Seminar Graphenfärbung
Greedy-Färbung und Verwandtes

Stefan Heinz

4. Juni 2004

Zusammenfassung

Das Problem der Berechnung der chromatischen Zahl $\chi(G)$ ist $\mathcal{NP}$-schwer, so dass es vermutlich keinen Algorithmus gibt, der in polynomieller Zeit diese Zahl berechnen kann (falls $\mathcal{P} \neq \mathcal{NP}$). Aus diesem Grund gewinnen Schranken für $\chi(G)$ an Bedeutung. Falls ein Algorithmus A eine zulässige Färbung auf einem Eingabegraph konstruiert, so ist die Anzahl der verwendeten Farben eine obere Schranke für $\chi(G)$.

Diese Ausarbeitung stellt in erster Linie einen GREEDY Algoritmus und die SMALLEST-LAST Heuristik für das Färben von Graphen vor. Dabei werden obere Schranken sowie die Möglichkeit der Bestimmung der Färbungszahl eines Graphen mit Hilfe dieser beiden Heuristiken, diskutiert.

Weiterhin gibt es noch zwei kleine Abschnitte zum Thema Iterated Greedy und zum Thema Färben mit Hilfe von Ähnlichkeits-Matrizen. In diesen beiden Abschnitten werden kurz die Ideen dieser Heuristiken vorgestellt und an einem einfachen Beispiel angewandt.

1 Einführung

In diesem Abschnitt werden die benötigten Notationen und Definitionen vorgestellt, sowie ein Beispiel aus der Praxis gegeben, wo man Graphenfärbung finden kann.

1.1 Notationen

Hier sollten alle für diese Ausarbeitung nötigen Notationen erklärt sein.

G	bezeichnet einen endlichen, einfachen, ungerichteten Graphen ($G = (V, E)$)
V	Knotenmenge eines Graphen G
E	Kantenmenge eines Graphen G ($E \subseteq V \times V$)
n	Anzahl der Knoten eines Graphen ($n = \|V\|$)
$\Delta(G)$	maximaler Grad eines Knoten von G
$\delta(G)$	minimaler Grad eines Knoten von G
$\chi(G)$	chromatische Zahl von G
$d(v)$	Grad des Knoten v
$\Gamma(v)$	Menge der Nachbarknoten vom Knoten v
$G[W]$	der von der Knotenmenge $W \subseteq V$ induzierte Subgraph von G

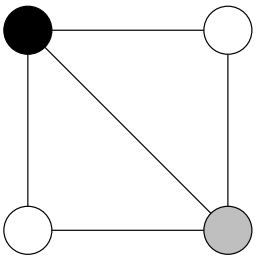
Alle Graphen, die im Folgenden auftauchen, sind immer endlich, einfach und ungerichtet.

1.2 Definitionen

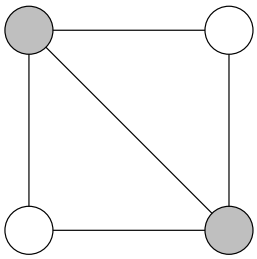
Gegeben sei ein Graph $G = (V, E)$, wobei V die Menge der Knoten und E die Menge der Kanten darstellen. Eine *k-Färbung* von einem Graphen G ist eine Abbildung $c : V \to \{1, \ldots, k\}$, so dass $c(v) \neq c(u)$ gilt, falls $uv \in E$ (bzw. $vu \in E$) ist, d.h. dass zwei benachbarte Knoten nicht die gleiche Farbe tragen dürfen. Somit ist jeder Graph $|V|$-färbbar und nur der leere Graph ($E = \emptyset$) ist 1-färbbar. Die *chromatische Zahl* $\chi(G)$ ist das kleinste k, so dass G eine k-Färbung besitzt. Eine *Farbklasse* in einer

Färbung von G ist die Menge von Knoten, welche dieselbe Farbe tragen. Dabei stellt jede Farbklasse eine stabile Menge des Graphen G dar.

Beispiel 1.1.



zulässige Färbung



unzulässige Färbung

1.3 Praxisbeispiel

In der Praxis spielt das Problem der Graphenfärbung eine große Rolle, sogar in unserer unmittelbaren Umgebung findet man es wieder.

