

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. Zu untersuchen waren die Regeln als solche, das Verhalten verschiedener Regeln in der Praxis und das Ergebnis von deren Kombination. Für die Umsetzung und die Implementierung der Algorithmen wurden die Programmiersprache *C++* und die *C++*-Library LEDA verwendet.

- Habe ich das so erwartet?

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Problemstellung	3
1.3	Zielsetzung	3
1.4	Gliederung der Arbeit	4
2	Grundlagen	5
2.1	Knotenüberdeckung	5
2.2	Einfache Reduktionsregeln	5
2.3	Nemhauser-Trotter Reduktionsregeln	6
2.4	Kronenregel	8
3	Analyse	10
3.1	Anforderungen	10
3.2	Bewertung der Reduktionsregeln	10
3.3	Anwendung der Reduktionsregeln	11
3.4	Interpretation der Ergebnisse	17
3.5	Zusammenfassung	17
4	Implementierung	18
4.1	Kronenregel	18
4.2	Nemhauser-Trotter-Regel	20
5	Diskussion und Ausblick	21
6	Anhang	22
	Literaturverzeichnis	23

Abbildungsverzeichnis

1.1	Graph eines Stromnetzes	1
2.1	Laufzeit der Nemhauser-Trotter-Regel am Testset	8
2.2	Laufzeit der Kronenregel am Testset	9
3.1	Graph ₃ nach Anwendung von Grad ₁ -Regel und Kronenregel.	13
3.2	Anwendung der Kronenregel.	13
3.3	Einfache Krone	13
3.4	Anwendung von Nemhauser-Trotter-Regel, Kronenregel und Grad ₁ -Regel.	16
4.1	Beispielgraph für die Anwendung der Kronenregel	18

Tabellenverzeichnis

1.1	Exponentielle Laufzeit	3
3.1	Bewertung der Reduktionsregeln	11
3.2	Anwendung einzelner Reduktionsregeln	12
3.3	Besondere Graphen für die Kronenregel	14
3.4	Anwendung kombinierter Reduktionsregeln	16
3.5	Besondere Graphen für die Dreierkombinationen von Regeln	17
4.1	Mindestens ein Knoten mit Einschränkung	19
4.2	Beide Knoten mit Einschränkung	19
6.1	Statistiken über Graph_5	22

1. Einleitung

Die Einleitung besteht aus der Motivation, der Problemstellung, der Zielsetzung und einem ersten Überblick über den Aufbau der Arbeit.

1.1 Motivation

Das *Knotenüberdeckungsproblem*, oder englisch Vertex Cover, ist ein nachgewiesen NP-vollständiges Problem [1], kann also von einem nichtdeterministischen Algorithmus in Polinomialzeit gelöst werden. Um das Problem zu erklären, betrachten wir ein Netz von Haushalten, bei dem wir die Möglichkeit haben, 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 erheblich 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. In

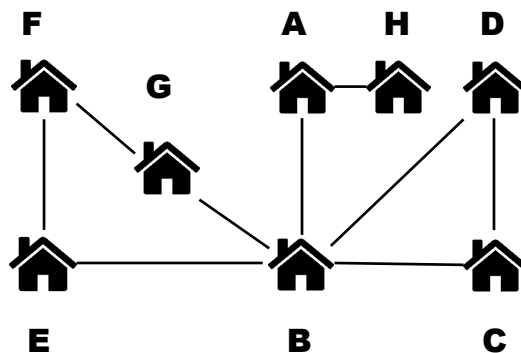


Abbildung 1.1: Graph eines Stromnetzes

Abbildung 1.1¹ sieht man ein Beispiel für eine solche Probleminstanz. Die Haushalte sind jeweils mit einem Buchstaben gekennzeichnet; jede Linie, beziehungsweise Kante eine Leitung zum Nachbarhaus. 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 zum Beispiel durch ausprobieren leicht zu sehen, dass es keine Lösung mit weniger Häusern, beziehungsweise Generatoren gibt. Um 4 Häuser, die das Problem lösen zu finden, könnten alle Möglichkeiten ausprobiert werden. Ein solcher naiver Algorithmus würde eine Laufzeit von $O(n^k \cdot m)$ (m = Anzahl der Leitungen) brauchen, da für jede k große Kombination aus Häusern, die Versorgung der Leitungen überprüft werden muss.

