Programm zur Visualisierung und automatisierten Ableitung von Beziehungen zwischen Graphparametern

Bachelorarbeit

Fabian Spieß

Matrikelnummer: 331661

Eingereicht am: 10. Februar 2014

Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik
Institut für Softwaretechnik und Theoretische Informatik
Fachbereich Algorithmik und Komplexitätstheorie

Betreuer Manuel Sorge, Dr. Sepp Hartung
Gutachter Prof. Dr. Rolf Niedermeier

Eidesstattliche Versicherung

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

Berlin, den

. . . . . .
Unterschrift

Ich möchte mich bei meinen Betreuern Manuel Sorge und Dr. Sepp Hartung für die sehr gute Betreuung bedanken. Die vielen Diskussionen und hilfreichen Ratschläge halfen mir nicht nur bei der Entwicklung von $P_a H_i D_e$, sondern auch sehr bei der Verfassung dieser Bachelorarbeit.

Inhaltsverzeichnis

Zusammenfassung

Seit den frühen 1990gern setzt sich die Komplexitätstheorie mit parametrisierten Problemen auseinander. Bis heute wurde für hunderte dieser Probleme entschieden, ob sie FPT oder W-schwer sind. W-Schwere bedeutet unter anderem, dass nach heutigen Erkenntnissen wahrscheinlich kein FPT-Algorithmus für das Problem existiert. Algorithmen von FPT-Problemen sind bei einem kleinen Parameter π und einer moderat wachsenden Laufzeitfunktion schnell zu berechnen[10]. Daher stellt die parametrisierte Komplexitätstheorie einen interessanten Ansatz dar, um NP-schwere Probleme zu lösen.

Parameter bilden eine Hierarchie bzgl. ihrer Beziehungen $\pi' \leq h(\pi)$, wobei h eine Funktion von $\mathbb{R} \to \mathbb{R}$ ist und π, π' zwei Parameter sind. In dem Fall sagt man, dass π' durch π beschränkt wird. Neben der Möglichkeit einer langsamer wachsenden Laufzeitfunktion, kann man die Laufzeit von FPT-Algorithmen bei manchen Eingabeinstanzen auch durch einen kleineren Parameter optimieren. Ist ein Problem parametrisiert durch π' in FPT, dann ist es auch parametrisiert für π in FPT. Ist es allerdings für π W-schwer, dann ist es auch für π' W-schwer. Daher sollte man, solange das Problem für π in FPT ist, nicht aufhören Parametrisierungen für kleinere Parameter π' zu finden, um so mögliche Laufzeitoptimierungen nutzen zu können. Da wir bereits sehr viele Parameter kennen und immer wieder neue hinzukommen, ist es schwierig den Überblick über die Parameterhierarchie zu behalten. Aus diesem Grunde wurde im Rahmen dieser Bachelorarbeit das Programm $P_aH_iD_e$ entwickelt, welches eine Parameterhierarchie für Graphparameter aller ungerichteten Graphen strukturiert darstellen kann. Aus gegebenen Beziehungen zwischen Graphparametern leitet $P_aH_iD_e$ weitere Beziehungen ab. Dies ist vor allem interessant, wenn neue Graphparameter in die Hierarchie eingetragen werden, da sich so schnell ein größeres Wissen über alle Graphparameter und ihre Größen ansammeln lässt. $P_aH_iD_e$ kann außerdem Hinweise auf ein sinnvolles Erweitern der Graphparameterhierarchie geben.

Diese Programmfunktionen können dem Benutzer viel Arbeit in Bezug auf Beweise abnehmen und ihm helfen die Graphparameterhierarchie in ihrer Gesamtheit im Überblick zu behalten.

Abstract

Since the early 1990s, computational complexity theory works on parameterized problems. Today there are hundreds of problems which were classified to be FPT or W-hard. Among other things, for all we know W-hardness implies that there is no FPT algorithm for a problem. Algorithms of FPT problems are fast if the problem's parameter π is small and the running time function f grows moderately[10]. Hence the parameterized complexity theory is an interesting possibility to solve NP-hard problems.

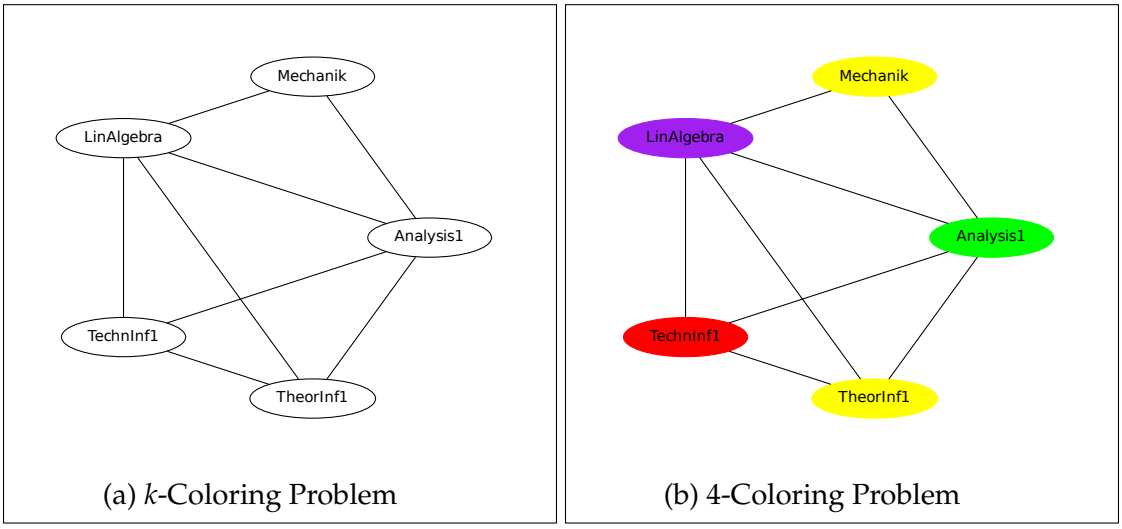
Parameters build a hierarchy of their mutual relations in the way $\pi' \leq h(\pi)$, where h is a function $\mathbb{R} \to \mathbb{R}$ and π, π' are parameters. In this case we say π' is upper bounded by π. Besides using a slower growing running time function, for some instances parameterizing a problem by a smaller parameter can also lead to an improvement of the running time. If a problem is in FPT parameterized by π', then it is also in FPT parameterized by π. Vice versa if a problem is W-hard when parameterized by π, then it is also W-hard when parameterized by π'. So we should not stop to find smaller parameters π', as long as the problem is for π in FPT. Because many parameters are known and we will get to know more, it is hard to have a structured overview on the parameter hierarchy. That is why we developed the program $P_aH_iD_e$, which draws a parameter hierarchy for graph parameters of all undirected graphs. Moreover $P_aH_iD_e$ derives new graph parameter relations from given graph parameter relations. This is mostly interesting if we add new graph parameters into the hierarchy, because it quickly creates a bigger knowledge about the graph parameters and their sizes. Moreover $P_aH_iD_e$ is able to give hints for a usefull extending of the graph parameter hierarchy.

These program features can be very helpful for the user because it saves time for proofs.

1 Einleitung

In der theoretischen Informatik beschäftigt sich ein Teilgebiet der Komplexitätstheorie mit dem Lösen von NP-schweren Problemen. Diese Probleme existieren nicht nur in theoretischen Überlegungen, sondern lassen sich überall in unserem Lebensalltag wiederfinden. Ausgehend von der Annahme, man hat eine Menge von Studenten, eine Menge von Veranstaltungen und eine Menge von Terminen, wobei alle Studenten bestimmte Vorlesungen besuchen möchten. Die Frage ist, wie viele Termine mindestens gebucht werden müssen, sodass alle Studenten ihre Vorlesungen besuchen können und sich ihre Termine nicht überschneiden. Das hier vorgestellte Problem ist eines der in der Komplexitätstheorie bekannten *Scheduling-Probleme* und ist NP-schwer. Zu bemerken ist, dass nicht alle Scheduling-Probleme NP-schwer sein müssen. Für eine Universität mit einigen tausend Studenten wird dieses Problem offensichtlich sehr aufwendig zu lösen sein, wenn man nicht die Rechenleistung von Computern nutzt. Doch auch das Lösen dieses Problems mittels eines Computers ist in der Praxis schwierig, da das Wachstum der Laufzeitfunktion eines problemlösenden Algorithmus für NP-schwere Probleme exponentiell steigt und damit ineffizient ist. Das bedeutet, dass bei großer Komplexität solcher Probleme selbst die schnellsten Computer tausende von Jahren benötigen könnten, um diese zu lösen.

„Die Tatsache, dass [diese Probleme] NP-[schwer] sind und sich folglich einer effizienten Lösbarkeit entziehen, verringert weder den Wunsch noch die Notwendigkeit, sie zu lösen“[6, S. 130]. Hierfür werden in der Komplexitätstheorie verschiedene Ansätze betrachtet und erforscht. Zum einen gibt es den Ansatz der *Randomisierten Algorithmen*. Diese sind effizient (in polynomieller Zeit zu lösen), jedoch können die Lösungen beispielsweise Fehler beinhalten, da sie Gebrauch von Zufallsentscheidungen machen. *Approximationsalgorithmen* stellen einen zweiten Ansatz dar. Hier können die Lösungen auch Fehler enthalten, doch für diese Algorithmen existieren Beweise, die eine obere Schranke für den Fehler angeben. Ein dritter Ansatz sind exakte Algorithmen, die versuchen exakte Lösungen zu errechnen, jedoch nicht effizient sind. Forschungen in Bezug auf diese Algorithmen zeigen, dass viel Potenzial besteht, die Laufzeit zu beschleunigen [7]. Zum Beispiel lässt sich das oben beschriebene Scheduling-Problem als das *k-Coloring Problem* der Graphentheorie modellieren. Ein Beispielgraph unseres Scheduling-Problems ist in Abbildung 1.1a zu sehen. Für dieses Beispiel ist es ausreichend, sich zunächst einmal unter einem Graphen eine Struktur wie in Abbildung 1.1a vorzustellen. Sie besteht aus Knoten (hier mit Fächernamen beschriftet) und ihren Verbindungen (Kanten).



(a) k-Coloring Problem (b) 4-Coloring Problem

Abbildung 1.1: Scheduling Problem

Veranstaltungen für Informatikstudenten bestehen aus den Fächern Technische Informatik 1 (TechInf1), Theoretische Informatik 1 (TheorInf1), Analysis 1 (Analysis1) und Lineare Algebra (LinAlgebra). Bauingenieure müssen die Fächer Mechanik, Lineare Algebra und Analysis 1 belegen. Folglich dürfen die Informatikfächer nicht zeitgleich stattfinden und sich auch nicht mit den Terminen der Mathematikfächer überschneiden. Mechanik darf nicht an den Terminen der Mathematikfächer stattfinden.

Das Scheduling-Problem kann demnach in folgender Weise als k-Coloring Problem dargestellt werden: Können wir mit k Farben (Termine) den Graph aus Abbildung 1.1a einfärben, sodass keine zwei Knoten, die durch eine Kante verbunden sind, dieselbe Farbe haben?

Das kleinste k, mit dem wir das Problem mit „Ja“ entscheiden können, ist vier (siehe Abbildung 1.1b).

Man benötigt also vier Termine, um die Veranstaltungen so auszurichten, dass jeder Student seine Veranstaltungen ohne Schwierigkeiten besuchen kann. Ein naiver Algorithmus eines *k-Coloring Problems* hat eine Laufzeit von $\mathcal{O}(k^n \cdot n^2)$, wobei k die Anzahl der Farben bezeichnet und n die Anzahl der Knoten in dem Graphen. Die $\mathcal{O}$-Notation stellt hier die obere Schranke dar. Das bedeutet, dass die Laufzeitfunktion des Algorithmus das asymptotische Wachstum von $k^n \cdot n^2$ nicht überschreitet. Der naive Algorithmus probiert alle Farbkombinationen durch, was in diesem sehr kleinen Beispiel schon maximal $4^5 \cdot 5^2 = 25.600$ Rechenschritte benötigen würde. Mittlerweile ist das k-Coloring Problem gut erforscht und es existiert ein Algorithmus mit einer oberen Schranke von $\mathcal{O}(2^n \cdot n)$, unabhängig von der Farbanzahl k [2]. In diesem Fall müsste der Algorithmus nur maximal $2^5 \cdot 5 = 160$ Schritte berechnen.

Es ist somit offensichtlich, dass uns die Forschung an NP-schweren Problemen in

der Realität helfen kann. Eine relativ neue und vielversprechende Idee ist das Betrachten NP-schwerer Probleme in der *Parametrisierten Komplexitätstheorie*[4][14][5]. Diese wurde von Downey und Fellows in den 1990ger Jahren eingeführt [4]. Das Ziel ist einen Algorithmus in seinen exponentiellen Laufzeitanteilen auf einen Parameter κ zu beschränken. In dieser Arbeit werden ausschließen Parameter für Graphprobleme, also Graphparameter betrachtet.

Definition 1 (**Graphparameter**)**.** Ein *Graphparameter* ist eine Funktion κ mit $\kappa : \mathbb{G} \to \mathbb{R}$, wobei $\mathbb{G}$ die Menge aller endlichen ungerichteten Graphen ist und $\mathbb{R}$ die Menge der reellen Zahlen.

Unter einem Graphparameter versteht man eine Eigenschaft eines Graphen, beispielsweise den größten Knotengrad, die Größe einer größtmöglichen Clique oder den längsten kürzesten Pfad. Haben wir uns auf einen Graphparameter κ festgelegt, durch den wir den exponentiellen Laufzeitanteil eines Problems beschränken, dann ist das Problem durch κ *parametrisiert* (siehe Beispiel 3). Alle in dieser Arbeit erwähnten Graphparameter sind in dem zusätzlichen Dokument „The Graph Parameter Hierarchy“ definiert, welches im Rahmen dieser Arbeit entstanden ist. Wenn im Folgenden konkret mit bestimmten Graphparametern gearbeitet wird, sind diese vorher in der Arbeit definiert. Allerdings werden auch einige Graphparameter nur als Beispiel genannt und können in dem Zusatzdokument nachgeschlagen werden.

Die „Parametrisierte Komplexitätstheorie“ unterscheidet grundlegend die Komplexitätsklassen **FPT** und **XP**. Die Algorithmen von FPT-Problemen sind bei einer kleinen Parametergröße schnell und können so eine Verbesserung gegenüber anderen exakten Algorithmen darstellen. Im Folgenden werden FPT und XP für Graphenprobleme definiert.

