INSTITUT FÜR INFORMATIK
Algorithmische Bioinformatik

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Aggregieren von gerichteten azyklischen Graphen unter Verwendung von ganzzahliger linearer Programmierung

Max Oerter

Bachelorarbeit

Beginn der Arbeit:	15. Mai 2020
Abgabe der Arbeit:	17. August 2020
Gutachter:	Prof. Dr. Gunnar W. Klau
	Prof. Dr. Stefan Dietze

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 17. August 2020

Max Oerter

Zusammenfassung

In der realen Welt lässt sich die Beziehung zwischen Daten häufig gut als Rangfolge darstellen, beispielsweise die Ergebnisse einer Suchmaschine bezüglich ihrer Relevanz oder den Präferenzen eines Nutzers. Hierbei sind aber nur sehr selten alle zur Verfügung stehenden Daten ein Teil der Rangfolge oder es fehlen Informationen über einige Beziehungen, weshalb sich die Darstellung in Form eines gerichteten azyklischen Graphen (DAGs) anbietet, gegenüber der einer kompletten Rangliste über alle Daten. Häufig ergibt sich damit dann das Problem, mehrere DAGs zu einem einzigen neuen DAG zusammenfassen zu müssen. Dabei sollen möglichst viele Informationen erhalten bleiben und auch widersprüchliche Beziehungen in den ursprünglichen Graphen berücksichtigt werden. Dieses Problem ist im allgemeinen von Computern nicht effizient lösbar und ist somit NP-schwer.

In dem Artikel „Beyond rankings: comparing directed acyclic graphs“ von Eric Malmi und Gionis (2015) wurde nun eine Verallgemeinerung von Kendalls Tau, ein Maß für Korrelationen, das also die Verknüpfung zwischen zwei Variablen misst, dafür benutzen, den Zusammenhang zwischen DAGs zu ermitteln. Voraussetzung dafür ist, dass die Beziehung zwischen den Knoten bekannt ist. Außerdem wurde ein einfacher, approximativer Algorithmus für das Aggregieren von DAGs vorgestellt (Algorithmus 2), welcher einfach der Reihe nach die, dem Zusammenhangsmaß nach, besten Kanten in den finalen Graphen aufnimmt, solange diese nicht mit den Eigenschaften eines DAGs im Konflikt stehen.

In dieser Arbeit soll nun das Aggregieren von gerichteten azyklischen Graphen als ganzzahliges lineares Programm (ILP) formuliert werden. Das Ziel dabei ist es, zu untersuchen, bis zu welcher Instanzgröße das Problem sinnvoll mit einem ILP gelöst werden kann. Außerdem wird der Algorithmus von Eric Malmi mit dem ILP verglichen, hinsichtlich Effizienz, Geschwindigkeit und der Qualität des resultierendem Graphen. Dabei soll vor allem festgestellt werden, welche Vor- und Nachteile eine Implementierung als ILP mit sich bringt und ob sich daher der, mit einem ILP verbundene, Aufwand lohnt.

Inhaltsverzeichnis

1 Einleitung

Die Beziehungen zwischen Daten lässt sich in der Realität oft gut in einer Art Rangfolge darstellen. Da die zur Verfügung stehenden Daten aber nicht immer vollständig sind oder man schlichtweg nicht die Beziehung zwischen sämtlichen Daten kennt, ist eine (vollständige) Rangliste nicht immer das geeignete Mittel. Hier kommen nun gerichtete azyklische Graphen (DAGs) ins Spiel, da man diese als eine Verallgemeinerung von Ranglisten betrachten kann. Diese bieten durch ihre Kreisfreiheit auch die Möglichkeit, eine Rangfolge über die Daten darzustellen, allerdings ist hier das Auslassen von Informationen kein Problem. Folglich ist es daher interessant, DAGs miteinander vergleichen und aggregieren zu können.

Das Ziel beim Aggregieren von gerichteten azyklischen Graphen ist es, mehrere DAGs zu einem einzigen, zentrierten DAG zusammen zu fassen und dabei möglichst viele Informationen zu erhalten, aber auch widersprüchliche Beziehungen zwischen Knoten zu berücksichtigen. Als Voraussetzung dafür wird gefordert, dass die Beziehung zwischen den Knoten, der zu aggregierenden DAGs, bekannt ist, also das man jedem Knoten aus einem Graphen genau einem anderen Knoten aus jedem anderen Graphen zuordnen kann.

1.1 Anwendungsbeispiele

1.1.1 Informationsverbreitung

Die Verbreitung von Informationen in sozialen Netzwerken kann gut in Form von DAGs dargestellt werden, wo es dann relevant ist, diese miteinander zu vergleichen und zu aggregieren (Eric Malmi und Gionis, 2015). Führt ein Knoten (Account) eine Aktion aus, so führen einige benachbarte Knoten ähnliche Aktionen durch, da diese beispielsweise den veröffentlichten Beitrag teilen. Beobachtet man diese Verbreitung von Aktionen, so kann man diese in Form einer unvollständigen Rangliste darstellen. Nicht alle Teilnehmer des Netzwerks oder einer Gruppe nehmen daran teil und auch nicht alle Beziehungen zwischen den Mitgliedern, bei Verbreitung dieser Information, sind bekannt. Um nun den Einfluss bestimmter Nutzer auf andere Nutzer zu untersuchen oder die Art und Weise wie Informationen in sozialen Netzwerken verbreitet werden, ist es sinnvoll, mehrere solcher Ranglisten zusammen zu fassen und miteinander vergleichen zu können.

Auch bei Wahlen oder in der Werbeindustrie spielt die Verbreitung und der Einfluss verschiedener Teilnehmer eine große Rolle und kann auf ähnliche Weise dargestellt werden. Allerdings ist es hier schwieriger an möglichst viele der relevanten Daten zu kommen.

1.1.2 Identifikation relevanter Merkmale

Auch in der Bioinformatik kann das Aggregieren von DAGs eine Rolle spielen (Li et al., 2017). In der Biologie fallen bei der Untersuchung eines Problems oft riesige Mengen genetischer Daten, aus unterschiedlichen Forschungseinrichtungen, an. Diese Daten sind oft nicht einheitlich, weisen viele kleinere Unterschiede auf, entweder weil sie von Natur aus eine gewisse Variation aufweisen oder da die verschiedenen Labore leicht Unterschiede in den Methoden haben, mit denen sie an die Daten kommen. Auch können verschiedene Methoden zur Datengewinnung genutzt werden und so können die Daten oft nicht direkt miteinander verglichen werden. Um diese Daten nun zusammenzufassen und beziehungsweise neue Daten zu integrieren, werden oft Metaanalysen basierend auf Ranglisten genutzt. Da die Daten hier aber auch unvollständig seien können oder nicht immer vollständigen Ranglisten ergeben müssen, kann es auch hier Vorteilhaft sein, DAGs zur Darstellung zu nutzen. Generell können DAGs auch immer dort verwenden werden, wo bisher Ranglisten genutzt wurden und eine Verallgemeinerung Sinn macht.

