TECHNISCHE UNIVERSITÄT
CHEMNITZ

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

Diplomarbeit

Effiziente Färbungsalgorithmen für k-färbbare Graphen

vorgelegt von:

cand. Inf. Tobias Baumann
Matrikelnummer 23373

Betreuer: Prof. Dr. Hanno Lefmann

Chemnitz, 2. September 2004

Zusammenfassung

Das Problem, einen Graphen mit einer gegebenen Anzahl Farben zu färben, ist als NP-vollständig bekannt. Hier werden einige Algorithmen vorgestellt, die für dieses Problem eine gute Approximation liefern. Des Weiteren wird ein allgemeines Färbungsverfahren hergeleitet, das für k-färbbare Graphen den bisher besten existierenden Algorithmus darstellt. Es können k-färbbare Graphen mit $\tilde{O}(n^{\alpha(k)})$ Farben gefärbt werden, wobei $\alpha(2) = 0$, $\alpha(3) = 3/14$ und $\alpha(k) = 1 - \frac{6}{k+4+3(1-2/k)\frac{1}{1-\alpha(k-2)}}$ für $k \geq 4$ gilt [20]. Diese Formel wurde für neue Basisalgorithmen verallgemeinert.

Inhaltsverzeichnis

Abkürzungsverzeichnis

$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{N}$	Menge der natürlichen Zahlen
G, H	Graphen
V	Knotenmenge
E	Kantenmenge
$\|M\|$	Anzahl der Elemente einer Menge M
$n = \|V\|$	Knotenanzahl
$m = \|E\|$	Kantenanzahl
$u, v, w, x \in V$	Knoten
$\{u, v\} \in E$	Kante
$S, T \subseteq V$	Teilmengen der Knotenmenge
$N(v)$	Nachbarn eines Knotens $v \in V$
$N(S) = \bigcup_{v \in S} N(v)$	Nachbarn einer Menge $S \subseteq V$
$d(v) = \|N(v)\|$	Grad eines Knotens $v \in V$
$\Delta(G) = \max_{v \in V} d(v)$	maximaler Knotengrad des Graphen G
$d_{min} = \min_{v \in V} d(v)$	minimaler Knotengrad des Graphen G
$D(S) = \sum_{v \in S} d(v)$	Summe der Knotengrade in $S \subseteq V$
$d_T(v) = \|N(v) \cap T\|$	Anzahl der Nachbarn von $v \in V$ innerhalb der Menge $T \subseteq V$
$D_T(S) = \sum_{v \in S} d_T(v)$	Summe der Knotengrade zwischen den Mengen $S \subseteq V$ und $T \subseteq V$
$C \subseteq V$	Clique
$I \subseteq V$	unabhängige Menge
$i, j \in \mathbb{N}$	Indizes
k	Färbbarkeitszahl eines Graphen
α_k	Exponent bei einer n^{α_k}-Färbung eines k-färbbaren Graphen
$\chi(G)$	chromatische Zahl eines Graphen G
$\alpha(G)$	Unabhängigkeitszahl eines Graphen G
$\omega(G)$	Cliquenzahl eines Graphen G
$f(n)$	Funktion in Abhängigkeit von n
$O(f(n)), \Omega(f(n)), \Theta(f(n))$	Größenordnung der Funktion $f(n)$
$\tilde{O}(f(n)), \tilde{\Omega}(f(n))$	Größenordnung der Funktion $f(n)$ ohne logarithmische Faktoren
$\log_b n$	Logarithmus von n zur Basis b
$\log n$	Logarithmus von n zur Basis 2
$\Gamma(A)$	Güte eines Algorithmus A
$c_A(G)$	Anzahl der Farben, die Algorithmus A verwendet, um den Graphen G zu färben
p; $P[A]$	Elementarwahrscheinlichkeit; Wahrscheinlichkeit des Ereignisses A

Einleitung

Das Problem der Färbbarkeit von Graphen ist eines der am weitestgehend untersuchten Probleme der Theoretischen Informatik.

Definition 0.1. Ein *Graph* $G = (V, E)$ ist ein geordnetes Paar, bestehend aus einer *Knotenmenge* $V = \{v_1, v_2, ..., v_n\}$ und einer Menge ungerichteter *Kanten* $E \subseteq \{\{v_i, v_j\} : v_i, v_j \in V\}$.

Definition 0.2. Gegeben sei ein Graph $G = (V, E)$. Eine *k-Färbung* von G ist eine Funktion $c : V \to \{1, ..., k\}$, so dass für jede Kante $\{u, v\} \in E$ gilt $c(u) \neq c(v)$. Eine solche Funktion wird auch *zulässige* k-Färbung genannt. Eine Funktion $c : V \to \{1, ..., k\}$, die diese Bedingung nicht erfüllt, wird auch *unzulässige* Färbung von G genannt. Ein Graph G, für den es eine zulässige k-Färbung gibt, heißt *k-färbbar*.

Definition 0.3. Gegeben sei ein Graph $G = (V, E)$. Die kleinste natürliche Zahl k, für die es eine zulässige k-Färbung gibt, heißt *chromatische Zahl* $\chi(G)$. Ein Graph G ist *k-chromatisch*, wenn $\chi(G) = k$ gilt.

Die Bestimmung der chromatischen Zahl $\chi(G)$ ist für $\chi(G) > 2$ ein NP-hartes Problem. Viele Algorithmen zur Approximation von $\chi(G)$ sind im Laufe der Zeit veröffentlicht worden. In dieser Arbeit werden einige Herangehensweisen im Hinblick auf ihre Approximationsgüte untersucht und die neuesten Verfahren vorgestellt.
Daneben gibt es die Frage nach einer Färbung eines Graphen G mit möglichst wenig Farben. Hierbei sind zwei Ausgangspunkte zu unterscheiden: Einerseits ist die chromatische Zahl $\chi(G)$ bekannt, und andererseits nicht. Das letztere Problem führt wieder zur Bestimmung der chromatischen Zahl. Im ersten Fall dagegen gibt es eine Reihe von Approximationsalgorithmen, die seit den 90er Jahren vorgestellt wurden. Ziel dieser Arbeit ist es, diese Algorithmen hinsichtlich ihrer Laufzeit und Güte zu analysieren. Zusätzlich wird ein neuer, allgemeiner Approximationsalgorithmus zur Färbung k-färbbarer Graphen vorgestellt, der auf der Arbeit von Halperin, Nathaniel und Zwick [20] basiert.

Kapitel 1 liefert noch einige für den Fortgang wichtige Definitionen. In Kapitel 2 werden einige Algorithmen zur Approximation der chromatischen Zahl $\chi(G)$ eines Graphen G und zur Färbung von Graphen mit unbekannter chromatischer Zahl vorgestellt. Der Spezialfall der Färbung 3-färbbarer Graphen wird in Kapitel 3 betrachtet und in Kapitel 4 auf allgemeines k ausgeweitet. Dort befindet sich auch der neue allgemeine Färbungsalgorithmus für k-färbbare Graphen. Kapitel 5 beschäftigt sich mit einigen Ansätzen zur Färbung von Graphen aus praktischen Anwendungen. In Kapitel 6 finden sich dann andere Instanzen von Färbungsprobemen: die Online-Färbung von Graphen (Abschnitt 6.1), die Färbung von Hypergraphen (Abschnitt 6.3) und dem Färben verwandte Probleme. Schließlich wurden einige Färbungsalgorithmen ausgewählt und an Beispielgraphen getestet. Die Ergebnisse befinden sich in Kapitel 7.

Kapitel 1

Definitionen

Definition 1.1. Für einen Graphen $G = (V, E)$ und einen Knoten $v \in V$ sei die Menge $N(v) = \{w \in V : \{v, w\} \in E\}$ die *Nachbarschaft* von v. Analog sei für eine Knotenmenge $S \subseteq V$ die Menge $N(S) = \{w \in V : \exists v \in S : \{v, w\} \in E\} = \bigcup_{v \in S} N(v)$ als Nachbarschaft der Menge S definiert.

Definition 1.2. Gegeben sei ein Graph $G = (V, E)$. Es ist $d(v) = |\{e \in E : v \in e\}| = |N(v)|$ der *Grad* des Knotens v und $\Delta(G) = \max_{v \in V}\{d(v)\}$ der *maximale Grad* des Graphen.

Definition 1.3. Sei $G = (V, E)$ ein Graph und $S \subseteq V$ eine beliebige Knotenmenge in G. Dann sei $G[S]$ der auf S *induzierte Subgraph* von G, d.h. $G[S] = (S, E \cap \{\{v, w\} : v \in S, w \in S\})$.

Definition 1.4. Sei $G = (V, E)$ ein Graph. Eine *unabhängige Menge* in G ist eine Knotenmenge $I \subseteq V$, so dass der auf I induzierte Subgraph von G keine Kanten besitzt, d.h. $G[I] = (I, \emptyset)$. Eine *Clique* in G ist eine Knotenmenge $C \subseteq V$, in der alle möglichen Kanten enthalten sind, d.h. $G[C] = (C, \{\{u, v\} : u, v \in C\})$. Eine unabhängige Menge $I \subseteq V$ (Clique $C \subseteq V$) heißt *maximal*, wenn es keinen Knoten $v \in V \setminus I$ ($v \in V \setminus C$) gibt, so dass $I \cup \{v\}$ ($C \cup \{v\}$) eine unabhängige Menge (Clique) ist.

Definition 1.5. Eine Färbung c eines Graphen G heißt *besser* als eine andere Färbung d, wenn c weniger Farben als d verwendet.

Definition 1.6. Sei c eine Färbung eines Graphen $G = (V, E)$. Die *Farbklassen* von c sind Mengen $I_j \subseteq V$ mit $I_j = \{v \in V : c(v) = j\}$. Jede Farbklasse I_j von c ist eine unabhängige Menge in G. Zwei Färbungen c und d eines Graphen heißen *verschieden*, wenn sie verschiedene Farbklassen implizieren.

Definition 1.7. Sei $G = (V, E)$ ein Graph. Der *Komplementärgraph* von G ist der Graph $\bar{G} = (V, \{e \in [V]^2 : e \notin E\}$. Eine unabhängige Menge in G ist gleichzeitig eine Clique in $\bar{G}$ und umgekehrt.

Der einfachste Fall ist die exakte Färbung eines 2-färbbaren Graphen. Dies kann mit Hilfe der Breitensuche in dem folgenden Algorithmus 1.1 durchgeführt werden. Ein 2-färbbarer Graph wird auch bipartit genannt.

Algorithmus 1.1. ***2-Color(G)***

Eingabe: Ein Graph $G = (V, E)$.

Ausgabe: Eine zulässige 2-Färbung von G, sofern möglich.

1. v_0 := ein beliebiger Startknoten;
   $Color(v_0) := 0$
   $Q := \emptyset$
2. `for each` $u \in N(v_0)$:
   $Color(u) := 1$;
   $Enqueue(Q, u)$;
3. `while` $Q \neq \emptyset$:
   $v := First(Q)$;
   `for each` $u \in N(v)$:
   · `if` $Color(u) = Color(v) \Rightarrow$ `return` Nicht 2-färbbar.
   · `else if` $u \notin Q \Rightarrow \{Color(u) := 1 - Color(v); Enqueue(Q, u)\}$

Für die Approximation der chromatischen Zahl wurde eine untere Schranke gefunden: Garey und Johnson zeigten, dass es ein NP-vollständiges Problem ist, die chromatische Zahl $\chi(G)$ auf den Wert $a\chi(G) + b$ mit $a, b \in \mathbb{R}^+,\ a < 2$ zu approximieren: ein Polynomialzeitalgorithmus, der einen Graphen G mit $(2 - \varepsilon)\chi(G)$ Farben färbt, impliziert einen exakten Polynomialzeitalgorithmus für das Färbbarkeitsproblem [17].
Einige Probleme sind so strukturiert, dass man bereits im Voraus weiß, dass der Graph G k-färbbar ist. Diese Information nutzt für $2 < k < n = |V|$ nicht viel; so haben z.B. Khanna, Linial und Safra gezeigt, dass ein 3-färbbarer Graph nicht in Polynomialzeit mit 4 Farben gefärbt werden kann, es sei denn, P=NP [25].

Es stellt sich nun die Frage, wie man einen k-färbbaren Graphen erhält. Wenn man einen zufälligen Graphen G erstellt, dann kann man zwar einen Bereich berechnen, in dem sich $\chi(G)$ aufhält, aber man kann die chromatische Zahl nicht exakt bestimmen.[1] Dagegen gibt es verschiedene Verfahren, mit denen man zufällige Graphen konstruieren kann, die garantiert k-färbbar sind.
Algorithmus 1.2 wurde gegenüber einem Verfahren aus [4] erweitert, mit dem ein 3-färbbarer Graph konstruiert wurde.

Algorithmus 1.2. ***makeGraph1*(*n*,*k*,*p*)**

Eingabe: Knotenanzahl n, gewünschte Farbanzahl k, Kantenwahrscheinlichkeit p.

Ausgabe: ein k-färbbarer Graph G.

$a_1, ..., a_{k-1} := k-1$ paarweise verschiedene zufällig gewählte Zahlen aus der Menge $\{1, ..., n\}$ mit $a_1 < a_2 < ... < a_{k-1}$.

`for each` $u, v \in V, u < v$:

· `if` $u < a_i$ und $v \geq a_{i+1} \Rightarrow$ füge Kante $\{u, v\}$ mit Wahrscheinlichkeit p in E ein.

$\Pi(V) :=$ eine zufällige Permutation der Knotenmenge.

Gib $G = (\Pi(V), \Psi(E))$ mit $\{u, v\} \in E \Leftrightarrow \{\pi(u), \pi(v)\} \in \Psi(E)$ zurück. ($\Psi(E)$ ist die zu $\Pi(V)$ gehörende Permutation der Kanten)

Algorithmus 1.3 stammt aus [34] und konstruiert ebenso einen k-färbbaren Graphen.

Algorithmus 1.3. ***makeGraph2*(*n*,*k*,*p*)**

Eingabe: Knotenanzahl n, gewünschte Farbanzahl k, Kantenwahrscheinlichkeit p.

Ausgabe: ein k-färbbarer Graph G.

`for each` $v \in V$: color(v):=random$(1, ..., k)$ gemäß Gleichverteilung

`for each` $u, v \in V$:

· `if` color(u)$\neq$ color(v) $\Rightarrow$ füge Kante $\{u, v\}$ mit Wahrscheinlichkeit p in E ein.

Lösche das Array color.

Beide Algorithmen weisen den Knoten zunächst eine zufällige Farbe zu, um danach die Kanten einzufügen. Die Graphen, die von den Algorithmen erstellt werden, sind auf jeden Fall k-färbbar[2]; es kann aber durchaus sein, dass der entstandene Graph auch mit weniger Farben

[1] Für einen zufälligen Graphen G gilt $\chi(G) = \Theta(n/\log n)$ [10, 29].

[2] Es werden nur k Farben verwendet, und wenn zwei Knoten u, v dieselbe Farbe erhalten, dann kann keine Kante $\{u, v\}$ entstehen.

gefärbt werden kann. Die auf diese Weise konstruierten Graphen sind keinesfalls gleichförmig aus sämtlichen k-färbbaren Graphen gewählt. Vielmehr wird jedem k-färbbaren Graphen G mit n Knoten eine positive Wahrscheinlichkeit zugeordnet, mit der er dann gezogen wird. Graphen, die sich auf viele verschiedene Weisen darstellen lassen, erhalten dadurch eine entsprechend höhere Wahrscheinlichkeit als Graphen, die kaum Isomorphieeigenschaften haben. So kann z.B. der leere Graph $G = (V, \emptyset)$ nur auf eine einzige Weise dargestellt werden und erhält die Wahrscheinlichkeit $P[(V, \emptyset)] = (1 - p)^n$. Ein Graph, bei dem zwei beliebige Knoten denselben Grad haben, kann dagegen durch Vertauschen dieser Knoten auf eine andere Weise dargestellt werden. Färbungsalgorithmen, die mit Graphen, die von Algorithmus 1.3 konstruiert wurden, arbeiten, werden in Kapitel 5 näher betrachtet.

Beide Algorithmen können nicht garantieren, dass es sich bei den entstehenden Graphen um k-chromatische Graphen handelt, da mit einer gewissen Wahrscheinlichkeit auch keine Kanten zwischen zwei Farbklassen verlaufen können.[3] Außerdem ist es möglich, dass eine Farbklasse i gar nicht auftritt, weil z.B. die Parameter n und k ungünstig gewählt wurden. Weiterhin müssen die k-Färbungen nicht eindeutig sein, da eine andere Partition der Knotenmenge ebenfalls zu einer zulässigen k-Färbung führen kann (oder sogar zu Färbungen mit weniger Farben).

1.1 Verwandte Probleme

Eine k-Färbung partitioniert einen Graphen in k disjunkte unabhängige Mengen. im Komplementärgraphen werden aus den unabhängigen Mengen Cliquen. Somit ist das Problem Minimum Independent Set Cover ebenso wie Minimum Clique Cover zum Problem Minimum Coloring äquivalent.

Definition 1.8. Das Problem Minimum Independent Set Cover ist wie folgt definiert: Gegeben sei ein Graph $G = (V, E)$; gesucht ist die minimale Anzahl unabhängiger Mengen $I_j \subseteq V$, so dass $\bigcup I_j = V$ gilt. Analog hierzu ist das Problem Minimum Clique Cover definiert, nur ist hier die minimale Anzahl Cliquen $C_j \subseteq V$ gesucht, so dass $\bigcup C_j = V$ gilt.

Man kann das Färbbarkeitsproblem auch auf die Zählvariante ausweiten: Gegeben sei ein Graph; wie viele zulässige k-Färbungen gibt es für den Graphen? Dieses Problem ist dann #P-vollständig [33].

[3] Ist dies der Fall, dann können die beiden Farbklassen fusioniert werden, und es entsteht eine $k-1$-Färbung.

Definition 1.9. Die Klasse #P ist die Menge der Zählprobleme (Wie viele Lösungen gibt es?), deren Entscheidungsvariante in NP liegt.

Eng verwandt ist auch das Problem MAX-k-CUT. Hierbei wird eine k-Partition der Knotenmenge V gesucht, so dass die Anzahl der Kanten zwischen den Mengen maximiert wird. Wenn der Graph k-färbbar ist, dann impliziert eine zulässige k-Färbung gerade die Lösung des MAX-k-CUT-Problems. In der anderen Richtung ist eine Lösung von MAX-k-CUT, bei der alle Kanten zwischen den Partitionen verlaufen, eine zulässige k-Färbung des Graphen G. Wenn ein Graph G eine Clique der Größe k enthält, dann ist dieses k auch eine untere Schranke für die Anzahl der zu verwendenden Farben. Die Lösung des Problems MAXIMUM CLIQUE kann also auch einen Schritt zur erfolgreichen Färbung mit wenig Farben bedeuten, da es für die Knoten der Clique nur eine einzige zulässige Färbung gibt.[4] Kapitel 5 wird dazu noch einige Betrachtungen liefern.

Um eine Färbung mit wenig Farben zu erhalten, ist es auch hilfreich, eine große unabhängige Menge zu finden, diese mit einer Farbe zu färben, und dann mit einem kleineren Graphen fortzufahren. Dies führt zu den Problemen MAXIMUM INDEPENDENT SET und MINIMUM VERTEX COVER.

Auch in praktischen Anwendungen kann man Probleme finden, die auf die Färbbarkeit zurückzuführen sind: Zu nennen wären da *Register Allocation*, *Scheduling* und *Frequency Assignment*. Letzteres stellt wieder eine Sonderform dar: Es müssen Radiofrequenzen regional verteilt werden, so dass sich an einem Punkt die Frequenzen verschiedener Sender nicht überlappen. Diese Fragestellung führt zur Färbung von Hypergraphen (siehe Kapitel 6). Die Kanten eines Hypergraphen sind nicht notwendigerweise 2-elementige Teilmengen der Knotenmenge.

Alle aufgeführten Probleme sind NP-vollständig, d.h. mit einem exakten Polynomialzeitalgorithmus für eines der Probleme erhält man auch einen Polynomialzeitalgorithmus für alle NP-vollständigen Probleme.

[4] Allerdings kann es auch sein, dass ein Graph keine großen Cliquen enthält, aber trotzdem viele Farben benötigt, so z.B. die *Mycielski-Graphen*, deren maximale Clique die Größe 2 hat, aber die chromatische Zahl kann beliebig groß werden.

Kapitel 2

Bestimmung der chromatischen Zahl

Die Bestimmung der chromatischen Zahl $\chi(G)$ geht Hand in Hand mit einer möglichen optimalen Färbung von G. Die wenigsten Algorithmen bestimmen nur die chromatische Zahl, ohne eine Färbung (bzw. Partition der Knotenmenge) anzugeben. Anders als bei Problemen wie Clique oder Independent Set kann man hier nicht von einem Polynomialzeitalgorithmus zur Bestimmung der chromatischen Zahl auf einen Polynomialzeitalgorithmus für eine $\chi(G)$-Färbung schließen [25]. Andererseits bemerkt man deutliche Unterschiede in der Herangehensweise zwischen Algorithmen für Graphen mit bekannter chromatischer Zahl und für Graphen, deren chromatische Zahl unbekannt ist.
In diesem Kapitel werden zunächst die grundlegenden *schnellen* Färbungsalgorithmen vorgestellt, um damit zu exakten Färbungsalgorithmen überzugehen. Den Abschluss bilden Algorithmen, die bei unbekannter chromatischer Zahl approximativ die besten Färbungen erzielen.

2.1 Einfache Algorithmen

Im allgemeinen Fall weiß man über den gegebenen Graphen recht wenig Bescheid. Effizient erreichbare Ergebnisse sind rar gesät, wie z.B. die Verteilung der Knotengrade oder die Entscheidung, ob der Graph 2-färbbar ist. Wenn man a priori keine weiteren Informationen bekommt, dann ist es erforderlich, die chromatische Zahl des Graphen mit einem Färbungsalgorithmus zu approximieren[1].

Definition 2.1. Sei **A** ein Approximationsalgorithmus zur Färbung eines Graphen G. Dann

[1]Ein spezieller Algorithmus für k-färbbare Graphen kann dagegen durch Rekursion bessere Ergebnisse erzielen.

ist die *Güte* Γ von $\mathbf{A}$ definiert als:

$$\Gamma(\mathbf{A}) = \max_G \frac{c_{\mathbf{A}}(G)}{\chi(G)},$$

wobei $c_{\mathbf{A}}(G)$ die Anzahl der von Algorithmus $\mathbf{A}$ verwendeten Farben für den Graphen G ist.

Die Güte betrachtet also den schlechtestmöglichen Fall, dass ein Graph G eine kleine chromatische Zahl besitzt, der Algorithmus $\mathbf{A}$ aber viele Farben benötigt.
Ein recht einfaches Verfahren zur Approximation der chromatischen Zahl ist der Greedy-Algorithmus, der sequentiell die Knoten mit der kleinsten möglichen Farbe aus der Menge $\mathbb{N} = \{1, 2, ...\}$ färbt.

Algorithmus 2.1. ***colorGreedy*(*G*)**

Eingabe: Ein Graph G.

Ausgabe: Eine zulässige Knotenfärbung von G.

`while` nicht alle Knoten sind gefärbt:

· $v :=$ ein beliebiger ungefärbter Knoten

· Färbe v mit der kleinsten Farbe, die noch kein Nachbar von v hat.

Der Greedy-Algorithmus bietet kaum Anhaltspunkte, seine Güte einzuschätzen. Der Algorithmus wird maximal $\Delta(G) + 1$ Farben verwenden, da im schlechtesten Fall die Nachbarn eines Knoten komplett eingefärbt sind, und zwar alle mit verschiedenen Farben. Für viele Anwendungen bleibt die chromatische Zahl im Verhältnis zur Knotenzahl sehr klein, während der maximale Grad $\Delta(G)$ meist proportional zur Anzahl der Knoten ist. Man kann sogar Graphen G mit n Knoten finden, bei denen der Greedy-Algorithmus $\Theta(n)$ Farben verwendet, obwohl die chromatische Zahl konstant ist (siehe Abschnitt 6.1). Damit erreicht die Güte von Algorithmus 2.1 den Wert $\Gamma(\textit{colorGreedy}) = O(n)$.
Eine Verbesserung von Algorithmus 2.1 ist das *Minimum Choice*-Prinzip. In einfachen Worten erklärt, wird dabei immer ein Knoten ausgewählt, der innerhalb der Teilfärbung eine minimale Auswahlmöglichkeit bietet.

Definition 2.2. Für einen teilweise gefärbten Graphen $G = (V, E)$ sei $\tilde{d}(v)$ die Anzahl der ungefärbten Nachbarn von $v \in V$ und $C(v)$ die Anzahl der in der Färbung zulässigen Farben für v, ohne eine neue Farbe hinzuzunehmen.

Wenn der gesamte Graph ungefärbt ist, dann gilt $C(v) = 0$ und $\tilde{d}(v) = d(v)\ \forall v \in V$.

Algorithmus 2.2. ***minChoice*(*G*)**

Eingabe: Ein Graph G.

Ausgabe: Eine zulässige Knotenfärbung von G.

`while` nicht alle Knoten sind gefärbt:

· v_0 := ein ungefärbter Knoten mit $C(v_0) = \min_v C(v)$; gibt es mehrere solcher Knoten, dann wähle denjenigen mit $\tilde{d}(v_0) = \max_v \tilde{d}(v)$.

· Färbe v_0 mit der kleinsten Farbe, die noch kein Nachbar von v_0 hat.

Mit diesem Algorithmus werden die Restriktionen ausgenutzt, die beim Färben eines Knotens entstehen. Es wird ein Knoten ausgewählt, für den schon viele Farben ausgeschlossen werden konnten, und mit dem für viele weitere ungefärbte Knoten möglicherweise eine Farbe ausgeschlossen werden kann. Zu Beginn wird ein Knoten v maximalen Grades gewählt, dieser gefärbt, und der nächste Knoten w in der Nachbarschaft von v gesucht. Dann werden die Knoten in der gemeinsamen Nachbarschaft der Knoten v und w betrachtet und der Knoten gewählt, der die meisten ungefärbten Nachbarn besitzt. Das wird so lange fortgesetzt, bis die gemeinsame Nachbarschaft der bisher gefärbten Knoten leer ist. Zu diesem Zeitpunkt wurde eine maximale Clique konstruiert. Ausgehend von dieser Clique, deren Größe eine untere Schranke für die zu verwendende Farbanzahl darstellt, kommt man im Allgemeinen zu einer recht guten Annäherung an $\chi(G)$, obwohl dies nicht gut quantitativ eingeschätzt werden kann. Die entstehende Färbung nach Algorithmus 2.2 verwendet im Allgemeinen aber weniger Farben als eine Färbung nach Algorithmus 2.1. In Kapitel 5 wird nochmals ausführlich auf Algorithmus 2.2 eingegangen. Algorithmus 2.2 wird im Allgemeinen auch als *DSATUR*-Algorithmus bezeichnet, abgeleitet von der Sättigung der zu färbenden Knoten.

2.2 Exakte Algorithmen

Zunächst wird ein Algorithmus vorgestellt, der auf dem bereits im vorangegangenen Abschnitt vorgestellten Greedy-Algorithmus 2.1 basiert. Er taucht in dieser Form auch bei Coudert [10] auf. Hier wird einfach nacheinander den Knoten eine Farbe zugewiesen und so lange systematisch durchprobiert, bis die erreichte Färbung mit einer unteren Schranke übereinstimmt oder sämtliche Möglichkeiten ausprobiert worden sind. Die untere Schranke wird durch eine Clique

repräsentiert, die z.B. durch einen Greedy-Algorithmus oder ein anderes Verfahren bestimmt wird.

Algorithmus 2.3. ***exactGreedy*(*G*)**

Eingabe: Ein Graph G mit n Knoten.

Ausgabe: Eine optimale Färbung von G.

$C :=$ eine maximale Clique von G.

Färbe C eindeutig mit $|C|$ Farben.

`return` *exactGreedyRec*(G,$|C|$,$n+1$,$|C|$).

***exactGreedyRec*(*G*,*k*,*best*,*lb*)**

Eingabe: ein mit k Farben zum Teil gefärbter Graph G, die bisher beste bekannte Farbanzahl *best*, eine untere Schranke *lb*.

Ausgabe: eine neue beste Färbung.

`if` G ist komplett gefärbt $\Rightarrow$ {Speichere die neue beste Färbung; `return` k}.

$v :=$ ein ungefärbter Knoten von G.

`for` $c := 1..\min(k+1, best-1)$

· `if` kein Nachbar von v ist mit c gefärbt

... Färbe v mit c

... $best :=$ *exactGreedyRec*(G,$\max(c,k)$,*best*,*lb*)

... Entfärbe v

... `if` $lb = best$ `return` *best*

`return` *best*

Die Entfärbung des Knotens v hat den Sinn, dass eine neue Knotenfärbung ausprobiert werden kann, die evtl. besser ist. Sollte der Algorithmus so weit kommen, dass der komplette Graph gefärbt ist, dann wird diese Färbung als neue beste Färbung gespeichert. Diese beste Färbung wird am Ende des Algorithmus ausgegeben. Algorithmus 2.3 bricht nur dann vorzeitig ab, wenn eine vorher berechnete untere Schranke erreicht wird. Die einzige untere Schranke, die man für die Färbung eines Graphen angeben kann, ist eine größte Clique im Graphen. Abgesehen davon, dass das Problem Maximum Clique ebenfalls NP-vollständig ist, kann man nicht mit Sicherheit sagen, dass ein Graph G, dessen größte Clique k Knoten umfasst,

auch wirklich mit k Farben färbbar ist. Eine ausführliche Betrachtung dieser Problematik findet sich in Kapitel 5.

Der nächste Algorithmus basiert auf dem Minimum Choice-Prinzip, ebenso wie Algorithmus 2.2. Gegenüber der Greedy-Variante (Algorithmus 2.3) hat der Algorithmus genau wie Algorithmus 2.2 gegenüber Algorithmus 2.1 den Vorteil, dass zuerst Knoten mit kleiner Auswahl gefärbt werden, so dass viele neue Restriktionen erstellt werden. Es wird wieder auf Definition 2.2 hingewiesen.

Algorithmus 2.4. ***exactMinChoice*(*G*)**

Eingabe: Ein Graph G.

Ausgabe: Eine optimale Färbung von G.

$C :=$ eine maximale Clique von G.

Färbe C eindeutig mit $|C|$ Farben.

`return` *exactMinChoiceRec*(G,$|C|$,$n+1$,$|C|$).

***exactMinChoiceRec*(*G*,*k*,*best*,*lb*)**

Eingabe: ein mit k Farben teilweise gefärbter Graph G, die bisher beste Farbanzahl *best*, eine untere Schranke *lb*.

Ausgabe: eine neue beste Färbung.

`if` G ist komplett gefärbt $\Rightarrow$ {Speichere die neue beste Färbung; `return` k}.

$v_0 :=$ ein ungefärbter Knoten wie in Algorithmus 2.2 mit $C(v_0) = \min_v C(v)$ und $\tilde{d}(v_0) = \max\{\tilde{d}(v) : C(v) = C(v_0)\}$.

`for` $c := 1..\min(k+1, best-1)$

· `if` kein Nachbar von v_0 ist mit c gefärbt

... Färbe v_0 mit c

... $best :=$ *exactMinChoiceRec*(G,$\max(c,k)$,*best*,*lb*)

... Entfärbe v_0

... `if` $lb = best$ `return` *best*

`return` *best*

Die beiden Algorithmen haben sehr unterschiedliche Laufzeiten. Während Algorithmus 2.3 einfach die Knoten sequentiell färbt, ohne Rücksicht auf die Eigenschaften der Knoten, sucht sich Algorithmus 2.4 immer den Knoten aus, der möglichst wenige Umfärbungen zulässt und

gleichzeitig vielen seiner Nachbarn zusätzliche Bedingungen aufzwingt. Obwohl die Laufzeiten beider Algorithmen asymptotisch exponentiell sind, unterscheidet sich die Effizienz der beiden Algorithmen erheblich: Mit Algorithmus 2.3 hat man bei der Auswahl des nächsten zu färbenden Knoten keine Probleme. Dies geschieht in konstanter Zeit. Algorithmus 2.4 dagegen sucht sich den nächsten zu färbenden Knoten nach einer bestimmten Vorgabe aus. Es werden sämtliche ungefärbten Knoten verglichen, um den richtigen Knoten zu finden. Bei geeigneter Implementierung erhält man hier eine lineare Laufzeit zur Aufrechterhaltung der Knotendaten.[2]

Einen dritten - randomisierten - Algorithmus kann man noch anführen. Dieser Algorithmus basiert auf dem *Local Search*-Prinzip, d.h. ausgehend von einer bestehenden zulässigen Färbung sucht er eine neue Färbung, die weniger Farben verwendet. Dabei wird immer nur die Farbe eines Knotens geändert.

Definition 2.3. Sei $G = (V, E)$ ein Graph und $f : V \to \{1, ..., k\}$ eine (nicht notwendigerweise zulässige) Färbung von G. Die Anzahl $t(f)$ der *Konflikte* in der Färbung f ist bestimmt durch $t(f) = |\{\{u, v\} \in E : f(u) = f(v)\}|$.

Algorithmus 2.5. ***reduceColors*(*G*,*k*)**

Eingabe: Ein mit den k Farben $\{1, ..., k\}$ gefärbter Graph G mit n Knoten.

Ausgabe: Eine Färbung mit $k - 1$ Farben, soweit möglich.

`for each` $v \in V$: `if` color(v)=k $\Rightarrow$ color(v):=random(1,...,$k-1$)

`repeat`

· wähle den Knoten $v_0 \in V$ und die Farbe $c_0 \in \{1, .., k-1\}$, so dass f_0 eine neue Färbung mit $t(f_0) = \min\{f' : f'(v) = f(v)\ \forall v \neq v_0, f'(v) = c_0\}$ ist.

· `if` $t(f_0) = t(f) \Rightarrow$ wähle zufällig $\frac{n}{k}$ Knoten und färbe sie zufällig

`until` $t(f_0) = 0$ oder MAX erfolglose Versuche.

`if` $t(f_0) = 0 \Rightarrow$ `return` $k-1$ `else return` k

Es kann passieren, dass Algorithmus 2.5 in einer suboptimalen Lösung stagniert, d.h. man kann ausgehend von einer Färbung mit wenigen Konflikten durch die Farbänderung eines einzelnen Knotens keine Verbesserung erzielen. An dieser Stelle tritt die Randomisierung in

[2]Bei jedem Schritt wird ein Knoten gefärbt. Es müssen also nur seine Nachbarn aktualisiert werden.

Kraft, die jedem Knoten v eine Zufallszahl $r_v \in [0,1]$ zuweist. Wenn $r_v < \frac{1}{k}$ gilt, dann wird dem Knoten v eine neue Farbe aus $\{1,...,k-1\}$ zugewiesen. Im Erwartungswert erhalten also n/k Knoten eine neue Farbe. Die so entstehende Färbung beinhaltet wieder einige Konflikte, die mit dem Local Search-Verfahren abgebaut werden können. Wenn dies keinen Erfolg bringt (weil z.B. k zu klein wird), sichert die Schranke MAX, dass der Algorithmus abbricht.
Durch die Randomisierung wird gewährleistet, dass man nicht in einer Lösung festhängt, wie dies bei lokalen Optima der Fall sein kann. Algorithmus 2.5 ist kein exakter Algorithmus, er kann aber so modifizert werden, dass man einen exakten Algorithmus erhält: Man startet mit einer zulässigen Färbung (z.B. mit Algorithmus 2.1), und reduziert die Farben so lange, bis Algorithmus 2.5 keine Verbesserung mehr findet.
Schließlich gibt es noch einen exakten Färbungsalgorithmus von Lawler [30], der von Eppstein verbessert wurde [12]. Zunächst zur Idee von Lawler: Eine zulässige optimale Färbung eines k-färbbaren Graphen ist eine Partition der Knotenmenge V in k jeweils unabhängige Teilmengen, von denen mindestens eine maximal ist. Wenn man nun sämtliche maximalen unabhängigen Mengen bestimmt und diese Prozedur mit den jeweiligen Restgraphen rekursiv fortsetzt, kann man die chromatische Zahl eines Graphen mit n Knoten und m Kanten in der Zeit $O(mn(1+3^{1/3})^n) \approx O(m \cdot n \cdot 2.4422^n)$ bestimmen. Diese Grenze blieb (für allgemeines k) lange Zeit ungeschlagen (Lawler stellte den Algorithmus bereits 1976 vor). Im Jahr 2000 schließlich gelang es Eppstein, diese Schranke zu verbessern [12]. Er ging davon aus, dass es genügt, nur kleinere maximale unabhängige Mengen zu bestimmen, um zu einer korrekten Färbung zu kommen.

Satz 2.4. [12]. *Sei $G = (V,E)$ ein Graph mit n Knoten und k eine nichtnegative ganze Zahl. Dann gibt es in G höchstens $3^{4k-n} \cdot 4^{n-3k}$ maximale unabhängige Mengen $I \subseteq V$ mit $|I| \leq k$.*

Beweis. Induktion über n: Bei $n = 0$ gibt es genau eine maximale unabhängige Menge $I = \emptyset$. Das bedeutet, für jedes $k \geq 0$ ist $1 \leq 3^{4k} \cdot 4^{-3k} = \left(\frac{81}{64}\right)^k$. Nun gibt es in Abhängigkeit der Knotengrade in G verschiedene Fälle:

a) G enthält einen Knoten v mit $d(v) \geq 3$: Jede maximale unabhängige Menge I von G kann entweder v enthalten (womit $I \setminus \{v\}$ auch eine maximale unabhängige Menge von $G \setminus (N(v) \cup \{v\})$ ist) oder nicht. Damit ist nach Induktionsvoraussetzung die Anzahl

maximaler unabhängiger Mengen mit maximal k Knoten höchstens

$$3^{4k-(n-1)} \cdot 4^{(n-1)-3k} + 3^{4(k-1)-(n-4)} \cdot 4^{(n-4)-3(k-1)}$$
$$= \left(\frac{3}{4} + \frac{1}{4}\right) \cdot 3^{4k-n} \cdot 4^{n-3k} = 3^{4k-n} \cdot 4^{n-3k}.$$

b) G enthält einen Knoten v mit Grad 1: Jede maximale unabhängige Menge I muss entweder v oder seinen Nachbarn u enthalten. Wenn der jeweilige Knoten aus I entfernt wird, entsteht eine maximale unabhängige Menge von $G \setminus (N(u) \cup \{u\})$ bzw. $G \setminus (N(v) \cup \{v\})$. Sei $d(u) = d$, so ergibt sich

$$3^{4(k-1)-(n-2)} \cdot 4^{(n-2)-3(k-1)} + 3^{4(k-1)-(n-d-1)} \cdot 4^{(n-d-1)-3(k-1)}$$
$$= \left(\frac{4}{9} + 3^{d-3} \cdot 4^{2-d}\right) \cdot 3^{4k-n} \cdot 4^{n-3k} \leq \frac{8}{9} \cdot 3^{4k-n} \cdot 4^{n-3k}.$$

c) G enthält einen isolierten Knoten v. Dann ist v in jeder maximalen unabhängigen Menge enthalten, und es ergibt sich eine Schranke von

$$3^{4(k-1)-(n-1)} \cdot 4^{(n-1)-3(k-1)} = \frac{16}{27} \cdot 3^{4k-n} \cdot 4^{n-3k}.$$

d) G enthält eine Kette u-v-w-x paarweise verschiedener Knoten mit Grad 2. Jede maximale unabhängige Menge I kann nun entweder u oder v enthalten, oder es liegen weder u noch v, aber dafür w in I (dieser Fall tritt ein, wenn der andere Nachbar von u enthalten ist). Es entsteht dann eine obere Schranke von

$$2 \cdot 3^{4(k-1)-(n-3)} \cdot 4^{(n-3)-3(k-1)} + 3^{4(k-1)-(n-4)} \cdot 4^{(n-4)-3(k-1)} = \frac{11}{12} \cdot 3^{4k-n} \cdot 4^{n-3k}.$$

e) G erfüllt keinen der vorangestellten Fälle. Dann ist G nur aus disjunkten Dreiecken zusammengesetzt: Alle Knoten haben Grad 2, und es gibt keine Viererkette verschiedener Knoten. Dann haben alle maximalen unabhängigen Mengen genau $\frac{n}{3}$ Knoten, und es gibt genau $3^{n/3}$ maximale unabhängige Mengen. Ist nun $k \geq \frac{n}{3}$, so gilt $3^{n/3} \leq 3^{4k-n} \cdot 4^{n-3k}$, und wenn $k < \frac{n}{3}$ ist, dann gibt es gar keine maximalen unabhängigen Mengen mit höchstens k Knoten.

Somit liegen alle möglichen Fälle innerhalb der angegebenen Schranke. □

Mit diesem Ergebnis können alle maximalen unabhängigen Mengen mit höchstens k Knoten in der Zeit $O(3^{4k-n} \cdot 4^{n-3k})$ angegeben werden. Man muss nur der Analyse des Satzes 2.4 folgen und die Größe des Graphen verkleinern. Die maximale Rekursionstiefe von n verschwindet bei der exponentiellen Laufzeit des Verfahrens.

Mit dieser Voraussetzung präsentiert Eppstein nun einen Algorithmus zur Bestimmung der chromatischen Zahl eines Graphen $G = (V, E)$: Er führt ein Array X der Länge 2^n ein, welches sämtliche Teilmengen von V repräsentiert und die entsprechenden chromatischen Zahlen der induzierten Subgraphen speichert. Initialisiert wird das Array, indem für jede Teilmenge $S \subseteq V$ bestimmt wird, ob $\chi(G[S]) \leq 3$ ist. Ein Graph kann mit einem beliebigen exakten Algorithmus auf 3-Färbbarkeit getestet werden, z.B. dem von Eppstein selbst entwickelten (siehe [13]). Die Laufzeit ist $O(1.3289^{|S|})$ und damit kleiner als die Anzahl der Teilmengen von $V(G)$. Außerdem ist für $S \subset S'$ immer $\chi(S) \leq \chi(S')$; wenn also eine Knotenmenge S mit $\chi(S) > 3$ gefunden wurde, dann gilt auch $\chi(S') > 3$ für alle Obermengen $S' \supseteq S$. Für $\chi(S) \leq 3$ wird der entsprechende Wert eingetragen, sonst wird $X[S] := \infty$ gesetzt. Als nächstes werden alle Teilmengen von $V(G)$ in einer Reihenfolge wachsender Kardinalität besucht. Diese Reihenfolge soll sicherstellen, dass zuerst alle Teimengen von S besucht werden, bevor S selbst betrachtet wird. Wenn die Menge S an der Reihe ist und $3 \leq X[S] < \infty$ gilt, dann wird für alle maximalen unabhängigen Mengen $I \subseteq V \setminus S$ mit $|I| \leq \frac{|S|}{X[S]}$ der Eintrag $X[S \cup I]$ aktualisiert, also auf höchstens $X[S] + 1$ gesetzt.

Algorithmus 2.6. ***ChromaticNumber(G)***

Eingabe: Ein Graph $G = (V, E)$ mit n Knoten; X ist ein Array von 2^n Einträgen

Ausgabe: $\chi(G)$.

`for each` $S \subseteq V$

· `if` $\chi(G[S]) \leq 3 \Rightarrow X[S] := \chi(G[S])$

· `else` $X[S] := \infty$

`for each` $S \subseteq V$

· `if` $3 \leq X[S] < \infty$

... `for each` maximale unabhängige Menge $I \subseteq V(G \setminus S)$ mit $|I| \leq \frac{|S|}{X[S]}$:

..... $X[S \cup I] := \min\{X[S \cup I], X[S] + 1\}$

`return` $X[V]$.

Definition 2.5. Sei $G = (V, E)$ ein Graph. Eine Menge $S \subseteq V$ heißt *k-chromatisch*, wenn

$\chi(G[S]) = k$ gilt. Die Menge $S \subseteq V$ heißt *maximale* k-chromatische Menge, wenn $S \cup \{v\}$ für alle $v \in V \setminus S$ $(k+1)$-chromatisch ist.

Lemma 2.6. *Sei M eine maximale $(k+1)$-chromatische Teilmenge von G und (S, I) eine Partition von M in eine k-chromatische Menge S und eine unabhängige Menge I, so dass unter allen solchen Partitionen $|S|$ maximal ist. Dann ist I eine maximale unabhängige Menge von $G \setminus S$ mit $|I| \leq \frac{|S|}{k}$, und S ist eine maximale k-chromatische Teilmenge von G.*

Beweis. Bei einer $k+1$-Färbung von M erfüllt die Separation der kleinsten Farbklasse I von den anderen Farbklassen gerade die Ungleichung $|I| \leq \frac{|S|}{k}$; dies ist also auch wahr, wenn $|S|$ maximiert wird. Wäre I keine maximale unabhängige Menge, also $I \subsetneq I' \subseteq V(G) \setminus S$, dann würde $S \cup I'$ einen größeren $(k+1)$-chromatischen Graphen induzieren, was der vorausgesetzten Maximalität von M widerspricht.
Sei nun $S' \supsetneq S$ eine k-chromatische Menge. Ist $S' \cap I = \emptyset$, so würde $S' \cup I$ eine $(k+1)$-chromatische Obermenge von M bilden, was wiederum der Maximalität von M widerspricht. Ist andererseits $S' \cap I \neq \emptyset$, ergibt $(S' \cap M, S' \setminus I)$ eine bessere Partition als (S, I). Es ergibt sich also in jedem Fall ein Widerspruch. □

Lemma 2.7. *Sei M eine maximale $(k+1)$-chromatische Teilmenge von G. Beim Besuch von M in der Schleife von Algorithmus 2.6 gilt $X[M] = \chi(M)$.*

Beweis. Bei der Initialisierung des Arrays werden die Werte für $\chi(M) \leq 3$ korrekt gesetzt. Sei nun (S, I) wie in Lemma 2.6. Nach Induktion über $|M|$ ist $X[S] = \chi(S)$, da S selbst eine maximale k-chromatische Teilmenge von M ist. Es folgt weiter, dass $X[S] \geq 3$ ist und $|I| \leq \frac{|S|}{X[S]}$. Damit wird beim Besuch von S auch I betrachtet, und es wird $X[M]$ auf $X[S] + 1 = \chi(M)$ gesetzt. □

Satz 2.8. *Die chromatische Zahl $\chi(G)$ eines Graphen G mit n Knoten kann mit Algorithmus 2.6 in der Zeit $O\left(\left(\frac{4}{3} + \frac{3^{4/3}}{4}\right)^n\right)$ mit dem Platzbedarf $O(2^n)$ exakt bestimmt werden.*

Beweis. Algorithmus 2.6 ist korrekt: V ist selbst eine maximale $\chi(G)$-chromatische Teilmenge von G. Damit ist nach Lemma 2.7 $X[V] = \chi(G)$. Der Platzbedarf des Algorithmus ist offensichtlich die Größe des Arrays X, und somit $O(2^n)$.