Definition 2 (**FPT**)**.** Es sei Π ein durch $\kappa : \mathbb{G} \to \mathbb{R}$ parametrisiertes Problem und $\mathbb{G}$ die Instanzenmenge aller Graphen von Π.
Π ist *fixed-parameter tractable* bezüglich κ, falls es eine Funktion $f : \mathbb{R} \to \mathbb{R}$ gibt, sodass für alle $G \in \mathbb{G}$ die Laufzeit eines Algorithmus für Π, mit der Eingabegröße der Instanz $n = |G|$, durch

$$f(\kappa(G)) \cdot n^{\mathcal{O}(1)}$$

abgeschätzt werden kann.
FPT ist die Menge aller Probleme Π, die *fixed-parameter tractable* sind.

Liegt ein Problem in „FPT“, so lassen sich Aussagen über das effiziente Lösen des Problems durch die Betrachtung seiner Funktion f treffen. Da f sehr häufig eine Funktion wie etwa $f(x) = c^x$ mit $c \in \mathbb{R}$ ist, sind FPT-Algorithmen schnell, wenn der Graphparameter x klein ist und f moderat wächst [10, S. 19].

Definition 3 (**XP**)**.** Es sei Π ein durch $\kappa : \mathbb{G} \to \mathbb{R}$ parametrisiertes Problem und $\mathbb{G}$ die Instanzenmenge aller Graphen von Π.

Π ist in XP bezüglich κ, falls berechenbare Funktionen f,$g : \mathbb{R} \to \mathbb{R}$ existieren, sodass für alle $G \in \mathbb{G}$ die Laufzeit eines Algorithmus mit der Eingabengröße der Instanz $n = |G|$ durch

$$f(\kappa(G)) \cdot n^{g(\kappa(G))}$$

abgeschätzt werden kann.

Man weiß, dass **FPT** $\subsetneq$ **XP** gilt [4, Kapitel 15]. Wurde ein Problem bezüglich eines Graphparameters als fixed-parameter tractable nachgewiesen, ist eine Optimierungsmöglichkeit, das Funktionswachstum von $f(\kappa)$ zu verringern, um die Laufzeit zu verbessern.[1]

Ein anderer interessanter Ansatz ist Probleme mit „kleineren“ Graphparametern zu lösen [10]. Ein Graphparameter π ist kleiner als ein anderer Graphparameter π', wenn eine Funktion f existiert, sodass $\pi \leq f(\pi')$ gilt und das keine Funktion h existiert mit der $\pi' \leq h(\pi)$ gilt (Definition 16). Bestätigen sich diese Beziehungen, dann gilt, dass π' den Graphparameter π strikt beschränkt oder π kleiner als π' bzw. π' größer als π ist. Der Graphparameter π ist nur in dem Sinne kleiner, als dass sein Wert immer kleiner oder gleich $f(\pi')$ ist. Es gilt nicht unbedingt, dass $\pi \leq \pi'$ ist. Diese Beziehung lässt sich hierarchisch darstellen.

Da Graphparameter alle möglichen Eigenschaften von Graphen beschreiben, können Probleme mit verschiedenen Graphparametern parametrisiert werden. Das Parametrisieren eines FPT-Problems P mit verschieden Graphparametern κ benötigt unterschiedliche Funktionen f, mit denen man P in $f(\kappa(G)) \cdot n^{\mathcal{O}(1)}$ lösen kann. Schneller wachsende Funktion f und ein kleinerer Graphparameter κ können bei manchen Eingabeinstanzen zu einem besseren Gesamtergebnis führen [10].

Ist ein Problem P für einen Graphparameter π in FPT , dann gilt für alle größeren Graphparameter als π, durch die man P parametrisieren kann, dass P mit diesen Graphparametern auch in FPT liegt. Ob P auch mit kleineren Graphparametern in FPT liegt, muss untersucht werden.

Allerdings lässt sich eine Aussage treffen, wenn ein parametrisiertes Problem für einen Graphparameter W-schwer ist. Die Parametrisierte Komplexitätstheorie lässt sich in weitere Klassen (W-Klassen) unterteilen, die sich über bestimmte Probleme definieren. Es gilt:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[t] \subseteq W[P] \subseteq XP.$$

Bei dem heutigen Forschungsstand geht man von $FPT \neq W[1]$ aus [7, S. 16]. Das bedeutet, dass es für W-schwere Probleme keinen FPT-Algorithmus gibt. Die W-Schwere überträgt sich in der Hierarchie auf kleinere Graphparameter. Ist P parametrisiert durch π W-schwer, dann gilt für alle kleineren Graphparameter als π, dass P mit diesen Graphparametern auch W-schwer ist. Diese Erkenntnis liefert uns die Möglichkeit Grenzen festzustellen. Wäre P mittels π' W-schwer und mit π fixed-parameter

[1] Internetseite für FPT „Rennen“: http://fpt.wikidot.com/fpt-races

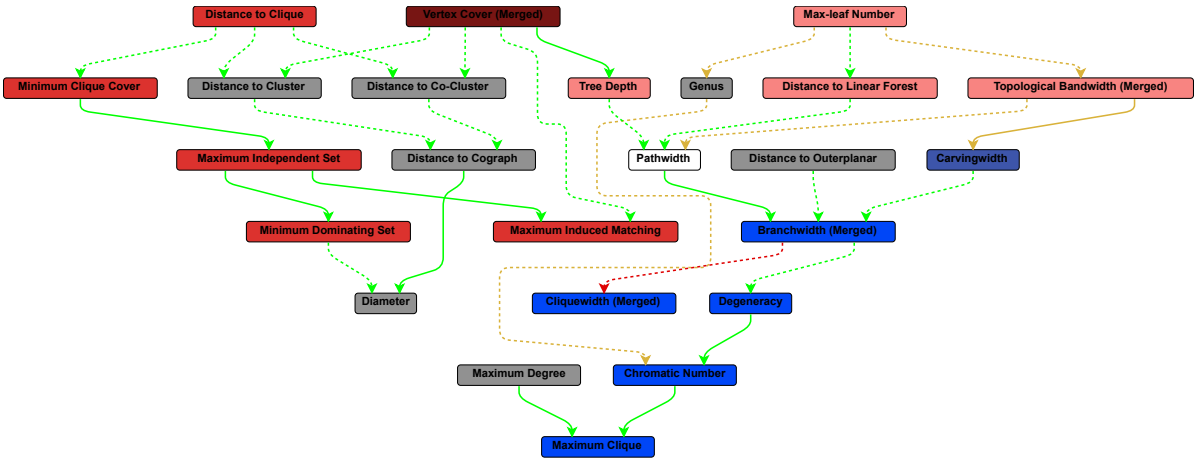


Abbildung 1.2: Darstellung der Graphparameterhierarchie

tractable, dann muss einer der Graphparameter, die in der Hierarchie zwischen π und π' liegen, der kleinste sein, mit welchem P in FPT ist. Beispielsweise ist das Graphproblem BANDWIDTH in FPT, wenn es mit dem Graphparameter *Vertex Cover* parametrisiert wird, aber W-schwer, wenn der Graphparameter die *Treewidth* ist. Ein Blick in die Graphparameterhierarchie wird zeigen, dass einige Graphparameter zwischen *Vertex Cover* und *Treewidth* liegen, durch die Parametrisierungen für BANDWIDTH in FPT möglich wären. „Wir sollten uns also nicht zufrieden geben, bis wir genau wissen, ab welchem Graphparameter [die Komplexität von BANDWIDTH] von fixed-parameter tractable zu W-schwer wechselt“[7, S.30].[2]

Das im Rahmen dieser Bachelorarbeit entstandene Programm $P_aH_iD_e$ kann mittels einer Datenbank von Graphparametern und deren Beziehungen die erwähnte Graphparameterhierarchie für Graphenprobleme aufbauen. Durch die hierarchische Darstellung ist die Größe der Graphparameter schnell auszumachen. Außerdem lassen sich durch ein Einfärben der Graphparameter schnell bekannte Beziehungen zwischen zwei Graphparametern erkennen und geben in Kombination mit der hierarchischen Darstellung Hinweise auf weitere Beziehungen, die noch zu beweisen sind. Des Weiteren kann $P_aH_iD_e$ neue korrekte Beziehungen aus gegebenen Beziehungen ableiten, was den Benutzer in seiner Arbeit unterstützt und eventuell noch nicht bekannte Beziehungen hinzufügt. Dies spart Zeit, denn oftmals sind nur wenige Beweise von Beziehungen notwendig, um einige neue Beziehungen ableiten zu können. Die hierarchische Darstellung hilft interessante Graphparameter für Laufzeitverbesserungen von FPT-Problemen auszuwählen oder Grenzen, wie es bei BANDWIDTH der Fall ist, zu finden. Ein Beispiel für die Graphparameterhierarchie, so wie sie $P_aH_iD_e$ mit einigen aktivierten Darstellungsoptionen anzeigt, ist in der Abbildung 1.2 zu sehen.

[2]Original: We should not be satisfied until we know exactly where the complexity of the problem switches from FPT to $W[1]$-hard.

$P_aH_iD_e$ wurde zur Visualisierung der Graphparameterhierarchie entwickelt, jedoch ist die Darstellung anderer Parameterhierarchien mit $P_aH_iD_e$ nicht ausgeschlossen. Für die Bedienung von $P_aH_iD_e$ wurde im Rahmen der Arbeit eine Bedienungsanleitung erstellt, welche zusammen mit dem Programm erhältlich ist. Aus diesem Grunde wird sich diese Bachelorarbeit ausschließlich mit den inhaltlichen Aspekten des Programms befassen.

2 Grundlegende Definitionen

Um ausführlicher auf die einzelnen Funktionen von $P_aH_iD_e$ eingehen zu können, werden im Folgenden grundlegende Begriffe der Graphtheorie definiert. $P_aH_iD_e$ arbeitet mit Graphparametern von ungerichteten Graphen.

Definition 4 (Ungerichteter Graph). Ein *ungerichteter Graph* G ist ein Tupel $G = (V, E)$ zweier disjunkter Teilmengen, wobei V eine Menge von **Knoten** und $E \subseteq [V]^2$ eine Menge von **Kanten** ist, welche Knoten miteinander verbindet. $[V]^2$ ist hier die Menge aller zweielementigen Teilmengen von V. Die Menge aller ungerichteten Graphen wird im Folgenden durch $\mathbb{G}$ bezeichnet.

$P_aH_iD_e$ stellt die Graphparameter und ihre Beziehungen in einer Hierarchie grafisch dar. Diese Hierarchie wird durch einen gerichteten Graphen gezeichnet.

Definition 5 (Gerichteter Graph). Ein *gerichteter Graph* G ist ein Tupel $G = (V, E)$ zweier disjunkter Teilmengen, wobei V eine Menge von Knoten und $E \subseteq \{(v_x, v_y) \mid v_x, v_y \in V\}$ eine Menge mit geordneten Tupeln ist.

Um gewisse Informationen und Darstellungen berechnen zu können, ist es nicht ausreichend, die Graphparameterhierarchie durch einen gerichteten Graphen darzustellen. Es ist weiterhin notwendig, diesen Graphen zu gewichten, um z.B. bestimmte Kanten filtern zu können.

Definition 6 (Gewichteter Graph). In einem *gewichteten Graphen* $G = (V, E)$ existiert eine Funktion f mit $f : E \to \mathbb{R}$, die jeder Kante $e \in E$ eine Zahl $x \in \mathbb{R}$ zuordnet.

Definition 7 (Größe eines Graphen). Die *Größe* eines Graphen $|G|$ ist die Anzahl seiner Knoten.

Definition 8 (Benachbart). Zwei Knoten v_1, v_2 sind in einem Graphen $G = (V, E)$ *benachbart*, wenn $\exists e \in E : e = \{v_1, v_2\}$.

Definition 9 (Pfad). Ein *Pfad* P in einem Graphen ist ein n-Tupel $P = (v_x, \ldots, v_n)$ mit $v_x \neq \ldots \neq v_n \in V$, in dem alle zwei aufeinander folgenden Knoten benachbart sind.

Definition 10 (Pfadlänge). Die *Länge eines Pfades* P beschreibt die Anzahl der Kanten in einem Pfad und wird durch $|P|$ bezeichnet.

Definition 11 (Kürzester Pfad). Sei P die Menge aller Pfade von v_1 zu v_n in einem Graphen $G = (V, E)$ mit $v_1, \ldots, v_n \in V$ und $n \in \mathbb{N}$.
Der *kürzeste Pfad* in P ist der, welcher die kleinste Anzahl an Kanten beinhaltet. Bei einem gewichteten Graphen G besitzt der *kürzeste Pfad* P_x von P das kleinste Gesamtgewicht aller Kanten in P_x.

Definition 12 (Kreise). Ein Graph $G = (V, E)$ beinhaltet einen Kreis, wenn es einen Pfad $P = (v_x, \ldots, v_x)$ mit $v_x \in V$ gibt.

Definition 13 (Knotengrad). Der Knotengrad $deg_G(v)$ eines Knoten v in einem Graphen ist die Anzahl der benachbarten Knoten von v.

3 Graphparameter

Wie in der Einleitung kurz geschildert, liegt die Hauptaufgabe von $P_aH_iD_e$ in der grafischen Darstellung der Graphparameterhierarchie. Bevor jedoch auf die Funktionsweisen und die Arbeit mit $P_aH_iD_e$ eingegangen werden kann, müssen zunächst die Konzepte der Graphparameter und ihrer Beziehungen untereinander geklärt werden.

3.1 Beispiele für Graphparameter

Wie bereits festgestellt, beschreiben Graphparameter bestimmte Eigenschaften von Graphen. Oft sind diese Eigenschaften Größen von Teilmengen des Graphen. Aber auch bestimmte Pfadlängen oder Aspekte bestimmter Knoten können Graphparameter sein. Darüber hinaus gibt es noch andere Möglichkeiten. Um solche Eigenschaften zu veranschaulichen, werden nun als Beispiele die Graphparameter *Vertex Cover Number* und *Local Modificationbound* vorgestellt.

Beispiel 1 (*Vertex Cover Number* κ)**.** Ein *Vertex Cover* in einem Graphen $G = (V, E)$ ist eine Teilmenge $S \subseteq V$ von Knoten, sodass für jede Kante $e = \{a, b\} \in E$ und $a, b \in V$ gilt, dass $a \in S$ oder $b \in S$. Der Graphparameter *Vertex Cover Number* ist die Größe des kleinsten Vertex Covers in G.

