



h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK

HOCHSCHULE DARMSTADT

SEMINAR "WISSENSCHAFTLICHES ARBEITEN 1"
SOMMERSEMESTER 2016

Das Färben von Knoten in Graphen am Beispiel von Sudoku und dem Backtracking-Algorithmus

Marcel Bös

marcel.boes@stud.h-da.de

Dimitri Neumann

dimitri.neumann@moonpie.co

betreut durch

Prof. Dr. Inge Schestag

18. Juni 2016

Zusammenfassung

Die Graphentheorie, als Gebiet der Mathematik, beschäftigt sich mit Graphen, welche eine abstrakte Struktur aus Knoten und derer möglichen Verbindung bilden. Mithilfe der Graphentheorie und der algorithmischen Verarbeitung von Graphen können diverse Problemstellungen der Informatik gelöst oder optimale Lösungen approximiert werden.

Im Rahmen der Seminararbeit möchten wir anhand eines konkreten Beispiels einer Problemstellung aus dem Bereich der Informatik zeigen, wie sich dieses Problem algorithmisch mittels Knotenfärbung lösen lässt. Dabei möchten wir im Vorfeld darauf eingehen, wie ein Graph definiert ist, welche Typen von Graphen existieren und wie sich Graphen algorithmisch verarbeiten lassen.

Ebenso möchten wir kurz auf die Komplexität solcher Problemstellungen eingehen und aufzeigen, dass die Klasse dieser Probleme nicht in polynomieller Zeit lösbar sind, sofern andere Verfahren verwendet werden. Im Anschluss möchten wir einen Algorithmus und dessen Funktionsweise zur Knotenfärbung vorstellen, der das ausgewählte Problem löst.

Inhaltsverzeichnis

1. Grundlagen der Graphentheorie	1
1.1. Grundlegende Definitionen	1
1.2. Darstellung von Graphen als Adjazenzmatrix	2
2. Das Färbungsproblem	2
2.1. Definition und Problemstellung	3
2.2. Komplexität und Laufzeit	3
3. Algorithmische Verarbeitung	5
3.1. Unterschiedliche Verfahren	5
3.2. Backtracking-Algorithmus	5
4. Anwendungsbeispiel: Sudoku	5
4.1. Problemstellung	5
4.2. Backtracking-Algorithmus zur Lösung von Sudokus	7
4.3. Anwendung des Backtracking-Algorithmus	8
5. Weitere Anwendungsfälle der Graphenfärbung	9
Literatur	10
A. Pseudocode: Backtracking-Algorithmus	11

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Darmstadt, den 15. Juni 2016

(Marcel Bös)

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Darmstadt, den 15. Juni 2016

(Dimitri Neumann)

1. Grundlagen der Graphentheorie

Um das Färbungsproblem genauer zu betrachten werden einige wenige Definitionen und Erklärungen der Graphentheorie benötigt. Es wird eine Definition eines Graphen gegeben, sowie ein kurzes Beispiel, wie diese visuell dargestellt werden können.

1.1. Grundlegende Definitionen

Für die Betrachtung des Färbungsproblems genügt es meist, die Definition von ungerichteten, einfachen Graphen zu kennen.

Definition 1.1 (Graph).

Ein ungerichteter, einfacher Graph $G = (V, E)$ besteht aus einer nichtleeren Menge V von Objekten, welche Knoten genannt werden und einer Menge E von Kanten.

Die Menge der Knoten (engl. Vertices) wird auch als $V(G)$ geschrieben und die Menge der Kanten (engl. Edges) als $E(G)$. Für eine Kante $e = (u, v)$ werden die Knoten u und v als adjazent, oder benachbart bezeichnet.

Beispiel

Sei G ein Graph. Dieser enthält die Menge V der Knoten

$$V = \{v_1, v_2, v_3, v_4\},$$

die Menge E hat folgende Kanten, welche 2-Tupel, bestehend aus den Knoten aus V , sind.

$$E = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_3, v_4)\}$$

Graphen können visuell durch einfache Diagramme dargestellt werden. Die folgende Abbildung 1 zeigt das Diagramm zur obengenannten Menge V und E .

