Ein monitorbarer reduktionsbasierter Approximationsalgorithmus für totale Knotenüberdeckung

Masterarbeit
Fach Informatik
Universität Trier

Vorgelegt von
Florian Kruschewski-Kursawe
Matrikelnummer 1023615
Robert-Schuman-Allee 12
54296 Trier
s4flkrus@uni-trier.de
Studiengang Informatik
Fachsemester: 5

1. Prüfer: Univ.-Prof. Dr. Henning Fernau
2. Prüfer: Univ.-Prof. Dr. Stefan Näher
Eingereicht am: 13.02.2019

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

In der Informatik begegnen wir häufig Optimierungsproblemen, welche sich vermutlich nicht effizient lösen lassen, sogenannte NP-Schwere Probleme. Für diese Probleme existieren bislang keine bekannten und tatsächlich funktionierenden Möglichkeiten, eine Lösung in Polynomialzeit zu finden.
Stattdessen können diese Probleme bisher nur in exponentiell wachsender Zeit exakt gelöst werden, wodurch sich zwar kleine Probleme in annehmbarer Zeit lösen lassen, diese annehmbare Zeit ist aber sehr schnell überschritten (bei wachsender Problemgröße). Hierbei sind diese nicht mehr in annehmbarer Zeit lösbaren Probleme jedoch keinesfalls unrealistisch groß, sondern genau genommen sogar in der Realität eher kleine Probleme. Betrachten wir beispielsweise Probleme auf Graphen und ein Straßennetz als solchen Graphen. Hierbei betrachten wir eine Straße als Kante und jede Kreuzung oder Sackgasse als einen Knoten im Graphen. Wollen wir ein NP-schweres Problem, beispielsweise auf der Straßenkarte eines Dorfes oder eines kleinen Städtchens lösen, so lässt sich dies oftmals in Minuten oder sogar Sekunden durchführen.
Betrachten wir jedoch bereits eine Großstadt, so kann die Lösung des Problems bereits einige Stunden oder Tage in Anspruch nehmen. Wollen wir dann sogar das Straßennetz eines ganzen Landes betrachten, so gelangen wir sehr schnell in den Bereich einiger Jahre zur exakten Lösung unseres Problems. Da unter anderem auch Probleme wie Routenplanung und ähnliches unter die Kategorie solcher Probleme fallen, wäre es natürlich nicht umsetzbar, wenn beispielsweise jedes mal bei der Nutzung eines Navigationsgerätes zunächst eine mehrjährige Berechnung notwendig würde, um eine optimale Route quer durch Deutschland zu berechnen.
Da uns solche Probleme in der Realität immer wieder begegnen, wurden in der Vergangenheit verschiedene Ansätze betrachtet, mit derartigen Problemen umzugehen. Einer dieser Ansätze ist, dass wir nicht zwangsweise eine optimale Lösung benötigen, sondern es uns ausreichen kann, eine Lösung zu finden, die „gut genug“ ist, dafür aber in deutlich kürzerer Zeit zu finden ist. Dieser Ansatz ist die sogenannte Approximation eines Problems. In dieser Arbeit wollen wir uns mit einer solchen Approximation beschäftigen.
Das Problem welches wir in dieser Arbeit betrachten wollen, ist das sogenannte Total Vertex Cover Problem. Betrachten wir dazu folgende Situation:
Nehmen wir an, in einer Stadt versucht die Polizei Beamte so zu verteilen, dass für jede Straße mindestens ein Beamter an mindestens einem ihrer Enden ist. Betrachten wir das Straßennetz dieser Stadt nun als Graphen, so ist die Bedingung, dass für jede Kante gilt, dass auf mindestens einem ihrer Knoten ein Beamter positioniert ist. Die einfachste Lösung wäre offensichtlich, auf jede Kreuzung (Knoten) einen Beamten zu positionieren. Aus Kostengründen sollen aber so wenig Beamte wie möglich eingesetzt werden. Es ist also nötig, die minimale Menge der Knoten zu finden, für die gilt, dass

für jede Kante mindestens einer ihrer Knoten in dieser Menge ist. Dies ist eine sogenannte Knotenüberdeckung. Weiterhin soll sichergestellt werden, dass kein Beamter von anderen Beamten isoliert ist, also für jeden Beamten muss gelten, dass sich mindestens ein Kollege höchstens eine Straße entfernt von ihm befindet, um sicherzustellen, dass schnellstmöglich Unterstützung möglich ist. Als Graphenproblem ausgedrückt bedeutet dies also, dass wir eine Knotenüberdeckung suchen, bei der zusätzlich gilt, dass für jeden Knoten in der Überdeckung auch mindestens einer seiner Nachbarn ebenfalls in der Überdeckung liegt. Dies ist das sogenannte „Totale Knotenüberdeckungsproblem“. Wir wollen uns im folgenden mit der Frage beschäftigen, ob ein Polynomialzeitalgorithmus für die Approximation dieses Problems existiert.

1.2 Ziel und Aufbau der Arbeit

In der folgenden Arbeit wollen wir einen **monitorisierbaren Approximationsalgorithmus** für das Totale Knotenüberdeckungsproblem (Total vertex cover problem) entwickeln. Das Ziel hierbei ist, einen Algorithmus zu entwickeln, welcher in polynomieller Laufzeit das gegebene Problem approximiert und dabei in der Lage ist, seine eigene Güte selbstständig zu überwachen. Hierbei betrachten wir Graphen mit beschränktem Durchschnittsgrad, wobei wir zunächst einen funktionierenden Algorithmus für Graphen mit Durchschnittsgrad ≤ 4 betrachten.
Für diesen Graphen erläutern wir die wichtigen Reduktionsregeln für Knoten mit kleinem Grad und zeigen die Korrektheit des Algorithmus für diese Menge von Graphen.
Dann betrachten wir einige Anpassungen für den Algorithmus, um zum einen Möglichkeiten zu finden, wie wir unseren Algorithmus zu Lasten der Laufzeit verbessern können, und zum anderen, wie wir den Algorithmus in Kaufnahme einer schlechteren Approximation für Graphen mit einem beliebigen Durchschnittsgrad anpassen können. Hierbei behalten wir immer im Auge, dass der Algorithmus in der Lage bleibt, sich selbst zu überwachen.
Nach der theoretischen Betrachtung des Algorithmus kommen wir dann zur praktischen Betrachtung verschiedener Aspekte des Algorithmus sowie einer tatsächlichen Implementierung des Algorithmus zu Testzwecken.
Zum Abschluss werden wir dann noch eine Reihe von speziellen Graphen und Graphenklassen betrachten und untersuchen welche Ergebnisse unser Algorithmus in diesen Fällen erzielt.

1.3 Benötigte Begriffe

Hier wollen wir auf einige Begriffe eingehen, welche im Verlaufe dieser Arbeit verwendet werden und sie insbesondere im Zusammenhang mit dieser Arbeit definieren. Beginnen wir zunächst mit der grundlegenden Definition eines Graphen:
Ein **Graph** $G = (V, E)$ ist eine Struktur, bestehend aus einer Menge von **Knoten** V (Vertices) und einer Menge von **Kanten** E (Edges). Eine Kante $e = (v1, v2)$ ist hierbei eine Struktur die zwei Knoten $v1, v2 \in V$ miteinander verbindet. Zwei Knoten $v1, v2$

bilden ein sogenanntes **Knotenpaar**, wenn zwischen ihnen eine Kante $e = (v1, v2)$ existiert.
Die **Nachbarschaft** N eines Knotens $v \in V$ ist die Menge aller Knoten $v2$, für die eine Kante $e = (v, v2)$ existiert. Gilt für zwei Knoten $v1, v2$, dass die Nachbarschaft $N1$ von $v1$ und $N2$ von $v2$ exakt gleich ist, so nennt man $v2$ einen **Zwilling** von $v1$.
Der **Grad** g eines Knotens v beschreibt die Anzahl der Kanten, die an v anliegen, also $g = |N|$ für N Nachbarschaft von v. Der **minimale** beziehungsweise **maximale Grad** eines Graphen ist dann der niedrigste beziehungsweise höchste Grad eines Knotens, welcher sich in diesem Graphen befindet.
Der **Durchschnittsgrad** Dg des Graphen ist dann der Durchschnitt des Grades aller Knoten im Graphen.
Im Folgenden werden wir **durchschnittsgraderhaltende Algorithmen** behandeln, daher betrachten wir das Lemma 7 aus der Arbeit von [1]:

In einem Graphen G mit Durchschnittsgrad Dg gilt:
wenn man x Knoten aus dem Graphen entfernt und dazu $\frac{Dg}{2}x$ Kanten, dann hat der restliche Graph G' einen Durchschnittsgrad $D'g \leq Dg$.

Insbesondere dieser Durchschnittsgrad eines Graphen ist für die vorliegende Arbeit relevant, da wir Graphen mit beliebig hohem maximalem Grad betrachten können, ohne dass diese Knoten für uns ein Problem darstellen, solange der Durchschnittsgrad klein genug ist. Bei Algorithmen, bei denen die Güte hingegen vom maximalen Grad des Graphen abhängt, können extreme Ausbrüche zu einer deutlich schlechteren Abschätzung führen. Diese extremen Knoten sind allerdings eher eine Ausnahme. Ein Beispiel für eine solche Situation sind soziale Netzwerke. Hier haben wir oftmals einige wenige Mitglieder mit extremer Anzahl an "Freunden"/Followern. Allerdings haben wir eine deutliche Mehrzahl an Accounts, die nur sehr wenige Freunde hinzugefügt haben, oder auch nur wegen einzelner Personen angelegt wurden. Daher ist der Durchschnitt der Anzahl an Beziehungen zwischen zwei Accounts im Vergleich zum Maximalwert extrem gering. Wenn wir nun Accounts als Knoten und deren Beziehungen als Kanten darstellen, so haben wir einen Graphen mit extremem Maximalgrad, aber sehr niedrigem Durchschnittsgrad.
Bei den Regeln, welche wir im Folgenden für die Bearbeitung solcher Graphen verwenden, handelt es sich um sogenannte **Reduktionsregeln**. Das bedeutet, wenden wir eine solche Reduktionsregel auf einen Graphen $G = (V, E)$ an, so wählt die Regel eine Menge von Knoten X und deren anliegende Kante Y aus und entfernt diese aus dem Graphen (ein Teil von X wird in die Überdeckung gelegt). Dadurch entsteht ein Graph $G' = (V', E')$ mit $V' = V \setminus X$ und $E' = E \setminus Y$. Auf G' kann nun die nächste Reduktionsregel angewendet werden. Ein **Reduktionsalgorithmus** terminiert, wenn nach Anwendung einer Reduktionsregel gilt, dass $V, E = \emptyset$.
Die Grundidee für den Algorithmus, welchen wir in dieser Arbeit betrachten, ist ein **Durchschnittsgrad erhaltender Reduktionsalgorithmus**. Das bedeutet, wir wenden solange Reduktionsregeln an, bis entweder keine der Regeln mehr anwendbar ist oder bis der gesamte Graph vollständig bearbeitet wurde, sich also keine Knoten mehr im Graphen befinden. Dabei muss jederzeit sichergestellt werden, dass der Durchschnittsgrad

unter einem gewissen Wert bleibt, oder dass dieser Wert durch spätere Regeln wieder erreicht wird. Führen wir also eine Regel aus, welche den Durchschnittsgrad erhöht, so müssen wir garantieren, dass eine andere Regel ausgeführt wird, die den Durchschnittsgrad wieder senkt, oder der Graph vollständig bearbeitet wurde, was die Ausführung der senkenden Regel überflüssig macht.

Kommen wir nun zur formalen Definition unseres Problems:
Für einen Graphen $G = (V, E)$ suchen wir eine minimale Menge U von Knoten mit $U \subseteq V$, so dass gilt, für jede Kante $e \in E$ mit $e = (v1, v1)$ und $v1, v2 \in V$ ist $v1 \in U$ und/oder $v2 \in U$. Zusätzlich muss gelten, dass für jeden Knoten $v1 \in U$ ein Knoten $v2 \in U$ existiert, so dass eine Kante $e = (v1, v2)$ mit $e \in E$ existiert. Umgangssprachlich bedeutet dies, dass für jede Kante von G mindestens einer der beiden Knoten in der Überdeckung liegt, und für jeden Knoten in der Überdeckung gilt, dass auch mindestens einer seiner Nachbarn im Ursprungsgraphen ebenfalls in der Überdeckung liegt. Dies nennt man eine **Totale Knotenüberdeckung**.
Für eine natürliche Zahl K kann man nun also die Frage stellen, ob für einen Graphen G eine Totale Knotenüberdeckung U existiert mit $|U| \leq K$. Diese Frage ist das **Totale Knotenüberdeckungsproblem** und dieses ist NP-Schwer. Die Frage lässt sich also nicht in Polynomieller Zeit beantworten, da wir nicht in der Lage sind, in polynomieller Zeit zu garantieren, dass eine gefundene Totale Knotenüberdeckung minimal ist.
Aus diesem Grund werden wir im Folgenden einen sogenannten **Approximationsalgorithmus** betrachten, also einen Algorithmus, welcher für das gegebene Problem eine Lösung berechnet, welche nicht zwangsweise minimal, aber möglichst gut ist. Hierbei erreicht der Algorithmus einen **Approximationsfaktor**. Dieser Faktor gibt an, um wie viel die gefundene Lösung höchstens von einer minimalen Lösung abweicht. Eine Lösung mit 150 Knoten und einem Approximationsfaktor von $1,5$ bedeutet also, dass eine minimale Lösung zwischen 100 und 150 Knoten enthalten muss.
Der Approximationsfaktor berechnet sich hierbei aus der Anzahl der Knoten in der approximierten Überdeckung geteilt durch die Anzahl der gefundenen minimalen Knoten. Ein minimaler Knoten ist hierbei ein Knoten, für den wir sicher sagen können, dass eine minimale Überdeckung existiert, welche genau diesen Knoten enthält, oder dass er Teil einer Menge von Knoten ist, von denen eine Mindestanzahl von Knoten in einer minimalen Überdeckung liegen muss. Legen wir beispielsweise ein Paar von Knoten in die Überdeckung, so muss gemäß der Bedingungen für die Knotenüberdeckung mindestens einer der beiden Knoten in einer vollständigen Überdeckung liegen, also auch in einer minimalen Überdeckung. Wir können also einen dieser Knoten als minimal betrachten, auch wenn wir nicht sicher sagen können, welcher von beiden Knoten der tatsächliche minimale Knoten ist. Das Gegenstück dazu sind approximierte Knoten. Dies sind Knoten, die in die Überdeckung gelegt werden, wobei dann aber noch nicht sicher ist, ob diese Knoten tatsächlich auch in einer minimalen Überdeckung liegen müssen. In unserem vorherigen Beispiel eines Knotenpaares wäre also der zweite Knoten dann ein approximierter Knoten, da wir nicht sicher sagen können, ob dieser Knoten nicht doch in einer minimalen Überdeckung liegen muss, beispielsweise um die Bedingung der Totalen Knotenüberdeckung zu erfüllen.

2 Total vertex cover Algorithmus

Wir betrachten im Folgenden einen Algorithmus für eine totale Knotenüberdeckung, welcher eine Erweiterung des Algorithmus von [1] für das allgemeine Knotenüberdeckungsproblem darstellt. Wie auch bei [1] betrachten wir zunächst das Problem für Graphen mit Durchschnittsgrad $d_{avg} = 4$ und Minimalgrad $\delta = 2$.
Da wir für die totale Knotenüberdeckung nicht alle Fälle von Knoten mit Grad 1 und Grad 2 minimal lösen können, müssen wir einen weiteren Reduktionsschritt in Kauf nehmen. Unser Maximalwert für die Approximation des Problems ist daher Faktor $\frac{5}{3}$ beziehungsweise 1.66. Wir können also für jeden minimalen Knoten bis zu zwei Knotenpaare in die Überdeckung legen. Dieser Faktor wird jedoch nur in extremen Grenzfällen erreicht (siehe dazu das Kapitel 5).
Da der Algorithmus in der Lage ist, für jede angewendete Regel die Zahl der minimalen und der approximierten Knoten zu bestimmen, kann am Ende jedes Durchlaufes auch der erreichte Approximationsfaktor für das bestimmte Problem genauer abgeschätzt werden.
Später werden wir auch die Möglichkeiten des Algorithmus für Graphen mit Minimalgrad $\delta = 1$ bzw. Durchschnittsgraden $d_{avg} > 4$ betrachten. Diese lassen sich jedoch nicht mehr garantiert lösen, sondern nur unter bestimmten Bedingungen.
Wie beim Algorithmus von [1] verwalten wir eine Bank, um sicherzustellen, dass der Approximationsfaktor eingehalten wird. Hierbei darf die Bank auch ins Negative gehen, also eine Art Kredit am Approximationsfaktor aufgenommen werden, solange sichergestellt ist, dass dieser später wieder ausgeglichen wird. Solange die Bank einen positiven Wert hat, gibt dieser an, wie viele Knotenpaare wir in die Knotenüberdeckung legen dürfen, ohne den angepeilten Approximationsfaktor zu überschreiten.
Zusätzlich verwalten wir eine Hilfsliste. Diese enthält die Knoten, die zwar in eine minimale Knotenüberdeckung müssen, für die aber die Totalitätsbedingung noch nicht erfüllt ist. Das heißt, wir überwachen diese Knoten im weiteren Verlauf des Algorithmus in Hinblick auf die Frage, ob irgendeiner ihrer Nachbarn auch in die Knotenüberdeckung gelegt wird und entfernen dementsprechend den Knoten aus der Hilfsliste, da die Totalitätsbedingung für diesen Knoten erfüllt ist. Oder wir wählen einen geeigneten Nachbarn aus und legen diesen in die Knotenüberdeckung. Dies lässt sich leider nicht immer exakt erfüllen, weswegen Knoten in der Hilfsliste zunächst nicht als minimale Regel gelten können. Dies wird im Punkt ***Behandlung der Hilfsliste*** näher erläutert.
Außerdem gilt für jeden Knoten im Graphen, dass er weiß, ob einer seiner Nachbarn bereits in die Knotenüberdeckung gelegt wurde. Liegt bereits ein Nachbar eines Knoten in der Knotenüberdeckung, so bedeutet dies, dass dieser Knoten in die Knotenüberdeckung gelegt werden darf, ohne dass ein weiterer Nachbar ebenfalls hineingelegt werden muss.
Dies beeinflusst einige der Regeln, da diese immer sicherstellen müssen, dass die Totalitätsbedingung für alle Knoten, die in die Überdeckung gelegt werden, erfüllt wird.

Daher kann es unter Umständen sein, dass Knoten approximiert in die Überdeckung gelegt werden müssen, um dies sicherzustellen, während Knoten, die bereits *abgedeckt* wurden, ohne Probleme auch einzeln betrachtet und minimal gelöst werden können. Wir werden dies im Folgenden dadurch ausdrücken, dass sich ein Knoten im Zustand *nicht abgedeckt*, oder *abgedeckt* befinden kann. Abgedeckt bedeutet dabei, dass irgendeiner seiner Nachbarn im Ursprungsgraphen entweder in der Knotenüberdeckung liegt oder in der Hilfsliste, da auch das garantiert, dass dieser Nachbar später in die Knotenüberdeckung gelegt wird. Ein Knoten in der Hilfsliste wird sogar automatisch in die Knotenüberdeckung gelegt, sobald ein Nachbar in die Knotenüberdeckung gelegt wird, da diese beiden Knoten ja nun gegenseitig die Totalitätsbedingung erfüllen.

Betrachten wir nun den eigentlichen Algorithmus. Anders als beim ursprünglichen Algorithmus von[1] können wir Fälle, in denen Knoten vom Grad 1 erzeugt werden, nicht direkt innerhalb der Regeln abfangen, sondern müssen nach Anwendung bestimmter Regeln zunächst die Grad-1-Regeln erschöpfend anwenden. Dies passiert jedoch nur im Umfeld der ursprünglich angewendeten Regel und wird als Teil dieser Regel aufgefasst. Die Regeln für die einzelnen Fälle gehen immer davon aus, dass kein anderer Knoten im Graphen existiert, welcher eine vorherige Regel erfüllt. In der Theorie sollte also immer solange Regel 1 angewendet werden, wie ein Knoten im Graph existiert, welcher Regel 1 erfüllt. Dies gilt analog für Regel 2 und folgende.
In der Praxis ist es jedoch möglich, die Regeln auch in geänderter Reihenfolge anzuwenden, solange sichergestellt wird, dass im Umfeld der angewendeten Regel kein Knoten existiert, welcher eine vorherige Regel erfüllt. Ist dies der Fall, so betrachten wir stattdessen diesen Knoten. Das Umfeld der Regel sind hierbei alle Knoten, welche durch die Regel aus dem Graphen entfernt werden sowie alle Knoten, die im Graphen verbleiben, aber zu einem der entfernten Knoten benachbart sind. Knoten, die durch eine etwaige folgende Anwendung der Grad-1-Regeln beeinflusst werden, müssen hierbei nicht beachtet werden.
Der Ablauf des Algorithmus sieht also aus wie folgt:

1. Existiert ein Knoten mit Grad 1, so wende die entsprechende Grad-1-Regel an, passe die Bank entsprechend an und beginne wieder bei Punkt 1, ansonsten gehe zu Punkt 2.

2. Existiert ein Knoten mit Grad 2, so wende die entsprechende Grad-2-Regel an, passe die Bank entsprechend an und beginne wieder bei Punkt 1, ansonsten gehe zu Punkt 3.

3. Existiert ein Knoten mit Grad 3, so wende die entsprechende Grad-3-Regel an, passe die Bank entsprechend an und beginne wieder bei Punkt 1, ansonsten gehe zu Punkt 4.

