nun zu diesem Kandidaten und betrachtet wiederum alle möglichen weiteren Kandidaten. Das wird solange wiederholt, bis sich der Navigator im Ziel befindet.

Der Greedy-Navigationspfad kann Abkürzungen in einem Pfad nehmen und andererseits kann es sein, dass ein Navigationspfad entsteht der nicht der kürzest mögliche Weg ist. Man betrachte folgende Abbildung.



(a) Sample network (b) Shortest path spanning tree rooted at 9

Abbildung 2.2g: Greedy Navigation aus "*Navigational Evaluation of Breadth First Search Spanning Trees"* [8]

Der Pfad 4 → 1 ist ein Greedy Navigationspfad. Es existiert ein Sprung von 3 → 8. Diese Verbindung existiert im Graphen, aber nicht im „Shortest-Path Spanning Tree". Dieses Beispiel in Abbildung 2.2g veranschaulicht, dass der Greedy Navigator eine Abkürzung benutzt. Trotzdem ist der Greedy Navigationspfad mit (4,3,8,1) größer als der kürzeste Weg im Graphen mit (4,2,1).

2.2.3 Bewertungskriterien

Um die Qualität der Navigation zu bewerten, möchte ich einige Bewertungskriterien erläutern.

2.2.3.1 Stretch

Als „*stretch*" bezeichnet man das Verhältnis des Greedy Navigationspfades $h(s,t)$ zum kürzesten Pfad $d^G(s,t)$.

$$\tau(s,t) = \frac{h(s,t)}{d^G(s,t)}, s \neq t.$$

Der Stretch ist 1 wenn der Greedy Navigationspfad gleich dem kürzesten Weg ist.

2.2.3.2 Global Stretch

Der globale Stretch ist der Durchschnitt der Stretches aller möglichen Knotenpaare.

$$\tau = \frac{1}{n(n-1)} \sum_{s \neq t} \frac{h(s,t)}{d^G(s,t)}.$$

2.2.3.3 Success Rate

Wenn wir n verschiedene Navigationen zwischen zufälligen Start- und Endknoten durchführen so ist die „*Success Rate*“ das Verhältnis aller n Navigationen zu den Navigationen bei denen sich der Navigator am Ende im Zielknoten befindet.

Der Wertebereich der Success Rate liegt also zwischen 0 und 1 bzw. zwischen inklusive 0% und 100 %.

Bei eingeführten Abbruchkriterien, wie z.B. die Anzahl von maximalen Navigationsschritten, kann zwar ein Pfad zum Zielknoten existieren, aber es wird vorher abgebrochen, da das Abbruchkriterium erfüllt ist.

Man sollte also nicht annehmen, dass alle erfolgreichen Navigationsversuche auch alle Start- und Endknotenpaare beinhalten, bei denen es einen Pfad gibt.

2.2.3.4 Durchschnittliche Pfadlänge - Average Path Length

Wie der Name schon andeutet, ist die durchschnittliche Pfadlänge jene, die sich ergibt, wenn alle Pfadlängen eines Experimentes mit einer gewählten Hierarchie summiert und durch die Anzahl der Experimente dividiert.

Man kann auch zwischen durchschnittliche Pfadlänge bei erfolgreicher und durchschnittliche Pfadlänge bei nicht erfolgreicher Navigation unterscheiden.

3 Hierarchien

Nun kommen wir zum Hauptthema dieser Arbeit – Hierarchien und dessen Extraktion. [25]

Um sich in einem Graphen effizient fortbewegen zu können benötigt man Informationen. Man könnte von jedem Ausgangspunkt, zu jedem Knoten den kürzesten Weg berechnen, ihn speichern und als Hintergrundwissen weitergeben. Das wäre natürlich mit einer optimalen Lösung verbunden.

In der Praxis ist diese Vorgehensweise jedoch nicht möglich, da einerseits die Rechenlaufzeit mit jedem zusätzlichen Knoten exponentiell steigt und andererseits immense Datenmengen von kürzesten Wegen entstehen würden.

Die Aufgabe der Hierarchie ist es nun, mit relativ kleiner Zusatzinformation gute Navigationsergebnisse zu erreichen. Gute Navigationsergebnisse spiegeln sich in der „*Success Rate*“ und in der „*average pathlength*“ wider. Es ist also eine hohe Erfolgsrate bei gleichzeitig kürzest möglicher durchschnittlicher Pfadlänge gewünscht.

Eine Hierarchie wird erstellt, indem man einen willkürlich gewählten Knoten als Wurzel definiert und sich nun eine Methode überlegt, wie man am besten möglichst viele Knoten hierarchisch zu einer Menge zusammenfassen kann. Man könnte zum Beispiel jeden Nachbarn eines betrachteten Knotens im Graphen, eine Ebene zuweisen. Weißt man jedem Knoten eine Ebene zu, in der der Index der Tiefe entspricht und wiederholt das solange, bis man alle erreichbaren Knoten eingetragen hat, entspräche dies im Groben einer Erstellung einer Hierarchie mittels Breitensuche.

Wenn man nun eine Hierarchie erstellt hat, kann man diese verwenden um Entfernungen verschiedener Knoten schnell ausfindig zu machen. Das geschieht indem man sich nun den Zielknoten in der Hierarchie sucht und sich von ihm in der Hierarchie Schrittweise entfernt. Man speichert sich zu jedem Knoten der Hierarchie die bereits benötigten Schritte, bis man die Hierarchie komplett abgearbeitet hat. Nun wechselt man zum Graphen, betrachtet alle Nachbarn des Knotens und speichert sie sich. Mit der Information, welche Nachbarn der derzeitige Knoten hat, wirft man wieder einen Blick in die Hierarchie und sucht sich dort diese Nachbarn heraus. Der Nachbar mit der geringsten Schrittweite, den man sich vorher gespeichert hat, wird nun als neuer Knoten angenommen bzw. wird zum neuen Navigationszielknoten. Von diesem Knoten aus wiederholt man den Vorgang solange, bis man sich im Knoten mit der Entfernung 0 befindet, welcher dann der Zielknoten ist.

In der Praxis funktioniert diese Vorgehensweise sehr gut, solange sich Start- und Zielknoten in der Hierarchie befinden.

Die Problematik bei der Erstellung von guten Hierarchien ist jedoch, dass man einerseits möglichst kurze Wege erreichen möchte, was sich direkt auf die Breite der einzelnen Ebenen der Hierarchien niederschlägt und andererseits möchte man eine hohe Erfolgsrate erreichen, welche sich jedoch verringert, wenn sich der Zielknoten in einer anderen Teilebene befindet, die keinen gerichteten Weg aufzeigt.

Das Ziel ist es Hierarchien aus beliebigen Informationsnetzwerken zu erstellen die eine effiziente und möglichst kurze Navigation [16] ermöglichen. In meinem Fall wird in erster Linie Wikipedia als Informationsnetzwerk untersucht. Allgemeine Messergebnisse rund um Wikipedia liefert uns auch J. Voss mit „*Measuring Wikipedia“*. [33]

3.1 Produktion von Bäumen

Jetzt stellt sich die Frage, wie man nun aus einem Graphen eine gute Hierarchie produzieren kann. Unter den möglichen Vorgehensweisen findet man unter anderem die Tiefensuche - „*depth-first search*“ (**DFS**) und die Breitensuche - „*breadth-first search*“ (**BFS**).

3.1.1 Tiefensuche - DFS

Die Tiefensuche ist ein gängiges Verfahren in der Informatik. Sie wird meistens verwendet um einen Knoten in einem Graphen zu suchen. Sie zählt zu den uninformierten Suchalgorithmen. [35]

Der Suchablauf funktioniert durch fortlaufende Expansion des jeweils ersten Nachfolgeknotens im Graphen. Das heißt, dass man sich schnell vom Startknoten in die Tiefe entfernt. Sobald kein Nachfolgeknoten mehr existiert, bewegt man sich im Graphen einen Schritt zurück und zwar solange, bis es einen unbesuchten Kindknoten gibt, oder man sich im Startknoten befindet und kein weiterer nicht besuchter Knoten existiert. Wenn ein unbesuchter Knoten gefunden wurde, expandiert man den ersten nicht besuchten Nachfolgeknoten und bewegt sich wieder in die Tiefe.

DFS informell [35]

1. Setze einen Startknoten
2. Expandiere den Knoten und speichere einen noch nicht besuchten Nachfolger in einem Stack
3. Rekursiver Aufruf von DFS für jeden Knoten in dem Stack
    a. Falls das gesuchte Element gefunden wurde – abbrechen und das gefundene Element zurückgeben

b. Gibt es keine noch nicht besuchten Nachfolger - lösche den obersten Knoten im Stack und wende auf den neuen obersten Knoten DFS an

c. Falls der Stack leer ist wurde das Element nicht gefunden

```
DFS(node, goal)
{
  if (node == goal) {
    return node;
  } else
  {
    stack := expand (node)
    while (stack is not empty)
    {
      node' := pop(stack);
      DFS(node', goal);
    }
  }
}
```

Listing 3.1.a: DFS Pseudocode [35]

Laufzeit:

Die Laufzeit der Breitensuche beträgt $O(|V| + |E|)$.

Ist der Graph als Adjazenzmatrix gespeichert, so steigt sie auf: $O(|V|^2)$

Speicherplatzverbrauch:

Die obere Schranke des Speicherverbrauches entspricht der Tiefe des größten Astes und wird dann benötigt, wenn sich der Algorithmus beim letzten Blatt des längsten Astes befindet. [35]

Vollständigkeit

Wenn im Graphen kein Test auf Zyklen durchgeführt wird, so kann es sein, dass ein Ergebnis nicht gefunden wird, obwohl es existiert. Das bedeutet, dass die Tiefensuche nicht vollständig ist. [35]

Optimalität

Die Tiefensuche ist im Allgemeinen nicht optimal wenn sie in ungewichteten Graphen mit monoton steigenden Pfadkosten eingesetzt wird, da als Lösung möglicherweise Pfade gefunden werden, die mit alternativen Routen jedoch viel kürzer ausfallen würden. Der Vorteil der Tiefensuche ist, dass meistens ein Ergebnis schneller gefunden wird und sie weniger Speicherbedarf hat, als die Breitensuche.

Es gibt auch eine Kombination von Tiefen- und Breitensuche die in der Praxis eingesetzt wird, sie nennt sich iterative Tiefensuche. [35]

Folgendes Beispiel zeigt, wie sich die Tiefensuche verhält.

Legt man als Startknoten die Wurzel des Beispielbaumes fest, so würde der Algorithmus in der Reihenfolge der Nummerierung diesen Baum durchlaufen um einen Knoten zu finden.

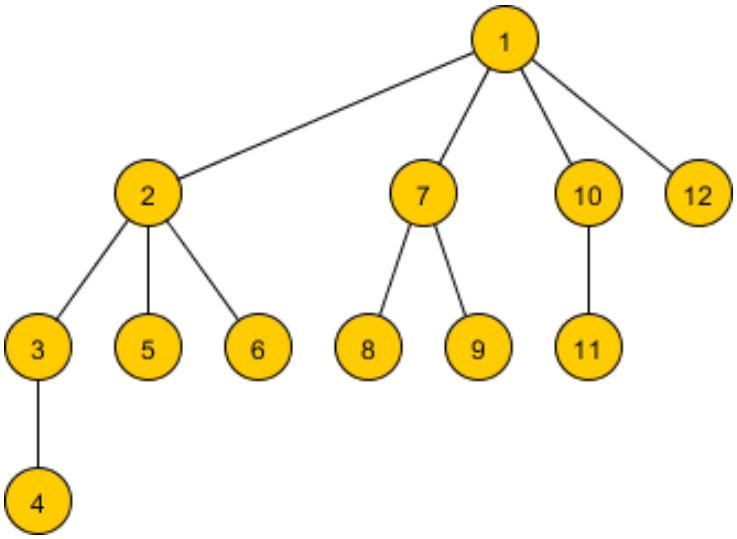


Abbildung 3.1a: Ein Baum dessen Knotennummerierung der Durchlaufreihenfolge von DFS entspricht

Man kann nun mit der Durchlaufreihenfolge einen Baum produzieren. Es werden alle Knoten entsprechend des Durchlaufes in einer Baumstruktur abgespeichert um eine Hierarchie zu erstellen.

Bei einem gerichteten Graphen können mit DFS produzierte Bäume relativ leicht entarten. Dazu folgendes Beispiel: Gegeben ist der gerichteten Graphen $G(V, E)$:

Würde man als Startknoten den Knoten 1 wählen, so könnte ein mit DFS produzierter Baum als Hierarchie dementsprechend entarten. Zur Erstellung einer Hierarchie zur Navigation in einem Graphen ist daher DFS nicht sehr sinnvoll, da hierbei lange und schmale Hierarchien entstehen werden und der Navigator entsprechend der Tiefe der entstandenen Hierarchie, viele Sprünge benötigen wird um ein Ziel zu erreichen. Das Ziel ist es jedoch, kurze Wege zu erreichen, daher eignet sich eine mit DFS erstellte Hierarchie als Hintergrundinformation nicht.

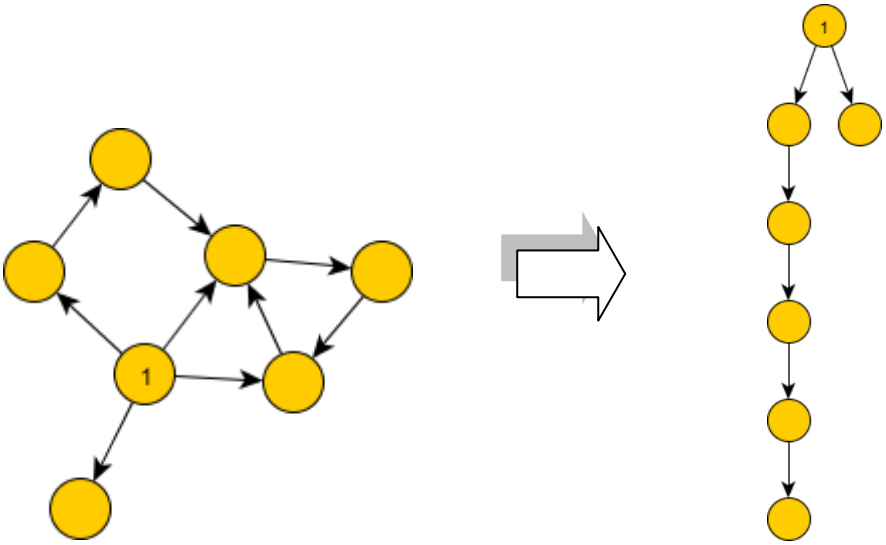


Abbildung 3.1b: Eine mögliche Hierarchie die entstehen kann, wenn man sie mit der Tiefensuche produziert

3.1.2 Breitensuche – BFS

Die Breitensuche verhält sich im Gegensatz zu der Tiefensuche genau umgekehrt.

Sie ist, ähnlich wie die Tiefensuche, ein gängiges Verfahren zum Suchen von Knoten in der Informatik und zählt auch zu den uninformierten Suchmethoden.

Es wird anstelle eines Stacks eine Warteschlange verwendet. Das Ergebnis ist eine Suche in die Breite.

BFS informell [35]

1. Setzen eines Startknotens und speichern des Startknotens in eine Warteschlange
2. Herausnehmen des ersten Knotens der Warteschlange und markieren des Knotens
    a. Falls das gesuchte Element gefunden wurde – abbrechen und das gefundene Element zurückgeben
    b. Andernfalls alle noch nicht markierten Nachfolger des Knotens, die sich noch nicht in der Warteschlange befinden ans Ende der Warteschlange hängen
3. Falls die Warteschlange leer ist, dann wurden alle Knoten durchlaufen und der Zielknoten wurde nicht gefunden. Beenden und „nicht gefunden“ zurückliefern.

4. Schritt 2 wiederholen

```
BFS(start_node, goal_node) {
 for(all nodes i) visited[i] = false;
 queue.push(start_node);
 visited[start_node] = true;
 while(! queue.empty() ) {
  node = queue.pop();
  if(node == goal_node) {
   return true;
  }
  foreach(child in expand(node)) {
   if(visited[child] == false) {
    queue.push(child);
    visited[child] = true;
   }
  }
 }
 return false;
}
```

Listing 3.1.b: BFS Pseudocode [35]

Laufzeit:

Bei einem Graphen $G(V,E)$ beträgt die Laufzeit im schlechtesten Fall $O(|V| + |E|)$, wenn der Graph als Adjazenzliste abgespeichert wurde. [35]

Speicherplatzverbrauch:

Alle entdeckten Knoten werden gespeichert, daher beträgt der Speicherplatzverbrauch $O(|V| + |E|)$. [35]

Vollständigkeit

Existieren in jedem Knoten endlich viele Alternativen, dann ist BFS vollständig. Wenn eine Lösung existiert wird diese auch gefunden, egal ob der Graph endlich ist oder nicht. BFS divergiert wenn keine Lösung existiert in einem unendlichen Graphen. [9]

Optimalität

Die Breitensuche ist im Allgemeinen optimal. Man kann die Breitensuche auch mit gewichteten Kanten benutzen, dann wird, statt der Erhöhung um den Wert 1, der noch nicht markierte Knoten mit dem Wert des Gewichtes seiner Kante erhöht. In diesem Fall wäre das Ergebnis, je nach Gewichtung, nicht notwendigerweise auch der tatsächlich kürzeste Weg, da in diesem Fall das Ergebnis der Pfad mit den geringsten Pfadkosten ist. [35]

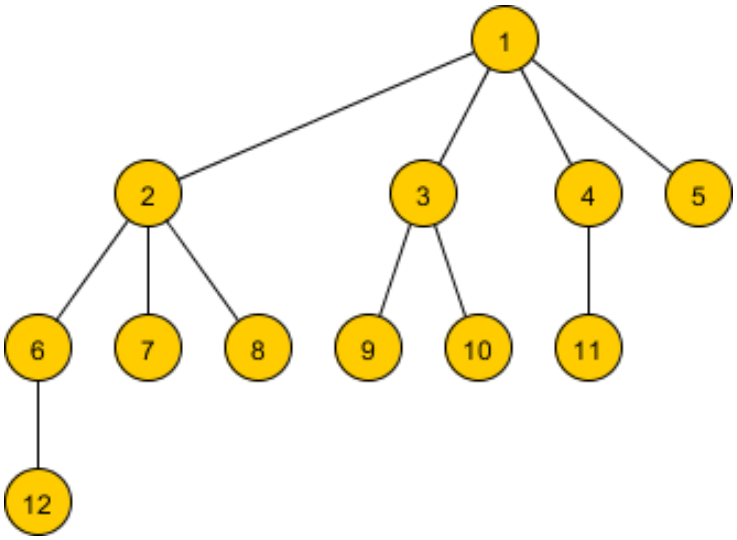


Abbildung 3.1c: Der Baum aus Abbildung 3.1a dessen Knotennummerierung der Durchlaufreihenfolge von BFS entspricht

Um eine Hierarchie mit Hilfe der Breitensuche zu erstellen, muss anstelle der Markierung nur die Tiefe, was gleichzeitig, der Distanz zum Wurzelknoten entspricht, gespeichert werden. Es muss bei einem Kindknoten nur der Distanzwert Vaterknoten um 1 erhöht werden.

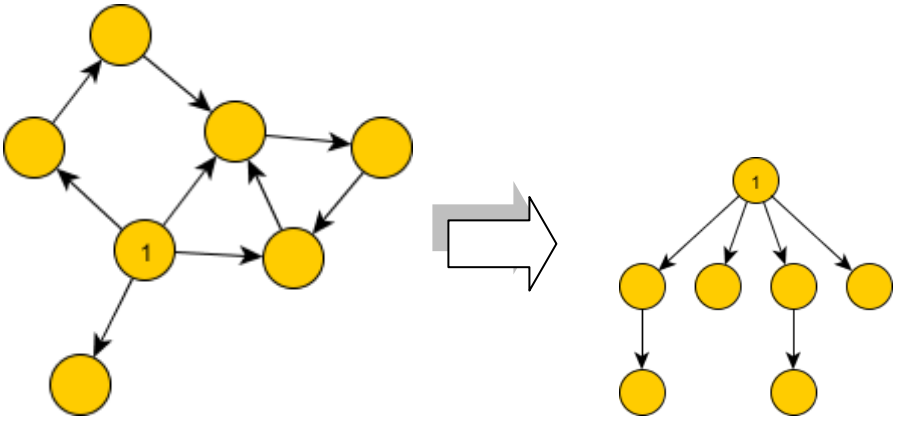


Abbildung 3.1d: Eine mögliche Hierarchie die entstehen kann, wenn man sie mit der Breitensuche produziert.

