Effiziente Approximation unabhängiger Mengen in Graphen

Diplomarbeit

Technische Universität Chemnitz

Professur für Theoretische Informatik und Informationssicherheit
Prof. Dr. Hanno Lefmann

Diplomarbeit

Effiziente Approximation unabhängiger Mengen in Graphen

cand. Inf. Matthias Baumgart
Matrikelnummer 22445

Chemnitz, 14. September 2004

Effiziente Approximation unabhängiger Mengen in Graphen
Baumgart, Matthias. -89 S., -10 Algorithmen, -7 Abb., -4 Tab.
Chemnitz: Technische Universität Chemnitz, Fakultät für Informatik, Diplomarbeit.

eingereicht von:	Matthias Baumgart
	geboren am 24. März 1980 in Mittweida

ausgegeben am:	1. April 2004
eingereicht am:	14. September 2004

Betreuer:	Prof. Dr. Hanno Lefmann
	Dr. Ulrich Tamm

Aufgabenstellung

Thema: Effiziente Approximation unabhängiger Mengen in Graphen

Die Approximierbarkeit der Unabhängigkeitszahl von Graphen in Polynomialzeit soll quantitativ untersucht werden. Speziell sind hierbei die Ergebnisse von Halldórsson und Boppana sowie die neuere Arbeit von U. Feige zu dieser Thematik auszuarbeiten, zu bewerten und zu analysieren. Des Weiteren sind die Ergebnisse von Alon und Kahale hinsichtlich des Einsatzes der Thetafunktion auszuarbeiten. Auch soll die effiziente Approximierbarkeit der Unabhängigkeitszahl in ausgewählten Graphenklassen sowie die Online Situation betrachtet und analysiert werden.

Für ausgewählte Verfahren sollen darüber hinaus praktische Performancetests durchgeführt werden.

Chemnitz, 1. April 2004

INHALTSVERZEICHNIS

1 EINLEITUNG

Eine unabhängige Menge in einem Graphen $G = (V, E)$ ist eine Teilmenge $I \subseteq V$ der Knotenmenge V mit paarweise nicht benachbarten Knoten. Im Allgemeinen sucht man nicht beliebige unabhängige Mengen sondern unabhängige Mengen maximaler Kardinalität, das heißt eine Lösung für das Problem INDEPENDENT SET. Das Bestimmen einer solchen Lösung für einen gegebenen Graphen ist ein wesentliches Problem in der Informatik und ein wichtiges Forschungsgebiet in der Kombinatorik und Graphentheorie.

Anfang der siebziger Jahre des vergangenen Jahrhunderts gelang Cook [8] ein Durchbruch in der Komplexitätstheorie, indem er nachwies, dass das Problem SATISFIABILITY oder kurz SAT $\mathcal{NP}$-hart ist. Später veröffentlichten Karp [28] sowie Garey und Johnson [12] polynomielle Reduktionen des Problems SAT auf eine Reihe anderer kombinatorischer Probleme darunter auch INDEPENDENT SET, um zu zeigen, dass diese ebenfalls $\mathcal{NP}$-hart sind. Man weiß aus komplexitätstheoretischen Betrachtungen, dass für jedes dieser Probleme ein Polynomialzeitalgorithmus existiert, wenn bereits für ein beliebiges $\mathcal{NP}$-hartes Problem ein Algorithmus polynomieller Laufzeit angegeben werden kann. Da in über 30 Jahren für keines dieser Probleme ein effizienter Algorithmus gefunden werden konnte, vermutet man, dass es für $\mathcal{NP}$-harte Probleme keine Polynomialzeitverfahren gibt.

Um dennoch unabhängige Mengen großer Kardinalität bestimmen zu können, versucht man gute Lösungen zu approximieren, indem man clevere Heuristiken angibt, die in Polynomialzeit eine in der Praxis ausreichende Näherung an das Optimum berechnen. Ein Maß für die Qualität einer von einem Approximationsalgorithmus gelieferten Lösung ist dessen Güte, welche als worst-case Verhältnis von Wert der Optimallösung und Wert der approximierten Lösung definiert ist.

Erste Approximationsalgorithmen für INDEPENDENT SET wurden von Johnson bereits 1974 analysiert [26], wobei er zeigte, dass diese Algorithmen für allgemeine Graphen auf n Knoten nur eine Approximationsgüte von $O(n)$ haben [18]. Mit Hilfe eines Färbungsalgorithmus wurde die Approximationsgüte 1983 zu $O(n/\log n)$ verbessert [45]. Eine weitere Verbesserung erzielten Boppana und Halldórsson mit dem Algorithmus CLIQUE-REMOVAL [7] im Jahre 1992. Die Approximationsgüte dieses Algorithmus beträgt $O(n/(\log n)^2)$. Kürzlich gelang es Feige, diese zu $O(n(\log\log n)^2/(\log n)^3)$ zu verbessern [11]. Die beiden zuletzt genannten Verfahren beruhen dabei auf dem Prinzip der iterativen Entfernung von bestimmten Untergraphen, um eine unabhängige Menge zu approximieren. Eine andere Vorgehensweise verwenden Goemans

und Williamson in ihrem 1994 veröffentlichten Approximationsalgorithmus für das Problem MAX CUT [13, 34], indem sie das Prinzip der semidefiniten Programmierung [42] nutzen. Eine Anwendung dieses Prinzips auf das Problem COLORING [27, 30] und schließlich auch auf das Problem INDEPENDENT SET [3] folgten, wobei der Algorithmus zum letztgenannten Problem zusätzlich noch auf die sogenannte Lovász ϑ-Funktion [35] zurückgreift.

Neben der Optimierung der Güte von Approximationsalgorithmen zur Bestimmung einer bezüglich Kardinalität maximalen unabhängigen Menge besteht ebenfalls das Bestreben, die Laufzeit zur exakten Berechnung der Unabhängigkeitszahl beziehungsweise einer unabhängigen Menge maximaler Kardinalität zu optimieren. Den ersten Algorithmus zur Lösung dieses Problems mit einer Laufzeit geringer als $O(2^n)$ veröffentlichten Tarjan und Trojanowski [40] 1977. Die Laufzeit dieses Verfahrens ist $O(1.2599^n)$. Unabhängig voneinander entwickelten Jian [25] und Robson [38] im Jahre 1986 weitere Verbesserungen, wobei der von Jian entwickelte Algorithmus eine Laufzeit von $O(1.2345^n)$ erreicht, während der Algorithmus von Robson eine Laufzeitkomplexität von $O(1.2243^n)$ aufweist. Neuere Algorithmen stammen von Beigel [5] oder Robson [39], wobei letzterer mit einer Laufzeit von $O(1.1889^n)$ einen exponentiellen Speicherplatzbedarf besitzt. Diese Entwicklung sei aber nur am Rande erwähnt und ist nicht Inhalt der vorliegenden Diplomarbeit.

Zur Motivation unserer Betrachtungen sollte auch ein praktischer Aspekt der Approximation von unabhängigen Mengen in Graphen nicht außer Acht gelassen werden. Bei dem Problem der Verteilung von Ressourcen an einzelne Prozesse tritt dieses Problem meist bei Betriebssystemen im Rahmen der Optimierung eines Schedules auf. Wie dieses beispielsweise aussehen könnte, werden wir im Folgenden darstellen.

Gegeben seien n Prozesse, welche um eine gewisse Anzahl von Ressourcen konkurrieren. Aufgabe ist es nun, eine in der Kardinalität größtmögliche Menge von Prozessen zu bestimmen, so dass nicht mehr als ein Prozess auf eine Ressource zugreift. Ziel dabei ist es, möglichst viele Prozesse innerhalb eines vorgegebenen Zeitraumes bearbeiten zu können.

Dieses Problem kann als Graph $G = (V, E)$ modelliert werden, mit der Aufgabe eine unabhängige Menge maximaler Kardinalität zu bestimmen. Der Graph $G = (V, E)$ besitzt als Knotenmenge V die einzelnen Prozesse, wobei mit jedem Prozess genau ein Knoten $v \in V$ assoziiert ist. Zwei Knoten sind durch Kanten miteinander verbunden, falls die dazugehörigen Prozesse auf gleiche Ressourcen zugreifen, das heißt zwischen diesen beiden Prozessen besteht eine Abhängigkeit.

Findet man nun in diesem modellierten Graphen eine unabhängige Menge maximaler Kardinalität, dann erhält man also eine (maximale) Menge von Prozessen, welche gleichzeitig ausgeführt werden können, ohne dass Komplikationen, etwa durch gleichzeitiges Zugreifen auf die gleiche Ressource, zum Beispiel RAM oder Festplatte, auftreten können.

Die Gliederung dieser Diplomarbeit geschieht in folgender Weise. Zunächst sollen in Kapitel 2 grundlegende Definition behandelt werden, wobei hier der Schwerpunkt auf Definitionen aus der Graphentheorie liegt. Anschließend werden wir den Algorithmus Ramsey kennenlernen und einen kurzen Einblick in die Ramsey-Theorie nehmen. Um die Analyse dieses Algorithmus zu erleichtern, werden wir uns in einem Abschnitt mit Bäumen beschäftigen. Darauf aufbauend schließt das dritte Kapitel mit dem Algorithmus Clique-Removal und einer Analyse seiner Approximationsgüte ab. Das vierte Kapitel enthält den Algorithmus von Feige, welcher die Approximationsgüte des Problems Independent Set auf $O(n(\log\log n)^2/(\log n)^3)$ verbessert. Anschließend werden wir in Kapitel 5 eine andere Klasse von Algorithmen kennenlernen, die auf dem Prinzip der semidefiniten Programmierung beruhen. Dazu gehört unter anderem auch der Einsatz der Lovász ϑ-Funktion [35]. Mit der Online Situation des Problems Independent Set beschäftigt sich das sechste Kapitel. Insbesondere werden wir hier das Multisolution Modell betrachten [20]. Praktische Untersuchungen sind Gegenstand des siebten und letzten Kapitels. Dabei werden ausgewählte Algorithmen darunter beispielsweise der Algorithmus Ramsey oder der Algorithmus Clique-Removal im Computeralgebrasystem Matlab implementiert und anhand einiger Graphen unterschiedlicher Knoten- und Kantenanzahl getestet. Mit einer Analyse der Testergebnisse schließt diese Diplomarbeit ab.

2 GRUNDLEGENDE DEFINITIONEN

Im Folgenden sollen grundlegende Definitionen aus der Graphentheorie betrachtet werden.

Definition 2.1 (Graph) *Ein Graph $G = (V, E)$ besteht aus einer endlichen Menge V von Knoten und einer Menge $E \subseteq \{\{v, w\} : v, w \in V \text{ und } v \neq w\}$ von (ungerichteten) Kanten.*

Graphen sind demnach durch Kanten miteinander verbundene Knoten. Falls zwei Knoten $v \in V$ und $w \in V$ durch eine Kante $\{v, w\} \in E$ verbunden sind, nennen wir die Knoten v und w benachbart beziehungsweise adjazent.

Definition 2.2 (adjazent, Nachbarschaft $N(v)$) *Es sei $G = (V, E)$ ein Graph. Ein Knoten $v \in V$ ist adjazent zu einem Knoten $w \in V$, falls $\{v, w\} \in E$ gilt. Die Nachbarschaft $N(v)$ des Knotens v ist die Menge aller adjazenten Knoten von v.*

Wir suchen nach Knotenmengen, deren Knoten paarweise nicht adjazent beziehungsweise benachbart sind. Solche Knotenmengen heißen unabhängige Mengen.

Definition 2.3 (unabhängige Menge) *Es sei $G = (V, E)$ ein Graph. Eine Teilmenge I der Knotenmenge V, $I \subseteq V$, heißt unabhängige Menge, falls für je zwei verschiedene Knoten $v \in I$ und $w \in I$ gilt $\{v, w\} \notin E$.*

Im Allgemeinen sind wir nicht an beliebigen unabhängigen Mengen interessiert, da ja jeder einzelne Knoten nach Definition 2.3 bereits eine solche ist, sondern an unabhängigen Mengen maximaler Kardinalität. In diesem Zusammenhang definieren wir die Unabhängigkeitszahl $\alpha(G)$ eines Graphen $G = (V, E)$.

Definition 2.4 (Unabhängigkeitszahl $\alpha(G)$) *Es sei $G = (V, E)$ ein Graph und I_{max} eine bezüglich Kardinalität maximale unabhängige Menge. Dann ist die Unabhängigkeitszahl $\alpha(G)$ des Graphen G definiert als $\alpha(G) = |I_{max}|$.*

Insbesondere sind wir an unabhängigen Mengen $I \subseteq V$ interessiert, für die $|I| = \alpha(G)$ gilt.

Definition 2.5 (Problem Independent Set) *Das Problem des Auffindens einer bezüglich Kardinalität maximalen unabhängigen Menge in einem Graphen $G = (V, E)$ wird im Folgenden mit* Independent Set *bezeichnet.*

Das Problem INDEPENDENT SET ist eines der zentralen Probleme der Informatik und von wesentlicher Bedeutung in der Kombinatorik sowie in der Graphentheorie. Aufgrund komplexitätstheoretischer Betrachtungen [12] vermutet man, dass dieses Problem nicht effizient lösbar ist, das heißt, es gibt vermutlich keinen Algorithmus zur Lösung des Problems INDEPENDENT SET, dessen Laufzeit polynomiell[1] beschränkt ist. Zur Erfassung derartiger Probleme definiert man daher die Komplexitätsklasse $\mathcal{NP}$ [44], die die Klasse der mit einer nichtdeterministischen Turingmaschine[2] in Polynomialzeit lösbaren Probleme ist.

Definition 2.6 (Komplexitätsklasse $\mathcal{NP}$) *Die Komplexitätsklasse $\mathcal{NP}$ ist die Klasse der Entscheidungsprobleme Π, für die es eine nichtdeterministische Turingmaschine gibt, deren worst-case Rechenzeit polynomiell beschränkt ist.*

Bei INDEPENDENT SET handelt es sich um ein Optimierungsproblem für das keine „größenordnungsmäßig schwereren[3]“ Entscheidungsprobleme in $\mathcal{NP}$ existieren. Derartige Probleme bezeichnen wir im Folgenden als $\mathcal{NP}$-hart (vgl. Anhang), das heißt INDEPENDENT SET ist $\mathcal{NP}$-hart [12] (vgl. auch Satz A.4). Für sämtliche $\mathcal{NP}$-harten Probleme sind bisher keine effizienten Algorithmen zur Lösung bekannt. Man begnügt sich in diesem Fall mit Lösungen, die möglichst in der Nähe des Optimums liegen, das bedeutet man versucht gute Lösungen zu approximieren. Für das Problem INDEPENDENT SET bedeutet dies, dass wir uns auch mit unabhängigen Mengen zufrieden geben, welche nicht in jedem Fall eine unabhängige Menge maximaler Kardinalität bilden. Nun kann es aber passieren, dass ein Algorithmus eine Lösung liefert, die sehr schlecht ist, weil sie „sehr weit“ vom Optimum entfernt liegt. Um die Qualität eines Approximationsalgorithmus deshalb beurteilen zu können, verwendet man den Begriff der (worst-case) Güte.

Definition 2.7 (Güte, worst-case Güte) *Für eine Eingabe I sei $w(A(I))$ der Wert der Lösung $A(I)$ eines Algorithmus A für ein Optimierungsproblem Π und $OPT(I)$ der Wert einer optimalen Lösung. Dann ist die Güte der Lösung $A(I)$*

$$R_A(I) = \max\left\{\frac{w(A(I))}{OPT(I)}, \frac{OPT(I)}{w(A(I))}\right\}.$$

Die worst-case Güte ist definiert durch

$$R_A := \inf\{r \geq 1 \mid R_A(I) \leq r \quad \text{für alle } I \in D_\Pi\},$$

wobei D_Π die Menge der zulässigen Eingaben für das Optimierungsproblem Π ist.

[1] Das heißt die Laufzeit eines Algorithmus für das Problem INDEPENDENT SET ist durch ein Polynom $p(n) = \sum_{i=1}^{k} a_i \cdot n^i$ für einen Graphen $G = (V, E)$ in der Anzahl Knoten $|V|$ und Kanten $|E|$ beschränkt.

[2] Mit einer nichtdeterministischen Turingmaschine verbindet man die Vorstellung eines Orakels, das die Lösung eines Problems errät. Die Laufzeit ist dann durch die Verifizierzeit der erratenen Lösung definiert.

[3] Eine formale Definition dieses Sachverhalts erfolgt im Anhang.

Wir werden bei unseren Betrachtungen stets die worst-case Güte verwenden, um eine Aussage darüber machen zu können, welche Qualität ein Algorithmus garantieren kann.

Bisher haben wir nur unabhängige Mengen in Graphen betrachtet. Dazu komplementäre Mengen sind die Cliquen.

Definition 2.8 (Clique) *Es sei $G = (V, E)$ ein Graph. Eine Menge $C \subseteq V$ heißt Clique falls für je zwei verschiedene Knoten $v \in C$ und $w \in C$ gilt $\{v, w\} \in E$.*

Eine Clique im Graphen $G = (V, E)$ ist also eine Teilmenge der Knotenmenge V, so dass zwischen je zwei Knoten eine Kante verläuft, während eine unabhängige Menge eine Teilmenge der Knotenmenge V ist, so dass zwischen je zwei Knoten keine Kante verläuft. Aufgrund dieses Zusammenhangs zwischen unabhängigen Mengen und Cliquen ist es sinnvoll Komplementgraphen zu definieren.

Definition 2.9 (Komplementgraph $\overline{G}$) *Es sei $G = (V, E)$ ein Graph. Der Komplementgraph $\overline{G} = (V, \overline{E})$ ist definiert durch $\overline{E} = \{\{v, w\} \mid \{v, w\} \notin E\}$.*

Damit ist eine unabhängige Menge im Graphen $G = (V, E)$ gerade eine Clique im Komplementgraphen $\overline{G}$. Analog zu unabhängigen Mengen sind wir auch hier an Cliquen maximaler Kardinalität interessiert, das heißt an Cliquen deren Kardinalität gerade der Cliquenzahl des Komplementgraphen $\overline{G}$, also $\omega(\overline{G})$, entspricht.

Definition 2.10 (Cliquenzahl $\omega(G)$) *Es sei $G = (V, E)$ ein Graph und C_{max} eine bezüglich Kardinalität maximale Clique. Dann ist die Cliquenzahl $\omega(G)$ des Graphen G definiert als $\omega(G) = |C_{max}|$.*

Einen wesentlichen Zusammenhang zwischen der Unabhängigkeitszahl $\alpha(G)$ eines Graphen $G = (V, E)$ und der Cliquenzahl $\omega(\overline{G})$ des entsprechenden Komplementgraphen $\overline{G}$ macht der folgende Satz.

Satz 2.11 *Es sei $G = (V, E)$ ein Graph und $\overline{G}$ der dazugehörige Komplementgraph. Dann gilt*

$$\alpha(G) = \omega(\overline{G}) \ .$$

Beweis: Es sei I eine maximale unabhängige Menge im Graphen G. Wir zeigen, dass die Clique C mit $C = I$ dann eine maximale Clique im Graphen $\overline{G}$ ist. Nehmen wir an, dass es eine Clique C^* mit $|C^*| > |C|$ gibt, dann gibt es nach Definition zwischen je zwei verschiedenen Knoten aus C^* in $\overline{G}$ eine Kante. Dies bedeutet jedoch, dass im Graphen G innerhalb von C^* keine Kanten verlaufen. Wir haben also eine unabhängige Menge I^* mit $I^* = C^*$ konstruiert, für die $|I^*| > |I|$ gilt, im Widerspruch zur Maximalität von I. Umgekehrt erfolgt der Beweis analog. □

Da das Komplement $\overline{\overline{G}} = (V, \overline{\overline{E}})$ des Komplementgraphen $\overline{G} = (V, \overline{E})$ wieder der ursprüngliche Graph $G = (V, E)$ ist, gilt offensichtlich auch

$$\alpha(\overline{G}) = \omega(G) \ . \tag{2.1}$$

Bevor wir uns einen weiteren Zusammenhang zwischen Cliquen und unabhängigen Mengen anschauen, werden wir das Problem CLIQUE definieren.

Definition 2.12 (Problem Clique) *Das Problem, eine Clique maximaler Kardinalität in einem Graphen $G = (V, E)$ aufzufinden, wird im Folgenden mit* CLIQUE *bezeichnet.*

Aufgrund der gerade beschriebenen Dualität der Probleme INDEPENDENT SET und CLIQUE können wir auch eine Aussage über einen Zusammenhang bezüglich deren Approximierbarkeit machen.

Satz 2.13 *Es gibt einen in der Laufzeit polynomiell beschränkten Approximationsalgorithmus A für das Problem* INDEPENDENT SET *mit Güte $R_A = r$ genau dann, wenn es einen in der Laufzeit polynomiell beschränkten Approximationsalgorithmus B für das Problem* CLIQUE *mit Güte $R_B = r$ gibt.*

Beweis: Angenommen es gibt einen Polynomialzeit-Approximationsalgorithmus A für das Problem INDEPENDENT SET mit Güte $R_A = r$. Wir konstruieren nun einen Algorithmus B, der das Problem CLIQUE mit Güte $R_B = r$ approximiert und in der Laufzeit ebenfalls polynomiell beschränkt ist. Der Algorithmus B bildet aus dem Eingabegraphen $G = (V, E)$ für das Problem CLIQUE zunächst den Komplementgraphen $\overline{G} = (V, \overline{E})$. Dieses erfolgt in Polynomialzeit. Anschließend wird auf $\overline{G}$ der Approximationsalgorithmus A angewendet, dessen Laufzeit polynomiell beschränkt ist. Die von A gefundene Lösung gibt der Algorithmus B aus. Die Gesamtlaufzeit des Approximationsalgorithmus B ist offenbar auch polynomiell beschränkt. Für den Wert $w(A(\overline{G}))$ gilt wegen der Approximationsgüte des Algorithmus A

$$\frac{\alpha(\overline{G})}{w(A(\overline{G}))} \leq r \ .$$

Nach obigen Überlegungen ist eine Knotenmenge $C \subseteq V$ im Graphen G genau dann eine Clique, wenn die Menge C im Graphen $\overline{G}$ eine unabhängige Menge ist. Damit ist die Ausgabe des Algorithmus B korrekt, das heißt B liefert eine Clique. Wegen (2.1) gilt $\alpha(\overline{G}) = \omega(G)$ und somit erhalten wir

$$\frac{\omega(G)}{w(B(G))} \leq r \ .$$

Umgekehrt erfolgt der Beweis analog. □

3 DER RAMSEY ALGORITHMUS

3.1 Die Idee des Algorithmus

Ein einfacher Algorithmus zur Approximation von unabhängigen Mengen ist der in [22, 23] betrachtete Min-Greedy Algorithmus, welcher in [4] ebenfalls analysiert wurde. Dieser Algorithmus konstruiert eine unabhängige Menge I eines Graphen $G = (V, E)$, indem er ausgehend von der leeren Menge $I := \emptyset$ einen Knoten $v \in V$ minimalen Grades wählt, diesen zur Menge I hinzufügt und dann den Knoten v, seine Nachbarn $N(v)$ sowie die zu $N(v) \cup \{v\}$ inzidenten Kanten löscht. Dieses Verfahren wird auf dem reduzierten Graphen iteriert, bis kein Knoten mehr vorhanden ist. Um eine bezüglich Inklusion maximale unabhängige Menge zu bestimmen, müssen wir jedoch nicht unbedingt einen Knoten $v \in V$ minimalen Grades wählen. Es reicht in diesem Fall aus, wenn wir einen beliebigen Knoten $v \in V$ in jeder Iteration wählen, obwohl dadurch nur eine Güte von $O(n)$ erreicht wird. Diese Vorgehensweise führt uns zum Algorithmus Greedy-Independent-Set.

Algorithmus 3.1 Greedy-Independent-Set($G = (V, E)$)

```
1: if V = ∅ then
2:     return (∅)
3: choose v ∈ V
4: V ← V \ (N(v) ∪ {v})
5: G ← G[V]
6: I ← Greedy-Independent-Set(G)
7: return ({v} ∪ I)
```

Analog zum Algorithmus Greedy-Independent-Set können wir auch Cliquen in einem Graphen $G = (V, E)$ suchen. Dazu wählen wir einen Knoten $v \in V$ aus und fügen diesen zur Clique C hinzu, die anfangs auf die leere Menge $C := \emptyset$ gesetzt wird. Anschließend suchen wir in der Nachbarschaft $N(v)$ des Knotens $v \in V$ weiter nach Cliquen. Offensichtlich erhalten wir eine bezüglich Inklusion maximale Clique C. Das Verfahren ist im Algorithmus Greedy-Clique dargestellt.

Diese beiden Algorithmen konstruieren sehr schnell eine bezüglich Inklusion maximale unabhängige Menge beziehungsweise eine bezüglich Inklusion maximale Clique. Ein wesentlicher Nachteil dieser Greedy Verfahren ist, dass abhängig vom gewählten Knoten (in Zeile 3 der beiden Algorithmen) jeweils unabhängige Mengen beziehungsweise Cliquen berechnet werden,

Algorithmus 3.2 Greedy-Clique($G = (V, E)$)

```
1: if V = ∅ then
2:     return (∅)
3: choose v ∈ V
4: V ← N(v)
5: G ← G[V]
6: C ← Greedy-Clique(G)
7: return ({v} ∪ C)
```

deren Kardinalität erheblich variieren kann, weshalb der Greedy Algorithmus im Allgemeinen keine guten Lösungen garantiert. Beim Algorithmus Greedy-Independent-Set kann nämlich der Fall eintreten, dass ein Knoten $v \in V$ gewählt wird, dessen Nachbarn eine große unabhängige Menge bilden. Diese unabhängige Menge geht allerdings verloren, da in der nächsten Phase nur die Knoten außer v betrachtet werden, welche nicht in der Nachbarschaft $N(v)$ liegen. Wir sollten also parallel in der Nachbarschaft $N(v)$ des Knotens v suchen. Analog gilt dieses auch für den Algorithmus Greedy-Clique. Das Ergebnis dieser Überlegungen ist der von Halldórsson und Boppana entwickelte Algorithmus Ramsey [7].

Algorithmus 3.3 Ramsey($G = (V, E)$)

```
1: if V = ∅ then
2:     return (∅, ∅)
3: choose v ∈ V
4: V_1 ← N(v)
5: G_1 ← G[V_1]
6: V_2 ← V \ (N(v) ∪ {v})
7: G_2 ← G[V_2]
8: (C_1, I_1) ←Ramsey(G_1)
9: (C_2, I_2) ←Ramsey(G_2)
10: if |C_1 ∪ {v}| > |C_2| then
11:     C ← C_1 ∪ {v}
12: else
13:     C ← C_2
14: if |I_2 ∪ {v}| > |I_1| then
15:     I ← I_2 ∪ {v}
16: else
17:     I ← I_1
18: return (C, I)
```

Bevor wir uns dem Verständnis des Algorithmus Ramsey zuwenden, werden wir einen Einblick in die Ramsey-Theorie [15] geben. Anschließend werden wir uns noch mit Bäumen beschäftigen, da sich damit die Arbeitsweise des Algorithmus Ramsey sehr gut veranschaulichen lässt.

3.2 Ein kurzer Einblick in die Ramsey-Theorie

Der Algorithmus RAMSEY ist, wie der Name schon vermuten lässt, eng verwandt mit einem klassischen Problem aus der Graphentheorie. Die im Nachfolgenden betrachtete Ramsey-Zahl $R(k,l)$ wurde nach dem englischen Mathematiker Frank P. Ramsey[1] benannt, der als Erster zeigen konnte, dass diese Zahl wohldefiniert ist.

Definition 3.1 (Ramsey-Zahl $R(k,l)$) *Es seien $k \geq 2$ und $l \geq 2$ natürliche Zahlen. Dann gibt es eine kleinste Zahl $R(k,l)$, genannt Ramsey-Zahl, so dass in jedem Graphen $G = (V,E)$ auf $n \geq R(k,l)$ Knoten eine Clique C der Kardinalität $|C| = k$ oder eine unabhängige Menge I der Kardinalität $|I| = l$ enthalten ist.*

Informal könnte man das Problem folgendermaßen beschreiben: Treffen sich $n \geq R(k,l)$ Personen, so gibt es immer k Personen, die sich alle gegenseitig kennen, oder es gibt l Personen, die sich paarweise nicht kennen.

Satz 3.2 *Für die Ramsey-Zahl $R(k,l)$ gilt die Rekursion*

$$R(k,l) \leq R(k,l-1) + R(k-1,l)\ . \tag{3.1}$$

Beweis: Es sei $G = (V,E)$ ein beliebiger Graph mit

$$|V| = R(k-1,l) + R(k,l-1)$$

Knoten. Wir müssen nun zeigen, dass G entweder eine Clique C der Kardinalität $|C| = k$ oder eine unabhängige Menge I der Kardinalität $|I| = l$ enthält. Zunächst wählen wir einen beliebigen Knoten $v \in V$ und setzen

$$\begin{aligned} I_1(v) &:= \{w \mid w \in V : \{v,w\} \in E\} \\ I_2(v) &:= \{w \mid w \in V : \{v,w\} \notin E\}\ . \end{aligned}$$

Für die Kardinalitäten der Mengen $I_1(v)$ und $I_2(v)$ gilt

$$|I_1(v)| + |I_2(v)| = |V| - 1\ .$$

Nach dem Schubfachprinzip gilt demnach entweder

$$|I_1(v)| \geq R(k-1,l)$$

oder

$$|I_2(v)| \geq R(k,l-1)\ .$$

Fall 1: Es sei $|I_1(v)| \geq R(k-1,l)$. Aufgrund der Definition von R ist mindestens eine der folgenden zwei Bedingungen erfüllt:

[1] Ramsey wurde am 22. Februar 1903 geboren und verstarb am 19. Januar 1930. Trotz seines frühen Todes veröffentlichte er bedeutende wissenschaftliche Arbeiten in den Bereichen Logik sowie Kombinatorik und hierbei insbesondere in der Graphentheorie.