4. Existiert kein Knoten mit Grad < 4, aber ein Knoten mit Grad > 4, so wende die Regel für den Ausgleichsschritt an, passe die Bank entsprechend an und beginne wieder bei Punkt 1, ansonsten gehe zu Punkt 5.

5. Existieren nur noch Knoten mit Grad 4, so wende die Grad-4-Regel an, passe die Bank entsprechend an und beginne wieder bei Punkt 1, ansonsten terminiere den Algorithmus.

Für Graphen mit Durchschnittsgrad $d_{avg} = 4$ und Minimalgrad $\delta = 2$ terminiert der Algorithmus immer mit einem leeren Graphen. Wir werden später für andere Arten von Graphen noch andere Terminierungsbedingungen betrachten.
Kommen wir nun zu den im Einzelnen verwendeten Regeln. Wir zeigen hierbei beispielhafte Ausschnitte aus dem Graphen, um die Fälle zu verdeutlichen. Dabei bedeuten gepunktete Linien, dass der Knoten weitere Nachbarn besitzen muss, die aber für den Fall direkt keine Bedeutung haben, also nur mehr Nachbarn vorhanden sein müssen als in dem gezeigten Ausschnitt des Graphen vorhanden sind. Durchgezogene Linien ohne Knoten am Ende bedeuten, dass genau die vorhandene Anzahl an Kanten vorhanden sein muss, aber die restlichen Knoten für den Fall nicht von Bedeutung sind. Außerdem ist bei einigen Fällen das Vorhandensein von Knoten-Zwillingen wichtig. Diese werden in den Beispielen durch Knoten mit dem selben Namen dargestellt und sind so zu verstehen, dass beliebig viele solcher Knoten vorhanden sein können.

2.1 Einige Basisregeln

Zunächst betrachten wir einige grundlegende Regeln für die Knotenüberdeckung. Diese finden keine direkte Anwendung im Algorithmus, sondern bilden nur die Basis für andere Regeln.

Isolierte Knoten Regel: Ein Knoten vom Grad 0 ist für die Knotenüberdeckung irrelevant und wird unter keinen Umständen in die Knotenüberdeckung gelegt werden. Daher kann ein solcher Knoten einfach aus dem Graphen entfernt werden.
Dies findet insbesondere bei Grad-1-Regeln Anwendung, da, wenn der Nachbar eines Grad 1 Knoten in die Knotenüberdeckung gelegt wird, alle seine Kanten entfernt werden und dementsprechend ein isolierter Knoten erzeugt wird.

Knotenpaar Regel: Existiert für zwei Knoten x und y eine Kante $e = (x, y)$, so muss mindestens einer der beiden Knoten in einer minimalen Knotenüberdeckung enthalten sein. Legen wir beide Knoten x und y in die Knotenüberdeckung, so ist also mindestens einer der beiden Knoten minimal gelöst.

Dreiecksregel: Existieren für drei Knoten x, y und z die Kanten $e1 = (x, y), e2 = (y, z)$ und $e3 = (x, z)$, so müssen mindestens zwei der drei Knoten in einer minimalen Knotenüberdeckung enthalten sein. Legen wir also alle drei Knoten in die Knotenüberdeckung, so sind mindestens zwei der Knoten minimal gelöst.

Totalitätsbedingung: Damit eine Knotenüberdeckung total ist, muss für jeden Knoten

in der Überdeckung gelten, dass ein Nachbar existiert, welcher ebenfalls in der Knotenüberdeckung liegt. Die Totalitätsbedingung für einen Knoten x ist also erfüllt, wenn ein Knoten y existiert, welcher im ursprünglichen Graphen benachbart war zu x und y in der Knotenüberdeckung liegt.

2.2 Bank

Im Folgenden werden wir bei den Fällen auch jeweils die Änderungen an der Bank betrachten. Die Bank dient hierbei zur Überwachung des Approximationsfaktors. Hierbei wird sichergestellt, dass pro fünf Knoten in der Überdeckung mindestens drei Knoten dieser Knoten garantiert in der minimalen Überdeckung liegen müssen, damit der Faktor $\frac{5}{3}$ nicht überschritten wird. Hierbei bedeutet eine Erhöhung der Bank um 1, dass genau 1 beliebiges Knotenpaar in die Überdeckung gelegt werden darf. Da für ein Knotenpaar gilt, dass mindestens einer der beiden Knoten in die Überdeckung gelegt werden muss, ist also einer der beiden Knoten dann als minimal zu betrachten. Es handelt sich also um einen Faktor-2-Approximationsschritt. Für die Erhöhung der Bank gibt es zwei Möglichkeiten:

Einzelner Knoten: Jeder Knoten, der minimal gelöst wird, wird als einzelner Knoten betrachtet. Um unter den erwünschten $\frac{5}{3}$ zu bleiben, können also noch zwei Knotenpaare in die Überdeckung gelegt werden. Dann wurden 5 Knoten in die Bank gelegt, von denen mindestens 3 (der minimal gelöste Knoten sowie jeweils ein Knoten des Knotenpaares) in der Überdeckung liegen müssen. Daher erhöhen wir für jeden minimalen Knoten die Bank im Folgenden um 2. Werden mehrere Knoten gleichzeitig minimal gelöst, so wird für jeden Knoten die Bank um 2 erhöht. Wird also bei einer Regel die Bank beispielsweise um 6 erhöht, so wurden drei Knoten von der Regel minimal gelöst.

Knoten-Tripel: Von einigen Regeln werden Tripel von Knoten in die Überdeckung gelegt, die jedoch nicht alle minimal gelöst wurden. Dies sind beispielsweise Dreiecksregeln, aber auch andere Regeln können eine solche Situation erzeugen. Allerdings ist in diesen Fällen immer sicher gestellt, dass mindestens zwei der Knoten in der Überdeckung liegen müssen. Im Sinne der Bank können wir diese Situationen also so betrachten, als wäre ein minimaler Knoten und bereits ein Knotenpaar in die Überdeckung gelegt worden. Daher erhöhen wir die Bank nur noch um 1, da wir mit einem weiteren Knotenpaar wieder den Faktor $\frac{5}{3}$ erreichen. Es wird nie mehr als ein Tripel von einer Regel in die Überdeckung gelegt. Daher wird die Bank in solchen Fällen immer nur um 1 erhöht, eine größere Erhöhung bedeutet immer, dass Knoten minimal gelöst werden.

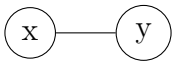
Das Hinzufügen von Paaren zur Knotenüberdeckung geschieht vor allem im Ausgleichsschritt, welcher im Abschnitt 2.9 näher erläutert wird. Allerdings gibt es auch Regeln, die ebenfalls Paare in die Überdeckung legen. Diese könnten daher dazu führen, dass die Bank auch ins Negative geht, wenn diese Regeln angewendet werden, bevor andere Regeln angewendet wurden. Allerdings wird in diesen Fällen auch immer sichergestellt,

dass die Bank später wieder ausgeglichen wird. Dies wird bei den einzelnen Fällen erläutert. Weiterhin ist dieser Ausgleich insbesondere bei der Betrachtung des Erhalts des Durchschnittsgrades entscheidend. Dies sehen wir ebenfalls Kapitel 2.10.
Hat die Bank einen positiven Wert, so muss dieser nicht zwangsweise ausgeglichen werden. Genau genommen bedeutet eine positive Bank, dass unser Approximationsfaktor aktuell besser ist als der Zielfaktor. Wenn der Graph vollständig gelöst wurde, haben wir also in diesem Fall einen besseren Faktor als $\frac{5}{3}$ erreicht (Siehe hierzu auch das Kapitel 2.11).

2.3 Grad 1

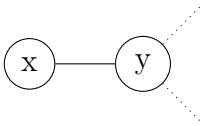
Wir betrachten nun die Regeln für Knoten mit Grad 1. Hierbei wird immer ein Knoten x mit Grad 1 betrachtet. Der Zustand von x ist, sofern nicht explizit angegeben, irrelevant.

Fall 1:



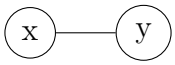
Der Knoten x ist im Zustand abgedeckt, der Nachbarknoten y hat ebenfalls Grad 1. Lege Knoten x in die Knotenüberdeckung und entferne den Knoten y sowie die Kante $e = (x, y)$. Erhöhe die Bank um 2.
Beweis: Da die Kante $e = (x, y)$ besteht, muss mindestens einer der beiden Knoten in die Knotenüberdeckung, und da bei Knoten x die Totalitätsbedingung bereits erfüllt ist, wählt man diesen Knoten. Würde man den Knoten y wählen, so müsste der Knoten x ebenfalls wieder in die Knotenüberdeckung gelegt werden, um die Totalitätsbedingung für y zu erfüllen, dies wäre also nicht mehr minimal.

Fall 2:



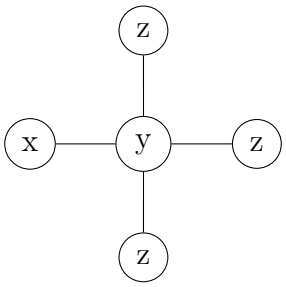
Der Nachbarknoten y ist im Zustand *abgedeckt*. Lege den Knoten y in die Knotenüberdeckung und entferne alle Nachbarn von y mit Grad 1 sowie alle anliegenden Kanten. Alle anderen Nachbarn von y erhalten Zustand *abgedeckt*. Erhöhe die Bank um 2.
Beweis: Da für den Knoten y die Totalitätsbedingung erfüllt ist und durch y in der Knotenüberdeckung sichergestellt wird, dass x nicht mehr in die Knotenüberdeckung muss, existiert immer eine minimale Lösung, welche y enthält.

Fall 3:



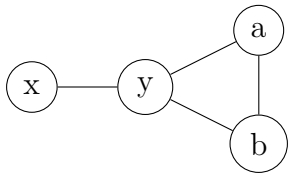
Der Nachbarknoten y hat Grad 1 und beide Knoten x, y sind im Zustand *nicht abgedeckt*. Lege die Knoten x und y in die Knotenüberdeckung, entferne die Kante $e = (x, y)$ und erhöhe die Bank um 4.
Beweis: Da die Knoten x und y benachbart sind, muss mindestens einer der beiden Knoten in die Knotenüberdeckung. Da die beiden Knoten jedoch keinen weiteren Nachbarn haben, welcher in die Knotenüberdeckung gelegt werden könnte, muss auch immer der andere Knoten in die Knotenüberdeckung gelegt werden, um die Totalitätsbedingung für einander zu erfüllen.

Fall 4:



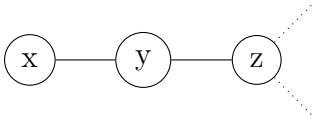
Der Nachbarknoten y hat Grad > 1 und es gilt: Für alle Nachbarn z von y ist Grad von $z = 1$. Lege x und y in die Knotenüberdeckung und entferne alle Knoten z sowie alle anliegenden Kanten. Erhöhe die Bank um 4.
Beweis: wenn Knoten y nicht in die Knotenüberdeckung kommt, so müssen alle seine Nachbarn in die Knotenüberdeckung, also mindestens 2 Knoten. Für diese muss wieder die Totalitätsbedingung erfüllt werden, womit y wieder in die Knotenüberdeckung müsste, da kein weiterer Nachbar existiert, welcher die Totalitätsbedingung für den Grad-1-Knoten erfüllen kann (Eine Ausnahme läge vor, wenn genau zwei Grad-1-Knoten existieren und beide im Zustand abgedeckt sind. Allerdings müssten auch dann wieder zwei Knoten in die Überdeckung gelegt werden, die Lösung wäre also nicht besser, als wenn y und einer der beiden Knoten in die Überdeckung gelegt wird). Also ist y minimal, benötigt allerdings einen weiteren Knoten, um die Totalitätsbedingung zu erfüllen. Da alle Nachbarn vom Grad 1 sind, also keine weiteren Nachbarn haben, beeinflusst die Wahl, welcher Knoten als Partner in die Knotenüberdeckung kommt, keinen anderen Knoten im Graphen. Daher existiert für jeden Nachbarn z von y eine minimale Lösung, welche z und y enthält. Wir wählen daher der Einfachheit halber den ursprünglichen Knoten x.

Fall 5:



Der Nachbarknoten y hat Grad > 2 und es existieren zwei Nachbarn a, b von y für die gilt, dass die Kante $e = (a, b)$ existiert. Lege Knoten y in die Knotenüberdeckung, entferne alle Nachbarn von y mit Grad 1 sowie alle anliegenden Kanten. Alle Nachbarn von y erhalten Zustand *abgedeckt*, erhöhe die Bank um 2.
Beweis: Da eine Kante zwischen a und b existiert, muss mindestens einer der beiden Knoten schlussendlich in die Knotenüberdeckung. Daher ist garantiert, dass die Totalitätsbedingung für den Knoten y im späteren Verlauf des Algorithmus erfüllt wird. Damit kann y direkt in die Knotenüberdeckung gelegt werden.

Fall 6:



Der Nachbarknoten y hat genau einen Nachbarn z mit Grad > 1 oder genau einen weiteren Nachbarn z mit beliebigem Grad. Lege y, z in die Knotenüberdeckung, entferne alle Nachbarn von y, z mit Grad 1 und alle anliegenden Kanten. Alle anderen Nachbarn erhalten Zustand *abgedeckt*. erhöhe die Bank um 4.
Beweis: Der Knoten z deckt y ab, damit gilt für y wieder Fall 2. Da y einen Nachbarn benötigt, um die Totalitätsbedingung zu erfüllen, und alle anderen Nachbarn außer z definitiv nicht in die Knotenüberdeckung müssen, wenn y in der Knotenüberdeckung ist und *abgedeckt* wird, existiert immer eine minimale Lösung, welche auch z enthält.

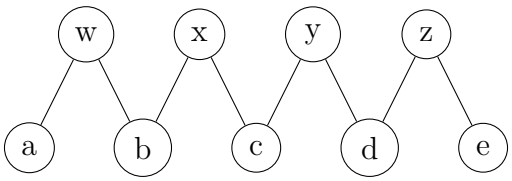
Fall 7:
Keiner der vorherigen Fälle gilt, aber der Nachbar y hat einen Nachbarn z und z liegt in einer Menge von Nachbarn eines Knoten in der Hilfsliste. Lege die Knoten y und z in die Knotenüberdeckung, entferne alle Nachbarn vom Grad 1 sowieso alle anliegenden Kanten. Alle anderen Nachbarn von y und z erhalten Zustand *abgedeckt* (insbesondere wird der Nachbar von z aus der Hilfsliste in die Knotenüberdeckung gelegt), erhöhe die Bank um 0,5.
Beweis: Wenn x in die Knotenüberdeckung käme, müsste ebenfalls y in die Knotenüberdeckung, um die Totalitätsbedingung für x zu erfüllen. Daher existiert immer eine minimale Lösung, welche y enthält. Da z für zwei Knoten, y und den Knoten aus der Hilfsliste, die Totalitätsbedingung erfüllt, können wir diesen Knoten ebenfalls in die

Knotenüberdeckung legen, da damit zwei minimale Knoten ebenfalls in die Knotenüberdeckung gelegt werden. Da der Knoten z jedoch nicht in der Knotenüberdeckung liegen muss, ist dieser nur eine Approximation. Wir legen also drei Knoten in die Knotenüberdeckung, von denen zwei minimal sind und könnten dementsprechend die Bank um 1 erhöhen. Da jedoch bereits 0,5 in die Bank gelegt wurden(siehe Fall 8), legen wir hier weitere 0,5 in die Bank. Für eine genauere Erläuterung, warum der Knoten z keine exakte Lösung ist, siehe das Kapitel Behandlung der Hilfsliste.

Fall 8:
Keiner der vorherigen Fälle gilt. Lege den Nachbarn y in die Hilfsliste und entferne alle Nachbarn vom Grad 1. Alle anderen Nachbarn erhalten Zustand *abgedeckt*. Entferne alle anliegenden Kanten und erhöhe die Bank um 0,5.
Beweis: Siehe Fall 7. Da wir mehrere solcher Knoten erhalten können, bevor diese weiter behandelt werden, erhöhen wir die Bank um 0,5, da pro zwei Knoten maximal ein weiterer Knoten approximiert werden muss, um diese Knoten abzudecken. Je nachdem wie der Knoten später behandelt wird, wird die Bank weiter erhöht, aber wir können bereits mit dieser Mindestmenge in der Bank arbeiten.
Anmerkungen:
Bei den Grad-1-Regeln ist es irrelevant, ob sie Nachbarn in der Hilfsliste haben. Wir versuchen soweit möglich, abgedeckte Knoten zu wählen (liegt ein Nachbar in der Hilfsliste, so ist der Knoten auch im Zustand *abgedeckt*). Wenn ein abgedeckter Knoten trotzdem entfernt wird, so ist es immer ein Grad 1 Knoten. Daher wollen wir, wenn möglich, andere Knoten wählen, um sie in die Knotenüberdeckung zu legen. Der Fall 8 deckt damit alle Fälle ab, die nicht von vorherigen Fällen abgedeckt wurden.

2.4 Behandlung der Hilfsliste

Das Ziel der Hilfsliste ist es, sicherzustellen, dass einzelne Knoten einen Partner erhalten, um die Totalitätsbedingung für diese Knoten zu erfüllen. Diese einzelnen Knoten entstehen durch Regeln, durch die sichergestellt ist, dass sie minimal gelöst sind, also eine Minimale Knotenüberdeckung existiert, welche diese Knoten enthält. Da die Totalitätsbedingung für diese Knoten gelten muss, muss auch mindestens einer der Nachbarn eines solchen minimalen Knotens in der Überdeckung liegen. Ein Knoten wird also aus der Hilfsliste entfernt, sobald einer seiner Nachbarn in die Überdeckung gelegt wird. Würde am Ende des Durchlaufes des Algorithmus kein Nachbar durch irgend eine andere Regel in die Überdeckung gelegt, so gilt, dass für jeden Nachbarn x eines Knoten in der Hilfsliste eine Überdeckung existiert, welche x enthält. Wir können jedoch nicht mit Sicherheit feststellen, welcher dieser Nachbarn eine minimale Knotenüberdeckung bildet. Betrachte hierzu folgenden Graphen:



Dies ist eine sehr vereinfachte Variante zur Veranschaulichung. Die Knoten w, x, y und z sind minimal gelöste Knoten in der Hilfsliste. In diesem Fall ist durch simples "drauf schauen" erkennbar, dass die Knoten b und d eine minimale Lösung wären, da damit alle 4 Knoten aus der Hilfsliste abgedeckt wären. Diese Knoten allerdings tatsächlich durch den Algorithmus finden zu lassen, ist eine Variante des sogenannten „hitting-set Problems". Dies ist jedoch ebenfalls ein NP-schweres Problem, also können wir nicht annehmen, eine minimale Lösung effizient finden zu können. Es ist also möglich, dass eine Heuristik zur Lösung zunächst den Knoten c findet und damit die Knoten x und y abdeckt. Allerdings haben dann die Knoten w und z nur noch Nachbarn, welche zu keinem anderen Knoten in der Hilfsliste benachbart sind. Daher müssen zwei weitere Knoten in die Überdeckung gelegt werden. Wir müssen daher Lösungen, welche einen Knoten auswählen, der zwei oder mehr Knoten verbindet, als approximiert annehmen. Da wir maximal einen Knoten für jeweils zwei Knoten in der Hilfsliste hinzufügen, ist dies eine $\frac{3}{2}$ Approximation. Anders ist die Situation, wenn keiner der Nachbarn eines Knotens aus der Hilfsliste ebenfalls Nachbar eines anderen Knoten in der Hilfsliste ist. In diesem Fall ist egal, welcher der Nachbarn in die Knotenüberdeckung gelegt wird, da auf jeden Fall einer hineingelegt werden muss, um die Totalitätsbedingung für den Knoten in der Hilfsliste zu erfüllen. Daher existiert für jeden Nachbarn x eine minimale Knotenüberdeckung, welche x enthält, und dies ist daher eine minimale Regel. Dies führt uns zu drei Möglichkeiten, wie ein Knoten x aus der Hilfsliste bearbeitet werden kann.

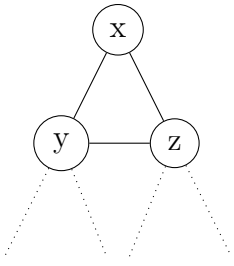
Fall 1:
Ein Nachbar y von x wird durch eine andere Regel in die Überdeckung gelegt oder soll in die Hilfsliste gelegt werden. Da x und y damit gegenseitig die Totalitätsbedingung erfüllen, entferne x aus der Hilfsliste und lege x und y in die Überdeckung. Wir können in diesem Fall die Bank um 1,5 beziehungsweise 3 (falls y eigentlich in die Hilfsliste gelegt werden sollte) erhöhen.(Die Bank wurde bereits durch die Anwendung der entsprechenden Regel um jeweils 0,5 erhöht) Dieser Fall wird permanent vom Algorithmus überprüft, also für jeden Knoten, der in die Hilfsliste oder die Knotenüberdeckung gelegt wird, wird auch überprüft, ob einer der ehemaligen Nachbarn in der Hilfsliste liegt. In der Praxis kann dies umgesetzt werden, indem jeder Knoten zusätzlich markiert wird, wenn einer seiner Nachbarn in die Hilfsliste gelegt wird. Dann muss nur überprüft werden, ob diese Markierung gegeben ist. Für andere Knoten ist dann die Überprüfung, ob ein Nachbar in der Hilfsliste liegt, nicht erforderlich.

Fall 2:
Regel 7 für Knoten mit Grad 1. Siehe dort für die Erläuterung. Dies ist der nicht exakt gelöste Fall. Dies stellt sicher, dass zu keinem Zeitpunkt ein Knoten im Graphen für mehr als einen Knoten in der Hilfsliste die Totalitätsbedingung erfüllen könnte.