2 Graphentheorie

Dieser Abschnitt startet mit den grundlegenden Begriffen der Graphentheorie, die im weiteren Verlauf dieser Arbeit benötigt werden und führt die Notation ein, die für den Rest dieser Arbeit benutzt wird. Im Anschluss wird erläutert wie DAGs miteinander verglichen werden können.

2.1 Definitionen

Gerichteter Graph

Ein gerichteter Graph $G = (V, E)$ besteht aus einer Menge von Knoten V und einer Menge von gerichteten Kanten E. Eine gerichtete Kante $e \in$ E besteht dabei aus einem geordnetem Paar $(i, j) \in V \times V$ mit $i \neq j$. Zwischen jedem Knotenpaar darf es nur eine der beiden möglichen Kanten geben.

Gerichteter azyklischer Graph

Ein gerichteter azyklischer Graph (DAG) ist ein gerichteter Graph, welcher keine gerichteten Kreise enthält.

(Vollständige) Rangliste

Eine Rangliste ist ein gerichteter kreisfreier Graph in dem es eine feste Ordnung über alle Knoten gibt, sodass nur Kanten (i, j) existieren, wobei der Knoten i vor

dem Knoten j in der Ordnung auftritt. Eine Rangliste ist vollständig, wenn es zwischen jedem Knotenpaar $\{i, j\} \in V$ eine der beiden möglichen Kanten gibt.

Kendall-Tau

Kendall-Tau ist ein Distanzmaß für vollständige Ranglisten, mit dem der Grad der Gemeinsamkeit zwischen den beiden Graphen bestimmt werden kann.

Die Kendall-Tau Distanz $K(G_1, G_2)$ zwischen den beiden Graphen G_1 und G_2 berechnet sich über die Summe der Knotenpaare die nicht miteinander übereinstimmen. Ein Knotenpaar stimmt nicht überein, ist diskordant, wenn in einem der beiden Graphen die Kante (i, j) (bzw. (j, i)) existiert und in dem anderen die Kante (j, i) (bzw. (i, j)). Stimmt das Knotenpaar hingegen überein, also gibt es in beiden Graphen die Kante (i, j) (oder (j, i)), so handelt es sich um ein konkordantes Knotenpaar.

2.2 Distanzbestimmung in DAGs

Um nun mehrere DAGs zu einem einzigen zusammen fassen zu können, muss bestimmt werden, wie stark sich die Graphen jeweils unterscheiden. Grundsätzlich sind DAGs nun Ranglisten sehr ähnlich, da es auch hier keine Kreise gibt. DAGs können daher auch als Verallgemeinerung von Ranglisten betrachtet werden. Für Ranglisten ist nun Kendall-Tau als Distanzmaß bekannt, da es aber nicht zwangsweise eine Kante zwischen jedem Knotenpaar in einem DAG gibt, kann Kendall-Tau in dieser Form leider nicht verwendet werden. In Eric Malmi und Gionis (2015) wurde hierfür nun eine Verallgemeinerung von Kendall-Tau vorgestellt, welche ebenfalls in dieser Arbeit verwendet und im folgenden erklärt wird:

Ohne Beschränkung der Allgemeinheit kann angenommen werden, dass alle DAGs über die gleiche Menge an Knoten verfügen. Sollte einer der DAGs einen Knoten nicht enthalten, so kann er einfach diesem Graphen einzeln hinzugefügt werden, also ohne dass neue Kanten hinzukommen. Das heißt insbesondere, dass die Beziehung der Knoten zwischen den verschiedenen DAGs bekannt ist, also jeder Knoten aus einem DAG kann eindeutig einem Knoten aus jedem anderem DAG zugeordnet werden. Genau wie die bereits bekannte Kendall-Tau Distanz wird auch die verallgemeinerte Variante über Paare von Knoten definiert, mit dem Unterschied, dass bei DAGs nicht zwangsläufig eine klare Aussage über das Verhältnis der beiden Knoten getroffen werden kann. Ist in einem oder beiden Graphen keine Kante zwischen zwei Knoten vorhanden, so ist diese weder vollständig konkordant noch diskordant. Diese Fälle müssen also mit berücksichtigt werden und sollten schwächer zur Distanz beitragen, als ein eindeutig diskordantes Paar. Um nun den Grad der Übereinstimmung zweier Kanten zu benennen, werden zwei

zusätzliche Parameter $0 \leq p, q \leq 1$ benötigt. Außerdem wird für die Definition noch eine Ordnung über die Kanten benötigt, welche aber willkürlich gewählt werden kann, also insbesondere nichts mit der Ordnung einer Rangliste zu tun hat.

Sei nun $G_1 = (V, E_1)$ und $G_2 = (V, E_2)$ zwei DAGs und sei $0 \leq p, q \leq 1$ zwei feste Parameter. Sei weiterhin i und j zwei Knoten in V, wobei $i < j$ in der gewählten Ordnung, und die Kanten $e = (i, j)$ und $f = (j, i)$.

Definition: Sei $K_{i,j}^{p,q}(G_1, G_2)$ die Distanz bezüglich der beiden Knoten i und j in den DAGs G_1 und G_2 unter Berücksichtigung der beiden Parameter p und q. Um die Distanz zu bestimmen muss zwischen vier Fällen unterschieden werden:

Fall 1: $e \in E_1$ und $e \in E_2$ oder $f \in E_1$ und $f \in E_2$. In diesem Fall stimmen G_1 und G_2 bezüglich i und j überein und die Distanz wird auf $K_{i,j}^{p,q}(G_1, G_2) = 0$ gesetzt.

Fall 2: $e \in E_1$ und $f \in E_2$ oder $f \in E_1$ und $e \in E_2$. In diesem Fall stimmen G_1 und G_2 bezüglich i und j nicht überein und die Distanz wird auf $K_{i,j}^{p,q}(G_1, G_2) = 1$ gesetzt.

Fall 3: $e \vee f \in E_1$ und $e, f \notin E_2$ oder $e, f \notin E_1$ und $e \vee f \in E_2$. In diesem Fall ist die Beziehung zwischen i und j nur in einem der beiden Graphen bekannt und die Distanz wird auf $K_{i,j}^{p,q}(G_1, G_2) = p$ gesetzt.