1. Es existiert eine unabhängige Menge $I \subseteq I_1(v)$ der Kardinalität $|I| = l$ und wir sind fertig.

2. Es existiert eine Clique $C \subseteq I_2(v)$ der Kardinalität $|C| = k - 1$.

Falls Letzteres gilt, setzen wir $C^* = C \cup \{v\}$, so dass $|C^*| = k$ gilt und C^* eine Clique ist.

Fall 2: Es sei $|I_2(v)| \geq R(k, l-1)$. Wie in Fall 1 ist wieder mindestens eine der folgenden zwei Bedingungen erfüllt:

1. Es existiert eine unabhängige Menge $I \subseteq I_1(v)$ der Kardinalität $|I| = l - 1$.

2. Es existiert eine Clique $C \subseteq I_2(v)$ der Kardinalität $|C| = k$ und wir sind fertig.

Falls Ersteres gilt, setzen wir $I^* = I \cup \{v\}$, um eine unabhängige Menge der Kardinalität $|I^*| = l$ zu erhalten. □

Damit lässt sich auch die folgende Abschätzung herleiten.

Satz 3.3 *Für die Ramsey-Zahl $R(k, l)$ gilt die obere Schranke*

$$R(k, l) \leq \binom{k+l-2}{k-1}.$$

Beweis: Wir beweisen diesen Satz mittels Induktion über k und l. Den Induktionsanfang bildet

$$R(k, 2) = k = \binom{k+2-2}{k-1}$$

sowie

$$R(2, l) = l = \binom{2+l-2}{2-1}.$$

Der Induktionsschritt erfolgt durch Einsetzen von

$$R(k, l-1) \leq \binom{k+l-1-2}{k-1}$$

und

$$R(k-1, l) \leq \binom{k+l-1-2}{k-1-1}$$

in die Rekursionsformel (3.1) aus Satz 3.2, welche mit Hilfe der Definition der Binomialkoeffizienten [1, Kapitel 1] vereinfacht wird:

$$\begin{aligned} R(k, l) &\leq R(k, l-1) + R(k-1, l) \\ &\leq \binom{k+l-3}{k-1} + \binom{k+l-3}{k-2} \\ &\leq \binom{k+l-2}{k-1}. \end{aligned}$$

□

3.3 Bäume

Definition 3.4 (Baum, Wurzel) *Ein Graph $T = (V, E)$ heißt Baum, falls er zusammenhängend ist und keine Kreise enthält. Die Wurzel ist ein speziell ausgezeichneter Knoten w der Knotenmenge V.*

Bäume werden meistens hierarchisch, abhängig von der Entfernung zur Wurzel w, in Ebenen dargestellt. Die Entfernung zur Wurzel ist dabei ein eindeutig definierter Wert, da je zwei Knoten in einem Baum T durch einen eindeutigen Weg verbunden sind [1, Kapitel 6].

Beispiel 3.5 Der Graph $G = (V, E)$ mit

$$\begin{aligned} V &:= \{1, 2, 3, 4, 5, 6, 7, 8\} \\ E &:= \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 5\}, \{3, 6\}, \{3, 7\}\{7, 8\}\} \end{aligned}$$

ist ein Baum. Falls die Wurzel w der Knoten 1 ist, kann der Baum folgendermaßen dargestellt werden.

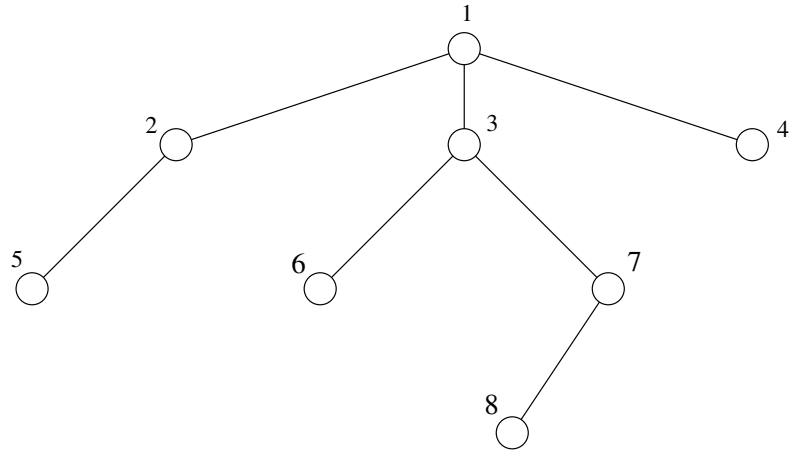

Abbildung 3.1: Visualisierung eines Baumes

Die Wurzel 1 befindet sich in Ebene 0, die Knoten $\{2, 3, 4\}$ mit Abstand 1 von der Wurzel werden Ebene 1 zugeordnet. Die Knoten $\{5, 6, 7\}$ sind demnach in Ebene 2 und der Knoten 8 in Ebene 3.

Definition 3.6 (Vater, Sohn, Blatt, innere Knoten) *Es sei $T = (V, E)$ ein Baum mit Wurzel w. Ein Knoten x heißt Vater des Knotens y, falls $\{x, y\} \in E$ und die Entfernung von x zur Wurzel w um genau Eins kleiner ist als die Entfernung von y zur Wurzel w. Der Knoten y wird in diesem Fall als Sohn von x bezeichnet. Falls ein Knoten v eines Baumes T keinen Sohn besitzt, dann bezeichnen wir v als Blatt, sonst heißt x innerer Knoten von T.*

Für das Beispiel 3.5 bedeutet diese Definition, dass beispielsweise der Knoten 3 der Vater der Knoten 6 und 7 ist. Der Knoten 8 ist hier Sohn des Knotens 7 und die Knoten $4, 5, 6$ und 8 sind Blätter.

Eine spezielle Art von Bäumen sind die binären Bäume.

Definition 3.7 (binärer Baum, linker (rechter) Sohn) *Es sei $T = (V, E)$ ein Baum mit Wurzel w. Dieser Baum T heißt binärer Baum, falls jeder innere Knoten x von T höchstens zwei Söhne besitzt. Wir unterscheiden in diesem Fall linken Sohn x_l und rechten Sohn x_r. Die Kante $\{x, x_l\}$ bezeichnen wir als Linkskante, die Kante $\{x, x_r\}$ als Rechtskante.*

Definition 3.8 (äußerer Knoten) *Es sei $T = (V, E)$ ein binärer Baum. Ein Knoten $x \notin V$ heißt äußerer Knoten, falls es einen Knoten $y \in V$ gibt, so dass der Baum T durch „anhängen“ von x an y ein binärer Baum bleibt.*

Beispiel 3.9 Der in Abbildung 3.2 dargestellte binäre Baum T auf der Knotenmenge $V = \{1, \ldots, 6\}$ hat 7 äußere Knoten, welche hier schwarz markiert sind. Im Allgemeinen werden die äußeren Knoten bei einer graphischen Veranschaulichung nicht dargestellt, da diese Knoten nicht Teil der Knotenmenge des Baumes T sind.

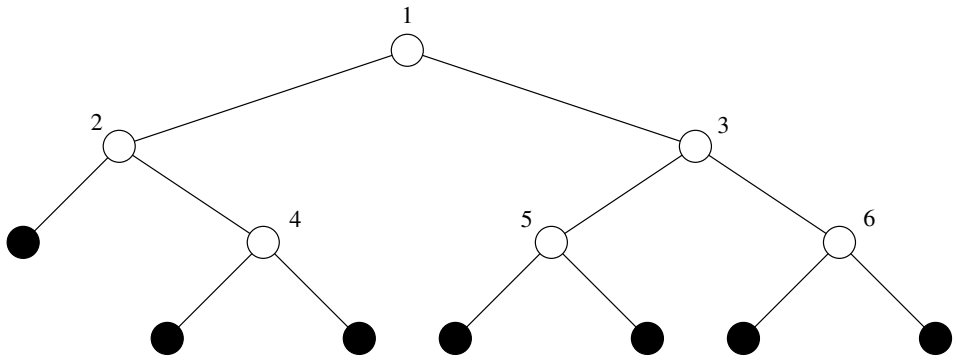

Abbildung 3.2: Äußere Knoten eines binären Baumes

Lemma 3.10 *Ein binärer Baum T mit n Knoten hat genau $n + 1$ äußere Knoten.*

Beweis: Wir beweisen dieses durch Induktion über die Anzahl n der Knoten. Für $n = 1$, das heißt, falls der Baum T nur aus der Wurzel besteht, gilt offensichtlich

$$\#\text{ äußere Knoten} = 2 = n + 1\ ,$$

da man an die Wurzel genau 2 verschiedene Knoten anhängen kann, so dass T ein binärer Baum bleibt. Im Induktionsschritt $n \to n + 1$ fügen wir nun ein Blatt so zum Baum T hinzu, dass weiterhin ein binärer Baum vorliegt, das heißt, wir können beispielsweise ein Blatt an ein bereits vorhandenes Blatt anhängen. Dieses ist leicht möglich, da jeder endliche Baum

mindestens ein Blatt hat. Durch das Hinzufügen verlieren wir einen äußeren Knoten, nämlich gerade an der Position, an der wir das Blatt angehängt haben. Auf der anderen Seite erhöht sich die Anzahl äußerer Knoten um zwei, nämlich um die zwei äußeren Knoten, die wir an das hinzugefügte Blatt anhängen können. Damit gilt

$$\#\text{ äußere Knoten} = (n+1) - 1 + 2 = n + 2\ . \qquad \square$$

3.4 Verständnis des Algorithmus

Die Berechnungen des Algorithmus RAMSEY für einen Graphen $G = (V, E)$ kann man sich am besten durch einen binären Baum T veranschaulichen, wobei jeder innere Knoten von T als Schlüsselwert gerade den in *Zeile 3* des Algorithmus RAMSEY (vgl. Algorithmus 3.3) gewählten Knoten $v \in V$ enthält. Die Aufrufe mit leerem Graphen entsprechen den äußeren Knoten (vgl. Definition 3.8) dieses Rekursionsbaumes. Der linke Sohn eines inneren Knotens soll dabei dem Rekursionsaufruf „Ramsey(G_1)“ (*Zeile 8*) und der rechte Sohn dem Rekursionsaufruf „Ramsey(G_2)“ (*Zeile 9*) entsprechen. Die Abbildung 3.3 stellt diesen Sachverhalt dar. An dieser Stelle sei bemerkt, dass der Graph $G_1 = (V_1, E_1)$ gerade der auf die Nachbarschaft $N(v)$ des Knotens v induzierte Untergraph von G ist. Entsprechend enthält der Graph $G_2 = (V_2, E_2)$ gerade alle die Knoten außer v, welche keine Nachbarn des Knotens v sind.

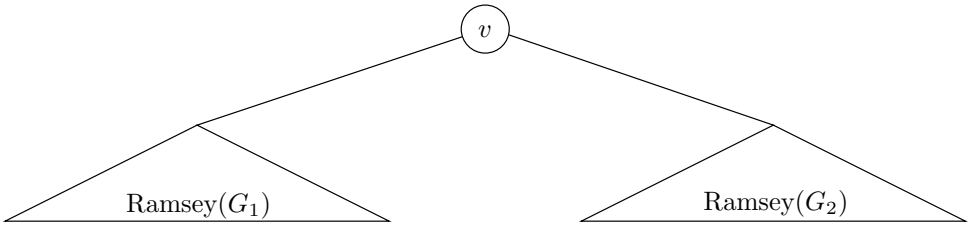

Abbildung 3.3: Rekursionsbaum des Algorithmus RAMSEY

Aufgrund dieser Eigenschaft gilt für jeden inneren Knoten $v \in V$ des binären Baumes, dass alle Knoten $x \in V$, die sich im linken Teilbaum unter v befinden, im Graphen $G = (V, E)$ mit dem Knoten v durch eine Kante $\{v, x\} \in E$ verbunden sind. Analog dazu gilt, dass alle Knoten $y \in V$, die sich im rechten Teilbaum unter v befinden, nicht zur Nachbarschaft $N(v)$ des Knotens v im Graphen $G = (V, E)$ gehören.

Mit dieser Sichtweise können wir die vom Algorithmus RAMSEY gefundene unabhängige Menge mit einem Pfad, welcher maximal viele rechte Kanten (vgl. Definition 3.7) enthält, vergleichen. Es sei $\{v_1, \ldots, v_n\}$ ein Pfad von der Wurzel v_1 zum Blatt v_n des Rekursionsbaumes, welcher maximal viele Rechtskanten enthält. Die unabhängige Menge, welche der Algorithmus RAMSEY findet, ist dann gerade

$$I = \{v_n\} \cup \{v_i \mid v_{i+1} \text{ ist rechter Sohn von } v_i \text{ für } i = 1, \ldots, n-1\}\ . \tag{3.2}$$

Die Kardinalität der unabhängigen Menge I entspricht also der Anzahl Rechtskanten auf dem Pfad $\{v_1, \ldots, v_n\}$ plus eins. Analog dazu sei $\{v_1, \ldots, v_m\}$ ein Pfad von der Wurzel v_1 zum Blatt v_m des Rekursionsbaumes, welcher maximal viele Linkskanten enthält. Die vom Algorithmus RAMSEY gefundene Clique ist dann

$$C = \{v_m\} \cup \{v_i \mid v_{i+1} \text{ ist linker Sohn von } v_i \text{ für } i = 1, \ldots, m-1\} . \tag{3.3}$$

3.5 Die Kardinalität der Mengen I und C

Wir wollen nun mit diesen Erkenntnissen die Größe der approximierten Clique C und der approximierten unabhängigen Menge I abschätzen. Angenommen der Algorithmus RAMSEY liefert eine Clique C der Kardinalität $|C| = k$ und eine unabhängige Menge der Kardinalität $|I| = l$ zurück. Aufgrund der Überlegungen im vorangegangenen Abschnitt entspricht der Rekursionsbaum des Algorithmus RAMSEY dabei gerade einem binären Baum, dessen maximale Anzahl an Linkskanten in einem Pfad $k-1$ ist und dessen maximale Anzahl an Rechtskanten in einem Pfad $l-1$ ist. Diese Betrachtung wollen wir nun verallgemeinern, um damit die Kardinalität der gefundenen Clique C und der gefundenen unabhängigen Menge I bestimmen zu können.

Definition 3.11 ($r(k,l)$) *Es sei $r(k,l)$ die kleinste natürliche Zahl n, so dass alle binären Bäume auf n Knoten eine der folgenden zwei Bedingungen erfüllen:*

1. *Es gibt einen Pfad, welcher mindestens $k-1$ Linkskanten enthält.*
2. *Es gibt einen Pfad, welcher mindestens $l-1$ Rechtskanten enthält.*

Um die Zahl $r(k,l)$ abzuschätzen, benötigen wir die folgende Definition:

Definition 3.12 (Baum $T_{k,l}$) *Es sei $T_{k,l}$ ein größter binärer Baum, der folgende Bedingungen erfüllt:*

1. *Jeder Pfad enthält höchstens $k-2$ Linkskanten.*
2. *Jeder Pfad enthält höchstens $l-2$ Rechtskanten.*

Der Baum $T_{k,l}$ erfüllt also keine der zwei Bedingungen aus Definition 3.11.

Die Definition 3.12 wollen wir uns am folgenden Beispiel veranschaulichen.

Beispiel 3.13 Der durch den Graphen $G = (V, E)$ mit

$$\begin{aligned} V &:= \{1,2,3,4,5\} \\ E &:= \{\{1,2\},\{1,3\},\{2,4\},\{3,5\}\} \end{aligned}$$

gegebene Baum mit der Wurzel 1 ist gerade der, bis auf Isomorphie eindeutige, Baum $T_{3,3}$.

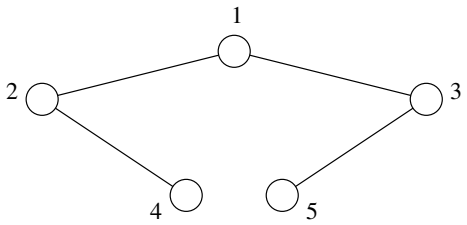

Abbildung 3.4: Graphische Darstellung des Baumes $T_{3,3}$

Wollen wir an diesen Baum zusätzlich noch einen Knoten anhängen, dann erhalten wir in jedem Fall offensichtlich einen Pfad mit 2 Linkskanten oder einen Pfad mit 2 Rechtskanten.

Es gilt offenbar

$$\#\text{ Knoten in } T_{k,l} = r(k,l) - 1\ . \tag{3.4}$$

Nach Lemma 3.10 gilt für die Anzahl äußerer Knoten in $T_{k,l}$ demzufolge

$$\#\text{ äußere Knoten in } T_{k,l} = r(k,l)\ . \tag{3.5}$$

Damit wollen wir nun die Zahl $r(k,l)$ abschätzen. Wir zeigen zunächst das folgende Lemma:

Lemma 3.14 *Es gilt*

$$r(k,l) = r(k-1,l) + r(k,l-1)\ . \tag{3.6}$$

Beweis: Sei $T_{k,l}$ wie oben ein größter Baum, welcher einen Pfad mit höchstens $k-2$ Linkskanten beziehungsweise höchstens $l-2$ Rechtskanten enthält und sei w die Wurzel dieses Baumes,

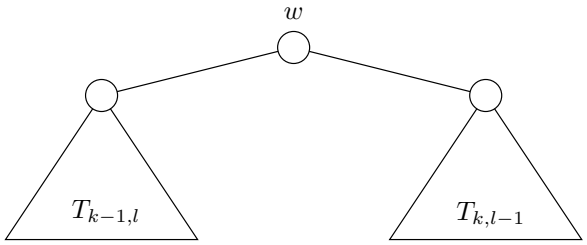

Abbildung 3.5: Zusammensetzung des Baumes $T_{k,l}$

dann ist der linke Teilbaum ein Baum $T_{k-1,l}$ und der rechte Teilbaum ein Baum $T_{k,l-1}$ (vgl. Abbildung 3.5) und folglich gilt:

$$\begin{aligned} r(k,l) &= \#\text{ Knoten in } T_{k,l} + 1 \\ &= \underbrace{\#\text{ Knoten in } T_{k-1,l} + \#\text{ Knoten in } T_{k,l-1} + 1}_{\#\text{ Knoten in } T_{k,l}} + 1 \\ &= \underbrace{r(k-1,l) - 1}_{\#\text{ Knoten in } T_{k-1,l}} + \underbrace{r(k,l-1) - 1}_{\#\text{ Knoten in } T_{k,l-1}} + 2 \\ &= r(k-1,l) + r(k,l-1)\ . \end{aligned}$$

□

Damit zeigen wir nun:

Satz 3.15 *Es gilt*

$$R(k,l) \leq r(k,l) = \binom{k+l-2}{l-1} .$$

Beweis: Wir beweisen diesen Satz mittels Induktion über k und l und nutzen gleichzeitig die Gleichung (3.5). Der Baum $T_{k,2}$ besteht aus $k-2$ Linkskanten und somit $k-1$ Knoten. Außer dem Blatt, welches 2 äußere Knoten besitzt, hat jeder Knoten einen äußeren Knoten. Damit gilt

$$\#\text{ äußere Knoten in } T_{k,2} = r(k,2) = k = \binom{k+2-2}{1} .$$

Der Baum $T_{2,l}$ besteht aus $l-2$ Rechtskanten und somit $l-1$ Knoten. Außer dem Blatt, welches 2 äußere Knoten besitzt, hat jeder Knoten wiederum einen äußeren Knoten. Damit gilt

$$\#\text{ äußere Knoten in } T_{2,l} = r(2,l) = l = \binom{2+l-2}{1} .$$

Der Rest folgt analog zum Beweis von Satz 3.3. Die Behauptung $R(k,l) \leq r(k,l)$ gilt, da wir in Lemma 3.14 die Identität $r(k,l) = r(k-1,l) + r(k,l-1)$ gezeigt haben. □

Diese Überlegungen nutzen wir, um den nächsten Satz, welcher eine Aussage über die Größe der unabhängigen Menge I und die Größe der Clique C macht, zu beweisen.

Satz 3.16 *Es sei $G = (V,E)$ ein Graph auf $n = |V|$ Knoten. Dann gibt der Algorithmus* Ramsey *bei Eingabe von G eine Clique C und eine unabhängige Menge I aus, so dass gilt*

$$\binom{|C|+|I|}{|C|} - 1 \overset{(i)}{\geq} r(|C|+1, |I|+1) - 1 \overset{(ii)}{\geq} n . \tag{3.7}$$

Beweis: Teil (i) der Formel (3.7) folgt sofort aus Satz 3.15. Zum Beweis von (ii) nehmen wir das Gegenteil an, das heißt es gelte

$$\begin{aligned} & r(|C|+1, |I|+1) - 1 \;<\; n \\ \iff \qquad & r(|C|+1, |I|+1) \;\leq\; n . \end{aligned}$$

Diese Aussage bedeutet, dass jeder Baum auf n Knoten einen Pfad mit (mindestens) $|C|$ Linkskanten oder einen Pfad mit (mindestens) $|I|$ Rechtskanten besitzt. In diesem Fall hätte der Algorithmus Ramsey jedoch eine Clique C^* der Kardinalität $|C^*| = |C|+1$ oder eine unabhängige Menge I^* der Kardinalität $|I^*| = |I|+1$ ausgeben müssen, Widerspruch. □

Aus diesem Satz können wir das folgende Korollar ableiten.

Korollar 3.17 *Es sei $G = (V,E)$ ein Graph auf $n = |V|$ Knoten. Dann gibt der Algorithmus* Ramsey *bei Eingabe von G eine Clique C und eine unabhängige Menge I aus, so dass gilt*

$$|I| \cdot |C| \geq \frac{1}{4} \cdot (\log n)^2 .$$

Beweis: Aufgrund von (3.7) gilt

$$n \leq \binom{|I| + |C|}{|C|},$$

wobei die rechte Seite für den Fall der Gleichheit $|I| = |C|$ minimal wird [37]. Damit gilt

$$n \leq \binom{2|C|}{|C|} \leq 2^{2|C|} \iff |C| \geq \frac{1}{2} \cdot \log n$$

und wir erhalten wegen $|I| = |C|$

$$|I| \cdot |C| \geq \frac{1}{4} \cdot (\log n)^2 .$$

□

3.6 Eine Approximationsgüte von $O(n/(\log n)^2)$

In Abschnitt 3.1 haben wir den Algorithmus Ramsey kennengelernt. Hier wollen wir nun zeigen, wie wir mit Hilfe dieses Algorithmus auf Graphen mit n Knoten eine Approximationsgüte von $O(n/(\log n)^2)$ erreichen.
Angenommen der gegebene Graph $G = (V, E)$ enthält nur Cliquen geringer Kardinalität. Da für die ausgegebene unabhängige Menge I und die ausgegebene Clique C wegen Korollar 3.17

$$|I| \cdot |C| \geq \frac{1}{4} \cdot (\log n)^2$$

gilt, muss die gefundene unabhängige Menge I in diesem Fall entsprechend groß sein.
Umgekehrt, das heißt, wenn der gegebene Graph $G = (V, E)$ Cliquen hoher Kardinalität enthält, können wir leider keine nützlichen Aussagen über die Qualität der unabhängigen Menge machen. Wenn wir jedoch diese Cliquen hoher Kardinalität aus dem Graphen G entfernen und erneut den Algorithmus Ramsey auf den Restgraphen $G' = (V', E')$ anwenden, können wir bessere Ergebnisse erwarten. Wir könnten also die folgenden zwei Schritte ausführen:

1. Entferne aus dem Graphen $G = (V, E)$ eine maximale Menge knotendisjunkter Cliquen der Kardinalität k. Der dadurch entstandene Graph sei $G' = (V', E')$.

2. Wende den Algorithmus Ramsey auf den Graphen G' an.

Wenn wir diesen Ansatz verfolgen, stoßen wir jedoch auf folgende Probleme:

(i) Das erste Problem ist, dass, falls wir als Eingabe beliebige Graphen $G = (V, E)$ zulassen, der Restgraph $G' = (V', E')$ leer ist, das heißt keine Knoten mehr enthält.

(ii) Das zweite Problem ist, dass alle bisher bekannten Algorithmen für das Auffinden einer Clique der Kardinalität $k > 3$ eine Laufzeit von $\Theta(n^k)$ Operationen benötigen. Dieses ist jedoch nicht für alle auftretenden Werte von k und n polynomiell beschränkt.

Diese beiden Probleme können jedoch einfach gelöst werden. Falls wir nämlich wissen, dass der gegebene Graph G eine hinreichend große unabhängige Menge enthält, dann können wir garantieren, dass ausreichend viele Knoten im Graphen G' verbleiben und der Restgraph folglich nicht leer ist. Dieses werden wir im folgenden Lemma zeigen.

Lemma 3.18 *Es sei $G = (V, E)$ ein Graph mit einer Unabhängigkeitszahl $\alpha(G)$ und k die kleinste natürliche Zahl, so dass für ein $\varepsilon > 0$ gilt*

$$\alpha(G) \geq \left(\frac{1}{k} + \varepsilon\right) \cdot n \,. \tag{3.8}$$

Weiter seien $C_1, \ldots, C_m$ knotendisjunkte Cliquen der Kardinalität $|C_i| = k$, $i = 1, \ldots, m$, in G. Dann gilt für die Anzahl Knoten im Graphen $G[V']$ mit $V' := V \setminus (C_1 \cup \ldots \cup C_m)$

$$|V'| \geq \frac{\varepsilon}{1 - 1/k} \cdot n \geq \varepsilon \cdot n \,. \tag{3.9}$$

Beweis: Um dieses Lemma zu beweisen, werden wir einen worst-case Fall konstruieren. Genauer gesagt wollen wir einen Graphen $G = (V, E)$ konstruieren, der hauptsächlich aus den Cliquen C_i, $i = 1, \ldots, m$ besteht, so dass nach dem Löschen dieser Cliquen möglichst wenig Knoten im Restgraphen $G' = (V', E')$ verbleiben.
Wir beginnen die Konstruktion mit m Cliquen der Kardinalität $|C_i| = k$, $i = 1, \ldots, m$, welche wir untereinander mit Kanten verbinden, so dass eine unabhängige Menge I der Kardinalität $|I| = m$ (von jeder Clique ein Knoten) erhalten bleibt. Diese sorgfältige Wahl der Kanten zwischen den Cliquen soll bezwecken, dass wir wenige zusätzliche Knoten hinzufügen müssen, um die Bedingung (3.8) zu erfüllen. Falls wir nur diese m Cliquen verwenden, gilt aber nur

$$\alpha(G) = |I| = m = \frac{n}{k} \,.$$

Um (3.8) zu erfüllen, reicht es jedoch, wenn wir einen einzelnen Knoten v zum Graphen G hinzufügen, welcher zusammen mit der Menge I eine unabhängige Menge $I^* = I \cup \{v\}$ bildet, da in diesem Fall

$$\begin{aligned}
\alpha(G) = |I^*| = m + 1 &\geq \left(\frac{1}{k} + \varepsilon\right) \cdot n \\
\iff \qquad m + 1 &\geq \frac{n}{k} + \varepsilon \cdot n \\
\iff \qquad k \cdot m + k &\geq \underbrace{m \cdot k + 1}_{=n} + \varepsilon \cdot k \cdot n \\
\iff \qquad k - 1 &\geq \varepsilon \cdot k \cdot n \\
\iff \qquad \frac{k-1}{k \cdot n} &\geq \varepsilon
\end{aligned} \tag{3.10}$$

gilt. Wegen $k \geq 2$ haben wir somit ein $\varepsilon > 0$ bestimmt.
Nach dem Entfernen der Cliquen C_i, $i = 1, \ldots, m$, enthält der Graph $G' = (V', E')$ offenbar nur noch den einzelnen Knoten v. Mit (3.10) folgt

$$|V'| = 1 \geq \frac{\varepsilon}{1 - 1/k} \cdot n \geq \varepsilon \cdot n$$

und somit ist auch (3.9) erfüllt. □

Das Problem (ii) lässt sich auch einfach lösen, da wir eigentlich gar keine Cliquen suchen müssen. Der Algorithmus RAMSEY berechnet nämlich beides, eine unabhängige Menge I und eine Clique C. Diese Clique C wird dann aus dem Graphen entfernt und der Algorithmus RAMSEY erneut auf den Restgraphen angewendet. Falls durch das Entfernen von C der Graph leer wird, dann bestand der Graph nur aus dieser Clique C. Die größte unabhängige Menge I würde in diesem Fall nur aus einem Knoten bestehen. Der Algorithmus RAMSEY hätte dann jedoch einen einzelnen Knoten ausgegeben. Im Algorithmus CLIQUE-REMOVAL ist diese gerade beschriebene Vorgehensweise dargestellt.

Algorithmus 3.4 Clique-Removal($G = (V, E)$)

```
1:  i ← 1
2:  (C_i, I_i) ←Ramsey(G)
3:  I ← I_i
4:  while V ≠ ∅ do
5:      V ← V \ C_i
6:      G ← G[V]
7:      i ← i + 1
8:      (C_i, I_i) ←Ramsey(G)
9:      if |I_i| > |I| then
10:         I ← I_i
11: return (I, {C_1, C_2, ..., C_i})
```

Der Algorithmus CLIQUE-REMOVAL ruft den Algorithmus RAMSEY iterativ auf und entfernt die gefundene Clique, bis der Graph $G = (V, E)$ keinen Knoten mehr enthält. Das Ergebnis des Algorithmus ist eine größte, vom Algorithmus RAMSEY gefundene unabhängige Menge I sowie eine Zerlegung der Knotenmenge V in disjunkte Cliquen. Diese Partition der Knotenmenge ist eine Approximation des Problems CLIQUE COVER (vgl. Definition A.6). Umgekehrt, wenn wir als Eingabe für den Algorithmus CLIQUE-REMOVAL den Komplementgraphen $\overline{G} = (V, \overline{E})$ des Graphen $G = (V, E)$ verwenden, erhalten wir eine Approximation für das Problem CLIQUE und für das Problem COLORING (vgl. Definition A.5). Wir müssen jedoch nicht unbedingt den Komplementgraphen $\overline{G}$ verwenden, um diese beiden Probleme zu approximieren. Stattdessen genügt es, den Algorithmus CLIQUE-REMOVAL so zum Algorithmus INDEPENDENT-SET-REMOVAL zu modifizieren, dass wir in jeder Iteration die gefundene unabhängige Menge I aus dem Graphen entfernen. Die Knoten dieser unabhängigen Mengen können offenbar jeweils mit derselben Farbe gefärbt werden, so dass die Anzahl der Iterationen eine Approximation des Problems COLORING ist.