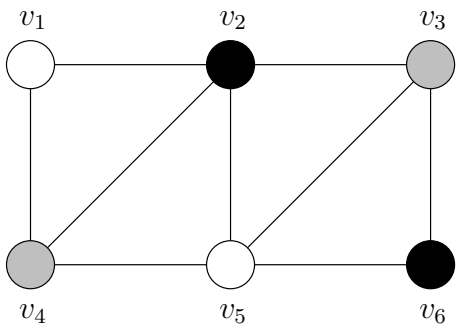
In Berlin werden jedes Jahr im Februar die Abiturklausuren geschrieben. Dafür steht den Schulen ein Zeitraum von 14 Tagen zur Verfügung. Innerhalb dieser 14 Tage muss eine bestimmte Anzahl von Abiturklausuren platziert werden, wobei jede Klausur (natürlich) nur einmal stattfinden darf. Weiterhin ist darauf zu achten, dass sich die Klausuren eines Schülers (in der Regel zwei Leistungskursklausuren und eine Grundkursklausur) nicht überschneiden.

Dieses Problem läßt sich in ein Graphenfärbungsproblem umwandeln. Im folgenden Beispiel haben wir dies exemplarisch dargestellt.

Beispiel 1.2. Eine Schule soll $T = \{t_1, \ldots, t_6\}$ Abiturklausuren innerhalb von drei Tagen durchführen. In der folgenden Tabelle ist aufgelistet, welcher Schüler welche Klausuren schreibt.

Schüler	**Klausur 1**	**Klausur 2**	**Klausur 3**
s_1	t_1	t_2	t_4
s_2	t_3	t_5	t_6
s_3	t_2	t_4	t_5
s_4	t_2	t_3	t_5

Man konstruiert nun einen Graphen $G = (V, E)$. Für jede Klausur t_i erzeugt man einen Knoten v_i in G. Dann fügt man für jeden Schüler s_i drei Kanten ein, und zwar für jedes Paar von Klausuren, die der Schüler schreibt, die Kante zwischen den zugehörigen Knoten in G.
Dies führt zu folgendem Graph G mit einer zulässigen Färbung:



Aus dieser Färbung läßt sich nun eine Lösung für das Ausgangsproblem konstruieren. Jede Farbklasse dieser Färbung repräsentiert Klausuren, die am selben Tag stattfinden können. Damit ordnet man jeder Farbklasse einen Tag zu und erhält für unser Beispiel eine mögliche Lösung:

Termin	Klausuren
Tag 1	t_1, t_5
Tag 2	t_2, t_6
Tag 3	t_3, t_4

2 Greedy Algorithmus

Im Folgenden untersuchen wir eine Greedy-Heuristik für das Graphenfärbungsproblem. Seien dazu $G = (V, E)$ ein Graph und $\sigma : \{1, \ldots, n\} \to V$ eine beliebige Anordnung der Knoten des Graphen G gegeben.

Um den Algorithmus und zukünftige Formeln übersichtlicher zu halten, führen wir folgende Notationen ein:

- $v_i := \sigma(i), i = 1, \ldots, n$, und schreiben $\sigma = (v_1, \ldots, v_n)$
- für $i = 1, \ldots, n$ setze $V_i := \{v_1, \ldots, v_i\}$
- Subgraph $G_i := G[V_i]$

2.1 Algorithmus

Der GREEDY Algorithmus geht die Knoten des Graphen in der Reihenfolge σ durch und färbt jeden Knoten mit der niedrigsten zulässigen Zahl.

Input : Graph $G = (V, E)$ und eine Anordnung $\sigma = (v_1, \ldots, v_n)$ der Knoten von G.
Output : Anzahl der benötigten Farben. (bzw. Färbung von G)

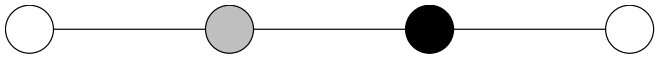
$c(v_1) \leftarrow 1$;
for $i \leftarrow 2$ ***to*** n **do**
  $c(v_i) \leftarrow \min\{k \in \mathbb{N} \mid k \neq c(u) \forall u \in \Gamma(v_i) \cap V_{i-1}\}$;
return $|\{c(v) : v \in V\}|$(bzw. c);

Algorithmus 1: GREEDY

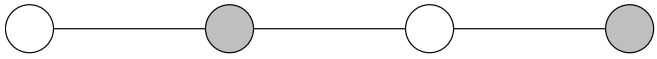
Aus der Definition einer zulässigen Färbung folgt, dass der Algorithmus GREEDY eine zulässige Knotenfärbung c konstruiert.