Fall 3:
Bis auf einen Nachbarn y von x werden alle Nachbarn im Verlaufe des Algorithmusdurchlaufes aus dem Graphen entfernt und nicht in die Knotenüberdeckung gelegt. Da damit nur noch der Knoten y ein potentieller Partner für x ist, um die Totalitätsbedingung für x zu erfüllen, können wir den Knoten y direkt in die Überdeckung legen, genau wie den Knoten x, welcher damit auch aus der Hilfsliste entfernt wird. Wir können in diesem Fall die Bank um 3,5 erhöhen.(1,5 für den Knoten in der Hilfsliste, da bereits eine Erhöhung um 0,5 stattgefunden hat, sowie 2 für den minimalen Knoten y). Dies funktioniert, da wir in diesem Fall annehmen dürfen, dass alle Nachbarn des Knoten x nicht in die Überdeckung gelegt werden. In diesem Fall muss aber trotzdem die Totalitätsbedingung für den Knoten x erfüllt werden. Daher ist x sowie ein beliebiger seiner Nachbarn eine minimale Lösung. Der Nachbar, den wir in diesem Fall wählen, ist daher dieser letzte im Graphen verbleibende Nachbar. Dieser würde entweder sowieso in die Überdeckung gelegt werden oder er ist der Knoten, den wir auswählen dürfen. Daher können wir annehmen, dass eine minimale Knotenüberdeckung existiert, die diesen Knoten enthält.
Anmerkung: Die Hilfsliste dient dazu, die Knoten zu verwalten, für die gilt, dass sie in einer minimalen Knotenüberdeckung liegen müssen, für die aber noch kein Nachbar in der Überdeckung die Totalitätsbedingung erfüllt. Allerdings ist keine minimale Lösung garantiert und außerdem müssen die Knoten in der Hilfsliste permanent vom Algorithmus überwacht werden. Daher versuchen wir in den folgenden Regeln möglichst oft, Knoten aus der Hilfsliste durch Erfüllen der Totalitätsbedingung zu entfernen, selbst wenn wir dafür bei einzelnen Regeln eine möglicherweise überflüssige Approximation in Kauf nehmen. Im Kapitel 3 betrachten wir eine Anpassung, um dieses Problem zu reduzieren.

2.5 Grad 2

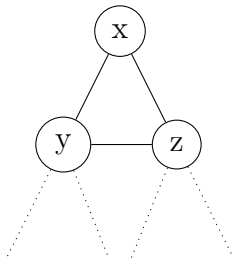
Als Nächstes betrachten wir die Regeln für Knoten mit Grad 2. Hierbei wird immer ein Knoten x mit Grad 2 betrachtet. Der Zustand ist, sofern nicht explizit angegeben, irrelevant.
Fall 1:



Der Knoten x ist nicht in einer Menge von Nachbarn eines Knoten in der Hilfsliste enthalten. Für die Nachbarn y und z gilt, es existiert eine Kante $e = (y, z)$. Dann lege die Knoten y und z in die Knotenüberdeckung, entferne x und alle anliegenden Kanten.

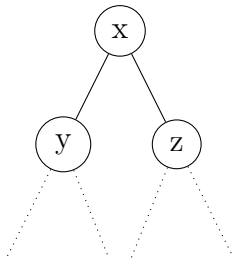
Erhöhe die Bank um 4. Alle Nachbarn von y und z erhalten Zustand *abgedeckt*.
Beweis: Da x, y und z ein Dreieck bilden gilt die Dreiecksregel. Wenn y und z in der Knotenüberdeckung liegen, sind alle Kanten von x abgedeckt. Würden wir x in die Überdeckung legen, so müssten wir sicherstellen, dass entweder y oder z nicht in die Überdeckung kommen, da y und z eine minimale Lösung darstellen, und daher x, y und z in die Überdeckung keine minimale Lösung wäre. Da aber y und z auf jeden Fall eine minimale Lösung darstellen, würde x in die Überdeckung aufzunehmen nur unnötigen Aufwand bedeuten. Der Knoten x muss daher definitiv nicht in die Knotenüberdeckung.

Fall 2:



Der Knoten x liegt in einer Menge von Nachbarn eines Knoten in der Hilfsliste. Für die Nachbarn y und z gilt, es existiert eine Kante $e = (y, z)$. Lege x, y und z in die Knotenüberdeckung, entferne alle anliegenden Kanten. Erhöhe die Bank um 2,5. Alle Nachbarn von y und z erhalten Zustand *abgedeckt*.
Beweis: Da x, y und z ein Dreieck bilden, gilt die Dreiecksregel. Wir nehmen hierbei die Approximation für das Dreieck in Kauf, dafür wird aber der Nachbar von x in der Hilfsliste abgedeckt und damit minimal gelöst. Daher können wir die Bank um 1 erhöhen für die Dreiecksregel und um 1,5 für den minimal gelösten Knoten(0,5 wurden ja bereits durch die ursprüngliche Regel hinzugefügt).

Fall 3:



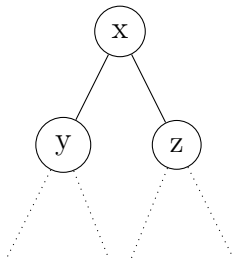
Der Knoten x ist im Zustand abgedeckt und für die Nachbarn y und z gilt, dass weder eine Kante $e = (y, z)$ existiert, noch ein Knoten a, für den gilt, dass die Kanten $e1 = (a, y)$ sowie $e2 = (y, z)$ existieren (weder Dreieck noch Viereck). Der Knoten y wird mit dem Knoten z zusammengelegt, das heißt, alle Nachbarn von z, die nicht auch schon Nachbarn von y sind, werden zu Nachbarn von y. Dann werden x und z zusammen mit allen

verbliebenen Kanten aus dem Graphen entfernt (allerdings zunächst weder gelöscht noch in die Überdeckung gelegt). Was mit x und z in Bezug auf die Knotenüberdeckung passiert, ist abhängig davon, was später mit dem Knoten y passiert. Es gibt 2 Möglichkeiten:

Möglichkeit 1:
Der Knoten y wird im Verlaufe des Algorithmus in die Knotenüberdeckung gelegt (dies beinhaltet auch den Fall, dass der Knoten zunächst in die Hilfsliste gelegt wird). In diesem Fall werden ebenfalls die beiden Knoten x und z in die Überdeckung gelegt. Die Bank wird um 1 erhöht.

Möglichkeit 2:
Der Knoten y wird gelöscht und landet daher nicht in der Überdeckung. In diesem Fall lege x in die Überdeckung, lösche z und erhöhe die Bank um 2. Beweis:
Da x bereits abgedeckt ist, kann x eine minimale Lösung für das Tripel x, y und z sein. Daher wäre, x, y und z einfach in die Knotenüberdeckung zu legen, ein Faktor-3-Approximationsschritt, also für diesen Algorithmus völlig ungeeignet. Durch das übertragen der Nachbarn von z auf y wird sichergestellt, dass entweder alle Nachbarn der beiden Knoten in die Überdeckung gelegt werden, oder mindestens einer der beiden Knoten. Dies ist dadurch ausgedrückt, dass y in die Überdeckung kommt im Verlaufe des Algorithmus. Hierbei ist irrelevant welcher der beiden Knoten y oder z tatsächlich in die Überdeckung gelegt werden sollte. Da einer der beiden Knoten in unserer minimalen Lösung liegt, muss auch mindestens einer der anderen beiden Knoten in die Überdeckung, da ja eine Kante zwischen ihnen existiert. Dies ist dann der Fall 4 und wird hier genauso behandelt. Siehe dazu auch den Beweis dort. Wird der Knoten y im Verlauf des Algorithmus jedoch gelöscht, so bedeutet dies, dass alle Nachbarn von y und z in die Überdeckung gelegt wurden außer x. Da x eine zulässige Lösung ist, können wir also einfach x in die Überdeckung legen, und y und z löschen. Damit sind alle Nachbarn von y und z in der Überdeckung, und damit müssen y und z garantiert nicht mehr in die Überdeckung.

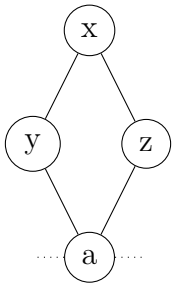
Fall 4:



Der Knoten x ist im Zustand *nicht abgedeckt* und für die Nachbarn y und z gilt, dass weder eine Kante $e = (y, z)$ existiert, noch ein Knoten $a \neq x$, für den gilt, dass die Kanten $e1 = (a, y)$ sowie $e2 = (y, z)$ existieren (weder Dreieck noch Viereck). Lege x, y und z in die Knotenüberdeckung und entferne alle anliegenden Kanten. Erhöhe die Bank

um 1, alle Nachbarn von y und z erhalten Zustand *abgedeckt*.
Beweis: da x *nicht abgedeckt* ist, ist x alleine keine gültige Lösung für das Tripel x, y und z. Wenn x in die Knotenüberdeckung kommt, muss auf jeden Fall auch entweder y oder z in die Knotenüberdeckung, um die Totalitätsbedingung für x zu erfüllen. Es müssen also auf jeden Fall zwei der Knoten in die Knotenüberdeckung gelegt werden. Daher betrachten wir dies wie eine Dreiecksregel.

Fall 5:



Die Nachbarn y und z haben beide Grad 2 und es existiert genau ein Knoten $a \neq x$ mit Kanten $e1 = (a, y)$ und $e2 = (a, z)$. Dieser Fall lässt sich immer minimal lösen, die Lösung hängt jedoch von den Zuständen der vier Knoten ab. Es ergeben sich folgende Möglichkeiten:

Möglichkeit 1:
Die Knoten x und a haben Zustand *abgedeckt*. Lege x und a in die Knotenüberdeckung, lösche y und z und entferne alle anliegenden Kanten. Die Bank wird um 4 erhöht und alle Nachbarn von a erhalten Zustand *abgedeckt*.

Möglichkeit 2:
Knoten a hat Grad 2, aber den Zustand *nicht abgedeckt*. Der Knoten y hat den Zustand *nicht Abgedeckt* (hat y Zustand *abgedeckt*, aber z Zustand *nicht abgedeckt*, so vertausche im Folgenden die Knoten y und z analog). Lege die Knoten x, z und a in die Überdeckung, entferne den Knoten y und erhöhe die Bank um 6. Entferne alle anliegenden Kanten.

Möglichkeit 3:
Knoten x hat Zustand *abgedeckt*. Knoten a hat Grad 3, aber den Zustand *nicht abgedeckt*. Sei b der dritte Nachbar von a. Lege x, a und b in die Knotenüberdeckung, lösche die Knoten y und z und alle anliegenden Kanten. Erhöhe die Bank um 6, alle Nachbarn von b erhalten Zustand *abgedeckt*.

Möglichkeit 4:
Der Knoten x hat den Zustand abgedeckt, nicht aber der Knoten a. Lege x in die Knotenüberdeckung, a in die Hilfsliste und lösche y und z. Entferne alle anliegenden Kanten

und erhöhe die Bank um 2,5. Alle Nachbarn von a erhalten Zustand *abgedeckt*.

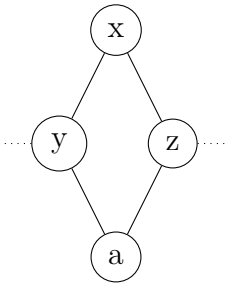
Möglichkeit 5:
Der Knoten x hat Zustand *nicht abgedeckt*. Die Knoten y und z haben Zustand *abgedeckt*. Lege y und z in die Knotenüberdeckung, entferne den Knoten x und alle anliegenden Kanten. Erhöhe die Bank um 4, der Knoten a erhält Zustand *abgedeckt*.

Möglichkeit 6:
Der Knoten x hat Zustand *nicht abgedeckt* und einer der Knoten y oder z hat ebenfalls Zustand *nicht abgedeckt*. Lege die Knoten y, z und a in die Überdeckung, entferne den Knoten x und erhöhe die Bank um 6. Entferne alle anliegenden Kanten. Alle Nachbarn von a erhalten Zustand *abgedeckt*. (Dieser Fall gilt immer, falls keine der vorherigen Lösungen greift. Damit sind alle Möglichkeiten abgedeckt)

Beweis: Da die vier Knoten x, y, z und a einen Kreis der Länge 4 bilden, müssen immer mindestens 2 nicht benachbarte Knoten (x und a oder y und z) in die Überdeckung gelegt werden. Dabei muss sichergestellt werden, dass die Totalitätsbedingung für jeden Knoten, der in die Überdeckung gelegt wird, erfüllt ist.
Zu 1: x und a decken den Kreis ab, daher müssen y und z definitiv nicht in die Überdeckung. Außerdem ist die Totalitätsbedingung sowohl für x als auch a erfüllt. Damit ist die Lösung minimal.
Zu 2: In diesem Fall kann kein Paar nicht benachbarter Knoten den Kreis abdecken und dabei alleine die Totalitätsbedingung für beide Knoten erfüllen, da keiner der Knoten einen weiteren Nachbarn hat dafür. Daher müssen drei der Knoten in die Überdeckung gelegt werden, um die Totalitätsbedingung für alle Knoten zu erfüllen. Diese drei Knoten sind daher eine minimale Lösung. Die Wahl der drei Knoten ist genau genommen irrelevant. Wir wählen daher einen Knoten zur Entfernung, der nicht den Zustand *abgedeckt* hat, da die anderen Knoten dann möglicherweise noch Knoten aus der Hilfsliste abdecken können.
Zu 3: Hier wird wieder x und a gewählt, um den Kreis abzudecken. Würden wir die Knoten y und z wählen, so müssten trotzdem entweder a oder sein Nachbar b in die Überdeckung gelegt werden, also mindestens drei Knoten. Allerdings wäre nicht garantiert, dass die Totalitätsbedingung für a oder b erfüllt ist. Daher legen wir b in die Überdeckung, womit die Totalitätsbedingung für a und b erfüllt ist. Da mindestens drei der Knoten in die Überdeckung müssen, ist die Lösung minimal.
Zu 4: Auch hier wird wieder x und a gewählt, um den Kreis abzudecken. Da a Grad ≥ 4 hat, können wir a nicht einfach abdecken. Daher legen wir a in die Hilfsliste, um eine spätere Abdeckung zu garantieren. Würden wir y und z wählen, so wäre nicht garantiert, dass a nicht in die Überdeckung muss (Totalitätsbedingung für y und z), also könnten drei Knoten in die Überdeckung müssen, während bei x und a nur zwei hineinkommen. Daher ist die Lösung minimal.
Zu 5: In diesem Fall können wir den Knoten a nicht direkt behandeln. Da aber, wenn x in die Überdeckung gelegt werden soll, auch auf jeden Fall y oder z in die Überdeckung müssen, um die Totalitätsbedingung für x zu erfüllen und eben a um den Kreis abzu-

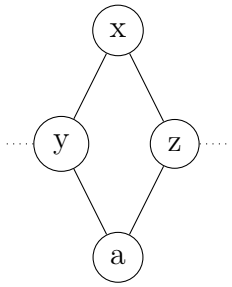
decken, wären es also auf jeden Fall drei Knoten, die in die Überdeckung gelegt werden müssen. Da aber y und z abgedeckt sind, können wir mit ihnen den Kreis abdecken und die Totalitätsbedingung für y und z ist erfüllt. Damit ist der Kreis mit zwei Knoten abgedeckt und diese sind daher minimal. Unabhängig davon, ob a später in die Überdeckung kommt oder nicht.
Zu 6: Hier existiert kein Paar nicht benachbarter Knoten, welches den Kreis abdecken kann und dabei die Totalitätsbedingung für jeden Knoten, der in die Überdeckung kommt, erfüllt. Daher müssen mindestens drei Knoten in die Überdeckung gelegt werden, um dies zu erreichen. Dabei ist jede Dreierkombi aus den vier Knoten minimal, aber wir wählen wieder einen Knoten ohne Nachbarn zum Entfernen, da die anderen Knoten dann potentiell Knoten aus der Hilfsliste abdecken können.

Fall 6:



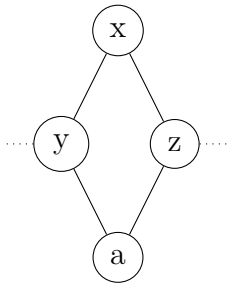
Der Knoten x ist im Zustand *abgedeckt*. Für die Nachbarn y und z gilt, es existiert genau ein Knoten $a \neq x$ mit Kanten $e1 = (a, y)$ und $e2 = (a, z)$. Der Knoten a hat ebenfalls Zustand *abgedeckt*. Lege die Knoten x und a in die Knotenüberdeckung und entferne alle anliegenden Kanten. Erhöhe die Bank um 4, alle Nachbarn von x und a erhalten Zustand *abgedeckt*.
Beweis: Da die Knoten x, y, z und a ein Viereck bilden, müssen mindestens zwei dieser Knoten in die Überdeckung. Genauer gesagt, entweder das Paar x und a oder das Paar y und z müssen in die Knotenüberdeckung. Da jedoch keine Kanten zwischen den Paaren existieren, muss zusätzlich sichergestellt werden, dass die Totalitätsbedingung für die Knoten, welche in die Überdeckung kommen, erfüllt ist. Da dies für das Paar x und a gilt, können wir diese in die Überdeckung legen, ohne dass weitere Knoten nötig sind und damit das Viereck abdecken.

Fall 7:



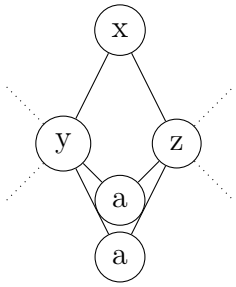
Für die Nachbarn y und z gilt, es existiert mindestens ein Knoten $a \neq x$ mit Kanten $e1 = (a, y)$ und $e2 = (a, z)$. Die Knoten y und z haben den Zustand *abgedeckt*. Lege die Knoten y und z in die Knotenüberdeckung, entferne alle anliegenden Kanten und lösche den Knoten x, sowie alle Knoten die nach Löschung der Kanten Grad 0 haben. Erhöhe die Bank um 4, alle Nachbarn von y und z erhalten Zustand *abgedeckt*.
Beweis: Da x, y, z und a ein Viereck bilden, müssen mindestens zwei nicht benachbarte Knoten in die Überdeckung gelegt werden. Da y und z bereits abgedeckt sind, muss man sich keine weiteren Gedanken über die Totalitätsbedingung machen, und da mindestens zwei der vier Knoten in jede minimale Überdeckung müssen, ist diese Lösung minimal.

Fall 8:



Für die Nachbarn y und z gilt, es existiert genau ein Knoten $a \neq x$ mit Kanten $e1 = (a, y)$ und $e2 = (a, z)$. Lege die Knoten y, z und a in die Knotenüberdeckung, entferne alle anliegenden Kanten und lösche den Knoten x. Erhöhe die Bank um 1, alle Nachbarn von y, z und a erhalten Zustand *abgedeckt*.
Beweis: Da die Knoten x, y, z und a ein Viereck bilden, müssen mindestens zwei dieser Knoten in die Überdeckung. Genauer gesagt entweder das Paar x und a oder das Paar y und z müssen in die Knotenüberdeckung. Da jedoch keine Kanten zwischen den Paaren existieren, muss zusätzlich sichergestellt werden, dass die Totalitätsbedingung für die Knoten, welche in die Überdeckung kommen, erfüllt ist. Um die Totalitätsbedingung für y und z zu erfüllen, legen wir zusätzlich den Knoten a in die Überdeckung, womit drei Knoten hineinkommen, von denen mindestens zwei in einer Knotenüberdeckung liegen müssen.

Fall 9:



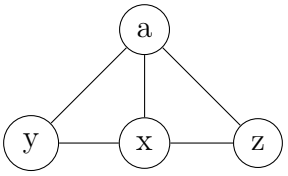
Für die Nachbarn y und z gilt, es existieren mindestens zwei weitere Knoten $a \neq x$ mit Kanten $e1 = (a, y)$ und $e2 = (a, z)$. Wähle aus diesen Knoten einen Knoten b für den gilt, dass $\text{Grad}(b) \geq \text{Grad}(a)$ für alle solchen Knoten a. Lege die Knoten y, z und b in die Knotenüberdeckung, entferne alle anliegenden Kanten und alle Knoten die nun Grad 0 haben. Erhöhe die Bank um 1 und alle Nachbarn von y, z und b erhalten den Zustand *abgedeckt*.
Beweis: Die Knoten y und z sind hier Teil gleich mehrerer Vierecke, daher gilt, dass entweder y und z in die Knotenüberdeckung gelegt werden oder alle ihre Nachbarn. Insbesondere decken y und z sämtliche Vierecke ab, während dadurch auf jeden Fall der Knoten x entfernt werden kann. Um jedoch die Totalitätsbedingung für y und z zu erfüllen, müssen wir wieder einen zusätzlichen Knoten in die Überdeckung legen, wir wählen hierzu den mit dem höchsten Grad.

Anmerkungen:
Bei diesen Fällen ist der Fall 4 derjenige, welcher greift, wenn kein anderer Fall gilt. Damit ist sichergestellt, dass alle Knoten mit Grad 2 behandelt werden. Allerdings muss für die Fälle 5-9 der Fall 4 ausgeschlossen werden, damit diese funktionieren. Jeder Knoten, der keinen der Fälle 5-9 erfüllt, wird aber spätestens bei Fall 4 behandelt werden.

2.6 Grad 3

Wir behandeln nun die Regeln für Knoten mit Grad 3. Hierbei wird immer der Knoten x mit Grad 3 betrachtet. Der Zustand ist, sofern nicht explizit angegeben, irrelevant.