Der Parameter *Vertex Cover Number* beschreibt die Größe einer Lösungsmenge für ein klassisches Graphenproblem. Parameter können aber auch noch weitere Eigenschaften beschreiben. Hierfür dient als Beispiel die *Local Modificationbound*, welche eine Anzahl an Kantenveränderungen (Kanten zwischen zwei Knoten hinzufügen oder löschen) in einem Graphen beschreibt.

Beispiel 2 (*Local Modificationbound* τ)**.** Sei $G = (V, E)$ ein ungerichteter Graph und S eine Menge von Kantenveränderungen für G. Die Menge S ist lokal-τ-beschränkt, wenn für alle Knoten $v \in V$ gilt, $|S \cap \{\{u, v\} \in E,\ u \in V \setminus \{v\}\}| \leq \tau$ [9, S. 21].

Informell bedeutet dies, dass eine lokal-τ-beschränkte Menge S maximal τ Kantenmodifizierungen an jedem Knoten aus dem Eingabegraphen vornimmt.

Nun kann man Graphparameter nutzen, indem man Graphenprobleme durch sie parametrisiert. Das folgende Beispiel zeigt das parametrisierte Graphproblem VERTEX COVER.

Beispiel 3 (Vertex Cover). Eine Parametrisierung eines Problems kann man darüber erhalten, indem man den in der Probleminstanz vorkommenden Graphparameter wählt. Diese wird häufig als natürliche Parametrisierung bezeichnet. Bei VERTEX COVER ist das der Fall.

	VERTEX COVER
Gegeben:	Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$.
Graphparameter:	k
Frage:	Existiert eine Menge $S \subseteq V$, mit $\|S\| \leq k$, von Knoten, sodass mindestens einer der Knoten $x, y \in V$ von jeder Kante $e = \{x, y\} \in E$, in S ist?

Da der Graphparameter *Vertex Cover Number* κ die Anzahl des kleinsten Vertex Cover in G beschreibt, können wir mit $k = \kappa(G)$ das Problem mit „Ja“ entscheiden.

Das Vertex Cover Problem ist fixed-parameter tractable [10] und ist in einer Laufzeit von $1,2738^{\kappa(G)} \cdot poly(n)$ zu lösen [3]. Hier ist $n = |G|$ die Größe der Instanz mit $G \in \mathbb{G}$. Es existieren auch Probleme, die sich über andere Graphparameter parametrisieren lassen, zum Beispiel das CLUSTER EDITING Problem.

Beispiel 4 (Cluster Editing). Im Fall von CLUSTER EDITING können für eine Parametrisierung die Graphparameter *Local Modificationbound* τ und die *Obere Schranke von Clustern in einem Clustergraphen cc* verwendet werden [10]. Ein Clustergraph ist ein Graph, der aus einer oder mehreren Komponenten besteht, wobei jede Komponente eine Clique ist.

	CLUSTER EDITING
Gegeben:	Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$.
Graphparameter:	*Local Modificationbound* $\tau(G)$, Obere Schranke an Clustern in dem entstehenden Clustergraphen $cc(G)$
Frage:	Kann man G in einen Clustergraphen mit maximal k Kantenveränderungen transformieren?

Mit dieser Parametrisierung ist CLUSTER EDITING in FPT und kann mit einer Laufzeit von $2^{\mathcal{O}(cc \cdot \tau)}$ gelöst werden [10].

Wie schon erwähnt, steht bei $P_a H_i D_e$ der Aspekt der Laufzeitoptimierung von FPT-Problemen durch kleinere Graphparameter im Vordergrund. Um zu diesem Ziel zu gelangen, muss zunächst geklärt werden, wie genau sich kleinere Graphparameter definieren und welche Beziehungen zwischen Graphparametern sonst noch existieren. Erst dadurch lässt sich die Graphparameterhierarchie vollständig aufbauen und kleinere Graphparameter finden.

3.2 Beziehungen von Graphparametern

Der erste Schritt um kleinere Graphparameter bestimmen zu können, ist der Beweis einer *oberen Schranke* zwischen zwei Graphparametern. Diese Beziehung bildet auch die Grundlage für die Graphparameterhierarchie.

Definition 14 (Obere Schranke). Seien $G = (V, E)$ ein Graph und α, β zwei Graphparameter, dann ist β eine *obere Schranke* von α (oder β *beschränkt* α), wenn

$$\exists f : \mathbb{R} \to \mathbb{R}.\ \forall G \in \mathbb{G} : \alpha(G) \leq f(\beta(G))$$

gilt, kurz $\alpha \preceq_{\mathbb{G}} \beta$.

Eine obere Schranke wird im folgenden als *linear* bezeichnet, wenn f eine lineare Funktion ist. Sie heißt *polynomiell*, wenn f polynomiell ist und dementsprechend *exponentiell*, wenn f eine exponentielle oder stärker wachsende Funktion ist.

Eine obere Schranke bedeutet, dass der Wert der Funktion f, welche als Eingabe den Wertebereich des Graphparameter β erhält, immer größer/gleich dem Wert des Graphparameters α ist. Beispielsweise beschränkt der Graphparameter *Maximum Degree* Δ, welcher der größte Knotengrad aller Knoten in einem Graphen ist, den Graphparameter *Maximum Clique* γ. *Maximum Clique* beschreibt die Anzahl der Knoten in der größten Clique eines Graphen. Der Wert $\gamma - 1$ ist das kleinstmögliche Δ in einem Graphen G. Deswegen gilt $\gamma \preceq_{\mathbb{G}} \Delta$ zum Beispiel mit der Funktion $f(x) = x + 1$.

Kann man hingegen zeigen, dass für zwei Graphparameter keine Funktion f existiert, für die $\alpha(G) \leq f(\beta(G))$ gilt, so gilt in einem solchen Fall, dass β den Graphparameter α nicht beschränkt.

Definition 15 (Unbeschränktheit). Seien $G = (V, E)$ ein Graph und α, β zwei Graphparameter. Wenn

$$\nexists f : \mathbb{R} \to \mathbb{R}.\ \forall G \in \mathbb{G} : \alpha(G) \leq f(\beta(G))$$

gilt, dann ist α durch β *unbeschränkt*, kurz geschrieben $\alpha \not\preceq_{\mathbb{G}} \beta$.

Dieser Fall gilt auch, wenn man Δ und γ betrachtet. Denn der *Maximum Degree* ist unbeschränkt durch die *Maximum Clique*. Bei Graphen, die Sterne darstellen (nur Kanten von einem Knoten zu allen anderen), kann Δ beliebig groß werden, γ hingegen ist immer maximal zwei. Daher gilt also $\gamma \not\preceq_{\mathbb{G}} \Delta$.

Des Weiteren wird nun der Begriff der „strikten oberen Schranke“ eingeführt. Dieser Fall ist gegeben, wenn ein Graphparameter α einen anderen Graphparameter β beschränkt, jedoch umgekehrt eine Unbeschränktheit vorliegt.

Definition 16 (Strikte obere Schranke). Seien α, β zwei Graphparameter. Dann ist β eine *strikte obere Schranke* von α, wenn $(\alpha \preceq_{\mathbb{G}} \beta) \wedge (\beta \not\preceq_{\mathbb{G}} \alpha)$, geschrieben $\alpha \prec_{\mathbb{G}} \beta$, gilt. Alternativ ist α *kleiner* als β.

Auch hier dienen wieder die Graphparameter *Maximum Clique* und *Maximum Degree* als Beispiel, da genau die beiden Eigenschaften für diese Graphparameter gelten.

Tritt allerdings dieser Fall nicht auf, sondern α und β beschränken sich gegenseitig, so spricht man von einer „Gleichheit“.

Definition 17 (Gleichheit). Seien α, β zwei Graphparameter. Die Graphparameter α und β sind *gleich*, wenn $(\alpha \preceq_{\mathbb{G}} \beta) \wedge (\beta \preceq_{\mathbb{G}} \alpha)$ gilt. Im weiteren wird die Notation $\alpha \equiv_{\mathbb{G}} \beta$ verwendet.

Hierfür kann man als Beispiel die Graphparameter *Vertex Cover* und *Maximum Matching* sehen. Denn es gilt, dass der Graphparameter *Vertex Cover* α den Graphparameter *Maximum Matching* β für alle $G \in \mathbb{G}$, mit $\beta(G) \leq \alpha(G)$ beschränkt [15, Theorem 4.5]. Aber β beschränkt auch α mit $\alpha(G) \leq 2 \cdot \beta(G)$ [15, Theorem 4.6].

Der letzte noch zu betrachtende Fall ist, wenn bewiesen ist, dass keinerlei Schranken zwischen zwei Graphparametern existieren. Diese Situation nennt sich „Unvergleichbarkeit“.

Definition 18 (Unvergleichbarkeit). Seien α, β Graphparameter. Wenn $(\alpha \not\preceq_{\mathbb{G}} \beta) \wedge (\beta \not\preceq_{\mathbb{G}} \alpha)$, also $\alpha \parallel_{\mathbb{G}} \beta$ gilt, dann sind α und β *unvergleichbar*.

Dies gilt zum Beispiel für die beiden Graphparameter *Dominating Set* und *Maximum Induced Matching* [15, Theorem 4.1, Theorem 4.2].

Die Beziehung zwischen zwei Graphparametern ist vollständig, wenn sie *gleich*, *unvergleichbar* oder der eine *kleiner* als der andere ist. Kennt man nur eine Beziehung zwischen zwei Graphparametern, also eine obere Schranke oder eine Unbeschränktheit, dann ist die Beziehung unvollständig. Warum es möglich ist, durch diese Beziehungen eine Hierarchie der Graphparameter zu bilden, wird im folgenden Kapitel erläutert.

3.3 Die Ordnung von Graphparametern

Beziehungen von Graphparametern stehen in einer Ordnungsrelation. Genau betrachtet bilden sie eine sogenannte Halbordnung, die per Definition *reflexiv*, *transitiv* und *antisymmetrisch* ist. Diese Voraussetzungen gelten für die Beziehungen der Graphparameter. Seien α, β, γ drei Graphparameter. Dann gilt, dass $\alpha \preceq_{\mathbb{G}} \beta \wedge \beta \preceq_{\mathbb{G}} \gamma \Rightarrow \alpha \preceq_{\mathbb{G}} \gamma$ (*Transitivität*), $\alpha \equiv_{\mathbb{G}} \alpha$ (*Reflexivität*) und $\alpha \preceq_{\mathbb{G}} \beta \wedge \beta \preceq_{\mathbb{G}} \alpha \Rightarrow \alpha \equiv_{\mathbb{G}} \beta$ (*Antisymmetrie*).

Um solche Halbordnungen graphisch zu repräsentieren, bietet sich die Verwendung eines „Hasse-Diagramms“ an [12, S. 51]. Ein solches Diagramm stellt eine Ordnung als gerichteten Graphen hierarchisch dar. Als Beispiel dient hier eine Darstellung der Graphparameterhierachie, so wie sie $P_a H_i D_e$ darstellt (siehe Abbildung 3.1). Die schwarzen Kanten bezeichnen die oberen Schranken der jeweiligen Graphparameter. Transitive Kanten sind zur besseren Übersicht ausgeblendet. Ferner sind alle

Graphparameter α, β für die $\alpha \equiv_G \beta$ gilt, in einem Knoten zusammengefasst, damit die Antisymmetrie in dem Hasse-Diagramm gegeben ist. Sie sind farbig gekennzeichnet und werden in der Hierarchie durch einen der gleichen Graphparameter repräsentiert.

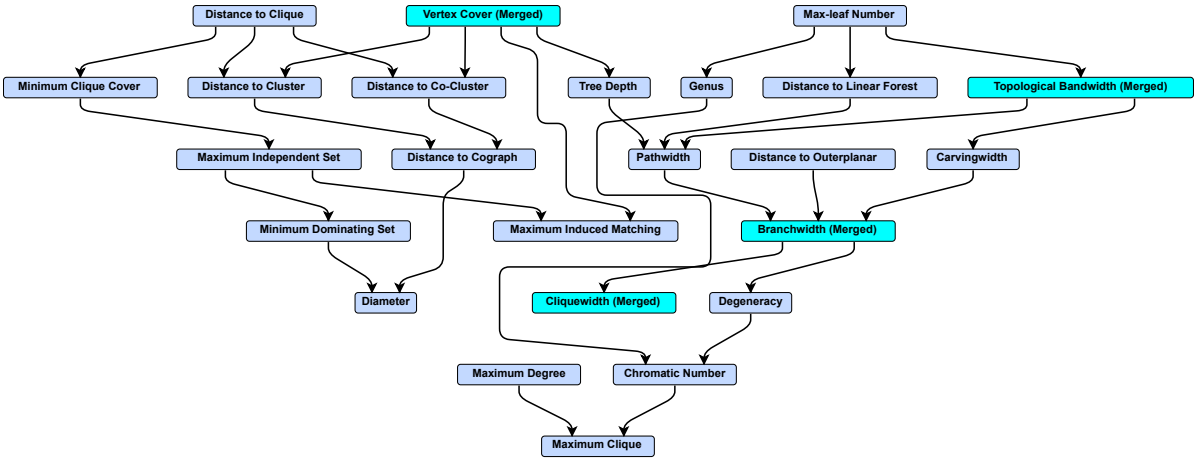


Abbildung 3.1: Hasse-Diagramm der Graphparameterhierarchie

3.4 Beispielbeweis für Graphparameterbeziehungen

Nachdem nun die Beziehungen der Graphparameter geklärt sind, folgt nun ein Beispiel für den Beweis einer strikten oberen Schranke. Der Beweis bezieht sich auf die Graphparameter *Minimum Dominating Set* und *Durchmesser*. Beweise für obere Schranken folgen immer dem selben Grundprinzip. Daher sind die Beweise, die in der Datenbank von $P_a H_i D_e$ gespeichert sind, mit dem Folgenden vergleichbar.

Definition 19 (*Minimum Dominating Set* γ)**.** Sei $G = (V, E)$ ein ungerichteter Graph. Ein Dominating Set ist eine Menge von Knoten $S \subseteq V$, für die gilt: $\forall v \in V : v \in S$ oder ein Nachbar x von v existiert mit $x \in S$. Wir sagen S überdeckt V. Der Graphparameter *Minimum Dominating Set* ist die Anzahl $|S|$ des kleinst möglichen Dominating Set S von G.

Definition 20 (*Durchmesser d*)**.** Der Durchmesser d ist $|P|$, wobei P der längste von allen kürzesten Pfaden in einem ungerichteten Graphen G ist.