Bei dem gerichteten Graphen, wie in Abbildung 3.1c gezeigt, wird eine Hierarchie entstehen die so aussehen könnte wie in Abbildung 3.1d.

Wie man gut erkennt, eignet sich die Breitensuche sehr gut um Distanzen zu einem Startknoten darzustellen.

Wenn man den erzeugten Baum betrachtet, sollte er gute Eigenschaften für eine Hierarchie ergeben, er wird automatisch sehr breit werden und in den ersten paar Ebenen werden sich im Durchschnitt die meisten Knoten befinden. Die Pfadlänge wird im Durchschnitt auch recht gering sein, es werden sich bei zunehmender Verzweigung in einem Graphen aber die durchschnittlichen Pfadlängen verschlechtern, da die Hierarchie eine konstante Kantenanzahl haben wird. Weil jeder Knoten nur eine eingehende Kante besitzt, wird die Hierarchie $n-1$ Kanten besitzen, wobei n die Anzahl der Knoten in der Hierarchie ist.

Mit den wenigen Kanten können also mit Sicherheit nicht alle kürzesten Wege in einem komplexen Graphen abgedeckt werden.

3.2 Power Law und strongly-connected components

Das Power Law oder Potenzgesetz [29] ist eine funktionale Beziehung zwischen zwei Mengen. Sie trifft dann zu, wenn die Frequenz des Auftretens eines Ereignisses in der Potenz einer Eigenschaft des Attributes dieses Ereignisses auftritt.

Die Gesetzmäßigkeit hat die Form: $\boldsymbol{y = a * x^b}$.

Das Potenzgesetz tritt bei vielen natürlichen Phänomenen auf. Beispielsweise verhält sich die Population von Städten nach dem Potenzgesetz und auch bei den Worthäufigkeiten findet man dessen Gültigkeit.

In Fall der Degree Verteilung der Hierarchien zeigt sich auch ein direkter Zusammenhang.

Im Paper "*The Web as a graph*" [10] wird auch über dieses Phänomen diskutiert. Es geht darum, das Web bzw. dessen Seiten als Knoten und dessen Hyperlinks als Kanten eines großen gerichteten Graphen anzusehen und diesen Graphen, der *„Web Graph"* zu untersuchen.

Der durchschnittliche Knoten hat in diesem Fall im Schnitt ungefähr 7 ausgehende Kanten (=Hyperlinks). Es wurde in dem Paper auch die Degree Verteilung diskutiert. Das resultierende Ergebnis lautet, dass die Anzahl der Webseite mit dem in-degree i proportional zu $1/i^2$ für einige $x > 1$ ist. Dieses Ergebnis wurde von Albert [11] und Broder [13] bestätigt, indem sie Seiten der Notre Dame Universität zu 200 Millionen Knoten im Web verfolgt haben. In allen Experimenten war der Exponent x im Potenzgesetz konsistent.

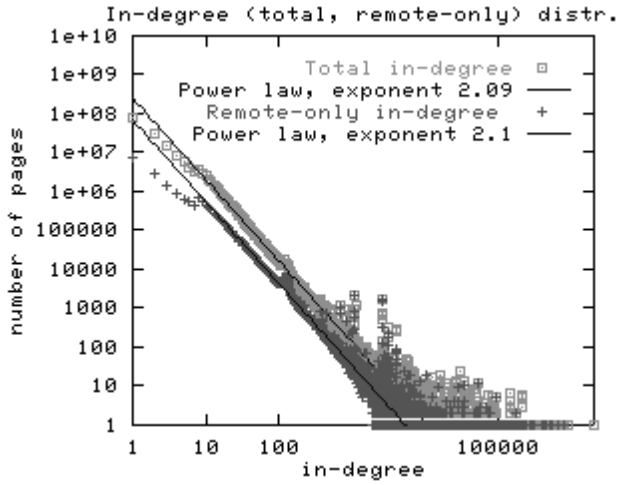


Abbildung 3.2a: In-Degree Verteilung Notre Dame Experimentes aus dem Paper „*The Web as a graph*“ [10].

In dem log-log Plot von Abbildung 3.2a und 3.2b kann man deutlich erkennen, dass das Potenzgesetz eindeutig zutrifft.

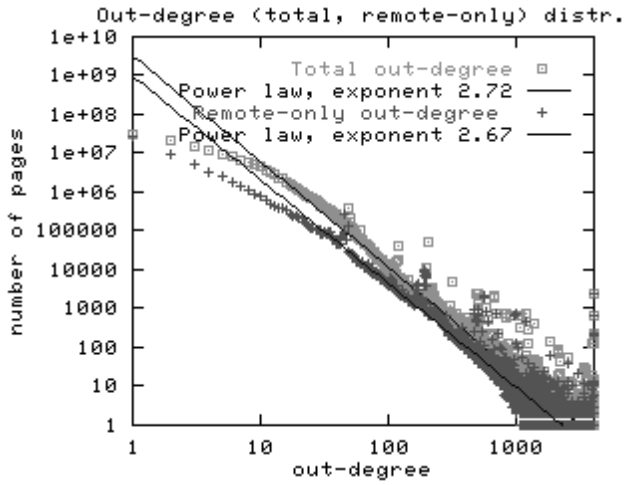


Abbildung 3.2b: Out-Degree Verteilung Notre Dame Experimentes aus dem Paper „*The Web as a graph*“ [10].

Im Paper folgen aus den Experimenten genaue Werte für verschiedene Arten von Komponenten. Das Experiment wurde mit Hilfe eines von der Suchmaschine Altavista bereitgestellten Crawl-Datensatzes von Oktober 1999 mit 200 Millionen Seiten und 1,5 Milliarden Links durchgeführt. [12]

Es wurde evaluiert, dass in einer schwach verbundene Komponente („*weakly-connected component*“) mehr als 90 % aller Seiten, 186 Millionen Kanten, des Datensatzes liegen.

Eine schwach verbundene Komponente ist ein Set aus Seiten welche über Links erreicht werden können, wenn man Ihnen in beide Richtungen folgt.

Die größte stark verbundene Komponente (*„strongly-connected component“*), welche dem Set aller Seitenpaare entspricht, die man mit einem gerichteten Pfad im Graphen erreichen kann, enthielt 56 Millionen Knoten. Die zweitgrößte nur mehr 50.000 Knoten. Die Anzahl der Komponenten, egal ob stark oder schwach verbunden, unterliegen auch dem Potenzgesetz.

Broder führte ein Modell des Webs als Karte ein, wie in Abbildung 3.2b zu sehen ist.

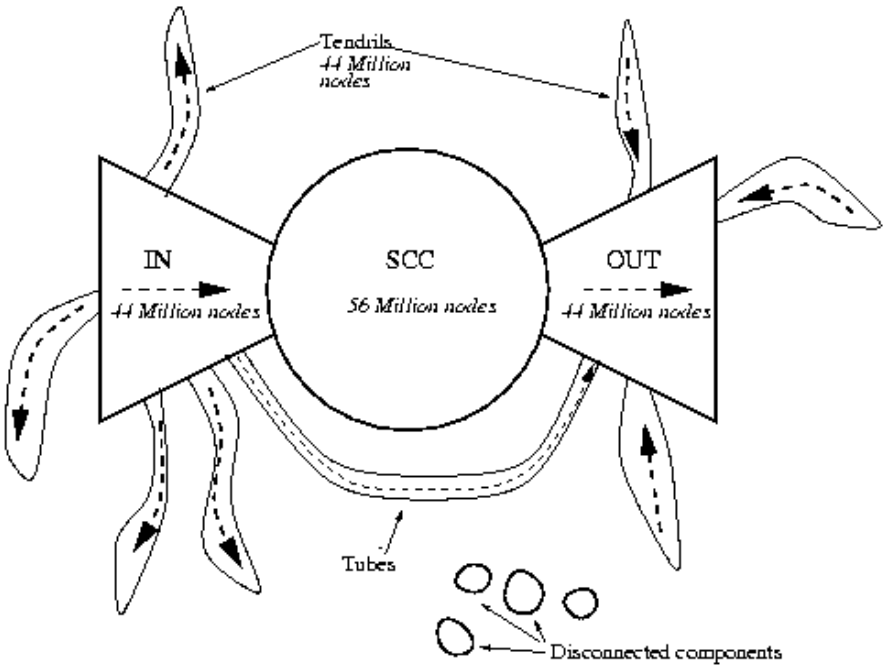


Abbildung 3.2c: Eine Darstellung des Webs aus *„Graph structure in the web: experiments and models”* [13]

Mit IN wurden alle Knoten bezeichnet, die über einen Pfad zu der SCC verfügen. Alle Knoten in OUT können von allen Knoten im SCC erreicht werden. *TENDRILS* sind Knoten sind alle Knoten, die über keine Verbindung zum SCC verfügen.

Die Ergebnisse von Albert [11] ergaben eine Voraussage, dass es für die meisten Paare von Webseiten u und v eine direkte Verbindung mit der

Pfadlänge 19 gibt, wenn diese existiert. Die Ergebnisse dieser Arbeit spielen sich auch in diesem Pfadlängenbereich ab.

In der Zwischenzeit kam es jedoch noch zu weiteren Forschungsergebnissen, welche besagen, dass die meisten Paare von Seiten eine unendliche gerichtete Distanz besitzen.

Im Experiment von Kapitel 4 besteht der größte verbundene Teilgraph aus 5.826.222 Knoten. Er beinhaltet also ungefähr 63 % aller Knoten des Graphen.

IN und OUT betragen in etwa 10%, was natürlich im direkten Zusammenhang mit der bestmöglichen Erfolgsrate im nachfolgenden Experiment steht.

3.3 Degree Verteilung der entstandenen Hierarchien

Auch in meinem Experiment hat sich gezeigt, dass sich die Degree Verteilungen nach dem „*Power Law*“ verhalten.

3.3.1 Degree Verteilung von BFS Hierarchien

Der In-Degree aller Knoten unterhalb des Wurzelknotens beträgt in den extrahierten Hierarchien immer 1, da es sich ja um einen Baum handelt. Hat ein Knoten den Grad 1, dann hat er eine eingehende Kante und keine ausgehende, beim Grad 5 wären es eine eingehende und 4 ausgehende Kanten usw.

Bei den folgenden Auswertungen handelt es sich also um die Anzahl von Knoten mit einem gewissen Knotengrad. Es gibt etwas Aufschluss über die Struktur der Hierarchie und soll zeigen, dass sich in allen Fällen die Verteilungen so verhalten wie im Potenzgesetz beschrieben.

Betrachtet werden die jeweils besten Hierarchien aus den folgenden Kategorien:

- BFS in-high
- BFS in-low
- BFS out-high
- BFS out-low
- DFS in-high
- DFS in-low
- DFS out-high
- DFS out-low

Bei Hierarchien erstellt durch **BFS mit hohem In-Degree** als Wurzelknoten zeigte sich folgendes Bild.

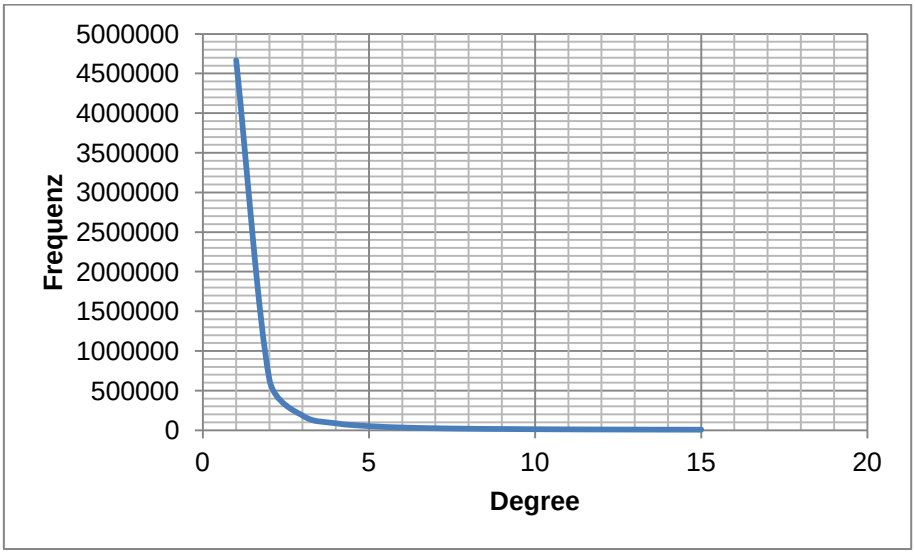


Abbildung 3.3a: Degree Verteilung einer **BFS High In-Degree** Hierarchie

94,23 % aller Knoten haben einen Degree von 1-3 in dieser Hierarchie.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	4667170	**11**	9178	**21**	2580
2	639338	**12**	7615	**22**	2410
3	183299	**13**	6608	**23**	2315
4	85779	**14**	5779	**24**	2011
5	50109	**15**	5047	**25**	1898
6	32518	**16**	4324	**26**	1701
7	23193	**17**	3926	**27**	1599
8	17416	**18**	3459	**28**	1570
9	13864	**19**	3074	**29**	1373
10	11224	**20**	2948	**30**	1286

Tabelle 3.3a: Degree Verteilung einer **BFS High In-Degree** Hierarchie im Überblick.

Bei Hierarchien erstellt durch **BFS mit niedrigem In-Degree** ergab sich folgendes.

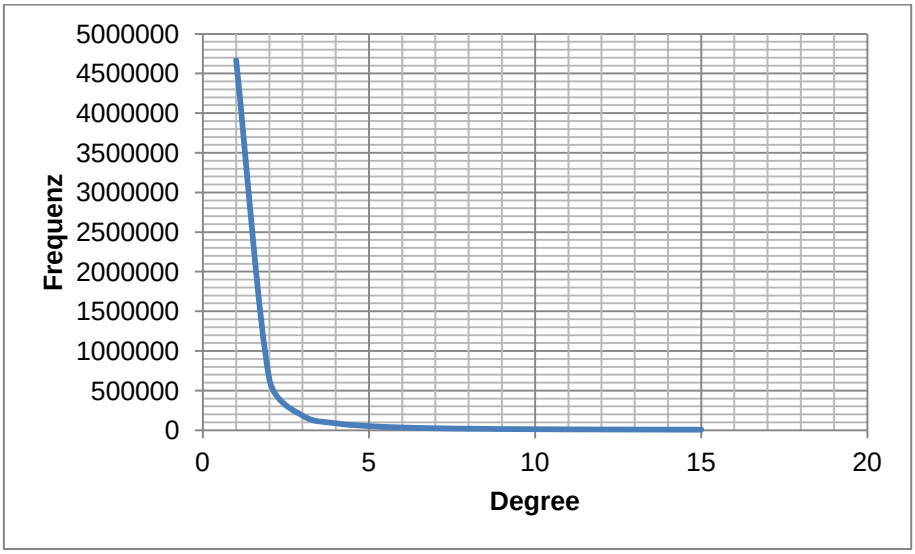


Abbildung 3.3b: Degree Verteilung einer **BFS Low In-Degree** Hierarchie

94,21 % aller Knoten haben einen Degree von 1-3 in dieser Hierarchie.

Als niedrigen In-Degree wurde der Degree 100 definiert und relativ ähnliche Werte sind das Ergebnis.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	4666069	**11**	9153	**21**	2458
2	640019	**12**	7728	**22**	2437
3	183066	**13**	6579	**23**	2113
4	86221	**14**	5767	**24**	2028
5	50425	**15**	5037	**25**	1830
6	32800	**16**	4405	**26**	1737
7	23238	**17**	3976	**27**	1599
8	17679	**18**	3586	**28**	1572
9	13537	**19**	3153	**29**	1378
10	11009	**20**	2838	**30**	1299

Tabelle 3.3b: Degree Verteilung einer **BFS Low In-Degree** Hierarchie im Überblick.

Bei Hierarchien erstellt durch **BFS mit hohem Out-Degree** als Wurzelknoten zeigte sich ebenfalls die Gesetzmäßigkeit des Potenzgesetzes:

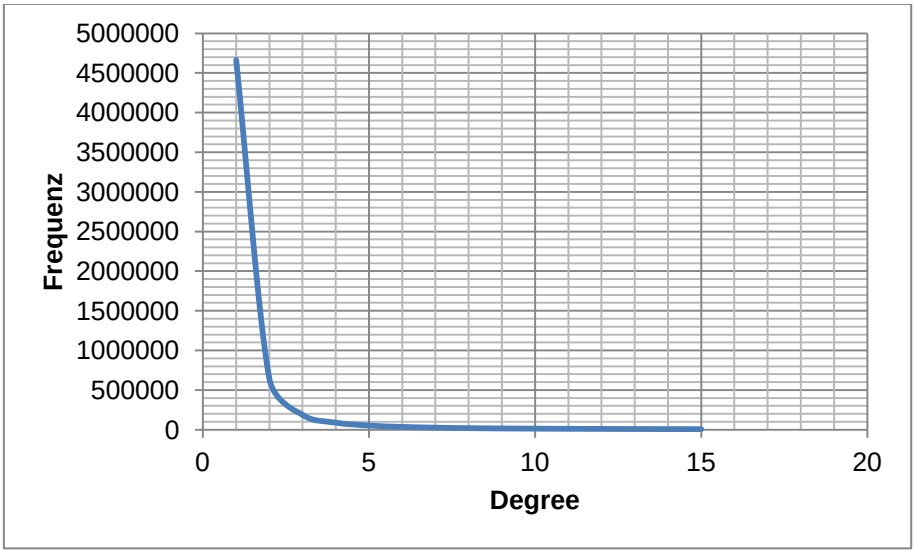


Abbildung 3.3c: Degree Verteilung einer **BFS High Out-Degree** Hierarchie

Mit **94,18 %** aller Knoten mit Degree von 1-3 ähnelt auch diese Hierarchie den anderen mit BFS erstellten Hierarchien stark aber es gibt kleine Unterschiede.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	4661807	**11**	9214	**21**	2584
2	640397	**12**	7874	**22**	2438
3	185134	**13**	6561	**23**	2175
4	86873	**14**	5698	**24**	1983
5	50494	**15**	4907	**25**	1807
6	33368	**16**	4551	**26**	1796
7	23728	**17**	3872	**27**	1578
8	17664	**18**	3462	**28**	1481
9	13746	**19**	3213	**29**	1347
10	11171	**20**	2834	**30**	1230

Tabelle 3.3c: Degree Verteilung einer **BFS High Out-Degree** Hierarchie im Überblick.