¹Icons in der Graphik von <https://www.flaticon.com/>

Mit einem einfachen Suchbaumalgorithmus betrachtet man jede Leitung nur einmal und entscheidet, welches der beiden Häuser mit einem Generator versorgt werden soll, da mindestens eins der beiden Häuser in der Lösungsmenge ist. Dadurch ergibt sich für jede Entscheidung eine Verzweigung. 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 eines 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, ersetzt man die Häuser durch Knoten eines Graphen und die Leitungen durch dessen Kanten, zum Knotenüberdeckungsproblem transformieren lässt. Zu diesem Problem lassen sich zwei Fragestellungen unterscheiden: Zum Einen das Finden einer minimalen Knotenüberdeckung [2]:

EINGABE : Graph $G = (V, E)$, positive Integer $k \leq |V|$
AUSGABE : $S \subseteq V$ mit $|S| \leq k$,
 sodass jede Kante aus E einen Endpunkt in S hat.

Zum Anderen das Entscheidungsproblem, ob es eine Knotenüberdeckung einer Größe $\leq k$ gibt:

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

Letzteres ist in der Liste von Karp's 21 NP-vollständigen Problemen zu finden [3]. Es existieren weitere reale Probleme, für die das Knotenüberdeckungsproblem anwendbar ist. Abu-Khzam et al [4] schlagen eine Anwendung in der Bioinformatik vor. Ein nichtdeterministischer Algorithmus kann jede vermeintliche Lösungskombination aus Knoten, beziehungsweise Haushalten in Polynomialzeit testen [1], 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 also sinnvoll, die Eingabe möglichst klein zu halten und 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 reduzieren. Das Ziel der in dieser Arbeit verwendeten Reduktionsregeln ist für einen gegebenen Graphen $G = (V, E)$ und 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. Für den Suchbaumalgorithmus (mit $O(2^k \cdot m)$) bedeutet jede Reduzierung von k eine Verdoppelung der Geschwindigkeit.

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 eines Algorithmus' mit einer Laufzeit von $O(2^n)$

1.2 Problemstellung

Das Knotenüberdeckungsproblem ist in der Literatur viel vertreten und es wurden verschiedene Reduktionsregeln 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 Regel greift an verschiedenen Stellen eines Graphen und es scheint, dass die Reihenfolge, in der verschiedene Regeln auf einen Graphen angewandt werden einen Einfluss auf die insgesamt erreichte Reduktion hat. In dieser Arbeit wird untersucht, wie sich verschiedene Regeln in der Praxis verhalten, sowohl einzeln als auch in Kombination, und verglichen.

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 [5] implementiert, welche Algorithmen und eingetragene Datentypen bereitstellt. Unter anderem liefert LEDA Datenstrukturen für Graphen, Algorithmen für Graphen, wie zum Beispiel eine Funktion, mit der ein Maximum-Bipartite-Matching in einem Graphen gefunden und ausgegeben wird. Durch die Vielzahl an Funktionen eignet sich die Library für diese Arbeit. Jede Regel 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, um dann 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 Kapitel wird auf Besonderheiten in der Implementierung und Umsetzung der Reduktionsregeln eingegangen, um abschließend im Fazit 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 Grad₀-, Grad₁- und Buss-Regel lediglich den Grad einzelner Knoten betrachtet. Darüber hinaus können die Regeln weiter unterschieden werden. All die zu zuvor genannten Regeln entfernen induzierte Teilgraphen aus dem Graphen, an dem sie angewandt werden. Die Folding-Regel [6] (oder Grad₂-Regeln) zum Beispiel nicht. Hier werden die Nachbarn u und v eines Knoten x des Grades 2 verschmolzen, vorausgesetzt, es gibt keine Kante zwischen diesen Knoten. Es wird ein neuer Knoten x' mit $N(x') = N(u) \cup N(v) \setminus \{x, u, v\}$ eingesetzt und k verringert. Diese Konstruktion muss dann nach dem Durchlaufen des Algorithmus wieder entpackt werden, da sonst nach der Reduktion kein äquivalenter Problemgraph entsteht. Eine schnelle Laufzeit, um zu entscheiden, ob eine Knotenüberdeckung in einem gegebenen Graphen G , Parameter k und einer Knotenmenge n zu finden ist, wird von Chen et al [6] mit $O(kn + 1.271^k k^2)$ vorgeschlagen, welche allerdings unter anderem die Folding-Regel verwendet; die schnellste bekannte Methode, alle Knotenüberdeckungen eines Graphen zu zählen von Mölle et al [7] mit einer Laufzeit von $O(1.3803^k)$.

