Heuristische Berechnung von Feedback-Arc-Sets mittels Zentralitätsmaßen

Bachelorarbeit
von

Christoph König

An der Fakultät für Informatik und Mathematik
Lehrstuhl für Informatik mit Schwerpunkt Theoretische Informatik

Gutachter: Prof. Dr. Ignaz Rutter

Bearbeitungszeit: 26. Januar 2023 – 29. April 2023

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig und ohne unzulässige Hilfe verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt sowie die wörtlich uns sinngemäß übernommenen Passagen aus anderen Werken kenntlich gemacht habe. Die Arbeit ist weder von mir noch von einer anderen Person an der Universität Passau oder an einer anderen Hochschule zur Erlangung eines akademischen Grades bereits eingereicht worden.

Passau, 28. April 2023

Zusammenfassung

Sei $G = (V, E)$ ein gerichteter Graph. Benötigt man einen azyklischen Graphen, müssen einige Kanten aus diesem entfernt werden. Welche Kanten man wählt, um möglichst wenige entfernen zu müssen, ist ein NP-schweres Problem. Deshalb werden bekannte Heuristiken vorgestellt und erörtert. Es wurde ein neuer zentralitätsmaßbasierter Algorithmus genauer untersucht, verschiedene Zentralitätsmaße miteinander verglichen und Kriterien und Parameter bei PageRank getestet, um diesen Ansatz weiter zu verbessern und besser in bekannte Heuristiken einordnen zu können. Auch wurden *Edge PageRank* eingeführt, wodurch die Laufzeit erheblich verbessert werden konnte. Aber auch bekannte Heurisiken wie GreedyFAS und SortFAS konnten für Graphen mit kleinem Ausgangsgrad verbessert werden.

Inhaltsverzeichnis

1. Einleitung

„Es wird niemals so viel gelogen wie vor der Wahl, während des Krieges und nach der Wahl.“ Dieses Zitat von Bismarck zeigt, dass die Verbreitung von Falschinformationen kein neues Phänomen ist, das man erst seit dem Krieg in der Ukraine oder den Präsidentschaftswahlen in den USA kennt. Nicht nur soziale Medien, sondern auch die Möglichkeit, falsche Behauptungen mit von einer KI generierten falschen Bilden als „Beweise“ zu versehen, machen deren Verbreitung kinderleicht. Der Schaden ist jedoch oft riesig.

Es gibt Algorithmen, wie der von Simpson et al. [SST16a], die Falschinformationen in sozialen Netzwerken erkennen und entfernen können, indem sie diese als gerichtete Graphen auffassen. Dazu benötigt er jedoch einen azyklischen Graphen, aus dem zuvor möglichst wenig Kanten entfernt wurden. Eine möglichst kleine Teilmenge der Kanten E eines gerichteten Graphen $G = (V, E)$ bezeichnet man als *Feedback Arc Set* (FAS), wenn nach Entfernen dieser Kanten der Graph keine gerichteten Kreise mehr enthält [SST16b].

Auch bei der hierarchischen Darstellung gerichteter Graphen [STT81] oder bei der Analyse sozialer Netzwerke durch Label Propagation [GGKF14], wird eine FAS benötigt.

Die Berechnung der FAS eines Graphen ist aber ein NP-schweres Problem [Kar10].

Deshalb wurden mehrere Heuristiken entwickelt, die eine akzeptable Annäherung generieren. Simpson et al. haben hierzu die gängigen Heuristiken miteinander verglichen, wobei sich GreedyFAS und SortFAS* als die mit der kleinsten FAS herausstellten und die zudem bei kleineren Graphen eine annehmbaren Laufzeit haben. Diese Heuristiken basieren auf der Fragestellung nach einem Linear Arrangement, was ebenfalls ein NP-schweres Problem ist.

Geladaris, Lionakis und Tollis stellten eine Methode zur Berechnung einer FAS vor, die eine vereinfachte Version von PageRank verwendet. Diese hat zwar eine größere Laufzeit als die beiden anderen Heuristiken, jedoch lieferte deren Algorithmus eine bessere FAS [GLT22].

In dieser Arbeit wurde die Idee, Zentralitätsmaße zur Berechnung einer FAS zu verwenden, aufgegriffen, und die Eignung weiterer Zentralitätsmaße, wie Closeness- und Betweenness-Zentralität, untersucht.

Geladaris et al. reduzierten die Graphen auch auf deren starke Zusammenhangskomponenten, um die Laufzeit ihres Algorithmus zu verbessern. Wir überprüften daher auch, wie sich diese Methode zur Verbesserung von GreedyFAS und SortFAS* eignet.

Auch testeten wir verschiedene Parameter bei der Berechnung von PageRank und stellen Edge-PageRank vor, das die Laufzeit auf bis zu 23% bei identischer FAS-Größe gegenüber der von Geladaris et al. verkürzten kann.

In Kapitel 2 werden die oben genannten Heuristiken genauer vorgestellt und deren Funktionsweise anhand von Beispielen erklärt.

In Kapitel 3 wird die Funktionsweise der zentralitätsmaßbasierten FAS-Algorithmen vorgestellt, es werden verschiedene Zentralitätsmaße beschrieben und die unterschiedlichen Versionen von PageRank diskutiert, sowie die in dieser Arbeit verwendeten FAS-Heuristiken aufgeführt. Des Weiteren wird der Einfluss einer Reduzierung auf die starken Zusammenhangskomponenten und die Vor- und Nachteile der verschiedenen Zentralitätsmaße ausgewertet.

Kapitel 4 beschäftigt sich mit der Optimierung des PageRank-Ansatzes. Dazu wurde in Abschnitt 4.1 PageRank so angepasst, dass auf die Erstellung eines Line Graphen verzichtet werden kann, indem die Kanten direkt bewertet werden. Außerdem wurde mit der Anzahl der Iterationen experimentiert und weitere Abwandlungen von PageRank getestet, so etwa eine dynamische PageRank-Version und eine Version von PageRank mit Dämpfungsfaktor.

2. Existierende FAS-Heuristiken

In diesem Kapitel werden bekannte Algorithmen zur Berechnung von Feedback Arc Sets, ein neuer Ansatz von Geladaris, Lionakis und Tollis, vorgestellt und deren Funktionsweisen anhand von Beispielen erklärt. Dazu dient der Graph $G = (V, E)$ in Abbildung 2.1 als Beispielgraph, wobei V die Menge der Knoten, E die Menge der Kanten, $n = |V|$, $m = |E|$ und $\mathrm{d}^{\mathrm{out}}(v)$ und $\mathrm{d}^{\mathrm{in}}(v)$ der Ausgangsgrad, bzw. der Eingangsgrad eines Knotens v ist.

Einige Heuristiken, wie *GreedyFAS* oder *SortFAS*, basieren auf einer äquivalenten Fragestellung nach einer minimalen FAS, nämlich der nach einem *Linear Arrangement*.

Ein Linear Arrangement (LA) ist eine Sortierung der Knoten mit einer minimalen Anzahl an Rückwärtskanten. Da nach Entfernen aller Rückwärtskanten in einem LA ein gerichteter Graph azyklisch ist, bildet die Menge dieser Rückwärtskanten eine FAS [GLT22]. Oder anders formuliert: Ein gerichteter Graph ist azyklisch genau dann, wenn ein LA existiert, das keine Rückwärtskanten enthält.

2.1 GreedyFAS

Der GreedyFAS-Algorithmus berechnet zunächst ein LA und gibt anschließend alle sich ergebenden Rückwärtskanten als FAS aus. Die Idee dieses Algorithmus ist, eine Knotenfolge zu berechnen, indem alle blätterähnlichen Knoten rechts und alle wurzelähnlichen Knoten links angeordnet werden, um so möglichst wenig Kanten zu erhalten, welche von rechts nach links gerichtet sind.

Dazu werden die Eingangs- und Ausgangsgrade der Knoten betrachtet, die Knoten nacheinander aus dem Graphen entfernt und in einer von zwei Knotensequenzen, s_1 und s_2, eingefügt.

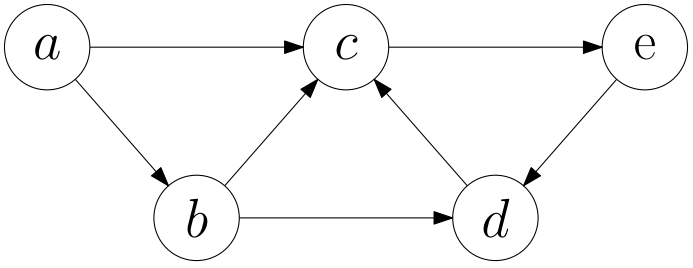


Abbildung 2.1: Beispielgraph.

Tabelle 2.1: Greedy-FAS auf Beispielgraph.

Runde 1	Da der Graph keinen Blattknoten enthält, aber mit Knoten a einen Wurzelknoten besitzt, d. h. $\mathrm{d}^{\mathrm{in}}(a) = 0$, wird Knoten a aus dem Graphen entfernt und an s_1 angehängt.
Runde 2	Da $\mathrm{d}^{\mathrm{in}}(b) = 0$, gibt es erneut einen Wurzelknoten, weshalb Knoten b aus dem Graphen entfernt und an s_1 angehängt wird. Somit gilt: $s_1 = a, b$.
Runde 3	Für alle verbleibenden Knoten c, d und e gilt $\mathrm{d}^{\mathrm{in}} = \mathrm{d}^{\mathrm{out}} = 1$. Somit gibt es keinen Wurzelknoten und keinen Blattknoten. Da auch für alle verbleibenden Knoten $v \in \{c, d, e\} : \delta(v) = \mathrm{d}^{\mathrm{out}}(v) - \mathrm{d}^{\mathrm{in}}(v) = 0$, wird der zufällig gewählte Knoten c aus dem Graphen entfernt und an s_1 angehängt. Somit gilt: $s_1 = a, b, c$.
Runde 4	Da nun der Ausgangsgrad des Knoten d Null ist, wird Knoten d aus dem Graphen entfernt und an s_2 vorne eingefügt. Somit gilt: $s_2 = d$.
Runde 5	Da nun der Ausgangsgrad des Knoten e Null ist, wird Knoten e aus dem Graphen entfernt und an s_2 vorne eingefügt. Somit gilt: $s_2 = e, d$. Da der Graph jetzt leer ist, ist die Folge $s_1 s_2 = a, b, c, e, d$ ein LA, das die Rückwärtskante (d, c) enthält. Somit bildet diese Kante eine FAS für den Beispielgraphen in Abbildung 2.1.

