Technische Universität Berlin

Graphclustern durch Zerstören langer induzierter Pfade

Bachelorarbeit

am Fachgebiet Algorithmik und Komplexitätstheorie

Prof. Dr. Rolf Niedermeier

Fakultät IV Elektrotechnik und Informatik

Technische Universität Berlin

vorgelegt von

Felix Bohlmann

Betreuer: Prof. Dr. Rolf Niedermeier,
Dr. Falk Hüffner,
Dr. Christian Komusiewicz

Felix Bohlmann

Matrikelnummer: 339266

Barther Straße 78

13051 Berlin

Zusammenfassung

Beim Clustern von Graphen geht es darum, dichte Teilgraphen zu finden, die wenige Kanten zum Rest des Graphen aufweisen, und diese mit möglichst wenigen Änderungen zu separieren. Diese dichten Teilgraphen werden durch ein konkretes Modell charakterisiert. Ein solches ist zum Beispiel das P_5-frei-Modell, mit welchem wir uns in dieser Arbeit befassen wollen. Das P_5-frei-Modell ist besonders für Graphen mit einer vermuteten Kern-Peripherie-Struktur geeignet, weil es in P_5-freien Graphen einen dominierenden Teilgraph in Form einer Clique oder eines P_3 sowie weitere zu diesem adjazente Knoten gibt. Das *optimale P_5-frei-Editierungsproblem* ist das $\mathcal{NP}$-schwere Problem einen gegebenen Graphen durch Hinzufügen und Löschen einer minimalen Anzahl an Kanten in einen P_5-freien Graphen zu transformieren. Wir werden einen parametrisierten Algorithmus vorstellen, der in $O(10^k \cdot m^2)$ Laufzeit eine optimale Lösung für das Problem findet. Der Fokus dieser Arbeit liegt darauf, eine möglichst leistungsstarke Implementierung dieses Algorithmus zu konstruieren. Dazu gehen wir auf Algorithmen zur Suche von P_5s ein und stellen Datenreduktionsregeln sowie untere Schranken für die minimale Anzahl der Änderungen auf. Anschließend wenden wir den Algorithmus auf von uns generierten synthetischen Graphen sowie aus wissenschaftlichen Datenbanken entnommenen Interaktions- und Proteinähnlichkeitsgraphen an und bewerten wie gut das P_5-frei-Modell zu den einzelnen Graphen passt. Wir kommen zu dem Schluss, dass das P_5-frei-Modell für die von uns gewählten Graphen ein gutes Modell ist. Wir konnten zum Beispiel für den Zachary-Karate-Club-Graphen mit 34 Knoten eine optimale Lösung mit 13 Editierungen berechnen, welche den Graphen in die selben Partitionen wie einst Zachary aufteilt. Allerdings ist das Finden einer optimalen Lösung deutlich schwieriger als für viele andere ähnliche Modelle, wie zum Beispiel das Clustergraph- oder das Co-Graph-Modell.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Notation und Definitionen 4
1.2 Das P_5-frei-Editierungsproblem 5
1.3 Parametrisierte Algorithmen 6
1.4 Testgraphen 7
1.4.1 Synthetische Graphen 7
1.4.2 Reale Anwendungsgraphen 9
1.4.3 Vergleich der Eigenschaften der Testgraphen 10
1.5 Implementierungsdetails 13

2 Algorithmen zum Finden von P_5s **15**

2.1 Algorithmen zum Finden eines P_5 15
2.1.1 Endpunktsuche 16
2.1.2 Mittelpunktsuche 17
2.1.3 Hoàng-Algorithmus 18
2.1.4 Evaluierung 20
2.1.4.1 Vergleich der Algorithmen untereinander 20
2.1.4.2 Laufzeitvergleich der Graphklassen 21
2.2 Algorithmen zum Finden aller $P_5 s$ 22
2.3 Algorithmen zur heuristischen Suche mehrerer P_5s 24

3 Lösungsalgorithmus **26**

3.1 Suchbaumalgorithmus 26
3.2 Untere Schranken 27
3.2.1 Knotendisjunktes Packing 28
3.2.2 Knotenpaardisjunktes Packing 28
3.2.3 Hitting-Set-Approximation 31
3.2.4 Fazit 34

3.3 Datenreduktion 35
3.4 Annotationen 40
3.5 Weitere Verbesserungsansätze 42
3.5.1 Separates Lösen von Zusammenhangskomponenten 42
3.5.2 Gezieltes Verzweigen 43
3.5.3 Speichern von P_5-Listen 43
3.6 Ergebnisse 43
3.7 Fazit 49

4 Ausblick **50**

Anhang **51**

Literaturverzeichnis **IV**

Erklärung der Urheberschaft **VIII**

Kapitel 1

Einleitung

Die Graphentheorie ist ein wichtiges Gebiet für die theoretische Informatik. Graphen erlauben eine universelle Modellierung einer Vielzahl von Anwendungsproblemen aus den unterschiedlichsten Gebieten, angefangen von der Informatik selbst, über die Biologie bis hin zu den Sozial- und Wirtschaftswissenschaften [1, 2, 3, 4]. Bei dem Teilgebiet der Graphmodifikationsprobleme geht es ganz allgemein darum, ein durch einen Graph modelliertes System mit möglichst wenig Aufwand in einen gewünschten Zielzustand zu überführen. Ein typisches Graphmodifikationsproblem ist das *Clustern* von Graphen. Dabei geht es darum dichte Teilgraphen zu finden, die wenige Kanten zum Rest des Graphen aufweisen und diese mit möglichst wenigen Änderungen zu separieren [5]. Diese dichten Teilgraphen werden oft als *Cluster* bezeichnet und mit einem konkreten Modell charakterisiert. Am bekanntesten hierfür ist das Cliquen-Modell [6], welches Teilgraphen spezifiziert, bei denen alle Knoten zueinander adjazent sind, sogenannte *Cliquen*. Das Clustern nach diesem Modell ergibt einen sogenannten Clustergraph, dessen Zusammenhangskomponenten allesamt Cliquen sind. Beim Clustern werden die Knoten des Graphen gruppiert, weshalb Clustern Ähnlichkeit zum unüberwachten Lernen bei der automatisierten Mustererkennung hat [7].

In dieser Arbeit werden wir uns mit dem P_5-frei-Modell beschäftigen und speziell mit dem Problem in ungerichteten und ungewichteten Graphen alle induzierten P_5s aufzulösen, indem wir eine minimale Anzahl an Kanten hinzufügen und löschen. Ein P_5 ist hierbei ein zu Abbildung 1.1 isomorpher Graph. Wir bezeichnen dieses Problem als *optimales P_5-frei-Editierungsproblem*.

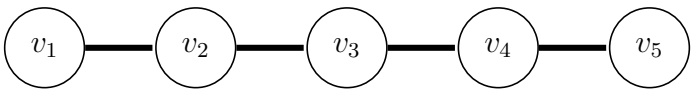

Abbildung 1.1: Ein P_5

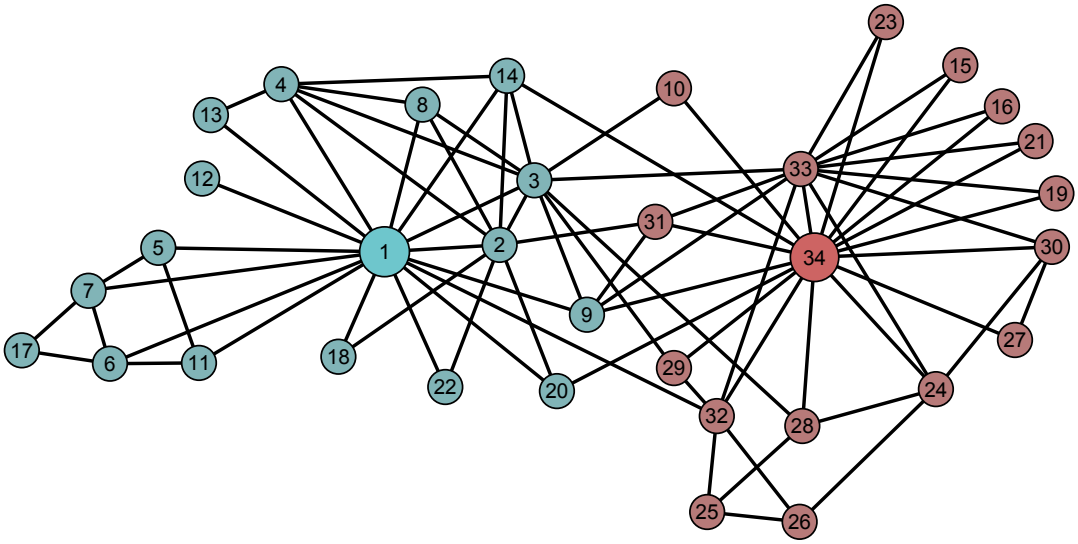

Abbildung 1.2: Zachary-Karate-Club-Graph: Die Färbung der Knoten repräsentiert die Clubzugehörigkeit des Mitglieds nach der Trennung.

Als erstes Beispiel schauen wir uns den berühmten Zachary-Karate-Club-Graph in Abbildung 1.2 an [8, S. 71]. Die Knoten stellen die Mitglieder eines Karate-Clubs dar und die Kanten repräsentieren die Freundschaften zwischen ihnen. Zachary analysierte diesen Graphen und stellte unter anderem fest, dass die Mitglieder des Clubs sich grob in zwei Gruppen einteilen ließen, was den Mitgliedern selbst nicht bewusst war. Die eine Gruppe versammelte sich um den Trainer (Knoten 1) und die andere um den Präsidenten (Knoten 34). Außerdem gab es einige Mitglieder, die mit beiden Gruppen befreundet waren. Zachary versuchte mit Hilfe eines minimalen Schnittes zu entscheiden zu welchem Lager diese Mitglieder eher gehörten. Von diesen theoretischen Überlegungen unbeeinflusst kam es zum Streit zwischen dem Trainer und dem Präsidenten, welcher letztendlich zur Aufspaltung des Clubs und seiner Mitglieder führte. Die Färbung des Graphen repräsentiert die Clubzugehörigkeit der Mitglieder nach der Trennung. Dabei stellte sich heraus, dass der Club sich nahezu genau in die beiden Gruppen aufteilte, die Zacharys Modell zuvor bestimmt hatte. (Die einzige Ausnahme ist Knoten 9, welcher nach dem Modell des minimalen Schnittes zur Gruppe des Präsidenten (Knoten 34) gehört.) Wir werden das optimale P_5-frei-Editierungsproblem für den Karate-Club-Graph lösen und mit der Lösung das P_5-frei-Modell mit dem Modell des minimalen Schnittes sowie dem tatsächlich eingetretenen Ausgang vergleichen.

Eine weitere konkrete Anwendung für das optimale P_5-frei-Editierungsproblem liegt in der Analyse von *Proteininteraktionsnetzwerken*. Bei diesen entspricht eine Kante zwischen zwei Knoten der Interaktion zwischen zwei Proteinen. Gesucht sind so ge-

nante Komplexe. Das *Kern-Peripherie-Modell* nimmt an, dass diese in einem ungestörten Netzwerk in Form von separaten Kern-Peripherie-Strukturen vorliegen. Diese bestehen jeweils aus einem Kern, einem dichten Teilgraphen des Netzwerkes, und einer Peripherie, welche weitere Proteine umfasst, die mit dem Kern interagieren. Um solche Strukturen in einem gestörten Graphen wiederherzustellen, wurde unter anderem das *Split-Cluster-Graph-Modell* vorgeschlagen [9]. Alle Zusammenhangskomponenten eines Split-Cluster-Graphen sind sogenannte Split-Graphen, dass heißt sie können in zwei Teilmengen V_1, V_2 aufgeteilt werden, sodass alle Knoten in V_1 zueinander adjazent sind und alle Knoten in V_2 paarweise nicht adjazent sind. Die Menge V_1 stellt dann den Kern dar und die Menge V_2 enthält die Knoten der Peripherie. Split-Cluster-Graphen können durch eine Menge von verbotenen Teilgraphen charakterisiert werden, diese enthält unter anderem einen P_5. Wir schlagen stattdessen vor, dass optimale P_5-frei-Editierungsproblem für das Netzwerk zu lösen. Dieser Ansatz ist weniger restriktiv, da er sich mit weniger verbotenen Teilgraphen charakterisieren lässt und daher üblicherweise weniger Editierungen benötigt. Wir werden versuchen, die selben Proteininteraktionsnetzwerke wie in [9] zu lösen und die Ergebnisse zu vergleichen.

Das Problem, eine minimale Anzahl an Modifikationen zu finden, um einen Graph zu einem P_5-freien Graph beziehungsweise zu einem Split-Cluster-Graph zu transformieren, ist $\mathcal{NP}$-schwer [9][1]. Letzteres hat eine Laufzeit von $O(10^k \cdot (n+m))$ [9]; hierbei ist k die Anzahl der benötigten Änderungen, n die Anzahl der Knoten des Graphen und m die Anzahl der Kanten. Für das optimale P_5-frei-Editierungsproblem werden wir einen Algorithmus mit einer Laufzeit von $O(10^k \cdot m^2)$ vorstellen. Die verwandten Probleme P_3-freie beziehungsweise P_4-freie Graphen zu erzeugen sind ebenfalls $\mathcal{NP}$-schwer und bereits ausführlich untersucht worden [10, 11, 12]. In der Literatur werden P_3-freie Graphen häufig als Clustergraphen und P_4-freie Graphen als Co-Graphen bezeichnet. Um Graphen entsprechend zu editieren wurden Algorithmen mit Laufzeiten von $O(1.62^k + m + n)$ für P_3-frei [13] und $O(4.612^k + n^{4.5})$ für P_4-frei [14] vorgestellt. Ein weiteres ähnliches $\mathcal{NP}$-vollständiges Problem ist das Minimum-Flip-Consensus-Tree-Problem [15], bei welchem nur bestimmte P_5s aufgelöst werden müssen. Für dieses Problem konnte ein Algorithmus mit $O(4.83^k)$ Laufzeit aufgestellt werden [16]. Wir gehen in dieser Arbeit speziell auf P_5-freie Graphen ein, aber viele der vorgestellten Ansätze können für beliebige P_l mit $l > 2$ eingesetzt werden.

[1] Die NP-Schwere für die optimale P_5-frei-Editierung folgt implizit aus der Herleitung für die $\mathcal{NP}$-Schwere der optimalen Split-Cluster-Graph-Editierung. Die zitierte Arbeit enthält jedoch kein eigenständiges Theorem dazu.

Es stellt sich die Frage, ob es überhaupt notwendig ist ein solch schweres Problem optimal zu lösen oder ob eine angenäherte Lösung nicht ausreichend ist. Zum einen gibt es kein bekanntes approximatives Lösungsverfahren und zum anderen ist eine optimale Lösung notwendig, um beurteilen zu können, ob P_5-freie Graphen ein gutes Modell für die angesprochenen Graphen darstellen.

Abschließend ist es noch erwähnenswert, dass auf P_5-freien Graphen einige wichtige $\mathcal{NP}$-vollständige Probleme in polynomieller Zeit gelöst werden können. Dazu zählen das Independent-Set-Problem [17] sowie das k-Knotenfärbungsproblem [18].

Das Hauptziel dieser Arbeit ist es für das optimale P_5-frei-Editierungsproblem eine möglichst leistungsstarke Implementierung zu entwickeln. Dafür werden wir verschiedene klassische Lösungs- und Optimierungskonzepte betrachten, umsetzen und deren Leistung anhand von selbst generierten und wissenschaftlichen Datenbanken entnommen Testgraphen vergleichen. Wir werden zunächst das Problem in Abschnitt 1.2 formal beschreiben und anschließend in Abschnitt 1.3 auf parametrisierte Algorithmen eingehen, welche die Grundlage für unseren Lösungsalgorithmus bilden. Danach gehen wir näher auf die Graphen ein, welche wir für unsere Leistungstest ausgewählt haben. Eine wichtige Teilaufgabe stellt die Konstruktion von Algorithmen zur Suche nach P_5s in einem Graphen dar, worauf wir ausführlich in Kapitel 2 eingehen werden. In Kapitel 3 werden wir weitere Verbesserungsansätze vorstellen, unter anderem sind dies Datenreduktionsregeln und untere Schranken. Am Schluss von Kapitel 3 werden wir die Ergebnisse unserer Laufzeittests vorstellen, diskutieren und insbesondere darauf eingehen, ob P_5-freie Graphen ein gutes Modell für den Karate-Club-Graph und die Proteininteraktionsnetzwerke sind.

1.1 Notation und Definitionen

Im Folgenden sei $G := (V, E)$ ein ungerichteten *Graph*, wobei V die Menge der *Knoten* und $E \subseteq \{\{u, v\} \mid u, v \in V \wedge u \neq v\}$ die Menge der *Kanten* von G ist. Im Allgemeinen meint $n := |V|$ die Anzahl an Knoten und $m := |E|$ die Anzahl Kanten in G. Die *Nachbarschaft* $N(v) := \{u \mid \{u, v\} \in E\}$ bezeichnet die Menge aller Knoten, die zu Knoten v adjazent sind. Ein *Pfad* $(v_1, \ldots, v_k)$ ist eine geordnete Menge von Knoten aus G wobei gilt, dass zwei aufeinander folgende Knoten des Pfades im Graphen G benachbart sind. Ein Pfad in dem jeder Knoten einmalig ist nennen wir einen *Weg*. Die *Länge* eines Pfades beziehungsweise eines Weges ist definiert als die Anzahl seiner Knoten. Ein durch eine Menge S induzierter Teilgraph von G ist definiert als $G[S] := (S, \{\{u, v\} \in E \mid u, v \in S\})$. Wir verwenden die selbe Notation analog, für

einen durch eine geordnete Menge induzierten Teilgraph, wobei die Ordnung keinerlei Relevanz für das Ergebnis hat. Damit definieren wir einen $P_l := G[w]$ für einen Weg $w := (v_1, \ldots, v_l)$ mit $\forall i \in [1, l].\nexists j \in [1, l] \backslash \{i-1, i+1\} \ : \ E(v_i, v_j)$. Eine Menge von paarweise adjazent Knoten nennen wir eine *Clique*. Eine Clique ist P_3-frei und insbesondere auch P_5-frei. Graphen deren Zusammenhangkomponenten alle Cliquen sind, nennen wir *Clustergraphen*. Der *Abstand* zweier Knoten in einem Graphen ist die Länge des kürzesten Pfades, welcher beide Knoten enthält, minus eins. Der Abstand ist also über die Anzahl der Kanten definiert. Die *Exzentrizität* eines Knoten ist der maximale Abstand zu einem anderen Knoten des Graphen. Das Maximum aller Exzentrizitäten bezeichnen wir als den *Durchmesser*. Des Weiteren definieren wir den *maximalen Durchmesser* auf unzusammenhängenden Graphen als den größten Durchmesser der einzelnen Zusammenhangskomponenten.

1.2 Das P_5-frei-Editierungsproblem

Bevor wir das P_5-frei-Editierungsproblem formal definieren, schauen wir uns zunächst erst einmal an, was es für einen Graphen überhaupt bedeutet P_5-frei zu sein.

Definition 1.1. *Ein Graph ist* P_5-frei *genau dann, wenn er keinen P_5 als induzierten Teilgraph enthält.*

Alle P_5-freien Graphen erfüllen die folgende Eigenschaft.

Lemma 1.1 ([19])**.** *Ein zusammenhängender P_5-freier Graph enthält entweder eine dominierende Clique oder einen dominierenden P_3.*

Die Umkehrung des Lemmas gilt nicht, weil ein P_5 selbst bereits einen dominierenden P_3 enthält. P_5-freie Graphen lassen sich außerdem wie folgt charakterisieren.

Lemma 1.2 ([19])**.** *Ein zusammenhängender Graph G ist P_5-frei genau dann, wenn jeder zusammenhängende, induzierte Teilgraph $H \subseteq G$ eine dominierende Clique oder einen dominierenden C_5 enthält.*

Wir führen nun das P_5-frei-Editierungsproblem ein.

Problem 1. P_5-FREI-EDITIERUNGSPROBLEM
Eingabe: Ein ungerichteter Graph G und eine natürliche Zahl k.
Frage: Ist es möglich den Graph G durch Löschen und Hinzufügen von höchstens k Kanten in einen P_5-freien Graph zu transformieren?

Das Tupel (G, k) ist eine Instanz des P_5-frei-Editierungsproblems. Das *Verhältnis* zweier Knoten u, v kann entweder eine Kante oder eine Nicht-Kante sein, je nach dem ob $\{u, v\} \in E$ ist oder nicht. Wenn wir im Folgenden vom *Invertieren* des Verhältnisses zweier Knoten sprechen meinen wir, dass in einem Graphen eine Kante zu einer Nicht-Kante wird oder umgekehrt.

Definition 1.2. *Für einen Graphen G und eine Menge M von Knotenpaaren definieren wir den Graphen $G \triangle M := \big(V, (E \setminus M) \cup (M \setminus E)\big)$.*

Mit Hilfe dieser Definition bestimmen wir nun was es für eine Menge von Knotenpaaren bedeutet eine Lösung für das P_5-frei-Editierungsproblem zu sein.

Definition 1.3. *Eine Menge L von Knotenpaaren ist eine* Lösung *für das P_5-frei-Editierungsproblem (G, k), wenn der Graph $G \triangle L$ ein P_5-freier Graph ist und L höchstens k viele Elemente enthält.*

Wenn wir im Folgenden davon sprechen, dass L eine Lösung für Graph G ist, dann meinen wir damit, dass L eine Lösung für alle P_5-frei-Editierungsprobleme (G, l') ist, wobei $l' \geq |L|$. Die Elemente einer Lösung nennen wir *Editierungen*. Unser Ziel ist es die kleinste Lösung für ein P_5-frei-Editierungsproblem eines Graphen zu finden. Wir definieren eine solche Lösung wie folgt.

