Übung 12 Abgabe 19.7.2012

Aufgabe 1 – Das 2-Erfüllbarkeitsproblem (2-SAT) im Fußball 30 Punkte

In der Fußballbundesliga müssen jährlich neue Spielpläne ausgearbeitet werden. Da jeder zweimal gegen jeden spielt, gibt es bei 18 Mannschaften zwei Halbserien zu je 17 Spieltagen. Zuerst werden für jeden Spieltag die Paarungen bestimmt, und dann wird festgelegt, welche der beiden Mannschaften zu Hause bzw. auswärts antreten muss (die zweite Halbserie ist dabei identisch zur ersten, nur dass Heim- und Auswärtsspiele vertauscht werden). Idealerweise wechseln sich Heim- und Auswärtsspiele immer ab, aber es kann mathematisch gezeigt werden, dass dies maximal für zwei Mannschaften erreicht werden kann. Alle anderen Mannschaften haben mindestens eine Doppelung (einmal zwei Heim- oder Auswärtsspiele hintereinander). Wir nehmen in dieser Aufgabe an, dass die Paarungen bereits gegeben sind, und suchen eine optimale Heim-/Auswärts-Zuordnung (maximal eine Doppelung pro Mannschaft). Dazu gehen wir wie folgt vor:

1. Wir suchen zunächst nach dem Gegenteil, nämlich nach einer Zuordnung, bei der eine Mannschaft *nur* zu Hause spielt, eine zweite *nur* auswärts, und alle anderen Mannschaften haben einen Wechsel (spielen also zuerst nur auswärts, dann nur zu Hause oder umgekehrt).
2. Haben wir eine solche Zuordnung gefunden, kehren wir an allen geradzahligen Spieltagen die Zuordnung um. Offensichtlich ist damit eine optimale Zuordnung gefunden.

Schritt 1 kann effizient als 2-SAT-Problem formuliert werden (der naive Ansatz, alle Möglichkeiten zu probieren, ist zu langsam). Dazu führen wir Variablen h_{ms} und a_{ms} ein, die den Wert `true` haben, wenn Mannschaft m am Spieltag s ein Heimspiel bzw. ein Auswärtsspiel hat. Offensichtlich muss immer $h_{ms} = \neg a_{ms}$ gelten[1], weil keine Mannschaft gleichzeitig zu Hause und auswärts spielen kann. Außerdem muss $h_{ms} = a_{ns}$ gelten, wenn am Spieltag s Mannschaft m gegen Mannschaft n spielt (Konsistenz der Paarungen). Wir entscheiden nun willkürlich, dass Mannschaft k nur zu Hause spielt. Damit ist für alle anderen Mannschaften je ein Auswärtsspiel festgelegt.

Nehmen wir an, dass dieses Auswärtsspiel für Mannschaft $m \neq k$ am Spieltag s_m stattfindet, dass also $= \texttt{true}$ gilt. Damit *vor* Spieltag s_m höchstens ein Wechsel stattfindet, darf es in diesem Zeitraum keinen Wechsel von auswärts nach heim geben, denn sonst wäre mindestens ein zweiter Wechsel von heim nach auswärts nötig, weil der Spieltag s_m ja ein Auswärtsspiel ist. Um diese verbotenen Wechsel auszuschließen, muss für alle Spieltage mit $t < s_m$ gelten

$$\forall\, m \neq k,\, t < s_m\colon\ \neg(a_{mt} \wedge h_{m(t+1)}) \tag{1}$$

Eine analoge Bedingung muss für alle Spieltage mit $t > s_m$ erfüllt sein

$$\forall\, m \neq k,\, t > s_m\colon\ \neg(h_{m(t-1)} \wedge a_{mt}) \tag{2}$$

damit *nach* Spieltag s_m höchstens ein Wechsel stattfindet. Die Forderung, dass insgesamt höchstens ein Wechsel stattfinden darf, erfüllt man schließlich durch die Bedingung, dass keine Mannschaft (außer k) sowohl am Anfang als auch am Ende der Halbserie ein Heimspiel hat:

$$\forall\, m \neq k\colon\ \neg(h_{m\,1} \wedge h_{m\,17}) \tag{3}$$

Die Paarungen sind unter http://hci.iwr.uni-heidelberg.de/Staff/ukoethe/download/bundesliga-paarungen-12-13.json als JSON-File mit folgender Struktur gespeichert:

```
{
    "Mannschaft_1": {
        "Mannschaft_2": Spieltag,
        "Mannschaft_3": Spieltag,
        ...
    },
    ...
}
```

[1] Wir verwenden die übliche Notation für logische Ausdrücke: ¬ (Negation), ∧ (Konjunktion – und), ∨ (Disjunktion – oder), ∀ (für alle) und ∃ (es existiert).