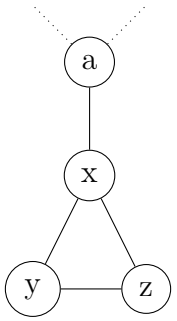
Fall 1:



Für die Nachbarn y, z und a existieren die Kanten $e1 = (y, a)$ und $e2 = (z, a)$. Lege a in die Knotenüberdeckung, entferne alle anliegenden Kanten und erhöhe die Bank um 2.

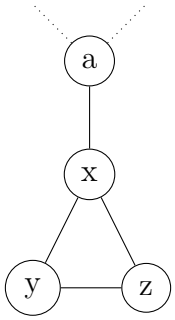
Alle Nachbarn von a erhalten Zustand *abgedeckt*.
Beweis: Da die Knoten x, y und a, beziehungsweise x, z und a jeweils ein Dreieck bilden, müssen jeweils mindestens zwei dieser Knoten in die Überdeckung. Kommt der Knoten a nicht in die Überdeckung, so müssen alle seine Nachbarn, also insbesondere x, y und z, in die Überdeckung gelegt werden. In diesem Fall können wir aber auch den Knoten a statt x in die Überdeckung legen und dafür x entfernen, ohne dass dadurch zusätzliche Knoten in die Überdeckung gelegt werden müssen. Daher können wir annehmen, dass immer eine minimale Lösung existiert, welche den Knoten a enthält und diesen daher als minimal gelöst annehmen. Da durch die Kanten in seiner Nachbarschaft sichergestellt ist, dass mindestens einer seiner Nachbarn später in die Knotenüberdeckung gelegt wird, ist garantiert, dass später die Totalitätsbedingung für a erfüllt wird, und daher müssen wir a nicht in die Hilfsliste legen.

Fall 2:



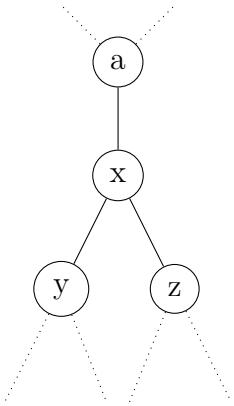
Für die Nachbarn y und z existiert die Kante $e = (y, z)$ und der verbleibende Nachbar a hat den Zustand *abgedeckt*. Lege die Knoten y, z und a in die Überdeckung, entferne alle anliegenden Kanten und lösche den Knoten x. Erhöhe die Bank um 6, alle Nachbarn von y, z und a erhalten Zustand *abgedeckt*.
Beweis: die Knoten x, y und z bilden ein Dreieck, also gilt die Dreiecksregel. Die Knoten y und z decken dieses Dreieck ab. Da x und a ein Paar bilden, muss also einer der beiden Knoten in die Überdeckung. Wenn a nicht in die Überdeckung kommt, so müssen alle seine Nachbarn in die Überdeckung, also mindestens drei (es existieren keine Knoten vom Grad < 3 mehr im Graphen). Wenn wir a in die Überdeckung legen und x löschen, müssen keine weiteren Knoten in die Überdeckung gelegt werden, da für a ja bereits die Totalitätsbedingung erfüllt ist und y und z diese gegenseitig erfüllen.

Fall 3:



Für die Nachbarn y und z existiert die Kante $e = (y, z)$. Lege die Knoten x, y und z in die Überdeckung, entferne alle anliegenden Kanten und erhöhe die Bank um 1. Alle Nachbarn von x, y und z erhalten Zustand *abgedeckt*.
Beweis: Dreiecksregel. Da in diesem Fall der verbleibende Nachbar *a nicht abgedeckt* ist, müsste mindestens ein weiterer Knoten in die Überdeckung gelegt werden, falls a in die Überdeckung kommt. Daher wählen wir hier einfach die Dreiecksregel, um die Knoten x, y und z zu händeln.

Fall 4:



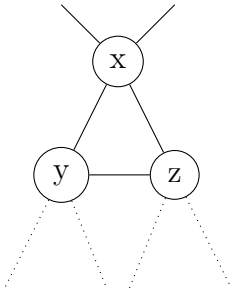
Es existieren keine Nachbarn y und z, für die eine Kante $e = (y, z)$ existiert. Wähle den Nachbarn y mit höchstem Grad, sowie einen Nachbarn a von y mit höchstem Grad, lege y und a in die Knotenüberdeckung und entferne alle anliegenden Kanten. Reduziere die Bank um 1, alle Nachbarn von y und a erhalten Zustand *abgedeckt*.
Beweis: Wir wenden eine Reduktion an, wodurch der Knoten x Grad 2 erhält. Durch die Behandlung dieses Grad 2 Knoten wird die Bank garantiert wieder ausgeglichen.

Anmerkungen:
Da der Fall 4 alle Fälle abdeckt, in denen der Knoten x nicht Teil eines Dreiecks ist, und der Fall 3 alle Fälle, wo er Teil eines Dreiecks ist aber keinen vorherigen Fall erfüllt, sind alle möglichen Fälle für x abgedeckt.

2.7 Grad 4

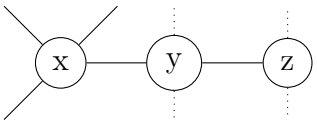
Wir betrachten nun die Regeln für Knoten mit Grad 4. Hierbei wird immer der Knoten x mit Grad 4 betrachtet. Der Zustand ist irrelevant. Diese Regeln finden ausschließlich Anwendung, wenn ein $4-regulärer$ Graph behandelt wird.

Fall 1:



Für die Nachbarn y und z existiert die Kante $e = (y, z)$. Lege die Knoten x, y und z in die Überdeckung, entferne alle anliegenden Kanten und erhöhe die Bank um 1. Alle Nachbarn von x, y und z erhalten Zustand *abgedeckt*.
Beweis: Dreiecksregel.

Fall 2:



Es existieren keine Nachbarn y und z für die eine Kante $e = (y, z)$ existiert. Wähle den Nachbarn y mit höchstem Grad sowie einen Nachbarn a von y mit höchstem Grad, lege y und a in die Knotenüberdeckung und entferne alle anliegenden Kanten. Reduziere die Bank um 1, alle Nachbarn von y und a erhalten Zustand *abgedeckt*.
Beweis: Wir wenden hier eine Reduktion an, so dass der Knoten x als Knoten vom Grad 3 weiter gehändelt werden kann. Da diese Regel nur angewendet wird, wenn der Graph $4-regulär$ ist und wir bei den restlichen Regeln keine Kanten umändern oder hinzufügen, kann nach Anwendung dieser Regel nicht wieder ein $4-regulärer$ Graph entstehen. Es entsteht also eine Kaskadierende Reduktion, womit garantiert ist, dass der Graph irgendwann klein genug ist, damit eine Regel angewendet wird, welche die Bank wieder ausgleicht.

2.8 Erzeugte Knoten vom Grad 1

Anders als beim Algorithmus von [1] ist es hier bei den Regeln für Grad > 1 möglich, dass Knoten vom Grad 1 erzeugt werden. Diese Knoten müssen daher einzeln behandelt

werden. Allerdings wird die Behandlung dieser Knoten als Teil der Regel betrachtet, wenn wir den Erhalt des Durchschnittsgrades diskutieren.
Wir können diese Grad 1 Knoten nicht in den Regeln direkt abfangen, weil wir nicht einfach einzelne Knoten in die Überdeckung legen können, sondern immer dafür sorgen müssen, dass die Totalitätsbedingung für diese Knoten erfüllt ist. Daher müssen wir die einzelnen Regeln beachten, um dies sicherzustellen.

2.9 Ausgleichsschritt

Da die Regeln nur solange funktionieren, wie sich Knoten mit Grad ≤ 4 im Graphen befinden, können wir die Knoten, die übrig bleiben, nicht mehr direkt mit dem gewünschten Approximationsfaktor lösen, sondern müssen diese behandeln, indem wir Knotenpaare in die Überdeckung legen. Dies ist ein Faktor-2-Approximationsschritt. Allerdings wird dies dadurch ausgeglichen, dass durch die vorher angewendeten Regeln ein besserer als der gewünschte Approximationsfaktor erreicht wird. Ausgedrückt wird dies durch die Bank. Diese gibt an, wie viele solcher Faktor-2-Approximationsschritte wir durchführen dürfen, ohne dass der Approximationsfaktor insgesamt über den Zielfaktor steigt.
Theoretisch können wir einfach ein beliebiges Knotenpaar wählen und in die Überdeckung legen, da ja die Knotenpaar Regel gilt, also jedes Paar eine Faktor-2-Approximation darstellt. Allerdings gibt es verschiedene Möglichkeiten, diese Paare sinnvoller auszuwählen. Wir betrachten hier zwei solcher Methoden.

Methode 1: Wähle einen Knoten x im Graphen mit maximalem Grad sowie einen Nachbarn y von x mit maximalem Grad. Lege x und y in die Knotenüberdeckung, reduziere die Bank um 1 und alle Nachbarn von x und y erhalten Zustand *abgedeckt*. Bei dieser Methode wird ein Knotenpaar gewählt, welches möglichst viele Kanten abdeckt. Die Idee hierbei ist, dass wenn die Knoten mit hohem Grad nicht in die Überdeckung kommen, dann alle ihre Nachbarn in die Überdeckung müssen. Durch die Wahl dieser Knoten für die Reduktion ist zumindest sichergestellt, dass diese Knoten in der Überdeckung liegen.

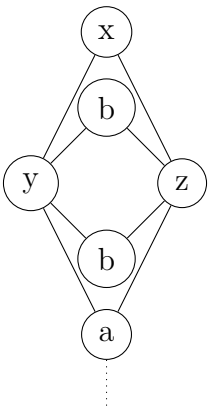
Methode 2: Für einen Knoten x im Graphen mit minimalem Grad wähle einen Nachbarn y von x mit maximalem Grad sowie einen Nachbarn z von y mit maximalem Grad. Lege y und z in die Knotenüberdeckung, reduziere die Bank um 1 und alle Nachbarn von y und z erhalten Zustand *abgedeckt*. Hierbei ist das Ziel, den Grad des Knoten x zu reduzieren, um schnellstmöglich einen Knoten vom Grad < 4 zu erhalten. Die Hoffnung hierbei ist, dass möglichst viele Knoten mit Regeln abgearbeitet werden können, deren Approximationsfaktor möglichst gering ist.

2.10 Erhalt des Durchschnittsgrades

Folgendes gilt: In einem Graphen G mit Durchschnittsgrad d_{avg} gilt, werden x Knoten entfernt und mindestens $\frac{d_{avg}}{2}x$ Kanten, so gilt für den verbleibenden Graphen G' $d'_{avg} \leq d_{avg}$ (Siehe Lemma 7 in [1]).
Desweiteren gilt, führen wir einen Ausgleichsschritt aus, so haben alle Knoten mindestens Grad 4 und es existiert mindestens ein Knoten mit Grad ≥ 5. Entfernen wir also ein Paar, so entfernen wir ebenfalls mindestens $5 + 4 - 1 = 8$ Kanten (Wir nehmen den Knoten mit maximalem Grad, also mindestens 5, dazu einen Nachbarn mit Grad mindestens 4. Da die beiden ein Paar bilden, muss dann noch eine Kante abgezogen werden da sie sonst doppelt gezählt würde).
Wir betrachten zunächst die Regeln für Grad ≥ 2. Die Behandlung von Grad 1 Knoten folgt danach.

Zu Grad 2 **Regeln**:
Wir entfernen entweder $1, 2, 3$ oder 4 Knoten. Da alle entfernten Knoten mindestens Grad 2 haben, ergeben sich daraus folgende Fälle:

Fall 1: Wir entfernen 1 Knoten und mindestens 2 Kanten. Damit bleibt der Durchschnittsgrad erhalten.
Fall 2: Wir entfernen 2 Knoten und mindestens 3 Kanten. Führen wir einen Ausgleichsschritt aus, so werden 4 Knoten und mindestens 11 Kanten entfernt. Damit bleibt der Durchschnittsgrad erhalten.
Fall 3: Wir entfernen 3 Knoten und genau 3 Kanten. In diesem Fall wird die Bank garantiert um 4 erhöht, da dies immer minimal gelöst wird (es handelt sich bei diesem Fall um ein Dreieck mit 3 Knoten vom Grad 2). Führen wir zwei Ausgleichsschritte aus, so werden 7 Knoten und mindestens 19 Kanten entfernt. Damit bleibt der Durchschnittsgrad erhalten.
Fall 4: Wir entfernen 3 Knoten und mehr als 3 Kanten. Führen wir einen Ausgleichsschritt aus, so werden 5 Knoten und mindestens 11 Kanten entfernt. Damit bleibt der Durchschnittsgrad erhalten.
Fall 5: Wir entfernen 4 Knoten und mindestens 4 Kanten. Führen wir einen Ausgleichsschritt aus, so werden 6 Knoten und mindestens 12 Kanten entfernt. Damit bleibt der Durchschnittsgrad erhalten.
Anmerkung: Theoretisch gibt es noch einen sechsten Fall, bei dem beliebig viele Knoten entfernt werden können. Dies passiert, wenn an einem Kreis der Länge 4 noch weitere Knoten vom Grad 2 anhängig sind. Betrachte folgende Grafik:



In diesem Fall jedoch sind für die eigentlichen Regeln nur die Knoten x, y, z und a relevant. Wir können also bei der Betrachtung des Durchschnittsgrades zunächst annehmen, dass die Knoten b entfernt werden. Dies wird hier durch den Fall 1 abgedeckt. Danach verbleiben nur noch die Knoten x, y, z und a, deren Löschung wird durch den Fall 5 abgedeckt. Daher benötigen wir keinen weiteren Fall.

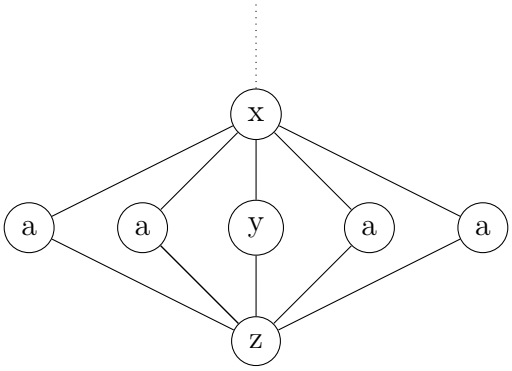
Zu Grad 3 **Regeln**:
Wir entfernen entweder $1, 2, 3$ oder 4 Knoten. Da das Entfernen von zwei Knoten immer ein Reduktionsschritt ist, betrachten wir diese Fälle zuletzt.

Fall 1: Wir entfernen 1 Knoten und mindestens 3 Kanten. Damit bleibt der Durchschnittsgrad erhalten.
Fall 2: Wir entfernen 3 Knoten und mindestens 6 Kanten. Damit bleibt der Durchschnittsgrad erhalten.
Fall 3: Wir entfernen 4 Knoten und mindestens 8 Kanten. Damit bleibt der Durchschnittsgrad erhalten.
Fall 4: Wir entfernen 2 Knoten und mindestens 5 Kanten. Damit würde der Durchschnittsgrad erhalten bleiben, allerdings führen wir hier einen Reduktionsschritt aus, können also einen Ausgleichsschritt weniger ausführen, welcher uns 8 Kanten bringen würde. Daher fehlen uns theoretisch drei Kanten. Allerdings wird dieser Reduktionsschritt nur ausgeführt, wenn sich kein Dreieck mit einem Grad 3 Knoten in der Umgebung befindet. Daher wird der benachbarte Knoten dann zu einem Grad 2 Knoten, aber alle weiteren Knoten haben mindestens Grad 3. Wir können also bei der Betrachtung der Grad 2 Regeln jeweils eine Kante mehr pro zusätzlich entferntem Knoten annehmen. Damit werden wieder überall genug Kanten entfernt.

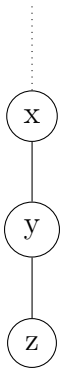
Zu Grad 4 **Regeln**:
Diese Regeln finden nur Anwendung, wenn keine Knoten mit Grad > 4 mehr im Graphen existieren. Daher ist der Durchschnittsgrad in diesem Fall irrelevant, da er keinesfalls mehr über 4 kommen kann.

Zu den Grad-1**-Regeln**:
Die Regeln für Knoten mit Grad 1 stellen für sich alleine betrachtet ein Problem für

den Erhalt des Durchschnittsgrades dar. Allerdings wird die Existenz solcher Knoten zu Beginn der Bearbeitung ausgeschlossen. Die einzige Möglichkeit, dass solche Knoten im Graphen existieren, sind Zwischenschritte beim Durchlauf des Algorithmus. Daher können wir einzelne Situationen anders betrachten.
Das Hauptproblem für den Erhalt des Durchschnittsgrades liegt in der Existenz beliebig vieler Knoten mit Grad 1. Das kann dazu führen, dass beliebig viele Knoten entfernt werden, aber nur ein einzelner Knoten in die Überdeckung gelegt werden muss, wodurch nur ein einziger Ausgleichsschritt möglich ist. Da wir jedoch die Situation betrachten können, bevor die Knoten vom Grad 1 erzeugt werden, stellt sich das Ganze etwas anders da. Siehe dazu folgenden Teilgraphen:



Nehmen wir nun an, dass der Knoten x durch eine andere Regel entfernt wird. Damit entstehen fünf Knoten vom Grad 1, nämlich y und alle a. Allerdings sind nur die Knoten y und z dann relevant für jede angewendete Grad-1-Regel (y sei hierbei der Knoten, der irgendeine Bedingung für eine Grad-1-Regel erfüllt, oder wenn alle Knoten gleich sind bezüglich Zustand, einfach ein beliebig ausgewählter Knoten). Wir können das Ganze daher zunächst wieder so betrachten, dass die Knoten a zunächst gelöscht werden. Da sie Grad 2 haben, werden also für jeweils einen Knoten 2 Kanten entfernt. Damit bleibt der Durchschnittsgrad nach dem Entfernen dieser Knoten erhalten. Damit bleibt dann der folgende Teilgraph erhalten:



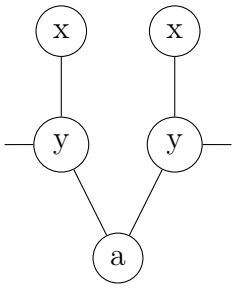
Nun wird der Knoten x durch die Regel entfernt. Für die verbleibende Betrachtung des Durchschnittsgrades muss dieser Knoten Grad 2 haben, also zwei Kanten entfernt

werden. Damit bleiben die Knoten y und z erhalten, mit einer Kante zwischen ihnen. Dies ist nun die Situation, für die wir den Erhalt des Durchschnittsgrades sicherstellen müssen (Dies ist NICHT die Situation, welche die Grad-1-Regel betrachten muss, da dann möglicherweise die Knoten a nicht abgedeckt würden. Da aber eben diese Knoten abgedeckt werden müssen, können wir davon ausgehen, dass eine Situation entsteht, in der die Knoten a einfacht entfernt werden). Hierbei wird also nicht die Grad-1-Regel im einzelnen betrachtet, sondern eine induktive Entwicklung des Durchschnittsgrades für eine Regel und alle daraus entstehenden Grad-1-Knoten.
Damit ergibt sich, dass entweder $1, 2$ oder 5 Knoten entfernt werden müssen, für die wir den Erhalt des Durchschnittsgrades sicherstellen müssen. Hierbei sind die Fälle, bei denen 1 oder 2 Knoten entfernt werden, immer minimal gelöste Regeln. Dies ergibt zunächst zwei Fälle:

Fall 1: Wir entfernen 1 Knoten und mindestens 1 Kante. Führen wir einen Ausgleichsschritt aus, so werden 3 Knoten und mindestens 9 Kanten entfernt. Damit bleibt der Durchschnittsgrad erhalten.

Fall 2: Wir entfernen 2 Knoten und mindestens 1 Kante. Führen wir einen Ausgleichsschritt aus, so werden 4 Knoten und mindestens 9 Kanten entfernt. Damit bleibt der Durchschnittsgrad erhalten.

Ein spezieller Fall ist, wenn zwei Knoten in die Hilfsliste gelegt werden, aber zwischen ihnen ein Knoten existiert. Dies ist der Fall, wo wir die Bank nur um $0, 5$ pro gelöstem Knoten erhöhen. In diesem Fall haben aber die beiden Knoten, welche in die Hilfsliste gelegt wurden, zu diesem Zeitpunkt mindestens Grad 3 (hätten sie nur Grad 2 oder 1, so würden andere Regeln greifen und die Knoten wären nicht in die Hilfsliste gelegt worden).
Es wird also ein Teilgraph betrachtet der ungefähr wie folgt aussieht:



Die Knoten x sind hierbei die Grad 1 Knoten, weswegen die Knoten y in der Hilfsliste landen. Der Knoten a ist der Verbindungsknoten. Wir betrachten diesen Teilbaum bezüglich des Erhalts des Durchschnittsgrades. Es werden also 5 Knoten und mindestens 6 Kanten entfernt und die Bank um 1 erhöht, also ein Ausgleichsschritt ausgeführt. Damit werden 7 Knoten und mindestens 14 Kanten entfernt, womit der Durchschnittsgrad erhalten bleibt.

Zu der Grad-1-Regel:
Die Regel für Grad 0 Knoten ist isoliert betrachtet eine Katastrophe für den Durchschnittsgrad, dies ist allerdings in diesem Fall kein Problem. Da zu Beginn des Durchlaufes keine solchen Knoten existieren dürfen, können sie nur währenddessen erzeugt werden. Dies sind jedoch genau die Speziallfälle, die wir bereits für die Grad 1 beziehungsweise 2 Regeln betrachtet haben. Daher müssen wir die Grad-0-Regel im Bezug auf den Erhalt des Durchschnittsgrades nicht extra betrachten .

