Untersuchung von Graphreduktionsregeln beim Knotenüberdeckungsproblem

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor (B.Sc.)

Universität Trier
FB IV - Informatikwissenschaften
Lehrstuhl für theoretische Informatik

Gutachter:	Prof. Dr. Henning Fernau
	Prof. Dr. Stefan Näher
Betreuer:	Prof. Dr. Henning Fernau

Vorgelegt am xx.xx.xxxx von:

Benedikt Lüken-Winkels
Bahnhofstraße 32
54292 Trier
s4beluek@uni-trier.de
Matr.-Nr. 1138844

Zusammenfassung

Gegenstand dieser Arbeit ist der Vergleich und die Anwendung von Graphreduktionsregeln für das Knotenüberdeckungsproblem. Die Regeln als solche, das Verhalten verschiedener Regeln in der Praxis und der Effekt von deren kombinierter Anwendung wurden untesucht. Umsetzung und Implementierung der Algorithmen geschah in der Programmiersprache *C++* mithilfe der *C++*-Library LEDA. Es stellte sich heraus, dass das Ergebnis der Reduktion in hohem Maße sowohl von der jeweiligen Kombination, als auch der Reihenfolge, in der die Reduktionsregeln an einem Problemgraphen angewandt werden, abhängt. Zudem wurde eine Konfiguration für die hier implementierte Kronenregel gefunden, welche die Reduktion der Regel in der Praxis deutlich verbessert. Die Abhängigkeit der Regel von dieser Einstellung wurde bereits durch andere Untersuchungen zum Thema entdeckt [1]. Die besten Ergebnisse bei der Reduktion pro Graph am eigens erstellten Testset erreichte die Anwendung in der Reihenfolge Nemhauser-Trotter-Regel - Kronenregel - Grad_1-Regel. Hier zeigten sich einige besondere Graphen, welche sich im Vergleich zu den anderen Graphen aus der gleichen Kategorie, beziehungsweise mit der gleichen Kantenanzahl, untypisch verhalten haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

1.1 Motivation

Das *Knotenüberdeckungsproblem*, oder englisch Vertex Cover, ist ein nachgewiesen NP-vollständiges Problem [2], kann also von einem nichtdeterministischen Algorithmus in Polynomialzeit gelöst werden. Um das Problem zu erklären, wird ein Netz von Haushalten betrachtet, bei dem es die Möglichkeit gibt, in jedem Haushalt einen Stromgenerator zu platzieren, sodass durch alle mit diesem Haus verbundenen Leitungen Strom fließt. Ziel ist, ein stabiles Stromnetz zu schaffen, bei dem jede Leitung an mindestens eine Stromquelle angeschlossen ist. Nun ergibt sich das Problem, dass die Kosten für Stromgeneratoren beträchtlich sind und daher nur maximal k Geräte angeschafft werden können. Es gilt also, aus den n Häusern k oder weniger auszuwählen, sodass jede Leitung von einem der Häuser in der Auswahl versorgt wird. Ein

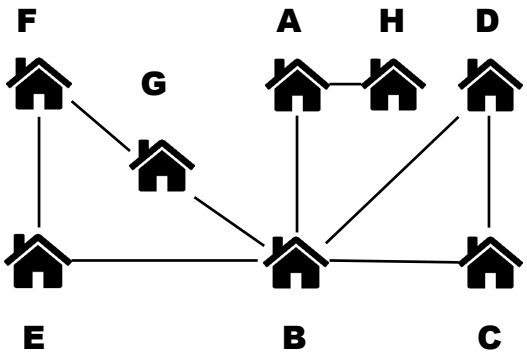

Abbildung 1.1: Graph eines Stromnetzes mit Häusern und Leitungen

Bespiel für eine solche Probleminstanz ist in Abbildung 1.1[1] zu sehen. Die Haushalte sind jeweils mit einem Buchstaben gekennzeichnet; jede Linie, beziehungsweise Kante stellt eine Leitung zum Nachbarhaus dar. Hier ist es möglich, eine Lösung für $k = 4$ zu finden, also 4 Häuser mit Generatoren auszustatten, sodass alle Leitungen versorgt sind. Bei einem kleinen Beispiel wie hier, ist durch ausprobieren leicht zu sehen, dass es keine Lösung mit weniger Häusern, beziehungsweise Generatoren gibt. Um eine Problemlösung mit 4 Häusern zu finden, könnten alle Möglichkeiten ausprobiert werden. Ein solcher naiver Algorithmus würde eine Laufzeit von $O(n^k \cdot m)$ (m

[1] Icons in der Graphik von https://www.flaticon.com/

= Anzahl der Leitungen) brauchen, da für jede k große Kombination aus Häusern die Versorgung der Leitungen überprüft werden muss.
Mit einem einfachen Suchbaumalgorithmus wird jede Leitung nur einmal betrachtet und es muss entschieden werden, welches der beiden Häuser mit einem Generator versorgt werden soll, da mindestens eins der beiden Häuser in der Lösungsmenge enthalten sein muss. Dadurch ergibt sich für jede Entscheidung eine Verzweigung im Suchbaum. Nach k Verzweigungen sind dann k Generatoren verteilt und es muss überprüft werden, ob jede Leitung versorgt ist, was wiederum in m Schritten gemacht werden kann. Es ergibt sich eine Laufzeit von $O(2^k \cdot m)$. Nach der Verwendung einer dieser Algorithmen, finden sich die Lösungen, dass wenn die Haushalte F, B, A und C oder F, B, A und D mit einem Generator ausgestattet werden, durch alle Leitungen Strom fließt. Für eine Problemgröße, wie sie hier geschildert wird, bieten diese Methoden eine Lösung. Für eine Häusermenge $n = 2000$ mit entsprechend vielen Leitungen bedeutet das ungefähr $1.148 \cdot 10^{602}$ mögliche Kombinationen. Es ist leicht zu sehen, dass sich das Stromnetzproblem, werden die Häuser durch Knoten eines Graphen und die Leitungen durch dessen Kanten ersetzt, zum Knotenüberdeckungsproblem transformieren lässt. Zu diesem Problem lassen sich zwei Fragestellungen unterscheiden: Zum einen das Finden einer minimalen Knotenüberdeckung [3]:

$$
\begin{aligned}
&EINGABE: \textit{Graph } G = (V, E),\ \textit{positive Integer } k \leq |V| \\
&AUSGABE: S \subseteq V \textit{ mit } |S| \leq k, \\
&\qquad\qquad \textit{sodass jede Kante aus } E \textit{ einen Endpunkt in } S \textit{ hat.}
\end{aligned}
$$

Zum anderen das Entscheidungsproblem [4], ob es eine Knotenüberdeckung einer Größe $\leq k$ gibt:

$$
\begin{aligned}
&EINGABE: \textit{Graph } G = (V, E),\ \textit{positive Integer } k \leq |V| \\
&AUSGABE: \textit{Existiert ein } S \subseteq V \textit{ mit } |S| \leq k, \\
&\qquad\qquad \textit{sodass jede Kante aus } E \textit{ einen Endpunkt in } S \textit{ hat?}
\end{aligned}
$$

Letzteres ist in der Liste von Karps 21 NP-vollständigen Problemen zu finden [5]. Es existieren weitere reale Probleme, für die das Knotenüberdeckungsproblem anwendbar ist. Abu-Khzam et al [6] schlagen zum Beispiel eine Anwendung in der Bioinformatik vor. Ein nichtdeterministischer Algorithmus kann jede vermeintliche Lösungskombination aus Knoten, beziehungsweise Haushalten in Polynomialzeit testen [2], da lediglich überprüft werden muss, ob die Größe der Lösungsmenge den Wert von k überschreitet und ob alle Kanten abgedeckt (Leitungen versorgt) sind. Allerdings würde ein naiver Algorithmus für große Eingaben in der Realität nicht in absehbarer Zeit terminieren, wie beispielhaft in Tabelle 1.1 zu sehen ist. Es ist

Tabelle 1.1: Exponentielle Laufzeit

Eingabegröße	**Laufzeit**
2^1	0.00002s
2^{10}	0.02s
2^{100}	ca. $2 \cdot 10^{14}$ Jahrtausende

Beispiel für die Laufzeit einen Algorithmus' mit eine Laufzeit von $O(2^n)$

also sinnvoll, die Eingabe möglichst klein zu halten oder auf einen Problemkern zu reduzieren, indem einfache Teile des Graphen durch Vorverarbeitung, englisch *Preprocessing*, entfernt werden und gegebenenfalls in die Lösungsmenge aufgenommen werden. Dies sollte in Polynomialzeit geschehen, um die gesamte Laufzeit zu verringern. Das Ziel der in dieser Arbeit verwendeten Reduktionsregeln ist, für einen gegebenen Graphen $G = (V, E)$, einer natürlichen Zahl k und der Problemstellung $VC(G, k)$ das Finden und Entfernen eines induzierten Teilgraphen $G' \subseteq G$ und Verkleinerung von k, sodass für den Problemkern $G'' = G \setminus G'$ gilt, dass $VC(G, k) = VC(G'', k'') \cup C \; mit \; C = VC(G', k'')$, wobei $C = VC(G', k'')$ in Polynomialzeit durch die Regeln berechnen lässt [7]. Für den Suchbaumalgorithmus (mit $O(2^k \cdot m)$) bedeutet jede Reduzierung von k eine Verdoppelung der Geschwindigkeit.

