Formale Techniken der Software-Entwicklung

Übung am 22.05.2015

Christian Kroiß

28. Mai 2015

Knoten-Färbung eines Graphen als SAT-Problem

Schema:
Sei $G = (V, E)$ ein ungerichteter Graph mit der Knotenmenge $V = \{v_1, \ldots, v_N\}$ und der Kantenmenge $E = \{e_1, \ldots, e_M\}$. Sei ferner $C = \{c_1, \ldots, c_K\}$ die Menge an *Farben*, mit denen die Knoten gefärbt werden sollen.

Für alle $1 \leq i \leq N$ und $1 \leq j \leq k$ bezeichne $C_{i,j}$ die Aussage, dass der Knoten v_i mit der Farbe c_j gefärbt wird.

Knoten-Färbung eines Graphen als SAT-Problem (2)

Es ergeben sich folgende Constraints:

1. Jeder Knoten muss **genau eine** Farbe haben.
   1.1 Jeder Knoten muss mindestens eine Farbe haben:

$$\bigwedge_{i \in [1,N]} (C_{i,1} \vee \ldots \vee C_{i,K})$$

   1.2 Kein Knoten darf mehr als eine Farbe haben:

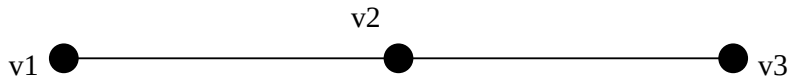
$$\bigwedge_{i \in [1,N]} \bigwedge_{k,l \in [1,K], k \neq l} \neg (C_{i,k} \wedge C_{i,l})$$

2. Benachbarte Knoten dürfen nicht die gleiche Farbe haben.

$$\bigwedge_{i \in [1,N]} \bigwedge_{j \in [1,N], \{i,j\} \in E} \bigwedge_{k \in [1,K]} \neg (C_{i,k} \wedge C_{j,k})$$

Aufgabe 2 b)

Gegeben sei der folgende ungerichtete Graph G:



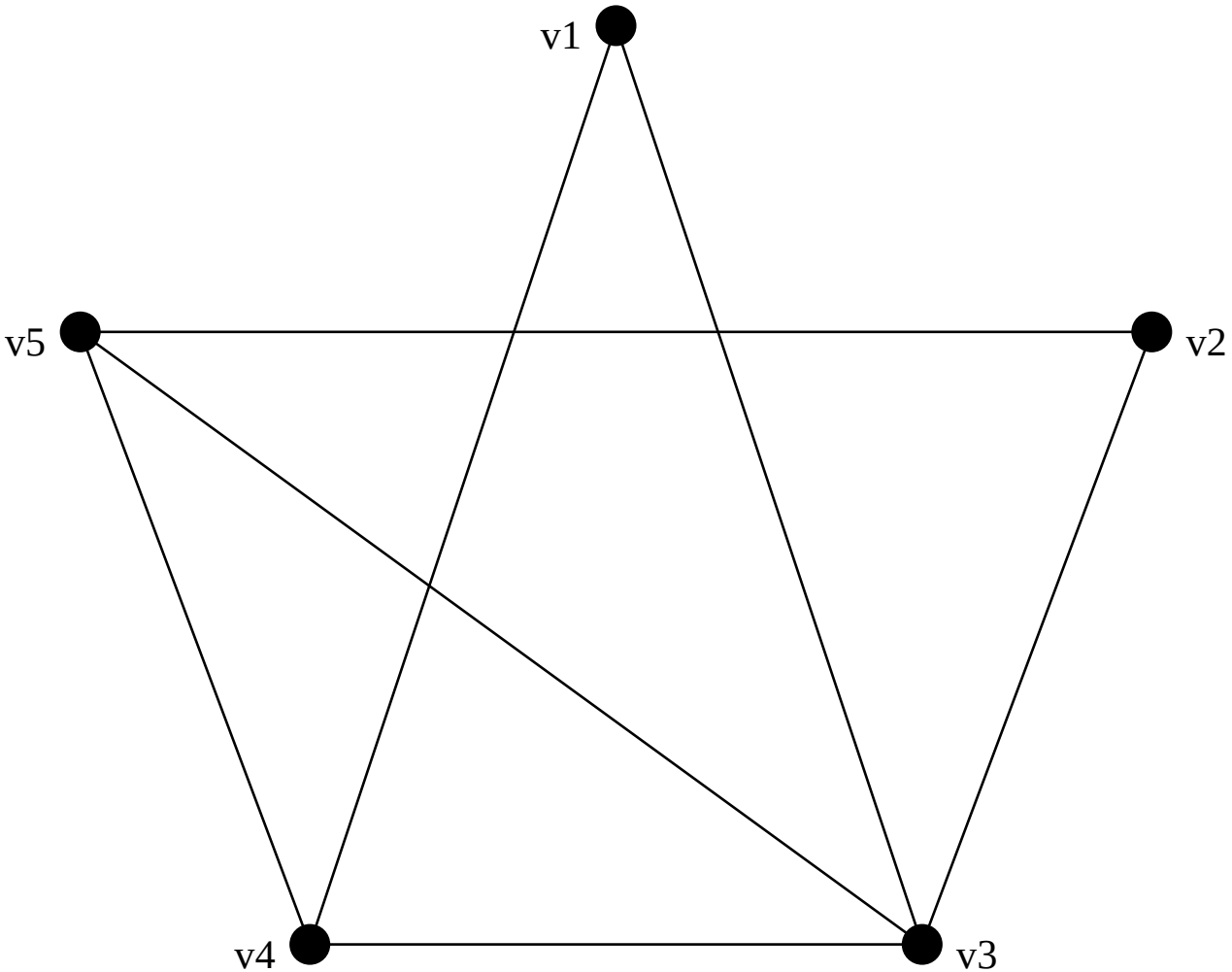
Aufgabe: Erstelle CNF-Formel, die eine gültige Kantenfärbung von G mit 2 Farben (z.b. Rot und Blau) modelliert.