2.11 Monitoring

Wie bereits erwähnt liegt ein großer Vorteil des hier vorgestellten Algorithmus in der Möglichkeit, den Faktor unserer Approximation jederzeit während des Durchlaufes des Algorithmus zu überwachen. Hierbei machen wir uns insbesondere zunutze, dass jede Regel in sich abgeschlossen ist und wir daher nach jedem Abschluss einer Regel für alle Knoten, die von dieser Regel in die Knotenüberdeckung gelegt wurden, sagen können, ob sich diese Knoten in einer minimalen Überdeckung befinden müssen. Hierbei wird jedoch zunächst nicht unterschieden zwischen Knoten, die garantiert in der Überdeckung liegen müssen (beziehungsweise sogar in **jeder** minimalen Überdeckung liegen müssen) und Knotenmengen, von denen eine Teilmenge garantiert in der minimalen Überdeckung liegen muss (Beispielsweise müssen immer mindestens zwei Knoten eines Dreiecks in jeder gültigen Überdeckung liegen, also sind auch in jeder minimalen Überdeckung mindestens zwei dieser Knoten enthalten).
Das eigentliche Monitoring setzt sich daher aus zwei verschiedenen Faktoren zusammen, aus denen bei Bedarf ein sehr gutes Bild der Zusammensetzung der Überdeckung abgeleitet werden kann.
Der wichtigste Aspekt des Monitoring ist der tatsächlich erreichte Approximationsfaktor. Da wir, wie bereits erwähnt, nach jeder Regel genau bestimmen können, wie viele der Knoten, welche von der Regel in die Knotenüberdeckung gelegt wurden, sich in der minimalen Überdeckung befinden müssen, können wir bereits während des Durchlaufes des Algorithmus die Anzahl dieser Knoten sowie die Anzahl aller Knoten in der Überdeckung überwachen. Dies lässt sich am einfachsten durch zwei Werte, min = Anzahl der Knoten die in einer minimalen Überdeckung liegen müssen, und all = Anzahl der Knoten in der gefundenen Überdeckung, ausdrücken. Der Approximationsfaktor, welchen der Durchlauf damit tatsächlich erreicht hat, ist also einfach zu berechnen durch $\frac{all}{min}$. Außerdem lässt sich hierdurch direkt auch eine untere Grenze für die Größe einer minimalen Überdeckung angeben, da keine minimale Überdeckung U existieren kann mit $|U| < min$. Allerdings lässt sich hieraus keine exakte Aussage über die Größe einer minimalen Überdeckung treffen, es sei denn der berechnete Approximationsfaktor ist tatsächlich genau 1 (Dies geschieht beispielsweise bei Bäumen, siehe Kapitel 5).
Wir können jedoch mit diesem berechneten Approximationsfaktor und dem erzielten Minimum der Größe einer minimalen Überdeckung meistens eine bessere Abschätzung der Größe einer minimalen Überdeckung angeben, als wenn wir nur einen festen worst-case

Approximationsfaktor annehmen. Außerdem bietet sich die Möglichkeit, zu mindestens für Probleme mit annehmbarer Laufzeit des Algorithmus verschiedene Versionen des Algorithmus (beispielsweise verschiedene Reihenfolge von Regelanwendungen, siehe Kapitel 4, oder verschiedene Anwendung von Regeln, siehe Kapitel 3) laufen zu lassen und die Ergebnisse zu vergleichen. Zwar werden solche verschiedenen Versionen zumeist ähnliche Ergebnisse liefern, aber insbesondere je kleiner der tatsächlich erreiche Approximationsfaktor ist, desto mehr Informationen lassen sich aus Schwankungen in der Lösung gewinnen. Nehmen wir beispielsweise an, Version 1 liefert uns $all1 = 10100, min1 = 10000$ und Version 2 liefert $all2 = 10200, min2 = 10075$. Dann können wir annehmen, dass für eine minimale Überdeckung U gilt, dass $10075 \leq |U| \leq 10100$, also erreichen wir einen tatsächlichen Approximationsfaktor von $\frac{10100}{10075} \approx 1,0025$.
Der zweite Faktor des Monitoring sind die tatsächlich angewendeten Regeln. Auch dies ist sehr simpel durch mehrere mitlaufende Indizes für jeden Fall zu erreichen, welche nach jeder vollständig ausgeführten Regel angepasst werden. Damit lässt sich nach dem vollständigen Durchlauf des Algorithmus genau feststellen, welche Regeln wie oft angewendet wurden. Eine mögliche Anwendung dieser Informationen wäre beispielsweise nach dem Durchlauf einer laufzeitminimalen Version des Algorithmus, welche Version eines Approximationsfaktoroptimierten Algorithmus sinnvoll sein könnte, um bessere Ergebnisse zu erzielen (beispielsweise kann für einen Graphen, bei dem viele Dreiecksregeln angewendet wurden, ein Durchlauf mit einem auf die Lösung von Dreiecken optimierte Version des Algorithmus genutzt werden). Außerdem lassen sich bei einem bereits optimierten Algorithmus aus den angewendeten Regeln möglicherweise noch Aussagen über die Qualität der Lösung auf statistischer Basis ableiten, beispielsweise weil bestimmte Regeln zwar mathematisch nicht exakt sind, aber in der Realität zumeist eine tatsächlich minimale Lösung liefern, während andere Regeln oftmals überflüssige Knoten in die Überdeckung legen.
Zusammengenommen liefert uns das Monitoring somit eine ganze Reihe zusätzlicher Informationen, mit denen die Qualität der gefundenen Lösung besser eingeschätzt werden kann. Außerdem ist der Aufwand für das Monitoring vergleichsweise gering, da die Veränderungen, die jede Regel erzielt fix sind, und nicht jedes Mal berechnet werden müssen. Daher reicht es aus, nach vollständiger Ausführung einer Regel drei Indizes um festgelegte Werte zu erhöhen (all, min und den Index der angewendeten Regel).

2.12 Laufzeit

Da die tatsächliche Laufzeit des Algorithmus stark davon abhängt, welche Regeln in welcher Form und Reihenfolge angewendet werden, betrachten wir im Folgenden nur eine grobe Abschätzung der Laufzeit. Dabei zeigen wir vor allem, dass der Algorithmus das Problem in Polynomialzeit löst, also kein Exponentialzeitalgorithmus ist.
Betrachten wir dazu zunächst einige der Grundoperationen des Algorithmus:

Knoten finden: Vor jeder Regelanwendung müssen wir einen Knoten mit minimalem Grad finden, welcher die entsprechende Regel erfüllt. Dazu müssen wir im Worst-Case

den gesamten Graphen durchlaufen(selbst bei einer Sortierung der Knoten nach Grad, da dieser bei regulären Graphen für alle Knoten gleich ist). Um alle Knoten abzuarbeiten müssen wir wiederum im Worst-Case jeden Knoten einmal suchen. Daher läuft diese Operation in $\mathcal{O}(n^2)$. Verwenden wir jedoch eine Implementierung, bei der immer der erste Knoten mit minimalem Grad bearbeitet wird, unabhängig davon, welche Regel er erfüllt (Siehe hierzu das Kapitel 4), sowie eine Sortierung der Knoten nach Grad, so ist die Suchzeit nur noch $\mathcal{O}(1)$. Außerdem müssen wir für einen vollständigen Durchlauf niemals alle Knoten betrachten, sondern können annehmen, dass jede Regel mindestens zwei Knoten aus dem Graphen entfernt (genau genommen ist die Regel 1 für Knoten vom Grad 3 die einzige Regel, die nur genau einen Knoten aus dem Graphen entfernt, und die Anwendung dieser Regel führt direkt zu einer Regel, bei der mindestens drei Knoten aus dem Graphen entfernt werden. Daher werden bei diesen zwei aufeinanderfolgenden Regeln mindestens vier Knoten entfernt, also zwei pro Regel im Schnitt). Daher können wir hier eine Abschätzung von $\mathcal{O}(m)$ mit $m = \frac{n}{2}$ vornehmen.
Selbiges gilt für die Hilfsliste, in welche wir die Knoten legen können. Da für jeden Knoten, der in die Hilfsliste gelegt wird, mindestens ein Knoten gelöscht wird, können wir auch hier für das Durchlaufen der Hilfsliste eine Laufzeit von $\mathcal{O}(m)$ annehmen. Je nach Implementierung kann das Finden eines Knotens in der Hilfsliste sogar auf $\mathcal{O}(1)$ reduziert werden.
Nachbarn durchlaufen: Bei verschiedenen Schritten des Algorithmus ist es nötig, sämtliche Nachbarn eines Knotens zu durchlaufen, beispielsweise um deren Grad, Nachbarschaft zu anderen Knoten oder ihren Zustand festzustellen, oder alle Kanten zu entfernen, wenn der Knoten aus dem Graphen entfernt werden soll. Dabei ist die Laufzeit abhängig vom maximalen Grad des Graphen, und da theoretisch ein Knoten existieren kann, welcher mit allen anderen Knoten benachbart ist, schätzen wir hier wieder gegen die Gesamtzahl der Knoten im Graphen ab, also $\mathcal{O}(n)$. Man könnte argumentieren, dass in der Realität zum einen die Anzahl von Knoten, die mit allen anderen Knoten im Graphen benachbart sind, begrenzt ist, und außerdem diese Knoten in einem normalen Durchlauf sehr schnell eliminiert würden. Außerdem betrachten wir einen Graphen mit beschränktem Durchschnittsgrad, womit man auch annehmen könnte, dass wir im Schnitt nur jeweils Durchschnittsgrad viele Nachbarn überprüfen müssten. Allerdings ist insbesondere die Annahme bezüglich des Durchschnittsgrades nicht ganz korrekt, da zum einen der Durchschnittsgrad im Verlaufe des Algorithmus schwanken kann (wir stellen nur sicher, dass er immer wieder ausgeglichen werden kann), und außerdem ist die Wahrscheinlichkeit, dass wir die Nachbarn einen Knotens mit Grad > Durchschnittsgrad durchlaufen müssen höher, da diese Knoten natürlich auch öfter mit anderen Knoten benachbart sind als solche mit niedrigem Grad. Daher bleiben wir hier bei der Abschätzung $\mathcal{O}(n)$, auch wenn diese in der Realität meist besser abgeschätzt werden könnte.
Dreiecke finden: Bei dieser Operation wird im Grunde nur das Durchlaufen aller Nachbarn für alle Nachbarn eines Knotens nötig, also zweimal die eben betrachtete Operation. Daher können wir diese Operation abschätzen mit $\mathcal{O}(n^2)$. Analog ist auch das Finden anderer Zusammenhänge wie Vierecke und anderes möglich.
Knoten aus Graphen entfernen: Unabhängig davon ob der Knoten gelöscht, in die Überdeckung oder in die Hilfsliste gelegt wird, immer ist das Entfernen des Knoten aus

dem Graphen nötig. Dabei werden alle Kanten des Knoten gelöscht und der Knoten aus der Speicherstruktur des Graphen entfernt. Da für das Entfernen einer einzelnen Kante immer nur eine feste Reihenfolge von Schritten nötig ist, können wir dies mit $\mathcal{O}(1)$ abschätzen. Auch das Entfernen des Knotens selbst ist in $\mathcal{O}(1)$ möglich. Da wir jedoch jede Kante löschen müssen, ist dies wieder ein Fall von alle Nachbarn durchlaufen, also ist auch hier die Abschätzung wieder $\mathcal{O}(n)$.
Sonstiges: Zusätzlich zu den vorherigen Operationen sind nur noch Dinge notwendig wie die Bank anzupassen, das Monitoring, oder Anpassen des Grades eines Knotens und so weiter. All diese Operationen sind offensichtlich simple Rechenoperationen und lassen sich in $\mathcal{O}(1)$ ausführen.

Zur polynomiellen Laufzeit: Sämtliche Operationen des Algorithmus lassen sich durch die betrachteten Operationen abdecken. Da keine der Operationen nicht in Polynomialzeit ausgeführt werden kann, ist auch der gesamte Algorithmus in Polynomialzeit lösbar.
Zur realistischen Laufzeit: Wie bereits erwähnt hängt die reale Laufzeit stark von den gewählten Faktoren ab. Allerdings sind beispielsweise Operationen wie das Auffinden von Dreiecken nicht zu vermeiden. Daher können wir die Laufzeit mit $\mathcal{O}(n^2)$ abschätzen, wobei hierbei nicht nur die Anzahl der Knoten im Graphen, sondern auch der tatsächliche Durchschnittsgrad und die Struktur des Graphen entscheidend sind. Je nach Implementierung können wir sogar $\mathcal{O}(n)$ erreichen.

3 Modifikationen des Total Vertex Cover Algorithmus

Im vorherigen Kapitel haben wir den Algorithmus für eine totale Knotenüberdeckung für Graphen mit Durchschnittsgrad $d_{avg} = 4$ und Minimalgrad $\delta = 2$ betrachtet. Hierbei wurden die Regeln zunächst möglichst einfach gehalten. Es bieten sich jedoch verschiedene Möglichkeiten, einzelne Regeln zu verbessern. Es ist allerdings nicht möglich, den Faktor generell zu verbessern, sondern nur im Durchschnitt bessere beziehungsweise genauere Ergebnisse im Sinne des Monitoring zu erzielen. Dafür wird die Laufzeit jedoch erhöht. Es ist also eine Abwägung erforderlich, ob, beziehungsweise welche zusätzlichen Regeln sinnvoll sind, oder nur der Grundalgorithmus für das Problem verwendet wird. Wir betrachten diese Anpassungen in diesem Kapitel. Ebenfalls ist für die spezielle Art von Graphen garantiert, dass der Algorithmus korrekt terminiert. Allerdings kann der Algorithmus trotzdem auch auf Graphen angewendet werden, die diese Bedingungen nicht erfüllen. In diesem Fall ist jedoch nicht mehr garantiert, dass der Algorithmus korrekt funktioniert. Wir betrachten im Folgenden daher ebenfalls, welche Anpassungen nötig sind, damit der Algorithmus auch für andere Graphen funktionieren kann, beziehungsweise in welchen Fällen der Algorithmus mit hoher Wahrscheinlichkeit nicht mit einer vollständigen Lösung terminiert.

3.1 Optimierung

Da die Regeln, welche bereits zu minimalen Lösungen führen, offensichtlich nicht mehr verbessert werden können, betrachten wir im Folgenden nur solche Regeln, bei denen eine Approximation stattfindet. Wir unterscheiden hierbei oft zwischen dem ursprünglichen und dem aktuellen Graphen. Der ursprüngliche Graph stellt die initiale Eingabe dar, also unverändert durch den Algorithmus. Der aktuelle Graph ist der Graph, welcher vom Algorithmus verändert wurde durch Entfernen von Knoten und Kanten.

Grad 1 Fall 7 / Behandlung der Hilfsliste
Der Fall 7 für Grad 1 Knoten sorgt dafür, dass jedes Mal, wenn zwei Knoten in der Hilfsliste durch einen Knoten verbunden sind, diese Knoten in die Überdeckung gelegt werden. Dies ist ein simpler Faktor-$\frac{3}{2}$-Approximationsschritt. Wir können diesen Faktor nicht generell verbessern. Allerdings können wir diese Approximationen teilweise vermeiden, indem wir zunächst beide Knoten in die Hilfsliste legen. Dies erfordert eine Anpassung der Behandlung der Hilfsliste, erlaubt aber verschiedene Möglichkeiten, wie Knoten minimal gelöst werden können.
Wir entfernen also den Fall 7 und legen die Regel mit dem Fall 8 zusammen. Also werden nun alle Knoten, denen nicht direkt ein Nachbar in der Knotenüberdeckung zugewiesen werden kann, in die Hilfsliste gelegt, selbst wenn sie einen gemeinsamen Nachbarn mit einem Knoten haben, welcher bereits in der Hilfsliste liegt.
Daher müssen wir nun die Behandlung der Hilfsliste anpassen, um dieser Änderung gerecht zu werden. Wir entfernen den Fall 2, welcher ja dem Fall 7 für Grad 1 Knoten entspricht. Wir behandeln nun also im Verlaufe des Algorithmus möglicherweise auch Knoten, welche mehr als einen ehemaligen Nachbarn haben, welcher nun in der Hilfsliste liegt. Soll ein solcher Knoten durch eine andere Regel in die Überdeckung gelegt werden, so gilt wieder Fall 1 für die Behandlung der Hilfsliste, und wir können diese Nachbarn aus der Hilfsliste entfernen und in die Überdeckung legen. In diesem Fall können wir die Bank anders anpassen, und zwar können wir für jeden Knoten, der aus der Hilfsliste entfernt wird, die Bank um 1,5 erhöhen.
Interessant wird es aber, wenn ein solcher Knoten durch eine Regel entfernt werden soll. In diesem Fall müssen wir sicherstellen, dass zunächst der Knoten korrekt behandelt wird. Wenn wir den Knoten löschen und dadurch die ehemaligen Nachbarn nun nur noch Knoten als Nachbarn besitzen ohne weitere Nachbarn in der Hilfsliste, so werden diese als angeblich minimale Knoten gelöst werden. Dies wäre jedoch nicht korrekt, da, wenn der so gelöschte Knoten in die Überdeckung gelegt werden würde, weniger Knoten in der Überdeckung liegen würden, ergo wäre unsere Lösung nicht länger minimal. Daher betrachten wir 2 neue Fälle, um dies zu verhindern.
Es gilt jeweils, dass ein Knoten x gelöscht werden soll, für den gilt, dass dieser im ursprünglichen Graphen mindestens zwei Nachbarknoten y und z hat, welche im aktuellen Zustand des Algorithmus in der Hilfsliste liegen.

Fall 1:
Für jeden Nachbarn von x aus dem ursprünglichen Graphen, der im aktuellen Zustand des Algorithmus in der Hilfsliste liegt, gilt, dass ein Knoten $a \neq x$ im aktuellen Graphen existiert, mit a hat mindestens genau so viele Nachbarn in der Hilfsliste wie x. In diesem Fall entferne x gemäß der angewendeten Regel.

Fall 2:
Fall 1 gilt nicht. Daher wird die angewendete Regel ausgesetzt und stattdessen wird der Knoten x in die Überdeckung gelegt. Damit werden alle ehemalig benachbarten Knoten, die nun in der Hilfsliste liegen, abgedeckt. Sind dies nur zwei Knoten, so handelt es sich wieder um die ursprüngliche Approximation, also ein Faktor-$\frac{3}{2}$-Approximationsschritt. Ist der Knoten x jedoch noch zu weiteren Knoten in der Hilfsliste benachbart, so werden diese nun ebenfalls abgedeckt. Damit wird für diese Knoten die ursprüngliche Regel 3 für die Behandlung der Hilfsliste erfüllt, und sie können minimal gelöst werden.

Diese Abänderung des Fall 7 für Grad 1 Knoten beziehungsweise der Behandlung der Hilfsliste ermöglicht es, dass in vielen Fällen eine bessere Approximation des Graphen erzielt wird. Allerdings kann dies nicht garantiert werden. Dafür findet hier ein größerer Aufwand statt, da jedes Mal beim Entfernen eines Knotens geprüft werden muss, ob dieser überhaupt entfernt werden darf, oder eben gemäß dieser angepassten Regel in die Überdeckung gelegt werden muss. Daher erhöht die Anwendung dieser Regel die Laufzeit möglicherweise merklich (abhängig davon wie viele Knoten überhaupt in die Hilfsliste gelegt werden), erzielt aber nicht immer eine Verbesserung. Allerdings kann man davon ausgehen, dass, je mehr die Laufzeit erhöht wird, desto mehr Knoten in die Hilfsliste gelegt werden und Fälle überprüft werden müssen, desto häufiger treten auch Fälle auf, in denen die angepasste Regel die Approximation verbessert. Für einen Laufzeitminimalen Durchlauf ist aber trotzdem die ursprüngliche Regel zu bevorzugen, da auch sie eine gültige Approximation garantiert, aber deutlich weniger Aufwand benötigt.

Grad 2 Fall 2
In diesem Fall nutzen wir die Dreiecksregel, da wir dadurch einen Knoten in der Hilfsliste abdecken und somit minimal lösen können. Dies ist genau genommen jedoch nicht nötig. Entfernen wir den Knoten x, statt ihn in die Überdeckung zu legen, so haben wir hier eine minimale Regel, statt einer Approximation. Es gelten die Bedingungen der Hilfsliste. Wir behandeln diesen Fall also nun wie den Fall 1 für Knoten vom Grad 2, daher siehe den dortigen Beweis zu Korrektheit der Regel, und beachte die dortige Anpassung der Bank.
Die ursprüngliche Regel erleichtert uns den Umgang mit der Hilfsliste und ist daher generell einfacher für den Gesamtdurchlauf des Algorithmus, während diese Version der Regel meistens einen besseren Approximationsfaktor erreicht. Dafür müssen wir den Knoten in der Hilfsliste anders behandeln.

Grad 2 Fall 4
Diese Regel können wir nicht direkt verbessern. Jedoch können wir sie nach Abschluss des Algorithmus noch einmal betrachten und dabei möglicherweise verbessern. Es gibt hierbei 3 mögliche Fälle:

Fall 1:
Für die Knoten y und z gilt jeweils: Es existiert ein Nachbarknoten a von y (für z analog) im ursprünglichen Graphen, mit $a \neq x$ und a liegt in der Überdeckung. In diesem Fall ist für die Knoten y und z also die Totalitätsbedingung erfüllt. Daher ist der Knoten x in diesem Fall überflüssigerweise in der Überdeckung und kann gelöscht werden, womit dieser Fall minimal gelöst ist.