Definition 1.4. *Eine Lösung L für Graph G ist genau dann* optimal*, wenn es für G keine weitere Lösung L' mit $|L'| < |L|$ gibt.*

Im nächsten Abschnitt werden wir uns mit parametrisierten Algorithmen beschäftigen, welche die Grundlage für die Suche nach einer optimalen Lösung sein werden.

1.3 Parametrisierte Algorithmen

Parametrisierte Algorithmen wurden bereits erfolgreich für viele ähnliche Graphmodifikationsprobleme, aber insbesondere auch für das Cluster- und Co-Graph-Editierungsproblem [10, 14] sowie das Minimum-Flip-Consensus-Tree-Problem [15, 16] eingesetzt. Wir werden in diesem Abschnitt zeigen, wie sich das P_5-frei-Editierungsproblem mit Hilfe von parametrisierten Algorithmen entscheiden lässt.

Um eine optimale Lösung für einen Graphen G mit Hilfe parametrisierter Algorithmen zu finden, müssen wir einen Lösungsalgorithmus konstruieren der in $f(k) \cdot \text{poly}(n)$ Zeit das P_5-frei-Editierungsproblem (G, k) entscheidet. Dabei ist f eine beliebige berechenbare Funktion, welche nur von k abhängt. Für ein festes k ergibt sich eine polynomielle Laufzeit für den Algorithmus.

Wir können einen einfachen Suchbaum verwenden, um in $f(k) \cdot \text{poly}(n)$ Zeit das P_5-frei-Editierungsproblem (G, k) zu entscheiden. Wir beobachten, dass wir um einen P_5 aufzulösen entweder eine der vier vorhandenen Kanten löschen oder eine der sechs möglichen Kanten einfügen müssen. Der Suchbaum wählt dann einen beliebigen P_5 des Graphen, führt eine der zehn Editierungen l aus und entscheidet dann rekursiv das P_5-frei-Editierungsproblem $(G \triangle \{l\}, k-1)$. Falls nach k Editierungen der Graph nicht P_5-frei ist, dann müssen die letzten Änderungen rückgängig gemacht und eine der anderen Möglichkeiten getestet werden, bis der Suchbaum vollständig erschöpft ist oder wir einen P_5-freien Graph erzeugt haben. In letzterem Falle können alle durchgeführten Modifikationen beim Wiederaufstieg zusammengesammelt und so eine Lösung für (G, k) zurückgegeben werden. Für alle k Schichten des Suchbaumes müssen im schlechtesten Falle jeweils alle zehn Editierungen getestet werden, woraus sich ein Verzweigungsgrad von zehn und eine Laufzeit von $O(10^k)$ für $f(k)$ ergeben. Die Laufzeit von $\text{poly}(n)$ hängt davon ab wie schnell der nächste aufzulösende P_5 gefunden werden kann. Dafür können Algorithmen mit polynomieller Laufzeit konstruiert werden, worauf wir in Kapitel 2 genauer eingehen werden.

Die beiden Teilfunktionen $f(k)$ und $\text{poly}(n)$ bieten zwei Ansatzpunkte für die Laufzeitoptimierung. Erstens werden wir in Kapitel 2 mit verschiedenen Methoden experimentieren, um die Zeit zum Finden des nächsten P_5 zu minimieren und zweitens werden wir versuchen den hohen Verzweigungsgrad des Suchbaumes zu verringern.

1.4 Testgraphen

Um die Leistungsfähigkeit unserer Algorithmen zu vergleichen benötigen wir verschiedenartige Testgraphen, welche wir im Folgenden vorstellen werden. Dabei unterscheiden wir zwischen Graphen, die wir aus verschiedenen wissenschaftlichen Datenbanken entnommen haben und extra für unsere Tests generierten Graphen, die wir im Folgenden als synthetische Graphen bezeichnen.

1.4.1 Synthetische Graphen

Gilbertgraphen $G(n, p)$ sind Graphen, bei denen unabhängig voneinander mit der selben Wahrscheinlichkeit p zwischen zwei Knoten eine Kante ist [20]. Als Parameter haben wir eine Knotenanzahl n von 10 bis 200 Knoten mit jeweils 19%, 36% und 51% Kan-

tenwahrscheinlichkeit zwischen zwei Knoten gewählt [2]. Von jeder Kombination wurden 10 Instanzen erstellt.

Die Gilbertgraphen spiegeln allerdings nicht die zu erwartenden Struktur der Anwendungsdaten wieder. Um derartig strukturierte Daten zu simulieren, benutzen wir eine Klasse von Graphen, die wir im Folgenden als *synthetische Clustergraphen* bezeichnen. Für ihre Konstruktion erzeugen wir zunächst zufällige Clustergraphen. Um Messfehler zu simulieren, invertieren wir dann für zufällig ausgewählte Knotenpaare das Nachbarschaftsverhältnis. Als Parameter haben wir eine Knotenanzahl von 10 bis 200 Knoten, welche zufällig auf $\lfloor\sqrt{n}\rfloor$ viele Cliquen verteilt werden. Die Anzahl der invertierten Verhältnisse beträgt jeweils 2%, 5%, 10% oder 20% der maximalen Kantenanzahl $\binom{n}{2}$, auf ganze Zahlen abgerundet. Von jeder Kombination wurden wieder 10 Instanzen erstellt. Bei den synthetischen Clustergraphen lässt sich die Größe einer optimalen Lösung mit der Anzahl der eingebauten Kantenfehler als obere Grenze abschätzen. Dies folgt daraus, dass wir einen P_3-freien Clustergraphen erhalten, wenn wir genau diese Kanten beziehungsweise Nicht-Kanten wiederherstellen.

Speziell für die Tests der P_5-Suchalgorithmen führen wir außerdem die Klasse der *synthetischen P_5-freien Graphen* ein. Wir konstruieren für jede Knotenanzahl n zwischen 10 und 200 einen Graphen, wie für $n = 50$ in Abbildung 1.3 dargestellt. Zunächst erstellen wir aus der Hälfte der Knoten eine Clique, die wir Kernclique nennen. Die restlichen Knoten verteilen wir gleichmäßig auf $\lfloor\sqrt{n/2} + 1\rfloor$ viele weitere Cliquen, die Peripheriecliquen. Danach verbinden wir jede Peripherieclique mit einer zufällig ausgewählten Menge an Knoten aus der Kernclique, sodass diese Teilmenge und die Peripherieclique zusammen wiederum eine Clique bilden. Die Anzahl der zufällig ausgewählten Kernknoten liegt dabei uniform verteilt zwischen eins und der Anzahl der Peripheriecliquen. Es gilt folgendes Lemma.

Lemma 1.3. *Die synthetischen P_5-freien Graphen enthalten keinen induzierten P_5.*

Beweis. Sei K die Menge der Knoten aus der Kernclique und sei $P := \{P_1, \ldots, P_i\}$ die Menge der Mengen der jeweiligen Peripheriecliquenknoten. Aufgrund der Art und Weise ihrer Konstruktion sind die Peripheriecliquen knotendisjunkt. Außerdem sind je zwei Knoten aus zwei verschiedenen Peripheriecliquen nicht adjazent. Falls ein Knoten aus K zu einem Knoten aus einer der Peripheriecliquen adjazent ist, dann ist er zu allen anderen Knoten dieser Peripherieclique ebenfalls adjazent. Umgekehrt gilt, dass falls ein Knoten aus P_j zu einem Knoten $k_1 \in K$ adjazent ist, dann sind alle Knoten aus P_j adjazent zu k_1.

[2] Die ungewöhnliche Wahl des Parameters p ist auf einen Fehler bei der Generierung der Graphen, aufgrund einer unzureichenden Dokumentation, zurückzuführen.

Angenommen es gibt einen P_5 dessen Endpunkt $v_1 \in K$ ist. Falls v_2 Teil einer Peripherieclique P_j ist, dann gibt es keinen v_3 der nicht adjazent zu v_1 ist. Falls $v_2 \in K$, dann muss v_3 Teil einer Peripherieclique sein. Dann gibt es jedoch keinen Knoten v_4 mehr der nicht zu v_2 adjazent ist.

Angenommen v_1 ist ein Knoten einer Peripherieclique. Falls v_2 Teil der selben Peripherieclique ist, dann gibt es keinen v_3 der nicht zu v_1 adjazent ist. Falls $v_2 \in K$ und v_3 teil einer Peripherieclique ist, dann gibt es keinen v_4 der nicht zu v_2 adjazent ist. Falls $v_2 \in K$ und $v_3 \in K$, dann muss v_4 Teil einer Peripherieclique sein. Dann gibt es jedoch keinen Knoten v_5 der nicht zu v_3 adjazent ist.

Damit ist erschöpfend gezeigt, dass es keinen P_5 in einem synthetischen P_5-freien Graph geben kann. □

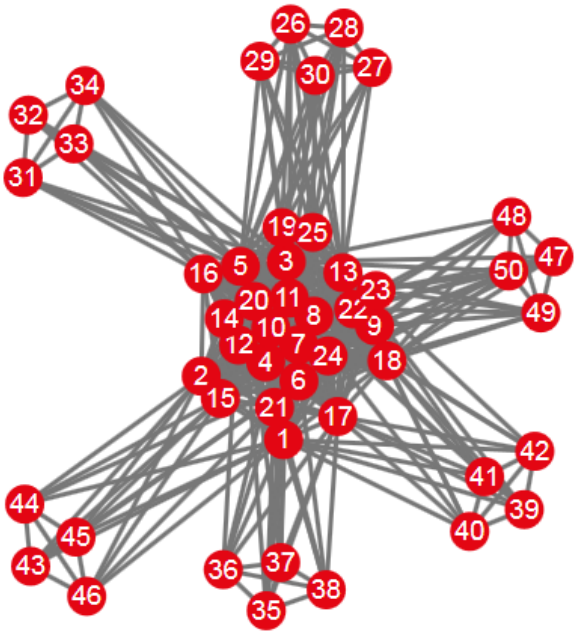

Abbildung 1.3: Beispiel für einen synthetischen P_5-freien Graph mit 50 Knoten

1.4.2 Reale Anwendungsgraphen

Für unsere Versuche haben wir repräsentativ jeweils drei Graphen aus den Sozialwissenschaften [21] sowie der Biologie [22] ausgewählt. Es handelt sich bei den Graphen um sogenannte Interaktionsnetzwerke. Bei den sozialwissenschaftlichen Graphen entspricht eine Kante zwischen zwei Knoten einer Interaktion zwischen zwei Individuen oder Gruppen und bei den biologischen Daten bedeutet eine Kante, dass zwei Proteine

miteinander interagieren. Wir nennen diese Graphen im Folgenden die *Interaktionsgraphen*. Die Tabelle 1.1 enthält einen Überblick der Grapheneigenschaften.

Graph	n	max. Grad	Dichte	max. Ø	max. Radius	# $P_5 s$
Hochlandstämme	16	10	0,48	3	2	323
Karate-Club-Graph	34	17	0,28	5	3	1.583
Jazzmusiker	198	100	0,14	4	4	6.238.440
translation	188	75	0,13	8	4	1.112.733
cell-cycle	196	64	0,04	6	4	268.558
transcription	215	39	0,03	9	5	187.183

Tabelle 1.1: Überblick über die Eigenschaften der Interaktionsgraphen, max. Ø ist der maximale Durchmesser des Graphen.

Zuletzt betrachten wir noch die Menge der *Proteinähnlichkeitsgraphen* [23, 24]. Diese Graphen stammen aus der COG (clusters of orthologous groups of proteins) Datenbank und beschreiben die Ähnlichkeit zwischen Proteinen, welche auf 21 verschiedenen Genomen verschiedenartiger Bakterien kodiert sind. Die Ähnlichkeit zweier Proteine wird als rationaler Wert angegeben, wobei ein positiver Werte für einen hohen Grad an Ähnlichkeit steht. Da unser Modell allerdings für ungewichtete Graphen konzipiert ist, wurden die Daten so uminterpretiert, dass alle positiven Werte als Kanten und alle negativen als Nicht-Kanten modelliert werden. Ähnliche Proteine bilden daher Cliquen, weswegen die Struktur der Proteinähnlichkeitsgraphen denen der synthetischen Clustergraphen ähnelt. Wir verwerfen alle Graphen mit weniger als fünf und mehr als 235 Knoten, um grob etwa die selbe maximale Knotenanzahl wie die anderen Graphklassen zu erhalten. Insgesamt enthält die Menge der Proteinähnlichkeitsgraphen damit 3073 Graphen.

1.4.3 Vergleich der Eigenschaften der Testgraphen

Da die Laufzeit der Algorithmen stark von den Eigenschaften der Graphen abhängig sein wird, werden wir diese im Folgenden kurz diskutieren. Des Weiteren lohnt es sich die Eigenschaften der synthetischen und realen Anwendungsgraphen zu vergleichen, um zu überprüfen, ob die gewählten Parameter angemessen sind. Dazu werden wir die Dichte, den maximalen Durchmesser sowie die Anzahl der enthaltenen P_5s jeweils innerhalb der Graphklassen als auch untereinander vergleichen.

In Abbildung 1.4 zeigt sich, dass die Proteinähnlichkeitsgraphen im Bereich von 1 bis 100 Knoten die höchste Dichte aufweisen. Allerdings streut die Dichte mit zunehmender Knotenanzahl immer stärker mit Tendenz zu einem niedrigeren bis gar zum

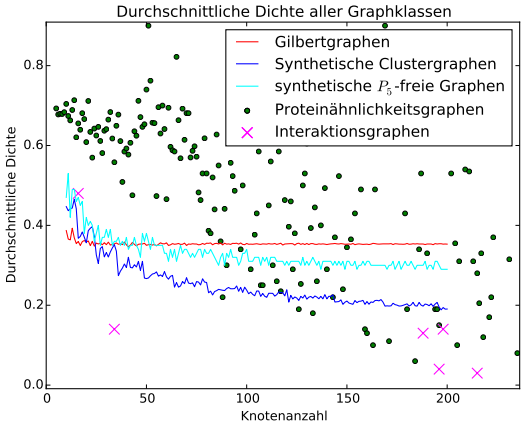

Abbildung 1.4: Durchschnittliche Dichte der Graphen in Abhängigkeit der Knotenanzahl

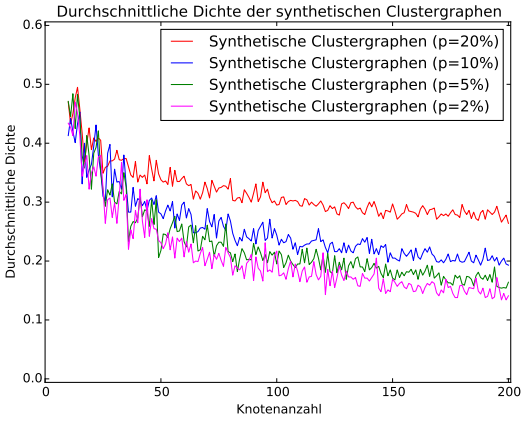

Abbildung 1.5: Durchschnittlichen Dichte in Abhängigkeit der Knotenanzahl für alle Parameter p der synthetischen Clustergraphen

niedrigsten Bereich aller Graphklassen. Die konstante durchschnittlich Dichte der Gilbertgraphen ist aufgrund der Art und Weise ihrer Generierung zu erwarten gewesen, da ihre durchschnittliche Dichte gleich dem verwendeten Parameter p ist. In Abbildung 1.5 ist die Dichte der synthetischen Clustergraphen in Abhängigkeit des Parameters p dargestellt. Es zeigt sich, dass die Graphen eine umso höhere durchschnittliche Dichte aufweisen, desto mehr zufällige Knotenverhältnisse invertiert wurden. Dies lässt sich darauf zurückführen, dass es in Graphen mit einer Dichte von unter 50% wahrscheinlicher ist zufällig eine neue Kante einzufügen als eine bereits existierende zu löschen. Zusammenfassend lässt sich festhalten, dass die Dichte der Graphen durchweg relativ gering ist, wie für Graphen aus realen Anwendungen üblich ist und auf eine gute Wahl der Parameter für die synthetischen Graphen hindeutet. Die geringe Dichte sollte bei der Konstruktion der Algorithmen beachtet werden.

Auffällig ist außerdem der mit steigender Knotenanzahl tendenziell steigende durchschnittliche Durchmesser bei den realen Anwendungsdaten, im Gegensatz zu dem leicht sinkenden und konvergierenden Kurven für die synthetischen Graphen (Abbildung 1.6). Der Durchmesser der Gilbertgraphen in Abhängigkeit der Knotenanzahl konvergiert dabei in Abhängigkeit vom gewählten Parameter p. Für $p = 36\%$ und $p = 51\%$ konvergiert der Durchmesser im Bereich von 50 bis 100 Knoten gegen zwei und für $p = 19\%$ gegen drei. Die Kurve des durchschnittlichen maximalen Durchmessers der synthetischen Clustergraphen in Abhängigkeit der Knotenanzahl zeigt ebenfalls ein konvergierendes Verhalten, wie in Abbildung 1.7 zu sehen.

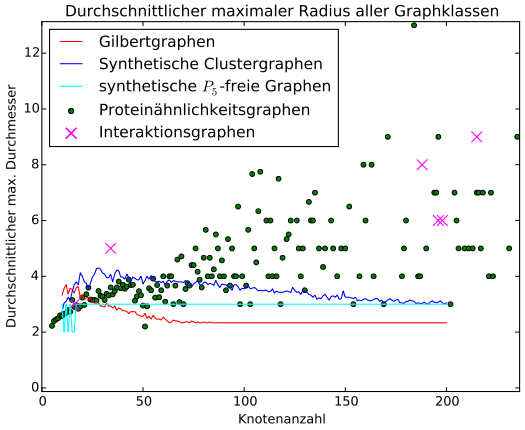

Abbildung 1.6: Durchschnittlicher maximaler Durchmesser der Graphen in Abhängigkeit der Knotenanzahl

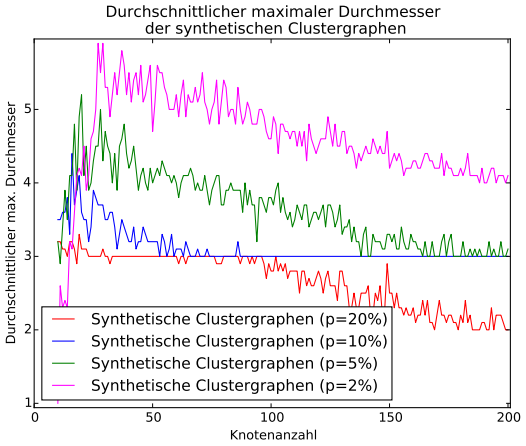

Abbildung 1.7: Durchschnittlicher maximaler Durchmesser in Abhängigkeit der Knotenanzahl für alle Parameter p der synthetischen Clustergraphen

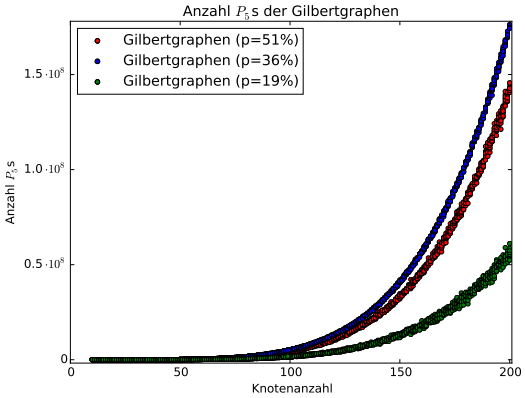

Abbildung 1.8: Vergleich der durchschnittlichen Anzahl an P_5s in Abhängigkeit der Knotenanzahl der Gilbertgraphen für verschiedene Parameter p

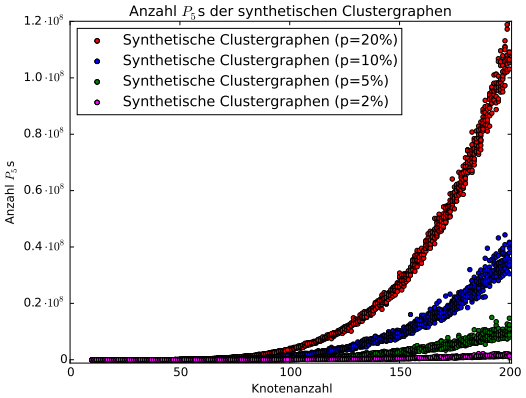

Abbildung 1.9: Vergleich der durchschnittlichen Anzahl an P_5s in Abhängigkeit der Knotenanzahl der synthetischen Clustergraphen für verschiedene Parameter p

Die Eigenschaften der synthetischen Graphen liegen im Durchschnitt der realen Anwendungsgraphen. Allerdings folgen sie nicht immer deren Tendenzen. Der größte Unterschied liegt in der Anzahl der enthaltenen P_5s ab in etwa 100 Knoten. Während die synthetischen Clustergraphen mit $p = 2\%$ beziehungsweise $p = 5\%$ im Durchschnitt der realen Anwendungsgraphen liegen, lässt sich für $p = 10\%$ ein deutlich erhöhtes P_5-Vorkommen erkennen, welches auf dem selben Niveau wie die komplett zufälligen Gilbertgraphen mit 19% Kantenwahrscheinlichkeit liegt. Die Anzahl der enthaltenen P_5s der synthetischen Clustergraphen mit $p = 20\%$ liegen in etwa in der selben Größenordnung wie die der Gilbertgraphen mit $p = 36\%$, beziehungsweise $p = 51\%$ und zeigen

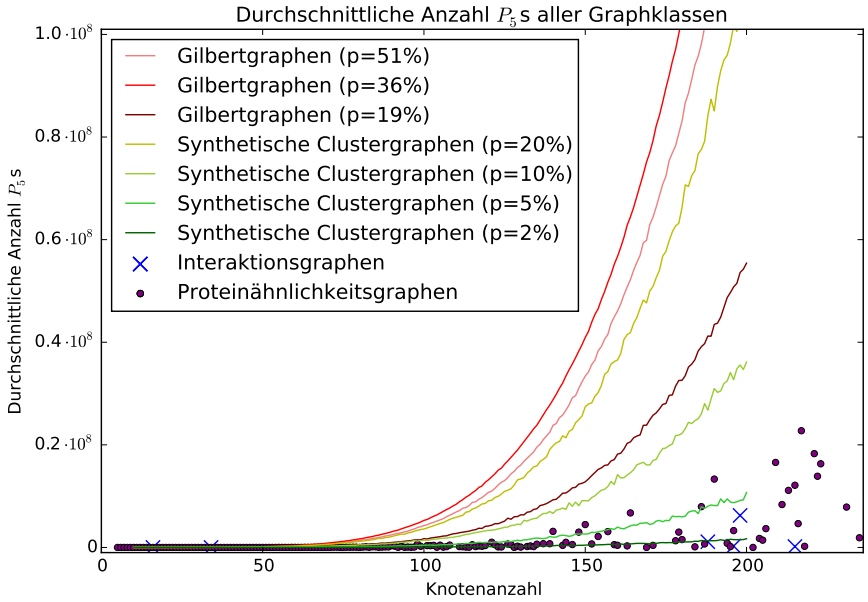

Abbildung 1.10: Übersicht aller Graphklassen über die durchschnittliche Anzahl der enthaltenen P_5s in Abhängigkeit der Knotenanzahl.

ein deutlich anderes Verhalten als die Anwendungsgraphen. Interessant ist, dass die Gilbertgraphen mit $p = 51\%$ weniger P_5s enthalten als die Gilbertgraphen mit $p = 36\%$.