Es bleibt noch die Laufzeitanalyse des Algorithmus. Die Initialisierung des Arrays X benötigt eine Gesamtzeit von

$$\sum_{S \subseteq V} O(1.3289^{|S|}) = O\left(\sum_{i=0}^{n} \binom{n}{i} 1.3289^i\right) = O((1 + 1.3289)^n) = O(2.3289^n),$$

da im schlechtesten Fall für jede Teilmenge ein exakter 3-Färbungsalgorithmus durchgeführt werden muss. Der beste bekannte 3-Färbungsalgorithmus stammt von Eppstein [13] und hat eine Laufzeit von $O(1.3289^n)$. Die Laufzeit der Hauptschleife von Algorithmus 2.6 setzt sich wie folgt zusammen:
Zur Konstruktion der maximalen unabhängigen Mengen von $V(G) \setminus S$ kann Satz 2.4 verwendet werden. Im schlechtesten Fall ist $X[S] = 3$, und die Größe der zu konstruierenden unabhängigen Mengen kann auf maximal $\frac{|S|}{3}$ festgelegt werden. Zur Bestimmung des Wertes $X[S \cup I]$ für jede Menge I wird jeweils eine konstante Zeit aufgewendet. Damit kann die Gesamtzeit mit

$$\sum_{S \setminus V(G)} O(3^{4^{\frac{|S|}{3}} - |V(G) \setminus S|} \cdot 4^{|V(G) \setminus S| - 3\frac{|S|}{3}}) = O\left(\sum_{i=0}^{n} \binom{n}{i} 3^{\frac{7i}{3} - n} \cdot 4^{n-2i}\right) = O\left(\left(\frac{4}{3} + \frac{3^{\frac{4}{3}}}{4}\right)^n\right)$$

nach oben beschränkt werden. □

Algorithmus 2.6 berechnet nur die chromatische Zahl $\chi(G)$ des Graphen. Eine zulässige $\chi(G)$-Färbung wird nicht mit angegeben. Hierfür genügt nach Abschluss des Algorithmus ein erneuter Durchlauf rückwärts durch das Array X:

Algorithmus 2.7. ***Eppstein*(*G*)**

Eingabe: Ein Graph G.

Ausgabe: Eine zulässige $\chi(G)$-Färbung von G.

Berechne das Array X mit Algorithmus 2.6

$S := V(G)$; $i := 1$

`for each` $T \subseteq V(G)$ *rückwärts*

· `if` ($T \subset S$ `and` $X[S \setminus T] = 1$ `and` $X[T] = X[S] - 1$)

... Färbe alle $v \in S \setminus T$ mit Farbe i.

... $i := i + 1$

... $S := T$

Satz 2.9. *Eine $\chi(G)$-Färbung eines Graphen G mit n Knoten kann in zusätzlicher Zeit $O(2^n)$ und ohne zusätzlichen Platz bestimmt werden.*

Beweis. Der Ressourcenbedarf von Algorithmus 2.7 ist klar: Zuerst wird Algorithmus 2.6 dazu benutzt, das Array X aufzubauen. Die weitere Arbeit bewegt sich auf diesem Array, benötigt also keinen weiteren Platz. Algorithmus 2.7 besteht im Wesentlichen aus einer Schleife über alle Teilmengen von $V(G)$, also 2^n Durchläufe. Es werden innerhalb eines Durchlaufs nur elementare Operationen durchgeführt, also bleibt die Laufzeit bei $O(2^n)$. Es bleibt, die Korrektheit des Algorithmus zu prüfen.
Eine Farbe wird einer Menge $S \setminus T$ nur dann zugewiesen, wenn $S \setminus T$ eine unabhängige Menge ist. Außerdem muss T mit genau einer Farbe weniger färbbar sein. Dies ist nur dann erfüllt, wenn $S \setminus T$ eine maximale unabhängige Menge ist und S somit in $(T, S \setminus T)$ partitioniert wird. □

Ein letztes exaktes Verfahren, welches hier vorgestellt wird, basiert auf sogenannten *kritischen Graphen*.

Definition 2.10. Ein Graph $G = (V, E)$ heißt *kritisch*, wenn für alle induzierten Subgraphen G' von G gilt: $\chi(G') \lneq \chi(G)$.

Herrmann und Hertz [21] nutzen folgende Tatsache aus, um die chromatische Zahl eines Graphen zu bestimmen:

Fakt 2.11. *Alle nicht-kritischen Graphen G enthalten einen induzierten kritischen Subgraph G' mit $\chi(G') = \chi(G)$.*

Der daraus resultierende Algorithmus ist eine Mischung aus einem Approximationsalgorithmus $H(G)$ (hierfür kann jede beliebige Heuristik angewendet werden, z.B. Algorithmus 2.2) und einem exakten Verfahren $E(G)$. Auch hier kann ein beliebiges Verfahren genutzt werden, z.B. Algorithmus 2.4.
Ziel ist es, den Graphen so weit zu reduzieren, dass ein exaktes Verfahren weniger Aufwand zur Bestimmung der chromatischen Zahl hat.
Der Algorithmus besteht aus 3 Phasen: der Initialisierung, dem Absteigen und der Zuwachsphase.

Definition 2.12. Sei $G = (V, E)$ ein Graph. Für einen Knoten $v \in V$ sei $G \setminus v$ definiert als $G[V \setminus \{v\}]$.

Algorithmus 2.8. ***colorCritical*(G)**

Eingabe: Ein Graph $G = (V, E)$.

Ausgabe: Eine optimale Färbung von G.

1. Bestimme eine obere Schranke $k = H(G) \geq \chi(G)$.
2. $G' := G$

   `for each` $v \in V$:

   · Bestimme eine obere Schranke $H(G' \setminus v)$ für $\chi(G' \setminus v)$

   · `if` $H(G' \setminus v) = k \Rightarrow G' := G' \setminus v$
3. Bestimme $\chi(G') = E(G')$ mit dem exakten Verfahren.

   `if` $\chi(G') = k \Rightarrow$ STOP: k ist die chromatische Zahl von G.
4. Setze $List := \emptyset$

   `for each` $v \in (V \setminus V(G'))$

   · Bestimme eine obere Schranke $H(G' \cup \{v\})$ für $\chi(G' \cup \{v\})$.

   · `if` $H(G' \cup \{v\}) > \chi(G') \Rightarrow$

   ... Bestimme $\chi(G' \cup \{v\}) = E(G' \cup \{v\})$

   ... `if` $\chi(G' \cup \{v\}) = \chi(G') + 1 \Rightarrow List := List \cup \{v\}$
5. `if` $List \neq \emptyset \Rightarrow$ wähle ein $v \in List$ und setze $G' := G' \cup \{x\}$

   `else` wähle ein $v \in G \setminus G'$ und setze $G' := G' \cup \{v\}$.
6. `if` $G' = G$ `or` $\chi(G') = k \Rightarrow$ STOP: $\chi(G')$ ist die chromatische Zahl von G.

   `else goto` 4.

Anmerkung: In Schritt 5 ist die chromatische Zahl des entstehenden Subgraphen einfach zu erkennen: Wurde ein Knoten aus *List* genommen und zu G' hinzugefügt, so erhöht sich die chromatische Zahl exakt um 1. Wurde ein anderer Knoten gewählt, so bleibt die chromatische Zahl konstant.

Fakt 2.13. *Sei k eine obere Schranke für die chromatische Zahl $\chi(G)$ eines Graphen G, und sei G' ein beliebiger Subgraph von G. Wenn $\chi(G') = k$ ist, dann ist auch $\chi(G) = \chi(G') = k$.*

Beweis. Es ist immer $\chi(G) \geq \chi(G')$, da G' ein Subgraph von G ist. Außerdem ist $k \geq \chi(G)$, da k eine obere Schranke für $\chi(G)$ ist. Es folgt damit $\chi(G) = \chi(G')$, wenn $\chi(G') = k$ ist. □

Fakt 2.14. *Sei $H(G)$ die in Schritt 1 von Algorithmus 2.8 berechnete obere Schranke für $\chi(G)$, und sei G' der am Ende von Schritt 2 resultierende Graph. Wenn $\chi(G') = H(G)$, dann ist G' ein kritischer Graph und hat dieselbe chromatische Zahl wie G.*

Beweis. Nach Fakt 2.13 gilt $\chi(G') = \chi(G)$. Der Subgraph G' ist durch das Entfernen sämtlicher Knoten entstanden, die die obere Schranke $H(G)$ nicht verändern. Es ist also gleichzeitig $H(G') = H(G)$ und $H(G' \setminus v) < H(G') \forall v \in V(G')$ erfüllt. Damit gilt

$$\chi(G' \setminus v) \leq H(G' \setminus v) < H(G') = H(G) = \chi(G) \; \forall v \in V(G').$$

Das bedeutet nach Definition 2.10, dass G' ein kritischer Graph ist. □

In Schritt 3 wird die chromatische Zahl von G' berechnet. Nach Fakt 2.13 kann der Algorithmus beendet werden, wenn $\chi(G') = H(G)$ ist. Nun kann es natürlich passieren, dass $\chi(G') < H(G)$ ist. In diesem Fall beginnt die Zuwachsphase.
Es werden alle Knoten $v \in V$ betrachtet, die nicht mehr in G' enthalten sind. Zunächst wird für einen Knoten v eine obere Schranke $H(G' \cup \{v\})$ für den Subgraphen berechnet, der entsteht, wenn v zu G' hinzugefügt wird. Wenn dieser Wert größer ist als $\chi(G')$, dann gibt es Grund zur Vermutung, dass auch die chromatische Zahl größer wird. Deshalb wird der exakte Färbungsalgorithmus angewendet, um diese Vermutung zu bestätigen. Wenn das Ergebnis $\chi(G' \cup \{v\}) = \chi(G') + 1$ ist, dann wird v in *List* eingefügt. Am Ende des Schrittes 4 hat jeder Knoten v in *List* die Eigenschaft, dass die chromatische Zahl um 1 wächst, wenn der Knoten v zum Graphen hinzugefügt wird; bei alle anderen Knoten wird die chromatische Zahl beibehalten. Die Schritte 4 und 5 werden so lange wiederholt, bis ein Subgraph $\hat{G}$ entsteht, so dass $\chi(\hat{G}) = H(G)$. Damit kann nach Fakt 2.13 der Algorithmus angehalten werden. Anderenfalls werden Knoten so lange zu G' hinzugefügt, bis $G' = G$ ist, und die chromatische Zahl $\chi(G)$ im Verlauf dieses Verfahrens bereits bestimmt wurde.
Es kann schließlich noch der Fall auftreten, dass der in Schritt 5 entstehende Graph G', der durch Hinzufügen eines Knotens aus *List* konstruiert wird, kein kritischer Graph ist, d.h. es können andere Knoten aus G' entfernt werden, ohne dass die chromatische Zahl geändert wird. Dies kann verhindert werden, indem innerhalb des Schrittes 5 noch eine vereinfachte Abstiegsphase eingefügt wird.

Der zuletzt präsentierte Algorithmus 2.8 hat einen entscheidenden Nachteil: Wenn die verwendete Heuristik $H(G)$ nicht sehr gut ist, dann wird bei der Zuwachsphase viel Rechenzeit darauf verwendet, eine exakte Färbung zu finden. Offensichtlich ist der Algorithmus nutzlos, wenn die verwendete Heuristik eine andere obere Abschätzung als $\chi(G)$ errechnet. Unter dieser Voraussetzung kann man genauso gut die betreffende Heuristik verwenden. Andererseits kann dieser Algorithmus in vielen Fällen die Arbeit sehr stark erleichtern, da das exakte Verfahren auf einen kleineren Graphen G' statt auf den Ausgangsgraphen G angewendet wird. Die worst-case-Laufzeit ist demnach $O(\sum_{i=n'}^{n} 2.3289^i) = O(2.3289^n)$, wobei n' die Größe des kleinsten im Algorithmus verwendeten Graphen G' bezeichnet. Es ergibt sich also im allgemeinen Fall kein Vorteil.

2.3 Approximation der chromatischen Zahl

Eine k-Färbung eines Graphen partitioniert die Knotenmenge in k unabhängige Mengen. Ein einfacher Algorithmus zur Färbung eines Graphen ist daher die Bestimmung von disjunkten unabhängigen Mengen. Der hier angeführte Algorithmus stammt von Johnson [23] und bietet eine Güte $\Gamma(\mathit{GreedyISColoring}) = O(\frac{n}{\log_k n})$.

Algorithmus 2.9. ***GreedyISColoring*(*G*)**

Eingabe: Ein Graph G.

Ausgabe: Eine Knotenfärbung von G.

1. $i := 1$
2. $W :=$ die Menge ungefärbter Knoten. `if` $W = \emptyset \Rightarrow$ STOP `else` $U := W$
3. $u :=$ ein Knoten minimalen Grades in $G[U]$. Färbe u mit Farbe i und setze $U := U \setminus (\{u\} \cup N(u))$
4. `if` $U = \emptyset \Rightarrow \{i := i + 1;$ `goto` $2\}$
   `else goto` 3.

Satz 2.15. *Algorithmus 2.9 verwendet zur Färbung eines k-färbbaren Graphen maximal $\lceil \frac{3n}{\log_k n} \rceil$ Farben.*

Beweis. Da der Graph G ein k-färbbarer Graph ist, muss eine unabhängige Menge der Größe $\frac{n}{k}$ in G enthalten sein. Aus dieser Beobachtung folgt, dass es einen Knoten mit einem Grad von

maximal $n \cdot \frac{k-1}{k}$ in G geben muss.[3] Nun wird Schritt 3 zu dem Zeitpunkt betrachtet, wo die Farbe i gerade neu zugewiesen wurde. Die Menge W hat beim ersten Durchlauf m Knoten und enthält eine unabhängige Menge der Größe $\frac{m}{k}$, da der Graph auf W ein k-färbbarer Graph ist. Es wird ein Knoten u minimalen Grades ($d(u) \leq m \cdot \frac{k-1}{k}$) gewählt und u und seine Nachbarschaft aus W entfernt. Diese Überlegung kann bei jeder Durchführung von Schritt 3 angestellt werden; daraus folgt nun, dass mindestens $\lfloor \log_k m \rfloor$ Knoten mit der Farbe i gefärbt werden können. Nun muss noch bestimmt werden, wie oft eine neue Farbe verwendet werden muss: Beim Start des Algorithmus ist der komplette Graph ungefärbt, d.h. $|W| = n$. Eine Farbe wird für mindestens $\log_k |W| > \frac{1}{2} \log_k n$ Knoten verwendet, wenn $|W| > \frac{n}{\log_k n}$ ist. Das heißt, in diesem Schritt werden maximal $\lceil \frac{2n}{\log_k n} \rceil$ Farben benötigt.
Nun sind nur noch $\lceil \frac{n}{\log_k n} \rceil$ ungefärbte Knoten übrig. Selbst wenn jeder Knoten eine andere Farbe erhält, so verwendet Algorithmus 2.9 doch nicht mehr als $\lceil \frac{n}{\log_k n} \rceil$ Farben. Zusammen mit der vorangestellten Überlegung entsteht also eine zulässige Färbung mit höchstens $\lceil \frac{3n}{\log_k n} \rceil$ Farben. □

Wigderson stellte einen Algorithmus vor, der die chromatische Zahl bis auf einen Faktor $3n(\frac{\log \log n}{\log n})^2$ approximiert [36]. Damit wird Algorithmus 2.9 von Johnson [23] um einen Faktor $O(\frac{(\log \log n)^2}{\log n})$ verbessert.
Ausgehend von einem Algorithmus für 3-färbbare Graphen (siehe [36] und Kapitel 3) erstellt er zunächst einen rekursiven Algorithmus für k-färbbare Graphen, indem er k als Eingabegröße verwendet. Es sei

$$f_k(n) = n^{1-\frac{1}{k-1}}. \tag{2.1}$$

Algorithmus 2.10. ***Wigderson1*(*k*,*G*,*i*)**

Eingabe: ein k-färbbarer Graph G, eine Zahl i, die angibt, bei welcher Farbe zu starten ist.

Ausgabe: Die Anzahl Farben, mit denen G gefärbt wurde.

1. $i_0 := i$

   `if` $k = 2 \Rightarrow$ Benutze Algorithmus 1.1, um G mit den Farben i und $i+1$ zu färben; `return` 2.

   `if` $k \geq \log n \Rightarrow$ Färbe G mit den Farben $i, i+1, ..., i+n-1$; `return` n.

[3] Hätte jeder Knoten einen größeren Grad als $n \cdot \frac{k-1}{k}$, dann enthält jeder Teilgraph der Größe $\frac{n}{k}$ mindestens eine Kante, und kann somit keine unabhängige Menge bilden.

2. `while` $\Delta(G) \geq \lceil f_k(n) \rceil$ `do`:
   - $\cdot$ Sei v ein Knoten mit $d(v) = \Delta(G)$
   - $\cdot$ $H := G[N(v)]$
   - $\cdot$ $j :=$ *Wigderson1*$(k-1,H,i)$
   - $\cdot$ Färbe v mit Farbe $i+j$
   - $\cdot$ $i := i+j$
   - $\cdot$ $G := G[V \setminus (N(v) \cup \{v\})]$
3. Färbe G mit den Farben $i, i+1, ..., i+s-1$; `return` $(i+s-i_0)$.

Satz 2.16. *Der Algorithmus 2.10 färbt für $k \geq 2$ jeden k-färbbaren Graphen mit n Knoten mit maximal $2k\lceil f_k(n) \rceil = 2k\lceil n^{1-\frac{1}{k-1}} \rceil$ Farben.*

Beweis. Der Beweis wird mit Induktion über k geführt:
$k = 2$: Bipartite Graphen werden mit $2 < 4n^{1-1/1} = 4$ Farben gefärbt.
$k \geq \log n$: Der Graph wird mit $n < 2kn^{1-\frac{1}{k-1}}$ Farben gefärbt.
$2 < k < \log n$: Die Behauptung sei für jeden k-färbbaren Graphen wahr, und es sei ein $(k+1)$-färbbarer Graph gegeben. Sei m die Anzahl der Schleifendurchläufe in Schritt 2. Wenn $m = 0$ ist, dann wurde G mit der Brute-Force-Strategie in Schritt 3 gefärbt. Dieser Schritt benötigte höchstens $\Delta(G) + 1 = \lceil f_{k+1}(n) \rceil \leq 2(k+1)\lceil f_{k+1}(n) \rceil$ Farben. Ist $m > 0$, dann seien $v_1, v_2, ..., v_m$ die jeweils ausgewählten Knoten, H_i die jeweils dazugehörigen Subgraphen und t_i die Anzahl der Knoten in H_i. Nach der Induktionsvoraussetzung wurden alle Graphen H_i zulässig gefärbt. Da jeweils verschiedene Farben verwendet wurden, gibt es auch keine Konflikte zwischen den jeweiligen Teilgraphen. Jeder Knoten v_i erhielt eine größere Farbe als all seine Nachbarn. Schließlich wurde noch Schritt 3 einmal mit einer neuen Farbmenge ausgeführt. Damit ist klar, dass die entstandene Färbung zulässig ist. Nun wird die Anzahl der verwendeten Farben bestimmt. Dazu dient ein bekannter Satz aus der konvexen Analysis:

Definition 2.17. ***Konkave Funktion.*** Sei $S \subseteq \mathbb{R}^n$ eine konvexe Menge. Eine Funktion $f : S \to \mathbb{R}$ ist konkav, wenn für jedes Paar $x, y \in S$ und $0 \leq \lambda \leq 1$ gilt: $\lambda f(x) + (1-\lambda) f(y) \leq f(\lambda x + (1-\lambda)y)$.

Satz 2.18. Ungleichung von Jensen. *Sei $S \subseteq \mathbb{R}^n$ und $f : S \to \mathbb{R}$ eine konkave Funktion, und seien $x_1, x_2, ..., x_m \in S$. Dann gilt:*

$$\frac{\sum_{i=1}^{m} f(x_i)}{m} \leq f\left(\frac{\sum_{i=1}^{m} x_i}{m}\right) \tag{2.2}$$

Wenn man die in Gleichung (2.1) definierten Funktionen $f_k(n)$ auf die reellen Zahlen $x > 0$ ausweitet, dann sieht man bei der 2. Ableitung, dass es sich um konkave Funktionen handelt. Weiterhin ist folgendes bekannt: Bei jedem Teilgraphen H_i mit t_i Knoten werden nach Induktionsvoraussetzung nie mehr als $2k\lceil t_i^{1-\frac{1}{k-1}}\rceil$ Farben verwendet, wobei gilt $\sum_{i=1}^{m} t_i \leq n$. Da $t_i = \Delta(G) \geq n^{1-1/k}\ \forall i$ gilt, muss $m < n^{1/k}$ sein.[4] Damit ergibt sich für die While-Schleife:

$$\begin{aligned}
\sum_{i=1}^{m} 2k\lceil t_i^{1-\frac{1}{k-1}}\rceil &< 2k\left(m + \sum_{i=1}^{m} t_i^{1-\frac{1}{k-1}}\right) \\
&= 2km\left(1 + \frac{\sum_{i=1}^{m} t_i^{1-\frac{1}{k-1}}}{m}\right) \\
&< (2k+1)m\frac{\sum_{i=1}^{m} t_i^{1-\frac{1}{k-1}}}{m} \qquad (2.3) \\
&\leq (2k+1)m\left(\frac{n}{m}\right)^{1-\frac{1}{k-1}}, \text{ nach (2.2)} \\
&= (2k+1)(nm^{\frac{1}{k-2}})^{1-\frac{1}{k-1}} \\
&< (2k+1)n^{1-\frac{1}{k}} \\
&\leq (2k+1)\lceil n^{1-\frac{1}{k}}\rceil.
\end{aligned}$$

Die Umformung zur Ungleichung (2.3) ist zulässig, wenn die Zahl in Klammern größer als $2k$ ist. Dies ist der Fall, wenn $k < \log n$ gilt. In Schritt 4 werden maximal $\lceil n^{1-1/k}\rceil$ Farben verwendet, womit sich die obere Schranke $2(k+1)\lceil n^{1-1/k}\rceil$ ergibt. □

Der Algorithmus 2.10 ist ein praktikabler Algorithmus, d.h. er ist so implementierbar, dass er eine Laufzeit von $O(k(|V|+|E|))$ besitzt. Für die genauen Spezifikationen der Implementation wird auf [36] verwiesen.

Algorithmus 2.10 besitzt einen entscheidenden Nachteil: Er nimmt an, dass die chromatische Zahl des Graphen bereits bekannt ist. Diese Information wurde allerdings nicht gegeben. Man kann aber Algorithmus 2.10 so modifizieren, dass er `nein` ausgibt, wenn er keine zulässige Färbung mit der aus den gegebenen Werten berechenbaren maximalen Farbanzahl findet (weil der angegebene Wert k zu klein war), und `ja`, wenn er eine solche Färbung findet. So kann mit binärer Suche ein geeigneter Parameter k bestimmt werden.

[4] In jedem Schleifendurchlauf i werden t_i Knoten entfernt. Nach m Durchläufen enthält der Restgraph also höchstens $n - m \cdot n^{1-1/k}$ Knoten.

Algorithmus 2.11. ***Wigderson2*(G)**

Eingabe: Ein Graph G.

Ausgabe: Eine zulässige Färbung von G.

1. Führe den Algorithmus *Wigderson1*$(k,G,1)$ (Algorithmus 2.10) mit $k = 2^l$, $l = 1, 2, ...$ aus, bis zum ersten Mal die Antwort `ja` kommt.
2. Suche binär nach dem kleinsten $k \in (2^{l-1}, 2^l]$, für das *Wigderson1*$(k,G,1)$ mit `ja` antwortet. Sei k_0 dieser Wert.
3. Färbe G mit der aus *Wigderson1*$(k_0,G,1)$ erhaltenen Färbung.

In Abänderung von Schritt 3 kann auch die beste erzielte Färbung verwendet werden. Die Analyse lässt sich aber nur für k_0 durchführen. Es zeigt sich, dass mit diesem Verfahren asymptotisch recht gute Erfolge erzielt werden können.

Lemma 2.19. $k_0 \leq \chi(G)$.

Beweis. Algorithmus *Wigderson1*$(k,G,1)$ (Algorithmus 2.10) atwortet immer dann mit `ja`, wenn er eine zulässige Färbung mit $2k\lceil f_k(n)\rceil$ Farben findet.
Der Beweis ist nun relativ einfach: Für *jedes* $k \geq \chi(G)$ wird *Wigderson1*$(k,G,1)$ nach Satz 2.16 mit `ja` antworten. Da $k_0 = \min\{k \in N :$ *Wigderson1*$(k,G,1)$ antwortet mit `ja`$\}$ ist, muss die Behauptung gelten, also $k_0 \leq \chi(G)$. □

Satz 2.20. *Für jeden Graphen $G = (V, E)$ mit n Knoten benutzt Algorithmus 2.11 maximal $2\chi(G)\left\lceil n^{1-\frac{1}{\chi(G)-1}}\right\rceil$ Farben.*

Beweis. Sei k_0 wie in Algorithmus 2.11 definiert. Sei $C(G)$ die verwendete Anzahl Farben. Nach Satz 2.16 ist $C(G) \leq 2k_0\left\lceil n^{1-\frac{1}{k_0-1}}\right\rceil$, da die Färbung von *Wigderson1*$(k_0,G,1)$ verwendet wird. Nach Lemma 2.19 ist $k_0 \leq \chi(G)$, womit $C(G) \leq 2\chi(G)\left\lceil n^{1-\frac{1}{\chi(G)-1}}\right\rceil$ gilt. □

Die Laufzeit des Algorithmus ist $O((|V| + |E|)\chi(G)\log\chi(G))$: Die binäre Suche, um den geeigneten Wert k_0 zu bestimmen, benötigt maximal $2\log\chi(G)$ Aufrufe des Algorithmus 2.10. Dieser hat jeweils eine Laufzeit von $O((|V| + |E|)\chi(G))$.

Korollar 2.21. *Für jedes feste k ist die Laufzeit von Algorithmus 2.11 auf Graphen G mit $\chi(G) \leq k$ linear.*

Algorithmus 2.9 aus [23] bzw. [36] hat eine Güte von $O(\frac{n}{\log n})$ für alle Graphen. Dieser Wert wird von Algorithmus 2.11 nur für Graphen mit relativ kleiner chromatischer Zahl verbessert, genauer für Graphen G mit $\chi(G) < \frac{\log n}{\log \log n}$. Wigderson konstruierte aus diesen beiden Algorithmen ein Hybridverfahren, welches die Färbung verwendet, die weniger Farben benutzt.

Algorithmus 2.12. ***WigdersonFinal*(*G*)**

Eingabe: Ein Graph G.

Ausgabe: Eine Färbung für G.

Färbe G mit Algorithmus 2.9

Färbe G mit Algorithmus 2.11

`return` die Färbung, die weniger Farben verwendet.

Satz 2.22. *Die Güte des Hybridalgorithmus 2.12 ist* $\Gamma(\text{WigdersonFinal}) \leq 6n\frac{(\log \log n)^2}{(\log n)^2}$.

Beweis. Sei G ein beliebiger k-färbbarer Graph mit n Knoten. Aus den Sätzen 2.15 und 2.20 ist bekannt, dass die Algorithmen 2.9 und 2.11 die Güten $f_A = \Gamma(\textit{GreedyISColoring}) \leq \frac{3n}{k \log_k n}$ und $f_B = \Gamma(\textit{Wigderson2}) \leq 2\lceil n^{1-\frac{1}{k-1}} \rceil$ erfüllen. Sei nun n fest und f_A und f_B seien Funktionen in Abhängigkeit von k. Es zeigt sich, dass f_A monoton fällt, während f_B monoton wächst. Es genügt also, einen Wert für k zu bestimmen, bei dem beide Funktionen nicht größer als $6n\frac{(\log \log n)^2}{(\log n)^2}$ werden. Dies ist bei $k = \lceil \frac{\log n}{2 \log \log n} \rceil$ der Fall. □

Die Idee der Splittung des Verfahrens nach dem maximalen Knotengrad ist grundlegend für die Färbungsalgorithmen aus den Kapiteln 3 und 4. So wird z.B. nur ein Algorithmus konstruiert, der für Graphen mit einem großen Knotengrad sehr gut ist, und dieses Verfahren wird dann mit einem bereits bekannten Algorithmus kombiniert.

Ein weiteres Verfahren wird aus folgender Überlegung hergeleitet:

Lemma 2.23. [19]. *Ein Algorithmus, der in einem k-färbbaren Graphen G garantiert, eine unabhängige Menge der Größe $f_k(n) = \Omega(\sqrt{n})$ zu finden, kann dazu verwendet werden, eine Färbung von G mit maximal $\frac{2n}{f_k(n)}$ Farben zu produzieren.*

Beweis. Der Algorithmus liefert eine unabhängige Menge, die mit einer Farbe gefärbt werden und dann aus dem Graphen G entfernt werden kann. Bei wiederholter Anwendung dieses

Verfahrens wird jede Farbe mindestens $f_k(n')$ Knoten zugewiesen, wobei n' die Anzahl der noch vorhandenen Knoten ist. In jedem Schritt erhält mindestens ein Knoten eine Farbe. Mit $\frac{1}{f_k(n')}$ Farben pro Knoten und der Annahme, dass mindestens ein Knoten pro Runde entfernt wird, ergibt sich eine maximale Farbanzahl $\hat{k}$ von

$$\hat{k} \leq \sum_{i=1}^{n} \frac{1}{f_k(i)}. \tag{2.4}$$

Wenn $f_k(n) = \Omega(n^{1/t})$ gilt, dann folgt aus Ungleichung (2.4):

$$\hat{k} \leq \frac{t \cdot n}{(t-1) \cdot f_k(n)}.$$

Für $t = 2$ folgt die Behauptung. □

Wenn man also einen Algorithmus kennt, der in einem k-färbbaren Graphen garantiert eine unabhängige Menge der Größe $\Omega(n^\alpha), 0 < \alpha < 1$, findet, dann kann durch wiederholte Anwendung dieses Algorithmus eine $O(n^{1-\alpha})$-Färbung produziert werden.
Zuerst gingen Boppana und Halldórsson diesen Weg, um sofort die beste bisher bekannte Schranke für die Färbung eines Graphen mit unbekannter chromatischer Zahl festzuschreiben [8, 19].
Boppana und Halldórsson stellten einen Algorithmus vor [8], der durch sukzessives Entfernen von Cliquen eine unabhängige Menge der Größe $\Omega\left(n^{\frac{1}{k-1}} (\log n)^{\frac{k-2}{k-1}}\right)$ findet. Die Konstruktion des Algorithmus basiert auf der Ramsey-Funktion $R(s,t)$.

Definition 2.24. Sei $R(s,t)$ die kleinste natürliche Zahl n, so dass alle Graphen mit n Knoten entweder eine Clique der Größe s oder eine unabhängige Menge der Größe t enthalten. Die Funktion $R(s,t)$ heißt *Ramsey*-Funktion.

Definition 2.25. Sei $G = (V,E)$ ein Graph und $v \in V$. Dann ist $\bar{N}(v) = V \backslash (N(v) \cup \{v\})$ die Menge der Nicht-Nachbarn von v.

Algorithmus 2.13. ***Ramsey(G)***

Eingabe: Ein Graph G.

Ausgabe: eine unabhängige Menge I und eine Clique C.

`if` $G = \emptyset$ `then return` $(\emptyset, \emptyset)$

wähle ein $v \in V$

(C_1, I_1) :=*Ramsey*$(G[N(v)])$

(C_2, I_2) :=*Ramsey*$(G[\bar{N}(v)])$

`return` $(\max\{C_1 \cup \{v\}, C_2\}, \max\{I_1, I_2 \cup \{v\}\})$

Der Algorithmus Ramsey liefert immer eine Clique C und eine unabhängige Menge I, so dass $R(|C|, |I|) \geq |V|$ gilt. Von Erdős und Szekeres [14] stammt eine obere Schranke für $R(s,t)$:

$$R(s,t) \leq \binom{s+t-2}{s-1}. \tag{2.5}$$

Eine Umkehrfunktion für $R(s,t)$ lautet $t_s(n) = \min\{t : R(s,t) \geq n\}$. Wenn der Graph G also keine Clique der Größe $k+1$ enthält, dann muss die gefundene unabhängige Menge eine Größe von mindestens $t_k(n)$ haben. Gleiches gilt natürlich für k-färbbare Graphen. Mit der Beziehung (2.5) kann $t_k(n)$ in der Größenordnung von $k \cdot n^{1/(k-1)}$ für $k \leq 2 \log n$ und $\frac{\log n}{\log \frac{k}{\log n}}$ für $k \geq 2 \log n$ angegeben werden, wie sich durch Einsetzen überprüfen lässt.
Algorithmus 2.13 findet allerdings nur dann eine große unabhängige Menge, wenn der Graph keine große Clique enthält. Wenn diese Bedingung nicht gegeben ist, dann kann über die Qualität der Lösung kaum eine Aussage getroffen werden. Was liegt also näher, als einige große Cliquen zu entfernen, um dann eine größere unabhängige Menge finden zu können?

Algorithmus 2.14. ***CliqueRemoval*(G)**

Eingabe: Ein Graph G

Ausgabe: Eine unabhängige Menge

$i := 1$

(C_i, I_i) :=*Ramsey(G)*

`while` $G \neq \emptyset$ `do`

$\cdot$ $G := G \setminus C_i$

$\cdot$ $i := i + 1$

$\cdot$ (C_i, I_i) :=*Ramsey(G)*

`return` I_0 mit $|I_0| = \max_{j=1..i} |I_j|$

Satz 2.26. *Sei $G = (V, E)$ ein Graph mit n Knoten, k die kleinste ganze Zahl, so dass $\alpha(G) > \frac{n}{k}$ und $\varepsilon = \frac{\alpha(G)}{n} - \frac{1}{k} > 0$. Sei $t_s(n)$ wie vorher definiert. Algorithmus 2.14 liefert eine unabhängige Menge $I \subseteq V$ der Größe $|I| \geq \max\{t_k(\varepsilon n), t_{k+1}(\frac{n}{k^2})\}$.*

Beweis. Der Algorithmus *CliqueRemoval* findet dann garantiert keine k-Clique mehr, wenn $\frac{\varepsilon}{1-1/k}n \geq \varepsilon n$ Knoten im Graphen verbleiben. Diese Beziehung ergibt sich aus der Voraussetzung, dass mehr als $\frac{n}{k}$ Knoten des Graphen eine unabhängige Menge bilden. Auf der anderen Seite wird der Algorithmus CliqueRemoval keine $(k+1)$-Clique finden, sobald $(\varepsilon + \frac{1}{k} - \frac{1}{k+1}) \cdot (\frac{1}{1-1/k})n \geq \frac{n}{k^2}$ Knoten im Graphen übrig bleiben. □

Korollar 2.27. [8]. *Gegeben sei G ein Graph mit n Knoten und der Unabhängigkeitszahl $\alpha(G) = tn \geq \frac{n}{\log n}$. Algorithmus 2.14 wird eine unabhängige Menge der Größe $e^{-1}\frac{n^t}{t}$ finden.*

Beweis. Nach Satz 2.26 gilt $|I| \geq t_k(\varepsilon n) \approx k(\varepsilon n)^{1/(k-1)}$, da $\frac{1}{k} = t \geq \frac{1}{\log n}$. Damit folgt $|I| \geq \frac{\varepsilon^{t-1}n^{t-1}}{t} \geq e^{-1}\frac{n^t}{t}$. □

Lemma 2.28. *Sei S ein Subgraph von G mit höchstens $\frac{n}{k} \cdot \frac{\log n}{2\log\log n}$ Knoten. Wenn S eine unabhängige Menge der Größe $\frac{n}{k}$ enthält, dann liefert Algorithmus 2.14, angewendet auf S, eine unabhängige Menge der Größe $\Omega\left(\frac{(\log n)^3}{(6\log\log n)}\right)$.*

Beweis. Mit $|S| \leq \frac{n\log n}{2k\log\log n}$ findet Algorithmus 2.14 nach Korollar 2.27 eine unabhängige Menge der Größe von mindestens

$$e^{-1}|S|^{\frac{(n/k)}{|S|}} \cdot \frac{|S|}{(n/k)} = e^{-1}\frac{k}{n}|S|^{\frac{n}{k|S|}+1} = e^{-1}\frac{k}{n}e^{\ln|S|(\frac{n}{k|S|}+1)}.$$

Dieser Ausdruck wird minimiert, wenn $|S|$ maximal ist, da die erste Ableitung im Bereich $\frac{n}{k} \leq |S| \leq \frac{n\log n}{2k\log\log n}$ negativ ist. Also folgt:

$$e^{-1}\frac{k}{n}\left(\frac{n\log n}{2k\log\log n}\right)^{\frac{2\log\log n}{\log n}+1} = e^{-1}\frac{\log n}{2\log\log n}\left(\frac{n\log n}{2k\log\log n}\right)^{\frac{2\log\log n}{\log n}} \geq e^{-1}\frac{\log^3 n}{2\log\log n}.$$

Die letzte Ungleichung ist für alle $n \geq 2$ und $k \leq \log n$ erfüllt, wie man durch Einsetzen überprüfen kann:

$$\begin{aligned} e^{-1}\frac{\log n}{2\log\log n}\left(\frac{n\log n}{2k\log\log n}\right)^{\frac{2\log\log n}{\log n}} &\geq e^{-1}\frac{\log n}{2\log\log n}\left(\frac{\log n}{2\log n\log\log n}\right)^{\frac{2\log\log n}{\log n}} \\ &= e^{-1}\frac{\log^3 n}{2\log\log n}(2\log\log n)^{-\frac{2\log\log n}{\log n}}. \end{aligned}$$

Mit $(2\log\log n)^{-\frac{2\log\log n}{\log n}} \leq 1$ für $n \geq 2$ folgt die Ungleichung. □

Angenommen, es sei ein k-färbbarer Graph G mit n Knoten gegeben, und die größte Farbklasse in einer zulässigen Färbung von G sei A. Weiterhin sei I eine Menge von $\log_k n$ zufällig gewählten Knoten. Dann kann Folgendes festgestellt werden:

Fakt 2.29. *Sei G ein k-färbbarer Graph mit n Knoten, und $A \subseteq V$ in einer zulässigen k-Färbung von G die größte Farbklasse. Sei weiterhin I eine Menge von $\log_k n$ zufällig gewählten Knoten. Dann ist*

1. *$I \subseteq A$ (und damit auch $A \subseteq I \cup \bar{N}(I)$) mit einer Wahrscheinlichkeit von mindestens $\frac{1}{n}$.*
2. *$\bar{N}(I)$ ist k-färbbar.*
3. *Wenn $\bar{N}(I)$ klein ist und $A \subseteq I \cup \bar{N}(I)$, dann findet Algorithmus 2.14 eine große unabhängige Menge in $\bar{N}(I)$.*

Beweis. Zunächst ist $|A| \geq \frac{n}{k}$. Daraus folgt, dass das Verhältnis der $\log_k n$-Teilmengen, die in A enthalten sind, zu allen $\log_k n$-Teilmengen mindestens $\frac{1}{n}$ ist.[5] Die zweite Beobachtung ist offensichtlich: Wenn man aus einem k-färbbaren Graphen einige Knoten und Kanten entfernt, dann ist er immer noch k-färbbar. Es kann also ein rekursiver Algorithmus gebildet werden. Die dritte Überlegung zielt auf Lemma 2.28 ab. Wenn der Algorithmus keine große Approximation für eine unabhängige Menge liefern kann, dann muss $|\bar{N}(I)|$ groß sein, was wiederum einen weiteren Rekursionsschritt ermöglicht. □

Nun kann Algorithmus 2.14 zum Auffinden einer großen unabhängigen Menge eingesetzt werden.

Algorithmus 2.15. ***SampleIS*(*G*,*k*)**

Eingabe: Ein k-färbbarer Graph G.

Ausgabe: Eine unabhängige Menge.

`if` $|V| \leq 1 \Rightarrow$ `return` G

`forever do`

· Wähle eine zufällige Knotenmenge I mit $|I| = \log_k n$ Knoten

· `if` I ist eine unabhängige Menge $\Rightarrow$

... `if` $|\bar{N}(I)| \geq \frac{n \log n}{2k \log \log n} \Rightarrow$ `return` $(I \cup SampleIS(\bar{N}(I),k))$

... `else`

..... $I_2 := CliqueRemoval(\bar{N}(I)) \cup I$

..... `if` $|I| \geq \frac{\log^3 n}{6 \log \log n} \Rightarrow$ `return` $(I \cup I_2)$

[5] Es gilt $\frac{\binom{n/k}{\log_k n}}{\binom{n}{\log_k n}} = \frac{O((n/k)^{\log_k n})}{O(n^{\log_k n})} = \Omega((\frac{1}{k})^{\log_k n}) = \Omega(\frac{1}{n})$.

Satz 2.30. *Gegeben sei ein k-färbbarer Graph G mit n Knoten. Algorithmus 2.15 findet in erwarteter Polynomialzeit eine unabhängige Menge I mit*

$$|I| \geq \frac{\log_k n \log n}{2 \max\{\log(k \cdot \frac{2 \log\log n}{\log n}), 1\}}. \tag{2.6}$$

Beweis. Der Beweis arbeitet mit Induktion über die Anzahl n der Knoten. Für $n = 1$ ist $k = 1$, und der Algorithmus gibt eine unabhängige Menge der Größe $1 > 0$ zurück. Sei die Behauptung nun für alle Graphen mit weniger als n Knoten erfüllt, d.h. in einem k-färbbaren Graphen G' mit $n' < n$ Knoten findet *SampleIS*(G') eine unabhängige Menge I der Größe $|I| = \frac{\log_k n' \log n'}{2 \max\{\log(k \cdot \frac{2 \log\log n'}{\log n'}), 1\}}$. Es ist zu zeigen, dass die Behauptung auch für Graphen mit genau n Knoten erfüllt ist:

Jeder Schleifendurchlauf des Algorithmus kann auf drei verschiedene Weisen enden (vorausgesetzt, I ist eine unabhängige Menge):

a) $|\bar{N}(I)| \geq \frac{n \log n}{2k \log\log n}$. Dann gibt nach Induktionsvoraussetzung *SampleIS*($\bar{N}(I)$,k) eine unabhängige Menge $\Im$ der Größe

$$|\Im| = \mathit{SampleIS}(\bar{N}(I)) \geq \frac{(\log |\bar{N}(I)|)^2}{2 \log k \max\{\log(k \cdot \frac{2 \log\log |\bar{N}(I)|}{\log |\bar{N}(I)|}), 1\}}$$

zurück. Mit $\frac{n \log n}{2k \log\log n} \leq |\bar{N}(I)| \leq n$ (und damit auch $\frac{2k \log\log n}{\log n} \geq 1$) ergibt sich

$$\begin{aligned}
|\Im| &\geq \frac{(\log \frac{n \log n}{2k \log\log n})^2}{2 \log k \max\{\log \frac{2k \log\log n}{\log n}, 1\}} \\
&= \frac{\log^2 n}{2 \log k \max\{\log \frac{2k \log\log n}{\log n}, 1\}} + \frac{\log \frac{2k \log\log n}{\log n} \cdot (\log \frac{2k \log\log n}{\log n} - 2 \log n)}{2 \log k \max\{1, \log \frac{2k \log\log n}{\log n}\}} \\
&= \frac{\log^2 n}{2 \log k \max\{\log \frac{2k \log\log n}{\log n}, 1\}} + \min\{\log \frac{2k \log\log n}{\log n}, 1\} \cdot \frac{\log \frac{2k \log\log n}{\log n} - 2 \log n}{2 \log k} \\
&\geq \frac{\log^2 n}{2 \log k \max\{\log \frac{2k \log\log n}{\log n}, 1\}} - \frac{\log n}{\log k}.
\end{aligned}$$

Die Größe von I ist $\log_k n = \frac{\log n}{\log k}$. Die Menge $\Im \cup I$ besitzt also $\frac{\log^2 n}{2 \max\{\log \frac{2k \log\log n}{\log n}, 1\}}$. Knoten und ist eine unabhängige Menge.

b) *CliqueRemoval* gab eine unabhängige Menge der Größe $\frac{\log^3 n}{(6 \log\log n)}$ zurück. Dann ist die Behauptung für $k \geq 3$ erfüllt.

c) $|\bar{N}(I)| < \frac{n \log n}{2k \log \log n}$, und *CliqueRemoval* war nicht erfolgreich. Dann war I nicht in der größten Farbklasse des Graphen enthalten.

Nach im Erwartungswert n Durchläufen wird $I \subseteq A$ gelten (siehe Fakt 2.29), was zu einer der beiden Möglichkeiten a) oder b) führen wird. □

Satz 2.31. Halldórsson [19]. *Das Problem* Graph Coloring *ist mit* $O\left(n\frac{(\log \log n)^2}{(\log n)^3}\right)$ *approximierbar, d.h. die chromatische Zahl kann bis auf einen Faktor* $O\left(n\frac{(\log \log n)^2}{(\log n)^3}\right)$ *approximiert werden.*

Beweis. Algorithmus 2.15 kann für alle Werte von k gestartet werden, bis eine unabhängige Menge gefunden wird, die mindestens so groß ist wie die von *SampleIS*(G,$\chi(G)$). Gleichzeitig wird Algorithmus 2.14 benutzt, um letztendlich die größte gefundene unabhängige Menge auszugeben. Wenn $\chi \leq \frac{\log n}{2 \log \log n}$ ist, dann wird nach Lemma 2.23 die Anwendung von *CliqueRemoval* eine Färbung mit $O(\frac{n^{1-1/\chi}}{\chi})$ Farben konstruieren. Andererseits wird bei $\chi \geq \frac{\log n}{2 \log \log n}$ mit *SampleIS* eine Färbung mit $O\left(\frac{n \log \chi \max\{\log(\chi \frac{2 \log \log n}{\log n}),1\}}{(\log n)^2}\right)$ Farben produziert. Diese beiden Ergebnisse werden nun ins Verhältnis zu χ gesetzt sowie n und k fixiert. Die erste Funktion, $\frac{n^{1-1/\chi}}{\chi^2}$, ist monoton fallend und die zweite Funktion, $\frac{n \log \chi \max\{\log(\chi \frac{2 \log \log n}{\log n}),1\}}{\chi(\log n)^2}$, wächst monoton mit χ. Somit genügt es, den maximal erreichbaren Funktionswert zu betrachten. Dieser liegt gerade bei $\chi = \frac{\log n}{2 \log \log n}$, und beide Funktionen haben dort einen Wert von $O(n\frac{(\log \log n)^2}{(\log n)^3})$. □

Die Approximationsalgorithmen von Wigderson [36] und Halldórsson [8, 19] versuchen zuerst, die chromatische Zahl des Graphen zu bestimmen: Ein Algorithmus wird mit verschiedenen Parametern für die chromatische Zahl gestartet, bis das Ergebnis herauskommt, welches der Algorithmus mit der richtigen chromatischen Zahl erhalten hätte. Dies führt einerseits zu den exakten Algorithmen aus dem letzten Abschnitt, die allerdings nicht effizient durchführbar sind (P≠NP vorausgesetzt), andererseits zu speziellen Algorithmen für k-färbbare Graphen, die in Kapitel 4 näher betrachtet werden.