Lemma 1 ($d \prec_G \gamma$)**.** Sei der Graphparameter d der *Durchmesser* eines Graphen $G \in \mathbb{G}$ und γ der Graphparameter *Minimum Dominating Set*, dann gilt für alle G, dass γ den Graphparameter d strikt beschränkt.

Beweis. Im ersten Schritt wird gezeigt dass γ eine obere Schranke für d darstellt. Im zweiten Schritt wird gezeigt, dass γ eine strikte obere Schranke für d ist.
Schritt 1:
Zu zeigen: $\gamma \succeq_{\mathbb{G}} d$, das heißt, $\exists f : \mathbb{R} \to \mathbb{R}.\ \forall G \in \mathbb{G} :\ d(G) \leq f(\gamma(G))$.
Sei $G = (V, E) \in \mathbb{G}$ und $p = (v_1, v_2, \ldots, v_n)$ mit $n \in \mathbb{N}$ ein Pfad mit $v_1, \ldots, v_n \in V$, $\{v_x, v_{x+1}\} \in E$ mit $x \in \{1, \ldots, n-1\}$ und $|p| = d(G)$. Des Weiteren sei S ein Minimum Dominating Set.
Ferner sei $p_i = (v_{3\cdot(i-1)+1}, \ldots, v_{3\cdot(i-1)+3})$ ein Tripel mit $i \in \{1, \ldots, \frac{n}{3}\}$, welches jeweils drei aufeinander folgende Knoten von p enthält. Ein Minimum Dominating Set wäre, wenn für jedes p_i genau einen Knoten $k_i \in S$ existiert, für den gilt $(k_i, y) \in E$ mit $y \in \{v_{3\cdot(i-1)+1}, \ldots, v_{3\cdot(i-1)+3}\}$. Gelte auch $(k_i, z) \in E$ mit $z \in \{v_1, v_2, \ldots, v_n\} \setminus \{v_{3\cdot(i-1)+1}, \ldots, v_{3\cdot(i-1)+3}\}$, dann wäre $|p| \neq d(G)$, da p kein kürzester Pfad mehr wäre. Ist die Anzahl der Knoten von p nicht glatt durch drei teilbar, dann würde noch ein weiterer Knoten für S benötigt. Daraus folgt, dass für $f(x) = 3 \cdot x$ gilt: $d(G) \leq f(\gamma(G))$.
Schritt 2:
Zu zeigen: $\gamma \not\succeq_{\mathbb{G}} d$, das heißt, $\nexists f : \mathbb{R} \to \mathbb{R}.\ \forall G \in \mathbb{G} :\ \gamma(G) \leq f(d(G))$.
Sei $G = (V, E)$ ein Graph wie folgt: $V = \{x, a_1, \ldots, a_n, b_1, \ldots, b_n\}$ mit $n \in \mathbb{N}$ und $E = \{\{x, y\} \mid y \in \{a_1, \ldots, a_n\}\} \cup \{\{a_1, b_1\}, \{a_2, b_2\}, \ldots, \{a_n, b_n\}\}$, dargestellt in Abbildung 3.2.

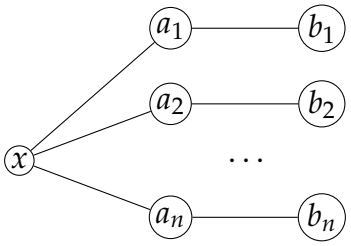


Abbildung 3.2: Der Graph G

Für G ist $d(G) = 4$ für alle n. Jedoch ist $\gamma(G) = n$. Für jede Funktion $f(x) = c$ wählt man $n = c + 1$. Daraus folgt $\nexists f :\ \gamma(G) \leq f(d(G))$. □

Durch $\gamma \succeq_{\mathbb{G}} d$ und $\gamma \not\succeq_{\mathbb{G}} d$ folgt nach Definition 16, dass $d \prec_{\mathbb{G}} \gamma$ gilt. Es wurde bewiesen, dass *Durchmesser* ein kleinerer Graphparameter als *Minimum Dominating Set* ist. Können wir ein Problem durch beide Graphparameter parametrisieren, könnte die Laufzeit für das Problem, parametrisiert durch den *Durchmesser*, kleiner sein und wäre damit eine bessere Wahl.

4 Das Programm $P_aH_iD_e$

Um mit Graphparametern und ihren Beziehungen besser arbeiten zu können, wurde $P_aH_iD_e$ entwickelt. Die Hauptaufgabe liegt darin, die Graphparameterhierarchie in einer übersichtlichen Weise darzustellen. Zu diesem Zweck gibt es eine Datenbank von Graphparametern und ihren Beziehungen, die $P_aH_iD_e$ verwendet und die sich durch das Programm verwalten bzw. modifizieren lässt. Es ist außerdem möglich, alle Informationen über die Beziehungen von zwei bestimmten Graphparametern auszulesen. Ein weiterer Aspekt, ist neue Beziehungen - sofern möglich - aus gegebenen Einträgen der Datenbank abzuleiten, um so die Hierarchie automatisch zu erweitern. Aus der Fülle an Informationen, die $P_aH_iD_e$ liefert, soll es möglich sein, Ansätze oder Hinweise zu bekommen, wie die Datenbank sinnvoll zu erweitern ist. So können durch wenige neue Beweise viele neuen Beziehungen entstehen. Ein Ziel ist es neue Parametrisierungen für Graphprobleme zu finden. Hierfür soll $P_aH_iD_e$ helfen Graphparameter auszuwählen, welche für ein Problem in Frage kommen bzw. welche sinnvoll sind, um eine Laufzeitverbesserung zu erhalten. Wie diese Anforderungen im Einzelnen umgesetzt wurden, wird in den folgenden Abschnitten näher erläutert.

4.1 Darstellung

Zur Darstellung der Graphparameterhierarchie wird die Java Bibliothek JGraphX verwendet [11]. Diese Bibliothek erlaubt es einen gerichteten Graphen G zu erstellen, zu zeichnen und diesem ein Hierarchielayout (*Hasse-Diagramm*) zu geben. Um dies korrekt bereitzustellen, verwendet JGraphX den „Sugiyama“ Algorithmus [8, S.174], welcher nicht nur G eine Hierarchie zuweist, sondern vor allem geeignet ist, um Kantenkreuzungen heuristisch zu minimieren. In $G = (V, E)$ steht jeder Knoten für einen Graphparameter π und ist nach ihm benannt. Jede gerichtete Kante des Graphen beschreibt eine obere Schranke. Also $(\pi, \pi') \in E \Rightarrow \pi' \preceq_{\mathbb{G}} \pi$. Zu beachten ist, dass nicht beschränkende Beziehungen (eine Unbeschränktheit) nicht eingezeichnet werden, da sich die Halbordnung nur auf obere Schranken bezieht. Um strikte obere Schranken oder Unbeschränktheiten zu erkennen, bietet $P_aH_iD_e$ bestimmte Darstellungsmöglichkeiten, welche in Abschnitt 4.3.2 erläutert werden. Graphparameter, welche nach Definition 17 gleich sind, werden zwar als Kreise dargestellt, können aber durch einen Filter als ein einziger Graphparameter gezeichnet werden, um die Definition der Halbordnung nicht zu verletzen.

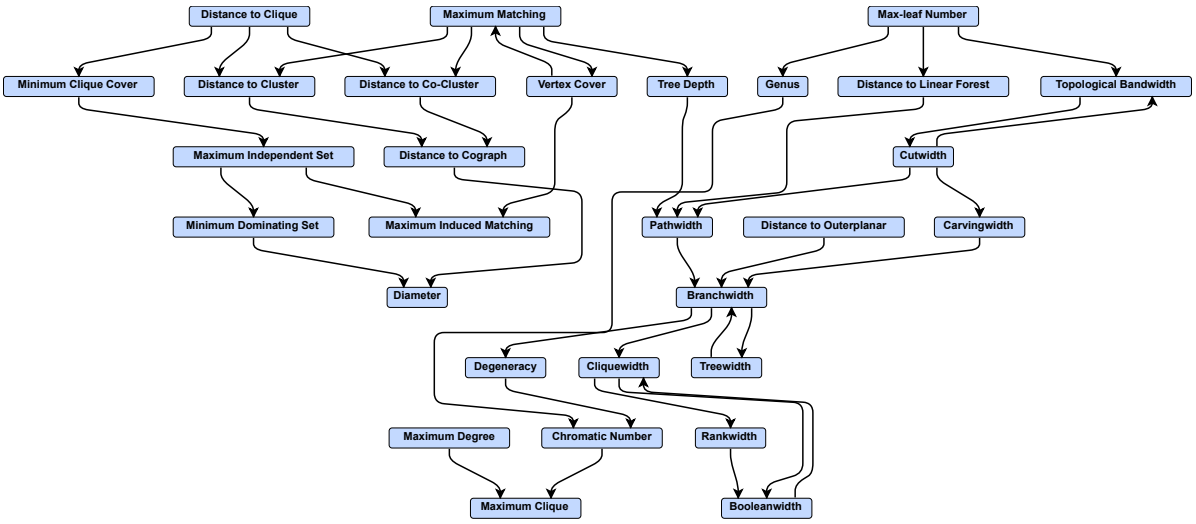


Abbildung 4.1: Standarddarstellung der Graphparameterhierarchie.

Wird die zuvor erwähnte Datenbank in $P_aH_iD_e$ geladen, erzeugt $P_aH_iD_e$ automatisch die Knoten und Kanten. Außerdem erzeugt es weitere Beziehungen, die sich aus der Transitivität der Halbordnung bzw. bestimmter Regeln, die später eingeführt werden, ableiten lassen. Die Datenbank wurde auch im Rahmen dieser Bachelorarbeit erstellt und enthält zum Zeitpunkt der Abgabe 29 Graphparameter und 57 Beziehungen zwischen ihnen. Durch das automatisierte Ableiten entstehen 260 zusätzliche Beziehungen, sodass man 29 Graphparameter und 317 Beziehungen, bestehend aus 157 oberen Schranken und 160 Unbeschränktheiten, betrachten kann.

Da die Darstellung von 29 Graphparametern und 157 oberen Schranken nicht übersichtlich ist, wird für die Standarddarstellung der Graphparameterhierarchie ein transitiver Kantenfilter aktiviert. Dieser blendet transitiven Kanten aus. Es kann jedoch die Situation entstehen, dass eine transitive obere Schranke vom Benutzer in die Datenbank eingetragen wurde, deren Funktion f ein kleineres Wachstum als das der generierten transitiven Schranke hat. Sei s eine bewiesene obere Schranke von dem Graphparameter π_1 durch den Graphparameter π_3 mit der Funktion f und $G_H = (V_H, E_H)$ der Hierarchiegraph mit $(\pi_2, \pi_1), (\pi_3, \pi_2) \in E_H$ und $\pi_1, \pi_2, \pi_3 \in V_H$. Das heißt $\exists g, h : \mathbb{R} \to \mathbb{R}.\ \pi_1(G) \leq g(\pi_2(G)) \wedge \pi_2(G) \leq h(\pi_3(G)) \Rightarrow \pi_1(G) \leq g(h(\pi_3(G)))$, mit $G \in \mathbb{G}$. Wenn f eine lineare Funktion und g oder h eine Funktion mit polynomiellem oder größerem Wachstum ist, dann ist s eine bessere obere Schranke als die durch g und h implizierte. Folglich würde der Filter s nicht ausblenden. Die entstandene Darstellung der Datenbank wird in Abbildung 4.1 visualisiert.

Durch diesen Filter und weitere Funktionen lässt sich die Darstellung ändern, vereinfachen und anpassen. Zur ausführlicheren Erläuterung müssen jedoch zunächst die Datenstrukturen innerhalb des Programms dargelegt werden.

4.2 Datenstruktur

Lädt man die Datenbank der Graphparameterhierarchie in das Programm, wird für jeden Graphparameter, jede obere Schranke und für jede Unbeschränktheit ein Knoten bzw. eine Kante erzeugt. Die dafür benutzte Graphstrukur ist ein gerichteter und gewichteter Graph, der Kreise zulässt. Der Graph beinhaltet zwei Arten von Kanten. Die eine Art beschreibt obere Schranken, die andere Unbeschränktheiten. Diese Knoten und Kanten sind Objektklassen, die an die Datenbank angepasst sind. So können die erzeugten Objekte alle Informationen der Datenbank beinhalten. Diese werden dann nicht sofort als Graph gezeichnet, sondern erst einmal als Graphdatenstruktur intern abgelegt. Für diese Datenstruktur nutzt $P_aH_iD_e$ die Java Bibliothek JGraphT [13]. Diese Bibliothek macht das Verwalten von Graphen nicht nur einfacher, sondern JGraphT hat auch schon viele gängige Graphalgorithmen wie pfadsuchende Algorithmen oder Kreiserkennungsalgorithmen, implementiert.

Insgesamt werden zwei Graphen erzeugt, wenn man eine Datenbank öffnet. Der erste Graph enthält alle 29 Graphparameter und die 317 Beziehungen. Er beinhaltet also alle Informationen, die sich nur irgend möglich aus der Datenbank ableiten lassen. Im Folgenden wird dieser Graph als *Original-Graph* bezeichnet, da dieser nicht weiter modifiziert wird, sondern nur zum Abgleich für weitere Funktionen von $P_aH_iD_e$ verwendet wird. Das Erzeugen dieses Graphen hat zwar eine Laufzeit von $\mathcal{O}(m^2 \cdot n)$ mit $n = |V|$, $m = |E|$ und $m > n$, er ermöglicht aber während der Laufzeit Zugriff auf alle existierenden Informationen der Datenbank mit einem Aufwand von $\mathcal{O}(1)$. Um später eindeutig mit den Eigenschaften des Original-Graphen arbeiten zu können, wird dieser im Folgenden definiert.

Definition 21 (Original-Graph und modifizierbarer Graph). Sei $G = (V, E)$ der Original-Graph bzw. der modifizierbare Graph, dann steht jeder Knoten $v \in V$ für einen Graphparameter. Die Menge der gerichteten Kanten E besteht aus zwei disjunkten Teilmengen S und U, mit $(\alpha, \beta) \in S \Rightarrow \beta \preceq_G \alpha$ und $(\alpha, \beta) \in U \Rightarrow \beta \npreceq_G \alpha$. Alle Kanten in S sind zusätzlich gewichtet, um später erklärte Funktionen zu ermöglichen.