Fall 4: $e, f \notin E_1$ und $e, f \notin E_2$. In diesem Fall ist die Beziehung zwischen i und j in keinem der beiden Graphen bekannt und die Distanz wird auf $K_{i,j}^{p,q}(G_1, G_2) = q$ gesetzt.

Hierbei ist zu beachten, dass $q > 0$ gewählt werden sollte, da sonst zwei leere Graphen ($G = (V, E)$ mit $E = \{\}$) sich genau so stark unterscheiden, wie zwei DAGs, welche in jeder Kante übereinstimmen. Intuitiv ist außerdem schnell klar, dass $q \leq p$ gewählt werden sollte. Denn ist in beiden DAGs eine Kante nicht vorhanden, dann stimmt das Knotenpaar maximal genau so stark nicht übereinstimmt, wie wenn in einem der beiden Graphen eine der beiden möglichen Kanten vorhanden ist. Für das Aggregieren von DAGs würde $p > q$ zudem bedeuten, dass optimaler Weise der resultierende DAG ein vollständiger Graph ist.

Da nun die Beziehung zwischen zwei Knoten dargestellt werden kann, wird die Kendall-Tau Distanz zwischen zwei DAGs wie folgt definiert:

Seien weiterhin $G_1 = (V, E_1)$ und $G_2 = (V, E_2)$ zwei DAGs und sei $0 < p \leq q \leq 1$. Dann ist die Distanz zwischen den beiden DAGs $K^{p,q}(G_1, G_2)$ die Summe über alle Knotenpaare $(i, j) \in V \times V$, sodass $i < j$.

$$K^{p,q}(G_1, G_2) = \sum_{\substack{(i,j) \in V \times V \\ i<j}} K_{i,j}^{p,q}(G_1, G_2)$$

Sollte es sich bei den beiden Graphen um eine Rangliste handeln, so reduziert sich die Distanzbestimmung zum klassischen Kendall-Tau.

Beispiel 1.

Abbildung 1: Graphen für Beispiel 1

Im folgenden soll nun die Distanz $K^{p,q}(G_1, G_2)$ zwischen den Graphen G_1 und G_2 aus Abbildung 1 berechnet werden. Zwischen den Graphen gibt es ein konkordantes Knotenpaar, da die Kante (a, b) in beiden Graphen vorhanden ist und ein diskordantes Knotenpaar, da in G_1 die Kante (a, c) und in G_2 die Kante (c, a) vorhanden ist. Außerdem gibt es zwei Knotenpaare vom Fall 3, nämlich (b, c) und (b, d). Die restlichen sind alle von Fall 4, da hier keine entsprechenden Kanten in den beiden Graphen vorhanden sind. Damit ergibt sich die Kendall-Tau Distanz zu:

$$K^{p,q}(G_1, G_2) = 0 + 1 + 2p + 2q$$

3 Aggregieren von DAGs

Das folgende Kapitel beschäftigt sich nun mit dem Aggregieren von DAGs. Das Problem hierbei besteht darin, dass man mehrere DAGs übergeben bekommt und diese zu einem einzigen DAG zusammenfassen möchte. Als Voraussetzung gilt nach wie vor, dass alle gegeben Graphen über die selbe Knotenmenge verfügen. Das Ziel ist es, möglichst viele relevante Informationen, also Kanten die in mehreren DAGs vorhanden sind, zu erhalten. Beide hier vorgestellten Lösungen dieses Problems bauen dabei auf dem zuvor aufgestellte Distanzmaß auf. Das hat zur Folge, dass beide Methoden auch etwaige Vor- und Nachteile der Distanzbestimmung mittels Kendall-Tau haben. Diese hat Eric Malmi und Gionis (2015) in seiner Arbeit genauer untersucht. Insgesamt lässt sich daraus aber festhalten, dass Kendall-Tau gut geeignet ist als Maß für die Distanz zwischen DAGs.

Das Problem wird nun zunächst formal definiert und es wird gezeigt, wie man dies mit dem vorgestelltem Distanzmaß erreichen kann. Danach werden die beiden Lösungsmethoden vorgestellt, einmal die Variante als ILP und ein einfacherer gieriger Algorithmus.

3.1 Problemdefinition

Gegeben sei eine Menge M von DAGs $G_1, ..., G_M$ und die zwei Parameter p und q. Gesucht ist der DAG $C = (V, E)$, der die Summe

$$\sum_{m=1}^{M} K^{p,q}(G_m, C)$$

minimiert.

Um dies zu erreichen sei zuerst $b(i, j)$ die Kosten dafür, (i, j) nicht in unserem finalen DAG C zu haben. Dafür sei H_{leer} ein Graph ohne Kanten und

$$b(i, j) = \sum_{m=1}^{M} K_{i,j}^{p,q}(G_m, H_{leer}).$$

Analog dazu sei $w(i, j)$ die Kosten dafür, die Kante (i, j) in C zu haben, mit

$$w(i, j) = \sum_{m=1}^{M} K_{i,j}^{p,q}(G_m, H_{komplett}).$$

$H_{komplett}$ sei hier ein gerichteter Graph, welcher alle möglichen Kanten enthält. Für jedes mögliche Knotenpaar $(i, j) \in V \times V, i \neq j$ gibt es also eine gerichtete Kante.

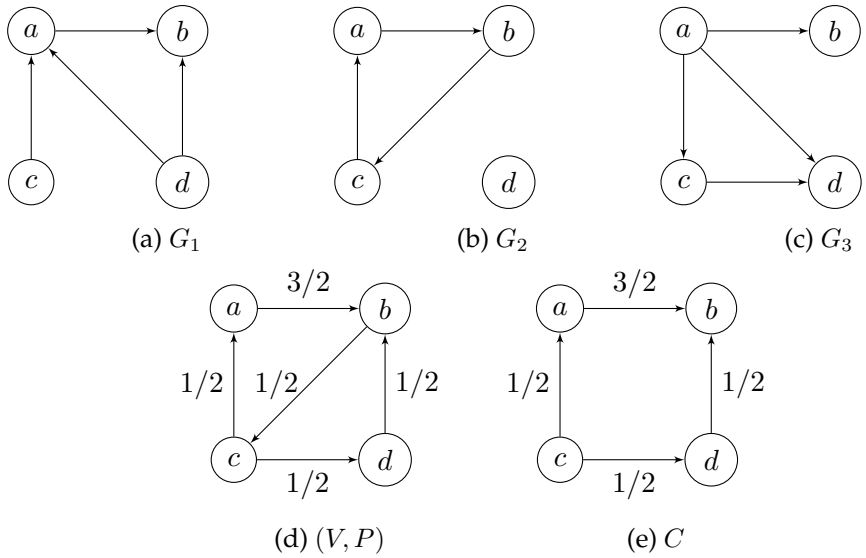
Damit ergibt sich insgesamt:

$$\sum_{m=1}^{M} K^{p,q}(G_m, C) = \sum_{(i,j) \notin E} b(i, j) + \sum_{(i,j) \in E} w(i, j)$$