1.2 Problemstellung

Das Knotenüberdeckungsproblem ist in der Literatur viel vertreten und es wurden verschiedene Möglichkeiten, eine Probleminstanz zu reduzieren, entdeckt. In dieser Arbeit werden einfachere Reduktionsregeln, genauer gesagt die Grad_0-, Grad_1- und Buss-Regel betrachtet, als auch Nemhauser-Trotter-Regel und Kronenregel, welche sich weitaus komplexer gestalten. Jede dieser Regeln wird an verschiedenen Stellen eines Graphen ausgeslöst und es scheint, dass die Reihenfolge, in die Regeln auf einen Graphen angewandt werden, einen Einfluss auf die insgesamt erreichte Reduktion hat. In dieser Arbeit wird untersucht, wie sich die oben genannten Regeln in der Praxis verhalten, sowohl einzeln als auch in Kombination, um dann verglichen zu werden.

1.3 Zielsetzung

Ein Großteil der Behandlung von Reduktionsregeln in der Literatur findet ausschließlich theoretisch statt. Ziel dieser Arbeit ist, herauszufinden, wie sich die Regeln in der Praxis verhalten, wenn sie einzeln oder in Kombination miteinander angewandt werden. Hierzu wurden die Regeln mithilfe der *C++*-Library LEDA [8] implementiert,

welche Algorithmen und eigene Datentypen bereitstellt. Unter anderem liefert LEDA Datenstrukturen und Algorithmen für Graphen, wie zum Beispiel eine Funktion, mit der ein Maximum-Bipartite-Matching in einem Graphen gefunden und ausgegeben werden kann. Durch die Vielzahl an Funktionen eignet sich die Library für diese Arbeit. Jede der Regeln wird dann unter verschiedenen Aspekten untersucht:

- Wie verhält sich die theoretische Laufzeit im Vergleich zur Praxisanwendung?
- Passt die theoretisch erwartete Reduktion zum Ergebnis am Testset?
- Wie gut ist das Ergebnis der Reduktion im Vergleich zu anderen Regeln?
- Welche Kombination von Regeln liefert eine hohe/niedrige Reduktion? Warum?
- Wie sehen Graphen aus, auf die keine Regel anwendbar ist?
- Wie sehen Graphen aus, auf die genau eine Regel anwendbar ist?

1.4 Gliederung der Arbeit

Zunächst werden einige Grundlagen geschaffen, um auf gewisse Begrifflichkeiten in späteren Kapiteln zurückgreifen zu können und die entsprechenden Algorithmen vorzustellen. Kapitel 3 beschäftigt sich mit der Analyse der Praxisanwendung der Regeln und zeigt mögliche Gründe für die gefundenen Ergebnisse. Im Folgenden wird auf Besonderheiten in der Implementierung und Umsetzung der Reduktionsregeln eingegangen, um abschließend ein Fazit mit einen Ausblick auf mögliche zukünftige Problemstellungen zu geben.

2. Grundlagen

Wie zuvor erwähnt, wird unter den in dieser Arbeit verwendeten Reduktionsregeln zwischen einfacheren und komplexeren Regeln unterschieden. Die komplexeren Nemhauser-Trotter- und Kronenregel erstellen während der Anwendung weitere Subgraphen für den Problemgraphen, während Grad_0-, Grad_1- und Buss-Regel lediglich den Grad einzelner Knoten im Graphen betrachten. Darüber hinaus können die Regeln weiter unterschieden werden. All die zuvor genannten Regeln entfernen induzierte Teilgraphen aus der Probleminstanz, an der sie angewandt werden. Die Folding-Regel [9] (oder Grad_2-Regel) zum Beispiel nicht. Hier werden die Nachbarknoten u und v eines Knoten x des Grades zwei verschmolzen, vorausgesetzt, es gibt keine Kante zwischen u und v. Es wird ein neuer Knoten x' mit $N(x') = N(u) \cup N(v) \setminus \{x, u, v\}$ eingesetzt und k um eins verringert. Diese Konstruktion muss dann nach dem Durchlaufen des Algorithmus' wieder entpackt werden, da sonst nach der Reduktion kein äquivalenter Problemgraph entsteht. Ein Algorithmus mit schneller Laufzeit, um zu entscheiden, ob eine Knotenüberdeckung in einem gegebenen Graphen G, Parameter k und einer Knontenmenge n zu finden ist, wird von Chen et al [9] mit $O(kn + 1.271^k k^2)$ vorgeschlagen. Diese verwendet unter anderem die Folding-Regel verwendet; die schnellste bekannte Methode, alle Knotenüberdeckungen eines Graphen zu zählen von Mölle et al [10] mit einer Laufzeit von $O(1.3803^k)$.

2.1 Knotenüberdeckungsproblem

Das Knotenüberdeckungsproblem ist ein gut erforschtes Beispiel für ein Fixed-Parameter-Tracable Problem [11]. Ein solches Problem zeichnet sich dadurch aus, dass es bei einer Problemgröße n und Parametrisiertung k in $O(f(k) \cdot n^c)$ entscheidbar ist, wobei f eine Funktion und c eine von n und k unanhängige Konstante ist [12]. Für das Knotenüberdeckungsproblem bedeutet dies, dass die Parametrisierung über k eine Vergleichbarkeit verschiedener Algorithmen ermöglicht.

2.2 Einfache Reduktionsregeln

Aus einfachen Beobachtungen über den Problemgraphen lassen sich für einen Graphen $G = (V, K)$ und der natürlichen Zahl k einige Reduktionsregeln ableiten.

1. Knoten, die isoliert stehen und keine Kanten haben, können aus dem Graphen entfernt werden. Sie können nicht zu einer minimalen Knotenüberdeckung gehören können, da sie keine Kanten abdecken (Grad_0-Regel).

2. Bei Knoten, die genau eine Kante, beziehungsweise genau einen Nachbarknoten haben, wird der entsprechende Nachbarknoten automatisch in die Lösungsmenge aufgenommen und beide aus dem Graphen entfernt, da einer der beiden Knoten in der Knotenüberdeckung vorkommt. Der Nachbarknoten hat mindestens den Grad eins, deckt also möglicherweise mehr Kanten ab (Grad_1-Regel).

3. Knoten mit $k + 1$ Kanten, werden der Knotenüberdeckung hinzugefügt und aus dem Graphen entfernt, da sonst alle Nachbarknoten aufgenommen werden und somit mehr, als k Knoten in der Lösungsmenge wären (Buss-Regel).

Angewandt an eine Probleminstanz bedeutet das Einsetzen einer der Regeln, dass die entsprechenden Knoten und Kanten von einer weiterführenden Betrachtung ausgeschlossen sind. Buss-Regel und Grad_1-Regel unterscheiden sich von der Grad_0-Regel dahingehend, dass das Auslösen einer dieser Regeln auch den Parameter k verringert, da ein Knoten in die Überdeckung aufgenommen wird. Das Beispiel in Abbildung 1.1 kann durch die wiederholte Anwendung der Regeln sogar gelöst werden:

1. Es stehen 4 Generatoren zur Verfügung, allerdings hat Haus B 5 ausgehende Leitungen ⇒ Buss-Regel: B erhält einen Generator und dessen Leitungen haben Strom.

2. Drei disjunkte Mengen von Häusern ({E, G, F}, {D, C} und {A, H}) bleiben übrig, bei denen jeweils einmal die Grad_1-Regel angewandt werden kann. Die 3 restlichen Generatoren werden verteilt, wobei sich bei jeder der Häusergruppen mehrere Möglichkeiten bieten.

3. Es steht kein Generator mehr zur Verfügung und aus der Restmenge E, G, F bleibt ein Haus ohne stromlose Leitungen zurück ⇒ das Haus wird aus der Betrachtung entfernt und das Problem ist gelöst.

Nach der wiederholten Anwendung der Buss-Regel sind alle Knoten mit mehr als k Kanten in der Überdeckung und von der weiteren Betrachtung ausgeschlossen. Jeder übrige Knoten kann also maximal k Kanten haben. Wenn nun noch mehr als k^2 Kanten nicht indiziert sind, kann es keine Lösung dieser Probleminstanz geben, da lediglich k Knoten in der Lösungsmenge sein dürfen und somit nur maximal k^2 Kanten abdecken. Es bleiben höchstens $2k^2$ Knoten (mit mindestens einer Kante), bei denen noch nicht alle Kanten abgedeckt sind, übrig [13], da jede der Kanten

zwei Knoten hat. Wird nun die Grad_1-Regel angewandt, bis sie nicht mehr greift, sind höchstens $k^2 + k$ Knoten übrig, da jeder dieser Knoten jetzt zwischen zwei und k Kanten hat. Um einen Knoten aus der Knotenmenge V mit mehr als k Kanten zu finden, werden im schlimmsten Fall $k \cdot n$ ($n = |V|$) Schritte benötigt, da jeder Knoten überprüft werden muss. Falls alle Knoten den Grad k haben, wird die Buss-Regel nicht ausgelöst. Ansonsten wird der gefundene Knoten mit all seinen Kanten aus dem Graphen entfernt, was wiederum höchstens d (wobei d der größte Grad unter den Knoten aus G ist und $d \geq k$ gilt) Schritte benötigt. Die Grad_1-Regel benötigt im schlimmsten Fall $2 \cdot n$ Schritte, um den Graphen einmal zu durchlaufen, da die Betrachtung eines Knoten abgebrochen werden kann, sobald er mehr als eine oder keine Kante hat, da sie dann für diese Regel uninteressant ist. Wird eine Kante gefunden, kann dessen Nachbarknoten und damit alle von diesem ausgehenden Kanten entfernt werden, also im schlimmsten Fall d. Wurde die Buss-Regel vor der Grad_1-Regel angewandt, bedeutet das, dass für die Entfernung maximal k Schritte gebraucht werden. Um alle Häuser ohne Leitungen zu entfernen, muss lediglich für jedes Haus überprüft werden, ob es Leitungen hat, also n Schritte.