Bei der letzten BFS Kategorie, eine Hierarchie erstellt durch **BFS mit niedrigem Out-Degree** als Wurzelknoten zeigte sich Folgendes:

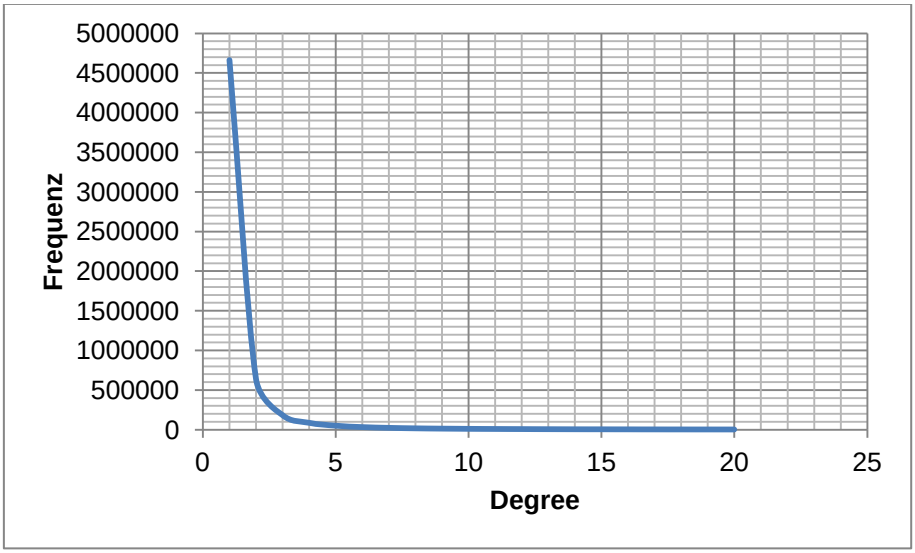


Abbildung 3.3d: Degree Verteilung einer BFS Low Out-Degree Hierarchie

94,20 % aller Knoten haben hierbei einen Degree von 1-3.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	4665050	**11**	9156	**21**	2608
2	639611	**12**	7794	**22**	2362
3	183687	**13**	6557	**23**	2134
4	86877	**14**	5560	**24**	2000
5	50628	**15**	5103	**25**	1922
6	32780	**16**	4448	**26**	1746
7	23371	**17**	3854	**27**	1589
8	17282	**18**	3430	**28**	1459
9	13792	**19**	3209	**29**	1420
10	11117	**20**	2879	**30**	1312

Tabelle 3.3d: Degree Verteilung einer **BFS Low Out-Degree** Hierarchie im Überblick

Mit der Wahl des Wurzelknotens kann man die Degree-Verteilung der Hierarchie leicht beeinflussen, das Potenzgesetzt gilt in jedem Fall eindeutig.

3.3.2 Degree Verteilung von DFS Hierarchien

Bei Hierarchien erstellt durch **DFS mit hohem In-Degree** als Wurzelknoten zeigte sich folgende Degree Verteilung:

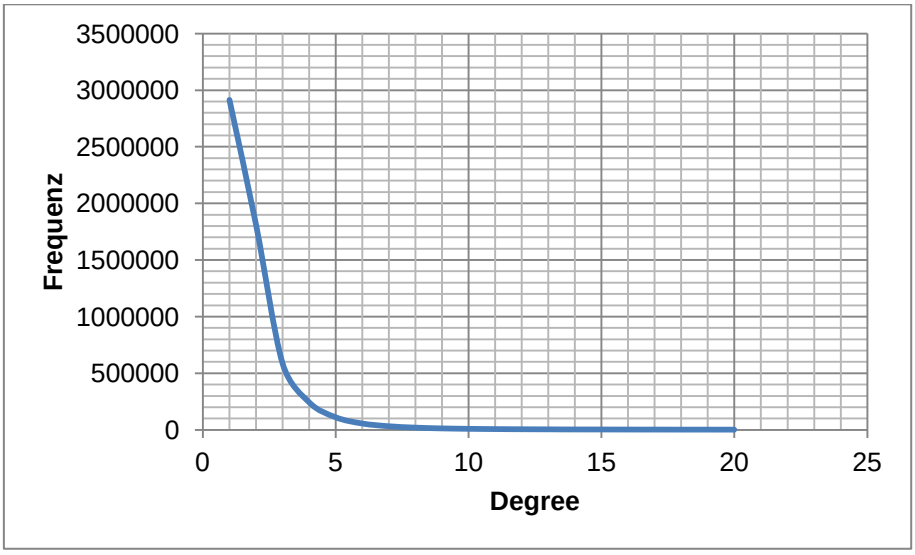


Abbildung 3.3e: Degree Verteilung einer **DFS High In-Degree** Hierarchie

Im Vergleich zu den BFS Hierarchien, erkennt man, dass sich weniger Knoten in die Kategorie von 1-3 Graden pro Knoten kategorisieren lassen.

91,12 % aller Knoten haben einen Degree von 1-3 in dieser Hierarchie.

Mit ca. **3%** weniger Knoten in dieser Kategorie gibt es also einen signifikanten Unterschied zu BFS Hierarchien.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	2914268	**11**	5830	**21**	865
2	1812093	**12**	4568	**22**	783
3	585485	**13**	3462	**23**	684
4	242334	**14**	2684	**24**	575
5	110101	**15**	2237	**25**	468
6	55243	**16**	1801	**26**	494
7	30961	**17**	1530	**27**	406
8	18738	**18**	1257	**28**	401
9	11884	**19**	1113	**29**	342
10	8363	**20**	939	**30**	326

Tabelle 3.3e: Degree Verteilung einer **DFS High In-Degree** Hierarchie im Überblick

Bei Hierarchien erstellt durch **DFS mit niedrigem In-Degree** als Wurzelknoten ergab sich ebenfalls, wie in Abbildung 3.3f gezeigt, einen Verlauf der der Gesetzmäßigkeit des Potenzgesetzes entspricht.

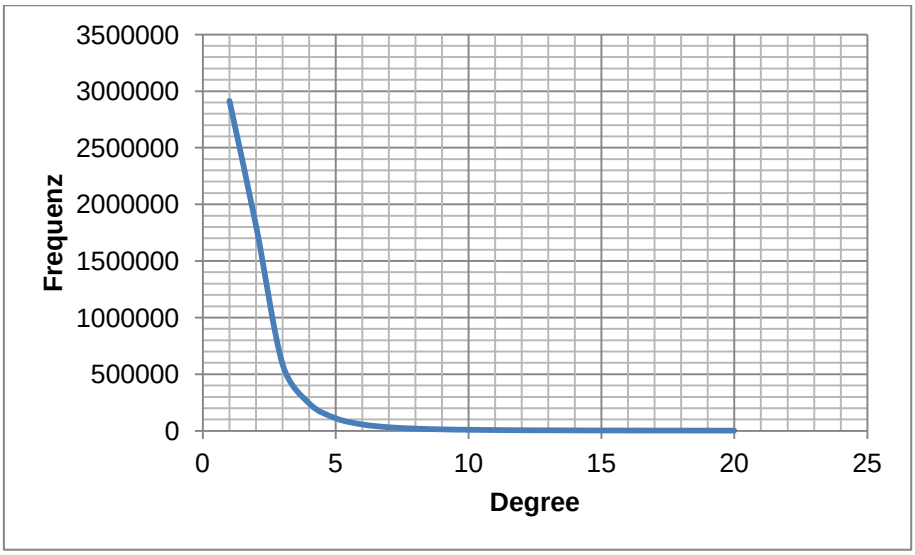


Abbildung 3.3f: Degree Verteilung einer **DFS Low In-Degree** Hierarchie

Die entstandene DFS Hierarchie ist praktisch identisch mit der DFS High In-Degree Hierarchie.

Auch **91,12%** aller Knoten haben einen Degree von 1-3. Erst in der dritten Kommastelle unterscheidet sich hier die Degree Verteilung der Hierarchie.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	2914220	**11**	5940	**21**	851
2	1811971	**12**	4506	**22**	760
3	586114	**13**	3487	**23**	690
4	241454	**14**	2787	**24**	563
5	110181	**15**	2192	**25**	497
6	55630	**16**	1787	**26**	436
7	30630	**17**	1528	**27**	440
8	18754	**18**	1330	**28**	402
9	12011	**19**	1109	**29**	342
10	8290	**20**	960	**30**	326

Tabelle 3.3f: Degree Verteilung einer **DFS Low In-Degree** Hierarchie im Überblick

Wie man erkennen kann, ändert sich das Ergebnis durch die Wahl eines Wurzelknotens mit hohem In-Degree oder niedrigem In-Degree bei der Erstellung einer Hierarchie mit DFS kaum.

Folgendes Ergebnis brachten die Hierarchien erstellt durch **DFS mit hohem Out-Degree** als Wurzelknoten.

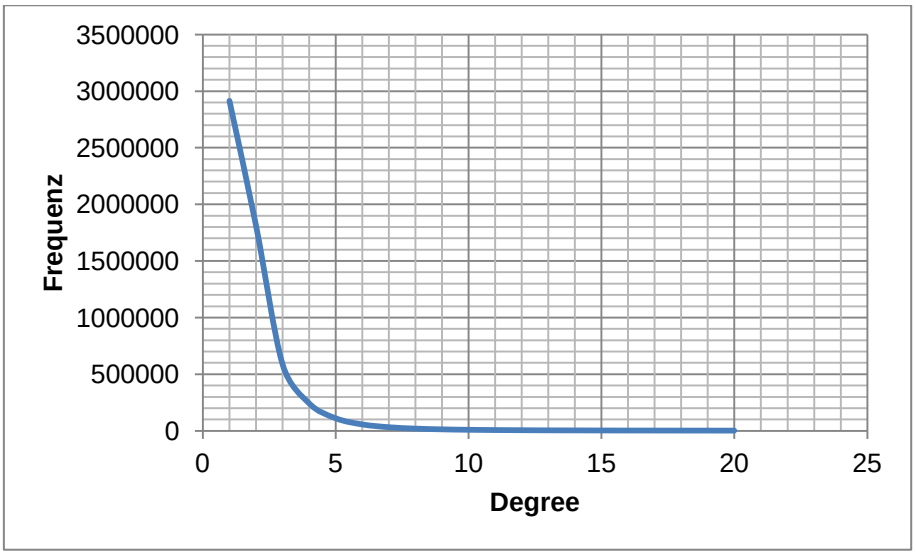


Abbildung 3.3g: Degree Verteilung einer **DFS High Out-Degree** Hierarchie

Wie auch in den vorigen durch DFS erstellten Hierarchien finden wir hier wieder **91,12 %** aller Knoten in der Kategorie Degree 1-3.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	2914324	**11**	5967	**21**	851
2	1812162	**12**	4520	**22**	727
3	585747	**13**	3375	**23**	665
4	241595	**14**	2781	**24**	583
5	110035	**15**	2162	**25**	505
6	55618	**16**	1801	**26**	471
7	30958	**17**	1526	**27**	436
8	18756	**18**	1318	**28**	378
9	11986	**19**	1154	**29**	328
10	8199	**20**	985	**30**	327

Tabelle 3.3g: Degree Verteilung einer **DFS High Out-Degree** Hierarchie im Überblick

Scheinbar ändert sich dieses Verhalten auch nicht bei der Wahl eines hohen In- oder Out-degree Wurzelknotens. Das nächste Ergebnis bestätigt das.

Aus der letzten Kategorie **DFS mit niedrigem Out-Degree** als Wurzelknoten ergibt ebenfalls folgende Degree-Verteilung.

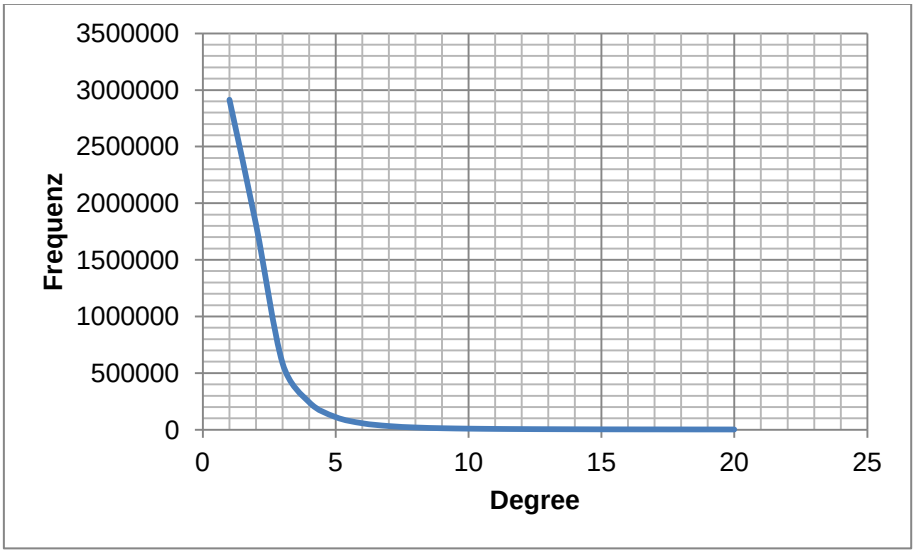


Abbildung 3.3h: Degree Verteilung einer **DFS Low Out-Degree** Hierarchie

Wie vermutet bildet sich auch hierbei ein unverändertes Bild.

91,12 % aller Knoten haben einen Degree von 1-3 in dieser Hierarchie.

Degree	Frequenz	Degree	Frequenz	Degree	Frequenz
1	2914555	**11**	5843	**21**	815
2	1812107	**12**	4446	**22**	759
3	584928	**13**	3489	**23**	657
4	241738	**14**	2777	**24**	580
5	110279	**15**	2192	**25**	551
6	55884	**16**	1804	**26**	466
7	31012	**17**	1524	**27**	416
8	18594	**18**	1299	**28**	409
9	12009	**19**	1145	**29**	357
10	8311	**20**	933	**30**	325

Tabelle 3.3h: Degree Verteilung einer **DFS Low Out-Degree** Hierarchie im Überblick

Es hat sich gezeigt, dass man durch Wahl eines Wurzelknotens bei der Erstellung einer Hierarchie mit DFS zumindest die entstehende Degree Verteilung kaum verändern kann.

4 Experiment: Wikipedia

4.1 Implementierung der Hierarchieerstellung

Zuerst wurden die Breitensuche und die Tiefensuche in SNAP adaptiert, um damit Bäume bzw. Hierarchien erzeugen zu können.

Dazu wurde der Pseudocode der beiden Algorithmen von Kapitel 3, in C++ implementiert und als Klasse in SNAP eingefügt.

Bei der Evaluierung der mit der Tiefensuche erzeugten Hierarchien, musste das Limit der möglichen rekursiven Aufrufe erhöht werden, weil im Navigator das Einlesen der Hierarchien über eine rekursive Funktion durchgeführt wird. Die Anzahl der existierenden Ebenen ist in dem Fall gleichzeitig die Anzahl der rekursiven Aufrufe. (1,9 Millionen)

4.2 Laufzeit

Die Laufzeit der Hierarchieerstellung besteht einerseits aus der Erzeugung von k zufälligen Knotenpaaren bestehend aus Start- und Zielknoten, aus dem Einlesen des Graphen mit n Knoten und m Kanten, der Berechnung der kürzesten Wege der k zufälligen Paare, der Anwendung der Breiten- oder Tiefensuche.

Der größte Rechenaufwand ergab sich ganz klar beim Errechnen der kürzesten Distanzen für die 100.000 zufälligen Knotenpaare.

Es wurde von der TU-Graz ein Cluster zur Verfügung gestellt, um mit 16 Prozessoren, die parallele Erstellung und Auswertung per batch script zu ermöglichen.

4.3 Datengröße

Der Wikipedia Graph liegt im Textformat als auch binär vor.

Die Kanten und Knoten benötigen im Textformat 3,9 Gigabyte Speicher. Der binär gespeicherte Graph benötigt dagegen nur mehr 2,3 Gigabyte.

Der verwendete Wikipedia Graph [34] besteht aus 9.305.679 Knoten und 255.517.245 Kanten. Der größte zusammenhängende Teilgraph besitzt 5.826.222.

4.4 Random Pairs

Es wurden 100.000 verschiedene Random Pairs erzeugt. Das heißt, 100.000 Knotenpaare, die einen Start- und einen Zielknoten beinhalten.

4.5 Shortest Paths

Von den 100.000 Random Pairs, wurden auch die „Shortest Paths“ berechnet. Die kürzeste Wege Berechnung in einem derartigen Graphen, nahm einige Wochen in Anspruch. Von den 100.000 Zufallspaaren, waren nur 62630 tatsächlich verbunden. Dass diese Paare tatsächlich zufällig gewählt wurden, lässt sich auch im Verhältnis der 5.826.222 zusammenhängenden Knoten mit der Anzahl der gesamten 9.305.679 Knoten belegen. Dieses Verhältnis beträgt 0,626.

Bei 100.000 Knotenpaaren sollten also im Schnitt mindestens 62600 Paare auch tatsächlich einen Pfad besitzen, was in diesem Experiment mit 62630 tatsächlich verbundenen Paaren sehr genau eingetroffen ist.

Der durchschnittliche kürzeste Weg der 62630 zufälligen Paare in diesem Experiment beträgt: 5,426792272. [17]

4.6 Ablauf der Experimente

Die erste Aufgabe war es die 20 Knoten im Graphen zu suchen, die die meisten eingehenden Kanten (in-degree) besitzen.

Die Implementierung der Hierarchieerzeugung wurde dementsprechend erweitert, dass automatisch die 20 mit dem höchsten Grad eingehender Knoten gewählt werden können.

Es waren auch die 20 Knoten mit den meisten ausgehenden Kanten, die 20 Knoten mit In-Degree 100 und 20 Knoten mit Out-Degree 100 interessant.

Es wurde bewusst als untere Degree-Schranke 100 gewählt, da man ansonsten mit keinen aussagekräftigen Ergebnissen rechnen kann. Knoten mit nur einer ausgehenden oder eingehenden Kante als Wurzelknoten zu wählen ist generell nicht sinnvoll. Das Verhalten des Grades der Knoten soll gezeigt werden, deswegen wurde diese untere Schranke eingeführt um eine qualitativ hochwertige Evaluierungen zu gewährleisten.

Diese 80 Knoten bilden nun den Wurzelknoten der jeweiligen Hierarchien, die mit der Breitensuche erstellt werden um weitere Aussagen treffen zu können. Mit Hilfe der Implementierung wurden nun 80 Hierarchien mit der Breitensuche erzeugt und mit den jeweils 100.000 zufälligen Start- und Endknotenpaaren dessen Qualität über den Navigator evaluiert.

Das Experiment wurde um die Erstellung von Hierarchien mit der Tiefensuche erweitert.

4.7 Visualisierung der entstandenen Hierarchien

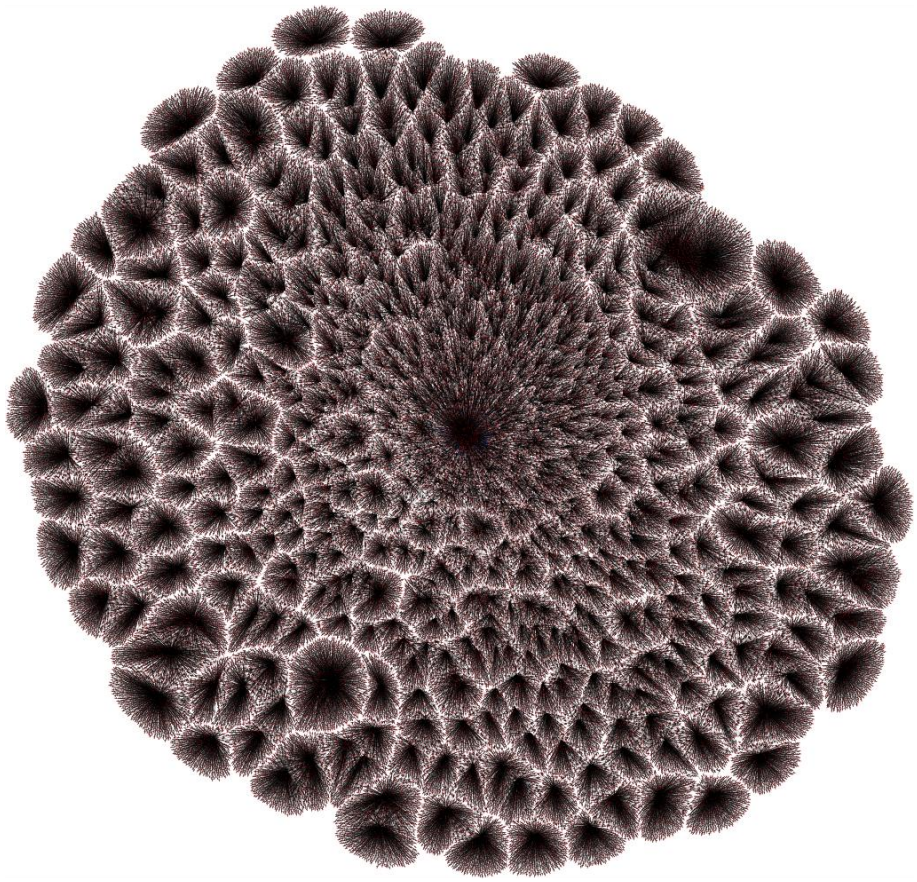


Abbildung 4.7a: Die ersten **drei** Ebenen (inklusive Wurzelknoten) der Hierarchie mit Wurzelknoten 4689264 (BFS In-High).