Dazu werden zunächst alle Blätter, d. h. Knoten mit Ausgangsgrad Null, aus dem Graphen entfernt und an s_2 vorne angehängt. Nachdem der Graph keine Blätter mehr enthält, werden alle Knoten mit Eingangsgrad Null aus dem Graphen entfernt und an s_1 hinten angehängt. Anschließend wird der wurzelähnlichste Knoten, also der Knoten, für den $\delta(v) = \mathrm{d}^{\mathrm{out}}(v) - \mathrm{d}^{\mathrm{in}}(v)$ maximal ist, aus dem Graphen entfernt und an s_1 hinten angehängt. Diese drei Schritte werden solange wiederholt, bis der Graph keine Knoten mehr enthält. Die Folge $s_1 s_2$ dient nun als Linear Arrangement und deren Rückwärtskanten werden als FAS ausgegeben.

Zur Verdeutlichung der Funktionsweise ist in Tabelle 2.1 die Ausführung auf dem Beispielgraph in Abbildung 2.1 mit den Entscheidungskriterien und Zwischenergebnissen angegeben.

Die Laufzeit beträgt bei einer direkten Implementierung $O(n^2)$, mit der Implementierung von Simpson, Srinivasan und Thoma läuft dieser Algorithmus in $O(n + m)$ Zeit. Dadurch eignet sich dieser Algorithmus auch für große Graphen [SST16b].

2.2 SortFAS

Auch der Algorithmus SortFAS berechnet zunächst ein LA und gibt anschließend die sich daraus ergebenden Rückwärtskanten als FAS aus. Hierzu werden die Knoten in ihrer initialen Reihenfolge $(v_1 \dots v_n)$ nacheinander von links nach rechts abgearbeitet. Dabei wird für einen Knoten jede mögliche Position weiter links anhand der sich daraus ergebenden Rückwärtskanten bewertet und an die bestmögliche Stelle eingefügt. Sollte es mehrere Positionen geben, an denen die Anzahl der sich ergebenden Rückwärtskanten minimal ist, wird die am weitesten links liegende Position gewählt.

Entwickelt wurde SortFAS von Brandenburg und Hanauer [BH11]. Deren direkte Implementierung hat eine Laufzeit von $O(n^3)$. Die Implementierung von Simpson et al. (2016)

Tabelle 2.2: SortFAS auf Beispielgraph.

	Knotensortierung
Runde 1	$\boldsymbol{a}$, b, c, d, e
Runde 2	a, $\boldsymbol{b}$, c, d, e
Runde 3	a, b, $\boldsymbol{c}$, d, e
Runde 4	a, b, $\boldsymbol{d}$, c, e
Runde 5	$\boldsymbol{e}$, a, b, d, c

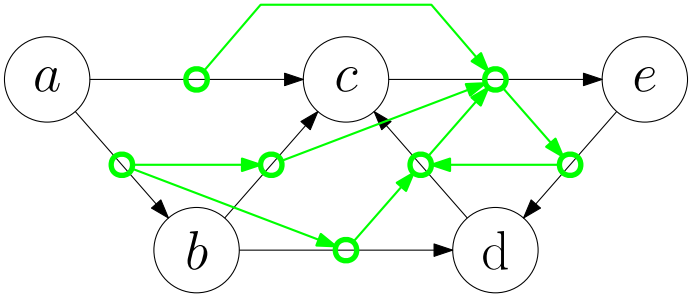


Abbildung 2.2: Line Graph des Beispielgraphen.

läuft jedoch bei einer Implementierung mit Adjazenzlisten in $O(n^2 \log(\mathrm{d}_{\max})$ Zeit, wobei $\mathrm{d}_{\max}$ der maximale Knotengrad des Graphen ist [SST16b].

Zum besseren Verständnis sind in Tabelle 2.2 für jede Runde die Zwischenergebnisse, d. h. die Sortierung der Knoten nach jeder Runde, angegeben. Die zuletzt betrachteten Knoten sind fett gedruckt. Die Knotensortierung nach Runde 5 stellt das LA dar. Somit liefert SortFAS die Kante (c, e) als FAS.

Eine Weiterentwicklung von Simpson, Srinivasan und Thoma ist SortFAS*, bei der die Knotensortierung mehrfach ausgeführt wird.

2.3 PageRankFAS

Diese Arbeit greift die Idee des *PageRankFAS*-Algorithmus von Geladaris et al. (2022) auf. Sie basiert auf der Annahme, dass sich die Anzahl der Kreise, die eine bestimmte Kante enthalten, sich im PageRank-Score der Kanten widerspiegelt. Durch das Entfernen dieser Kante wäre es demnach wahrscheinlich, viele Kreise zu durchbrechen [GLT22].

Um dieses Verfahren erklären zu können, wird zunächst noch auf einige Begriffe eingegangen.

Der *Line Graph* $L(G)$ eines gerichteten Graphen $G = (V, E)$ mit der Knotenmenge $V = \{a, b, c, d, \dots\}$ hat für jede Kante in G einen Knoten. Eine Kante in $L(G)$ existiert genau dann zwischen zwei Knoten (a, b) und (c, d), wenn $b = c$, d. h. wenn im Originalgraphen der Zielknoten der einen Kante gleich dem Ausgangsknoten der anderen Kante ist [GYZ14].

Für den Beispielgraphen ist in Abbildung 2.2 der zugehörige Line Graph in Grün dargestellt.

Ein Graph ist stark zusammenhängend, wenn es für je zwei Knoten einen Kreis gibt, in dem diese enthalten sind. Ist ein Graph nicht stark zusammenhängend, so kann dieser mittels *Tarjan*-Algorithmus in Linearzeit ($O(n+m)$) in seine starken Zusammenhangskomponenten (SCC) zerlegt werden [GPP03]. Ein SCC ist ein maximal stark zusammenhängender Teilgraph, bei dem das Hinzufügen jedes weiteren Knoten diese Eigenschaft verletzen würde.

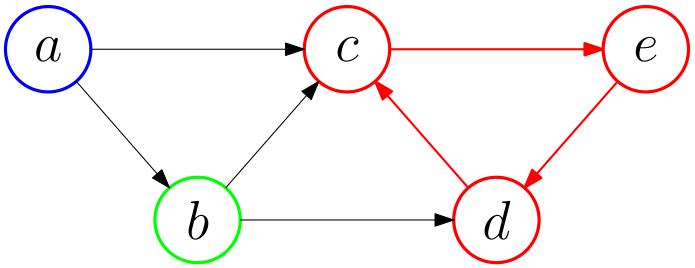


Abbildung 2.3: Starke Zusammenhangskomponenten des Beispielgraphen.

Für den Beispielgraphen in Abbildung 2.1 ergeben sich drei SCCs: $\{a\}$, $\{b\}$, $\{c, d, e\}$, wie auch in Abbildung 2.3 dargestellt ist.

Ein Graph ist genau dann ein gerichteter azyklischer Graph (*DAG*), wenn jedes seiner SCCs aus nur einem Knoten besteht. Daraus folgt, dass sich Rückwärtskanten nur innerhalb eines SCCs befinden können [GLT22].

Daher berechnet PageRankFAS zunächst die starken Zusammenhangskomponenten des Graphen, erstellt für jedes SCC einen Line Graph und berechnet so nur auf diesen Teilgraphen eine zu entfernende Kante, um dessen Laufzeit zu verbessern [GLT22].

Den Knoten dieser Line Graphen wird mittels eines vereinfachten PageRank-Algorithmus ein Wert zugewiesen und anschließend die entsprechende Kante des am höchsten bewerteten Knoten aus dem Originalgraphen entfernt und der FAS hinzugefügt.

Dies wird solange wiederholt, bis der Graph azyklisch ist.

3. Zentralitätsmaßbasierte FAS-Algorithmen

Die folgenden Heuristiken basieren alle auf dem gleichen Prinzip. Solange der Graph nicht azyklisch ist, werden die folgenden drei Schritte ausgeführt:

1. Reduziere den Graph auf seine SCCs.
2. Erstelle für jedes SCC einen Line Graph.
3. Finde in jedem Line Graphen den am höchsten bewerteten Knoten, füge die entsprechende Kante der FAS hinzu und lösche sie aus dem Originalgraphen.

Für die Berechnung der SCCs wird der *Tarjan*-Algorithmus verwendet (Laufzeit $O(n+m)$). Anschließend werden alle Kanten aus dem Graphen entfernt, deren Start- und Zielknoten in unterschiedlichen SCCs liegen. Des Weiteren werden alle Knoten aus dem Graphen gelöscht, die ein eigenes SCC bilden. Diese Reduzierung des Graphen benötigt eine Laufzeit von $O(n+m)$.

3.1 Zentralitätsmaße

Zentralitätsmaße gehen der Frage nach, wie zentral ein Knoten oder eine Kante sich in einem Graphen befindet. Zentralität ist nicht einheitlich definiert, sondern je nach Vorstellung, wann ein Knoten oder eine Kante zentral ist, existieren verschiedene Zentralitätsmaße.