2.1 Knotenüberdeckung

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, da sie nicht zu einer minimalen Knotenüberdeckung gehören können (Grad₀-Regel)
2. Bei Knoten, die genau eine Kante, beziehungsweise genau einen Nachbarknoten haben, wird der entsprechende Nachbarknoten automatisch in die Lösungsmenge aufgenommen, da einer der beiden Knoten in der Knotenüberdeckung vorkommt und der Nachbarknoten mindestens den Grad 1 hat, also möglicherweise mehr Kanten abdeckt (Grad₁-Regel)
3. Knoten mit $k + 1$ Kanten, werden der Knotenüberdeckung hinzugefügt, da sonst alle Nachbarknoten aufgenommen werden und somit mehr, als k Knoten in der Lösungsmenge wären (Buss-Regel)

Angewandt an eine Problem Instanz 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 \Rightarrow 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, wo 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 \Rightarrow 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 sind von der weiteren Betrachtung ausgeschlossen und quasi aus dem Graphen entfernt. 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 Problem Instanz 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[8], da jede der Kanten an 2 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 2 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 $d \geq k$ (wobei d der größte Grad unter den Knoten aus G ist) 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 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 [9]:

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 [10] können diese Mengen mithilfe des Erstellens eines auf dem Problemgraphen basierenden bipartiten Graphen in Polinomialzeit gefunden werden. Der im Zuge dieser Arbeit verwendete Algorithmus ist wie folgt:

Algorithmus 2.1: Nemhauser-Totter-Regel.

```

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

```

Das Erstellen des Bipartiten Graphen in den Zeilen 1-2 benötigt $n \cdot d$, 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:

$$n \cdot d + \sqrt{2n} \cdot 2m + 2n \quad (2.1)$$

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

Die Laufzeitabschätzung von $O(n + \sqrt{n} \cdot m)$ entsteht, wenn d als Konstante betrachtet wird. Man sieht, dass die Laufzeit des Algorithmus' hauptsächlich vom Finden des Matchings dominiert wird, der Algorithmus eignet sich also für eine Vorverarbeitung des Graphen in Polinomialzeit. 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 $S \subset E$, 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 Problemerkern 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 [11]. Wie in Abbildung 2.1 zu sehen ist, verhält sich die Laufzeit des implementierten Algorithmus' entsprechend der Abschätzung linear zur Eingabe.

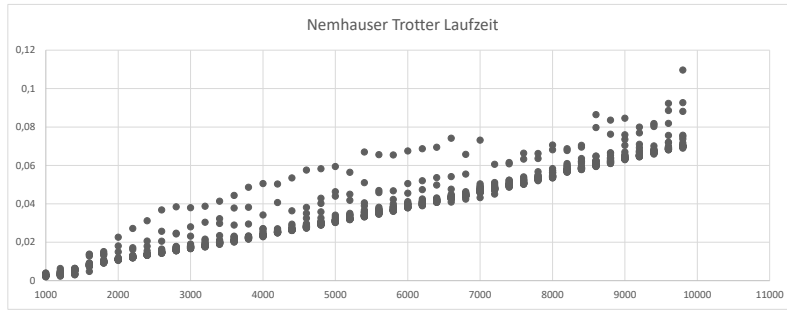


Abbildung 2.1: Laufzeit der Nemhauser-Trotter-Regel am Testset

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. $H = N(I)$,
2. I ist ein Independent Set, was bedeutet: $\forall v, w \in I$ gilt $(vw) \notin E$ und
3. die Kanten zwischen H und I enthalten ein Matching 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 keine Kanten untereinander hat. Werden also nur die Nachbarknoten, beziehungsweise H in die Überdeckung aufgenommen, kann man sicher sein, dass keine Kanten vernachlässigt werden. Für die Untersuchung wurde der folgende Algorithmus implementiert:

Algorithmus 2.2: Kronenregel.

```

0   $G = (V, E), n := |V|, m := |E|, d := \text{maximaler Grad der Knoten aus } G$ 
1   $// M_1 = \text{Maximal Matching von } G$ 
2   $M_1 \leftarrow \emptyset$ 
3  foreach  $e \in E$  do
4       $M_1 \leftarrow M_1 \cup e$ 
5      Entferne  $e$  und  $N(e)$  aus der weiteren Betrachtung
6  od
7   $O \leftarrow \text{nicht gepaarte Knoten in } M_1$ 
8   $M_2 \leftarrow \text{Maximum Matching von } B = (O, N(O), \{uv | u \in O \wedge v \in N(O)\})$ 
9   $I \leftarrow \text{nicht gepaarte Knoten aus } O \text{ in } M_2$ 
10  $I' \leftarrow \emptyset$ 
11 while  $I' \neq I$  do
12      $I' \leftarrow I$ 
13      $H \leftarrow N(I)$ 
14      $I \leftarrow I \cup \{\forall u \in O | \exists v \in H (uv \in M_2)\}$ 
15 od