Fall 2:
Für den Knoten y gilt (für z analog), dass alle Nachbarknoten a von y im ursprünglichen Graphen nun in der Überdeckung liegen. Außerdem gilt für alle Knoten a, dass ein Nachbarknoten $b \neq y$ im ursprünglichen Graphen existiert, welcher ebenfalls in der Überdeckung liegt. Damit ist der Knoten y überflüssigerweise in der Überdeckung und kann gelöscht werden. Damit ist auch dieser Fall minimal gelöst.
Anmerkung: Gilt dies für beide Knoten y und z, so wird zunächst einer der beiden Knoten entfernt. Damit gilt aber für den Knoten x, dass dieser nun keinen Nachbarn mehr in der Überdeckung besitzt außer dem anderen der beiden Knoten. Damit ist die Bedingung nicht mehr erfüllt. Es ist also nicht möglich, dass sowohl y als auch z gelöscht werden.

Fall 3:
Keiner der beiden vorherigen Fälle gilt. In diesem Fall können wir keine exakte Aussage darüber treffen, ob einer der Knoten überflüssigerweise in der Überdeckung liegt. Daher bleibt die Approximation erhalten.
Da wir nicht garantieren können, dass die Regel nach dem Durchlauf minimal gelöst werden kann, aber immer Aufwand betreiben müssen, um die Bedingungen zu überprüfen, ist hier die Abwägung zwischen Laufzeit und einer möglichen Verbesserung der Approximation vorzunehmen. Die ursprüngliche Regel liefert uns eine sehr schnelle und einfache Lösung, die jedoch immer den Faktor $\frac{3}{2}$ hat. Die Anpassung kann diesen Faktor jedoch nicht garantiert verbessern, sondern nur mit einer gewissen Wahrscheinlichkeit auf 1 verbessern.

Grad 2 Fall 8
In diesem Fall haben wir den Knoten a in die Überdeckung gelegt, um die Totalitätsbedingung für y und z zu garantieren. Wenn jedoch gilt, dass alle Nachbarn b von a im ursprünglichen Graphen (insbesondere y und z) nach dem Durchlauf des Algorithmus in der Knotenüberdeckung liegen, und außerdem für jeden Nachbarn b gilt, dass ein Nachbar c von b im ursprünglichen Graphen in der Überdeckung liegt, mit $c \neq a$, so ist a überflüssig. Daher kann a aus der Überdeckung entfernt werden und die Regel ist minimal gelöst.

Wieder kann kein Erfolg garantiert werden, aber wir haben immer Laufzeit für die Überprüfung, daher muss abgewägt werden ob man diese Laufzeit für eine möglicherweise bessere Approximation in Kauf nehmen möchte.

Grad 2 Fall 9
Auch in diesem Fall wählen wir einen Knoten b und legen ihn in die Überdeckung um die Totalitätsbedingung für y und z zu gewährleisten. Hier gibt es allerdings zwei Wege, die Regel zu verbessern bezüglich einer möglichen minimalen Lösung. Zum Einen direkt während des Durchlaufes des Algorithmus, zum Anderen wie bei der vorherigen Regel nach dem kompletten Durchlauf des Algorithmus.
Anpassung während des Durchlaufes:
In der ursprünglichen Regel wählen wir sinnvoll einen Knoten b und legen diesen direkt in die Überdeckung, um die Totalitätsbedingung für die Knoten y und z zu garantieren. Der Knoten b ist daher eine Approximation. Wir können jedoch unter bestimmten Umständen auch zunächst auf diese Approximation verzichten. Dafür muss gelten, es existiert mindestens ein Knoten $a \neq x$ mit Kanten $e1 = (a, y)$ und $e2 = (a, z)$ und Grad ≥ 3. In diesem Fall legen wir die beiden Knoten y und z in die Hilfsliste statt in die Überdeckung. Dann werden alle anliegenden Kanten entfernt, und ebenso alle Knoten die danach Grad 0 haben.
Existiert kein solcher Knoten a, so wird, wie bei der ursprünglichen Regel, einfach einer der Knoten ausgewählt, welcher benachbart zu y und z ist, und in die Überdeckung gelegt. Diese Anpassung funktioniert nur, wenn auch die Anpassung für den Fall 7 für Grad-1-Regeln beziehungsweise die Hilfsliste erfolgt ist. Nur dann wird sichergestellt, dass die Knoten y und z in der Hilfsliste ordentlich abgedeckt werden, ansonsten kann es passieren, dass beide Knoten durch zwei andere Knoten abgedeckt werden und dies als minimale Lösung angesehen wird, obwohl ein einzelner Knoten a beide Knoten hätte abdecken können.

Anpassung nach dem Durchlauf:
Hier kann das selbe wie beim vorherigen Fall angewendet werden.
Wird die Anpassung für die Hilfsliste vorgenommen, so ist es auch sinnvoll die Anpassung während des Durchlaufes anzuwenden. Die Überprüfung des Grades der Knoten findet auch bei der ursprünglichen Regel statt und die erhöhte Laufzeit für die Hilfsliste wird ja sowieso bereits in Kauf genommen. Für die Anpassung nach dem Durchlauf gilt das selbe wie für den vorherigen Fall.

Grad 3 Fall 3
Auch hier gibt es wieder Möglichkeiten, die Regel sowohl während des Durchlaufes als auch danach anzupassen.
Während des Durchlaufes:
Gilt, die Knoten y und z haben Grad ≥ 4, so lege y und z in die Überdeckung. Dies entspricht der Dreiecksregel insofern, als dass mindestens zwei der Knoten des Dreiecks in die Überdeckung gelegt werden müssen. Entfernt man die Kanten von y und z, so ist x nun ein Grad 1 Knoten. Dieser wird entsprechend behandelt. Hat mindestens einer

der Knoten y oder z Grad 3, so wende die ursprüngliche Regel an.
Nach dem Durchlauf:
Wir können nach Anwendung der Dreiecksregel für alle drei Knoten überprüfen, ob der Knoten überflüssig ist in der Überdeckung. Da jedoch mindestens zwei der Knoten in der Überdeckung liegen müssen, können wir mit der Überprüfung aufhören, sobald ein Knoten aus der Überdeckung entfernt wurde. Wir beschreiben das Vorgehen nun für den Knoten x. Sollte dieser nicht aus der Überdeckung entfernt werden, so gehe für y bzw. z Analog vor.
Gilt für den Knoten x, dass für jeden Nachbarn a von x im ursprünglichen Graphen gilt, dass a in der Überdeckung liegt, und es existiert ein Nachbar $b \neq x$ von a im ursprünglichen Graphen, welcher ebenfalls in der Überdeckung liegt, so ist x überflüssig in der Überdeckung. Entferne in diesem Fall x aus der Überdeckung. Damit sind nun die Knoten y und z minimal gelöst.

Auch hier gilt wieder, dass wir eine erhöhte Laufzeit in Kauf nehmen für eine Chance, aber keine Garantie, auf eine Verbesserung der Approximation.

Grad 3 Fall 4 / Grad 4 Fall 1 / Grad 4 Fall 2 / Ausgleichsschritt
Wir können diese Fälle hier zusammenfassen, da die Anpassungen der Regeln weitestgehend gleich sind. Eine Ausnahme ist die Regel 1 für Grad 4, da diese eine Dreiecksregel statt einer Reduktionsregel ist, daher gibt es für diese Regel keine Verbesserung während des Durchlaufes. Allerdings können die Anpassungen nach dem Durchlauf ebenfalls für diese Regel angewendet werden. Wir haben also wieder die zwei Möglichkeiten, die Regeln während oder nach dem Durchlauf zu verbessern.
Während des Durchlaufes:
Bei den ursprünglichen Regeln entfernen wir immer ein Paar von Knoten. Dies ist sehr einfach und durch die Verwaltung der Bank auch abgedeckt. Allerdings ist nicht ausgeschlossen, dass zum Zeitpunkt der Anwendung dieser Regeln noch Dreiecke im Graph vorhanden sind. Daher können wir diese Regeln jeweils anpassen, indem wir zunächst überprüfen, ob wir nicht stattdessen ein Dreieck entfernen können. Bei den Regeln 4 für Grad 3 beziehungsweise 2 für Grad 4 müssen wir nur überprüfen, ob einer der Nachbarn von x Teil eines Dreiecks ist. In diesem Fall, lege alle Knoten des Dreiecks in die Überdeckung. Der Knoten x kann nicht Teil dieses Dreiecks sein, da ansonsten bereits eine andere Regel gegriffen hätte. Daher wird der Grad des Knoten x wie gewünscht reduziert. Außerdem können wir nun statt die Bank um 1 zu reduzieren die Bank um 1 erhöhen, da wir statt eines Faktor-2-Approximationsschrittes einen Faktor-$\frac{3}{2}$-Approximationsschritt durchführen. Für den Ausgleichsschritt gilt soweit dasselbe, nur dass wir in diesem Fall im gesamten Graphen nach Dreiecken suchen. Der ursprüngliche Ausgleichsschritt wird also nur noch angewendet, wenn sich im Graphen keine Dreiecke mehr befinden.

Nach dem Durchlauf:
Bei diesen Regeln werden ausschließlich Paare oder Dreiecke in die Überdeckung gelegt. Wir können für beide Fälle überprüfen, ob Knoten in der Überdeckung überflüssig sind. Außerdem können wir für Paare eine Bedingung überprüfen, ob das Paar eine minimale

Lösung ist.

Paare:
Gilt für die zwei Knoten x und y des Paares, dass kein anderer Nachbar von x und y in der Überdeckung liegt, so sind x und y minimal gelöst, keiner der Knoten ist also mehr approximiert. Alternativ, gilt für einen der Knoten x oder y, dass für alle Nachbarn a von x (oder y analog) im ursprünglichen Graphen, ein Nachbar $b \neq x$ im ursprünglichen Graphen existiert, mit a und b liegen in der Überdeckung. Dann ist x überflüssig, und kann aus der Überdeckung entfernt werden. Dann ist y minimal gelöst (oder entsprechend x).

Dreiecke:
Wir können für alle drei Knoten überprüfen, ob der Knoten überflüssig ist in der Überdeckung. Da jedoch mindestens zwei der Knoten in der Überdeckung liegen müssen, können wir mit der Überprüfung aufhören, sobald ein Knoten aus der Überdeckung entfernt wurde. Wir beschreiben das Vorgehen nun für den Knoten x. Sollte dieser nicht aus der Überdeckung entfernt werden, so gehe für y beziehungsweise z Analog vor.
Gilt für den Knoten x, dass für jeden Nachbarn a von x im ursprünglichen Graphen gilt, dass a in der Überdeckung liegt, und es existiert ein Nachbar $b \neq x$ von a im ursprünglichen Graphen, welcher ebenfalls in der Überdeckung liegt, so ist x überflüssig in der Überdeckung. Entferne in diesem Fall x aus der Überdeckung. In diesem Fall sind nun die Knoten y und z minimal gelöst.

Auch bei all diesen Anpassungen gilt, dass wir jeweils die Chance haben, die Approximation zu verbessern, aber dafür eine erhöhte Laufzeit in Kauf nehmen müssen.

Zusatz: Kreise ungerader Länge:
Eine weitere mögliche Anpassung für den Ausgleichsschritt ergibt sich durch Kreise ungerader Länge, wie auch Dreiecke welche sind. Für diese gilt, werden alle Knoten eines Kreises der Länge l (l ungerade) in die Überdeckung gelegt, so ist dies ein Faktor-$\frac{l}{\lceil \frac{l}{2} \rceil}$-Approximationsschritt.
Will man also eine möglichst gute Approximation erreichen, so kann es sinnvoll sein, zunächst Kreise ungerader Länge zu entfernen, bevor man Paare entfernt für den Ausgleichsschritt. Da diese trotzdem den Approximationsfaktor über den gewünschten Wert anheben würden, müssen wir trotzdem die Bank um 1 reduzieren wenn wir einen Kreis entfernen (Dreiecke und Kreise der Länge 5 ausgenommen). Wir führen hierbei aber eine Approximation mit einem Faktor aus, der besser als der Faktor 2 ist, welcher durch das hinzufügen von Paaren in die Überdeckung erreicht wird.
Auch hier wird wieder die Laufzeit erhöht und dafür möglicherweise (wenn kein Kreis mehr vorhanden ist, wird trotzdem versucht, einen zu finden, daher ist auch hier kein Erfolg garantiert) eine bessere Approximation erreicht. Da die Laufzeit um so mehr steigt, je länger die Kreise werden, andererseits aber der Faktor immer schlechter wird, sollte man trotzdem die Länge der gesuchten Kreise begrenzen. Dies ist also auch wieder eine

Abwägungssache, wie viel zusätzliche Laufzeit man in Kauf nehmen möchte.

3.2 Beliebiger Minimalgrad

Im vorherigen Kapitel haben wir den Algorithmus für Graphen mit Minimalgrad ≥ 2 betrachtet und gezeigt, dass der Algorithmus in diesen Fällen mit einer vollständigen Lösung terminiert. Für Graphen mit niedrigerem Minimalgrad ist dies zumindest nicht mehr garantiert. Trotzdem können wir den Algorithmus grundsätzlich auch auf solche Graphen anwenden. Die einzige dazu nötige Anpassung ist die Grad-1-Regel als feste Regel statt als Hilfsregel zu betrachten, also dass isolierte Knoten im Graphen einfach entfernt werden. Dann sind grundsätzlich alle Fälle abgedeckt, und der Algorithmus kann fehlerfrei den Graphen bearbeiten. Allerdings ist es möglich, dass ein nicht leerer Graph mit Minimalgrad ≥ 5 erzeugt wird und die Bank leer ist. In diesem Fall terminiert der Algorithmus, allerdings ohne vollständige Lösung.
Der Grund hierfür ist, dass die Knoten vom Grad 0 und 1 für den Erhalt des Durchschnittsgrades ein Hindernis darstellen. Insbesondere bei Grad 0 Knoten ist sogar garantiert, dass durch ihr Entfernen der Durchschnittsgrad erhöht wird entsprechend der gegebenen Regeln, denn es gibt keine Möglichkeit, dass diese Knoten in die Überdeckung gelegt werden. Daher werden hierbei immer genau ein Knoten und keine Kante entfernt. Um den Durchschnittsgrad zu erhalten, müssten aber mindestens zwei Kanten entfernt werden.
Auch die Knoten mit Grad 1 können zu Problemen führen, insbesondere dadurch, dass mehrere solcher Knoten mit dem selben Knoten verbunden sein können.
Bei der Behandlung von Grad 1 Knoten werden maximal zwei Knoten in die Überdeckung gelegt, und die Bank damit um maximal 4 erhöht. Es können also maximal vier Ausgleichsschritte ausgeführt werden, um den Durchschnittsgrad zu erhalten. Pro Ausgleichsschritt werden zwar mindestens acht Kanten entfernt, aber auch zwei Knoten. Damit bleiben also noch maximal $4 * 4 = 16$ Kanten um den Durchschnittsgrad auszugleichen. Liegen also mehr als 16 Knoten mit Grad 1 an einem Knoten an, so werden mindestens 17 Knoten und mindestens 16 Kanten entfernt. Um den Durchschnittsgrad zu erhalten müssten mindestens 34 Kanten entfernt werden, wir erhalten aber im Zweifel nur 16 weitere Kanten durch die Ausgleichsschritte. Damit würde sich der Durchschnittsgrad erhöhen. Im Fall, dass der Knoten, an dem die Grad 1 Knoten anliegen, in die Hilfsliste gelegt wird, wird die Bank sogar nur um 0,5 erhöht, damit kommen wir also nur noch auf mindestens 2 Kanten, die wir als Ausgleich erhalten. In diesem Fall werden durch den Grad 1 Knoten und den Knoten, der in die Hilfsliste gelegt wird, mindestens drei Kanten entfernt (1 von dem Grad 1 Knoten + 2 weitere von dem Knoten in der Hilfsliste, da wenn dieser nur einen weiteren Nachbarn hätte, eine andere Regel greifen würde). Liegen zwei weitere Grad 1 Knoten an, so werden 4 Knoten und mindestens 5 Kanten entfernt. Mit den Kanten vom Ausgleichsschritt kommen wir auf mindestens 7 entfernte Kanten, es werden jedoch mindestens 8 benötigt um den Durchschnittsgrad zu erhalten.

Damit ist zumindest mathematisch nicht mehr garantiert, dass der Durchschnittsgrad erhalten bleibt. Somit kann die Terminierung ohne vollständige Lösung vorkommen. In der Praxis ist es jedoch oftmals der Fall, dass trotzdem eine vollständige Lösung erreicht wird, da an anderer Stelle mehr Kanten als benötigt entfernt werden, oder der Durchschnittsgrad nicht exakt 4, sondern niedriger war, und daher eine Erhöhung nicht zwangsweise den Durchschnittsgrad über 4 erhöht.
Wir können den Algorithmus also prinzipiell auf solche Graphen anwenden und werden oftmals eine vollständige Lösung erhalten. Wir betrachten im Folgenden Bedingungen, unter welchen die Graphen sogar garantiert gelöst werden können. Diese gelten immer für einen Durchschnittsgrad ≤ 4.
Beschränkter Maximalgrad:
Befinden sich keine Knoten im Graph mit Grad ≥ 5, so werden garantiert keine Ausgleichsschritte benötigt, um eine vollständige Lösung zu erreichen. Daher ist der Durchschnittsgrad in diesem Fall irrelevant, da wir unter keinen Umständen den Grad eines Knotens erhöhen, und damit der Durchschnittsgrad auch nie über 4 steigen kann. Daher haben wir immer Knoten im Graphen, welche wir mit einer unserer Regeln bearbeiten können.

Beschränkung der Nachbarn vom Grad 1:
Gilt für jeden Nachbarn eines Knotens mit Grad 1, dass dieser Nachbar keine weiteren Nachbarn vom Grad 1 besitzt, so kann der Graph vollständig vom Algorithmus gelöst werden. Betrachte hierzu den Abschnitt zum Erhalt des Durchschnittsgrades bei Knoten vom Grad 1. Hier werden die Fälle betrachtet, in denen nur ein einzelner Grad 1 Knoten als Nachbar vorhanden ist, und gezeigt, dass in diesen Fällen der Durchschnittsgrad erhalten bleibt.

Akzeptabler Durchschnittsgrad nach Entfernen der störenden Knoten:
Gilt keine der vorherigen Bedingungen, so bleibt noch die Option, den zweiten Fall zu erzeugen falls möglich. Für die Grad-1-Regeln ist es irrelevant, wie viele Nachbarn vom Grad 1 vorhanden sind (eine Ausnahme ist der Fall 4 für Grad 1 Knoten, aber dieser wird durch das Entfernen der zusätzlichen Knoten vom Grad 1 zu einem Fall 3, und die Lösung bleibt dieselbe). Daher entfernen wir zunächst alle Knoten vom Grad 0 sowie alle Knoten vom Grad 1, deren Nachbar mindestens einen weiteren Nachbarn vom Grad 1 hat (dies darf nicht gleichzeitig geschehen, da immer genau ein Knoten vom Grad 1 übrig bleiben muss als Nachbar, und bei gleichzeitigem Entfernen aller Nachbarn vom Grad 1 entfernt werden würden). Danach betrachten wir den Durchschnittsgrad. Ist dieser immer noch ≤ 4, so gilt der vorherige Fall, und der Algorithmus wird den Graphen vollständig lösen. Steigt der Durchschnittsgrad über 4, so ist ein Fall gegeben, in dem die vollständige Lösung nicht mehr garantiert ist.

Wurde das Entfernen der störenden Knoten durchgeführt, aber der Durchschnittsgrad ist in diesem Fall > 4, so kann wie gesagt der Algorithmus trotzdem normal gestartet werden, da die Chance besteht, dass eine vollständige Lösung erzeugt wird. Eine genauere Betrachtung von Graphen mit einem Durchschnittsgrad > 4 erfolgt im folgenden

Abschnitt.

3.3 Beliebiger Durchschnittsgrad

Im Folgenden betrachten wir alle Graphen mit Durchschnittsgrad $Dg > 4$. Hierbei betrachten wir der Einfachheit halber nur aufgerundete ganze Durchschnittsgrade. Generell können wir hierbei annehmen, dass der Graph mit Faktor $\frac{(Dg*2)-3}{Dg-1}$ gelöst werden soll. Dies ist jedoch bei einem Durchschnittsgrad > 4 nicht mehr garantiert. Wir werden im Folgenden zunächst betrachten, welche Anpassungen nötig sind, um die Wahrscheinlichkeit zu erhöhen, dass der Graph vollständig gelöst werden kann. Danach betrachten wir, welche Fälle dazu führen, dass der Algorithmus in diesem Fall scheitert und keine vollständige Lösung erzeugt. Zum Abschluss werden wir betrachten, wie wir ein vollständiges Lösen des Graphen garantieren können.