Aufgrund der Dualität der Probleme INDEPENDENT SET und CLIQUE besitzen die beiden Algorithmen CLIQUE-REMOVAL und INDEPENDENT-SET-REMOVAL die gleiche Approximationsgüte. Dies werden wir bei der Analyse der Güte dieser Algorithmen ausnutzen. Zunächst werden wir das folgende Lemma beweisen.

Algorithmus 3.5 Independent-Set-Removal($G = (V, E)$)

```
1:  i ← 1
2:  (C_i, I_i) ←Ramsey(G)
3:  C ← C_i
4:  while V ≠ ∅ do
5:      V ← V \ I_i
6:      G ← G[V]
7:      i ← i + 1
8:      (C_i, I_i) ←Ramsey(G)
9:      if |C_i| > |C| then
10:         C ← C_i
11: return (C, {I_1, I_2, ..., I_i})
```

Lemma 3.19 *Es sei $\mathcal{A}$ ein Algorithmus, welcher in einem Graphen $G = (V, E)$ mit $|V| = n$ eine unabhängige Menge I der Kardinalität $|I| = f(n)$ findet, wobei f eine positive ($f(n) > 0$ für alle n) monoton steigende Funktion ist. Dann erhält man durch iterative Anwendung von $\mathcal{A}$ auf G eine Färbung der Knotenmenge V mit*

$$\#\ \textit{Farben} \leq \sum_{i=1}^{n} \frac{1}{f(i)}\ . \tag{3.11}$$

Beweis: Wir beweisen diese Aussage mittels Induktion über die Anzahl n der Knoten V. Für $n = 1$ erhalten wir

$$\#\ \text{Farben} = 1 \leq \sum_{i=1}^{1} \frac{1}{f(i)} = 1\ ,$$

da jeder vernünftige Algorithmus $\mathcal{A}$ in einem Graphen, welcher aus einem Knoten besteht, eine unabhängige Menge I der Kardinalität $|I| = 1$ findet und demzufolge $f(1) = 1$ gilt.

Es sei nun $n > 1$. Der Algorithmus $\mathcal{A}$ findet laut Voraussetzung eine unabhängige Menge I der Kardinalität $|I| = f(n)$. Diese $f(n)$ vielen Knoten können mit einer Farbe gefärbt werden, da nach Definition einer unabhängigen Menge je zwei Knoten der Menge I nicht benachbart sind. Durch das Entfernen der Menge I aus dem Graphen G erhalten wir einen neuen Graphen $G' = (V', E')$ mit $|V'| = n' = n - f(n)$ vielen Knoten.

Wenn wir nun

$$\sum_{i=n'+1}^{n} \frac{1}{f(i)} = \sum_{i=n-f(n)+1}^{n} \frac{1}{f(i)} \geq 1 \tag{3.12}$$

zeigen können, sind wir fertig, da wir in diesem Fall

$$1 + \sum_{i=1}^{n'} \frac{1}{f(i)} \leq \sum_{i=1}^{n} \frac{1}{f(i)}$$

erhalten und damit die Anzahl Farben wie in (3.11) beschränkt ist.

Es gilt

$$\sum_{i=n-f(n)+1}^{n} \frac{1}{f(i)} = \frac{1}{f(n-f(n)+1)} + \frac{1}{f(n-f(n)+2)} + \ldots + \frac{1}{f(n)} \; . \tag{3.13}$$

Die rechte Seite in Formel (3.13) enthält offenbar $f(n)$ viele Summanden. Da $f(n)$ eine monoton steigende Funktion ist, gilt

$$\frac{1}{f(n-f(n)+1)} \geq \frac{1}{f(n-f(n)+2)} \geq \ldots \geq \frac{1}{f(n)}$$

und demzufolge auch

$$\sum_{i=n'+1}^{n} \frac{1}{f(i)} \geq f(n) \cdot \frac{1}{f(n)} = 1 \; .$$

□

Wir haben nun die Grundlagen geschaffen, um zu beweisen, dass der Algorithmus Clique-Removal eine unabhängige Menge I und der Algorithmus Independent-Set-Removal eine Clique C approximiert, wobei jeweils eine Approximationsgüte von $O(n/(\log n)^2)$ erreicht wird.

Satz 3.20 *Der Algorithmus* Independent-Set-Removal *approximiert für einen Graphen* $G = (V, E)$ *eine Clique* C *mit Güte* $O(n/(\log n)^2)$.

Beweis: Es seien C und $I_1, \ldots, I_{COLORS}$ die vom Algorithmus Independent-Set-Removal gefundene Approximation für das Problem Clique und das Problem Coloring, wobei die approximierte Anzahl Farben genau $COLORS$ ist. Aufgrund von Korollar 3.17 gilt in jeder Iteration i

$$|C_i| \cdot |I_i| \geq \left(\frac{1}{2} \cdot \log |n_i| \right)^2 \; ,$$

wobei mit n_i die Anzahl Knoten des Restgraphen in Iteration i gemeint ist. Die beiden Mengen C_i und I_i sind dementsprechend die in Iteration i gefundene Clique beziehungsweise die gefundene unabhängige Menge. Für $|I_i| = f(n_i)$ gilt folglich

$$f(n_i) \geq \frac{1}{4} \cdot \frac{(\log n_i)^2}{|C_i|} \geq \frac{1}{4} \cdot \frac{(\log n_i)^2}{|C|} \; , \tag{3.14}$$

da C die größte gefundene Clique ist.
Um Lemma 3.19 anwenden zu können, muss die eingesetzte Funktion $f(n)$ positiv ($f(n) > 0$ für alle n) und monoton steigend sein. Die Funktion $f(n_i)$ ist jedoch nur monoton steigend und nicht positiv, da für $n_i = 1$ der Funktionswert 0 angenommen wird. Die Bedeutung der Funktion $f(n)$ aus Lemma 3.19 ist folgende: Bei Eingabe eines Graphen $G = (V, E)$ mit $n = |V|$ Knoten liefert der Algorithmus $\mathcal{A}$ eine unabhängige Menge I der Größe $f(n)$. Offensichtlich gilt stets $f(n) \geq 1$ und insbesondere $f(1) = 1$, da die Knotenmenge eines Graphen, welcher nur aus einem Knoten besteht, selbst unabhängig ist. Wir könnten daher

die untere Schranke in (3.14) für $n_i = 1$ auf $f(1) = 1$ erhöhen. Zur einfacheren Rechnung [37] setzen wir allerdings nur $f(1) := 1/(4|C|) \leq 1$ und erhalten

$$\begin{aligned}
\#\text{ Farben} &= COLORS \\
&\leq \frac{1}{f(1)} + \sum_{i=2}^{n} \frac{4|C|}{(\log i)^2} \\
&= 4|C| + \sum_{i=2}^{n} \frac{4|C|}{(\log i)^2} \\
&\leq 4|C| \cdot \left(1 + \sum_{i=2}^{\lfloor\sqrt{n}\rfloor} \frac{1}{(\log i)^2} + \sum_{i=\lfloor\sqrt{n}\rfloor+1}^{n} \frac{1}{(\log i)^2} \right) \\
&\leq 4|C| \cdot \left(\sqrt{n} + \frac{n}{(\log \sqrt{n})^2} \right) \\
&= O\left(|C| \cdot \frac{n}{(\log n)^2} \right) .
\end{aligned}$$

Wir betrachten nun das Produkt der beiden Approximationsgüten:

$$\frac{\omega(G)}{|C|} \cdot \frac{COLORS}{\chi(G)} = \frac{\omega(G)}{\chi(G)} \cdot O\left(\frac{n}{(\log n)^2} \right) . \tag{3.15}$$

Es gilt stets $\omega(G) \leq \chi(G)$, da zum Färben einer Clique auf $\omega(G)$ Knoten genau $\omega(G)$ viele Farben benötigt werden. Approximationsgüten sind nach Definition mindestens 1. Da das Produkt beider durch $O(n/(\log n)^2)$ beschränkt ist, muss jeder einzelne Faktor in (3.15) durch $O(n/(\log n)^2)$ beschränkt sein und somit erhalten wir eine Approximationsgüte von $O(n/(\log n)^2)$ für die Probleme Clique und Coloring[2]. □

Korollar 3.21 *Der Algorithmus* Clique-Removal *approximiert für einen Graphen* $G = (V, E)$ *eine unabhängige Menge* I *mit Güte* $O(n/(\log n)^2)$.

Beweis: Die beiden Algorithmen Independent-Set-Removal und Clique-Removal unterscheiden sich nur dahingehend, dass ersterer eine Lösung für das Problem Clique approximiert, indem er unabhängige Mengen aus dem Graphen entfernt, und letzterer eine Lösung für das Problem Independent Set approximiert, indem er Cliquen entfernt. Da Cliquen gerade das Komplement von unabhängigen Mengen sind, haben beide Algorithmen die gleiche Approximationsgüte. Das Korollar folgt damit direkt aus Satz 3.20. □

Wir haben also gezeigt, dass wir unabhängige Mengen mit einer Güte von $O(n/(\log n)^2)$ approximieren können. An dieser Stelle wollen wir explizit die Größe einer unabhängigen Menge I bestimmen. Insbesondere sind wir natürlich an unteren Schranken bezüglich der Kardinalität dieser unabhängigen Menge interessiert. Zunächst betrachten wir folgende Definition:

[2] Für das Problem Coloring sind bereits bessere Approximationsalgorithmen, beispielsweise von Halldórsson und Radhakrishnan [21] mit einer Approximationsgüte von $O(n(\log\log n)^2/(\log n)^3)$, bekannt.

Definition 3.22 ($l_k(n)$) *Die Zahl $l_k(n)$ sei die kleinste Zahl l, so dass jeder Graph auf n Knoten eine Clique der Größe k oder eine unabhängige Menge der Größe l hat:*

$$l_k(n) = \min\{l \mid r(k,l) \geq n\} \geq \min\left\{ l \;\middle|\; \binom{k+l-2}{l-1} \geq n \right\} .$$

Falls ein Graph $G = (V, E)$ auf n Knoten keine Clique der Größe k hat, dann enthält G demnach eine unabhängige Menge der Größe $l_k(n)$. Für den Algorithmus Clique-Removal können wir die Größe der gefundenen unabhängigen Menge I folgendermaßen abschätzen:

Satz 3.23 *Es sei $G = (V, E)$ ein Graph und k die kleinste natürliche Zahl, so dass $\alpha(G) > n/k$ gilt. Weiter sei $\varepsilon = \alpha(G)/n - 1/k$. Dann findet der Algorithmus* Clique-Removal *eine unabhängige Menge I der Kardinalität*

$$|I| \geq \max\{l_k(\varepsilon \cdot n), l_{k+1}(n/k^2)\} .$$

Beweis: Da die Unabhängigkeitszahl $\alpha(G)$ des Graphen $G = (V, E)$ echt größer als n/k ist, findet der Algorithmus Clique-Removal schließlich irgendwann keine k-Clique mehr, welche entfernt werden könnte. Für die Kardinalität der Knotenmenge V' des Graphen $G' = (V', E')$, der zu diesem Zeitpunkt vorliegt, gilt dann wegen Lemma 3.18

$$|V'| \geq \frac{\varepsilon}{1 - 1/k} \cdot n \geq \varepsilon \cdot n .$$

Da der Graph G' keine k-Clique mehr hat, muss es eine unabhängige Menge I der Kardinalität

$$|I| \geq l_k(|V'|) \geq l_k(\varepsilon \cdot n)$$

geben, welche vom Algorithmus Ramsey effizient gefunden wird.
Der Fall, dass keine $k+1$-Clique gefunden wird, tritt schon eher ein. Wir wollen nun die Anzahl Knoten abschätzen, welche in diesem Fall noch im Graphen $G' = (V', E')$ sind. Es gilt

$$\alpha(G) \geq \left(\frac{1}{k} + \varepsilon\right) \cdot n = \left(\frac{1}{k+1} + \varepsilon + \frac{1}{k} - \frac{1}{k+1}\right) \cdot n .$$

Wir setzen

$$\varepsilon' = \varepsilon + \frac{1}{k} - \frac{1}{k+1}$$

und erhalten mit Hilfe von Lemma 3.18

$$\begin{aligned} |V'| &\geq \frac{\varepsilon'}{1 - 1/(k+1)} \cdot n \\ &= \frac{\varepsilon + 1/k - 1/(k+1)}{1 - 1/(k+1)} \cdot n \\ &\geq \frac{n}{k^2} . \end{aligned}$$

Falls im Graphen G' keine Clique der Größe $k+1$ enthalten ist, können wir mit Hilfe des Algorithmus RAMSEY effizient eine unabhängige Menge I bestimmen, für die gilt

$$|I| \geq l_{k+1}(|V'|) \geq l_{k+1}(n/k^2) \; .$$

Damit gilt für die vom Algorithmus CLIQUE-REMOVAL gefundene unabhängige Menge I

$$|I| \geq \max\{l_k(\varepsilon \cdot n), l_{k+1}(n/k^2)\} \; . \qquad \square$$

Wir wollen nun den Wert $l_k(n)$ abschätzen. Wir unterscheiden dabei abhängig von den Werten für l und k verschiedene Fälle. Zunächst betrachten wir den Fall, dass der gegebene Graph $G = (V, E)$ eine große unabhängige Menge und eine kleine Clique enthält. Dann gilt

$$\begin{aligned}
& \binom{k+l-2}{k-1} && \geq && n \\
\Longrightarrow \quad & \left(\frac{e \cdot (k+l-2)}{k-1}\right)^{k-1} && \geq && n \\
\Longleftrightarrow \quad & \frac{e \cdot (k+l-2)}{k-1} && \geq && n^{1/(k-1)} \\
\Longleftrightarrow \quad & l && \geq && e^{-1} \cdot (k-1) \cdot n^{1/(k-1)} - k + 2 \; . \qquad (3.16)
\end{aligned}$$

Damit erhalten wir nun das folgende Korollar:

Korollar 3.24 *Es sei $G = (V, E)$ ein Graph und k die kleinste natürliche Zahl, so dass $\alpha(G) \geq n/k + m$ gilt. Dann findet der Algorithmus* CLIQUE-REMOVAL *eine unabhängige Menge der Größe $\Omega(m^{1/(k-1)})$.*

Beweis: Nach Lemma 3.18 enthält der Graph G nach dem Entfernen aller Cliquen der Größe k mindestens m Knoten. Aufgrund obiger Überlegungen enthält die gefundene unabhängige Menge $\Omega(m^{1/(k-1)})$ Knoten. $\square$

Wir haben hier vorausgesetzt, dass der Wert k klein ist, das heißt, der Graph enthält nur Cliquen geringer Größe. Falls k groß ist, dann ist die Abschätzung (3.16) von der Art $l \geq 1$, das bedeutet, wir hätten eine Aussage, welche trivialerweise erfüllt ist, da der Ausdruck $n^{1/(k-1)}$ für $n \to \infty$ und $k \to n$ gegen 1 konvergiert. Für diesen Fall werden wir daher eine andere Abschätzung vornehmen.

$$\begin{aligned}
& \binom{k+l-2}{l-1} && \geq && n \\
\Longrightarrow \quad & \left(\frac{e \cdot (k+l-2)}{l-1}\right)^{l-1} && \geq && n \\
\Longleftrightarrow \quad & (l-1) \cdot \log\left(\frac{e \cdot (k+l-2)}{l-1}\right) && \geq && \log n \; .
\end{aligned}$$

Für kleine Werte von l und große Werte für k, etwa $l \leq \log n$ und $k \geq 2 \log n$, erhalten wir

$$l \geq \frac{\log n}{\log(k/\log n)} \; .$$

4 DER ALGORITHMUS VON FEIGE

In Kapitel 3 haben wir den von Boppana und Halldórsson entwickelten Algorithmus Clique-Removal [7] zur Bestimmung unabhängiger Mengen betrachtet. Dieser Algorithmus entfernt in jedem Schritt die vom Algorithmus Ramsey gefundene Clique. Durch das Entfernen einer Clique, das heißt eines vollständigen Teilgraphen, wird die Kantendichte im Restgraphen $G' = (V', E')$ geringer. Der Gedanke dabei ist, dass eine große unabhängige Menge in G' einfacher zu finden ist, wenn G' wenige Kanten enthält. Das Entfernen einer Clique verringert die Größe einer unabhängigen Menge nur um maximal einen Knoten, da jede unabhängige Menge höchstens einen Knoten dieser Clique enthalten kann.
Umgekehrt, falls wir eine Lösung für das Problem Clique approximieren wollen, können wir aufgrund der Dualität zwischen den Problemen Clique und Independent Set analog vorgehen. Das bedeutet, dass wir dann in jedem Schritt die vom Algorithmus Ramsey gefundene unabhängige Menge entfernen müssen. In Kapitel 3 wurde dieses Vorgehen durch den Algorithmus Independent-Set-Removal verdeutlicht.
Wir wollen uns in diesem Kapitel nun eine Weiterentwicklung dieser Gedanken von Feige [11] anschauen. Der Grundgedanke von Feige war, dass wir zur Approximation einer Clique nicht nur unabhängige Mengen aus dem gegebenen Graphen $G = (V, E)$ entfernen, sondern dass wir zu diesem Zweck auch dünnbesetzte Teilgraphen, das heißt Teilmengen $S \subseteq V$ der Knotenmenge V entfernen, wobei zwischen Knoten aus S nur „wenige“ Kanten verlaufen.

Wir wollen zunächst definieren, was wir unter einem Teilgraphen, der nur „wenige“ Kanten enthält, verstehen.

Definition 4.1 (schwache Knotenmenge) *Sei $G = (V, E)$ ein Graph mit einer Clique C der Kardinalität $|C| \geq n/k$. Eine Knotenmenge S in G heißt schwach, wenn der auf S induzierte Untergraph von G keine Clique C_S der Kardinalität $|C_S| \geq |S|/(2k)$ enthält.*

Lemma 4.2 *Sei $G = (V, E)$ ein Graph und S_1 sowie S_2 zwei beliebige disjunkte, schwache Knotenmengen von G. Dann ist die Knotenmenge $S_1 \cup S_2$ ebenfalls schwach.*

Beweis: Nach Definition 4.1 wissen wir, dass der auf S_1 induzierte Untergraph von G keine Clique C_{S_1} der Kardinalität $|C_{S_1}| \geq |S_1|/(2k)$ besitzen kann. Analog gilt, dass der auf S_2 induzierte Untergraph von G keine Clique C_{S_2} der Kardinalität $|C_{S_2}| \geq |S_2|/(2k)$ enthält.

Wir nehmen nun an, dass die Knotenmenge $S_1 \cup S_2$ nicht schwach ist, das heißt, der auf

$S_1 \cup S_2$ induzierte Untergraph von G enthält eine Clique C_S, für die gilt

$$|C_S| \geq \frac{|S_1| + |S_2|}{2k} .$$

Falls die Menge S_1 mehr als $|S_1|/(2k)$ Knoten der Clique C_S enthält, dann bilden diese Knoten natürlich auch in S_1 eine Clique im Widerspruch zur Voraussetzung, dass S_1 eine schwache Knotenmenge ist. Umgekehrt, nämlich, falls die Menge S_1 weniger als $|S_1|/(2k)$ Knoten der Clique C_S enthält, befinden sich in der Knotenmenge S_2 mindestens $|S_2|/(2k)$ Knoten der Clique C_S, welche offenbar eine Clique bilden, im Widerspruch dazu, dass die Knotenmenge S_2 schwach ist. □

Satz 4.3 *Sei $G = (V, E)$ ein Graph mit einer Clique C der Kardinalität $|C| \geq n/k$. Seien $S_1, \ldots, S_l$ beliebige disjunkte schwache Knotenmengen von G. Sei $G' = (V', E')$ der auf $V \setminus (S_1 \cup \ldots \cup S_l)$ induzierte Untergraph von G. Dann gilt für die Anzahl der Knoten in G'*

$$|V'| \geq \frac{n}{2k} .$$

Außerdem enthält der Graph $G' = (V', E')$ eine Clique C' der Kardinalität

$$|C'| > \frac{|V'|}{k} . \tag{4.1}$$

Beweis: Aufgrund von Definition 4.1 besitzt der auf S_i induzierte Untergraph keine Clique der Größe $|S_i|/(2k)$, $i = 1, \ldots, l$. Weiterhin ist wegen Lemma 4.2 die Vereinigung $S = \bigcup_{i=1}^{l} S_i$ wieder eine schwache Knotenmenge. Jeder Untergraph von $G = (V, E)$ hat höchstens n Knoten. Also hat der auf S induzierte Untergraph von G höchstens n Knoten und keine Clique der Kardinalität $n/(2k)$.

Nach Voraussetzung hat G eine Clique C der Größe $|C| \geq n/k$. Da der auf S induzierte Untergraph von G keine Clique der Größe $n/(2k)$ haben kann, müssen mindestens $n/(2k)$ Knoten der Clique C im Graphen G' verblieben sein. Damit gilt

$$|V'| \geq \frac{n}{2k} .$$

Die Formel (4.1) beweisen wir, indem wir annehmen, dass für alle Cliquen C' im Graphen $G' = (V', E')$ gilt

$$|C'| \leq \frac{|V'|}{k} .$$

Dieses bedeutet jedoch, dass sich mindestens

$$\frac{|V|}{k} - \frac{|V'|}{k} = \frac{|S|}{k}$$

Knoten der Clique C in der Knotenmenge S befinden und damit auch eine Clique C_S der Größe

$$|C_S| \geq \frac{|S|}{k}$$

im auf S induzierten Untergraphen von G bilden, welches im Widerspruch dazu steht, dass S eine schwache Knotenmenge ist. □

Algorithmus 4.1 Feige$(G = (V, E), k, t)$

```
1:  V' ← V
2:  E' ← E
3:  loop
4:      V'' ← V'
5:      E'' ← E'
6:      C ← ∅
7:      loop
8:          if |V''| < 6kt then
9:              break
10:         P_1, ..., P_l ← P_i ⊆ V'' : |P_i| = 2kt, ∀ i ≠ j : P_i ∩ P_j = ∅
11:         good ← false
12:         i ← 1
13:         repeat
14:             for all S ⊂ P_i, |S| = t do
15:                 clique ← true
16:                 for all v, w ∈ S, v ≠ w do
17:                     if {v, w} ∉ E'' then
18:                         clique ← false
19:                         break
20:                 if clique = true then
21:                     N(S) ← {v | v ∈ V'' \ S, ∀ w ∈ S : {v, w} ∈ E''}
22:                     if |N(S)| ≥ |V''|/(2k) − t then
23:                         good ← true
24:                         break
25:             i ← i + 1
26:         until good = true or i = l
27:         if good = true then
28:             C ← C ∪ S
29:             V'' ← N(S)
30:             E'' ← {{v, w} | v ∈ V'' and w ∈ V'' and {v, w} ∈ E''}
31:         else
32:             break
33:     if C ≥ t · log_{3k}(|V'|/(6kt)) then
34:         return C
35:         break
36:     if good = false then
37:         V' ← V' \ V''
38:         E' ← E' \ {{v, w} | v ∈ V'' or w ∈ V''}
```

4.1 Verständnis des Algorithmus

Der Algorithmus von Feige liefert nach Eingabe der Parameter k und t sowie eines Graphen $G = (V, E)$, welcher eine Clique der Größe $|V|/k$ enthält, eine Clique C der Kardinalität $|C| \geq t \cdot \log_{3k}(|V|/t - 3)$ für einen Parameter t, der im Abschnitt zur Laufzeitanalyse des Algorithmus noch genauer spezifiziert wird.

Man kann diesen Algorithmus in Phasen (*die Zeilen 3 bis 38*) und Iterationen (*die Zeilen 7 bis 32*) einteilen, wobei jede Phase aus mehreren einzelnen Iterationen bestehen kann. Eine Phase erhält als Eingabe einen Graph $G' = (V', E')$, welcher eine Clique der Größe $|V'|/k$ enthält. In der ersten Phase wird mit dem Graphen $G = (V, E)$, der nach Voraussetzung eine Clique dieser Größe enthält, gestartet (*vgl. Zeilen 1 und 2*).

Nach Abarbeitung mehrerer einzelner Iterationen endet eine Phase, wobei eine der folgenden zwei Bedingungen erfüllt ist:

1. Eine Clique C der Kardinalität $|C| \geq t \cdot \log_{3k}(|V'|/(6kt))$ wurde gefunden.
2. Ein schwacher Untergraph $G'' = (V'', E'')$ wurde gefunden.

Falls die erste Bedingung erfüllt ist, wird die Clique C ausgegeben (*Zeile 34*) und der Algorithmus *terminiert* (*Zeile 35*). Bei Erfüllung der zweiten Bedingung wird der schwache Untergraph $G'' = (V'', E'')$, welcher in den einzelnen Iterationen berechnet wird, aus dem Graphen $G' = (V', E')$ entfernt (*die Zeilen 37 und 38*) und eine neue Phase mit dem Restgraphen beginnt. Wegen Satz 4.3 ist die Schleifeninvariante erfüllt, das heißt der Restgraph enthält eine Clique der Kardinalität $|V'|/k$. Zusätzlich gilt, dass $|V'|$ nicht kleiner als $n/(2k)$ werden kann.

Bevor wir uns einer einzelnen Iteration zuwenden werden, wollen wir die ausgegebene Clique C genauer analysieren. Aufgrund der Bedingung in *Zeile 33* hat diese Clique eine Kardinalität von

$$\begin{aligned} |C| &\geq t \cdot \log_{3k} \frac{|V'|}{6kt} \\ &\geq t \cdot \log_{3k} \frac{n}{2k \cdot 6kt} && (\text{da } |V'| \geq n/(2k)) \\ &= t \cdot \left(\log_{3k} \frac{n}{t} - \log_{3k} 12k^2 \right) . \end{aligned}$$

Wir schätzen nun den Term

$$f(k) := \log_{3k} 12k^2 = \frac{\ln(12k^2)}{\ln(3k)}$$

mit 3 ab, da folgende Äquivalenz erfüllt ist

$$\begin{aligned} & \log_{3k} 12k^2 \leq 3 \\ \iff & 12k^2 \leq (3k)^3 \\ \iff & 1 \leq \frac{9}{4}k . \end{aligned}$$

Damit gilt

$$t \cdot (\log_{3k}(n/t) - \log_{3k}(12k^2)) > t \cdot (\log_{3k}(n/t) - 3)$$

und der Algorithmus liefert eine Clique C der Größe

$$|C| \geq t \cdot (\log_{3k}(|V|/t) - 3) \; . \tag{4.2}$$

Wir analysieren nun eine einzelne Iteration. In jeder Iteration wird ein Untergraph $G'' = (V'', E'')$ von G beziehungsweise von G' (*vgl. Zeilen 4 und 5*) betrachtet. Für die erste Iteration einer Phase wird der Graph $G' = (V', E')$ verwendet sowie die Menge C auf die leere Menge gesetzt (*Zeile 6*). In *Zeile 8* wird zunächst überprüft, ob die Knotenmenge V'' weniger als $6kt$ Knoten enthält. Ist dies der Fall, wird die Schleife der Iterationen beendet. Die Clique C hat dann die Kardinalität $|C| \geq t \cdot \log_{3k}(|V'|/(6kt))$, wie wir später beweisen werden, und der Algorithmus terminiert daraufhin.

Anderenfalls partitionieren wir in *Zeile 10* die Knotenmenge V'' in disjunkte Knotenmengen P_i, $i = 1, \ldots, l$ ($l = \lceil V''/(2kt) \rceil$), der Größe $|P_i| = 2kt$. Zur Vereinfachung können wir annehmen, dass $2kt$ ein Teiler von $|V''|$ ist. Der Algorithmus kann einfach modifiziert werden, um auch den Fall, dass $2kt$ kein Teiler von $|V''|$ ist, zu bewältigen, ohne dass dies größere Auswirkungen auf die Clique C, welche der Algorithmus ausgibt, hat. Dazu fügen wir beispielsweise isolierte Dummyknoten hinzu, das heißt Knoten, die mit keinem anderen Knoten verbunden sind.

Zum weiteren Verständnis benötigen wir die folgende Definition.

Definition 4.4 (gute Knotenmenge) *Es sei der Graph $G = (V, E)$, der eine Clique C der Größe $|C| \geq |V|/k$ enthält, sowie die Parameter t und k die Eingabe für den Algorithmus von Feige. Weiterhin sei $S \subseteq V$, $|S| = t$, und $N(S) \subseteq V \setminus S$ die Menge aller Knoten aus $V \setminus S$, die mit jedem Knoten aus S inzidieren. Wir bezeichnen S als gute Knotenmenge, wenn folgende zwei Bedingungen erfüllt sind:*

1. *S ist eine Clique.*
2. *Es gilt $|N(S)| \geq |V|/(2k) - t$.*

In den *Zeilen 13 bis 26* betrachten wir nun nacheinander alle t-elementigen Knotenmengen von P_i, $i = 1, \ldots, l$, bis wir entweder eine *gute Knotenmenge* S gefunden haben (*siehe Zeilen 15 bis 24*) oder alle Knotenmengen P_i abgearbeitet sind. Falls wir eine gute Knotenmenge S gefunden haben, fügen wir diese zur Clique C hinzu und starten die nächste Iteration, welche als neue Eingabe den auf $N(S)$ (vgl. Definition 4.4) induzierten Untergraphen von G'' erhält.

Im anderen Fall, das heißt, falls wir in keiner der Knotenmengen P_i, $i = 1, \ldots, l$, eine gute Knotenmenge S gefunden haben, ist die Knotenmenge V'' schwach. Die Schleife der Iterationen wird daraufhin in *Zeile 32* verlassen und der auf V'' induzierte Untergraph auf G' aus dem Graphen G' entfernt (*Zeilen 37 und 38*) und eine neue Phase beginnt.