2.3 Nemhauser-Trotter Reduktionsregeln

Der Nemhauser-Trotter-Regel liegt das Nemhauser-Trotter-Theorem zugrunde [14]: *Für einen Graphen $G = (V, E)$ können zwei disjunkte Mengen C_0 und V_0 gefunden werden, sodass*

1. *C_0 in einer minimalen Knotenüberdeckung von G enthalten ist,*
2. *der Teilgraph $G[V_0]$ eine Knotenüberdeckung der Größe $\leq |V_0|/2$ hat,*
3. *und $VC(G) = VC(G[V_0]) \cup C_0$ gilt.*

Nach dem Satz von König und unter Verwendung des Algorithmus' von Hopcroft und Karp [15] können diese Mengen mithilfe des Erstellens eines auf dem Problemgraphen basierenden bipartiten Graphen in Polynomialzeit gefunden werden. Der im Zuge dieser Arbeit verwendete Algorithmus [16] lautet wie folgt:

Algorithmus 2.1: Nemhauser-Trotter-Regel.

0 $G = (V, E), n := |V|, m := |E|, d := \textit{maximaler Grad der Knoten aus } G$
1 `Bipartiten Graphen erstellen` $B = (V, V', E')$
2 `mit` $E' := \{\{x, y'\}, \{x', y\} | \{x, y\} \in E\}$
3 `Maximum Matching` M `von` B `bestimmen`
4 $C_B \leftarrow VC(B)$
5 $C_0 \leftarrow \{x \in V \mid x \in C_B \textit{ und } x' \in C_B\}$
6 $V_0 \leftarrow \{x \in V \mid \textit{entweder } x \in C_B \textit{ oder } x' \in C_B\}$

Das Erstellen des bipartiten Graphen in den Zeilen 1-2 benötigt $n \cdot d$ Schritte, da der Graph kopiert und jede Kante doppelt eingetragen wird. Zeilen 3-4 brauchen mit dem verdoppelten Graphen durch die Verwendung des Algorithmus' von Hopcroft und Karp $\sqrt{2n} \cdot 2m$ Schritte. Um die Zugehörigkeit der einzelnen Knoten in den Zeilen 5-6 zu bestimmen, muss der Graph jeweils lediglich einmal durchgegangen werden, also $2n$ Schritte. Insgesamt ergibt das:

$$n \cdot d + \sqrt{2n} \cdot 2m + 2n \tag{2.1}$$

$$\Rightarrow O(\sqrt{n} \cdot m) \tag{2.2}$$

Die Laufzeitabschätzung von $O(\sqrt{n} \cdot m)$ entsteht, wenn d als Konstante betrachtet wird. Die Laufzeit des Algorithmus' wird hauptsächlich vom Finden des Matchings dominiert. Der Algorithmus eignet sich also für eine Vorverarbeitung des Graphen in Polynomialzeit. Ein wichtiger Teil des Algorithmus' ist das Finden eines Maximum Matchings. Im Allgemeinen ist ein Matching oder eine Paarung für einen Graphen $G = (V, E)$ eine Menge von Kanten $S \subseteq E$, für die gilt, dass keine zwei Kanten aus S einen gemeinsamen Knoten haben. Ein Maximal Matching M wiederum ist ein Matching, für das gilt, dass M keine weitere Kante aus E hinzugefügt werden kann, ohne die Matchingeigenschaft zu zerstören. Maximum Matchings erfüllen die Eigenschaften eines Maximal Matchings und enthalten außerdem die größte Anzahl an Kanten im Vergleich zu anderen Matchings.
Mit der Nemhauser-Trotter-Regel ist es theoretisch möglich, den Problemgraphen auf einen Problemkern von $2k$ zu reduzieren, allerdings sorgt die Verwendung dieser Regel dafür, dass nicht alle Knotenüberdeckungen in einem Graphen gefunden werden können, sondern lediglich mindestens eine [16]. Wie in Abbildung 2.1 zu

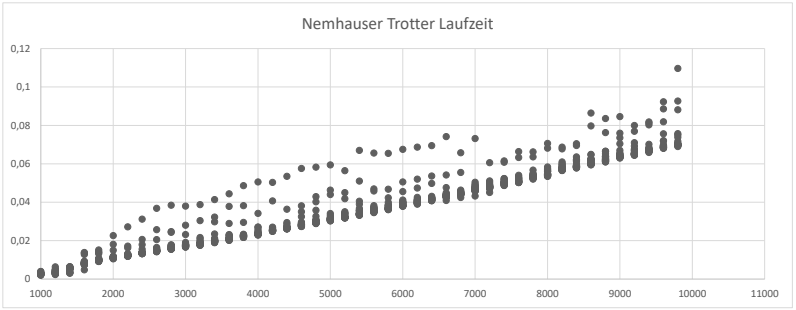

Abbildung 2.1: Laufzeit der Anwendung der Nemhauser-Trotter-Regel am Testset

sehen ist, verhält sich die Laufzeit des implementierten Algorithmus' entsprechend der Abschätzung linear zur Eingabe. Jeder Punkt repräsentiert einen Graphen, für den auf der X-Achse die entsprechende Kantenanzahl und auf der Y-Achse die Zeit, für ein Durchlauf der Nemhauser-Trotter-Regel im Schnitt für diesen Graphen in Sekunden gebraucht hat.

2.4 Kronenregel

Ziel der Kronenregel ist, im Problemgraphen eine Krone zu finden, welche sich folgendermaßen definiert:
Für einen Graphen $G = (V, E)$ besteht eine Krone aus $H \subseteq V$ und $I \subseteq V$ mit $H \cap I = \emptyset$, sodass

1. *$H = N(I)$ gilt,*
2. *I ein Indipendent Set ist, also $\forall v, w \in I$ gilt $(vw) \notin E$ und*
3. *die Kanten zwischen H und I ein Matching enthalten in dem alle Knoten aus H enthalten sind.*

Ist eine solche Krone gefunden, kann davon die Menge H in eine Knotenüberdeckung für G hinzugefügt werden, um den Graphen um H und I zu reduzieren. Dies ist für das Knotenüberdeckungsproblem sinnvoll, da I ein Independent Set ist und damit innerhalb der Menge keine Kanten existieren. Werden also nur die Nachbarknoten, beziehungsweise H in die Überdeckung aufgenommen, ist sichergestellt, dass keine Kanten vernachlässigt sind. Für die Untersuchung wurde der folgende Algorithmus [17] implementiert:

Algorithmus 2.2: Kronenregel.

```
0  G = (V, E), n := |V|, m := |E|, d := maximaler Grad der Knoten aus G
1  //M_1 = Maximal Matching von G
2  M_1 ← ∅
3  foreach e ∈ E do
4     M_1 ← M_1 ∪ e
5     Entferne e und N(e) aus der weiteren Betrachtung
6  od
7  O ← nicht gepaarte Knoten in M_1
8  M_2 ← Maximum Matching von B = (O, N(O), {uv|u ∈ O ∧ v ∈ N(O)})
9  I ← nicht gepaarte Knoten aus O in M_2
10 I' ← ∅
11 while I' ≠ I do
12    I' ← I
13    H ← N(I)
14    I ← I ∪ {∀u ∈ O|∃v ∈ H (uv ∈ M_2)}
15 od
16 return I
```

Um das Maximal Matching M_1 zu erstellen, werden im schlimmsten Fall $m \cdot d$ Schritte benötigt, da bis zu d Nachbarkanten deaktiviert werden müssen. Ist $d = 1$, so wird jede Kante aus E betrachtet, beziehungsweise in M_1 aufgenommen. Gilt $|M_1| > k$, so kann für den Graphen keine Knotenüberdeckung $\leq k$ gefunden werden. Das

Finden der in M_1 nicht gepaarten Knoten aus V kostet n Schritte. B kann in etwas weniger als $\frac{nd}{2}$ erstellt werden, da n bis zu $2k$ kleiner ist, allerdings fällt dies für die Worst Case Abschätzung nicht sehr schwer ins Gewicht. Um das Matching M_2 in diesem bipartiten Graphen zu finden, werden wie bei der Nemhauser-Trotter-Regel $\sqrt{n} \cdot m$ Schritte benötigt. Ist das Matching $M_2 > k$, so kann für den Graphen keine Knotenüberdeckung $\leq k$ gefunden werden. Die Zeilen 8 bis 13 kosten maximal $n \cdot d$ Schritte, da die Knoten aus O und deren Nachbarknoten überprüft werden müssen. Insgesamt ergibt das eine Abschätzung von:

$$m \cdot d + n + \frac{nd}{2} + \sqrt{n} \cdot m + n + n \cdot d \tag{2.3}$$

$$= d(m+n) + 2n + \frac{nd}{2} + \sqrt{n} \cdot m \tag{2.4}$$

$$\Rightarrow O(\sqrt{n} \cdot m) \tag{2.5}$$

Für die Laufzeitabschätzung ergibt sich $O(\sqrt{n} \cdot m)$, wird d als Konstante betrachtet. Der Algorithmus eignet sich also für eine Vorverarbeitung des Graphen in Polynomialzeit. Für das Matching M_1 wird ein Maximal Matching verwendet. Dies hat vermutlich den Grund, dass Maximal Matching-Algorithmen vermeintlich eine bessere Laufzeit, als die Algorithmen haben, die ein Maximum Matching in einem allgemeinen Graphen suchen. Allerdings gibt es nach Blum [18] einen Algorithmus, der ein Maxmimum Matching in einem allgemeinen Graphen in $O(\sqrt{n} \cdot m)$ findet, was die Gesamtlaufzeit der Kronenregel zwar um den Faktor zwei verschlechtert, allerdings eventuell das Ergebnis der Reduktion verbessern würde. Abu-Khzam et al. [1] vermutet, dass das Matching M_1 einen großen Einfluss auf das Ergebnis der Reduktion hat, geht aber in seinem Artikel nicht weiter darauf ein. Insgesamt kann mit der Kronenregel ein Problemkern mit einer Knotenmenge von bis zu $3k$ erreicht werden. Wie in Abbildung 2.2 zu sehen ist, verhält sich die Laufzeit des implementierten