Der *modifizierbare Graph* ist anfangs eine exakte Kopie des Original-Graphen. An diesem Graph können Modifikationen während der Laufzeit vorgenommen werden. Außerdem ist dieser Graph der Hierarchiegraph, welcher mittels des Programms gezeichnet wird. Viele der später erklärten Filter und Funktionen ändern den Graphen oder löschen Informationen daraus. Deswegen nutzt $P_aH_iD_e$ den Original-Graphen, um gelöschte Information zurück zu erhalten.

4.3 Darstellungsmodifikationen

Es gibt weitere Möglichkeiten, um die in der Datenbank enthaltenen Informationen zu zeichnen. Die Standarddarstellung zeigt bisher nur die Graphparameter und die Beziehungen des Typs $\alpha \preceq_G \beta$. Diese Beziehung ist aber noch nicht vollständig. Wir können erst sagen, dass α kleiner als β ist, wenn $\alpha \prec_G \beta$ gilt. Es ist außerdem interessant das Wachstum der Funktionen f zu kennen, durch die $\alpha(G) \leq f(\beta(G))$ mit $G \in \mathbb{G}$ gilt. Denn sie trifft eine Aussage über das Größenverhältnis der beiden Graphparameter.

4.3.1 Funktionswachstum der oberen Schranke

Um die Schranken der Graphparameter zu erkennen, bietet $P_a H_i D_e$ eine farbliche Darstellungsweise, die alle Kanten in dem Hierarchiegraphen einfärbt. Dabei unterscheidet das Programm in vier Typen von Kanten. Sei $G = (V, E)$ der Original-Graph, dann gilt $\forall \alpha, \beta \in V.\ (\alpha, \beta) \in S \Rightarrow \exists f : \mathbb{R} \to \mathbb{R}$ mit $\beta(G) \leq f(\alpha(G)$. Hat f ein lineares Wachstum, dann wird (α, β) grün eingefärbt. Ist f eine polynomielle Funktion, dann wird (α, β) gelb gezeichnet. Wenn f exponentiell oder schneller steigt, wird (α, β) rot eingefärbt. Ist f der Kante (α, β) unbekannt, so bleibt sie schwarz. Wenn eine Schranke der Datenbank hinzugefügt werden soll, muss man die Art der Funktion angeben. Aus diesem Grund kann $P_a H_i D_e$ einmalig über jede Kante iterieren und sie gemäß ihrer Funktion einfärben. Abbildung 4.2 zeigt eine solche Darstellung.

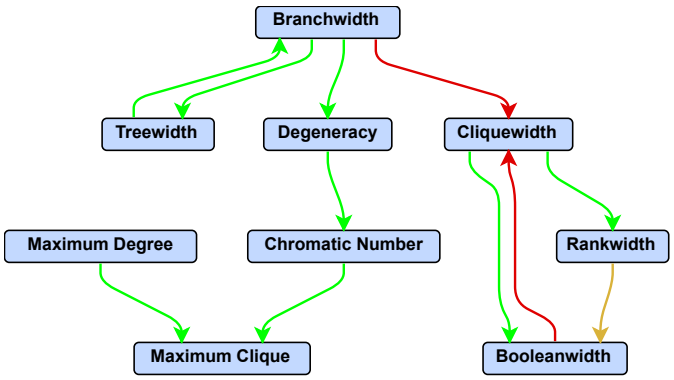


Abbildung 4.2: Darstellung der oberen Schranken gemäß ihrer Funktionen.

4.3.2 Strikte Obere Schranke

Durch eine weitere Darstellung kann man sich die strikten oberen Schranken anzeigen lassen. Wir können dadurch erkennen, ob eine Beziehung zwischen zwei Graphparametern vollständig ist und ob der beschränkte Graphparameter tatsächlich

kleiner ist. Aktiviert man diese Darstellungsmöglichkeit, dann wird in dem Original-Graphen folgende Eigenschaft überprüft: Wenn für zwei Graphparameter α, β gilt, dass $(\alpha, \beta) \in S \wedge (\beta, \alpha) \in U$, dann gilt per Definition $\beta \prec_{\mathbb{G}} \alpha$. Für alle Graphparameter, die sich strikt beschränken, bleibt die Kantendarstellung unverändert. Wenn $(\alpha, \beta) \in S \wedge (\beta, \alpha) \notin U$ gilt, dann wird die repräsentierende Kante im Hierarchiegraph gestrichelt gezeichnet. Ein Beispiel dazu ist in Abbildung 4.3 zu sehen.

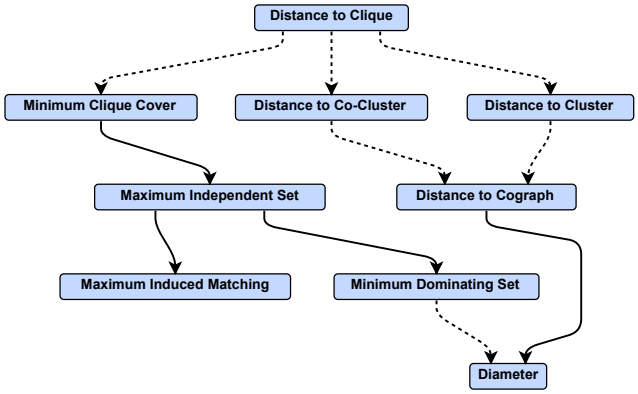


Abbildung 4.3: Darstellung mit strikten oberen Schranken.

4.4 Graphfilter

4.4.1 Transitiver Kantenfilter

Im Folgenden wird nun der transitive Kantenfilter eingeführt, welcher für die Standarddarstellung verwendet wird. Im Abschnitt 4.1 wurde dieser schon kurz erklärt. $P_a H_i D_e$ bietet zwei verschiedene Kantenfilter. Der eine Filter blendet alle Kanten in dem Hierarchiegraph aus, für die es auch einen alternativen Pfad gibt. Der Andere blendet nur die Kanten aus, die für den Benutzer keine neuen Informationen beinhalten. Welche Kanten nicht gefiltert werden, obwohl sie transitive Kanten sind, hängt von der „Stufe“ des Pfades ab, durch welchen die Transitivität gilt.

Der „kleinste“ Pfad der Stufe 1 ist ein Pfad, in dem die Graphparameter nur durch lineare Funktionen f beschränkt sind. Die zweite Stufe ist ein Pfad, in dem mindestens eine Schranke polynomiell ist und beliebig viele weitere linear oder polynomiell sind. Die dritte Stufe beinhaltet genau eine exponentielle Schranke, während die anderen enthaltenen Kanten lineare oder polynomielle Schranken repräsentieren. Und die letzte Stufe wäre ein Pfad, der mindestens zwei exponentielle Schranken enthält. Existiert eine Schranke in einem Pfad, bei der die Funktion noch unbekannt ist, so wird dieser Pfad auch der Stufe 4 zugeordnet.

Konkret werden also Kanten von dem Filter ausgenommen:

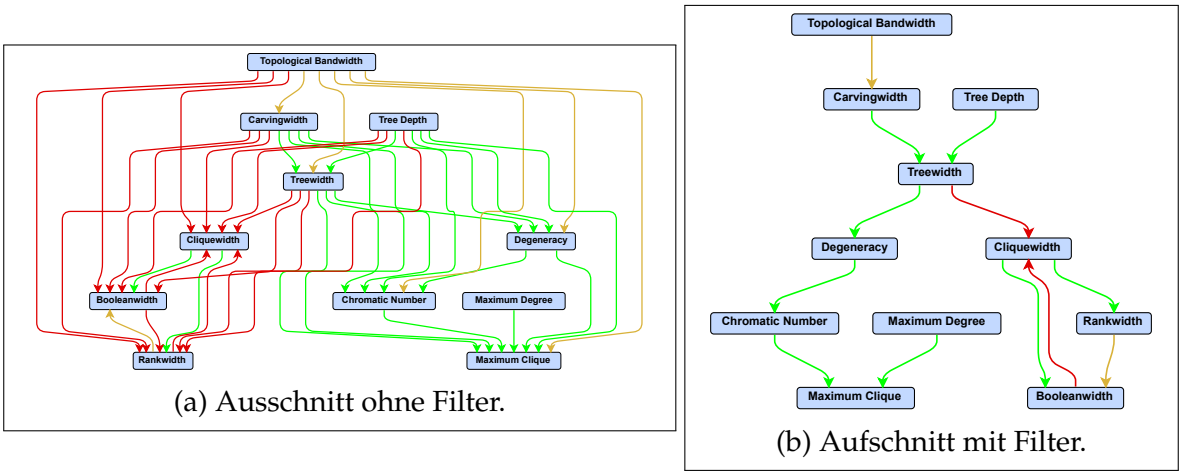
- Wenn die direkte Schranke **linear** (Stufe 1) ist und der Pfad eine oder mehrere **polynomielle/exponentielle oder unbekannte** Schranken enthält (Stufe 2,3,4).
- Wenn die direkte Schranke **polynomiell** (Stufe 2) ist und der Pfad mindestens eine **exponentielle oder unbekannte** Schranke besitzt (Stufe 3,4).
- Wenn die direkte Schranke **exponentiell** (Stufe 3) ist und der Pfad mindestens **zwei exponentielle oder eine unbekannte** Schranke/n besitzt (Stufe 4).

Diese Unterteilung ist aus folgenden Gründen sinnvoll. Sei $\alpha(G) \leq f(\beta(G))$, für alle $G \in \mathbb{G}$, die obere Schranke, die durch einen Pfad P in dem Original-Graphen $G_o = (V_o, E_o = \{S_o \cup U_o\})$ gilt und $\alpha(G) \leq g(\beta(G))$ die obere Schranke, die durch eine direkte Kante $(\beta, \alpha) \in E_o$ gilt. Sind f und g beide lineare Funktionen, dann kann α in beiden Fällen linear größer werden als β und es lägen durch g keine neuen Informationen vor. Steigt f allerdings polynomiell, exponentiell oder stärker, dann ist durch g erkennbar, dass α nur linear größer wird als β. Somit stellt g eine bessere und präzisere Schranke dar und wird daher nicht ausgeblendet. Gleiches gilt auch, wenn f exponentiell oder stärker und g polynomiell wächst bzw. f über-exponentiell oder unbekannt und g exponentiell ist. In jedem dieser Fälle erhielte man durch g eine genauere bzw. kleinere Schranke und g würde daher in dem modifizierbaren Graphen nicht ausgeblendet werden.

Realisiert wird der Filter dadurch, dass intern jeder Kante in S_o ein Gewicht zugewiesen wird, dass abhängig vom asymptotischen Wachstum ihrer Funktion ist. In G_o haben Kanten von linearen Schranken das Gewicht 1, von polynomiellen Schranken das Gewicht n mit $n = |G_o|$ und Kanten von exponentiellen Schranken das Gewicht n^2. Kanten mit unbekannten Funktionswachstum erhalten das Gewicht n^3. Der längste Pfad in einem Graphen kann maximal $(n-1)$ Kanten beinhalten. Das bedeutet, dass der längste lineare Pfad von G_o immer ein kleineres Gewicht als eine Kante mit polynomiellen Gewicht hat. Genauso verhält es sich mit polynomiellen Pfaden im Vergleich zu einem exponentiellen. Das Gewicht des größtmöglichen polynomiellen Pfades in G_o ist $n \cdot (n-1)$, welches kleiner als n^2 ist. Exponentielle Pfade sind somit auch immer kleiner als eine Kante mit unbekannter Funktion. Wendet man den *Dijkstra-Algorithmus* an, ist immer eindeutig festzustellen, ob die Funktionen der transitiven Kanten ein größeres Wachstum als der alternative Pfad haben. Denn der *Dijkstra-Algorithmus* findet mit einem Aufwand von $\mathcal{O}(n^2)$ den kürzesten Pfad zwischen zwei Knoten. Der kürzeste Pfad ist nicht der Pfad mit der kleinsten Anzahl von Kanten, sondern der mit dem niedrigsten Gesamtgewicht aller Kanten [16, Kapitel 8.4.3]. Es muss also nur das Gesamtgewicht des Pfades mit dem Gewicht der transitiven Kante verglichen werden, um zu wissen, ob die transitive Kante auszublenden ist oder nicht.

Im Weiteren löscht $P_a H_i D_e$ alle Kanten aus dem modifizierbaren Graphen, welche keine neuen Informationen bieten und zeichnet diesen dann erneut.

Das Ergebnis des hier beschriebenen Verfahrens ist in Abbildung 4.4 zu sehen. Das linke Bild zeigt einen Ausschnitt des modifizierbaren Graphen ohne aktivierten Filter. Das rechte Bild zeigt den Graphen, nachdem der transitive Kantenfilter mit Berücksichtigung von „besseren“ Funktionen aktiviert wurde.



(a) Ausschnitt ohne Filter.

(b) Aufschnitt mit Filter.

Abbildung 4.4: Der transitive Kantenfilter.

4.4.2 Knotenfilter

Ein weiterer Filter implementiert die Möglichkeit, Knoten aus der Darstellung auszublenden bzw. ausgeblendete Knoten wieder einzublenden. Dies dient der Übersichtlichkeit, wenn man nur an ganz bestimmten Graphparametern interessiert ist. Wichtig ist es, die Korrektheit der anderen Funktionen, die das Programm besitzt, beizubehalten. Dies gilt insbesondere für das Funktionswachstum der Kanten, die neu entstehen bzw. ersetzt werden. Wurde ein Knoten $k \in G_m$ zum Ausblenden ausgewählt, müssen zunächst der transitive Kantenfilter und der im folgenden Abschnitt erklärte Verschmelzungsfilter deaktiviert werden. Danach gleicht der modifizierbare Graph $G_m = (V_m, E_m = S_m \cup U_m)$, abgesehen von bereits ausgeblendeten Knoten wieder dem Original-Graphen. Nun kann man k aus G_m und alle $(x,k),(k,y) \in E_m$ mit $x,y \in V_m$ löschen. Alle Kanten $(x,y),(y,x) \in E_m$ müssen nun stattdessen gezeichnet werden. Da diese nun nicht mehr transitiv sind, wird der transitive Kantenfilter diese Kanten nicht mehr ausblenden. In früheren Versionen von des Programms, in der es noch keinen Original-Graphen gab, mussten diese Kanten erst mit ihren Informationen berechnet werden. Dieser stellt also einen effizienten Vorteil dar. Zuletzt werden alle Filter wieder aktiviert, die vor der Modifizierung von G_m deaktiviert wurden.