Die Abbildung 4.7a zeigt eine der am besten abgeschnittenen Hierarchien. Es handelt sich um eine mit BFS erstellter Hierarchie, dessen Wurzelknoten einen hohen In-Degree besitzt.

Man erkennt, dass die 3. Ebene extreme Ausmaße angenommen hat. Die 2. Ebene ist blau eingefärbt und fast vollständig durch die 3. Ebene bedeckt. Bei der Visualisierung der 4. Ebene gab es Speichertechnisch zu wenige Möglichkeiten, um diese Hierarchie zu rendern.

In der Abbildung 4.7a wurden um die 125.000 Kanten und Knoten visualisiert.

Mit einer Success Rate von 97,30 % und einem Stretch von 3,62 ist diese Hierarchie eine mit den besten Ergebnissen.

Der Wurzelknoten hat 119282 eingehende und 1090 ausgehende Kanten. Die durchschnittliche Pfadlänge war 19,64.

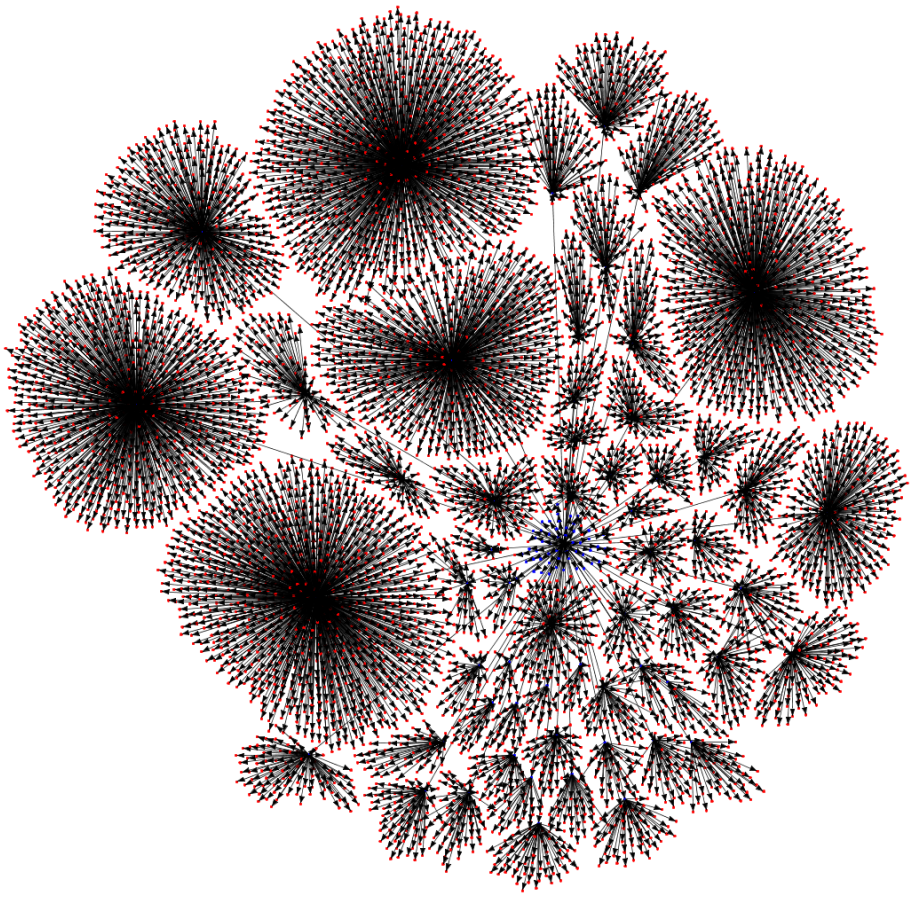


Abbildung 4.7b: Die ersten **drei** Ebenen (inklusive Wurzelknoten) der Hierarchie 1492637 (BFS In-Low)

Die Abbildung 4.7b zeigt eine der am besten abgeschnittenen Hierarchien in der Kategorie BFS In-Low. Es handelt sich um eine mit BFS erstellter Hierarchie, dessen Wurzelknoten einen niedrigen In-Degree besitzt.

Im Gegensatz zu Abbildung 4.7a befinden sich hier in den ersten 3 Ebenen viel weniger Kanten und Knoten. Demensprechend ist auch die durchschnittliche Pfadlänge gegenüber der in 4.7a dargestellten Hierarchie größer. Diese Hierarchie hat jedoch trotzdem recht gut abgeschnitten.

In der Abbildung 4.7b wurden zum Vergleich nur um die 6000 Kanten und Knoten visualisiert.

Mit einer Success Rate von 96,49 % und einem Stretch von 3,99 schneidet diese Hierarchie gut ab.

Der Wurzelknoten hat 100 eingehende und 109 ausgehende Kanten.

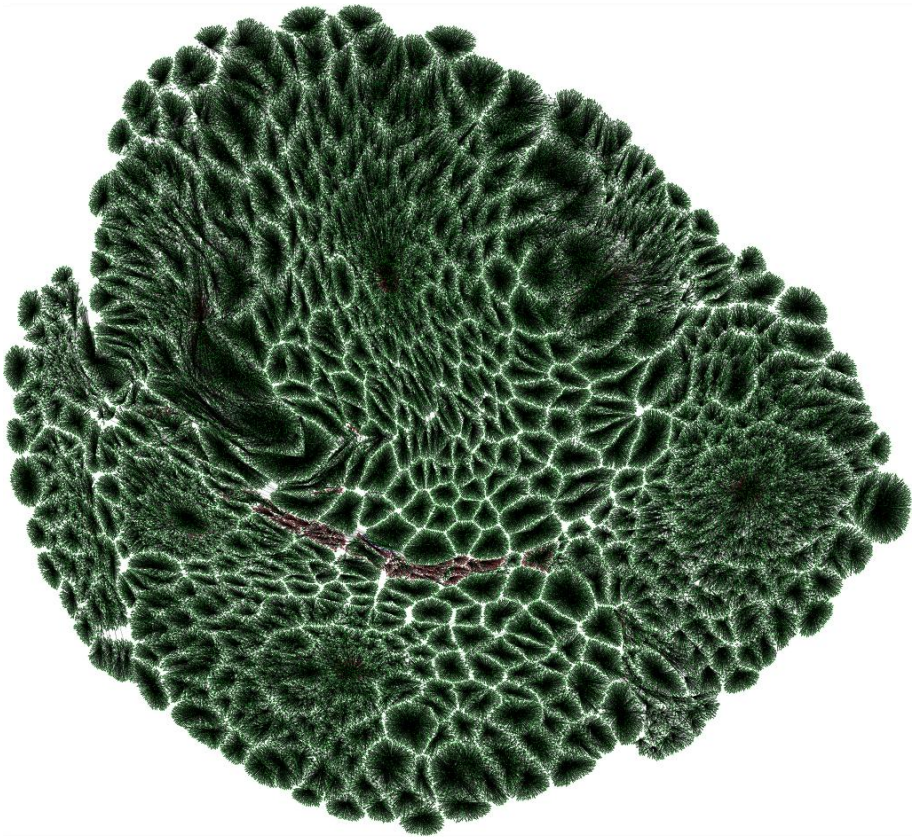


Abbildung 4.7c: Die ersten **vier** Ebenen (inklusive Wurzelknoten) der Hierarchie 1492637 (BFS In-Low)

Diese Abbildung zeigt die Hierarchie der Abbildung 4.7b in **vier** Ebenen.

In der vierten Ebene ergibt sich hierbei schon eine Summe von ca. 260.000 Knoten und Kanten. Zum Vergleich: Die Hierarchie in der Abbildung 4.7a hätte in der Visualisierung der 4. Ebene bereits um die 2.000.000 Knoten und Kanten.

Durch diesen Umstand ergibt sich natürlich auch die etwas längere durchschnittliche Pfadlänge im Gegensatz der Hierarchie in Abbildung 4.7a.

In 4.7c kann man gut erkennen, dass sich in der 4. Ebene dieser Hierarchie einige Knoten befinden, die einen großen Degree zu nicht bekannten Knoten haben, da dies in viele Kinderknoten resultiert.

Die zweite Ebene wurde blau, die dritte Ebene rot und die vierte Ebene grün eingefärbt. Der rote schmale Bereich entspricht der Abbildung 4.7b. Wie man sieht ist musste dieser Bereich sehr stark skaliert werden was auf eine gewaltige Ausdehnung schließen lässt.

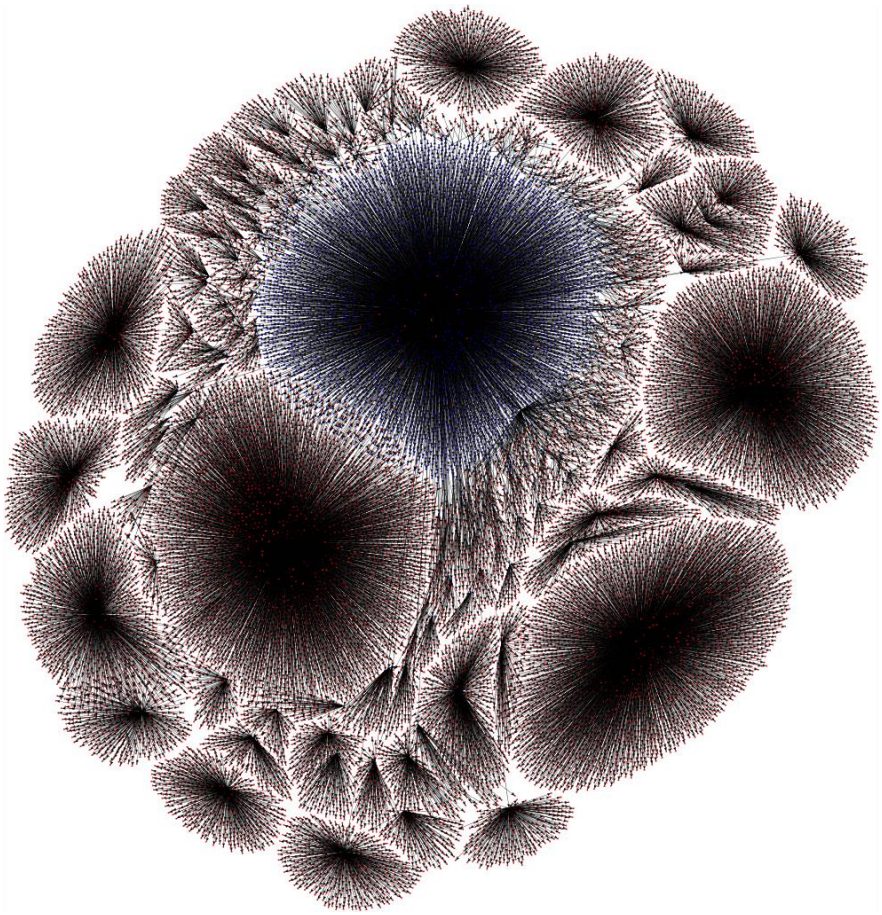


Abbildung 4.7d: Die ersten **drei** Ebenen (inklusive Wurzelknoten) der Hierarchie 1256259 (BFS Out-High).

Die Abbildung 4.7d zeigt eine der am besten abgeschnittenen Hierarchien in der Kategorie BFS Out-High. Es handelt sich um eine mit BFS erstellter Hierarchie, dessen Wurzelknoten einen hohen Out-Degree besitzt.

Es befinden sich hier in den ersten 3 Ebenen wieder etwas mehr Kanten und Knoten. Trotzdem schneidet sie gegenüber der in 4.7b und 4.7c dargestellten Hierarchie schlechter ab.

In der Abbildung 4.7d wurden um die 25000 Kanten und Knoten visualisiert.

Mit einer Success Rate von 93,74 % und einem Stretch von 4,09 schneidet diese Hierarchie passabel ab.

Der Wurzelknoten hat 34 eingehende und 4702 ausgehende Kanten.

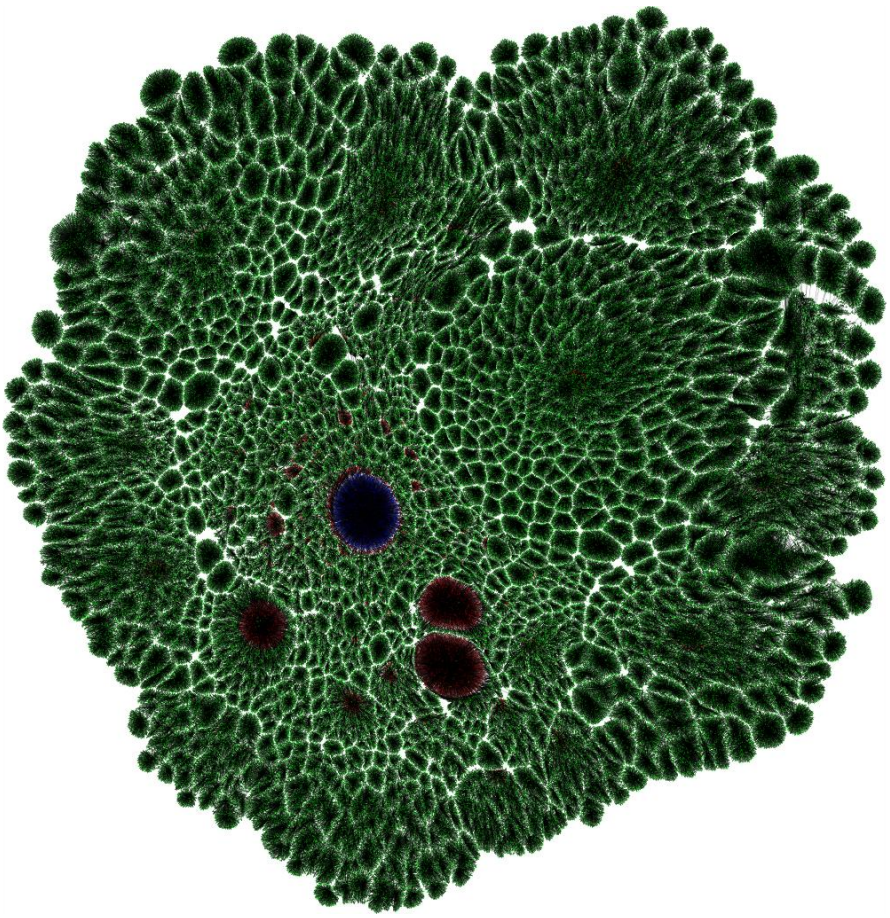


Abbildung 4.7e: Die ersten **vier** Ebenen (inklusive Wurzelknoten) der Hierarchie 1256259 (BFS Out-High).

Diese Abbildung zeigt die Hierarchie der Abbildung 4.7d in **vier** Ebenen.

In der vierten Ebene ergibt sich hierbei schon eine Summe von ca. 537.000 Knoten und Kanten.

Auch hier wurde die zweite Ebene wurde blau, die dritte Ebene rot und die vierte Ebene grün eingefärbt.

Man kann sehr gut sehen, dass die ausgehenden Kanten des High-Out-Degree Wurzelknotens auch in der vierten Ebene noch immer gut zu erkennen sind. Im Zentrum des Blauen Bereiches befindet sich der Wurzelknoten. Und auch in der 3. Ebene scheint es 3 signifikante weitere Knoten mit hohem Out-Degree zu geben.

Damit man sich besser vorstellen kann, wie sich die dritte Ebene in dieser Darstellung einbringt, betrachte man 4.7d. Man kann wunderbar erkennen wie sich die Ebene entwickelt hat.

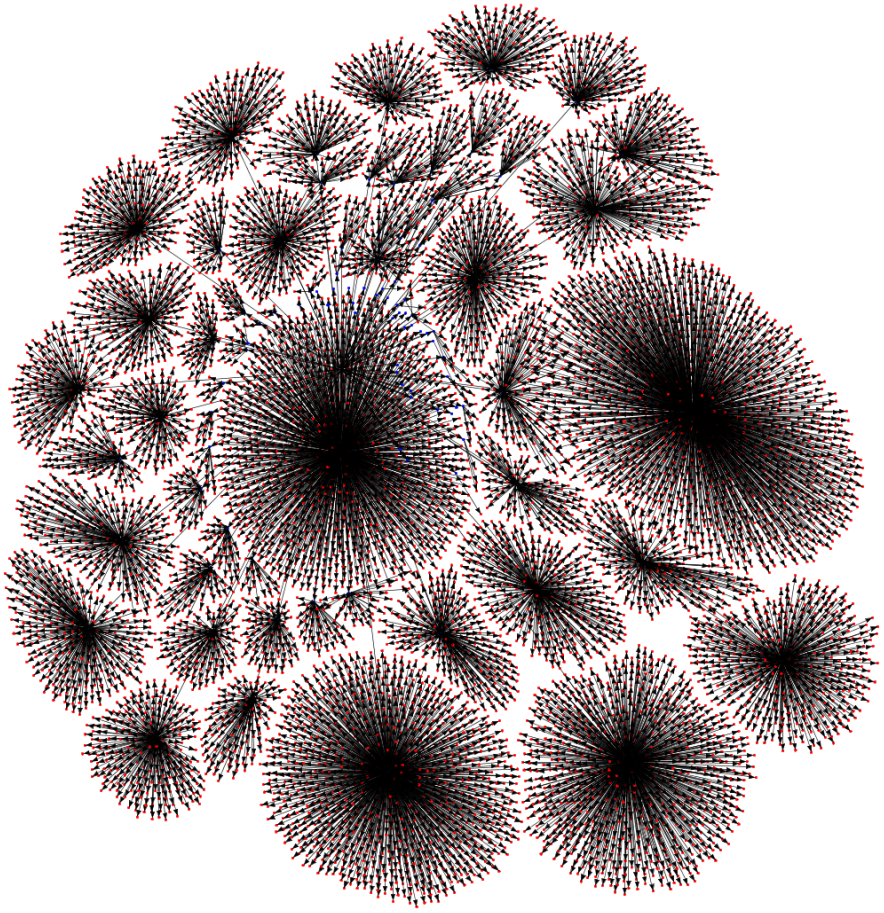


Abbildung 4.7f: Die ersten **drei** Ebenen (inklusive Wurzelknoten) der Hierarchie 3543305 (BFS Out-Low)

Die Abbildung 4.7f zeigt eine der am besten abgeschnittenen Hierarchien in der Kategorie BFS Out- Low. Es handelt sich um eine mit BFS erstellter Hierarchie, dessen Wurzelknoten einen niedrigen Out-Degree besitzt.

In der Abbildung 4.7f wurden um die 9000 Kanten und Knoten visualisiert.

Mit einer Success Rate von 95,57 % und einem Stretch von 3,82 schneidet diese Hierarchie besser ab als die beste der Kategorie BFS Out-High.

Der Wurzelknoten hat 58 eingehende und 100 ausgehende Kanten.