3.3.1 Anpassungen

Für einen Graphen mit Durchschnittsgrad Dg gilt, dass entweder ein Knoten x existiert mit Grad von $x < Dg$, oder der Graph ist Dg-Regulär. Es gilt, dass wir Knoten mit Grad ≤ 3 immer mit Faktor höchstens $\frac{5}{3}$ lösen können. Existiert kein Knoten mit Grad ≤ 3, und kein Knoten mit minimalem Grad, welcher Teil eines Dreiecks ist, so müssen wir den Grad eines Knotens solange reduzieren, bis wir einen Knoten vom Grad ≤ 3 erzeugt haben. Existiert ein Knoten mit minimalem Grad, welcher Teil eines Dreiecks ist, so wende die Dreiecksregel an und lege alle Knoten des Dreiecks in die Überdeckung. Ansonsten verwenden wir für die Reduktion des Grades von x Regeln, die Analog zu den Grad-4-Regeln sind. Für einen Knoten x mit minimalem Grad gilt also, wir entfernen einen Nachbarn y von x und entweder zwei Knoten z und a für die gilt, dass y, z und a ein Dreieck bilden, oder einen Nachbarn z von y, wobei y der Nachbar von x mit maximalem Grad ist, und z der Nachbar von y mit maximalem Grad ist ($z \neq x$).
Da x höchstens Grad $Dg - 1$ hat, werden also höchsten $Dg - 4$ solcher Reduktionen nötig, damit x Grad ≤ 3 hat. Damit werden also höchstens $Dg - 4$ Knotenpaare in die Überdeckung gelegt, von denen jeweils mindestens ein Knoten in der Überdeckung liegen muss. Da die Knoten vom Grad 3 mit Faktor-$\frac{5}{3}$ gelöst werden können, ergibt sich damit also insgesamt der Faktor $\frac{(2*(Dg-4))+5}{(Dg-4)+3} = \frac{(Dg*2)-3}{Dg-1}$.
Für Dg-Reguläre Graphen gilt das selbe wie für 4-Reguläre Graphen im Kapitel 2.7.
Zusätzlich können wir auch die Regeln um weitere Kreise ungerader Länge erweitern, je höher der Durchschnittsgrad ist. Legen wir beispielsweise alle sieben Knoten eines Kreises der Länge 7 in die Überdeckung, so müssen mindestens vier dieser Knoten in einer minimalen Überdeckung liegen. Es handelt sich also um einen Faktor-$\frac{7}{4}$-Approximationsschritt. Ist unser Zielfaktor gleich oder schlechter als $\frac{7}{4}$, so lohnt sich diese Approximation auf jeden Fall.

3.3.2 Probleme mit dem Durchschnittsgrad

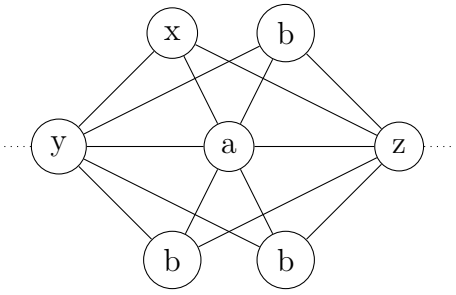
Analog zu den Problemen, welche Knoten vom Grad 1 für den Erhalt des Durchschnittsgrades bei Graphen mit Durchschnittsgrad ≤ 4 bedeuten, werden für Graphen mit Durchschnittsgrad > 4 auch Knoten mit höherem Grad potentiell zu einem Problem. Betrachten wir als Beispiel einen dreiecksfreien 9-regulären Graphen mit zwei Knoten x und y, für die gilt, dass keine Kante $e = (x, y)$ existiert. Füge dem Graphen nun solange Knoten z hinzu, mit Kanten $e1 = (x, z)$ und $e2 = (y, z)$, bis der Durchschnittsgrad des Graphen < 5 ist.

Der Graph wird nun also als Graph vom Durchschnittsgrad 5 behandelt mit einem Zielfaktor von $\frac{7}{4}$. Wird nun der erste Knoten vom Grad 2 behandelt, so werden die Knoten x und y in die Überdeckung gelegt, zusammen mit einem weiteren Knoten aus der Menge der Grad 2 Knoten. Die Bank wird um 1 erhöht. Da zwei Knoten aus dem ursprünglich 9-regulären Graphen entfernt wurden, hat der Graph nun Minimalgrad ≥ 8. Damit können wir keine Regel mehr anwenden für Graphen mit Durchschnittsgrad 5. Da die Bank noch nicht leer ist, können wir einen weiteren Ausgleichsschritt ausführen. Damit ist der Minimalgrad ≥ 7, aber der Graph ist nicht leer, die Bank ist leer und es existieren keine Dreiecke mehr im Graphen. Damit ist der Algorithmus nicht in der Lage, den Graphen weiter zu bearbeiten.
Je höher der Durchschnittsgrad des Graphen, desto höhere Grade von Knoten können diesen Effekt erzielen. Ab Durchschnittsgrad 6 beispielsweise auch Knoten vom Grad 3 und so weiter. Allerdings sind auch hier diese Fälle sehr extreme Beispiele von Graphen. In den meisten Fällen werden die fehlenden Kanten beim Entfernen solcher Zwillinge an anderer Stelle ausgeglichen, beziehungsweise der Durchschnittsgrad wird nicht so weit erhöht, dass es ein Problem darstellt. Wir können jedoch dementsprechend nicht jeden Graphen sicher lösen, daher betrachten wir nun, wie wir sicherstellen können, dass der Graph vollständig vom Algorithmus gelöst wird.

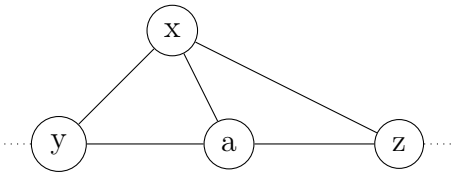
3.3.3 Lösungsstrategien

Wie bereits erwähnt, ist das Auftreten von Fällen, in denen der Algorithmus für Graphen vom Durchschnittsgrad > 4 nicht mit einer vollständigen Lösung terminiert, eher die Ausnahme. Daher können wir prinzipiell analog zu Graphen mit Durchschnittsgrad < 4 den Algorithmus auch einfach auf jeden Graphen starten (entsprechend angepasst an den Durchschnittsgrad) und die Möglichkeit hinnehmen, dass der Algorithmus nicht mit einer vollständigen Lösung terminiert. Alternativ können wir aber auch wieder die entsprechend störenden Zwillinge eliminieren und den Graphen beziehungsweise den Durchschnittsgrad nach dem Entfernen dieser Knoten betrachten, und erst dementsprechend den Algorithmus starten und den Zielfaktor bestimmen. Hierbei gilt, dass für

einen Durchschnittsgrad Dg alle Zwillinge vom Grad $\lceil \frac{Dg}{2} \rceil - 1$ ein Problem darstellen. Es gilt allerdings, dass wir nicht generell alle bis auf einen Zwilling entfernen dürfen wie es bei Knoten vom Grad 1 der Fall ist. Betrachte hierzu folgenden Graphen:



Hierbei sind die Knoten b alles Zwillinge vom Knoten x. Würde nun alle diese Zwillinge b vor dem Durchlauf des Algorithmus aus dem Graphen entfernt werden, so bliebe folgender Teilgraph erhalten:



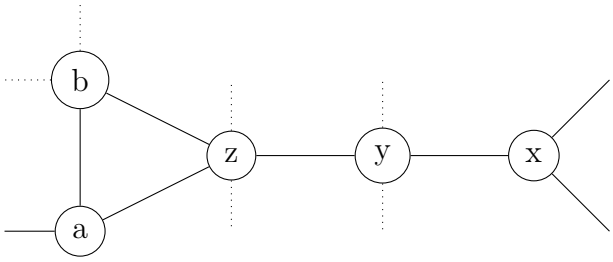
Für diesen Teilgraphen sind nun die Knoten x und a eine mögliche minimale Lösung. Allerdings wäre diese Lösung falsch, da sämtliche Kanten zwischen y beziehungsweise z und den bs nicht abgedeckt wären. Daher müssen genügend Zwillinge erhalten bleiben, um die minimale Lösung nicht zu beeinflussen.
Da wir aber dementsprechend nicht sicherstellen können, dass nicht trotzdem eine Menge von Zwillingen im Graphen existiert, welche unseren Durchschnittsgrad zu stark erhöht, können wir so wieder nur die Wahrscheinlichkeit erhöhen, dass der Graph vollständig gelöst wird.
Wollen wir daher sicher garantieren, dass der Algorithmus immer eine vollständige Lösung liefert, so müssen wir auf die Beschränkung des Zielfaktors verzichten. Dies geschieht, indem wir zulassen, dass die Bank beliebig ins Negative geht, auch ohne zu garantieren, dass diese wieder ausgeglichen wird. Damit können wir immer beliebig viele Ausgleichsschritte ausführen und den Grad der Knoten solange reduzieren, bis wir wieder Knoten vom Grad 3 erzeugt haben, welche dann von unseren Regeln gelöst werden können. Da wir keine Regeln anwenden, deren Faktor schlechter ist als 2, können wir auch garantieren, dass der Algorithmus keinesfalls eine schlechtere Lösung als eine Faktor-2-Approximation liefert. Allerdings können wir in diesem Fall den Faktor vor den Start des Algorithmus nicht mehr besser bestimmen.
Da das Monitoring weiterhin stabil funktioniert, können wir jedoch auch in diesem Fall nach dem Durchlauf des Algorithmus feststellen, welcher Faktor schlechtesten falls erreicht wurde. Dies wird weiterhin in den meisten Fällen signifikant besser sein als Faktor

2, aber eben nicht mehr garantiert.

4 Implementierung

In diesem Kapitel betrachten wir nun einige praktische Aspekte im Hinblick auf eine Implementierung des Algorithmus. Hierbei handelt es sich zunächst um generelle Überlegungen, wie mit bestimmten Aspekten umgegangen werden kann und sollte, und welche Vor- und Nachteile bestimmte Herangehensweisen bieten.
Betrachten wir zunächst den wohl wichtigsten Punkt, die strikte oder nicht strikte Einhaltung der Regel-Reihenfolge. Die Regeln des Algorithmus sind in einer bestimmten Reihenfolge aufgestellt. Diese soll sicher stellen, dass die Regeln möglichst optimal angewendet werden, also möglichst viele Knoten gefunden werden, die garantiert in einer minimalen Überdeckung liegen müssen.
Allerdings funktioniert der Algorithmus auch problemlos, wenn diese Reihenfolge nicht strikt eingehalten wird. Es muss nur sichergestellt werden, dass keine direkte Beeinflussung eines Knotens passiert, welcher die Bedingungen einer früheren Regel erfüllen würde. Daher darf durch die Anwendung einer Regel kein Knoten aus dem Graphen entfernt werden, welcher eine vorherige Regel erfüllt. Dies muss sichergestellt sein, da die Regeln jeweils unter der Annahme gelten, dass im Umfeld des Knotens, auf den die Regel angewendet wird, kein Knoten mehr existiert, der eine vorherige Regel erfüllt.
Der entscheidende Vorteil einer nicht strikten Einhaltung der Reihenfolge der Regeln liegt in der Laufzeit des Algorithmus. Wollen wir sicherstellen, dass die korrekte Reihenfolge eingehalten wird, so müssen wir die Knoten mit minimalem Grad entweder solange überprüfen, bis wir einen Knoten finden, welcher jeweils die erste Regel für Knoten mit diesem Grad erfüllt, oder alle dieser Knoten durchlaufen, und danach den auswählen, welcher den frühsten Fall erfüllt. Da insbesondere die ersten Fälle für jeden Grad sehr spezielle Fälle sind, ist die Wahrscheinlichkeit sehr hoch, dass kein Knoten existiert, welcher diesen Fall erfüllt, und wir dementsprechend jedes Mal die komplette Menge der Knoten mit minimalem Grad durchlaufen müssen. Dabei werden dann möglicherweise für jeden Knoten sämtliche Fälle überprüft. Dies ist also laufzeittechnisch sehr problematisch. Wenn wir jedoch immer direkt den ersten Knoten wählen, welchen wir mit minimalem Grad finden (beispielsweise bei einer sortierten Liste der Knoten im Graph ist dies immer der erste Knoten in der Liste, also eine Suchzeit von $\mathcal{O}(1)$). Dann müssen wir für diesen Knoten überprüfen, welchen Fall er erfüllt, und dann für alle seine Nachbarn vom selben Grad, ob einer von ihnen einen früheren Fall erfüllt. Sollte dies der Fall sein, so müssen wir erneut für die Nachbarn nun dieses Knotens überprüfen, ob einer der Nachbarn vom selben Grad einen früheren Fall erfüllt. Allerdings ist zum einen die Wahrscheinlichkeit, dass ein solcher Nachbar existiert um so geringer, je früher der Fall in der ursprünglichen Reihenfolge auftaucht, und zum anderen ist die Zahl der Regeln endlich, also müssen höchstens (Anzahl der Regeln für diesen Grad)-mal solche Nachbarn überprüft werden. Damit kann es im Worst-Case zwar immer noch vorkom-

men, dass wird sämtliche Knoten mit minimalem Grad kontrollieren müssen, dies kann allerdings nur noch bei sehr kleinen (Rest-)Graphen vorkommen.
Allerdings ist es möglich, dass, obwohl kein Knoten, welcher eine vorherige Regel erfüllt, direkt beeinflusst werden darf, ein Knoten am Rand durch das Entfernen von Nachbarn indirekt beeinflusst wird. Betrachte hierzu folgendes Beispiel:



Sei der Knoten x der gefundene Knoten. Dieser ist kein Teil eines Dreiecks und auch keiner seiner Nachbarknoten ist Teil eines Dreiecks. Daher wird nun die Regel 4 für Knoten vom Grad 3 angewendet. Seien nun die Knoten y und z die Knoten, welche als Paar entfernt werden, um den Grad von x zu reduzieren. Dies ist eine Faktor 2 Approximation. Der Knoten a war jedoch vorher ebenfalls vom Grad 3, und Teil eines Dreiecks. Wäre die korrekte Reihenfolge eingehalten worden, so wären zunächst die Knoten a, b und z als Dreieck entfernt worden. Dies wäre eine Faktor $1,5$ Approximation.
Hierbei ist nicht zwangsweise gegeben, dass die gefundene Lösung automatisch schlechter wird. Allerdings sind derartige Beeinflussungen möglich und ändern somit auch die gefundene Lösung. Insgesamt ist der Gewinn in der Laufzeit jedoch insbesondere für größere Graphen so signifikant, dass die nicht strikte Implementierung bevorzugt werden sollte, da sie nicht signifikant schlechtere Ergebnisse liefert, aber eben um bis zu Faktor n schneller ist.
Weitere Aspekte, die bei der Implementierung eine Rolle spielen, betreffen insbesondere die Komplexität beziehungsweise Verwendbarkeit des Algorithmus. Die Daten- und Organisationsstrukturen für den Grundalgorithmus können sehr viel simpler sein als die für eine optimierte Version für sämtliche Graphen.
Beispielsweise ist bei Graphen mit Durchschnittsgrad ≤ 4 ein Sortieren der Knoten vom Grad ≥ 5 nicht unbedingt nötig. Zum einen werden die Ausgleichsschritte so selten wie möglich ausgeführt, also ist der Zugriff auf diese Knotenmenge eher selten nötig, und zum anderen ist es auch hier möglich, einfach nur den ersten Knoten für den Ausgleichsschritt zu wählen, um somit eine möglichst schnelle Implementierung zu erreichen. Soll der Algorithmus allerdings für einen beliebigen Durchschnittsgrad funktionieren, so ist es ratsam, eine Struktur zu implementieren, welche zumindest in der Lage ist, alle Knoten mit Grad $\leq$ Durchschnittsgrad des Graphen zu sortieren, um einen schnellen Zugriff auf diese zu ermöglichen.
Eine andere Sache ist die Hilfsliste. In der ursprünglichen Version ist es nicht möglich, dass ein Knoten zu mehr als einem Knoten in der Hilfsliste benachbart war. Es ist also nicht nötig, eine vollständige Struktur für die Hilfsliste zu implementieren. Vielmehr können wir eine simple Hilfsstruktur verwenden, welche einen Verweis auf den Knoten

in der Hilfsliste enthält, sowie eine (beispielsweise boolsche) Information, ob der Knoten bereits versorgt wurde oder nicht. Die Nachbarn des Knotens in der Hilfsliste erhalten also nur einen Verweis auf diese Struktur, und wenn einer dieser Knoten in die Überdeckung gelegt wird, so wird zunächst nur überprüft, ob die Hilfsstruktur aussagt, ob der Knoten bereits behandelt wurde oder nicht. Wenn der Knoten noch nicht behandelt wurde, so lege ihn dann ebenfalls in die Überdeckung, und markiere ihn über die Hilfsstruktur als erledigt. Soll ein Knoten, welcher bereits einen solchen Verweis besitzt, einen weiteren Verweis erhalten, also ein weiterer Nachbar in die Hilfsliste gelegt werden, so gibt es zwei Möglichkeiten:

1. Der Knoten aus dem ursprünglichen Verweis wurde bereits behandelt. In diesem Fall ersetze einfach den alten durch den neuen Verweis.
2. Der Knoten aus dem ursprünglichen Verweis wurde noch nicht behandelt. In diesem Fall greift unsere Regel, und sowohl der Knoten aus dem alten, wie auch der aus dem neuen Verweis werden direkt in die Überdeckung gelegt, zusammen mit dem Knoten bei dem dies festgestellt wurde. Dann werden beide Verweise als erledigt markiert.

Verwenden wir jedoch die Variante, bei der ein Knoten zunächst zu beliebig vielen Knoten in der Hilfsliste benachbart gewesen sein darf, so ist diese simple Datenstruktur nicht länger möglich. In diesem Fall ist es nötig, dass zum einen jeder Knoten die Möglichkeit erhält, eine beliebige Menge von Verweisen auf Elemente in der Hilfsliste zu erhalten, und zum anderen dass jedes Element in der Hilfsliste in der Lage ist, alle potentiellen Partner, welche sich noch im Graphen befinden, zu kennen. Daher ist es hierbei nötig, eine wesentlich komplexere Datenstruktur zu nutzen, welche es ermöglicht, einen Knoten aus der Hilfsliste aktiv bei Knoten im Graphen zu entfernen, um sicherzustellen, dass die Knoten in den Verweisen auch tatsächlich noch in der Hilfsliste liegen, und zum anderen, um die Anzahl der Nachbarn in der Hilfsliste für jeden potentiellen Partner zu erfragen, um festzustellen, ob eine Regel getriggert werden muss, oder ein potentieller Partner aus dem Graph entfernt werden darf.
Ein weiterer interessanter Aspekt der Implementierung ist die Frage der Parallelisierbarkeit. Eine generelle Parallelisierbarkeit ist zunächst nicht gegeben, da die generelle Idee der Reduktionsalgorithmen vorraussetzt, dass immer eine Regel ausgeführt wird, und danach die nächste Regel den Graphen betrachtet, welcher nach Abschluss der vorherigen Regel erzeugt wurde. Damit ändert sich nach jeder Regel der zu betrachtende Graph und es können keine zwei Regeln auf dem selben Graphen betrachtet werden. Es gibt jedoch verschiedene Teile des Algorithmus, die sich durchaus parallelisieren lassen wobei sich hierbei die Frage ergibt, ob sich dies lohnt.
Ein Aspekt, der sich Paralellisieren ließe, wären auf jeden Fall Teilgraphen. Insbesondere im späteren Verlauf des Algorithmus werden durch das Entfernen der Knoten häufig Teilgraphen entstehen. Da das entfernen von Knoten aus einem Teilgraphen keinen Knoten aus einem anderen Teilgraphen beeinflussen kann, wäre hier eine parallele Anwendung der Regeln auf beide Graphen möglich. Allerdings setzt dies voraus, dass der Algorithmus weiß, dass Teilgraphen existieren. Es müsste also wiederum eine zusätzliche Überprüfung stattfinden, welche solche Teilgraphen aufspürt, und darauf eine parallele Anwendung

startet. Dies kostet jedoch wiederum Laufzeit. Daher müsste man auch hier abwägen, ob sich der zusätzliche Aufwand lohnt. Dies hängt stark von Größe und Struktur der zu betrachtenden Graphen ab. Dichte Graphen zerfallen beispielsweise sehr viel weniger wahrscheinlich in Teilgraphen als bestimmte andere Graphen. Außerdem ist auch die zur Verfügung stehende Hardware entscheidend, da eine Parallelisierung auf zwei Kerne beispielsweise sehr viel weniger wirkungsvoll wäre als auf 16 oder mehr Kerne.
Dieselbe Abwägung ist auch für den zweiten Aspekt nötig, die Überprüfung der einzelnen Regeln. Oftmals müssen wir beispielsweise überprüfen, ob Knoten Teil von Dreiecken sind. Dies ließe sich problemlos auf verschiedene Threads aufteilen, allerdings wäre möglicherweise der Aufwand des Verteilens größer, als die Überprüfung einfach auf einem Kern auszuführen.
Die interessanteste Möglichkeit wäre allerdings die Überprüfung der Regeln bei strikter Anwendung der Regelreihenfolge. In diesem Fall könnte der Nachteil, dass wir eine große Anzahl von Knoten durchlaufen müssen, um den Richtigen zu finden, deutlich reduziert werden, wenn wir die Knoten auf mehreren Kernen überprüfen lassen. Da wir bei der Überprüfung der Knoten auch keine Änderungen am Graphen vornehmen, sondern erst bei Ausführung der Regel, stellt dies kein Problem dar. Bei genügend großer Anzahl von Kernen kann also die parallelisierte strikte Regelanwendung sehr interessant werden.

5 Praktische Beispiele

Wir betrachten zunächst zwei spezielle Arten von Graphen, und zeigen für diese den erwarteten Faktor. Danach betrachten wir einige beispielhafte Ergebnisse einer Implementierung des Algorithmus.

Die erste Art Graphen sind Bäume. Hierbei handelt es sich um Kreisfreie Graphen mit Durchschnittsgrad 2. Für diese Graphen findet der Algorithmus immer eine Lösung mit Approximationsfaktor 1. Die gefundene Lösung ist also immer optimal. Dies folgt direkt daraus, dass wir ausschließlich Grad-1-Regeln anwenden zur Lösung solcher Graphen. Hierbei gilt, dass jeweils die Äste in der untersten Ebene des Baumes die Blätter mit Grad 1 haben, und ansonsten nur mit genau einem anderen Ast-Knoten verbunden sind. Daher wird immer entweder die Regel 2 oder die Regel 6 angewendet, und dadurch neue Blätter erzeugt. Allerdings kann es passieren, dass eine nicht strikte Umsetzung des Algorithmus hierbei einen fehlerhaften Approximationsfaktor ausgibt. Da hierbei nicht garantiert ist, dass ausschließlich Äste auf der untersten Ebene des Baumes betrachtet werden beziehungsweise solche Knoten die Regel 2 erfüllen, ist es möglich dass eine Regel 7 zur Anwendung kommt. Diese ist in diesem Fall trotzdem minimal, allerdings ist unser Algorithmus nicht in der Lage dies zu erkennen, sondern misst den Schritt hierbei mit Faktor $1,5$. Die gefundene Lösung ist trotzdem minimal, da die nicht strikte Anwendung trotzdem nicht zulässt, dass minimale Fälle zerstört werden.