Die einfachste Form ist die *Degree*-Zentralität. Diese bezeichnet einen Knoten als zentral, wenn er viele direkte Verbindungen zu anderen Knoten hat. Dieses Maß ist jedoch so lokal, dass es für unseren Anwendungsfall kaum geeignet sein dürfte. Die *Closeness*-Zentralität bezeichnet den Knoten als zentral, der die kürzesten Distanzen zu allen anderen Knoten hat. Ein weiteres Zentralitätsmaß ist die *Betweenness*-Zentralität. Demnach ist ein Knoten zentral, wenn er in vielen kürzesten Wegen zwischen je zwei Knoten enthalten ist [THS+11] [KGKK+20]. *PageRank* ist ein Zentralitätsmaß das eingeführt wurde, um Webseiten nach ihrer Wichtigkeit zu ordnen. Der Knoten eines Graphen hat einen hohen PageRank-Score, wenn die Summe der PageRank-Scores der Knoten, die auf ihn verweisen, hoch ist. Somit wird ein Knoten hoch bewertet, wenn er viele eingehende Kanten hat, aber auch, wenn er wenige eingehende Kanten von hoch bewerteten Knoten hat [PBMW99].

3.1.1 PageRank

Sei v ein Knoten, F_v die Menge der Nachfolgerknoten und B_v die Menge der Vorgängerknoten von v. Sei $N_v = |F_v|$ die Anzahl der ausgehenden Kanten von v, und c der Normalisierungsfaktor, damit die Gesamtbewertung aller Knoten konstant bleibt. Der Normalisierungsfaktor c muss kleiner Eins sein, da für Blattknoten der Score verloren geht. Vereinfacht ist der PageRank-Score $C_{\mathrm{PR}}(v)$ eines Knoten v definiert als:

$$C_{\mathrm{PR}}(v) = c \sum_{w \in B_v} \frac{C_{\mathrm{PR}}(w)}{N_w}.$$

Es können beliebige Startwerte gewählt werden. Ausgeführt wird diese Zuweisung, bis die Werte konvergieren.

Ein Problem, das bei dieser vereinfachten Version von PageRank auftreten kann, sind *Rank Sinks*. Verweisen beispielsweise zwei Knoten nur aufeinander und hat einer dieser Knoten eine weitere eingehende Kante, so sammelt diese Schleife die Scores der anderen Knoten, ohne diese weiter zu verteilen. Solche Schleifen werden als Rank Sinks bezeichnet. Um dieses Problem zu umgehen, wurde die *Rank Source* $E(w)$ eingeführt, um Rank Sinks auszugleichen. Dazu wird einem Knoten nur ein Teil seines errechneten Wertes zugewiesen, dafür erhalten aber alle Knoten einen zusätzlichen Wert, unabhängig von ihren Kanten. Die Rank Source eignet sich nicht nur zum Ausgleich der Rank Sinks, sondern erwies sich auch als mächtiger Parameter bei der Berechnung von PageRank [PBMW99].

Geladaris, Lionakis und Tollis verwenden bei ihrer Implementierung von PageRank lediglich eine vereinfachte Version, da sie möchten, dass die PageRank-Scores die tatsächliche Wichtigkeit der Knoten widerspiegeln [GLT22].

Bei ihrer Berechnung beginnen sie mit gleichverteilten Startwerten, d. h. jedem Knoten in einem Graphen $G = (V, E)$ mit $|V| = n$ Knoten wird ein Wert von $\frac{1}{n}$ zugewiesen. Anschließend wird für eine vorher bestimmte Anzahl an Iterationen der Wert eines jeden Knoten auf all seine Nachfolger gleichmäßig verteilt. Wenn ein Knoten keinen Nachfolger hat, behält er seinen Wert. Nach jeder Iteration werden den Knoten ihre neuen Werte zugewiesen.

Die Anzahl der Iterationen richtet sich nach der Größe des Graphen. Geladaris et al. (2022) sind der Ansicht, dass für kleine und mittlere Graphen ca. fünf Iterationen ausreichend sind, damit die Werte konvergieren. Sei k die Anzahl der Iterationen, so beträgt die Laufzeit von PageRank $O(k(n + m))$ [GLT22].

Der Pseudocode des vereinfachten PageRank ist in Algorithmus 3.1 angegeben.

Für den Beispielgraphen aus Abbildung 2.1 sind die Ergebnisse für die ersten beiden Iterationen der einfachen PageRank-Version in Tabelle 3.1 aufgeführt. Dieses Beispiel zeigt auch das Problem der Sink Ranks bei nicht geeigneten Graphen.

3.1.2 Closeness-Zentralität

Die Closeness-Zentralität eines Knoten spiegelt die Nähe zu allen anderen Knoten wider. Sei $\mathrm{d}(v, u)$ die Distanz, also Länge des kürzesten Pfades von Knoten v zu Knoten u. Die Closeness-Zentralität $C_C(v)$ eines Knoten v ist der Kehrwert der Summe der Distanzen vom Knoten v zu allen anderen Knoten n:

$$C_C(v) = \frac{1}{\sum_{w, w \neq v} \mathrm{d}(v, w)}.$$

Algorithm 3.1: VEREINFACHTES PAGERANK

Input: Digraph $G = (V, E)$, number of iterations k
Data: Set of nodes pointing to node v B_v
Output: PageRank scores of G

```
foreach node v in G do
    C_PR(v) ← 1/|V|
for k iterations do
    foreach node v in G do
        C_PR_new(v) ← Σ_{w∈B_v} C_PR(w) / d^out(w)
    C_PR ← C_PR_new
    C_PR_new ← 0
return C_PR
```

Tabelle 3.1: Vereinfachtes PageRank auf Beispielgraph.

Knoten	Initialer Score	Score nach Runde 1	Score nach Runde 2
a	0,2	0	0
b	0,2	0,1	0
c	0,2	0,4	0,35
d	0,2	0,3	0,25
e	0,2	0,2	0,4

Ein Knoten mit hohem Closeness-Score ist weniger auf die Vermittlung durch andere Knoten angewiesen [Mut10]. Damit die Summen der Distanzen vergleichbar sind, muss der Graph stark zusammenhängend sein [BKN12].

Die kürzesten Wege für jeden Knoten werden mit dem Dijkstra-Algorithmus berechnet. Anschließend werden die Distanzen aufsummiert. Somit ergibt sich eine Laufzeit von $O(n(m + n \log n))$.

3.1.3 Betweenness Zentralität

Wie auch die Closeness-Zentralität basiert die Betweenness-Zentralität auf den kürzesten Wegen des Graphen.

Die Betweeness Zentralität $C_B(v)$ eines Knoten v entspricht der Summe der Häufigkeiten, mit der dieser Knoten in allen kürzesten Wegen zwischen je zwei Knoten vorkommt. Formal gilt somit: Sei σ_{st} die Anzahl der kürzesten Wege zwischen zwei Knoten s und t, wobei $s \neq t$ und $\sigma_{\mathrm{st}}(v)$ die Anzahl der kürzesten Wege, die v enthalten, so gilt [Bra01]:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{\mathrm{st}}(v)}{\sigma_{\mathrm{st}}}.$$

In Tabelle 3.2 sind die kürzesten Wege des Beispielgraphen aufgeführt und in Tabelle 3.3 die sich daraus ergebende Betweeness.

Der Algorithmus von Brandes (2001) benötigt eine Laufzeit von $O(nm)$.

3.2 FAS-Algorithmen

Tabelle 3.4 bietet einen Überblick über die verwendeten FAS-Heuristiken. Geladaris, Lionakis und Tollis gaben in ihrer Arbeit an, die Graphen zuerst auf SCCs zu reduzieren, um

Tabelle 3.2: Kürzeste Wege im Beispielgraphen.

Startknoten / Zielknoten	a	b	c	d	e
a	-	$a \to b$	$a \to c$	$a \to b \to d$	$a \to c \to e$
b	-	-	$b \to c$	$b \to d$	$b \to c \to e$
c	-	-	-	$c \to e \to d$	$c \to e$
d	-	-	$d \to c$	-	$d \to c \to e$
e	-	-	$e \to d \to c$	$e \to d$	-

Tabelle 3.3: Betweenness auf Beispielgraph.

Kante	Betweeness-Score
a	0
b	1
c	3
d	1
e	1

die Laufzeit zu verkürzen [GLT22]. Damit überprüft werden kann, ob eine Reduzierung auf SCCs auch Auswirkungen auf die FAS-Größe hat, wurden auch Versionen ohne eine solche Reduzierung implementiert.

Implementiert wurden diese Algorithmen in C++ und unter Verwendung des Open Graph Drawing Framework (OGDF)[1] [CGJ+13] in der Version Dogwood (Februar 2022). Dieses enthält auch Algorithmen zur Überprüfung, ob ein Graph azyklisch ist, zur Berechnung von SCCs und zum Erstellen von zufälligen Graphen. Die Ausführung fand auf einem Cluster bestehend aus 8 Knoten mit Intel Core i7-4770 Prozessoren mit je vier Kernen, einer Taktrate von 3,4 GHz und 32 GB RAM statt. Da jeder Berechnung ein Kern zugeteilt wurde, konnten 32 Berechnungen gleichzeitig durchgeführt werden. Als Betriebssystem installiert war Debian GNU/Linux 11 (bullseye) und kompiliert wurde mit GCC in der Version 10.2.1.

Um die verschiedenen Algorithmen vergleichen zu können, wurden zwei Mengen an Graphen generiert, eine mit kleiner Knotenanzahl aber höherem durchschnittlichem Ausgangsgrad und eine, mit mehr Knoten, jedoch mit geringerem Ausgangsgrad, um das Verhalten bei großen und dichten Graphen zu testen.

Eine Menge hatte Graphen mit den durchschnittlichen Ausgangsgraden $1,5$, 2, 3, 5, 6, 8, 10 und 12, und eine Knotenanzahl von 50, 100 und 150. Im Folgenden wird diese Menge als OutDeg-Graphen bezeichnet. Die Andere hatte Graphen mit 50, 100, 200, 400, 700, 1000, 1500, 2000, 2500 und 3000 Knoten und einen durchschnittlichen Ausgangsgrad zwischen $1,5$ und $2,7$ mit Schrittweite $0,2$. Sie wird Knoten-Graphen genannt.