Beispiel 2.1. Betrachten wir folgenden Graphen G:

Nun wenden wir GREEDY auf die Anordnungen $\sigma_1 = (v_1, v_2, v_4, v_3)$ und $\sigma_2 = (v_1, v_2, v_3, v_4)$ an. Dies führt zu folgenden Färbungen.



Färbung für σ_1



Färbung für σ_2

Wie man sieht, hängt die Anzahl der verwendeten Farben von der Anordnung σ ab.

2.2 Obere Schranke durch GREEDY

Wie bereits erwähnt, liefern Algorithmen, die zulässige Färbungen konstruieren, obere Schranken für die chromatische Zahl, so auch der Algorithmus GREEDY.

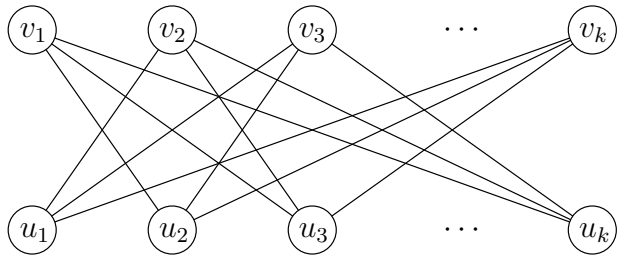
Proposition 2.2. *Für jede Anordnung σ der Knoten eines Graphen G gilt*

$$\chi(G) \overset{(1)}{\leq} \text{GREEDY}(G, \sigma) \overset{(2)}{\leq} \Delta(G) + 1.$$

Beweis. Ungleichung (1) ist trivial. Für Ungleichung (2) gilt, dass in jedem Färbungsschritt höchstens $|\Gamma(v_i) \cap V_{i-1}| \leq d(v_i) \leq \Delta(G)$ viele Farben für den Knoten v_i verboten sein können. Also kommt GREEDY unabhängig von σ stets mit den Farben $\{1, \ldots, \Delta(G) + 1\}$ aus. □

Für ungerade Kreise und vollständige Graphen gilt $\chi(G) = \Delta(G)+1$, somit liefert der Algorithmus GREEDY in diesen Fällen für jede Anordnung σ die chromatische Zahl. Jedoch gibt es Graphen, beidenen dieser Algorithmus auf einer Anordnung σ beliebig schlecht wird, wie folgendes Beispiel zeigt.

Beispiel 2.3. Wir betrachten den folgenden bipartiten Graphen $G_k = (U, V, E)$ mit $|U| = |V| = k$, $U = \{u_1, \ldots u_k\}$, $V = \{v_1, \ldots, v_k\}$ und $E = (U \times V) \backslash \{\{u_i, v_i\} : i = 1, \ldots, k\}$:



G_k ist 2-färbbar, der Algorithmus GREEDY benötigt aber bei jeder Anordnung, bei der auf u_i stets v_i folgt für $i = 1, \ldots, k$,

$$k = \Delta(G_k) + 1$$

viele Farben. Dies zeigt, dass GREEDY i.a. beliebig schlechte Färbungen liefert.

2.3 Bestimmung der chromatischen Zahl mit GREEDY

Das folgende Lemma liefert eine Möglichkeit, die chromatische Zahl mit Hilfe des Algorithmus GREEDY zu bestimmen. (Natürlich nicht in polynomieller Zeit.)