Idealerweise sollen nun natürlich genau die Kanten in unserem finalen DAG C enthalten sein, für die es günstiger ist sie aufzunehmen, als es nicht zu tun. Dies entspricht der Menge an Kanten P mit $P = \{(i, j) \in V \times V \mid w(i, j) < b(i, j)\}$. Da in P aber möglicherweise Kreise enthalten sind, kann man nicht unbedingt alle Kanten aus P nehmen. Folglich ist also die Menge der Kanten von C eine Teilmenge von P. Um nun eine Auswahl der Kanten vornehmen zu können, sei für jede Kante $(i, j) \in P \setminus E$ $r(i, j) = b(i, j) - w(i, j)$ der Regret, also der Kostenunterschied der bezahlt werden muss, falls die Kante aus P nicht genommen werden kann. Dann lässt sich die Kostenfunktion letztendlich wie folgt definieren:

$$\sum_{m=1}^{M} K^{p,q}(G_m, C) = \sum_{(i,j) \notin P} b(i, j) + \sum_{(i,j) \in P} w(i, j) + \sum_{(i,j) \in P \setminus E} r(i, j)$$

Beispiel 2.



(a) G_1 (b) G_2 (c) G_3

(d) (V, P) (e) C

Abbildung 2: Graphen für Beispiel 2

Seien die Graphen G_1, G_2 und G_3 aus Abbildung 2 gegeben und die Parameter $p = 1/2$ und q = $1/3$. Um den optimalen DAG C zu berechnen, muss zuerst die Kantenmenge P bestimmt werden.

Dazu wird nun der Regret für jede Kante berechnet. Für die Kante (a, b) ergibt sich:

$$r(a, b) = 1/2 + 1/2 + 1/2 = 3/2$$

Der Regret für die Kante (c, a) ist:

$$r(c, a) = 1/2 + 1/2 + 1/2 - 0 - 0 - 1 = 1/2$$

Für $(b, c), (c, d)$ und (b, d):

$$r(b, c) = (c, d) = (d, b) = 1/4 + 1/2 + 1/4 - 1/4 - 0 - 1/4 = 1/2$$

Für die Kante (a, d) (und auch für (d, a)) folgt:

$$r(a, d) = 1/2 + 1/4 + 1/2 - 1 - 1/2 - 0 = -1/4$$

Der Regret ist hier negativ, somit werden diese Kanten nicht in P aufgenommen. Bei den restlichen Kanten ist der Regret ebenfalls negativ und die Kanten werden nicht genommen. Insgesamt erhält man so die Menge P und damit den Graphen (V, P), welcher ebenfalls in Abbildung 2 abgebildet ist. Der Graph (V, P) enthält aber zwei Kreise, also können nicht alle Kanten genommen werden. Bei diesem Beispiel ist nun noch relativ einfach zu erkennen, dass man den DAG C am besten durch löschen der Kante (b, c) erhält, da dadurch beide Kreise, mit minimalen Kosten, entfernt werden.

Um nun beliebig große Instanzen lösen zu können, besteht die größte Schwierigkeit dar-

in zu wählen, welche Kanten gelöscht werden. Für dieses Problem werden nun zwei Möglichkeiten vorgestellt:

3.2 Formulierung als ILP

Eine optimale Lösung für das DAG Aggregation Problem bietet eine Formulierung als ILP, auf welcher auch der Schwerpunkt dieser Arbeit liegt.

Sei dafür $P = \{(i,j) \in V \times V \mid w(i,j) < b(i,j)\}$ und der Regret $r(i,j) = b(i,j) - w(i,j)$ der Kostenunterschied der bezahlt werden muss, wenn man die Kante $(i,j) \in P$ nicht nimmt.

Definiere: Sei $x_{i,j} = 0$ wenn die Kante genommen wird, sonst $x_{i,j} = 1$.

$$min \sum_{(i,j) \in P} r(i,j) \cdot x_{i,j}$$

s.t.:

$$\sum_{(i,j) \in C} x_{i,j} \geq 1$$

für alle gerichteten Kreise C in (V, P).

3.3 Gieriger Algorithmus

In der Arbeit von Eric Malmi und Gionis (2015) wurde zudem ein deutlich einfacherer Algorithmus zum Vereinigen von DAGs vorgestellt. Dieser ordnet die Kanten in P aufsteigend nach ihrem Regreten und fügt diese dem finalen DAG C nacheinander zu. Dabei werden die Kanten, welche durch hinzufügen einen Kreis erzeugen würden, einfach ausgelassen. Wie sich bereits erahnen lässt, liefert diese Variante keine optimale Lösung. Dennoch soll dieser Algorithmus in der Praxis sehr effektiv sein; wie effektiv genau wird in Kapitel 5 untersucht.

3.4 Komplexität

Wie bereits angesprochen, besteht das Hauptproblem beim Aggregieren von DAGs darin, den resultierenden Graphen kreisfrei zu bekommen. Die Bestimmung der DAG Distanz und damit das Bilden der Menge P ist vergleichsweise einfach, da hierfür für jede Kante nur der Regret berechnet werden muss. Dieser kann in linearer Zeit bestimmt werden, damit hängt die benötigte Zeit insgesamt nur von der Anzahl der Kanten, beziehungsweise Knoten ab, was zu einer Komplexität von $\mathcal{O}(|V|^2)$ führt. Benötigt man daher

als Ergebnis keinen kreisfreien Graphen, so kann man das Problem in Polynomialzeit berechnen.

Soll der resultierende Graph nun ein DAG sein, so reduziert sich das DAG Aggregation Problem im wesentlichen auf eine Instanz des Feedback Arc Set Problems (FAS). Bei dem Feedback Arc Set Problem möchte man einen gegebenen Graphen, durch entfernen möglichst weniger Kanten, kreisfrei bekommen. FAS ist NP-vollständig, damit ist auch das DAG Aggregation Problem NP-schwer, wie in Eric Malmi und Gionis (2015) gezeigt wurde. Die Laufzeit des ILPs hängt nun hauptsächlich von der Anzahl der gerichteten Kreise in dem Graphen (V, P) ab. Es muss für jede Kante entschieden werden, ob diese genommen wird oder nicht, damit liegt die Laufzeit des ILPs in $\mathcal{O}(2^{|V|})$. Die Laufzeit des gierigen Algorithmus wird mit $\mathcal{O}(min(|V|^{3/2}, |P|^{1/2})|P|)$ angegeben.