4.2 Korrektheit des Algorithmus

In diesem Abschnitt wollen wir die Korrektheit des Algorithmus beweisen. Zum einen müssen wir natürlich sicherstellen, dass die ausgegebene Menge C eine *Clique* der Kardinalität

$$|C| \geq t \cdot \log_{3k} \frac{|V'|}{6kt}$$

im induzierten Untergraphen von $G = (V, E)$ ist. Andererseits müssen wir zeigen, dass der Algorithmus einen schwachen Untergraphen $G'' = (V'', E'')$ sicher erkennt, das heißt, dieser Untergraph darf keine Clique C der Größe $|C| \geq |V''|/(2k)$ enthalten. Wir zeigen zunächst das Letztere.

Satz 4.5 *Wenn eine Knotenmenge V'' vom Algorithmus von Feige als schwach erkannt wird, dann enthält der auf V'' induzierte Untergraph von $G = (V, E)$ tatsächlich keine Clique C der Kardinalität $|C| \geq |V''|/(2k)$.*

Beweis: Angenommen der auf $V'' = P_1 \cup \ldots \cup P_l$ induzierte Untergraph von G enthält eine Clique C der Kardinalität

$$|C| \geq \frac{|V''|}{2k} = t \cdot l \ .$$

Nach dem Schubfachprinzip gibt es dann ein $i \in \{1, \ldots, l\}$, so dass P_i mindestens t Knoten dieser Clique enthält. Die Menge $S \subset P_i$, welche t dieser Knoten enthält, ist eine Clique und $N(S)$ enthält die restlichen Knoten der Clique C, so dass gilt

$$|N(S)| \geq \frac{V''}{2k} - t \ .$$

Damit sind beide Bedingungen aus Definition 4.4 erfüllt und V'' wird nicht als *schwach* bezeichnet. □

Nun zeigen wir, dass die Ausgabe des Algorithmus tatsächlich eine Clique C der Kardinalität

$$|C| \geq t \cdot \log_{3k} \frac{|V'|}{6kt}$$

ist.

Satz 4.6 *Endet eine Phase mit der Ausgabe einer Menge C, dann enthält diese mindestens*

$$|C| \geq t \cdot \log_{3k} \frac{|V'|}{6kt} \tag{4.3}$$

Knoten, welche eine Clique in $G' = (V', E')$ und damit auch in $G = (V, E)$ bilden.

Beweis: Zuerst wollen wir einsehen, dass die Menge C in der Tat eine Clique ist. Dazu schauen wir uns den Algorithmus noch einmal genauer an. Vor der ersten Iteration einer

Phase wird die Menge C auf die leere Menge $\emptyset$ gesetzt, womit C trivialerweise eine Clique ist. Damit haben wir zu Beginn einer Phase immer eine Clique vorliegen.

Wir weisen nun nach, dass die Knotenmenge C im Laufe der Iterationen immer eine Clique bleibt. Dies bedeutet insbesondere, dass *Zeile 28*, also die einzige Zeile mit einer Operation auf der Menge C, überprüft werden muss. Diese Operation wird allerdings nur ausgeführt, wenn in den *Zeilen 13 bis 26* eine *gute Knotenmenge* S gefunden wurde. Aufgrund von Definition 4.4 ist S in diesem Fall eine Clique. Nach der ersten Operation $C \leftarrow C \cup S$ ist C also immer noch eine Clique.

Vor Beginn der nächsten Iteration verringert sich die Knotenmenge V'' in der Art, dass nur jene Knoten in V'' verbleiben, die mit jedem Knoten aus S durch eine Kante verbunden sind (vgl. $N(S)$ in Definition 4.4). Falls wir in dieser neuen (verringerten) Knotenmenge $V'' = N(S)$ eine Clique (in diesem Fall beliebiger Größe) finden und zu C hinzufügen, bleibt C auch weiterhin eine Clique, da jeder Knoten von $N(S)$ mit jedem Knoten aus C verbunden ist.

Wir werden nun zeigen, dass die ausgegebene Menge C mindestens

$$|C| \geq t \cdot \log_{3k} \frac{|V'|}{6kt}$$

Knoten enthält. In jeder, außer der letzten, Iteration der Phase, die mit der Ausgabe der Menge C endet, werden t Knoten zu C addiert. Um die Anzahl Iterationen nach unten abzuschätzen sei $|V''|$ die Anzahl der Knoten am Anfang einer Iteration.

Wegen der Bedingung

$$|N(S)| \geq \frac{|V''|}{2k} - t$$

in *Zeile 22* des Algorithmus, startet die nächste Iteration mit wenigstens $|V''|/(2k) - t$ Knoten. Wenn eine neue Iteration gestartet werden soll, muss wegen *Zeile 8* natürlich

$$|V''| \geq 6kt$$

und damit

$$t \leq \frac{|V''|}{6k}$$

gelten.

Damit startet die nächste Iteration mit wenigstens

$$\begin{aligned} \frac{|V''|}{2k} - t &\geq \frac{|V''|}{2k} - \frac{|V''|}{6k} \\ &= \frac{|V''|}{3k} \end{aligned}$$

Knoten.

Allgemein folgt also, dass die $(i+1)$'te Iteration mit mindestens

$$\frac{|V'|}{(3k)^i}$$

Knoten startet.

Die Frage ist nun: Wie viele Iterationen müssen durchlaufen werden, damit die Anzahl Knoten kleiner als $6kt$ ist? Es gilt folgende Äquivalenz:

$$\begin{aligned} & \frac{|V'|}{(3k)^x} < 6kt \\ \iff \quad & \frac{|V'|}{6kt} < (3k)^x \\ \iff \quad & \log_{3k} \frac{|V'|}{6kt} < x \ . \end{aligned}$$

Es werden also mindestens $\log_{3k}(|V'|/(6kt))$ Iterationen durchlaufen, wobei in jeder Iteration t Knoten zu der Menge C hinzugefügt werden. Folglich enthält die Menge C mindestens

$$|C| \geq t \cdot \log_{3k} \frac{|V'|}{6kt}$$

Knoten und der Satz ist damit bewiesen. □

4.3 Laufzeitanalyse des Algorithmus

Wir werden in diesem Abschnitt die Laufzeit des Algorithmus von Feige analysieren und zeigen, dass diese polynomiell beschränkt ist. Wir beginnen zunächst mit der Analyse der Phasen, bevor wir die einzelnen Iterationen betrachten.

Mehrere Phasen können nur dann auftreten, wenn, bis auf die letzte Phase, alle mit der Erkennung einer schwachen Knotenmenge enden. Die letzte Phase endet generell mit der Ausgabe der Clique C. Wenn eine schwache Knotenmenge S entdeckt wurde, dann gilt für die Kardinalität dieser Knotenmenge $|S| \geq 6kt$. Dadurch erhält man eine obere Schranke für die Anzahl Phasen mit

$$\#\text{ Phasen} \leq \frac{n}{6kt} \ .$$

Die Anzahl Iterationen einer Phase ist offensichtlich durch

$$\#\text{ Iterationen} \leq \frac{|V'|}{t} \leq \frac{n}{t}$$

nach oben beschränkt, da sich die Knotenmenge V' in jeder Iteration um mindestens t Knoten verringert.

Im folgenden wollen wir die Laufzeit einer Iteration analysieren.

Die Anzahl Knotenmengen P_i ist durch

$$\#\text{ Knotenmengen } P_i = \frac{|V''|}{2kt} \leq \frac{n}{2kt}$$

beschränkt. Im schlimmsten Fall müssen alle diese Mengen P_i untersucht werden. Dabei werden für jede Knotenmenge P_i mit $|P_i| = 2kt$ alle möglichen Teilmengen S der Kardinalität $|S| = t$ betrachtet, von denen es

$$\#\ t\text{-elementige Knotenmengen von } P_i = \binom{2kt}{t}$$

viele gibt. Diese Knotenmengen S müssen nun auf die Eigenschaft *gut* getestet werden, das heißt, es muss überprüft werden, ob S eine Clique bildet. Zusätzlich dazu muss die Knotenmenge $N(S)$ berechnet werden. Beides zusammen erfolgt in Laufzeit

$$O\left(\frac{t^2 - t}{2} + t \cdot (|V''| - t)\right) .$$

Damit ergibt sich eine Gesamtlaufzeit des Algorithmus von

$$O\left(\frac{n}{kt} \cdot \frac{n}{t} \cdot \frac{n}{kt} \cdot \binom{2kt}{t} \cdot (t \cdot n)\right) = O\left(\binom{2kt}{t} \cdot \frac{n^4}{k^2 \cdot t^2}\right) .$$

Ein Polynomialzeitverfahren haben wir also vorliegen, wenn der Term

$$\binom{2kt}{t} \tag{4.4}$$

polynomiell in n beschränkt ist. Dieses ist für beliebige Werte des Parameters t nicht der Fall, da (4.4) exponentiell in t wächst. Dann ist (4.4) polynomiell in n beschränkt, wenn es eine Konstante $c > 0$ gibt, so dass gilt

$$\begin{aligned} \binom{2kt}{t} &\leq \left(\frac{e \cdot 2kt}{t}\right)^t = (2ek)^t \\ &\leq \text{poly}(n) \leq n^c . \end{aligned}$$

Für t erhalten wir also

$$t \leq \frac{c \cdot \log n}{\log(2ek)} .$$

Dieser Wert wird maximal, falls k minimal ist. In Abschnitt 4.4 werden wir sehen, dass für den Wert k gilt

$$\frac{\log n}{2 \log\log n} < k < (\log n)^3 .$$

Wir setzen für k die untere Schranke ein und erhalten

$$\begin{aligned} t &\leq \frac{c \cdot \log n}{\log(2ek)} \\ &\leq \frac{c \cdot \log n}{\log(2e \log n) - \log(2 \log\log n)} \\ &= \Theta\left(\frac{\log n}{\log\log n}\right) . \end{aligned} \tag{4.5}$$

Wir haben nun den Wert für den Parameter t maximiert, so dass (4.4) polynomiell in n beschränkt ist. Wir haben jedoch noch nicht überprüft, ob (4.5) einen möglichst großen Wert in (4.3) realisiert. Dazu betrachten wir (4.3) als Funktion in Abhängigkeit von t und bilden die erste Ableitung:

$$\begin{aligned} f(t) &= t \cdot \log_{3k} \frac{|V'|}{6kt} \\ &= \frac{1}{\ln 3k} \cdot t \cdot \ln \frac{|V'|}{6kt} \\ f'(t) &= \frac{1}{\ln 3k} \cdot \left(1 \cdot \ln \frac{|V'|}{6kt} + t \cdot \frac{6kt}{|V'|} \cdot (-1) \cdot \frac{1}{t^2} \cdot \frac{|V'|}{6k} \right) \\ &= \frac{1}{\ln 3k} \cdot \left(\ln \frac{|V'|}{6kt} - 1 \right) . \end{aligned}$$

Die Funktion $f(t)$ ist monoton steigend, falls $f'(t) > 0$ gilt:

$$\begin{aligned} & f'(t) > 0 \\ \iff \qquad & \frac{|V'|}{6ek} > t . \end{aligned}$$

Aufgrund von

$$\begin{aligned} t &= \Theta\left(\frac{\log n}{\log \log n} \right) \\ &< \frac{n}{\log n} \\ &\leq \frac{|V'|}{6ek} \end{aligned}$$

erhalten wir damit das Maximum in (4.3), falls wir t wie in Formel (4.5) setzen.

4.4 Eine Approximationsgüte von $O(n(\log \log n)^2/(\log n)^3)$

Am Ende des letzten Abschnitts haben wir $t = \Theta(\log n / \log \log n)$ gesetzt, um eine polynomiell beschränkte Laufzeit zu erhalten. Mit Hilfe von (4.2) wollen wir die Größe $|C|$ der gefundenen Clique C genauer abschätzen. Aufgrund der Θ-Notation [32] gibt es Werte $c > 0$, $\overline{c} > 0$ und n_0, so dass für alle $n \geq n_0$ gilt

$$\overline{c} \cdot \frac{\log n}{\log \log n} \leq t \leq c \cdot \frac{\log n}{\log \log n} .$$

Dann erhalten wir für die Kardinalität der Clique C

$$\begin{aligned} |C| &\geq t \cdot (\log_{3k}(|V|/t) - 3) \\ &\geq t \cdot \left(\frac{\log n - \log t}{b \cdot \log \log n} - 3 \right) \qquad (k < (\log n)^b \text{ siehe später}) \\ &\geq \overline{c} \cdot \frac{\log n}{\log \log n} \cdot \frac{\log n - \log(c \cdot \log n)/\log \log n}{b \cdot \log \log n} - 3c \cdot \frac{\log n}{\log \log n} \end{aligned}$$

$$
\begin{aligned}
&= \frac{\overline{c}}{b} \cdot L^2 - \overline{c} \cdot L \cdot \frac{\log c}{b \cdot \log\log n} + \overline{c} \cdot L \cdot \frac{\log L}{b \cdot \log\log n} - 3c \cdot L \qquad \left(\text{mit } L = \frac{\log n}{\log\log n}\right) \\
&= \Omega\left(\left(\frac{\log n}{\log\log n}\right)^2\right)
\end{aligned}
$$

Für Graphen $G = (V, E)$ mit $\omega(G) = O(n/\log n)$ erhalten wir also sofort eine Approximationsgüte von $O(n(\log\log n)^2/(\log n)^3)$. Falls sogar $\omega(G) \leq n/(\log n)^3$ gilt, erhalten wir eine Approximationsgüte von $O(n/(\log n)^3)$, indem wir einfach einen beliebigen Knoten des Graphen G ausgeben. Um eine Güte von $O(n(\log\log n)^2/(\log n)^3)$ auch für allgemeine Graphen zu erhalten, werden wir den Algorithmus von Feige etwas modifizieren und uns den Algorithmus Independent-Set-Removal aus Kapitel 3 noch einmal anschauen.
Für den Beweis des folgenden Fakts verweisen wir auf [21]:

Fakt 4.7 *Es sei $G = (V, E)$ ein Graph mit $\omega(G) \geq \sigma \cdot n$, wobei $\sigma \geq 1/\log n$. Dann findet der Algorithmus* Independent-Set-Removal *eine Clique C mit mindestens $|C| \geq n^\sigma/(e \cdot \sigma)$ Knoten, wobei $e = 2,7182818\ldots$ die Eulersche Zahl ist.*

Damit können wir den folgenden Satz beweisen.

Satz 4.8 *Es sei $G = (V, E)$ ein Graph mit einer Clique C der Kardinalität $|C| \geq 2n \cdot \log\log n/\log n$. Dann gibt der Algorithmus* Independent-Set-Removal *eine Clique C mit mindestens $|C| \geq (\log n)^3/(6\log\log n)$ Knoten aus.*

Beweis: Mit $\sigma = 2\log\log n/\log n \geq 1/\log n$ können wir Fakt 4.7 anwenden und erhalten

$$
\begin{aligned}
|C| &\geq \frac{n^\sigma}{e \cdot \sigma} \\
&= \frac{n^{2\log\log n/\log n} \cdot \log n}{e \cdot 2\log\log n} \\
&= \frac{(\log n)^3}{2e \cdot \log\log n} \\
&\geq \frac{(\log n)^3}{6\log\log n}\,.
\end{aligned}
$$

□

Für Graphen $G = (V, E)$ mit $\omega(G) = \Omega(n\log\log n/\log n)$ erhalten wir also eine Güte von $O(n\log\log n/(\log n)^3)$. Wie bereits oben erwähnt, werden wir für den Fall $\omega(G) = O(n\log\log n/\log n)$ den Algorithmus von Feige modifizieren. Ohne eine Modifizierung erhalten wir in diesem Fall nämlich nur eine Approximationsgüte von $O(n(\log\log n)^3/(\log n)^3)$. Wir müssen also einen Faktor von $\Omega(\log\log n)$ einsparen[1], um eine Approximationsgüte von $O(n(\log\log n)^2/(\log n)^3)$ zu erreichen. Zunächst werden wir uns die Änderungen am Quelltext anschauen. Diese betreffen im Wesentlichen nur die Definition einer *guten Knotenmenge*.

[1] Auf eine ähnliche Art verbesserte Halldórsson [21] die Approximationsgüte eines Algorithmus von Berger und Rompel [6] von $O(n(\log\log)^3/(\log n)^3)$ auf eine Güte von $O(n(\log\log n)^2/(\log n)^3)$ für das Problem Coloring.

Genauer gesagt müssen wir *Zeile 23* im Algorithmus von Feige durch die *Zeilen 23a bis 23h* des modifizierten Algorithmus von Feige ersetzen.

Algorithmus 4.2 Feige-modifiziert$(G = (V, E), k, t)$

```
1-22:  //wie Algorithmus von Feige
23a:                    n_test ← f(V'')
23b:                    if |N(S)| > n_test − t then
23c:                        good ← true
23d:                    else
23e:                        (C, I) ← Independent-Set-Removal(G[S ∪ N(S)])
23f:                        if |C| ≥ (log n_test)^3/(6 log log n_test) then
23g:                            C ← C ∪ C
23h:                            return C
24-38: //wie Algorithmus von Feige
```

Wir müssen also Definition 4.4 erweitern zu:

Definition 4.9 (gute Knotenmenge (erweitert)) *Eine Knotenmenge $S \subseteq V''$, wobei V'' die Knotenmenge einer Iteration ist, heißt gute Knotenmenge, falls die folgenden zwei Bedingungen erfüllt sind:*

1. *S ist eine Clique.*
2. *$|N(S)| > n_{test} - t$, wobei n_{test} der größte Wert ist, welcher*

$$n_{test} \leq \frac{\log n_{test}}{2 \log \log n_{test}} \cdot \frac{|V''|}{2k}$$

erfüllt.

Der in dieser Definition auftauchende Wert n_{test} kann mit binärer Suche in Zeit $O(\log n)$ bis auf konstant viele Nachkommastellen berechnet werden. Diese Berechnung erfolgt im modifizierten Algorithmus von Feige durch die Funktion f in *Zeile 23a* in Abhängigkeit von der Kardinalität der Knotenmenge V''.
Zusätzlich enthält der modifizierte Algorithmus von Feige in *Zeile 23e* einen Aufruf des Algorithmus Independent-Set-Removal, falls für die Knotenmenge $N(S)$ gilt

$$\frac{V''}{2k} - t \leq |N(S)| \leq n_{test} - t \; .$$

Wir müssen nun noch die Korrektheit des modifizierten Algorithmus zeigen. Diese Analyse erfolgt ähnlich zu den Sätzen 4.5 und 4.6.

Satz 4.10 *Wenn eine Knotenmenge V'' vom modifizierten Algorithmus von Feige als schwach erkannt wird, dann enthält der auf V'' induzierte Untergraph von $G = (V, E)$ tatsächlich keine Clique C der Kardinalität $|C| \geq |V''|/(2k)$.*

Beweis: Zunächst verläuft dieser Beweis wie der Beweis von Satz 4.5. Angenommen der auf V'' induzierte Untergraph von G enthält eine Clique C der Kardinalität $|C| \geq |V''|/(2k)$, dann existiert nach dem Schubfachprinzip eine Knotenmenge P_i, $i \in \{1, \ldots, l\}$, die mindestens t Knoten der Clique C enthält. Dann gibt es jedoch auch eine t-elementige Menge $S \subset P_i$, welche eine Clique ist und für die $|N(S)| \geq |V''|/(2k) - t$ gilt. Gilt nun zusätzlich

$$|N(S)| > n_{test} - t \; ,$$

dann sind wir fertig, da in diesem Fall S eine gute Knotenmenge nach Definition 4.9 ist und V'' nicht als *schwach* bezeichnet wird. Anderenfalls gilt

$$\frac{|V''|}{2k} - t \leq |N(S)| \leq n_{test} - t \; .$$

Falls $|V_S| = |S \cup N(S)| < n_{test}$ ist, dann fügen wir zur Knotenmenge $V_S = S \cup N(S)$ noch beliebige Knoten aus V'' hinzu, so dass die Anzahl Knoten n_{test} ist. In jedem Fall enthält der auf V_S induzierte Untergraph von G eine Clique C der Größe

$$\begin{aligned} |C| &\geq \frac{|V''|}{2k} \\ &= \frac{n_{test} \cdot 2 \log\log n_{test}}{\log n_{test}} \; . \end{aligned}$$

Wenn wir nun den Algorithmus Independent-Set-Removal auf den Graphen $G[V_S]$ anwenden, erhalten wir nach Satz 4.8 eine Clique C der Kardinalität

$$|C| \geq \frac{(\log n_{test})^3}{6 \log\log n_{test}}$$

und damit endet die Phase, ohne dass V'' vom Algorithmus als *schwach* bezeichnet wird. □

Satz 4.11 *Es seien* $t = \log n / \log\log n$ *und* $\log n/(2\log\log n) < k < \log n$ *die Parameter für den modifizierten Algorithmus von Feige. Endet eine Phase dieses Algorithmus mit der Ausgabe einer Menge* C*, dann ist* C *eine Clique auf* $\Omega(t \cdot \log_b |V'|)$ *Knoten, wobei* $b = \Theta(k \cdot \log\log|V'|/\log|V'|)$*. Insbesondere findet der modifizierte Algorithmus von Feige eine Clique* C *der Größe* $|C| = \Omega((\log n)^2/\log\log n)$*, wenn der Graph der Eingabe eine Clique der Größe* $\Theta(n \log\log n/\log n)$ *hat.*

Beweis: Analog zum Beweis von Satz 4.6 zeigen wir zunächst, dass C tatsächlich eine Clique ist. Im modifizierten Algorithmus von Feige gibt es eine zusätzliche Operation auf der Menge C, nämlich in *Zeile 23g*. Dabei wird die vom Algorithmus Independent-Set-Removal gefundene Clique $\mathcal{C}$ zur bereits bestehenden Clique C hinzugefügt. Die Menge C bleibt dabei eine Clique, da jeder Knoten der Clique $\mathcal{C} \subseteq S \cup N(S)$ mit jedem Knoten aus C (vor der Operation $C \leftarrow C \cup \mathcal{C}$) verbunden ist (vgl. Beweis zu Satz 4.6).

Wir zeigen nun, dass die Clique C die geforderte Größe hat. Zunächst betrachten wir nur Iterationen, bis $|V''| < \sqrt{|V'|}$ gilt. Findet der Algorithmus INDEPENDENT-SET-REMOVAL vorher, das heißt für $|V''| \geq \sqrt{|V'|}$, eine Clique C der Kardinalität

$$|C| \geq \frac{(\log n_{test})^3}{6 \log\log n_{test}} ,$$

dann erhalten wir wegen

$$n_{test} \geq \frac{|V''|}{2k} \geq \frac{\sqrt{|V'|}}{2k}$$

folgende Kardinalität der Clique C:

$$\begin{aligned} |C| &\geq \frac{(\log n_{test})^3}{6 \log\log n_{test}} \\ &\geq \frac{(\log(\sqrt{|V'|}/(2k)))^3}{6 \log\log(\sqrt{|V'|}/(2k))} \\ &= \Omega\left(\frac{(\log |V'|)^3}{\log\log |V'|}\right) . \end{aligned}$$

Falls dies nicht der Fall ist, gilt in jeder Iteration $|N(S)| > n_{test} - t$, wobei n_{test} von $|V''|$ und damit von jeder einzelnen Iteration abhängt. Damit beginnt eine neue Iteration mit mindestens

$$\frac{|V''| \cdot \log n_{test}}{4k \cdot \log\log n_{test}} = \Theta\left(\frac{|V''| \cdot \log |V'|}{k \cdot \log\log |V'|}\right)$$

vielen Knoten. Die Reduzierung der Knotenmenge V'' zwischen zwei aufeinanderfolgenden Iterationen erfolgt dann mit einem Faktor von

$$O\left(\frac{k \cdot \log\log |V'|}{\log |V'|}\right) .$$

Im Beweis von Satz 4.6 war dieser Faktor $O(k)$.
Wir berechnen nun die Anzahl Iterationen x, die nötig sind, um die Knotenmenge auf $6kt$ zu reduzieren. Es gilt folgende Äquivalenz:

$$\begin{aligned} & \left(\frac{\log |V'|}{k \cdot \log\log |V'|}\right)^x \cdot |V'| < 6kt \\ \iff \quad & \frac{\log(|V'|/(6kt))}{\log(\underbrace{k \cdot \log\log |V'|/\log |V'|}_{=b})} < x . \end{aligned}$$

In jeder Iteration werden t Knoten zur Clique C hinzugefügt, so dass schließlich gilt

$$|C| = \Omega(t \cdot \log_b |V'|) .$$

Für Graphen $G = (V, E)$ mit einer Cliquenzahl $\omega(G)$, für die gilt

$$\omega(G) = \Theta\left(\frac{n \cdot \log\log n}{\log n}\right) ,$$

erhalten wir also eine Clique C der Kardinalität

$$\begin{aligned} |C| &\geq \frac{\log n}{\log\log n} \cdot \frac{\log |V'|}{\log(k \cdot \log\log |V'|/\log |V'|)} \\ &\geq \frac{\log n}{\log\log n} \cdot \frac{\log |V'|}{\log(\log n \cdot \log\log |V'|/(2\log\log n \cdot \log |V'|))} \\ &= \Theta\left(\frac{(\log n)^2}{\log\log n}\right) . \end{aligned}$$

□

An dieser Stelle wollen wir unsere Erkenntnisse noch einmal zusammenfassen. Die Eingabe sei ein Graph $G = (V, E)$, welcher eine Clique C der Kardinalität

$$\omega(G) = |C| \geq \frac{|V|}{k} \tag{4.6}$$

enthält, wobei der Parameter k minimal ist. Dieser Parameter ist zunächst unbekannt. Für k kommen jedoch aufgrund der Ganzzahligkeit der Cliquenzahl $\omega(G)$ nur die folgenden n verschiedenen Werte in Frage:

$$k \in \left\{ \frac{|V|}{1}, \frac{|V|}{2}, \ldots, \frac{|V|}{|V|-1}, 1 \right\} . \tag{4.7}$$

In Abhängigkeit von k unterscheiden wir 3 Fälle:

Fall 1: Es sei $k \geq (\log n)^3$. Wir wählen einen beliebigen Knoten $v \in V$. Offensichtlich ist v eine Clique auf einem Knoten. Wir bezeichnen mit C_1 diese Clique auf einem Knoten, also $C_1 := \{v\}$. Für die Cliquenzahl $\omega(G)$ gilt aufgrund des Parameters k

$$\omega(G) \leq \frac{n}{(\log n)^3} .$$

Da die gefundene Clique die Kardinalität $|C_1| = 1$ hat, haben wir sogar eine Approximationsgüte von $O(n/(\log n)^3)$ für den Fall, dass $k \geq (\log n)^3$ gilt.

Fall 2: Es sei $(\log n)^3 > k > \log n/(2\log\log n)$. In diesem Fall benutzen wir den modifizierten Algorithmus von Feige. Falls zusätzlich $\log n \leq k \leq (\log n)^3$ gilt, reicht sogar der unmodifizierte Algorithmus von Feige. Da der Parameter k für den Graphen G jedoch nicht explizit bekannt ist, starten wir den modifizierten Algorithmus mehrfach mit jeweils verschiedenen Werten für k entsprechend (4.7), wobei wir den modifizierten Algorithmus von Feige nur für Werte von k, für die $k < (\log n)^3$ gilt, aufrufen. Damit können wir die Anzahl an Aufrufen des Algorithmus auf $(\log n)^3$ beschränken. Die größte hierbei berechnete Clique sei C_2.

Fall 3: Es sei $k \leq \log n/(2\log\log n)$. Wir wenden den Algorithmus Independent-Set-Removal an und erhalten eine Clique C_3.

Wenn wir nun alle drei Fälle kombinieren und die größte Clique C,

$$C := \max\{C_1, C_2, C_3\} ,$$

ausgeben, dann erhalten wir nach obigen Überlegungen eine Clique mit Approximationsgüte $O(n(\log\log n)^2/(\log n)^3)$, da in mindestens einem der drei Fälle eine Clique mit Güte $O(n(\log\log n)^2/(\log n)^3)$ approximiert wird. Algorithmisch lässt sich dieses folgendermaßen darstellen:

Algorithmus 4.3 Clique-Approximation($G = (V, E)$)

```
1:  if V = ∅ then
2:      return ∅
3:  choose v ∈ V
4:  C_1 ← {v}
5:  C_2 ← ∅
6:  t ← log|V| / log log|V|
7:  for each k ∈ {|V|/1, |V|/2, ..., 1} do
8:      C ← Feige-modifiziert(G, k, t)
9:      if |C| > |C_2| then
10:         C_2 ← C
11: (C_3, I) ← Independent-Set-Removal(G)
12: if |C_1| ≥ |C_2| and |C_1| ≥ |C_3| then
13:     return C_1
14: else
15:     if |C_2| ≥ |C_3| then
16:         return C_2
17:     else
18:         return C_3
```

5 APPROXIMATION UNABHÄNGIGER MENGEN MIT DER ϑ-FUNKTION

In diesem Kapitel werden die Ergebnisse von Alon und Kahale [3] hinsichtlich des Einsatzes der Thetafunktion vorgestellt. Begonnen wird zunächst mit einem Abschnitt, der grundlegende Notationen und Definitionen der linearen Algebra, insbesondere der Matrix- und Vektorrechnung, klärt. Im Anschluss daran soll die nach ihrem Entdecker benannte Lovász ϑ-Funktion betrachtet und untersucht werden. Weiterhin werden wir einen Zusammenhang der ϑ-Funktion mit den Erkenntnissen aus Kapitel 3 zeigen. Daran anschließend betrachten wir den in der Arbeit von Alon und Kahale [3] vorgestellten Algorithmus zur Approximation einer unabhängigen Menge, der auf einem Algorithmus von Karger, Motwani und Sudan [27] aufbaut.

5.1 Mathematische Vorbetrachtungen

Im Folgenden sollen zunächst die mathematischen Grundlagen, und zwar wesentliche Notationen und Definitionen der Matrix- und Vektorrechnung, kennengelernt werden.