Die zweite Art von Graphen die für uns interessant ist sind Worst-Case Graphen. Also solche Graphen, bei denen tatsächlich der schlecht möglichste Faktor erreicht wird. Für

Graphen mit Durchschnittsgrad ≤ 4 und minimalem Grad 2 ist dies nur für 3- oder 4-reguläre dreiecksfreie Graphen möglich. Diese wenden nur die Regel 4 für Knoten mit Grad 3 an, und erzeugen damit immer abgedeckte Knoten vom Grad 2. Werden für diese ausschließlich die Regeln 3, 8 oder 9 für Knoten vom Grad 2 angewendet, so entsteht eine Lösung mit Approximationsfaktor $\frac{5}{3}$. für alle anderen Graphen erzeugt der Algorithmus eine Lösung mit einem besseren Approximationsfaktor. Diese Graphen sind also möglich, und auch nicht schwer zu konstruieren, allerdings ist das Auftreten solcher Graphen als Zufallsgraph sehr unwahrscheinlich. Dementsprechend ist das erreichen des schlecht möglichsten Approximationsfaktors also zwar Möglich, aber eben nicht sehr Wahrscheinlich. In den meisten Fällen werden wir entsprechend bessere Approximationsfaktoren erzielen.

Zum Abschluss wollen wir nun ein paar Ergebnisse betrachten, die der Algorithmus auf Zufallsgraphen erreicht.
Ein beispielhafte Implementierung einer nicht strikten Regelumsetzung findet man unter folgendem Link:

https://github.com/S4flkrus/MA.git

Betrachten wir nun eine beispielhafte Ausgabe des Algorithmus für Zufallsgraphen mit Durchschnittsgrad zwischen 2 und 14:

vertices: 1024 edges: 1024 average degree: 2
vertices in the cover: 495 faktor: 1 Runtime1: 0ms
vertices: 1024 edges: 1536 average degree: 3
vertices in the cover: 575 faktor: 1.20545 Runtime1: 0ms
vertices: 1024 edges: 2048 average degree: 4
vertices in the cover: 513 faktor: 1.40164 Runtime1: 0ms
vertices: 1024 edges: 2560 average degree: 5
vertices in the cover: 400 faktor: 1.44404 Runtime1: 0ms
vertices: 1024 edges: 3072 average degree: 6
vertices in the cover: 371 faktor: 1.44358 Runtime1: 0ms
vertices: 1024 edges: 3584 average degree: 7
vertices in the cover: 351 faktor: 1.43852 Runtime1: 0ms
vertices: 1024 edges: 4096 average degree: 8
vertices in the cover: 310 faktor: 1.42202 Runtime1: 0ms
vertices: 1024 edges: 4608 average degree: 9
vertices in the cover: 378 faktor: 1.39483 Runtime1: 0ms
vertices: 1024 edges: 5120 average degree: 10
vertices in the cover: 436 faktor: 1.42951 Runtime1: 15ms
vertices: 1024 edges: 5632 average degree: 11
vertices in the cover: 424 faktor: 1.41806 Runtime1: 0ms
vertices: 1024 edges: 6144 average degree: 12
vertices in the cover: 514 faktor: 1.46439 Runtime1: 15ms

vertices: 1024 edges: 6656 average degree: 13
vertices in the cover: 555 faktor: 1.48794 Runtime1: 0ms
vertices: 1024 edges: 7168 average degree: 14
vertices in the cover: 556 faktor: 1.51087 Runtime1: 0ms

vertices: 2048 edges: 2048 average degree: 2
vertices in the cover: 999 faktor: 1 Runtime1: 0ms
vertices: 2048 edges: 3072 average degree: 3
vertices in the cover: 1118 faktor: 1.18936 Runtime1: 0ms
vertices: 2048 edges: 4096 average degree: 4
vertices in the cover: 993 faktor: 1.4206 Runtime1: 0ms
vertices: 2048 edges: 5120 average degree: 5
vertices in the cover: 726 faktor: 1.49691 Runtime1: 0ms
vertices: 2048 edges: 6144 average degree: 6
vertices in the cover: 601 faktor: 1.51005 Runtime1: 0ms
vertices: 2048 edges: 7168 average degree: 7
vertices in the cover: 562 faktor: 1.49468 Runtime1: 15ms
vertices: 2048 edges: 8192 average degree: 8
vertices in the cover: 496 faktor: 1.4806 Runtime1: 0ms
vertices: 2048 edges: 9216 average degree: 9
vertices in the cover: 563 faktor: 1.46615 Runtime1: 0ms
vertices: 2048 edges: 10240 average degree: 10
vertices in the cover: 615 faktor: 1.45735 Runtime1: 0ms
vertices: 2048 edges: 11264 average degree: 11
vertices in the cover: 676 faktor: 1.46638 Runtime1: 0ms
vertices: 2048 edges: 12288 average degree: 12
vertices in the cover: 641 faktor: 1.50469 Runtime1: 8ms
vertices: 2048 edges: 13312 average degree: 13
vertices in the cover: 768 faktor: 1.49126 Runtime1: 10ms
vertices: 2048 edges: 14336 average degree: 14
vertices in the cover: 787 faktor: 1.60941 Runtime1: 10ms

vertices: 4096 edges: 4096 average degree: 2
vertices in the cover: 1967 faktor: 1 Runtime1: 0ms
vertices: 4096 edges: 6144 average degree: 3
vertices in the cover: 2255 faktor: 1.16718 Runtime1: 0ms
vertices: 4096 edges: 8192 average degree: 4
vertices in the cover: 1899 faktor: 1.43105 Runtime1: 0ms
vertices: 4096 edges: 10240 average degree: 5
vertices in the cover: 1414 faktor: 1.55899 Runtime1: 3ms
vertices: 4096 edges: 12288 average degree: 6
vertices in the cover: 1104 faktor: 1.60232 Runtime1: 5ms
vertices: 4096 edges: 14336 average degree: 7
vertices in the cover: 955 faktor: 1.56301 Runtime1: 6ms

vertices: 4096 edges: 16384 average degree: 8
vertices in the cover: 838 faktor: 1.59013 Runtime1: 10ms
vertices: 4096 edges: 18432 average degree: 9
vertices in the cover: 815 faktor: 1.59491 Runtime1: 10ms
vertices: 4096 edges: 20480 average degree: 10
vertices in the cover: 857 faktor: 1.53036 Runtime1: 10ms
vertices: 4096 edges: 22528 average degree: 11
vertices in the cover: 799 faktor: 1.53065 Runtime1: 10ms
vertices: 4096 edges: 24576 average degree: 12
vertices in the cover: 973 faktor: 1.57189 Runtime1: 12ms
vertices: 4096 edges: 26624 average degree: 13
vertices in the cover: 1068 faktor: 1.6036 Runtime1: 19ms
vertices: 4096 edges: 28672 average degree: 14
vertices in the cover: 1199 faktor: 1.65608 Runtime1: 30ms

vertices: 8192 edges: 8192 average degree: 2
vertices in the cover: 3952 faktor: 1.00025 Runtime1: 0ms
vertices: 8192 edges: 12288 average degree: 3
vertices in the cover: 4403 faktor: 1.18679 Runtime1: 8ms
vertices: 8192 edges: 16384 average degree: 4
vertices in the cover: 3713 faktor: 1.46354 Runtime1: 10ms
vertices: 8192 edges: 20480 average degree: 5
vertices in the cover: 2679 faktor: 1.57311 Runtime1: 10ms
vertices: 8192 edges: 24576 average degree: 6
vertices in the cover: 1949 faktor: 1.60941 Runtime1: 10ms
vertices: 8192 edges: 28672 average degree: 7
vertices in the cover: 1552 faktor: 1.63025 Runtime1: 20ms
vertices: 8192 edges: 32768 average degree: 8
vertices in the cover: 1342 faktor: 1.63659 Runtime1: 18ms
vertices: 8192 edges: 36864 average degree: 9
vertices in the cover: 1245 faktor: 1.61061 Runtime1: 20ms
vertices: 8192 edges: 40960 average degree: 10
vertices in the cover: 1186 faktor: 1.61801 Runtime1: 30ms
vertices: 8192 edges: 45056 average degree: 11
vertices in the cover: 1230 faktor: 1.60365 Runtime1: 30ms
vertices: 8192 edges: 49152 average degree: 12
vertices in the cover: 1353 faktor: 1.62425 Runtime1: 50ms
vertices: 8192 edges: 53248 average degree: 13
vertices in the cover: 1701 faktor: 1.71472 Runtime1: 68ms
vertices: 8192 edges: 57344 average degree: 14
vertices in the cover: 2006 faktor: 1.72485 Runtime1: 80ms

vertices: 16384 edges: 16384 average degree: 2
vertices in the cover: 7828 faktor: 1 Runtime1: 4ms

vertices: 16384 edges: 24576 average degree: 3
vertices in the cover: 9055 faktor: 1.17674 Runtime1: 2ms
vertices: 16384 edges: 32768 average degree: 4
vertices in the cover: 7540 faktor: 1.46893 Runtime1: 10ms
vertices: 16384 edges: 40960 average degree: 5
vertices in the cover: 5536 faktor: 1.59035 Runtime1: 20ms
vertices: 16384 edges: 49152 average degree: 6
vertices in the cover: 3627 faktor: 1.67606 Runtime1: 30ms
vertices: 16384 edges: 57344 average degree: 7
vertices in the cover: 3020 faktor: 1.69663 Runtime1: 40ms
vertices: 16384 edges: 65536 average degree: 8
vertices in the cover: 2440 faktor: 1.72682 Runtime1: 50ms
vertices: 16384 edges: 73728 average degree: 9
vertices in the cover: 2032 faktor: 1.72642 Runtime1: 68ms
vertices: 16384 edges: 81920 average degree: 10
vertices in the cover: 1890 faktor: 1.72603 Runtime1: 81ms
vertices: 16384 edges: 90112 average degree: 11
vertices in the cover: 1888 faktor: 1.7009 Runtime1: 89ms
vertices: 16384 edges: 98304 average degree: 12
vertices in the cover: 2300 faktor: 1.74639 Runtime1: 131ms
vertices: 16384 edges: 106496 average degree: 13
vertices in the cover: 2782 faktor: 1.77085 Runtime1: 190ms
vertices: 16384 edges: 114688 average degree: 14
vertices in the cover: 3428 faktor: 1.81568 Runtime1: 251ms

Diese Ergebnisse sind nur beispielhaft, geben aber einen Schnitt von Ergebnissen wieder. Wir können hieraus eine Reihe interessanter Folgerungen ableiten.
Zum einen ist zu sehen, dass auch für Zufallsgraphen mit Durchschnittsgrad 2 fast immer eine exakte Lösung gefunden wird. Dies ist bereits ebenso bei dem Algorithmus von [1], und setzt sich also für totale Knotenüberdeckung fort.
Außerdem sind auch die Graphen mit Durchschnittsgrad > 10 trotzdem noch nahe an dem ursprünglichen Zielfaktor von $\frac{5}{3}$. Dies zeigt die Vermutung, dass es sich lohnen kann den Algorithmus auch auf Graphen mit Durchschnittsgrad ≥ 4 anzuwenden ohne weitere Modifikationen.
Allerdings ist der von diesem Algorithmus erzielte Faktor im Durchschnitts signifikant höher als bei der Implementierung des Algorithmus von [1] aus [2]. Dies entsteht vor allem, da es nicht möglich ist, alle Fälle optimal zu lösen, welche der Algorithmus für die normale Knotenüberdeckung Problemlos minimal lösen kann. Allerdings ist hierbei oftmals nur das Monitoring schlechter, weil die Regeln nicht als minimal angenommen werden können, obwohl sie es sind. Ein interessanter Vergleich wären noch die optimierten Versionen der beiden Algorithmen.
Da die Graphen jeweils 2^x Knoten enthalten, lassen sich auch erste Schlüsse auf die reale Laufzeit ziehen. Hierbei kann man mit zusätzlichen Daten ablesen, dass der Algorithmus

jeweils bezüglich der Anzahl der Knoten und der Anzahl der Kanten eine Laufzeit von $\mathcal{O}(n)$ hat.
Betrachten wir als letztes noch eine beispielhafte Ausgabe für größere Graphen mit vollständigem Monitoring:

vertices: 65536 edges: 131072 average degree: 4
vertices in the cover: 28902 faktor: 1.48253
Grad 1 Fall 1: 1 Grad 1 Fall 2: 2482 Grad 1 Fall 3: 0 Grad 1 Fall 4: 0 Grad 1 Fall 5: 0 Grad 1 Fall 6: 390 Grad 1 Fall 7: 0 Grad 1 Fall 8: 6030
Grad 2 Fall 1: 15 Grad 2 Fall 2: 1 Grad 2 Fall 3: 15595 Grad 2 Fall 4: 6532 Grad 2 Fall 5: 0 Grad 2 Fall 6: 8 Grad 2 Fall 7: 68 Grad 2 Fall 8: 0 Grad 2 Fall 9: 10
Grad 3 Fall 1: 0 Grad 3 Fall 2: 5 Grad 3 Fall 3: 1 Grad 3 Fall 3: 1093 Grad 4 Fall 1: 0 Grad 4 Fall 2: 0 Ausgleichsschritte: 0
Runtime1: 80ms

vertices: 131072 edges: 262144 average degree: 4
vertices in the cover: 57795 faktor: 1.48558
Grad 1 Fall 1: 1 Grad 1 Fall 2: 5073 Grad 1 Fall 3: 0 Grad 1 Fall 4: 0 Grad 1 Fall 5: 3 Grad 1 Fall 6: 803 Grad 1 Fall 7: 0 Grad 1 Fall 8: 12081
Grad 2 Fall 1: 8 Grad 2 Fall 2: 2 Grad 2 Fall 3: 31190 Grad 2 Fall 4: 13017 Grad 2 Fall 5: 0 Grad 2 Fall 6: 6 Grad 2 Fall 7: 66 Grad 2 Fall 8: 0 Grad 2 Fall 9: 6
Grad 3 Fall 1: 1 Grad 3 Fall 2: 8 Grad 3 Fall 3: 0 Grad 3 Fall 3: 2308 Grad 4 Fall 1: 0 Grad 4 Fall 2: 0 Ausgleichsschritte: 0
Runtime1: 193ms

vertices: 262144 edges: 524288 average degree: 4
vertices in the cover: 114668 faktor: 1.48221
Grad 1 Fall 1: 2 Grad 1 Fall 2: 10298 Grad 1 Fall 3: 0 Grad 1 Fall 4: 0 Grad 1 Fall 5: 0 Grad 1 Fall 6: 1565 Grad 1 Fall 7: 0 Grad 1 Fall 8: 24335
Grad 2 Fall 1: 18 Grad 2 Fall 2: 2 Grad 2 Fall 3: 62260 Grad 2 Fall 4: 26193 Grad 2 Fall 5: 0 Grad 2 Fall 6: 12 Grad 2 Fall 7: 70 Grad 2 Fall 8: 0 Grad 2 Fall 9: 6
Grad 3 Fall 1: 0 Grad 3 Fall 2: 9 Grad 3 Fall 3: 2 Grad 3 Fall 3: 4467 Grad 4 Fall 1: 0 Grad 4 Fall 2: 0 Ausgleichsschritte: 0
Runtime1: 437ms

vertices: 524288 edges: 1048576 average degree: 4
vertices in the cover: 229894 faktor: 1.48315
Grad 1 Fall 1: 1 Grad 1 Fall 2: 21008 Grad 1 Fall 3: 0 Grad 1 Fall 4: 0 Grad 1 Fall 5: 3 Grad 1 Fall 6: 3055 Grad 1 Fall 7: 0 Grad 1 Fall 8: 48859
Grad 2 Fall 1: 18 Grad 2 Fall 2: 3 Grad 2 Fall 3: 124061 Grad 2 Fall 4: 52522 Grad 2 Fall 5: 1 Grad 2 Fall 6: 11 Grad 2 Fall 7: 88 Grad 2 Fall 8: 0 Grad 2 Fall 9: 8
Grad 3 Fall 1: 0 Grad 3 Fall 2: 4 Grad 3 Fall 3: 0 Grad 3 Fall 3: 8984 Grad 4 Fall 1: 0 Grad 4 Fall 2: 0 Ausgleichsschritte: 0
Runtime1: 969ms

vertices: 1048576 edges: 2097152 average degree: 4
vertices in the cover: 459889 faktor: 1.48485
Grad 1 Fall 1: 8 Grad 1 Fall 2: 41698 Grad 1 Fall 3: 0 Grad 1 Fall 4: 0 Grad 1 Fall 5: 1 Grad 1 Fall 6: 6087 Grad 1 Fall 7: 0 Grad 1 Fall 8: 97345
Grad 2 Fall 1: 19 Grad 2 Fall 2: 2 Grad 2 Fall 3: 248845 Grad 2 Fall 4: 104961 Grad 2 Fall 5: 0 Grad 2 Fall 6: 10 Grad 2 Fall 7: 107 Grad 2 Fall 8: 0 Grad 2 Fall 9: 9
Grad 3 Fall 1: 0 Grad 3 Fall 2: 11 Grad 3 Fall 3: 1 Grad 3 Fall 3: 18136 Grad 4 Fall 1: 0 Grad 4 Fall 2: 0 Ausgleichsschritte: 0
Runtime1: 2109ms

vertices: 2097152 edges: 4194304 average degree: 4
vertices in the cover: 915861 faktor: 1.48467
Grad 1 Fall 1: 9 Grad 1 Fall 2: 82756 Grad 1 Fall 3: 0 Grad 1 Fall 4: 0 Grad 1 Fall 5: 3 Grad 1 Fall 6: 12345 Grad 1 Fall 7: 0 Grad 1 Fall 8: 192614
Grad 2 Fall 1: 20 Grad 2 Fall 2: 1 Grad 2 Fall 3: 499395 Grad 2 Fall 4: 209678 Grad 2 Fall 5: 0 Grad 2 Fall 6: 9 Grad 2 Fall 7: 123 Grad 2 Fall 8: 0 Grad 2 Fall 9: 8
Grad 3 Fall 1: 0 Grad 3 Fall 2: 5 Grad 3 Fall 3: 0 Grad 3 Fall 3: 36641 Grad 4 Fall 1: 0 Grad 4 Fall 2: 0 Ausgleichsschritte: 0
Runtime1: 4672ms

Wir sehen, dass der Algorithmus auch weiterhin sehr effektiv arbeitet. Außerdem können wir erkennen, dass die Laufzeit des Algorithmus bei ähnlich bleibender Komplexität des Graphen sogar nur $\mathcal{O}(n)$ beträgt. Allerdings ist dies eine laufzeitoptimierte Version des Algorithmus, also für anders optimierte Varianten kann die anders aussehen.
Anhand des Monitorings können wir auch für jeden Graphen genau ablesen, welche der Regeln verwendet wurden, um die vollständige Überdeckung zu erzeugen.

6 Fazit

Wir haben in dieser Arbeit gezeigt, dass ein Reduktionsalgorithmus existiert, welcher eine approximierte Lösung für das totale Knotenüberdeckungsproblem mit überwachbarer Güte in Polynomialzeit liefert. Um jedoch der Totalität gerecht zu werden sind eine ganze Reihe von Anpassungen nötig, um sicherzustellen, dass die Bedingung für alle Knoten erfüllt wird, ohne dabei übermäßig viele überflüssige Knoten in die approximierte Überdeckung zu legen. Mit der vorhandenen Implementierung lässt sich zeigen, dass der Algorithmus tatsächlich funktioniert, und sich auch tatsächlich nutzbar machen lässt.
Auch ergeben sich aus der Existenz eines solchen Algorithmus verschiedene Fragen, die eine spätere Betrachtung wert sein könnten.
Zum einen ist die Frage, ob sich die Grundidee eines Reduktionsalgorithmus mit überwachbarer Güte auch auf andere Probleme anwenden lässt. Die Existenz für das totale

Knotenüberdeckungsproblem zeigt beispielsweise, dass auch Probleme ohne Polynomiellen Kern funktionieren.
Außerdem kann man betrachten, inwieweit sich der Algorithmus insbesondere für Graphen mit höherem Durchschnittsgrad erweitern lässt. In der aktuellen Version werden für Graphen mit Durchschnittsgrad > 4 ausschließlich Reduktionsschritte betrachtet, um Knoten vom Grad < 4 zu erreichen. Allerdings könnte es möglich sein, dass weitere Regeln existieren, welche sich für Knoten mit höherem Grad anwenden lassen, und somit die Approximation dieser Probleme deutlich verbessern können.
Auch die Anwendbarkeit von Ergebnissen dieser Arbeit auf andere NP-Schwere Graphenprobleme könnte einer Betrachtung wert sein.

Literatur

[1] L. Brankovic, H. Fernau, A novel parameterised approximation algorithm for minimum vertex cover, in: Theoretical Computer Science 511 (2013) 85–108

[2] F. Kruschewski-Kursawe, Datenreduktionsbasierte Approximationsalgorithmen für das Knotenüberdeckungsproblem: Implementierung und Analyse, Bachelorarbeit Universität Trier (2016)

Erklärung zur Masterarbeit

Hiermit erkläre ich, dass ich die Masterarbeit selbstständig 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 nicht veröffentlicht.

Ort, Datum

Unterschrift