4 Implementierung

Der gesamte Programmcode für diese Arbeit und auch die noch folgenden Experimente befindet sich auf GitLab unter `https://gitlab.cs.uni-duesseldorf.de/albi/albi-students/bachelor-max-oerter` (Zuletzt aufgerufen: 17.08.2020).

Alle Algorithmen für diese Arbeit wurde in Python 3.3 geschrieben und zum Lösen des ILPs wurde der Gurobi Optimizer in der Version 9.0 verwendet.

4.1 Benutzung

Der Programmcode dieser Arbeit ist auf mehrere Dateien aufgeteilt, die einzeln genutzt werden können oder für die Experimente im nächsten Kapitel gemeinsam. Die Datei „distanceMeasure“ enthält alle Funktionen zur Distanzbestimmung zwischen DAGs, wie zuvor vorgestellt. Die Bestimmung der Distanz zwischen einem DAG und einem leeren oder vollständigen Graphen wurde dabei, aus Geschwindigkeitsgründen, als eigenständige Funktion implementiert. Diese wird für die Bestimmung der Menge P benötigt (siehe Kapitel 3.1). Von der Datei „dag“ werden die Funktionen zum eigentlichen Lösen des DAG Aggregation Problems bereitgestellt. Einzeln ausgeführt werden die DAGs aus einer JSON-Datei eingelesen und beide vorgestellten Methoden zum Lösen des DAG Aggregation Problems ausgeführt. Die Ergebnisse werden im Anschluss in einer Tabelle in der Konsole angezeigt und der resultierende Graph kann angezeigt werden, mit farblichen Markierungen, welche Kanten jeweils gelöscht werden. Da bei der Lösung mittels ILP mitunter exponentiell viele Kreise entstehen können, werden hier lazy constraints verwendet. Diese fügen dann dynamisch weitere Nebenbedingungen für jeden neu gefundenen Kreis hinzu. Die Datei „randomDAGGenerator“ stellt zwei Methoden zur zu-

fälligen Erzeugung von DAGs bereit, wobei die dafür notwendigen Parameter über die Konsole eingegeben werden können, falls die Datei einzeln ausgeführt wird. Die Methoden zur Generierung von DAGs werden im nächsten Abschnitt vorgestellt. In der letzten Datei „experiments“ sind die in Kapitel 5 gezeigten Experimente enthalten. Zur Ausführung werden dabei alle zuvor genannten Dateien benötigt.

4.2 Datengenerierung

DAGs können mit zwei verschiedenen Methoden erstellt werden, komplett zufällig oder mit einem einstellbaren Grad an Gleichheit. Bei beiden kann die Anzahl an DAGs bestimmt werden und die Anzahl an Knoten pro Graph. Die Knoten können wie zuvor bereits angesprochen in jedem Graphen als dieselben angenommen werden.

Die erste Variante erstellt komplett zufällige DAGs, wobei noch die Wahrscheinlichkeit angeben werden kann, mit der eine Kante zwischen zwei Knoten generiert wird. Basierend auf der Tatsache, dass jede DAG als eine Liste aus seinen Knoten dargestellt werden kann, in der es nur Vorwärts- beziehungsweise Rückwärtskanten gibt, wird für jeden zu erzeugenden DAG eine zufällige Ordnung über alle Knoten erstellt, sodass nur Kanten (i, j) möglich sind, wo $i < j$. Eine Wahrscheinlichkeit von 0 entspricht dabei einem leeren Graphen, wobei eine von 100 einer vollständigen Rangliste entspricht.

Die zweite Variante basiert auf einer ähnlichen Methode wie der von Eric Malmi und Gionis (2015), da im Verlauf des nächsten Kapitels auch eines seiner Experimente nachgestellt wird, um die Qualität des resultierenden DAGs zu bewerten und die Ergebnisse zu vergleichen. Hier werden die DAGs nun basierend auf einem Schlüssel erzeugt, sodass alle erzeugten Graphen am Ende über ein bestimmte Ähnlichkeit verfügen. Wie stark sich die DAGs am Ende unterschieden, kann über zwei weitere Parameter angegeben werden. n_{swap} gibt die Anzahl an, wie oft zwei zufällige Knoten in dem aktuellen Graphen miteinander getauscht werden, sodass dann also alle zu einem Knoten inzidenten Kanten zu dem anderen Knoten gehören. Mit p_{change} wird die Wahrscheinlichkeit für jede Kante angegeben, mit der diese gelöscht und gleichzeitig eine neue Kante, an zufälliger Position, erzeugt wird. Das Löschen und Erzeugen von Kanten ist dabei unabhängig voneinander, also falls eine Kante gelöscht wird bedeutet dies nicht, dass dann auch zwangsweise eine neue hinzugefügt wird. Die Wahrscheinlichkeit zum Hinzufügen und Löschen einer Kante ist hier mit Absicht gleich groß, da so die DAGs am Ende im Mittel über die gleiche Anzahl an Kanten verfügen.

5 Auswertung

5.1 Vergleich

In diesem Abschnitt werden nun die beiden vorgestellten Versionen zum Aggregieren von DAGs, anhand von künstlich erstellten DAGs, miteinander verglichen.

In den folgenden Experimenten besitzen die generierten DAGs überwiegend eine hohe bis sehr hohe Dichte an Kanten, warum genau dies so gewählt wurde, wird sich im weiteren Verlauf dieser Arbeit noch zeigen. Aus diesem Grund werden nun die Parameter $p = 1/2$ und $q = 1/4$ festgelegt. Diese Werte haben sich nach den Experimenten von Eric Malmi und Gionis (2015) und auch in meinen eigenen als sinnvoll für dichte DAGs herausgestellt.

5.1.1 Instanzgröße

Zuerst soll nun untersucht werden, bis zu welcher Instanzgröße das ILP überhaupt sinnvoll verwendet werden kann. Für das Experiment werden komplett zufällig generierte Graphen verwendet und die Größe der Instanz wird einmal über die steigende Anzahl der Knoten und über die Chance, dass eine Kante generiert wird, geregelt. Die Anzahl der anfänglichen DAGs wird auf 10 gesetzt. Diese spielt beim Lösen des ILPs keine Rolle und nimmt nur Einfluss auf die Dauer, die zur Berechnung der Distanz benötigt wird. Mit der Kantenanzahl ist die Anzahl der Kanten in dem Graphen (V, P) gemeint, diese soll, in Verbindung mit der Knotenanzahl, eine bessere Bestimmung der Instanzgröße ermöglichen.