Jeder Vektor $x \in \mathbb{R}^n$ sei im Folgenden grundsätzlich ein Spaltenvektor, dessen Einträge durch natürliche Zahlen $i \in \{1, 2, \ldots, n\}$ beziehungsweise durch Knoten $v \in V$ eines Graphens $G = (V, E)$ mit $n = |V|$ indiziert sind. Wir legen meistens eine Indizierung, die auf den Knoten eines Graphen basiert, zugrunde, da wir diese Vorbetrachtungen auf Graphenprobleme anwenden werden. Für einen gegebenen Spaltenvektor $x \in \mathbb{R}^n$ erzeugen wir den dazugehörigen Zeilenvektor x^T durch Transponieren. Ähnliches gilt auch für Matrizen $A \in \mathbb{R}^{n \times n}$, wobei wir unsere Betrachtungen auf quadratische Matrizen, das heißt Matrizen mit gleicher Anzahl Spalten und Zeilen, beschränken werden.

Zunächst betrachten wir einige Notationen. Für zwei Vektoren $x \in \mathbb{R}^n$ und $y \in \mathbb{R}^n$ gilt $x \geq y$, falls $x_v \geq y_v$ für alle $v \in V$ erfüllt ist, wobei mit x_i das Element in Zeile i des Spaltenvektors x gemeint ist. Für eine Matrix $A \in \mathbb{R}^{n \times n}$ bezeichnen wir mit $A_v \in \mathbb{R}^n$ die Spalte v der Matrix A. Weiterhin bezeichnen wir mit $A_{uv} \in \mathbb{R}$ das Element in Zeile u und Spalte v der Matrix A. Den Nullvektor und die Nullmatrix werden wir mit „0“ kennzeichnen. Die Einheitsmatrix $\mathcal{I} \in \mathbb{R}^{n \times n}$ sei die Matrix, welche auf der Hauptdiagonalen die Einträge „1“ hat und deren restliche Einträge gleich Null sind. Die Matrix $\mathcal{J} \in \mathbb{R}^{n \times n}$ ist die Matrix, deren Einträge alle 1 sind. Das Skalarprodukt zweier Vektoren $x \in \mathbb{R}^n$ und $y \in \mathbb{R}^n$ kürzen wir mit $x^T y$ ab.

Nun folgen noch einige wichtige Definitionen und Bemerkungen.

Definition 5.1 (Länge eines Vektors, Einheitsvektor) *Die Länge eines Vektors $a \in \mathbb{R}^n$ ist definiert als $\|a\| = \sqrt{a^T a}$. Gilt $\|a\| = 1$, dann bezeichnen wir a als Einheitsvektor.*

Bemerkung 5.2 Die Cauchy-Schwarz Ungleichung besagt, dass für zwei Vektoren $a \in \mathbb{R}^n$ und $b \in \mathbb{R}^n$ gilt

$$a^T b \leq \|a\| \cdot \|b\| \; . \tag{5.1}$$

Ein wesentlicher Bestandteil unserer Betrachtungen sind orthogonale und insbesondere orthonormale Vektoren, die wir an dieser Stelle definieren werden.

Definition 5.3 (orthogonale und orthonormale Vektoren) *Zwei Vektoren $a \in \mathbb{R}^n$ und $b \in \mathbb{R}^n$ sind orthogonal, wenn $a^T b = 0$ gilt. Die Vektoren a und b heißen orthonormal, falls a und b orthogonal sind und beide Einheitsvektoren sind, das heißt es gilt $\|a\| = \|b\| = 1$.*

Im Folgenden werden wir einige spezielle Arten von Matrizen kennenlernen.

Definition 5.4 (orthogonale Matrix) *Eine Matrix $Q \in \mathbb{R}^{n \times n}$ heißt orthogonal, wenn das Produkt $Q^T \cdot Q$ der Matrizen Q^T und Q die Einheitsmatrix $\mathcal{I}$ ergibt, also*

$$Q^T \cdot Q = \mathcal{I} \; .$$

Da für jede Matrix A gilt

$$(A^T \cdot A)_{uv} = \sum_{k=1}^{n} ((A^T)_{uk} \cdot A_{kv}) = \sum_{k=1}^{n} (A_{ku} \cdot A_{kv}) = A_u^T \cdot A_v \; ,$$

ist Q also genau dann orthogonal, wenn je zwei verschiedene Spalten aus Q orthonormal sind. Weiterhin gilt, dass die transponierte Matrix Q^T einer orthogonalen Matrix Q wiederum eine orthogonale Matrix ist, da $Q^T \cdot Q = \mathcal{I}$ impliziert, dass Q^T die inverse Matrix Q^{-1} zu Q ist und folglich gilt

$$\begin{aligned} & & Q^T \cdot Q &= \mathcal{I} \\ \iff & \underbrace{Q^{-T} \cdot Q^T}_{=\mathcal{I}} \cdot Q \cdot Q^T &= & \underbrace{Q^{-T} \cdot \mathcal{I} \cdot Q^T}_{=\mathcal{I}} \\ \iff & & Q \cdot Q^T &= \mathcal{I} \; . \end{aligned}$$

Definition 5.5 (Diagonalmatrix) *Eine Matrix $D \in \mathbb{R}^{n \times n}$ heißt Diagonalmatrix, wenn $d_{ij} = 0$ für alle $i \neq j$ gilt.*

Definition 5.6 (symmetrische Matrix) *Eine Matrix $A \in \mathbb{R}^{n \times n}$ heißt symmetrisch, falls gilt*

$$A = A^T \; .$$

Eine besondere Eigenschaft von symmetrischen Matrizen ist im folgenden Fakt dargestellt.

Fakt 5.7 *Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt symmetrisch genau dann, wenn es eine Diagonalmatrix $D \in \mathbb{R}^{n\times n}$ und eine orthogonale Matrix $Q \in \mathbb{R}^{n\times n}$ gibt, so dass gilt*

$$A = Q \cdot D \cdot Q^T \ . \tag{5.2}$$

Damit wir die Bedeutung dieses Faktes verstehen, benötigen wir noch zwei Kenngrößen einer Matrix.

Definition 5.8 (Eigenwert, Eigenvektor) *Es sei $A \in \mathbb{R}^{n\times n}$ eine Matrix. Ein Vektor $x \in \mathbb{R}^n$ heißt Eigenvektor, falls für ein $\lambda \in \mathbb{R}$ gilt*

$$A \cdot x = \lambda \cdot x \ .$$

Der Wert λ ist dann ein Eigenwert der Matrix A.

Die Identität (5.2) ist wegen

$$A \cdot Q = Q \cdot D \cdot \underbrace{Q^T \cdot Q}_{=\mathcal{I}}$$

äquivalent zur Gleichung

$$A \cdot Q = Q \cdot D \ ,$$

welche wiederum äquivalent ist zu

$$A \cdot Q_v = Q_v \cdot \lambda_v$$

mit $\lambda_v = D_{vv}$ für alle $v \in V$. Die Elemente auf der Hauptdiagonalen der Matrix D sind demnach die Eigenwerte der Matrix A und die Spalten der Matrix Q sind die entsprechenden Eigenvektoren. Die Gleichung (5.2) wird als Eigenwertzerlegung beziehungsweise Spektralzerlegung der Matrix A bezeichnet.

Mit Hilfe der Eigenwerte lassen sich auch positiv semidefinite Matrizen definieren:

Definition 5.9 (positiv semidefinite Matrix) *Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt positiv semidefinit, falls für alle Eigenwerte $\lambda_i \geq 0$, $i = 1, \ldots, n$, gilt.*

Im nächsten Abschnitt werden wir uns die Lovász $\vartheta(G)$-Zahl eines Graphen $G = (V, E)$ anschauen. Dabei wird jedem Knoten $v \in V$ ein Vektor zugeordnet, so dass diese Vektoren bestimmte Bedingungen erfüllen.

Definition 5.10 (Orthogonale Repräsentation) *Sei $G = (V, E)$ ein Graph. Eine Zuweisung eines Vektors $a_v \in \mathbb{R}^k$ zu jedem Knoten $v \in V$ heißt orthogonale Repräsentation von G, falls für alle $\{v, w\} \notin E$ die Vektoren a_v und a_w orthogonal zueinander sind, das heißt, es gilt $a_v^T a_w = 0$.*

Diese Definition bedeutet insbesondere, wenn für zwei Vektoren $a_v^T a_w \neq 0$ gilt, dann ist die Kante $\{v, w\}$ im Graphen G enthalten. Analog dazu können wir auch orthonormale Repräsentationen definieren.

Definition 5.11 (Orthonormale Repräsentation) *Eine orthogonale Repräsentation (a_v) eines Graphen G heißt orthonormale Repräsentation, falls für jeden Knoten $v \in V$ der Vektor a_v ein Einheitsvektor ist.*

5.2 Die Lovász $\vartheta(G)$-Zahl eines Graphen

Wir werden nun die nach Lovász benannte $\vartheta(G)$-Zahl [35] eines Graphen $G = (V, E)$ definieren. In der Literatur [31] wird $\vartheta(G)$ auch häufig als ϑ-Funktion bezeichnet.

Definition 5.12 (Lovász $\vartheta(G)$-Zahl eines Graphen G) *Es sei $G = (V, E)$ ein Graph mit $|V| = n$. Dann ist die Lovász $\vartheta(G)$-Zahl des Graphen G gleich dem Minimum des Ausdrucks*

$$\max_{v \in V} \frac{1}{(d^T a_v)^2}$$

über alle Einheitsvektoren $d \in \mathbb{R}^k$ und alle orthonormalen Repräsentationen (a_v), $a_v \in \mathbb{R}^k$, von G. Damit ist die Lovász $\vartheta(G)$-Zahl definiert durch

$$\vartheta(G) = \min_{\{d,(a_v)\}} \max_{v \in V} \frac{1}{(d^T a_v)^2} \; . \tag{5.3}$$

Die Lovász $\vartheta(G)$-Zahl wurde bereits umfassend untersucht [17] und hat die besondere Eigenschaft, dass sie stets, wie ein Sandwich [31], zwischen zwei weiteren Kenngrößen des Graphen G beziehungsweise des Komplementgraphen $\overline{G}$ liegt. Diese beiden Größen sind zum einen die Unabhängigkeitszahl $\alpha(G)$ des Graphen G, das heißt die Kardinalität $|I|$ einer maximalen unabhängigen Menge I, und zum anderen die chromatische Zahl $\chi(\overline{G})$ des Komplementgraphen $\overline{G}$, das heißt die minimale Anzahl Farben für eine zulässige Färbung [33] der Knotenmenge V von $\overline{G}$.
Genauer gesagt gilt

$$\alpha(G) \leq \vartheta(G) \leq \chi(\overline{G}) \; . \tag{5.4}$$

Analog gilt auch [31]

$$\omega(G) \leq \vartheta(\overline{G}) \leq \chi(G) \; .$$

Die Berechnung der $\vartheta(G)$-Zahl eines Graphen $G = (V, E)$ kann mit beliebiger Genauigkeit in Polynomialzeit erfolgen [16], wie wir später noch sehen werden. Weiterhin ist bekannt [17, 31, 35], dass die Lovász $\vartheta(G)$-Zahl, also insbesondere (5.3), einige äquivalente Beschreibungen zulässt, von denen wir zwei im folgenden Fakt darstellen werden.

Fakt 5.13 *Die Lovász $\vartheta(G)$-Zahl eines Graphen $G = (V, E)$ hat folgende äquivalente Beschreibungen:*

1. *Es sei $G = (V, E)$ ein Graph auf $n = |V|$ Knoten. Dann gilt für die Lovász $\vartheta(G)$-Zahl*

$$\vartheta(G) = \max_B \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} \ , \tag{5.5}$$

wobei $B \in \mathbb{R}^{n \times n}$ eine symmetrische, positiv semidefinite Matrix ist, für die gilt

$$\sum_{i=1}^{n} b_{ii} = 1 \tag{5.6}$$

und

$$b_{ij} = 0$$

für alle Kanten $\{i, j\} \in E$ des Graphen G. Aufgrund der Symmetrie gilt in diesem Fall auch $b_{ji} = 0$.

2. *Es sei (b_v) eine orthonormale Repräsentation des Graphen $\overline{G}$ und $d \in \mathbb{R}^k$ ein Einheitsvektor. Dann gilt*

$$\vartheta(G) = \max_{\{d,(b_v)\}} \sum_{i=1}^{n} (d^T b_i)^2 \ . \tag{5.7}$$

Bemerkung 5.14 Der Ausdruck (5.6) ist äquivalent zu

$$\text{Tr } B = 1 \ ,$$

wobei Tr B die Spur, das heißt die Summe der Hauptdiagonalelemente, der Matrix B ist:

$$\text{Tr } B = \sum_{i=1}^{n} b_{ii} \ .$$

Dementsprechend können wir (5.5) auch in folgender Weise aufschreiben:

$$\vartheta(G) = \max_B \text{Tr } (B \cdot \mathcal{J}) \ . \tag{5.8}$$

Mit Hilfe der Darstellung (5.7) lässt sich sofort die linke Ungleichung in (5.4) zeigen:

Satz 5.15 *Es sei $G = (V, E)$ ein Graph auf $n = |V|$ Knoten. Dann gilt*

$$\alpha(G) \leq \vartheta(G) \ .$$

Beweis: Es sei I eine beliebige maximale unabhängige Menge des Graphen G, das heißt, es gilt $|I| = \alpha(G)$. Wir konstruieren nun eine orthonormale Repräsentation (b_v) der Knoten des Komplementgraphen $\overline{G}$. Die Dimension der Vektoren der orthonormalen Repräsentation (b_v) setzen wir auf

$$k := n - \alpha(G) + 1 \ .$$

Weiterhin sei $e \in \mathbb{R}^k$ der Einheitsvektor, welcher an erster Position eine 1 und an den restlichen Positionen Nullen enthält:

$$e := (\ 1 \quad 0 \quad \ldots \quad 0\ )^T\ .$$

Für alle Knoten $v \in I$ der unabhängigen Menge I, setzen wir $b_v := e$. Wir müssen nun noch den $(n-|I|)$ vielen Knoten, die nicht in der unabhängigen Menge I liegen, einen Einheitsvektor zuweisen. Dafür konstruieren wir eine Familie von paarweise orthogonalen Einheitsvektoren $f_i \in \mathbb{R}^k$ mit $i = 1, \ldots, (n-|I|)$, indem der Vektor f_i an der $(i+1)$'ten Position eine 1 und an allen anderen Positionen nur Nullen enthält. Wir weisen nun jedem Knoten $v \in V \setminus I$ genau einen Vektor f_i zu. Diese Zuweisung kann beliebig erfolgen, die einzige Einschränkung ist, dass wir jeden Vektor f_i genau einmal verwenden. Offensichtlich gilt $e^T f_i = 0$ für alle $i = 1, \ldots, (n-|I|)$. Wir müssen nun noch überprüfen, ob die zugewiesenen Vektoren eine orthonormale Repräsentation (b_v) des Komplementgraphen $\overline{G}$ bilden. Nach unserer Konstruktion gilt für je zwei Vektoren b_i und b_j mit $i \neq j$ genau dann $b_i^T b_j \neq 0$, wenn die Knoten i und j in der unabhängigen Menge I enthalten sind. In diesem Fall sind jedoch diese beiden Knoten im Graphen $\overline{G}$ durch eine Kante miteinander verbunden und folglich muss $b_i^T b_j = 0$ wegen Definition 5.12 nicht gelten. Wir vervollständigen unseren Beweis durch folgende Rechnung

$$\begin{aligned}
\alpha(G) = |I| &= \underbrace{\sum_{v \in I} (e^T e)^2}_{=|I|} + \underbrace{\sum_{v \notin I} (e^T f_v)^2}_{=0} \\
&= \sum_{i=1}^{k} (e^T b_v)^2 \\
&\leq \max_{\{e,(b_v)\}} \sum_{i=1}^{k} (e^T b_i)^2 = \vartheta(G)\ .
\end{aligned}$$

□

5.3 Eine Verbindung der ϑ-Funktion mit der Ramsey-Theorie

In Kapitel 3 haben wir den Algorithmus RAMSEY kennengelernt, welcher in einem Graphen $G = (V, E)$ mit $n = |V|$ effizient eine Clique C und eine unabhängige Menge I berechnet, so dass gilt (vergleiche Satz 3.16)

$$r(|C| + 1, |I| + 1) \geq n\ .$$

Den Algorithmus RAMSEY haben wir im Algorithmus CLIQUE-REMOVAL iterativ angewendet und damit eine unabhängige Menge approximiert. Wir haben gesehen, dass bei Eingabe eines Graphen $G = (V, E)$ mit $|V| = n$, welcher eine unabhängige Menge I der Größe $|I| \geq n/k + m$ enthält, eine unabhängige Menge der Größe $l_k(m)$ (vergleiche Definition 3.22) in Polynomialzeit gefunden werden kann. Die Idee hinter dem Algorithmus CLIQUE-REMOVAL war, dass wir wiederholt Cliquen aus dem Graphen G entfernt haben, bis der Graph G keine Clique C

der Kardinalität $|C| = k$ hatte. Wegen Lemma 3.18 enthält der Graph nach dem Entfernen der Cliquen mindestens m Knoten und folglich können wir effizient eine unabhängige Menge I bestimmen mit der Kardinalität

$$|I| \geq l_k(m) = \Omega(m^{1/(k-1)}) \; .$$

Wir werden im Folgenden dieses Ergebnis mit der Lovász $\vartheta(G)$-Zahl in Verbindung bringen. Eine genauere Betrachtung des Algorithmus Clique-Removal führt zu folgendem Satz:

Satz 5.16 *Es sei $G = (V, E)$ ein Graph und $\overline{G}$ der Komplementgraph des Graphen G. Gilt für die chromatische Zahl $\chi(\overline{G}) \geq n/k + m$, dann kann in G eine unabhängige Menge I der Größe $|I| \geq l_k(m)$ in Polynomialzeit gefunden werden.*

Beweis: Offenbar entspricht die chromatische Zahl $\chi(\overline{G})$ des Graphen $\overline{G}$ gerade der Cliquenüberdeckungszahl $\overline{\chi}(G)$ (vgl. Definition A.6) des Graphen G. Da in jedem Schritt eine Clique aus dem Graphen G entfernt wird, verringert sich die Cliquenüberdeckungszahl $\overline{\chi}(G)$ in jedem Schritt um höchstens 1. Zu dem Zeitpunkt, da keine Clique der Größe k im Graphen enthalten ist, sind höchstens n/k viele Cliquen entfernt worden. Für die Cliquenüberdeckungszahl $\overline{\chi}(G')$ des Restgraphen G' zu diesem Zeitpunkt gilt $\overline{\chi}(G') \geq m$. Damit enthält der Graph G' mindestens m Knoten und keine Clique C der Kardinalität $|C| \geq k$. Nach den Überlegungen aus Kapitel 3 kann dann mit dem Algorithmus Ramsey im Graphen G' eine unabhängige Menge der Größe $l_k(m)$ in Polynomialzeit gefunden werden. □

Korollar 5.17 *Es sei $G = (V, E)$ ein Graph auf $n = |V|$ Knoten mit $\vartheta(G) \geq n/k + m$. Dann kann in G eine unabhängige Menge I der Kardinalität $|I| \geq l_k(m)$ in Polynomialzeit gefunden werden.*

Beweis: Wegen (5.4) gilt auch $\chi(\overline{G}) \geq n/k + m$ und wir können Satz 5.16 anwenden. □

5.4 Eine Verbesserung von Alon und Kahale

Im vorangegangenen Abschnitt haben wir gesehen, dass wir unter der Bedingung $\vartheta(G) \geq n/k + m$ eine unabhängige Menge der Größe $\Omega(m^{1/(k-1)})$ approximieren können. Im Folgenden werden wir zeigen, wie man mit Hilfe der semidefiniten Programmierung und dabei insbesondere der Lovász ϑ-Funktion eine unabhängige Menge der Größe $\tilde{\Omega}(m^{3/(k+1)})$ finden kann, wobei die Notation $\tilde{\Omega}(n)$ mit $\Omega(n/poly(\log n))$ gleichbedeutend ist.
Die Verbesserung von Alon und Kahale beruht im Wesentlichen auf einem Approximationsalgorithmus von Karger, Motwani und Sudan [27] für das Problem Coloring. Die folgenden Definitionen und Fakten entstammen dieser Arbeit, wobei wir die Fakten an dieser Stelle nicht beweisen werden und auf [27] verweisen.

Definition 5.18 (vektorchromatische Zahl, starke vektorchromatische Zahl)
Es sei $G = (V, E)$ ein Graph. Die vektorchromatische Zahl $\chi_v(G)$ des Graphen G ist die kleinste reelle Zahl h, so dass es eine Belegung der Knoten $v \in V$ mit Einheitsvektoren a_v gibt, so dass für jede Kante $\{v, w\} \in E$ gilt

$$a_v^T a_w \leq -\frac{1}{h-1} .$$

Die kleinste reelle Zahl h, so dass es eine Belegung der Knoten $v \in V$ mit Einheitsvektoren a_v gibt, so dass für jede Kante $\{v, w\} \in E$ gilt

$$a_v^T a_w = -\frac{1}{h-1} ,$$

nennen wir starke vektorchromatische Zahl $\chi_{sv}(G)$ des Graphen G.

Bemerkung 5.19 Aufgrund von Definition 5.18 ist sofort klar, dass die vektorchromatische Zahl stets nach oben beschränkt ist durch die starke vektorchromatische Zahl, das heißt, es gilt

$$\chi_v(G) \leq \chi_{sv}(G) . \tag{5.9}$$

Mit Hilfe dieser Definition können wir uns den wesentlichen Ergebnissen in [27] zuwenden.

Fakt 5.20 *Es sei $G = (V, E)$ ein Graph. Gilt für die vektorchromatische Zahl $\chi_v(G) \leq h$, wobei $h \geq 3$ eine feste natürliche Zahl ist, dann kann G von einem randomisierten Polynomialzeitalgorithmus mit $\tilde{O}(n^{1-3/(h+1)})$ Farben gefärbt werden.*

Bemerkung 5.21 Mahajan und Ramesh haben gezeigt [36], dass sich der Algorithmus von Karger, Motwani und Sudan [27] zu einem deterministischen Polynomialzeitalgorithmus derandomisieren lässt.

Eine Verbindung zur Lovász $\vartheta(G)$-Zahl macht der folgende Fakt.

Fakt 5.22 *Es sei $G = (V, E)$ ein Graph. Die starke vektorchromatische Zahl des Graphen G ist gleich der Lovász $\vartheta(\overline{G})$-Zahl des Komplementgraphen $\overline{G}$, also*

$$\chi_{sv}(G) = \vartheta(\overline{G}) .$$

Wir haben damit die Grundlage für den Satz von Alon und Kahale [3] geschaffen.

Satz 5.23 *Es sei $G = (V, E)$ ein Graph. Falls für eine feste natürliche Zahl $k \geq 3$ gilt*

$$\vartheta(G) \geq \frac{n}{k} + m , \tag{5.10}$$

dann kann eine unabhängige Menge der Größe

$$\tilde{\Omega}(m^{3/(k+1)}) \tag{5.11}$$

mit einem randomisierten Polynomialzeitalgorithmus gefunden werden.

Beweis: Wir beginnen den Beweis, indem wir zeigen, dass der Graph $G = (V, E)$ eine unabhängige Menge der Größe $\tilde{\Omega}(m^{3/(k+1)})$ enthält, wenn $\vartheta(G) \geq n/k + m$ gilt. Es sei d ein Einheitsvektor und (b_v) eine orthonormale Repräsentation des Graphen $\overline{G}$, die das Maximum in (5.7) erfüllen, das heißt, wir wählen d und (b_v) so, dass gilt

$$\vartheta(G) = \sum_{i=1}^{n} (d^T b_i)^2 \; . \tag{5.12}$$

Wie wir einen Einheitsvektor d und eine orthonormale Repräsentation des Graphen $\overline{G}$ finden, werden wir weiter unten beschreiben. Wir verwenden die Familie (b_v) von Vektoren um eine große unabhängige Menge im Graphen G zu konstruieren. Ohne Beschränkung der Allgemeinheit seien die Knoten des Graphen G so von 1 bis n nummeriert, dass gilt

$$(d^T b_1)^2 \geq (d^T b_2)^2 \geq \ldots \geq (d^T b_n)^2 \; . \tag{5.13}$$

Wegen (5.10) und (5.12) gilt

$$(d^T b_1)^2 + (d^T b_2)^2 + \ldots + (d^T b_n)^2 \geq \frac{n}{k} + m \; . \tag{5.14}$$

Weiterhin gilt aufgrund der Cauchy-Schwarz Ungleichung für alle Knoten $v \in V$

$$(d^T b_v)^2 \leq 1 \; ,$$

da d und die Vektoren b_v für alle $v \in V$ Einheitsvektoren sind. Damit gilt aber auch

$$(d^T b_m)^2 \geq \frac{1}{k} \; , \tag{5.15}$$

da anderenfalls (5.14) nicht erfüllt ist. Falls nämlich gilt

$$(d^T b_m)^2 < \frac{1}{k} \; ,$$

dann folgt wegen (5.13)

$$(d^T b_j)^2 < \frac{1}{k} \qquad \text{für alle } j \in \{m, \ldots, n\}$$

und wir erhalten

$$\begin{aligned} (d^T b_1)^2 + (d^T b_2)^2 + \ldots + (d^T b_n)^2 &\leq (m-1) \cdot 1 + (n-m+1) \cdot \frac{1}{k} \\ &= \frac{n}{k} + (m-1) \cdot \left(1 - \frac{1}{k}\right) \\ &< \frac{n}{k} + m \; , \end{aligned}$$

was im Widerspruch zu (5.14) steht.

Wir betrachten nun nur noch die Knotenmenge $V' = \{1, \ldots, m\}$. Es sei K der auf V' induzierte Untergraph von G. Die Vektoren $b_1, \ldots, b_m$ bilden offensichtlich eine orthonormale

Repräsentation $(b_{v'})$ mit $v' \in V'$ des Komplementgraphen $\overline{K}$. Wir stellen im Folgenden einen Zusammenhang mit der Lovász $\vartheta(\overline{K})$-Zahl her. Wegen (5.15) gilt

$$\max_{v' \in V'} \frac{1}{(d^T b_{v'})^2} \leq k \; .$$

Damit folgt sofort nach Definition 5.12

$$\vartheta(\overline{K}) = \min_{\{d', (b_{v'})\}} \max_{v' \in V'} \frac{1}{(d'^T b_{v'})^2} \leq \max_{v' \in V'} \frac{1}{(d^T b_{v'})^2} \leq k \; .$$

Nach Fakt 5.22 gilt deshalb für die starke vektorchromatische Zahl des Graphen K

$$\chi_{sv}(K) \leq k$$

und folglich gilt wegen (5.9) für die vektorchromatische Zahl des Graphen K

$$\chi_v(K) \leq k \; .$$

Aufgrund von Fakt 5.20 kann K dann mit $\tilde{O}(m^{1-3/(k+1)})$ Farben von einem randomisierten Polynomialzeitalgorithmus gefärbt werden. Es seien $I_1, \ldots, I_{COLORS}$ die Mengen von Knoten, welche jeweils mit den Farben $1, 2, \ldots, COLORS$ gefärbt sind, wobei wir annehmen, dass gilt

$$|I_1| \geq \ldots \geq |I_{COLORS}| \; .$$

Offensichtlich bildet jede dieser Mengen eine unabhängige Menge im Graphen K und demnach auch im Graphen G. Nach dem Schubfachprinzip gilt für die Kardinalität der Menge I_1

$$|I_1| \geq \tilde{\Omega}(m^{3/(k+1)}) \; .$$

Wir haben damit gezeigt, dass der Graph G eine unabhängige Menge der Größe (5.11) enthält. Wir müssen nun noch zeigen, wie wir in Polynomialzeit einen Einheitsvektor d und eine orthonormale Repräsentation (b_v) des Graphen $\overline{G}$ finden, so dass (5.12) erfüllt ist. Es reicht sogar, wenn gilt

$$\sum_{i=1}^{n} (d^T b_i)^2 \geq \vartheta(G) - 1 \; , \tag{5.16}$$

da die bisherigen Erkenntnisse ihre Gültigkeit behalten, wenn wir m durch $m - 1$ ersetzen, dass heißt insbesondere, dass sich an der Lösung größenordnungsmäßig nichts ändert. Im Folgenden werden wir eine andere Beschreibung der Lovász $\vartheta(G)$-Zahl verwenden, die wir in Fakt 5.13 bereits vorgestellt haben. Für die Lovász $\vartheta(G)$-Zahl des Graphen $G = (V, E)$ mit $n = |V|$ gilt demnach

$$\vartheta(G) = \max_{B} \operatorname{Tr} (B \cdot \mathcal{J}) \; ,$$

wobei $B \in \mathbb{R}^{n \times n}$ eine symmetrische, positiv semidefinite Matrix ist, so dass die folgenden zwei Bedingungen erfüllt sind:

$$\operatorname{Tr} B = 1 \tag{5.17}$$

$$b_{ij} = 0 \qquad \text{für alle Kanten } \{i, j\} \in E. \tag{5.18}$$

Eine Matrix B, welche die Bedingungen (5.17) und (5.18) erfüllt und für die zusätzlich gilt

$$\text{Tr}\,(B \cdot \mathcal{J}) \geq \vartheta(G) - 1\,, \tag{5.19}$$

kann mit der Ellipsoidmethode [16] beziehungsweise mit Innere-Punkte-Verfahren [2, 24] in Polynomialzeit gefunden werden. Da $B \in \mathbb{R}^{n\times n}$ eine symmetrische, positiv semidefinite Matrix ist, gibt es nach Fakt 5.7 eine Diagonalmatrix $D \in \mathbb{R}^{n\times n}$ mit den Einträgen $d_{ii} \geq 0$, $i = 1, \ldots, n$, und eine orthogonale Matrix $Q \in \mathbb{R}^{n\times n}$, so dass gilt

$$B = Q \cdot D \cdot Q^T\,.$$

Wir bilden nun eine Matrix $C \in \mathbb{R}^{n\times n}$

$$C = Q \cdot \tilde{D}\,,$$

wobei $\tilde{D} \in \mathbb{R}^{n\times n}$ eine Diagonalmatrix mit folgenden Einträgen ist:

$$\tilde{d}_{ii} = \sqrt{d_{ii}} \qquad \text{für alle } i \in \{1, \ldots, n\}\,.$$

Offenbar gilt dann auch

$$B = C \cdot C^T\,.$$

Die Matrix C kann mit Hilfe der Cholesky-Zerlegung [14, Kapitel 4.2.8] in Polynomialzeit gefunden werden kann.