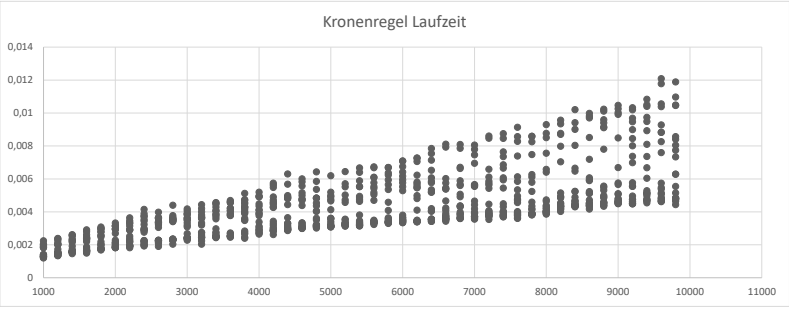

Abbildung 2.2: Laufzeit der Anwendung der Kronenregel am Testset

Algorithmus' entsprechend der Abschätzung linear zur Eingabe. Wie Abbildung 2.1 repräsentiert jeder Punkt einen Graphen, für den auf der X-Achse die entsprechende Kantenanzahl und auf der Y-Achse die Zeit, für ein Durchlauf der Kronenregel im Schnitt für diesen Graphen in Sekunden gebraucht hat.

3. Analyse

3.1 Anforderungen

Um einen Graphen, der zur Diskussion steht, auf einen Problemkern zu reduzieren, können also, wie zuvor gezeigt, verschiedene Reduktionsregeln verwendet werden. Zunächst werden die Regeln anhand der folgenden Kriterien verglichen:

- Laufzeit
- Erwartete Reduktion

Dann wird die Anwendung und der Effekt der Regeln auf das eigens erstellte Testset untersucht, um folgende Fragen zu beantworten:

- Wie effektiv sind die Reduktionsregeln in der Anwendung?
  - Wie oft sind die Regeln anwendbar?
  - Wie viel wird reduziert?
- Wie (gut) funktionieren die Reduktionsregeln in Kombination?
- Wie sehen Graphen aus, auf die keine Regel anwendbar ist?
- Wie sehen Graphen aus, auf die genau eine Regel anwendbar ist?

3.2 Bewertung der Reduktionsregeln

Für jede Regel wird vorausgesetzt, dass zusätzlich zur Regel selbst alle Kanten des Grades 0 aus dem Graphen entfernt werden. Ist dies nicht der Fall, werden zum Beispiel nach der Anwendung der Buss-Regel beliebig viele Knoten mit Grad 0 übrig bleiben und die Knotenmenge der reduzierten Instanz kann nicht durch k abgeschätzt werden. In Tabelle 3.1 stehen die Laufzeiten in der O-Notation und die erwartete obere Grenze der Knotenanzahl des reduzierten Problemkerns der Nemhauser-Trotter-Regel, der Kronenregel und die der Buss-Regel gegenüber. Zwar zeigte sich

bei der Analyse der Laufzeit der Algorithmen, dass die Laufzeit der Nemhauser-Trotter-Regel um einige ganzzahlige Faktoren länger ist als die der Kronenregel, allerdings hängt diese bei beiden überwiegend vom Finden von Matchings ab. Im folgenden Abschnitt wird betrachtet, wie sich die Reduktionsregeln in der Praxis verhalten. Zu erwarten ist, dass die Nemhauser-Trotter-Regel mehr Knoten aus den Graphen entfernt als die Kronenregel, wobei sich in den eher lichten Graphen des Testsets womöglich viele einfache Kronen finden lassen.

Tabelle 3.1: Bewertung der Reduktionsregeln

Reduktionsregel	**Laufzeit**	**Reduktion**
Nemhauser-Trotter	$O(\sqrt{n} \cdot m)$	$\leq 2k$
Kronenregel	$O(\sqrt{n} \cdot m)$	$\leq 3k$
Buss	$O(k \cdot n)$	$\leq k^2$

Für einen Problemgraphen $G = (V, E)$ ist k die obere Schranke für die Größe einer Knotenüberdeckung, $n = |V|$ und $m = |E|$
Reduktion bezieht sich hier auf den übrig bleibenden Problemkern nach der Reduktion

3.3 Methodik

Das Testset, an dem die Algorithmen angewandt wurden, besteht aus ungerichteten Graphen, die jeweils eine Knotenmenge von 1000 Knoten und eine Kantenanzahl von 1000 bis 10000 (aufsteigend in 200er Schritten) umfassen. Von jeder *Graphklasse*, die sich durch die Kantenmenge auszeichnet, gibt es 20 Exemplare. Dadurch entsteht ein Testset von 900 Graphen. Zum Erstellen der Graphen wurde die LEDA-Funktion random_simple_undirected_graph(|V|,|E|) [8] verwendet. Da Zufallsgraphen verwendet werden, gibt es keine Vorgabe für den Parameter k. Die obere Beschränkung der Kantenmenge von 10000 hat sich bei den Tests ergeben, da ab einer bestimmten Dichte, beziehungsweise Knotenanzahl keine Reduktionsregel mehr erfolgreich ist, beziehungsweise keine Änderung am eingegebenen Graphen mehr erzielte. Dies kann mit der erwarteten Reduktion (siehe Tabelle 3.1) erklärt werden: Je dichter der Graph wird, desto größer wird k und sobald $k > \frac{|V|}{2}$ wird, ist bei der Nemhauser-Trotter-Regel und somit auch bei den anderen die theoretische Größe des reduzierten Problemkerns größer oder gleich der Menge der Knoten, was bedeutet, dass theoretisch keine Reduktion stattfinden kann. Es eignen sich viele Benchmarktests [1] [2] nicht, da sich diese durch sehr schwere Probleme mit dichten Graphen und Werten für k, die sich der Größe der Knotenmenge im jeweiligen Problem annähern, auszeichnen. Eine Knotenmenge von 1000 Knoten pro Graph für das Testset erwies sich

[1] http://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/
[2] http://sites.nlsde.buaa.edu.cn/ kexu/benchmarks/graph-benchmarks.htm

bei der Anwendung als ausreichend hoher Wert, um das Einsetzen der Reduktionsregeln zu beobachten.
Die Reduktionsregeln wurden jeweils solange auf einen Graphen angewandt, bis sich keine Änderungen mehr ergeben haben. Nach jedem Einzeldurchlauf einer Regel sorgte die Anwendung der Grad_0-Regel dafür, dass isolierte Knoten aus dem Graphen enfernt wurden. Erzielte mindestens eine der Regeln eine Reduktion, wurde der gesamte Vorgang wiederholt. In der Anwendung sorgte dies dafür, dass für manche Regeln einige weitere Iterationen, beziehungsweise Durchläufe möglich wurden. In Tabelle 3.3 ist dieser Effekt zu beobachten. Die Anwendung der Grad_1-Regel nach der Kronenregel sorgte dafür, dass bei Graph_1 drei und bei Graph_2 sogar fünf weitere Iterationen der Kronenregel eine Reduktion erzielten.

3.4 Anwendung der Reduktionsregeln

Ausgeführt wurden die Experimente auf einem 2.6 GHz, Zweikern, Intel Core i5-3320M mit 8 GB Arbeitsspeicher. In der Tabelle 3.2 werden die durchschnittliche Anzahl der Anwendungen pro Graph, die durchschnittliche Reduktion (Anzahl der Knoten, die aus dem Graphen entfernt wurden) und die durchschnittliche CPU-Zeit in Sekunden, die ein Durchlauf im Schnitt pro Graph gedauert hat gezeigt. Die Buss-Regel wurde von der Untersuchung ausgeschlossen, da kein Wert für k vorliegt, welcher für deren Anwendung essentiell ist, während die restlichen Regeln auch ohne diesen Parameter verwendbar sind.
Tabelle 3.2 spiegelt die erwarteten Werte (Tabelle 3.1) insoweit wieder, als das die Nemhauser-Trotter-Regel eine deutlich bessere Reduktion als die Kronenregel erreicht, zumindest, wenn sie einzeln angewandt wird. Dabei werden weniger Durchläufe, allerdings ein höherer Rechenaufwand benötigt.

Tabelle 3.2: Anwendung einzelner Reduktionsregeln

Reduktionsregel	**Anwendungen**	**Reduktion**	**CPU-Zeit**
Nemhauser-Trotter	0.27	50.3	0.014s
Kronenregel	0.46	19.77	0.005s
Grad 1	1.32	99.06	0.001s

Anwendungen beschreibt, wie oft nach einem Durchlauf der Reduktionsregel mindestens ein Knoten entfernt wurde; *Reduktion*, wie viele Knoten insgesamt pro Graph und *CPU-Zeit* in Sekunden die ein Durchlauf im Schnitt pro Graph gedauert hat.