```

16 **return** I

Um das Maximal Matching M_1 zu erstellen werden im schlimmsten Fall $m \cdot d$ Schritte benötigt, da dann d Nachbarkanten deaktiviert werden. Ist $d = 1$, so wird jede Kante aus E betrachtet, beziehungsweise in M_1 aufgenommen. Gilt $|M_1| > 2k$, 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 k kleiner ist, allerdings fällt dies für die Worst Case Abschätzung nicht sehr schwer ins Gewicht. Um das Matching 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 $n \cdot d$, da die Knoten aus O und deren Nachbarknoten überprüft werden müssen. Insgesamt ergibt das eine Abschätzung von:

$$m \cdot d + n + \frac{nd}{2} + \sqrt{n} \cdot m + n + n \cdot d \quad (2.3)$$

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

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

Wieder erhält man $O(\sqrt{n} \cdot m)$, 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 vermutlich wegen der schnelleren Laufzeit, als das Finden eines Maximum Matchings in einem allgemeinen Graph verwendet. Allerdings gibt es nach Blum [13] einen Algorithmus, der ein Maximum Matching in einem allgemeinen Graphen in $O(\sqrt{n} \cdot m)$ findet, was die Gesamtlaufzeit der Kronenregel zwar um den Faktor 2 verschlechtern, allerdings eventuell das Ergebnis der Reduktion verbessern würde. Abu-Khzam et al. [12] 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

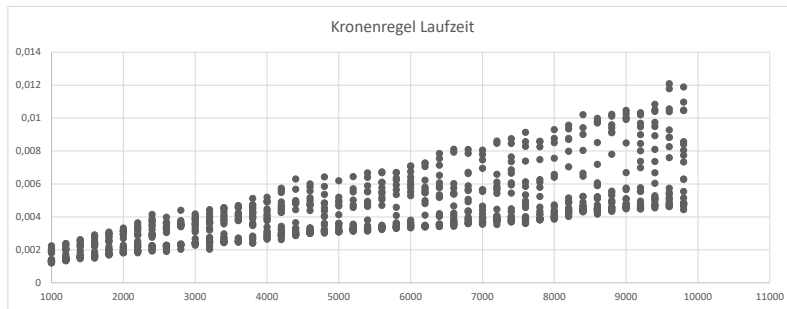


Abbildung 2.2: Laufzeit der Kronenregel am Testset

sehen ist, verhält sich die Laufzeit des implementierten Algorithmus' entsprechend der Abschätzung linear zur Eingabe.

3. Analyse

3.1 Anforderungen

Um einen Graphen, der zur Diskussion steht auf einen Problemkern zu reduzieren können verschiedene Reduktionsregeln verwendet werden.

Zunächst werden die Regeln anhand der folgenden Kriterien beurteilt:

- Laufzeit
- Erwartete Reduktion
- Ressourcenverbrauch

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

Um die verwendeten Reduktionsregeln zu bewerten, beziehungsweise zu vergleichen, betrachten wir auf der einen Seite die theoretisch mögliche Reduktion und auf der anderen, wie sich der implementierte Algorithmus verhält. Für jede Regel wird vorausgesetzt, dass zusätzlich alle Kanten des Grades 0 aus dem Graphen entfernt werden, da die Knotenmenge der reduzierten Instanz ansonsten nicht durch k abgeschätzt werden kann.

In Tabelle 3.1 stehen die Laufzeiten in der O -Notation und erwartete Knotenanzahl der reduzierten Problemkerns der Nemhauser-Trotter-Regel [11], der Kronenregel [11] [5] und die der Buss Regel [11] gegenüber.