Wir verstehen die Anzahl der P_5s als Indiz für die Ähnlichkeit zu einem P_5-freien Graphen. Daher erwarten wir, dass sich Graphen desto schneller lösen lassen, je weniger P_5s sie enthalten. Daraus folgern wir, dass die Interaktions- und Proteinähnlichkeitsgraphen zusammen mit den die synthetischen Clustergraphen mit $p = 2\%$ am leichtesten zu lösen sein werden und das P_5-frei Modell entsprechend gut zu ihnen passt. Danach folgen mit einigem Abstand die synthetischen Clustergraphen mit $p = 5\%$ und die Gilbertgraphen mit $p = 19\%$. Die letzten drei Graphklassen folgen in einem weiteren, größeren Abstand.

1.5 Implementierungsdetails

Die grundlegende Datenstruktur zur Repräsentation der Graphen wird ebenfalls entscheidende Auswirkungen auf die Laufzeit der verschiedenen Algorithmen haben. Prinzipiell gibt es zwei verschiedene, gängige Ansätze mit jeweils gegenläufigen Vor- und Nachteilen. Für unser Problem eines ungerichteten und ungewichteten Graphen bietet es sich an die Daten entweder in Form einer Adjazenzmatrix oder in Form von Adjazenzlisten zu speichern. Erstere speichert für jedes Knotenpaar, ob dieses adjazent ist und

hat den Vorteil von schnellen Zugriffs- und Berechnungszeiten, um auf einzelne Kanten zuzugreifen. Bei lichten Graphen enthält die Matrix allerdings viele Leerstellen für die Nicht-Kanten. Dies können wir mit Adjazenzlisten vermeiden, weil wir bei diesem Ansatz für jeden Knoten eine Liste mit allen adjazenten Knoten speichern. Dies erlaubt direkt auf die Nachbarschaft eines Knoten zuzugreifen, welches für unsere Algorithmen eine entscheidende Rolle spielt. Allerdings ist es umständlicher direkt auf eine spezifische Kante zuzugreifen, da diese in der Liste der benachbarten Knoten erst gesucht werden muss.

Wir haben uns dazu entschieden den Graphen in Form von Adjazenzlisten zu speichern, da sich diese für viele bekannte Algorithmen als effizienter erweisen und auch für die Suche nach P_5 vermutlich besser geeignet sind, um die durchschnittlich geringe Dichte auszunutzen.

Unsere in Python implementierte Datenstruktur nutzt allerdings zu Gunsten einer höheren Flexibilität eine hinsichtlich der Worst-Case-Laufzeit nicht optimale Hash-Tabelle (Python Dictionary), welche die Knotennamen auf die Menge der adjazenten Knoten (Python Set) abbildet. Davon unbeeinflusst werden wir in den folgenden Laufzeitbetrachtungen die theoretisch besten Laufzeitschranken für Adjazenzlisten verwenden.

Die Testmaschine besitzt einen vierkernigen Intel 4790k Prozessor mit einer Taktrate von 4.0 GHz und 8 MB Cache sowie 8 GB Hauptspeicher. Das Betriebssystem ist Windows 7 Service Pack 1. Sämtlicher Code wurde in Python geschrieben und mit der Python Version 2.7.8 interpretiert.

Kapitel 2

Algorithmen zum Finden von P_5s

Wie bereits im Abschnitt über parametrisierte Algorithmen 1.3 beschrieben ist das Finden der P_5s ein fundamentaler Schritt. Da wir tendenziell eher mit lichten Graphen konfrontiert werden, sollten die Algorithmen versuchen dies auszunutzen. Wir werden im Folgenden drei verschiedene Varianten zum Finden der P_5s betrachten:

1. Algorithmen um einen P_5 zu finden.
2. Algorithmen um alle P_5s zu finden.
3. Algorithmen zur heuristischen Suche mehrerer, aber nicht notwendigerweise aller P_5s.

2.1 Algorithmen zum Finden eines P_5

Zunächst wenden wir uns dem Problem zu, einen beliebigen P_5 in einem Graphen zu finden. Diesen Ansatz können wir bei der Umsetzung des parametrisierten Algorithmus verwenden, um nach jedem aufgelösten P_5 nach einem weiteren P_5 zu suchen. Die in diesem Abschnitt vorgestellten Algorithmen sind in dem Sinne vollständig, dass sie jeden P_5 in einem beliebigen Graphen finden können.

Der *naive* Ansatz, alle Knotenmengen der Größe fünf darauf zu testen ein P_5 zu sein, liefert eine erste obere Laufzeitschranke von $O(n^5)$, welche wir im Folgenden verbessern wollen. In den folgenden Laufzeitbetrachtungen gehen wir davon aus, dass der Graph sowohl als Adjazenzmatrix als auch in Form von Adjazenzlisten vorliegt. Die so aufgestellten, theoretischen Laufzeiten decken sich mit den erwarteten Laufzeiten unserer Python-Implementierungen.

2.1.1 Endpunktsuche

Bei der *Endpunktsuche* betrachten wir jeden Knoten v_1 als potentiellen Grad-1-Knoten eines P_5. Wir versuchen von v_1 aus einen P_5 zu finden, indem wir schrittweise einen Weg aufbauen. Dazu verzweigen wir von v_1 auf alle seine Nachbarn $N(v_1)$. Wir testen für jeden Knoten v_2 aus $N(v_1)$, ob der Weg (v_1, v_2) zu einem P_5 erweitert werden kann. Dafür verzweigen wir nun auf alle Nachbarn des letzten Knotens des Weges u, welche zu keinem Knoten des Weges außer u selbst adjazent sind und fügen sie dem Weg hinzu. Dies wiederholen wir solange bis wir entweder einen Weg der Länge fünf erreicht haben und somit erfolgreich einen P_5 gefunden haben, oder keine Verzweigung einen P_5 ergeben hat.

Algorithmus 1 *Endpunktsuche* Algorithmus zum Finden eines P_5

```
for all v1 in V do
    for all v2 in N(v1) do
        for all v3 in N(v2) \ {v1} do
            if {v1, v3} ∉ E then
                for all v4 in N(v3) \ {v2} do
                    if {{v1, v4}, {v2, v4}} ∩ E = ∅ then
                        for all v5 in N(v4) \ {v3} do
                            if {{v1, v5}, {v2, v5}, {v3, v5}} ∩ E = ∅ then
                                return (v1, v2, v3, v4, v5)
```

Der Algorithmus ist eine Verbesserung des naiven Ansatzes, da wir nun nach jedem neu hinzugenommen Knoten testen, ob die Kombination noch zu einem P_5 führen kann. Der Aufwand liegt allerdings weiterhin im ungünstigsten Fall in $O(n^5)$ Zeit. Da wir allerdings gezielt über die Nachbarknoten verzweigen, können wir die Laufzeit über den maximalen Knotengrad Δ mit $O(n \cdot \Delta^4)$ genauer abschätzen.

Die Laufzeit lässt sich außerdem auch in Abhängigkeit von der Kantenanzahl abschätzen. Die beiden äußersten Schleifen iterieren über die Menge der Kanten $\{\{v_1, v_2\} | v_1 \in V \wedge v_2 \in N(v_1)\}$. Dabei wird jede Kante $\{a, b\}$ des Graphen genau zwei mal betrachtet, einmal für $v_1 = a \wedge v_2 = b$ und umgekehrt. Daraus folgt eine Laufzeit von $O(m)$. Das selbe Argument kann auch auf die dritte und vierte Schleife angewendet werden, wodurch sich zusammen mit der letzten Schleife über alle Nachbarn von v_4 die Laufzeit des Algorithmus mit $O(m^2 \cdot \Delta)$ abschätzen lässt.

2.1.2 Mittelpunktsuche

Bei der *Mittelpunktsuche* betrachten wir jeden Knoten als potentielle Mitte eines $P_5 := G[(v_1, \ldots, v_5)]$. Die zentrale Idee dabei ist zuerst für den Mittelknoten v_3 die Menge $D_1 := N(v_3)$ der Knoten mit Abstand eins zu v_3 sowie die Menge $D_2 := \bigcup\{N(d_1) \mid d_1 \in D_1\} \setminus (D_1 \cup \{v_3\})$ der Knoten mit Abstand zwei zu v_3 zu berechnen. Dadurch wird der Graph wie in Abbildung 2.1 dargestellt aufgeteilt.

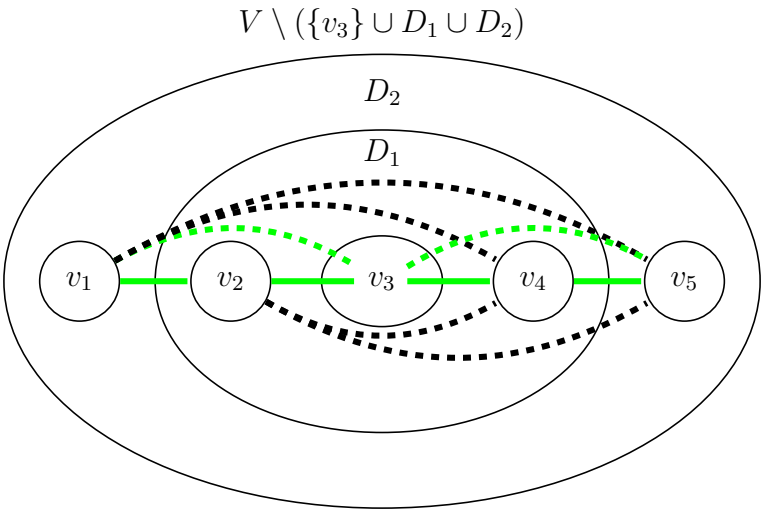

Abbildung 2.1: Bereits abgedeckte Knotenpaare sind grün dargestellt.

Dies kann mit Hilfe einer einfachen, in der Tiefe beschränkten Breitensuche in $O(m)$ Zeit umgesetzt werden. Danach gilt $v_1, v_5 \in D_2$ und $v_2, v_4 \in D_1$ für alle P_5 die v_3 als mittleren Knoten enthalten. Damit lässt sich die P_5-Suche auf das Problem reduzieren zwei Kanten $\{v_1, v_2\}, \{v_4, v_5\} \in E$ zu finden, sodass die Knoten der einen Kante zu den Knoten der anderen Kante paarweise nicht adjazent sind und $v_2, v_4 \in D_1 \wedge v_1, v_5 \in D_2$ gilt, wie in Abbildung 2.2 dargestellt. Man nennt dies auch das $2K_2$-Problem. Das $2K_2$-Problem auf einem Graph G zu lösen ist äquivalent zu der Aufgabe einen Kreis der Länge vier auf der Invertierung von G zu finden, welches in $O(n^{3,376})$ Zeit möglich ist [25]. Daraus folgt für den kompletten Algorithmus eine Laufzeit von $O(n^{4,376})$.

Wir sehen aus praktischen Gründen jedoch von einer Implementierung dieses Algorithmus ab, da er auf einer ausgeklügelten Matrizenmultiplikation beruht, die erst bei sehr großen Matrizen vorteilhaft ist. Stattdessen haben wir den in Algorithmus 2 beschriebenen Algorithmus umgesetzt. Die Idee ist den P_5 vom Mittelknoten her aufzubauen, indem wir zuerst versuchen ein Ende zu finden und dann das andere. Bei der Auswahl der Knoten werden deren Nachbarknoten jeweils aus den Mengen D_1 und D_2 entfernt um den Suchraum für die nächsten Knoten weiter einzuschränken. Die Laufzeit

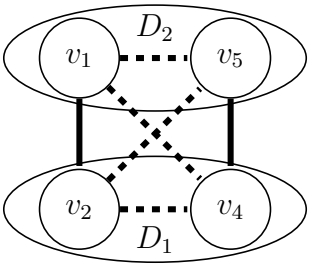

Abbildung 2.2: Reduktion des Problems auf das Finden eines $2K_2$

dieser Variante lässt sich mit $O(n \cdot \Delta^4)$ abschätzen wie in Algorithmus 2 gezeigt wird. Wir können außerdem eine Laufzeitschranke in Abhängigkeit von m aufstellen, indem wir der selben Argumentation wie im Abschnitt der Endpunktsuche folgen. Sie liegt bei $O(m^2 \cdot \Delta)$ Zeit.

Algorithmus 2 *Mittelpunktsuche* Algorithmus zum Finden eines P_5

```
for all v_3 in V do                                          # O(n)
    D_1 ← N(v_3)                                             # O(Δ)
    D_2 ← Knoten mit Abstand 2 zu v_3                        # O(m)
    for all v_2 in D_1 do                                    # O(Δ)
        V_1 ← N(v_2) ∩ D_2                                   # O(n)
        markiere(N(v_2))                                     # O(Δ)
        for all v_1 in V_1 do                                # O(Δ)
            markiere(N(v_1))                                 # O(Δ)
            for all v_4 ← unmarkierter Knoten aus D_1 do     # O(Δ)
                v_5 ← benachbart mit v_4 und unmarkierter aus D_2   # O(Δ)
```

Gesamtlaufzeit: $O(n \cdot \Delta^4)$

2.1.3 Hoàng-Algorithmus

Im Folgenden stellen wir einen 2013 veröffentlichten Algorithmus vor, welcher mit der zur Zeit besten Laufzeit von $O(m^2)$ entscheidet, ob ein Graph P_5-frei ist [25]. Der Algorithmus, ab jetzt nach dem Erstautor *Hoàng* benannt, besteht aus einer äußeren Schleife, die über alle Kanten iteriert, und eine Teilroutine (Algorithmus 3) mit der aktuellen Kante aufruft. Diese testet in $O(m)$ Zeit ob es für zwei benachbarte Knoten v_1 und v_2 einen $P_5 := (v_1, v_2, v_3, v_4, v_5)$ gibt. Die Gesamtlaufzeit liegt daher bei $O(m^2)$. (Es sei an dieser Stelle angemerkt, dass für jede Kante zwischen Knoten v_1 und v_2 sowohl die Kombination (v_1, v_2), als auch (v_2, v_1) durch die Teilroutine getestet werden muss, da ansonsten ein möglicher $P_5 := (v_2, v_1, v_3, v_4, v_5)$ nicht gefunden wird.)

Algorithmus 3 *Hoàng* Algorithmus zum Finden eines P_5

INPUT: $\{v_1, v_2\}$

Require: $\{v_1, v_2\} \in E$

```
B ← V \ N(v_1)          # alle Knoten die nicht zu v_1 adjazent sind
B' ← B \ N(v_2)         # alle Kandidaten für v_4 und v_5
C_1, C_2, ..., C_t ← alle Zusammenhangskomponenten aus G[B']
initialisiere Zähler c_1, ..., c_t mit 0
L ← leere Liste
for all v_3 in N(v_2) ∩ B do        # für alle v_3 Kandidaten
    for all v_4 in N(v_3) ∩ B' do       # für alle v_4 Kandidaten
        j ← Index der Zusammenhangskomponente, die v_4 enthält
        c_j ← c_j + 1       # Anzahl Knoten in C_j zu denen v_3 adjazent ist
        if c_j = 1 then Füge j der Liste L hinzu
                # merke, es gibt mindestens einen v_4 Kandidaten in der Komponente j
    for all j in L do       # *
        if c_j < |C_j| then     # **
            return "yes"        # Der Graph enthält mindestens einen P_5.
        c_j ← 0
    L ← leere Liste
return "no"
```

* Für jede Zusammenhangskomponente die einen v_4 Kandidaten enthält:

** Falls v_3 nicht zu allen Knoten der Zusammenhangskomponente adjazent ist, dann gibt es einen Knoten v_5.

Die Laufzeit $O(m)$ der Teilroutine ergibt sich aus den beiden folgenden Beobachtungen. Die induzierten Zusammenhangskomponenten $C_1, \ldots, C_t$ können in $O(m)$ Zeit berechnet werden. Und für die Schleifen gilt, dass die ersten beiden For-Schleifen über alle Kanten $\{v_2, v_3\}$ iterieren und die dritte For-Schleife wird höchstens so oft ausgeführt wie die Zweite.

Der Algorithmus lässt sich leicht so modifizieren, dass er nicht nur das P_5-frei Problem entscheidet, sondern auch gegebenenfalls einen konkreten P_5 berechnet [25]. Die ersten drei Knoten v_1, v_2 und v_3 sind bereits bekannt. Die induzierte Zusammenhangskomponente C_j lässt sich in zwei nicht leere Teilmengen von Knoten zerlegen. Und zwar in die Mengen der Knoten die adjazent beziehungsweise nicht adjazent zu v_3 sind. Nun muss man nur noch eine Kante zwischen den beiden Teilmengen finden und erhält so die gesuchten Knoten für v_4 und v_5. Eine solche Kante muss existieren, weil C_j zusammenhängend ist.

2.1.4 Evaluierung

Im Folgenden werden wir zunächst die Laufzeiten der einzelnen Algorithmen untereinander vergleichen und anschließend genauer auf die Laufzeiten für die unterschiedlichen Testgraphklassen eingehen.

2.1.4.1 Vergleich der Algorithmen untereinander

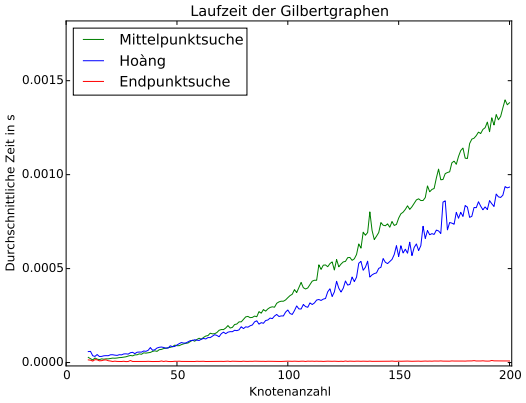

Abbildung 2.3: Durchschnittliche Laufzeit der Gilbertgraphen in Abhängigkeit der Knotenanzahl

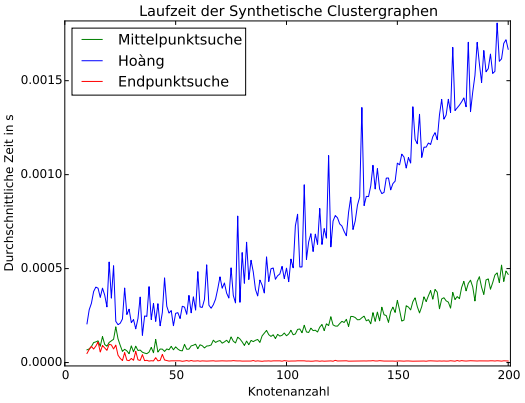

Abbildung 2.4: Durchschnittliche Laufzeit der synthetischen Clustergraphen in Abhängigkeit der Knotenanzahl

Am Auffälligsten ist die nahezu konstante Laufzeit des Endpunktsuche-Algorithmus auf den Gilbertgraphen und den synthetischen Clustergraphen (siehe Abbildungen 2.3, 2.4). Dies lässt sich damit erklären, dass fast alle Knoten Teil eines P_5 sind, weshalb die Endpunktsuche schnell zu einem Ziel gelangt. Die beiden anderen Algorithmen hingegen müssen für ihre Suche Schnittmengen von Knoten berechnen, welches mit zunehmender Knotenanzahl aufwendiger wird. Bei den synthetischen P_5-freien Graphen müssen alle Kombinationen getestet werden, daher hat die Endpunktsuche hier ebenfalls eine zunehmende Laufzeit (siehe Abbildung 2.5).

Für die Klassen der Proteinähnlichkeitsgraphen und der Interaktionsgraphen haben alle drei Algorithmen in etwa die gleiche Zeit benötigt.

Die Klasse der synthetischen P_5-freien Graphen ist für die Analyse der Algorithmen von größerer Bedeutung, da sich die Anzahl der P_5s im Graphen während der Editierung tendenziell verringern und gegen Null laufen wird. Deshalb wird die Laufzeit der Suchalgorithmen während der Editierung ebenfalls steigen und sich der Laufzeit der synthetischen P_5-freien Graphen annähern.