Kapitel 3

Resultate für 3-färbbare Graphen

Fast alle Approximationsalgorithmen für k-färbbare Graphen arbeiten rekursiv mit $k-1$-färbbaren Graphen, um gute Ergebnisse zu erzielen. Daher ist es notwendig, einen guten Rekursionsanfang zu liefern. Ein Färbungsalgorithmus für bipartite Graphen ist mit Algorithmus 1.1 bereits vorgestellt worden; in diesem Kapitel folgen nun die besten Algorithmen für 3-färbbare Graphen.

3.1 Analyse der Ergebnisse

In [4] wurde bereits ein Überblick über Approximationsalgorithmen für 3-färbbare Graphen gegeben. Vier Algorithmen werden hier noch einmal kurz umschrieben. Das Ziel dieser Algorithmen ist die Färbung eines 3-färbbaren Graphen mit $\tilde{O}(n^{\alpha_3})$ Farben. Nach und nach wurde der Wert α_3 von $\frac{1}{2}$ auf $\frac{3}{14}$ verbessert. Am Ende steht ein Hybridalgorithmus, der eben diese $\tilde{O}(n^{3/14})$ Farben verwendet, um einen 3-färbbaren Graphen zu färben.

Der erste Algorithmus für 3-färbbare Graphen stammt von Wigderson [36]. Er legte den Grundstein für eine $O(\sqrt{n})$-Färbung eines 3-färbbaren Graphen. Das Verfahren besteht aus zwei Teilen. Solange der Graph einen großen maximalen Knotengrad besitzt, kommt der 2-Färbungsalgorithmus 1.1 zur Anwendung. Hierbei wird ein Knoten maximalen Grades gewählt und seine Nachbarschaft mit zwei Farben gefärbt. Sobald allerdings der maximale Knotengrad einen bestimmten Schwellwert unterschreitet, werden die übriggebliebenen Knoten mit einer einfachen Strategie (z.B. mit dem Greedy-Algorithmus 2.1) gefärbt.

Die exakten Schwellwerte lassen sich durch analytische Überlegungen bestimmen: Sei G ein 3-färbbarer Graph mit n Knoten, sowie Δ der Schwellwert, bei dem auf den Greedy-Algorithmus gewechselt wird. In der ersten Phase werden jeweils die Nachbarn von Knoten mit einem

Grad von mindestens Δ gefärbt. Damit erhalten jeweils mindestens Δ Knoten immer genau 2 Farben. Es werden dadurch maximal $\frac{2n}{\Delta}$ Farben verwendet. Die Anzahl der verwendeten Farben in der zweiten Phase richtet sich nach dem maximalen Knotengrad des Restgraphen. Da dieser auf Δ beschränkt ist, werden hier maximal Δ Farben verwendet. Die minimale Anzahl der Farben erhält man nun durch Gleichsetzen der beiden Summanden Δ und $\frac{2n}{\Delta}$. Dies ergibt einen Schwellwert von $\Delta = \sqrt{2n}$. Hier folgt nun der Algorithmus im Einzelnen:

Algorithmus 3.1. ***Wigderson3-Color*(*G*)**

Eingabe: Ein 3-färbbarer Graph G mit n Knoten.

Ausgabe: Eine $O(\sqrt{n})$-Färbung von G.

1. $i := 1$
2. `while` $\Delta(G) \geq \lceil\sqrt{2n}\rceil$ `do:`
   - Wähle v mit $d(v) = \Delta(G)$
   - $H := G[N(v)]$
   - Färbe H mit 2 Farben $i, i+1$
   - Färbe v mit $i+2$
   - $i := i+2$
   - $G := G[V \setminus (N(v) \cup \{v\})]$
3. $(\Delta(G) < \lceil\sqrt{2n}\rceil)$. Färbe G mit Algorithmus 2.1.

Satz 3.1. [36, 4, 6]. *Algorithmus 3.1 färbt jeden 3-färbbaren Graph mit n Knoten mit maximal $\sqrt{8} \cdot \sqrt{n}$ Farben.*

Beweis. In der ersten Phase werden nach den obigen Überlegungen maximal $\frac{2n}{\sqrt{2n}} = \sqrt{2n}$ Farben verwendet, in der zweiten Phase noch einmal maximal $\sqrt{2n}$. Dies ergibt eine maximale Farbanzahl von $2 \cdot \sqrt{2n} = \sqrt{8n}$. □

Für den weiteren Fortgang werden noch einige Definitionen benötigt:

Definition 3.2. Sei $G = (V, E)$ ein Graph. Seien weiterhin $S, T \subseteq V$ und $v \in V$. Dann ist $D(S) = \sum_{v \in S} d(v)$ der Grad von S, $d_T(v) = |N(v) \cap T|$ der Grad von v in die Menge T und $D_T(S) = \sum_{v \in S} d_T(v)$ die Summe der Knotengrade von Knoten aus S in die Menge T.[1]

[1] Wenn $S \cap T = \emptyset$ gilt, dann ist $D_T(S)$ die Anzahl der Kanten zwischen S und T. Anderenfalls werden alle Kanten, die innerhalb von $S \cap T$ verlaufen, doppelt gezählt.

Blum [6] gab verschiedene Algorithmen an, wie man einen 3-färbbaren Graphen mit möglichst wenig Farben färben kann. Er nutzte dazu hauptsächlich Lemma 2.23, welches besagt, dass man mit einer unabhängigen Menge der Größe $f(n)$ einer $O(\frac{n}{f(n)})$-Färbung näherkommen kann. Daraus und aus einigen anderen Prozeduren (wie z.B. aus einem Approximationsalgorithmus für unabhängige Mengen in Graphen ohne kurze ungerade Kreise [32]) stellte Blum einen Algorithmus zusammen, der 3-färbbare Graphen mit n Knoten mit $O(n^{3/8}\log^{8/5} n)$ Farben färbt. Hierzu benötigte er einen Algorithmus, der in einem Graphen eine große unabhängige Menge findet, wenn bekannt ist, dass etwa die Hälfte der Knoten eine unabhängige Menge bildet. Dies leistet der Algorithmus von Monien und Speckenmeyer [32]:

Algorithmus 3.2. ***BE/MS*(G)**

Eingabe: Ein Graph $G = (V, E)$.

Ausgabe: Eine unabhängige Menge I.

1. Entferne die Knoten aller Kreise mit ungerader Länge $\leq 2l + 1$, wobei $l = \lfloor \frac{\log n - 3}{6} \rfloor$ gilt.
2. $I := \emptyset$

   Wähle ein $v \in V$.
3. Setze $V_i := \{u \in V : dist(u, v) = i\}, i = 0, ..., l$
4. Setze $S_i := \bigcup_{k=0}^{\lfloor i/2 \rfloor} V_{i-2k}, i = 0, ..., l.$
5. Suche i_0 mit $|N(S_{i_0})| \leq n^{1/(l+1)}|S_{i_0}|$
6. $I := I \cup S_{i_0}$

   $V := V \setminus (S_{i_0} \cup N(S_{i_0}))$

   $n := |V|$

   `if` $n \neq 0 \Rightarrow$ `goto` 3

   `else return` I

Einen Kreis ungerader Länge t erkennt man folgendermaßen: Sei $A \in \{0, 1\}^{n \times n}$ die Adjazenzmatrix von G. Dann ist A^t die Matrix mit den Wegen der Länge t, d.h. der Eintrag $A^3_{i,j}$ gibt an, wie viele Wege der Länge 3 es von Knoten i zu Knoten j gibt. Das bedeutet, auf der Hauptdiagonalen stehen sämtliche Dreiecke des Graphen G. Sei nun $G' := G[\{i \in V : A^3_{i,i} = 0\}]$ der Subgraph von G, der keine Dreiecke induziert. Nun kann induktiv fortgefahren werden: Sei

$H = (V, E)$ ein Graph, der keine Kreise mit ungerader Länge von maximal t enthält. Dann ist $H' := H[\{i \in V : A_{i,i}^{t+2} = 0\}]$ ein Graph, der keine ungeraden Kreise mit höchstens $t + 2$ Knoten enthält.

Im endgültigen Verfahren wird der Algorithmus 3.2 auf einen Teilgraphen des zu färbenden Graphen angewendet, von dem bekannt ist, dass etwa die Hälfte seiner Knoten eine unabhängige Menge bilden. Diese Bedingung stellt einerseits sicher, dass nach dem ersten Schritt (Entfernung der kurzen Kreise ungerader Länge) noch genügend Knoten übrig bleiben, und andererseits, dass eine unabhängige Menge der Größe $\Omega(\frac{n}{\log n})$ gefunden wird, wenn der Teilgraph anfangs n Knoten besaß [6].

Wie bereits in Lemma 2.23 angedeutet, kann man mit einer großen unabhängigen Menge eine gute Färbung erlangen. Blum [6] vereinfachte diese Formulierung, indem er nicht unbedingt ständig eine neue unabhängige Menge finden muss. Er weist in einem Graphen Eigenschaften nach, die es ermöglichen, einen *Fortschritt* für einen Färbungsalgorithmus zu erzielen. Die Idee ist folgende: Wenn man in einem Graphen nacheinander Fortschritte für eine $O(f(n))$-Färbung erzielen kann, dann kann man den kompletten Graphen mit $O(f(n))$ Farben färben. Der erste Fortschritt zielt auf Lemma 2.23 ab: Wenn im Graphen eine unabhängige Menge der Größe $\frac{n}{f(n)}$ gefunden wird, dann wurde ein Fortschritt für eine $O(f(n))$-Färbung erzielt: Die unabhängige Menge wird gefärbt und der Färbungsalgorithmus startet von neuem mit einem kleineren Graphen und neuen Farben. Das entsprechende Verfahren wird *Progress-1* genannt. Blum gab noch zwei weitere Arten von Fortschritt an, die aber kaum ins Gewicht fallen. *Progress-2* zielt darauf ab, wiederholt disjunkte unabhängige Mengen mit jeweils kleiner Nachbarschaft zu suchen. Die Vereinigungen dieser Mengen ergeben dann eine große unabhängige Menge, und das Verfahren *Progress-1* ist wieder anwendbar. Als letzten Fortschrittstyp führt Blum *Progress-3* ein: Wenn zwei Knoten in jeder zulässigen 3-Färbung des Graphen dieselbe Farbe erhalten müssen, dann kann man diese beiden Knoten verschmelzen und mit einem kleineren Graphen fortfahren, ohne eine Farbe verwendet zu haben.

Die genannten drei Fortschrittstypen kommen hauptsächlich zum Tragen, wenn der Graph relativ dünn besetzt ist, d.h. wenige Kanten besitzt. Da es aber auch 3-färbbare Graphen mit relativ vielen Kanten geben kann (bzw. wenn ein Graph sogenannte *dichte* Regionen besitzt), ist es erforderlich, auch in dichten Regionen einen Fortschritt zu erzielen. Dazu dient der Algorithmus *Dense-Region-Progress* mit seinen Prozeduren *LargeSet*, *SharedNeighborhood*

und *SmallIS*.

Zunächst folgt die Prozedur *LargeSet*(G, S). Sie dient dazu, in einer mittelgroßen Menge S festzustellen, ob es eine zulässige 3-Färbung des Graphen gibt, so dass fast alle Knoten der Menge S dieselbe Farbe erhalten. Wenn dies der Fall ist, dann kann ein Fortschritt erzielt werden, anderenfalls wird eine Garantie zurückgegeben, dass nicht so viele Knoten dieselbe Farbe erhalten können. Für die Details wird auf [6] verwiesen.

Algorithmus 3.3. ***LargeSet*(G,S)**

Eingabe: Ein Graph $G = (V, E)$ mit n Knoten und eine Menge $S \subseteq V$ mit mindestens $\frac{n \log^2 n}{(f(n))^2}$ Knoten.

Ausgabe: *Fortschritt* oder *Garantie*, dass in jeder zulässigen 3-Färbung von G weniger als $(1 - \frac{1}{4 \log n})|S|$ Knoten in S dieselbe Farbe haben.

1. $G' := G[S]; S' := S$;
2. `While` S' ist nicht unabhängig:
   - Wähle $\{u, v\} \in E(G')$;
   - $S' := S' \setminus \{u, v\}; G' := G'[S']$
3. `if` $|S'| < |S|(1 - \frac{1}{2 \log n})$ `return` Garantie
4. `if` $|N(S')| \leq \frac{n \log^2 n}{f(n)} \Rightarrow$ `return` *Progress-2*
5. $T := N(S'); s_0 := |S'|$
6. `for each` $w \in T$: markiere eine Kante von w nach S'.
   Setze $N'(v) := \{w \in T : \{v, w\}$ ist markiert$\}$ $\forall v \in S'$.
7. Setze $S_i := \{v \in S' : |N'(v)| \in [2^i, 2^{i+1})\},\ i = 0, 1, ...$
8. Bestimme i_0 mit $|N'(S_{i_0})| = max_i\{|N'(S_i)|\}$
9. $I :=$*BE/MS*$(N'(S_{i_0}))$; (Algorithmus 3.2)
   `if` $|I| \geq \frac{n}{f(n)} \Rightarrow$ `return` *Progress-1*
10. $S' := S' \setminus S_{i_0}; T := N(S')$. `if` $|S'| \geq \frac{s_0}{3} \Rightarrow$ `goto` 6.
11. `return` Garantie.

Wenn zwei Knoten u, v viele gemeinsame Nachbarn besitzen, dann kann man mit der Prozedur *SharedNeighborhood* einen Fortschritt erzielen: Sei $S = N(u) \cap N(v)$ die gemeinsame Nachbarschaft von u und v. Besitzt $N(S)$ relativ wenige Knoten, dann erzielt man mit *Progress-2*

einen Fortschritt. Ist dagegen die Menge $N(S)$ relativ groß und 2-färbbar, so führt *Progress-1* zum Erfolg. Ist die Menge $N(S)$ schließlich nicht 2-färbbar, dann müssen die Knoten u und v in jeder zulässigen 3-Färbung des Graphen dieselbe Farbe erhalten. Wenn u nämlich mit Farbe 1 gefärbt wäre und v mit Farbe 2, dann wäre S einfarbig mit Farbe 3 gefärbt. Die Nachbarschaft von S wäre dann bipartit. Wenn also der Versuch fehlschlägt, die Menge $N(S)$ mit zwei Farben zu färben, ergibt sich automatisch der Fortschritt *Progress-3*.

Algorithmus 3.4. ***SharedNeighborhood*(*G*,*u*,*v*)**

Eingabe: Ein 3-färbbarer Graph G mit n Knoten und zwei Knoten u, v, die mindestens $\frac{n}{f(n)}^2$ gemeinsame Nachbarn haben.
Ausgabe: *Fortschritt* für eine $O(f(n))$-Färbung von G.

Setze $S := N(u) \cap N(v)$
`if` $|N(S)| \leq \frac{n}{f(n)} \Rightarrow$ *Progress-2*
`else if` $N(S)$ ist 2-färbbar $\Rightarrow$ *Progress-1*
`else` *Progress-3*

Die dritte Prozedur, *SmallIS*, verwendet eine relativ kleine unabhängige Menge. Man kann damit garantieren, dass entweder eine kleinere unabhängige Menge (mit der Größe $\frac{n}{f(n)^2}$) bereits einen Fortschritt bringt, oder nicht alle Knoten der unabhängigen Menge können dieselbe Farbe erhalten.

Algorithmus 3.5. ***SmallIS*(*G*,*S*)**

Eingabe: Ein 3-färbbarer Graph G mit n Knoten und eine unabhängige Menge S mit $|S| \geq \frac{n}{f(n)}^2$.
Ausgabe: *Fortschritt* für eine $O(f(n))$-Färbung von G oder *Garantie*, dass nicht alle Knoten dieselbe Farbe erhalten.

`if` $N(S)$ ist 2-färbbar `and` $|N(S)| \geq \frac{n}{f(n)} \Rightarrow$ *Progress-1*
`else if` $|N(S)| \leq \frac{n}{f(n)} \Rightarrow$ *Progress-2*
`else return` Garantie.

Nun kann der Algorithmus *Dense-Region-Progress* vorgestellt werden. Als Eingabe werden zwei Knotenmengen S und T verwendet, die zusätzlich noch einige Eigenschaften erfüllen

müssen. So muss z.B. die Menge S 2-färbbar sein, und zwischen S und T sollten viele Kanten verlaufen. Damit bilden die Mengen S und T eine so genannte dichte Region.

Algorithmus 3.6. ***Dense-Region-Progress*(*G*,*S*,*T*)**

Eingabe: Ein Graph $G = (V, E)$ mit n Knoten, eine 2-färbbare Menge S und eine Menge T mit $(D_T(S))^3 \geq (|S| + \max_{v \in S} d_T(v)) \cdot (|S||T|^2 \frac{n \log n}{(f(n))^2} + |S|^2|T|\frac{n^2}{(f(n))^4})$ und $D_T(S) \geq |S|\frac{n \log^2 n}{(f(n))^2}$.

Ausgabe: *Fortschritt* für eine $O(f(n))$-Färbung von G.

1. `for each` $v \in S$: `if` $|N(v) \cap T| \geq \frac{n \log^2 n}{(f(n))^2} \Rightarrow$ *LargeSet*(G,$N(v) \cap T$)
2. `if` $\exists u, v \in S : |N(u) \cap N(v)| \geq \frac{n}{(f(n))^2} \Rightarrow$ *SharedNeighborhood*(G,u,v)
3. `for each` $v \in T$:
   - $Y := N(N(v) \cap S) \cap T$;
   - $Z := \{w \in S : d_Y(w) \geq \frac{n}{f(n)^2}\}$;
   - `if` Z ist unabhängig $\Rightarrow$ *SmallIS*(G,Z)
   - `for each` $w \in Z$: `if` $(Y \cap N(w))$ ist unabhängig $\Rightarrow$ *SmallIS*(G,$Y \cap N(w)$)

Nun wird behauptet, dass der Algorithmus 3.6 immer einen Fortschritt erzielt. Dies ist zum Einen in den Eigenschaften der Mengen S und T begründet. Nach Blum [6] wird mindestens eine der Mengen Z und $Y \cap N(w)$ für $w \in Z$ komplett einfarbig sein, was dazu führt, dass die entsprechende Nachbarschaft 2-färbbar ist. Die Prozedur *SmallIS* wird dann auf jeden Fall einen Fortschritt erzielen.

Mit diesen Prozeduren kann nun der Algorithmus von Blum vorgestellt werden (in [6] wird der Algorithmus *Improved-Approx* genannt).

Definition 3.3. Gegeben sei ein (3-färbbarer) Graph $G = (V, E)$ mit n Knoten. Es sei $\delta := \delta(n) = \frac{1}{5 \log n}$.

Algorithmus 3.7. ***Blum3-Color*(*G*)**

Eingabe: Ein 3-färbbarer Graph G mit n Knoten.

Ausgabe: Eine $O(n^{3/8} \log^{8/5} n)$-Färbung von G.

1. Setze $f(n) = n^{3/8} \log^{8/5} n$.

2. `for each` $v \in V$: `if` $d(v) < f(n) \Rightarrow$ *Progress-2*
3. `for each` $v \in V; i, j \in \{0, 1, ..., 5 \log^2 n\}$:
   · Setze $I_j := \{v \in V : d(v) \in [(1+\delta)^j, (1+\delta)^{j+1})\}$
   · Setze $S = S_{v,j} := N(v) \cap I_j$
   · Setze $T_{v,i,j} := \{v \in N(S) : d_S(v) \in [(1+\delta)^j, (1+\delta)^{j+1})\}$
   · `if` $|T_{v,i,j}| \geq \frac{n^{5/8}}{\log^{3/2} n} \Rightarrow I :=$*BE/MS*$(T_{v,i,j})$; (Algorithmus 3.2)
   ... `if` $|I| \geq \frac{n}{f(n)} \Rightarrow$ *Progress-1*
   · `if` $(D_T(S))^3 \geq (|S| + \max_{v \in S} d_T(v)) \cdot (|S||T|^2 \frac{n \log n}{(f(n))^2} + |S|^2 |T| \frac{n^2}{(f(n))^4})$ `and` $D_T(S) \geq |S| \frac{n \log^2 n}{f(n)^2} \Rightarrow$ *Dense-Region-Progress*

In wenigen Worten ausgedrückt, sucht der Algorithmus nach guten Eigenschaften der Knoten, um eine Färbung vorzunehmen. Die Verifikation von Algorithmus 3.7 zeigt dann noch, dass das Verfahren immer korrekt arbeitet, d.h. dass immer mindestens eine der Voraussetzungen gegeben ist, um einen Fortschritt zu erzielen [6, 4].

Satz 3.4. [6, 4]. *Algorithmus 3.7 färbt jeden 3-färbbaren Graphen mit $O(n^{3/8} \log^{8/5} n)$ Farben.*

Einen ganz anderen Weg schlugen Karger, Motwani und Sudan [24] ein. Sie nutzten eine Repräsentation, die der θ-Funktion von Lovasz [31] sehr ähnlich ist. Hierzu sind einige Definitionen aus der Vektoranalysis notwendig:

Definition 3.5. Seien $u, v \in \mathbb{R}^n$ Vektoren. Dann ist $\langle u, v \rangle = \sum_{i=1}^{n} u_i \cdot v_i$ das *Skalarprodukt* von u und v und $\|u\| = \sqrt{\langle u, u \rangle}$ der *Betrag* von u. Speziell heißt ein Vektor u mit $\|u\| = 1$ *Einheitsvektor*.

Definition 3.6. Gegeben sei ein k-färbbarer Graph $G = (V, E)$. Eine *Vektor-k-Färbung* von G ist eine Funktion $f \colon V \to \mathbb{R}^n$ mit $\|f(v)\| = 1$ und $\langle f(v), f(w) \rangle \leq -\frac{1}{k-1}$ für alle Kanten $\{v, w\} \in E$. Eine *Matrix-k-Färbung* von G ist eine symmetrische, positiv semidefinite Matrix $M = (m_{ij})$ mit $m_{ii} = 1 \; \forall i$ und $m_{ij} \leq -\frac{1}{k-1}$, wenn $\{v_i, v_j\} \in E$ ist.

Hier folgen nun noch einige Eigenschaften von Vektor- und Matrix-Färbungen. Zunächst soll gezeigt werden, dass es für jeden k-färbbaren Graphen G mit n Knoten auch eine Vektor-k-Färbung gibt. Offensichtlich muss dafür nur gezeigt werden, dass es im n-dimensionalen reellen Raum immer $k \leq n + 1$ Einheitsvektoren gibt, so dass das Skalarprodukt von jeweils zwei

dieser Vektoren maximal $-\frac{1}{k-1}$ ergibt. Danach können die k Farben des Graphen eineindeutig den k Einheitsvektoren zugeordnet werden.

Der Basisfall $k = 2$ ist einfach: man nimmt zwei diametral gegenüberliegende Vektoren, z.B. $(1, 0, ..., 0)$ und $(-1, 0, ..., 0)$. Das Skalarprodukt der beiden Vektoren ist $-1 = -\frac{1}{k-1}$, unabhängig von n.

Seien nun k Vektoren $v_1, ..., v_k$ gegeben, so dass $\langle v_i, v_j \rangle \leq -\frac{1}{k-1}$ für $i \neq j$ gilt. Aus diesen Vektoren werden nun die Vektoren $u_1, ..., u_{k+1}$ wie folgt konstruiert:

$$u_i := \left(\frac{\sqrt{(k-1)(k+1)}}{k} v_i^1, ..., \frac{\sqrt{(k-1)(k+1)}}{k} v_i^k, -\frac{1}{k} \right),$$

wobei v_i^j die j-te Komponente von v_i bezeichnet. Hinzu kommt der Vektor $u_{k+1} := (0, ..., 0, 1)$. Die neuen Vektoren sehen also genauso aus wie die alten, sie wurden nur um eine Komponente erweitert und skaliert, so dass wieder Einheitsvektoren entstehen. Offensichtlich ist das Skalarprodukt von u_{k+1} mit jedem der ersten k Vektoren genau $-\frac{1}{k}$. Es bleibt nur noch zu zeigen, dass auch die Vektoren $u_1, ..., u_k$ untereinander den geforderten Wert $-\frac{1}{k}$ nicht überschreiten. Es gilt

$$\begin{aligned} \langle u_i, u_j \rangle &= \sum_{i=1}^{k} u_i + \frac{1}{k^2} \\ &= \frac{(k-1)(k+1)}{k^2} \langle v_i, v_j \rangle + \frac{1}{k^2} \\ &= \frac{(k-1)(k+1) \cdot (-1)}{k^2 \cdot (k-1)} + \frac{1}{k^2} \\ &= -\frac{k+1}{k^2} + \frac{1}{k^2} = -\frac{1}{k}. \end{aligned}$$

Nun ist also eine Vektor-k-Färbung $v_1, ..., v_n$ gegeben. Man kann diese in eine Matrix-k-Färbung umwandeln. Hierzu ist $M \in \mathbb{R}^{n \times n}$ die Matrix mit den Einträgen $m_{i,j} = \langle u_i, u_j \rangle$. Die Matrixeinträge auf der Hauptdiagonale sind offensichtlich gleich 1, da es sich ausschließlich um Einheitsvektoren handelt.

Ebenso lässt sich eine Matrix-k-Färbung in eine Vektor-k-Färbung umwandeln. Die Cholesky-Zerlegung einer symmetrischen, positiv semidefiniten Matrix M liefert eine Matrix U, so dass $UU^T = M$ gilt. Da auf der Hauptdiagonale der Matrix-Färbung nur Einsen stehen, bilden die Zeilen der Matrix U jeweils Einheitsvektoren. Das bedeutet, dass die Matrix U gerade eine Vektor-k-Färbung ergibt.

Der letzte Schritt von der Vektor-k-Färbung zur echten k-Färbung ist leider nicht möglich. Es können Beispiele für Graphen gefunden werden, deren Vektor-chromatische Zahl deutlich unter ihrer chromatischen Zahl liegt (s. Seite 69).
Zur Berechnung der Matrix-k-Färbung eines Graphen $G = (V, E)$ gibt es ein ein semidefinites Optimierungsprobem:

$$\begin{aligned} \text{minimiere} \quad & \alpha \quad \text{mit} \\ M = (m_{ij}) \quad & \in \quad \mathbb{R}^{n \times n} \text{ symmetrisch, positiv semidefinit} \\ m_{ii} \quad & = \quad 1 \\ m_{ij} = m_{ji} \quad & \leq \quad \alpha \; \forall \{v_i, v_j\} \in E. \end{aligned} \tag{3.1}$$

Es gibt verschiedene Methoden, um ein solches Problem in Polynomialzeit beliebig genau zu lösen [27]. Das bedeutet, es kann in Polynomialzeit eine Matrix-k-Färbung (und damit auch eine Vektor-k-Färbung) eines k-färbbaren Graphen gefunden werden.
Nun stellt sich noch die Frage, wie ausgehend von den Vektoren $f(v)$ eine zulässige Knotenfärbung entstehen soll. Hierfür haben Karger, Motwani und Sudan eine randomisierte Rundungstechnik entwickelt:

Definition 3.7. Seien $c_1, ...c_t \in \mathbb{R}^n$ die Ortsvektoren zufällig gewählter Punkte (*Zentren*) und $a \in \mathbb{R}^n$ ein beliebiger Vektor. Das Zentrum c_i *fängt* den Vektor a, wenn für alle $j \neq i$ gilt: $\langle c_j, a \rangle < \langle c_i, a \rangle$.

Die t zufälligen Zentren werden nun mit jeweils einer Farbe identifiziert. Jeder Knoten v, dessen in der Vektor-k-Färbung zugeordneter Vektor u_v vom Zentrum c_i gefangen wird, erhält dann die Farbe i.
Mit einer solchen Technik kann es aber immer noch sein, dass ein adjazentes Knotenpaar von demselben Zentrum gefangen wird und damit dieselbe Farbe erhält. Wenn dieses Ereignis nicht allzu häufig auftritt, dann kann man aber immer noch von einer *Semifärbung* des Graphen sprechen.

Definition 3.8. Eine *k-Semifärbung* eines Graphen $G = (V, E)$ mit n Knoten ist eine nicht zulässige k-Färbung, so dass maximal $\frac{n}{4}$ Kanten zu jeweils gleich gefärbten Knoten inzident sind.

Wenn man nun den korrekt gefärbten Teil des Graphen unverändert lässt und für die restlichen Knoten die Färbungsprozedur wiederholt, dann entsteht aus einer t-Semifärbung eine zulässige $(t \log n)$-Färbung des Graphen. Um die erwartete Anzahl nicht zulässig gefärbter Knoten auf das gewünschte Maß zu reduzieren, muss nur noch der Parameter t bestimmt werden. Bei geeigneter Wahl ([24, 4]) erhält man für $t = \Delta(G)^{1/3} \log^{4/3} \Delta(G)$ eine $(t \log n)$-Färbung, also eine Färbung mit $\Delta(G)^{1/3} \log^{4/3} \Delta(G) \log n$ Farben. Damit wurde ein Algorithmus konstruiert, der speziell auf Graphen mit kleinem maximalem Grad ausgelegt ist. Kombiniert mit dem ersten Teil von Algorithmus 3.1 kann die Anzahl der verwendeten Farben in Abhängigkeit von n beschrieben werden: Zunächst wird der maximalen Grad des Graphen mit Hilfe von Algorithmus 3.1 auf einen Schwellwert d reduziert. Dann startet der Rundungsalgorithmus, um eine $O(t \log n)$-Färbung des Restgraphen zu erhalten. Man erhält ein Verfahren, das mit $O(\frac{n}{d} + t \log n)$ Farben auskommt. Durch Gleichsetzen ergibt sich der Wert $d = \frac{n^{3/4}}{\log n}$, und mit $t = t(d)$ entsteht ein $O(n^{1/4} \log n)$-Färbungsalgorithmus.

Algorithmus 3.8. ***KMS3-Color*(*G*)**

Eingabe: Ein 3-färbbarer Graph G.

Ausgabe: Eine $O(n^{1/4} \log n)$-Färbung von G.

$i := 1$;

`while` $\Delta(G) \geq \frac{n^{3/4}}{\log n}$

· Wähle $v \in V$ mit $d(v) = \Delta(G)$.

· Färbe $N(v)$ mit Farben $i, i+1$ und v mit $i+2$.

· $G := G[V \setminus (N(v) \cup \{v\})]$

· $i := i + 2$

Finde eine Matrix-3-Färbung M von G mit Aufgabe (3.1).

Extrahiere mittels der Cholesky-Zerlegung aus M eine Vektor-3-Färbung $f(v)$.

`while` G ist noch nicht komplett gefärbt:

· Wähle $t = \Delta^{1/3} \log^{4/3} \Delta$ zufällige Punkte $c_1, ..., c_t \in \mathbb{R}^n$.

· `for each` $v \in V$: Setze $c(v) := j$ mit $\langle c_j, f(v) \rangle > \langle c_l, f(v) \rangle \forall l \neq j$.

· $S := \{v \in V : \exists w \in V : \{v, w\} \in E \wedge c(v) = c(w)\}$;

· `for each` $v \in V \setminus S$: färbe v mit Farbe $i + c(v)$

· $G := G[S]$; $i := i + t$.

Satz 3.9. [24, 4]. *Algorithmus 3.8 färbt jeden 3-färbbaren Graphen mit $O(n^{1/4} \log n)$ Farben.*

Der letzte Algorithmus für 3-färbbare Graphen stammt von Blum und Karger [7] und geht auf die beiden vorangestellten Algorithmen 3.7 und 3.8 zurück. Man kann feststellen, dass Algorithmus 3.8 auf Graphen mit kleinem maximalem Grad spezialisiert ist. Knoten mit großem Grad werden am Anfang des Algorithmus durch eine andere Technik beseitigt. Dagegen hat Algorithmus 3.7 eher bei Graphen mit großem maximalem Grad eine Chance durch die Prozedur *Dense-Region-Progress*. Die Idee, die hinter dem neuen Algorithmus steckt, ist folgende:
Eine $O(n^c)$-Färbung eines Graphen mit n Knoten mit $c \in (0, 1]$ enthält eine unabhängige Menge mit $\Omega(n^{1-c})$ Knoten, die durch die größte Farbklasse definiert ist. Wenn also ein Teilgraph von G mit $\Theta(n)$ Knoten so gefärbt werden kann, dass eine zulässige Färbung mit $O(n^c)$ Farben entsteht, dann entsteht dadurch ein Fortschritt vom Typ *Progress-1* für eine $\tilde{O}(n^c)$-Färbung des kompletten Graphen.

Korollar 3.10. [7, 4]. *Seien $c \in \mathbb{R}$ und $0 < \rho < 1$. Sei $\boldsymbol{A}$ ein Algorithmus, der für jeden 3-färbbaren Graphen mit n Knoten und einem durchschnittlichen Grad von cn^ρ einen Fortschritt für eine $O(n^\alpha)$-Färbung erzielt. Dann kann jeder 3-färbbare Graph mit $\tilde{O}(\max\{n^{\rho/3}, n^\alpha\})$ Farben gefärbt werden.*

Beweis. Wenn der gegebene Graph G einen durchschnittlichen Grad von mindestens n^ρ hat, dann wird Algorithmus **A** angewendet. Im anderen Fall gibt es mindestens $n/2$ Knoten mit einem Grad von höchstens $2cn^\rho$. Damit kann der entsprechende Subgraph mit $\tilde{O}(n^{\rho/3} \log n)$ Farben gefärbt werden (Algorithmus 3.8) und es wird mit *Progress-1* ein Fortschritt für eine $\tilde{O}(n^{\rho/3})$-Färbung des gesamten Graphen erzielt. □

Durch Anwendung verschiedener Sätze [7, 4] erzielt man eine Färbung mit $\tilde{O}(n^{3/14})$ Farben:

Algorithmus 3.9. ***Best-3-Color*(*G*)**
Eingabe: Ein 3-färbbarer Graph G.
Ausgabe: Eine $\tilde{O}(n^{3/14})$-Färbung von G.

`if` $\bar{d}(G) \leq n^{9/14} \Rightarrow$

· $S := \{v \in V : d(v) \leq 2n^{9/14}\}$; $H := G[S]$

· *KMS3-Color*(H)

· $I :=$ größte Farbklasse in H

· *Progress-1*

`for each` $v \in V; i, j \in \{0, 1, ..., 5 \log^2 n\}$:

· Setze $I_j := \{v \in V : d(v) \in [(1+\delta)^j, (1+\delta)^{j+1})\}$

· Setze $S_{v,j} := N(v) \cap I_j$

· Setze $T_{v,i,j} := \{v \in N(S) : d_S(v) \in [(1+\delta)^j, (1+\delta)^{j+1})\}$

· `if` $|T_{v,i,j}| \geq n^{11/14} \Rightarrow I :=$*BE/MS*$(T_{v,i,j})$

... `if` $|I| \geq \frac{n^{11/14}}{\log n} \Rightarrow$ *Progress-1*

· `if` $D_T(S) \geq |S| n^{8/14} \log^2 n$ `and`
$(D_T(S))^3 \geq (|S| + \max_{v \in S} d_T(v)) \cdot (|S||T|^2 n^{8/14} \log n + |S|^2 |T| n^{16/14}) \Rightarrow$ *Dense-Region-Progress*

Im Wesentlichen wird der Algorithmus 3.7 mit $f(n) = \tilde{O}(n^{3/14})$ verwendet. Der Graph G soll mit $f(n)$ Farben gefärbt werden. Aus den Sätzen 8, 9 und 10 aus [7] folgt ein Algorithmus, der jeden 3-färbbaren Graphen mit n Knoten mit einem durchschnittlichen Grad von mindestens n^ρ mit maximal $n^\alpha = n^{\frac{3}{5}(1-\rho)}$ Farben färbt. Nach Korollar 3.10 entsteht nun ein Algorithmus, der jeden 3-färbbaren Graphen mit $\tilde{O}(\max\{n^{\frac{3}{5}(1-\rho)}, n^{\frac{\rho}{3}}\})$ Farben färbt. Mit $\rho = 9/14$ folgt der angegebene Algorithmus 3.9.[2]

Satz 3.11. [7, 4]. *Algorithmus 3.9 färbt jeden 3-färbbaren Graphen mit $\tilde{O}(n^{3/14})$ Farben.*

Algorithmus 3.9 von Blum und Karger ist der bisher beste bekannte Algorithmus für 3-färbbare Graphen. Eine leichte Verbesserung konnten Halperin, Nathaniel und Zwick erzielen [20], aber nicht durch neue Techniken, sondern durch eine verfeinerte Analyse des Algorithmus 3.8. Dort gelang ihnen eine Verbesserung im polylogarithmischen Bereich: Statt $O((\Delta^{1-2/k} \log^{1/2} \Delta) \cdot \log n)$ Farben für einen k-färbbaren Graphen G erhalten sie mit $\bar{d}$ als durchschnitlichem Grad des Graphen eine Färbung mit $O((\bar{d}^{1-2/k} \log^{1/k} \bar{d}) \log n$ Farben. Da sich aber im Hauptterm nichts ändert, kann man nur von einer polylogarithmischen Verbesserung sprechen. Die Schranke $\tilde{O}(n^{3/14})$ für 3-färbbare Graphen bleibt also bestehen.

[2] Durch Gleichsetzen erhält man $\rho/3 = \frac{3}{5}(1-\rho)$ und damit $\rho = 9/14$.

3.2 Erweiterungsmöglichkeiten auf k-färbbare Graphen

Der Algorithmus von Karger, Motwani und Sudan [24], Algorithmus 3.8, kann leicht auf alle k-färbbaren Graphen ausgedehnt werden. Die Definition einer Vektor-k-Färbung (Definition 3.6) bleibt schließlich bestehen. Eine genaue Analyse der Gütegarantie für eine solche Erweiterung wird in Kapitel 4 angegeben.
Der Algorithmus von Blum [6] dagegen nutzt die Eigenschaft, dass der gegebene Graph 3-färbbar ist und somit eine 2-Färbung der Nachbarschaft eines Knotens leicht gefunden werden kann. Nichtsdestotrotz bietet Blum in [6] auch einen Algorithmus an, der k-färbbare Graphen zulässig färbt. Auch dieser Algorithmus findet sich in Kapitel 4.
Eine Möglichkeit, einen k-färbbaren Graphen zu färben, baut auf einem einfachen Prinzip auf:

Fakt 3.12. *Gegeben ist ein k-färbbarer Graph $G = (V, E)$. Für jeden Knoten $v \in V$ gilt: $G[N(v)]$ ist $k-1$-färbbar.*

Beweis. Sei im Gegensatz zur Behauptung die Nachbarschaft $N(v)$ k-chromatisch. Dann gibt es für jede Farbe $c \in \{1, ..., k\}$ mindestens einen Knoten $w \in N(v)$ mit der Farbe c. Um nun eine zulässige Färbung zu erhalten, muss v die Farbe $k+1$ erhalten. Damit kann G nicht k-färbbar sein. □

Ein einfacher rekursiver Algorithmus sieht demnach so aus:

Algorithmus 3.10. ***kColor*(*G*,*k*)**
Eingabe: Ein k-färbbarer Graph G.
Ausgabe: Eine Färbung von G.

`if` $k = 2 \Rightarrow$ `return` 2-Färbung von G
`while` $E(G) \neq \emptyset$:
· Wähle einen Knoten v mit $d(v) = \Delta(G)$
· *kColor(*$N(v)$,$k-1$*)*
· $G := G[V(G) \setminus N(v)]$
Färbe die übriggebliebene unabhängige Menge mit einer Farbe.

Algorithmus 3.10 ist im Wesentlichen identisch mit Algorithmus 2.10 von Wigderson [36]. Der einzige Unterschied ist, dass Wigderson einen Wert für den maximalen Grad festlegt, ab dem ein Graph mit einer anderen Strategie gefärbt wird. Die Analyse von Algorithmus 2.10 befindet sich in Kapitel 2 bzw. in [36].

Das Prinzip der rekursiven Färbung eines Graphen wird auch von Blum [6] aufgegriffen. Er sucht sich (analog zu Wigderson [36]) einen Basisfall, den man effizient behandeln kann. Bei Wigderson war dies ein 2-färbbarer Graph; Blum kann nun auf einen Algorithmus zurückgreifen, der einen 3-färbbaren Graphen mit relativ wenig Farben färben kann.

Allgemein kann angenommen werden, dass es einen Algorithmus **A** gibt, der jeden k_0-färbbaren Graphen mit n Knoten mit $O(cn^\alpha \log^\beta n)$ Farben färbt, wobei $\alpha, \beta, c \in \mathbb{R}$ Konstanten sind. Ein Algorithmus für k-färbbare Graphen, $k > k_0$, sieht dann so aus:

Algorithmus 3.11. ***Recursive-Color*(*G*,*k*)**

Eingabe: Ein k-färbbarer Graph $G = (V, E)$ und ein Algorithmus **A**, der jeden k_0-färbbaren Graphen mit höchstens $cn^\alpha \log^\beta n$ Farben färbt.

Ausgabe: Eine zulässige Färbung von G.

1. `if` $k = k_0 \Rightarrow$ Algorithmus **A**
2. Setze $r := k_0 - \frac{1}{1-\alpha}$
   $f(n,k) := n^{1-\frac{1}{k-r}} (\log n)^{\frac{\beta}{(1-\alpha)(k-r)}}$
3. `while` $\exists v \in V : d(v) \geq f(n,k)$:
   · Wähle $N'(v) \subseteq N(v)$ mit $|N'(v)| = f(n,k)$;
   · *Recursive-Color*$(G[N'(v)]$,$k-1)$
   · $G := G[V \setminus N'(v)]$
4. Färbe G mit $f(n,k)$ Farben. $(\Delta(G) < f(n,k))$

Satz 3.13. *Sei* ***A*** *ein Algorithmus, der jeden k_0-färbbaren Graphen mit n Knoten mit maximal $c \cdot n^\alpha \cdot \log^\beta n$ Farben färbt. Dann kann Algorithmus 3.11 jeden k-färbbaren Graphen $(k \geq k_0)$ mit maximal*

$$C_k(n) = (c + (k - k_0)) \cdot n^{1-\frac{1}{k-r}} \cdot (\log n)^{\beta \frac{k_0 - r}{k-r}} \tag{3.2}$$

Farben färben. Hierbei ist $r = r(\boldsymbol{A}) = k_0 - \frac{1}{1-\alpha}$ abhängig von Algorithmus ***A****.*

Beweis. Der Beweis funktioniert mit Induktion über k:
Im Falle $k = k_0$ gilt $C_{k_0} = c \cdot n^{1-\frac{1}{1/(1-\alpha)}} \cdot (\log n)^\beta = c \cdot n^\alpha \cdot \log^\beta n$. Sei nun $c_k = c + k - k_0$ und $f(n,k) = n^{\frac{k-r-1}{k-r}} \cdot (\log n)^{\beta\frac{k_0-r}{k-r}}$ wie in Algorithmus 3.11. Nach der Induktionsvoraussetzung ist $C_{k-1}(n) \leq c_{k-1} \cdot f(n, k-1)$. Es ist nun zu zeigen, dass auch $C_k(n) \leq c_k \cdot f(n,k)$ gilt:
In Schritt 3 von Algorithmus 3.11 werden bei jedem Durchlauf genau $f(n,k)$ Knoten entfernt. Das bedeutet, dass dieser Schritt höchstens $\frac{n}{f(n,k)}$-mal durchgeführt werden kann. Somit ergibt sich die Rekursion

$$C_k(n) \leq C_{k-1}(f(n,k)) \cdot \frac{n}{f(n,k)} + f(n,k).$$

Wenn diese Rekursion aufgelöst wird, dann sieht man

$$\begin{aligned}
C_k(n) &\leq \left(c_{k-1} \cdot f(n,k)^{\frac{k-r-2}{k-r-1}} \cdot (\log f(n,k))^{\beta\frac{k_0-r}{k-r-1}}\right) \cdot \frac{n}{f(n,k)} + f(n,k) \\
&< c_{k-1} \cdot f(n,k)^{\frac{k-r-2}{k-r-1}} \cdot (\log n)^{\beta\frac{k_0-r}{k-r-1}} \cdot \frac{n}{f(n,k)} + f(n,k) \\
&= c_{k-1} \cdot n \cdot f(n,k)^{\frac{-1}{k-r-1}} \cdot (\log n)^{\beta\frac{k_0-r}{k-r-1}} + f(n,k) \\
&= c_{k-1} \cdot n \cdot \left(n^{\frac{k-r-1}{k-r}}\right)^{\frac{-1}{k-r-1}} \cdot \left((\log n)^{\beta\frac{k_0-r}{k-r}}\right)^{\frac{-1}{k-r-1}} \cdot (\log n)^{\beta\frac{k_0-r}{k-r-1}} + f(n,k) \\
&= c_{k-1} \cdot n^{1-\frac{1}{k-r}} \cdot (\log n)^{\beta\frac{k_0-r}{k-r}} + f(n,k) \\
&= c_{k-1} \cdot f(n,k) + f(n,k) \\
&= c_k f(n,k).
\end{aligned}$$

Damit ist die in Formel (3.2) angegebene Eigenschaft gezeigt. □

Mit Satz 3.13 und einem guten Approximationsalgorithmus für k_0-färbbare Graphen kann man nun auch k-färbbare Graphen effizient färben, mit $k > k_0$.
Wenn als Basisfall $k_0 = 2$ gewählt wird und Algorithmus **A** 2-färbbare Graphen mit 2 Farben färbt (Algorithmus 1.1), dann entsteht mit $k_0 = 2, \alpha = 0$ und $r(\mathbf{A}) = 1$ ein rekursiver Algorithmus, der jeden k-färbbaren Graphen mit $C_k(n) = k \cdot n^{1-\frac{1}{k-1}}$ Farben färbt. Dies entspricht genau der Güte von Algorithmus 2.11 aus [36].
Nun kann als Basisalgorithmus der beste bekannte Algorithmus für 3-färbbare Graphen, Algorithmus 3.9 aus [7] genutzt werden. Dann entsteht mit $k_0 = 3, \alpha = \frac{3}{14}$ und $r(\mathbf{A}) = 3 - \frac{14}{11} = \frac{19}{11}$ ein Algorithmus mit $\tilde{O}(n^{1-\frac{1}{k-19/11}})$ Farben für einen k-färbbaren Graphen.
Es ergeben sich für kleine Werte von k die in Tabelle 3.1 ablesbaren Ergebnisse, die mittels Algorithmus 3.11 aus den Algorithmen des Abschnitts 3.1 abgeleitet werden können.

	k =3	4	5	6	7
Algorithmus 2.11	1/2	2/3	3/4	4/5	5/6
[36]	0.5	0.667	0.75	0.8	0.833
Algorithmus 3.7	3/8	8/13	13/18	18/23	23/28
[6]	0.375	0.615	0.722	0.783	0.821
Algorithmus 3.8	1/4	4/7	7/10	10/13	13/16
[24]	0.25	0.571	0.7	0.769	0.8125
Algorithmus 3.9	3/14	14/25	25/36	36/47	47/58
[7]	0.214	0.56	0.694	0.766	0.810

Tabelle 3.1: Exponenten für verschiedene Basisalgorithmen kombiniert mit *Recursive-Color*. Algorithmus 2.11 nutzt als Basis $k_0 = 2$, alle anderen $k_0 = 3$.