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	$O(k \cdot n)$	$\leq k^2$

* Die Reduktion dieser Regeln hängt stark von der Struktur des Graphen ab. Eine formale Bestimmung der zu erwartenden Reduktion steht noch aus.
 k beschreibt jeweils den Eingabeparameter der Knotenüberdeckung für einen Graphen $G = (V, E)$

3.3 Anwendung der Reduktionsregeln

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|)` [5] verwendet. Da Zufallsgraphen verwendet werden, gibt es keine Vorgabe für den Parameter k . Die obere Beschränkung der Kantenmenge (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. Daher 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 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. 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 möglich wurden. In Tabelle 3.3 kann man diesen Effekt beobachten. Die Anwendung der Grad₁-Regel nach der Kronenregel sorgte dafür, dass bei Graph₁ 3 und bei Graph₂ sogar 5 weitere Iterationen der Kronenregel eine Reduktion erzielten. 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, die pro Graph aufgebracht wurde. Die

¹<http://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/>

²<http://sites.nlsde.buaa.edu.cn/kexu/benchmarks/graph-benchmarks.htm>

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 (3.1) in soweit wieder, als das die Nemhauser-Trotter-Regel eine deutlich bessere Reduktion als die Kronenregel erreicht, zumindest, wenn sie isoliert 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

Pro Zeile sieht man die jeweilige Reduktionsregel und den Durchschnitt pro Graph im Testset. *Anwendungen* beschreibt, wie oft nach einem Durchlauf mindestens ein Knoten entfernt wurde; *Reduktion*, wie viele Knoten insgesamt pro Graph; *CPU-Zeit* in Sekunden wie lange 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 3.2 zu sehen ist, setzten sich diese deutlich vom Durchschnitt ab. In Tabelle 3.3 werden die drei Graphen mit einer Kantenzahl über (beziehungsweise gleich) 3000 und einer durch die Kronenregel erreichten Reduktion > 480 Knoten betrachtet. Keiner dieser Graphen ist bipartit. 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 Grad₁-Regel, wohingegen die Anwendung in Kombination einen großen Einfluss auf die Reduktion hatte. Kombiniert man die Regeln, scheint die Reihenfolge, in der sie eingesetzt werden schwer ins Gewicht zu fallen. Besonders fiel das bei Graph₁ (3200 Kanten) auf. Wurde zuerst die Grad₁ und dann die Kronenregel verwendet, war jeweils lediglich ein Durchlauf (Anwendung) zu beobachten. Die Grad₁-Regel scheint den Graphen derart zu verändern, dass die Struktur keine Reduktion durch die Kronenregeln mehr zulässt. Betrachtet man dagegen das Ergebnis von Graph₂, erzeugte die Anwendung in gerade dieser Reihenfolge eine Lösung des Problems: 1000 von 1000 Knoten reduziert, wovon 619 eine Knotenüberdeckung für Graph₂ bilden. Auch hier war die Reihenfolge wieder wichtig, wie man bei der Reduktionsmenge beim entgegengesetzten Experiment (erst Kronenregel, dann Grad₁-Regel) sieht, wo 858 Knoten bei der Reduktion aus dem Graphen entfernt wurden. Bei Graph₁ und Graph₃ zeichnet sich der übrig gebliebene Problemkern nach der jeweils größten Reduktion dadurch aus, dass der Großteil der Knoten vom Grad 2 ist. Der Problemkern G'_3 von Graph₃, 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 2. 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.

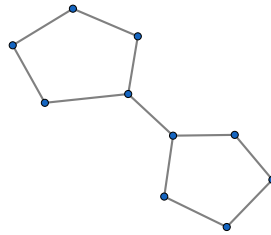


Abbildung 3.1: Graph₃ nach Anwendung von Grad₁-Regel und Kronenregel.

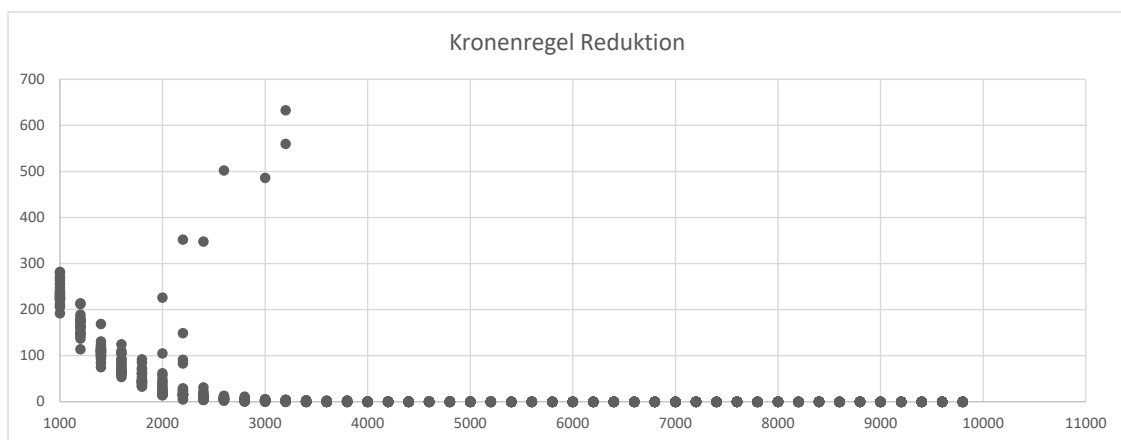


Abbildung 3.2: Anwendung der Kronenregel.