Um zufallsbedingte Verzerrungen in den Ergebnissen auszugleichen, wurde für jede Kombination aus Ausgangsgrad und Knotenanzahl zehn Graphen generiert.

3.3 Auswertung

3.3.1 Reduzierung auf SCCs

Im Vergleich zwischen GreedyFAS und SCCGreedyFAS zeigt sich, dass SCCGreedyFAS eine kleinere FAS liefert, jedoch nur bei Graphen mit niedrigem Ausgangsgrad und geringer Knotenanzahl.

[1] http://www.ogdf.net

Bei den OutDeg-Graphen verringert sich FAS bei einem Ausgangsgrad von $1,5$ von $5,29\%$ auf $5,01\%$. Bei einem Ausgangsgrad von Zwei beträgt der Unterschied $0,28\%$ und bei allen anderen Ausgangsgraden ist der Unterschied kleiner als $0,02\%$ und konvergiert mit zunehmendem Ausgangsgrad gegen Null (siehe Abbildung 3.1b). Bei den Knoten-Graphen ist der Unterschied bei 50 und 100 Knoten kleiner als $0,26\%$, aber auch dieser verkleinert sich weiter und liegt bei 3000 Knoten nur noch bei $0,008\%$.

Die Auswirkung auf die Laufzeit ist hierbei nur minimal. So ist bei den Knoten-Graphen bei 3000 Knoten SCCGreedyFAS um $0,00028s$ schneller, bei den OutDeg-Graphen wiederum ist GreedyFAS bei einem Ausgangsgrad von $2,7$ um $0,001s$ besser. Alle anderen Zeiten liegen dazwischen.

Bei SortFAS* ist der Unterschied zu der Variante mit SCC etwas deutlicher. Bei den OutDeg-Graphen ist die FAS bei einem Ausgangsgrad von $1,5$ um $0,94\%$ kleiner kleiner als die FAS ohne SCC-Reduzierung mit $4,98\%$. Doch auch hier konvergiert der Unterschied mit zunehmendem Ausgangsgrad gegen Null. Bei Ausgangsgrad Zwei beträgt der Unterschied noch $0,41\%$, bei Drei noch $0,31\%$ und ab einem Ausgangsgrad von Fünf ist der Unterschied kleiner $0,0023\%$. Dabei war mal SortFAS*FAS und mal SCCSortFAS*FAS schneller, jedoch maximal um $0,00051s$.

Auf den Knoten-Graphen war bei den Knoten zwischen 100 und 3000 Knoten die FAS von SCCSortFAS* zwischen $0,65\%$ und $0,76\%$ kleiner und mit größer werdender Knotenanzahl wurde auch der zeitliche Gewinn deutlich. Beträgt der Unterschied bei 1000 Knoten lediglich $0,06s$, so beträgt er bei 3000 Knoten bereits $0,68s$, was nur 40% der Laufzeit von SortFAS* entspricht.

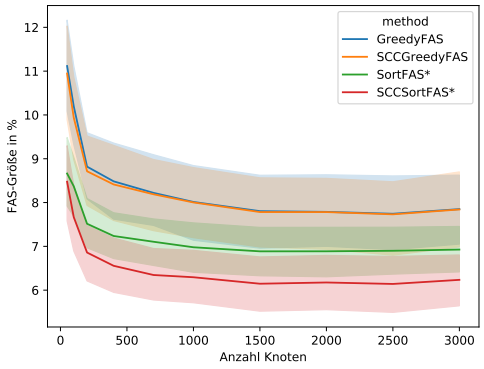
Die Ergebnisse sind graphisch in Abbilung 3.1 dargestellt.

Der Effekt der konvergierenden FAS-Größen mit dichter werdenden Graphen bei GreedyFAS und SortFAS* passt zur Entwicklung der Größe des größten ersten SCCs. Besteht es bei einem Ausgangsgrad von $1,5$ aus durchschnittlich $30,3\%$ der Knoten, enthält es bei Drei schon 88% und für die Graphen mit einem Ausgangsgrad größer Fünf über $98,6\%$. Diese Entwicklung ist in Abbildung 3.2 dargestellt. Da diese beiden Algorithmen auf jedem Teilgraphen nur einen Durchgang benötigen, um alle Kanten der FAS zu bestimmen, nimmt der Unterschied bei zunehmender SCC-Größe ab, da als Teilgraph der gesamte Graph betrachtet wird. Deshalb ist bei kargen Graphen eine Verbesserung erkennbar, bei dichteren Graphen hingegen ist eine Zerlegung in die SCCs jedoch überflüssig.

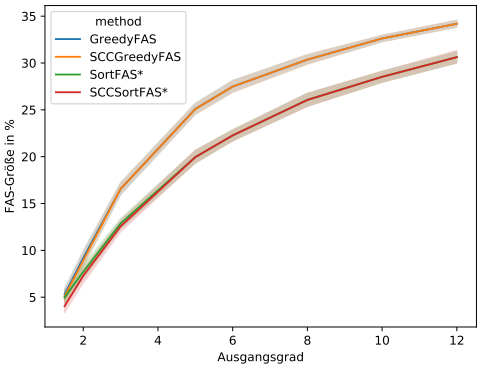
Ganz anders wirkt sich die Reduzierung auf die SCCs bei den zentralitätsmaßbasierten Heuristiken aus. Es verbessert nicht nur die Laufzeit, sondern hat auch einen erheblichen Einfluss auf die FAS-Größe.

Da auf den OutDeg-Graphen schon bei einem kleinen Ausgangsgrad von $1,5$ die FAS bei den zentralitätsmaßbasierten Heuristiken ohne SCC zwischen 64% und 89% liegt, bei den Versionen mit Reduzierung jedoch bei nur $4,8\%$ bis $5,2\%$, ist die FAS bei den Versionen ohne Reduzierung 13- bis 17-mal größer. Mit zunehmender FAS-Größe bei größer werdendem Ausgangsgrad sinkt dieser Faktor, jedoch liefert LGPRItFAS(5) bei den OutDeg-Graphen bei einem Ausgangsgrad von 12 eine FAS, die mehr als dreimal größer ist, wie die von SCCLGPRItFAS(5). Bei Closeness und Betweenness beträgt dieser Faktor noch $2,63$ bzw. $2,46$ (siehe Abbildung 3.3a).

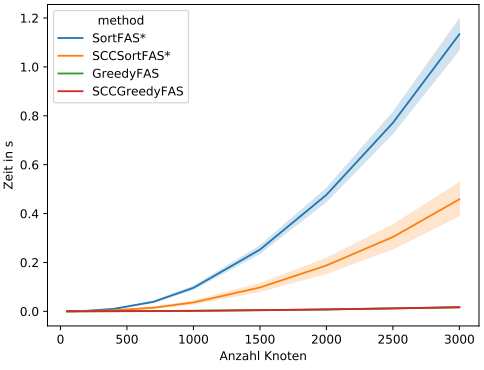
Somit liefern bei diesem Ansatz die Versionen ohne SCC-Reduzierung nicht nur eine unbrauchbar große FAS, sondern benötigen hierzu auch noch deutlich mehr Zeit, da die Berechnung der Zentralitätsmaße, die den Großteil der Laufzeit ausmacht, für jede entfernte Kante ausgeführt werden musste, weshalb hier die Reduzierung auf die SCC unverzichtbar ist.



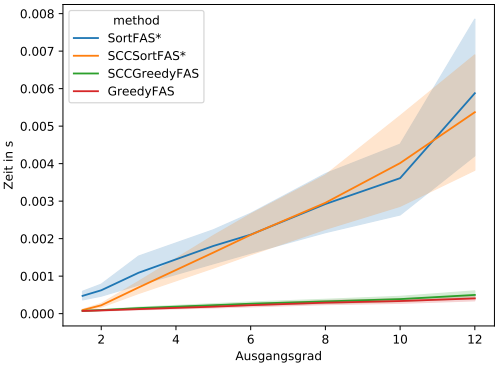
(a) FAS-Größen bei den Knoten-Graphen.



(b) FAS-Größe bei den OutDeg-Graphen.



(c) Laufzeit bei den Knoten-Graphen.



(d) Laufzeit bei den OutDeg-Graphen.

Abbildung 3.1: Auswirkungen der SCC-Reduzierung bei GreedyFAS und SortFAS*.

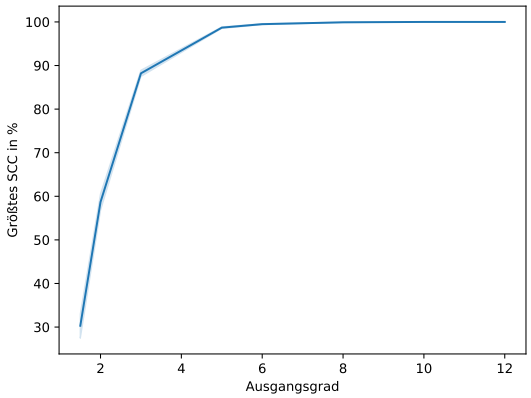


Abbildung 3.2: Anteil des größten SCCs der OutDeg-Graphen.

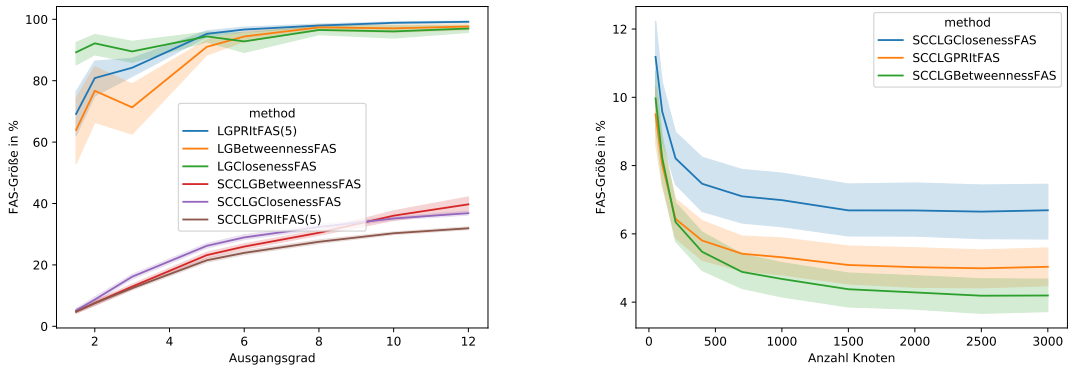
3.3.2 Zentralitätsmaße

Auch wurde untersucht, welches Zentralitätsmaß sich zur Berechnung einer FAS am besten eignet. Dazu wurden die Algorithmen SCCLGPRItFAS(5), SCCLGBetweennessFAS und SCCLGClosenessFAS auf den Knoten- und OutDeg-Graphen ausgeführt. Die Ergebnisse sind in Abbildung 3.3 dargestellt.