Während sich die Reduktion durch die Nemhauser-Trotter und die Grad_1 Regel bei der Anwendung an dichteren Graphen, also Graphen mit einer höheren Kantenzahl, erwartungsgemäß stetig verringerte, zeigten sich bei der Kronenregel einige Ausnahmen. Wie in Abbildung 6.1 zu sehen ist, setzten sich einige wenige Graphen deutlich

vom Durchschnitt ab. In Tabelle 3.3 werden die drei Graphen mit einer Kantenmenge mit über 3000 Kanten und einer durch die Kronenregel erreichten Reduktion von > 480 Knoten betrachtet. Keiner dieser Graphen ist bipartit oder regulär. Bei allen drei Graphen führte die Anwendung der Nemhauser-Trotter-Regel zu keinerlei nennenswertem Effekt, weder isoliert, noch in Kombination mit den anderen Regeln. Ähnlich verhielt sich zunächst die Grad_1-Regel, wohingegen die Anwendung in Kombination einen großen Einfluss auf die Reduktion hatte. Werden die Regeln miteinander kombiniert, scheint die Reihenfolge, in der sie eingesetzt werden, schwer ins Gewicht zu fallen. Besonders fiel dies bei Graph_1 (3200 Kanten) auf. Wurde zuerst die Grad_1 und dann die Kronenregel verwendet, war jeweils lediglich ein Durchlauf (Anwendung) zu beobachten. Die Grad_1-Regel scheint den Graphen derart zu verändern, dass die Struktur keine Reduktion durch die Kronenregeln mehr zulässt. Das Ergebnis von Graph_2, erzeugte die Anwendung in gerade dieser Reihenfolge eine Lösung des Problems: 1000 von 1000 Knoten reduziert, wovon 619 eine Knotenüberdeckung für Graph_2 bilden. Auch hier war die Reihenfolge wichtig, wie bei der Reduktionsmenge beim entgegengesetzten Experiment zu sehen ist (zunächst Kronenregel, dann Grad_1-Regel), wo 858 Knoten bei der Reduktion aus dem Graphen entfernt wurden. Bei Graph_1 und Graph_3 zeichnet sich der übrig gebliebene Problemkern nach der jeweils größten Reduktion dadurch aus, dass der Großteil der Knoten vom Grad zwei ist. Der Problemkern G'_3 von Graph_3, zu sehen in Abbildung 3.1, lässt sich in die Knotenmengen V_1 und V_2 mit $|V_1| = |V_2| = 5$ aufteilen. Bis auf die Knoten $v_1 \in V_1$ und $v_2 \in V_2$, welche vom Grad 3 sind, hat jeder andere Knoten in G'_3 durch die vorherige Reduktion den Grad zwei. Für v_1 und v_2 existiert eine Kante (v_1, v_2) in G'_3, welche die Knotenmengen verbindet. Innerhalb der Knotenmengen existiert einen Zyklus (Kreis), mit ungerader Knotenzahl, woraus sich folgern lässt, dass G'_3 nicht bipartit ist. Hier ist keine der Reduktionsregeln mehr erfolgreich.

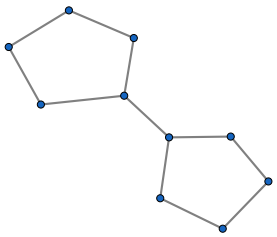

Abbildung 3.1: Graph_3 nach Anwendung von Grad_1-Regel und Kronenregel.

Die in Tabelle 3.4 zusammengefassten Ergebnisse zeigen, dass die Nemhauser-Trotter-Regel in Kombination mit anderen Regeln nicht die gleiche Reduktion erzeugt, wie Grad_1 und Kronenregel. Des Weiteren lassen sich eine Reihe von Beobachtungen anstellen.

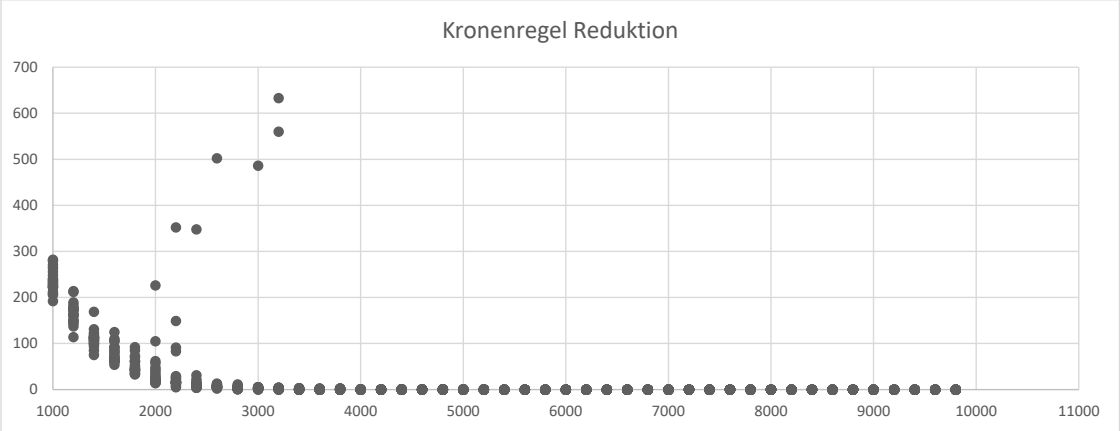

Abbildung 3.2: Anzahl der reduzierten Knoten pro Graph nach der Anwendung der Kronenregel

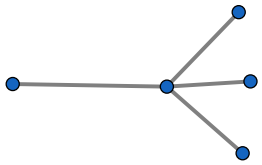

Abbildung 3.3: Einfache Krone

Da durch die Grad_1-Regel einfache 1-gradige Knoten, beziehungsweise deren Nachbarn und damit der Kopf einer einfachen Krone, reduziert wird, stellt sie eine vereinfachte Form der Kronenregel dar. Daher sollten intuitiv nach der erschöpfenden Anwendung der Kronenregel kaum Probleminstanzen, bei denen die Grad_1-Regel erfolgreich ist, übrig bleiben. Wie die Tabelle allerdings zeigt, ist die Kombination beider Regeln im Vergleich zu den anderen Zweierkombination diejenige, die im Durchschnitt am meisten Knoten aus den Problemgraphen entfernt. Zu dem besseren Ergebnis der Reduktion kommt hier noch die stark erhöhte durchschnittliche Anwendung pro Graph. Ein Grund hierfür könnte die Auswahl des ersten Matchings in der Kronenregel sein, welches maßgeblich für die darauffolgende Reduktion ist. Eine einfache Krone, wie sie in Abbildung 3.1 zu sehen ist, wird mit großer Wahrscheinlichkeit von der Kronenregel, so wie sie hier verwendet wird, ignoriert. Im Abschnitt 4.1 wird genauer erläutert, warum das der Fall ist: Bei der Anwendung auf das Testset wird die größte Reduktion mit der Kronenregel dann erreicht, wenn bei der Erstellung des Matchings M_1 zunächst Kanten betrachtet werden, bei denen beide Knoten höhergradig sind, genauer gesagt einen höheren Grad, als der durchschnittliche Knoten im aktuellen Graphen haben. Wäre die abgebildete Krone Teil eines größeren Graphen, kann es sein, dass der Kopf dieser Krone nicht in das Matching M_1 aufgenommen und so bei der weiteren Reduktion nicht berücksichtigt wird. Die Grad_1-Regel reduziert demnach jene einfachen Kronen, die von der Kronenregel

Tabelle 3.3: Besondere Graphen für die Kronenregel

Graph	**Reduktionsregeln**	**Anwendungen$_1$**	**Anwendungen$_2$**	**Reduktion**
Graph$_1$	Kronenregel	11	-	560
	Nemhauser-Trotter	1	-	4
	Grad$_1$	1	-	18
	Grad$_1$-Kronenregel	1	1	22
	Kronenregel - Grad$_1$	14	11	946
Graph$_2$	Kronenregel	13	-	486
	Nemhauser-Trotter	1	-	6
	Grad$_1$	2	-	40
	Grad$_1$-Kronenregel	12	13	1000
	Kronenregel - Grad$_1$	18	12	858
Graph$_3$	Kronenregel	15	-	633
	Nemhauser-Trotter	1	-	4
	Grad$_1$	2	-	46
	Grad$_1$-Kronenregel	18	11	990
	Kronenregel - Grad$_1$	15	9	971

Durchschnittliche Anzahl der Anwendungen der Regeln und der Durchschnitt der entfernten Knoten pro Graph für die verschiedenen Kombinationen von Regeln für die besonderen Graphen der Kronenregel-Reduktion

ignoriert, beziehungsweise nicht erkannt werden. Wird nun die Reihenfolge geändert, in der die beiden Regeln angewandt werden, zeigen sich nur minimale Änderungen in der durchschnittlichen Reduktion und der durchschnittlichen Anwendung pro Graph. Entgegen der Beobachtung für die Sonderfälle bei der Anwendung der Kronenregel zeigt sich im Allgemeinen, dass die Grad$_1$-Regel und die Kronenregel überwiegend verschiedene Arten von Kronen abdecken, da annähernd gleiche Ergebnisse unabhängig von der Reihenfolge der Regeln erzielt werden. Die Unterschiede in den Ergebnissen zeichnen sich dadurch aus, dass die als Zweites verwendete Regel im Schnitt weniger Anwendungen hat, als wenn sie als Erstes verwendet wird. Für die Kronenregel bedeutet das im Schnitt eine Differenz von 0.42 Durchläufen pro Graph, für die Grad$_1$-Regel nur 0.07. Diese 0.07 Durchläufe der Grad$_1$-Regel mehr pro Graph verschlechtern allerdings die Gesamtreduktion. Diesen Effekt kann auch bei dem Extrembeispiel von Graph$_1$ in Tabelle 3.3 beobachten. Es scheint also im Großteil der Fälle für eine maximale Reduktion von Vorteil zu sein, wenn die Kronenregel vor der Grad$_1$-Regel angewandt wird, was die anderen Beispielen in Tabelle 3.4 wiederum untermalen.