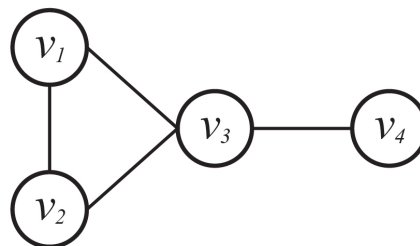


Abbildung 1: Diagramm eines Graphen

Des Weiteren gibt es in der Graphentheorie zahlreiche Unterscheidungen von Graphen und weitere Definitionen, z.B. gerichtete Graphen (auch Digraphen genannt), Teilgraphen und Multigraphen. Zur Betrachtung des Färbungsproblems verwenden wir in dieser Seminararbeit nur einfache, ungerichtete Graphen.

1.2. Darstellung von Graphen als Adjazenzmatrix

Um Graphen für die algorithmische Verarbeitung verwenden zu können, benötigt es einer Darstellungsweise, welche maschinenlesbar ist. Hierbei gibt es mehrere Möglichkeiten, von denen wir eine der gebräuchlichsten vorstellen möchten.

Adjazenzmatrix

Zur Anschauung verwenden wir den Graph aus Beispiel 1.1.

Die Abbildung 2 zeigt den obigen Graphen und die dazugehörigen Adjazenzmatrix. Dabei werden Verbindungen zwischen Knoten durch eine 1 in der Matrix und fehlende Kanten durch eine 0 gekennzeichnet.

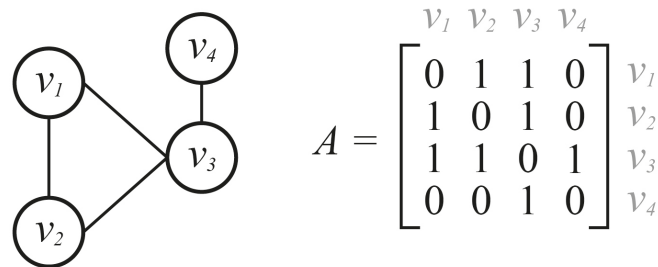


Abbildung 2: Diagramm eines Graphen

Matrizen als Darstellungsweise bieten sich insbesondere wegen der einfachen Anschauung an. Jedoch kann man Graphen auch durch eine Adjazenzliste darstellen. Die Wahl der Datenstruktur, hinsichtlich der Effizienz, hängt vom jeweiligen Algorithmus und dem zu lösenden Problem ab.

2. Das Färbungsproblem

Es existieren viele Problemstellungen aus unterschiedlichen Wissenschaftsbereichen, insbesondere der Informatik. Durch das Erfordernis, effiziente Berechnungsmethoden zu finden, ist besonders in der Informatik das Finden einer Lösung oder gar einer Approximation zu komplex und manchmal effektiv nicht berechenbar ist. Die Komplexität ergibt sich aus der großen Anzahl möglicher Lösungsschritte, welche zu einer Lösung führen können, sodass ein reines algorithmisches Ausprobieren nicht zum Erfolg führt.

Im letzten Kapitel wurde die Definition von einem Graph bereits vorgestellt. Ein Graph besteht aus einer Anzahl von n Knoten, die wiederum durch m Kanten verbunden sind.

Das Färbungsproblem beschäftigt sich damit, jedem dieser Knoten eine bestimmte Farbe zuzuordnen. Wichtig dabei ist, dass die Knoten, die durch eine Kante verbunden sind, nicht die gleiche Farbe aufweisen, aber trotzdem so wenig wie möglich unterschiedliche

Farben für die Färbung des Graphen benutzen. Hierfür gibt es unterschiedliche Verfahren, auf die wir später genauer eingehen.