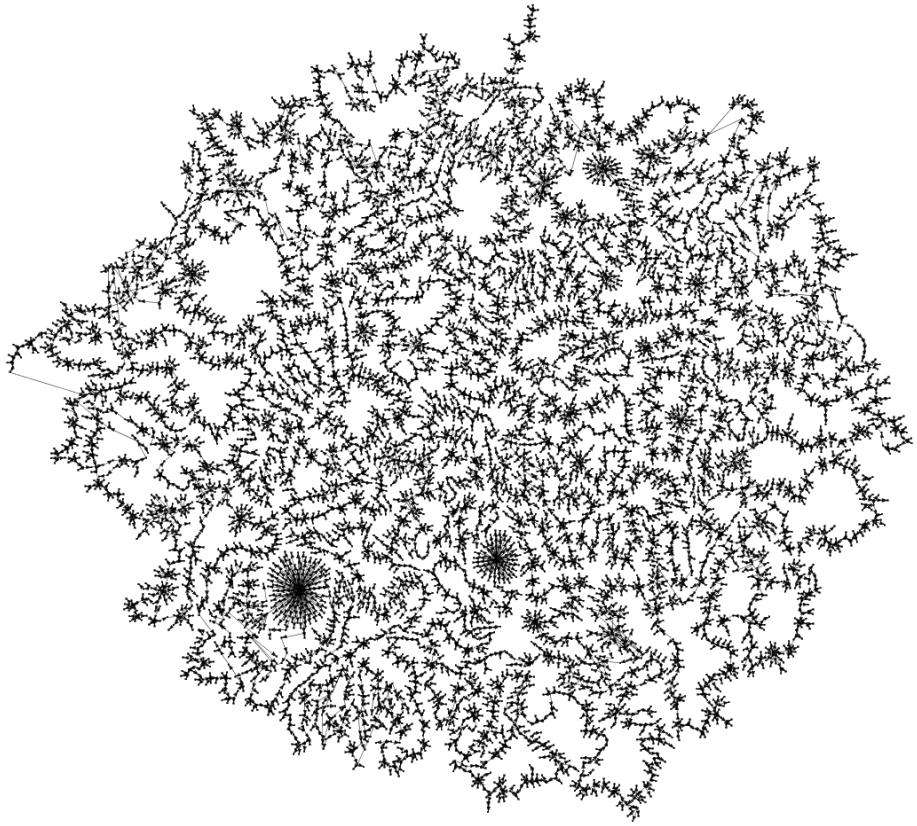


Abbildung 4.7g: Die ersten **3000** Ebenen (inklusive Wurzelknoten) der Hierarchie 3434750 (DFS In-High)

Nun kommen wir zu der ersten mit DFS erstellten Hierarchie.

Es handelt sich hier um die ersten 3000(!) Ebenen der Hierarchie.

Die Abbildung 4.7g zeigt eine mit DFS erstellte Hierarchie, dessen Wurzelknoten einen hohen In-Degree besitzt.

In dieser Abbildung wurden trotz 3000 Ebenen nur um die 9.000 Kanten und Knoten visualisiert.

Mit einer Success Rate von 9,86 % und einem Stretch von 10,59 ist diese Hierarchie eine mit den schlechtesten Ergebnissen.

Der Wurzelknoten hat 491777 eingehende und 1738 ausgehende Kanten. Die durchschnittliche Pfadlänge war 57,47, was jedoch durch die Abbruchbedingung von maximaler Pfadlänge 100 begrenzt wurde.

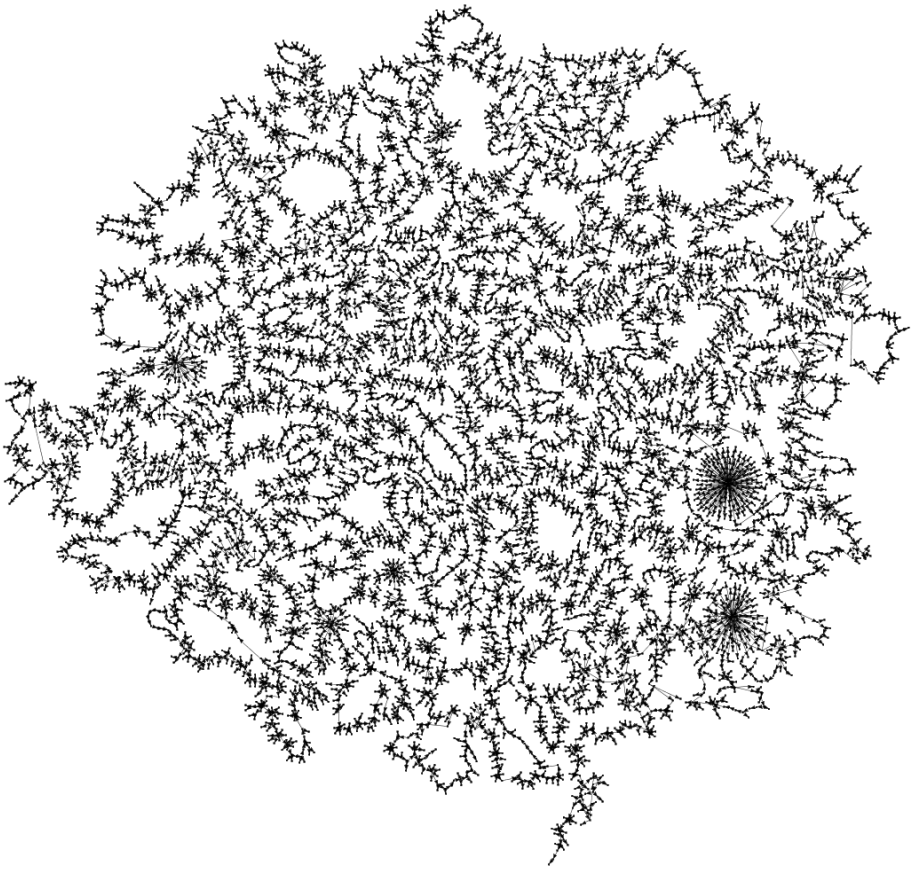


Abbildung 4.7h: Die ersten **3000** Ebenen (inklusive Wurzelknoten) der Hierarchie 140972 (DFS In-Low)

Die Abbildung 4.7h zeigt eine mit DFS erstellte Hierarchie, dessen Wurzelknoten einen niedrigen In-Degree besitzt.

Es ist im Prinzip ein langer Pfad, der als Hierarchie nicht geeignet ist. Die ersten 3000 Ebenen umfassen hierbei auch gerade einmal 9000 Kanten und Knoten.

Der Wurzelknoten dieser Hierarchie hat 100 eingehende und eine ausgehende Kante.

Die durchschnittliche Pfadlänge der 9,54% erfolgreichen Pfade, liegt nur bei 57,44.

Der durchschnittliche Stretch der erfolgreichen Pfade liegt bei 10,58.

Somit ist auch diese Hierarchie als Hintergrundinformation zur Navigation in großen Graphen nicht geeignet.

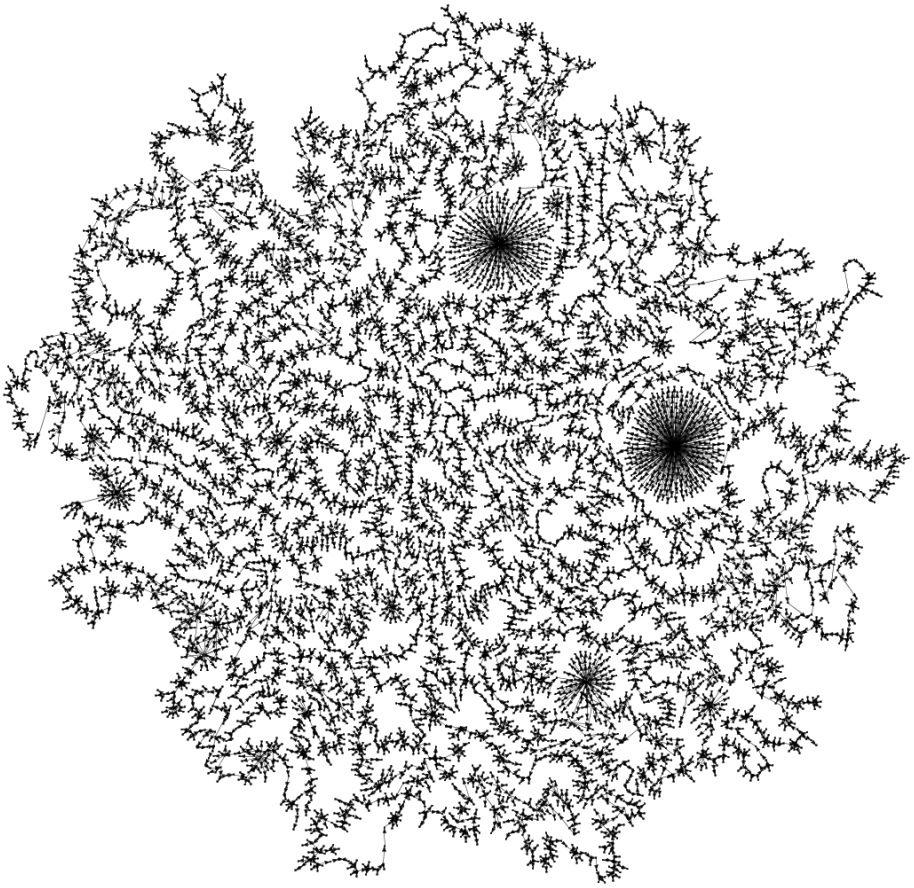


Abbildung 4.7i: Die ersten **3000** Ebenen (inklusive Wurzelknoten) der Hierarchie 3357308 (DFS Out-High)

Der nächste Versuch liegt in der Erstellung der Hierarchien mit DFS mit hohem Out-Degree Wurzelknoten.

Wie auch in den vorherigen beiden Visualisierungen, ergibt sich hierbei nur ein Pfad, der als Hierarchie nicht geeignet ist.

Einige Knoten scheinen doch einen höheren Degree zu haben, trotzdem sind in den ersten 3000 Ebenen davon nur eine Hand voll zu finden.

Es lässt sich daraus schließen, dass die auffälligen Knoten praktisch Sackgassen oder Endstationen waren, in denen der DFS Schritte zurück gehen musste.

Um einige Werte zu nennen, handelt es sich hierbei um einen Wurzelknoten mit 4 eingehenden und 4276 ausgehenden Knoten. Die durchschnittliche Pfadlänge bewegt sich mit 57,44 im selben Bereich wie bei den anderen mittels DFS extrahierten Hierarchien.

Die Erfolgsrate von 9,56% und der Stretch von 10,58 sagen aus, dass es sich ebenfalls um keine nützliche Hierarchie handelt.

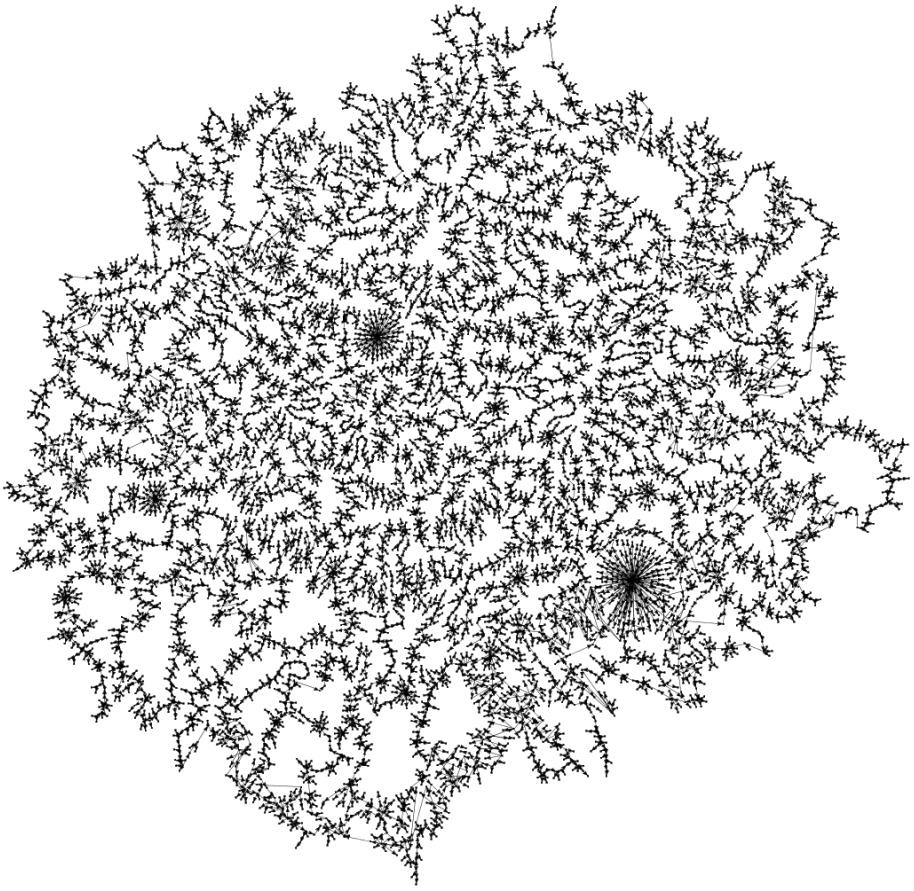


Abbildung 4.7j: Die ersten **3000** Ebenen (inklusive Wurzelknoten) der Hierarchie 3543305 (DFS Out-Low)

Die letzte Visualisierung einer mittels DFS extrahierten Hierarchie entstand mit einem Wurzelknoten der wenig ausgehende Kanten verzeichnet.

Wiederrum sind in den ersten 3000 Ebenen sehr wenige Kanten, im Vergleich zu mittels BFS erstellten Hierarchien ersichtlich. Es handelt sich ebenfalls um ca. 9000 Knoten und Kanten.

Um die Eckdaten zu nennen handelt es sich beim Wurzelknoten um einen Knoten mit 58 eingehenden und 100 ausgehenden Kanten. Die Erfolgsrate liegt bei durchschnittlich 9,82% und die durchschnittliche Pfadlänge beträgt 57,31.

Es handelt sich um denselben Wurzelknoten wie in Abbildung 4.7f, nur dass hierbei statt BFS mit DFS extrahiert wurde.

Die Abbildungen wurden mit Hilfe des Programmes GraphViz visualisiert, nachdem zuerst die jeweilige Hierarchie für GraphViz aufbereitet wurde. Anschließend musste die entstandene SVG Datei noch in ein kleineres brauchbares Bildformat konvertiert werden.

5 Ergebnisse und Folgerungen

5.1 Evaluierung der Hierarchien die mit Breitensuche erstellt wurden

Alle unter 5.1 gezeigten Ergebnisse und Tabellen beziehen sich auf Hierarchien die mit Hilfe der Breitensuche (BFS) erstellt wurden.

5.1.1 Evaluierung der 20 Hierarchien mit höchstem out-degree Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	Success Rate	Stretch
6078971	16	3824	28,80	83,03%	5,31
5426051	223	3838	30,59	82,95%	5,64
13931641	21	4022	23,21	93,30%	4,28
18554848	3519	4160	23,09	93,94%	4,25
100735	108	4206	25,75	88,57%	4,74
5483372	6	4247	30,44	77,95%	5,61
3357308	4	4276	32,11	76,73%	5,92
26421705	4	4398	26,11	86,44%	4,81
26421645	3	4462	26,11	86,44%	4,81
12089729	20	4518	29,72	78,18%	5,48
3284269	14	4580	30,88	76,57%	5,69
12577319	2	4620	22,23	92,50%	4,10
12562598	34	4702	22,18	93,74%	4,09
1147130	6	4769	24,16	85,21%	4,45
419390	15	5234	27,93	80,23%	5,15
274621	8	5310	25,46	90,09%	4,69
7772500	7591	5468	34,76	68,41%	6,41
21824714	2	5516	25,45	90,75%	4,69
6514471	3	5669	27,15	85,99%	5,00
24669262	2	8103	24,01	80,83%	4,42
		Mean:	27,01	84,59%	4,98

Es wird sich zeigen, dass die durchschnittliche Pfadlänge von 27 noch nicht optimal ist. Auch die Erfolgsrate von durchschnittlich 84,59 % kann noch verbessert werden. Die gleichzeitige Verringerung der Pfadlänge und Erhöhung der Erfolgsrate ist wünschenswert.

5.1.2 Evaluierung von 20 Hierarchien mit out-degree 100 Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	Success Rate	Stretch
5734	556	100	26,86	94,34%	4,95
1632845	251	100	24,40	95,36%	4,50
9248897	93	100	24,77	93,03%	4,56
32437	479	100	21,67	97,36%	3,99
1581567	38	100	21,86	97,24%	4,03
563367	500	100	23,88	95,27%	4,40
4468591	58	100	21,18	94,57%	3,90
3543305	58	100	20,75	95,57%	3,82
1098679	148	100	27,97	94,23%	5,15
1178726	87	100	26,03	94,47%	4,80
32627852	46	100	26,11	86,44%	4,81
20583037	111	100	26,71	94,11%	4,92
1440478	167	100	25,85	95,32%	4,76
1888433	86	100	24,73	95,79%	4,56
242821	120	100	24,23	94,78%	4,46
16767019	79	100	25,15	95,94%	4,64
2400759	114	100	25,31	95,74%	4,66
187849	222	100	23,04	96,67%	4,25
96429	254	100	23,13	95,40%	4,26
292169	82	100	22,51	96,47%	4,15
		Mean:	24,31	94,90%	4,48

In Abbildung 5.1a wurden die 40 Knotenpaare der 2 Testdurchläufe mit niedrigem und hohem Out-Degree in Bezug auf ihre durchschnittliche Pfadlänge aufgetragen. Man kann gut erkennen, dass die Hierarchien mit hohem Out-Degree Wurzelknoten eine deutlich höhere durchschnittliche Pfadlänge aufweisen, als die Hierarchien mit Out-Degree 100 Wurzelknoten. Auch das schlechteste Ergebnis zeigte sich in der Gruppe mit hohem Out-Degree, das beste in der Gruppe mit niedrigem Out-Degree.

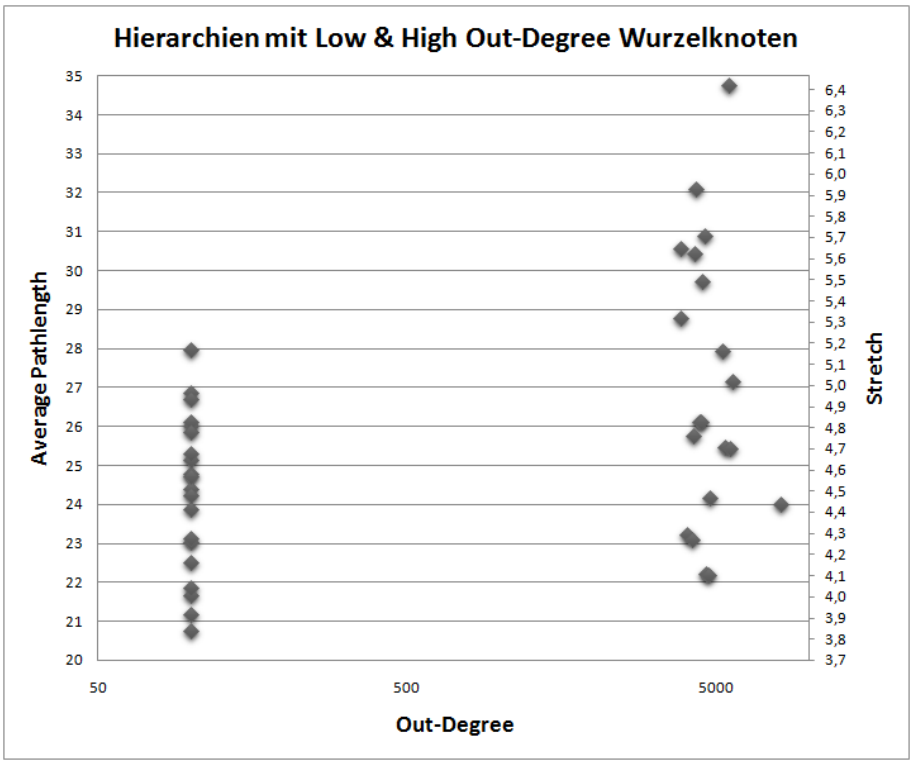


Abbildung 5.1a: Durchschnittliche Pfadlänge der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **ausgehenden** Kanten besitzen, als Wurzelknoten

Auch in der Abbildung 5.1b wird deutlich, dass die Erfolgsraten der Hierarchien mit hohem Out-Degree wesentlich schlechtere Werte aufweisen, als die Erfolgsrate der Hierarchien mit niedrigem Out-Degree. Vermutlich sind die Knoten mit hohen ausgehenden Kanten eine Art von Link Sammlungen im Informationsnetzwerk, welche jedoch selber nicht sehr oft referenziert werden.