Dabei zeigte sich, dass bei den Knoten-Graphen SCCLGClosenessFAS mit einer FAS-Größe von $7,5\%$ bis $6,6\%$ bei 400 bis 1000 Knoten jeweils um ca. $1,6\%$ größer ist als die von SCCLGPRItFAS(5), und deshalb am schlechtesten abschneidet. Auch die Laufzeit von SCCLGClosenessFAS ist bei 3000 Knoten mit durchschnittlich 48 Minuten deutlich größer als die von SCCLGPRItFAS(5) mit nur $0,78s$. SCCLGBetweennessFAS liefert bei 50 Knoten noch eine FAS, die um $0,47\%$ größer ist als die von SCCLGPRItFAS(5), ab mehr als 200 Knoten kehrt sich dieses Verhältnis jedoch um und SCCLGBetweennessFAS berechnet eine FAS, die bei 3000 Knoten um $0,84\%$ kleiner ist. Die Laufzeit dafür beträgt hierzu aber über elf Minuten, wohingegen SCCLGPRItFAS(5) lediglich $0,78s$ benötigt.

Vergleicht man die verschiedenen Maße auf den OutDeg-Graphen, so ist PageRank den beiden anderen überlegen, da die FAS-Größe bei einem Ausgangsgrad von $1,5$ um $0,04\%$ kleiner ist als bei SCCLGBetweennessFAS. Dieser Unterschied wird bei zunehmender Dichte immer größer und beträgt bei einem Ausgangsgrad von Zwölf bereits $7,8\%$. Auch SCCLGClosenessFAS ist von Beginn an um $0,39\%$ schlechter als SCCLGPRItFAS(5) und auch dieser Abstand vergrößert sich weiter und beträgt bei 3000 Knoten $4,9\%$. Ist bis zu einem Ausgangsgrad von ca. Neun SCCLGBetweennessFAS bei der FAS-Größe SCCLGClosenessFAS überlegen, so hat dieser bei einem Ausgangsgrad von Zwölf mit $39,7\%$ eine um $2,9\%$ größere FAS als SCCLGClosenessFAS. Auch auf diesen Graphen ist SCCLGPRItFAS(5) bei der Laufzeit den beiden anderen deutlich überlegen. Beträgt sie bei einem Ausgangsgrad von Zwölf lediglich $0,7s$ so braucht SCCLGBetweennessFAS durchschnittlich $39,7s$ und SCCLGClosenessFAS sogar 28 Minuten.

Daher ist SCCLGBetweennessFAS SCCLGPRItFAS(5) nur bei Graphen mit größerer Knotenanzahl und niedrigem Ausgangsgrad etwas überlegen, jedoch bei einer viel größeren Laufzeit. Bei den dichteren OutDeg-Graphen liefert SCCLGPRItFAS(5) bei einem Ausgangsgrad von Zwölf eine um $7,8\%$ kleinere FAS und das bei einer viel besseren Laufzeit.



(a) FAS-Größen bei den OutDeg-Graphen.

(b) FAS-Größen bei den Knoten-Graphen.

Abbildung 3.3: Vergleich der zentralitätsmaßbasierten Heuristiken.

Tabelle 3.4: FAS-Algorithmen

SCCLGPRItFAS	Entspricht dem Algorithmus von Geladaris et al. (2022). Da sie PageRank fünf Iterationen lang ausführten, wird deren Version im Folgenden als SCCLGPRItFAS(5) bezeichnet.
SCCLGClosenessFAS	Reduziert den Graphen auf seine SCCs und erstellt für jedes SCC einen Line Graph. Die entsprechende Kante des von Closeness Zentralität am höchsten bewerteten Knoten wird der FAS hinzugefügt.
SCCLGBetweennessFAS	Reduziert den Graphen auf seine SCCs und erstellt für jedes SCC einen Line Graph. Die entsprechende Kante des von Betweenness Zentralität am höchsten bewerteten Knoten wird der FAS hinzugefügt.
LGPRItFAS	Solange der Graph nicht kreisfrei ist, wird der Line Graph des gesamten Graphen erstellt und die entsprechende Kante des von PageRank am höchsten bewerteten Knoten entfernt.
LGClosenessFAS	Solange der Graph nicht kreisfrei ist, wird der Line Graph des gesamten Graphen erstellt und die entsprechende Kante des von Closeness am höchsten bewerteten Knoten entfernt.
LGBetweennessFAS	Solange der Graph nicht kreisfrei ist, wird der Line Graph des gesamten Graphen erstellt und die entsprechende Kante des von Betweenness am höchsten bewerteten Knoten entfernt.
GreedyFAS	Entspricht dem ArrayFAS-Algorithmus von Simpson et al. (2016).
SCCGreedyFAS	Reduziert den Graphen auf seine SCCs und führt auf jedem dieser Teilgraphen GreedyFAS aus.
SortFAS*	Entspricht dem SortFAS*-Algorithmus von Simpson et al. (2016) mit fünf Iterationen.
SCCSortFAS*	Reduziert den Graphen auf seine SCCs und führt auf jedem dieser Teilgraphen SortFAS* aus.

4. Optimierung des PageRank-Ansatzes

Da sich in Kapitel 3 gezeigt hat, dass PageRank als Zentralitätsmaß für die Berechnung von Feedback Arc Sets am besten geeignet ist, wurde versucht, SCCLGPRIt(5) zu verbessern.

4.1 Edge PageRank

Um die Laufzeit von SCCLGPRIt zu verbessern, stellen wir einen Algorithmus vor, der auf das Erstellen eines Line Graphen verzichten kann, indem PageRank so angepasst wurde, dass die Kanten direkt bewertet werden. Dazu wird allen Kanten der Wert $\frac{1}{\mathrm{m}}$ zugewiesen, wobei m der Anzahl der Kanten im Teilgraphen entspricht. Der Wert einer Kante wird auf die ausgehenden Kanten des Zielknotens aufgeteilt. Im Folgenden wird dieses angepasste Zentralitätsmaß als *Edge PageRank* bezeichnet. Es wird vorausgesetzt, dass vor dem Aufruf alle Kanten zwischen verschiedenen SCCs entfernt wurden. Der Pseudocode ist in Algorithmus 4.1 angegeben. Dass dieser Algorithmus dasselbe Ergebnis wie Algorithmus 3.1 liefert, besagt das Theorem 4.1.

Dieser Algorithmus ermöglicht die Einführung der FAS-Heuristik *SCCEdgePRItFAS*, welche exakt wie SCCLGPRItFAS funktioniert, nur mit dem Unterschied, dass zur Bestimmung einer Kante, welche der FAS hinzugefügt werden soll, der Algorithmus 4.1 anstatt des Algorithmus 3.1 verwendet wird.

Theorem 4.1. *Sei $L(G) = (V_L, E_L)$ der Line Graph des Graphen $G = (V_G, E_G)$. Sei $\mathrm{LG}(e) = v \in V_L$ der entsprechende Knoten einer beliebigen Kante $e \in E_G$. Sei $C_{\mathrm{PR}_{\mathrm{node}}}(v)$ der PageRank-Score des Knoten v des Algorithmus 3.1 und $C_{\mathrm{PR}_{\mathrm{edge}}}(e)$ der PageRank-Score der Kante e des Algorithmus 4.1, so gilt für jede beliebige Kante e:*

$$C_{\mathrm{PR}_{\mathrm{edge}}}(e) = C_{\mathrm{PR}_{\mathrm{node}}}(v)$$

Beweis. Dies lässt sich mit vollständiger Induktion beweisen.

Sei $B_V(v)$ die Menge der Vorgängerknoten des Knoten v, $F_V(v)$ die Menge der Nachfolgerknoten des Knoten v, $B_E(e)$ die Menge der beim Startknoten der Kante e eingehenden Kanten und $F_E(e)$ die Menge der vom Zielknoten der Kante e ausgehenden Kanten.

Da im Line Graphen für jede Kante im Graphen (hier Subgraphen) ein Knoten erstellt wird, ist die Anzahl der Knoten im Line Graph gleich der Anzahl der Kanten im Subgraph, d. h. $|V_L| = |E_G|$

Algorithm 4.1: EDGE PAGERANK

Input: Digraph $G = (V, E)$, number of iterations k, number of subgraph edges m, list of nodes of SCC
Data: Set of edges pointing to node e.source B_e
Output: Edge PageRank scores of SCC subgraph in G

```
1 foreach edge e in G do
2     C_PR(e) ← 1/m
3 for k iterations do
4     foreach edge e in G do
5         if e.source ∈ SCC then
6             C_PR_new(e) ← Σ_{a∈B_e} C_PR(a) / d^out(a.target)
7     C_PR ← C_PR_new
8     C_PR_new ← 0
9 return C_PR
```

Zu Beginn werden allen Knoten und Kanten gleichverteilte Werte zugewiesen. Somit gilt $\forall v \in V_L \colon C_{\mathrm{PR}_{\mathrm{node}}}(v) = \frac{1}{|V_L|}$ und $\forall e \in V_G \colon C_{\mathrm{PR}_{\mathrm{edge}}}(e) = \frac{1}{|E_G|}$. Da $|V_L| = |E_G|$, haben alle Knoten und alle Kanten vor der ersten Iteration den selben PageRank-Score. Daraus folgt: *Induktionsanfang:* $C_{\mathrm{PR}_{\mathrm{node}}}(v) = \frac{1}{|V_L|} = \frac{1}{|E_G|} = C_{\mathrm{PR}_{\mathrm{edge}}}(e)$.