Lemma 2.4. *Für jeden Graphen G gibt es eine Anordnung σ^* seiner Knoten, auf der* GREEDY *eine optimale Färbung konstruiert, d.h. es gilt*

$$\chi(G) = \min_{\sigma \in S_n} \text{GREEDY}(G, \sigma)$$

Beweis. Seien $S_1, \ldots, S_{\chi(G)}$ die Farbklassen von G in einer $\chi(G)$-Färbung. $\sigma^* = (v_1, \ldots, v_n)$ enthalte zunächst die Knoten von S_1 in beliebiger Reihenfolge, dann die von S_2 u.s.w., d.h. $\sigma^* = (S_1, \ldots, S_{\chi(G)})$. Es folgt $\text{GREEDY}(G, \sigma^*) = \chi(G)$. □

Das Problem ist, dass man eine Anordnung σ^* i.a. im Voraus nicht kennt. Man könnte alle möglichen Permutation durchlaufen und so $\chi(G)$ bestimmen. Dies ist aber kein polynomieller Algorithmus, da es $n!$ Permutationen gibt und nach der Stirlingschen Formel

$$n! \sim \sqrt{s\pi n}\left(\frac{n}{e}\right)^n$$

gilt.

3 Smallest-Last Heuristik

Wir haben gesehen, dass die Anordnug $\sigma = (v_1, \ldots, v_n)$ der Knoten eines Graphen einen Einfluß auf die Färbungszahl hat, die der Algorithmus GREEDY liefert. In diesem Abschnitt betrachten wir die SMALLEST-LAST Heuristik, bei der man die Knoten eines Graphen erst sortiert, bevor man den Algorithmus GREEDY anwendet.

3.1 Smallest-Last Anordnung

Für den i-ten Knoten v_i sind im Algorithmus GREEDY höchstens $d_{G_i}(v_i)$ viele Farben verboten. Somit kommt GREEDY stets mit $\max_{1\leq i\leq n} d_{G_i}(v_i) + 1$ vielen Farben aus. Wir schreiben im Folgenden abkürzend

$$b(\sigma) := \max_{1\leq i\leq n} d_{G_i}(v_i),$$

falls $\sigma = (v_1, \ldots, v_n)$. Durch Minimierung über alle $n!$ Anordnungen $\sigma \in S_n$ der Knotenmenge erhält man damit folgende obere Schranke für $\chi(G)$

$$\chi(G) \leq \min_{\sigma\in S_n} b(\sigma) + 1, \tag{1}$$

welche abhängig von σ ist.

Eine Anordnung der Knoten, bei der dieses Minimum angenommen wird, lässt sich mit dem Algorithmus 2 konstruieren.

Input : Graph $G = (V, E)$
Output : Anordnung $\sigma_{SL} = (v_1, \ldots, v_n)$

for $i \leftarrow n$ ***downto*** 1 **do**
  wähle einen Knoten minimalen Grades in G als v_i der Anordnung σ_{SL};
  $G := G - v_i$;
return σ_{SL};

Algorithmus 2: SL-ANORDNUNG

Jede Permutation der Knotenmenge eines Graphen, die man so erhalten kann, heißt *smallest-last Anordnung*. Es gilt folgende interessante Maximum-Minimum-Beziehung.

Proposition 3.1. *Für jede smallest-last Anordnung σ_{SL} der Knotenmenge eines Graphen gilt:*

$$b(\sigma_{SL}) = \max_{H} \delta(H) = \min_{\sigma\in S_n} b(\sigma),$$

wobei das Maximum über alle Subgraphen H von G genommen wird.

Beweis. Für jede smallest-last Anordnung $\sigma_{SL} = (v_1, \ldots, v_n)$ der Knotenmenge eines Graphen gilt

$$b(\sigma_{SL}) \leq \max_{1\leq i\leq n} d_{G_i}(v_i) \leq \max_{1\leq i\leq n} \delta(G_i) \leq \max_{H} \delta(H),$$

da aus dem Algorithmus SL-ANORDNUNG folgt, dass $d_{G_i}(v_i) = \delta(G_i)$ für alle i.

Sei andererseits H^* ein Subgraph von G mit $\delta(H^*) = \max_H \delta(H)$. Dann gilt für jede Permutation σ der Knotenmenge von G, wenn $j = j(\sigma)$ den kleinsten Index bezeichnet, so dass H^* Subgraph von G_j ist:

$$\max_{H} \delta(H) = \delta(H^*) \leq d_{H^*}(v_j) \leq d_{G_j}(v_j) \leq b(\sigma).$$

Also gilt $\max_H \delta(H) \leq \min_{\sigma\in S_n} b(\sigma)$. Die Ungleichung $\min_{\sigma\in S_n} b(\sigma) \leq b(\sigma_{SL})$ schließlich ist trivial. □

3.2 Algorithmus und obere Schranke

Die Färbungsheuristik SMALLEST-LAST ordnet die Knoten eines Graphen G zunächst mit Hilfe des Algorithmus SL-ANORDNUNG in eine smallest-last Anordnung und wendet auf diese den Algorithmus GREEDY an. Aus der Proposition 3.1 erhalten wir die folgende Schranke für $\chi(G)$.