Die in Tabelle 3.4 zusammengefassten Ergebnisse zeigen, dass die Nemhauser-Trotter-Regel in Kombination mit anderen Regeln nicht die gleiche Reduktion erzeugt, wie Grad₁ und Kronenregel. Des Weiteren lassen sich eine Reihe von Beobachtungen anstellen.

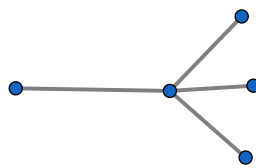


Abbildung 3.3: Einfache Krone

Da durch die Grad₁-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 Grad₁-Regel erfolgreich ist, übrig bleiben. Wie die Tabelle allerdings zeigt, ist die Kombination beider Regeln im Vergleich zu den anderen Zweierkombination diejenige, die am meisten Knoten aus den Problemgraphen entfernt. Zu dem besseren Ergebnis der Reduktion kommt hier noch die stark erhöhte durchschnittliche Anwendung pro Graph. Ein

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

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, 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 Grad₁-Regel reduziert demnach jene einfachen Kronen, die von der Kronenregel ignoriert, beziehungsweise nicht erkannt werden. Ändert man nun die Reihenfolge, in der die beiden Regeln angewandt werden, zeigen sich nur minimale Änderungen in der durchschnittlichen Reduktion und der durchschnittlichen Anwendung pro Graph. Das könnte die Vermutung stützen, dass die Grad₁-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 eine Differenz von 0.42 durchschnittlichen Durchläufen, für die Grad₁-Regel nur 0.07. Diese

0.07 Grad₁-Regel-Durchläufe mehr pro Graph verschlechtern allerdings die Gesamt-reduktion. Diesen Effekt kann auch bei dem Extrembeispiel von Graph₁ in Tabelle 3.3 beobachten, wo der gleiche Effekt eintritt. Es scheint also im Großteil der Fälle für eine maximale Reduktion von Vorteil zu sein, wenn die Kronenregel vor der Grad₁-Regel angewandt wird, was die die anderen Beispielen in Tabelle 3.4 wiederum untermalen.

Betrachtet man die Grad₁-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 Grad₁-Regel abgedeckten Reduktion wird auch von der Nemhauser-Trotter-Regel entfernt und dem entsprechend auch anders herum. Die Differenz bei verschiedener Reihenfolge beträgt bei der Grad₁-Regel 0.2 und bei der NT-Regel 0.26 Iterationen pro Graph, was vermuten lässt, dass die Grad₁-Regel Bereiche des Graphen, wo beide Regeln greifen effizienter, mit weniger Durchläufen entfernt. Vergleicht man diese Kombination mit einer Grad₁-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 Grad₁-Regel einschränkt. In die andere Richtung scheint diese Einschränkung auch zu gelten. Die gesamte Reduktion pro Graph erhöht sich nicht sonderlich, vergleicht man die Werte, die die Grad₁-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 Grad₁-Regel der Fall ist. Zwei Interessante Ergebnisse liefern die Kombinationen Grad₁ - Nemhauser-Trotter - Kronenregel und Grad₁ - Kronenregel - Nemhauser-Trotter. Die Position, beziehungsweise die Reihenfolge der Regeln beeinflusst die Anwendungshäufigkeit von Grad₁ und Nemhauser-Trotter-Regel deutlich, während Gesamtreduktion und Anwendungen der Kronenregel annähernd gleich bleiben. Dies könnte wiederum ein Indikator dafür sein, dass NT und Grad₁-Regel ähnliche Teile des Graphen reduzieren und die Kronenregel den Graphen scheinbar für die jeweilige Reduktion günstig verändert. Werden Nemhauser-Trotter-Regel, Kronenregel und Grad₁-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 (Graph₄), während andere Graphen dieser Größe annähernd komplett gelöst wurden. Zum Anderen ein Graph mit 7200 Kanten, bei dem 762 Knoten entfernt werden konnten (Graph₅). Keiner der Graphen ist - weder vor noch nach der Reduktion - regulär oder bipartit. Die Kronenregel greift bei Graph₅ nur, wenn die Grad₁-Regel den einen eingradigen und dessen Nachbarknoten entfernt. Bis auf die Grad₁-Regel kann keine Reduktionsregel auch nur einen Knoten des Graphen entfernen, wenn sie alleine angewandt werden. Eine weitere interessante Beobachtung für Graph₅ 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, kann findet keine Reduktion statt, auch nicht in Kombination mit den anderen Regeln. Wählt man als Einschränkung stattdessen Grad > 14 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 Graph₄ zeigt sich, wie bei Graph₅, dass die Nemhauser-Trotter-Regel keinen Einfluss auf die Gesamtreduktion hat. Kronenregel und Grad₁-Regel entfernen in Kombination die Größte Menge an Knoten.