Schleifeninvariante: Sei $e \in E_G$ eine beliebige Kante, und ist $\mathrm{LG}(e) = v$, so gilt $C_{\mathrm{PR}_{\mathrm{node}}}(v) = C_{\mathrm{PR}_{\mathrm{edge}}}(e)$.

Induktionsschritt: Die Berechnungen der neuen PageRank-Scores entsprechen den folgenden Zuweisungen:

$$C_{\mathrm{PR}_{\mathrm{node}}}(v) = \sum_{w \in B_V(v)} \frac{C_{\mathrm{PR}_{\mathrm{node}}}(w)}{|F_V(w)|}$$

$$C_{\mathrm{PR}_{\mathrm{edge}}}(e) = \sum_{a \in B_E(e)} \frac{C_{\mathrm{PR}_{\mathrm{edge}}}(a)}{|F_E(a)|}.$$

Da $B_V(v)$ die Menge der Vorgängerknoten des Knoten v ist und $B_E(e)$ die Menge der beim Startknoten der Kante e eingehenden Kanten und da für jede Kante in G ein Knoten im Line Graphen existiert, gilt: $\forall w \in B_V(v) \colon \exists a \in B_E(e) \colon \mathrm{LG}(a) = w$.

Außerdem folgt aus der Induktionsannahme: $C_{\mathrm{PR}_{\mathrm{node}}}(w) = C_{\mathrm{PR}_{\mathrm{edge}}}(a)$.

Sei $a = (x, y) \in F_E(e)$. Aus der Definition des Line Graphen folgt, dass für jede von x ausgehende Kante ein Knoten in $F_{V(v)}$ existiert und ein Knoten nur existiert, wenn es eine Kante gibt. So ist die Menge der Nachfolgerknoten $F_V(w)$ gleich groß der Menge der Nachfolgerkanten $F_E(a)$.

Zusammengefasst gilt somit, dass für eine beliebige Kante e ein Knoten v mit $\mathrm{LG}(e) = v$ existiert, und dass für jede Vorgängerkante a von e ein Vorgängerknoten w von v existiert, wobei $C_{\mathrm{PR}_{\mathrm{node}}}(w) = C_{\mathrm{PR}_{\mathrm{edge}}}(a)$. Da auch für jede Nachfolgerkante von a genau ein Nachfolgerknoten von w exisiert und daher $|F_V(w)| = |F_E(a)|$, gilt die Behauptung.

□

Um die Auswirkung zu messen, wurde eine Menge an Testgraphen mit den Ausgangsgraden $1,5$, 2, 3, 5, 7, 8, 10, 12 und 15 und einer Knotenanzahl von 50, 100, 200, 400, 700, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500 und 5000 generiert. Für jede Kombination aus Knotenanzahl und Ausgangsgrad wurden zehn Graphen erzeugt, also insgesamt 1260 Graphen. Die Umgebung ist die gleiche wie in Kapitel 3.

Werden die Graphen nach ihrem Ausgangsgrad gruppiert, so braucht SCCEdgePRIt(5) durchschnittlich nur 62% der Laufzeit von SCCLGPRIt(5), gruppiert nach der Anzahl der Knoten sind es nur 46%.

Der größte relative Zeitgewinn wurde auf dieser Graphenmenge bei den Graphen mit 700 Knoten erzielt. Hier benötigte SCCEdgePRIt(5) lediglich 23% der Laufzeit von der Version mit Line Graph. Die Laufzeit konnte so von $52s$ auf $11,7s$ verkürzt werden. Doch auch bei 5000 Knoten könnte durch den Verzicht auf die Generierung der Line Graphen die durchschnittliche Laufzeit von 49 Minuten auf 33 Minuten verkürzt werden, weshalb hier die relative Laufzeit bei 67% liegt. Betrachtet man die Laufzeit der Graphen nach deren Ausgangsgrad, so profitiert man bei einem Ausgangsgrad von 5 am meisten von dieser Version, da hier die Laufzeit mit $31s$ nur 34% der Laufzeit von SCCLGPRIt(5) mit $92s$ entspricht. Auch mit zunehmendem Ausgangsgrad steigt die relative Laufzeit und beträgt bei 15 noch 66%. Bei einem Ausgangsgrad von nur $1,5$ hingegen ist diese Version um 5% schlechter, allerdings brauchen beide eine Laufzeit von unter $0,02s$, weshalb diese Zeiteinbuße zu vernachlässigen ist.

Die absoluten Laufzeiten sowie die Entwicklung der relativen Laufzeiten sind in Abbildung 4.1 dargestellt.

Die Größe der FAS ist bei diesen beiden Algorithmen natürlich identisch.

Zusammenfassend kann man sagen, dass SCCEdgePRIt eine deutliche Verbesserung gegenüber SCCLGPRIt darstellt.

4.2 Anzahl Iterationen

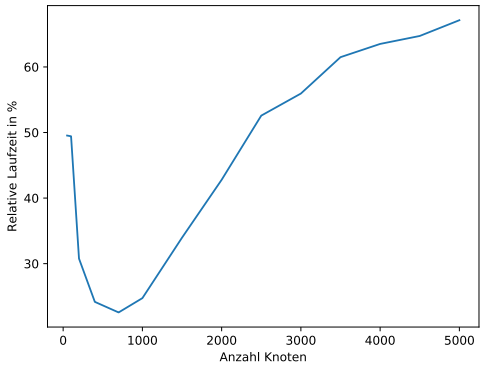
Um untersuchen zu können, wie sich die Anzahl der Iterationen bei PageRank auf die FAS-Größe und die Laufzeit auswirkt, haben wir SCCEdgePRItFAS mit den Parametern 3, 5, 7, 10 und 25 für die Anzahl der Iterationen auf den Testgraphen ausgeführt.

Es zeigte sich, dass mit zunehmenden Wiederholungen der PageRank-Routine die FAS kleiner wurde und die Laufzeit zunahm. Nahm jedoch die Laufzeit proportional zur Anzahl der Iterationen zu, so beginnt die FAS-Größe zu konvergieren. Betrachtet man hierzu die durchschnittliche FAS-Größen bei den Graphen ab 1000 Knoten, wie in Abbildung 4.2a dargestellt ist, so hat die FAS bei drei Wiederholungen eine durchschnittliche Größe von $19,85\%$, bei fünf Iterationen $19,57\%$, bei zehn Iterationen $19,45\%$ und bei 25 Iterationen $19,4\%$.

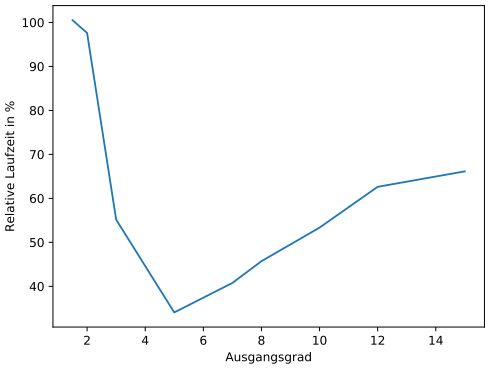
Lag hingegen die Laufzeit bei drei Wiederholungen bei den Graphen mit 5000 Knoten bei 22 Minuten und 33 Minuten bei fünf Wiederholungen, so betrug sie bei 25 Wiederholungen 2:45 Stunden. Für die bei 5000 Knoten um $0,17\%$ kleinere FAS bei 25 Iterationen gegenüber der mit 5 Iterationen, benötigte SCCEdgePRItFAS(25) die fünffache Laufzeit.

Auch in Abhängigkeit zum Ausgangsgrad ist die FAS bei einem Ausgangsgrad von 15 mit 25 Iterationen lediglich um $0,03\%$ kleiner als die mit fünf und auch das Verhältnis der Laufzeiten ist ähnlich.

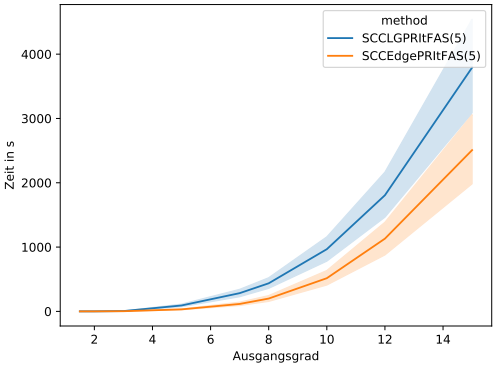
Die Ergebnisse sind graphisch in Abbildung 4.2 dargestellt.



(a) Relative Laufzeit nach Anzahl der Knoten.

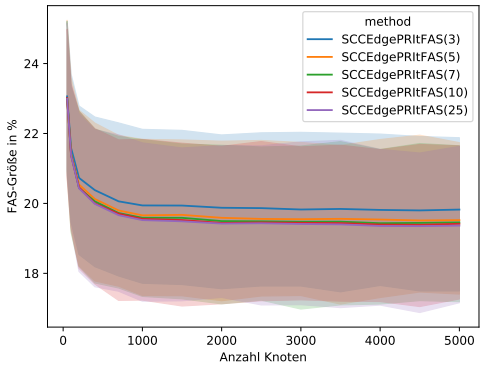


(b) Relative Laufzeit nach Ausgangsgrad.

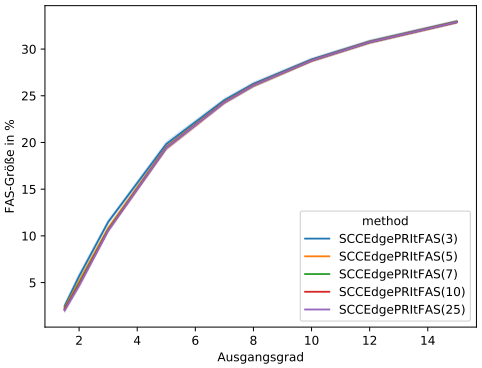


(c) Durchschnittliche Laufzeiten nach Ausgangsgrad.