Da die Mittelpunktsuche bei den synthetischen P_5-freien Graphen am Besten abschneidet ist zu vermuten, dass sie am Besten für den parametrisierten Algorithmus ge-

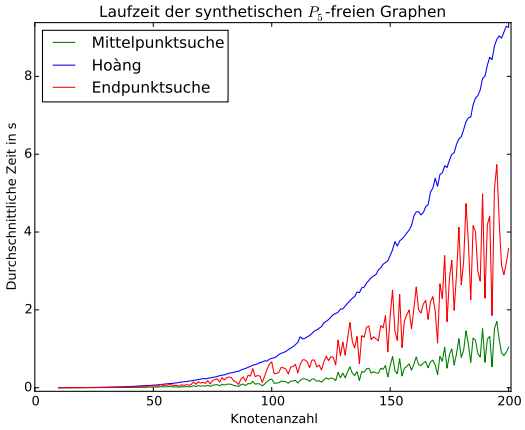

Abbildung 2.5: Durchschnittliche Laufzeit der synthetischen P_5-freien Graphen in Abhängigkeit der Knotenanzahl

eignet ist. Jedoch wird aus den Daten nicht klar, ab welchem P_5-Anteil sich die Laufzeit zu Gunsten der Mittelpunktsuche verschiebt, sodass der anfängliche Geschwindigkeitsvorteil der Endpunktsuche möglicherweise doch den ausschlaggebenden Faktor darstellen könnte.

Der Hoàng-Algorithmus kann auf keiner der Graphklassen positive Resultate vorzeigen. Dies führen wir auf die eher theoretische Natur des Algorithmus zurück, welche für den Worst-Case optimiert ist.

2.1.4.2 Laufzeitvergleich der Graphklassen

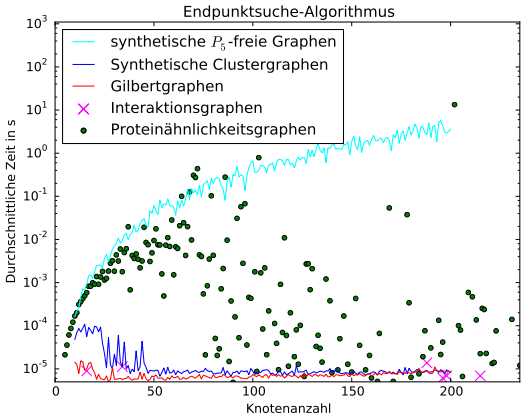

Abbildung 2.6: Durchschnittliche Laufzeit des Endpunktsuche-Algorithmus für alle Graphklassen in Abhängigkeit der Knotenanzahl

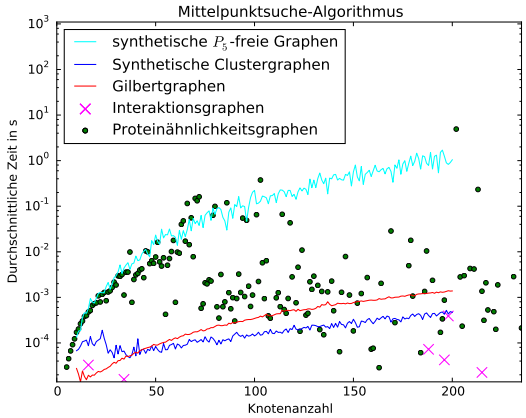

Abbildung 2.7: Durchschnittliche Laufzeit des Mittelpunktsuche-Algorithmus für alle Graphklassen in Abhängigkeit der Knotenanzahl

Wie zu erwarten war, benötigen alle Algorithmen für die synthetischen P_5-freien Graphen im Durchschnitt am längsten. Danach folgen die Proteinähnlichkeitsgraphen, welche jedoch ab etwa 100 Knoten durchschnittlich dieselbe Laufzeit aufweisen wie die synthetischen Clustergraphen und die Gilbertgraphen. Auffällig ist, dass einzig beim Mittelpunktsuchalgorithmus die Laufzeit der Gilbertgraphen die der synthetischen Clustergraphen übersteigt (siehe Abbildung 2.7). Die Wahl des Parameters p bei den synthetischen Graphen spiegelt sich in allen Fällen in einem nahezu konstanten zusätzlichen Zeitaufwand entsprechend der Anzahl an P_5s wider. Graphen mit mehr P_5s haben eine kürzere Laufzeit.

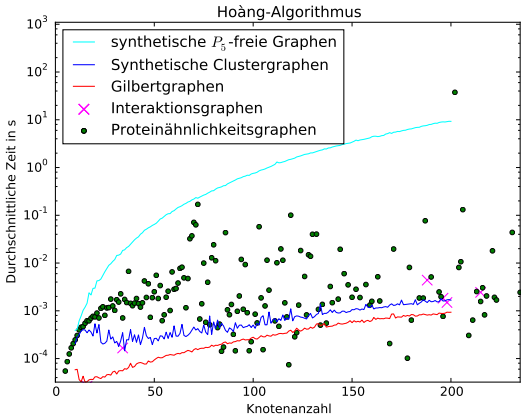

Abbildung 2.8: Durchschnittliche Laufzeit des Hoàng-Algorithmus für alle Graphklassen in Abhängigkeit der Knotenanzahl

2.2 Algorithmen zum Finden aller P_5s

Alle P_5s eines Graphen aufzulisten kann nicht schneller als in $O(n^5)$ Zeit geschehen, weil es in der Größenordnung n^5 viele P_5s in einem Graphen geben kann. Als Beispiel für einen solchen Graphen betrachten wir einen fünf-partiten Graph, den wir folgendermaßen konstruieren: Alle Knoten aus der ersten Partition sind mit allen aus der zweiten Partition verbunden, alle aus der zweiten Partition mit allen aus der Dritten und so weiter. Dann ist jeder induzierter Teilgraph, welcher je genau einen Knoten aus jeder Partition enthält, ein P_5. Dem zu Folge enthält ein solcher Graph in der Größenordnung von $O((n/5)^5)$ viele P_5s. Eine genauere obere Abschätzung für die Anzahl an P_5 in Graphen liegt bei $O(n \cdot m^2)$, welche überdies in $O(n \cdot m^2)$ Zeit aufgelistet werden können [25].

Die Algorithmen zum Finden eines P_5 lassen sich leicht anpassen, sodass sie anstatt den ersten gefunden P_5 zurückzugeben eine Liste von P_5s erstellen. Da alle genannten Algorithmen zum Finden eines P_5 vollständig sind, also jeden P_5 finden, folgt dass die wie oben beschrieben modifizierten Versionen alle im Graphen enthaltenen P_5 auflisten.

An dieser Stelle sei angemerkt, dass obwohl der Hoàng-Algorithmus eine Laufzeit von $O(m^2)$ hat, er nicht alle P_5s in selbiger Laufzeit aufzählen kann, da wir für jeden gefundenen P_5 noch dessen letzten beiden Knoten berechnen müssen. Daher hat der Hoàng-Algorithmus sogar eine schlechtere Laufzeit von $O(m^3)$.

Die zur Laufzeit generierten P_5-Listen werden mitunter so groß, dass Python einen Speicherzugriffsfehler wirft. Für jede Graphklasse wurde das Benchmark abgebrochen, sobald 15 Graphen in Folge mit einem Speicherzugriffsfehler terminierten. Da die Anzahl der P_5s bei den synthetischen Graphen sehr stark vom Parameter p abhängen, wurden für jede Variante des Parameters eine eigene Testreihe mit der selben Abbruchbedingung durchgeführt.

Die Laufzeit des Hoàng-Algorithmus ist etwa um den Faktor 30 höher als die der beiden anderen Algorithmen, weswegen wir ihn nicht weiter diskutieren werden.

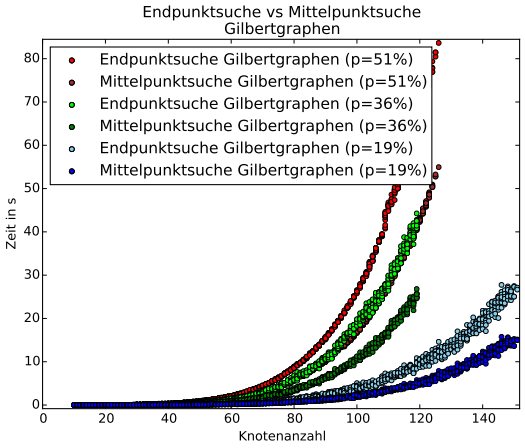

Abbildung 2.9: Laufzeitvergleich zum Finden aller P_5s der Gilbertgraphen für Endpunktsuche und Mittelpunktsuche

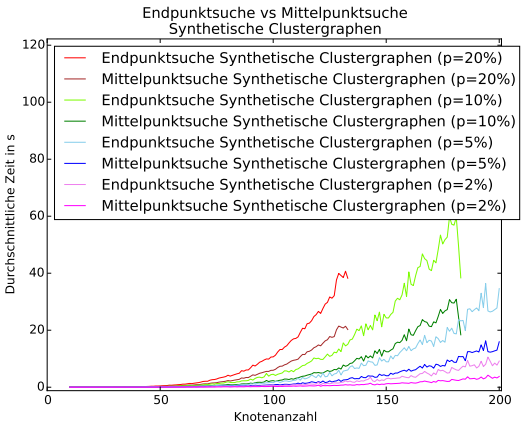

Abbildung 2.10: Laufzeitvergleich zum Finden aller P_5s der synthetischen Clustergraphen für Endpunktsuche und Mittelpunktsuche

Bei etwa 14 Millionen gefundenen P_5s wirft Python einen Speicherzugriffsfehler. Dies ist der Grund warum in der Abbildung der Gilbertgraphen (2.9) und synthetischen Clustergraphen (2.10) manche Kurven bereits frühzeitig abbrechen. Als Ergebnis können wir festhalten, dass die Mittelpunktsuche auf allen Graphklassen in etwa nur die halbe Zeit benötigt um alle P_5s aufzulisten und damit der bevorzugte Algorithmus zum Aufzählen aller P_5s ist.

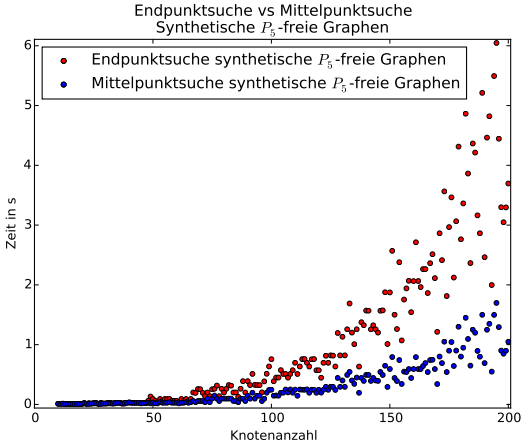

Abbildung 2.11: Laufzeitvergleich zum Finden aller P_5s der synthetischen P_5-freien Graphen für Endpunktsuche und Mittelpunktsuche

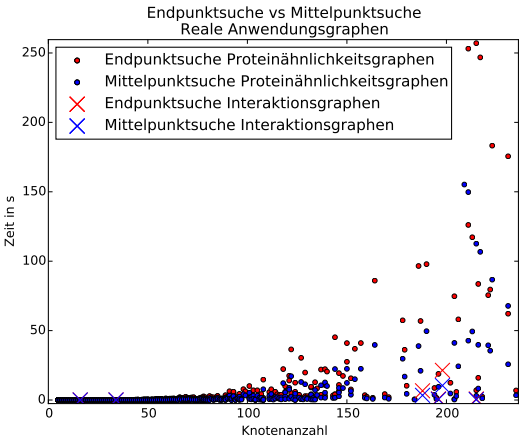

Abbildung 2.12: Laufzeitvergleich zum Finden aller P_5s der realen Anwendungsgraphen für Endpunktsuche und Mittelpunktsuche

2.3 Algorithmen zur heuristischen Suche mehrerer P_5s

In diesem Abschnitt werden wir uns mit Algorithmen befassen, welche zugunsten einer kürzeren Laufzeit nicht garantieren können, dass sie alle P_5s in einem Graphen finden. Ein naheliegender Ansatz ist es, die kürzesten Wege zwischen allen Knoten zu berechnen. Bekannte Algorithmen für diese Aufgabe sind der Dijkstra-Algorithmus und der Bellman-Ford-Algorithmus sowie der Algorithmus von Floyd und Warshal. In unserem Spezialfall von ungerichteten und ungewichteten Kanten ist jedoch eine einfache Breitensuche bereits ausreichend. Jeder kürzeste Weg der Länge fünf ist ein P_5. Dieser Ansatz hat eine im Vergleich sehr geringe Laufzeitklasse von $O(n \cdot m)$, allerdings findet er nicht alle P_5s. Ein nicht P_5-freier Beispielgraph, in welchem kein P_5 gefunden wird, ist ein Kreis aus sechs Knoten. Allgemein können mit diesem Ansatz keine P_5s in Graphen gefunden werden, deren Durchmesser kleiner gleich drei ist. In Abbildung 1.6 wird deutlich, dass die synthetischen Testgraphen im Schnitt einen Durchmesser zwischen drei und vier haben und daher dieser Ansatz nicht geeignet ist. Jedoch haben viele der realen Anwendungsgraphen einen größeren Durchmesser. Ein weiterer Schwachpunkt dieses Ansatzes liegt darin, dass P_5-freie Graphen einen Durchmesser von drei oder weniger haben, weshalb er nicht für Graphen geeignet ist, welche bereits eine ähnliche Struktur wie P_5-freie Graphen aufweisen.

Auf keiner der Graphklassen kann die Breitensuche ein Ergebnis nahe der korrekten P_5-Anzahl liefern (Abbildungen 2.13, 2.14, 2.15). Die korrekte Lösung für die Teilklasse mit der niedrigsten Anzahl an P_5s wurde als Vergleich ebenfalls dargestellt. Für die

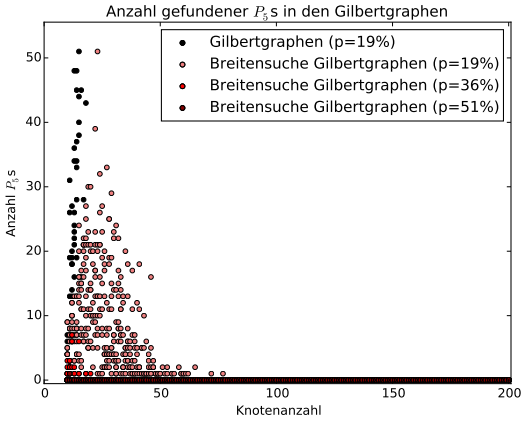

Abbildung 2.13: Anzahl der gefundenen P_5s in den Gilbertgraphen

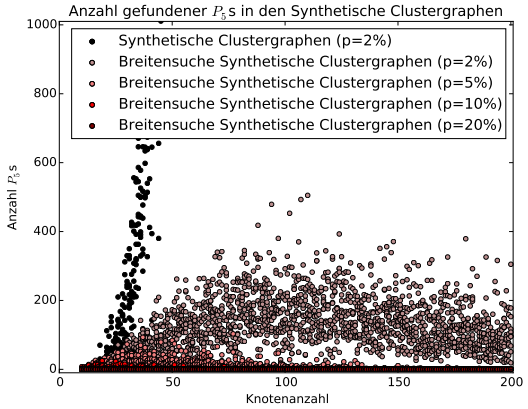

Abbildung 2.14: Anzahl der gefundenen P_5s in den synthetischen Clustergraphen

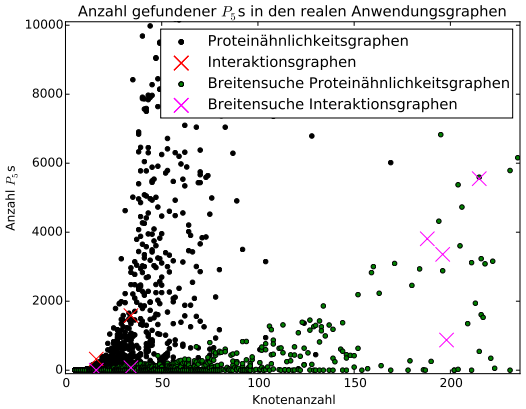

Abbildung 2.15: Anzahl der gefundenen P_5s in den Anwendungsgraphen

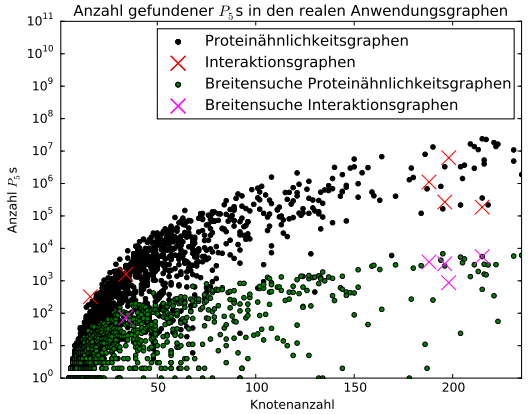

Abbildung 2.16: Logarithmische Anzahl der gefundenen P_5s in den Anwendungsgraphen, falls mindestens ein P_5 gefunden wurde.

synthetischen Clustergraphen mit $p = 2\%$ kann für alle Graphen eine kleine Teilmenge der P_5s gefunden werden. Bei allen anderen synthetischen Graphen sinkt, wie bereits beschrieben, der durchschnittliche Durchmesser mit zunehmender Knotenanzahl auf drei, sodass ab diesem Zeitpunkt mit der Breitensuche keine nichtleeren Teillösungen mehr gefunden werden können. Für viele der Proteinähnlichkeitsgraphen kann ebenfalls eine, im Vergleich zu den synthetischen Clustergraphen mit $p = 2\%$, relativ große Teilmenge gefunden werden. In Abbildungen 2.16 haben wir die Anzahl der gefunden P_5s für die Anwendungsgraphen mit einer logarithmischen Skala dargestellt, um einen besseren Vergleich zu der optimalen Lösung zu ermöglichen.

Kapitel 3

Lösungsalgorithmus

In diesem Kapitel widmen wir uns dem Suchbaumalgorithmus zum Finden einer Lösung für das optimale P_5-frei-Editierungsproblem sowie dessen Verbesserung mit Hilfe unterer Schranken für den Parameter k, Datenreduktionsregeln, Annotationsregeln und einigen weiteren Ansätzen. Wir werden außerdem unsere Implementierungen auf die Testgraphen anwenden, um zu ermitteln wie gut sich die einzelnen Graphklassen im Vergleich lösen lassen, für welche optimale Lösungsgrößen wir eine Lösung berechnen können und wie sich unsere Verbesserungen auf dieses Ergebnis auswirken.

3.1 Suchbaumalgorithmus

Wir sind in Kapitel 1 bereits auf parametrisierte Algorithmen eingegangen und darauf wie wir mit Hilfe eines einfachen Suchbaumes das P_5-frei-Editierungsproblem für einen Graphen G und eine natürliche Zahl k entscheiden können. Nun gehen wir genauer darauf ein, wie wir mit diesem Ansatz eine optimalen Menge von Editierungen finden können, um einen gegebenen Graph zu einem P_5-freien Graph zu transformieren.

Problem 2. OPTIMALES P_5-FREI-EDITIERUNGSPROBLEM
Eingabe: Ein ungerichteter Graph G.
Ausgabe: Eine optimale Menge L von Knotenpaaren, sodass der Graph $G \triangle L$ keinen induzierten P_5 enthält.

Um das Problem der optimalen P_5-frei-Editierung zu lösen, verwenden wir den *Suchbaumalgorithmus*. Dieser findet eine optimale Lösung für einen Graphen G, indem er mit schrittweise steigendem Parameter j versucht das P_5-frei-Editierungsproblem (G, j) zu entscheiden, bis die erste positive Antwort gefunden wird.

Der in Abschnitt 2.1.3 vorgestellte Hoàng-Algorithmus zum finden eines P_5 hat eine Laufzeit von $O(m^2)$, woraus sich zusammen mit der maximalen Suchbaumgröße für ein gegebenes k eine Laufzeit von $O(10^k \cdot m^2)$ für den Suchbaumalgorithmus ergibt.

3.2 Untere Schranken

Eine untere Schranke versucht die notwendige Anzahl an Editierungen zu bestimmen, welche für eine optimale Lösung eines P_5-frei-Editierungsproblems notwendig sind. Wir definieren den Begriff der unteren Schranke folgendermaßen:

Definition 3.1. *Sei $k(G)$ die kleinste natürliche Zahl, welche für ein P_5-frei-Editierungsproblem von G eine positive Antwort zurückgibt. Dann ist jede natürliche Zahl $s \leq k(G)$ eine* untere Schranke *für die Probleme (G, j) mit $j \in \mathbb{N}$.*

Eine untere Schranke s für die Größe einer optimalen Lösung erlaubt uns zum einen am Anfang des Suchbaumalgorithmus zum optimalen Lösen des p_5-frei-Editierungsproblems unnötige Lösungsversuche für (G, j) mit $j < s$ zu überspringen und zum anderen im Suchbaumalgorithmus frühzeitig abzubrechen, falls wir mit der verbliebenen Editierungsanzahl keinen P_5-freien Graph mehr erzeugen können. Daraus folgt, dass untere Schranken die tatsächliche Kantenänderungsanzahl unterschätzen, aber nicht überschätzen dürfen, weil sonst durch ein vorzeitiges Beschneiden des Suchbaumes mögliche Lösungen verloren gingen.

In diesem Abschnitt werden wir drei *Packing-basierte* Ansätze zur Bestimmung unterer Schranken für das P_5-frei-Editierungsproblem vorstellen.

Die Idee des Packing ist eine Menge von P_5s zu finden, die jeweils einzeln aufgelöst werden müssen. Dafür führen wir zunächst den Begriff des Packing formal ein.