Es zeigt sich, dass mit einem einfachen rekursiven Algorithmus für große k auch ein guter Basisalgorithmus kaum Verbesserung bringt. Es ist also notwendig, eine nicht-triviale Kombination der angegebenen Algorithmen zu finden.

Kapitel 4

Approximationsalgorithmen für k-färbbare Graphen

4.1 Einige spezielle Lösungsansätze

In diesem Abschnitt soll es um direkte Erweiterungen der in Abschnitt 3.1 vorgestellten Algorithmen gehen. Während Algorithmus 3.11 nur einen Algorithmus für 3-färbbare Graphen (allgemein: k_0-färbbare Graphen) als Ausgangpunkt benutzt und die Graphen rekursiv färbt, sollen hier die Prinzipien der entsprechenden Algorithmen direkt auf k-färbbare Graphen ausgeweitet werden. Im Allgemeinen sind solche Algorithmen dann langsamer als der rekursive Algorithmus, sie bieten allerdings auch eine Färbung mit weniger Farben. Um einen guten Trade-off zu erhalten, kann man für einen Wert k_1 den erweiterten (besseren) Algorithmus nutzen und diesen dann als Basis für den rekursiven Algorithmus verwenden.

4.1.1 Der Algorithmus von Blum

Zur Erweiterung eines einfacheren Algorithmus von Blum aus [6, 4] ist eine Verallgemeinerung der Eigenschaften 3-färbbarer Graphen auf k-färbbare Graphen notwendig. Bei einigen Feststellungen ist dies auch ohne weiteres möglich, aber der Algorithmus *Dense-Region-Progress* wiederum verlangt direkt, dass der Graph 3-färbbar ist, um einen Fortschritt zu erzielen.
Eine zentrale Überlegung findet alerdings doch noch ihre Beachtung. Zunächst muss jedoch der Wert von δ aus Definition 3.3 angepasst werden.

Definition 4.1. Für einen k-färbbaren Graphen $G = (V, E)$ mit n Knoten sei δ definiert als $\delta = \delta(k) = \frac{1}{4^k \log n}$.
Weiter sind $I_j = \{v \in V : d(v) \in [(1+\delta)^j, (1+\delta)^{j+1})\}, j = 0, 1, ...$ Knotenmengen, die so

nach den Knotengraden eingeteilt sind, dass sich innerhalb einer Menge I_j die Knotengrade um einen Faktor von maximal $1+\delta$ unterscheiden. Ebenso seien innerhalb der Nachbarschaft einer Menge $S \subseteq V$ die Mengen $N_i(S) = \{v \in N(S) : d_S(v) \in [(1+\delta)^i, (1+\delta)^{i+1})\}, i = 0, 1, ...$ analog aufgebaut.

Satz 4.2. *Seien $G = (V, E)$ ein k-färbbarer Graph mit n Knoten, so dass keine zwei Knoten mehr als s Nachbarn gemeinsam haben und $\delta = \Theta(1/\log n)$. Wenn G einen minimalen Grad von $d_{min} \geq (10 \log n)/\delta^2$ hat und eine unabhängige Menge R enthält, so dass $D_R(V \setminus R) \geq \lambda D(V \setminus R)$ für eine Konstante $\lambda \in [0, 1]$ gilt, dann ist für einen Knoten $v \in V$, sowie Indizes $i, j \in \{0, 1, ..., \log_{1+\delta} n\}$ folgende Eigenschaft erfüllt:*
Die Menge $T_{v,i,j} = N_i(N(v) \cap I_j)$ hat eine Größe von $\Omega(\frac{d_{min}^2}{s \log^7 n})$ und mindestens $\lambda(1 - 5\delta)|T_{v,i,j}|$ Knoten aus $T_{v,i,j}$ liegen in R.

Dieser Satz ist eine Abwandlung von Korollar 9 aus [6] und sagt aus, dass es für eine geeignete Kombination der Indizes v, i, j eine Menge $T_{v,i,j}$ gibt, die relativ groß ist und eine große unabhängige Menge enthält. Mit einer solchen Menge kann man dann eine gute Färbung finden. Eine abgewandelte Form von Satz 4.2 mit Beweis befindet sich am Ende dieses Kapitels als Satz 4.17. Der Beweis zu Satz 4.2 verläuft analog zum Beweis von Satz 4.17.
Nun soll ein k-färbbarer Graph mit möglichst wenig Farben gefärbt werden. Um die Anzahl der verwendeten Farben zu bestimmen, wird eine Funktion angegeben, die eine obere Schranke für die Farbanzahl darstellen soll. Es sei

$$f_k(n) = n^{\alpha_k} \log^{\beta_k} n, \tag{4.1}$$

wobei α_k durch eine monoton nicht fallende rekursive Funktion definiert ist, und β_k eine ebenfalls monoton nicht fallende Funktion ist mit $\beta_k \leq \frac{11}{2}$. Der Wert β_k kann durch die Beschränkung zugunsten einer einfacheren Analyse bei $\beta_k = \frac{11}{2}$ fixiert werden.
Zunächst wird der Rekursionsschritt definiert. Hierbei werden sämtliche Knotenpaare u, v darauf überprüft, ob sie viele gemeinsame Nachbarn besitzen. Ist dies der Fall, so kann auf jeden Fall ein Fortschritt erzielt werden.

Satz 4.3. *Es sei $G = (V, E)$ ein k-färbbarer Graph mit n Knoten, der zwei Knoten u, v enthält, die mindestens $n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}$ Nachbarn gemeinsam haben. Weiterhin sei $\boldsymbol{A}$ ein Algorithmus, der jeden $(k-2)$-färbbaren Graphen mit $f_{k-2}(n)$ Farben färbt. Dann kann man einen Fortschritt für eine $f_k(n)$-Färbung von G erzielen.*

Beweis. Für den Beweis dient der folgende Algorithmus:

Algorithmus 4.1. ***Sharing-Progress*(*G*,*k*)**

Eingabe: Ein k-färbbarer Graph G mit zwei Knoten x und y, die mindestens $n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}$ Nachbarn gemeinsam haben und ein Algorithmus **A**, der jeden $(k-2)$-färbbaren Graphen mit $f_{k-2}(n)$ Farben färben kann.

Ausgabe: Fortschritt für eine $f_k(n)$-Färbung von G.

1. $S := N(x) \cap N(y)$ mit $|S| \geq n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}$
2. Wende **A** auf $G[S]$ an.
3. `if` $\leq f_{k-2}(|S|)$ Farben wurden verwendet $\Rightarrow$ *Progress-1*
   `else` *Progress-3*.

Wenn in Schritt 2 der von S induzierte Subgraph $k-2$-färbbar ist, dann entsteht mit Algorithmus **A** eine Färbung mit

$$f_{k-2}(|S|) = |S|^{\alpha_{k-2}} \cdot (\log |S|)^{\beta_{k-2}} \leq |S|^{\alpha_{k-2}} \cdot (\log n)^{\beta_k}$$

Farben. Diese Färbung impliziert sofort eine unabhängige Menge der Größe

$$\Omega\left(\frac{|S|^{1-\alpha_{k-2}}}{(\log n)^{\beta_k}}\right) = \Omega\left(\frac{n^{1-\alpha_k}}{(\log n)^{\beta_k}}\right) = \Omega\left(\frac{n}{f_k(n)}\right).$$

Dies ergibt einen Fortschritt vom Typ 1 (*Progress-1*). Wenn mehr als $f_{k-2}(|S|)$ Farben verwendet wurden, dann ist der von S induzierte Subgraph nicht $(k-2)$-färbbar. Dieser Fall kann aber nur dann eintreten, wenn x und y in jeder zulässigen k-Färbung von G dieselbe Farbe erhalten (*Progress-3*). □

Dieser Algorithmus ist eine Abwandlung der Überlegungen zu 3-färbbaren Graphen. Dort wurden ebenfalls zwei Knoten u, v gesucht, die viele gemeinsame Nachbarn besitzen. Wenn u und v verschiedene Farben bekommen sollen, dann muss die gemeinsame Nachbarschaft einfarbig sein. Ist die Nachbarschaft dagegen keine unabhängige Menge, dann müssen u und v dieselbe Farbe erhalten und können mit *Progress-3* verschmolzen werden. Ähnliches kann man sich auch für 4-färbbare Graphen überlegen: Ist die gemeinsame Nachbarschaft von zwei Knoten nicht bipartit, dann müssen die beiden Knoten dieselbe Farbe erhalten, andernfalls gibt es zwei relativ große unabhängige Mengen.

Mit Algorithmus 4.1 sind bereits zwei Teile für den neuen Algorithmus fertig: Man kann einerseits garantieren, dass der gegebene Graph G einen gewissen Minimalgrad hat[1], und andererseits ausschließen, dass in G zwei Knoten viele gemeinsame Nachbarn haben. Nun muss nur noch dieser Fall bearbeitet werden. Dies geschieht mit Hilfe von Satz 4.2. Das Ergebnis ist ein *Bootstrapping*-Algorithmus, der entweder einen Fortschritt erzielt, oder den gegebenen Graphen so zerlegt, dass mindestens einer der zurückgegebenen Teilgraphen eine große unabhängige Menge enthält.

Algorithmus 4.2. ***Bootstrap*(*H*,*a*,*b*)**

Eingabe: Zwei Werte $a \in [0,1], b > 0$ und $\delta = \frac{1}{\Theta(\log n)}$. Ein Teilgraph H mit m Knoten eines Graphen G mit n Knoten, $m > \frac{1}{\delta^2}$, so dass H eine unabhängige Menge R mit $|R| \geq \lambda m$ für ein $\lambda > 0$ enthält.

Ausgabe: Entweder Fortschritt für eine $O(n^a \log^b n)$-Färbung von G, oder höchstens $m/2$ Teilgraphen $G_0, ..., G_{m/2-1}$ von H, so dass mit hoher Wahrscheinlichkeit mindestens ein G_i einen minimalen Grad von $\delta^2 \cdot \frac{m}{n} \cdot n^a \log^b n$ hat *und* innerhalb von G_i die Bedingung $D(R \cap V(G_i)) \geq (\lambda - 2\delta) \cdot D(V(G_i))$ gilt.

1. Setze $G_0 = (V_0, E_0) = H$.
   `for` $i := 1, ..., \frac{m}{2} - 1$: Erstelle $G_i = (V_i, E_i)$ aus G_{i-1}, indem eine Kante $\{u, v\} \in E_{i-1}$ zufällig gewählt wird und die Knoten u, v aus V_{i-1} gelöscht werden.
2. `for each` G_i mit $|V(G_i)| \geq \delta m$: `while` $d_{min} < \delta^2 \cdot m \cdot n^{a-1} \log^b n$: Lösche den Knoten v mit minimalem Grad aus G_i und setze $W_i := W_i \cup \{v\}$.
3. `if` $\exists i : |W_i| > \delta^2 m \Rightarrow$ *Progress-1*
   `else return` $(G_0, G_1, ..., G_{\frac{m}{2}-1})$.

Satz 4.4. *Sei $\delta = \Theta(\frac{1}{\log n})$. Gegeben sei ein Teilgraph H mit $m > \frac{1}{\delta^2}$ Knoten eines Graphen G mit n Knoten, so dass H eine unabhängige Menge R mit $|R| \geq \lambda |V(H)|$ für eine Konstante $\lambda > 0$ enthält. Dann liegt nach Ablauf von Algorithmus 4.2 einer der folgenden Fälle vor:*

(i) Es wurde ein Fortschritt für eine $O(n^a \log^b n)$-Färbung von G erzielt.

[1] Wenn ein Knoten einen kleinen Grad hat, dann wird mit *Progress-2* ein Fortschritt erzielt.

(ii) Mindestens einer der zurückgegbenen Subgraphen $G_i = (V_i, E_i)$ *besitzt einen minimalen Grad* $d_{min} \geq \delta^2 \cdot m \cdot n^{a-1} \log^b n$ *und innerhalb von* G_i *gilt mit hoher Wahrscheinlichkeit* $D(R \cap V_i) \geq (\lambda - 2\delta) \cdot D(V_i)$.

Beweis. Nach [6]. Die Folge der Subgraphen wird so konstruiert, dass nacheinander jeweils zwei Knoten mit einer Wahrscheinlichkeit proportional zu ihrem Knotengrad entfernt werden. Wenn es im Graphen H eine große unabhängige Menge $R \subseteq V$ gibt, dann werden mit großer Wahrscheinlichkeit zunächst Knoten aus $V \setminus R$ entfernt. Zumindest aber wird in jedem Schritt maximal ein Knoten aus R entfernt. Sei nun $N = m\frac{1-\delta}{2}$. Die Menge V_N enthält damit genau δm Knoten, ist also für Schritt 2 das Abbruchkriterium. Nun wird gezeigt, dass mit hoher Wahrscheinlichkeit mindestens ein $i \leq N$ existiert, für das $D(R_i) \geq (\lambda - \delta) \cdot D(V_i)$ gilt.
Sei A_i, $i \leq N$, das Ereignis, dass bei der Konstruktion von G_{i+1} aus G_i eine Kante mit einem Endknoten in R_i gelöscht wird. Die Anzahl der Kanten mit Endknoten in R_i ist genau $D(R_i)$ (Definition 3.2), da R_i eine unabhängige Menge ist, und damit keine Kanten doppelt gezählt werden. Damit ergibt sich für die Wahrscheinlichkeit, dass A_i eintritt:

$$P[A_i] = \frac{D(R_i)}{|E_i|} = \frac{2D(R_i)}{D(V_i)}.$$

Wenn für einen Index $i \leq N$ die Ungleichung $D(R_i) < (\lambda - \delta) \cdot D(V_i)$ gilt, dann ist die Wahrscheinlichkeit, dass A_i eintritt, maximal $2(\lambda - \delta)$. Seien nun $p = 2(\lambda - \delta)$ und entgegen dem zu Beweisenden $D(R_i) < (\lambda - \delta) \cdot D(V_i),\ \forall i \leq N$. Damit ist die Wahrscheinlichkeit für jedes Ereignis A_i kleiner als p. Da bis zum Erreichen des Teilgraphen G_N genau N-mal eine Kante entfernt wird, ist die Wahrscheinlichkeit, dass mehr als $pN \cdot (1 + \delta)$ Knoten aus R entfernt werden, höchstens $e^{-\delta^2 \cdot \Omega(pN)}$ (Chernoff-Schranke)[2]. Da $pN = \Omega(m)$ und $m > \frac{1}{\delta^2}$ gelten, ist die Wahrscheinlichkeit, dass mehr als $pN \cdot (1 + \delta)$ Knoten aus R entfernt werden,

[2]Für die Chernoff-Schranke gilt: Seien $x_1, ..., x_N$ unabhängige Zufallsvariablen aus $\{0, 1\}$ und $X = \sum_{i=1}^{N}$. Dann gilt für jedes $\delta \in [0, 1]$: $P[X \geq (1 + \delta) \cdot pN] \leq e^{-\frac{\delta^2 pN}{3}}$ [33].

gleich $o(1)$. Damit gilt mit hoher Wahrscheinlichkeit

$$\begin{aligned}
|R_N| &\geq \lambda m - pN \cdot (1+\delta) \\
&= \lambda m - 2(\lambda-\delta) \cdot (m \cdot \frac{1-\delta}{2}) \cdot (1+\delta) \\
&= m(\lambda - 2(\lambda-\delta) \cdot \frac{1-\delta}{2} \cdot (1+\delta)) \\
&= m(\lambda - (\lambda-\delta) \cdot (1-\delta^2)) \\
&= m(\lambda\delta^2 + \delta - \delta^3) \\
&= \delta m + m\delta^2 \cdot (\lambda - \delta) \\
&> \delta m = |V_N|.
\end{aligned}$$

Damit ist mit hoher Wahrscheinlichkeit $|R_N| > |V_N|$, und dies ist ein Widerspruch, da nach Voraussetzung $R_N \subseteq V_N$ ist. Mit hoher Wahrscheinlichkeit ist also die Annahme falsch, dass für jedes $i \leq N$ die Beziehung $D(R_i) < (\lambda - \delta) \cdot D(V_i)$ gilt.

Vor Schritt 2 gilt für mindestens ein $i_0 \leq N$ die Beziehung $D(R_{i_0}) \geq (\lambda-\delta) \cdot D(V_{i_0})$. Nun gibt es zwei mögliche Ausgänge: Wenn weniger als $\delta^2 m$ Knoten entfernt wurden, um den gewünschten minimalen Grad zu erhalten, dann wurden aus dem Subgraphen G_{i_0} maximal $\delta|V_{i_0}|$ Knoten entfernt, jeder mit einem maximalen Grad von $\delta^2 \cdot m \cdot n^{a-1} \log^b n$. Es wurden höchstens $\delta D(V_{i_0})$ der gesamten Gradsumme aus V_{i_0} entfernt. Selbst wenn sämtliche entfernten Knoten aus R_{i_0} wären, gilt immer noch $D(R_{i_0}) \geq (1-2\delta) \cdot D(V_{i_0})$, wie behauptet.

Im anderen Fall wurden aus einem Teilgraphen G_j mindestens $\delta^2 m$ Knoten entfernt. In diesem Fall enthält die Menge W_j der entfernten Knoten mindestens $\delta^2 m$ Knoten, die alle einen maximalen Grad von $\delta^2 \cdot m \cdot n^{a-1} \log^b n$ haben. Innerhalb dieser Menge kann man mit einem Greedy-Algorithmus eine unabhängige Menge der Größe

$$\frac{\delta^2 m}{\delta^2 \cdot m \cdot n^{a-1} \log^b n} = \frac{n}{n^a \log^b n}$$

finden, womit man einen Fortschritt vom Typ *Progress-1* erzielt. □

Aus den Algorithmen 4.1 und 4.2 entsteht nun ein rekursiver Algorithmus für k-färbbare Graphen. Als Rekursionsanfänge werden $k = 2$ und $k = 3$ verwendet.

Algorithmus 4.3. ***Multi-Stage-Color(G,k)***

Eingabe: Ein k-färbbarer Graph G.

Ausgabe: Fortschritt für eine $O(n^{\alpha_k} \log^{\beta_k})$-Färbung von G.

Setze $f(n) = n^{\alpha_k} \log^{\beta_k} n$.

1. `if` $k = 2 \Rightarrow$ Färbe G exakt mit 2 Farben (Algorithmus 1.1)
   `else if` $k = 3 \Rightarrow$ Nutze einen der Approximationsalgorithmen aus Kapitel 3 (z.B. Algorithmus 3.9).
2. `for each` $v \in V$: `if` $d(v) < f(n) \Rightarrow$ *Progress-2*
3. `for each` $u, v \in V$: `if` $|N(u) \cap N(v)| \geq n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}} \Rightarrow$ *Sharing-Progress*
4. `for each` $v \in V, i, j \in \{0, ..., \log_{1+\delta} n\}$: $G_{v,i,j} := G[N_i(N(v) \cap I_j)]$.
5. `for each` $G_{v,i,j}$: *Iterate-Neighbors*(n,k,$G_{v,i,j}$,$k-3$)
   `if` Fortschritt erzielt $\Rightarrow$ HALT
   `else` seien $G_1, ..., G_q$ die zurückgegebenen Graphen.
6. `for each` G_i: *BE/MS*(G_i) (Algorithmus 3.2).

Algorithmus 4.4. ***Iterate-Neighbors*(n,k,G',*iter*)**

Eingabe: Werte n und k; ein Teilgraph G' mit m Knoten und eine Anzahl *iter* von Iterationen.

Ausgabe: $O((m^2(\log_{1+\delta} m)^2)^{iter})$ *Teilgraphen* von G' oder *Fortschritt* für eine $O(n^{\alpha_k} \log^{\beta_k})$-Färbung von G.

1. `if` $iter = 0 \Rightarrow$ `return` G'.
2. *Bootstrap*(G',α_k,β_k).
3. `if` Fortschritt erzielt $\Rightarrow$ HALT.
   `else` seien $H_0, ..., H_{m/2-1}$ die zurückgegebenen Teilgraphen.
4. `for each` H_l,$v \in V(H_l)$,$i, j, \in \{0, ..., \log_{1+\delta} m\}$:
   (a) Sei $G_{l,v,i,j}$ der Subgraph von H_l, der von $N_i(N(v) \cap I_j)$ induziert wurde.
   (b) *Iterate-Neighbors*(n,k,$G_{l,v,i,j}$,$iter-1$)

Mit der Prozedur *Iterate-Neighbors* kann Folgendes festgestellt werden: Es werden im Wesentlichen $O\left((m^2(\log_{1+\delta} m)^2)^{iter}\right)$ Teilgraphen zurückgegeben, aber *iter* erhält für jeden Graphen $G_{v,i,j}$ den Anfangswert $k-3$. Damit läuft Algorithmus 4.3 nicht mehr in Polynomialzeit, da k nicht beschränkt werden kann; so kann z.B. $k = \Omega(\sqrt{n})$ bzw. $k = \Omega(\log n)$ gelten. Allerdings können die grundlegenden Prinzipien dieses Algorithmus für ein Polynomialzeitverfahren verwendet werden.

Satz 4.5. *Mit Algorithmus 4.3 kann man jeden k-färbbaren Graphen mit* $O(n^{\alpha_k} \log^{5.5} n)$ *Farben färben, wobei* $\frac{1}{1-\alpha_k} = 2 - \frac{1}{2^{k-2}} + \frac{1}{1-\alpha_{k-2}} \cdot (1 - \frac{1}{2^{k-2}})$ *sowie* $\alpha_2 = 0$ *und* $\alpha_3 = \frac{3}{14}$ *gelten.*

Beweis. Für $k \leq 3$ ist der Fall klar: Bei $k = 2$ wird der exakte Färbungsalgorithmus 1.1 ausgeführt, und bei $k = 3$ kann man Algorithmus 3.9 verwenden. Die Rekursionsanfänge $\alpha_2 = 0$ und $\alpha_3 = 3/14$ sind also korrekt.
Sei nun $k > 3$. Weiter sei $s_k(n) := n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}$. Die Schritte 2 und 3 sichern, dass der gegebene Graph G einen minimalen Grad $d_{min} \geq n^{\alpha_k} \log^{\beta_k} n$ hat und dass keine zwei Knoten eine gemeinsame Nachbarschaft von mehr als $s_k(n)$ Knoten besitzen. Der Graph G ist k-färbbar. Das bedeutet, in G gibt es eine unabhängige Menge R mit $D_R(V \setminus R) \geq \frac{1}{k-1} \cdot D(V \setminus R)$. Nach Satz 4.2 heißt dies, dass mindestens einer der Graphen $G' = G_{v,i,j}$, die in Schritt 4 erzeugt werden, mindestens

$$\begin{aligned} m_1 &= \frac{d_{min}^2}{s_k(n) \log^7 n} \\ &= \frac{n^{2\alpha_k} \log^{2\beta_k} n}{s_k(n) \log^7 n} \end{aligned} \tag{4.2}$$

Knoten besitzt und eine unabhängige Menge enthält, die einen Anteil von mindestens

$$\lambda_1 = \frac{1}{k-1} \cdot (1 - 5\delta) \geq (\frac{1}{k-1} - 5\delta) \tag{4.3}$$

seiner Knoten umfasst.
An dieser Stelle wird die Funktion *Iterate-Neighbors* aufgerufen. Zunächst enthält der Graph G' mindestens m_1 Knoten, von denen mindestens $\lambda_1 m_1$ Knoten eine unabhängige Menge bilden. Der Ablauf der Funktion *Iterate-Neighbors* beim Graphen G' sieht folgendermaßen aus: Zunächst wird versucht, mit dem Algorithmus 4.2 (*Bootstrap*) einen Fortschritt zu erzielen. Schlägt dies fehl, so heißt das nach Satz 4.4, dass mit hoher Wahrscheinlichkeit einer der konstruierten Graphen H_l einen minimalen Grad von $\delta \cdot m_i \cdot n^{\alpha(k)-1} \log^{\beta(k)} n$ hat und eine unabhängige Menge R' mit

$$D(R') \geq (\lambda_i - 2\delta) \cdot D(V(H_l)) = (\lambda_i - 2\delta) \cdot (D(V(H_l) \setminus R') + D(R'))$$

enthält. Daraus ergibt sich $D(R') \geq \frac{\lambda_i - 2\delta}{1 - \lambda_i + 2\delta} \cdot D(V(H_l) \setminus R')$. Nach Satz 4.2 wird wieder einer der in Schritt 4(a) der Funktion *Iterate-Neighbors* konstruierten Graphen $G_{l,v,i,j}$ mindestens m_{i+1} Knoten besitzen, von denen mindestens $\lambda_{i+1} m_{i+1}$ eine unabhängige Menge bilden. Dabei

gilt wieder wie bei der Bestimmung von m_1 und λ_1:

$$\begin{aligned} m_{i+1} &= \frac{\delta^4 \cdot m_i^2 \cdot n^{2\alpha_k - 2} (\log n)^{2\beta_k}}{s_k(n) \log^7 n} \\ &= \Omega\left(\frac{m_i^2 \cdot n^{2\alpha_k - 2} (\log n)^{2\beta_k}}{s_k(n) \log^{11} n} \right) \end{aligned}$$

und

$$\begin{aligned} \lambda_{i+1} &\geq \frac{\lambda_i - 2\delta}{1 - \lambda_i + 2\delta} - 5\delta = \frac{\lambda_i - 4\delta + 2\delta}{1 - \lambda_i + 2\delta} - 5\delta \\ &\geq \frac{\lambda_i - 4\delta}{1 - \lambda_i} - 5\delta, \quad (\text{da } \frac{a+c}{b+c} \geq \frac{a}{b} \; \forall a, b, c \geq 0, \; b \geq a). \end{aligned}$$

Mit $\beta(k) \leq 5.5 \; \forall k$ und $\lambda_i \leq 1/2$ ergeben sich dann:

$$m_{i+1} = \Omega\left(\frac{m_i^2 \cdot n^{2\alpha_k - 2}}{s_k(n)} \right) \tag{4.4}$$

$$\lambda_{i+1} \geq \frac{\lambda_i}{1 - \lambda_i} - 13\delta, \quad (\text{da } \lambda_i \leq 1/2). \tag{4.5}$$

Unter der Annahme, dass in Schritt 5 kein Fortschritt erzielt wird, besitzt einer der Graphen G_l, die zurückgegeben werden, mindestens m_{k-2} Knoten und enthält eine unabhängige Menge der Größe $\lambda_{k-2} m_{k-2}$. Es bleibt die Bestimmung von m_{k-2} und λ_{k-2}.

Behauptung 1: $\lambda_i \geq \frac{1}{k-i} - 4^{i+2}\delta$ für $0 \leq i \leq k-2$.

Beweis. Für $i = 1$ gilt $\lambda_1 \geq \frac{1}{k-1} - 5\delta \geq \frac{1}{k-1} - 4^3\delta$ (Gleichung (4.3)). Per Induktion folgt nun mit Beziehung (4.5):

$$\begin{aligned} \lambda_i &\geq \frac{\frac{1}{k-i+1} - 4^{i+1}\delta}{\frac{k-i}{k-i+1} + 4^{i+1}\delta} - 13\delta \\ &\geq \frac{\frac{1}{k-i+1} - 2 \cdot 4^{i+1}\delta}{\frac{k-i}{k-i+1}} - 13\delta \\ &\geq \frac{1}{k-i} - 2 \cdot 4^{i+1}\delta \left(\frac{k-i+1}{k-i} \right) - 13\delta \\ &\geq \frac{1}{k-i} - 3 \cdot 4^{i+1}\delta - 13\delta \\ &\geq \frac{1}{k-i} - 4^{i+2}\delta, \end{aligned}$$

wie behauptet. □

Mit Definition 4.1 ($\delta(k) = \frac{1}{4^k \log n}$) ergibt sich dann

$$\lambda_{k-2} \geq \frac{1}{2} - \frac{1}{\log n}. \tag{4.6}$$

Behauptung 2: $m_i = \Omega\left(n^{(2^{i+1}-2)\alpha_k} \cdot n^{2-2^i} \cdot (s_k(n))^{1-2^i}\right)$.

Beweis. Für $i = 1$ gilt nach Gleichung (4.2) und mit $\beta = 5.5$:

$$m_1 = \frac{n^{2\alpha_k} \log^{2\beta} n}{s_k(n) \log^7 n} = \Omega\left(\frac{n^{2\alpha_k}}{s_k(n)}\right).$$

Der Induktionsschritt zeigt nun mit Gleichung (4.4), dass

$$\begin{aligned}
m_{i+1} &= \Omega\left(\frac{m_i^2 \cdot n^{2\alpha_k-2}}{s_k(n)}\right) \\
&= \Omega\left(\frac{\left[n^{(2^{i+1}-2)\alpha_k} \cdot n^{2-2^i} \cdot (s_k(n))^{1-2^i}\right]^2 \cdot n^{2\alpha_k-2}}{s_k(n)}\right) \\
&= \Omega\left(n^{(2^{i+2}-4+2)\alpha_k} \cdot n^{4-2^{i+1}-2} \cdot (s_k(n))^{2-2^{i+1}-1}\right) \\
&= \Omega\left(n^{(2^{i+2}-2)\alpha_k} \cdot n^{2-2^{i+1}} \cdot (s_k(n))^{1-2^{i+1}}\right)
\end{aligned}$$

erfüllt ist. □

Damit gilt für m_{k-2}:

$$m_{k-2} = \Omega\left(n^{(2^{k-1}-2)\alpha_k} \cdot n^{2-2^{k-2}} \cdot (s_k(n))^{1-2^{k-2}}\right). \tag{4.7}$$

In Schritt 5 von Algorithmus 4.3 werden also mehrere Subgraphen konstruiert, von denen mindestens einer

(i) mindestens m_{k-2} Knoten enthält (Gleichung (4.7)), und

(ii) eine unabhängige Menge enthält, die fast die Hälfte seiner Knoten umfasst (siehe Gleichung (4.6)).

Schritt 6 von Algorithmus 4.3 wird dann eine unabhängige Menge der Größe $\frac{m_{k-2}}{\log n}$ finden (siehe [6, 32, 4]), indem Algorithmus 3.2 auf jeden Teilgraphen $G_{l,v,i,j}$ angewendet wird.
Zuletzt muss noch α_k so festgelegt werden, dass $m_{k-2}/\log n = \Omega(\frac{n}{n^{\alpha_k} \log^{\beta_k} n})$ gilt. Mit $\beta = 5.5$ genügt es, zu zeigen, dass $m_{k-2} = \Omega(n^{1-\alpha_k})$ ist:
Es gilt mit $s_k(n) = n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}$:

$$m_{k-2} = \Omega\left(n^{(2^{k-1}-2)\alpha_k} \cdot n^{2-2^{k-2}} \cdot \left(n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}\right)^{1-2^{k-2}}\right).$$

Wenn man hiervon den Logarithmus $\log_n$ bildet, dann muss nun noch gezeigt werden, dass

$$1 - \alpha_k \leq \alpha_k(2^{k-1} - 2) + (2 - 2^{k-2}) + \left(\frac{1 - \alpha_k}{1 - \alpha_{k-2}}\right) \cdot (1 - 2^{k-2})$$

gilt. Durch Umstellen ergeben sich folgende Beziehungen:

$$\begin{aligned}
1 - \alpha_k - \alpha_k(2^{k-1} - 2) - (2 - 2^{k-2}) &\leq \frac{1 - \alpha_k}{1 - \alpha_{k-2}} \cdot (1 - 2^{k-2}) \\
2^{k-1} - 1 - 2^{k-1}\alpha_k + \alpha_k - 2^{k-2} &\leq \frac{1 - \alpha_k}{1 - \alpha_{k-2}} \cdot (1 - 2^{k-2}) \\
(1 - \alpha_k) \cdot (2^{k-1} - 1) - 2^{k-2} &\leq \frac{1 - \alpha_k}{1 - \alpha_{k-2}} \cdot (1 - 2^{k-2}) \\
2^{k-1} - 1 - \frac{2^{k-2}}{1 - \alpha_k} &\leq \frac{1}{1 - \alpha_{k-2}} \cdot (1 - 2^{k-2}) \\
\frac{1}{1 - \alpha_k} &\geq 2 - \frac{1}{2^{k-2}} - \frac{1}{1 - \alpha_{k-2}} \cdot (\frac{1}{2^{k-2}} - 1). \qquad (4.8)
\end{aligned}$$

Wenn α_k die Rekursionsbeziehung (4.8) erfüllt, dann ergibt sich eine Färbung mit $\tilde{O}(n^{\alpha_k})$ Farben.

Die Funktion $f(x) = \frac{1}{1-x}$ ist für $x \in [0, 1)$ monoton wachsend, da $f'(x) = \frac{1}{(1-x)^2} > 0$ gilt. Wenn also x den kleinstmöglichen Wert x_0 annimmt, dann ist auch $f(x_0)$ ein Minimum. Umgekehrt gilt genauso, wenn für $f(x_0)$ der Minimalwert gewählt wird, dann wird auch x_0 minimal sein. Für die Beziehung (4.8) heißt das: Wenn α_k klein bleiben soll, dann muss $\frac{1}{1-\alpha_k} = 2 - \frac{1}{2^{k-2}} - \frac{1}{1-\alpha_{k-2}}(\frac{1}{2^{k-2}} - 1)$ gelten. □

Blum [6] gab für Algorithmus 4.3 als Basisalgorithmus für $k = 3$ seinen Algorithmus 3.7 mit $\alpha(3) = \frac{3}{8}$ an. Im Laufe der Zeit verbesserten sich aber die Algorithmen für 3-färbbare Graphen, und so gilt heute eine Basis von $\alpha(3) = \frac{3}{14}$ (Algorithmus 3.9). Der große Schönheitsfehler von Algorithmus 4.3 ist, dass er kein Polynomialzeitverfahren ist. Das Problem stellt der Algorithmus von Monien und Speckenmeyer [32] (Algorithmus 3.2) in Schritt 6 dar, der als Eingabe einen Teilgraphen benötigt, der eine unabhängige Menge enthält, die fast die Hälfte seiner Knoten umfasst. Um einen solchen Graphen zu finden, der auch noch hinreichend groß ist, wird Algorithmus 4.4 verwendet, der sehr viele (exponentiell in k) Teilgraphen extrahiert, von denen mindestens einer die gewünschten Eigenschaften erfüllt.

Eine Verbesserung in der Laufzeit von $\Theta(n^k)$ zu $O(n^c)$, $c = konst.$ kann ein Algorithmus von Alon und Kahale [2] bringen, der mit Hilfe semidefiniter Programmierung eine große unabhängige Menge findet, auch wenn es im Graphen nur eine unabhängige Menge gibt,

die höchstens einen Anteil von $\frac{1}{k-1}$ der Knoten umfasst. Da Alon und Kahale aber bereits semidefinite Programmierung verwenden, wird die Kombination der Algorithmen von Blum [6] und Alon/Kahale [2] gemeinsam mit einem Algorithmus von Karger/Motwani/Sudan [24] in Abschnitt 4.2.2 angegeben. Die Variante von Algorithmus 4.3 mit der Technik von Alon und Kahale [2] erzielt eine Färbung mit $\tilde{O}(n^{\alpha_k})$ Farben, wobei $\alpha_k = 1 - \frac{6}{k+6+\frac{3}{1-\alpha_{k-2}}}$ für $k \geq 4$ und $\alpha_2 = 0$ sowie $\alpha_3 = \frac{3}{14}$ gelten. Eine Übersicht über die Exponenten α_k bei der Anwendung des Färbungsalgorithmus 4.3 mit verschiedenen Basisalgorithmen ist in Tabelle 4.1 angegeben.

	k =3	4	5	6	7
Algorithmus 3.11	3/14	14/25	25/36	36/47	47/58
$\alpha_3 = 3/14$	0.214	0.56	0.694	0.766	0.810
Algorithmus 4.3	3/8	3/5	91/131	105/137	5301/6581
$\alpha_3 = 3/8$	0.375	0.6	0.695	0.766	0.806
Algorithmus 4.3	1/4	3/5	49/73	105/137	3007/3775
$\alpha_3 = 1/4$	0.25	0.6	0.671	0.766	0.796
Algorithmus 4.3	3/14	3/5	175/263	105/137	10881/13697
$\alpha_3 = 3/14$	0.214	0.6	0.665	0.766	0.794
Variante von	3/14	7/13	97/163	25/37	317/449
Algorithmus 4.3 mit [2]	0.214	0.538	0.595	0.675	0.706

Tabelle 4.1: Exponenten α_k für die rekursiven Algorithmen 3.11 und 4.3 mit verschiedenen Basen im Vergleich.

Man sieht in Tabelle 4.1, dass die Performance für gerades k bei allen Basisalgorithmen für *Multi-Stage-Color* gleich ist. Das liegt ganz einfach daran, dass die Rekursion immer in Zweierschritten vonstatten geht. Ein veränderter Basisalgorithmus für $k_0 = 3$ wirkt sich also nur direkt auf k-färbbare Graphen aus, wenn k ungerade ist. Offensichtlich ist die polynomiell laufende Variante von Algorithmus 4.3 besser als das exponentiell laufende original angegebene Verfahren von Blum [6]. Das liegt daran, dass in der Prozedur *Iterate-Neighbors* (Algorithmus 4.4) der Ausgangsgraph schrittweise verkleinert wird, so dass die gefundene unabhängige Menge relativ klein wird, obwohl sie noch die geforderte Größe besitzt. Tabelle 4.2 zeigt die wichtigsten Daten der Algorithmen von Monien/Speckenmeyer [32] und Alon/Kahale [2].

4.1.2 Der Algorithmus von Karger, Motwani und Sudan

In diesem Abschnitt wird der Algorithmus von Karger, Motwani und Sudan [24] (Algorithmus 3.8) von 3-färbbaren Graphen auf k-färbbare Graphen erweitert. Dies ist relativ einfach, da die Definition einer Vektor-k-Färbung bereits bekannt ist (Definition 3.6).

	Monien/Speckenmeyer [32]	Alon/Kahale [2]
Mindestgröße für ein IS	$(\frac{1}{2} - O(\frac{1}{\log m})) \cdot m$	$(\frac{1}{k-1} - O(\frac{1}{\log m})) \cdot m$
Gefundenes IS	$\Omega(\frac{m}{\log m})$	$\Omega(m^{3/k})$
Einsatz bei $m =$	$\Omega(n^{1-\alpha_k})$	$\Omega(n^{\frac{k(1-\alpha_k)}{3}})$
Rekursion für α_k	$\frac{1}{1-\alpha_k} = 2 - \frac{1}{2^{k-2}} + \frac{1-2^{2-k}}{1-\alpha_{k-2}}$	$\alpha_k = 1 - \frac{6}{k+6+\frac{3}{1-\alpha_{k-2}}}$

Tabelle 4.2: Vergleich von Algorithmus 3.2 mit dem Algorithmus von Alon/Kahale (hier in Abschnitt 4.2.2 angeführt).

Gegeben ist wieder ein k-färbbarer Graph $G = (V, E)$ mit n Knoten. Wie bei Algorithmus 3.8 wird zunächst mit Beziehung (3.1) eine Matrix-k-Färbung gesucht, d.h. eine Matrix $M \in \mathbb{R}^{n \times n}$, bei der die Einträge $m_{ij} \leq -\frac{1}{k-1}$ für jede Kante $\{i, j\} \in E$ und die Hauptiagonalelemente $m_{ii} = 1$ sind. Diese Matrix wird mit der Cholesky-Zerlegung in n Vektoren zerlegt, die ihrerseits eine Vektor-k-Färbung bilden. Schließlich wird durch Runden der Vektoren auf zufällige Zentren (s. Definition 3.7) jedem Knoten eine Farbe zugewiesen [24, 4].
Zu bestimmen ist noch die Anzahl t der zu verwendenden Farben, so dass mit großer Wahrscheinlichkeit eine t-Semifärbung entsteht.

Satz 4.6. [24]. *Sei $P_k(n, t)$ die Wahrscheinlichkeit, dass zwei adjazente Knoten in einem k-färbbaren Graphen $G = (V, E)$ mit n Knoten bei der Rundung auf t Farben dieselbe Farbe erhalten. Dann gilt:*

$$P_k(n, t) = \Theta(t^{\frac{-k}{k-2}}).$$

Beweis. Da die t Zentren unabhängig voneinander gewählt wurden, genügt es, die Wahrscheinlichkeit zu bestimmen, dass zwei Vektoren u_1, u_2 mit $\langle u_1, u_2 \rangle \leq -\frac{1}{k-1}$ von einem bestimmten Zentrum, z.B. c_1, gefangen werden. Dies mit t multipliziert ergibt dann die Wahrscheinlichkeit, dass u_1 und u_2 von irgendeinem Zentrum gefangen werden.
Außerdem genügt es, sich auf $P_k(2, t)$ zu beschränken. Die Zentren können auch dann immer noch als unabhängig voneinander gewählte zufällige Punkte betrachtet werden, wenn man nur die Ebene betrachtet, die von den beiden Vektoren u_1 und u_2 aufgespannt wird.
Zur Berechnung der Wahrscheinlichkeit $P_k(2, t)$ sei $\langle u_1, u_2 \rangle \leq -\frac{1}{k-1}$. Das bedeutet, der Winkel zwischen u_1 und u_2 beträgt mindestens $2\theta = \arccos(-\frac{1}{k-1})$. O.B.d.A. sei der Vektor u_1 weiter von c_1 entfernt, d.h. der Winkel zwischen c_1 und u_1 ist größer als θ. Das Zentrum c_1 fängt u_1 nur dann, wenn kein anderes der t Zentren ein größeres Skalarprodukt mit u_1 hat. Sei R der Abschnitt der Ebene, der in einem Winkel von maximal $\varepsilon < \theta$ um u_1 liegt, und in

diese Region fallen r Zentren. Es ist klar, dass die Projektion von c_1 auf die näher gelegene Begrenzungslinie von R größer sein muss als jedes einzelne Zentrum in R selbst, sonst kann u_1 nicht gefangen werden. Dies ist nur eine notwendige Bedingung, keine hinreichende.
Sei nun A das Ereignis, dass die Projektion von c_1 auf eine Linie, die einen Winkel von $\phi = \theta - \delta$ zu u_1 bildet, größer ist als alle Zentren innerhalb der Region R.

Lemma 4.7. *Sei $P(A, r)$ die Wahrscheinlichkeit für das Ereignis A unter der Annahme, dass r Zentren in R liegen. Dann gilt $P(A) = \binom{r+q}{r}^{-1}$, wobei $q = 1/cos^2\phi$ ist.*

Beweis. Die Längen der Zentren sind nach [24, 4] exponentialverteilt mit dem Parameter $\lambda = \frac{1}{2}$. Sei X die Größe der Projektion von c_1 auf die Begrenzungslinie von R, und $Y_1, ..., Y_r$ die Längen der Zentren innerhalb von R. Dann ist die gesuchte Wahrscheinlichkeit äquivalent zu $P[X \geq q \cdot \max Y_i]$. Mit Hilfe der Eulerschen Γ-Funktion[3] ergibt sich hierfür ein Wert von $\binom{r+q}{r}^{-1}$. □

Weiter im Beweis von Satz 4.6. Die Anzahl der Vektoren in R ist durch die Binomialverteilung $B(t, p)$ mit $p = \frac{\varepsilon}{\pi}$ bestimmt. Damit ergibt sich als Wahrscheinlichkeit für das Ereignis A:

$$\begin{aligned}
P[A] &= \sum_{r=0}^{t} \binom{t}{r} p^r (1-p)^{t-r} \cdot \binom{r+q}{r}^{-1} \\
&= \binom{t+q}{t}^{-1} \cdot \sum_{r=0}^{t} \binom{t+q}{t-r} p^r (1-p)^{t-r} \\
&= \binom{t+q}{t}^{-1} \cdot \sum_{s=0}^{t} \binom{t+q}{s} p^{t-s} (1-p)^s \\
&\leq \binom{t+q}{t}^{-1} \cdot \sum_{s=0}^{t} \binom{t+\lceil q \rceil}{s} p^{t-s} (1-p)^s \\
&= p^{-\lceil q \rceil} \binom{t+q}{t}^{-1} \cdot \sum_{s=0}^{t} \binom{t+\lceil q \rceil}{s} p^{t+\lceil q \rceil - s} (1-p)^s \\
&\leq p^{q-\lceil q \rceil} \left(p \binom{t+q}{t} \right)^{-1} \cdot (p + (1-p))^{t+\lceil q \rceil} \\
&= O(p^{q-\lceil q \rceil} (pt)^{-q}). \qquad (4.9)
\end{aligned}$$

Mit $\varepsilon = \frac{1}{\log t}$ ergeben sich folgende Parameter: $p = \frac{1}{\pi \log t}$ und $q = \frac{1}{\cos^2(\theta - \varepsilon)} = \frac{2(k-1)}{k-2} - O(\varepsilon)$. Zusammen mit Gleichung (4.9) ergibt dies unter der Beachtung, dass $p^{q-\lceil q \rceil} = \Theta(1)$ (nach [24])

[3] Mit der Γ-Funktion $\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dx$ kann der Binomialkoeffizient $\binom{n}{k}$ auf komplexe Zahlen n ausgeweitet werden: $\binom{n}{k} = \frac{\Gamma(n+1)}{\Gamma(k+1)\Gamma(n-k+1)}$.

gilt:

$$\begin{aligned} P_k(2,t) &= O(t(pt)^{-q}) \\ &= O(t(\frac{t}{\log t})^{-\frac{2(k-1)}{k-2}(1-O(\frac{1}{\log t}))}) \\ &= O(t^{-\frac{k}{k-2}} \log^{\frac{2k-2}{k-2}} t). \end{aligned}$$

Dies ist der zu beweisende Wert. □

Aus Satz 4.6 kann man schließen, dass $\Delta^{1-\frac{2}{k}+o(1)}$ Farben genügen, um eine Semifärbung zu erhalten. Um eine absolute Approximationsgüte zu berechnen, wird ein Schwellwert Δ_k definiert. Solange der maximale Grad den Wert Δ_k übersteigt, wird ein Knoten v mit maximalem Grad gewählt, seine Nachbarschaft $N(v)$ rekursiv gefärbt und dann $N(v)$ aus dem Graphen entfernt, da die Nachbarschaft eines Knotens in einem k-färbbaren Graphen $(k-1)$-färbbar ist. Damit kann wieder eine Funktion α_k bestimmt werden, so dass jeder k-färbbare Graph mit n Knoten mit n^{α_k} Farben gefärbt wird. Es ergibt sich folgender Algorithmus:

Algorithmus 4.5. ***KMS-k-Color*(*G*,*k*)**

Eingabe: Ein k-färbbarer Graph G.

Ausgabe: Eine zulässige Färbung von G.

$i := 1$;

`while` $\Delta(G) \geq \Delta_k$

· Wähle $v \in V$ mit $d(v) = \Delta(G)$.

· Färbe $N(v)$ rekursiv mit *KMS-k-Color*$(G[N(v)],k-1)$.

· $G := G[V \backslash N(v)]$

Finde eine Vektor-k-Färbung $f(v)$ von G mittels Semidefiniter Programmierung.