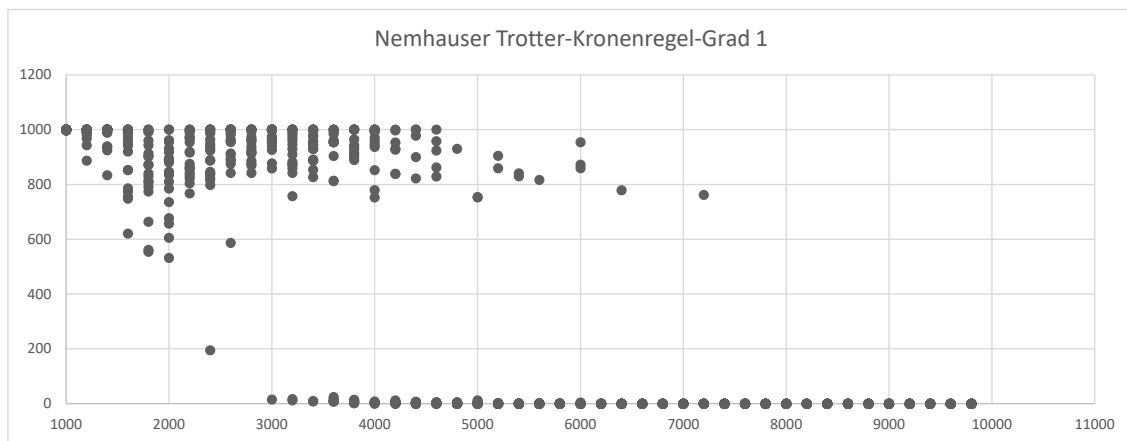


Abbildung 3.4: Anwendung von Nemhauser-Trotter-Regel, Kronenregel und Grad₁-Regel.

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 - Grad₁ - Reduktion

3.4 Interpretation der Ergebnisse

- Gegenüberstellung der Reduktionsergebnisse: Einzeln, wie in der Theorie, Kombination sehr anders und Abhängig von der Reduktionsregel
- Wie zu erwarten, keine großen Reduktionen bei dichten Graphen. Graphen werden dichter $\Rightarrow k$ erhöht sich \Rightarrow Keine Reduktion mehr erfolgreich

3.5 Zusammenfassung

Am Ende sollten ggf. die wichtigsten Ergebnisse nochmal in *einem* kurzen Absatz zusammengefasst werden.

4. Implementierung

Die Regeln wurden in der Programmiersprache *C++* unter Verwendung der Bibliothek *LEDA* [5] 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 bei der Implementierung der Kronenregel aufgefallen ist [12]. 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. Wenn nun beim Finden von M_1 zunächst bevorzugt Kanten mit Knoten hö-

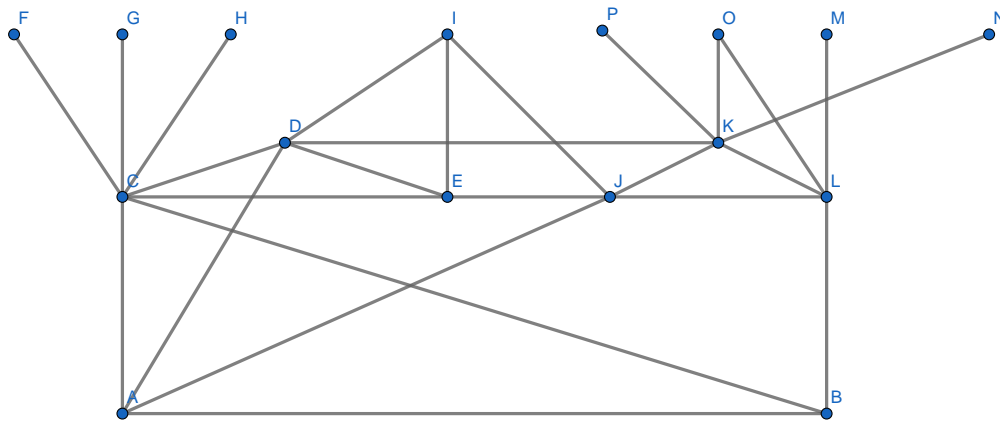


Abbildung 4.1: Beispielgraph für die Anwendung der Kronenregel

heren 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. Zum einen (Tabelle 4.1) wurden 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). *Größte Anzahl* und *Durchschnittliche Anzahl* ergeben sich jeweils aus der Menge an Knoten eines bestimmten Grades. Bei Ersterem werden Knoten, deren Grad im aktuellen Graphen am häufigsten vorkommt bevorzugt.