Repräsentiert man nun die Farben als natürliche Zahlen, so lässt sich dies definieren als $c(V) \in \{1, 2, \dots, k\}$, wobei k minimal ist. Um z.B. einem Knoten die Farbe 2, die als Grau hinterlegt ist, zuzuweisen, schreibt man dies in folgender Form auf: $c(V) = 2$.

2.1. Definition und Problemstellung

Die Eigenschaften, die für das Färben notwendig sind, lassen sich wie folgt beschreiben.

Definition 2.1. *"A colouring of a graph is called complete if all vertices $v \in V$ are assigned a colour $c(v) \in \{1, \dots, k\}$; else the colouring is considered partial."*

Definition 2.2. *"A clash describes a situation where a pair of adjacent vertices $u, v \in V$ are assigned the same colour (that is, $\{u, v\} \in E$ and $c(v) = c(u)$). If a colouring contains no clashes, then it is considered proper; else it is considered improper."*

Definition 2.3. *"A colouring is feasible if and only if it is both complete and proper."*

Definition 2.4. *"The chromatic number of a graph G , denoted by $\chi(G)$, is the minimum number of colours required in a feasible colouring of G . A feasible colouring of G using exactly $\chi(G)$ colours is considered optimal."*

[Lew16a]

Da jetzt das Färben eines Graphen definiert wurde, steht nun die Frage im Raum, wie man optimal die Farben einem Knoten zuordnet, sodass die verwendeten Farben minimal bleiben, also $\chi(G)$ minimal ist. Eine Lösung zu berechnen ist in vielen praktischen Fällen nicht durchführbar, sodass Algorithmen verwendet werden, welche eine optimale Lösung approximieren. Es ist nicht immer möglich zu bestimmen, ob ein Graph eine bestimmte n -Färbung besitzt. So war es lange Zeit z.B. nicht bewiesen, dass man für jede Landkarte eine 4-Färbung finden kann (siehe Vier-Farben-Problem [Tur13a]).

2.2. Komplexität und Laufzeit

Für jeden beliebigen, einfachen Graph mit n Knoten, existiert die triviale n -Färbung. Damit durch die Anwendung der Graphenfärbung eine Problemstellung optimiert werden kann, wird, wie bereits erwähnt, gefordert eine n -Färbung mit möglichst kleinem n zu finden.

Die Berechnung für das Finden solch einer optimalen Färbung kann unter Umständen sehr ineffizient werden. Die einfachste Möglichkeit, einen Graphen zu färben, ist, alle möglichen Farbkombinationen durch Zuweisung von möglichen Farben zu den Knoten, auszuprobieren und bei jeder sich ergebenden Färbung zu prüfen, ob diese gültig ist.

Bei dem soeben beschriebenen Verfahren durch Ausprobieren ergeben sich bei n Knoten maximal n^n Möglichkeiten, den Graphen zu färben. Man erkennt schnell, dass bei zunehmender Größe eines Graphen die Laufzeit enorm steigen würde. Bei einer Größenordnung von $n = 50$ Knoten ist es selbst mit den heutigen Rechnern nicht mehr möglich, auf ein Ergebnis zu kommen.

”To illustrate, a graph with $n = 50$ vertices would lead to over $50^{50} \approx 8.8 \times 10^{84}$ different assignments: a truly astronomical number. This would make the task of creating and checking all of these assignments, even for this modestly sized problem, far beyond the computing power of all of the world’s computers combined. (For comparison’s sake, the number of atoms in the known universe is thought to be around 1082.)”

Eine wesentlich effizientere Herangehensweise ist es, die Symmetrie eines ungerichteten Graphen zu benutzen. Zur Veranschaulichung der Symmetrie betrachte man die Adjazenzmatrix eines Graphen. Diese ist immer symmetrisch, wodurch einige Färbungen, durch Vertauschen der Farben (unter Beachtung der Symmetrie), erneut eine gültige Färbung ergibt. Hat man z.B einen Graphen mit den Farben $c(v_1) = 1, c(v_2) = 4, c(v_3) = 2, c(v_4) = 3, c(v_5) = 1$ ist dies in unserem Fall gleichzusetzen mit $c(v_1) = 4, c(v_2) = 1, c(v_3) = 1, c(v_4) = 3, c(v_5) = 2$ Somit ergibt sich folgende Gleichung unter Betrachtung der Symmetrie des Graphen: $k! = k \times (k - 1) \times (k - 2) \times \dots \times 2 \times 1$.