`while` G ist noch nicht komplett gefärbt:

· Wähle $\Delta^{1-2/k}$ zufällige Punkte $c_1, ..., c_t \in \mathbb{R}^n$.

· `for each` $v \in V$: Setze $c(v) := j$ mit $\langle c_j, f(v)\rangle > \langle c_{j'}, f(v)\rangle \forall j' \neq j$.

· $S := \{v \in V : \exists w \in V : \{v,w\} \in E \wedge c(v) = c(w)\}$;

· `for each` $v \in V \backslash S$: färbe v mit Farbe $i + c(v)$

· $i := i + t$

Nun muss noch bestimmt werden, wie viele Farben bei diesem Algorithmus verwendet werden. Hierzu sei v ein Knoten mit $d = d(v) = \Delta(G)$. Seine Nachbarschaft $N(v)$ (d Knoten) wird mit $d^{\alpha_{k-1}}$ Farben gefärbt. Das heißt, für jeden Knoten werden im Durchschnitt $d^{\alpha_{k-1}-1} \leq \Delta_k^{\alpha_{k-1}-1}$ Farben verwendet. Die maximale Anzahl der in diesem Schritt verwendeten Farben ist damit höchstens $n\Delta_k^{\alpha_{k-1}-1}$, da höchstens n Knoten in der ersten While-Schleife gefärbt werden können. Wenn der maximale Grad kleiner als Δ_k wird, dann wird der restliche Graph direkt mit $\Delta_k^{1-2/k}$ Farben gefärbt. Durch Gleichsetzen ergibt sich

$$n\Delta_k^{\alpha_{k-1}-1} = \Delta_k^{1-2/k},$$

woraus die Beziehungen

$$n = \Delta_k^{2-2/k-\alpha_{k-1}}$$
$$\Delta_k = n^{\frac{1}{2-2/k-\alpha_{k-1}}}$$

folgen. Damit ergibt sich eine Färbung mit $2\Delta_k^{1-2/k} = 2n^{\frac{1-2/k}{2-2/k-\alpha_{k-1}}}$ Farben. Das heißt:

$$\alpha_k = \frac{1-\frac{2}{k}}{2-\frac{2}{k}-\alpha_{k-1}}. \tag{4.10}$$

Satz 4.8. *Jeder k-färbbare Graph $G = (V, E)$ mit n Knoten kann mittels Semidefiniter Programmierung mit $\tilde{O}(n^{1-\frac{3}{k+1}})$ Farben gefärbt werden.*

Beweis. Der Beweis wird mit Induktion über k geführt. Für $k = 3$ ist $\alpha_k = \frac{1}{4} = 1 - \frac{3}{4}$ mit Algorithmus 3.8. Sei nun α_k durch die in Gleichung (4.10) gegebene Rekursion bestimmt und $\alpha_{k-1} = 1 - \frac{3}{k}$. Dann gilt:

$$\begin{aligned}\alpha_k &= \frac{1-\frac{2}{k}}{2-\frac{2}{k}-\alpha_{k-1}} = \frac{1-\frac{2}{k}}{2-\frac{2}{k}-(1-\frac{3}{k})} \\ &= \frac{(k-2)/k}{(k+1)/k} = \frac{k-2}{k+1} = 1 - \frac{3}{k+1},\end{aligned}$$

wie behauptet. □

Es ergibt sich damit im Vergleich zu Algorithmus 4.3 von Blum eine wesentliche Verbesserung hinsichtlich der Anzahl verwendeter Farben, die in Tabelle 4.3 aufgeführt ist.
Offensichtlich ist Algorithmus 4.3 selbst dann noch schlechter als Algorithmus 4.5, wenn er im Basisfall Algorithmus 4.5 mit $k_1 = 4$ verwendet. Man sollte also versuchen, bei größeren k den Algorithmus von Karger, Motwani und Sudan stärker einfließen zu lassen, um bessere Resultate zu erhalten.

	k =3	4	5	6	7
Algorithmus 4.3	3/14	3/5	175/263	105/137	10881/13697
$\alpha_3 = 3/14$	0.214	0.6	0.665	0.766	0.794
Algorithmus 4.5	1/4	2/5	1/2	4/7	5/8
	0.25	0.4	0.5	0.571	0.625
Algorithmus 4.3	3/14	2/5	175/263	5/7	10881/13697
$\alpha_3 = 3/14, \alpha_4 = 2/5$	0.214	0.4	0.665	0.714	0.794

Tabelle 4.3: Die Exponenten α_k der Algorithmen 4.3 und 4.5 für kleine Werte von k.

Ein weiterer Aspekt ist die Beziehung zwischen der Vektor-chromatischen Zahl und der chromatischen Zahl eines Graphen. Es ist bereits bekannt, dass jeder k-chromatische Graph auch vektor-k-färbbar ist. Umgekehrt kann jedoch gezeigt werden, dass ein vektor-k-färbbarer Graph nicht unbedingt k-färbbar sein muss. Bereits Karger, Motwani und Sudan [24] konnten für sogenannte *Kneser*-Graphen mit konstanter vektor-chromatischer Zahl k eine chromatische Zahl $\chi > n^{0.07}$ bestimmen.

Definition 4.9. Ein Kneser-Graph $K(m, r, t)$ ist ein Graph $G = (V, E)$ mit der Knotenmenge $V = \{S \subseteq \{1, ..., m\} : |S| = r\}$ und der Kantenmenge $E = \{\{i, j\} : |S_i \cap S_j| < t\}$. Es ist damit $|V| = \binom{m}{r} = n$.

Als Beispiel wird der Kneser-Graph $K(m, \frac{m}{2}, \frac{m}{8})$ angeführt, der zwar vektor-3-färbbar ist, aber für die chromatische Zahl gilt $\chi(K(m, \frac{m}{2}, \frac{m}{8})) \approx n^{0.0113}$ [24]. Feige, Langberg und Schechtman [15] konstruierten eine Klasse von Graphen, die vektor-k-färbbar sind, und die mit hoher Wahrscheinlichkeit keine unabhängige Menge der Größe $\frac{n}{\Delta^{1-\frac{2}{k}}}$ enthalten.[4] Diese Graphen basieren auf einem *kontinuierlichen* Graphen G^c, der aus einer d-dimensionalen Einheitskugel gewonnen wird. Dann wird in mehreren Phasen dieser Graph diskretisiert, bis nur noch relativ wenige Knoten vorhanden sind. Für detaillierte Beschreibungen wird auf [15] verwiesen.
Der Algorithmus von Karger, Motwani und Sudan (Algorithmus 4.5) färbt solche Graphen zwar nahezu exakt, aber es zeigt auch, dass man mit einem Vektor-k-färbbaren Graphen keine genauere Aussage treffen kann, als dass die chromatische Zahl mit Sicherheit unter $\Delta^{1-\frac{2}{k}}$ liegen wird.

[4] Ein k-färbbarer Graph enthält immer eine unabhängige Menge der Größe $\frac{n}{k}$.

4.2 Kombination von Algorithmen

Hier werden die beiden Algorithmen von Blum [6] und Karger, Motwani und Sudan [24] so kombiniert, dass eine Verringerung der Farbanzahl erreicht wird.

4.2.1 Der Algorithmus von Xie, Ono und Hirata

Xie, Ono und Hirata [37] kombinierten einfach den Algorithmus von Blum und Karger (Algorithmus 3.9) mit dem erweiterten Algorithmus von Karger, Motwani und Sudan (Algorithmus 4.5). Die Idee stammt wieder von Wigderson [36]: Solange ein Knoten mit großem Grad existiert, wird seine Nachbarschaft rekursiv gefärbt, und diese Knoten dann aus dem Graphen entfernt. Sobald der maximale Knotengrad einen Schwellwert unterschreitet, wird Algorithmus 4.5 verwendet, um den Restgraphen zu färben. Als Basisfall dient hier der Algorithmus von Blum und Karger (Algorithmus 3.9, siehe [7]). Zunächst der Algorithmus, der einen k-färbbaren Graphen mit n Knoten mit $\tilde{O}(n^{\alpha_k})$ Farben färbt, wobei α_k wieder näher zu bestimmen ist:

Algorithmus 4.6. ***Japanese-k-Color*(*G*,*k*)**

Eingabe: Ein k-färbbarer Graph G mit n Knoten.

Ausgabe: Eine $\tilde{O}(n^{\alpha_k})$-Färbung von G.

1. `if` $k = 2 \Rightarrow$ Färbe G mit 2 Farben (Algorithmus 1.1)
2. `if` $k = 3 \Rightarrow$ *Best-3-Color* (Algorithmus 3.9)
3. `while` $\Delta(G) > n^{\alpha_k/(1-2/k)}$:
   - $v :=$ Knoten mit $d(v) = \Delta(G)$
   - $N'(v) :=$ Teilmenge von $N(v)$ mit $|N'(v)| = n^{\alpha_k/(1-2/k)}$
   - *Japanese-k-Color*$(G[N'(v)],k-1)$
   - $G := G[V \setminus N'(v)]$
4. *KMS-k-Color*(G,k) (Algorithmus 4.5)

Satz 4.10. *Sei $k \geq 3$. Algorithmus 4.6 färbt jeden k-färbbaren Graphen mit $\tilde{O}(n^{\alpha_k})$ Farben, wobei $\alpha_3 = \frac{3}{14}$ und $\alpha_k = \frac{k-2}{2k-2-k\alpha_{k-1}}$ gelten.*

Beweis. Der Algorithmus ist korrekt, da nur Verfahren verwendet werden, die bereits als korrekte Verfahren bekannt sind. Bei jeder Anwendung wird ein neuer Satz Farben verwendet.

Somit können keine Konflikte auftreten.

Ein 3-färbbarer Graph wird mit Algorithmus 3.9 mit $\tilde{O}(n^{3/14})$ Farben gefärbt. Es ist demnach $\alpha_3 = \frac{3}{14}$. Sei nun $k \geq 4$. In Schritt 3 wird eine Knotenmenge der Größe $m = n^{\frac{\alpha_k}{1-2/k}}$ rekursiv mit $\tilde{O}(m^{\alpha_{k-1}})$ Farben gefärbt. Diese Beziehung ist zulässig, da die Knotenmenge aus der Nachbarschaft eines Knotens stammt und demnach $k-1$-färbbar ist. Die Anzahl verwendeter Farben in einem Durchlauf von Schritt 3 ist demnach $n^{\frac{\alpha_k \alpha_{k-1}}{1-2/k}}$. Bei jedem Durchlauf werden diese m Knoten aus dem Graphen gelöscht. Die Schleife kann also höchstens $\frac{n}{m} = n^{1-\frac{\alpha_k}{1-2/k}}$-mal durchlaufen werden. Da bei jedem Durchlauf neue Farben verwendet werden, werden in Schritt 3 maximal

$$n^{1-\frac{\alpha_k}{1-2/k}} \cdot \tilde{O}\left(n^{\frac{\alpha_k \alpha_{k-1}}{1-2/k}}\right) = \tilde{O}\left(n^{1+\frac{\alpha_k(\alpha_{k-1}-1)}{1-2/k}}\right)$$

Farben benutzt. Im letzten Schritt werden nochmals $\tilde{O}(\Delta(G)^{1-2/k}) = \tilde{O}(n^{\alpha_k})$ Farben verwendet, da mit $\Delta(G) = n^{\alpha_k/(1-2/k)}$ der maximale Grad des Restgraphen bereits bekannt ist. Insgesamt benutzt Algorithmus 4.6 nur $\tilde{O}(n^{\alpha_k} + n^{1+\frac{\alpha_k(\alpha_{k-1}-1)}{1-2/k}})$ Farben. Um hier das Minimum an Farben zu bestimmen, werden die beiden Summanden gleichgesetzt:

$$\alpha_k = 1 + \frac{\alpha_k(\alpha_{k-1}-1)}{1-2/k}.$$

Durch Umstellen ergibt sich die zu beweisende Rekursionsformel $\alpha_k = \frac{k-2}{2k-2-k\alpha_{k-1}}$. □

Die Auflösung dieser Rekursion ergibt Folgendes:

$$\begin{aligned}
\frac{1}{1-\alpha_k} &= \frac{1}{1-\frac{k-2}{2k-2-k\alpha_{k-1}}} \\
&= \frac{k-2}{k-k\alpha_{k-1}} + 1 \\
\frac{k(k-1)}{1-\alpha_k} &= k(k-1) + \frac{(k-1)(k-2)}{1-\alpha_{k-1}}.
\end{aligned}$$

Mit $\beta_k = \frac{k(k-1)}{1-\alpha_k}$ ergibt sich eine vereinfachte Rekursionsgleichung: $\beta_k = k(k-1) + \beta_{k-1}$. Die Anfangsbedingung $\alpha_3 = \frac{3}{14}$ liefert $\beta_3 = \frac{84}{11}$.

Behauptung: $\beta_k = \frac{1}{3}(k-1)k(k+1) - \frac{4}{11}$ für $k \geq 3$.

Beweis. Induktion über k: Für $k = 3$ ist $\beta_3 = \frac{1}{3} \cdot 2 \cdot 3 \cdot 4 - \frac{4}{11} = \frac{24}{3} - \frac{4}{11} = \frac{84}{11}$. Sei

nun $k > 3$. Es gilt:

$$\begin{aligned}
\beta_k &= k(k-1) + \beta_{k-1} \\
&= k(k-1) + \frac{1}{3}(k-2)(k-1)k - \frac{4}{11} \\
&= k(k-1)(1 + \frac{k-2}{3}) - \frac{4}{11} \\
&= k(k-1)\frac{k+1}{3} - \frac{4}{11} \\
&= \frac{(k-1)k(k+1)}{3} - \frac{4}{11},
\end{aligned}$$

wie behauptet. □

Damit ergibt sich für α_k:

$$\begin{aligned}
\alpha_k &= 1 - \frac{k(k-1)}{\beta_k} \\
&= 1 - \frac{k(k-1)}{\frac{(k-1)k(k+1)}{3} - \frac{4}{11}} \\
&= 1 - \frac{1}{\frac{k+1}{3} - \frac{4}{11k(k-1)}}.
\end{aligned}$$

Ausgewählte Ergebnisse für Algorithmus 4.6 sind in Tabelle 4.4 auf Seite 83 aufgeführt.

4.2.2 Der Algorithmus von Halperin, Nathaniel und Zwick

Eine deutlichere Verbesserung des Algorithmus von Karger, Motwani und Sudan [24] lieferten Halperin, Nathaniel und Zwick [20]. Algorithmus 4.6 zeigt, dass eine triviale Kombination der Algorithmen von Blum (Algorithmus 4.3) und Karger (Algorithmus 4.5) bereits an ihre Grenzen gestoßen ist. Halperin, Nathaniel und Zwick fügen nun einen Algorithmus ein, der ähnlich dem Algorithmus 3.2 eine große unabhängige Menge findet. Das Verfahren stammt von Alon und Kahale [2] und findet in einer Knotenmenge M, die eine unabhängige Menge S der Größe $\frac{|M|}{\varkappa}$ enthält, eine unabhängige Menge I der Größe $\tilde{\Omega}(|M|^{f(\varkappa)})$ mit

$$f(\varkappa) = \frac{\varkappa(\varkappa-1)}{\lfloor\varkappa\rfloor(\varkappa(\varkappa - \lfloor\varkappa\rfloor) + \frac{(\lfloor\varkappa\rfloor-1)(\lfloor\varkappa\rfloor+1)}{3})} \,. \tag{4.11}$$

Für ganze Zahlen $\varkappa = k$ ergibt sich $f(k) = \frac{3}{k+1}$, und für den Fall $\varkappa = k + O(\frac{1}{\log n})$ gilt $f(\varkappa) = \frac{3}{k+1} - O(1/\log n)$.

Behauptung: Die Funktion $f(\varkappa)$ erfüllt die Rekursion $f(\varkappa) = \frac{1}{1+\frac{1-2/\varkappa}{f(\varkappa-1)}}$ für $\varkappa \geq 2$.

Beweis. Sei $k = \lfloor \varkappa \rfloor$. Zunächst ist für $1 < \varkappa < 2$ die Gleichung

$$f(\varkappa) = \frac{\varkappa(\varkappa - 1)}{\varkappa(\varkappa - 1) + \frac{(1-1)(1+1)}{3}} = 1$$

erfüllt. Sei nun $2 \leq \varkappa < 3$: Dann gilt einerseits

$$f(\varkappa) = \frac{1}{1 + \frac{1-2/\varkappa}{f(\varkappa-1)}} = \frac{\varkappa}{2(\varkappa - 1)}$$

und andererseits

$$f(\varkappa) = \frac{\varkappa(\varkappa - 1)}{2(\varkappa(\varkappa - 2) + 1)} = \frac{\varkappa}{2(\varkappa - 1)}.$$

Sei nun $\varkappa \geq 3$. Nach Induktionsvoraussetzung ist

$$f(\varkappa - 1) = \frac{(\varkappa - 1)(\varkappa - 2)}{(k-1) \cdot ((\varkappa - 1) \cdot ((\varkappa - 1) - (k-1)) + \frac{(k-2)k}{3})}.$$

Durch Einsetzen in die Rekursionsgleichung ergibt sich

$$\begin{aligned}
f(\varkappa) &= \frac{(\varkappa - 1)(\varkappa - 2)}{(\varkappa - 1)(\varkappa - 2) + (1 - \frac{2}{\varkappa})(k-1) \cdot ((\varkappa - 1)(\varkappa - k) + \frac{(k-2)k}{3})} \\
&= \frac{\varkappa(\varkappa - 1)}{\varkappa(\varkappa - 1) + (k-1)(\varkappa - 1)(\varkappa - k) + \frac{(k-1)(k-2)k}{3}} \\
&= \frac{\varkappa(\varkappa - 1)}{k(\varkappa^2 - k\varkappa + k - 1 + \frac{(k-1)(k-2)}{3})} \\
&= \frac{\varkappa(\varkappa - 1)}{k(\varkappa(\varkappa - k) + \frac{k^2-1}{3})} \\
&= \frac{\varkappa(\varkappa - 1)}{k(\varkappa(\varkappa - k) + \frac{(k-1)(k+1)}{3})},
\end{aligned}$$

was der Definition der Funktion $f(\varkappa)$ entspricht. □

Satz 4.11. *Sei $G = (V, E)$ ein Graph mit n Knoten, der eine unabhängige Menge der Größe $\frac{n}{\varkappa}$ enthält, mit $\varkappa \geq 1$. Dann kann in G in Polynomialzeit eine unabhängige Menge der Größe $\tilde{\Omega}(n^{f(\varkappa)})$ gefunden werden. Hierbei ist $f(\varkappa)$ wie in Gleichung (4.11) definiert.*

Der Beweis des Satzes basiert auf den folgenden Lemmata:

Lemma 4.12. *Sei $G = (V, E)$ ein Graph mit n Knoten, der eine unabhängige Menge der Größe $\frac{n}{\varkappa}$ enthält mit $\varkappa \geq 2$. Dann kann man eine Menge $S \subseteq V$ mit $|S| \geq \frac{n}{\log n}$ und eine Vektor-$(\varkappa + O(1/\log n))$-Färbung von $G[S]$ in Polynomialzeit finden.*

Beweis. Sei $V = \{1, ..., n\}$ und folgende Relaxierung des MAXIMUM INDEPENDENT SET-Problems gegeben:

$$\text{Maximiere} \sum_{i=1}^{n} \frac{1 + \langle v_0, v_i \rangle}{2}$$
$$\text{NB} : \langle (v_0 + v_i), (v_0 + v_j) \rangle = 0,\ \{i, j\} \in E$$
$$\|v_i\| = 1,\ i = 0, ..., n.$$

Hierbei seien $v_i \in \mathbb{R}^n$ den Knoten $i \in V$ zugeordnete Einheitsvektoren und $v_0 \in \mathbb{R}^n$ ein Einheitsvektor. Bei einer Lösung liegen die Vektoren v_i einer unabhängigen Menge I so, dass sie alle in dieselbe Richtung zeigen wie v_0. Eine fast optimale Lösung $v_0, v_1, ..., v_n$ dieses Problems kann in Polynomialzeit gefunden werden. Wenn $v_0, v_1, ..., v_n$ nur um maximal $\frac{n}{\log n}$ vom Optimum ($\frac{n}{\varkappa}$) abweicht, dann ergibt die Zielfunktion den Wert $\sum_{i=1}^{n} \frac{1+\langle v_0, v_i \rangle}{2} \geq \frac{n}{\varkappa} - \frac{n}{\log n}$. Damit kann angenommen werden, dass

$$\sum_{i=1}^{n} \langle v_0, v_i \rangle \geq \left(\frac{2}{\varkappa} - 1 - \frac{1}{\log n} \right) n$$

gilt.

Nun gilt aber: Wenn $\sum_{i=1}^{n} x_i \geq \gamma n$ mit $x_i \leq 1\ \forall i$ gelten soll, dann müssen für jedes $\epsilon > 0$ mindestens ϵn der x_i die Bedingung $x_i \geq \frac{\gamma - \epsilon}{1 - \epsilon}$ erfüllen.[5]

Mit $\epsilon < \frac{1+\gamma}{2}$ gilt nun $\frac{\gamma-\epsilon}{1-\epsilon} > \gamma - 2\epsilon$. Ist nun $x_i = \langle v_0, v_i \rangle$, $\gamma = (\frac{2}{\varkappa} - 1 - \frac{1}{\log n})$ und $\epsilon = \frac{1}{\log n}$, so sieht man, dass mindestens $\frac{n}{\log n}$ der Vektoren v_i die Bedingung $\langle v_0, v_i \rangle \geq \frac{2}{\varkappa} - 1 - \frac{3}{\log n}$ erfüllen. Es sei $S := \{1 \leq i \leq n : \langle v_0, v_i \rangle \geq \frac{2}{\varkappa} - 1 - \frac{3}{\log n}\}$. Damit gilt $|S| \geq \frac{n}{\log n}$. Nun wird noch folgende Aussage benötigt:

> *Behauptung:* Seien $v_0, v_i, v_j \in \mathbb{R}^n$ drei Einheitsvektoren, so dass $v_i \neq v_0$, $v_j \neq v_0$ und $\langle (v_0 + v_i), (v_0 + v_j) \rangle = 0$. Seien v_i' und v_j' die normalisierten Projektionen von v_i und v_j auf die Hyperebene senkrecht zu v_0. Dann gilt
>
> $$\langle v_i', v_j' \rangle = -\sqrt{\frac{1 + \langle v_0, v_i \rangle}{1 - \langle v_0, v_i \rangle}} \cdot \sqrt{\frac{1 + \langle v_0, v_j \rangle}{1 - \langle v_0, v_j \rangle}}.$$
>
> *Beweis.* Zunächst sei $a_i = \langle v_i, v_0 \rangle$ und $a_j = \langle v_j, v_0 \rangle$. Dann ist
>
> $$v_i' = \frac{v_i - a_i v_0}{\|v_i - a_i v_0\|} = \frac{v_i - a_i v_0}{\sqrt{1 - a_i^2}},$$

[5] Wäre diese Behauptung nicht erfüllt (also $x_i < \frac{\gamma-\epsilon}{1-\epsilon}\ \forall i$), dann ist $\sum_{i=1}^{n} < (1 - \epsilon)n \cdot \frac{\gamma-\epsilon}{1-\epsilon} + \epsilon n = \gamma n$ ein Widerspruch.

da v_i ein Einheitsvektor ist. Analoges gilt für v'_j. Damit ist

$$\langle v'_i, v'_j \rangle = \frac{\langle (v_i - a_i v_0), (v_j - a_j v_0) \rangle}{\sqrt{(1-a_i^2)(1-a_j^2)}} = \frac{\langle v_i, v_j \rangle - a_i a_j}{\sqrt{(1-a_i^2)(1-a_j^2)}}.$$

Wegen $\langle (v_0 + v_i), (v_0 + v_j) \rangle = 0$ ist $\langle v_i, v_j \rangle = -1 - a_i - a_j$. Somit ergibt sich

$$\langle v'_i, v'_j \rangle = -\frac{(1+a_i)(1+a_j)}{\sqrt{(1-a_i^2)(1-a_j^2)}} = -\sqrt{\frac{(1+a_i)(1+a_j)}{(1-a_i)(1-a_j)}},$$

woraus dann durch Rücksubstitution von a_i und a_j die behauptete Eigenschaft entsteht. □

Nun wird der Beweis zu Lemma 4.12 weiter ausgeführt: Es ist $S = \{i : \langle v_o, v_i \rangle \geq \beta\}$ mit $\beta = \frac{2}{\varkappa} - 1 - \frac{3}{\log n}$. Weiter ist bekannt, dass $|S| \geq \frac{n}{\log n}$ ist. Da $v_0, ..., v_n$ nur eine fast optimale Lösung des Maximierungsproblems darstellt, ist anzunehmen, dass $v_i \neq v_0 \ \forall i$ gilt. Anderenfalls kann v_0 etwas verändert werden, so dass die Bedingung erfüllt ist. Seien nun noch zwei Knoten $i, j \in S$ durch eine Kante $\{i, j\} \in E$ verbunden. Dann ist $\langle v_0, v_i \rangle \geq \beta$, $\langle v_0, v_j \rangle \geq \beta$ und $\langle (v_0 + v_i), (v_0 + v_j) \rangle = 0$. Seien nun v'_i und v'_j die normalisierten Projektionen von v_i und v_j auf die Hyperebene senkrecht zu v_0. Der Ausdruck $\langle v'_i, v'_j \rangle$ ist in beiden Richtungen ($\langle v_0, v_i \rangle$ und $\langle v_0, v_j \rangle$) monoton fallend. Damit ist

$$\langle v'_i, v'_j \rangle \leq -\frac{1+\beta}{1-\beta} = -\frac{1}{\varkappa - 1 + O(\frac{1}{\log n})}.$$

Nach Definition 3.6 entsteht damit wie gefordert eine Vektor-$(\varkappa + O(\frac{1}{\log n}))$-Färbung von $G[S]$. □

Lemma 4.13. *Sei $\varkappa \geq 1$ und $G = (V, E)$ ein Vektor-$\varkappa$-färbbarer Graph mit n Knoten. Dann kann eine unabhängige Menge der Größe $\tilde{\Omega}(n^{f(\varkappa)})$ in Polynomialzeit gefunden werden.*

Beweis. Der Beweis läuft per Induktion über $k = \lfloor \varkappa \rfloor$. Sei zuerst $k = 1$. Ein Graph ist nur dann Vektor-$\varkappa$-färbbar, für $\varkappa < 2$, wenn der Graph keine Kanten enthält. Damit formt die Knotenmenge V eine unabhängige Menge der Größe $n = n^{f(\varkappa)}$.
Sei nun $k \geq 2$ und Δ der maximale Grad des Graphen G. Es gibt (mindestens) zwei Möglichkeiten, eine unabhängige Menge in G zu finden. Mit Algorithmus 4.5 wird eine unabhängige Menge der Größe $\tilde{\Omega}(\frac{n}{\Delta^{1-2/\varkappa}})$ gefunden. Alternativ sei v ein Knoten mit Grad Δ und $N(v)$

seine Nachbarschaft. Der Subgraph $G[N(v)]$ ist Vektor-$(\varkappa-1)$-färbbar. Nach der Induktionsannahme kann in $G[N(v)]$ eine unabhängige Menge der Größe $\tilde{\Omega}(\Delta^{f(\varkappa-1)})$ gefunden werden. Die größere dieser beiden unabhängigen Mengen hat eine Größe von

$$\tilde{\Omega}\left(max\left\{\frac{n}{\Delta^{1-\frac{2}{\varkappa}}}, \Delta^{f(\varkappa-1)}\right\}\right) \geq \tilde{\Omega}\left(n^{\frac{1}{1+\frac{1-2/\varkappa}{f(\varkappa-1)}}}\right) = \tilde{\Omega}(n^{f(\varkappa)}),$$

wie oben beschrieben. □

Nun kann Satz 4.11 bewiesen werden:

Beweis. (*Satz 4.11*) Der Graph $G = (V, E)$ enthält eine unabhängige Menge der Größe $\frac{n}{\varkappa}$. Nach Lemma 4.12 kann mit $\varkappa' = \varkappa + O(\frac{1}{\log n})$ eine Knotenmenge $S \subseteq V$ mit mindestens $\frac{n}{\log n}$ Knoten und eine Vektor-$\varkappa'$-Färbung von $G[S]$ gefunden werden. Mit $|S| = \tilde{\Omega}(n)$ und $f(\varkappa') = f(\varkappa) + O(\frac{1}{\log n})$ bedeutet dies für die unabhängige Menge eine Größe von $\tilde{\Omega}(n^{f(\varkappa)})$, wie behauptet. □

Mit diesen Eigenschaften ergibt sich nun der folgende Algorithmus, der einen k-färbbaren Graphen mit $\tilde{O}(n^{\alpha_k})$ Farben färbt, wobei α_k folgende Rekursion erfüllt:

$$\alpha_2 = 0,\ \alpha_3 = \tfrac{3}{14},$$
$$\alpha_k = 1 - \frac{6}{k+4+3(1-\frac{2}{k})\frac{1}{1-\alpha_{k-2}}}.$$

Nach jedem Schritt kann der Algorithmus abgebrochen werden, sofern ein Fortschritt erzielt wurde.

Algorithmus 4.7. ***Combined-Color*(*G*,*k*)**

Eingabe: Ein k-färbbarer Graph $G = (V, E)$ mit n Knoten.

Ausgabe: Eine $\tilde{O}(n^{\alpha_k})$-Färbung von G.

1. `if` $k = 2 \Rightarrow$ Färbe G mit 2 Farben in Linearzeit.
2. `if` $k = 3 \Rightarrow$ Färbe G mit Algorithmus 3.9 mit $\tilde{O}(n^3/14)$ Farben.
3. Entferne nacheinander aus G Knoten v mit $d(v) < n^{\frac{\alpha_k}{1-2/k}}$ im aktuellen Graphen. Sei U die Menge der entfernten Knoten.
   `if` $|U| \geq \frac{n}{2} \Rightarrow$

· *KMS-k-Color*($G[U]$,k) (Algorithmus 4.5)

· *Progress-1* ($|I| = \tilde{\Omega}(n^{1-\alpha_k})$, wenn I die größte Farbklasse in $G[U]$ ist.)

4. ($|U| < \frac{n}{2}$) $W := V \backslash U$

   `for each` $u, v \in W$:

   · $S := N(u) \cap N(v)$

   · `if` $|S| \geq n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}} \Rightarrow$

   ... *Combined-Color*($G[S]$,$k-2$)

   ... `if` $\tilde{O}(|S|^{\alpha_{k-2}})$ Farben wurden verwendet $\Rightarrow$ *Progress-1*

   ... `else` *Progress-3*

5. Konstruiere mithilfe von Satz 4.17 eine Sammlung $\mathcal{T}$ von Mengen $T \subseteq W$, so dass eine der Mengen T die Bedingung $|T| = \tilde{\Omega}\left(n^{\frac{2\alpha_k}{1-2/k} - \frac{1-\alpha_k}{1-\alpha_{k-2}}}\right)$ erfüllt und T eine unabhängige Menge der Größe $(\frac{1}{k-1} - O(\frac{1}{\log n}))|T|$ enthält.
   Wende den Algorithmus von Alon und Kahale [2] auf $G[T]$, $T \in \mathcal{T}$ an. In mindestens einem Durchlauf wird eine unabhängige Menge I gefunden mit $|I| = \tilde{\Omega}\left(n^{\left(\frac{2\alpha_k}{1-2/k} - \frac{1-\alpha_k}{1-\alpha_{k-2}}\right)\frac{3}{k}}\right) = \tilde{\Omega}(n^{1-\alpha_k})$ (Gleichung (4.12)). *Progress-1.*

Behauptung: Die in Schritt 5 gefundene unabhängige Menge enthält $\tilde{\Omega}(n^{1-\alpha_k})$ Knoten.

Beweis. Es wird eine unabhängige Menge I der Größe $\tilde{\Omega}\left(n^{\left(\frac{2\alpha_k}{1-2/k} - \frac{1-\alpha_k}{1-\alpha_{k-2}}\right)\frac{3}{k}}\right)$ gefunden. Mit $\alpha_k = 1 - \frac{6}{k+4+3\frac{1-2/k}{1-\alpha_{k-2}}}$ ergibt sich

$$\begin{aligned}
\left(\frac{2\alpha_k}{1-\frac{2}{k}} - \frac{1-\alpha_k}{1-\alpha_{k-2}}\right)\frac{3}{k} &= \frac{3}{k}\left(\frac{2\left(1 - \frac{6}{k+4+3\frac{1-2/k}{1-\alpha_{k-2}}}\right)}{1-\frac{2}{k}} - \frac{6}{(1-\alpha_{k-2})\left(k+4+3\frac{1-2/k}{1-\alpha_{k-2}}\right)}\right) \\
&= \frac{3}{k} \cdot \frac{2\left(k-2+3\frac{1-2/k}{1-\alpha_{k-2}}\right)(1-\alpha_{k-2}) - 6\left(1-\frac{2}{k}\right)}{\left(k+4+3\frac{1-2/k}{1-\alpha_{k-2}}\right)\left(1-\frac{2}{k}\right)(1-\alpha_{k-2})} \\
&= \frac{3}{k} \cdot \frac{2(k-2)(1-\alpha_{k-2})}{\left(k+4+3\frac{1-2/k}{1-\alpha_{k-2}}\right)\left(1-\frac{2}{k}\right)(1-\alpha_{k-2})} \\
&= \frac{6}{k+4+3\frac{1-2/k}{1-\alpha_{k-2}}} = 1-\alpha_k, \qquad (4.12)
\end{aligned}$$

was der Behauptung entspricht. □

Satz 4.14. *Algorithmus 4.7 färbt jeden k-färbbaren Graphen mit n Knoten mit* $\tilde{O}(n^{\alpha_k})$ *Farben, wobei* $\alpha_2 = 0$, $\alpha_3 = \frac{3}{14}$ *und* $\alpha_k = 1 - \frac{6}{k+4+\frac{3(1-2/k)}{1-\alpha_{k-2}}}$ *für* $k \geq 4$ *sind.*

Beweis. Der Algorithmus wird nun Schritt für Schritt betrachtet. Sobald ein Fortschritt erzielt wird, kann gestoppt und der Algorithmus neu gestartet werden; diesmal mit weniger Knoten. Zunächst werden die Rekursionsanfänge definiert. Die verwendeten Algorithmen sind korrekt und laufen in Polynomialzeit. Danach werden Knoten mit geringem Grad aus G entnommen. Wenn D der durchschnittliche Grad der Knoten aus $G[U]$ ist, dann gilt $D \leq 2n^{\frac{\alpha_k}{1-2/k}}$. Enthält die Menge U mindestens $\frac{n}{2}$ Knoten, dann liefert Algorithmus 4.5 eine $\tilde{O}(D^{1-2/k})$-Färbung und damit eine unabhängige Menge der Größe $\tilde{\Omega}\left(\frac{n}{D^{1-2/k}}\right) = \tilde{\Omega}(n^{1-\alpha_k})$.
Im anderen Fall ist $|U| < \frac{n}{2}$ und damit $|W| = |V \setminus U| \geq \frac{n}{2}$. Weiterhin gilt für den minimalen Grad d_{min} von $G[W]$: $d_{min} \geq n^{\frac{\alpha_k}{1-2/k}}$. Nun wird für jedes Knotenpaar $u, v \in W$ die Menge $S = N(u) \cap N(v)$ betrachtet (Hier können die Nachbarn in G betrachtet werden, nicht nur die in $G[W]$!). Wenn diese Menge groß ist, dann kann der Algorithmus *Combined-Color* rekursiv auf $G[S]$ angewendet und die Anzahl der dabei verwendeten Farben bestimmt werden. Dabei werden folgende Fälle unterschieden:

a) Die Knoten u und v haben in einer zulässigen k-Färbung des Graphen G verschiedene Farben. Dann muss S auf jeden Fall $k-2$-färbbar sein, da sich S aus der gemeinsamen Nachbarschaft zweier Knoten zusammensetzt. Die rekursive Anwendung des Algorithmus auf S liefert dann eine $\tilde{O}(|S|^{\alpha_{k-2}})$-Färbung. Aus dieser Färbung ist eine unabhängige Menge der Größe $\tilde{\Omega}(|S|^{1-\alpha_{k-2}}) = \tilde{\Omega}(n^{1-\alpha_k})$ leicht zu extrahieren. Damit lässt sich ein Fortschritt erzielen.

b) Die Knoten u und v haben in jeder zulässigen k-Färbung des Graphen G dieselbe Farbe. Dann ist $G[S]$ nicht unbedingt $k-2$-färbbar, und es kann sein, dass die rekursive Anwendung des Algorithmus auf $G[S]$ mehr Farben als $\tilde{O}(|S|^{\alpha_{k-2}})$ Farben benötigt. Wenn dies auftritt, dann können ohne Probleme die beiden Knoten u und v miteinander verschmolzen und ein Fortschritt erzielt werden.

Auf jeden Fall wird ein Fortschritt erzielt, wenn es zwei Knoten $u, v \in W$ gibt, die viele Nachbarn gemeinsam haben.
Schließlich kann es noch sein, dass es kein Knotenpaar im Graphen gibt, das mindestens $s = n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}$ gemeinsame Nachbarn besitzt. Dann kann der Satz von Blum [6] (Satz 4.2

bzw. 4.17) angewendet werden. Es wird eine Sammlung $\mathcal{T}$ von Teilmengen $T \subset W$ konstruiert, von denen mindestens eine relativ groß ist ($|T| = \tilde{\Omega}\left(\frac{d_{\min}^2}{s}\right)$) und eine große unabhängige Menge I enthält ($|I| \geq (\frac{1}{k-1} - O(\frac{1}{\log n}))|T|$). Da nicht bekannt ist, welche der Mengen $T \in \mathcal{T}$ diese Bedingungen erfüllen, müssen alle diese Mengen darauf überprüft werden. Dies geschieht mit der Erweiterung des Algorithmus von Alon und Kahale [2] (Satz 4.11). Da $\mathcal{T}$ nur $\tilde{O}(n)$ Mengen umfasst, ist dieser Schritt in Polynomialzeit abgeschlossen. Für mindestens eine der Mengen gibt der Algorithmus von Alon und Kahale eine unabhängige Menge der Größe $\tilde{\Omega}\left(n^{(\frac{2\alpha_k}{1-2/k} - \frac{1-\alpha_k}{1-\alpha_{k-2}})\cdot\frac{3}{k}}\right) = \tilde{\Omega}(n^{1-\alpha_k})$ zurück. Damit wird dann ein Fortschritt *Progress-1* erzielt. □

Es bleibt der Beweis von Satz 4.2. Dieser Satz stammt in den grundlegenden Überlegungen von Blum [6] und wurde von Halperin, Nathaniel und Zwick [20] in etwas abgewandelter Form bewiesen. Es wird behauptet, dass unter gewissen Bedingungen schnell eine große Knotenmenge gefunden werden kann, die ihrerseits eine große unabhängige Menge enthält. Für den Beweis des Satzes (hier als Satz 4.17) werden die folgenden Lemmata benötigt:

Lemma 4.15. *Seien $x_1, ..., x_n \geq 0$ und $y_1, ..., y_n \geq 0$ so, dass $\sum_{i=1}^n x_i = \gamma n$ und $\sum_{i=1}^n x_i \geq \beta \sum_{i=1}^n y_i$ für $\gamma, \beta > 0$. Dann gibt es für jedes $0 < \delta \leq 1$ mindestens einen Index $i \in \{1, ..., n\}$, für den gilt:*

$$x_i \geq \delta\gamma \ , \ x_i \geq (1-\delta)\beta y_i.$$

Beweis. Sei $I = \{i \in \{1, ..., n\} : x_i \geq \delta\gamma\}$. Dann gilt:

$$\begin{aligned}\sum_{i\in I} x_i &= \sum_{i=1}^n x_i - \sum_{i\notin I} x_i \geq \gamma n - \delta\gamma n + \delta\gamma|I| \\ &\geq (1-\delta)\gamma n = (1-\delta)\sum_{i=1}^n x_i \geq (1-\delta)\beta\sum_{i=1}^n y_i \\ &\geq (1-\delta)\beta\sum_{i\in I} y_i.\end{aligned}$$

Damit muss es nach dem Schubfachprinzip mindestens ein $i \in I$ geben, so dass $x_i \geq (1-\delta)\beta y_i$ gilt. □

Lemma 4.16. *Sei $G = (V, E)$ ein Graph. Seien $\delta > 0$ sowie $R \subseteq V$ und $U \subseteq V \setminus R$ so, dass $d(v) \in [d, d(1+\delta))\ \forall v \in U$. Wenn $D_R(U) \geq \lambda D(U)$ für ein $\lambda > 0$ gilt, dann gibt es einen Knoten $v \in R$, so dass $D(N(v) \cap U) \geq (1-\delta)\lambda D(N(v) \cap U)$ gilt.*

Beweis. Angenommen, die Gleichung ist für alle Knoten $v \in R$ nicht erfüllt. Aufsummiert ergibt sich dann folgende Ungleichung:

$$\sum_{v \in R} D_R(N(v) \cap U) < (1-\delta)\lambda \sum_{v \in R} D(N(v) \cap U). \tag{4.13}$$

Die linke Seite ergibt

$$\sum_{v \in R} D_R(N(v) \cap U) = \sum_{v \in R} \sum_{u \in N(v) \cap U} d_R(u) = \sum_{u \in U} (d_R(u))^2,$$

da jeder Knoten $u \in U$ genau $d_R(u)$ viele Nachbarn einbringt und genau $d_R(u)$-mal aufgerufen wird. Auf der rechten Seite gilt analog

$$\sum_{v \in R} D(N(v) \cap U) = \sum_{v \in R} \sum_{u \in N(v) \cap U} d(u) = \sum_{u \in U} d_R(u) d(u),$$

da wiederum jeder Knoten $u \in U$ genau $d_R(u)$-mal aufgerufen wird, diesmal aber $d(u)$ viele Nachbarn einbringt. Mit der gegebenen oberen Schranke für $d(u)$ ergibt sich

$$\sum_{v \in R} D(N(v) \cap U) < d(1+\delta) \sum_{u \in U} d_R(u).$$

Für die Ungleichung (4.13) resultiert dann mit Hilfe der Cauchy-Schwartzschen Ungleichung

$$\begin{aligned} D_R(U) = \sum_{u \in U} d_R(u) &\leq \frac{|U| \sum_{u \in U} d_R(u)^2}{\sum_{u \in U} d_R(u)} \\ &\leq \lambda d \cdot (1-\delta)(1+\delta) \cdot |U| < \lambda D(U), \end{aligned}$$

und damit ein Widerspruch zur Anfangsbedingung. □

Satz 4.17. *Sei $G = (V, E)$ ein k-färbbarer Graph mit n Knoten und minimalem Grad d_{min}. Weiterhin gibt es in G kein Knotenpaar u, v, das mehr als s gemeinsame Nachbarn besitzt. Dann ist es möglich, in Polynomialzeit eine Sammlung $\mathcal{T}$ von Teilmengen $T \subseteq V$ zu konstruieren, so dass mindestens eine Menge $T \in \mathcal{T}$ die folgenden Bedingungen erfüllt: $|T| = \tilde{\Omega}(\frac{d_{min}^2}{s})$, und T enthält eine unabhängige Menge der Größe $(\frac{1}{k-1} - O(\frac{1}{\log n}))|T|$.*

Beweis. Sei $R \subseteq V$ die größte Farbklasse in einer zulässigen k-Färbung von G, d.h. R ist die Farbe, die die meisten Knoten umfasst. Das bedeutet, $D_R(V \setminus R) = D(R) \geq \frac{D(V \setminus R)}{k-1}$. Benötigt werden nun noch $\delta = 1/\log n$ und $I_j = \{v \in V \setminus R : d(v) \in [(1+\delta)^j, (1+\delta)^{j+1})\}$ für $j = 0, 1, ..., log_{1+\delta} n$.

Die Werte $x_j = D_R(I_j)$ und $y_j = D(I_j)$ erfüllen dann die Voraussetzungen von Lemma 4.15:

$$\sum_{j=0}^{\log_{1+\delta} n} x_j = \sum_{j=0}^{\log_{1+\delta} n} D_R(I_j) = D_R(V \setminus R) = D(R)$$

und

$$\sum_{j=0}^{\log_{1+\delta} n} y_j = \sum_{j=0}^{\log_{1+\delta} n} D(I_j) = D(V \setminus R) \leq \frac{D(R)}{k-1} = \frac{1}{k-1} \sum_{j=0}^{\log_{1+\delta} n} x_j.$$

Nach Lemma 4.15 gibt es nun eine Menge I_j, so dass gilt:

$$D_R(I_j) \geq \delta \frac{D_R(V \setminus R)}{\log_{1+\delta} n}\ ,\ D_R(I_j) \geq (1-\delta)\frac{D(I_j)}{k-1}. \tag{4.14}$$

Nun werden aus dem Graphen alle Knoten $v \in R$, die *wenige* Nachbarn in I_j haben, entfernt. Das heißt, es wird nur noch die Menge $R' := \{v \in R : d_{I_j}(v) \geq \frac{\delta^2 d_{min}}{\log_{1+\delta} n}\}$ betrachtet. Nun ergibt sich

$$\begin{aligned} D_{R'}(I_j) &= D_R(I_j) - \sum_{v \notin R'} d_{I_j}(v) > D_R(I_j) - \frac{\delta^2 d_{min}|R \setminus R'|}{\log_{1+\delta} n} \\ &\geq D_R(I_j) - \delta^2 \frac{D_R(V \setminus R)}{\log_{1+\delta} n} \geq (1-\delta) D_R(I_j) \geq (1-\delta^2)\frac{D(I_j)}{k-1}. \end{aligned} \tag{4.15}$$

Mit Gleichung 4.15 sind nun die Voraussetzungen für Lemma 4.16 erfüllt: In den Bezeichnungen von Lemma 4.16 seien $U := I_j$, $R := R'$ und $\lambda = \frac{(1-\delta)^2}{k-1}$. Nach Lemma 4.16 gibt es einen Knoten $v \in R'$, so dass gilt:

$$D_{R'}(N(v) \cap I_j) \geq (1-\delta)^3 D(N(v) \cap I_j). \tag{4.16}$$

Es sei $S := N(v) \cap I_j$ die Menge der Nachbarn von v innerhalb der Menge I_j. Da $v \in R'$ ist, folgt

$$|S| = |N(v) \cap I_j| \geq \frac{\delta^2 d_{min}}{\log_{1+\delta} n} = \tilde{\Omega}(d_{min}) \tag{4.17}$$

und $D_{R'}(S) \geq \frac{(1-\delta)^3}{k-1} \cdot D(N(v) \cap I_j)$ nach Gleichung (4.16).