Chance = 40				Chance = 60		
Knotenanzahl	Zeit (s)	Kantenanzahl		Knotenanzahl	Zeit (s)	Kantenanzahl
100	0.054	433		100	2.430	1192
120	0.106	639		110	7.745	1429
140	0.342	888		120	12.942	1686
160	0.869	1205		130	35.097	1956
180	4.147	1501		140	64.377	2331
200	10.350	1792		150	218.171	2545
220	30.149	2151				
240	86.316	2681				
260	∼2400	2800				

Tabelle 1: Zufällige DAGs, Instanzgröße ILP

Die vom ILP benötigte Zeit scheint exponentiell anzusteigen, obwohl die Zunahme der Knoten linear verläuft. Auch die Anzahl der resultierenden Kanten in (V, P) steigt an,

allerdings deutlich langsamer und scheinbar linear. Dies lässt darauf schließen, dass die benötigte Zeit, und damit auch die maximale Größe einer Instanz, vor allem von dem Verhältnis der Knoten zu den entstehenden Kanten abhängt. Dieses Verhältnis beeinflusst stark die Chance, dass sich in dem Graphen (V, P) Kreise bilden, welche dann entfernt werden müssen.

Das Ergebnis hängt natürlich ebenfalls von der verwendeten Hardware ab, allerdings sollte dies nur den Zeitpunkt des Anstiegs verzögern. Eine genaue Aussage, bis zu welchem Wert eine Instanz sinnvoll lösbar ist, lässt sich somit nicht treffen, aber man erkennt dennoch, dass die Instanzgröße beschränkt ist.

5.1.2 Vergleich ILP mit gieriger Version

In diesem Experiment soll nun untersucht werden wie stark sich die optimale ILP Lösung von der gierigen Variante, auf komplett zufällig generierten DAGs, mit unterschiedlichen Kantendichten, unterscheidet.

Angegeben sind dafür die Anzahl der Kanten, welche aus dem Graphen (V, P) gelöscht werden müssen, damit der finale DAG C keine Kreise enthält und die dafür benötigten Kosten. Die Kosten sind hierbei die Summe der Regreten der Kanten, welche von dem jeweiligen Algorithmus gelöscht wurden. Außerdem relevant ist natürlich die benötige Zeit, die die beiden Varianten benötigen. Die Laufzeit der gierigen Version sind allerdings in der nachfolgenden Tabelle nicht aufgeführt, da die Zeit immer deutlich unter einer Sekunde lag, also insbesondere auch deutlich unter der benötigten Zeit des ILPs.

Knoten: 100, Anzahl DAGs: 10

Dichte	Kosten		Gelöschte Kanten		Benötigte Zeit	Anzahl
	ILP	Greedy	ILP	Greedy	ILP (s)	Kanten in (V, P)
0.1	0.00	0.00	0	0	0.000	7
0.2	0.00	0.00	0	0	0.001	68
0.3	0.21	0.21	1	1	0.013	229
0.4	2.64	3.23	7	9	0.051	456
0.5	7.49	9.75	19	28	0.327	784
0.6	15.94	20.01	44	59	2.818	1158
0.7	24.41	31.73	62	85	7.765	1543
0.8	29.28	37.44	66	91	10.173	1861
0.9	28.15	35.78	53	72	7.198	1961
1.0	7.00	13.50	6	11	0.132	1663

Tabelle 2: Zufällige DAGs, Kosteneffizienz

Man kann gut erkennen, dass für die Lösung des ILPs in der Regel weniger Kanten gelöscht werden, als für die der gierigen Version. Wie zu erwarten sind auch die Kosten der gierigen Version meist höher, jedoch ist hier interessant zu beobachten, dass erst bei DAGs mit über 30% ihrer maximal möglichen Anzahl an Kanten ein Unterschied feststellbar ist. Insgesamt scheinen die Kosten auch maximal doppelt so hoch, wie die der optimalen Lösung zu sein. Das erst bei extrem dichten Graphen ein Unterschied feststellbar ist, liegt vermutlich an der Anzahl der Kanten des Graphen (V, P). Erst wenn die Anzahl der resultierenden Kanten im Verhältnis zu den Knoten einen gewissen Wert überschreitet, scheinen mehrere Kreise zu entstehen. Sind keine Kreise oder nur sehr wenige vorhanden, so ist es sehr wahrscheinlich, dass beide Methoden zum selben Ergebnis kommen und die gleichen Kanten löschen.

Auffällig ist außerdem, dass sobald die generierten DAGs sehr nahe einer vollständigen Rangliste kommen (vollständige Rangliste: Dichte = 1), die Anzahl der Kanten in (V, P) weniger stark zunimmt und für eine Rangliste sogar wieder stark abnimmt. Da hier, im Fall von vollständigen Ranglisten, alle möglichen Kanten vorhanden sind, sinkt allen Anschein nach die Wahrscheinlichkeit das Kanten am Ende zustande kommen. Damit nimmt dann natürlich auch die Anzahl der gelöschten Kanten, Kosten und die Laufzeit des ILPs wieder ab.

5.1.3 Experimente nach Eric Malmi

Die folgenden Untersuchungen (Abbildung 3) sollen möglichst gut die Experimente von Eric Malmi und Gionis (2015) aus Kapitel 7.1.3 nachstellen, um zu überprüfen, wie gut die Implementierung als ILP im Vergleich zu der dort vorgestellten gierigen Version ist.

Hier basieren die DAGs nun auf einem Schlüssel und werden unterschiedlich stark verändert (siehe Kapitel 4.2). Untersucht werden soll hier, wie gut der Algorithmus den ursprünglichen, unveränderten Graphen erkennt. Die Kosten sind in diesem Experiment die Distanz zwischen dem berechneten DAG C und dem ursprünglichen Schlüssel-DAG. Die optimal-Linie setzt sich aus den Punkten zusammen, welche aus der Distanz zwischen dem Schlüssel-Graphen und sich selbst berechnet wird und geben damit an, wie gut das Ergebnis bestenfalls sein kann. Dieser Wert ist insbesondere ungleich Null, außer es handelt sich um eine vollständige Rangliste, da Kanten welche nicht vorhanden sind trotzdem die Distanz um den Wert q erhöhen. Die Nulllinie ist die Distanz zwischen dem Schlüssel-Graphen und einem leeren Graphen und soll als obere Grenze dienen. Die Werte der Nulllinie sind aber keine feste Grenze, da beispielsweise ein DAG, welcher nur die entgegengesetzten Kanten des Schlüssel-DAGs enthält, eine viel größere Distanz aufweist.