Bei Letzterem dementsprechend Knoten, deren Grad im aktuellen Graphen dem Durchschnitt entspricht. 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 Machting 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 2 ist. Vermutlich erzielt die Bevorzugung des durchschnittlichen Grades das beste Ergebnis, da sich dieser Wert mit jedem Durchlauf verändert.

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.

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.

Das Problem bei diesem Vorgehen ist, dass einfache Kronen ignoriert werden können, weshalb die Kombination mit der Grad₁-Regel vermutlich so gute Ergebnisse erzielt, wie in Kapitel 3.3 zu sehen ist.

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 versucht, die Knoten des Graphen in zwei Farb 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  $v \in V$  do
2    if  $\text{Farbe}[v] = \text{leer}$  then do
3       $\text{Farbe}[v] \leftarrow \text{Farbe}_1$ 
4       $\text{Queue.push}(v)$ 
5      while  $\text{Queue.isNotEmpty}$  do
6         $v \leftarrow \text{Queue.pop}$ 
7        if  $\text{Farbe}[N(v)] = \text{leer}$  oder  $\text{Farbe}[N(v)] = \neg \text{Farbe}[v]$  then do
8           $\text{Farbe}[N(v)] \leftarrow \neg \text{Farbe}[v]$ 
9           $\text{Queue.push}(N(v))$ 
10       else do
11         return  $G$  ist 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 anbietet. Diese Modifikation gilt es weiterhin genauer zu untersuchen, da nicht garantiert ist, dass die in dieser Arbeit verwendete Einschränkung für das Finden des Matchings M_1 ist die beste ist. Außerdem wurde nicht getestet, welchen Einfluss die Verwendung eines Maximum Matchings an dieser Stelle hätte. In Abbildung 3.4 zeigte sich, dass es scheinbar große Unterschiede in der Reduzierbarkeit der Graphen mit 3000 bis 4600 Knoten. 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 [?] gezeigt, bereits eine einfache Verringerung von k die Laufzeit eines Suchbaumalgorithmus halbiert.

6. Anhang

Tabelle 6.1: Statistiken über Graph₅

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 - Grad₁ - Reduktion

ljhawdlhawldkhawlkdhawlkhdalwkhhdalwkhhdwlkh

Literaturverzeichnis

- [1] M. R. Garey and D. S. Johnson, *Computers and Intractability : a guide to the theory of NP-completeness*. Freeman New York, 2007.
- [2] G. Valiente, *Algorithms on Trees and Graphs*. Springer, Berlin, Heidelberg, 2002.
- [3] R. M. Karp, “Reducibility among combinatorial problems,” *Complexity of Computer Computations*, 85-103, 1972.
- [4] 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.
- [5] K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, 2006.
- [6] J. Chen, I. A. Kanj, and W. Jia, “Vertex cover: further observations and further improvements,” *Journal of Algorithms 41*, 280 - 301, 2001.
- [7] 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.
- [8] M. Cygan, *Parameterized algorithms*. Springer Cham, 2015.
- [9] G. Nemhauser and L. E. Trotter, “Vertex packings: Structural properties and algorithms,” *Mathematical Programming 8*, 232-248.
- [10] 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.
- [11] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
- [12] 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.
- [13] N. Blum, “A new approach to maximum matching in general graphs,” *ICALP: Automata, Languages and Programming*, 586-597, 1990.
- [14] E. C. Sewell, S. H. Jacobson, and H. Kaul, “Reductions for the stable set problem,” *Algorithmic Operations Research Vol.6* 40-55, 2011.

- [15] 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.
- [16] F. Gurski, *Exakte Algorithmen für schwere Graphenprobleme*. Springer Heidelberg, 2010.
- [17] R. G. Downey and M. R. Fellows, *Fundamentals of parameterized complexity*. Springer London, 2013.
- [18] ———, *Parametrized Complexity*. Springer New York, 1997.

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