Da $S \subseteq N(v)$ ist, gilt für jeden Knoten $u \in V$ laut Voraussetzung des Satzes $|N(u) \cap S| \leq s$, insbesondere also auch für jeden Knoten $u \in N(S)$. Mit $D(S) = \sum_{w \in S} d(w)$ ergibt dies

$$|N(S)| \geq \frac{D(S)}{s} = \frac{1}{s} \sum_{w \in N(v) \cap I_j} d(w) \geq \frac{d_{min}}{s}|N(v) \cap I_j| \geq \frac{\delta^2 d_{min}^2}{s \log_{1+\delta} n} = \tilde{\Omega}\left(\frac{d_{min}^2}{s}\right).$$

Hätten jetzt alle Knoten in $N(S)$ gleich viele Nachbarn in S, dann wären der Beweis beendet. Es würde dann $|N(S) \cap R'| \geq \frac{D'_R(S)}{s} \geq \frac{(1-\delta)^3|N(S)|}{k-1}$ gelten, was bedeutet, dass der Anteil der

Knoten aus $R \supseteq R'$ in der Menge $N(S)$ $\tilde{\Omega}(\frac{1}{k-1})$ entspricht. Allerdings können die Knoten in $N(S)$ durchaus noch stark unterschiedliche Knotengrade in die Menge S besitzen.
Aus diesem Grunde werden die Knoten aus $N(S)$ erneut nach der Anzahl ihrer Nachbarn in S partitioniert. Es werden die Mengen $N_i(S) := \{u \in N(S) : d_S(u) \in [(1+\delta)^i, (1+\delta)^{i+1})\},\ i = 0, ..., \log_{1+\delta} n$ konstruiert. Mit den Werten $x_i = D_{R' \cap N_i(S)}(S)$ und $y_i = D_{N_i(S)}(S)$ sind wieder die Voraussetzungen für Lemma 4.15 gegeben: Es gilt

$$\sum_{i=0}^{\log_{1+\delta} n} x_i = D_{R'}(S) \geq \frac{(1-\delta)^3}{k-1} D(S) = \frac{(1-\delta)^3}{k-1} \sum_{i=0}^{\log_{1+\delta} n} y_i.$$

Nach Lemma 4.15 gibt es nun eine Menge $N_i(S)$, so dass gilt:

$$\begin{aligned} D_{R' \cap N_i(S)}(S) &\geq \delta \frac{D_{R'}(S)}{\log_{1+\delta} n} && \text{wegen Lemma 4.15} \\ &\geq \frac{\delta(1-\delta)^3}{(k-1)\log_{1+\delta} n} D(S) && \text{nach Gleichung (4.16)} \\ &\geq \frac{\delta^3(1-\delta)^3}{(k-1)\log^2_{1+\delta} n} d^2_{min} && \text{da } D(S) \geq \frac{\delta^2 d^2_{min}}{\log_{1+\delta} n} \end{aligned} \tag{4.18}$$

und

$$D_{R' \cap N_i(S)}(S) \geq \frac{(1-\delta)^4}{k-1} D_{N_i(S)}(S).$$

Es gilt nun nach der Definition von $N_i(S)$:

$$D_{R' \cap N_i(S)}(S) \geq \frac{(1-\delta)^4}{k-1} D_{N_i(S)}(S) = \frac{(1-\delta)^4}{k-1} \sum_{w \in N_i(S)} d_S(w) \geq \frac{(1-\delta)^4}{k-1} (1+\delta)^i |N_i(S)|. \tag{4.19}$$

Wie oben gilt für jeden Knoten $u \in V$ die Beziehung $|N(u) \cap S| \leq s$; insbesondere gilt dies für jeden Knoten $u \in N_i(S)$. Damit gilt einerseits

$$|N_i(S)| \geq \frac{D_{N_i(S)}(S)}{s} = \tilde{\Omega}\left(\frac{d^2_{min}}{s}\right), \tag{4.20}$$

und andererseits

$$\begin{aligned} |N_i(S) \cap R'| &\geq \frac{D_{R' \cap N_i(S)}(S)}{(1+\delta)^{i+1}} \\ &\geq \frac{(1-\delta)^4 (1+\delta)^i}{(k-1)(1+\delta)^{i+1}} |N_i(S)| && \text{(nach Gleichung (4.19))} \\ &= \frac{1}{1-\delta^2} \cdot \frac{(1-\delta)^5}{k-1} |N_i(S)| \\ &\geq \frac{(1-\delta)^5}{k-1} |N_i(S)|, && \text{(da } \frac{1}{1-\delta^2} > 1\text{).} \end{aligned} \tag{4.21}$$

Mit $R' \subseteq R$ erhält man dann $|N_i(S) \cap R| \geq |N_i(S) \cap R'| \geq \frac{(1-\delta)^5}{k-1}|N_i(S)|$, und die zu beweisenden Eigenschaften sind mit den Gleichungen (4.20) und (4.21) gezeigt. Eine letzte Bemerkung ist noch zu machen: Natürlich ist nicht bekannt, welche Menge die Menge R ist. Aus diesem Grunde kann die Menge I_j nur aus sämtlichen Knoten zusammengestellt werden. Die sich ergebenden Beziehungen bleiben allerdings bestehen, da sich innerhalb der Menge $R \cap I_j$ keine Kanten befinden. Da die Menge R unbekannt ist, müssen auch sämtliche Kombinationen v, i, j ausprobiert werden. Deshalb wurde die Anzahl der Mengen in der Sammlung $\mathcal{T}$ polynomiell beschränkt. □

In Tabelle 4.4 sind die Algorithmen aus diesem Abschnitt noch einmal mit ihrer Güte zusammengefasst. Man erkennt deutlich, dass die Ergebnisse der Algorithmen 4.6 und 4.7 stark von Algorithmus 4.5 abhängen, da sie bei größeren Werten von k kaum eine Veränderung zu Algorithmus 4.5 zeigen. Die restlichen Zutaten (hauptsächlich aus Satz 4.17) können kaum eine Verbesserung erzielen.

	k =3	4	5	6	7
Algorithmus 3.11	3/14	14/25	25/36	36/47	47/58
$k_0 = 3$	0.214	0.56	0.694	0.766	0.810
Algorithmus 4.3	3/14	3/5	175/263	105/137	10881/13697
[6], $k_0 = 2; k_1 = 3$	0.214	0.6	0.665	0.766	0.794
Algorithmus 4.5	1/4	2/5	1/2	4/7	5/8
[24]	0.25	0.4	0.5	0.571	0.625
Algorithmus 4.6	3/14	7/18	54/109	218/383	383/614
[37]	0.214	0.389	0.495	0.569	0.624
Algorithmus 4.7	3/14	7/19	97/207	43/79	1391/2315
[20]	0.214	0.368	0.469	0.544	0.601

Tabelle 4.4: Exponenten α_k der bisherigen rekursiven Färbungsalgorithmen im Vergeich mit den Algorithmen von Xie, Ono und Hirata [37] bzw. Halperin, Nathaniel und Zwick [20].

Von dieser Position aus kann man sich nun in zwei Richtungen fortbewegen: Einerseits kann man nach Algorithmen suchen, die für Graphen mit großen Knotengraden eine stärkere Verbesserung erzielen, so wie es Halperin, Nathaniel und Zwick [20] taten, und diese mit Algorithmus 4.5 kombinieren, oder man sucht sich einen Algorithmus, der für Graphen mit kleinem maximalen Knotengrad bessere Ergebnisse erzielt.

4.3 Laufzeiten

Bisher wurden die vorgestellten Approximationsalgorithmen als Polynomialzeitalgorithmen bezeichnet. Allerdings gibt es trotzdem noch deutliche Unterschiede zwischen den Laufzeiten der einzelnen Algorithmen, die hier näher ausgeführt werden sollen. Es stellt sich immer wieder die Frage, ob man einen schnellen Algorithmus wie z.B. die Greedy-Strategie oder den DSATUR-Algorithmus verwenden sollte, oder lieber ein gutes Approximationsergebnis mit dem Algorithmus von Halperin, Nathaniel und Zwick [20] erzielen möchte. Als dritte Alternative steht dann immer noch ein exakter, aber in Exponentialzeit laufender Algorithmus, der aber auf Grund der geringen Problemgröße noch anwendbar sein könnte.

Hier werden die Laufzeiten der in diesem Kapitel vorgestellten Algorithmen betrachtet, einschließlich des Algorithmus 2.12, der als Grundstufe aller hier vorgestellten Algorithmen gelten darf.

Zunächst wird die Laufzeit von Algorithmus 2.12 von Wigderson [36] analysiert. Dieser Algorithmus besteht aus zwei Teilen, nämlich Algorithmus 2.11, dem rekursiven Verfahren, und 2.9 von Johnson [23]. Nach Korollar 2.21 ist die Laufzeit von Algorithmus 2.11 nur linear von der Knoten- und Kantenzahl abhängig ($O(|V| + |E|)$), wenn $\chi(G) \leq k$ für ein festes k gilt. Im Allgemeinen wird eine Laufzeit von $O(\chi(G) \cdot (|V| + |E|))$ für Algorithmus 2.11 angegeben. Analog dazu kann Algorithmus 2.9 in der Zeit $O(|V|^2)$ laufen. Somit ergibt sich für Algorithmus 2.12 eine Gesamtlaufzeit von $O(|V|^2)$. In Kapitel 7 wurde das Laufzeitverhalten von Algorithmus 2.12 bei einigen Beispielinstanzen untersucht.

Die folgenden Algorithmen verwenden als Rekursionsbasis hauptsächlich den Algorithmus 3.9 von Blum und Karger [7]. Daher folgt hier die Laufzeitanalyse für diesen Algorithmus.

Algorithmus 3.9 benutzt, solange der durchschnittliche Grad maximal $n^{9/14}$ beträgt, den Algorithmus von Karger, Motwani und Sudan [24], also Algorithmus 3.8. Mit Hilfe der Semidefiniten Programmierung entsteht eine $O(\Delta^{1/3})$-Färbung eines Teilgraphen mit $\Delta = O(n^{9/14})$. Die Laufzeit des Verfahrens wird von Klein und Lu [27] mit $O(|V| \cdot |E|)$ angegeben, da Semidefinite Optimierungsprobleme, die aus Graphproblemen entstehen, eine einfach zu handhabende Struktur aufweisen. In diesem Fall schlägt also der erste Teil des Algotithmus mit $O(|V| \cdot \Delta|V|) = O(|V|^2 \cdot n^{9/14}) = O(n^{37/14})$ zu Buche.

Der zweite Teil von Algorithmus 3.9 versucht, einen Fortschritt innerhalb von dichten Regionen des Graphen zu finden. Hierzu wird jedes Tripel (v, i, j) mit $v \in V$, $i, j \in \{0, 1, ..., 5 \log n\}$

betrachtet. Jede Menge $T_{v,i,j}$ kann theoretisch den gesamten Graphen umfassen, d.h. es gibt maximal $O(n \log^2 n)$ Mengen mit jeweils maximal n Knoten. Jede der Mengen $T_{v,i,j}$ muss mit Algorithmus 3.2 überprüft werden, um eine große unabhängige Menge zu finden. Die Laufzeit von Algorithmus 3.2 beträgt laut [6] $O(|V| \cdot |E|)$. Schließlich wird Algorithmus 3.6 mit jeder Menge $T_{v,i,j}$ unter bestimmten Bedingungen angewendet. Bei großzügiger Abschätzung (alle Mengen werden mit $O(n)$ angenommen) erhält man eine Laufzeit von $O(n^4)$. Insgesamt ergibt sich also für Algorithmus 3.9 eine Laufzeit von $O(n^{37/14} + n \log^2 n \cdot (nm + n^4)) = \tilde{O}(n^5)$.
Nun wird die Laufzeit von Algorithmus 4.5 betrachtet. Bei der semidefiniten Optimierungsaufgabe 3.1 ist es ohne Bedeutung, welchen Wert k hat. Die semidefinite Programmierung wird immer nach einer Zeit von $\tilde{O}(nm)$ abgeschlossen sein [27]. Die von Klein und Lu generierte Lösung wird zusätzlich noch schnell faktorisierbar sein, so dass man auch ohne die aufwändige Cholesky-Zerlegung der resultierenden Matrix auskommt. Beim Rundungsschritt wird jeder Knoten genau einmal angeschaut und gefärbt ($O(|V|)$). Allerdings kann es sein, dass einige Knoten unzulässig gefärbt werden und die Rundung für diese Knoten erneut vorgenommen werden muss. Insgesamt ergibt sich eine Laufzeit von $\tilde{O}(|V| \cdot |E| + |V|) = \tilde{O}(|V| \cdot |E|)$.
Die restlichen Algorithmen verwenden für k-färbbare Graphen rekursiv den Algorithmus für $(k-1)$- bzw. $(k-2)$-färbbare Graphen. Somit gilt als Rekursionsanfang jeweils der aktuell beste Algorithmus (also die Algorithmen 1.1 für 2-färbbare Graphen und 3.9 für 3-färbbare Graphen).
Algorithmus 4.6 ist einfach eine Variante von Algorithmus 4.5 mit dem etwas besseren Algorithmus 3.9 als Rekursionsanfang. Die Laufzeit wird damit durch $\tilde{O}(k|V^5|)$ dominiert.
Der wichtigste Algorithmus ist das Verfahren von Halperin, Nathaniel und Zwick [20] (Algorithmus 4.7). Wenn hier gezeigt werden kann, dass die Laufzeit des Algorithmus nicht wesentlich größer ist als die bisher erhaltenen Laufzeiten, dann ergibt sich auch ein praktikabler Algorithmus, der neben einer guten Approximation auch eine schnelle Laufzeit besitzt. Die zentralen Punkte von Algorithmus 4.7 sind der Rekursionsaufruf und die Untersuchung der Mengen $T_{v,i,j}$.
Die Mengen $T_{v,i,j}$ für $v \in V$, $i, j \in \{0, 1, ..., \log_{1+\delta} n\}$ werden immer dann untersucht, wenn $|T_{v,i,j}| = \tilde{\Omega}(\frac{d^2_{\min}}{s})$ gilt. Die andere Eigenschaft, nämlich dass T eine unabhängige Menge der Größe $|T|(\frac{1}{k-1} - O(\frac{1}{\log n}))$ enthält, kann nicht schnell überprüft werden, daher muss an dieser Stelle der Algorithmus von Alon und Kahale [2] angewendet werden. Nach Satz 4.17 erfüllt

mindestens eine Menge $T_{v,i,j}$ auch diese Eigenschaft.
Nun wird Satz 4.11 auf die Menge $T = T_{v,i,j}$ angewendet, um eine unabhängige Menge der Größe $|T|^{\frac{3}{k+1}}$ zu finden. Das bedeutet:

1. Bei $k = 1$ wird die Menge $|T|$ selbst ausgegeben.

2. Für $k \geq 2$ gilt nun folgendes: Algorithmus 4.5 findet eine unabhängige Menge der Größe $\tilde{\Omega}(\frac{n}{\Delta^{1-2/k}})$. Dies benötigt die Zeit $\tilde{O}(|T| \cdot E(G[T]))$.

3. Gleichzeitig wird ein Knoten v mit maximalem Grad Δ gewählt und der Algorithmus rekursiv mit der Nachbarschaft von v aufgerufen.

4. Zurückgegeben wird dann die größere der beiden unabhängigen Mengen.

Sei $A[n, k]$ die Laufzeit der Prozedur zu Satz 4.11 nach dem Algorithmus von Alon und Kahale [2]. Dann gilt:

$$\begin{aligned} A[|T|, k] &= \Delta(G[T]) \cdot \tilde{O}(|T| \cdot |E(G[T])|) + A[\Delta(G[T]), k-1], \\ A[\cdot, 1] &= 1. \end{aligned}$$

Über $\Delta(G[T])$ kann keine Aussage getroffen werden; im schlechtesten Fall ist aber $\Delta(G[T]) = |T| - 1$. Dies ergibt eine Abschätzung von $A[T, k] = \tilde{O}(|T|^2 \cdot |E(G[T])|)$.
Hinzu kommt nun der rekursive Aufruf von Algorithmus 4.7 selbst. Wenn es zwei Knoten gibt, deren Nachbarschaft sehr groß ist, dann wird diese Rekursion ausgeführt; man geht hier aber gleich um zwei Stufen nach unten.
Als Anfangsbedingungen stehen die Algorithmen 1.1 (Laufzeit $O(n + m)$) und 3.9 (Laufzeit $\tilde{O}(n^2 m)$). Wenn $H[n, k]$ die Laufzeit von Algorithmus 4.7 ist, dann gilt nun:

$$\begin{aligned} H[n, k] &\leq n^2 + O(n \cdot m) + H[n-2, k-2] + n \cdot \log_{1+\delta}^2 n \cdot A[n, k] \\ H[\cdot, 2] &= O(n + m) \\ H[\cdot, 3] &= \tilde{O}(n^2 m), \end{aligned}$$

da für alle Prozeduraufrufe möglichst große Mengen benötigt werden. Diese Mengen können jeweils maximal n Knoten besitzen. Es ergibt sich für diesen Fall eine Gesamtlaufzeit von $\tilde{O}(n^3 m) = \tilde{O}(n^5)$. Die Laufzeit von Algorithmus 4.7 basiert also hauptsächlich auf der Laufzeit von Algorithmus 3.9.

4.4 Ein allgemeiner Algorithmus

Mit den Beispielalgorithmen der vorangegangenen Abschnitte kann nun ein allgemeiner rekursiver Algorithmus beschrieben werden, der jeden k-färbbaren Graphen mit n Knoten mit $\tilde{O}(n^{\alpha_k})$ Farben in Polynomialzeit färbt. Hierzu werden folgende Komponenten verwendet:

1. Einen Algorithmus *Low-Degree*, der einen k-färbbaren Graphen mit mit durchschnittlichem Grad $\bar{d}$ mit $O(\bar{d}^{\beta_k})$ Farben färbt.

2. Einen Algorithmus *Find-IS*, der in einem k-färbbaren Graphen, der eine unabhängige Menge der Größe $\frac{n}{k-1}$ enthält, eine unabhängige Menge der Größe $\Omega(n^{\gamma_k})$ findet.

Beide Algorithmen können *beliebig* gewählt werden. Wenn ein neuer Algorithmus für (1.) oder (2.) gefunden wird, dann kann dieser prinzipiell problemlos eingesetzt und α_k neu berechnet werden.

Das Verfahren ist analog zu Algorithmus 4.7. Zunächst werden nacheinander Knoten mit kleinem Grad aus dem Graphen entfernt, bis eine Partition $V = U \dot{\cup} W$ entsteht, so dass sämtliche Knoten in $G[W]$ einen Grad von mindestens d_{min} besitzen. Daraus folgt automatisch, dass der durchschnittliche Grad $\bar{d}$ in $G[U]$ höchstens $2 \cdot d_{min}$ sein kann. Ist nun $G[U]$ groß, d.h. umfasst die Menge U mindestens $\frac{n}{2}$ Knoten, so wird Algorithmus *Low-Degree* angewendet, was in einer Färbung mit $O(\bar{d}^{\beta_k})$ Farben resultiert. Diese Färbung soll eine unabhängige Menge der Größe $\Omega(n^{1-\alpha_k})$ enthalten, damit ein Fortschritt erzielt werden kann. Mit $|U| = \Omega(n)$ ergibt sich eine unabhängige Menge der Größe $\Omega(\frac{n}{d^{\beta_k}})$.

Als nächstes werden die gemeinsamen Nachbarschaften von jeweils zwei Knoten $u, v \in W$ betrachtet. Wenn u und v verschiedene Farben bekommen, dann muss ihre gemeinsame Nachbarschaft $S = N(u) \cap N(v)$ $k-2$-färbbar sein. Wenn zudem $|S| \geq s$ gilt, dann kann der Algorithmus rekursiv auf S angewendet werden. Ist der Graph $G[S]$ tatsächlich $(k-2)$-färbbar, dann ergibt dies eine Färbung mit $O(s^{\alpha_{k-2}})$ Farben. Diese Färbung enthält eine unabhängige Menge der Größe $\Omega(s^{1-\alpha_{k-2}})$. Für einen Fortschritt muss dann asymptotisch gelten $n^{1-\alpha_k} = s^{1-\alpha_{k-2}}$. Werden dagegen im rekursiven Aufruf mehr als $O(s^{\alpha_{k-2}})$ Farben verwendet, dann kann daraus geschlossen werden, dass S nicht $(k-2)$-färbbar ist und damit die Knoten u und v dieselbe Farbe erhalten müssen.

Der dritte Schritt basiert auf Satz 4.17. Es ist ein k-färbbarer Graph $G[W]$ mit $\Omega(n)$ Knoten gegeben, der einen minimalen Grad d_{min} besitzt. Je zwei Knoten haben innerhalb von $G[W]$

maximal s gemeinsame Nachbarn. Nach Satz 4.17 kann man in Polynomialzeit eine Menge T finden, die folgende Eigenschaften besitzt: $t = |T| = \tilde{\Omega}(\frac{d_{min}^2}{s})$ und T enthält eine unabhängige Menge der Größe $\frac{t}{k-1}$. Mit dieser Menge T wird nun der Algorithmus *Find-IS* angewendet, welcher eine unabhängige Menge der Größe $\Omega(t^{\gamma_k})$ findet. Um einen Fortschritt zu erzielen, muss wieder asymptotisch gelten $t^{\gamma_k} = n^{1-\alpha_k}$.

Algorithmus 4.8. ***Final-Color*(*G*,*k*)**

Eingabe: Ein k-färbbarer Graph G mit n Knoten, Parameterfunktionen β_k und γ_k.

Ausgabe: Eine α_k-Färbung von G.

1. Entferne nacheinander Knoten v aus G mit $d(v) < n^{\alpha_k/\beta_k}$. Setze $U :=$ Menge der entfernten Knoten und $W := V \setminus U$.
2. `if` $|U| \geq \frac{n}{2} \Rightarrow$ Wende *Low-Degree* auf $G[U]$ an; *Progress-1*
3. `if` $\exists u, v \in W : |N(u) \cap N(v)| \geq n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}} \Rightarrow$ Setze $S := N(u) \cap N(v)$. Rufe *Final-Color*$(G[S],k-2)$ auf.
   `if` mehr als $|S|^{\alpha_{k-2}}$ Farben verwendet $\Rightarrow$ *Progress-3*
   `else` *Progress-1*
4. Erstelle eine Kollektion $\mathcal{T}$ von Mengen $T_{v,i,j} \subseteq W$ mit $T_{v,i,j} = N_i(N(v) \cap I_j)$, $N_i(U) = \{w \in N(U) : d_U(w) \in \left[(1+\delta)^i, (1+\delta)^{i+1}\right)\}$ für $U \subseteq V$, sowie $I_j = \{w \in W : d(w) \in \left[(1+\delta)^j, (1+\delta)^{j+1}\right)\}$, und wende *Find-IS* auf jeden Graphen $G[T_{v,i,j}]$ an. Einer der Durchläufe wird nach Satz 4.17 einen Fortschritt *Progress-1* erzielen.

Zusammenfassend muss also gelten

$$\begin{aligned} n^{1-\alpha_k} &= \frac{n}{\bar{d}^{\beta_k}} \\ n^{1-\alpha_k} &= s^{1-\alpha_{k-2}} \\ n^{1-\alpha_k} &= t^{\gamma_k}, \end{aligned}$$

wobei $\beta_k, \gamma_k > 0$ und $\alpha_{k-2} < 1$ sind. Dies ergibt folgende Parameter:

$$\begin{aligned} \bar{d} &= n^{\frac{\alpha_k}{\beta_k}} \\ s &= n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}} \\ t &= n^{\frac{1-\alpha_k}{\gamma_k}}. \end{aligned}$$

Des Weiteren gilt aber

$$t = \frac{d_{min}^2}{s} = \frac{d_{min}^2}{n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}}$$

und damit

$$\frac{d_{min}^2}{n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}} = n^{\frac{1-\alpha_k}{\gamma_k}}.$$

Mit $d_{min} = n^{\frac{\alpha_k}{\beta_k}}$ ergibt dies folgende Beziehung:

$$\frac{n^{\frac{2\alpha_k}{\beta_k}}}{n^{\frac{1-\alpha_k}{1-\alpha_{k-2}}}} = n^{\frac{1-\alpha_k}{\gamma_k}}. \tag{4.22}$$

Aus Gleichung (4.22) kann nun eine Rekursionsformel für α_k aufgestellt werden:

$$\begin{aligned} \frac{2\alpha_k}{\beta_k} - \frac{1-\alpha_k}{1-\alpha_{k-2}} &= \frac{1-\alpha_k}{\gamma_k} \\ \alpha_k &= \frac{\frac{1}{\gamma_k} + \frac{1}{1-\alpha_{k-2}}}{\frac{2}{\beta_k} + \frac{1}{1-\alpha_{k-2}} + \frac{1}{\gamma_k}} \\ &= 1 - \frac{2}{2 + \beta_k \cdot (\frac{1}{1-\alpha_{k-2}} + \frac{1}{\gamma_k})}. \end{aligned} \tag{4.23}$$

Mit $\beta_k = 1 - \frac{2}{k}$ und $\gamma_k = \frac{3}{k}$ (wie in Algorithmus 4.7) entsteht gerade die Rekursionsformel von Halperin, Nathaniel und Zwick [20].

In Tabelle 4.5 ist eine Übersicht über kleine Werte k bei bestimmten Parametern β_k und γ_k aufgeführt. Ausgangswerte sind immer $\alpha_2 = 0$ und $\alpha_3 = \frac{3}{14}$. Die erste Zeile zeigt den bekannten Algorithmus 4.7.

β_k	γ_k	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
$1-2/k$	$3/k$	7/19	97/207	43/79	1391/2315	175/271
		0.368	0.469	0.544	0.601	0.646
$1-3/k$	$3/k$	7/31	97/262	79/175	1294/2449	2155/3691
		0.226	0.370	0.451	0.528	0.584
$1-2/k$	$4/k$	1/3	333/773	1/2	1543/2775	3/5
		0.333	0.431	0.5	0.556	0.6
$1-2/k$	$3/\log k$					
		0.268	0.352	0.396	0.439	0.468

Tabelle 4.5: Vergleich der Exponenten α_k von Algorithmmus 4.8 bei verschiedenen Parameterfunktionen β_k und γ_k.

Offensichtlich kommen für eine Verbesserung des Algorithmus 4.8 nur kleinere Werte von β_k und größere Werte von γ_k in Frage. Hierbei besitzt anscheinend der Wert β_k einen größeren

Einfluss als γ_k. Allerdings kann man für β_k eine untere Schranke angeben: Ein Algorithmus, der jeden k-färbbaren Graphen mit durchschnittlichem Grad $\bar{d}$ mit $O(\bar{d}^{\beta_k})$ Farben färbt, impliziert bei $\beta_k = 1 - \frac{3}{k}$ einen Algorithmus, der jeden 3-färbbaren Graphen mit einer konstanten Anzahl Farben färbt: Es gilt dann nämlich $\alpha_3 = 1 - \frac{2}{2+(1-\frac{3}{3})(1+\frac{1}{\gamma_3})} = 0$.

Ebenso ist das Verfahren von dem Algorithmus *Find-IS* abhängig. Allerdings kann hier genauso wenig auf Verbesserungen gehofft werden, denn ein k-färbbarer Graph kann nur eine unabhängige Menge enthalten, die $\Theta(\frac{n}{k})$ Knoten umfasst. Der Algorithmus von Alon und Kahale [2], der im aktuell besten Algorithmus dieser Art verwendet wird, nutzt auch die semidefinite Programmierung, um eine große unabhängige Menge zu finden. Wenn man stattdessen einen Algorithmus entwickelt, der die Formel $\gamma_k = O(\frac{1}{\log k})$ erfüllt, dann wäre hier eine entscheidende Verbesserung geglückt (siehe Tabelle 4.5). Bisher ist allerdings noch kein Verfahren bekannt, das in einem Graphen $G = (V, E)$, der eine unabhängige Menge der Größe $\frac{1}{k-1}|V|$ enthält, eine unabhängige Menge der Größe $|V|^{O(\frac{1}{\log k})}$ findet.

Auf der anderen Seite kann Algorithmus 4.8 evtl. noch verbessert werden: Wenn man $\alpha_1 = 0$ setzt und mit der Formel (4.22) versucht, α_3 zu bestimmen, dann erhält man $\alpha_3 = 1 - \frac{2}{2+\beta_3+\frac{\beta_3}{\gamma_3}}$. Mit den bekannten Werten $\beta_k = 1 - \frac{2}{k}$ und $\gamma_k = \frac{3}{k}$ ergibt dies $\alpha(3) = \frac{1}{4}$. Der beste bekannte Algorithmus zur Färbung 3-färbbarer Graphen bietet aber $\alpha_3 = \frac{3}{14}$. Die Überlegungen müssen also auf den Algorithmus von Blum und Karger [7] (Algorithmus 3.9) verlagert werden. Allerdings gibt es keine Möglichkeit, den Algorithmus *Dense-Region-Progress* (Algorithmus 3.6) vom Fall $k = 3$ auf allgemeines k auszuweiten, da bei $k = 3$ explizit mit der Eigenschaft gearbeitet wird, dass die Nachbarschaft eines Knotens bipartit ist. Im allgemeinen Fall ist die Nachbarschaft eines Knotens v in einem k-färbbaren Graphen aber $(k-1)$-partit, woraus sich bis jetzt kein Vorteil schlagen lässt.

Demnach bildet Algorithmus 4.8 mit den Parametern $\beta_k = 1 - \frac{2}{k}$ (Algorithmus 4.5), $\gamma_k = \frac{3}{k}$ (Satz 4.11), sowie den Algorithmen 1.1 und 3.9 als Rekursionsanfänge ($\alpha_2 = 0$ und $\alpha_3 = \frac{3}{14}$) den derzeit besten Färbungsalgorithmus für k-färbbare Graphen bezüglich der verwendeten Anzahl Farben.

Kapitel 5

Ist *k*-Coloring im wahren Leben einfach?

In diesem Kapitel werden Fälle betrachtet, in denen es einfach erscheint, einen Graphen optimal zu färben. Zunächst wird ein Modell präsentiert, in dem ein zufälliger k-färbbarer Graph mit hoher Wahrscheinlichkeit mit einem einfachen Algorithmus mit k Farben gefärbt werden kann. Danach werden Färbungsprobleme betrachtet, die aus anwendungsbezogenen Graphen stammen. Man kann dort feststellen, dass ein exakter Algorithmus nur unwesentlich mehr Zeit benötigt als ein aufwändiger Approximationsalgorithmus.

5.1 Fast alle *k*-färbbaren Graphen sind leicht zu färben

Mit Algorithmus 1.3 wird ein Wahrscheinlichkeitsraum für k-färbbare Graphen $G = (V, E)$ konstruiert.

Definition 5.1. Sei $k \leq n$ und $p \in [0, 1]$. Der Wahrscheinlichkeitsraum, der durch den Algorithmus *makeGraph2*(n,k,p) (Algorithmus 1.3) definiert wird, sei $X_n(k, p)$.

Das bedeutet, jedem k-färbbaren Graphen mit n Knoten wird mit Algorithmus 1.3 eine positive Wahrscheinlichkeit zugeordnet. Bei genauerer Betrachtung entpuppt sich $X_n(k, p)$ als Erweiterung des bekannten Zufallsmodells: Ein k-färbbarer Graph ist auch $k + 1$-färbbar, $k + 2$-färbbar,... und n-färbbar. Das bedeutet, dass $X_n(n, p)$ identisch mit der Verteilung $G_n(p)$ ist. Hierbei ist $G_n(p)$ die Verteilung, die entsteht, wenn man jede Kante zufällig mit der Wahrscheinlichkeit p hinzufügt.
Turner [34] vetritt nun die Behauptung, dass man aus dem Wahrscheinlichkeitsraum $X_n(k, p)$ gewählte zufällige k-färbbare Graphen mit einem No-Choice-Algorithmus exakt färben kann.

Algorithmus 5.1. ***NoChoice*(*G*,*k*)**

Eingabe: ein Graph G, k als Farbanzahl.

Ausgabe: eine k-Färbung von G oder `no success`.

$C :=$ eine maximale Clique von G.

`if` $|C| < k \Rightarrow$ `return no success`.

Färbe die Knoten von C eindeutig mit den ersten $|C|$ Farben.

`repeat`

• `for each` $v \in V$: Setze avail$[v] := |\{i : 1 \leq i \leq k \wedge (\{v, w\} \in E \Rightarrow color(w) \neq i)\}|$

• Sei v ein Knoten mit avail$[v] = 1$. Färbe v mit der einzigen möglichen Farbe.

`until` (alle Knoten gefärbt `or` avail$[v] \neq 1 \; \forall v \in V$)

Algorithmus 5.1 scheitert, wenn er entweder keine k-Clique findet oder sich im Verlaufe des Algorithmus herausstellt, dass für jeden ungefärbten Knoten mehr als eine Farbe zur Verfügung steht. An dieser Stele müsste der Algorithmus einen Knoten wählen, der noch minimal viele Farben zur Verfügung hat (mindestens 2 Farben) und dann eine Farbe auswählen. Diese Entscheidung kann Algorithmus 5.1 allerdings nicht treffen. Das bedeutet, dass sämtliche k-Färbungen des Graphen G jeweils dieselbe Knotenpartition des Graphen implizieren müssen, wenn Algorithmus 5.1 erfolgreich sein soll.

Definition 5.2. Sei G ein k-färbbarer Graph. Wenn sämtliche zulässigen k-Färbungen von G dieselbe Knotenpartition implizieren, dann heißt G *eindeutig k-färbbar*.

Das bedeutet, dass jeder von Algorithmus 5.1 erfogreich gefärbte k-färbbare Graph G auch eindeutig k-färbbar sein muss.
Es sei $\lambda(c) = \lambda_n(c) = -\frac{\ln n}{\ln c} = -\log_c n$: Mit $c \in (0,1)$ und $n > 1$ ist $\lambda_n(c) > 0$ und $c^{\lambda_n(c)} = 1/n$, sowie $\lim_{n\to\infty} \lambda_n(c) = \infty$ für ein festes $c \in (0,1)$.

Definition 5.3. Sei $G = (V, E)$ ein k-färbbarer Graph. Der Graph G besitzt die *Cliquen-Eigenschaft*, wenn jede r-Clique ($r < k$) auf eine $r+1$-Clique erweiterbar ist.

Sei nun G ein k-färbbarer Graph, der mindestens eine k-Clique enthält, und sei $\{x_1, ..., k_k\}$ eine k-Clique von G. Für $i = 1, ..., k$ seien folgende Mengen definiert:

$$\begin{aligned} A_i^{(0)}(x_1, ..., x_k) &= \{x_i\} \\ A_i^{(t)}(x_1, ..., x_k) &= \{y \in V : \forall j \neq i \exists z_j \in A_j^{(t-1)}(x_1, ..., x_k) : \{y, z_j\} \in E\}. \end{aligned}$$

Die Menge $A_i^{(t)}(x_1, ..., x_k)$ besteht also aus den Knoten y_i, die in jeder Menge $A_j^{(t-1)}(x_1, ..., x_k)$ mit $j = 1, ..., k; j \neq i$ einen Nachbarn haben. Alle Knoten der Mengen $A_i^{(1)}(x_1, ..., x_k)$ haben demnach in einer k-Färbung der Knoten $x_1, ..., x_k$ genau eine Farbe zur Auswahl, nämlich die Farbe i.

Es gilt $A_i^{(t)}(x_1, ..., x_k) \subseteq A_i^{(t+1)}(x_1, ..., x_k)$ für alle i. Weiter sei $A_i^{(\infty)}(x_1, ..., x_k)$ definiert als die kleinste Menge $A_i^{(t)}(x_1, ..., x_k)$, für die gilt $A_i^{(t)}(x_1, ..., x_k) = A_i^{(t+1)}(x_1, ..., x_t)$, was den Fixpunkt bei der Iteration der Mengen $A_i^{(t)}(x_1, ..., x_k)$ bestimmt.

Ein k-färbbarer Graph $G = (V, E)$ ist nun nach [34] *einfach färbbar*, wenn G die *Cliquen-Eigenschaft* erfüllt und für alle Cliquen $\{x_1, ..., x_k\}$ gilt:

$$\bigcup_{i=1}^{k} A_i^{(3)}(x_1, ..., x_k) = V.$$

Diese Eigenschaft trifft natürlich nicht auf sämtliche Graphen zu, die von Algorithmus 5.1 korrekt gefärbt werden; aber bei jedem nach Definition 5.2 eindeutig färbbaren Graphen G wird Algorithmus 5.1 erfolgreich sein. Im Allgemeinen ist der No-Choice-Algorithmus bei allen k-färbbaren Graphen G erfolgreich, für die gilt $\bigcup_{i=1}^{k} A_i^{(\infty)}(x_1, ..., x_k) = V$, d.h. bei fortgesetzter Iteration der Mengen $A_i^{(t)}$ wird irgendwann die gesamte Knotenmenge abgedeckt. Turner zeigt mit Wahrscheinlichkeitsabschätzungen, dass fast alle Graphen $G \in X_n(k, p)$ einfach färbbar sind, also dass $A_j^{(3)}(x_1, ..., x_t) = I_j$ für alle j gilt, wenn $x_1, ..., x_k$ eine k-Clique in G ist [34]. Hierbei sind $I_1, ..., I_k$ die jeweiligen Farbklassen (Definition 1.6).

Korollar 5.4. *Seien $0 < \varepsilon < 1$, $0 < p < 1$ fest, $n \to \infty$ und $2 \leq k \leq (1 - \epsilon)\lambda(p)$. Fast alle Graphen $G \in X_n(k, p)$ sind eindeutig k-färbbar.*

Beweis. Nach [34] sind fast alle Graphen $G \in X_n(k, p)$ einfach färbbar. Für diese Graphen findet Algorithmus 5.1 eine zulässige k-Färbung. Da Algorithmus 5.1 nur eindeutig färbbare Graphen erfolgreich färben kann, gilt die Behauptung. □

Algorithmus 5.1 färbt fast alle k-färbbaren Graphen nach einer bestimmten Wahrscheinlichkeitsverteilung exakt und arbeitet in Linearzeit. Anders sieht das Ganze aus, wenn die Verteilung so gewählt wird, dass jedem k-färbbaren Graphen die gleiche Wahrscheinlichkeit zugeordnet wird, d.h. ohne Berücksichtigung von Knoten- und Farbpermutationen. Bisher ist aber kein Verfahren bekannt, das eine solche Wahrscheinlichkeitsverteilung produzieren kann, da das Problem Graph-Isomorphie ebenfalls NP-vollständig ist.

Algorithmus 5.1 ist eine verschärfte Variante von Algorithmus 2.2, welcher seinerseits nicht garantieren kann, dass seine produzierte Färbung optimal ist. Wenn beide Algorithmen miteinander verglichen werden, dann stellt man fest, dass sie sich bei den Graphen, die Algorithmus 5.1 erfolgreich behandelt, gleich verhalten: Bei beiden wird zunächst eine Clique der Größe $k' \leq k$ konstruiert, indem man jeweils den Knoten färbt, der bereits mit allen anderen gefärbten Knoten verbunden ist und im ungefärbten Restgraphen den maximalen Grad hat. Danach wird immer der Knoten genommen, der die meisten verschiedenfarbigen Nachbarn besitzt. Solange für jeden Knoten genau eine Farbe zur Auswahl bleibt, kommt Algorithmus 5.1 noch mit; sobald allerdings eine Auswahl zu treffen ist, scheitert die No-Choice-Variante.
Mit dieser Prämisse kann man feststellen, dass auch der *DSATUR*-Algorithmus 2.2 fast alle Graphen $G \in X_n(k,p)$ exakt färbt. Es gibt in dieser Betrachtung nur einen Schönheitsfehler: Die Aussagen gelten nur für kleine Werte von k, in der Arbeit von Turner [34] ist $k = O(|\log_p n|)$, und die gewählten Wahrscheinlichkeiten p sind konstant. Es bleibt eine offene Frage, ob zufällige k-färbbare Graphen mit $k = \Omega(n^c)$ mit $c \in [0,1]$, $c = konst.$ und $p = O(1/n)$ (also viel dünner besetzte Graphen, die viel mehr Farben benötigen) in Polynomialzeit exakt gefärbt werden können. Die Algorithmen 5.1 und 2.2 sind bei solchen Werten zum Scheitern verurteilt, da es sehr unwahrscheinlich ist, eine k-Clique zu erhalten, und damit vergrößert sich auch der Spielraum bei der Entscheidung für die geeignete Farbe.

5.2 1-perfekte Graphen und max-Clique

Zur Färbung eines Graphen kann eine gute untere Schranke für die Anzahl zu verwendender Farben hilfreich sein. Genauer: Wenn man weiß, wieviele Farben mindestens benötigt werden, dann kann man die Suche nach einer besseren Färbung einstellen, sobald man diese untere Schranke erreicht hat. Eine untere Schranke beim Färbungsproblem ist die Größe $\omega(G)$ einer größten Clique im Graphen $G = (V, E)$. Bei einer Clique der Größe k muss jeder der beteiligten Knoten eine andere Farbe bekommen; dies führt zu mindestens k Farben im Graphen.
Allerdings muss die Hoffnung sogleich durch drei Aspekte gedämpft werden:

1. Das Problem Maximum-Clique ist ebenfalls NP-vollständig.

2. Wenn $\omega(G) < \chi(G)$ ist, dann hilft die maximale Clique nicht, da trotzdem fast alle (potenziellen) $\omega(G)$-Färbungen durchprobiert werden müssen, was wieder nur in Expo-

nentialzeit möglich ist.

3. Fast alle Graphen G mit n Knoten erfüllen nach [29] die Eigenschaft

$$\omega(G) < 4 \log n < \frac{n}{3 \log n} < \chi(G).$$

Es gibt aber einen kleinen Anteil von Graphen, bei denen die letzten beiden Eigenschaften nicht zutreffen. Diese Graphen nennt man *1-perfekt*.

Definition 5.5. Ein Graph $G = (V, E)$ heißt *1-perfekt*, wenn $\omega(G) = \chi(G)$ gilt, d.h. die größte Clique der chromatischen Zahl entspricht. Ein Graph $G = (V, E)$ heißt *perfekt*, wenn alle Teilgraphen von G 1-perfekt sind.

Nach [18] sind perfekte Graphen mit der Ellipsoid-Methode in Polynomialzeit färbbar, allerdings ist das Verfahren für praktische Anwendungen viel zu langsam.
Hier wird das Verhalten von den einfachen exakten Färbungsalgorithmen 2.3 und 2.4 auf sogenannten *real-life*-Graphen betrachtet. Das sind Graphen, deren Färbung eine praktische Bedeutung hat. In diese Kategorie fallen z.B. Graphen, die bei den Problemen *Register Allocation*, *Scheduling*, *Frequency Assignment* und *Planar Routing* zur Anwendung kommen. Solche Graphen haben meist eine Art Gitterstruktur, d.h. die Knoten erhalten (ganzzahlige) Koordinaten eines mehrdimensionalen Raums, und die Kanten verlaufen nur zwischen Knoten mit benachbarten Koordinaten. Dies gibt ihnen eine grundlegende und für die exakte Färbung wichtige Eigenschaft. Viele dieser Graphen bilden eine große Clique aus, und danach richtet sich ihre chromatische Zahl. Man stellt also fest, dass ein großer Anteil der in praktischen Anwendungen auftretenden Graphen 1-perfekt ist.
Von der praktischen Seite her kann es also von Vorteil sein, zuerst eine größte Clique in G zu finden, vor allem, wenn man einen Algorithmus hat, der diese Clique schnell findet.
Coudert [10] zeigt nun, dass man mit einer maximalen Clique einen 1-perfekten Graphen schnell färben kann. Das Ziel ist ein Branch-and-Bound-Algorithmus, der viele Verzweigungsmöglichkeiten abblockt, durch die die Lösung verschlechtert wird.
Coudert verwendet zur Bestimmung der exakten Färbung Algorithmus 2.3, nachdem er eine maximale Clique im gegebenen Graphen G gefunden hat. Es kann aber auch Algorithmus 2.4 verwendet werden, der dieselbe Struktur aufweist wie Algorithmus 2.3, aber durch eine verbesserte Auswahl der zu färbenden Knoten die Anzahl der Backtracks stark verringert. Beide

Algorithmen verwenden eine Clique als Startwert. Wie oben beschrieben, kann eine größte Clique hier die Anzahl der Suchschritte drastisch vermindern.

Algorithmus 5.2. ***MaxClique*(*G*,*C*,*best*,*ub*)**

Eingabe: Ein Graph $G = (V, E)$, die in Konstruktion befindliche Clique C mit $C \cap V = \emptyset$, die bisher größte Clique *best* und eine obere Schranke *ub*.

Ausgabe: Eine größte Clique, entweder C oder *best*.

1. `if` G ist leer $\Rightarrow$ `return` C;
2. $t :=$ Anzahl der bei einer Färbung von G verwendeten Farben (z.B. mit Algorithmus 2.2)
   $ub := \min\{ub, |C| + t\}$
   `if` $ub \leq |best| \Rightarrow$ `return` *best*
3. $v :=$ Knoten mit maximalem Grad in G;
   $G' := G[N(v)]$
   best :=*MaxClique*(G',$C \cup v$,*best*,*ub*)
   `if` $ub = |best| \Rightarrow$ `return` *best*
4. $G'' := G[V \setminus \{v\}]$
   `return` *MaxClique*(G'',C,*best*,*ub*)

Der Algorithmus wird zuerst mit den Parametern G, $C = \emptyset$, $best = \emptyset$ und $ub = \infty$ aufgerufen. Algorithmus 5.2 basiert auf den folgenden grundlegenden Prinzipien:

(a) Ein k-färbbarer Graph G kann keine $k + 1$-Clique enthalten, folglich ist k eine obere Grenze für $\gamma(G)$ (*ub*).

(b) Wenn ein Knoten $v \in V$ in einer Clique $C \subseteq V$ enthalten ist, dann können außer v nur Knoten $w \in N(v)$ zu C gehören ($G' := G[N(v)]$).

Weiter kann Folgendes hinzugefügt werden, um weitere Verzweigungen abzuschneiden:

(c) Wenn $|C| + |V| \leq |best|$ ist, dann kann keine größere Clique als *best* mehr gefunden werden.

(d) Jeder Knoten v mit $d(v) < |best| - |C|$ kann kein Mitglied einer größeren Clique als *best* sein und kann demnach entfernt werden.

(e) Jeder Knoten v mit $d(v) \geq |V| - 2$ muss in die Clique aufgenommen werden, da es keine größere Clique ohne v geben kann.

(f) Jeder Knoten v, für den $V \setminus N(v)$ eine unabhängige Menge ist, muss in die Clique aufgenommen werden, da es keine größere Clique ohne v geben kann.

(g) Statt v müssen mindestens 2 Nicht-Nachbarn v_1, v_2 von v in einer Clique enthalten sein, die größer sein soll als $C \cup \{v\}$.