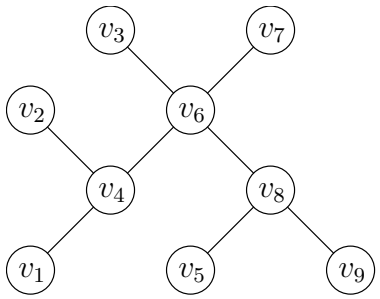
Korollar 3.2. *Sei G ein Graph und σ_{SL} eine smallest-last Anordnung seiner Knoten. Dann gilt*

$$\chi(G) \leq \text{GREEDY}(G, \sigma_{SL}) \leq \max_{H} \delta(H) + 1,$$

wobei das Maximum über alle Subgraphen H von G genommen wird.

Beweis. Folgt aus der Proposition 3.1 und der Ungleichung (1) von Seite 5. □

Beispiel 3.3. Gegeben sei folgender Graph G:



Als erstes erzeugen wir eine smallest-last Anordnung mit dem Algorithmus SL-ANORDNUNG. Der Algorithmus liefert die Anordnung

$$\sigma_{SL} = (v_9, v_8, v_6, v_7, v_5, v_4, v_3, v_2, v_1),$$

falls man im Falle einer nicht eindeutigen Auswahl den tie breaker kleinster Index benutzt.
Nun wenden wir GREEDY auf diese Anordnung σ_{SL} an und erhalten eine 2-Färbung.

$$S_1 = \{v_1, v_2, v_5, v_6, v_9\}, S_2 = \{v_3, v_4, v_7, v_8\}$$

GREEDY liefert eine optimale Färbung ($\chi(G) = 2$). Dies ist kein Zufall wie der folgende Satz zeigt.

Satz 3.4. *Die* SMALLEST-LAST *Heuristik färbt alle Wälder optimal.*

Beweis. Sei $G = (V, E)$ ein Wald. Falls $V = \emptyset$ ist nichts zu zeigen. Sei nun $V \neq \emptyset$.
1. Fall $E \neq \emptyset$
Für alle $W \subseteq V$ gilt, dass der Subgraph $G[W]$ wieder ein Wald ist. Somit folgt,

$$\max_{H} \delta(H) \leq 1, \tag{2}$$

wobei das Maximum über alle Subgraphen H von G genommen wird. Die Ungleichung 2 wird zur Gleichung,da ein Subgraph existiert, der ein Baum ist ($E \neq \emptyset$). Aus dem Korollar 3.2 folgt nun, dass GREEDY auf jeder smallest-last Anordnung der Knoten von G die chromatische Zahl $\chi(G) = 2$ bestimmt.
2. Fall $E = \emptyset$
Somit ist G 1-färbbar und der GREEDY Algorithmus liefert für jede Anordnung die chromatische Zahl. □

Ebenso wie für den GREEDY Algorithmus lässt sich auch für den SMALLEST-LAST Algorithmus ein Beispiel angeben, welches zeigt, dass der SMALLEST-LAST Algorithmus beliebig schlechte Färbung liefert. In [4] findet man eine Konstruktion für solch einen Graphen.

4 Iterated Greedy

Die meisten Algorithmen machen nur einen einzigen Durchlauf, um eine Färbung zu produzieren. Der Algorithmus ITERATED GREEDY, der in diesem Abschnitt vorgestellt wird, wendet hingegen den GREEDY mehrmals an (z.B. $10 \cdot |V|$ mal) und versucht, in jeder Iteration die Färbung zu verbessern. Dabei ist die Idee, die gewonnenen Informationen aus vergangenen Durchläufen für den nächsten Durchlauf zu nutzen.