Dadurch ergibt sich auch ein ungünstiges Verhältnis für in-degree und out-degree.

Im Paper „*Exploring the Differences and Similarities between Hierarchical Decentralized Search and Human Navigation in Information Networks*“ von Helic, Strohmaier, Singer und Tratter [15] wird ein Knoten anhand folgender Formel bewertet:

$$hs(n) = \frac{d_{in}(n)}{d_{out}(n)} \sqrt{d_{in}(n)}.$$

Die Qualität des Knotens wurde also direkt mit dem Verhältnis der ein und ausgehenden Kanten verknüpft, wobei die eigehenden Knoten eine höhere Rolle spielen sollten, was auch, bis auf einige Extremwerte, in meiner gesamten Evaluierung gut zutrifft.

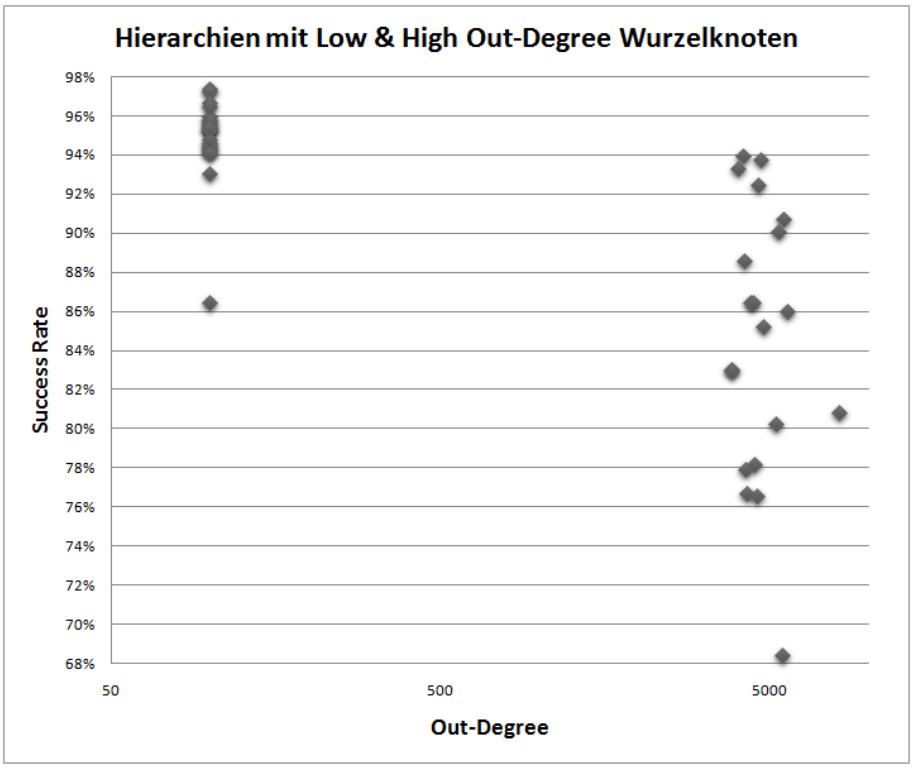


Abbildung 5.1b: Durchschnittliche Erfolgsrate der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **ausgehenden** Kanten besitzen, als Wurzelknoten

5.1.3 Evaluierung von 20 Hierarchien mit höchstem in-degree Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	Success Rate	Stretch
48361	708536	104	20,49	96,92%	3,78
3434750	491777	1738	22,20	96,95%	4,09
14919	311935	399	21,72	97,02%	4,00
30890	211490	613	22,56	96,77%	4,16
12653094	201344	383	22,30	95,51%	4,11
23410163	200926	364	25,40	96,10%	4,68
147101	188421	209	23,37	95,47%	4,31
31717	178840	1590	21,58	96,84%	3,98
2855554	175888	161	25,05	94,94%	4,62
5843419	172813	1838	21,75	95,96%	4,01
9316	156211	1808	22,79	95,52%	4,20
11867	151719	1209	22,00	96,32%	4,05
5042916	143025	821	21,50	96,78%	3,96
39736	134128	128	24,94	92,92%	4,60

11039790	119698	433	25,46	95,64%	4,69
47548	119598	388	23,02	96,30%	4,24
4689264	119282	1090	19,64	97,30%	3,62
53207	114909	434	22,91	94,94%	4,22
14532	109391	1356	21,49	96,40%	3,96
15573	109122	936	20,88	97,34%	3,85
		Mean:	22,55	96,10%	4,16

5.1.4 Evaluierung der 20 Hierarchien mit in-degree 100 Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	Success Rate	Stretch
29499150	100	1	26,11	86,44%	4,81
33196490	100	1	26,11	86,44%	4,81
453030	100	175	24,41	92,02%	4,50
6143564	100	91	23,99	95,68%	4,42
201481	100	181	23,87	95,15%	4,40
1591874	100	96	26,03	95,07%	4,80
5257371	100	103	25,03	95,86%	4,61
1649921	100	123	23,95	96,45%	4,41
77685	100	68	25,93	94,42%	4,78
2313158	100	88	27,73	93,97%	5,11
18613593	100	102	25,85	95,72%	4,76
1273586	100	80	24,31	94,50%	4,48
18388326	100	132	26,96	94,05%	4,97
1492637	100	109	21,67	96,49%	3,99
1275540	100	111	21,84	96,23%	4,02
1492663	100	111	21,80	96,25%	4,02
267090	100	1	25,22	94,89%	4,65
5605350	100	119	22,89	95,79%	4,22
9921187	100	607	29,30	89,30%	5,40
140972	100	1	27,66	92,12%	5,10
		Mean:	25,03	93,84%	4,61

In Abbildung 5.1c wurden die 40 Knotenpaare der zweiten Testreihe mit niedrigem und hohem In-Degree in Bezug auf ihre durchschnittliche Pfadlänge aufgetragen. Wie man gut erkennt, haben die Knotenpaare mit niedrigen In-Degree eine signifikant höhere durchschnittliche Pfadlänge. Der Durchschnitt, als auch das beste Ergebnis brachte eine Hierarchie mit einem hohen In-Degree.

Warum nun die Hierarchien dessen Wurzel einen niedrigen Out-Degree haben schlechter abschnitten, als die Hierarchien mit hohem Out-Degree, lässt sich anhand dieses Experimentes und der Tabelle gut schlussfolgern. Einige Knoten hatten im Test nur den Grad 1 bei ausgehenden Kanten. Dementsprechend ist der durchschnittliche Pfad immer mindestens um diesen Faktor größer.

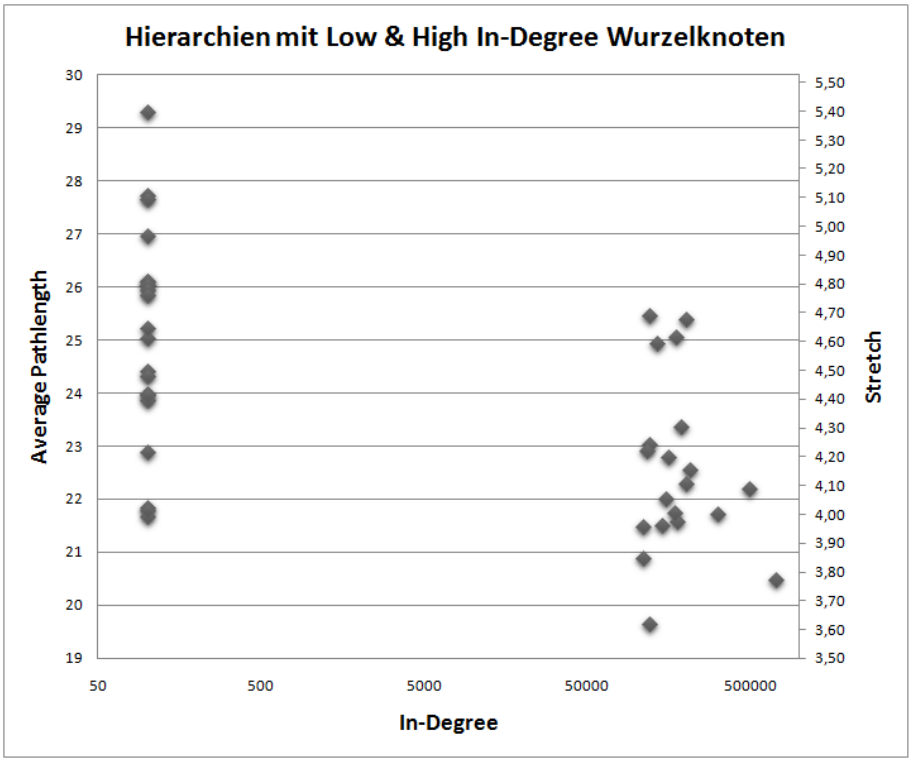


Abbildung 5.1c: Durchschnittliche Pfadlänge der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **eingehenden** Kanten besitzen, als Wurzelknoten

In der Abbildung 5.1d wurde die Erfolgsrate in Bezug auf die Anzahl der eingehenden Kanten der Wurzelknoten aufgetragen. Auch hier kann man erkennen, dass im Durchschnitt die entstandenen Hierarchien mit hohem Grad eingehender Kanten die höheren Erfolgsraten haben. Der schlechteste Wert kommt von einer Hierarchie aus der Testreihe mit niedrigem In-Degree, am besten hat wieder eine Hierarchie als Wurzelknoten einen Knoten mit hohem In-Degree hat, abschneiden.

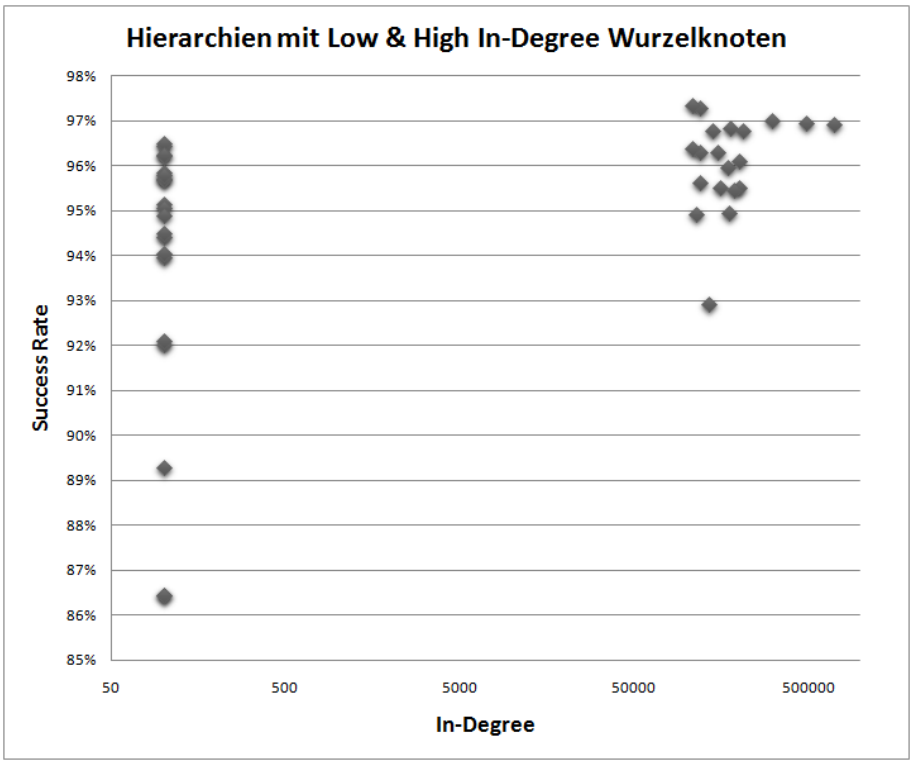


Abbildung 5.1d: Durchschnittliche Erfolgsrate der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **eingehenden** Kanten besitzen, als Wurzelknoten

Hierarchien die mit der Breitensuche erstellt wurden haben für Knoten mit hohem Grad an eingehenden Kanten eine höhere Erfolgsrate, weil die Wahrscheinlichkeit größer ist, dass man den Wurzelknoten, oder eine darunterliegende Ebene des Wurzelknotens, erreicht.

5.1.5 Ergebnisse der BFS Hierarchien im Vergleich

In der Abbildung 5.1e wurden alle Ergebnisse in Bezug auf den Grad der eingehenden Kanten mit der durchschnittlichen Pfadlänge aufgetragen.

Wie man sehr gut erkennt ergibt sich ein schöner linearer Trend. Je höher der Grad der eingehenden Kanten der Wurzelknoten ist, desto niedriger wurde im Mittel die durchschnittliche Pfadlänge.

Es besteht also kein Zweifel, dass einer der Einflussfaktoren für eine „gute“ Hierarchie der Grad der eingehenden Kanten des Wurzelknotens ist. Man sieht jedoch auch, dass es nicht der einzige Einflussfaktor zu sein scheint. Der Grad der ausgehenden Knoten spielt als weiterer Faktor eine wichtige Rolle.

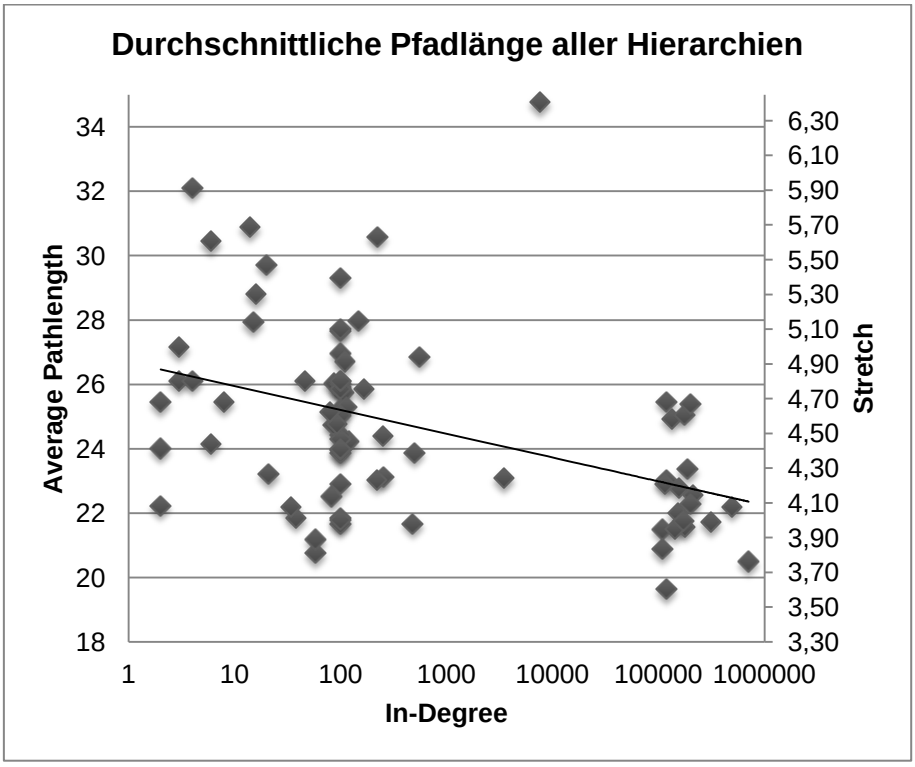


Abbildung 5.1e: Durchschnittliche Pfadlängen aller Hierarchien in Bezug auf den Grad der **eingehenden** Kanten des Wurzelknotens

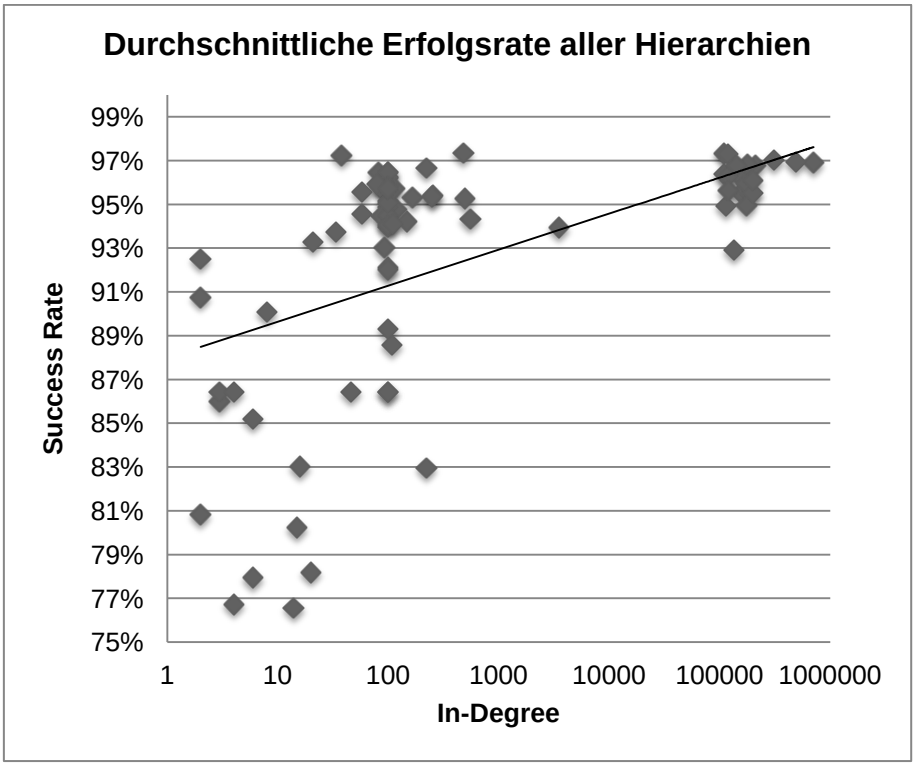


Abbildung 5.1f: Durchschnittliche Erfolgsrate aller Hierarchien in Bezug auf den Grad der **eingehenden** Kanten des Wurzelknotens

Für eine gute Erfolgsrate scheint der Faktor „in-degree“ sehr signifikant zu sein. Man kann sagen, je größer der in-degree ist, desto höher ist die Erfolgsrate im Mittel. Beim out-degree ist das nicht der Fall.

Wie man in Abbildung 5.1g erkennt scheint ein zu kleiner, aber auch ein zu großer Wert nicht optimal. Es muss anscheinend ein gutes Verhältnis zwischen out-degree und in-degree herrschen.

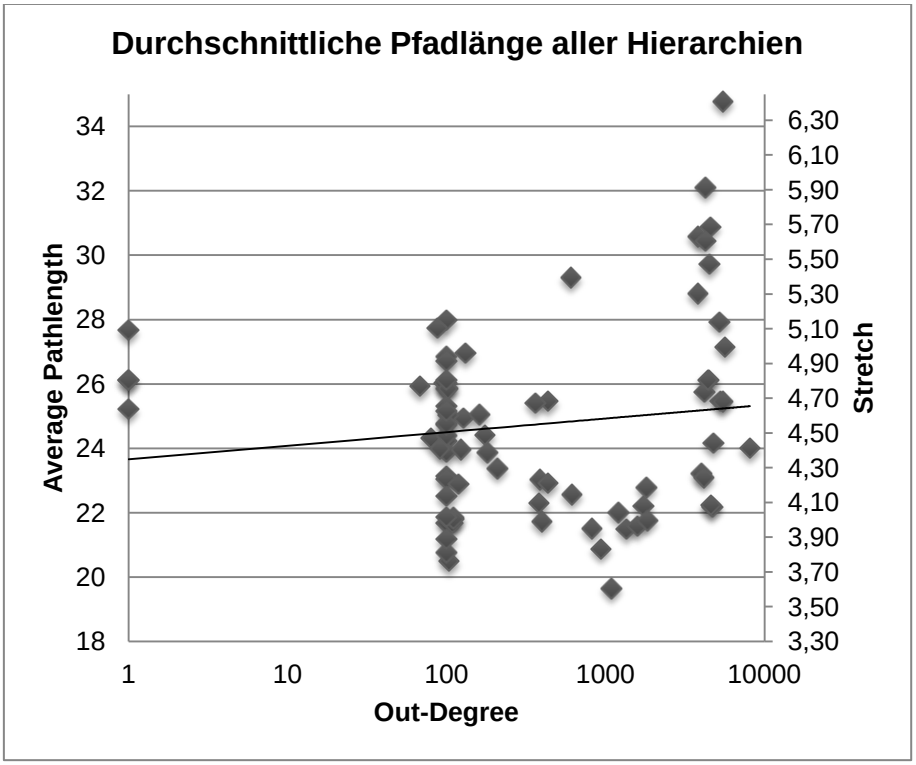


Abbildung 5.1g: Durchschnittliche Pfadlängen aller Hierarchien in Bezug auf den Grad der **ausgehenden** Kanten des Wurzelknotens

Die nächste Signifikanz zeigt sich in Abbildung 5.1h. Die Anzahl der ausgehenden Kanten eines Wurzelknotens hat zwar keine lineare Auswirkung auf die durchschnittliche Pfadlänge bei erfolgreichen Navigationen, aber sie hat eine direkte Auswirkung auf die Erfolgsrate.