Definition 3.2. *Eine Menge Q von P_5s ist ein* Packing *für einen Graphen G genau dann, wenn jeder P_5 in Q ein induzierter Teilgraph von G ist und für alle P_5s gilt, dass sie paarweise keine gemeinsame Kante oder Nicht-Kante enthalten.*

Wir beweisen zunächst, dass folgende untere Schranke gilt:

Lemma 3.1. *Die Anzahl der Elemente s eines Packings Q für G ist eine untere Schranke für alle P_5-frei-Editierungsprobleme (G, j) mit $j \in \mathbb{N}$.*

Beweis. Weil ein P_5 nur durch das Invertieren eines der Verhältnis zwischen zwei seiner Knoten aufgelöst werden kann und alle P_5s eines Packings paarweise keine gemeinsames Kante oder Nicht-Kante enthalten, löst Editierung im Graphen genau einen P_5 des

Packings auf. Damit gilt, dass $s = |Q|$ eine untere Schranke für alle P_5-frei Editierungsprobleme (G, j) mit $j \in \mathbb{N}$ ist. □

Es kann für einen Graph mehrere gültige Packings mit verschieden vielen Elementen geben. Ein Packing für einen Graph mit n Knoten und m Kanten kann nicht größer als $m/4$ und nicht größer als $\binom{n}{2}/10$ sein. Das Problem für einen Graphen G eine größtmögliches Menge von gleichen Teilgraphen T zu finden, deren Kanten paarweise disjunkt sind, wird in der Literatur als Kanten-Packing-Problem oder kurz $\mathrm{EPack_G}$ bezeichnet [26]. Falls G ein Baum ist, kann das Problem in Polynomzeit gelöst werden. Für allgemeine Graphen und Teilgraphen ist das Problem jedoch $\mathcal{NP}$-vollständig [26], insbesondere auch falls T ein P_5 ist.

Daher werden wir heuristische Algorithmen konstruieren, welche zu Gunsten einer kürzeren, polynomiellen Laufzeit eventuell nicht die größte untere Schranke bestimmen.

3.2.1 Knotendisjunktes Packing

Als erstes verfolgen wir die Idee ein Packing aus P_5s zu bilden, sodass alle P_5s paarweise keinen gemeinsamen Knoten enthalten. Dazu modifizieren wir den Endpunktsuche-Algorithmus aus Kapitel 2, indem wir jeden gefundenen P_5 speichern und beim Verzweigen nicht nur Knoten verwerfen, die zu einem anderen Knoten auf dem bisherigen Weg adjazent sind, sondern auch alle die Teil eines bereits gefundenen P_5 sind. Nachdem wir einen P_5 gefunden und dem Packing hinzugefügt haben müssen wir außerdem einen neuen Startknoten für die P_5-Suche wählen, da der aktuelle Startknoten Teil eines P_5s des Packings ist.

Ein knotendisjunktes Packing kann höchstens $n/5$ viele Elemente enthalten, weswegen die höchstmögliche untere Schranke, die der Algorithmus zurückgeben kann, ebenfalls durch diese Anzahl beschränkt ist. Aus Abbildung 3.1 wird ersichtlich, dass unser heuristischer Ansatz, besonders für die synthetischen Graphen, gute Ergebnisse liefert, die sehr nahe an dieser Grenze liegen. Bezüglich des Parameters p der synthetischen Graphen gibt es kleine Unterschiede, die mit der Anzahl der enthaltenen P_5s korrelieren.

3.2.2 Knotenpaardisjunktes Packing

Um ein gültiges Packing zu erhalten, müssen nicht alle P_5s knotendisjunkt sein. Falls alle P_5s paarweise höchstens einen Knoten gemeinsam haben, dann gibt es auch keine

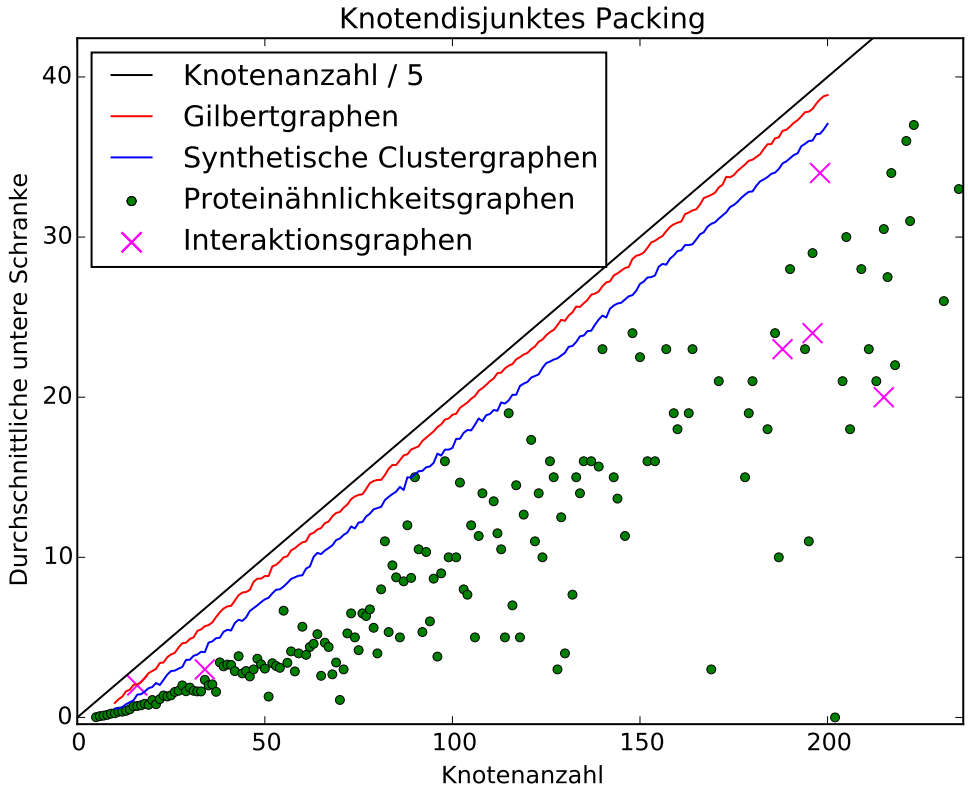

Abbildung 3.1: Aus der Größe der berechneten Packings ergeben sich die folgenden durchschnittlichen unteren Schranken für das knotendisjunkte Packing, dargestellt für alle Graphklassen.

Kante oder Nicht-Kante, die in mehr als einem P_5 enthalten ist. Mit dieser Beobachtung können wir den Ansatz des knotendisjunkten Packings verbessern.

Der Algorithmus 4 besteht aus zwei Teilen. Die erste Funktion initialisiert die Parameter für die zweite rekursive Funktion. Wir benötigen vier Variablen: Einen Zähler für die Anzahl der aktuell gefundenen P_5s, am Anfang gleich Null, eine Hashtabelle, die jeden Knoten aus V auf eine Menge von natürlichen Zahlen abbildet, in welcher wir die Nummern der gefunden P_5 speichern, in welchen der Knoten enthalten ist. Außerdem benötigen wir eine Menge mit markierten P_5s, welche ebenfalls lediglich deren Nummern enthält und am Anfang ebenfalls leer ist. Letztlich benötigen wir noch eine Liste in der wir den aktuell untersuchten Weg speichern, welche ebenfalls am Anfang leer initialisiert wird.

Die zweite Funktion arbeitet grundlegend wie der Endpunktsuche-Algorithmus zum Finden aller P_5s. Von einem Startknoten aus versuchen wir einen Weg der Länge fünf zu finden. Für jeden neu explorierten Knoten kommt eine Abfrage hinzu, ob er sich bereits in einem P_5 mit einem anderen Knoten aus dem aktuellen Weg befindet. Falls nicht

wird er dem aktuellen Weg hinzugefügt und die Menge der markierten P_5s wird um die Menge der P_5s erweitert, in denen der Knoten vorkommt. Für diese Abfrage verwenden wir die Hashtabelle. Falls wir während des Explorierens in eine Sackgasse geraten und den letzten Knoten des Weges wieder entfernen, dann müssen wir ebenfalls die Menge der markierten P_5s aktualisieren. Wenn wir einen P_5 finden, dann muss die Hashtabelle aktualisiert werden, indem wir für alle Knoten des P_5 ihrer jeweiligen P_5-Liste den neuen P_5 hinzufügen.

Wir bezeichnen diesen Algorithmus als *rekursives Packing*. Der hier dargestellte Algorithmus ist als Konzept zu verstehen. Die in den Versuchen verwendete Implementierung ist hinsichtlich der Laufzeit optimiert worden. Der verwendete Python-Code befindet sich ebenfalls auf dem beigelegten Datenträger.

Algorithmus 4 *Kantendisjunktes Packing* Algorithmus zum Berechnen einer unteren Schranke für das P_5-frei-Editierungsproblem von G

function REKURSIVESPACKING(G)
  P5Zähler ← 0 # Nummerierung der gefundenen P_5s
  KnotenP5s ← $\{n : \emptyset\} \forall n \in V$ # Hashtabelle: Knoten → Menge mit P_5-Nummern
  markierteP5s ← $\emptyset$ # Menge von natürlichen Zahlen
  Weg ← Leere Liste # Liste von Knoten aus G
  WEGERWEITERN(G, Weg, markierteP5s, KnotenP5s, P5Zähler)
  return P5Zähler
function WEGERWEITERN(G, Weg, markierteP5s, KnotenP5s, P5Zähler)
  for all v in V **do**
    Weg ← Weg + v # Füge v als letzten Knoten des Weges an
    if G[Weg] ist ein $P_{|\text{Weg}|} \wedge \left(\bigcup_{n \in Weg} \text{KnotenP5s[n]}\right) \cap \text{markierteP5s} = \emptyset$ **then**
      if Weg hat Länge 5 **then**
        for all $n \in$ Weg **do**: KnotenP5s[n] ← KnotenP5s[n] ∪ P5Zähler
        P5Zähler ← P5Zähler + 1
      else:
        NeuMarkierteP5s ← KnotenP5s[v] \ markierteP5s
        markierteP5s ← markierteP5s ∪ NeuMarkierteP5s
        WEGERWEITERN(G, Weg, markierteP5s, KnotenP5s, P5Zähler)
        markierteP5s ← markierteP5s \ NeueMarkierteP5s # Backtrack
    Weg ← Weg - v # Entferne v aus dem Weg

Für das rekursive Packing fällt die obere Schranke von $n/5$ des knotendisjunkten Packing weg, weil jeder Knoten jetzt in mehreren P_5s enthalten sein kann. Dies spiegelt sich auch in dem deutlich höheren gefundenen unteren Schranken wieder, wie in Abbildung 3.2 zu erkennen ist.

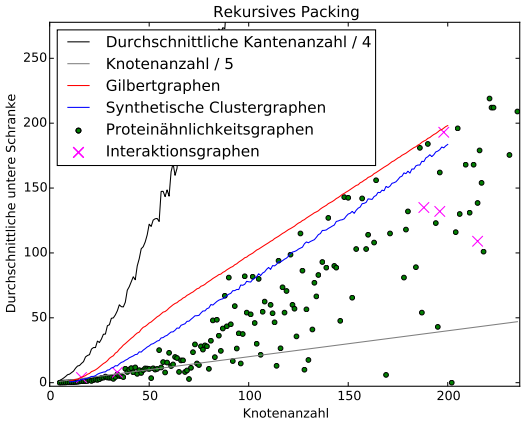

Abbildung 3.2: Untere Schranken des rekursiven Packings für alle Graphklassen

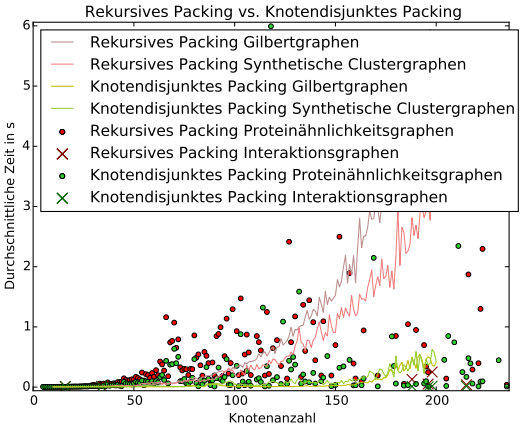

Abbildung 3.3: Laufzeitvergleich zwischen rekursiven und knotendisjunkten Packing

Allerdings steigt auch die Laufzeit etwa um den Faktor 10 im Vergleich zum knotendisjunkten Packing an, was sich besonders bei Graphen mit vielen Knoten bemerkbar macht. Dieser zeitliche Mehraufwand ist jedoch mehr als gerechtfertigt in Anbetracht der deutlich besseren Leistung und der daraus folgenden exponentiellen Zeitersparnis für den Suchbaumalgorithmus.

Die Laufzeitabbildung 3.3 ist ein Ausschnitt, es gibt einige wenige Ausreißer bei den Proteinähnlichkeitsgraphen. Der gemessene Höchstwert liegt bei 40 Sekunden. Bezüglich des Parameters p der synthetischen Graphen gibt es für das rekursive Packing, ähnlich wie beim knotendisjunkten Packing, kleine konstante Abweichungen, die mit der Anzahl der enthaltenen P_5s korrelieren.

3.2.3 Hitting-Set-Approximation

Bisher haben wir den Ansatz verfolgt eine Menge von knotenpaardisjunkten P_5s im Graphen zu suchen. Ein weiterer Ansatz ist es zunähst alle P_5s im Graphen aufzulisten und dann zu bestimmen wie viele Editierungen mindestens benötigt werden, um alle diese P_5s aufzulösen. Diese Aufgabe kann auf das 10-Hitting-Set-Problem reduziert werden, indem wir für jeden P_5 ein Set mit allen Knotenpaaren des P_5 konstruieren. Das Hitting-Set-Problem gehört allerdings zu den 21 klassischen $\mathcal{NP}$-vollständigen Problemen und lässt sich damit nicht in effizienter Zeit berechnen. Daher werden wir uns in diesem Abschnitt mit approximativen Lösungen des Hitting-Set-Problems beschäftigen und Möglichkeiten daraus untere Schranken für das P_5-frei-Editierungsproblem zu berechnen.

Zunächst müssen wir uns allerdings mit dem Problem befassen, dass alle heuristischen Lösungen des Hitting-Set-Problems größer oder im besten Fall gleich der optimalen Lösung sind, aber eine untere Schranke für das P_5-frei-Editierungsproblem die notwendigen Änderungen nicht überschätzen darf. Daher können wir nur Heuristiken verwenden, welche einen Approximationsfaktor besitzen. Dieser gibt an, um welchen Faktor die heuristische Lösung eines Algorithmus höchsten von der optimalen Lösung abweicht. In der Literatur gibt es zwei weit verbreitete heuristische Approximationen für das Hitting-Set-Problem [27]. Wir werden im Folgenden kurz auf drei Varianten eingehen, welche aber alle dem selben Schema folgen:

- Sei G ein Graph, M eine leere Menge.

1. Wir zählen alle P_5s auf. (Dazu verwenden wir den Mittelpunktsuche-Algorithmus aus Kapitel 2.)

2. Für jeden P_5 fügen wir die zehnelementige Menge aller Knotenkombinationen des P_5 der Menge M hinzu.

3. Sei s die Größe einer approximativen Lösung des Hitting-Set-Problems auf M.

4. Dann ist $\lceil s/$ (Approximationsfaktor des Algorithmus)$\rceil$ eine untere Schranke für das P_5-frei-Editierungsproblem von G.

Der erste Algorithmus wählt immer das Knotenpaar, welches in den meisten Sets enthalten ist und fügt es der Lösung hinzu. Für diesen Algorithmus wurde ein Approximationsfaktor von $\ln |M| + 1$ bewiesen. Die Strategie in jedem Schritt den Folgezustand zu wählen, welcher zu diesem Zeitpunkt am Erfolgversprechendsten ist, wird häufig als ”greedy” bezeichnet. Daher nennen wir diesen Algorithmus *HS-greedy*.

Die unteren Schranken des HS-greedy-Algorithmus sind durchweg sehr gering im Vergleich zum rekursiven Packing (siehe Abbildung 3.4). Dies lässt sich darauf zurückführen, dass die greedy Approximation des Hitting-Sets allgemein recht gute Ergebnisse liefert. Ironischer Weise benötigen wir jedoch möglichst schlechte Ergebnisse, um eine höhere untere Schranke zu erhalten. Da wir zunächst erst alle P_5s berechnen müssen, ist die Laufzeit von HS-greedy erwartungsgemäß deutlich höher, dies ist in Abbildung 3.5 deutlich zu erkennen. Interessant ist auch, dass die Laufzeiten des HS-greedy-Algorithmus für die synthetischen Graphen deutlich stärker vom Parameter p abhängen und sich mehrere Ausläufer auffächern, während die Laufzeiten der Graphen für das rekursive Packing eine kompakte Struktur bilden, auf der sie scheinbar gleichmäßig verteilt sind.

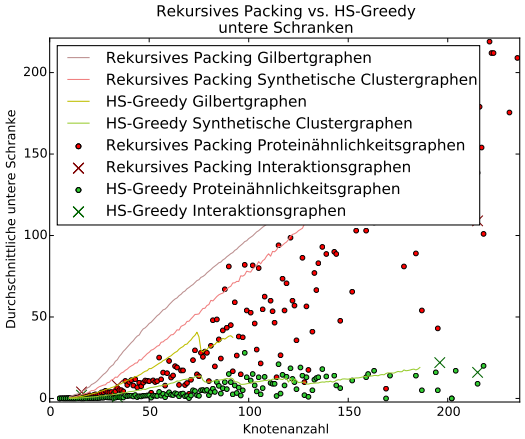

Abbildung 3.4: Vergleich der gefundenen unteren Schranken zwischen rekursiven Packing und HS-greedy

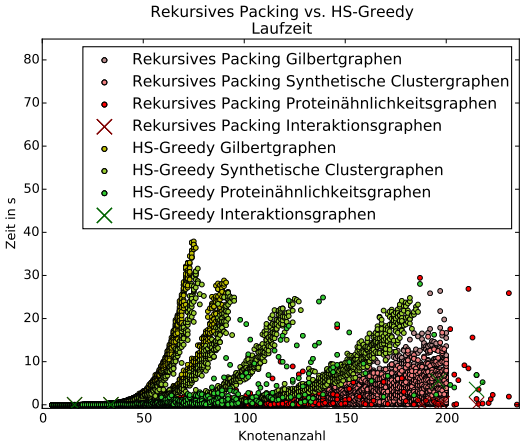

Abbildung 3.5: Laufzeitvergleich zwischen rekursiven Packing und HS-greedy

Der nächste Ansatz wählt ein zufälliges Set aus und fügt alle enthaltenen Elemente der Lösung hinzu. Dieser Ansatz hat einen Approximationsfaktor von 10. Wir bezeichnen ihn daher als *10-HS*.

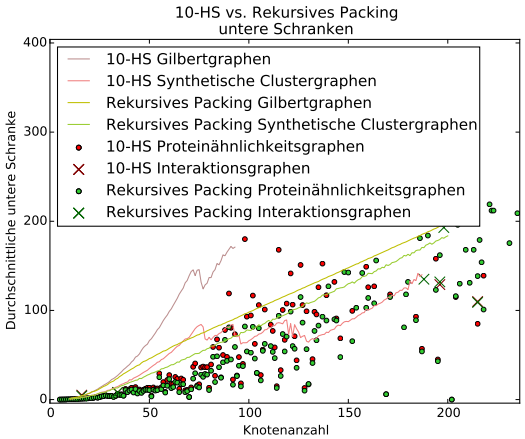

Abbildung 3.6: Vergleich der gefundenen unteren Schranken zwischen 10-HS und rekursiven Packing

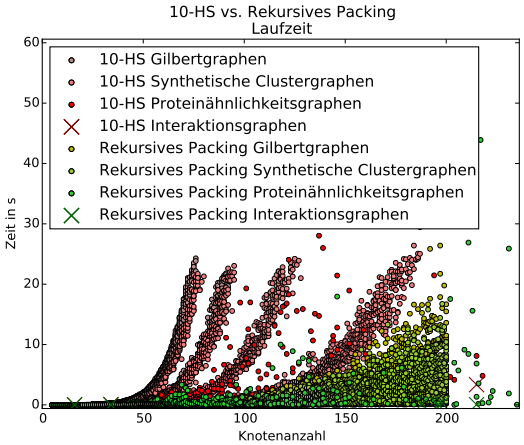

Abbildung 3.7: Laufzeitvergleich zwischen 10-HS und rekursiven Packing

Der 10-HS-Algorithmus kann deutlich bessere Ergebnisse als das rekursive Packing erzielen (siehe Abbildung 3.6), hat aber auch eine im Vergleich höhere Laufzeit (siehe Abbildung 3.7). Problematisch ist jedoch, dass aufgrund des hohen Speicherverbrauchs für das Hitting-Set bereits sehr frühe Abbrüche aufgrund von Speicherzugriffsfehlern entstehen. Dies ist auch der Grund für das anscheinend schlechtere Abschneiden auf den synthetischen Clustergraphen. Dieses Problem tritt ebenfalls bereits beim HS-greedy-

Algorithmus auf, hat aufgrund der relativ schlechten Resultate aber kaum sichtbare Auswirkung auf den Verlauf der Kurve.

Der Ansatz der 10-HS-Approximation lässt sich verbessern, indem wir nicht ein zufälliges Set auswählen, sondern das Set, dessen Kanten in den wenigsten anderen Sets enthalten sind. Aufgrund dieses Auswahlverfahrens bezeichnen wir diesen Ansatz als *greedy-10-HS*.