4.1 Algorithmus

Der ITERATED GREEDY bekommt wie GREEDY einen Graphen G und eine Anordnung σ der Knoten von G übergeben. Nun wird GREEDY iterativ angewandt, wobei zwischen jeder Iteration eine neue Anordnung σ in Abhängigkeit der aktuellen Färbung konstruiert wird. Die Behauptung ist, wenn σ eine Anordnung ist wo Knoten mit der gleichen Farbe benachbart sind, dann konstruiert der Algorithmus GREEDY im nächsten Durchlauf eine Färbung, die nicht schlechter ist als die vorherige.

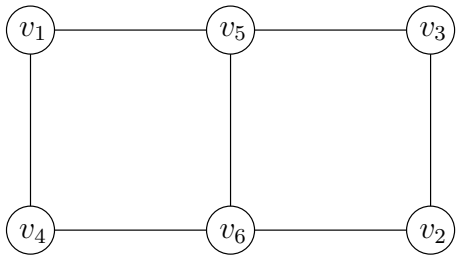
Lemma 4.1. *Sei $S_1, \ldots, S_k$ die Farbklassen einer k-Färbung des Graphen G, π eine Permutation dieser Farbklassen und $\sigma = (S_{\pi(1)}, S_{\pi(2)}, \ldots, S_{\pi(k)})$, dann gilt:*

$$\text{GREEDY}(G, \sigma) \leq k.$$

Beweis. Wenn man zeigt das $\forall i \in \{1, \ldots, k\} \forall v \in S_{\pi(i)} : c(v) \leq i$, folgt die Behauptung. Dies kann mit Hilfe der vollständigen Induktion gezeigt werden.
IA: Für $k = 1$ ist das klar.
IS: $(k-1) \rightarrow k$ Annahme $\exists v \in S_{\pi(k)} : c(v) = k + 1$
$\Rightarrow \exists v^* \in \Gamma(v) : c(v^*) = k$
Nach Induktionsvoraussetzung gilt, dass $S_{\pi(1)}, \ldots, S_{\pi(k-1)}$ mit $k-1$ Farben gefärbt werden.
$\Rightarrow v^* \in S_{\pi(k)}$
Dies ist aber ein Widerspruch zur Voraussetzung, dass $S_1, \ldots, S_k$ eine k-Färbung sind, denn dann kann v^* nicht gleichzeitig ein Nachbarknoten von v sein und sich in der selben Farbklasse wie v befinden.
$\Rightarrow$ Behauptung □

Es ist klar, dass die Anordnung der Knoten innerhalb einer Farbklasse keine Rolle für die nächste Färbung, die GREEDY konstruiert, spielt.

Beispiel 4.2. Betrachten wir folgenden Graphen G und die Anordnung $\sigma = (v_1, v_2, v_3, v_4, v_5, v_6)$.



Im ersten Durchlauf konstruiert GREEDY die Farbklasse

$$S_1 = \{v_1, v_2\}, S_2 = \{v_3, v_4\}, S_3 = \{v_5\}, S_4 = \{v_6\}.$$

Nun erzeugen wir uns eine neue Anordnung σ in Abhängigkeit der aktuellen Färbung. Dies tun wir, indem wir die Farbklassen in umgekehrter Reihenfolge anordnen und erhalten:

$$\sigma = (v_6, v_5, v_3, v_4, v_1, v_2).$$

Wendet man nun GREEDY erneut auf diese Anordnung an, erhalten wir eine 2-Färbung:

$$S_1 = \{v_1, v_3, v_6\}, S_2 = \{v_2, v_4, v_5\}$$

Der ITERATED GREEDY hat für dieses Beispiel eine optimale Fäbung produziert.

4.2 Umordnungen

Nun gibt es wieder viele Möglichkeiten, die Farbklassen nach einer Iteration umzuordnen. Im Folgenden stellen wir einige Ideen vor.

Umgekehrte Reihenfolge Bei dieser Umordnung dreht man einfach die Farbklassen um (siehe Beispiel 4.2).

Wachsende Mengen-Größe Mit Mengen-Größe ist die Anzahl der Elemente in einer Farbklasse gemeint, so dass Mengen mit wenigen Elemente nach vorn gepackt werden. Die Idee ist, dass Farbklassen mit wenigen Elementen auch (eventuell) weniger Nachbarknoten haben und somit die Wahrscheinlichkeit, dass andere Knoten noch dazu passen, groß ist.