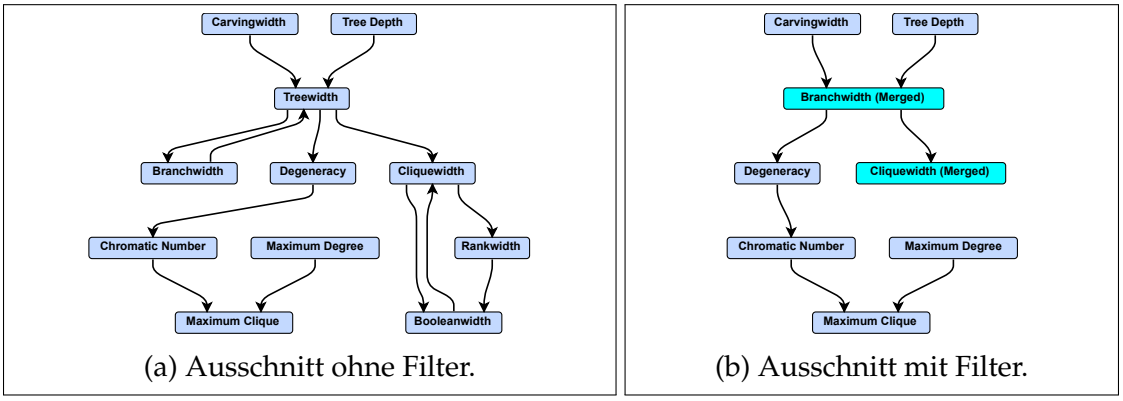
Bei dem Ausblenden von Knoten gehen Informationen über die ausgeblendeten Kanten in G_m verloren. Aus dem Grund ist es unter anderem sinnvoll, den Original-

Graphen $G_o = (V_o, E_o = S_o \cup U_o)$ beim Laden der Datenbank zu erstellen. Wenn ein Knoten k wieder eingeblendet werden soll, werden wieder alle Filter deaktiviert und alle Knoten $x, y \in V_o$ in G_o gesucht, für die $(x, k) \in S_o$ oder $(k, y) \in S_o$ gilt. Danach wird k wieder in G_m eingefügt. Auch hier besitzt G_o einen großen Vorteil, denn bei früheren Implementierungen mussten rekursiv alle Knoten in G_m gesucht werden, die nicht versteckt sind und eine Beziehung zu dem eingefügten Knoten haben. Dies ist mit Hilfe von G_o nicht mehr notwendig, da man durch ihn alle Beziehungen von k kennt. Es können einfach alle Kanten $(x, k), (k, y) \in S_o$ in G_m eingefügt werden. Wenn x oder y auch versteckt sind, dann fügt man diese Kanten erst ein, wenn x oder y wieder eingeblendet werden. Anschließend wird dasselbe Verfahren für alle Kanten, die eine Unbeschränktheit repräsentieren, angewendet. Alle Kanten $(x, y), (y, x) \in E_m$ sind nun wieder transitiv und werden von dem transitiven Kantenfilter, der nach dem Verfahren wieder aktiviert wird, erneut ausgeblendet.

4.4.3 Verschmolzene Knoten

Wie schon im Abschnitt 3.3 erwähnt, gilt in der Graphparameterhierarchie die Antisymmetrie, also $\alpha \preceq_{\mathbb{G}} \beta \wedge \beta \preceq_{\mathbb{G}} \alpha \Rightarrow \alpha \equiv_{\mathbb{G}} \beta$ für zwei Graphparameter α, β. Aus diesem Grund kann $P_a H_i D_e$ Graphparameter, die nach Definition 17 gleich sind, als einen einzigen Graphparameter behandeln. Um sich die Halbordnung der Graphparameter korrekt anzuzeigen bzw. gleiche Graphparameter auszublenden, ist es möglich, diese zu verschmelzen. Um diese Knoten zu identifizieren, wird die Kreiserkennung von JGraphT verwendet. Natürlich werden nur Kreise gesucht, deren Kanten in dem Original-Graphen eine obere Schranke repräsentieren. Ein Kreis von oberen Schranken ist ausreichend, um gleiche Graphparameter zu finden. Seien α, β, γ drei Graphparameter. Wenn in dem Original-Graphen $G_o = (V_o, E_o = S_o \cup U_o)$ gilt, dass $(\alpha, \beta), (\beta, \gamma), (\gamma, \alpha) \in S_o$ sind, dann folgt aus der Transitivität der Halbordnung $(\beta, \alpha), (\gamma, \beta), (\alpha, \gamma) \in S_o$ und es gilt per Definition $\alpha \equiv_{\mathbb{G}} \beta \equiv_{\mathbb{G}} \gamma$. Nachdem ein Kreis gefunden wurde, werden alle im Kreis enthaltenen Knoten aus dem modifizierbaren Graphen gelöscht. Stattdessen wird ein neuer Knoten eingefügt, der eine Liste der gelöschten Knoten beinhaltet. So sind auch in dem neuen Knoten alle Informationen der alten enthalten. Alle in den Kreis führenden Kanten werden nun zwischen ihren Quellen und dem neuen Knoten gezeichnet. Hat ein Knoten zwei Kanten, die in den Kreis führen, wird die Kante mit dem größten Funktionswachstum verwendet.

Wird das Verschmelzen von Knoten wieder rückgängig gemacht, sind zunächst die verschmolzenen Knoten zu löschen. Da alle diese Knoten eine Liste ihrer ursprünglichen Graphparameter beinhalten, können diese wieder eingefügt werden. Um die Kanten wieder herzustellen, wird das Verfahren benutzt, welches versteckte Knoten wiederherstellt. Da die verschmolzenen Knoten letztendlich auch nur versteckt wurden, spart man so eine erneute Implementierung des Vorgangs. Die Darstellung des Filters ist in Abbildung 4.5 zu sehen.



(a) Ausschnitt ohne Filter.

(b) Ausschnitt mit Filter.

Abbildung 4.5: Der Verschmelzungsfilter.

4.5 Schrankenberechnung

Es ist nützlich, die Funktionen der Schranken von einem Graphparameter durch einen anderen zu kennen. Wenn diese Graphparameter in der Hierarchie weit auseinander liegen, ist die obere Schranke oftmals nicht offensichtlich erkennbar, da sie aus Verkettungen von Funktionen resultiert. Hierfür bietet $P_aH_iD_e$ die Möglichkeit, zwei Graphparameter auszuwählen und sich deren genaue Beziehung anzeigen zu lassen.

Auch hier hilft wieder der Original-Graph, welcher beim Start des Programms erstellt wurde. Existiert nämlich eine obere Schranke zwischen zwei Graphparametern, die aus der Verkettung von mehreren oberen Schranken folgt, dann muss der Original-Graph die zugehörige Kante enthalten. Da jede Kante $e = (h, l) \in S$ des Original-Graphen die Funktion der repräsentierten Schranke in dem Format $l \leq f(h)$ gespeichert hat, kann $P_aH_iD_e$ auch für die generierten Kanten $e' \in S$ die obere Schranke berechnen. Seien α, β, γ drei Graphparameter und $G \in \mathbb{G}$. Gilt $\alpha(G) \leq f(\beta(G))$ und $\beta(G) \leq g(\gamma(G))$, dann folgt aus der Transitivität $\alpha(G) \leq f(g(\gamma(G))$. Genau diese obere Schranke wird dann in die generierte Kante (γ, α) in den Original-Graphen eingetragen. Da die generierten Kanten durch den Moore-Bellman-Ford-Algorithmus [16, Kapitel 8.3] berechnet wurden, ist sicher gestellt, dass die verkettete Funktion die „beste“ (mit dem niedrigsten Wachstum) ist.

Existiert also eine solche Kante im Original-Graphen, wird sie in einem Fenster in $\mathcal{O}(1)$ angezeigt. Existiert eine solche nicht, wird dem Benutzer jede andere Art von Beziehung gemeldet.

4.6 Automatisierte Ableitung weiterer Beziehungen

Eine der interessantesten Funktionen, die $P_aH_iD_e$ bietet, ist das Ableiten von neuen Beziehungen aus den bekannten Informationen der Datenbank. Diese werden beim

Laden einer Datenbank in $P_a H_i D_e$ automatisch generiert und ihre beschreibenden Kanten automatisch in den Original-Graphen eingefügt. Welche Beziehungen generiert werden können und wie unter anderem dadurch der Original-Graph entsteht, wird in diesem Abschnitt erläutert.

Zunächst werden alle Graphparameter und ihre Beziehungen aus der Datenbank in das Programm geladen und in einem Graphen G gespeichert. Dieser Graph ist das Grundkonstrukt des Original-Graphen. Danach wird eine Abfolge zur Erweiterung des Graphen gestartet, die solange wiederholt wird, bis keine neuen Beziehungen zwischen den Graphparametern mehr entstehen. Zu der Abfolge gehört zum einen der transitive Abschluss von G, zum anderen zwei Regeln, die weitere Unbeschränktheiten ableiten können. Diese Regeln können jeweils in $\mathcal{O}(n \cdot m^2)$ für einen Graphen berechnet werden, wobei n die Anzahl der Knoten und m die Anzahl der Kanten ist.

Lemma 2 (Regel 1)**.** Seien α, β, γ drei Graphparameter und G ein Graph mit $G \in \mathbb{G}$. Für die Beziehungen zwischen α, β, γ gilt:

$$\alpha \not\succeq_{\mathbb{G}} \beta \wedge \alpha \succeq_{\mathbb{G}} \gamma \Rightarrow \gamma \not\succeq_{\mathbb{G}} \beta$$

Beweis. Es gelte $\exists f : \mathbb{R} \to \mathbb{R} : f(\alpha(G)) \geq \gamma(G)$ und $\nexists g : \mathbb{R} \to \mathbb{R} : g(\alpha(G)) \geq \beta(G)$.
Aussage: $\gamma \succeq_{\mathbb{G}} \beta$
Träfe dieses zu, dann würde nach Transitivität aber auch $\alpha \succeq_{\mathbb{G}} \beta$ gelten. Nach der zweiten Voraussetzung ist dies aber nicht der Fall. Das impliziert $\gamma \not\succeq_{\mathbb{G}} \beta$.

□

Die graphische Darstellung dieser Regel ist in Abbildung 4.6 zu finden. Die gestrichelten Kanten stehen für eine Unbeschränktheit und die durchgezogene Kante beschreibt eine obere Schranke. Die blaue Kante bezeichnet die generierte Unbeschränktheit.

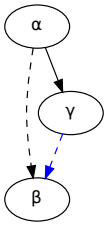


Abbildung 4.6: Regel 1 ($\alpha \succeq_{\mathbb{G}} \gamma \wedge \alpha \not\succeq_{\mathbb{G}} \beta \Rightarrow \gamma \not\succeq_{\mathbb{G}} \beta$).

Lemma 3 (Regel 2)**.** Seien α, β, γ drei Graphparameter und $G \in \mathbb{G}$ ein Graph. Für die Beziehungen zwischen α, β, γ gilt:

$$\alpha \succeq_G \beta \wedge \gamma \not\succeq_G \beta \Rightarrow \gamma \not\succeq_G \alpha$$

Beweis. Es gälte $\exists f : \mathbb{R} \to \mathbb{R} : f(\alpha(G)) \geq \beta(G)$ und $\nexists g : \mathbb{R} \to \mathbb{R} : g(\gamma(G)) \geq \beta(G)$.
Aussage: Es gilt $\gamma \succeq_G \alpha$
Dann müsste nach Transitivität folgen $\gamma \succeq_G \beta$. Dies ist aber wieder ein Widerspruch zu der zweiten Voraussetzung. Folglich ist das ein Widerspruch zur Aussage. Daraus folgt $\gamma \not\succeq_G \alpha$.

□

Die graphische Darstellung dieser Regel ist in Abbildung 4.7 zu finden. Auch hier stehen die gestrichelten Kanten für eine Unbeschränktheit und die durchgezogene Kante beschreibt eine obere Schranke. Die blaue Kante bezeichnet auch hier die generierte Unbeschränktheit.

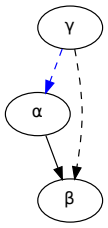


Abbildung 4.7: Regel 2 ($\alpha \succeq_G \beta \wedge \gamma \not\succeq_G \beta \Rightarrow \gamma \not\succeq_G \alpha$).

Der transitive Abschluss wird mit Hilfe des Moore-Bellman-Ford Algorithmus berechnet. Dieser Algorithmus berechnet in dem Original-Graphen $G = (V, E)$ alle kürzesten Wege von einem Knoten zu allen anderen in einem Durchgang mit einem Aufwand von $\mathcal{O}(n \cdot m)$. Der Wert n ist hier $|V|$ und $m = |E|$. Aus den kürzesten Pfaden entnimmt man alle Informationen und kann eine transitive Kante mit den Informationen in den Original-Graphen einfügen. Da alle möglichen Pfade zu berechnen sind, muss dieser Algorithmus n-mal durchgeführt werden und kommt auf eine Gesamtlaufzeit von $n \cdot \mathcal{O}(n \cdot m) = \mathcal{O}(n^2 \cdot m)$. Für einen transitiven Abschluss ist er effizienter als der Dijkstra-Algorithmus, da dieser für einen kürzesten Pfad zwischen zwei Knoten eine Laufzeit von $\mathcal{O}(n^2)$ besitzt und dieser dann $(n \cdot (n-1))$-mal aufgerufen werden müsste. So kommt der Dijkstra-Algorithmus auf eine Gesamtlaufzeit von $\mathcal{O}(n^4)$ für den transitiven Abschluss. Der Moore-Bellman-Ford Algorithmus ist auch in der Graphbibliothek JGraphT implementiert. JGraphT stellt außerdem noch den Floyd-Warshall Algorithmus [16, Kapitel 8.9] bereit, der den transitiven Abschluss

sogar in $\mathcal{O}(n^3)$ berechnen kann. Da in dem Original-Graphen $n < m$ gilt, sollte dieser Algorithmus noch schneller sein als der Moore-Bellman-Ford. Mit den Implementierungen von JGraphT und der Verwendung in $P_aH_iD_e$ war der Moore-Bellman-Ford allerdings trotzdem - vor allem bei schwächeren Systemen - merklich langsamer als der Floyd-Warshall.