Beweis. Zu (c): Im Rekursionsaufruf sind die Knoten der bisherigen Clique C nicht in der Menge V enthalten. Folglich bietet der Wert $|C| + |V|$ eine obere Schranke für die mit den aktuellen Knoten noch mögliche Clique. Ist dieser Wert nicht größer als die Größe der bisher größten Clique, dann kann die Rekursion abgebrochen werden.
Bei (d) fehlen zur bisher größten Clique *best* noch genau $|best| - |C|$ Knoten. Wenn v in die Clique aufgenommen wird, dann entsteht höchstens eine Clique mit $|C| + 1 + d(v) \leq |C| + |best| - |C| = |best|$, also eine Clique, die höchstens genauso groß ist wie *best*. An dieser Stelle kann die Rekursion ebenfalls abgebrochen werden, da eine Clique gesucht wird, die mehr Knoten enthält.
Für Punkt (e) sei $v \in V$ ein Knoten mit $d(v) = |V| - 2$. Dann gibt es genau einen Knoten $v' \in V$, mit dem v nicht durch eine Kante verbunden ist. Eine maximale Clique, die v nicht enthält, muss den Knoten v' enthalten, da sonst die Clique nicht maximal wäre. Nun hat v' aber ebenso einen maximalen Grad $|V| - 2$. Damit ist eine maximale Clique mit dem Knoten v' höchstens genauso groß wie die größte Clique mit v.
Der Punkt (f) ist eine Verallgemeinerung von (e). Allerdings benötigt das Testen auf eine unabhängige Menge zu viel Zeit und wird daher nicht mit aufgenommen.
Schließlich gibt es noch Punkt (g). Wenn der Knoten v nicht in einer Clique verwendet wird, dann muss in einer größten Clique mindestens ein Nicht-Nachbar von v enthalten sein. Weiter kann man die Anzahl der Nicht-Nachbarn von v in einer größten Clique mit mindestens zwei angeben: Angenommen, es gäbe nur einen Nicht-Nachbarn v' von v in der größten Clique. Dann wird v' durch v ersetzt und es entsteht eine Clique, die genauso groß ist. Demnach werden mindestens zwei Knoten benötigt, die nicht mit v benachbart sind, um eine Clique zu erhalten, die größer ist als eine größte Clique mit v. □

Der Hauptsatz von Coudert zielt nun darauf ab, den Suchraum noch weiter zu verringern:

Satz 5.6. [10] *Sei G der aktuelle Graph an irgendeinem Punkt der Rekursion von Algorithmus 5.2, C die zu konstruierende Clique und best die bisher größte gefundene Clique. Seien weiter $I_1, ..., I_t$ die in Schritt 2 von Algorithmus 5.2 gefundenen Farbklassen von G. Dann kann jeder Knoten v, der mit $q > |C| - |best| + t$ Farben gefärbt werden könnte, aus G entfernt werden.*

Aus diesem Satz lässt sich ein Schneeballeffekt herleiten: Wenn ein Knoten v entfernt wurde, dann verringert sich der Grad seiner Nachbarn; vielleicht wird dann auch bei einigen Knoten wieder eine Farbe frei, womit wieder ein Knoten die Grenze q erreicht. Im Laufe dieses Prozesses kann es sein, dass eine komplette Farbklasse geleert wird, so dass q selbst verkleinert wird usw. Es kann sogar der Fall eintreten, dass k zu klein wird, um überhaupt eine weitere Rekursion von diesem Punkt aus fortzusetzen.

Beweis. Die t-Färbung von G teilt die Knotenmenge V in t disjunkte Mengen $I_1, ..., I_t$. Sei v ein Knoten, der in der t-Färbung mit q Farben gefärbt werden kann. O.B.d.A. heißt das, dass $I_j \cup \{v\}$ für $I_1, ..., I_q$ jeweils unabhängige Mengen bilden (dem Knoten v kann jede der Farben 1,...,q zugewiesen werden). Sei nun C' die größte Clique, die entstehen kann, wenn v zu C hinzugefügt wird. Es gilt

$$\begin{aligned} |C'| &= |C \cup \{v\} \cup \mathbf{MaxClique}(N(v))| \\ &= |C| + 1 + \gamma(N(v)) \\ &\leq |C| + 1 + \chi(N(v)) \\ &\leq |C| + 1 + t - q \qquad (5.1) \\ &\leq |best| \end{aligned}$$

Die Ungleichung (5.1) gilt, weil $N(v) \subseteq \bigcup_{j=q+1}^{t} I_j$ ist, und damit $\{I_{q+1}, ..., I_t\}$ eine zulässige $(t-q)$-Färbung von $N(v)$ ist.
Da also keine größere Clique als *best* entstehen kann, kann der Knoten v aus G entfernt werden. □

Eine maximale Clique zu finden, ist bei 1-perfekten Graphen unerlässlich, wenn man ein gutes Abbruchkriterium für einen exakten Algorithmus nutzen möchte. Rechtzeitiges Abbrechen der Suche führt bei geringem (polynomiellen) Aufwand zu einem exponentiellen Vorteil. Wenn die Clique nicht maximal ist, dann entsteht aus der Bestimmung der Clique kein Vorteil.

Coudert belegt den Erfolg seiner Überlegungen mit einigen experimentellen Resultaten [10]. Es zeigt sich, dass für 1-perfekte Graphen der Aufwand zwar noch exponentiell ist, aber deutlich geringer als bei vergleichbaren Algorithmen, die keine maximale Clique bestimmen. Dagegen ist bei nicht-1-perfekten Graphen das Suchen der Clique quasi nutzlos, und damit auch der Aufwand für eine exakte Färbung ähnlich groß.

Einen anderen Weg zur exakten Färbung kann man mit schrittweiser Vergrößerung der unteren Schranke erzielen: Zuerst wird eine maximale (nicht notwendigerweise eine größte) Clique C im Graphen $G = (V, E)$ gesucht. Dann versucht man, eine $|C|$-Färbung von G zu finden. Gelingt dies nicht, dann sucht man eine $2|C|$-Färbung usw. Das heißt, man verdoppelt immer die untere Schranke, bis eine entsprechende k'-Färbung gefunden wurde. Dann ist bekannt, dass $k'/2 < \chi(G) \leq k'$ gilt. Nun wird Algorithmus 2.4 verwendet, um eine exakte Färbung zu finden. Als untere Schranke gilt dann nicht mehr $|C|$, sondern $k'/2$, und die obere Schranke ist durch k' bestimmt.

Vorteile kann dieses Vorgehen allerdings nur erzielen, wenn man schnell feststellen kann, dass ein Graph *nicht* k-färbbar ist. Die Frage der Nicht-k-Färbbarkeit ist allerdings co-NP-vollständig.

Kapitel 6

Andere Färbungsprobleme

In diesem Kapitel werden Färbungsprobleme behandelt, auf die die bisher vorgestellten Algorithmen nicht mehr angewendet werden können. Den ersten Abschnitt bilden Färbungsalgorithmen für Online-Situationen. Dabei sollen die Knoten eines Graphen nacheinander gefärbt werden, unter der Bedingung, dass nur ein Teil des Graphen (die bisher gefärbten Knoten) bekannt ist. Abschnitt 6.2 beschäftigt sich mit der Schwierigkeit, einen Graphen exakt zu färben. Im dritten Abschnitt werden Färbungen von Hypergraphen betrachtet und einige Schwierigkeitsresultate vorgestellt. Der letzte Abschnitt stellt noch einige Färbungsprobleme vor, die dem Ausgangsproblem COLORING noch verschiedene Bedingungen auferlegen.

6.1 Online-Algorithmen zur Graphenfärbung

Ein Online-Graph $G^{\prec} = (V, E, \prec)$ besteht aus einem Graphen $G = (V, E)$ und einer Ordnung $\prec$ über der Knotenmenge $V = \{v_1 \prec v_2 \prec ... \prec v_n\}$. Die Knoten werden einem Online-Färbungsalgorithmus entsprechend der Ordnung $\prec$ präsentiert: Gemeinsam mit dem Knoten $v \in V$ werden alle Kanten $\{u, v\}$ mit $u \prec v$ angegeben, also die Kanten zu bereits bekannten Knoten.

Eine Online-Färbung läuft so ab, dass einem Knoten zum Zeitpunkt seiner Einführung unwiderruflich eine Farbe zugewiesen wird.

In gewisser Weise ist damit der Greedy-Algorithmus 2.1 der erste und einfachste Online-Algorithmus zur Graph-Färbung: Es werden alle Knoten nacheinander betrachtet, und jedem Knoten wird sofort eine Farbe zugewiesen. Die maximal mögliche Farbe ist dann $\Delta(G) + 1$, wobei $\Delta(G)$ dem maximalen Grad des Graphen G entspricht. Die Ergebnisse aus [26] implizieren, dass für allgemeine Graphen der Wert $\Delta(G) + 1$ nicht verbessert werden kann.

Gleichzeitig kann Algorithmus 2.1 als First-Fit-Strategie gesehen und implementiert werden: Es wird immer die erstbeste Farbe genommen, die keine Konflikte auslöst.
Allerdings gibt es vielversprechende Resultate, die für spezielle Graphen gelten. Hierbei sei C_k ein Kreis mit k Knoten.

Satz 6.1. **[26]**. *Für jede Zahl n gibt es einen Online-Algorithmus, der jeden Graphen G mit n Knoten, der weder C_3 noch C_5 enthält, mit höchstens $2\sqrt{n}$ Farben färbt.*

Beweis. Es sei ein Graph G mit n Knoten gegeben, der weder einen Kreis der Länge 3 noch einen Kreis der Länge 5 enthält. Die Eingabefolge sei $v_1 \prec v_2 \prec ... \prec v_n$. Zum Färben wird folgender Algorithmus verwendet:

Algorithmus 6.1. ***Online-C3-C5-free*$(G^{\prec},n)$**
Eingabe: Ein Online-Graph $G^{\prec}$ mit n Knoten, der weder C_3 noch C_5 enthält.
Ausgabe: Eine $O(\sqrt{n})$-Färbung von $G^{\prec}$.

`for` $i := \lceil\sqrt{n}\rceil..\lceil 2\sqrt{n}\rceil : W_i := \emptyset$
`for` $i := 1..n$:

1. `if` für v_i die kleinste freie Farbe $t \leq \lceil\sqrt{n}\rceil \Rightarrow$ Färbe v_i mit t.
2. `else if` $\exists t > \sqrt{n}$ mit $v_i \in N(W_t) \Rightarrow$ Färbe v_i mit $\min t$.
3. `else` $t := min_l\{W_l = \emptyset\}$
   $W_t := \{v \in N^{\prec}(v_i) : Color(v) \leq \sqrt{n}\}$
   Färbe v_i mit t.

Algorithmus 6.1 arbeitet korrekt: Angenommen, zwei adjazente Knoten $u \prec v$ erhalten dieselbe Farbe j. Es ist offensichtlich, dass, da v als zweiter Knoten seine Farbe erhalten hat, v nicht im Schritt 1 gefärbt werden konnte. Folglich ist $j > \sqrt{n}$, und damit kann auch u nicht im Schritt 1 gefärbt worden sein. Nur der erste Knoten, der die Farbe j erhält, kann im Schritt 3 gefärbt worden sein; dieser Fall kann also höchstens auf u zutreffen – v wurde im Schritt 2 gefärbt.
Wenn nun u im Schritt 3 gefärbt wurde, dann bedeutet das, dass $W_j \subset N(u)$ und $v \in N(W_j)$ gelten. Das heißt, dass u und v einen gemeinsamen Nachbarn in W_j haben. Dies widerspricht der Annahme, dass $G^{\prec}$ keine Kreise der Länge 3 enthält.

Andererseits, wenn u in Schritt 2 gefärbt wurde, dann heißt das, dass $u \in N(W_j)$ und $v \in N(W_j)$ gelten. Das bedeutet, dass entweder u und v einen gemeinsamen Nachbarn in W_j haben, oder sie haben verschiedene Nachbarn, die jeweils zu dem Knoten adjazent sind, der zuerst mit Farbe j gefärbt wurde. Im ersten Fall ergibt dies wieder einen Kreis der Länge 3, im zweiten Fall einen Kreis der Länge 5, was wieder beides der Annahme zu $G^{\prec}$ widerspricht. Damit wird Algorithmus 6.1 eine zulässige Färbung produzieren. □

Eine Bemerkung sollte noch gemacht werden: Algorithmus 6.1 setzt voraus, dass die Anzahl der Knoten von vorneherein bekannt ist. In einer Online-Situation kann man sich aber nicht sicher sein, dass die Anzahl der Knoten dem Algorithmus übergeben wird.
Die First-Fit-Strategie (Algorithmus 2.1) ist, wie bereits oben erwähnt, die einfachste Variante, einen Online-Graphen zu färben. Man kann nach [26] sagen, dass die First-Fit-Strategie für Bäume sogar der optimale Online-Färbungsalgorithmus ist. Für allgemeine 2-färbbare Graphen ist dagegen First-Fit alles andere als optimal.

Satz 6.2. *Für jede positive Zahl n existiert ein 2-färbbarer Graph $G = (V, E)$ mit $2n$ Knoten, so dass First-Fit auf einer Eingabesequenz $\prec$ für G genau n Farben verwendet.*

Beweis. Der gewählte Graph sei $G_n = K_{n,n} \setminus M$, wobei $K_{n,n}$ der vollständige bipartite Graph mit $n+n$ Knoten und $M = (V, \{\{a_i, b_i\} : i \in \{1, ..., n\}\})$ ein perfektes Matching in $K_{n,n}$ ist. Als Eingabefolge $\prec$ wird $(a_1, b_1, a_2, b_2, ..., a_n, b_n)$ gewählt. Damit wird First-Fit jedes Knotenpaar a_i, b_i mit der Farbe i färben: Zuerst erhält a_1 die Farbe 1. Der Knoten b_1 ist nicht mit a_1 verbunden, also bekommt er ebenfalls die Farbe 1. Seien nun die Knoten $a_1, ..., a_i, b_1, ..., b_i$ mit den Farben $1, ..., i$ gefärbt. Der Knoten a_{i+1} hat die Knoten $b_1, ..., b_i$ als Nachbarn und muss demnach mit Farbe $i + 1$ gefärbt werden. Analog dazu hat b_{i+1} die Knoten $a_1, ..., a_i$ als Nachbarn und muss ebenso die Farbe $i + 1$ erhalten. □

Als Gegensatz dazu kann ein Online-Algorithmus angegeben werden, der jeden 2-färbbaren Graphen mit n Knoten mit höchstens $2 \log n$ Farben online färbt.

Satz 6.3. *Es gibt einen Online-Algorithmus, der jeden 2-färbbaren Graphen G mit n Knoten online mit maximal $2 \log n$ Farben färbt.*

Beweis. Sei $v_1 \prec v_2 \prec ... \prec v_n$ die Eingabesequenz des 2-färbbaren Graphen $G^{\prec}$. Wenn v_i ankommt, dann gibt es eine eindeutige Partition (I_1, I_2) der Zusammenhangskomponente V_i,

zu der v_i gehört, in unabhängige Mengen. Hierbei sei o.B.d.A. $v_i \in I_1$. Der Algorithmus weist v_i die kleinste Farbe zu, die noch kein Knoten in I_2 bekommen hat.
Es genügt zu zeigen, dass, wenn der Algorithmus t Farben für eine Zusammenhangskomponente von $G_i^{\prec}$ benutzt, dann muss diese Komponente mindestens $2^{\lfloor t/2 \rfloor}$ Knoten enthalten. Es wird Induktion über i genutzt:
Der Induktionsanfang ist klar: der Knoten v_1 erhält die Farbe 1. Damit wurde eine Farbe für $1 = 2^{\lfloor 1/2 \rfloor}$ Knoten verwendet. Sei nun $i > 1$. Wenn der Knoten v_i die Farbe $k+2$ erhält, dann muss es in I_2 einen Knoten v_p geben, der bereits die Farbe k bekommen hat, und ebenso einen weiteren Knoten, der die Farbe $k+1$ erhielt. Ebenso hat der Algorithmus einem Knoten $v_q \in I_1$ die Farbe k zugeordnet. Da beide Knoten v_p, v_q dieselbe Farbe haben und nun in zwei verschiedenen Teilen von V_i liegen, müssen sie im Graphen $G_r^{\prec}$, mit $r = \max\{p, q\}$, in verschiedenen Zusammenhangskomponenten gewesen sein. Nach Induktionsvoraussetzung muss jede dieser Komponenten mindestens $2^{\lfloor k/2 \rfloor}$ Knoten besessen haben, und damit hat die Komponente von $G_i^{\prec}$, in der v_i enthalten ist, mindestens $2 \cdot 2^{\lfloor k/2 \rfloor} = 2^{\lfloor (k+2)/2 \rfloor}$ Knoten. □

Damit ergibt sich folgender Algorithmus:

Algorithmus 6.2. ***Online-2-color*(G)**

Eingabe: Ein 2-färbbarer Online-Graph $G^{\prec}$ mit der Eingabefolge $v_1, v_2, ..., v_n$.

Ausgabe: Eine $O(\log n)$-Färbung von G.

`for` $i = 1, 2, ...$

· Teile die Zusammenhangskomponente, zu der v_i gehört, in zwei Teile (Algorithmus 1.1), so dass $v_i \in I_1$ ist.

· Setze $color(v_i) := \min\{\text{nicht zugewiesene Farbe in } I_2\}$.

Als Beispielgraph soll wie oben der Graph $G_n = K_{n,n} \setminus M$ dienen. Der Programmverlauf bei den ersten acht Knoten ist in Tabelle 6.1 aufgeführt. Danach wiederholen sich die letzten beiden Schritte.
Für Graphen, deren chromatische Zahl größer als 2 ist, kann eine solche Schranke wie bei Algorithmus 6.2 nicht angegeben werden, da nicht bekannt ist, wie eine k-Partition eines k-färbbaren Graphen schnell gefunden werden soll. Es wurde schließlich gezeigt [26], dass k-färbbare Graphen $\Omega(\log^{k-1} n)$ Farben mit einem Online-Algorithmus benötigen.

Knoten	I_1	I_2	Farbe
a_1	$\emptyset$	$\emptyset$	1
b_1	$\emptyset$	$\emptyset$	1
a_2	$\emptyset$	$\{b_1\}$	2
b_2	$\emptyset$	$\{a_1\}$	2
a_3	$\{a_1, a_2\}$	$\{b_1, b_2\}$	3
b_3	$\{b_1, b_2\}$	$\{a_1, a_2, a_3\}$	4
a_4	$\{a_1, a_2, a_3\}$	$\{b_1, b_2, b_3\}$	3
b_4	$\{b_1, b_2, b_3\}$	$\{a_1, a_2, a_3, a_4\}$	4

Tabelle 6.1: Durchlauf von Algorithmus 6.2 für den Graphen $K_{n,n} \backslash M$. First-Fit verwendet bei dieser Eingabesequenz n Farben, in diesem speziellen Fall werden unabhängig von n immer 4 Farben verwendet.

Auf der anderen Seite gibt es zwei Ergebnisse, die unabhängig voneinander stehen können (siehe [26]):

Satz 6.4. *Es gibt einen Online-Algorithmus, der jeden k-färbbaren Graphen G mit n Knoten mit $O(n\frac{\log^{(2k-3)} n}{\log^{(2k-4)} n})$ Farben färben kann. Hierbei ist* $\log^{(t)} m = \underbrace{\log \log ... \log}_{t-\text{mal}} m$.

Satz 6.5. *Für jede Zahl k gibt es einen Online-Algorithmus A_k und eine Zahl N, so dass A_k für jeden k-färbbaren Graphen G mit $n \geq N$ Knoten maximal $n^{1-1/k!}$ Farben verwendet.*

Für 3- und 4-färbbare Graphen gibt es noch spezielle Algorithmen, die im folgenden Satz zusammengefasst sind:

Satz 6.6. [26]. *Es gibt einen Online-Algorithmus, der jeden 3-färbbaren Graphen mit n Knoten online mit weniger als $20 \cdot n^{2/3} \log^{1/3} n$ Farben färbt.*
Es gibt einen Online-Algorithmus, der jeden 4-färbbaren Graphen mit n Knoten online mit weniger als $120 \cdot n^{5/6} \log^{1/6} n$ Farben färbt.

Die beiden Algorithmen sind insofern bemerkenswert, da sie auch in Polynomialzeit laufen. Die Güten der besten bekannten Offline-Algorithmen, die in Polynomialzeit laufen, können für 3- und 4-färbbare Graphen ebenfalls nur mit $O(n^c)$ abgeschätzt werden (Kapitel 4).

6.2 Mapmaker's Dilemma - ein Spiel

Eine völlig andere Richtung schlagen Beigel und Gasarch [5] ein. Hier wird ein k-färbbarer Graph online gegeben, und es soll jeweils eine k-Färbung des Graphen gesucht werden.
Die Aufgabenstellung lautet im Einzelnen:

> Wir versetzen uns ins 12. Jahrhundert und schauen einem Kartenzeichner bei der Arbeit zu. Er hat eine Landkarte und soll diese mit vier Farben färben. Dazu hat er potenziell unendlich viel Zeit. Von Zeit zu Zeit erhält der Kartenzeichner Nachrichten von Entdeckern, die ein neues Land entdeckt haben. Dieses neue Land besitzt einige Grenzen mit der bekannten Welt und soll nun mit eingetragen werden. Es kann nun aber sein, dass die aktuelle 4-Färbung nicht mehr gültig ist, wenn das neue Land hinzukommt. Aus diesem Grund muss der Kartenzeichner seine Karte umfärben. Nun sind aber die Materialien im 12. Jahrhundert noch sehr teuer, und der Kartenzeichner möchte möglichst selten in die Verlegenheit kommen, die Karte umfärben zu müssen. Gibt es eine Möglichkeit, die Anzahl der Umfärbungen zu minimieren?

Dieses Problem lässt sich einfach auf ein Spiel mit Graphenfärbung abbilden: Gegeben sei am Anfang des Spiels ein Parameter k und ein Subgraph H eines k-färbbaren Graphen $G = (V, E)$. Danach präsentiert der Entdecker sukzessive die restlichen Knoten von G. Der Kartenzeichner muss, ausgehend von der aktuellen Färbung, dem neuen Graphen eine zulässige k-Färbung zuweisen, möglichst ohne die Färbung von H zu verändern. Der Entdecker gewinnt, wenn der Kartenzeichner im Laufe des Spiels sämtliche k-Färbungen des Graphen H ausprobiert, um zur richtigen Lösung zu kommen.

Definition 6.7. Sei $G^{\prec} = (V, E, \prec)$ ein k-färbbarer Online-Graph mit N Knoten. Weiter sei $H = G_n$ der Subgraph von G mit den ersten n Knoten und c eine k-Färbung von H. Für $m > n$ ist die Färbung c G_m-erweiterbar, wenn es eine k-Färbung c' von G_m gibt, so dass $c'(v) = c(v)\ \forall v \in V(H)$ gilt.

Es kann nun gezeigt werden, dass es bei $k \geq 3$ für jede k-Färbung c des Ausgangsgraphen H einen k-färbbaren Obergraphen G_m, $m > n$ gibt, so dass die Färbung c nicht G_m-erweiterbar ist. Gleichzeitig sind aber alle anderen k-Färbungen $c' \neq c$ G_m-erweiterbar. Das heißt, der Entdecker kann nacheinander jeweils genau eine k-Färbung von H ungültig machen, so dass der Kartenzeichner sämtliche k-Färbungen von H ausprobieren muss. Es werden hierbei wieder nur nicht-isomorphische Färbungen von H betrachtet, d.h. zwei Färbungen c_1, c_2 sind genau dann verschieden, wenn sie zwei verschiedene Knotenpartitionen von $V(H)$ implizieren.
Die Anzahl möglicher verschiedener k-Färbungen eines leeren Graphen mit n Knoten ist durch $\sum_{s=1}^{k} S(n, s)$ bestimmt, wobei $S(n, s)$ die Anzahl der Partitionen einer n-elementigen Menge

in s nicht-leere Mengen bezeichnet (Stirling-Zahl 2.Art). Nach [1] ist dabei die Gleichung $S(n,s) = \frac{1}{s!}\sum_{i=0}^{s}(-1)^{s-i}\binom{s}{i}i^n$ erfüllt. Damit ist $\sum_{s=1}^{k} S(n,s) \approx \frac{k^n}{k!}$. Ein nicht-leerer Graph hat auf jeden Fall weniger mögliche k-Färbungen.

Das wichtigste Ergebnis von Beigel und Gasarch [5] ist, dass der Entdecker für $k \geq 3$ eine Gewinnstrategie hat, d.h. er kann anhand der aktuellen k-Färbung c immer einen neuen Obergraphen G' präsentieren, so dass die Färbung nicht G'-erweiterbar ist. Gleichzeitig sind aber alle anderen (noch möglichen) k-Färbungen $d \neq c$ G'-erweiterbar. Für $k = 2$ kann gezeigt werden, dass der Kartenzeichner eine Gewinnstrategie hat.

Zunächst folgt die Konstruktion eines Graphen, für den der Entdecker eine Gewinnstrategie hat.[1] Benötigt werden drei Graphenarten, die nachfolgend beschrieben sind.

Lemma 6.8. *Seien $k \geq 3$ und $m \geq 1$; sei $A = \{v_1, ..., v_m\}$ eine Knotenmenge und sei $v \notin A$ ein Knoten. Es existiert ein Graph $SAME(A, v)$, so dass*

1. *es eine k-Färbung c von $SAME(A, v)$ gibt, in der alle Knoten aus A dieselbe Farbe erhalten,*
2. *wenn c eine k-Färbung von $SAME(A, v)$ ist, in der alle Knoten aus A dieselbe Farbe haben, dann hat v auch diese Farbe, und*
3. *wenn c eine k-Färbung von A ist, in der nicht alle Knoten dieselbe Farbe haben, dann gibt es für jede Farbe $a \in \{1, ..., k\}$ eine Färbung c' von $SAME(A, v)$, die c erweitert, so dass $c'(v) = a$ gilt.*

Die erste Bedingung von Lemma 6.8 sichert, dass $A \cup \{v\}$ eine unabhängige Menge ist und im Graphen auch mit einer Farbe gefärbt werden kann.

Beweis. Bei $m = 1$ besteht $SAME(A, v)$ aus $k + 1$ Knoten: Die Knoten $v_1 \in A$ und v sind jeweils mit sämtlichen Knoten einer $k - 1$-Clique verbunden.

Für $m = 2$ sei $W = \{w_1, ..., w_{k-1}\}$ eine Clique mit $k - 1$ Knoten. Zur Konstruktion von $SAME(A, v)$ wird v mit jedem Knoten aus W verbunden, v_1 mit jedem Knoten aus $W \setminus \{w_{k-1}\}$ und v_2 mit w_{k-1}. Der Graph $SAME(A, v)$ erfüllt die Bedingungen von Lemma 6.8: Es gibt eine Färbung c, in der die Knoten aus A, nämlich v_1 und v_2, dieselbe Farbe erhalten: $c(w_i) = i \; \forall w_i \in W$ und $c(v_1) = c(v_2) = k$. Sei nun c eine k-Färbung von $SAME(A, v)$,

[1] Der Entdecker kann jeden beliebigen Graphen wählen, und die Knoten dann in einer beliebigen Reihenfolge ausgeben; er wird natürlich den Graphen mit der Knotenfolge wählen, so dass er gewinnt.

in der $c(v_1) = c(v_2)$ gilt (o.B.d.A. ist $c(v_1) = c(v_2) = 1$). Da $N(A) = W$ ist, kann kein Knoten $w_i \in W$ mehr die Farbe 1 annehmen. Somit müssen sich die restlichen $k-1$ Farben auf W verteilen. Es bleibt somit nur noch $c(v) = 1$ für v. Schließlich sei noch c eine Färbung von A mit $c(v_1) \neq c(v_2)$. Es können nun k Farben fast beliebig auf die k-Clique $W \cup \{v\}$ verteilt werden, das heißt, der Knoten v kann jede der k Farben erhalten.

Die restlichen Graphen bauen sich induktiv auf den Fall $m = 2$ auf: Sei $A = \{v_1, ..., v_m\}$ und seien $w_1, ..., w_{m-2}$ neue Knoten. Weiter sei $v = w_{m-1}$. Der Graph $SAME(A, v)$ setzt sich zusammen aus

$$SAME(A, v) = SAME(\{v_1, v_2\}, w_1) \cup \bigcup_{i=1}^{m-2} SAME(\{w_i, v_{i+2}\}, w_{i+1}). \tag{6.1}$$

Es bleibt zu zeigen, dass $SAME(A, v)$ die Bedingungen von Lemma 6.8 erfüllt. Man kann sich den resultierenden Graphen $SAME(A, v)$ als *Kette* von $m-1$ $SAME(A', \cdot)$-Graphen (mit $|A'| = 2$) vorstellen. Die Kettenglieder sind an den Knoten $w_1, ..., w_m - 2$ miteinander verbunden. Den Abschluss bildet der Knoten v.

Bedingung 1 gilt: Jedes Kettenglied enthält eine $k-1$-Clique. Sei jede Clique mit den Farben $1, ..., k-1$ gefärbt. Wie oben müssen nun die Knoten v_1, v_2 und w_1 im ersten Kettenglied, w_1, v_3 und w_2 im zweiten Kettenglied,..., w_{m-2}, v_m und $v = w_{m-1}$ im letzten Kettenglied die Farbe k erhalten.

Bedingung 2 gilt: Sei c eine k-Färbung von $SAME(A, v)$, in der $c(v_i) = c(v_j)$ $\forall v_i, v_j \in A$ gilt. Da die Kettenglieder selbst $SAME$-Graphen sind, gilt nun $c(w_1) = c(v_1) = c(v_2)$. Mit $c(v_3) = c(v_1) = c(w_1)$ lässt sich nun die Kette bis zum Ende vervollständigen. Es ist also auch $c(v) = c(v_1)$ erfüllt.

Bedingung 3 gilt: Sei c eine k-Färbung von A, und j der kleinste Index, für den v_j eine andere Farbe als v_{j+1} erhält. Zu bestimmen ist zunächst die Farbe von w_j. Da die Eigenschaft $c(v_i) = c(v_{i+1})$ $\forall i < j$ gilt, ist $c(w_{j-1}) = c(v_j)$ erfüllt. Damit ist $c(w_{j-1}) \neq c(v_{j+1})$, und für den Graphen $SAME(\{w_{j-1}, v_{j+1}\}, w_j)$ ist $c(w_j)$ frei wählbar. Nun gibt es zwei Möglichkeiten: Entweder ist $w_j = v$, dann ist der Beweis beendet. Im anderen Fall wird $c(w_j) \neq c(v_{j+2})$ gewählt. Damit wurde für das nachfolgende Kettenglied dieselbe Ausgangsbedingung geschaffen und die Reihe kann bis v fortgesetzt werden. □

Als nächstes wird ein k-färbbarer Graph $LESS(A)$ konstruiert, in dem eine Menge A mit $2 \leq j \leq k$ Knoten bei jeder zulässigen k-Färbung von $LESS(A)$ mindestens zwei Knoten

derselben Farbe besitzt.

Lemma 6.9. *Sei $2 \leq j \leq k$, und sei $A = \{v_1, ..., v_j\}$ eine Knotenmenge. Es gibt einen k-färbbaren Graphen $LESS(A)$, so dass*

1. *wenn c eine k-Färbung von $LESS(A)$ darstellt, dann gibt es mindestens zwei Knoten $v_a, v_b \in A$ mit $c(v_a) = c(v_b)$, d.h. $|\{c(v) : v \in A\}| < j$, und*
2. *jede $(j-1)$-Färbung von A kann zu einer k-Färbung von $LESS(A)$ erweitert werden.*

Beweis. Zur Konstruktion von $LESS(A)$ wird jeder Knoten aus A mit jedem Knoten aus einer $(k-j+1)$-Clique W verbunden. Es bleibt zu zeigen, dass die beiden Bedingungen erfüllt sind: Sei zuerst c eine k-Färbung von $LESS(A)$, in der alle Knoten aus A verschiedene Farben $1, ..., j$ erhalten. Dann sind für jeden Knoten $w \in W$ die Farben $1, ..., j$ verboten. Es bleiben $k-j$ Farben für eine $(k-j+1)$-Clique. Das heißt, die k-Färbung c ist nicht zulässig. Es muss also zwei Knoten in A geben, die dieselbe Farbe bekommen.
Sei nun c eine $j-1$-Färbung von A o.B.d.A. mit den Farben $1, ..., j-1$. Dann bleiben für die Knoten aus W mindestens noch $k-(j-1) = k-j+1$ Farben übrig. Man kann also eine beliebige $j-1$-Färbung von A auf eine k-Färbung von $LESS(A)$ erweitern. □

Schließlich wird mit Hilfe von Lemma 6.8 ein Graph $NEQ(A)$ konstruiert, der nicht erlaubt, dass die Knoten aus A mit einer Farbe gefärbt werden können.

Lemma 6.10. *Sei $k \geq 3$, und sei $A = \{v_1, ..., v_m\}$ eine Knotenmenge. Es gibt einen Graphen $NEQ(A)$, so dass*

1. *wenn c eine k-Färbung von $NEQ(A)$ ist, dann gibt es in A zwei Knoten v_i, v_j mit $c(v_i) \neq c(v_j)$, das heißt, $|\{c(v) : v \in A\}| > 1$, und*
2. *wenn c eine k-Färbung von A mit $|\{c(v) : v \in A\}| > 1$ ist, dann kann c zu einer k-Färbung von $NEQ(A)$ erweitert werden. Die Färbung c muss nicht unbedingt alle k Farben verwenden.*

Beweis. Es wird ein Graph I benötigt, der zwei neue Knoten x_1 und x_2 durch eine Kante verbindet. Dann ist

$$NEQ(A) = SAME(A, x_1) \cup SAME(A, x_2) \cup I. \tag{6.2}$$

Der Graph $NEQ(A)$ erfüllt die Bedingungen von Lemma 6.10: Sei zuerst c eine k-Färbung von $NEQ(A)$, in der alle Knoten aus A dieselbe Farbe, o.B.d.A. Farbe 1, erhalten. Nach Lemma 6.8 erhalten dann die Knoten x_1 und x_2 ebenfalls die Farbe 1, da sie jeweils in den Graphen $SAME(A, x_i)$ enthalten sind. Damit ist c aber nicht zulässig, da $\{x_1, x_2\}$ eine Kante in $NEQ(A)$ ist. Das heißt, für die Färbung von A müssen mindestens zwei Farben verwendet werden.
Sei nun c eine beliebige k-Färbung von A, d.h. eine Färbung, die mindestens zwei, aber höchstens k Farben verwendet. Nach Lemma 6.8 kann die Färbung c auf $SAME(A, x_1)$ und auf $SAME(A, x_2)$ erweitert werden. Ebenfalls nach Lemma 6.8 können die Farben von x_1 und x_2 beliebig gewählt werden, also auch so, dass dann $c'(x_1) \neq c'(x_2)$ gilt. □

Jetzt kann der Graph G' angegeben werden, der bei einer zulässigen k-Färbung c des Ausgangsgraphen H sicherstellt, dass c nicht G'-erweiterbar ist, aber gleichzeitig sind alle von c verschiedenen k-Färbungen G'-erweiterbar.

Lemma 6.11. *Seien $k \geq 3$, G ein k-färbbarer Graph, H ein Subgraph von G und c eine zulässige G-erweiterbare k-Färbung von H. Dann existiert ein Graph $G' = SPOIL(k, G, H, c)$, so dass*

1. *c nicht G'-erweiterbar ist,*
2. *wenn $d \neq c$ eine beliebige, G-erweiterbare Färbung von H ist, dann ist d auch G'-erweiterbar.*

Beweis. Es wird angenommen, dass c genau $j \leq k$ Farben verwendet. Sei zunächst $j = 1$. In diesem Fall haben alle Knoten aus H dieselbe Farbe. Dann ist $G' = G \cup NEQ(H)$. Der Graph G' erfüllt die Bedingungen von Lemma 6.11: Der Graph $NEQ(H)$ fordert nach Lemma 6.10 nur, dass für die Menge H mindestens zwei Farben verwendet werden. Jede von c verschiedene Färbung von H verwendet mindestens zwei Farben, und ist demnach $NEQ(H)$-erweiterbar.
Sei nun $j \geq 2$. Es seien $A_i = \{v \in H : c(v) = i\}$ und $v_1, ..., v_j$ neue Knoten. Dann ist

$$G' := G \cup \left(\bigcup_{i=1}^{j} SAME(A_i, v_i) \right) \cup LESS(\{v_1, ...v_j\}). \tag{6.3}$$

Auch hier ist noch zu zeigen, dass G' die geforderten Eigenschaften besitzt: Nach Lemma 6.8 müssen die Knoten $v_1, ...v_j$ jeweils dieselbe Farbe wie die Knoten aus $A_1, ..., A_j$ erhalten.

Das bedeutet, dass für $v_1, ..., v_j$ genau j Farben verwendet werden. Genau das schließt aber der Graph $LESS(\{v_1, ..., v_j\})$ aus. Damit ist c nicht G'-erweiterbar. Sei nun $d \neq c$ eine andere G-erweiterbare Färbung von H. Die Bedingung $d \neq c$ impliziert eine andere Partition der Knotenmenge von H. Das bedeutet, es gibt mindestens eine Menge A_i, in der es einen Knoten v_0 gibt mit $d(v_0) \neq i$. Damit kann im Graphen $SAME(A_i, v_i)$ die Farbe von v_i frei gewählt werden. Das bedeutet, die Färbung d ist G'-erweiterbar. □

Der konstruierte Graph $SPOIL(k, G, H, c)$ hat im Fall $|\{c(v) : v \in H\}| = 1$ die Knoten aus G; hinzu kommen zwei $k-1$-Cliquen für die Graphen $SAME$ und zwei Knoten x_1, x_2. Insgesamt entsteht für den Fall, dass nur eine Farbe für H verwendet wurde, ein neuer Graph G' mit $|V(G)| + 2k$ Knoten. Im anderen Fall ($|\{c(v) : v \in H\}| = j \geq 2$) kommen neben den Knoten aus G noch j Knoten $v_1, ..., v_j$, j Cliquen der Größe $k-1$, sowie eine Clique der Größe $k-j+1$ hinzu. Damit enthält G' genau $|V(G)|+j \cdot k+k-j+1 = |V(G)|+(j+1) \cdot (k-1)+2$ Knoten.
Damit kann nun der Hauptsatz dieses Abschnittes formuliert werden:

Satz 6.12. *Seien $3 \leq k \leq n$ und H ein k-färbbarer Graph mit n Knoten. Der Entdecker hat eine Gewinnstrategie beim Spiel gegen den Kartenzeichner.*

Beweis. Das Spiel startet mit $G_1 = H$ und geht etappenweise voran. In jeder Etappe $s \geq 2$ erstellt der Kartenzeichner eine k-Färbung c_s von H, die auf G_{s-1} erweiterbar ist. Der Entdecker konstruiert einen neuen Graphen $G_s = SPOIL(k, G_{s-1}, H, c_s)$. Nach Lemma 6.11 ist c_s nicht G_s-erweiterbar, aber alle anderen G_{s-1}-erweiterbaren Färbungen von H sind G_s-erweiterbar. Damit kann der Entdecker immer genau eine Färbung unzulässig machen, so dass der Kartenzeichner gezwungen ist, sämtliche k-Färbungen von H zu probieren. □

Bisher wurde angenommen, dass $k \geq 3$ gilt. Aber auch für den Fall $k = 2$ kann eine Aussage getroffen werden.

Satz 6.13. *Seien $k = 2$ und $n \in \mathbb{N}$. Der Kartenzeichner hat eine Gewinnstrategie für das Spiel mit einem 2-färbbaren Graphen.*
Das bedeutet, er muss nicht alle möglichen 2-Färbungen von H ausprobieren. Genauer gesagt, genügt es, höchstens n Färbungen zu probieren.

Beweis. Die Gewinnstrategie für $k = 2$ ist folgende: Zunächst färbt der Kartenzeichner den Graphen H beliebig mit 2 Farben. Der Entdecker präsentiert immer einen neuen Graphen G'. Jedes Mal, wenn die aktuelle Färbung nicht G'-erweiterbar ist, dann wählt der Kartenzeichner eine beliebige G'-erweiterbare Färbung. Dies geschieht nur dann, wenn der Entdecker zwei Zusammenhangskomponenten von H verbindet. Insgesamt kann der Graph H aber nur n Zusammenhangskomponenten besitzen (jeder Knoten ist isoliert). Das bedeutet, der Kartenzeichner muss nur n Färbungen verwenden. □

Beigel und Gasarch [5] verwenden dieses Ergebnis in der rekursiven Graphentheorie, um die P=NP-Frage widerzuspiegeln. Es wird gefragt, ob ein Problem, für das ein exponentieller Brute-Force-Algorithmus bekannt ist, auch effizienter gelöst werden kann. Es wurde damit gezeigt, dass der Brute-Force-Algorithmus optimal ist.

Definition 6.14. Ein Graph $G = (V, E)$ ist rekursiv, wenn jeder Knoten $v \in V$ einen endlichen Grad besitzt und $V \subseteq \mathbb{N}$ bzw. $E \subseteq \mathbb{N}^2$ entscheidbare Mengen sind.

Definition 6.15. Ein rekursiver Graph $G = (V, E)$ heißt rekursiv k-färbbar, wenn es eine berechenbare Funktion $f : V \to \{1, ..., k\}$ gibt, die eine zulässige Färbung von G ist.

Definition 6.16. Sei $G = (V, E)$ ein k-färbbarer rekursiver Graph. Eine lokale k-Färbung von G ist eine Funktion, die für eine endliche Menge $H \subseteq V$ eine G-erweiterbare k-Färbung produziert.

Es kann z.B. gezeigt werden [5], dass es einen 3-färbbaren Graphen gibt, der für alle k nicht rekursiv k-färbbar ist. Der Hauptsatz von Beigel und Gasarch [5] besagt, dass es für $k \geq 3$ einen k-färbbaren rekursiven Graphen G gibt, so dass jeder Färbungsalgorithmus (Kartenzeichner) alle möglichen lokalen k-Färbungen über eine beliebige Teilmenge $H \subseteq V$ ausprobieren muss. Eine Variante dieses Spiels erlaubt es dem Kartenzeichner, $m > k$ Farben zu verwenden. Der Entdecker muss aber den Graphen k-färbbar belassen. Man kann zeigen, dass der Kartenzeichner für $m \geq 2k$ eine Gewinnstrategie hat. Bei Werten $k < m < 2k$ konnte noch keine Aussage getroffen werden [5].

6.3 Färbungen bei Hypergraphen

Manche Probleme lassen sich nicht gut mit Graphen modellieren, obwohl eine graphenähnliche Struktur von Vorteil ist. Aus diesem Grunde wird der Begriff des Hypergraphen eingeführt.

Ein Hypergraph $H = (V, E)$ ist ein geordnetes Paar aus Knoten und Kanten, wobei die Kanten beliebige Teilmengen der Knotenmenge sein können, d.h. $E \subseteq \{e : e \subseteq V\}$. Für die Färbung solcher Graphen gibt es zwei verschiedene Konzepte.

Definition 6.17. Sei $H = (V, E)$ ein Hypergraph. Eine k-Färbung von H ist eine Funktion $c : V \to \{1, ..., k\}$, so dass $c(v_i) \neq c(v_j)$, wenn $v_i \neq v_j$ in einer Kante $e \subseteq V$ liegen.

Definition 6.18. Sei $H = (V, E)$ ein Hypergraph. Eine k-Färbung von H ist eine Funktion $c' : V \to \{1, ..., k\}$, so dass in jeder Kante $e \subseteq V$ mit $|e| \geq 2$ mindestens zwei Knoten v_i, v_j mit $c'(v_i) \neq c'(v_j)$ liegen.

Bei einer Färbung nach Definition 6.17 kann man sich einen korrespondierenden Graphen $G(H) = (V, E')$ vorstellen, in dem jede Kante aus H einer Clique in $G(H)$ entspricht. Eine Färbung in H ist damit gleichbedeutend mit einer Färbung in $G(H)$, da alle Knoten einer Kante verschiedene Farben erhalten müssen, ebenso wie bei einer Clique. Es genügt also, nur Färbungsprobleme nach Definition 6.18 zu betrachten.

Satz 6.19. *Es ist ein* NP*-vollständiges Problem, zu entscheiden, ob ein gegebener Hypergraph 2-färbbar ist.*

Beweis. Es wird das Problem NAE-Satisfiability auf Hypergraph 2-Coloring reduziert. Das Problem NAE-Satisfiability ist wie folgt definiert:
Eine Boolesche Formel in Konjunktiver Normalform ist im NAE-Sinne erfüllbar, wenn es eine erfüllende Variablenbelegung gibt, so dass in jeder Klausel nicht alle Literale denselben Wert haben (Not All Equal). Das impliziert sofort, wenn $x_1, ..., x_n$ eine erfüllende Belegung im NAE-Sinne ist, dann ist auch $\neg x_1, ..., \neg x_n$ eine erfüllende Belegung im NAE-Sinne.
Die Reduktion ist nun einfach: Als Knoten gelten die Literale $x_1, ..., x_n, \neg x_1, ..., \neg x_n$, und eine Kante entspricht einer Klausel. So wird z.B. die Klausel $x_1 \vee \neg x_2 \vee x_3$ zur Kante $\{x_1, \neg x_2, x_3\}$. Hinzu kommen die Kanten $\{x_i, \neg x_i\}$ $\forall i$. Eine zulässige 2-Färbung des Hypergraphen entspricht eine Zuweisung der Werte 0 und 1 zu den Variablen $x_1, ..., x_n$. Die Zweierkanten $\{x_i, \neg x_i\}$ sichern, dass x_i und $\neg x_i$ jeweils unterschiedliche Farben erhalten, also kann man eindeutig die Farbwerte von $x_1, ..., x_n$ den Booleschen Werten *true* und $false$ zuordnen. In jeder Kante müssen auf jeden Fall die Werte 0 und 1 auftreten, also ist die Ausgangsformel genau dann im NAE-Sinne erfüllbar, wenn der Hypergraph 2-färbbar ist. Der Beweis, dass NAE-Satisfiability selbst NP-vollständig ist, befindet sich in [33]. □

Nun kann man Hypergraphen noch einschränken, indem jede Kante genau t Knoten besitzen darf.

Definition 6.20. Ein Hypergraph $H = (V, E)$ heißt t-uniform, wenn $|e| = t$ für jede Kante $e \in E$ gilt.[2] Ein 2-uniformer Hypergraph ist demnach ein Graph im klassischen Sinne.

Man kann sogar sagen, dass die Entscheidung, ob ein t-uniformer Hypergraph 2-färbbar ist, für $t > 2$ NP-schwer ist. Sogar eine Färbung mit einer konstanten Anzahl Farben ist NP-schwer [11]. Damit ergibt sich für jedes Zahlentripel t, k, c mit $t > 2$ und $c > k > 1$, dass die Färbung eines k-färbbaren t-uniformen Hypergraphen mit c Farben ein NP-schweres Problem ist.
Die meisten Fortschritte zur (Nicht-)Approximierbarkeit konnte man mit Hilfe der probabilistisch überprüfbaren Beweise (PCP – probabilistic checkable proofs) erzielen [3].
Die Färbung von Hypergraphen findet vor allem auch in der Informationstheorie ihre Anwendung.

6.4 Weitere Probleme

In diesem Abschnitt werden noch einige Probleme vorgestellt, die mit der Färbbarkeit eng verwandt sind.

Definition 6.21. Eine zulässige k-Färbung eines Graphen heißt *achromatisch*, wenn jeweils zwei Farbklassen C_i, C_j mit $i \neq j$ durch mindestens eine Kante verbunden sind. Die achromatische Zahl $\psi(G)$ ist der maximale Wert k, für den eine achromatische k-Färbung existiert.