Fallende Mengen-Größe Analog zu wachsenden Mengen-Größen, nur das hier die größten Mengen nach vorne gepackt werden. Diese Idee entsteht aus der Tatsache, dass man manchmal nach einer größten unabhängigen Menge sucht.

zufällige Umordnung Diese Möglichkeit soll davor schützen, nicht in Schleifen zu gelangen, wie es bei den anderen Methoden möglich ist.

Neben diesen Möglichkeiten gibt es natürlich noch viele andere. In der Praxis hat sich erwiesen, dass eine Mischstrategie die besten Resultate liefert.

Der ITERATED GREEDY ist natürlich nicht in der Lage die Färbungszahl eines beliebigen Graphen zu bestimmen. Schlimmer noch, er ist genauso schlecht wie ein einzelner Durchlauf des GREEDY Algorithmus, wie das folgende Beispiel zeigt.

Beispiel 4.3. Betrachten wir nun wieder den Graphen aus Beispiel 2.3. Für diesen Graphen konstruiert GREEDY im worst case eine k-Färbung mit Farbklassen

$$S_1 = \{u_1, v_1\}, S_2 = \{u_2, v_2\}, \ldots, S_k = \{u_k, v_k\}.$$

Wenn dies passiert, gibt es keine Umordnung der Farbklassen, die zu einer Verbesserung der Färbung im nächsten Durchlauf führt.

5 Ähnlichkeits-Matrizen

In diesem Abschnitt stellen wir nun noch die Idee, Graphen mit Hilfe von Ähnlichkeits-Matrizen zu färben, vor. Betrachten wir dazu zuerst folgendes Beispiel.

Beispiel 5.1. Der Graph G aus Beispiel 4.2 und die Anordnung $\sigma = (v_1, v_2, v_3, v_4, v_5, v_6)$ führen zu einer 4-Färbung, wenn man den GREEDY Algorithmus anwendet. Der Graph G ist jedoch 2-färbbar ($S_1 = \{v_1, v_3, v_6\}, S_2 = \{v_2, v_4, v_5\}$).

Wenn man sich das Beispiel genauer anschaut, sieht man, dass sobald GREEDY die Knoten v_1 und v_2 mit der gleichen Farbe gefärbt hat, er eine k-Färbung mit $k > 2$ liefert. Die Frage ist nun, wie man dies verhindern kann? Man möchte in irgendeiner Form ausdrücken, dass v_1 und v_2 nicht „so ähnlich“ sind, wobei ähnlich in Bezug zur Farbe zu sehen ist, die die Knoten bei einer Färbung bekommen.

5.1 Definitionen

Definition 5.2. Sei $G = (V, E)$ ein beliebiger Graph, dann ist die *Konflikt-Matrix* $C \in \{0,1\}^{|V|\times|V|}$ definiert als

$$c_{ij} = \begin{cases} 1 & \text{falls } v_i \text{ und } v_j \text{ benachbart} \\ 0 & \text{sonst} \end{cases}$$

Definition 5.3. Sei $G = (V, E)$ ein beliebiger Graph und C die zugehörige Konflikt-Matrix, dann ist die *Ähnlichkeits-Matrix* $S \in \mathbb{N}^{|V|\times|V|}$ definiert als

$$s_{ij} = \begin{cases} 0 & \text{falls } c_{ij} = 1 \\ 1 & \text{falls } i = j \\ 1 + \sum_k (c_{ik} \wedge c_{jk}) & \text{sonst} \end{cases}$$

wobei

$$c_{ik} \wedge c_{jk} = \begin{cases} 1 & \text{falls } c_{ik} = c_{jk} = 1 \\ 0 & \text{sonst} \end{cases}$$

5.2 Algorithmus

Nun folgt der Algorithmus ÄHNLICHKEITS-GREEDY. Dieser ist nur ideenhaft aufgeschrieben.