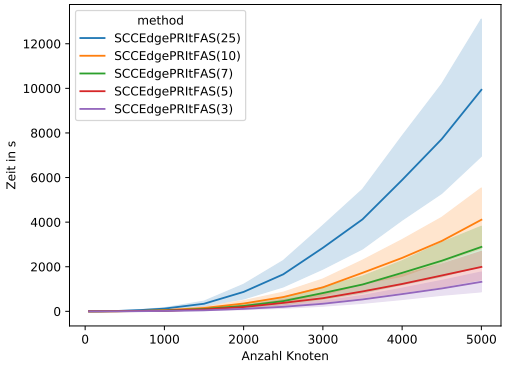
Abbildung 4.1: Laufzeiten von PageRankFAS mit und ohne Line Graph.



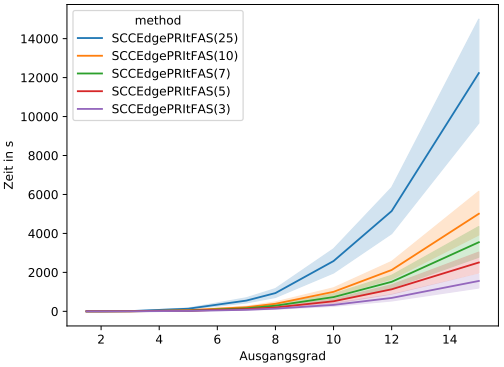
(a) FAS-Größe mit verschiedenen Iterationen bei zunehmender Knotenanzahl.



(b) FAS-Größe mit verschiedenen Iterationen bei zunehmendem Ausgangsgrad.



(c) Laufzeit mit verschiedenen Iterationen bei zunehmender Knotenanzahl.



(d) Laufzeit mit verschiedenen Iterationen bei zunehmendem Ausgangsgrad.

Abbildung 4.2: Auswirkungen von länger laufendem PageRank bei SCCEdgePRItFAS.

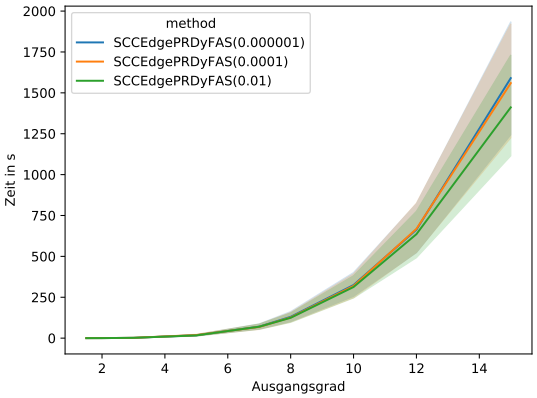


Abbildung 4.3: Laufzeiten von SCCEdgePRDyFAS.

4.3 Dynamisches PageRank

Für SCCEdgePRItFAS wird vor der Ausführung die Anzahl der Iterationen festgelegt, in der die PageRank-Scores konvergieren sollten. Dadurch können in einfachen Schleifen, wie beispielsweise das SCC des Beispielgraphen aus Kapitel 2, die Scores der Knoten nur weitergereicht werden, ohne dass diese weiter konvergieren würden. Ebenso hat sich in Abschnitt 4.2 gezeigt, dass mit mehreren Iterationen eine kleinere FAS erreicht werden kann, wenn die Scores noch länger konvergieren können. Deshalb wurde *SCCEdgePRDyFAS* implementiert, das wie SCCEdgePRItFAS funktioniert. Der einzige Unterschied hierzu ist, dass nicht eine vorher festgelegte Zahl entscheidend ist, ob eine weitere Iteration lang PageRank ausgeführt wird, sondern ein Grenzwert, um den sich der maximale Score verändert haben muss. Dazu wird der maximale Score der Knoten in der letzten Runde mit dem maximalen Score der vorletzten Runde verglichen. Wenn sich dieser um mehr als den Grenzwert vergrößert hat, wird PageRank eine weitere Runde ausgeführt. Dadurch sollte bei einfachen Kreisen vorzeitig abgebrochen werden und gleichzeitig die Präzision erhöht werden, wenn sich die Werte noch stark verändern.

Um diese Annahme zu überprüfen, wurde SCCEdgePRDyFAS mit den Grenzwerten $0,01$, $0,0001$ und $0,000001$ auf den Testgraphen ausgeführt. Der Grenzwert $0,01$ entspricht einer Änderung von 1%.

Sowohl nach Ausgangsgrad als auch nach der Anzahl der Knoten gruppiert zeigt sich, dass die FAS mit kleiner werdendem Grenzwert auch kleiner wird. Gruppiert nach Knoten ist der Unterschied zwischen SCCEdgePRDyFAS$(0,01)$ und SCCEdgePRDyFAS$(0,000001)$ aber kleiner als $0,1\%$, gruppiert nach Ausgangsgrad bei Zwei um $0,6\%$, ansonsten aber auch kleiner als $0,15\%$.

Auch die Laufzeit ist überwiegend identisch. Sie nimmt zwar mit größer werdendem Ausgangsgrad zu, macht sich aber nur bei dichteren Graphen bemerkbar. Bei einem Ausgangsgrad von 15 erhöht sie sich von 24 Minuten bei dem Grenzwert von $0,01$ auf 26 Minuten bei $0,0001$ und auf $26,5$ Minuten bei $0,000001$. Gegenüber dem Grenzwert von $0,01$ entspricht dies einer Zunahme von $10,5\%$ bzw. $12,7\%$ (siehe Abbildung 4.3).

Die Größe der FAS und die Laufzeit lässt sich somit bei SCCEdgePRDyFAS über die Größe des Grenzwertes steuern. Die FAS-Größe konvergiert jedoch sehr schnell, sodass sie sich nicht beliebig verkleinern lässt.

4.4 PageRank mit Dämpfungsfaktor

Laut Page et al. (1999) ist die Rank Source nicht nur notwendig, um Rank Sinks auszugleichen, sondern ist auch ein mächtiger Parameter bei der Berechnung des PageRank-Scores [PBMW99]. Um deren Einfluss in diesem Anwendungsfall zu überprüfen, wurde der Algorithmus *SCCLGPRDampItFAS* implementiert, eine Abwandlung von SCCLGPRItFAS. Dieser verwendet nicht das vereinfachte PageRank, sondern arbeitet mit einem Dämpfungsfaktor. Hierzu wird in jeder Iteration, in der die PageRank-Scores geupdatet werden, jedem Knoten im Line Graph 85% des errechneten Wertes zugewiesen und die restlichen 15% des globalen Scores auf alle Knoten gleich verteilt.

Betrachtet man die FAS-Größe in Abhängigkeit zur Anzahl der Knoten, so zeigt sich, dass die FAS von SCCLGPRDampItFAS(5) um ca. $1,3\%$ größer ist als die von SCCLGPRItFAS(5) und auch in Abhängigkeit zum Ausgangsgrad ist die FAS von SCCLGPRDampItFAS(5) zwischen $0,13\%$ bei einem Ausgangsgrad von $1,5$ und $3,0\%$ bei einem Ausgangsgrad von Drei größer. Bei den restlichen Graden ist die FAS zwischen $0,65\%$ und $1,9\%$ größer.

Bei der Laufzeit gibt es kaum einen Unterschied, lediglich bei sehr großer Knotenanzahl oder sehr hohem Ausgangsgrad. So braucht SCCLGPRDampItFAS(5) bei 5000 Knoten durchschnittlich 4% mehr Zeit und bei einem Ausgangsgrad von 15 $1,8\%$. Die größere Laufzeit lässt sich auf eine größere FAS zurückführen, weshalb öfter auf dem verbleibenden Graphen eine zu entfernende Kante bestimmt werden muss, bis der Graph azyklisch ist.

Somit bringt in diesem Anwendungsfall die Verwendung einer Rank Source keine Verbesserung. Das PageRank mit Dämpfungsfaktor konvergiert langsamer, weshalb die FAS größer und in Folge die Laufzeit schlechter ist.

4.5 Vergleich aller FAS-Heuristiken

Wir haben uns in den vorherigen Abschnitten und in Kapitel 3 mit den Auswirkungen einzelner Parameter bei PageRank, der Reduzierung auf die SCCs bei GreedyFAS und SortFAS* sowie mit der Verwendung verschiedener Zentralitätsmaße beschäftigt. In diesem Abschnitt werden nun die verschiedenen Versionen miteinander Verglichen.

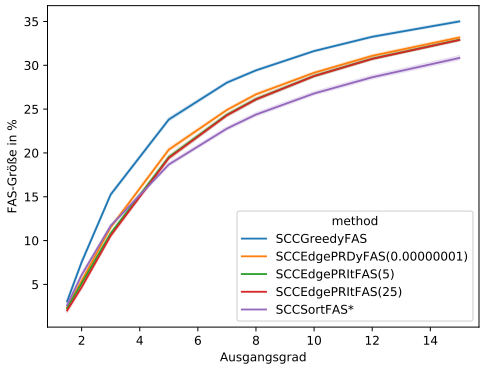
Durch mehrere Iterationen konnte der geringe Unterschied bei großen kargen Graphen von SCCBetweennessFAS gegenüber SCCEdgePRIt(5) bei 25 Iterationen nochmals halbiert werden, mit der Laufzeit bleibt dabei SCCEdgePRIt weiterhin weit zurück.