Zwar würde die Laufzeit durch das Ausnutzen der Symmetrie abnehmen, jedoch kaum effizienter werden. Denn bei gleicher Knotenanzahl $n = 50$ ist die Anzahl verschiedener Möglichkeiten mit $k! : 50! \approx 3,04 \times 10^{64}$ immer noch sehr hoch.

Auch unter Betrachtung von Untergraphen (Vgl. [Sve16]) eines gegebenen Graphen lässt sich die Laufzeit zwar verbessern, ist aber immer noch weit entfernt von einer akzeptablen Berechnungszeit. Eine akzeptable Berechnungszeit ist natürlich immer subjektiv zu betrachten, da es jedem selbst überlassen ist, wie optimal das Ergebnis sein soll, dennoch sprechen wir hier von einer Laufzeit von mehreren Tage, was in einem bestimmten realen Anwendungsfall nicht gewinnbringend sein muss.

Mit der Stirlingzahl $\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\}$ definiert man die Anzahl der Möglichkeiten, n Knoten in genau k nicht leere Teilmengen zu partitionieren. Dies lässt sich mit folgender Gleichung berechnen:

$$\left\{ \begin{smallmatrix} n \\ k \end{smallmatrix} \right\} = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{n}{j} j^n.$$

Somit lässt sich ein Graph der Knotenanzahl 3 in drei nicht leere Teilmengen zerlegen: $v_1, v_2, v_3, v_2, v_1, v_3$ und v_3, v_1, v_2 . Jetzt kann man einen Aufzählungsalgorithmus verwenden, der bei einem Startwert $k = 1$ anfängt. Die Variable k definiert hier die verwendete Farbe. In jedem Schritt überprüft der Algorithmus, ob für alle Teilmengen die Gültigkeit und die Abgeschlossenheit für den Graphen gilt ”Definition 2.3”. Falls eine Lösung gefunden wurde, kann der Algorithmus sofort aufhören nach weiteren Lösungen zu suchen, denn man hat bereits die beste gefunden.

Bestimmt man nun, wie oben erwähnt, den hohen Rechenaufwand für diesen Algorithmus, sieht man auch hier, dass die Laufzeit immer noch zu hoch ist. Bleiben wir bei dem Beispiel von 50 Knoten und versuchen diese in maximal 10 verschiedenen Farben darzustellen, erhalten wir mit der Formel $\left\{ \begin{smallmatrix} 50 \\ 10 \end{smallmatrix} \right\} \approx 2,6 \times 10^{43}$ wieder zu viele Kandidaten einer möglichen Färbung. (Vgl. [Lew16b])

3. Algorithmische Verarbeitung

Für das effiziente Färben von Graphen stehen unterschiedliche Algorithmen zur Verfügung, wobei sowohl die Laufzeit als auch die sinnvolle Anwendung für die entsprechende Problemstellung abgewogen werden muss. In den nächsten beiden Abschnitten stellen wir drei Algorithmen vor, wobei wir insbesondere den Backtracking-Algorithmus genauer betrachten möchten.

3.1. Unterschiedliche Verfahren

Einer der einfachsten Algorithmen, der zur Kolorierung von Graphen verwendet werden kann, ist der Greedy-Algorithmus. Dieser versucht zu jedem Ausführungsschritt den bestmöglichen, nächsten Rechenschritt (Auswahl aus einer Anzahl möglicherweise verschiedener Rechenschritte) zu treffen, der zu einer Teillösung führt. Die Teillösung wird so sukzessiv aufgebaut, bis schließlich eine vollständige Lösung gefunden wird.