Wenn Sie das File mittels `pairs = json.load(file('bundesliga-paarungen-12-13.json'))` einlesen, sagt Ihnen `pairs['VfL Wolfsburg']['1899 Hoffenheim']` beispielsweise, dass die beiden Mannschaften am 12. Spieltag gegeneinander spielen.

Aufgaben (alle Funktionen sollen im File "`bundesliga.py`" abgegeben werden):

a) Überführen Sie die Bedingungen (1)-(3) mit Hilfe der De Morganschen Gesetze in 2-Konjunktionen-Normalform. Um auszudrücken, dass Klauseln analog für alle Zeitpunkte von t=1 bis $t=s_m$-1 gelten, verwenden Sie den Und-Operator in der Form (analog zum Summenzeichen). Transformieren Sie die 2-Konjunktionen-Normalform danach in Implikationen-Normalform. *8 Punkte*

b) Schreiben Sie einen Algorithmus `graph = implicationGraph(pairs, k)`, der für die gegebenen Paarungen und gegebenes $k \in$ { "1. FC Nürnberg", "1. FSV Mainz 05", ..., "VfL Wolfsburg"} den zugehörigen gerichteten Implikationengraph erzeugt. *6 Punkte*

c) Schreiben Sie eine Funktion `isSatisfiable(graph)`, die den Strongly Connected Components-Algorithmus (siehe Vorlesung) implementiert. Geben Sie mit Hilfe dieser Funktion an, für welche k das 2-SAT-Problem erfüllbar ist. *8 Punkte*

d) Schreiben Sie eine Funktion `result = homeOrAway(pairs, k)`, die den randomisierten Lösungsalgorithmus (siehe Vorlesung) verwendet, um für ein geeignetes k eine Heim/Auswärtszuordnung zu bestimmen. Die Ausgabe soll die gleiche Struktur wie `pairs` haben, aber `result[m1][m2]` enthält den Spieltag genau dann als *negative Zahl*, wenn Mannschaft *m1* auswärts antreten muss (es gilt also stets `result[m2][m1] == -result[m1][m2]`). Speichern Sie `result` als JSON-File "`bundesliga_home_or_away.json`" und geben Sie dieses File ab. Messen Sie für verschiedene Initialisierungen, wie viele Schritte (= Variableninvertierungen) der Algorithmus benötigt, bis er zu einer Lösung gelangt. Werden tatsächlich im Mittel $O(n^2)$ Schritte benötigt, wie von der random-walk-Analyse vorhergesagt? *8 Punkte*

Aufgabe 2 – Der RANSAC-Algorithmus für Kreise 18 Punkte

Gegeben ist das File "http://hci.iwr.uni-heidelberg.de/Staff/ukoethe/download/noisy-circles.txt", das zufällig verteilte 2D-Punkte enthält (ein Punkt pro Zeile), von denen einige deutlich Kreise markieren.

a) Benutzen Sie `gnuplot` oder ein anderes geeignetes Programm, um die Punkte zu zeichnen. In `gnuplot` sollte man durch den Befehl "`set size square`" ein quadratisches Ausgabefenster erzwingen (sonst werden die Kreise zu Ellipsen). Der entsprechende Befehl bei `pylab` lautet "`pylab.axes().set_aspect('equal')`". *4 Punkte*

b) Implementieren Sie den RANSAC-Algorithmus für Kreise: *10 Punkte*

1. Wiederhole hinreichend oft:
   - (A) Wähle zufällig drei Punkte und bestimme den Umkreis des dadurch definierten Dreiecks.
   - (B) Zähle die Anzahl der "Inlier", d.h. die Anzahl der Punkte, deren Abstand von diesem Kreis höchstens p beträgt (für geeignet gewähltes p).
   - (C) Wenn dieser Kreis mehr "Inlier" hat als der beste bisher bekannte, speichere den Kreis und die zugehörigen Inlier.
2. Falls weitere Kreise detektiert werden sollen: Entferne die Inlier des letzten Kreises aus der Liste und gehe zu 1.

Der Algorithmus soll als Funktion `circles = circleRANSAC(points, p, numberOfCircles)` implementiert werden, die im File "`circles.py`" abzugeben ist. `circles` ist ein Array, das jeden Kreis durch Mittelpunktskoordinaten und Radius repräsentiert.

c) Zeichnen Sie die detektierten Kreise ebenfalls in Ihr Diagramm ein und geben Sie das Diagramm als Bilddatei (z.B. PNG oder PDF) ab. In `gnuplot` verwendet man für Kreise den Befehl *4 Punkte*

```
>>> plot [0:2*pi] r*cos(t)+xcenter,r*sin(t)+ycenter
```