Bei der Grad_1-Regel in Kombination mit der Nemhauser-Trotter-Regel, sorgt die Reihenfolge zwar für einen Unterschied in der durchschnittlichen Anwendung, allerdings ist das Ergebnis der Reduktion identisch. Ein Teil der von der Grad_1-Regel abgedeckten Reduktion wird also vermutlich auch von der Nemhauser-Trotter-Regel entfernt und dementsprechend auch anders herum. Die Differenz bei verschiedener Reihenfolge beträgt bei der Grad_1-Regel 0.2 und bei der NT-Regel 0.26 Iterationen pro Graph, was vermuten lässt, dass die Grad_1-Regel Bereiche des Graphen, wo beide Regeln greifen, effizienter, mit weniger Durchläufen entfernt. Beim Vergleich von diese Kombination mit einer Grad_1-Kronenregel-Reduktion, fällt auch auf, dass die Form des Graphen, die die Nemhauser-Trotter-Regel durch das Entfernen von Knoten und Kanten erzeugt, die Anwendungsmöglichkeit der Grad_1-Regel einschränkt. In die andere Richtung scheint diese Einschränkung auch zu gelten. Die gesamte Reduktion pro Graph erhöht sich nicht sonderlich, im Vergleich zu den Werten, die die Grad_1-Regel alleine erzeugt (Tabelle 3.2).

Kronenregel und Nemhauser-Trotter-Regel sorgen in Kombination dafür, dass die jeweilige Anwendung pro Graph leicht erhöht wird. Auch zeigt sich ein deutlicher Anstieg der Reduktion pro Graph im Vergleich zur Einzelanwendung, was darauf hindeutet, dass die beiden Regeln wiederum verschiedene Bereiche des Graphen entfernen. Bei den Dreierkombinationen zeigt sich dieser Trend ebenfalls: Wird hier die Kronenregel unmittelbar vor der Nemhauser-Trotter-Regel angewandt, finden deutlich mehr Iterationen (der NT-Regel) pro Graph statt, als es bei vorheriger Verwendung der Grad_1-Regel der Fall ist. Zwei interessante Ergebnisse liefern die Kombinationen Grad_1 - Nemhauser-Trotter - Kronenregel und Grad_1 - Kronenregel - Nemhauser-Trotter. Die Position, beziehungsweise die Reihenfolge der Regeln beeinflusst die Anwendungshäufigkeit von Grad_1 und Nemhauser-Trotter-Regel deutlich, während Gesamtreduktion und Anwendungen der Kronenregel annähernd gleich bleiben. Dies könnte wiederum ein Indikator dafür sein, dass Nemhauser-Trotter und Grad_1-Regel ähnliche Teile des Graphen reduzieren und die Kronenregel den Graphen scheinbar für die jeweilige Reduktion günstig verändert. Werden Nemhauser-Trotter-Regel, Kronenregel und Grad_1-Regel in dieser Reihenfolge angewandt, kann im Schnitt die größte Reduktion pro Graph erreicht werden. In Abbildung 3.4 lassen sich wiederum einige Ausnahmen erkennen: Zwei Graphen stechen besonders aus dem Durchschnitt heraus, zu sehen in Tabelle 3.5. Zum einen ein Graph mit 2000 Kanten und einer Reduktion von lediglich 195 Knoten (Graph_4), während andere Graphen dieser Größe annähernd komplett gelöst wurden. Zum anderen ein Graph mit 7200 Kanten, bei dem 762 Knoten entfernt werden konnten (Graph_5). Keiner der Graphen ist, weder vor noch nach der Reduktion, regulär oder bipartit. Die Kronenregel greift bei Graph_5 nur, wenn die Grad_1-Regel den einen eingradigen und

dessen Nachbarknoten entfernt. Bis auf die Grad_1-Regel kann keine Reduktionsregel auch nur einen Knoten des Graphen entfernen, wenn sie alleine angewandt werden. Eine weitere interessante Beobachtung für Graph_5 ist, dass die Konfiguration für die Kronenregel sehr wichtig für eine hohe Reduktion ist. Die Knoten des Graphen haben vor der Reduktion einen durchschnittlichen Grad von 14 (abgerundet). Werden beim Finden des Matchings M_1 zunächst Kanten, bei denen jeder Knoten den Grad ≥ 14 hat, bevorzugt, findet keine Reduktion statt, auch nicht in Kombination mit den anderen Regeln. Wird als Einschränkung stattdessen ein Grad > 14 gewählt, werden 702 Knoten reduziert. Das Beste Ergebnis wird bei einer Konfiguration erreicht, bei der der Grad beider Knoten größer als der durchschnittliche Grad aller Knoten ist. Auf dieses Phänomen wird in Kapitel 4.1 weiter eingegangen. Bei Graph_4 zeigt sich, wie bei Graph_5, dass die Nemhauser-Trotter-Regel keinen Einfluss auf die Gesamtreduktion hat. Kronenregel und Grad_1-Regel entfernen in Kombination die Größte Menge an Knoten.

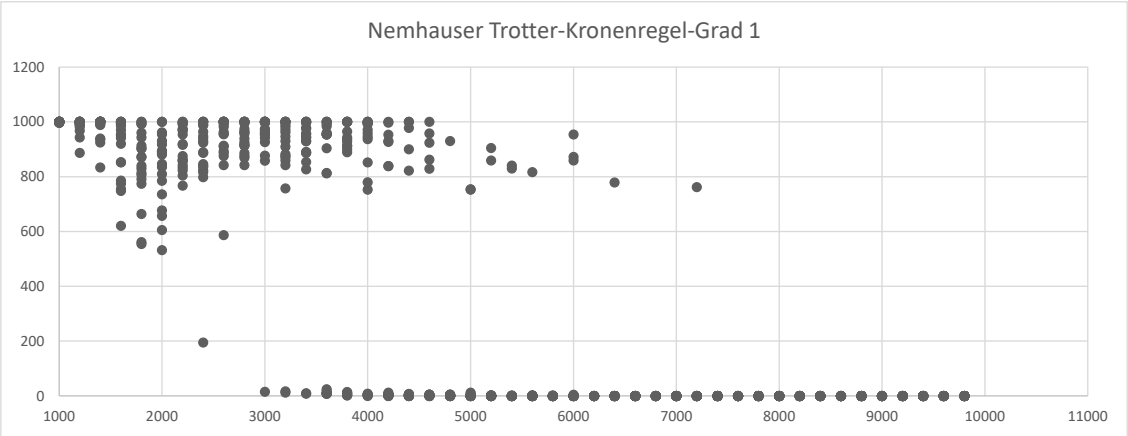

Abbildung 3.4: Anwendung von Nemhauser-Trotter-Regel, Kronenregel und Grad_1-Regel.

3.5 Zusammenfassung

Es hat sich gezeigt, dass die Regeln in der Einzelanwendung bei der Größe der Reduktion im Vergleich zueinander den Erwartungen aus Tabelle 3.1 entsprechen. Allerdings wurde festgestellt, dass die Anwendbarkeit der Regeln nicht ausschließlich von der Dichte eines Graphen abhängt, sondern stark von dessen Struktur, wie am Beispiel von Graph_5 zu sehen ist, wo genau zwei Knoten entfernt werden mussten, um die Anwendung der Kronenregel zu ermöglichen. Wie erwartet, nimmt die Anzahl der reduzierten Knoten pro Graph mit wachsender Dichte ab. Im schlimmsten Fall, zum Beispiel bei einer Clique, also einem (Teil-)Graphen, bei dem jeder Knoten eine Kante zu jedem anderen Knoten der Clique hat, ist keine Reduktion mehr möglich, da sich dann auch die Größe der kleinsten Knotenüberdeckung für diesen Graphen der Knotenmenge annähert.

Tabelle 3.4: Anwendung kombinierter Reduktionsregeln

Kombination	**Anwendungen$_1$**	**Anwendungen$_2$**	**Anwendungen$_3$**	**Reduktion**
K - G_1	3.63	4.3	-	331.8
G_1 - K	4.37	3.22	-	331.17
K - NT	0.8	0.38	-	68.28
NT - K	0.45	0.56	-	68.6
G_1 - NT	1.33	0.017	-	99.87
NT - G_1	0.28	1.13	-	99.87
K - G_1 - NT	3.61	4.29	0.11	334.67
K - NT - G_1	3.6	0.87	3.39	334.83
G_1 - NT - K	4.36	0.12	3.2	334.17
G_1 - K - NT	3.61	3.2	0.65	334.16
NT - K - G_1	0.39	3.44	4.03	335.2
NT - G_1 - K	0.91	3.42	3.2	334.16

Durchschnittliche Anzahl der Anwendungen der Regeln und der Durchschnitt der entfernten Knoten pro Graph für die verschiedenen Kombinationen von Regeln

Tabelle 3.5: Besondere Graphen für die Dreierkombinationen von Regeln

Graph	**Reduktionsregeln**	**Anwend.$_1$**	**Anwend.$_2$**	**Anwend.$_3$**	**Reduktion**
$Graph_4$	NT - K - G_1	1	5	6	195
	G_1 - NT - K	6	0	5	195
	Nemhauser-Trotter	1	-	-	11
	Kronenregel	1	-	-	11
	$Grad_1$	2	-	-	91
	Kronenregel - $Grad_1$	6	5	-	195
$Graph_5$	NT - K - G_1	1	9	2	762
	Nemhauser-Trotter	0	-	-	0
	Kronenregel	0	-	-	0
	$Grad_1$	1	-	-	2
	Kronenregel - $Grad_1$	9	3	-	762

Anzahl der Anwendungen der Kombinationen der Regeln und die Summe der entfernten Knoten für die besonderen Graphen der Nemhauser-Trotter - Kronenregel - $Grad_1$ - Reduktion

4. Implementierung

Die Regeln wurden in der Programmiersprache *C++* unter Verwendung der Bibliothek *LEDA* [8] implementiert. Sie wurden bei der Anwendung jeweils solange wiederholt, bis sich am Graphen keine Änderung mehr ergab.