Einen anderen Ansatz verfolgt der Algorithmus RLF (Recursive Largest First). Zunächst wird aus einer Menge von Knoten ein Untergraph erzeugt. Bei diesem werden alle Knoten in einer Farbe gefärbt. Im folgenden Schritt wird im bereits gefärbten Untergraph ein weiterer Untergraph gesucht, welcher dann in einer zweiten Farbe komplett gefärbt wird. Dieses Vorgehen führt dann zu einem kolorierten Graph.

3.2. Backtracking-Algorithmus

Auf einem ähnlichen Prinzip, wie das des Greedy-Algorithmus, basiert auch das Backtrackingverfahren. Dabei wird versucht, eine Teillösung zu konstruieren, von der aus eine Gesamtlösung oder optimale Approximation generiert werden kann. Gelangt jedoch die Ausführung des Algorithmus an einen Schritt, der zu einer unmöglichen Lösung führt, wird schrittweise zurückgegangen und eine alternative Teillösung konstruiert.

Zur Lösung des Sudoku-Problems wird im Allgemeinen das Backtrackingverfahren angewandt, welches wir im folgenden Abschnitt als einfachen Pseudocode für die Lösung von Sudoku-Rätseln vorstellen werden. Im Anhang findet sich ein weiteres Listing einer allgemeinen Variante des Backtracking-Algorithmus, welche durch eine Rekursion allgemein Graphen färben kann.

4. Anwendungsbeispiel: Sudoku

Inwiefern gehört das Spiel Sudoku und die Graphentheorie zusammen? Genau das möchten wir gerne in diesem Kapitel zur Darstellung bringen, denn ein Sudokufeld lässt sich mittels Färben von Graphen bestimmen.

4.1. Problemstellung

Zunächst möchten wir auf die Definitionen in Kapitel 2.1 verweisen, die sich mit dem Färben auseinander setzen und kurz auf die Spielregeln von Sudoku eingehen.

Bei der Betrachtung des Sudokuproblems beschränken wir uns auf ein reguläres 9×9 -Sudoku mit 81 Feldern. Dabei hat das Spielfeld eine Länge und eine Breite von jeweils 9 Feldern. Zusätzlich wird das Sudoku-Spielfeld in Blöcke von einer Länge und Breite von jeweils 3 Feldern unterteilt, also zu insgesamt 9 gleichgroßen Blöcken.

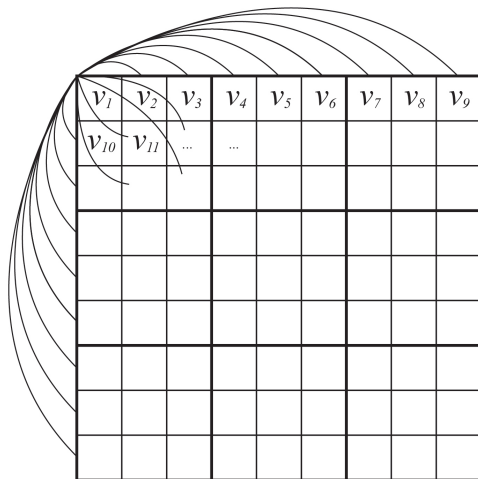
Ein Sudoku-Rätsel gilt als gelöst, wenn jedes Spielfeld eine Zahl von 1 bis 9 enthält und die Bedingung erfüllt ist, dass in der gleichen Zeile, Spalte oder dem Block eine Zahl nicht mehr als einmal auftritt.

Das Lösen des Sudokurätsels durch Einsetzen beliebiger Zahlen und Validieren des Rätsels (so wie auf das Verfahren in Kapitel 2 Bezug genommen wurde), würde maximal zu 5.525×10^{27} möglichen Kombinationen führen. Obwohl die Anzahl valider Sudoku-Lösungen bei 6.671×10^{21} Kombinationen liegt und sich die Anzahl der Kombinationen durch bereits vorausgefüllte Felder reduzieren würde, wäre die Laufzeit immer noch, selbst bei einem einfachen Algorithmus mit einer Zeitkomplexität von $O(n^4)$, nicht mit heutigen Rechnern effektiv zu bewältigen.