Um alle möglichen Informationen aus der Datenbank zu generieren, wird folgender Algorithmus durchgeführt:

Data: Graph G der Datenbank
Result: Original-Graph
alteGröße = $anzahlKanten(G)$;
neueGröße = $anzahlKanten(G) + 1$;
$G = TransitiverAbschluss(G)$;
while *alteGröße* $\neq$ *neueGröße* **do**
    $G = Regel1(G)$;
    $G = Regel2(G)$;
    alteGröße = neueGröße;
    neueGröße = $anzahlKanten(G)$;
end
return G

Es ist sinnvoll zuerst den transitiven Abschluss durchzuführen, da im Gegensatz zu mehreren Durchläufen, mehr Fälle für die Regeln in einem Durchlauf der Schleife betrachtet werden können. Hierdurch wird insbesondere bei großen Datenbanken die Laufzeit merklich reduziert. Durch diese Regeln können einige neue Information bezüglich der Beziehungen der Graphparameter entstehen. Von besonderem Interesse sind z.B. noch nicht bewiesene strikte obere Schranken, die bei Tests von $P_aH_iD_e$ generiert wurden. Aus den Information der Datenbank und dem beschriebenen Algorithmus ist der Original-Graph entstanden. So können im nächsten Schritt die offenen Fragen generiert werden.

4.7 Generierung und Darstellung der offenen Fragen

$P_aH_iD_e$ soll das Finden neuer sinnvoller Parametrisierungen für ein FPT-Problem unterstützen. Zu diesem Zweck wäre eine Graphparameterhierarchie sehr gut geeignet, die nur aus vollständigen Beziehungen besteht und alle existierenden Graphparameter beinhaltet. Von dem Ziel ist man noch weit entfernt, doch $P_aH_iD_e$ kann Hinweise auf fehlende Beziehungen geben. Außerdem liefert die Darstellungsweise Indizien, wie schwer der Beweis zu führen sein könnte oder wie viele neue Informationen durch ihn hinzukämen. Dies wird im Folgenden an dem Graphparameter *Maximum Degree* Δ demonstriert.

Definition 22 (*Maximum Degree* Δ)**.** Sei $G = (V, E)$ ein ungerichteter Graph. Dann ist der *Maximum Degree* $max(\{deg_G(v)|\forall v \in V\})$.

Es gibt zwei Möglichkeiten sich die bekannten Beziehungen anzeigen zu lassen. Die erste ist eine tabellarische Darstellung, welche über die Menüleiste aufgerufen werden kann. Dort kann man in einer Liste den gewünschten Graphparameter auswählen. Die Tabelle zeigt darauf hin alle Informationen zu seinen Beziehungen an. Die Alternative ist durch einen Rechtsklick auf den jeweiligen Graphparameter den Graphen gemäß seiner Beziehungen einzufärben. Eine Legende zeigt, welche Farbe für welche Beziehung steht.

Durch das Einfärben des Graphen bekommt man eine Übersicht über die existierenden Beziehungen. Die farbige Hierarchie der Graphparameter gibt zusätzliche Informationen darüber, welche neuen Beweise die Informationen der Datenbank erweitern. Außerdem lassen sich Vermutungen über die Art von Beziehungen anstellen. Bestehen zum Beispiel keinerlei Beziehungen zwischen zwei Graphparametern α und β, aber α steht weit oben in der Hierarchie und β weit unten, dann ist es wahrscheinlich, dass α den Graphparameter β beschränkt. Es lassen sich aber auch Vermutungen anstellen, welche Beweise leichter und welche schwerer zu erbringen sein könnten. Existiert zum Beispiel eine Unbeschränktheit des Graphparameters γ durch eine Gruppe von anderen Graphparametern, dann ist der Beweis einer oberen Schranke durch γ wahrscheinlich am einfachsten für den kleinsten Graphparameter der Gruppe zu zeigen. Hingegen wird der Beweis einer oberen Schranke für den in der Hierarchie am höchsten stehenden Graphparameter der Gruppe vermutlich schwieriger zu erbringen sein.

In Abbildung 4.8 sind alle bekannten Beziehungen von Δ (weißer Knoten) zu sehen. Dunkelblau bedeutet, dass *Maximum Degree* den Graphparameter *Maximum Clique* strikt beschränkt. Graue Knoten stellen dar, dass keine Beziehung bekannt ist. Wir wissen also momentan sehr wenig über *Maximum Degree* und es gilt mehr Beziehungen zu finden.

Im linken Teil der Abbildung 4.8 ist eine Gruppe von Graphparametern zu sehen, die zwischen *Distance to Clique* ϵ und *Diameter* d (Durchmesser) liegen. *Distance to Clique* beschränkt alle von diesen Graphparametern bis einschließlich *Diameter*. Kann man nun dieselbe Beziehung zwischen Δ und *Distance to Clique* bzw. Δ und *Diameter* zeigen, dann folgt aus der Transitivität oder den Regeln, dass diese Beziehung auch für alle Graphparameter gilt, die zwischen *Distance to Clique* und *Diameter* liegen. Da *Distance to Clique* ganz oben in der Hierarchie steht und Δ ganz weit unten, ist davon auszugehen, dass Δ den Graphparameter *Distance to Clique* nicht beschränkt. Die folgenden Beweise werden die Beziehungen $\Delta \parallel_{\mathbb{G}} \epsilon$ und $\Delta \parallel_{\mathbb{G}} d$ zeigen.

Definition 23 (*Distance to Clique* ϵ)**.** Sei $G = (V, E)$ ein ungerichteter Graph. Der Graphparameter *Distance to Clique* beschreibt die Anzahl an Knoten in V, welche minimal aus G gelöscht werden müssen, damit G eine Clique ist.

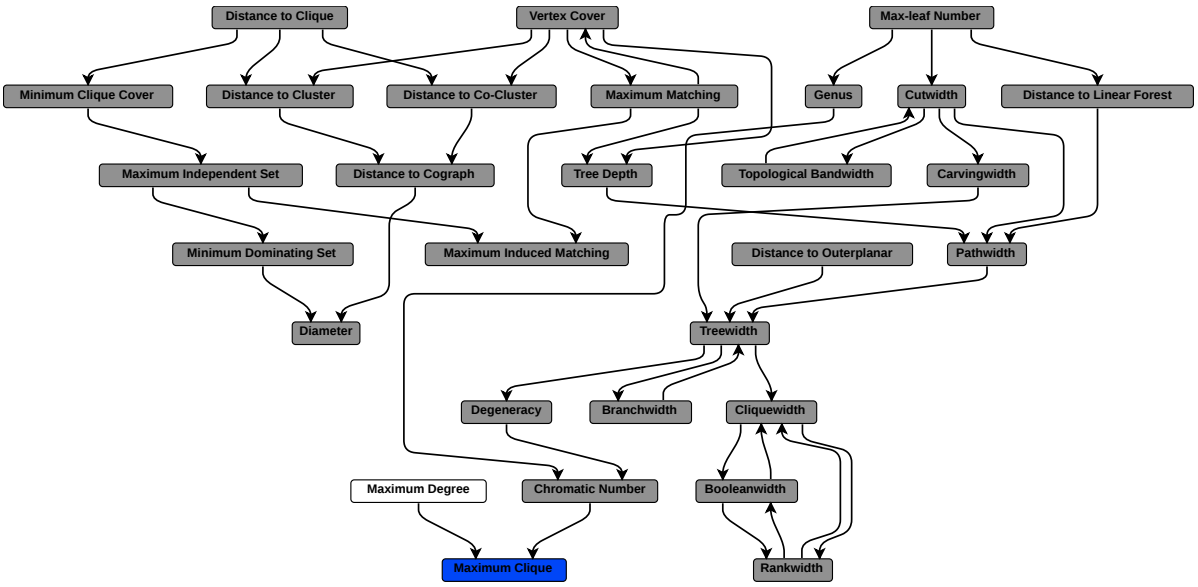


Abbildung 4.8: Offene Beziehungen für *Maximum Degree*.

Lemma 4 ($\Delta \parallel_{\mathbb{G}} \epsilon$)**.** Für die Menge aller ungerichteten Graphen $\mathbb{G}$ sind *Maximum Degree* und *Distance to Clique* unvergleichbar.

Beweis. Schritt 1:
Zu Zeigen: $\Delta \not\preceq_{\mathbb{G}} \epsilon$. Das heißt, $\nexists f : \mathbb{R} \to \mathbb{R}. \forall G \in \mathbb{G}. \Delta(G) \leq f(\epsilon(G))$.
Sei G eine beliebig große Clique. Dann ist $\epsilon(G) = 0$ und $\Delta(G) = n - 1$, wobei $n = |V|$ ist. Für jede Funktion $f(\epsilon(G)) = c$ wählen wir einen Graphen, der eine Clique ist und $|V| = c + 2$.
Schritt 2:
Zu Zeigen: $\epsilon \not\preceq_{\mathbb{G}} \Delta$. Das heißt, $\nexists f : \mathbb{R} \to \mathbb{R}. \forall G \in \mathbb{G}. \epsilon(G) \leq f(\Delta(G))$.
Sei G ein Binärbaum. In einem solchen Graphen ist maximal $\Delta(G) = 3$ für alle Binärbäume. Um allerdings G durch Knotenlöschungen in eine Clique zu transformieren, werden $n - 2$ Knoten gelöscht, was $\epsilon(G) = n - 2$ mit $n = |V|$ bedeutet. Für jede Funktion $f(\Delta(G)) = c$ wird ein Binärbaum mit $|V| = c + 3$ gewählt. □

Lemma 5 ($d \parallel_{\mathbb{G}} \Delta$)**.** Für die Menge aller ungerichteten Graphen $\mathbb{G}$ sind *Maximum Degree* und *Durchmesser* unvergleichbar.

Beweis. Schritt 1:
Zu Zeigen: $\Delta \not\preceq_{\mathbb{G}} d$. Das heißt, $\nexists f : \mathbb{R} \to \mathbb{R}. \forall G \in \mathbb{G}. \Delta(G) \leq f(d(G))$.
Sei G ein Graph mit $V = \{a, x_1, \ldots, x_n\}$ und
$E = \{(a, x_1), \ldots, (a, x_n)\}$. Für diesen Graphen ist $d(G) = 2$, egal für welches $n \in \mathbb{N}$. Jedoch ist $\Delta(G) = n - 1$. Daraus folgt für jede Funktion $f(d(G)) = c$, dass wir G mit $n = c + 2$ wählen und dann $\Delta(G) > d(G)$ gilt.

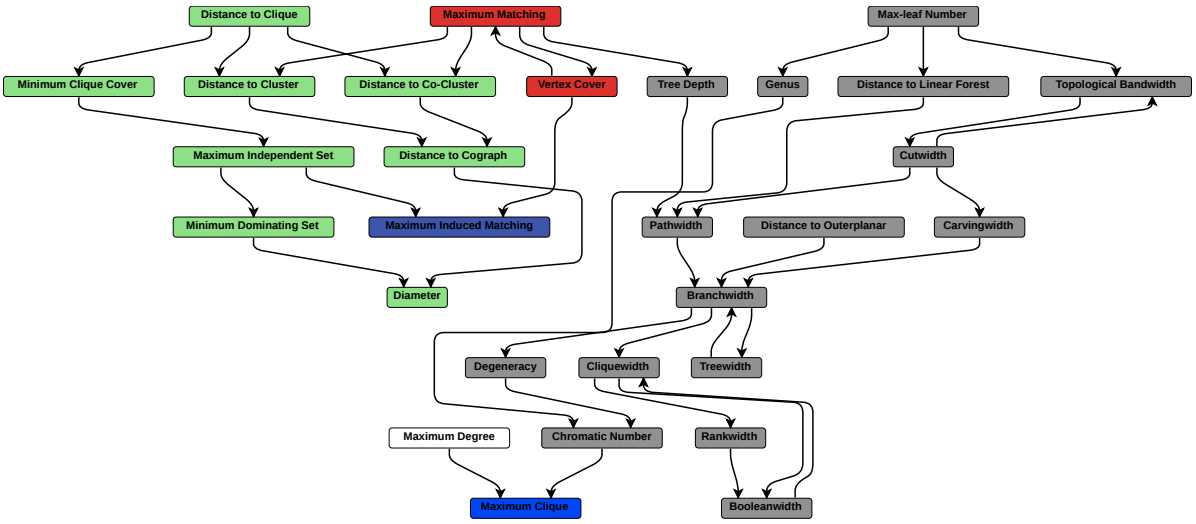


Abbildung 4.9: Offene Beziehungen für *Maximum Degree*.

Schritt 2:
Zu Zeigen: $d \not\preceq_{\mathbb{G}} \Delta$. Das heißt, $\nexists f : \mathbb{R} \to \mathbb{R}. \forall G \in \mathbb{G}. d(G) \leq f(\Delta(G))$.
Sei G ein Graph mit $V = \{a_1, \ldots, a_n\}$ mit $n \in \mathbb{N}$ und
$E = \{(a_1, a_2), (a_2, a_3), \ldots (a_{n-1}, a_n)\}$. In diesem Graphen ist $\Delta(G) = 2$ und
$d(G) = n - 1$. Für jede Funktion $f(\Delta(G)) = c$ wählt man G mit $n = c + 2$. So gilt für
G $d(G) > \Delta(G)$. □

Fügt man diese Beziehungen im Programm hinzu, werden die Beziehungen zwischen *Maximum Degree* und den besagten Graphparametern abgeleitet . Diese Beziehungen sind in Abbildung 4.9 zu sehen.

Die grünen Knoten stehen für eine Unvergleichbarkeit. Außerdem erkennt man, dass *Maximum Degree* die rot gezeichneten Graphparameter nicht beschränkt und das *Maximum Induced Matching* den Graphparameter *Maximum Degree* nicht beschränkt.

Wenn man Beschränkungen durch Δ finden will, ist es am leichtesten, dies für weit unten in der Hierarchie stehende Graphparameter zu zeigen. Ein Beispiel ist der Graphparameter *Degeneracy*.

Definition 24 (*Degeneracy* δ)**.** Sei $G = (V, E)$ ein ungerichteter Graph. Dann ist *Degeneracy* d der größte von allen kleinsten Knotengraden in jedem Teilgraph G' von G.

Aus den Definitionen folgt sofort, dass δ durch Δ beschränkt ist. Es gilt $\delta(G) \leq \Delta(G)$ für alle Graphen $G \in \mathbb{G}$, da jeder Knotengrad in jedem Teilgraphen von G auch nur maximal Δ groß ist.

Außerdem gilt für den Graphparameter *Carving-Width* cw und Δ die Beziehung $\Delta \preceq_{\mathbb{G}} cw$ [1, Observation 1]. Fügen wir auch diese beiden Beziehungen in die Datenbank ein, so erhalten wir das Bild in Abbildung 4.10.