Algorithmus 5.1 Cholesky-Zerlegung(B)

```
1: for k ← 1 to n do
2:     if B(k,k) > 0 then
3:         C(k,k) ← √B(k,k)
4:         for i ← k+1 to n do
5:             C(i,k) ← B(i,k)/C(k,k)
6:         for j ← k+1 to n do
7:             for i ← j to n do
8:                 B(i,j) ← B(i,j) − C(i,k)·C(j,k)
9: return C
```

Insbesondere existieren also Vektoren c_i und c_j mit $b_{ij} = c_i^T c_j$ für alle $i, j \in \{1, \ldots, n\}$. Wir konstruieren nun daraus eine orthonormale Repräsentation (b_v) mit $v \in V$. Es sei $I = \{i \mid c_i \neq 0\}$ und $\overline{I} = \{i \mid c_i = 0\}$. Wir setzen für alle $i \in I$

$$b_i = \frac{c_i}{\|c_i\|}\,. \tag{5.20}$$

Für die restlichen $(n - |I|)$ Vektoren b_j mit $j \in \overline{I}$ konstruieren wir $(n - |I|)$ Einheitsvektoren, welche paarweise orthogonal zueinander sind und für die zusätzlich gilt, dass jeder Vektor

b_j mit $j \in \overline{I}$ zu jedem Vektor b_i mit $i \in I$ orthogonal ist[1]. Also gilt für $v \in \overline{I}$ und $w \in \overline{I}$ beziehungsweise für $v \in I$ und $w \in \overline{I}$ oder $v \in \overline{I}$ und $w \in I$

$$b_v^T b_w = 0 \; . \tag{5.21}$$

Weiterhin sei

$$d = \frac{\sum_{i=1}^n c_i}{\| \sum_{i=1}^n c_i \|} \; . \tag{5.22}$$

Es muss nun noch gezeigt werden, dass die Vektoren $(b_v) = (b_1, \ldots, b_n)$ eine orthonormale Repräsentation des Graphen $\overline{G} = (V, \overline{E})$ bilden, das heißt für alle $\{v, w\} \notin E$ gilt $b_v^T b_w = 0$. Seien also v und w Knoten, so dass die Kante $\{v, w\}$ nicht im Graphen $\overline{G}$ enthalten ist. Die dazugehörigen Vektoren seien b_v und b_w. Wir müssen also zeigen, dass gilt

$$b_v^T b_w = 0 \; . \tag{5.23}$$

Ohne Beschränkung der Allgemeinheit seien beide Vektoren b_v und b_w nach Vorschrift (5.20) gebildet worden, das heißt, es gilt $v \in I$ und $w \in I$, da anderenfalls b_v und b_w wegen (5.21) orthogonal zueinander sind. Damit gibt es Vektoren $c_v \neq 0$ und $c_w \neq 0$, so dass gilt

$$b_v = \frac{c_v}{\|c_v\|} \text{ und } b_w = \frac{c_w}{\|c_w\|} \; .$$

Da die Kante $\{v, w\}$ nicht im Graphen $\overline{G}$ enthalten ist, muss sie im Komplementgraphen enthalten sein, also $\{v, w\} \in E$. Damit ist der entsprechende Eintrag b_{vw} in der Matrix B wegen (5.18) gleich Null, insbesondere gilt also $b_{vw} = c_v^T c_w = 0$ und wir erhalten

$$b_v^T b_w = \frac{c_v^T c_w}{\|c_v\| \cdot \|c_w\|} = 0 \; .$$

Damit ist gezeigt, dass die Vektoren $(b_v) = (b_1, \ldots, b_n)$ eine orthonormale Repräsentation bilden. Es fehlt noch der Beweis, dass damit (5.16) erfüllt ist. Zunächst sei bemerkt, dass wegen (5.17) gilt

$$\sum_{i=1}^n c_i^2 = 1 \; .$$

Außerdem gilt aufgrund von (5.19)

$$\left(\sum_{i=1}^n c_i \right)^2 \geq \vartheta(G) - 1 \; .$$

Damit erhalten wir durch Anwenden der Cauchy-Schwarz Ungleichung (5.1)

$$\sum_{i=1}^n (d^T b_i)^2 \quad = \quad \left(\sum_{i=1}^n c_i^2 \right) \cdot \left(\sum_{i=1}^n (d^T b_i)^2 \right)$$

[1] Wir beweisen kurz die Existenz dieser Vektoren. Es sei U der von den Vektoren c_i mit $i \in I$ aufgespannte Unterraum und V dessen orthogonales Komplement. Es gilt $\dim V = n - \dim U$ sowie $\dim U \leq k$ und folglich auch $\dim V \geq n - k$. Wir wählen nun $n - k$ beliebige Elemente einer Orthonormalbasis in V.

$$
\begin{aligned}
&\geq \left(\sum_{i=1}^{n} \|c_i\| \cdot (d^T b_i)\right)^2 \\
&= \left(\sum_{i=1}^{n} d^T c_i\right)^2 && \text{(da } b_i = c_i/\|c_i\|\text{)} \\
&= \left(d^T \cdot \left(\sum_{i=1}^{n} c_i\right)\right)^2 && \text{(aufgrund von (5.22))} \\
&= \left(\sum_{i=1}^{n} c_i\right)^2 \\
&\geq \vartheta(G) - 1 \; .
\end{aligned}
$$

Wir können daher nach obigen Überlegungen den Algorithmus von Karger, Motwani und Sudan [27] auf die ersten m Knoten nach Ordnung (5.13) anwenden und geben die Knoten der größten Farbklasse aus. □

Bemerkung 5.24 Die Verbesserung von Alon und Kahale [3] lässt sich offensichtlich ebenfalls derandomisieren, da wir dafür nur den Algorithmus von Karger, Motwani und Sudan [27] derandomisieren müssen, was aufgrund von [36] möglich ist.

Der Algorithmus von Alon und Kahale lässt sich im Pseudocode wie folgt beschreiben:

Algorithmus 5.2 SDP-Independent-Set-Approximation($G = (V, E)$)

```
 1: B ← Theta(G)
 2: C ← Cholesky-Zerlegung(B)
 3: c_1, ..., c_n ← C(1,:), ..., C(n,:)
 4: for i ← 1 to n do
 5:     if c_i ≠ 0 then
 6:         b_i ← c_i/‖c_i‖
 7:     else
 8:         Ī ← Ī ∪ i
 9: b_Ī ← Null(C(V \ Ī))
10: s ← Σ_{i=1}^{n} c_i
11: d ← s/‖s‖
12: v_1, ..., v_n ← Sort((d^T b_1)^2 ≥ (d^T b_2)^2 ≥ ... ≥ (d^T b_n)^2)
13: I_1, ..., I_COLORS ← KMS-Coloring(G[v_1, ..., v_m])
14: return max{I_1, ..., I_COLORS}
```

In *Zeile 1* wird zunächst eine Matrix $B \in \mathbb{R}^{n \times n}$ berechnet, so dass (5.5) gilt. Anschließend wird die Cholesky-Zerlegung der Matrix B bestimmt und die einzelnen Zeilen dieser Matrix C in den Vektoren $c_1, \ldots, c_n$ gespeichert. In den *Zeilen 4 bis 9* wird eine orthonormale Repräsentation (b_v) des Graphen $\overline{G}$ berechnet. Dabei wird zunächst für alle $c_i \neq 0$ der Wert von b_i

entsprechend (5.20) gesetzt. Alle restlichen Vektoren b_j, für die $c_j = 0$ gilt, werden in *Zeile 9* gesetzt. Dabei berechnet „Null“ paarweise orthogonale Einheitsvektoren, die wiederum orthogonal zu allen bisherigen Vektoren b_i verlaufen[2]. In *Zeile 12* wird eine Funktion aufgerufen, die die Knoten entsprechend (5.13) sortiert. Der auf die ersten m Knoten dieser Ordnung induzierte Untergraph von G wird durch den Aufruf „KMS-Coloring($G[v_1, \ldots, v_m]$)“ in *Zeile 13* mit dem Färbungsalgorithmus von Karger, Motwani und Sudan gefärbt und abschließend in *Zeile 14* die größte gefundene Farbenklasse zurückgegeben.

[2]Die gleichnamige Matlab-Funktion `null` berechnet diese Vektoren.

6 ONLINE INDEPENDENT SET

In diesem Kapitel werden wir uns mit der Onlinesituation des Problems Independent Set beschäftigen. Dabei handelt es sich um eine besondere Art von Approximationsalgorithmen, wie wir im Folgenden sehen werden. Bei den bisherigen Betrachtungen in dieser Diplomarbeit war die Eingabe für die Approximationsalgorithmen immer vollständig vorgegeben, das heißt die betrachteten Algorithmen hatten stets die ganze Eingabe, in unserem Fall also einen Graphen $G = (V, E)$, vorliegen, mit der Aufgabe, in diesem Graphen G eine unabhängige Menge maximaler Kardinalität zu approximieren.

Bei unseren Betrachtungen wandeln wir dieses dahingehend ab, dass die Eingabe nur noch Schritt für Schritt für den Algorithmus „sichtbar“ wird. In unserem Fall, das heißt bei dem Problem Independent Set, wird in jedem Schritt dem Onlinealgorithmus ein Knoten v gegeben. Zusätzlich zu diesem Knoten erhält der Algorithmus auch Informationen über Kanten, die zwischen dem Knoten v und bereits vorhandenen, in früheren Schritten gegebenen Knoten verlaufen. Bei Graphen $G = (V, E)$ auf $n = |V|$ Knoten bezeichnen wir diese Vorgehensweise als Onlinesequenz der Länge n.

Definition 6.1 (Onlinesequenz der Länge n, Runde) *Es sei $G = (V, E)$ ein Graph auf $n = |V|$ Knoten. Eine Onlinesequenz der Länge n für einen Onlinealgorithmus A ist gegeben durch eine Permutation π der Knotenmenge V und eine Kantenmenge*

$$E(\pi(i)) = E \cap \{\{\pi(i), \pi(j)\} \mid j < i\}\} .$$

Der Onlinealgorithmus A erhält dann im Schritt i den Knoten $\pi(i)$ sowie die zwischen dem Knoten $\pi(i)$ und der Menge $E(\pi(i))$ der Knoten $\{\pi(1), \ldots, \pi(i-1)\}$ verlaufenden Kanten als Eingabe. Ein einzelner Schritt wird oftmals als Runde bezeichnet, das heißt insbesondere, dass eine Onlinesequenz aus n Runden besteht.

Bemerkung 6.2 Zum besseren Verständnis gehen wir davon aus, dass die Knoten bereits so geordnet sind, wie sie einem Onlinealgorithmus gegeben werden, das bedeutet in Runde i erhält ein Onlinealgorithmus den Knoten v_i und die Kantenmenge

$$E(v_i) = E \cap \{\{v_i, v_j\} \mid \text{für alle } j < i\} .$$

Im Folgenden werden wir uns weniger mit speziellen Onlinealgorithmen, sondern vielmehr mit deren Fähigkeiten auseinandersetzen. Dieses bedeutet insbesondere, dass wir obere und untere Schranken bezüglich der Onlineapproximierbarkeit analysieren.

Ein Onlinealgorithmus für das Problem INDEPENDENT SET beginnt, ausgehend von der leeren Menge $I := \emptyset$, eine unabhängige Menge I zu konstruieren, indem er in jeder Runde i der Onlinesequenz entscheidet, ob der Knoten v_i zur unabhängigen Menge I hinzugefügt wird oder nicht. Natürlich kann der Onlinealgorithmus dieses nur tun, wenn die Menge I durch das Hinzufügen von v_i immer noch eine unabhängige Menge ist. Ziel ist es wiederum, eine möglichst große, nahe am Optimum liegende, unabhängige Menge I zu erhalten. Analog zur Güte bei Approximationsalgorithmen definieren wir bei Onlinealgorithmen auch die Qualität einer gelieferten Lösung.

Definition 6.3 (Kompetivität R_A eines Onlinealgorithmus A) *Es sei $\mathcal{H}_n$ die Menge aller Onlinesequenzen der Länge n aller Graphen $G = (V, E)$ auf $n = |V|$ Knoten und $A(H)$ der Wert der Lösung des Onlinealgorithmus A bei Eingabe einer Onlinesequenz $H \in \mathcal{H}_n$. Dann ist die Kompetivität des Onlinealgorithmus A definiert als*

$$R_A(n) = \max_{H \in \mathcal{H}_n} \frac{\alpha(H)}{A(H)} ,$$

wobei $\alpha(H)$ die Unabhängigkeitszahl des Graphen $G = (V, E)$ auf $n = |V|$ Knoten ist, welcher durch die Onlinesequenz H gegeben wird.

Um die Kompetivität eines Onlinealgorithmus zu analysieren, führt man ein sogenanntes Gegnermodell ein. Dieses bedeutet, dass wir die Existenz eines Gegners voraussetzen, welcher unseren Onlinealgorithmus sowie die jeweilige Entscheidung in jeder Runde des Algorithmus kennt und die restlichen Runden der Onlinesequenz so modifizieren kann, dass unser Onlinealgorithmus eine möglichst schlechte Lösung approximiert. Einen Gegner mit diesen Fähigkeiten bezeichnen wir als anpassungsfähigen beziehungsweise adaptiven Gegner.

Definition 6.4 (adaptiver Gegner) *Ein adaptiver Gegner eines Onlinealgorithmus hat die Fähigkeit, vor jeder Runde i die restliche Onlinesequenz zu modifizieren, das heißt, er kann die Reihenfolge der Knoten $\pi(i), \ldots, \pi(n)$ beliebig verändern und zusätzliche Kanten zwischen den Knoten $\pi(i), \ldots, \pi(n)$ und allen anderen Knoten $v \in V$ einfügen beziehungsweise löschen.*

Mit Hilfe dieses Gegnermodells können wir die Kompetivität eines Onlinealgorithmus A bestimmen, indem der adaptive Gegner in jeder Runde gerade die Onlinesequenz so wählt, dass die Entscheidungen des Onlinealgorithmus am schlechtesten werden. Es ist sofort klar, dass dieser adaptive Gegner sehr mächtig ist, insbesondere können durch die Fähigkeiten eines anpassungsfähigen Widersachers sehr schlechte Ergebnisse für die Onlinesituation des Problems INDEPENDENT SET entstehen, wie wir im folgenden Satz sehen werden.

Satz 6.5 *Es sei A ein beliebiger Onlinealgorithmus für die Onlinesituation des Problems* INDEPENDENT SET*, welcher als Eingabe eine Onlinesequenz der Länge n erhält. Dann hat A gegen einen adaptiven Gegner eine Kompetivität von*

$$R_A(n) \geq n - 1 .$$

Beweis: Es sei eine beliebige Onlinesequenz der Länge n gegeben, das heißt, diese Onlinesequenz definiert einen Graphen $G = (V, E)$ auf $n = |V|$ Knoten. In jedem Schritt erhält der Onlinealgorithmus A einen Knoten v des Graphen G sowie die, zu vorher gegebenen Knoten, inzidenten Kanten. Nach Definition 6.4 kann der adaptive Gegner in jedem Schritt die restliche Onlinesequenz modifizieren.

Die Onlinesequenz sieht nun folgendermaßen aus. Zunächst erhält der Onlinealgorithmus A in jeder Runde nur einzelne, isolierte Knoten, das heißt Knoten, die mit keinem der in vorherigen Runden gegeben Knoten durch eine Kante verbunden sind. Angenommen in Runde i, $i \in \{1, \ldots, n\}$, entscheidet sich der Onlinealgorithmus A zum ersten Mal den Knoten v_i in die unabhängige Menge I aufzunehmen. Es seien $v_{i+1}, \ldots, v_n$ die restlichen Knoten der Onlinesequenz. Der adaptive Gegner ändert nun die Onlinesequenz wie folgt. Für jeden der Knoten v_j, $j \in \{i+1, \ldots, n\}$, wird zusätzlich noch die Kante $\{v_i, v_j\}$ in jeder weiteren Runde gegeben. Die Reihenfolge der Knoten muss nicht geändert werden. Diese Modifikation hat zur Folge, dass der Onlinealgorithmus A keinen weiteren Knoten zur unabhängigen Menge I hinzufügen kann, da durch das Hinzufügen eines Knotens v_j, $j \in \{i+1, \ldots, n\}$, zu I die Unabhängigkeit dieser Menge zerstört wird.

Wir analysieren nun die Kompetivität des Onlinealgorithmus A. Der durch die Onlinesequenz definierte Graph $G = (V, E)$ mit $n = |V|$ wird beschrieben durch

$$\begin{aligned} V &:= \{v_1, \ldots, v_n\} \\ E &:= \{\{v_i, v_j\} \mid \text{ für alle } j \in \{i+1, \ldots, n\}\} , \end{aligned}$$

wobei v_i der erste und einzige Knoten der Onlinesequenz ist, welcher zur unabhängigen Menge I hinzugefügt wurde:

$$I = \{v_i\} .$$

Die maximale unabhängige Menge des Graphen G ist offensichtlich

$$I_{max} = \{v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n\} = V \setminus \{v_i\} ,$$

da jede Kante mit dem Knoten v_i inzidiert und es deshalb keine Kante gibt, die nur mit Knoten aus I_{max} inzidiert. Damit ist die Unabhängigkeitszahl $\alpha(G)$ des Graphen G

$$\alpha(G) = n - 1 .$$

Wir erhalten somit als Kompetivität des Onlinealgorithmus A

$$R_A(n) \geq \frac{n-1}{1} = n - 1 .$$

□

Wir haben damit gesehen, dass jeder Onlinealgorithmus für das Problem Independent Set unter Verwendung des obigen Modells eine sehr schlechte Kompetivität hat, da trivialerweise

$R_A(n) \leq n$ gilt. Die Ursache dieser schlechten Kompetivität liegt darin, dass wir die Entscheidungen des Onlinealgorithmus nicht rückgängig machen können, also etwa den in Schritt i gewählten Knoten wieder entfernen und dafür andere Knoten zur unabhängigen Menge I hinzufügen. Andererseits wäre es günstiger, wenn wir statt einer unabhängigen Menge I mehrere unabhängige Mengen $I_1, \ldots, I_k$ verwenden könnten und dann die größte dieser k Mengen ausgeben. Dieses Modell, welches man wegen der mehrfachen Lösungsmengen auch Multisolution Modell nennt, werden wir nun kennenlernen.

6.1 Das Multisolution Modell

Im bisherigen Modell der Onlinesituation des Problems INDEPENDENT SET konnte ein Onlinealgorithmus nur eine Lösungsmenge konstruieren. Wir haben gesehen, dass ein solcher Algorithmus schlechte Lösungen liefern kann. In diesem Abschnitt wollen wir dieses Modell dahingehend erweitern, dass ein Onlinealgorithmus mehrere Lösungsmengen zusammenstellen kann. Zur besseren Veranschaulichung werden wir diese Lösungsmengen auch Behälter nennen.

Definition 6.6 (Behälter) *Eine Lösungsmenge bezeichnen wir als Behälter.*

Wir können uns das Multisolution Modell so vorstellen, dass wir eine endliche Menge von Behältern betrachten. In Runde i kann ein Onlinealgorithmus nun den Knoten v_i zu mehreren dieser Behälter hinzufügen. Wie oft ein Knoten v_i zu verschiedenen Behältern hinzugefügt werden darf, ist durch einen Parameter $r(n)$ spezifiziert. Abbildung 6.1 stellt das Hinzufügen des dritten Knotens zu $r(n) = 3$ vielen verschiedenen Behältern graphisch dar.

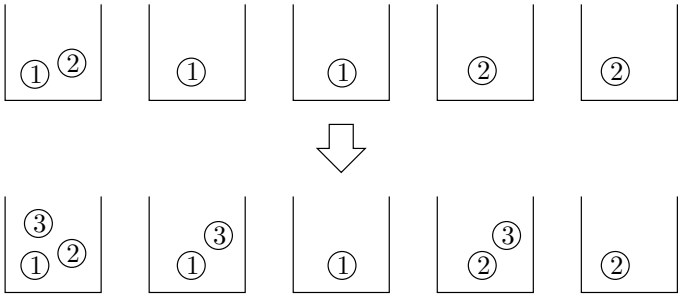

Abbildung 6.1: Hinzufügen eines Knotens im Multisolution Modell

Definition 6.7 (Parameter $r(n)$) *Es sei A ein Onlinealgorithmus für das Multisolution Modell. Dann wird durch den Parameter $r(n)$ festgelegt, zu wie vielen verschiedenen Behältern jeder Knoten v_i in Runde i höchstens hinzugefügt werden darf.*

Offenbar können wir unter der Bedingung $r(n) = 2^{n-1}$ alle möglichen Teilmengen der Knotenmenge V des durch die Onlinesequenz der Länge n definierten Graphen $G = (V, E)$ erzeugen

und das Problem wird trivial. Wir sind also insbesondere an Lösungen interessiert, so dass der Parameter $r(n)$ polynomiell in n beschränkt ist.

Definition 6.8 (Behältermenge $B(v)$ eines Knotens v) *Die Behältermenge $B(v)$ eines Knotens v ist die Menge aller Behälter, die den Knoten v enthalten.*

Für die Kardinalität der Behältermenge $B(v)$ gilt $|B(v)| \leq r(n)$ aufgrund von Definition 6.7.

6.1.1 Eine obere Schranke der Kompetivität

Satz 6.9 *Es sei A ein beliebiger Onlinealgorithmus für das Multisolution Modell der Onlinesituation des Problems* Independent Set*, welcher als Eingabe eine Onlinesequenz der Länge n erhält. Dann hat A gegen einen adaptiven Gegner eine Kompetivität von*

$$R_A(n) = O\left(\frac{n}{\log r(n)}\right) .$$

Beweis: Es sei $t = \lceil \log r(n) - 1 \rceil$. Wir unterteilen die gegebene Onlinesequenz der Länge n in einzelne Blöcke der Länge t, das heißt, wir fassen immer t viele Knoten zu einem Block zusammen. Im Folgenden betrachten wir die Vorgehensweise für einen beliebigen Block von t aufeinanderfolgenden Knoten.

Es gibt insgesamt $2^t - 1$ viele nichtleere Teilmengen I_j, $j = 1, \ldots, 2^t - 1$, von Knoten eines Blockes, der aus t Knoten besteht. Jede dieser nichtleeren Teilmengen I_j kann als Bitstring b_j der Länge t interpretiert werden, wobei an Position $b_j(i)$ genau dann eine Eins steht, wenn der Knoten v_i in der Teilmenge I_j enthalten ist. Anderenfalls ist der Eintrag $b_j(i)$ gleich Null. Für jeden Block der Länge t legen wir zunächst $2^t - 1$ verschiedene Behälter an. Jedem dieser $2^t - 1$ vielen Behälter weisen wir nun genau einen der eben beschriebenen Bitstrings b_j, $j = 1, \ldots, 2^t - 1$ zu.

Der Onlinealgorithmus A geht nun wie folgt vor. Es sei v_i der i'te Knoten eines Blockes der Länge t für den A eine Entscheidung treffen muss. Der Knoten v_i wird nun in alle die Behälter I_j mit $j = 1, \ldots, 2^t - 1$ hinzugefügt, für die gilt

$$b_j(i) = 1 .$$

Das bedeutet, dass der Knoten v_i in genau 2^{t-1} viele Behälter hinzugefügt wird, da v_i in genau der Hälfte aller (inklusive der leeren) Teilmengen der t Knoten enthalten ist. Für die Anzahl der Zuordnungen von Knoten v_i gilt demnach

$$\begin{aligned} 2^{t-1} &= 2^{\lceil \log r(n) - 1 \rceil - 1} \\ &\leq r(n) . \end{aligned}$$

Jede mögliche unabhängige Menge eines jeden Blocks der Länge t ist in einem der $2^t - 1$ vielen Behältern repräsentiert. Der Onlinealgorithmus A muss nun zunächst jeden Behälter

prüfen, ob die darin enthaltenen Knoten eine unabhängige Menge bilden. Die größte gefundene unabhängige Menge I_A gibt der Onlinealgorithmus A am Ende seiner Suche aus. Diese Menge I_A besteht aus Knoten eines Blockes der Länge t. Alle anderen Blöcke enthalten nur unabhängige Mengen, deren Kardinalität durch $|I_A|$ nach oben beschränkt ist.

Die Anzahl Blöcke der Länge t, welche der Onlinealgorithmus A betrachtet ist $\lceil n/t \rceil$. Es sei $\alpha(G)$ die Unabhängigkeitszahl des Graphen $G = (V, E)$, welcher durch die Onlinesequenz der Länge n definiert wird. Nach dem Schubfachprinzip ist die Kardinalität einer größten unabhängigen Menge I_{max} des Graphen G beschränkt durch

$$\alpha(G) = |I_{max}| \leq \left\lceil \frac{n}{t} \right\rceil \cdot |I_A| \, ,$$

da I_A anderenfalls nicht eine größte unabhängige Menge eines Blockes der Länge t ist. Damit gilt für die Kompetivität des Onlinealgorithmus A

$$\begin{aligned} R_A(n) &= \frac{\lceil n/t \rceil \cdot |I_A|}{|I_A|} = \left\lceil \frac{n}{t} \right\rceil \\ &= O\left(\frac{n}{\log r(n)} \right) \, . \end{aligned}$$

□

6.1.2 Untere Schranken der Kompetivität

Zunächst werden wir uns mit der allgemeinen Vorgehensweise für Beweise unterer Schranken für die Onlinesituation des Problems Independent Set vertraut machen, welche auch schon in [19] betrachtet wurden. Jeder Knoten erhält vom adaptiven Widersacher einen Status. Entweder wird ein Knoten als *gut* oder als *schlecht* bezeichnet, welches nun genauer spezifiziert wird.

Definition 6.10 (guter Knoten, schlechter Knoten) *Ein Knoten v_i, der in Runde i einem Onlinealgorithmus für das Problem* Independent Set *gegeben wird, erhält zunächst den Status gut und wird als guter Knoten bezeichnet. Der Knoten v_i besitzt solange den Status gut, bis er von einem adaptiven Gegner den Status schlecht erhält. Ab diesem Zeitpunkt bezeichnen wir v_i als schlechten Knoten. Ein schlechter Knoten wird zu keinem Zeitpunkt wieder zu einem guten Knoten. Die Kantenmenge $E(v_i)$, welche in Runde i zusätzlich zum Knoten v_i gegeben wird, ist einzig allein abhängig vom Status der bisher gegebenen Knoten. Der Knoten v_i ist mit allen Knoten v_j, $j < i$, die schlechte Knoten sind, verbunden. Verbindungen zwischen zwei guten Knoten existieren nicht, das heißt, die Kantenmenge $E(v_i)$ erfüllt*

$$E(v_i) = \{\{v_j, v_i\} \mid j < i \text{ und } v_j \text{ schlecht}\} \, .$$

Da jeder schlechte Knoten mit allen nachfolgenden Knoten verbunden ist, können keine weiteren Knoten zu einer unabhängigen Menge I beziehungsweise zu einem Behälter I hinzugefügt werden, sobald der erste schlechte Knoten zu I hinzugefügt wird. Zusätzlich sei bemerkt, dass

die Kardinalität einer unabhängigen Menge durch die Anzahl guter Knoten in einem Behälter beschrieben werden kann[1].

Mit Hilfe der Definition 6.10 können wir den Beweis von Satz 6.5 auch so veranschaulichen, indem wir uns vorstellen, dass der erste gewählte Knoten sofort ein schlechter Knoten wird. Die gefundene unabhängige Menge besteht dann gerade nur aus diesem einen Knoten, obwohl die größte unabhängige Menge aus allen anderen Knoten bestehen kann.

Wir werden zunächst eine untere Schranke für das Multisolution Modell für den Fall, dass $r(n) < n$ gilt, angeben.

Satz 6.11 *Es sei A ein beliebiger Onlinealgorithmus für das Multisolution Modell der Onlinesituation des Problems* Independent Set*, welcher als Eingabe eine Onlinesequenz der Länge n erhält, die einen Graphen auf n Knoten definiert. Dann hat A gegen einen adaptiven Gegner eine Kompetivität von*

$$R_A(n) \geq \frac{n}{2 \cdot (r(n)+1)} \qquad \text{für } r(n) < n \, .$$

Beweis: Der adaptive Gegner des Onlinealgorithmus A bestimmt in jeder Runde i der Onlinesequenz der Länge n den Status des jeweils gegebenen Knoten v_i wie folgt. Falls A den Knoten v_i in einen Behälter legt, der bereits mindestens $r(n)$ Knoten enthält, dann ändert der adaptive Gegner den Status und v_i wird ein schlechter Knoten. Anderenfalls, das heißt die Berechnungen des Algorithmus A ergeben, dass der Knoten v_i in einen Behälter mit weniger als $r(n)$ Knoten gelegt wird, bleibt v_i ein guter Knoten.

Jeder Behälter, der eine unabhängige Menge bildet, enthält nun höchstens $r(n)+1$ Knoten, da der $(r(n)+1)$'te und jeder weitere Knoten eines jeden Behälters nach obiger Darstellung ein schlechter Knoten ist und demzufolge keine weiteren Knoten aufgenommen werden können. Wir erhalten folglich als obere Schranke für die Größe einer vom Onlinealgorithmus A gelieferten Lösung I_A

$$|I_A| \leq r(n) + 1 \, .$$

Wir müssen nun noch eine untere Schranke für eine optimale Lösung I_{max} berechnen. Wie bereits oben festgestellt, entspricht die Anzahl guter Knoten (zuzüglich eines schlechten Knotens) der Unabhängigkeitszahl $\alpha(G)$ des durch die Onlinesequenz der Länge n gegebenen Graphen $G = (V, E)$. Ein schlechter Knoten tritt erst dann auf, wenn dieser zu einem Behälter mit $r(n)$ guten Knoten hinzugefügt wird. Da es von dem schlechten Knoten höchstens $r(n)$ Kopien gibt, ist die Anzahl guter Knoten in allen Behältern mindestens so groß wie die Anzahl schlechter Knoten in allen Behältern. Deshalb gibt es mindestens $n/2$ viele gute Knoten, welche eine unabhängige Menge in G bilden und demzufolge gilt

$$\alpha(G) = |I_{max}| \geq \frac{n}{2} \, .$$

[1] Genauer gesagt gilt, dass der erste schlechte Knoten ebenfalls noch dazu gehört, da dieser schlechte Knoten nicht mit den vorher gegebenen (guten) Knoten durch eine Kante verbunden ist.