$$\begin{aligned}\Phi \equiv\ & (C_{1,1} \vee C_{1,2}) \wedge (C_{2,1} \vee C_{2,2}) \wedge (C_{3,1} \vee C_{3,2}) \wedge \\ & \neg(C_{1,1} \wedge C_{1,2}) \wedge \neg(C_{2,1} \wedge C_{2,2}) \wedge \neg(C_{3,1} \wedge C_{3,2}) \wedge \\ & \neg(C_{1,1} \wedge C_{2,1}) \wedge \neg(C_{1,2} \wedge C_{2,2}) \wedge \\ & \neg(C_{2,1} \wedge C_{3,1}) \wedge \neg(C_{2,2} \wedge C_{3,2})\end{aligned}$$

Aufgabe 2 b) (2)

$$
\begin{aligned}
CNF(\Phi) \equiv\ & (C_{1,1} \vee C_{1,2}) \wedge (C_{2,1} \vee C_{2,2}) \wedge (C_{3,1} \vee C_{3,2}) \\
& (\neg C_{1,1} \vee \neg C_{1,2}) \wedge (\neg C_{2,1} \vee \neg C_{2,2}) \wedge (\neg C_{3,1} \vee \neg C_{3,2}) \wedge \\
& (\neg C_{1,1} \vee \neg C_{2,1}) \wedge (\neg C_{1,2} \vee \neg C_{2,2}) \wedge \\
& (\neg C_{2,1} \vee \neg C_{3,1}) \wedge (\neg C_{2,2} \vee \neg C_{3,2})
\end{aligned}
$$

Ein etwas komplexerer Graph



Ein etwas komplexerer Graph

- Anstatt das Schema manuell auf den Graph von der letzten Folie anzuwenden, kann das natürlich auch leicht automatisiert werden.
- Eine Beispiel-Implementierung dazu findet man im Eclipse-Projekt `graphdyer`, das in der ZIP-Datei zu dieser Übung enthalten ist.
- Da das Projekt den selbstgebauten SAT-Solver `simplesat` verwendet, müssen beide Projekte in den Eclipse-Workspace importiert werden.
- Mit dem JUnit-Test-Case in der Klasse `graphdyer.test.graphdyer.GraphDyerTest` kann für eine beliebige Farbanzahl eine KNF-Formel für den Graphen erstellt werden und (falls vorhanden) eine gültige Färbung berechnet werden.

Ein etwas komplexerer Graph

Ausgabe von `graphdyer.test.graphdyer.GraphDyerTest` für `numColors = 3`:

```
(v1_1 | v1_2 | v1_3) & (v2_1 | v2_2 | v2_3) &
(v3_1 | v3_2 | v3_3) & (v4_1 | v4_2 | v4_3) &
(v5_1 | v5_2 | v5_3) &
(!v1_1 | !v1_2) & (!v1_1 | !v1_3) & (!v1_2 | !v1_3) &
(!v2_1 | !v2_2) & (!v2_1 | !v2_3) & (!v2_2 | !v2_3) &
(!v3_1 | !v3_2) & (!v3_1 | !v3_3) & (!v3_2 | !v3_3) &
(!v4_1 | !v4_2) & (!v4_1 | !v4_3) & (!v4_2 | !v4_3) &
(!v5_1 | !v5_2) & (!v5_1 | !v5_3) & (!v5_2 | !v5_3) &
(!v1_1 | !v3_1) & (!v1_2 | !v3_2) & (!v1_3 | !v3_3) &
(!v1_1 | !v4_1) & (!v1_2 | !v4_2) & (!v1_3 | !v4_3) &
(!v2_1 | !v3_1) & (!v2_2 | !v3_2) & (!v2_3 | !v3_3) &
(!v2_1 | !v5_1) & (!v2_2 | !v5_2) & (!v2_3 | !v5_3) &
(!v3_1 | !v4_1) & (!v3_2 | !v4_2) & (!v3_3 | !v4_3) &
(!v3_1 | !v5_1) & (!v3_2 | !v5_2) & (!v3_3 | !v5_3) &
(!v4_1 | !v5_1) & (!v4_2 | !v5_2) & (!v4_3 | !v5_3)
```

Ein etwas komplexerer Graph

Ausgabe von `graphdyer.test.graphdyer.GraphDyerTest` für `numColors = 3` (Fortsetzung):

```
v1_1 -> false, v1_2 -> true, v1_3 -> false
v2_1 -> false, v2_2 -> false, v2_3 -> true
v3_1 -> true, v3_2 -> false, v3_3 -> false
v4_1 -> false, v4_2 -> false, v4_3 -> true
v5_1 -> false, v5_2 -> true, v5_3 -> false


Coloring:
-----------
  v1 -> 2
  v2 -> 3
  v3 -> 1
  v4 -> 3
  v5 -> 2
```

Ein etwas komplexerer aber jetzt hübsch bunter Graph