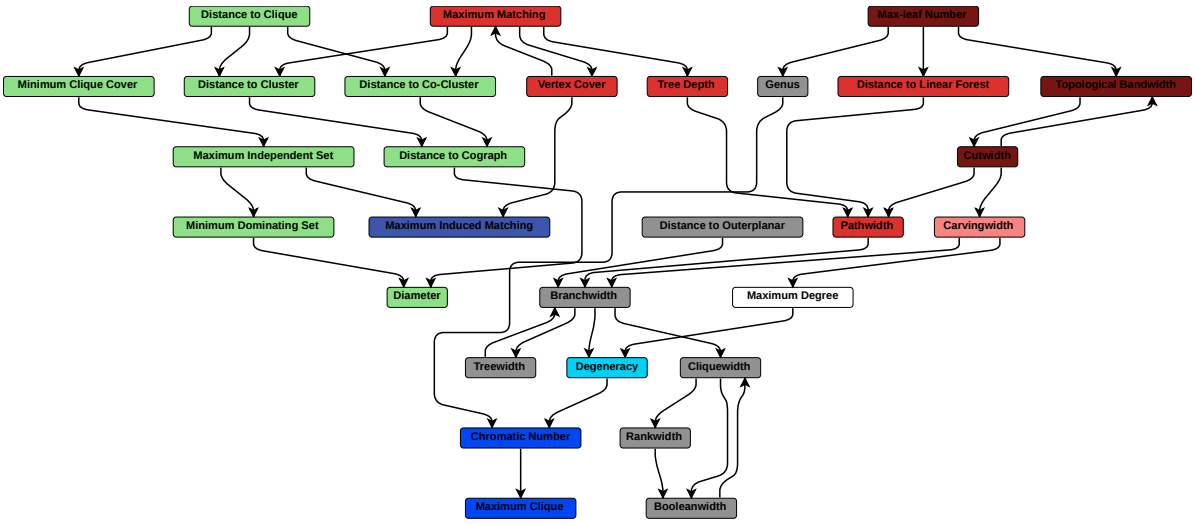


Abbildung 4.10: Offene Beziehungen für *Maximum Degree*.

Durch diese oberen Schranken konnten einige neue Beziehungen abgeleitet werden. Die dunkelroten Knoten bedeuten, dass sie Δ strikt beschränken. Hier sind also die Beziehungen vollständig. Der hellrote Knoten von *cw* zeigt die neu eingetragene Schranke $\Delta \preceq_G cw$ an. Die fünf roten Knoten α stehen für Beziehungen der Art $\Delta \not\succeq_G \alpha$. Nun kann man die Beweise $\Delta \preceq_G \alpha$ und $\Delta \not\preceq_G \alpha$ in Betracht ziehen, um die Beziehungen zu vervollständigen. Da die Knoten α teilweise wesentlich höher in der Hierarchie stehen, ist ein erfolgreicher Beweis der Art $\Delta \preceq_G \alpha$ wahrscheinlicher.

Zusammengefasst wurden für *Maximum Degree* durch die sechs neuen Beziehungen noch 26 weitere Beziehungen zu 20 weiteren Graphparametern abgeleitet, wobei einige dieser Beziehungen noch nicht vollständig sind. Außerdem konnten auch neue Beziehungen zwischen anderen Graphparametern abgeleitet werden, sodass man 74 abgeleitete Beziehungen erhält. Hierbei werden strikte oberen Schranken, Gleichheiten und Unvergleichbarkeiten jeweils als zwei Beziehungen gezählt.

Um das Einfärben des Graphen zu realisieren, verwendet $P_a H_i D_e$ den Original-Graphen. Sobald ein Graphparameter ausgewählt wurde, werden über jeden anderen Graphparameter in dem Original-Graphen iteriert und sein Knoten wird gemäß der bisher bekannten Beziehungen zu dem ausgewählten Graphparameter eingefärbt. Werden neue Kanten in die Hierarchie eingefügt, wird die Datenbank mit den neuen Beziehungen als Graph neu aufgebaut. Im Folgenden wird wieder der transitive Abschluss gebildet und Kanten hinzugefügt, die durch Regel 1 und 2 gelten. Im Anschluss kann man erneut den Graphen einfärben und alle existierenden Beziehungen von Δ sehen.

4.8 Exporte

Die Daten, welche durch $P_aH_iD_e$ visualisiert, verarbeitet und verändert wurden, lassen sich in zweierlei Arten aus dem Programm exportieren.

Die erste Möglichkeit besteht darin, den aktuell visualisierten Graphen der Graphparameterhierarchie als SVG-Datei (Scalable Vector Graphic) zu exportieren. Dieses Dateiformat ist XML-basiert und ein Bild der Parameterhierarchie, welches skalierbar ist oder auch in andere gängige Dateiformate umgewandelt werden kann. Hierfür stellt JGraphX die passenden Java-Klassen bereit.

Die andere Möglichkeit ist der Export aller Informationen der Datenbank als das LaTeX-Dokument „The Graph Parameter Hierarchy“. Diese Option nimmt dem Benutzer die Arbeit ab, alle eingetragenen und bearbeiteten Informationen über Graphparameter und Beziehungen ein zweites Mal in ein Dokument zu schreiben. Aus diesem Grund ist es wichtig, sich an die LaTeX-Syntax zu halten, wenn die Datenbank verändert wird. Denn es wird nicht geprüft, ob die LaTeX-Syntax korrekt ist, sondern es wird lediglich die Datei erstellt.

Das Dokument ist in drei Teile unterteilt. Der erste Teil beinhaltet die Graphparameter und ihre Definitionen. Der zweite Teil alle oberen Schranken. Ist zwischen zwei Graphparametern eine strikte obere Schranke, wird der Beweis der Unbeschränktheit direkt dazu geschrieben. Der dritte Teil beinhaltet alle Unbeschränktheiten, welche sonst noch in der Datenbank stehen und nicht zu einer strikten oberen Schranke gehören. Zu bemerken ist, dass $P_aH_iD_e$ exakt die Standarddarstellung der Datenbank exportiert. Das bedeutet, dass alle oberen Schranken, die der transitive Kantenfilter ausblendet, nicht exportiert werden. Unbeschränktheiten werden nur exportiert, wenn sie nicht generiert sind oder aber generiert und zu einer strikten oberen Schranke gehören.

Sind in der Datenbank Graphparameter oder Beweise zitiert worden, kann der Benutzer eine Bibtex-Datei mit dem Namen „*para_bibliography.bib*“ in dem Wurzelverzeichnis der Datenbank ablegen. $P_aH_iD_e$ sucht automatisch nach dieser Datei und integriert sie, falls vorhanden, in das LaTeX-Dokument.

4.9 Datenbank

Abschließend soll die Datenbank, welche das Kernstück von $P_aH_iD_e$ ist, erklärt werden. Diese bildet die Grundlage für $P_aH_iD_e$. Die Datenbank beinhaltet drei Ordner mit XML-Dateien.

Der erste Ordner *Parameters* enthält alle XML-Dateien der Graphparameter. Eine Parameter-XML enthält die Elemente *ID*, *Name*, *Definition* und *Referenz*. Jeder Graphparameter der Datenbank hat also eine eindeutige ID, wodurch bei der Verarbeitung in dem Programm jeder Graphparameter eindeutig ansteuerbar ist. Der Inhalt der anderen drei Felder ist optional, jedoch ist es immer sinnvoll, alle Informationen

einzutragen, da nur so alle Funktionen bereitgestellt werden können. Werden Graphparameter und ihre Definitionen von fremden Werken entnommen, ist es zusätzlich möglich diese im Referenzfeld zu zitieren.

Der zweite Ordner *Boundedness* beinhaltet die XML-Dateien, welche die Informationen über die oberen Schranken enthalten. Diese Dateien bestehen aus den Elementen *UpperBound*, *Functiontype*, *Proof*, *SourceNode*, *TargetNode*. Die UpperBound enthält die Funktion f, die die obere Schranke von zwei Graphparametern α, β mit $\alpha(G) \leq f(\beta(G))$ beschreibt. Die Funktionen sind immer in dem Format „**l \leq f(h)**“ einzutragen, da sonst verkettete Funktionen nicht korrekt berechnet werden. Der Funktionstyp von f ist in dem Element Functiontype enthalten. Hier gibt es die Möglichkeiten „linear“, „polynomiell“, „exponentiell“ und „unbekannt“. Diese Informationen dienen der farbigen Darstellung, welche in Kapitel 4.3.1 beschreiben wurde. Der „Proof“ enthält den Beweis der oberen Schranke. Das Element SourceNode ist die ID von β und TargetNode die ID von α.

Der dritte und letzte Ordner „Unboundedness“ beinhaltet die Beziehungen der Unbeschränktheit. Gilt eine Unbeschränktheit zwischen den Graphparametern α, β mit $\alpha \not\preceq_G \beta$, so enthalten die XML-Dateien die Elemente SourceNode (ID von β), TargetNode (ID von α) und Proof, welcher den zugehörigen Beweis beinhaltet.

5 Ausblick

Mit $P_aH_iD_e$ ist eine Hilfestellung und eine Motivation entstanden, die Graphparameterhierarchie weiter zu entwickeln und zu erforschen. Es wurde gezeigt, dass viele Beziehungen durch vergleichsweise wenige neue Informationen generiert werden können. Vor allem die entstehenden Unbeschränktheiten sind interessant, da sie häufig die Beziehungen zwischen zwei Graphparametern vervollständigen und damit die Größe der einzelnen Graphparameter klar wird. Eine interessante Erweiterung von $P_aH_iD_e$ könnte daher das Entwickeln weitere Regeln zum automatisierten Ableiten darstellen. Außerdem könnte man den transitiven Kantenfilter hinsichtlich der Filterregeln erweitern. Zum Beispiel unterscheidet $P_aH_iD_e$ noch nicht zwischen exponentiellen und noch stärker wachsenden Funktionen der oberen Schranken. Gerade wenn die Datenbank wächst, könnte dieser Fall relevant werden.

In der Einleitung wurde das Graphproblem BANDWIDTH kurz erwähnt, welches parametrisiert durch *Vertex Cover* fixed-parameter tractable ist. Parametrisiert durch die *Treewidth* ist es allerdings W-schwer. Durch die farbliche Darstellung der Hierarchie weiß man, durch welche Graphparameter BANDWIDTH in FPT ist bzw. sein könnte und durch welche es von der Komplexität W-schwer ist. Interessant für weitere Parametrisierungen von BANDWIDTH sind nach den in der Einleitung erwähnten Regeln Graphparameter, die kleiner als *Vertex Cover*, aber größer als *Treewidth* sind. Die Datenbank beinhaltet allerdings momentan nur wenige Graphparameter und Beziehungen. Es könnte noch viele weitere geben, die in der Hierarchie zwischen *Treewidth* und *Vertex Cover* liegen. Außerdem sind für einige Graphparameter die Beziehungen zu *Treewidth* noch nicht vollständig, sodass man noch keine sichere Aussage treffen kann. Es ist nicht sicher, ob sie tatsächlich Kandidaten sind für die BANDWIDTH in FPT ist. Man benötigt also so viele Information wie möglich um präzise Aussagen treffen zu können.

Die Hauptmotivationen, um die Datenbank zu erweitern sind neben der allgemeinen Übersicht möglichst gute FPT-Parametrisierungen zu finden. Es wurde durch $P_aH_iD_e$ eine Strukturiertheit der Graphparameterhierarchie und eine Aufwandsreduzierung in Bezug auf die Beweisführung erreicht, welche bei der Komplexität des Themas sinnvoll ist. Es bleibt jedoch den Benutzern vorbehalten, die Datenbank mit möglichst vielen Informationen zu füllen. Erst dann kann $P_aH_iD_e$ dazu beitragen, schnellere FPT-Algorithmen zu entwickeln.

Literatur

[1] Rémy Belmonte, Pim van 't Hof, Marcin Kaminski, Daniël Paulusma und Dimitrios M. Thilikos. „Characterizing graphs of small carving-width“. In: *Discrete Applied Mathematics* 161.13-14 (2013), S. 1888–1893 (siehe S. 34).

[2] Andreas Björklund, Thore Husfeldt und Mikko Koivisto. „Set Partitioning via Inclusion-Exclusion“. In: *SIAM Journal on Computing* 39 (2009), S. 546–563 (siehe S. 7).

[3] Jianer Chen, Iyad A. Kanj und Ge Xia. „Improved upper bounds for vertex cover“. In: *Theoretical Computer Science* 411.40–42 (2010), S. 3736 –3756. ISSN: 0304-3975 (siehe S. 15).

[4] Rodney. G. Downey und Michael R. Fellows. *Parameterized Complexity*. Springer, 1999 (siehe S. 8, 9).

[5] J. Flum und M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006 (siehe S. 8).

[6] Frank Gurski, Irene Rothe, Jörg Rothe und Egon Wanke. *Exakte Algorithmen für schwere Graphenprobleme*. Springer, 2010, S. 105 –143 (siehe S. 6).

[7] Bart M. P. Jansen. „The Power of Data Reduction: Kernels for Fundamental Graph Problems“. Diss. Universität Utrecht, 2013 (siehe S. 6, 9, 10).

[8] Michael Kaufmann und Dorothea Wagner, Hrsg. *Drawing Graphs, Methods and Models*. Bd. 2025. Lecture Notes in Computer Science. Springer, 2001 (siehe S. 20).

[9] Christian Komusiewicz. „Parameterized Algorithmics for Network Analysis: Clustering & Querying“. Diss. Technische Universität Berlin, 2011 (siehe S. 14).

[10] Christian Komusiewicz und Rolf Niedermeier. „New Races in Parameterized Algorithmics“. In: *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12)*. Bd. 7464. LNCS. Springer, 2012, S. 19–30 (siehe S. 4, 5, 8, 9, 15).

[11] JGraph Ltd. Hrsg. von JGraphX. URL: `http://www.jgraph.com/jgraph.html` (besucht am 02. 01. 2014) (siehe S. 20).

[12] Jiri Matousek und Jaroslav Nesetril. „Diskrete Mathematik“. In: Springer-Lehrbuch. Springer Berlin Heidelberg, 2007, S. 47–63 (siehe S. 17).

[13] Barak Naveh. Hrsg. von JGraphT. URL: http://www.jgrapht.org/ (besucht am 02. 01. 2014) (siehe S. 22).

[14] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (siehe S. 8).

[15] Róbert Sasák. „Comparing 17 graph parameters“. Masterarbeit. University of Bergen, 2010 (siehe S. 17).

[16] Volker Turau. *Algorithmische Graphentheorie (3. Aufl.)* Oldenbourg, 2009, S. I–XIII, 1–428 (siehe S. 25, 28, 30).