Wir haben nun eine obere Schranke für die Lösung I_A des Onlinealgorithmus A sowie eine untere Schranke für die Optimallösung und folglich gilt für die Kompetivität

$$R_A(n) \geq \frac{n}{2 \cdot (r(n)+1)} \; . \qquad \square$$

Damit haben wir eine erste untere Schranke der Kompetivität des Multisolution Modell der Onlinesituation des Problems INDEPENDENT SET kennengelernt. Falls für den Parameter $r(n)$ gilt, dass dieser wesentlich kleiner als n gewählt wird, ist diese Schranke bereits sehr gut. Anderenfalls, wenn der Parameter $r(n)$ ein Polynom in Abhängigkeit von n ist, haben wir eine wenig aussagekräftige untere Schranke vorliegen, da diese dann trivialerweise aufgrund $R_A(n) \geq 1$ erfüllt ist. Für diesen Fall folgt nun eine bessere Aussage.

Satz 6.12 *Es sei A ein beliebiger Onlinealgorithmus für das Multisolution Modell der Onlinesituation des Problems* INDEPENDENT SET*, welcher als Eingabe eine Onlinesequenz der Länge n erhält. Dann hat A gegen einen adaptiven Gegner eine Kompetivität von*

$$R_A(n) \geq \frac{n}{2 \cdot \log(n \cdot r(n))} \; .$$

Bevor wir uns dem Beweis zuwenden, benötigen wir noch einige Definitionen.

Definition 6.13 (guter Behälter, schlechter Behälter) *Ein Behälter b wird als gut bezeichnet, wenn er keinen schlechten Knoten enthält. Anderenfalls ist b ein schlechter Behälter.*

Definition 6.14 (Kosten eines Behälters, Gesamtkosten einer Lösung) *Es sei b ein Behälter und $|b|$ die Anzahl Knoten in b zu einem bestimmten Zeitpunkt. Dann sind die Kosten $cost_b$ des Behälters b zu diesem bestimmten Zeitpunkt definiert als*

$$cost_b = 2^{|b|-1} \; . \tag{6.1}$$

Die Gesamtkosten einer Lösung sind definiert als die Summe der Kosten aller guten Behälter.

Definition 6.15 (Kosten eines Knotens, effektive Kosten eines Knotens)
Die Kosten $cost(v)$ eines Knotens v sind definiert als die Summe der Kosten der Behälter, die den Knoten v enthalten, also die Kosten der Elemente der Behältermenge $B(v)$ des Knotens v

$$cost(v) = \sum_{b \in B(v)} cost_b = \sum_{b \in B(v)} 2^{|b|-1} \; .$$

Die effektiven Kosten $cost_e(v)$ eines Knotens v sind die Summe der Kosten der Behälter, die den Knoten v und mindestens einen weiteren Knoten enthalten

$$cost_e(v) = \sum_{\substack{b \in B(v) \\ |b| \geq 2}} cost_b = \sum_{\substack{b \in B(v) \\ |b| \geq 2}} 2^{|b|-1} \; .$$

Damit haben wir nun die Vorbereitungen für den Beweis von Satz 6.12 getroffen.

Beweis (Satz 6.12): Wie bereits im Beweis von Satz 6.11 besitzen auch hier die Knoten einen Status, das heißt, entweder bezeichnen wir einen Knoten als *gut* oder als *schlecht*.

Unsere Onlinesituation umfasst nun $n/2$ viele Runden. Dieses bedeutet, dass in jeder Runde zwei Knoten v_1 und v_2 sowie die entsprechenden Kanten der Onlinesequenz der Länge n dem Onlinealgorithmus A gegeben werden. Bisher hatten wir nur Onlinesituationen betrachtet, die dadurch gekennzeichnet waren, dass der Onlinealgorithmus A in jeder Runde nur einen einzigen Knoten als Eingabe erhält. Das Abweichen von diesem Modell in diesem Beweis soll nur der besseren Vorstellung dienen. Man könnte sich diese Situation auch wie folgt vorstellen. Der Onlinealgorithmus A erhält in jeder Runde einen Knoten. Nach jeweils zwei Runden kann der adaptive Gegner dann den Status der beiden zuvor gegebenen Knoten ändern. Dieses Vorgehen verringert die untere Schranke, da der Onlinealgorithmus vor der Entscheidung bezüglich des zweiten Knotens keine Information über Statusänderungen des ersten Knotens erfährt.

Der adaptive Widersacher ändert den Status der Knoten v_1 und v_2 nun in Abhängigkeit der vom Onlinealgorithmus A getroffenen Entscheidung so, dass einer der beiden Knoten den Status schlecht erhält, während der andere Knoten weiterhin ein guter Knoten bleibt. Der neue Status richtet sich nach den effektiven Kosten der beiden Knoten v_1 und v_2. Gilt $cost_e(v_1) > cost_e(v_2)$, dann wird v_1 ein schlechter Knoten und v_2 bleibt ein guter Knoten, anderenfalls wird v_2 ein schlechter Knoten und v_1 bleibt ein guter Knoten.

Die Anzahl guter Knoten ist aufgrund dieses Vorgehens genau so groß wie die Anzahl schlechter Knoten, wenn wir ohne Beschränkung der Allgemeinheit voraussetzen, dass die Anzahl der Runden gerade ist, das bedeutet, es gilt n ist gerade. Da alle guten Knoten nach Konstruktion nicht paarweise durch Kanten verbunden sind, bilden diese eine unabhängige Menge und wir erhalten als untere Schranke für die Unabhängigkeitszahl $\alpha(G)$ des durch die Onlinesequenz der Länge n definierten Graphen $G = (V, E)$

$$\alpha(G) \geq \frac{n}{2} .$$

Wir können annehmen, dass die beiden Knoten v_1 und v_2 nur zu guten Behältern, das heißt Behältern, die nur gute Knoten enthalten, hinzugefügt werden, da anderenfalls der entsprechende schlechte Behälter nach dem Hinzufügen eines Knotens keine unabhängige Menge mehr bildet.

Im Folgenden gehen wir ohne Einschränkung davon aus, dass der Knoten v_1 den Status *schlecht* erhält, das bedeutet, es gilt $cost_e(v_1) > cost_e(v_2)$. Jeder gute Behälter, der den Knoten v_1 aufnimmt, wird dadurch ein schlechter Behälter. Gute Behälter, die beide Knoten v_1 und v_2 aufnehmen, werden natürlich ebenfalls zu schlechten Behältern.

Wir betrachten nun, wie sich das Hinzufügen der Knoten v_1 und v_2 zu verschiedenen Behältern auf die Kosten der jeweiligen Behälter auswirkt. Behälter b, die nur den Knoten v_2 enthalten und folglich neu angelegt wurden, haben die Kosten 1. Alle anderen guten Behälter, die nur den Knoten v_2 aufgenommen haben, bleiben gute Behälter und verdoppeln aufgrund von (6.1) ihre Kosten, da sich die Anzahl der Elemente durch das Hinzufügen von v_2 um eins erhöht.

Durch eine Veränderung der Kosten der Behälter verändern sich natürlich auch die Gesamtkosten. Wir berechnen nun eine obere Schranke für die Gesamtkosten, wodurch wir eine obere Schranke für die Anzahl Knoten einer unabhängigen Mengen, die der Onlinealgorithmus A ausgibt, erhalten. Zunächst betrachten wir die effektiven Kosten. Es gilt nach Annahme

$$cost_e(v_1) > cost_e(v_2) \ . \tag{6.2}$$

Dadurch, dass Behälter, die den Knoten v_1 enthalten, schlechte Behälter werden, werden die effektiven Kosten von v_1 von den Gesamtkosten abgezogen. Dem gegenüber steht eine Erhöhung der Kosten durch die effektiven Kosten von v_2. Diese Erhöhung fällt wegen (6.2) jedoch geringer aus. Nur die Kosten von Behältern, die allein den Knoten v_2 beinhalten, können die Gesamtkosten erhöhen. Da v_2 höchstens zu $r(n)$ vielen Behältern hinzugefügt wird und es insgesamt $n/2$ Runden gibt, erhalten wir als obere Schranke für die Gesamtkosten $r(n) \cdot n/2$. Für die Größe $|b|$ eines Behälters b gilt dann

$$\begin{aligned} & 2^{|b|-1} \leq r(n) \cdot \frac{n}{2} \\ \iff \qquad & |b| \leq \log(r(n) \cdot n) \ . \end{aligned}$$

Folglich erhalten wir für die Kompetivität des Onlinealgorithmus A

$$R_A(n) \geq \frac{n}{2 \cdot \log(n \cdot r(n))} \ . \qquad \square$$

Die gerade bewiesene untere Schranke ist demnach von der Größenordnung $\Omega(n/\log n)$ für den Fall, dass $r(n) = n^k$ gilt, wobei k eine Konstante ist. Mit Satz 6.9 haben wir bereits eine obere Schranke von $O(n/\log n)$ dafür nachgewiesen, so dass keine wesentlichen Verbesserungen von Aussagen bezüglich der Kompetivität von Algorithmen des Multisolution Modell der Onlinesituation des Problems Independent Set zu erwarten sind.

7 PRAKTISCHE UNTERSUCHUNGEN

In den vorangegangenen Kapiteln haben wir uns mit der Approximierbarkeit der Unabhängigkeitszahl in theoretischer Hinsicht auseinandergesetzt. Wir haben verschiedene Algorithmen zur Approximation unabhängiger Mengen kennengelernt und deren Approximationsgüte analysiert. Auf eine praktische Relevanz dieser Algorithmen sind wir dabei nicht eingegangen.

In diesem Kapitel werden wir Implementierungen einiger ausgewählter Algorithmen kennenlernen und deren Leistungsfähigkeit analysieren. Dabei werden wir vor allem die Größe der gefundenen unabhängigen Mengen und die dafür erforderliche Rechenzeit gegenüberstellen. Diese Vorgehensweise erscheint sinnvoll, da bei praktischen Berechnungen nicht nur Wert auf die Kardinalität einer gefunden Lösung gelegt wird, sondern diese auch in einer angemessenen Zeit berechnet werden sollte, da es wohl einen Unterschied macht, ob die Ausgabe innerhalb weniger Millisekunden oder erst nach mehreren Minuten oder sogar Stunden erzeugt werden kann. Dass die betrachteten Algorithmen sich in der Laufzeit, wie eben beschrieben, stark unterscheiden, werden wir im Abschnitt 7.2 sehen.

Als Programmiersprache wurde das Computeralgebrasystem Matlab[1] in der Version 6.5 Release 13 ausgewählt, weil damit effiziente Matrixoperationen zur Verfügung gestellt werden, die die Bearbeitung der Adjazenzmatrizen der gegebenen Graphen erleichtern. Aus diesem Grund ist der Quelltext einfach zu verstehen und ähnelt vom Aufbau stark den Algorithmen im Pseudocode aus den vorangegangenen Kapiteln. Für die Berechnung der $\vartheta(G)$-Zahl eines Graphen $G = (V, E)$ wurden vordefinierte Funktionen aus dem Softwarepaket SDPT3 in der Version 3.02[2] verwendet [41].

Zur Bereitstellung geeigneter Eingabegraphen wurde ein Algorithmus zur Erzeugung randomisierter Graphen $G = (V, E)$ entworfen. Dabei kann einerseits die Knotenanzahl $|V| = n$ sowie eine Kantenwahrscheinlichkeit p frei gewählt werden. Der verwendete Pseudozufallszahlen-Generator ist direkt in Matlab integriert und liefert Pseudozufallszahlen im Intervall $(0, 1)$. Nähere Details dazu sind der in Matlab enthaltenen Hilfe zu entnehmen.

Alle implementierten Algorithmen, die wir uns im nachfolgenden Abschnitt 7.1 genauer anschauen werden, wurden auf einem Rechner mit AMD Athlon XP 1800+ Prozessor und 512MB Arbeitsspeicher getestet. Die Taktfrequenz, welche nicht aus der Prozessorbezeichnung ersichtlich ist, beträgt dabei 1.54GHz.

[1] Eine gute Einführung in Matlab 6.5 bietet [29]. Weitere Informationen gibt es unter www.mathworks.de.
[2] Das Softwarepaket SDPT3 ist unter http://www.math.nus.edu.sg/~mattohkc/sdpt3.html verfügbar.

7.1 Details der Implementierung

7.1.1 Erzeugung randomisierter Graphen

Zum Erzeugen randomisierter Graphen wurde eine Funktion `graph` implementiert, welche als Eingabe die Parameter n und p erhält. Dabei ist n die Knotenanzahl und p die Kantenwahrscheinlichkeit des zu erzeugenden Graphen, welcher durch die Adjazenzmatrix G repräsentiert wird. Diese Matrix G ist zugleich der Rückgabewert der Funktion `graph`. Wird der Parameter p nicht übergeben[3], dann erhält p den Wert 0.5 (siehe *Zeile 3*). Zunächst wird in *Zeile 6* eine $n \times n$ Matrix mit Nullen erzeugt. Anschließend werden die Kanten „ausgewürfelt“, indem in *Zeile 9* eine Pseudozufallszahl bestimmt wird. Mit der entsprechenden Wahrscheinlichkeit wird dann eine Kante $\{i, j\}$ zum Graphen hinzugefügt, das heißt, die Positionen (i, j) und (j, i) der Adjazenzmatrix G erhalten den Wert 1, da wir nur ungerichtete Graphen betrachten. Zuvor wird jedoch der Pseudozufallszahlen-Generator in der *Zeile 5* initialisiert. Durch die Verwendung des Befehls `rand('state',sum(100*clock))` ist der Zustand des Generators abhängig vom Zeitpunkt der Initialisierung.

```
function G = graph(n,p);
    if (nargin < 2)
        p = 0.5;
    end
5   rand('state',sum(100*clock));
    G = false(n);
    for i = 1:n
        for j = i+1:n
            r = rand(1);
10          if (r < p)
                G(i,j) = 1;
                G(j,i) = 1;
            end;
        end;
15  end;
```

7.1.2 Der Greedy Algorithmus

Ein einfacher Algorithmus zur Bestimmung einer unabhängigen Menge in einem Graphen $G = (V, E)$ ist der Greedy Algorithmus. Dieser Algorithmus bestimmt eine bezüglich Inklusion maximale unabhängige Menge I, indem er beginnend bei $I = \emptyset$ einen Knoten $v \in V$ zur unabhängigen Menge I hinzufügt und anschließend den Knoten v sowie seine Nachbarn $N(v)$

[3] Mit Hilfe der Variable `nargin` kann die Anzahl der Eingabeparameter in Matlab bestimmt werden.

aus dem Graphen löscht. Dieses Vorgehen wird solange iteriert bis der Graph G keine Knoten mehr enthält. Das Löschen der Knoten aus dem Graphen erfolgt in dieser Implementierung mit Hilfe der Funktion `setdiff` in *Zeile 10*. Anschließend wird in *Zeile 11* der Greedy Algorithmus rekursiv auf dem Restgraphen aufgerufen. Der Abbruch dieser Rekursion erfolgt, falls der Graph leer ist. In diesem Fall ist die Dimension der Adjazenzmatrix 0 und der Algorithmus terminiert. Ersetzt man die *Zeile 9* durch `v = nodes(ceil(rand*n));` und initialisiert den Pseudozufallszahlen-Generator zuvor mit dem Befehl `rand('state',sum(100*clock))`, dann erhält man eine randomisierte Variante des Greedy Algorithmus, das heißt, in jedem Schritt wird ein „zufälliger“ Knoten bestimmt.

```
function I = greedy(A,nodes)
    if (nargin < 2)
        nodes = [1:1:length(A)];
    end
    n = length(nodes);
    if (n == 0)
        I = [];
    else
        v = nodes(1);
        nnb = setdiff(nodes,[find(A(v,:)) v]);
        I1 = greedy(A,nnb);
        I = [v I1];
    end
```

7.1.3 Der Min-Greedy Algorithmus

Eine Variante des Greedy Algorithmus ist der Min-Greedy Algorithmus [4, 22, 23]. Die einzige Änderung ist, dass in jedem Schritt ein Knoten mit minimalem Knotengrad gewählt wird. Die Suche nach einem solchen Knoten ist in den *Zeilen 9 bis 17* implementiert. Um die Nachbarn eines beliebigen Knotens zu bestimmen, wird die Matlab-Funktion `find` verwendet. Mit dem Funktionsaufruf `find(A(v,:))` werden alle Nachbarn des Knotens v bestimmt, indem alle Einträge der Zeile v der Adjazenzmatrix A, die den Wert 1 haben, zurückgegeben werden.

```
function I = mingreedy(A,nodes)
    if (nargin < 2)
        nodes = [1:1:length(A)];
    end
    n = length(nodes);
    if (n == 0)
```

```
        I = [];
    else
        mi = nodes(1);
        md = length(find(A(mi,:)));
        for i = 2:n
            di = length(find(A(nodes(i),:)));
            if (di < md)
                md = di;
                mi = nodes(i);
            end
        end
        nnb = setdiff(nodes,[find(A(mi,:)) mi]);
        I1 = mingreedy(A,nnb);
        I = [mi I1];
    end
```

7.1.4 Der Ramsey Algorithmus

Es folgt eine Implementierung des Ramsey Algorithmus aus Kapitel 3. Wie bereits beim Greedy Algorithmus diskutiert, kann auch eine randomisierte Variante des Ramsey Algorithmus implementiert werden, indem die *Zeile 10* durch `v = nodes(ceil(rand*n));` ersetzt wird, wobei zuvor wieder eine Initialisierung des Pseudozufallszahlen-Generators etwa mit dem Befehl `rand('state',sum(100*clock))` erfolgen muss. Zur Erläuterung des Quelltextes des Ramsey Algorithmus ist noch zu sagen, dass die Matlab-Funktion `intersect` die Schnittmenge zweier Mengen bildet. Im vorliegenden Fall wurde diese Funktion in *Zeile 11* verwendet, um die Nachbarn des gewählten Knotens v zu bestimmen, die außerdem noch im Restgraphen vorhanden sind. Die Knoten des Restgraphen sind in der Variablen `nodes` gespeichert. Ansonsten entspricht die Implementierung dem Ramsey Algorithmus aus Kapitel 3 und wird folglich an dieser Stelle nicht genauer erläutert.

```
function [I,C] = ramsey(A,nodes)
    if (nargin < 2)
        nodes = [1:1:length(A)];
    end
    n = length(nodes);
    if (n == 0)
        I = [];
        C = [];
    else
```

```
        v = nodes(1);
        nb = intersect(nodes,find(A(v,:)));
        nnb = setdiff(nodes,[nb v]);
        [I1,C1] = ramsey(A,nb);
        [I2,C2] = ramsey(A,nnb);
        if (length([C1 v]) > length(C2))
            C = [v C1];
        else
            C = C2;
        end
        if (length([I2 v]) > length(I1))
            I = [v I2];
        else
            I = I1;
        end
    end
```

In Kapitel 3 haben wir den Ramsey Algorithmus als Grundlage für den nun folgenden Algorithmus Clique-Removal verwendet. Dieser ruft zunächst den Ramsey Algorithmus[4] auf und entfernt dann die gefundene Clique aus dem Eingabegraphen. Dieses Vorgehen wird solange auf dem Restgraphen durchgeführt, bis kein Knoten mehr vorhanden ist. Mit Hilfe der Variablen `nodes` werden die noch vorhandenen Knoten gespeichert.

```
function I = cliqueremoval(A)
    n = length(A);
    nodes = [1:1:n];
    [I,C] = ramsey(A,nodes);
    while length(nodes)>0
        nodes = setdiff(nodes,C);
        A1 = A(nodes,nodes);
        [I1,C] = ramsey(A1,1:1:length(A1));
        C = nodes(C);
        I1 = nodes(I1);
        if (length(I1) > length(I))
            I = I1;
        end
    end
```

[4]Damit der Ramsey Algorithmus nicht immer gleiche unabhängige Mengen bestimmt, wurde beim Testen des Algorithmus auf die randomisierte Variante des Ramsey Algorithmus zurückgegriffen.

7.1.5 Laufzeitmessungen

Um genaue Laufzeitmessungen der implementierten Algorithmen vorzunehmen, wurden die Matlab-Funktionen `tic` und `toc` verwendet. Dabei ermittelt der Befehl `toc` die Zeit, welche seit dem letzten Befehl `tic` vergangen ist. Eine Zeitmessung am Beispiel des Greedy Algorithmus könnte etwa in folgender Form als Matlab Funktion geschehen.

```
function [I,zeit] = laufzeitmessung(A)
    tic;
    I = greedy(A);
    zeit = toc;
```

7.2 Ergebnisse und Analyse der Experimente

Wir werden nun die im vorangegangenen Abschnitt 7.1 vorgestellten Algorithmen bezüglich ihrer Laufzeit und Kardinalität der gefundenen unabhängigen Menge anhand zufälliger Eingabegraphen testen. Bevor wir jedoch damit beginnen, betrachten wir die Funktion `graph`, welche zum Erzeugen von zufälligen Graphen benutzt wird. Die Zeiten zur Erzeugung zufälliger Graphen sind der Tabelle 7.1 zu entnehmen. In der Praxis werden diese Zeiten weniger eine Rolle spielen, da die Eingabegraphen in diesem Fall auf eine andere Weise erzeugt werden.

Anzahl Knoten n \ Kantenwahrscheinlichkeit p	0.10	0.25	0.50	0.75
100	0.0276	0.0277	0.0280	0.0282
500	0.6687	0.6720	0.6787	0.6857
1000	2.6729	2.6949	2.7149	2.7389
5000	66.7438	67.5050	68.9170	70.0366
10000	258.8723	262.6327	266.3880	267.8475

Tabelle 7.1: Zur Erzeugung zufälliger Graphen benötigte Zeit in Sekunden

Bei unterschiedlicher Kantenwahrscheinlichkeit p ist, wie auch zu erwarten war, die Zeit zum Erzeugen entsprechender zufälliger Graphen mit gleicher Knotenanzahl unterschiedlich. Dieser Unterschied in der Laufzeit ist darin begründet, dass bei geringerer Kantenwahrscheinlichkeit die *Zeilen 11 und 12* der Funktion `graph` weniger oft durchlaufen werden, das heißt, es werden weniger Operationen auf der Adjazenzmatrix G ausgeführt als bei einer höher gewählten Kantenwahrscheinlichkeit p.

Zum Testen der implementierten Algorithmen wurden randomisierte Graphen mit unterschiedlicher Anzahl an Knoten und unterschiedlichen Kantenwahrscheinlichkeiten erzeugt.

Die zugrunde liegenden, randomisierten Graphen, deren Knotenanzahl aufgrund des zur Verfügung stehenden Speichers auf 10000 Knoten beschränkt wurde, befinden sich auf einer der Diplomarbeit beigelegten CD im Verzeichnis `Graphen`. Die Parameter n und p, das heißt die Anzahl Knoten und die Kantenwahrscheinlichkeit, sind dabei im Dateinamen der Graphen, der die Form `G_x_y.mat` hat, kodiert. Dabei steht `x` für die Anzahl Knoten und `y` für die Kantenwahrscheinlichkeit, die hier in Prozent angegeben ist. Beispielsweise bedeutet die Kodierung `G_5000_25`, dass der randomisierte Graph, welcher in dieser Datei gespeichert ist, 5000 Knoten besitzt und die Kantenwahrscheinlichkeit $p = 0.25$ ist. In Matlab kann dieser Graph mit dem Befehl `load G_5000_25` in den Arbeitsspeicher[5] geladen werden.

Wir kommen nun zu den Ergebnissen der einzelnen Testläufe. Wie bereits erwähnt, sind wir an der Kardinalität einer gefundenen unabhängigen Menge und der dafür benötigten Zeit interessiert. Wir beschränken uns zunächst auf die vier ausgewählten Algorithmen, deren Implementierung wir im vorangegangen Abschnitt kennengelernt haben. Die Testergebnisse können der Tabelle 7.2 entnommen werden.

Graph	Greedy		Min-Greedy		Ramsey		Clique-Removal	
	$\|I\|$	Zeit in s	$\|I\|$	Zeit in s	$\|I\|$	Zeit in s	$\|I\|$	Zeit in s
`G_1000_10.mat`	48	0.0300	48	0.7610	48	0.7110	49	78.4820
`G_1000_25.mat`	22	0.0200	23	0.3500	22	0.7210	22	57.1520
`G_1000_50.mat`	8	0.0000	10	0.1800	10	0.7610	12	36.3420
`G_1000_75.mat`	5	0.0000	5	0.0610	6	0.7710	7	20.1590
`G_5000_10.mat`	60	0.1410	67	33.9190	60	7.3500	64	2994.2360
`G_5000_25.mat`	25	0.0400	30	12.9680	26	7.8810	29	2197.5400
`G_5000_50.mat`	12	0.0300	14	6.3290	14	8.5030	14	1454.9320
`G_5000_75.mat`	6	0.0200	7	3.8560	8	9.2040	9	773.5720
`G_10000_10.mat`	63	0.1710	67	160.9210	63	23.4130	72	16244.2980
`G_10000_25.mat`	24	0.0800	32	61.2380	27	25.4570	31	12453.0160
`G_10000_50.mat`	13	0.0400	12	30.1330	14	28.7520	15	8340.2130
`G_10000_75.mat`	7	0.0300	7	18.2760	9	32.0060	9	4576.6510

Tabelle 7.2: Ergebnisse der implementierten Algorithmen

Die Anzahl Knoten n der untersuchten zufälligen Graphen ist dabei 1000, 5000 oder 10000 Knoten. Größere Graphen, das heißt Graphen mit deutlich höherer Knotenanzahl als 10000, zu testen, war aus Speicherplatzgründen nicht möglich. Aufgrund der Laufzeiten in Tabelle 7.2, insbesondere der Spalte des Algorithmus Clique-Removal, erscheint es auch nicht sinnvoll zu sein, größere Graphen zu testen, da die Laufzeit im Falle von Clique-Removal dann extrem hoch wäre.

[5] Nach dem Laden des Graphen ist dann die Variable `G_5000_25` im Workspace von Matlab verfügbar.

Außerdem wurden mit $p \in \{0.1, 0.25, 0.5, 0.75\}$ jeweils vier verschiedene Kantenwahrscheinlichkeiten getestet.

Es fällt schnell auf, dass die benötigten Laufzeiten stark differieren. Während der Greedy Algorithmus selbst bei 10000 Knoten eine Ausgabe in einem Bruchteil einer Sekunde produziert, benötigt der Algorithmus Clique-Removal für derartige Graphen schon einige Stunden. Dabei ist die Laufzeit in allen Fällen abhängig von der Anzahl Knoten und der Kantenwahrscheinlichkeit und damit von der Anzahl Kanten. Bei gleich bleibender Kantenwahrscheinlichkeit p gilt, wie die Daten zeigen, dass die Laufzeit mit zunehmender Knotenanzahl ansteigt. Gleiches gilt beim Ramsey Algorithmus auch für die Kantenanzahl beziehungsweise die Kantenwahrscheinlichkeit. Umgekehrt sieht es bei den restlichen drei Algorithmen aus. Hier nimmt die Laufzeit zu, wenn wir p kleiner wählen.

Bei den Greedy Algorithmen ist die Begründung einfach, da bei höherer Kantenwahrscheinlichkeit in jeder Iteration mehr Knoten gelöscht werden als bei geringerer Wahrscheinlichkeit p. Dies bedeutet, dass die Knotenanzahl schneller sinkt und insgesamt weniger Iterationen durchlaufen werden und der Greedy Algorithmus sowie der Min-Greedy Algorithmus folglich kürzere Laufzeiten besitzen.

Beim Algorithmus Clique-Removal liegt die Ursache ebenfalls in der Anzahl der Iterationen. Wir vergegenwärtigen uns noch einmal kurz die Vorgehensweise dieses Algorithmus. Es wird der Ramsey Algorithmus, welcher neben einer unabhängigen Menge I auch eine Clique C liefert, aufgerufen. Anschließend wird die Clique C aus dem Graphen entfernt und dieses Vorgehen auf dem Restgraphen iteriert. Falls keine Knoten mehr vorhanden sind, terminiert der Algorithmus und gibt die größte bisher gefundene unabhängige Menge aus. Die Anzahl Iterationen ist demnach davon abhängig, wie viele Knoten in jeder Iteration aus dem Graphen entfernt werden. Wenn die Kantenwahrscheinlichkeit p erhöht wird, liefert der Ramsey Algorithmus auch eine größere Clique zurück und der Restgraph enthält folglich weniger Knoten. Dies erklärt die höhere Laufzeit bei geringerer Kantenwahrscheinlichkeit.

Wir betrachten nun die Kardinalität der gefundenen unabhängigen Menge bei den vier implementierten Algorithmen. Zunächst sei bemerkt, dass es keinen Algorithmus gibt, der bei allen getesteten Graphen den höchsten Wert geliefert hat. Da die Ergebnisse stark von den verwendeten Graphen abhängen, war dies auch nicht zu erwarten. Umgekehrt lieferte der Greedy Algorithmus erwartungsgemäß fast immer[6] die schlechteste Lösung. Die dafür verbrauchte Zeit ist dennoch erheblich geringer als bei allen anderen Algorithmen, so dass bei praktischen Anwendungen, bei denen die vergangene Zeit eine Rolle spielt, diese Variante bevorzugt eingesetzt werden sollte. Wir werden gleich sehen, dass wir bei mehrfacher und randomisierter Ausführung des Greedy Algorithmus größere unabhängige Mengen erhalten können. Ansonsten kann man den Testergebnissen leicht entnehmen, dass trotz hohem Rechenaufwands, den

[6] Einzig bei dem Graphen `G_10000_50` lieferte der Min-Greedy Algorithmus eine schlechtere Lösung.

der Algorithmus Clique-Removal benötigt, nicht immer auch beste Ergebnisse zu erwarten sind. Zum Teil liefern andere und wesentlich schnellere Algorithmen auch größere unabhängige Mengen.