[Fra05]

Unter Zuhilfenahme der Graphentheorie und des Backtracking-Algorithmus lassen sich Sudokus dennoch in adäquater Zeit lösen. Hierbei muss jede Feldeinheit des Spielfelds als Knoten eines Graphen betrachtet werden. Nun verbindet man Knoten durch Kanten, bei denen Konflikte, im Rahmen der oben genannten Spielregeln, auftreten können.

Der Färbungsprozess besteht hier in der Zuweisung der Zahlen von 1 bis 9 in die jeweiligen Felder, die jeweils repräsentativ für 9 unterschiedliche Farben stehen. Also $c(v_i) = k$ für $i \in \{1, 2, \dots, 81\}$ und $k \in \{1, 2, \dots, 9\}$.



Die Felder des Sudokus werden von v_1 bis v_{81} durchnummeriert, wobei ein Feld einem Knotenpunkt des Graphen entspricht. Dabei zeigt das Sudoku auf der linken Seite beispielhaft für das Feld v_1 , wie die Verbindungen zu den anderen Feldern wäre.

4.2. Backtracking-Algorithmus zur Lösung von Sudokus

Das weiter unten stehende Listing zeigt eine sehr stark vereinfachte Variante des Backtracking-Algorithmus zur Lösung von Sudoku-Rätseln. Die Implementierung, bzw. der Pseudocode der einzelnen Funktionen im folgenden Listing ist für das Verständnis nicht relevant und es wird nur eine kurze Beschreibung der Funktionen und Variablen gegeben. Eine ausführliche Pseudocode-Variante mit Rekursion findet sich im Anhang dieser Seminararbeit. Dabei befindet sich auf der nächsten Seite eine einfache Veranschaulichung, wie der Backtracking-Algorithmus vorgeht.

- Variable 'vertex':
Diese Variable enthält den Verweis auf eines der 81 Felder.
- Variable 'color':
Eine Zahl von 1 bis 10 wird in der Variable 'color' gespeichert.
- getNextVertex():
Gibt die Knoten 1 bis 81 bei aufruf zurück. Der erste Aufruf gibt also den ersten Knoten, also den Verweis auf das erste Feld zurück.
- getPrevVertex():
Gibt den vorherigen Knoten zurück.
- checkNeighbour():
Diese Funktion prüft, ob für die als Parameter übergebene Farbe 'color', mindestens ein benachbarter Knoten von 'vertex' die gleiche Farbe besitzt.
- deleteColorInField():
Löscht die Farbe im Feld 'vertex'.
- writeColorInField():
Schreibt die Farbe 'color'.

```
1 var color = 0;
2 var vertex = getNextVertex();
3
4 while (vertex != NULL)
5     color = color + 1;
6
7     if (checkNeighbour(vertex, color) == true)
8         if (color == 10)
9             deleteColorInField(vertex);
10            vertex = getPrevVertex();
11            color = 0
12     else
13         writeColorInField(color);
14         vertex = getNextVertex();
15         color = 0;
```

4.3. Anwendung des Backtracking-Algorithmus

		2			9	7		
8	4	1	3	6				
					8			
4		9					2	
	6						4	
	2					9		3
			2					
				9	1	3	7	8
		7	5			1		

Das linksstehende Ausgangssudoku soll gelöst werden. Der Algorithmus fängt in der oberen linken Ecke v_1 an und setzt nach entsprechender Prüfung (siehe Beschreibung des Backtracking-Algorithmus auf vorheriger Seite) eine Zahl von 1 bis 9 ein. In den nächsten beiden Sudoku-Tabellen sind neue Zuweisungen als hellgraue Zahl eingetragen.