Die Ergebnisse sind denen von Eric Malmi und Gionis (2015), wie zu erwarten, sehr ähnlich. In beiden Versuchen erkennen anfangs beiden Lösungsmethoden den ursprünglichen Graphen sehr gut. Erst ab einem gewissen Punkt steigt die Distanz im Vergleich zur optimalen Lösung stark an und konvergiert dann gegen einen Wert unterhalb der Nulllinie. Vor allem erkennt man hier aber, dass sich die beiden zu vergleichenden Versionen nicht unterscheiden. Auch nach vielen Wiederholgen ließ sich hier kein Unterschied zwischen beiden Varianten feststellen.

5.1.4 Experimente zur Qualität auf extrem dichten Graphen

Im Abschnitt 5.1.2 war ja bereits erkennbar, dass sich Unterschiede zwischen den beiden Versionen erst zeigen, sobald die erzeugten DAGs eine hohe Dichte an Kanten aufweisen. Im vorherigem Abschnitt ist die Kantendichte vergleichsweise sehr gering gewesen. Daher werden die Versuche hier noch einmal wiederholt (Abbildung 4), allerdings mit einer deutlich höheren Chance, dass eine Kante in einem DAG enthalten ist. Auch die Varietät der erzeugten Graphen wurde noch weiter erhöht, ebenfalls um die Anzahl der resultierenden Kreise zu erhöhen.

Die Nulllinie wurde hier aus Gründen der Übersichtlichkeit weggelassen, da diese bei solchen extrem Dichten DAGs deutlich höher liegt und somit der wesentliche Teil des Experiments kaum noch erkennbar ist (Nulllinie > 600).

Hier lassen sich nun Unterschiede in den beiden Varianten ausmachen. Wie zu erwarten ist die Distanz der ILP Version besser, allerdings ist auch gut zu erkennen, dass der Unterschied relativ gering ausfällt. Selbst mit stark steigender Variation der DAGs nimmt der Unterschied kaum zu und scheint zudem recht konstant zu bleiben.

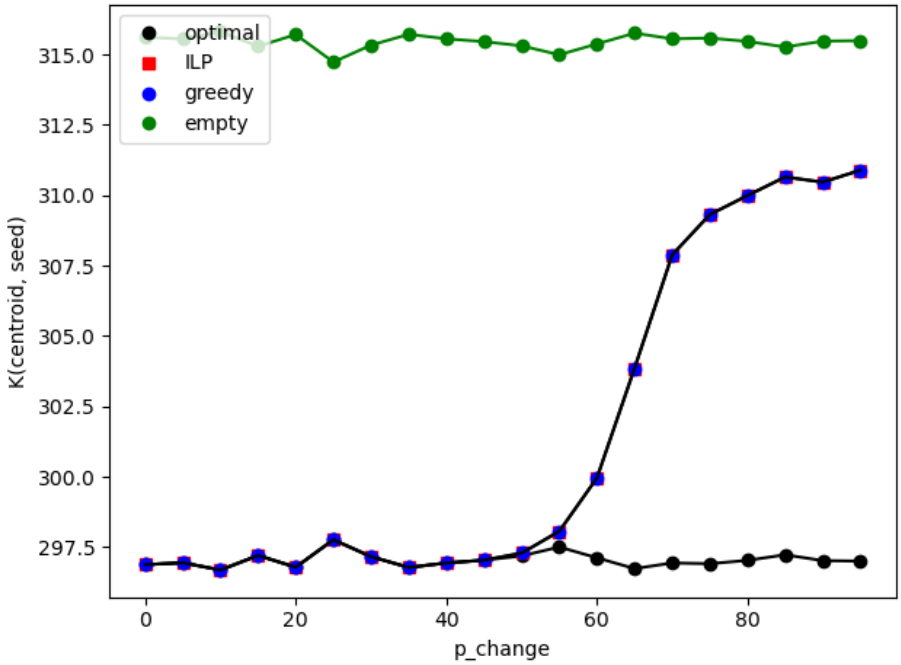
5.2 Abschluss

Nach Auswertung aller Experimente ist nun ziemlich deutlich zu erkennen, dass der gierige Algorithmus eine sehr gute Möglichkeit zum Aggregieren von DAGs bietet. Das ILP liefert auf den meisten Probleminstanzen gleich gute oder nur geringfügig bessere Ergebnisse, obwohl die Lösung optimal ist. Erst bei extrem dichten anfänglichen Graphen lässt sich ein Unterschied feststellen, aber dieser ist in der Regel nur sehr gering. Man sollte zudem beachten, dass ein ILP in der Praxis oft nicht sehr praktisch ist und auf größeren Instanzen mitunter sehr viel Zeit benötigt. Auf Probleminstanzen, wo sich ein Unterschied zwischen den beiden Varianten zeigt, braucht das ILP außerdem einige Minuten, im Vergleich zur gierigen Version, welche recht konstant bei unter einer Sekunde bleibt. Beide Versionen verhalten sich insgesamt sehr ähnlich, hängen also stark von der vorherigen Berechnung der Distanz mittels Kendall-Tau ab. So spielt am Ende dann auch

die Wahl der Parameter p und q eine große Rolle, da sie maßgeblich dazu beitragen, wie viele Kanten am Ende vorhanden sind.

In der Praxis ist eine Lösung des Problems mit einem ILP daher nicht gut geeignet, da die Qualität der Lösung in der Regel nicht viel besser ist als die einer einfachen gierigen Implementierung, die außerdem auch deutlich weniger Zeit beansprucht. In einzelnen Sonderfällen könnte der Unterschied vielleicht dennoch relevant sein, wenn die Optimalität der Lösung im Vordergrund steht. Dafür sollte aber absehbar sein, dass sich aufgrund der Daten viele Kreise in dem Graphen nach der Distanzbestimmung bilden und die Rechenzeit sollte keine zu große Rolle spielt.