Wie gerade diskutiert, sind die vom Greedy Algorithmus gefundenen unabhängigen Mengen im Vergleich zu den Ergebnissen der anderen implementierten Algorithmen in den meisten Fällen kleiner. Um bessere Ergebnisse zu erhalten, bietet es sich beispielsweise an, dass in jeder Iteration ein Knoten zufällig[7] bestimmt wird und dieser anschließend der unabhängigen Menge I hinzugefügt wird. Zusätzlich kann diese Vorgehensweise mit einem Multistartansatz verbunden werden, das heißt wir starten den Greedy Algorithmus einfach mehrfach und geben das beste berechnete Ergebnis aus. In den durchgeführten Tests lag die Anzahl bei 100 Testläufen. Gleiche Überlegungen gelten natürlich auch für den Ramsey Algorithmus, bei dem ebenfalls 100 Testläufe durchgeführt wurden. Die Ergebnisse sind in der Tabelle 7.3 enthalten.

Graph	Greedy		Greedy (r+m)[8]		Ramsey		Ramsey (r+m)[8]	
	$\|I\|$	Zeit in s	$\|I\|$	Zeit in s	$\|I\|$	Zeit in s	$\|I\|$	Zeit in s
G_1000_10.mat	48	0.0300	50	0.0300	48	0.7110	52	0.7310
G_1000_25.mat	22	0.0200	24	0.0100	22	0.7210	23	0.7410
G_1000_50.mat	8	0.0000	11	0.0000	10	0.7610	12	0.7710
G_1000_75.mat	5	0.0000	7	0.0000	6	0.7710	8	0.8110
G_5000_10.mat	60	0.1410	65	0.1100	60	7.3500	65	7.6810
G_5000_25.mat	25	0.0400	30	0.0500	26	7.8810	30	8.1620
G_5000_50.mat	12	0.0300	14	0.0200	14	8.5030	14	8.9830
G_5000_75.mat	6	0.0200	8	0.0200	8	9.2040	9	9.6440
G_10000_10.mat	63	0.1710	71	0.1900	63	23.4130	71	24.1250
G_10000_25.mat	24	0.0800	32	0.0900	27	25.4570	30	26.2780
G_10000_50.mat	13	0.0400	17	0.0500	14	28.7520	15	29.1420
G_10000_75.mat	7	0.0300	9	0.0300	9	32.0060	9	33.7790

Tabelle 7.3: Vergleich mit mehrfachgestarteten randomisierten Algorithmen

Die durchgeführten Experimente haben gezeigt, dass wir bei mehrmaliger Ausführung einer randomisierten Variante des Greedy Algorithmus beziehungsweise des Ramsey Algorithmus bessere Ergebnisse erzielen können. Die dabei auftretende Streuung der Ergebnisse war besonders beim Greedy Algorithmus sehr hoch. Aufgrund der Betrachtungen in Kapitel 3 war dies auch nicht anders zu erwarten. Natürlich erhöht sich auch die Gesamtlaufzeit linear mit der Anzahl der Durchläufe. Beim Greedy Algorithmus ist diese Laufzeit bei 100-maliger Ausführung aber immer noch im Bereich weniger Sekunden und darum in der Praxis einsetzbar.

[7] Wie dieses in Matlab realisiert werden kann, wurde bereits in Abschnitt 7.1.2 diskutiert.

[8] Die Abkürzung (r+m) bedeutet, dass eine randomisierte Variante mehrfach (100 mal) gestartet wurde. Ein Eintrag in dieser Spalte ist die Kardinalität der größten gefundenen unabhängigen Menge.

Zu guter Letzt wurde noch die $\vartheta(G)$-Zahl (vgl. Kapitel 5) einiger Graphen berechnet. Für diese Berechnung wurde die Funktion `thetaproblem` des Softwarepakets SDPT3 in der Version 3.02 verwendet. Diese Funktion liefert neben einigen anderen Parametern die $\vartheta(G)$-Zahl eines Graphen $G = (V, E)$, der durch eine Adjazenzmatrix gegeben ist, zurück. Wegen der Komplexität der semidefiniten Programme und des zur Berechnung benötigten hohen Speicherbedarfs wurden nur Graphen $G = (V, E)$ mit $|V| = n \leq 100$ getestet. Neben der Berechung der $\vartheta(G)$-Zahl wurden auch unabhängige Mengen mit Hilfe der implementierten Algorithmen berechnet und verglichen. Aufgrund von (5.4) konnte damit bei einigen Graphen die Unabhängigkeitszahl $\alpha(G)$ exakt berechnet werden. Die genauen Ergebnisse befinden sich in Tabelle 7.4.

Graph	Greedy (r+m)[9]	Min-Greedy	Ramsey (r+m)[9]	Clique-Removal	$\vartheta(G)$	$\alpha(G)$
`G_10_10.mat`	8	8	8	8	8.0000	8
`G_10_25.mat`	6	6	6	6	6.0000	6
`G_10_50.mat`	4	4	4	4	4.0000	4
`G_10_75.mat`	4	4	4	4	4.0000	4
`G_20_10.mat`	14	13	14	13	14.0000	14
`G_20_25.mat`	9	9	9	8	9.0000	9
`G_20_50.mat`	5	4	5	4	5.0577	5
`G_20_75.mat`	3	3	3	3	3.2823	3
`G_50_10.mat`	18	19	19	18	20.3332	$\in \{19, \ldots, 20\}$
`G_50_25.mat`	11	11	12	10	12.7272	12
`G_50_50.mat`	7	7	7	6	7.5500	7
`G_50_75.mat`	4	5	5	4	5.0000	5
`G_100_10.mat`	28	26	28	25	33.1137	$\in \{28, \ldots, 33\}$
`G_100_25.mat`	15	12	15	13	19.1228	$\in \{15, \ldots, 19\}$
`G_100_50.mat`	8	8	8	7	10.1930	$\in \{8, 10\}$
`G_100_75.mat`	5	5	5	5	5.7165	5

Tabelle 7.4: Vergleich der Ergebnisse ausgewählter Algorithmen mit der $\vartheta(G)$-Zahl

Bei kleinen Graphen, das heißt bis etwa 50 Knoten, lagen die berechneten Ergebnisse sehr dicht an der ϑ-Zahl des jeweiligen Graphen. Dies bedeutet, dass die Unabhängigkeitszahl α dieser Graphen in den meisten Fällen exakt bestimmt werden konnte. Die besten Ergebnisse lieferte dabei der randomisierte Ramsey Algorithmus, wobei 100 Durchläufe durchgeführt wurden und die größte gefundene unabhängige Menge ausgeben wurde. Jedoch auch alle anderen getesteten Algorithmen lieferten gute Ergebnisse.

[9] Die Abkürzung (r+m) bedeutet, dass eine randomisierte Variante mehrfach (100 mal) gestartet wurde.

A EIN KURZER EINBLICK IN DIE KOMPLEXITÄTSTHEORIE

Eines der grundlegendsten Konzepte der Komplexitätstheorie und der Theorie der Berechenbarkeit ist das Modell der Turingmaschine. Zur Definition der Komplexitätsklasse $\mathcal{NP}$ wurde in Kapitel 2 bereits von der Idee der nichtdeterministischen Turingmaschine Gebrauch gemacht. Hier soll das Turingmaschinenmodell etwas genauer beschrieben werden. Wir beschränken uns zunächst auf deterministische Turingmaschinen. Dabei handelt es sich um eine Rechenmaschine, die aus einem Eingabeband unbegrenzter Länge, einem Lese- und Schreibkopf und einer Kontrolleinheit mit endlich vielen, akzeptierenden und nicht akzeptierenden Zuständen besteht. Das Eingabeband ist dabei in einzelne Felder beziehungsweise Zellen, die ein Zeichen des Bandalphabetes enthalten, aufgeteilt.

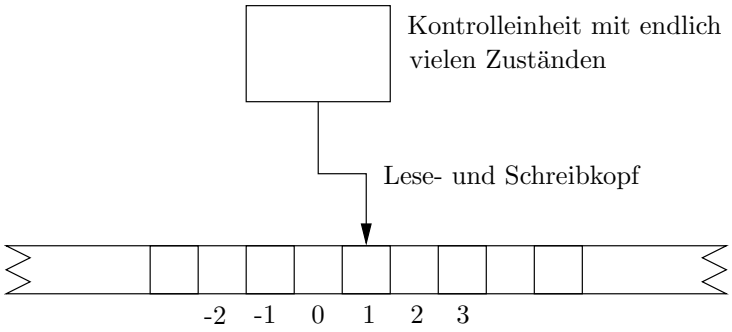

Abbildung A.1: Schematischer Aufbau einer Turingmaschine

In einem Rechenschritt wird das Zeichen auf dem Eingabeband an der Position des Lese- und Schreibkopfes gelesen und unter Berücksichtigung des aktuellen Zustands durch ein mittels einer Zustandsüberführungsfunktion eindeutig festgelegtes Zeichen des Bandalphabets überschrieben. Die Zustandsüberführungsfunktion definiert darüber hinaus einen Folgezustand und eine mögliche Bewegung des Lese- und Schreibkopfes um ein Feld nach links oder rechts. Im Unterschied zur deterministischen Turingmaschine handelt es sich bei der nichtdeterministischen Turingmaschine um eine Zustandsüberführungsrelation. Dies bedeutet, dass eine bestimmte Konfiguration[1] der nichtdeterministischen Turingmaschine mehrere Folgekonfigurationen zulässt. Eine genauere Charakterisierung wird in [33, 43, 44] gegeben.

[1]Eine Konfiguration ist durch Bandinhalt, Kopfposition und Zustand der Turingmaschine bestimmt.

Von grundlegender Bedeutung ist die Tatsache, dass das Modell der deterministischen Turingmaschine auf heutige Rechner übertragbar ist. Das heißt, dass unter der Annahme der Existenz eines in der Laufzeit polynomiell beschränkten Turingmaschinenprogramms für ein Problem gleichzeitig auch ein polynomiell beschränktes Programm zur Lösung dieses Problems auf einem realen Rechner existieren muss.

Aufbauend auf dem Modell der deterministischen Turingmaschine wird die sogenannte polynomielle Reduzierbarkeit von Sprachen definiert. Eine Sprache $\mathcal{L} \subseteq \Sigma^*$ entspricht dabei einer Menge von Zeichenketten beziehungsweise Wörtern $w \in \Sigma^*$ über einem beliebigen endlichen Alphabet Σ [43, 44].

Definition A.1 (polynomiell reduzierbar) *Es seien $\mathcal{L}_1$ und $\mathcal{L}_2$ Sprachen über Σ_1 und Σ_2. Dann heißt $\mathcal{L}_1$ polynomiell reduzierbar auf $\mathcal{L}_2$, wenn es eine von einer deterministischen Turingmaschine in polynomieller Zeit berechenbare Funktion $f\colon \Sigma_1^* \longrightarrow \Sigma_2^*$ gibt, so dass für alle Worte $w \in \Sigma_1^*$ gilt:*

$$w \in \mathcal{L}_1 \Longleftrightarrow f(w) \in \mathcal{L}_2 \; .$$

Man schreibt in diesem Fall $\mathcal{L}_1 \leq_p \mathcal{L}_2$.

Bereits in Kapitel 2 haben wir die Komplexitätsklasse $\mathcal{NP}$ definiert (vgl. Definition 2.6).

Definition A.2 ($\mathcal{NP}$-hart) *Eine Sprache $\mathcal{L}$ heißt $\mathcal{NP}$-hart, wenn für alle $\mathcal{L}' \in \mathcal{NP}$ gilt:*

$$\mathcal{L}' \leq_p \mathcal{L} \; .$$

Eine besondere Stellung nimmt nun das Problem SATISFIABILITY ein, das wie folgt definiert ist.

Definition A.3 (Problem Satisfiability) *Gegeben seien m Klauseln $C_1, \ldots, C_m$ über n Literalen $x_1, \ldots, x_n$, wobei wir unter einer Klausel eine Disjunktion endlich vieler Literale x_i oder deren Negation $\overline{x_i}$ verstehen. Bei dem Problem* SATISFIABILITY *wird gefragt, ob es eine Belegung der Literale mit den Wahrheitswerten true oder false beziehungsweise den Werten* 1 *oder* 0 *gibt, so dass alle Klauseln erfüllt sind, das heißt, in jeder Klausel gibt es mindestens ein Literal, die den Wert* 1 *erhält.*

Anfang der 70er Jahre gelang es Cook [8] für das Problem SATISFIABILITY (SAT) [33, 43, 44] die Eigenschaft $\mathcal{NP}$-hart nachzuweisen[2], das heißt, es gilt $\mathcal{L} \leq_p$ SAT für alle $\mathcal{L} \in \mathcal{NP}$. Um für ein Problem Π zu zeigen, dass es $\mathcal{NP}$-hart ist, genügt es nun, das Problem SAT polynomiell auf Π zu reduzieren, da die polynomielle Reduktion transitiv ist [44, Lemma 3.3.4].

Die folgenden drei Varianten des Problems INDEPENDENT SET sind paarweise polynomiell aufeinander reduzierbar [33, 44]:

[2] Die Idee von Cook war dabei, dass jeder mögliche Zustand, genauer jede mögliche Konfiguration, einer Turingsmaschine für eine beliebige Sprache $\mathcal{L} \in \mathcal{NP}$ in eine Reihe von Klauseln kodiert wird.

Variante 1: Gegeben sei ein Graph $G = (V, E)$ und eine Konstante $k \in \mathbb{N}$. Gefragt ist, ob es in G eine unabhängige Menge der Größe k gibt.

Variante 2: Gegeben sei ein Graph $G = (V, E)$. Gesucht ist die Kardinalität einer maximalen unabhängigen Menge in G.

Variante 3: Gegeben sei ein Graph $G = (V, E)$. Gesucht ist eine unabhängige Menge maximaler Kardinalität in G.

Es genügt daher die $\mathcal{NP}$-Härte von INDEPENDENT SET exemplarisch für eine dieser drei Varianten nachzuweisen. Im folgenden zeigen wir, dass Variante 1 $\mathcal{NP}$-hart ist.

Satz A.4 *Das Problem* INDEPENDENT SET *ist $\mathcal{NP}$-hart.*

Beweis: Wir reduzieren das Problem SAT polynomiell auf das Problem INDEPENDENT SET, das heißt, wir zeigen SAT $\leq_p$ INDEPENDENT SET.

Es seien $C_1, \ldots, C_m$ die Klauseln der Eingabe einer SAT-Instanz über den Literalen $x_1, \ldots, x_n$ sowie deren Komplementvariablen $\overline{x_1}, \ldots, \overline{x_n}$. Wir konstruieren nun daraus in Polynomialzeit eine Instanz für INDEPENDENT SET, das heißt, wir bilden einen Graphen $G = (V, E)$ und eine Konstante $k \in \mathbb{N}$. Für jede Klausel C_i, $i = 1, \ldots, m$, bilden wir einen vollständigen Graphen, wobei die Anzahl Knoten abhängig von der Anzahl der Literale einer Klausel ist. Falls eine Klausel C_i, $i = 1, \ldots, m$, l viele Literale enthält, dann bilden wir den vollständigen Graphen K_l auf l Knoten. Jeder Knoten eines vollständigen Graphen, die wir im Folgenden nur noch Komponenten nennen, repräsentiert somit ein Literal in der dazugehörigen Klausel. Die einzelnen Komponenten verbinden wir nun wie folgt: Wir verbinden zwei Knoten v und w unterschiedlicher Komponenten genau dann, wenn die dazugehörigen Literale gerade gegenseitig dem Komplement entsprechen, das heißt, repräsentiert der Knoten v das Literal x_j, $j = 1, \ldots, n$, dann fügen wir eine Kante zwischen v und w ein, wenn der Knoten w das Literal $\overline{x_j}$, $j = 1, \ldots, n$ repräsentiert. Den Parameter $k \in \mathbb{N}$ setzen wir auf m. Offensichtlich ist diese Transformation in Polynomialzeit durchführbar und demzufolge auch von einer deterministischen Turingmaschine in Polynomialzeit berechenbar.

Wir zeigen nun die folgende Äquivalenz: Die Klauseln $C_1, \ldots, C_m$ sind genau dann gleichzeitig erfüllbar, wenn es in G, dessen Konstruktion gerade beschrieben wurde, eine unabhängige Menge der Größe $k := m$ gibt.

„$\Longrightarrow$“: Es sei eine Belegung der Literale $x_1, \ldots, x_n$ mit den Wahrheitswerten *true* oder $false$ beziehungsweise 1 oder 0 gegeben, so dass in jeder Klausel mindestens ein Literal den Wert 1 hat. Wir wählen aus jeder Klausel ein solches Literal mit dem Wert 1, also insgesamt m Literale. Die diesen Literalen entsprechenden Knoten im Graphen G bilden offensichtlich eine unabhängige Menge der Größe $k = m$, da nach obiger Konstruktion nur Literale x_i und $\overline{x_i}$, $i = 1, \ldots, n$, unterschiedlicher Komponenten verbunden werden. Der Wert 1 ist jedoch nur für genau eines der beiden Literale möglich, damit eine zulässige Belegung der $x_1, \ldots, x_n$ vorliegt.

„$\Longleftarrow$“: Es sei I eine unabhängige Menge der Größe $|I| = k$. Offensichtlich kann nach obiger Konstruktion aus jeder Komponente nur ein Knoten in I enthalten sein. Da es genau $k = m$ Komponenten gibt, enthält jede Komponente genau einen Knoten der unabhängigen Menge I. Wir konstruieren aus I nun eine zulässige Belegung der Literale $x_1, \ldots, x_n$, indem die den Knoten aus I entsprechenden Literale den Wert 1 bekommen. Da dabei nach Konstruktion des Graphen G kein x_i und $\overline{x_i}$, $i = 1, \ldots, n$, den Wert 1 gleichzeitig erhält, ist die Belegung zulässig. Die Belegung der restlichen Literale ist dabei egal, vorausgesetzt sie erfolgt zulässig. Wir haben damit eine Belegung der Literale gefunden, die alle Klauseln gleichzeitig erfüllt, da in jeder Klausel mindestens eine, nämlich die dem Knoten aus I entsprechenden, Literale den Wahrheitswert *true* besitzt. □

Weitere Probleme, die wir in dieser Diplomarbeit erwähnen, werden wir nun definieren. Diese Probleme sind ebenfalls als $\mathcal{NP}$-hart bekannt [12], einen Beweis werden wir an dieser Stelle jedoch nicht führen.

Definition A.5 (Problem Coloring, chromatische Zahl $\chi(G)$) *Gegeben sei ein Graph $G = (V, E)$. Eine zulässige Färbung des Graphen G ist eine Abbildung $f\colon V \longrightarrow \mathcal{C}$, die jedem Knoten $v \in V$ ein Element aus $\mathcal{C}$ beziehungsweise eine „Farbe“ zuordnet, so dass alle Kanten $\{v, w\} \in E$ die Bedingung $f(v) \neq f(w)$ erfüllen, das heißt, adjazente Knoten sind unterschiedlich gefärbt. Bei dem Problem* Coloring *ist die minimale Anzahl Farben einer zulässigen Färbung des Graphen G gesucht, die als chromatische Zahl $\chi(G)$ bezeichnet wird.*

Definition A.6 (Problem Clique Cover, Cliquenüberdeckungszahl $\overline{\chi}(G)$) *Gegeben sei ein Graph $G = (V, E)$. Eine knotendisjunkte Partitionierung $V = C_1 \dot{\cup} \ldots \dot{\cup} C_m$ der Knotenmenge heißt Cliquenüberdeckung des Graphen G, wenn der auf C_i, $i = 1, \ldots, m$, induzierte Untergraph von G eine Clique ist. Bei dem Problem* Clique Cover *ist die minimale Anzahl Cliquen einer solchen Cliquenüberdeckung des Graphen G gesucht und wird als Cliquenüberdeckungszahl $\overline{\chi}(G)$ bezeichnet.*

ABBILDUNGSVERZEICHNIS

ALGORITHMENVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

$\exists$	„es gibt“
$\forall$	„für alle“
∞	unendlich
Σ	endliches Alphabet einer Sprache
Σ^*	Zeichenketten über dem endlichen Alphabet Σ
$\leq_p$	polynomielle Reduktion
$G = (V, E)$	Graph G mit Knotenmenge V und Kantenmenge E
$G[V']$	der auf die Knotenmenge V' induzierte Untergraph von G
$N(v)$	Nachbarschaft des Knotens $v \in V$
$\alpha(G)$	Unabhängigkeitszahl des Graphen G
$\omega(G)$	Cliquenzahl des Graphen G
$\chi(G)$	Chromatische Zahl des Graphen G
$\overline{\chi}(G)$	Cliquenüberdeckungszahl des Graphen G
$\vartheta(G)$	Lovász ϑ-Zahl des Graphen G
$\chi_v(G)$	vektorchromatische Zahl des Graphen G
$\chi_{sv}(G)$	starke vektorchromatische Zahl des Graphen G
$R(k, l)$	Ramsey-Zahl
e	Eulersche Zahl $e = 2,7182818\ldots$
$\|A\|$	Kardinalität einer Menge A
$\max A$	Maximum einer Menge A
$\min A$	Minimum einer Menge A
$\inf A$	Infimum einer Menge A
$\bigcup_{i=a}^{b} A_i$	Vereinigung $A_a \cup A_{a+1} \cup \ldots \cup A_b$
$\sum_{i=a}^{b} f(i)$	Summe $f(a) + f(a+1) + \ldots + f(b)$
$\prod_{i=a}^{b} f(i)$	Produkt $f(a) \cdot f(a+1) \cdot \ldots \cdot f(b)$
$\lfloor x \rfloor$	x abgerundet
$\lceil x \rceil$	x aufgerundet
$\log n$	Logarithmus von n zur Basis 2
$\ln n$	natürlicher Logarithmus von n
$[a, b]$	abgeschlossenes Intervall zwischen a und b
(a, b)	offenes Intervall zwischen a und b

$\binom{a}{b}$	Binomialkoeffizient „a über b“
$i \leftarrow \ldots$	Wertzuweisung
$\mathbb{N}$	Menge $\{0, 1, 2, \ldots\}$ der natürlichen Zahlen
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^n$	Menge der n-dimensionalen Vektoren über $\mathbb{R}$
$\mathbb{R}^{n \times n}$	Menge der $n \times n$-dimensionalen Matrizen über $\mathbb{R}$
$\mathcal{I}$	Einheitsmatrix
$\mathcal{J}$	Matrix, deren Einträge alle Null sind
$\mathrm{Tr}\, A$	Spur der Matrix A
$x^T y$	Skalarprodukt der Vektoren x und y
$\|x\|$	Länge des Vektors x
$\mathcal{P}$	Komplexitätsklasse $\mathcal{P}$
$\mathcal{NP}$	Komplexitätsklasse $\mathcal{NP}$
$O(g)$	Groß-„O“-Notation
$\Omega(g)$	Groß-„Omega“-Notation
$\Theta(g)$	Groß-„Theta“-Notation
$o(g)$	Klein-„O“-Notation
$\tilde{O}(g)$	Soft-Version der Groß-„O“-Notation, $\tilde{O}(n)$ entspricht $O(n \cdot poly(\log n))$
$\tilde{\Omega}(g)$	Soft-Version der Groß-„Ω“-Notation, $\tilde{\Omega}(n)$ entspricht $\Omega(n/poly(\log n))$

LITERATURVERZEICHNIS

[1] M. Aigner. *Diskrete Mathematik*. Vieweg Verlag, 4. durchgesehene Auflage, 2001.

[2] F. Alizadeh. Interior Point Methods in Semidefinite Programming with Application to Combinatorial Optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[3] N. Alon und N. Kahale. Approximating the Independence Number via the ϑ-Function. *Mathematical Programming*, 80:253–264, 1998.

[4] M. Baumgart. *Approximation von unabhängigen Mengen in dünnbesetzten Graphen*. Studienarbeit, Technische Universität Chemnitz, 2003.

[5] R. Beigel. Finding Maximum Independent Sets in Sparse and General Graphs. In *Proceedings of the 10th Annual ACM Symposium on Discrete Algorithms (SODA'99)*, 856–857, 1999.

[6] B. Berger und J. Rompel. A Better Performance Guarantee for Approximate Graph Coloring. *Algorithmica*, 5(3):459–466, 1990.

[7] R. Boppana und M. M. Halldórsson. Approximating Maximum Independent Sets by Excluding Subgraphs. *BIT*, 32(2):180–196, 1992.

[8] S. A. Cook. The Complexity of Theorem Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC'71)*, 151–158, 1971.

[9] T. Cormen, C. Leiserson und R. Rivest. *Introduction to Algorithms*. MIT Press, 2000.

[10] R. Diestel. *Graphentheorie*. Elektronische Ausgabe. Springer Verlag, Berlin, Heidelberg, New York, 2000.

[11] U. Feige. *Approximating Maximum Clique by Removing Subgraphs*. Manuskript, erscheint in SIAM Journal on Discrete Mathematics, 2002.

[12] M. R. Garey und D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[13] M. X. Goemans und D. P. Williamson. .878-Approximation for MAX CUT and MAX 2SAT. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC'94)*, 422–431, 1994.

[14] G. H. Golub und C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, 3. Auflage, 1996.

[15] R. Graham, B. Rothschild und J. Spencer. *Ramsey Theory*. Wiley Interscience, 2. Auflage, 1990.

[16] M. Grötschel, L. Lovász und A. Schrijver. The Ellipsoid Method and its Consequences in Combinatorial Optimization. *Combinatorica*, 1(2):169–197, 1981.

[17] M. Grötschel, L. Lovász und A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer Verlag, Berlin, Heidelberg, New York, 2. Auflage, 1993.

[18] M. M. Halldórsson. Approximations of Independent Sets in Graphs. In *Approximation Algorithms for Combinatorial Optimization, International Workshop (APPROX'98)*, 1–13, 1998.

[19] M. M. Halldórsson. Online Coloring Known Graphs. *Electronic Journal of Combinatorics*, 7(1):1–9, 2000.

[20] M. M. Halldórsson, K. Iwama, S. Miyazaki und S. Taketomi. Online Independent Sets. *Theoretical Computer Science*, 289(2):953–962, 2002.

[21] M. M. Halldórsson und J. Radhakrishnan. A Still Better Performance Guarantee for Approximate Graph Coloring. *Information Processing Letters*, 45:19–23, 1993.

[22] M. M. Halldórsson und J. Radhakrishnan. Greed is Good: Approximating Independent Sets in Sparse and Bounded-degree Graphs. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing (STOC'94)*, 439–448, 1994.

[23] M. M. Halldórsson und J. Radhakrishnan. Greed is Good: Approximating Independent Sets in Sparse and Bounded-degree Graphs. *Algorithmica*, 145–163, 1997.

[24] C. Helmberg, F. Rendl, B. Vanderbei und H. Wolkowicz. An Interior-Point Method for Semidefinite Programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996.

[25] T. Jian. An $O(2^{0.304n})$ Algorithm for Solving Maximum Independent Set Problem. *IEEE Transactions on Computers*, 35(9):847–851, 1986.

[26] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Science*, 9:256–278, 1974.

[27] D. Karger, R. Motwani und M. Sudan. Approximate Graph Coloring by Semidefinite Programming. *Journal of the ACM*, 45(2):246–265, 1998.

[28] R. M. Karp. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations. Plenum Press, New York*, 85–103, 1972.

[29] S. Katzenbeisser und C. Überhuber. *Matlab 6.5 – Eine Einführung*. Springer Verlag, Wien, New York, 2002.

[30] P. Klein und H. Lu. Efficient Approximation Algorithms for Semidefinite Programs Arising from Max Cut and Coloring. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC'96)*, 338–347, 1996.

[31] D. Knuth. The Sandwich Theorem. *The Electronic Journal of Combinatorics*, 1(A1):1–48, 1994.

[32] H. Lefmann. *Vorlesung Theoretische Informatik I.* Vorlesungsmanuskript, Technische Universität Chemnitz, 2002.

[33] H. Lefmann. *Vorlesung Theoretische Informatik II.* Vorlesungsmanuskript, Technische Universität Chemnitz, 2002.

[34] H. Lefmann. *Vorlesung Theoretische Informatik III.* Vorlesungsmanuskript, Technische Universität Chemnitz, 2002.

[35] L. Lovász. On the Shannon Capacity of a Graph. *IEEE Transactions on Information Theory*, IT-25:1–7, 1979.

[36] S. Mahajan und H. Ramesh. Derandomizing Semidefinite Programming based Approximation Algorithms. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, 162–169. IEEE Computer Society, 1995.

[37] H. J. Prömel, S. Hougardy, B. Kreuter, T. Emden-Weinert und A. Steger. *Einführung in Graphen und Algorithmen.* Vorlesungsmanuskript, Humboldt Universität Berlin, 1996. http://www.informatik.hu-berlin.de/Institut/struktur/algorithmen/ga/.

[38] J. M. Robson. Algorithms for Maximum Independent Sets. *Journal of Algorithms*, 7(3):425–440, 1986.

[39] J. M. Robson. *Finding a Maximum Independent Set in Time $O(2^{n/4})$*. Technical report, Université Bordeaux, 2001.

[40] R. E. Tarjan und A. E. Trojanowski. Finding a Maximum Independent Set. *SIAM Journal on Computing*, 6(3):266–283, 1977.

[41] R. H. Tütüncü, K. C. Toh und M. J. Todd. *SDPT3 – a Matlab Software Package for Semidefinite-Quadratic-Linear Programming, Version 3.0*. Technical report, National University of Singapore, 2001.

[42] L. Vandenberghe und S. Boyd. Semidefinite Programming. *SIAM Review*, 38(1):49–95, 1996.

[43] I. Wegener. *Kompendium Theoretische Informatik – eine Ideensammlung*. Teubner Verlag, 1996.

[44] I. Wegener. *Theoretische Informatik – eine algorithmische Einführung*. Teubner Verlag, 2. Auflage, 1999.

[45] A. Wigderson. Improving the Performance Guarantee for Approximate Graph Coloring. *Journal of the ACM*, 30(4):729–735, 1983.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig und ohne unerlaubte Hilfe angefertigt habe und keine anderen als die angegebenen Quellen benutzt habe.

Chemnitz, 14. September 2004 Matthias Baumgart