Die sequenziellen Algorithmen 2.1 und 2.2 produzieren zwangsläufig achromatische Färbungen, da immer nur dann eine neue Farbe verwendet wird, wenn ein Knoten zu allen anderen Farben adjazent ist.

Fakt 6.22. *Sei $G = (V, E)$ ein Graph und k die Anzahl der vom Greedy-Algorithmus 2.1 verwendeten Farben. Dann gilt: $\chi(G) \leq k \leq \psi(G)$.*

Beweis. Jeder Färbungsalgorithmus muss mindestens $\chi(G)$ Farben verwenden. Der Greedy-Algorithmus benutzt eine neue Farbe nur dann, wenn keine andere Farbe mehr frei ist, d.h.

[2] Mit Hilfe von t-uniformen Hypergraphen kann man z.B. die Definition eines *planaren* Graphen ausweiten: Ein t-planarer Hypergraph ist ein t-uniformer Hypergraph, der so in den Raum $\mathbb{R}^t$ gezeichnet werden kann, dass sich keine Kanten ($t-1$-dimensionale Hyperebenen) schneiden.

wenn der betrachtete Knoten zu allen bereits verwendeten Farben adjazent ist. Das heißt, der Greedy-Algorithmus verwendet höchstens $\psi(G)$ Farben. □

Für allgemeine Graphen G ist die Bestimmung der achromatischen Zahl $\psi(G)$ ein NP-schweres Problem. Kortsarz und Krauthgamer [28] gaben einen Algorithmus an, der für Graphen mit n Knoten die achromatische Zahl mit der Güte $O(n \cdot \frac{\log \log n}{\log n})$ approximiert. Weiterhin zeigen sie, dass unter der Annahme P≠NP kein Algorithmus existiert, der $\psi(G)$ auf einen Faktor $2-\varepsilon$, $\varepsilon > 0$, approximieren kann [28].

Ähnlich zum Problem der Bestimmung der achromatischen Zahl ist das Problem Pseudoachromatic Number. Hier wird nach einer Partition der Knotenmenge V in möglichst viele Mengen P_i gefragt, so dass jeweils zwei Mengen $P_i, P_j, i \neq j$ durch mindestens eine Kante verbunden sind [38]. Der Begriff achromatische Zahl wird hierbei sozusagen *aufgeweicht*, da keine zulässige Färbung mehr betrachtet werden muss.

Einen weiteren Schritt in dieser Richtung bildet das Problem (unweighted) Max-k-Cut. Hierbei wird die Knotenmenge V so in k disjunkte Mengen $P_i,\ i = 1, ..., k$ aufgeteilt, dass die Anzahl der Kanten zwischen den Mengen maximiert wird.

Schließlich gibt es in diesem Zusammenhang das Problem Weighted Max-k-Cut, bei dem die Kanten zusätzlich mit Gewichten versehen sind.

Karger, Motwani und Sudan [24] gaben neben dem Problem Vector-k-Coloring auch das Problem Strict Vector-k-Coloring an. Eine strenge Vektor-k-Färbung ist wieder eine Zuweisung von Einheitsvektoren u_i zu den Knoten $i \in V$ des Graphen, so dass die Bedingung $\langle u_i, u_j \rangle = -\frac{1}{k-1}$ für jede Kante $\{i, j\} \in E$ erfüllt ist. Die strenge Vektorchromatische Zahl ist äquivalent zur θ-Funktion des Komplementgraphen [24]. Jeder k-färbbare Graph ist auch streng Vektor-k-färbbar, und man vermutet, dass streng Vektor-k-färbbare Graphen im Allgemeinen eine kleinere chromatische Zahl als Graphen haben, die nur Vektor-k-färbbar sind. Das Problem Strict Vector-k-Coloring ist ebenfalls mittels semidefiniter Programmierung in Polynomialzeit lösbar.

Zusätzlich sollte in diesem Atemzug noch das Problem Strong Vector-k-Coloring genannt werden. Eine starke Vektor-k-Färbung verlangt zusätzlich zur strengen Vektor-k-Färbung, dass für alle Knoten $i, j \in V$ gilt: $\langle u_i, u_j \rangle \geq -\frac{1}{k-1}$ [9].

Bei manchen Aufgabenstellungen reicht es nicht aus, dass man möglichst wenig Farben verwendet. Eines dieser Probleme ist Minimum Equicoloring, das verlangt, dass zu jeder

Farbklasse gleich viele Knoten gehören bzw. die Größenunterschiede zwischen den Farbklassen maximal 1 ist. Das Problem ist äquivalent zur Partitionierung der Knotenmenge in k unabhängige Mengen $I_1, ..., I_k$, so dass $\lfloor n/k \rfloor \leq |I_j| \leq \lceil n/k \rceil \; \forall j$ gilt.
Auf der anderen Seite der Färbungsprobleme steht das Problem Edge Coloring. Hierbei sollen die Kanten eines Graphen so gefärbt werden, dass zwei inzidente Kanten (also Kanten, die einen gemeinsamen Knoten haben) verschiedene Farben erhalten. Eine Kantenfärbung kann durch eine Knotenfärbung des *Kantengraphen* dargestellt werden:

Definition 6.23. Sei $G = (V, E)$ ein Graph. Der Kantengraph $\mathcal{G}(G)$ ist der Graph $\mathcal{G}(G) = (E, \{\{e, f\} \in [E]^2 : e \cap f \neq \emptyset\})$. Die Knoten in $\mathcal{G}(G)$ entsprechen den Kanten aus G, und zwei Knoten in $\mathcal{G}(G)$ sind genau dann miteinander verbunden, wenn die entsprechenden Kanten einen Knoten gemeinsam haben.

Allerdings kann nicht jeder Graph G als Kantengraph aufgefasst werden. Einfachstes Beispiel ist der Graph $K_{1,3}$, also der Graph mit vier Knoten, der aus den Kanten $\{1, 2\}, \{1, 3\}$ und $\{1, 4\}$ besteht. Wäre der Graph $K_{1,3}$ ein Kantengraph, so müssten im Originalgraphen G die Kanten 2,3 und 4 jeweils mit der Kante 1 adjazent sein, aber nicht untereinander. Da eine Kante aber nur zwei inzidente Knoten besitzt, kann $K_{1,3}$ kein Kantengraph sein. Insgesamt gibt es neun Strukturen, die keine Kantengraphen sein können [35]. Taucht eine der Strukturen als induzierter Teilgraph eines Graphen G auf, so kann G kein Kantengraph sein.
Eine Kantenfärbung eines Graphen mit maximalem Grad Δ muss mindestens Δ Farben verwenden, da jede Kante, die zu einem Knoten inzident ist, mit einer eigenen Farbe gefärbt werden muss.

Kapitel 7

Implementation von Färbungsalgorithmen

Es wurden einige einfache Algorithmen zur Graphfärbung implementiert. Zunächst wird ein Graph erzeugt; hierfür gibt es vier Methoden:

1. Graph manuell einlesen: Zuerst wird die Anzahl n der Knoten angegeben und die Knotenmenge $V = \{0, ..., n-1\}$ erstellt, und dann wird jede Kante $\{i, j\}$ genau einmal per Hand eingegeben. Diese Methode eignet sich für kleine Graphen, an denen man verschiedene Algorithmen testen möchte.

2. Graph aus Datei einlesen: Der Graph muss im DIMACS-Format [16] gespeichert sein. Als Beispielgraphen dienen Instanzen wie *Book-Graphen*, *Queen-Graphen*, *Scheduling-Graphen* und zufällige Graphen.

3. Zufälligen Graphen erstellen: Nach Eingabe der Knotenanzahl n und der Kantenwahrscheinlichkeit p wird ein zufälliger Graph erzeugt, indem jede mögliche Kante v, w mit der Wahrscheinlichkeit p eingefügt wird.

4. Zufälligen k-färbbaren Graphen erstellen: Nach Eingabe von Knotenanzahl n, Kantenwahrscheinlichkeit p und gewünschter Farbanzahl k wird ein zufälliger Graph erzeugt, der garantiert k-färbbar ist. Hierfür wird Algorithmus 1.3 verwendet.

Im Programm wird der Graph durch seine Adjazenzmatrix repräsentiert. Zur Färbung des erzeugten Graphen werden nacheinander verschiedene Algorithmen verwendet und die jeweiligen Ergebnisse (Anzahl verwendeter Farben) ausgegeben. Implementiert wurden folgende Algorithmen:

1. Der Greedy-Algorithmus (Algorithmus 2.1).
2. Ein Algorithmus color-IS, der den Graphen mit Hilfe unabhängiger Mengen färbt. Die unabhängigen Mengen werden mit einem Greedy-Algorithmus bestimmt.
3. Der DSATUR-Algorithmus 2.2.
4. Der Algorithmus von Johnson (Algorithmus 2.9).
5. Der Algorithmus von Wigderson (Algorithmus 2.12 bzw. 2.11). Allerdings wird der Restgraph nach Abbruch der Schleife mit dem DSATUR-Algorithmus gefärbt.
6. Der exakte Algorithmus 2.3 (Greedy exakt). Hier wird die Anzahl der Backtracks gezählt und am Ende mit ausgegeben. Als Ausgangspunkt wird eine DSATUR-Färbung verwendet.
7. Der exakte Algorithmus 2.4 (DSATUR exakt). Zur Bestimmung der Clique als Ausgangspunkt wird Algorithmus 5.2 verwendet. Die Backtracks werden wieder gezählt und ausgegeben.

Bei allen Algorithmen wurde ebenfalls die Laufzeit gemessen. Zum Vergleich der Algorithmen wurden zunächst einige k-färbbare zufällige Graphen (Punkt 4) erzeugt und gefärbt. Es zeigt sich, dass bei großen Kantenwahrscheinlichkeiten $\Omega(\frac{1}{n})$ jeder Graph im Sinne von Turner [34] (siehe auch Kapitel 5) einfach färbbar ist. Die größten Unterschiede treten bei Graphen mit kleiner Kantenwahrscheinlichkeit auf. Hier lässt sich auch ein deutlicher Performance-Unterschied zwischen den beiden exakten Algorithmen feststellen. Selbst wenn der exakte Greedy-Algorithmus 2.3 mit einer kleinen Farbanzahl gestartet wird (als Ausgangspunkt dient der DSATUR-Algorithmus 2.2), benötigt er deutlich mehr Backtracks als Algorithmus 2.4. Selbst die schnellere Auswahl des nächsten zu färbenden Knoten (um den Faktor $O(n)$ schneller) kann die Laufzeit nicht drücken, da die Anzahl der Backtracks die von Algorithmus 2.4 um einen exponentiellen Faktor übersteigt.
Bei sämtlichen kleinen Graphen wurde außerdem festgestelt, dass unter den Approximationsalgorithmen der DSATUR-Algorithmus 2.2 mindestens eine genauso gute Approximation liefert wie die Algorithmen von Wigderson (2.11) und Johnson (2.9).
In Tabelle 7.1 ist für $k = 3$ aufgezeigt, wie viele Farben von den implementierten Algorithmen verwendet werden. Der Algorithmus von Wigderson wurde hierbei mit dem Algorithmus von

n	k	p	Greedy	DSATUR	Wigderson	Backtracks exact-DSATUR
100	3	10/n	6	4	4	111
200	3	10/n	6	4	4	360
400	3	10/n	6	4	4	1398
500	3	10/n	7	5	5	5896

Tabelle 7.1: Vergleich einiger Algorithmen bei verschiedenen 3-färbbaren Graphen

Johnson kombiniert, d.h. es wurde von beiden Algorithmen das bessere Ergebnis verwendet. Zunächst ist das Verfahren von Johnson deutlich besser, aber bei größeren Graphen (ab 500 Knoten) zeichnen sich leichte Vorteile für den Algorithmus von Wigderson ab.

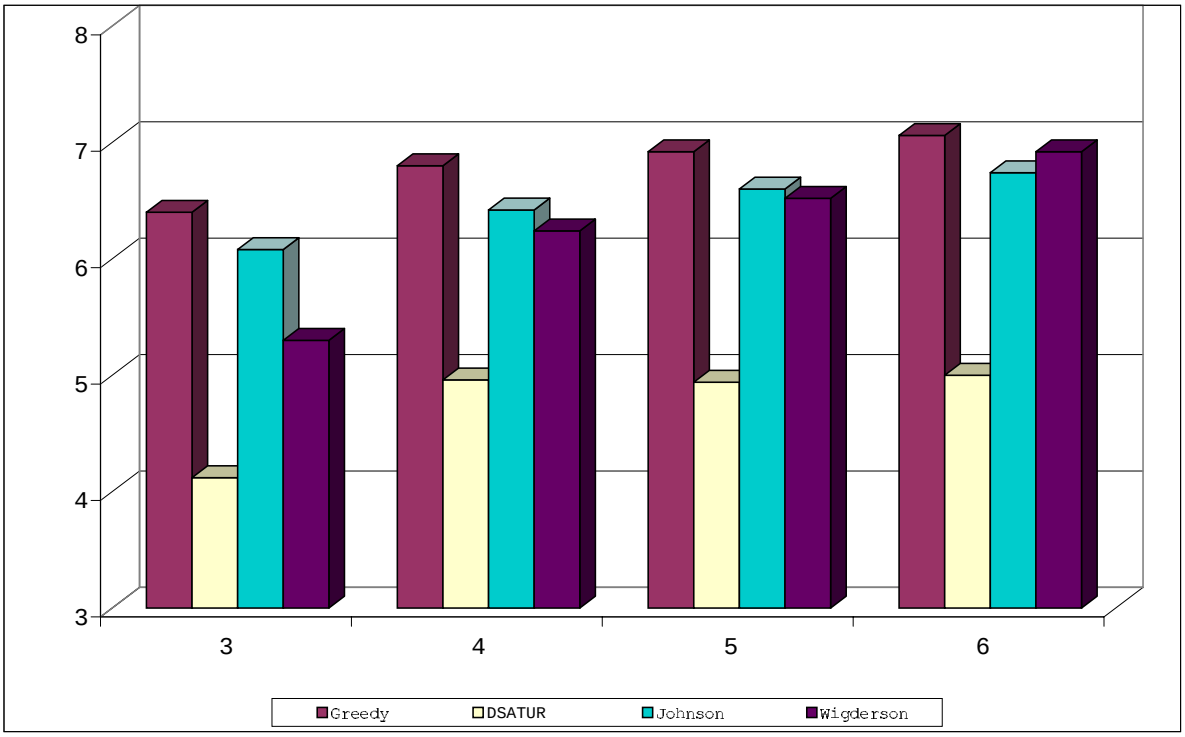

Abbildung 7.1: Anzahl verwendeter Farben bei zufälligen k-färbbaren Graphen

Abbildung 7.1 zeigt für kleine Werte von n und k, sowie für entsprechende Kantenwahrscheinlichkeiten ($O(1/n)$) die Anzahl der Farben, die zur Färbung eines zufälligen Graphen benutzt wurden.

In einem weiteren Versuch wurden zufällige Graphen generiert und diese mit den Approximationsalgorithmen Greedy (Algorithmus 2.1), DSATUR (Algorithmus 2.2) und von Wigderson/Johnson (Algorithmus 2.12) gefärbt. Die Ergebnisse sind in Tabelle 7.2 aufgeführt. Es

n	p	Greedy	DSATUR	Wigderson	Johnson
100	0.2	10	9	16	10
200	0.2	18	13	23	15
400	0.2	28	23	42	24
500	0.2	32	27	47	28
1000	0.2	55	47	82	47
100	$10/n$	7	6	10	7
200	$10/n$	7	5	9	8
400	$10/n$	8	6	10	7
500	$10/n$	7	5	5	7
1000	$10/n$	8	5	5	7

Tabelle 7.2: Anzahl verwendeter Farben bei zufälligen Graphen

zeigt sich, dass bei relativ hohen Kantenwahrscheinlichkeiten und kleinen Graphen ($n \leq 200$ und $p = O(1)$) die Algorithmen 2.2 (DSATUR) und 2.9 (Johnson) fast gleichauf liegen, während der Algorithmus von Wigderson (Algorithmus 2.11) weit abgeschlagen ist und fast doppelt so viele Farben verwendet wie das Verfahren DSATUR. Bei deutlich größeren Graphen ($n \geq 1000$) ist dagegen der Algorithmus von Johnson (Algorithmus 2.9) deutlich besser als der DSATUR-Algorithmus. Im Gegensatz dazu stehen die Ergebnisse mit kleiner Kantenwahrscheinlichkeit ($p = O(\frac{1}{n})$). Hier ist zwar immer noch der Algorithmus DSATUR ungeschlagen, aber das Verfahren von Johnson bleibt bei großen Graphen hinter dem Algorithmus von Wigderson zurück. Auffallend ist weiterhin, dass DSATUR durchweg weniger Farben verwendet als der Greedy-Algorithmus. Dies wird durch Abbildung 7.2 verdeutlicht. Dazu wurden zufällige Graphen verschiedener Größen generiert und mit den implementierten Färbungsalgorithmen gefärbt. Die Kantenwahrscheinlichkeit wurde wieder in Abhängigkeit der Knotenzahl gesetzt, so dass die entstehenden Graphen in etwa dieselbe chromatische Zahl besitzen.

Zu den Laufzeiten lässt sich folgendes sagen: Die Algorithmen, die unabhängige Mengen zur Färbung verwenden (der Algorithmus color-IS und Algorithmus 2.9), benötigen deutlich mehr Zeit als die einfachen Algorithmen 2.1 und 2.2.

Schließlich wurden die bereits genannten Approximationsalgorithmen mit verschiedenen vordefinierten Graphen gestartet, u.a. mit Queen-Graphen, Mycielski-Graphen und Leighton-Graphen. Die letzten beiden Graphklassen haben eine bekannte chromatische Zahl, für die Queen-Graphen kann man nur für kleine Knotenzahlen (Schachbrettgröße) eine Aussage treffen. In der Tabelle 7.3 sind die Ergebnisse aufgeführt, gemeinsam mit der chromatischen Zahl

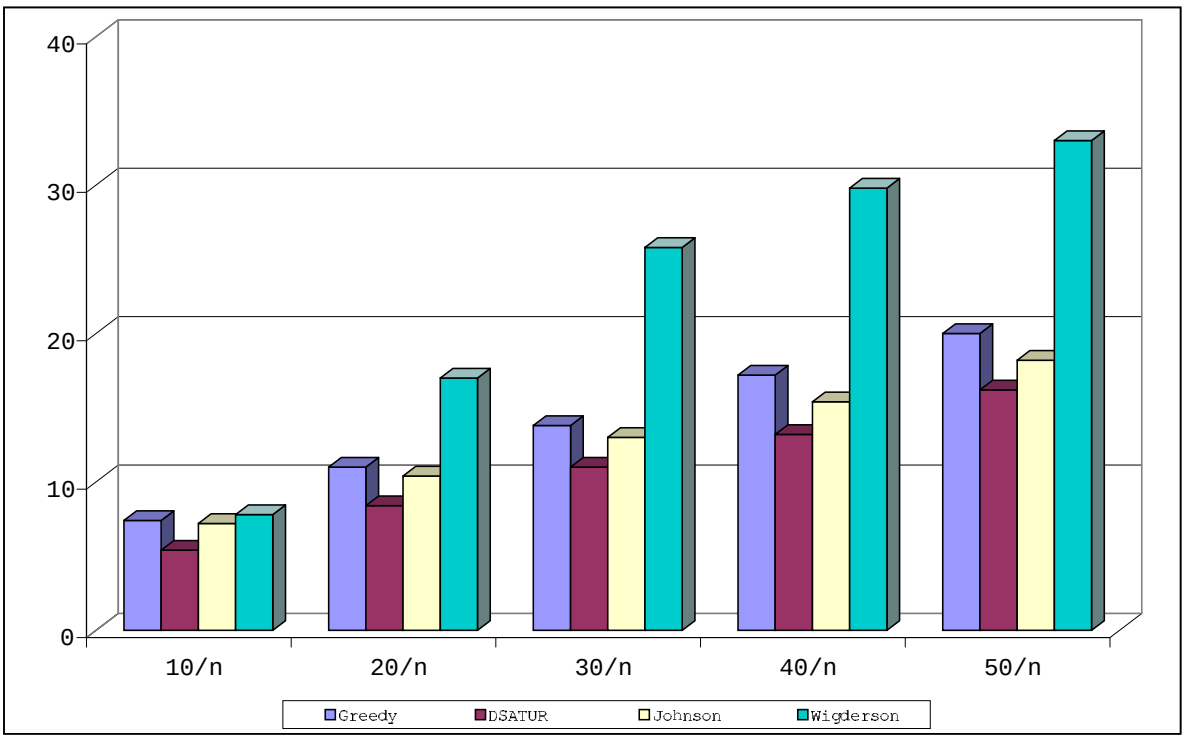

Abbildung 7.2: Anzahl verwendeter Farben bei zufälligen Graphen mit geringer Kantenwahrscheinlichkeit

des Graphen (soweit bekannt). Zunächst wurden einige konstruierte Graphen getestet.
Die Queen-Graphen der Größe n repräsentieren ein $n \times n$-Schachbrett, und die Kanten verbinden jeweils Knoten, die auf dem Schachbrett in derselben Zeile, Spalte oder Diagonale liegen. Damit werden die Zugmöglichkeiten einer Dame beim Schachspiel nachgestellt.
Die Mycielski-Graphen im zweiten Block sind Graphen, die aus der Mycielski-Transformation hergeleitet werden: Enthält ein k-färbbarer Graph G keine Dreiecke, dann entsteht durch die Mycielski-Transformation ein $k+1$-färbbarer Graph G', der ebenfalls keine Dreiecke enthält. Als Ausgangsgraph gilt hier ein Graph mit zwei Knoten x, y, die durch eine Kante verbunden sind. Dieser Graph enthält keine Dreiecke und hat die chromatische Zahl 2 (myciel1). Der Graph myciel$k$ hat die chromatische Zahl $k+1$.

Definition 7.1. Der Mycielski-Graph myciel1 besteht aus zwei Knoten x_1, x_2 die durch eine Kante verbunden sind. Der Mycielski-Graph myciel$(i+1) = (V', E')$ lässt sich aus dem

Name	n	$\chi(G)$	Greedy	DSATUR	Wigderson	Johnson
queen6_6	36	7	11	9	11	8
queen7_7	49	7	10	10	14	10
queen8_8	64	9	13	13	19	12
queen9_9	81	10	16	12	21	13
queen10_10	100	?	16	14	29	14
queen11_11	121	11	17	14	34	16
queen16_16	256	?	25	21	73	22
myciel3	11	4	4	4	5	4
myciel4	23	5	5	5	7	5
myciel5	47	6	6	6	8	6
myciel6	95	7	7	7	10	7
le450_5a	450	5	14	10	37	10
le450_5c	450	5	17	11	36	9
le450_15b	450	15	22	17	75	21
le450_15c	450	15	30	24	63	30
le450_25a	450	25	28	25	73	31
le450_25c	450	25	37	28	77	38
school1	385	?	42	20	25	36
school1_nsh	352	?	39	26	40	32
anna	138	11	12	11	16	13
david	87	11	12	11	14	13
homer	561	13	15	13	21	16
huck	74	11	11	11	13	11
jean	80	10	10	10	19	11
mulsol.i.1	197	49	49	49	50	51
mulsol.i.4	185	31	31	31	46	31
fpsol2.i.1	496	65	65	65	66	65
fpsol2.i.3	425	30	30	30	44	35
inithx.i.1	864	54	54	54	56	54
inithx.i.3	621	31	31	31	46	33
zeroin.i.1	211	49	49	49	62	51
zeroin.i.3	206	30	30	30	45	30

Tabelle 7.3: Verhalten von Approximationsalgorithmen auf Benchmark-Graphen

Graphen myciel$i = (V, E)$ wie folgt konstruieren:

$$V' := (V \times \{0,1\}) \cup \{x\} = \{(v,0),(v,1) : v \in V\} \cup x \text{ mit } x \notin V \text{ und}$$
$$E' := \{\{(v,0),(w,0)\} : \{v,w\} \in E\} \cup \{\{(v,0),(w,1)\} : \{v,w\} \in E\} \cup \{\{(v,1),x\} : v \in V\}.$$

Das heißt, die Knoten des Graphen myciel$i$ werden verdoppelt, und die Originalknoten werden mit den Kopien verbunden, wenn auch die Originalkante existiert. Zusätzlich werden die Kopien der Knoten mit einem neuen Knoten x verbunden. Die Konstruktion gewährleistet, dass jede Knotenkopie dieselbe Farbe erhalten muss wie ihr Original. Das bedeutet, der Zusatzknoten x muss eine neue Farbe bekommen. Gleichzeitig enthält der konstruierte Graph ebenso wie der Ausgangsgraph keine Kreise der Länge 3.
Den dritten Block bilden Leighton-Graphen. Sie haben eine feste chromatische Zahl, aber eine weite Streuung der Knotengrade. Die School-Graphen sind ein Spezialfall der Registerallokation. Hierbei soll ein Stundenplan mit möglichst wenig Unterrichtseinheiten erstellt werden, so dass ein Lehrer nie zwei Klassen gleichzeitig unterrichten muss. In der Variante school1_nsh soll zusätzlich die Raumbelegung berücksichtigt werden.
Die Book-Graphen wurden aus großen Werken der Weltliteratur konstruiert. Die Knoten sind hierbei die auftretenden Personen, und eine Kante zeigt an, ob sich die beiden Personen im Laufe der Geschichte begegnen. Als Beispiele wurden die Werke *Anna Karenina*, *David Copperfield*, Homers *Ilias*, *Huckleberry Finn* und *Les Miserables* (jean) verwendet.
Den zweiten Teil der Tabelle 7.3 bilden Graphen zur Registerallokation bei verschiedenen Rechenoperationen (Gleitkommaoperationen, Multiplikationen etc.)
Man sieht deutlich, dass bei den meisten der aus theoretischen Überlegungen erstellten Graphen kaum ein Algorithmus gute Ergebnisse liefert. Hierfür sind dann Approximationsalgorithmen gedacht, die die chromatische Zahl in Abhängigkeit von n bestimmen. Trotzdem liefert der Algorithmus 2.2 (DSATUR) fast immer die besten Ergebnisse bei den verglichenen Algorithmen. Der einzige Algorithmus, der auch nur annähernd in die Nähe von DSATUR kommt, ist der Algorithmus von Johnson [23] (Algorithmus 2.11).
Im Gegensatz dazu wurden sämtliche Instanzen zur Registerallokation exakt gelöst. Nur der Algorithmus von Wigderson gibt hier eine Färbung mit deutlich mehr Farben an.
Abbildung 7.3 zeigt das Verhältnis von verwendeter Farbanzahl zur tatsächlichen chromatischen Zahl. Hinzugekommen sind hier sogenannte *Flat-Graphen*. Das sind Graphen, die

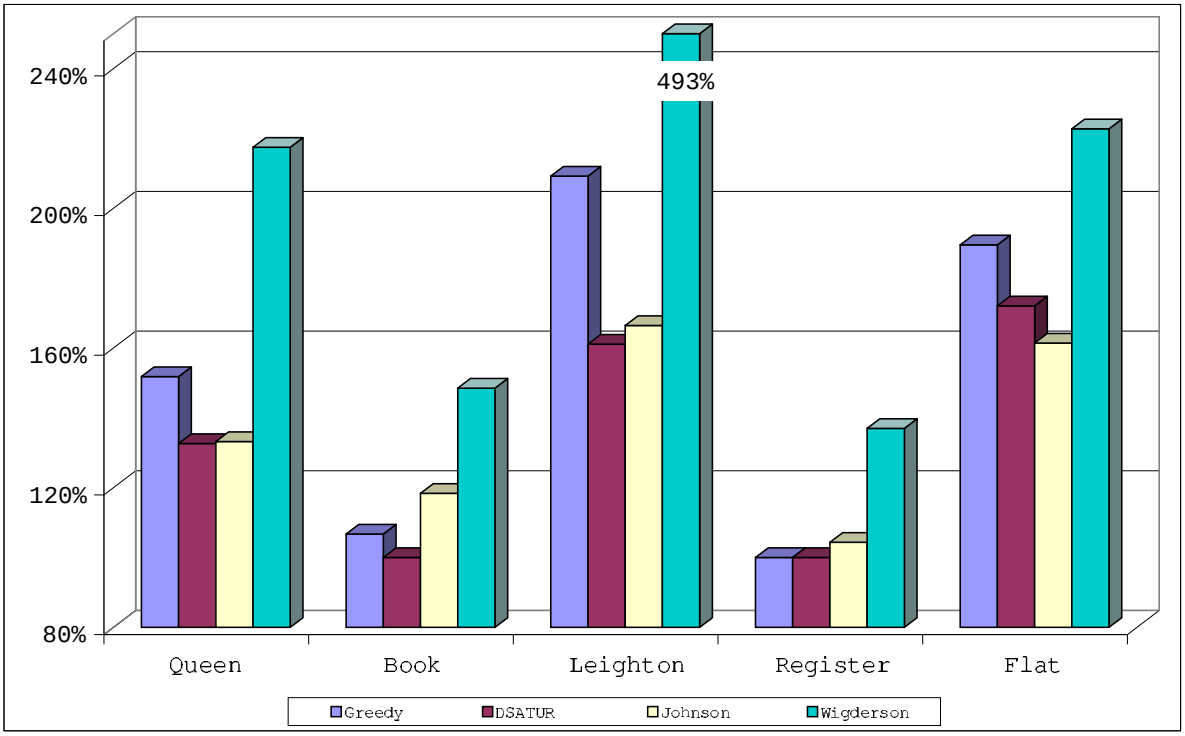

Abbildung 7.3: Güte der implementierten Algorithmen bei Benchmarkgraphen mit bekannter chromatischer Zahl

mit fester Knotenzahl und vorgegebener chromatischer Zahl nur eine geringe Varianz bei den Knotengraden zulassen. Dies führt zu einer erhöhten Schwierigkeit beim Färben des Graphen. Man erkennt, dass der Algorithmus von Wigderson deutlich hinter den anderen Algorithmen zurückbleibt. Das liegt daran, dass nach Satz 2.20 der Algorithmus von Wigderson maximal $2\chi(G) \cdot n^{1-\frac{1}{\chi(G)-1}}$ Farben verwendet. Für $\chi(G) = 4$ und $n = 100$ ergibt sich ein Maximum von 173 Farben. Das bedeutet, dass man genauso gut jedem Knoten v_i die Farbe i geben kann, ohne die angegebene Garantie zu verletzen. In Tabelle 7.4 sind für einige Werte von k die kleinsten Knotenzahlen n angegeben, für die der Algorithmus von Wigderson höchstens n Farben verwendet.

k	3	4	5	6	7	8	9	10
n_0	36	8^3	10^4	12^5	14^6	16^7	18^8	20^9
		512		$2.5 \cdot 10^5$	$7.5 \cdot 10^6$	$2.7 \cdot 10^8$	$1.1 \cdot 10^{10}$	$5.1 \cdot 10^{11}$
Johnson	11	114	$1.7 \cdot 10^3$	$3.6 \cdot 10^4$	$9.3 \cdot 10^5$	$2.9 \cdot 10^7$	$1 \cdot 10^9$	$4.4 \cdot 10^{10}$

Tabelle 7.4: Algorithmus 2.11 verwendet für k-färbbare Graphen höchstens $t = 2k \cdot n^{1-\frac{1}{k-1}}$ Farben. Für einige k sind hier die Werte n_0 angegeben, so dass $n \geq t \; \forall n \geq n_0$ gilt.

Man sieht, dass n_0 proportional zu $(2k)^{k-1}$ wächst. Bereits bei $k = 5$ besitzt der Graph bei der Repräsentation durch eine Adjazenzmatrix eine Größe von (mindestens) $10000 \cdot 10000 = 10^8$ Bits und ist aufgrund dieser Größe kaum noch zu handhaben.

Im Laufzeitvergleich sind die Algorithmen DSATUR, Wigderson und Greedy ähnlich. Einen größeren Unterschied gibt es zum Algorithmus von Johnson, der ab etwa 1000 Knoten bereits sehr langsam wird. In Tabelle 7.5 sind die Ergebnisse dazu dargestellt. Getestet wurden jeweils 100 Graphen mit 1000 bzw. 5000 Knoten und einer Kantenwahrscheinlichkeit $p = 0.5$.

Knoten	Greedy		DSATUR		Algorithmus	2.12
	Farben	Zeit (s)	Farben	Zeit (s)	Farben	Zeit (s)
1000	127	0.0635	115	0.4389	108	5.2363
5000	281	1.367	262	9.4391	233	367.859

Tabelle 7.5: Laufzeitvergleich der Algorithmen 2.1, 2.2 und 2.12 bei verschiedenen Graphgrößen.

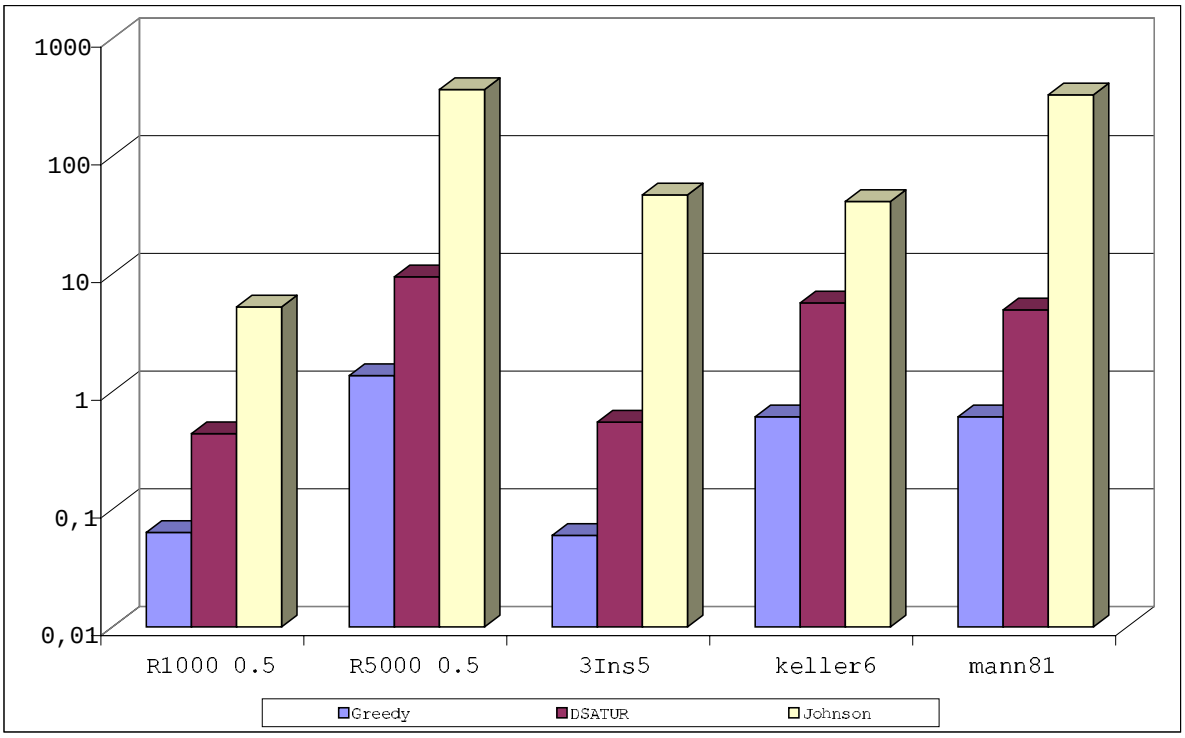

Abbildung 7.4: Laufzeiten verschiedener Algorithmen in Sekunden. Zu beachten ist, dass die Zeitachse logarithmisch skaliert ist.

Der Algorithmus von Wigderson (Algorithmus 2.12) benötigt die meiste Zeit, da hier der Algorithmus von Johnson (Algorithmus 2.9) mit eingearbeitet wurde. Im Vergleich zwischen

den Algorithmen 2.1 und 2.2 ist zwar die Greedy-Variante schneller, aber der DSATUR-Algorithmus liefert eine bessere Färbung. Der Algorithmus von Wigderson und Johnson liefert zwar die beste Färbung, jedoch steht der Mehraufwand in keinem Verhältnis zur erzielten Verbesserung.

Weitere Ergebnisse lassen sich in Abbildung 7.4 ablesen. Hier wurde unabhängig von der Güte des Ergebnisses nur die Laufzeit genommen, bis der Algorithmus eine Färbung ausgibt. Besondere Erwähnung sollte noch der Graph 4-FullIns_5.col finden. Der Graph besitzt 4146 Knoten und ist so konstruiert, dass er mit 9 Farben gefärbt werden kann. Der Greedy-Algorithmus verwendet 26 Farben in einer Zeit von 1.2 Sekunden. Nach 8.62 Sekunden bietet der DSATUR-Algorithmus eine 9-Färbung des Graphen an, und der Algorithmus von Johnson benötigt für seine Färbung 1038.53 Sekunden, also über 15 Minuten, um eine Färbung mit 11 Farben zu präsentieren. Der große Laufzeitunterschied ist vor allem in der Implementation des Algorithmus 2.9 von Johnson begründet. Für das Entfernen der jeweiligen Nachbarknoten wird immer der komplette Graph durchsucht. Bei einer geeigneteren Implementierung kann die Laufzeit vermutlich verbessert werden.

Für die Laufzeittests wurde ein AMD Athlon-Prozessor mit 600 MHz Taktfrequenz und 256 MB Arbeitsspeicher unter Windows 98 verwendet.

Kapitel 8

Zusammenfassung und Ausblick

Es wurde ausgehend vom Greedy-Algorithmus die Entwicklung verschiedener Färbungsalgorithmen bis hin zu neuesten Verfahren dargestellt. Der Algorithmus 2.7 ist der asymptotisch schnellste exakte Färbungsalgorithmus [12]. Vorgestellt wurde zudem der Färbungsalgorithmus von Halldórsson [19], der momentan der beste Approximationsalgorithmus für beliebige Graphen ist. Für k-färbbare Graphen konnte ausgehend von einem rekursiven Färbungsalgorithmus von Halperin et al. [20] ein neuer, allgemeiner Approximationsalgorithmus für k-färbbare Graphen hergeleitet werden. Die aktuell beste erreichbare Farbanzahl von n^{α_k} Farben, mit $\alpha_2 = 0, \alpha_3 = \frac{3}{14}$ und $\alpha_k = 1 - \frac{6}{k+4+\frac{3(1-2/k)}{1-\alpha_{k-2}}}$ für $k \geq 4$ konnte mit dem neuen Algorithmus 4.8 bestätigt werden. Des Weiteren konnte für den Fall, dass neue Algorithmen gefunden werden, die k-färbbare Graphen mit durchschnittlichem Grad $\bar{d}$ mit $O(\bar{d}^{\beta_k})$ Farben färben bzw. in k-färbbaren Graphen, die eine unabhängige Menge der Größe $\frac{n}{k-1}$ enthalten, eine unabhängige Menge der Größe $\Omega(n^{\gamma_k})$ finden, eine allgemeine Rekursionsgleichung für α_k bestimmt werden. Es gilt

$$\alpha_2 = 0,\ \alpha_3 = \frac{3}{14} \text{ und } \alpha_k = 1 - \frac{2}{2 + \frac{\beta_k}{1-\alpha_{k-2}} + \frac{\beta_k}{\gamma_k}} \text{ für } k \geq 4.$$

Weiterhin wurde das Verhalten von exakten Algorithmen auf speziellen Graphen betrachtet, sowie die Problematik von Online-Färbungen kurz umrissen. Es konnten auch einige obere und untere Schranken für Online-Algorithmen zur Färbung von Graphen angegeben werden. Die Schwierigkeit des Färbungsproblems konnte mit Hilfe eines Spiels mit exakten Färbungen näher beschrieben werden. Schließlich wurden noch einige Spezialfälle analysiert.

Einige der vorgestellten Algorithmen wurden implementiert und an verschiedenen Graphen getestet. Teilweise wurden die theoretischen Überlegungen verifiziert, allerdings wurden gera-

de bei kleineren Graphen Ergebnisse erzielt, die an der praktischen Anwendbarkeit der derzeit besten Algorithmen zweifeln lassen. Für die Tests wurden einige Graphinstanzen, die im Internet verfügbar sind [22], verwendet sowie verschiedene Graphen mit einem Zufallsgenerator konstruiert.

Boppana und Halldórsson [8] konnten zeigen, dass ihr Algorithmus zur Färbung beliebiger Graphen nicht mehr mit der von ihnen angewendeten Technik zu verbessern sind. Gleiches gilt für die Algorithmen, die mit semidefiniter Programmierung arbeiten [24, 20]. Das Ziel weiterer Arbeiten sollte es also sein, neue Techniken zur Färbung von Graphen zu entwickeln. Der allgemeine Färbungsalgorithmus für k-färbbare Graphen, Algorithmus 4.8, verwendet zwei Algorithmen, die ebenfalls semidefinite Programmierung benutzen, um ihr Ziel zu erreichen. Eine Loslösung von dieser Richtung könnte noch bessere Ergebnisse erzielen.

Auf der anderen Seite liegt die Optimierung der Laufzeit der vorgestellten Algorithmen. Klein und Lu [27] konnten für die Lösung semidefiniter Optimierungsprobleme, die von MaxCut und Coloring herrühren, einen effizienten Approximationsalgorithmus angeben. Somit bleibt noch die kombinatorische Analyse von Algorithmus 4.8. Vielleicht kann man den Satz von Blum [6] so modifizieren, dass man eine kleinere Anzahl von Knotenmengen $T \subseteq V$ untersuchen muss, um zum gewünschten Erfolg zu kommen.

Schließlich steht noch die generelle Schwierigkeit der Approximation zu Buche. Es bleibt die Frage offen, wie gut man unter der Prämisse $\mathtt{P}\neq\mathtt{NP}$ einen k-färbbaren Graphen in Polynomialzeit färben kann.

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

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

[2] N. Alon und N. Kahale. Approximating the independence number via the θ-function. *Mathematical Programming 80*, 253–264, 1998.

[3] S. Arora und S. Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1), 70–122, 1998.

[4] T. Baumann. *Färbungsalgorithmen für 3-färbbare Graphen.* Studienarbeit im Bereich Theoretische Informatik, Fakultät für Informatik, Technische Universität Chemnitz, 2003.

[5] R. Beigel und W. I. Gasarch. The mapmaker's dilemma. *Discrete Applied Mathematics*, 34(1-3), 37–48, 1991.

[6] A. Blum. New approximation algorithms for graph coloring. *J. ACM 41*, 470–516, 1994.

[7] A. Blum und D. Karger. An $\tilde{O}(n^{3/14})$-coloring algorithm for 3-colorable graphs. *Information Processing Letters 61*, 49–53, 1997.

[8] R. Boppana und M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. In *Lecture Notes in Computer Science No. 447*, 13–25. Springer-Verlag, 1990.

[9] M. Charikar. On semidefinite programming relaxations for graph coloring and vertex cover. *Proceedings of the 34th Annual ACM-SIAM Symposium on Discrete Algorithms*, 616–620, 2002.

[10] O. Coudert. Exact coloring of real-life graphs is easy. In *Design Automation Conference*, 121–126, 1997.

[11] I. Dinur, O. Regev und C. D. Smyth. The hardness of 3 - uniform hypergraph coloring. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, 33–42. IEEE Computer Society, 2002.

[12] D. Eppstein. Small maximal independent sets and faster exact graph coloring. *Lecture Notes in Computer Science* 2125, 462–470, 2000.

[13] D. Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proceedings of the 12th Symposium on Discrete Algorithms*, 329–337, 2001.

[14] P. Erdős und G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 463–470, 1935.

[15] U. Feige, M. Langberg und G. Schechtman. Graphs with tiny vector chromatic numbers and huge chromatic numbers. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, 283–292. IEEE Computer Society, 2002.

[16] Center for Discrete Mathematics and Theoretical Computer Science. *Clique and coloring problems graph format*, 1993. Verfügbar unter http://mat.gsia.cmu.edu/COLOR/general/ccformat.ps (03.06.2004)

[17] M. R. Garey und D. S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM 23*, 43–49, 1976.

[18] M. Grötschel, L. Lovász und A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica* 1, 169–197, 1981.

[19] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters 45*, pages 19–23, 1993.

[20] E. Halperin, R. Nathaniel und U. Zwick. Coloring k-colorable graphs using smaller palettes. In *12th Symposium on Discrete Algorithms*, 319–326, 2001.

[21] F. Herrmann und A. Hertz. Finding the chromatic number by means of critical graphs. *J. Exp. Algorithmics* 7 (10 Seiten), 2002.

[22] D. Johnson, A. Mehrotra und M. Trick. *Graph coloring instances*, 2002. Verfügbar unter http://mat.gsia.cmu.edu/COLORING03/ (15.06.2004)

[23] D. S. Johnson. Worst case behaviour or graph coloring algorithms. *Proc. 5th South-Eastern Conference on Combinatorics, Graph Theory and Computing*, 513–528, 1974.

[24] D. R. Karger, R. Motwani und M. Sudan. Approximate graph coloring by semidefinite programming. In *IEEE Symposium on Foundations of Computer Science*, 2–13, 1994.

[25] S. Khanna, N. Linial und S. Safra. On the hardness of approximating the chromatic number. *Combinatorica 20*, 393–415, 2000.

[26] H. A. Kierstead. Coloring graphs on-line. In *Online Algorithms - The State of the Art*, G. J. Woeginger und A. Fiat, 281–305, Springer-Verlag, 1998.

[27] P. Klein und H.-I Lu. Efficient approximation algorithms for semidefinite programs arising from max cut and coloring. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 338–347. ACM Press, 1996.

[28] G. Kortsarz und R. Krauthgamer. On approximating the achromatic number. In *Symposium on Discrete Algorithms*, 309–318, 2001.

[29] L. Kučera. *Combinatorial Algorithms*. Adam Hilger, 1990.

[30] E. L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters 5*, 66–67, 1976.

[31] L. Lovász. On the shannon capacity of graphs. *IEEE Trans. Inform. Theory 25*, 1–7, 1979.

[32] B. Monien und E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica 22*, 115–123, 1985.

[33] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[34] J. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 63–82, 1988.

[35] E. W. Weisstein. Line graph. Bei *MathWorld–A Wolfram Web Resource*. Verfügbar unter http://mathworld.wolfram.com/LineGraph.html (24.06.2004)

[36] A. Wigderson. Improving the performance guarantee for approximate graph coloring. *Journal of the ACM* 30, 729–735, 1983.

[37] X. Xie, T. Ono und T. Hirata. Simple combination of algorithms for 4-colorable graphs. Preprint, 6 Seiten, 2002.

[38] V. Yegnanarayanan. The pseudoachromatic number of a graph. *Southeast Asian Bulletin of Mathematics* 24, 129–136, 2000.

Ich, Tobias Baumann, erklre hiermit, dass ich die vorliegende Arbeit selbstndig und nur unter Verwendung der angegebenen Quellen angefertigt habe. Smtliche wissentlich aus anderen Schriften bernommene Passagen sind deutlich mit Herkunftsverweis gekennzeichnet.
Chemnitz, den

Tobias Baumann