(a) $n_{swap} = 0$



(b) $p_{change} = 0$

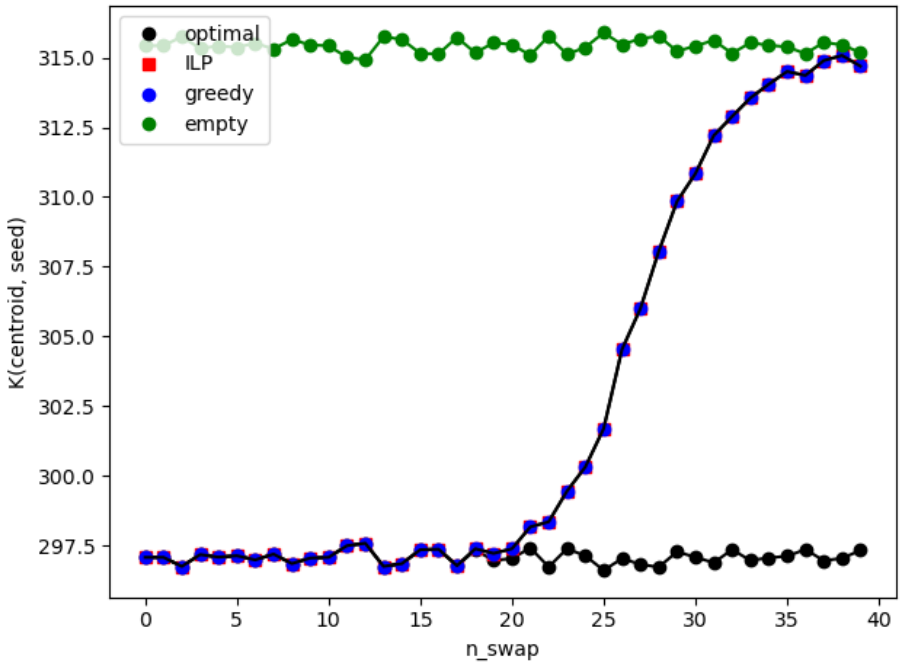
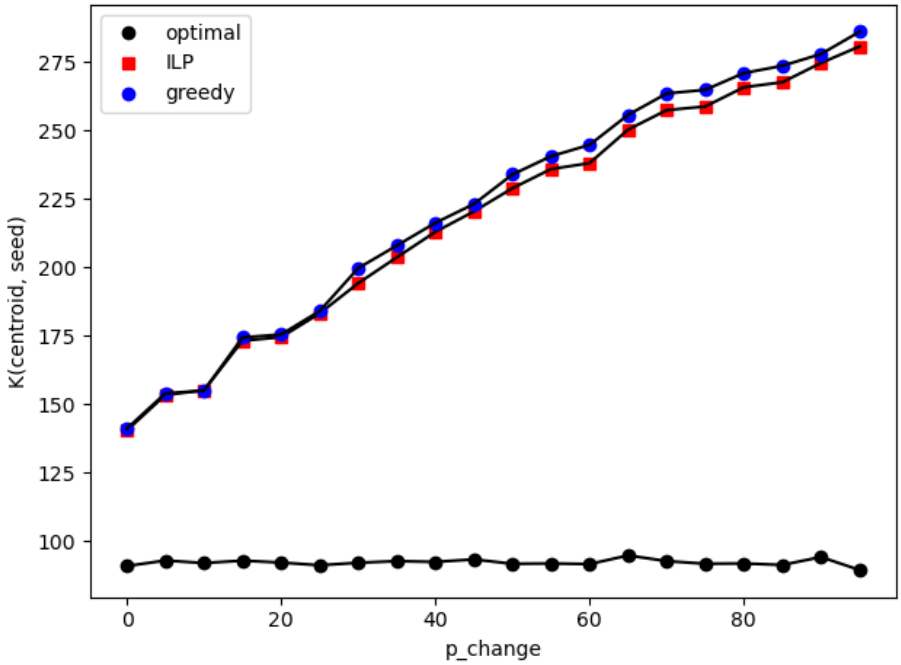


Abbildung 3: Experimente nach Eric Malmi
Knoten: 50, Anzahl DAGs: 100, Dichte: 0.03

(a) $n_{swap} = 20$



(b) $p_{change} = 40$

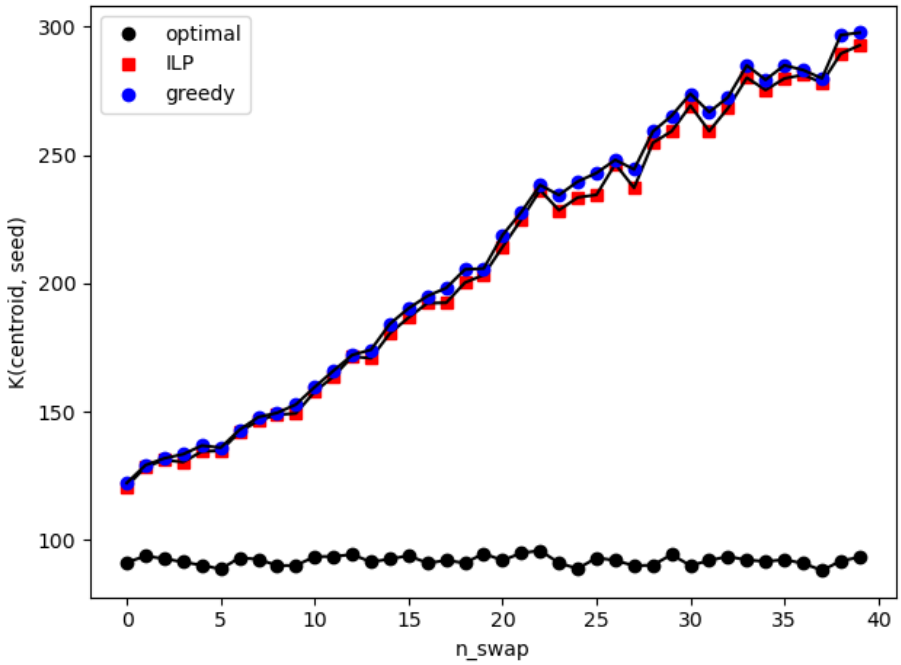


Abbildung 4: Experimente auf extrem dichten Graphen
Knoten: 50, Anzahl DAGs: 10, Dichte: 0.7

Literatur

Nikolaj Tatti Eric Malmi und Aristides Gionis (2015). „Beyond rankings: comparing directed acyclic graphs“. In: *Data mining and knowledge discovery* 29.5, S. 1233–1257.

Xue Li, Xinlei Wang und Guanghua Xiao (Aug. 2017). „A comparative study of rank aggregation methods for partial and top ranked lists in genomic applications“. In: *Briefings in Bioinformatics* 20.1, S. 178–189. eprint: https://academic.oup.com/bib/article-pdf/20/1/178/27689776/bbx101.pdf.

Eric Malmi (2018). „Collective Entity Resolution Methods for Network Inference“. Aalto University publication series DOCTORAL DISSERTATIONS, 73/2018. Diss.

Abbildungsverzeichnis

Tabellenverzeichnis