Je höher die Anzahl der ausgehenden Knoten in unserem Experiment war, desto niedriger war im Durchschnitt die zu erwartende durchschnittliche Erfolgsrate. Wobei als Faktor wieder erwähnt werden muss, dass bei den Hierarchien im rechten unteren Viertel in der Abbildung 5.1h der in-degree extrem niedrig war.

In Informationsnetzwerken scheint es so zu sein, dass Seiten mit vielen ausgehenden Links meistens nicht sehr häufig von anderen Seiten erwähnt werden.

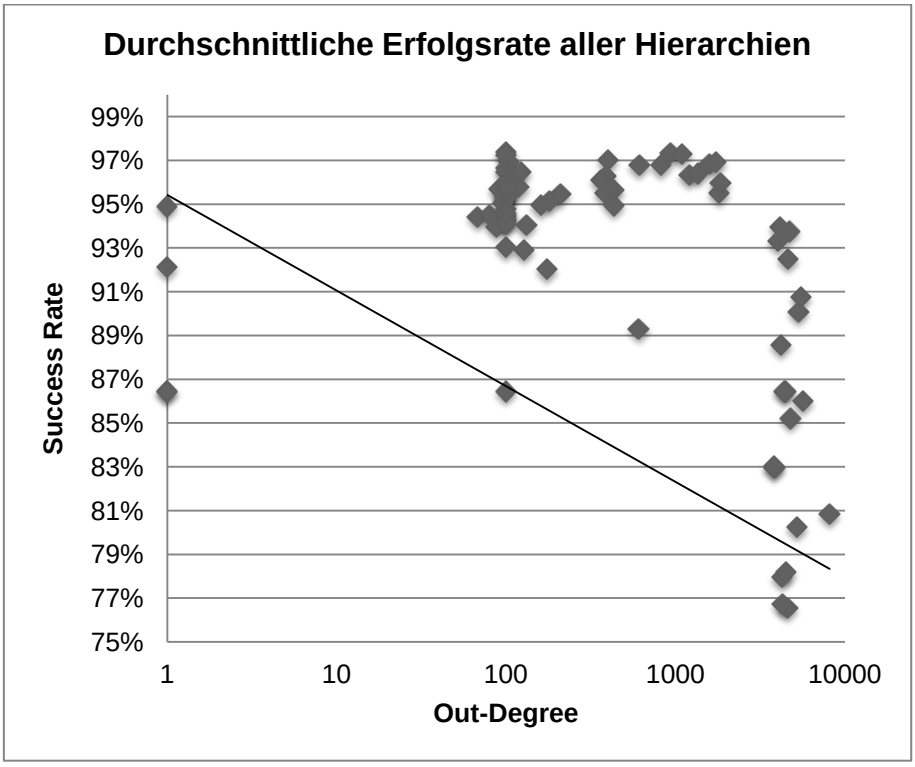


Abbildung 5.1h: Durchschnittliche Erfolgsrate aller Hierarchien in Bezug auf den Grad der **ausgehenden** Kanten des Wurzelknotens

5.2 Evaluierung der Hierarchien die mit der Tiefensuche erstellt wurden

Alle unter 5.2 gezeigten Ergebnisse und Tabellen beziehen sich auf Hierarchien die mit Hilfe der Tiefensuche (DFS) erstellt wurden.

Wie im Kapitel 3.1 beschrieben, wurde vermutet, dass Hierarchien die mit der Tiefensuche erstellt werden, sehr tief und schmal werden.

Die folgende Evaluierung zeigt, dass diese Vermutung richtig war.

Es wurden wieder 80 Hierarchien erzeugt die als Wurzelknoten, die Knoten mit höchstem in-degree, höchstem out-degree, in-degree 100 und out-degree 100 haben.

5.2.1 Evaluierung der 20 Hierarchien mit höchstem out-degree Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	SuccessRate	Stretch
24669262	2	8103	57,74	9,77%	10,64
6514471	3	5669	57,57	9,59%	10,61
21824714	2	5516	58,10	9,69%	10,71
7772500	7591	5468	57,89	9,72%	10,67
274621	8	5310	58,14	9,77%	10,71
419390	15	5234	58,32	9,64%	10,75
1147130	6	4769	57,67	9,73%	10,63
12562598	34	4702	58,10	9,71%	10,71
12577319	2	4620	58,05	9,61%	10,70
3284269	14	4580	58,06	9,78%	10,70
12089729	20	4518	57,45	9,53%	10,59
26421645	3	4462	57,51	9,56%	10,60
26421705	4	4398	58,06	9,70%	10,70
3357308	4	4276	57,44	9,56%	10,58
5483372	6	4247	57,69	9,85%	10,63
100735	108	4206	58,25	9,59%	10,73
18554848	3519	4160	57,78	9,64%	10,65
13931641	21	4022	58,09	9,84%	10,70
5426051	223	3838	58,35	9,77%	10,75
6078971	16	3824	57,62	9,66%	10,62
		Mean:	57,89	9,69%	10,67

5.2.2 Evaluierung von 20 Hierarchien mit out-degree 100 Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	SuccessRate	Stretch
1098679	148	100	57,52	9,49%	10,60
1178726	87	100	57,62	9,74%	10,62
1440478	167	100	58,12	9,70%	10,71
1581567	38	100	58,22	9,60%	10,73
1632845	251	100	58,12	9,77%	10,71
16767019	79	100	57,42	9,58%	10,58
187849	222	100	58,33	9,76%	10,75
1888433	86	100	57,74	9,56%	10,64
20583037	111	100	57,50	9,59%	10,59
2400759	114	100	57,45	9,75%	10,59
242821	120	100	57,63	9,64%	10,62
292169	82	100	57,82	9,52%	10,65
32437	479	100	58,05	9,73%	10,70
32627852	46	100	57,87	9,73%	10,66
3543305	58	100	57,31	9,82%	10,56
4468591	58	100	57,74	9,76%	10,64
563367	500	100	58,68	9,61%	10,81
5734	556	100	57,92	9,73%	10,67
9248897	93	100	58,19	9,68%	10,72
96429	254	100	57,80	9,55%	10,65
		Mean:	57,85	9,67%	10,66

Als nicht erfolgreich wurden auch alle Pfade deren Länge einen Wert von mehr als 100 ergaben, gewertet. Die Simulation wurde in einem solchen Fall abgebrochen, das wurde auch in der Reihe der Experimente mit der Breitensuche so gehandhabt.

Die Tabelle zeigt eindeutige Werte. Die Vermutung, dass Hierarchien die mit der Tiefensuche erstellt werden, gerade bei stark vernetzten Graphen, sehr tiefe Äste und eine sehr geringe Breiten besitzen und sich dadurch auch die Pfadlänge entsprechend verlängert, ist eingetroffen.

Die durchschnittlich entstandene Hierarchie hatte 1,9 Millionen Ebenen. Der resultierende Baum, ist dementsprechend extrem tief und schmal.

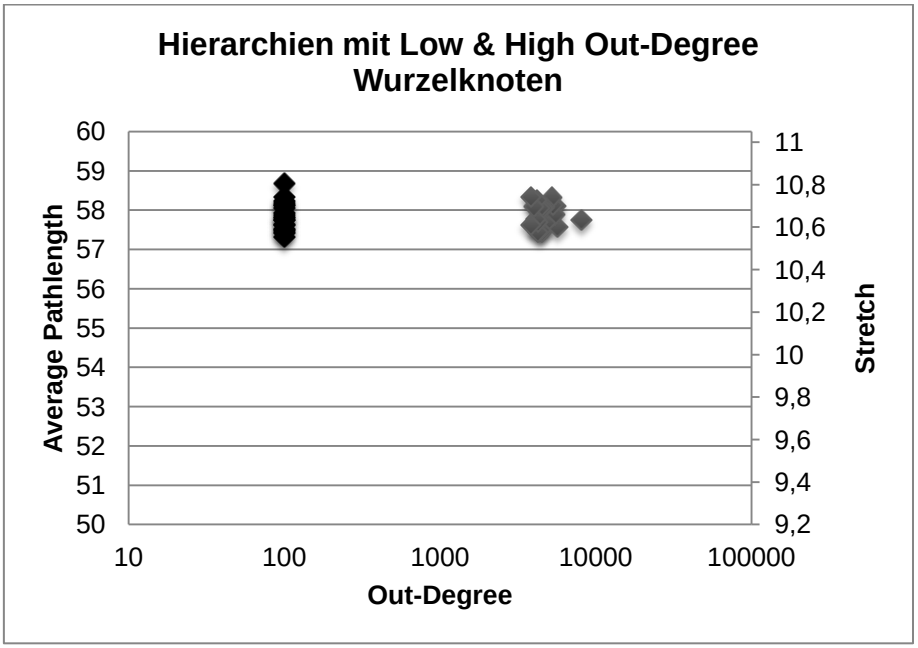


Abbildung 5.2a: Durchschnittliche Pfadlänge der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **ausgehenden** Kanten besitzen, als Wurzelknoten

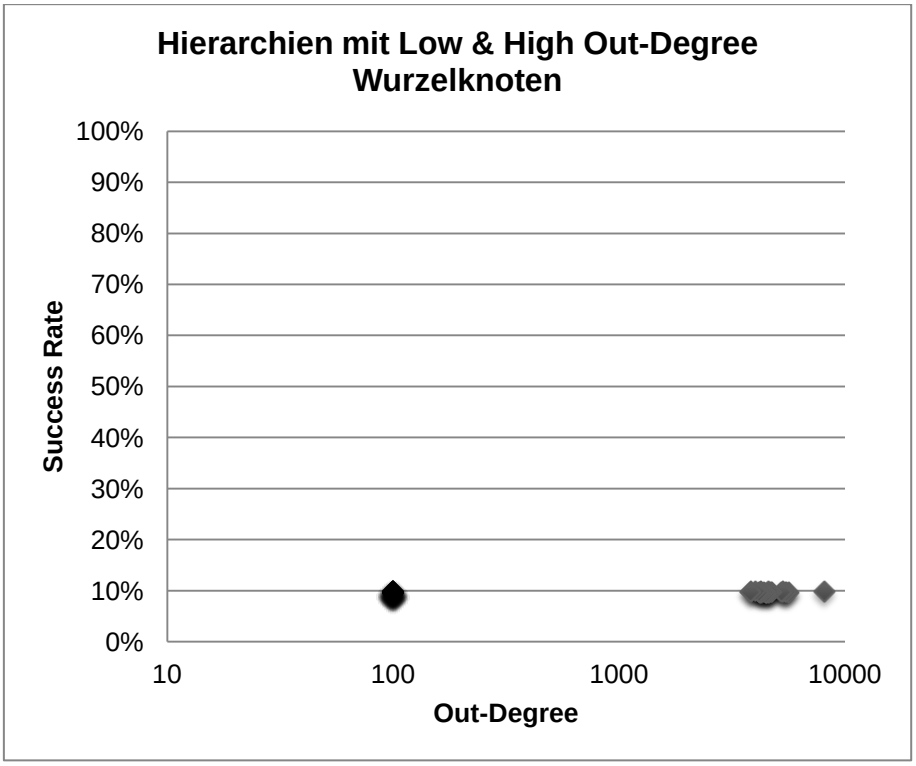


Abbildung 5.2b: Durchschnittliche Erfolgsrate der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **ausgehenden** Kanten besitzen, als Wurzelknoten

5.2.3 Evaluierung von 20 Hierarchien mit höchstem in-degree Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	SuccessRate	Stretch
48361	708536	104	57,94	9,65%	10,68
3434750	491777	1738	57,47	9,86%	10,59
14919	311935	399	58,18	9,64%	10,72
30890	211490	613	57,77	9,91%	10,64
12653094	201344	383	58,32	9,60%	10,75
23410163	200926	364	58,09	9,73%	10,70
147101	188421	209	58,36	9,60%	10,75
31717	178840	1590	57,92	9,69%	10,67
2855554	175888	161	57,46	9,70%	10,59
5843419	172813	1838	57,84	9,69%	10,66
9316	156211	1808	57,88	9,71%	10,67
11867	151719	1209	58,12	9,75%	10,71
5042916	143025	821	58,34	9,63%	10,75
39736	134128	128	57,65	9,64%	10,62
11039790	119698	433	58,00	9,94%	10,69
47548	119598	388	57,93	9,82%	10,68
4689264	119282	1090	58,00	9,71%	10,69
53207	114909	434	58,37	9,62%	10,76
14532	109391	1356	58,06	10,10%	10,70
15573	109122	936	57,84	9,85%	10,66
		Mean:	57,98	9,74%	10,68

Die Anforderungen die wir an Hierarchie haben, kann mit DFS erzeugten nicht entsprochen werden.

Wie man in der Tabelle sieht, liegt die durchschnittliche Erfolgsrate bei weniger als 10 %. Das bedeutet, dass nur jedes 10. Knotenpaar zueinander gefunden hat. Die anderen 90 % besitzen entweder eine extrem große Pfadlänge, mit einem Wert von mindestens 100, oder es wird mit der DFS erzeugten Hierarchien kein Pfad gefunden.

Die erhofften Ergebnisse von kurzen Wegen in Verbindung mit hohen Erfolgsraten werden mit der Tiefensuche bei weitem verfehlt.

Die gefundenen, als erfolgreich angesehenen Pfade haben einen Stretch von mehr als 10, was bedeutet, dass sie 10-mal länger sind als die optimale Lösung.

5.2.4 Evaluierung der 20 Hierarchien mit in-degree 100 Knoten als Wurzel

ID	InDegree	OutDegree	Average Pathlength	SuccessRate	Stretch
1273586	100	80	58,26	9,57%	10,73
1275540	100	111	58,16	9,75%	10,72
140972	100	1	57,44	9,54%	10,58
1492637	100	109	57,94	9,70%	10,68
1492663	100	111	58,06	9,76%	10,70
1591874	100	96	57,53	9,50%	10,60
1649921	100	123	57,85	9,65%	10,66
18388326	100	132	57,82	9,62%	10,65
18613593	100	102	57,77	9,70%	10,65
201481	100	181	58,44	9,62%	10,77
2313158	100	88	57,66	9,59%	10,63
267090	100	1	58,38	9,58%	10,76
29499150	100	1	57,83	9,77%	10,66
33196490	100	1	57,62	9,76%	10,62
453030	100	175	57,97	9,57%	10,68
5257371	100	103	57,44	9,62%	10,58
5605350	100	119	58,01	9,72%	10,69
6143564	100	91	57,52	9,64%	10,60
77685	100	68	58,10	9,64%	10,71
9921187	100	607	58,15	9,60%	10,72
		Mean:	57,90	9,64%	10,67

Dieses Ergebnis verschlechtert sich unter der Berücksichtigung, dass nur Pfade mit Pfadlängen unter 100 als erfolgreich angesehen wurden, nochmals deutlich in der Interpretation.

Die Ergebnisse zeigen in der folgenden Abbildung 5.2c konstant schlechte Werte.

Durch die starke Verzweigung des Graphen entarteten die Hierarchien die mit Hilfe der Tiefensuche erzeugt wurden. Es entstanden extrem lange Äste mit einer maximalen Tiefe von ungefähr 1,9 Millionen Ebenen.

Man kann sich gut vorstellen, dass der Tiefensuchalgorithmus in einem stark zusammenhängenden Graphen lange einen Knoten findet, der ihn wieder eine Ebene tiefer gehen lässt. Für eine Navigation mit kurzen Wegen eher kontraproduktiv.

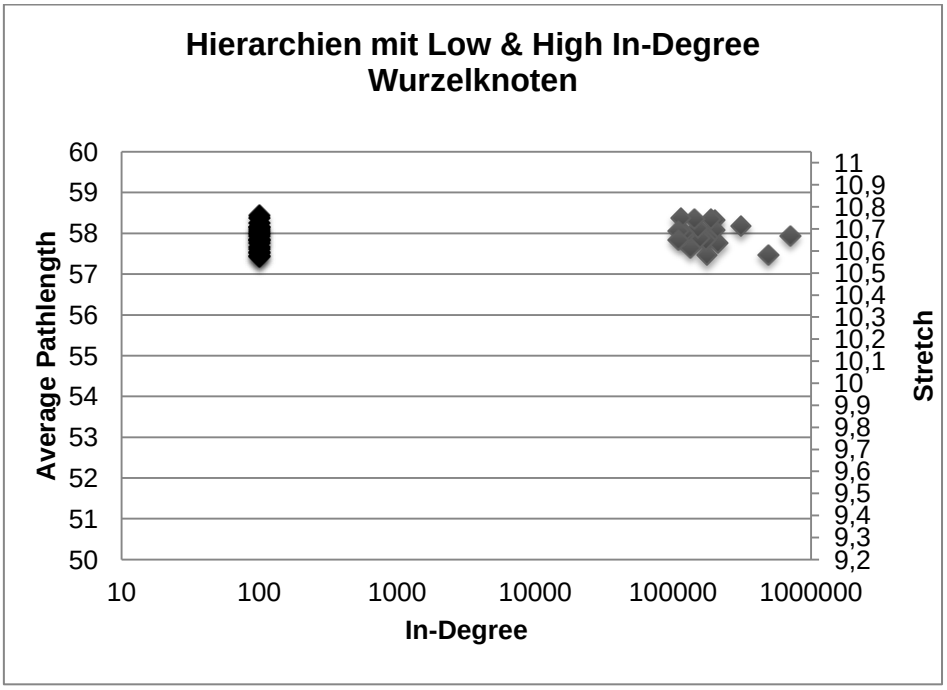


Abbildung 5.2c: Durchschnittliche Pfadlänge der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **eingehenden** Kanten besitzen, als Wurzelknoten

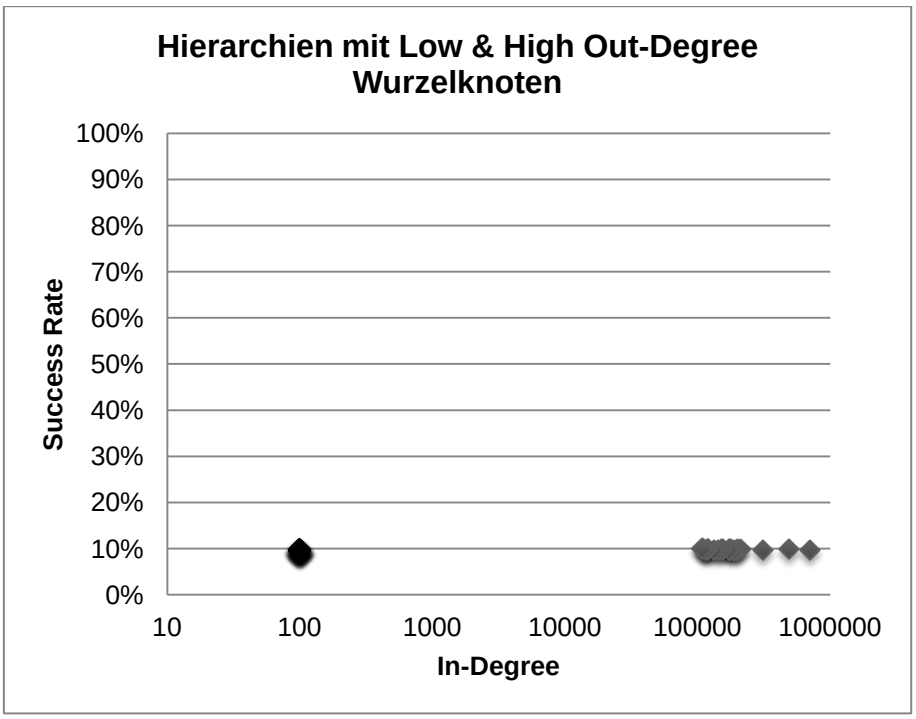


Abbildung 5.2d: Erfolgsrate der Hierarchien mit Knoten, die einen niedrigen und hohen Grad an **eingehenden** Kanten besitzen, als Wurzelknoten

5.2.5 Ergebnisse der DFS Hierarchien im Vergleich

In der Abbildung 5.2e wurden alle Ergebnisse in Bezug auf den Grad der eingehenden Kanten mit der durchschnittlichen Pfadlänge aufgetragen.