3	5	2	1	4	9	7	6	X
8	4	1	3	6				
					8			
4		9					2	
	6						4	
	2					9		3
			2					
				9	1	3	7	8
		7	5			1		

Nach mehreren Iterationen kann dem Feld v_9 (mit einem 'X' gekennzeichnet) keine Farbe, kleiner 9, zugewiesen werden. Nun geht der Algorithmus feldweise zurück, bis er einem bereits zuvor bearbeiteten Feld eine neue Farbe zuweisen kann (hier ist das Feld schwarz umrandet).

3	5	2	1	4	9	7	8	6
8	4	1	3	6	2	5	9	X
					8			
4		9					2	
	6						4	
	2					9		3
			2					
				9	1	3	7	8
		7	5			1		

Den nächsten Feldern, die nach dem geänderten Feld folgen, wird nach der Neu-zuweisung wieder eine neue Farbe von 1 bis 9 zugewiesen, bis eine Zuweisung erneut nicht möglich ist. Der Backtrack-Algorithmus terminiert, sobald das Sudoku gelöst wurde.

5. Weitere Anwendungsfälle der Graphenfärbung

Mit dem Färben von Graphen können viele Problemstellungen aus unterschiedlichen wissenschaftlichen Gebieten gelöst werden.

Typische Probleme sind beispielsweise das Testen von elektrischen Schaltkreisen auf Kurzschlüsse, Zuweisungsprobleme aus der Informatik (Register-Zuweisungsprobleme) oder die Flußkontrolle in Netzwerken.

Darunter gibt es auch viele Problemstellungen aus dem Bereich der Chemie und der Physik, welche mit dem Färben von Graphen gelöst werden können. [Lew16c]

Weitere Beispiele für Anwendungsfälle sind:

- Ampelschaltungen oder artgerechte Zoo-Gestaltung [Nit09]
- Planung von Sportligen oder Stundenplänen [Lew16c]

Literatur

- [Fra05] Bertram Felgenhauer Frazer Jarvis. „Enumerating possible Sudoku grids“. In: (2005), S. 2, 6 (siehe S. 6).
- [Lew16a] R.M.R. Lewis. „A Guide to Graph Coloring“. In: (2016), S. 10–11 (siehe S. 3).
- [Lew16b] R.M.R. Lewis. „A Guide to Graph Coloring“. In: (2016), S. 9–14 (siehe S. 4).
- [Lew16c] R.M.R. Lewis. „A Guide to Graph Coloring“. In: (2016) (siehe S. 9).
- [Nit09] Manfred Nitzsche. „Graphen für Einsteiger“. In: (2009), S. 203–205 (siehe S. 9).
- [Sve16] Hartmut Noltenmeier Sven Oliver Krumke. „Graphentheoretische Konzepte und Algorithmen“. In: (2016) (siehe S. 4).
- [Tur13a] Volker Turau. „Algorithmische Graphentheorie“. In: (2013), S. 152 (siehe S. 3).
- [Tur13b] Volker Turau. „Algorithmische Graphentheorie“. In: (2013), S. 151 (siehe S. 11).

A. Pseudocode: Backtracking-Algorithmus

```
1  var f : array[1..max] of Integer;
2  function faerbung(G : Graph; c : Integer) : Boolean;
3      q : Boolean;
4  begin
5      Initialisiere f mit 0;
6      versuche(1,c,q);
7      return q;
8  end
9
10 procedure versuche(i, c : Integer; var q : Boolean);
11     farbe : Integer;
12 begin
13     Initialisiere farbe mit 0;
14     repeat
15         farbe := farbe + 1;
16         q := false;
17         if moeglich(i,farbe) then begin
18             f[i] := farbe;
19             if i < n then begin
20                 versuche(i+1,c,q);
21                 if q = false then
22                     f[i] := 0;
23             end
24             else
25                 q := true;
26             end;
27         until q = true or farbe = c;
28 end
29
30 function moeglich(i,farbe : Integer) : Boolean;
31     j : Integer;
32 begin
33     for jeden Nachbar j von i do
34         if f[j] = farbe then
35             return false;
36     return true;
37 end
```

[Tur13b]