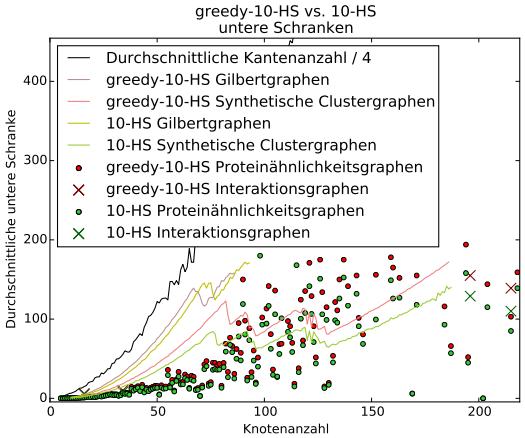

Abbildung 3.8: Vergleich der gefundenen unteren Schranken zwischen greedy-10-HS und 10-HS

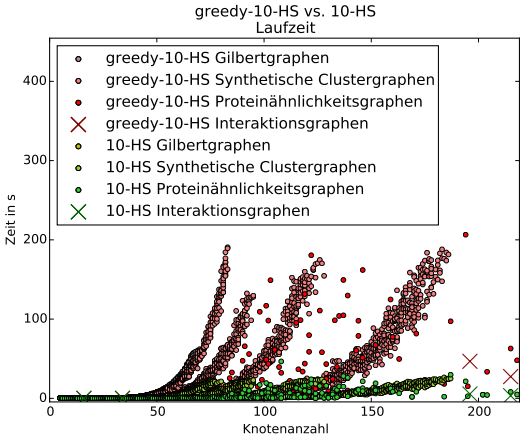

Abbildung 3.9: Laufzeitvergleich zwischen greedy-10-HS und 10-HS

Die greedy-10-HS-Approximation kann erneut deutlich höhere untere Schranken aufstellen und ist bereits sehr nahe an der oberen Grenze des Packings (siehe Abbildung 3.8). Allerdings vervielfacht sich erneut die benötigte Rechenzeit (siehe Abbildung 3.9). Dazu kommt noch weiterer Speicheraufwand um zu speichern wie oft jede Kante in allen Sets vorkommt. Daher sinkt erneut die durchschnittlich handhabbare Knotenanzahl.

3.2.4 Fazit

Das rekursive Packing liefert relativ schnelle und gute Ergebnisse. Ein weiterer, wichtiger Punkt ist, dass es auf allen Graphen anwendbar ist, ohne in Speicherprobleme zu geraten. Der greedy-10-HS-Algorithmus liefert die besten unteren Schranken und hat für Graphen bis durchschnittlich 50 Knoten eine annehmbare Laufzeit, in der auf unseren Instanzen außerdem keine Speicherfehler auftraten. Diese traten frühstens ab etwa 75 Knoten bei den Gilbertgraphen mit $p = 51\%$ auf.

Der greedy-10-HS-Algorithmus ist damit besser geeignet am Anfang des Suchbaumalgorithmus einmalig eine untere Schranke zu ermitteln, während das rekursive Packing besser für große Graphen mit vielen P_5s geeignet ist.

3.3 Datenreduktion

Datenreduktion ist ein häufig eingesetzter Ansatz zur Optimierung der Laufzeit von Graphmodifikationsproblemen. Die Idee besteht darin die Struktur des Graphen zu analysieren und falls möglich zu vereinfachen, zum Beispiel durch Entfernen von Knoten oder Kanten. Dies führt bei den meisten Problemen dazu, dass die Laufzeit vieler von der Knotenanzahl abhängiger Algorithmen gesenkt werden kann, aber auch zu einer Verkleinerung des Suchbaumes, falls notwendige Editierungen entdeckt werden.

Definition 3.3. *Eine Datenreduktionsregel ist eine Polynomialzeitabbildung einer Instanz des P_5-frei-Editierungsproblem (G, k) auf eine Instanz (G', k'), sodass (G, k) eine Lösung besitzt, genau dann wenn (G', k') eine Lösung besitzt.*

Oft werden auch so genannte *Kerne* entworfen. Dies sind Datenreduktionsregeln, deren Größe des Ergebnisses nicht von der ursprünglichen Knotenanzahl, sondern von der Anzahl an benötigten Änderungen k abhängen. Zum Beispiel gibt es für das P_3-frei-Editierungsproblems einen Kern, dessen Ergebnis höchstens $2k$ viele Knoten enthält [28]. Des Weiteren gibt es einen Kern für das P_4-frei-Editierungsproblem mit $O(k^3)$ vielen Knoten [29] sowie einen einen Kern für das Minimum-Flip-Consensus-Problem mit ebenfalls $O(k^3)$ vielen Knoten [30]. Es wurde bewiesen, dass es für jedes fixed-parameter tractable Problem einen in Polynomialzeit berechenbaren Kern gibt, allerdings wurde ebenfalls bereits bewiesen, dass es für das P_5-frei-Editierungsproblem und allgemein alle P_l-frei-Editierungsprobleme mit $l > 4$ keinen Kern in polynomieller Größe geben kann, falls $co\mathcal{NP} \not\subseteq \mathcal{NP}/poly$ gilt [31]. Deshalb werden wir uns auf praktisch relevante Datenreduktionsregeln und deren Korrektheitsbeweise beschränken.

Die Datenreduktionsregeln müssen in unserem Fall so konstruiert werden, dass wir zusätzlich noch eine Lösung für das ursprüngliche Problem konstruieren können. Wir werden im Folgenden vier Datenreduktionsregeln vorstellen.

Datenreduktionsregel 3.1. *Falls es in einem Graphen G eine P_5-freie Zusammenhangskomponente gibt, welche aus der Menge der Knoten Z besteht, dann kann das P_5-frei-Editierungsproblem (G, k) auf (G', k) mit $G' := G[V \setminus Z]$ reduziert werden.*

Diese Datenreduktionsregel bezeichnen wir als *P_5-frei-Reduktion*.

Beweis. Sei G ein Graph mit einer P_5-freien Zusammenhangskomponente, welche die Knoten Z enthält. Sei $G' := G[V \setminus Z]$. Sei L eine optimale Lösung für G.

Zunächst stellen wir fest, dass in L keine Editierungen mit Knoten aus Z enthalten sind, da keine Lösung optimal sein kann die zwei getrennte Zusammenhangskomponente verbindet oder Editierungen in einem P_5-freien Teilgraph durchführt. Sei $H := G \triangle L$ und I der Graph H ohne die Zusammenhangskomponente auf Z. Dann sind I und $F' := G' \triangle L$ isomorph zueinander. Daraus folgt, dass jede optimale Lösung L für G ebenfalls eine optimale Lösung für G' ist. Die Umkehrung gilt ebenfalls, weil jede Lösung L' für G' offensichtlich eine Lösung für G ist. Die Lösung L' ist optimal für G, weil, wie bereits gezeigt, eine kleinere Lösung für G ebenfalls eine kleinere Lösung für G' ist.

Damit ist bewiesen, dass P_5-freie Zusammenhangskomponenten entfernt werden dürfen. □

Für die nächste Datenreduktionsregel stellen wir zunächst folgendes Lemma auf.

Lemma 3.2. *Wenn es in einer Zusammenhangskomponente mit n' vielen Knoten einen Knoten u mit einem Knotengrad von mindestens $n' - 2$ gibt, dann kann dieser in keinem P_5 enthalten sein.*

Beweis. Für alle durch einen Weg w der Länge fünf induzierten Teilgraphen ($G[w]$) muss u zu mindestens drei anderen Knoten adjazent sein, weswegen dieser Graph kein P_5 sein kann. □

Ein Knoten mit noch weniger Kanten, also $n' - 3$ vielen, kann Teil eines P_5 sein. Dies ist zum Beispiel bei Grad-2-Knoten eines P_5 der Fall.

Datenreduktionsregel 3.2. *Wenn es in einer Zusammenhangskomponente G mit n vielen Knoten einen Knoten u mit Knotengrad mindestens $n - 2$ gibt, dann kann das P_5-frei-Editierungsproblem (G, k) auf $(G[V \setminus \{u\}], k)$ reduziert werden.*

Diese Datenreduktionsregel bezeichnen wir als *Universalknotenreduktion*.

Beweis. Sei G ein Graph, der einen Knoten u beinhaltet, welcher zu mindestens $n - 2$ anderen Knoten aus G adjazent ist. Sei $G' := G[V \setminus \{u\}]$. Sei L eine optimale Lösung für G und L' eine optimale Lösung für G'.

Dann unterscheiden sich F und F' genau durch Knoten u und seine Kanten, welche zusätzlich in F enthalten sind. Aus Lemma 3.2 folgt, dass u in keinem P_5 enthalten ist. Daher können durch Hinzufügen von u und seinen Kanten in G' keine P_5s entstehen. Daher ist u auch in keiner optimalen Lösung von G enthalten. Damit ist sowohl gezeigt,

dass L' eine optimale Lösung für G ist, als auch das jede optimale Lösung für G eine optimale Lösung für G' ist. □

Die folgende Datenreduktionsregel wurde ursprünglich für das Minimum-Flip-Consensus-Tree-Problem entworfen [30]. Sie ist aber ebenfalls auf das P_5-frei-Editierungsproblem anwendbar[30]. Zunächst müssen wir allerdings den Begriff des *Critical-Independent-Set*, kurz *CIS*, einführen.

Lemma 3.3. *Sei G ein Graph, dann nennen wir die Menge $I \subseteq V$ ein Critical-Independent-Set, falls für alle Knotenpaare $u, v \in I$ gilt, dass u und v nicht adjazent sind, $N(u) = N(v)$ ist und I maximal bezüglich dieser Eigenschaft ist.*

Alle Critical-Independent-Sets eines Graphen können in linearer Laufzeit gefunden werden [32].

Datenreduktionsregel 3.3 ([30])**.** *Sei G ein Graph und $I := \{i_1, \ldots, i_j\} \subseteq V$ ein Critical-Independent-Set. Falls $|I| > k + 1$, dann kann das P_5-frei-Editierungsproblem (G, k) auf $(G[V \setminus \{i_{k+2}, \ldots, i_j\}], k)$ reduziert werden.*

Diese Datenreduktionsregel bezeichnen wir als *CIS-Reduktion*.

In der letzten Datenreduktionsregel werden wir uns damit beschäftigen, unter welchen Bedingungen wir Grad-1-Knoten entfernen können.

Datenreduktionsregel 3.4. *Sei $u \in V$ und $B := \{b_1, \ldots, b_i\}$ eine Menge von i vielen Grad-1-Knoten, die zu u adjazent sind, sowie $C := \{c_1, \ldots, c_j\}$ eine Menge von j vielen Knoten mit Grad mindestens zwei, die ebenfalls adjazent zu u sind. Falls $i+2 \geq j$, dann kann das P_5-frei-Editierungsproblem (G, k) auf (G', k) mit $G' := G[V \setminus \{b_{j+2}, \ldots, b_i\}]$ reduziert werden.*

Abbildung 3.10 stellt die Regel schematisch dar. Im Folgenden bezeichnen wir diese Datenreduktionsregel als *Blattreduktion*. Für den Beweis beginnen wir mit einer grundlegenden Überlegung darüber, wie sich die Invertierung eines Knotenverhältnisses auf die Entstehung neuer $P_5 s$ in einem Graph auswirkt.

Lemma 3.4. *Falls nach der Invertierung des Verhältnisses zwischen den Knoten u und v neue induzierte $P_5 s$ entstehen, so müssen sowohl u als auch v in jedem von ihnen enthalten sein.*

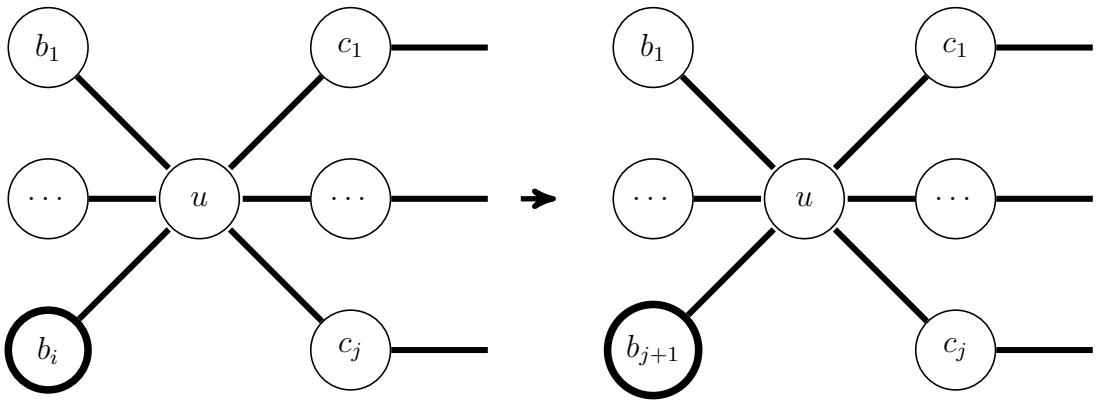

Abbildung 3.10: Schema für Blattreduktion 3.4

Beweis. Sei W_5 die Menge aller Wege der Länge fünf in G.

- Fall 1: Wenn die Kante $\{u, v\}$ aus G gelöscht wird und neue P_5s entstehen. Durch das Löschen einer Kante können keine neuen Wege entstehen, daher sind alle neuen P_5s in W_5 enthalten. Da jedoch nur das Verhältnis zwischen u und v verändert wurde können auch nur Wege die u und v enthalten zu neuen P_5s werden.

- Fall 2: Wenn die Kante $\{u, v\}$ in G eingefügt wird und neue P_5s entstehen. Durch das Hinzufügen einer Kante kann ein Weg der Länge fünf nicht zu einem P_5 werden. Daher sind alle neuen P_5s nicht in W_5 enthalten. Alle neu entstandenen Wege der Länge fünf müssen die einzige neue Kante $\{u, v\}$ enthalten und damit müssen auch die neu entstandenen P_5s diese Kante enthalten. □

Nun beginnen wir den eigentlich Beweis.

Beweis. Seien G, G' Graphen und B, C Mengen wie in Datenreduktionsregel 3.4 beschrieben.

Als erstes zeigen wir, dass jede optimale Lösung L' für G' ebenfalls eine optimale Lösung für G ist. Sei $F := G \triangle L'$ und $F' := G' \triangle L'$. Dann unterscheiden sich F und F' genau durch die Knoten in $\{b_{j+2}, \ldots, b_i\}$, welche nur in F enthalten sind. Wir werden zeigen, dass wir für jede Lösung L', die Editierungen mit Knoten aus $B^* := \{b_1, \ldots, b_{j+1}\}$ enthält, eine Lösung L'' mit weniger Elementen konstruieren können.

Fallunterscheidung nach Anzahl e der Editierungen in L', die einen Knoten aus B^* enthalten:

- Falls $e \leq j$, dann gibt es einen Knoten $v \in B^*$ der im Graphen F' zum Knoten u adjazent ist. Da F' ein P_5-freier Graph ist, gibt es keinen P_5 in dem v enthalten ist.

Dieselbe Argumentation gilt für alle Knoten aus B^*, weshalb die Menge $L'' := L' \setminus \{\{x, u\} \in L' \mid x \in B^*\}$ eine Lösung mit weniger Elementen ist.

- Falls $e > j$, dann ist die Menge $L'' := \Big(L' \setminus \{\{b', x\} \mid b' \in (B^* \cup \{u\}) \wedge x \in V'\}\Big) \cup \{\{u, c\} \mid c \in C\}$ eine Lösung mit mindestens einem Elementen weniger.

  Es bleibt zu zeigen, dass $F'' := (G' \triangle L'')$ ein P_5-freier Graph ist. Der Graph F' und F'' unterscheiden sich darin, dass der Knoten u zusammen mit den Knoten B^* in F'' eine P_5-freie Zusammenhangskomponente bilden, in der alle Knoten aus B^* nur zu u adjazent sind. Da F' ein P_5-freier Graph ist, müssen nach Lemma 3.4 in allen P_5s jeweils entweder der Knoten u oder einer der Knoten aus B^* enthalten sein. Einen P_5, der zwei Knoten aus unterschiedlichen Zusammenhangskomponenten enthält, kann es nicht geben. Daher ist F'' ebenfalls P_5-frei.

Damit ist bewiesen, dass eine optimale Lösung L' keine Editierung mit einem Knoten aus B^* enthält. Daraus folgt, dass wir beliebig viele zu u adjazente Grad-1-Knoten in den Graphen G' einfügen können, ohne dass dadurch P_5s entstehen. Ansonsten wären die Knoten aus B^* Teil eines P_5 und damit die Annahme über L' hinfällig. Damit ist L' ebenfalls eine optimale Lösung für G.

Nun zeigen wir, dass jede optimale Lösung L für G ebenfalls eine optimale Lösung für G' ist. Sei $F := G \triangle L$ und $F' := G' \triangle L$. Falls L Knoten aus B enthält ist $L' := \Big(L \setminus \{\{b', x\} \mid b' \in (B \cup \{u\}) \wedge x \in V\}\Big) \cup \{\{u, c\} \mid c \in C\}$ eine kleinere Lösung. Weil F ein P_5-freier Graph ist, müssen nach Lemma 3.4 in allen P_5s jeweils entweder der Knoten u oder einer der Knoten aus B enthalten sein. Einen P_5, der zwei Knoten aus unterschiedlichen Zusammenhangskomponenten enthält, kann es nicht geben. Damit ist $G \triangle L'$ ein P_5-freier Graph und L' eine Lösung. Daher ist kein Knoten aus B in L enthalten.

Damit unterscheiden sich F und F' genau durch die Knoten in $\{b_{j+2}, \ldots, b_i\}$, welche nicht in F' enthalten sind. Durch das Entfernen von Knoten können keine neuen P_5s entstehen, daher ist F' ebenfalls P_5-frei und L daher eine Lösung für G'. Es kann keine kleinere Lösung L'' für G' als L geben, weil, wie bereits gezeigt, jede Lösung für G' ebenfalls eine Lösung für G ist. □

Während der Ausarbeitung dieser Arbeit blieb die Frage offen, ob eine Datenreduktionsregel, welche alle Knoten entfernt, die nicht Teil eines P_5 sind, gültig ist oder nicht.

Wir haben die hier vorgestellten Datenreduktionsregeln auf unsere Testgraphen angewendet. Dabei zeigte sich, dass sich die synthetischen Graphen lediglich im Bereich

bis zu 20 Knoten reduzieren ließen und dies auch nur für den niedrigsten Parameter $p = 0.02$. Dies liegt daran, dass in dieser Konstellation gehäuft mehrere und dann zumeist P_5-freie Zusammenhangskomponenten in den Graphen enthalten sind. Die Anzahl der Grad-1-Knoten ist durchweg sehr gering, sodass die Blattreduktion nur sehr selten erfolgreich war. Umgekehrt gibt es aber auch nur sehr wenige Knoten, die sich mit der Universalknotenreduktion reduzieren ließen. Die CIS-Reduktionsregel ist von einem Parameter k abhängig. Es ist offensichtlich, dass die Regel umso erfolgreicher ist, desto geringer dieser Parameter gewählt ist. Wir haben für den Test der Regel das Ergebnis des rekursiven Packings verwendet, um zu bestimmen ob die Regel dazu geeignet ist am Anfang des Suchbaumalgorithmus ausgeführt zu werden. Für diesen Parameter konnten allerdings ebenfalls nur wenige Knoten in zumeist kleinen Graphen reduziert werden. Auffällig sind die Ergebnisse auf den Proteinähnlichkeitsgraphen. Es zeigte sich, dass vergleichsweise viele Graphen bereits P_5-frei sind. Dies gilt auch vereinzelt für Graphen mit sehr vielen Knoten. Außerdem ließen sich auch mit den anderen Reduktionsregeln noch für große, nicht P_5-freie Graphen einige Knoten reduzieren. Die Frage, ob sich die Datenreduktionsregeln vielleicht während der Laufzeit des Suchbaumalgorithmus erfolgreicher einsetzen lassen, bleibt vorerst offen. Wir werden darauf in Abschnitt 3.6 zurückkommen, wenn wir die Ergebnisse unserer Tests für den Suchbaumalgorithmus auswerten.

3.4 Annotationen

Unter bestimmten Umständen ist es möglich für Verhältnisse zweier Knoten mit absoluter Gewissheit zu sagen, ob sie für eine optimale Lösung invertiert werden müssen oder nicht. Dadurch lässt sich der Verzweigungsgrad beim Auflösen der P_5s reduzieren. Im Folgenden bezeichnen wir eine Kante die nicht mehr gelöscht werden darf als *permanent* und eine Nicht-Kante die nicht mehr eingefügt werden darf als *verboten*. Für ein gegebenes P_5-frei-Editierungsproblem (G, k) enthält die Menge Q die Knotenpaare der *verbotenen Nicht-Kanten* und die Knotenpaare der *permanenten Kanten*. Das Verhältnis zweier Knoten u, v zu fixieren ist aber nicht gleichbedeutend damit zu sagen, dass eine Editierung, welche $\{u, v\}$ enthält, nicht optimal ist, sondern dass es mindestens eine optimale Lösung gibt, welche dieses Verhältnis nicht invertiert.

Es ist leicht einzusehen, dass in einer optimalen Lösung keine Editierung mehr als einmal enthalten sein darf, da ansonsten eine kleinere Lösung existiert, welche den selben P_5-freien Graph erzeugt.

Annotationsregel 3.1. *Jedes invertierte Knotenverhältnis kann der Menge Q hinzugefügt werden.*

Diese Regel mag obsolet erscheinen, da wir eine Lösung als Menge von Knotenpaaren definiert haben und somit jede Editierung nur einmal enthalten sein kann. Jedoch hat diese Regel Einfluss auf die praktische Umsetzung des Algorithmus, weshalb wir uns entschieden haben sie mit anzugeben.

In der nächsten Annotationsregel betrachten wir die Knotenverhältnisse von Grad-1-Knoten und stellen fest, dass alle Nicht-Kanten zu Grad-1-Knoten fixiert werden können.