Bei den Versionen mit PageRank schneidet bei der Größe der FAS der dynamische Ansatz schlechter ab als der mit festgelegter Anzahl an Iterationen, bei der Laufzeit hingegen etwas besser. Dies kann aber durch Anpassen der Iterationen ausgeglichen werden, sodass eine Anpassung über die Iterationen sich als das mächtigere Werkzeug herausstellte. Denn auch wenn sich mit kleiner werdendem Grenzwert die FAS noch verkleinert, so ist, gruppiert nach der Knotenanzahl, die FAS von SCCEdgePRDyFAS(0.00000001) mit ca. 20% um etwa $0,5\%$ größer als die von SCCEdgePRItFAS(5) mit ca. $19,5\%$.

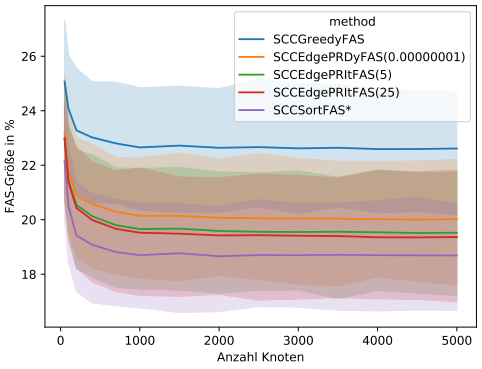
Auch bei einer Gruppierung nach den Ausgangsgraden zeigt sich ein ähnliches Bild. Bei einem Ausgangsgrad von Drei ist der relative Unterschied bei den FAS-Größen mit $6,4\%$ am größten. Dieser nimmt aber wieder ab, sodass der absolute Unterschied bei einem Ausgangsgrad von 15 $0,27\%$ bei ca. 33% FAS-Größe beträgt.

Betrachtet man jedoch die Laufzeit gruppiert nach dem Ausgangsgrad, so benötigt SCCEdgePRItFAS(5) bei 15 75% länger, sie liegt bei 42 Minuten gegenüber 24 Minuten bei SCCEdgePRDyFAS(0.00000001).

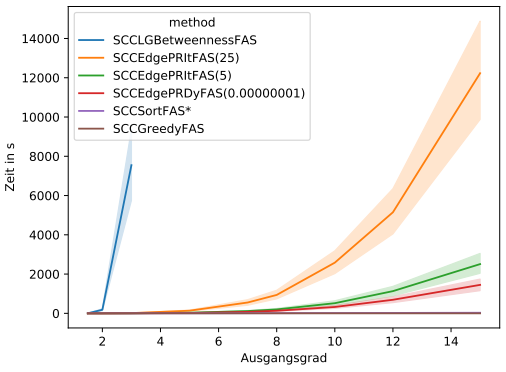
SCCEdgePRDyFAS berechnet also mit unseren Grenzwerten eine etwas größere FAS als SCCEdgePRItFAS(5), jedoch mit besserer Laufzeit.



(a) Durchschnittliche FAS-Größen bei zunehmendem Ausgangsgrad.



(b) Durchschnittliche FAS-Größen bei zunehmender Knotenanzahl.



(c) Durchschnittliche Laufzeit bei zunehmendem Ausgangsgrad.

Abbildung 4.4: Vergleich der unterschiedlichen Ansätze.

Das Verhalten der zentralitätsmaßbasierten FAS-Heuristiken kann also angepasst werden, jedoch haben all unsere Versionen eine besser FAS als GreedyFAS und eine bei dichteren Graphen schlechtere FAS als $SCCSortFAS^*$. Die Laufzeit von GreedyFAS ist weiterhin am besten, gefolgt von $SCCSortFAS^*$. Die zentralitätsmaßbasierten Heuristiken schneiden dabei am schlechtesten ab. Anschaulich dargestellt sind diese Vergleiche in Abbildund 4.4.

5. Zusammenfassung

In dieser Arbeit betrachteten wir die Idee von Geladaris et al. [GLT22], ein Zentralitätsmaß zur Berechnung einer FAS zu verwenden, genauer. Dazu verglichen wir Betweenness-, Closeness- und PageRank-Zentralität auf ihre Eignung bei diesem Anwendungsfall. Dabei zeigte sich, dass die Betweenness-Zentralität für größere Graphen mit sehr kleinem Ausgangsgrad eine kleinere FAS wie PageRank lieferte, jedoch bei hohem Zeitaufwand. Closeness-Zentralität hingegen benötigte nicht nur deutlich mehr Zeit, sondern berechnete auch größere Feedback Arc Sets als PageRank, weshalb PageRank sich als das am besten geeignete Zentralitätsmaß herausstellte.

Im Weiteren wurde deshalb versucht, den Ansatz von Geladaris et al. zu verbessern. Es wurde eine Version von PageRank vorgestellt, das die Kanten eines Graphen bewertet. Dadurch konnte auf das Erstellen eines Line Graph verzichtet werden und die Laufzeit je nach Graphen auf etwa ein Viertel der Laufzeit des ursprünglichen Algorithmus reduziert werden. Es wurden weitere Parameter getestet, um die FAS weiter zu verkleinern und die Laufzeit zu reduzieren. Page et al. (1999) erkannte, dass ein Dämpfungsfaktor ein mächtiger Parameter bei der Berechnung des PageRank-Scores sein kann [PBMW99]. Wir konnten aber zeigen, dass in diesem speziellen Anwendungsfall die Scores langsamer konvergieren und somit sich die FAS und die Laufzeit verschlechtern, weshalb hier eine vereinfachte Version zu bevorzugen ist.

Auch wurde eine dynamisch Version von PageRank getestet, welche bei der Laufzeit der von PageRank mit statischer Anzahl an Iterationen überlegen war, jedoch kann bei dieser der Grenzwert nicht so angepasst werden, dass sie eine kleinere FAS für unsere Beispielgraphen als die statische Version liefert.

Das beste Mittel, mit dem PageRank-Ansatz eine noch kleinere FAS zu berechnen, ist ein mehrfaches Ausführen der PageRank-Routine. Daher haben wir die Auswirkungen auf die FAS-Größe und die Laufzeit bei weiteren Wiederholungen untersucht. Dies erfordert jedoch eine Abwägung von einer etwas kleineren FAS und längerer Laufzeit.

Nebenbei konnten wir GreedyFAS und SortFAS durch eine Reduzierung auf die starken Zusammenhangskomponenten für karge Graphen, sowohl die Laufzeit, als auch die FAS-Größe betreffend, optimieren.

Es ist aber bei unseren Graphen mit dem PageRank-Ansatz nicht möglich, ab einem durchschnittlichen Ausgangsgrad von etwa Fünf, eine kleinere FAS als die von SCCSortFAS* zu erhalten. Somit sind bei großen und dichten Graphen Heuristiken wie GreedyFAS oder

SortFAS nicht zuletzt wegen der Laufzeit, besser geeignet. Bei kleineren Graphen hingegen sind die zentralitätsmaßbasierten FAS-Heuristiken, wenn das Augenmerk auf der FAS-Größe liegt, zu bevorzugen.

Möglicherweise ließe sich die Laufzeit noch weiter verbessern, wenn mehrere entfernt liegende Kanten aus dem Graphen gleichzeitig entfernt würden.

Literaturverzeichnis

[BH11] Franz J Brandenburg und Kathrin Hanauer: *Sorting heuristics for the feedback arc set problem*, 2011.

[BKN12] Ulrik Brandes, Sven Kosub und Bobo Nick: *Was messen Zentralitätsindizes?* Die integration von theorie und methode in der netzwerkforschung, Seiten 33–52, 2012.

[Bra01] Ulrik Brandes: *A faster algorithm for betweenness centrality.* Journal of mathematical sociology, 25(2):163–177, 2001.

[CGJ+13] Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W Klau, Karsten Klein und Petra Mutzel: *The Open Graph Drawing Framework (OGDF).* Handbook of graph drawing and visualization, 2011:543–569, 2013.

[GGKF14] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra und Christos Faloutsos: *Linearized and single-pass belief propagation.* arXiv preprint arXiv:1406.7288, 2014.

[GLT22] Vasileios Geladaris, Panagiotis Lionakis und Ioannis G. Tollis: *Computing a Feedback Arc Set Using PageRank*, 2022. `https://arxiv.org/abs/2208.09234`.

[GPP03] Raffaella Gentilini, Carla Piazza und Alberto Policriti: *Computing strongly connected components in a linear number of symbolic steps.* In: *SODA*, Band 3, Seiten 573–582, 2003.

[GYZ14] JL Gross, J Yellen und P Zhang: *Handbook of graph Theory Second Edition CRC Press Taylor and Francis Group.* 2014.

[Kar10] Richard M Karp: *Reducibility among combinatorial problems.* Springer, 2010.

[KGKK+20] Andreas Klärner, Markus Gamper, Sylvia Keim-Klärner, Irene Moor, Holger Von der Lippe und Nico Vonneilich: *Soziale Netzwerke und gesundheitliche Ungleichheiten: Eine neue Perspektive für die Forschung.* Springer Nature, 2020.

[Mut10] Peter Mutschke: *Zentralitäts-und Prestigemaße.* Springer, 2010.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani und Terry Winograd: *The PageRank citation ranking: Bringing order to the web.* Technischer Bericht, Stanford InfoLab, 1999.

[SST16a] Michael Simpson, Venkatesh Srinivasan und Alex Thomo: *Clearing contamination in large networks.* IEEE Transactions on Knowledge and Data Engineering, 28(6):1435–1448, 2016.

[SST16b] Michael Simpson, Venkatesh Srinivasan und Alex Thomo: *Efficient computation of feedback arc set at web-scale.* Proceedings of the VLDB Endowment, 10(3):133–144, 2016.

[STT81] Kozo Sugiyama, Shojiro Tagawa und Mitsuhiko Toda: *Methods for visual understanding of hierarchical system structures.* IEEE Transactions on Systems, Man, and Cybernetics, 11(2):109–125, 1981.

[THS+11] Mark Trappmann, Hans J Hummell, Wolfgang Sodeur, Mark Trappmann, Hans J Hummell und Wolfgang Sodeur: *Prestige, Zentralität und Zentralisierung.* Strukturanalyse sozialer Netzwerke: Konzepte, Modelle, Methoden, Seiten 27–72, 2011.