Input : Graph $G = (V, E)$ und zugehörige Ähnlichkeits-Matrix S
Output : gefärbter Graph G

$a \leftarrow$ größter Wert in S;
for $i \leftarrow a$ ***downto*** 1 **do**
  foreach *Knotenpaar* (v_i, v_j) *:* $s_{ij} = a$ **do**
    COLORPAIR(v_i, v_j);
  if G *gefärbt* **then**
    break;
return gefärbten Graph G;

Algorithmus 3: ÄHNLICHKEITS-GREEDY

Bleibt zu klären, was COLORPAIR mit den beiden Knoten macht. Dazu folgt nun eine kurze Algorithmusskizze, wobei vorher erwähnt werden sollte, dass ein Knoten v_i ignoriert werden kann, wenn gilt $d_G(v_i) < \#(\text{bereits benutzter Farbe})$. Denn in diesem Fall kann v_i immer einer bereits vorhanden Farbklasse zugewiesen werden.

COLORPAIR(v_i, v_j)

(a) v_i **und** v_j **sind gefärbt**

1. Gehe zum nächsten Paar

(b) **Einer der Knoten ist gefärbt (oBdA** v_i**)**

1. Falls $d_G(v_j) < \#(\text{benutzter Farbe})$, dann ignoriere v_j und gehe zum nächsten Paar
2. Falls v_j zur Farbklasse von v_i passt, dann färbe v_j wie v_i
3. Gehe zum nächsten Paar

(c) **Beide sind noch nicht gefärbt**

1. Falls $d_G(v_j) < \#(\text{benutzter Farbe})$ und $d_G(v_i) < \#(\text{benutzter Farbe})$, dann ignoriere v_j und v_i und gehe zum nächsten Paar
2. Finde Farbklasse, wo beide hinein passen und färbe sie mit dieser Farbe
3. Falls es keine solche Farbklasse gibt, dann benutze eine neue Farbe für v_i und v_j
4. Gehe zum nächsten Paar

Im Algorithmus ÄHNLICHKEITS-GREEDY wird jeder Knoten mindestes einmal angefasst, da $s_{ii} = 1$. Teilweise werden die Knoten mehrmals angefasst. Nun folgt noch ein Beispiel.

Beispiel 5.4. Betrachten wir nun den Graphen aus Beispiel 4.2 und färben ihn mit Hilfe von Ähnlichkeits-Matrizen. Dazu benötigen wir die Konflikt-Matrix C und die Ähnlichkeits-Matrix S.

Konflikt-Matrix C

C_{ij}	1	2	3	4	5	6
1	0	0	0	1	1	0
2	0	0	1	0	0	1
3	0	1	0	0	1	0
4	1	0	0	0	0	1
5	1	0	1	0	0	1
6	0	1	0	1	1	0

Ähnlichkeits-Matrix S

S_{ij}	1	2	3	4	5	6
1	1	1	2	0	0	3
2	1	1	0	2	3	0
3	2	0	1	1	0	3
4	0	2	1	1	3	0
5	0	3	0	3	1	0
6	3	0	3	0	0	1

Der Algorithmus ÄHNLICHKEITS-GREEDY konstruiert eine 2-Färbung

$$S_1\{v_1, v_3, v_6\}, S_2\{v_2, v_4, v_5\}.$$

Literatur

[1] Joseph C. Culberson. Iterated Greedy Graph Coloring and the Difficulty Landscape. techrep TR 92-07, ualtacs, Edmonton, Alberta, Canada T6G 2H1, 1992. ftp ftp.cs.ualberta.ca pub/TechReports.

[2] Joseph C. Culberson and Feng Luo. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26, chapter Exploring the k–colorable Landscape with Iterated Greedy, pages 245–284. American Mathematical Society, 1996. preprint available http://web.cs.ualberta.ca/˜joe/.

[3] Thomas Emden-Weinert, Stefan Hougardy, Bernd Kreuter, Hans Jürgen Prömel, and Angelika Steger. *Einführung in Graphen und Algorithmen*, chapter 5 Färbung, pages 128–152. 1996. http://www.informatik.hu-berlin.de/Institut/struktur/algorithmen/ga/.

[4] J. Mitchem. On Various Algorithms for Estimating the Chromatic Number of a Graph. *The Computer Journal*, Volume 19, Issue 2:182–183, 1976.

[5] D. C. Wood. A technique for colouring a graph applicable to large scale timetabling problems. *The Computer Journal*, Volume 12, Issue 4:317–319, 1969.