4.1 Kronenregel

Beim Austesten der Kronenregel hat sich gezeigt, dass die Auswahl des Matchings M_1 im in Kapitel 2.4 dargestellten Algorithmus das Ergebnis der Reduktion in großem Maße beeinflusst, was bereits zuvor aufgefallen ist [1]. Dies ist bei der Implementierung für diese Arbeit dadurch aufgefallen, dass beim Anwenden an einem für die Kronenregel gut geeigneten Testgraphen zunächst keine Reduktion stattgefunden hat. Dieser ist in Abbildung 4.1 zu sehen. Um zu funktionieren, müssen die Kanten $(A, C), (D, E), (J, K), (L, B)$ für das Matching M_1 ausgewählt werden, da ansonsten zum Beispiel K und L nicht korrekt als Kopf der Krone, beziehungsweise nicht als der Menge H zugehörig identifiziert werden.

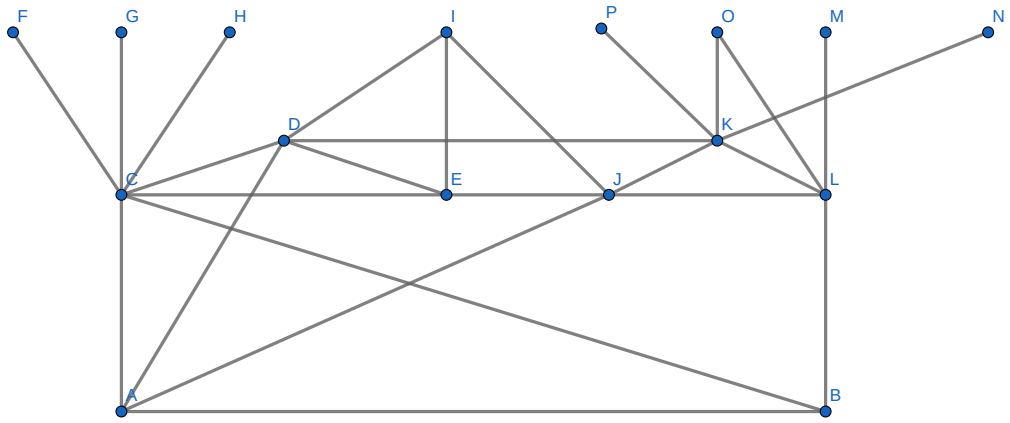

Abbildung 4.1: Beispielgraph für die Anwendung der Kronenregel

Wenn nun beim Finden von M_1 zunächst bevorzugt Kanten mit Knoten höheren Grades vor den anderen betrachtet werden, können bessere Ergebnisse in der darauf folgenden Reduktion erzielt werden und die richtigen Kanten werden getroffen. Daraufhin wurde untersucht, wie die höhergradigen Knoten, beziehungsweise die dazugehörigen Kanten ausgewählt werden müssen. Auf der einen Seite (Tabelle 4.1) wurden

Tabelle 4.1: Mindestens ein Knoten mit Einschränkung

Grad der Knoten	**Anwendungen**	**Reduktion**
keine Einschränkung	0.29	13.04
>1	0.29	13.04
>2	0.29	13.22
>3	0.27	12.92
>4	0.3	13.71
>5	0.31	13.38
>Größte Anzahl	0.32	13.44
>Durchschnittliche Anzahl	0.29	12.98

Grad der Knoten bezieht sich auf die Bedingung für die bevorzugte Auswahl der Kanten für M_1, bzw. dessen Knoten. *Anwendung* und *Reduktion* stellen jeweils den Durchschnittswert beim gesamten Testset dar.

Kanten betrachtet, bei denen das Auswahlkriterium auf mindestens einen der Knoten zutrifft, auf der anderen Seite Kanten, bei denen beide Knoten die Bedingung erfüllen (Tabelle 4.2). Die Einschränkungen *Größte Anzahl* und *Durchschnittliche Anzahl* ergeben sich jeweils aus der Menge an Knoten eines bestimmtes Grades. Bei Ersterem werden Knoten, deren Grad größer ist, als der, der im aktuellen Graphen am häufigsten vorkommt, bevorzugt. Bei Letzterem dementsprechend Knoten mit einem Grad, der größer ist, als der Durchschnittgrad im aktuellen Graphen. Diese Werte werden bei jeder Iteration des Algorithmus neu berechnet und passen sich dadurch während der Laufzeit an den Graphen an.
Generell wird eine bessere (größere) Reduktion mit der Kronenregel erreicht, wenn beim Matching M_1 zunächst Kanten betrachtet werden, bei denen die Einschränkung auf beide Knoten zutrifft. Die Reduktionsmenge bei Knoten mit $Grad > 2$ erzeugt im Vergleich mit anderen statischen Werten das beste Ergebnis. Dies könnte damit zusammenhängen, dass bei den Graphen, bei denen diese Regel sehr effektiv ist, der durchschnittliche Grad der Knoten zwei ist und es sich damit um einen eher dünnen Graphen handelt. Vermutlich erzielt die Bevorzugung des durchschnittlichen Grades das beste Ergebnis, da sich dieser Wert mit jedem Durchlauf verändert. Das Problem bei diesem Vorgehen ist, dass einfache Kronen ignoriert werden können, weshalb die Kombination mit der Grad_1-Regel vermutlich so gute Ergebnisse erzielt, wie in Kapitel 3.4 zu sehen ist.

Tabelle 4.2: Beide Knoten mit Einschränkung

Grad der Knoten	Anwendungen	Reduktion
keine Einschränkung	0.29	13.04
>1	0.36	15.34
>2	0.41	16.96
>3	0.39	16.52
>4	0.4	15.78
>5	0.4	15.72
>Größte Anzahl	0.29	13.06
>Durchschnittliche Anzahl	0.46	19.77

Grad der Knoten bezieht sich auf die Bedingung für die bevorzugte Auswahl der Kanten für M_1, bzw. dessen Knoten. *Anwendung* und *Reduktion* stellen jeweils den Durchschnittswert beim gesamten Testset dar.

4.2 Nemhauser-Trotter-Regel

Bei der Umsetzung des Algorithmus' der Nemhauser-Trotter-Regel sind keine Besonderheiten aufgefallen, wie es bei der Kronenregel der Fall war. Für das Erstellen des bipartiten Graphen B wurden zwei Referenzarrays angelegt, sodass für jeden Knoten aus G das entsprechende Paar in B gefunden werden konnte und anders herum, also für jeden Knoten aus B das entsprechende Urbild aufgerufen werden konnte. Die weiteren Berechnungen konnten dann mithilfe der LEDA-Funktion MAX_CARD_BIPARTITE_MATCHING aus mcb_matching.h und LEDA-Iteratoren angestellt werden. Um zu überprüfen, ob der erstellte Graph B tatsächlich bipartit ist und bei der Erstellung kein Fehler unterlaufen ist, wurde eigens eine entsprechende Funktion geschrieben, da LEDA eine solche nicht bereitstellt. In dieser Funktion wird getestet ob es möglich ist, die Knoten des Graphen in zwei Farben einzufärben, sodass keine zwei benachbarten Knoten die gleiche Farbe haben. Der Algorithmus funktioniert, wie folgt:

Algorithmus 4.1: Bipartit-Check

```
0 Eingabe: Graph G = (V, E)
1 foreach v ∈ V do
2   if Farbe[v]=leer then do
3     Farbe[v] ← Farbe_1
4     Queue.push(v)
5     while Queue.isNotEmpty do
6       v ← Queue.pop
7       if Farbe[N(v)]=leer oder Farbe[N(v)]=¬Farbe[v] then do
8         Farbe[N(v)] ← ¬Farbe[v]
9         Queue.push(N(v))
```

```
10        else do
11          return G is nicht bipartit
12        od
13      od
14    od
15  od
16  return G ist bipartit
```

Die alles umschließende Schleife (Zeile 1) sorgt dafür, dass auch nicht verbundene Teile auf Bipartition überprüft werden. Genau genommen überprüft der Algorithmus, ob in jedem nicht miteinander verbundenen Teil des Eingabegraphen eine Bipartition herrscht.

5. Diskussion und Ausblick

Eine Frage, die sich bei der Anwendung der Nemhauser-Trotter-Regel stellt, ist, warum sie in der Praxis so niedrige Reduktionen erreicht. Zwar zeigte die Einzelanwendung Nemhauser-Trotter-Regel, dass die Ergebnisse im Vergleich zur Einzelanwendung der Kronenregel wie erwartet besser sind, allerdings war dies im weiteren Verlauf der Untersuchung nicht mehr zu beobachten. Hierfür könnten Probleminstanzen, bei denen die Nemhauser-Trotter-Regel gute Ergebnisse liefert, genauer betrachtet werden, wodurch sich eventuell eine Modifikation, wie bei der Kronenregel ableiten lässt. Die Modifikation der Kronenregel gilt es weiterhin genauer zu untersuchen, da nicht garantiert ist, dass die in dieser Arbeit verwendete Einschränkung für das Finden des Matchings M_1 die besten Ergebnisse erzielt. Außerdem wurde nicht getestet, welchen Einfluss die Verwendung eines Maximum Matchings an dieser Stelle hätte. Des weiteren hat sich die Ähnlichkeiten bei der Reduktion zwischen Grad_1-Regel und Nemhauser-Trotter-Regel gezeigt. Bei der Kombination von Kronenregel und Grad_1-Regel führte die besagte Modifikation der Kronenregel dazu, dass verschiedene Kronen, beziehungsweise verschiedene Teile des Graphen betrachtet wurden und letztendlich zu einem drastisch besseren Ergebnis. Würde eine solche ebenfalls für die Nemhauser-Trotter-Regel werden, könnte diese in der Praxis besser anwendbar werden. In Abbildung 3.4 zeigte sich, dass es scheinbar große Unterschiede in der Reduzierbarkeit der Graphen mit 3000 bis 4600 Knoten gibt. Hier wäre interessant festzustellen, welche Eigenschaften diese Gruppen unterscheiden und ob eventuell andere, in dieser Arbeit nicht verwendete Graphreduktionsregeln, wie die Folding-Regel erfolgreich anwendbar wären.
Insgesamt scheint es ein sehr großes Ungleichgewicht zwischen der Zeit, die für Preprocessing und für das Finden einer Knotenüberdeckung im Anschluss aufgebracht wird, zu geben. Daher wäre es sinnvoll, viel Aufwand bei der Vorverarbeitung zu betreiben, da wie in Kapitel 1.1 gezeigt, bereits eine einfache Verringerung von k die Laufzeit eines Suchbaumalgorithmus' halbiert.