Annotationsregel 3.2. *Für jeden Grad-1-Knoten v und seinen Nachbarn u gilt, dass die Menge der Knotenpaare $K := \{\{v, x\} | x \in (V \setminus \{u, v\}\}$ der Menge Q hinzugefügt werden kann.*

Diese Annotationsregel bezeichnen wir als *Blattannotation*.

Beweis. Sei L eine optimale Lösung für G, welche eine Editierung mit dem Knoten v enthält. Dann lässt sich eine zweite Lösung L' so konstruieren, dass wir alle Editierungen in der Menge $A := \{\{v, x\} \in L\}$ aus der Lösung L entfernen und durch die Editierung $\{v, u\}$ ersetzen. Sei $F := G \triangle L$ und $F' := G \triangle L'$. Es bleibt zu zeigen, dass F' ein P_5-freier Graph ist. Die Graphen F und F' unterscheiden sich darin, dass in F' der Knoten v zu keinem anderen Knoten adjazent ist und entsprechend alle Kanten, die es in F zu ihm gibt, entfernt wurden. Da F ein P_5-freier Graph ist folgt nach Lemma 3.4, dass in allen P_5s in F' der Knoten v enthalten sein muss. Dies kann nicht der Fall sein, da v zu keinem anderen Knoten adjazent ist. Daher ist F' ebenfalls P_5-frei.

Daraus folgt, dass L' eine Lösung für G ist und weil $|L| \geq |L'|$ gilt und L optimal ist, muss auch L' eine optimal Lösung sein. □

Wir können die Datenreduktionsregel 3.4 durch eine zusätzlich Annotation deutlich verbessern.

Annotationsregel 3.3. *Sei $u \in V$ und $B := \{b_1, \ldots, b_i\}$ eine Menge von i vielen Grad-1-Knoten, die zu u adjazent sind, sowie $C := \{c_1, \ldots, c_j\}$ eine Menge von j vielen Knoten mit Grad mindestens zwei, die ebenfalls adjazent zu u sind. Falls $i + 2 \geq j$, dann kann das P_5-frei-Editierungsproblem (G, k) auf (G', k) mit $G' := G[V \setminus \{b_2, \ldots, b_i\}]$ reduziert werden, falls die Knotenpaare $\{\{b_1, x\} \mid x \in V' \setminus \{b_1\}\}$ der Menge Q hinzugefügt werden.*

Aus dem Beweis der Datenreduktionsregel 3.4 wird ersichtlich, dass eine optimale Lösung für G keine Editierung mit einem Knoten aus B enthält. Daher können wir die Menge der Knoten $\{b_{j+2}, \ldots, b_i\}$ aus dem Graphen entfernen. Die Knoten $B^* := \{b_1, \ldots, b_{j+1}\}$ können nicht entfernt werden, da ansonsten eine optimale Lösung für den reduzierten Graphen G' eine Editierung mit einem der Knoten aus B^* enthalten könnte und damit nicht mehr notwendiger Weise eine Lösung für den Graphen G ist. Wenn wir alle Knoten aus B^* bis auf b_1 entfernen, müssen wir also sicherstellen, dass b_1 in keiner optimalen Lösung von G' enthalten ist. Die Annotationsregel 3.3 leistet genau dies, indem sie alle Verhältnisse zwischen b_1 und allen anderen Knoten fixiert, weswegen b_1 in keiner Lösung enthalten sein kann.

3.5 Weitere Verbesserungsansätze

In diesem Abschnitt werden wir einige weitere Ansätze zur Verbesserung der Laufzeit besprechen. Wir schauen uns zunächst an, wie das Problem für unzusammenhängende Graphen vereinfacht werden kann. Dem folgen Überlegungen zur Wahl des aufzulösenden P_5 sowie der Verwaltung von P_5-Listen.

3.5.1 Separates Lösen von Zusammenhangskomponenten

Offensichtlicherweise kann eine Lösung, welche zwei getrennte Zusammenhangskomponenten verbindet, nicht optimal sein. Daher können alle Zusammenhangskomponenten als eigenständiges P_5-frei-Editierungsproblem gesehen werden. Dies stellt eine große Zeitersparnis dar, weil die Anzahl der Zustände, die der Suchbaumalgorithmus überprüfen muss, mit steigendem Parameter k exponentiell wächst. Die Suche nach getrennten Zusammenhangskomponenten kann mit Hilfe einer Tiefensuche in $O(m + n)$ Zeit ausgeführt werden und ist damit vergleichsweise schnell, sodass wir den Graphen nach jeder gelöschten Kante auf neue Zusammenhangskomponenten überprüfen können.

Außerdem empfiehlt es sich die Zusammenhangskomponenten, die sich durch die Editierungen während des Suchbaumalgorithmus ergeben, nicht nacheinander zu lösen, sondern parallel. Dazu versuchen wir alle noch nicht gelösten Zusammenhangskomponenten mit dem gleichen Parameterwert zu lösen. Für jede gelöste Zusammenhangskomponente reduzieren wir die noch zur Verfügung stehenden Kantenmodifikationen. Wir tun dies solange, bis entweder alle Zusammenhangskomponenten erfolgreich gelöst wurden oder die noch benötigten Änderungen die zur Verfügung stehenden übersteigen.

Zum einen ergibt sich aus der Anzahl der ungelösten Zusammenhangskomponenten eine weitere Abschätzung für k, weil jede ungelöste Zusammenhangskomponente mindestens noch eine Änderung benötigt. Zum anderen wird auf diese Weise erneut die Anzahl der zu überprüfende Zustände im Suchbaum verringert.

3.5.2 Gezieltes Verzweigen

Bisher haben wir zur Laufzeit des Suchbaumalgorithmus einen beliebigen P_5 ausgewählt und verzweigen dann auf alle Editierungen, welche diesen auflösen. Da wir nun Regeln zur Annotation von Knotenpaaren kennen, dessen Verhältnis nicht mehr editiert werden darf, liegt es nahe P_5s mit möglichst vielen annotierten Knotenverhältnissen auszuwählen, um den Verzweigungsgrad zu reduzieren.

Eine weitere Strategie besteht darin die Editierungen nicht in zufälliger Reihenfolge durchzuprobieren, sondern zunächst das Verhältnis für die Knotenpaare zu invertieren, welche in den meisten anderen P_5s enthalten sind. Dieses Vorgehen zielt darauf ab für ein gegebenes k mit möglichst wenigen Editierungsversuchen die k richtigen Editierungen vorzunehmen, ohne den kompletten Suchbaum der Tiefe k durchlaufen zu müssen.

3.5.3 Speichern von P_5-Listen

Es könnte zeitlich effizienter sein alle oder mehrere P_5s zu berechnen, anstatt nach jeder Kantenmodifikation erneut einen P_5 zu suchen. Allerdings muss man dabei beachten, dass durch jede Kantenmodifikationen bereits in der Liste enthaltene P_5s aufgelöst und neue P_5s entstehen können. Es gibt zwei Möglichkeiten damit umzugehen. Entweder wir entfernen nach jeder Kantenmodifikation alle ebenfalls aufgelösten P_5s und berechnen alle neuen P_5s oder wir ignorieren das Problem und testen einfach jeden aus der Liste entnommen P_5 darauf ob er noch gültig ist, solange bis wir einen gültigen finden oder die Liste leer ist. Im letzteren Falle berechnen wir eine neue Liste.

Ein weiterer Vorteil der P_5-Listen liegt darin, dass wir in kürzerer Zeit untere Schranken für den Parameter k berechnen können und gezielt verzweigen können.

3.6 Ergebnisse

Um die Leistungsfähigkeit des Suchbaumalgorithmus für das optimale P_5-frei-Editierungsproblem und der Optimierungsansätze zu vergleichen, werden wir eine Reihe von Testdurchläufen für verschiedene Varianten auf den Testgraphen durchführen.

Dabei werden wir die synthetischen Graphen wieder anhand des Parameters p trennen. Für jede der insgesamt neun Graphklassen werden wir nach steigender Knotenanzahl sortiert den Suchbaumalgorithmus für das optimale P_5-frei-Editierungsproblem auf alle Graphen anwenden, solange bis wir entweder erfolgreich alle Graphen gelöst haben oder 15 Graphen in Folge jeweils nicht mehr innerhalb von vier Minuten gelöst werden können.

Zunächst widmen wir uns der Frage, welcher der beiden Algorithmen zum Suchen eines P_5 besser geeignet ist. Die Endpunktsuche konnte mit einer kurzen Laufzeit auf Graphen mit vielen P_5s punkten, während die Mittelpunktsuche schneller auf Graphen mit wenigen P_5s war. Dafür testen wir den Suchbaumalgorithmus ohne zusätzliche Verbesserungen mit dem jeweiligen P_5-Suchalgorithmus.

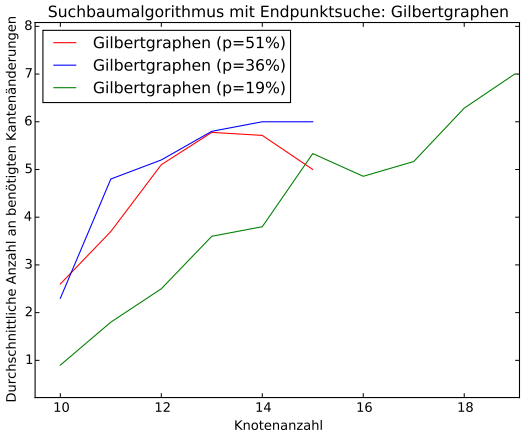

Abbildung 3.11: Durchschnittliche optimale Lösungsgröße der Gilbertgraphen

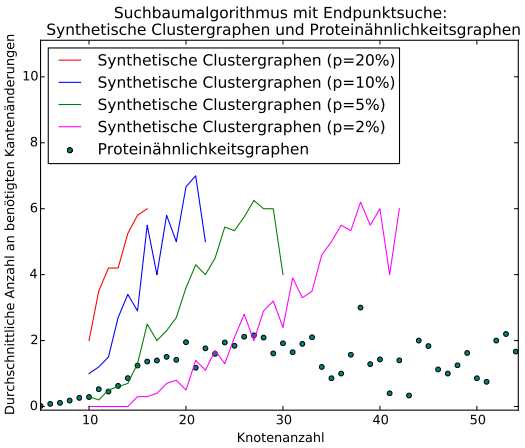

Abbildung 3.12: Durchschnittliche optimale Lösungsgröße der synthetischen Clustergraphen

Aus den Abbildungen 3.11 und 3.12 wird ersichtlich, dass wir das Problem lediglich für relativ kleine Graphen lösen können. Bei den synthetischen Graphen ist ein deutlicher Zusammenhang zum Parameter p zu erkennen. Für Graphen bis zu 50 Knoten gab es keinen bedeutenden Laufzeitunterschied zwischen beiden P_5-Suchalgorithmen und so verwundert es auch nicht, dass beide Varianten nahezu identische Resultate erbringen, weswegen wir nur die Ergebnisse der Endpunktsuche grafisch dargestellt haben. Da wir in den Abbildungen die durchschnittliche benötigte Änderungsanzahl verwenden, wollen wir an dieser Stelle erwähnen, dass die optimale Lösungsgröße der synthetischen Graphen bis auf wenige Ausnahmen konstant ansteigt. Diese Ausnahmen sind der Grund warum die Kurven teilweise am Schluss abfallen. Außerdem gibt es auch einige Proteinähnlichkeitsgraphen mit höheren Lösungsgrößen von bis zu sechs oder sieben, allerdings haben sehr viele Proteinähnlichkeitsgraphen eine kleinere Lösung.

Wir konnten allerdings für keinen Graphen eine Lösung größer als sieben berechnen. Unsere Vorhersage für den Schwierigkeitsgrad der einzelnen Graphklassen am Ende von Abschnitt 1.4.3 wird größtenteils bestätigt. Die Ausnahmen davon sind die Interaktionsgraphen, von denen wir unerwarteter Weise selbst den kleinsten mit 16 Knoten nicht lösen konnten und die synthetischen Clustergraphen mit $p = 5\%$, die sich deutlich besser als erwartet im Vergleich zu den Gilbertgraphen mit $p = 19\%$ lösen lassen.

Die Zeit zum Lösen der Graphen korreliert mit der durchschnittlichen optimalen Lösungsgröße (siehe Abbildungen 3.13 und 3.14). Außerdem wird aus den verschiedenen Laufzeiten der einzelnen synthetischen Graphklassen deutlich, dass für die von ungelösten Graphen die Laufzeit wesentlich stärker von der Größe der optimalen Lösung abhängig ist als von der Knotenanzahl des Graphen.

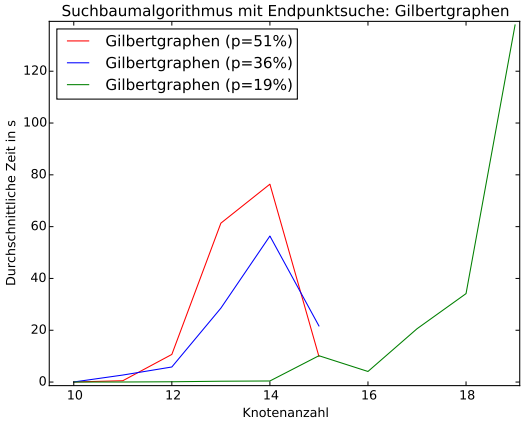

Abbildung 3.13: Durchschnittliche Laufzeit der Gilbertgraphen

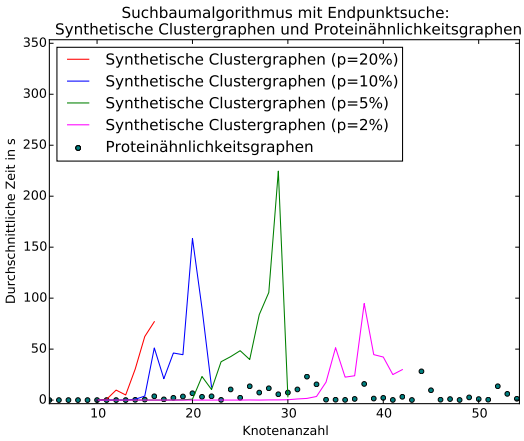

Abbildung 3.14: Durchschnittliche Laufzeit der synthetischen Clustergraphen

Wir betrachten als Nächstes die Variante einmalig alle Optimierungsansätze anzuwenden, bevor wir den Suchbaumalgorithmus für das optimale P_5-frei-Editierungsproblem starten. Zunächst teilen wir den Graph in seine Zusammenhangskomponenten auf und behandeln jede von ihnen wie ein eigenständiges Problem, wie beschrieben in Abschnitt 3.5.1. Danach führen wir die Universalknotenreduktion 3.2, die Blattannotation 3.2 und die verbesserte Version der Blattreduktion 3.3 durch. Die P_5-frei-Reduktionsregel wird durch das trennen der Zusammenhangskomponenten obsolet. Anschließend berechnen wir die untere Schranke für jede Zusammenhangskomponente mit dem greedy-10-HS-Algorithmus. Mit dieser Schranke führen wir abschließend die CIS-Reduktion 3.3 durch. Außerdem initialisieren wir den Parameter k des Suchbaumalgorithmus mit der berechneten unteren Schranke. Einzig die Annotation aller bereits editierten Knotenpaare 3.1 wird nach jeder Verzweigung im Suchbaum angewendet.

Unsere Tests ergeben, unabhängig vom gewählten Algorithmus zur P_5-Suche, nur eine minimale Änderung der Leistung. So können etwa 30 Graphen mehr gelöst werden, während sich an der Laufzeit nichts ändert. Daher gehen wir direkt zur nächsten Variante über.

In der letzte Variante führen wir wieder vorher alle Optimierungen der vorherigen Variante durch. Zusätzlich schätzen wir nun nach jeder Verzweigung im Suchbaum die benötigte Änderungsanzahl mit dem greedy-10-HS-Approximationsalgorithmus ab. Da wir hierfür jeweils alle P_5s berechnen müssen, nutzen wir dieses Zwischenergebnis um den P_5 zu suchen, dessen Knotenpaare in den meisten anderen P_5s enthalten sind. Außerdem verwenden wir den Ansatz des gezielten Verzweigens aus Abschnitt 3.5.2, indem wir zunächst die Knotenpaare des P_5s editieren, die in den meisten anderen P_5s enthalten sind.

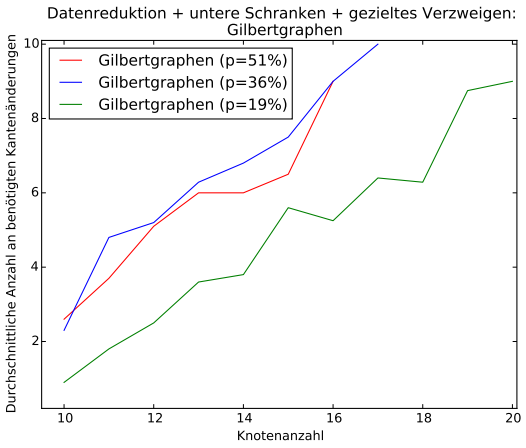

Abbildung 3.15: Durchschnittliche optimale Lösungsgröße der Gilbertgraphen

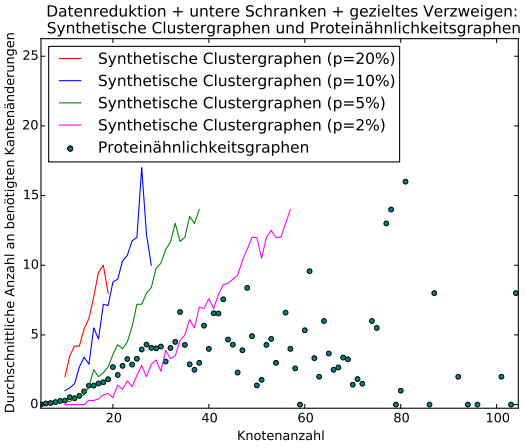

Abbildung 3.16: Durchschnittliche optimale Lösungsgröße der synthetischen Clustergraphen

Im Vergleich zu den den vorherigen Varianten konnten wir die Leistungsfähigkeit deutlich steigern. So steigt, wie in Abbildung 3.15 zu sehen, die maximal gefundene Lösungsgröße für die Gilbertgraphen um etwa drei und für die synthetischen Clustergraphen und die Proteinähnlichkeitsgraphen sogar um noch mehr (Abbildung 3.16). Auffallend ist, dass der Algorithmus nun einige sehr große Lösungen findet, während er bisher eine einheitliche Obergrenze hatte. Die größte gefundene Lösung liegt bei 37 für einen Graphen aus der Klasse der Proteinähnlichkeitsgraphen. Wir erklären dies über das Zusammenspiel von unteren Schranken und gezielten Verzweigen. Falls die untere Schranke bereits die optimale Lösungsgröße ist, besteht durch das gezielte Verzweigen eine relativ hohe Wahrscheinlichkeit die richtigen Editierungen zu finden, obwohl nur ein Bruchteil des Suchraumes abgearbeitet werden kann. Die Variante nach jedem Ver-

zweigen erneut eine Datenreduktion durchzuführen konnte keine besseren Ergebnisse liefern.

Bisher konnten wir mit keiner der Varianten einen der Interaktionsgraphen lösen. Daher haben wir ein weiteren Testdurchlauf mit bis zu sechs Stunden pro Graph für die Klasse der Interaktionsgraphen durchgeführt. Dabei haben wir die bisher erfolgreichste Variante verwendet, in jedem Schritt erneut die untere Schranke zu berechnen und gezielt zu verzweigen. Die Ergebnisse sind in Tabelle 3.1 dargestellt.

Graph	Gelöst?	greedy-10-HS	k am Ende	Verzweigungen	Zeit
Hochlandstämme	Nein	6	13	4.724.500	6h
Karate Club Graph	Ja	10	13	21.161	416s
Jazz Musiker	Nein	Speicherfehler			
translation	Nein	238	242	78	6h
cell-cycle	Nein	155	156	465	6h
transcription	Nein	139	140	778	6h

Tabelle 3.1: Überblick über die Ergebnisse der Interaktionsgraphen nach sechs Stunden Laufzeit; k am Ende ist der höchste Wert für den Parameter k, mit dem der Algorithmus zur Laufzeit versucht hat eine optimale Lösung zu finden. Der Wert stellt daher eine untere Schranke dar.

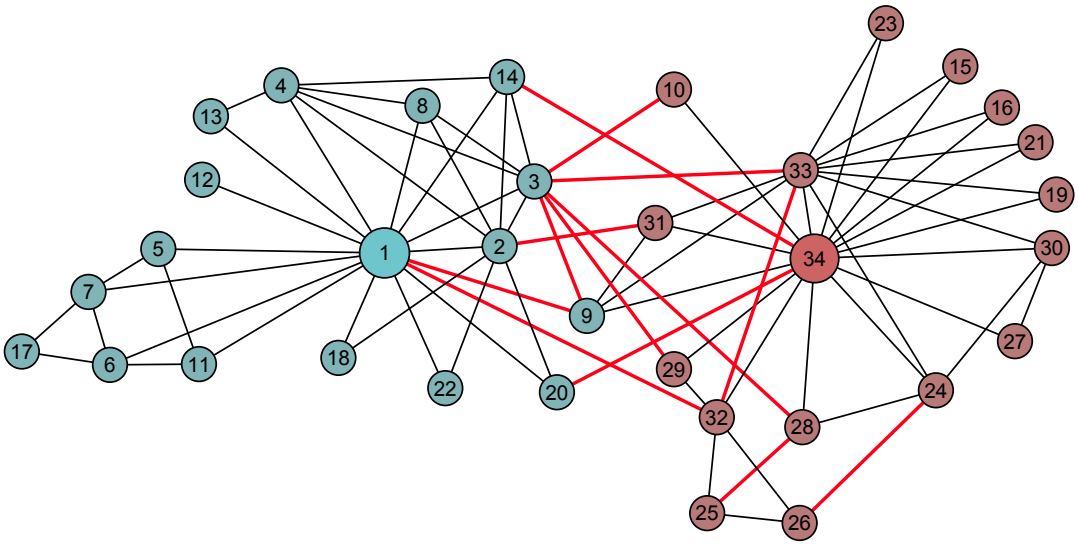

Abbildung 3.17: Zachary-Karate-Club-Graph nach P_5-frei-Editierung: Die Färbung der Knoten repräsentiert die Clubzugehörigkeit des Mitglieds nach der Trennung. Rot markierte Kanten wurden gelöscht.