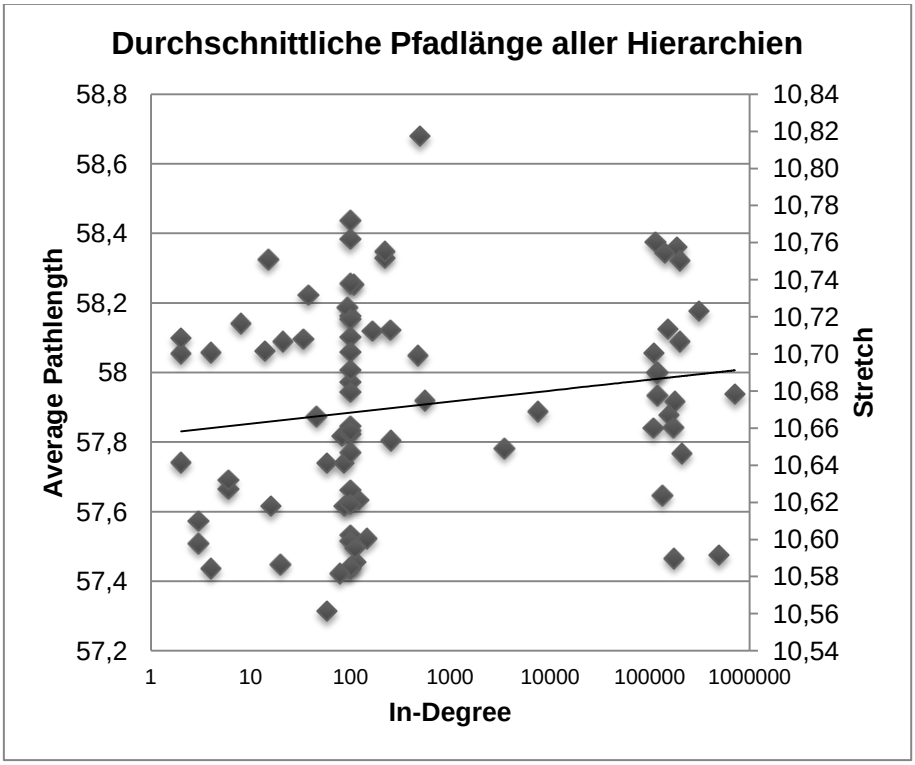


Abbildung 5.2e: Durchschnittliche Pfadlänge aller Hierarchien in Bezug auf den Grad der **eingehenden** Kanten des Wurzelknotens

Bei der Evaluierung aller mit Tiefensuche erstellten Hierarchien und dem Abbruchkriterium von 100 Schritten, ist die durchschnittliche Pfadlänge, wie man in Abbildung 5.2e gut erkennt relativ konstant. Mit einer Pfadlänge von ungefähr 58 und einer leichten Steigung der Pfadlänge mit zunehmenden eingehenden Knoten, ist im Gesamtbild die Tiefensuche zur Erstellung von Hierarchien nicht geeignet.

Der durchschnittliche Stretch von mindestens 10,5 ist sehr weit von effizientem Navigieren entfernt.

Die durchschnittliche Erfolgsrate von 10% ist für eine Navigation ungeeignet. Interessant ist jedoch, dass auch hier bei steigendem eingehenden Knotengrad die Erfolgsrate zumindest im Trend leicht gestiegen ist. Die Steigung ist mit 0,1% im gesamten Bereich aber kaum signifikant.

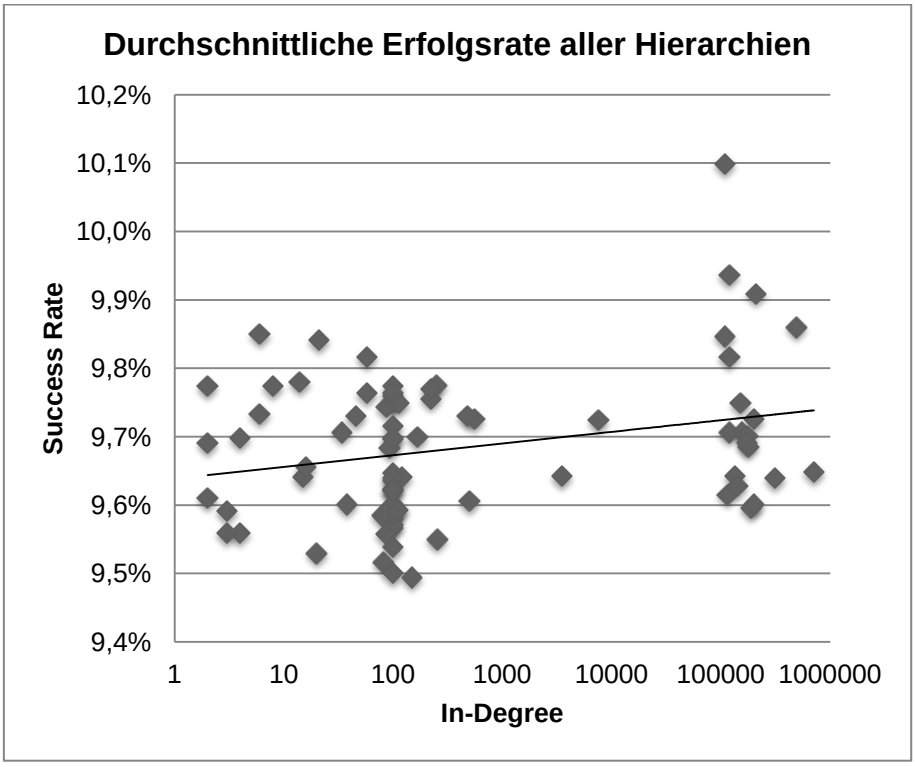


Abbildung 5.2f: Durchschnittliche Erfolgsrate aller Hierarchien in Bezug auf den Grad der **eingehenden** Kanten des Wurzelknotens

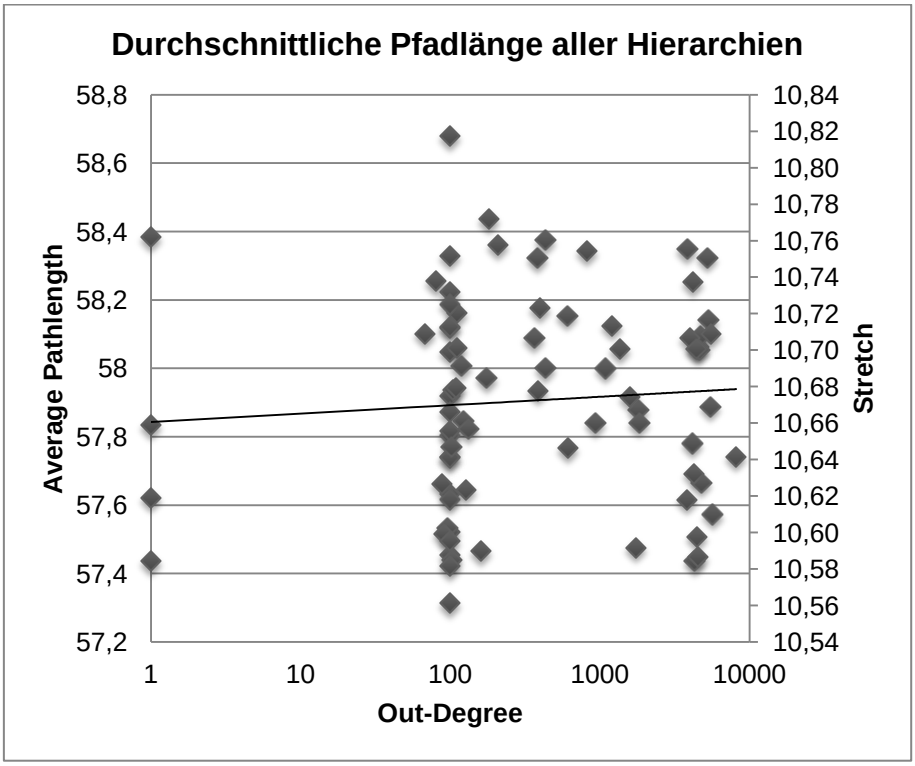


Abbildung 5.2g: Durchschnittliche Pfadlänge aller Hierarchien in Bezug auf den Grad der **ausgehenden** Kanten des Wurzelknotens

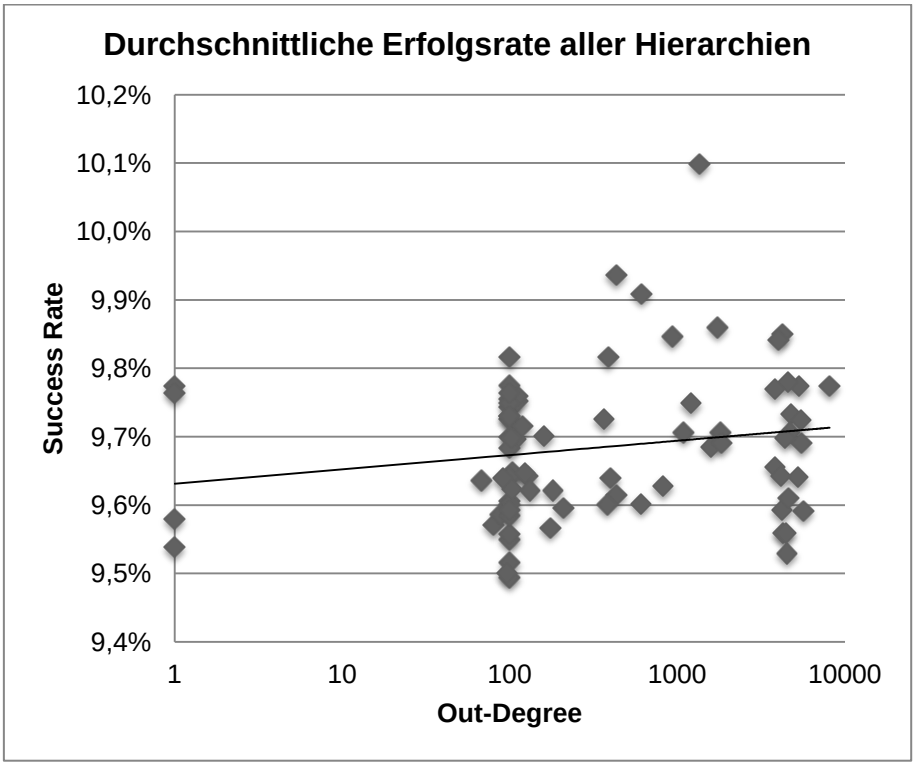


Abbildung 5.2h: Durchschnittliche Erfolgsrate aller Hierarchien in Bezug auf den Grad der **ausgehenden** Kanten des Wurzelknotens

Wie auch in den Abbildungen 5.2e bis 5.2h gut erkennbar ist, verhalten sich die Hierarchien, egal ob ihre Wurzel einen hohen Grad an eingehenden Kanten oder einen hohen Grad an ausgehenden Kanten besitzt, tendenziell gleich.

Beide steigen im Durchschnitt minimal in der durchschnittlichen Erfolgsrate bei steigendem Grad der Wurzelknoten.

6 Schlussfolgerung

Warum die Evaluierung gezeigt hat, dass Hierarchien mit Wurzelknoten die eine hohe Anzahl von eingehenden Kanten besitzen und zusätzlich mit der Breitensuche erzeugt wurden, die besten Ergebnisse liefern kann sich folgend erklären.

Knoten mit hohen eingehenden Wurzelknoten werden im Graph leichter gefunden als Knoten mit niedrigen eingehenden Wurzelknoten.

Je höher die Anzahl der eingehende Kanten des Knoten ist, desto höher ist auch die Wahrscheinlichkeit ihn einmal als Kindknoten in einem beliebigen Pfad zu finden.

Das bedeutet also, dass Knoten, die sich in der Hierarchie unter diesem Knoten befinden, dann auch erreicht werden, wenn der Wurzelknoten gefunden wird.

Davon ist nun auch direkt die Erfolgsrate abhängig. Wählt man einen Knoten als Wurzelknoten mit angenommen nur einer eingehenden Kante, so wird die daraus entstandene Hierarchie höchstwahrscheinlich nicht sehr zentral liegen und auch nicht effektiv werden. Das spiegelt sich dann in der durchschnittlichen Pfadlänge nieder. Es kann natürlich sein, dass der Graph sehr stark verzweigt ist, dann wird es möglich sein, eine gute Erfolgsrate mit einem Knoten, der relativ wenig eingehende Kanten hat, zu erreichen. Eine Balance mit den ausgehenden Kanten ist auch sehr wichtig. Der durchschnittliche Pfad würde darunter sehr leiden, wenn in einer Hierarchie, mit einem Knoten der nur eine ausgehende Kante hat, als Wurzelknoten immer über mindestens diese eine Kante navigiert werden und somit der durchschnittliche Pfad auch wiederrum im besten Fall mindestens um den Faktor 1 länger sein.

Will man dem Navigator eine gute Hintergrundinformation in Form einer Hierarchie mitgeben und wird diese Hierarchie auf Basis der Breitensuche aufgebaut, so lautet die Empfehlung Knoten als Wurzelknoten mit hohem in-degree zu wählen.

Glossar

BFS	breadth-first search, Breitensuche
BSD	Berkeley Software Distribution
DFS	depth-first search, Tiefensuche
High Out	Hoher Ausgangsgrad
IN	In-Degree, Eingangsgrad
Low In	Niedriger Eingangsgrad
MUN	Modeling User Navigation
OUT	Out-Degree, Ausgangsgrad
SNAP	Stanford Network Analysis Platform
SCC	strongly-connected component

Literaturverzeichnis

[1] D. Easley und J. Kleinberg, Networks, Crowds, and Markets: *Reasoning About a Highly Connected World*, Cambridge University Press, 2010.

[2] R. Diestel, *Graphentheorie*, Berlin: Springer Verlag, 2010.

[3] Peter Hedstrom. *Contagious collectivities: On the spatial diffusion of Swedish trade unions*. American Journal of Sociology, 99:1157–1179, 1994.

[4] F. Heart, A. McKenzie, J. McQuillian, and D. Walden. *ARPANET Completion Report*. Bolt, Beranek and Newman, 1978.

[5] Wayne Zachary. *An information flow model for conflict and fission in small groups*. Journal of Anthropological Research, 33(4):452–473, 1977.

[6] Lada Adamic and Natalie Glance. *The political blogosphere and the 2004 U.S. election: Divided they blog*. In Proceedings of the 3rd International Workshop on Link Discovery, pages 36–43, 2005.

[7] M. Eder. *Entwicklung eines Frameworks zur Navigationssimulation*. Technische Universität Graz, 2013.

[8] Denis Helic, Markus Strohmaier, Weronika Wojcik. *Navigational Evaluation of Breadth First Search Spanning Trees*. Graz University of Technology, 2013.

[9] Sven Oliver Krumke, Hartmut Noltemeier: *Graphentheoretische Konzepte und Algorithmen*. Springer Vieweg, 3. Auflage, 2012.

[10] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew S. Tomkins, Eli Upfal. *The Web as a graph*, 2000.

[11] R. Albert, H. Jeong, and A.-L. Barabasi. *Diameter of the World Wide Web*, 1999.

[12] K. Bharat and A. Broder. *A technique for measuring the relative size and overlap of public Web search engines*. Proc. 7th WWW Conf., 1998.

[13] A.Z. Broder, S.R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. *Graph structure in the web: experiments and models*. Proc. 9th WWW Conf., 1999.

[14] Joseph Kruskal: *On the shortest spanning subtree and the traveling salesman problem*. Proceedings of the American Mathematical Society, 1956.

[15] C. Trattner, P. Singer, D. Helic, and M. Strohmaier. *Exploring the differences and similarities between hierarchical decentralized search and human navigation in information networks.* Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies, 2012.

[16] J. Kleinberg, *The small-world phenomenon: an algorithm perspective* in Proceedings of the thirty-second annual ACM symposium on Theory of computing, 2000.

[17] J. M. Kleinberg, *Navigation in a small world*, 2000.

[18] S. Milgram, *The small world problem.* Psychology Today, 1967.

[19] D. Helic, M. Strohmaier, C. Trattner, M. Muhr, and K. Lerman, *Pragmatic evaluation of folksonomies.* Proceedings of the 20th international conference on World Wide Web, ser. WWW '11. New York, NY, USA: ACM, 2011.

[20] M. Strohmaier, D. Helic, D. Benz, C. K¨orner, and R. Kern. *Evaluation of folksonomy induction algorithms*. Sep. 2012.

[21] R. Hassin and A. Tamir. *On the minimum diameter spanning tree problem.* Information Processing Letters, vol. 53, 1995.

[22] A. Clauset, C. Moore, and M. E. J. Newman. *Hierarchical structure and the prediction of missing links in networks*. Nature, 2008.

[23] P. Heymann and H. Garcia-Molina. *Collaborative creation of communal hierarchical taxonomies in social tagging systems*. April 2006.

[24] P. Mika. *Ontologies are us: A unified model of social networks and semantics*. Web Semantics: Science, Services and Agents on the World Wide Web, March 2007.

[25] L. Muchnik, R. Itzhack, S. Solomon, and Y. Louzoun. *Self-emergence of knowledge trees: Extraction of the wikipedia hierarchies*. Jul 2007.

[26] R. W. White and S. M. Drucker. *Investigating behavioral variability in web search*. WWW, 2007.

[27] James Abello, Adam L. Buchsbaum, and JefferyWestbrook. *A functional approach to external graph algorithms*. In Proc. 6th European Symposium on Algorithms, 1998.

[28] Lada A. Adamic and Eytan Adar. *How to search a social network. Social Networks*, 2005.

[29] Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. *Search in power-law networks*. Physical Review E, 2001.

[30] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[31] David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. *Approximating betweenness centrality*. In Proc. 5th Workshop on Algorithms and Models for the Web Graph, 2007.

[32] David A. Bader and Kamesh Madduri. *SNAP: Small-world network analysis and partitioning: An open-source parallel graph framework for the exploration of large-scale networks*. In Proc. 22nd IEEE International Symposium on Parallel and Distributed Processing, 2008.

[33] J. Voss. *Measuring Wikipedia*. Proceedings of the 10th International Conference of the International Society for Scientometrics and Informetrics, Stockholm, 2005.

[34] *Stanford Large Network Dataset Collection*, [Online]. http://snap.stanford.edu/data/index.html. [Zugriff am 10. Mai 2013]

[35] Thomas H. Cormen, Charles Leiserson, Ronald L. Rivest, Clifford Stein: *Introduction to Algorithms*. MIT Press, 2nd edition 2001.