6. Anhang

Tabelle 6.1: Statistiken über Graph_5

Status	**Grad der Knoten**	**Knoten**	**Grad der Knoten**	**Knoten**
Vor der Reduktion	0	0	15	106
	1	1	16	99
	2	1	17	84
	3	0	18	66
	4	1	19	39
	5	3	20	27
	6	12	21	16
	7	13	22	18
	8	17	23	6
	9	37	24	3
	10	65	25	4
	11	59	26	5
	12	101	27	3
	13	99	28	1
	14	114		
Nach der Reduktion	0	0	6	18
	1	0	7	11
	2	55	8	8
	3	59	9	0
	4	46	10	0
	5	41	11	1

Anzahl der Knoten pro Grad vor und nach der Reduktion für den besonderen Graphen der Nemhauser-Trotter - Kronenregel - Grad_1 - Reduktion

Im Folgenden sind die Ergebnisse der restlichen Tests zu finden.

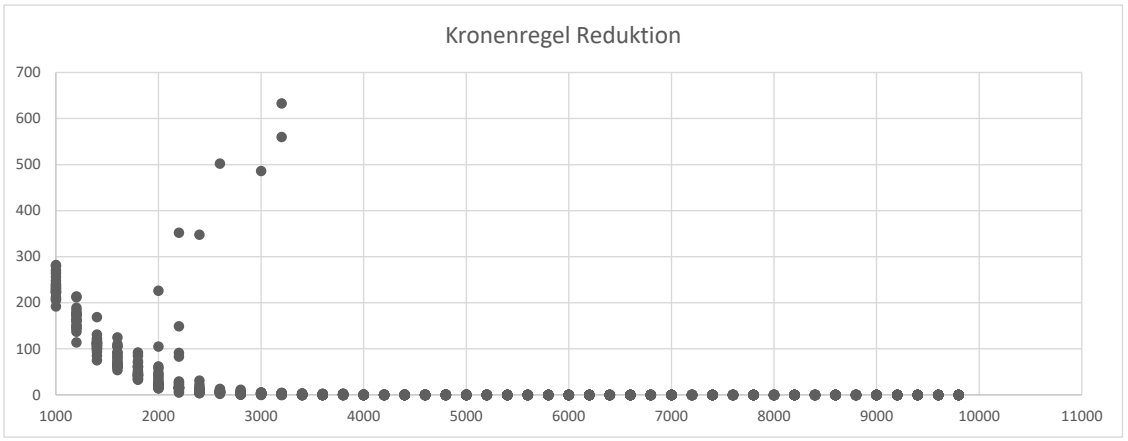

Abbildung 6.1: Anzahl der reduzierten Knoten pro Graph nach der Anwendung der Kronenregel

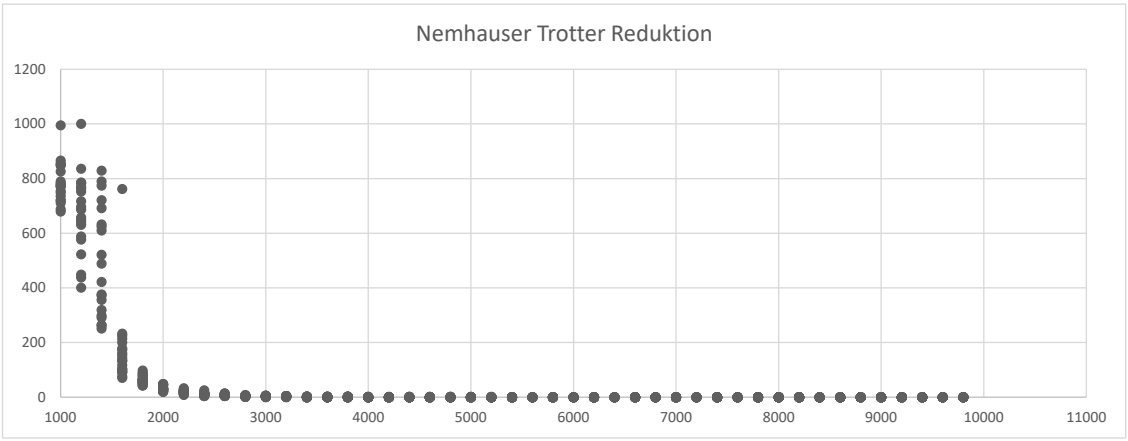

Abbildung 6.2: Anzahl der reduzierten Knoten pro Graph nach der Anwendung der Nehmhauer-Trotter-Regel

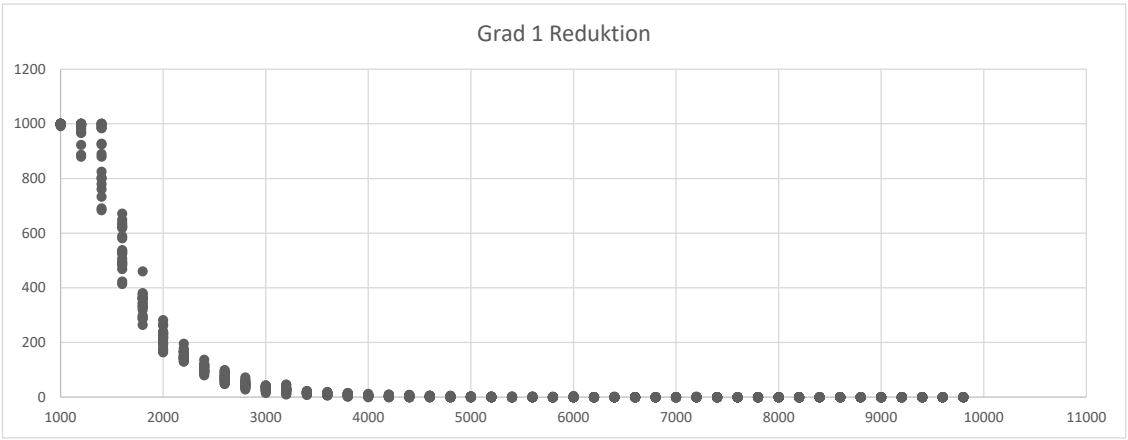

Abbildung 6.3: Anzahl der reduzierten Knoten pro Graph nach der Anwendung der Grad_1-Regel

Literaturverzeichnis

[1] F. N. Abu-Khzam, M. R. Fellows, M. A. Langston, and W. H. Suters, "Crown structures for vertex cover kernelization," *Theory of Computing Systems, vol 41, Issue 3, 411–430*, 2007.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability : a guide to the theory of NP-completeness.* Freeman New York, 2007.

[3] G. Valiente, *Algorithms on Trees and Graphs.* Springer, Berlin, Heidelberg, 2002.

[4] F. Gurski, *Exakte Algorithmen für schwere Graphenprobleme.* Springer Heidelberg, 2010.

[5] R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations, 85-103*, 1972.

[6] F. Abu-Khzam, R. Collins, M. Fellows, M. Langston, and W. S. C. Symons, "Kernelization algorithms for the vertex cover problem: Theory and experiments," *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX), ACM/SIAM, Proc. Applied Mathematics 115*, 2004.

[7] R. G. Downey and M. R. Fellows, *Fundamentals of parameterized complexity.* Springer London, 2013.

[8] K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing.* Cambridge University Press, 2006.

[9] J. Chen, I. A. Kanj, and W. Jia, "Vertex cover: further observations and further improvements," *Journal of Algorithms 41, 280 - 301*, 2001.

[10] D. Mölle, S. Richter, and P. Rossmanith, "Enumerate and expand: New runtime bounds for vertex cover variants," *Computing and Combinatorics. Lecture Notes in Computer Science, vol 4112, 265-273*, 2006.

[11] R. G. Downey and M. R. Fellows, *Parametrized Complexity.* Springer New York, 1997.

[12] F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C. T. Symons, "Scalable parallel algorithms for fpt problems," *Algorithmica, Volume 45, Number 3, 269-284*, 2006.

[13] M. Cygan, *Parameterized algorithms.* Springer Cham, 2015.

[14] G. Nemhauser and L. E. Trotter, "Vertex packings: Structural properties and algorithms," *Mathematical Programming 8, 232-248.*

[15] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing, vol. 2, No. 4, 225-231*, 1973.

[16] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[17] H. Fernau, "Parameterized algorithmics: A graph-theoretic approach," *Universität Tübingen*, 2005.

[18] N. Blum, "A new approach to maximum matching in general graphs," *ICALP: Automata, Languages and Programming, 586-597*, 1990.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher auch nicht veröffentlicht.

Trier, den 2. März 2018