Der Karate-Club-Graph konnte als einziger Graph mit knapp sieben Minuten Laufzeit innerhalb der vorgegebenen Zeit gelöst werden. Die Abbildung 3.17 zeigt den

Karate-Club-Graph und die dreizehn notwendigen Editierungen, um ihn zu einem P_5-freien Graph zu transformieren. Die dreizehn gelöschten Kanten wurden mit rot markiert. Es wurde keine neue Kante hinzugefügt. Interessanterweise erhalten wir genau die beiden gleichen Gruppen wie Zachary, sodass wir ebenfalls einzig Knoten 9 falsch klassifizieren. Von den dreizehn Editierungen werden zehn benötigt um die beiden Gruppen zu trennen und drei weitere, um innerhalb der Zusammenhangskomponenten mit Knoten 34 die P_5s aufzulösen. Die andere Zusammenhangskomponente mit dem Knoten 1 ist sogar ein natürlicher P_5-freier Graph. Um die Qualität dieser Lösung besser bewerten zu können, haben wir außerdem die optimalen Lösungen für das P_3-frei-Editierungsproblem sowie das P_4-frei-Editierungsproblem für den Karate-Club-Graph berechnet. [1] Die entsprechenden Abbildungen 5.1 und 5.2 befinden sich im Anhang. Um eine optimale P_3-freie Lösung zu erhalten, müssen 50 Editierungen im Graph durchgeführt werden. Der transformierte Graph besteht aus vielen kleinen Zusammenhangskomponenten und entspricht damit nicht dem gewünschten Ergebnis. Die optimale $P4$-freie Lösung hingegen klassifiziert die Knoten ebenfalls genau wie Zachary. Allerdings werden dafür insgesamt 20 Editierungen benötigt, also sieben mehr als für eine optimale Lösung des P_5-frei-Editierungsproblems notwendig sind. Zusammenfassend lässt sich sagen, dass das P_5-frei-Modell für den Karate-Club-Graph ein sehr gutes Modell ist.

Eine genauere Analyse des Hochlandstämme-Graph ergibt, dass dieser in seiner ursprünglichen Variante ein gewichteter Graph ist, der sowohl Freundschaften als auch Feindschaften zwischen den Stämmen modelliert. Das P_5-frei-Modell ist allerdings für Graphen mit einer Kern-Peripherie-Struktur gedacht, welche bei dieser Mischung aus Freundschaften und Feindschaften nicht zu erwarten ist. Ein erneuter Lösungsversuch, auf dem Graphen der nur die Freundschaftsverhältnisse enthält, konnte innerhalb von 0,2 Sekunden durchgeführt werden und ergab eine optimalen Lösung mit zwei Editierungen, welche die Stämme in zwei Gruppen einteilt. Zwischen den Stämmen beider Gruppen gibt es 22 Feindschaften, aber auch innerhalb der beiden Gruppen gibt es insgesamt 7 Feindschaften.

Weil wir keinen der Proteininteraktionsgraphen exakt lösen konnten, können wir keine genaue Aussage dafür treffen, ob das P_5-frei-Modell für die Proteininteraktionsgraphen geeignet ist, auch wenn wir die optimale Lösungsgröße für das Split-Cluster-Graphen-Editieren kennen [9]. Diese stellen eine obere Schranke dar, weil Split-Cluster-Graphen keine P_5s enthalten (siehe Tabelle 3.2).

[1] Für die Berechnung der optimalen P_3-freien beziehungsweise P_4-freien Lösung sind nur minimale Änderungen am Code für die P_5-freie Lösung notwendig. Die Datenreduktions- und Annotationsregeln müssen deaktiviert werden.

Graph	translation	cell-cycle	transcription
k am Ende	242	156	140
k Split-Cluster-Graph	308	321	273

Tabelle 3.2: Vergleich für die Proteininteraktionsnetzwerke zwischen dem höchsten getesteten Parameterwert k während der sechs Stunden Laufzeit für das P_5-frei-Editierungsproblem und der Größe der optimalen Lösungen für das Split-Cluster-Graph-Editieren

An der geringen Anzahl an Verzweigungen für die drei biologischen Interaktionsgraphen wird der Schwachpunkte unserer letzten Suchbaumvariante deutlich. Wir benötigen für große Graphen mit vielen P_5s zu viel Zeit zum Berechnen der unteren Schranke. Abhilfe könnte die in Abschnitt 3.5.3 vorgestellte P_5-Liste schaffen.

3.7 Fazit

Abschließend lässt sich sagen, dass das Finden einer optimalen Lösung für das P_5-frei-Editierungsproblem im Vergleich zu anderen ähnlichen Problemen sehr aufwendig ist, weil es einen sehr hohen Verzweigungsgrad für den Suchbaum aufweist und es außerdem keinen Kern in polynomieller Größe gibt. Wir konnten jedoch mit Hilfe von unteren Schranken für den Parameter k sowie gezieltem Verzweigen die Suchtiefe des naiven Suchbaumalgorithmus für das optimale P_5-frei-Editierungsproblem verbessern und haben zusätzlich auch eine heuristische Chance für noch schwerere Probleme eine Lösung zu finden. Unsere Implementierung ist insbesondere für unzusammenhängende Graphen beliebiger Größe geeignet, solange alle Zusammenhangskomponenten einzeln eine kleine optimale Lösungsgröße von in etwa zehn Editierungen haben.

Unsere Ergebnisse decken sich mit den Voraussagen darüber wie schwer die einzelnen Graphklassen zu lösen sein würden. Dies stützt die Annahme über den Zusammenhang zwischen der Anzahl der in den Graphen enthaltenen P_5s und der Strukturähnlichkeit zu P_5-freien Graphen. Die Proteinähnlichkeitsgraphen ließen sich für größere Graphen sogar besser lösen als die synthetischen Clustergraphen, dies stützt die These, dass das P_5-frei-Modell gut zu diesen Graphen passt.

Wir konnten bezüglich der Wahl des P_5-Suchalgorithmus keine signifikanten Auswirkungen auf die Laufzeit feststellen. Jedoch ist es erwähnenswert, dass der relativ einfache Ansatz der Endpunktsuche in der Praxis bessere Ergebnisse liefert als der Hoàng-Algorithmus mit der besten Laufzeitklasse.

Letztlich konnten wir mit unseren Datenreduktionsregeln ebenfalls keine signifikante Verbesserung der praktischen Laufzeit erreichen.

Kapitel 4

Ausblick

Leider konnten wir mit unserer Implementierung die Proteininteraktionsnetzwerke nicht lösen und daher auch nicht überprüfen, ob das P_5-frei-Modell für die Suche nach Proteinkomplexen geeignet ist, sodass dies vorerst eine offene Frage bleibt.

Mit Hilfe von Ganzzahliger linearer Optimierung (englisch integer linear programming (ILP)) wurde sowohl die Suche nach einer optimalen Lösung für das P_3-frei-Editierungsproblem [11] als auch für das P_4-frei-Editierungsproblem [12] erfolgreich gelöst. Insbesondere für das Split-Cluster-Graph-Editierungsproblem konnte eine effiziente Formulierung gefunden werden, mit der sich sogar die drei Proteininteraktionsnetzwerke lösen lassen [9]. Daher könnte der ILP-Ansatz auch für das optimale P_5-frei-Editierungsproblem erfolgreich sein.

Alle angesprochenen Verbesserungsansätze haben keine nachweisliche Auswirkung auf die theoretische Laufzeit des Suchbaumalgorithmus. Dafür sind beispielsweise Ansätze notwendig, die entweder den Verzweigungsgrad senken oder Datenreduktionsregeln, welche die notwendigen Editierungen erkennen können und somit die Suchbaumtiefe verringern. Interessant wäre es außerdem, die in Abschnitt 3.5.3 angesprochene P_5-Liste zu realisieren, um nicht nach jeder Verzweigung des Suchbaumes erneut alle P_5s berechnen zu müssen.

Ein weiteres in dieser Arbeit unbehandeltes Thema ist, anstelle einer optimalen Lösung des P_5-frei-Editierungsproblems eine approximative Lösung zu suchen, falls eine optimale Lösung nicht notwendig ist.

Anhang

Inhalt des beigelegten Datenträgers

Dieser Arbeit liegt ein Datenträger mit den folgenden Dateien bei.

- eine digitale Version dieser Arbeit
- der Quellcode für die implementierten Algorithmen
- die verwendeten Testgraphen
- der Code zur Generierung der synthetischen Clustergraphen sowie der synthetischen P_5-freien Graphen

Zusätzliche Abbildungen

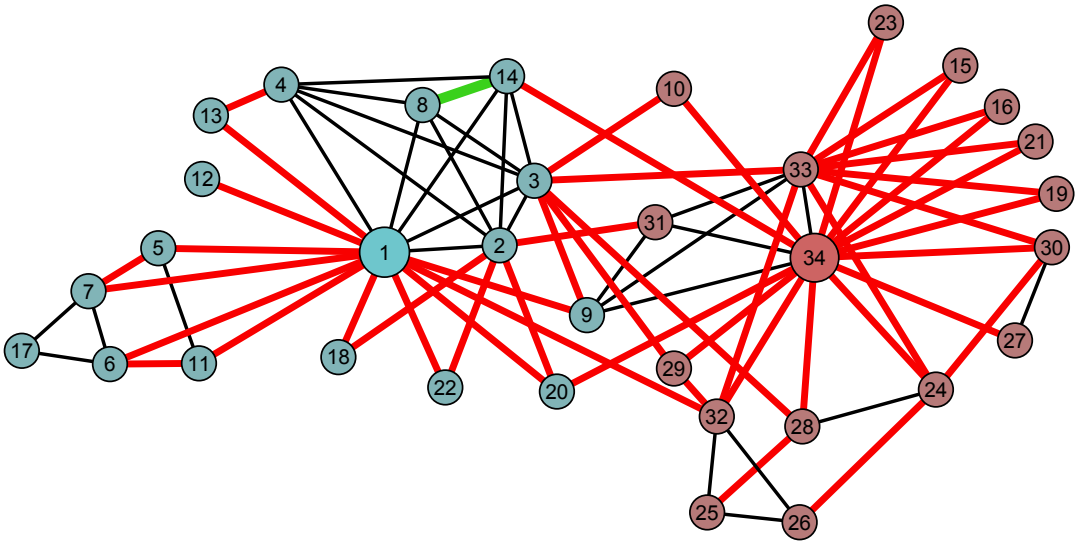

Abbildung 5.1: Zachary-Karate-Club-Graph nach optimaler P_3-frei-Editierung: Die Färbung der Knoten repräsentiert die Clubzugehörigkeit des Mitglieds nach der Trennung. Rot markierte Kanten wurden gelöscht, grüne markierte neu hinzugefügt.

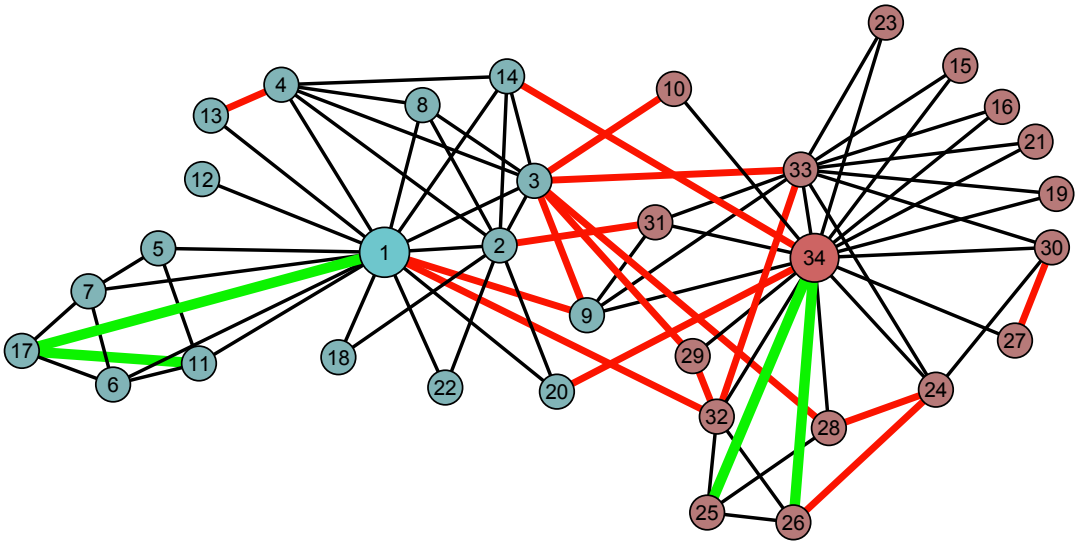

Abbildung 5.2: Zachary-Karate-Club-Graph nach optimaler P_4-frei-Editierung: Die Färbung der Knoten repräsentiert die Clubzugehörigkeit des Mitglieds nach der Trennung. Rot markierte Kanten wurden gelöscht, grüne markierte neu hinzugefügt.

Literaturverzeichnis

[1] BORNHOLDT, STEFAN und HEINZ GEORG SCHUSTER: *Handbook of Graphs & Networks*. Wiley Online Library, 2002. Zitiert auf S. 1.

[2] HELL, PAVOL und JAROSLAV NESETRIL: *Graphs and Homomorphisms*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford, 2004. Zitiert auf S. 1.

[3] CARRINGTON, PETER J, JOHN SCOTT und STANLEY WASSERMAN: *Models and methods in social network analysis*, Band 28. Cambridge university press, 2005. Zitiert auf S. 1.

[4] SLIKKER, MARCO und ANNE VAN DEN NOUWELAND: *Social and economic networks in cooperative game theory*, Band 27. Springer Science & Business Media, 2001. Zitiert auf S. 1.

[5] SCHAEFFER, SATU ELISA: *Survey: Graph Clustering*. Computer Science Review, 1(1):27–64, August 2007. Zitiert auf S. 1.

[6] SHAMIR, RON, RODED SHARAN und DEKEL TSUR: *Cluster graph modification problems*. In: *Graph-Theoretic Concepts in Computer Science*, Seiten 379–390. Springer, 2002. Zitiert auf S. 1.

[7] DUDA, RICHARD O, PETER E HART und DAVID G STORK: *Pattern classification*. John Wiley & Sons, 2012. Zitiert auf S. 1.

[8] EASLEY, DAVID und JON KLEINBERG: *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010. Zitiert auf S. 2.

[9] BRUCKNER, SHARON, FALK HÜFFNER und CHRISTIAN KOMUSIEWICZ: *A graph modification approach for finding core–periphery structures in protein interaction networks*. In: *Proceedings of the 14th Workshop on Algorithms in Bio-*

informatics (WABI '14), Band 8701 der Reihe *Lecture Notes in Bioinformatics*, Seiten 340–351. Springer, 2014. Zitiert auf S. 3, 48, and 50.

[10] BÖCKER, SEBASTIAN, SEBASTIAN BRIESEMEISTER, QUANG BAO ANH BUI und ANKE TRUSS: *Going Weighted: Parameterized Algorithms for Cluster Editing*. In: *Combinatorial Optimization and Applications, Second International Conference, COCOA 2008, St. John's, NL, Canada, August 21-24, 2008. Proceedings*, Seiten 1–12, 2008. Zitiert auf S. 3 and 6.

[11] BÖCKER, SEBASTIAN, SEBASTIAN BRIESEMEISTER und GUNNARW. KLAU: *Exact Algorithms for Cluster Editing: Evaluation and Experiments*. Algorithmica, 60(2):316–334, 2011. Zitiert auf S. 3 and 50.

[12] HELLMUTH, MARC, NICOLAS WIESEKE, MARKUS LECHNER, HANS-PETER LENHOF, MARTIN MIDDENDORF und PETER F STADLER: *Phylogenomics with paralogs*. PNAS, 2014. Zitiert auf S. 3 and 50.

[13] BÖCKER, SEBASTIAN: *A golden ratio parameterized algorithm for cluster editing*. Journal of Discrete Algorithms, 16:79–89, 2012. Zitiert auf S. 3.

[14] LIU, YUNLONG, JIANXIN WANG, JIONG GUO und JIANER CHEN: *Complexity and parameterized algorithms for Cograph Editing*. Theoretical Computer Science, 461:45–54, 2012. Zitiert auf S. 3 and 6.

[15] CHEN, DUHONG, OLIVER EULENSTEIN, DAVID FERNÁNDEZ-BACA und MICHAEL J. SANDERSON: *Minimum-Flip Supertrees: Complexity and Algorithms*. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 3(2):165–173, 2006. Zitiert auf S. 3 and 6.

[16] BÖCKER, SEBASTIAN, QUANGBAOANH BUI und ANKE TRUSS: *An Improved Fixed-Parameter Algorithm for Minimum-Flip Consensus Trees*. In: *Parameterized and Exact Computation*, Band 5018 der Reihe *Lecture Notes in Computer Science*, Seiten 43–54. Springer Berlin Heidelberg, 2008. Zitiert auf S. 3 and 6.

[17] LOKSHTANOV, DANIEL, MARTIN VATSHELLE und YNGVE VILLANGER: *Independent Set in P_5-Free Graphs in Polynomial Time*. In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, Seiten 570–581, 2014. Zitiert auf S. 4.

[18] HOÀNG, CHÍNH T., MARCIN KAMINSKI, VADIM V. LOZIN, JOE SAWADA und XIAO SHU: *A Note on k-Colorability of P5-Free Graphs*. In: *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, Seiten 387–394, 2008. Zitiert auf S. 4.

[19] BASCÓ, GÁBOR und ZSOLT TUZA: *Dominating cliques in P_5-free graphs*. Periodica Mathematica Hungarica, 21(4):303–308, 1990. Zitiert auf S. 5.

[20] GILBERT, EDGAR N: *Random graphs*. The Annals of Mathematical Statistics, Seiten 1141–1144, 1959. Zitiert auf S. 7.

[21] KUNEGIS, JÉRÔME: *KONECT – The Koblenz Network Collection*. In: *Proceedings International Conference on World Wide Web Companion*, Seiten 1343–1350, 2013. Zitiert auf S. 9.

[22] CHATR-ARYAMONTRI, ANDREW, BOBBY-JOE BREITKREUTZ, SVEN HEINICKE, BOUCHER LORRIE, ANDREW WINTER, CHRIS STARK, JULIE NIXON, LINDSAY RAMAGE, NADINE KOLAS, LARA O'DONNELL et al.: *The BioGRID interaction database: 2013 update*. Nucleic Acids Research, 41(D1):D816–D823, 2013. Zitiert auf S. 9.

[23] RAHMANN, SVEN, TOBIAS WITTKOP, JAN BAUMBACH, MARCEL MARTIN, ANKE TRUSS und SEBASTIAN BÖCKER: *Exact and Heuristic Algorithms for Weighted Cluster Editing*. In: *Proceedings of Computational Systems Bioinformatics (CSB 2007)*, Band 6, Seiten 391–401, 2007. Zitiert auf S. 10.

[24] BÖCKER, SEBASTIAN, SEBASTIAN BRIESEMEISTER, QUANG BAO ANH BUI und ANKE TRUSS: *A fixed-parameter approach for Weighted Cluster Editing*. In: *Proceedings of Asia-Pacific Bioinformatics Conference (APBC 2008)*, Band 5, Seiten 221–220. Imperial College Press, 2008. Zitiert auf S. 10.

[25] HOÀNG, CHÍNH T, MARCIN KAMIŃSKI, JOE SAWADA und R SRITHARAN: *Finding and listing induced paths and cycles*. Discrete Applied Mathematics, 161(4):633–641, 2013. Zitiert auf S. 17, 18, 19, and 22.

[26] VERGARA, JOHN PAUL C., LENWOOD S. HEATH, VIRGINIA POLYTECHNIC INSTITUTE und STATE UNIVERSITY. DEPARTMENT OF COMPUTER SCIENCE: *Edge-packing by Isomorphic Subgraphs*. Department of Computer Science, Virginia Polytechnic Institute and State University, 1990. Zitiert auf S. 28.

[27] HOCHBAUM, DORIT S.: *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997. Zitiert auf S. 32.

[28] CHEN, JIANER und JIE MENG: *A 2k kernel for the cluster editing problem*. Journal of Computer and System Sciences, 78(1):211–220, 2012. Zitiert auf S. 35.

[29] GUILLEMOT, SYLVAIN, FRÉDÉRIC HAVET, CHRISTOPHE PAUL und ANTHONY PEREZ: *On the (Non-)Existence of Polynomial Kernels for Pl-Free Edge Modification Problems*. Algorithmica, 65(4):900–926, 2013. Zitiert auf S. 35.

[30] KOMUSIEWICZ, CHRISTIAN und JOHANNES UHLMANN: *A cubic-vertex kernel for flip consensus tree*. Algorithmica, 68(1):81–108, 2014. Zitiert auf S. 35 and 37.

[31] CAI, LEIZHEN und YUFEI CAI: *Incompressibility of H-Free Edge Modification Problems*. Algorithmica, 71(3):731–757, 2015. Zitiert auf S. 35.

[32] HSU, WEN-LIAN und TZE-HENG MA: *Substitution decomposition on chordal graphs and applications*. In: *ISA'91 Algorithms*, Band 557 der Reihe *Lecture Notes in Computer Science*, Seiten 52–60. Springer Berlin Heidelberg, 1991. Zitiert auf S. 37.

Erklärung der Urheberschaft

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Ort, Datum Unterschrift