Lehrstuhl für Informatik I
Informationsstrukturen und wissensbasierte Systeme
Universität Würzburg

Prof. Dr. H. Noltemeier
Dr. S. O. Krumke

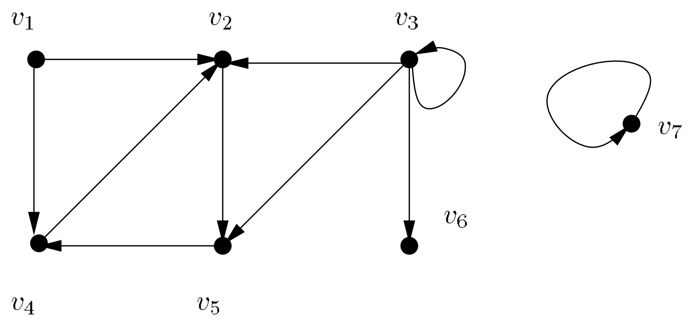
Würzburg, den 21. November 1997

Informationsstrukturen und wissensbasierte Systeme
(WS 97/98)

1. Übung mit Musterlösung

Aufgabe 1: (Grundlegende graphentheoretische Konzepte)

Gegeben sei der folgende gerichtete Graph $G = (V, R)$.



(a) Wenden Sie den *Depth-First-Search-Algorithmus* auf den oben gezeichneten Graphen G an. Beginnen Sie dabei mit der Ecke v_1. Geben Sie an, in welcher Reihenfolge die Ecken durch Depth-First-Search betrachtet werden und geben sie den bei Abbruch entstandenen Depth-First-Search-Wald an. **(2 Punkte)**

(b) In welcher Reihenfolge werden die Ecken von G durch den *Breadth-First-Search-Algorithmus* besucht, wenn man dabei ebenfalls mit v_1 startet? **(2 Punkte)**

(c) Geben Sie für den oben gezeichneten Graphen G alle starken Zusammenhangskomponenten an. **(1 Punkt)**

(d) Wie sehen die schwachen Zusammenhangskomponenten des Graphen G aus? **(1 Punkt)**

Aufgabe 2: (Erfüllbarkeitsprobleme in der Aussagenlogik)

Es sei $X = \{A_1, \ldots, A_n\}$ eine Menge von atomaren Formeln. Man nennt A_i und $\neg A_i$ *Literale*. Das Literal A_i ist *positiv*, das Literal $\neg A_i$ *negativ*.

Eine *Belegung* $\mathcal{B}$ für die Atomformeln in X ist eine Funktion $\mathcal{B}\colon X \to \{0,1\}$. Gilt $\mathcal{B}(A) = 1$, so sagen wir, daß A *unter* $\mathcal{B}$ *wahr* ist, ansonsten nennen wir A *unter* $\mathcal{B}$ *falsch*. Das Literal A ist genau dann wahr unter $\mathcal{B}$, wenn A unter $\mathcal{B}$ wahr ist. Das Literal $\neg A$ ist genau dann wahr unter $\mathcal{B}$, wenn A unter $\mathcal{B}$ falsch ist.

Eine *Klausel* ist eine Menge von Literalen über X, z.B. $C = \{A_1, \neg A_2, A_5\}$. Eine Klausel C repräsentiert die Disjunktion dieser Literale. Eine Klausel C wird durch die Belegung $\mathcal{B}$ *erfüllt* genau dann, wenn mindestens eines der Literale aus C unter $\mathcal{B}$ wahr ist. Eine Menge M von Klauseln heißt *erfüllbar*, wenn es eine *Belegung* gibt, die alle Klauseln in M erfüllt. Solch eine Belegung nennt man dann eine *erfüllende Belegung* für M.

Eine Instanz (X, M) des *Erfüllbarkeitsproblems* SAT ist durch eine Menge X von Atomformeln und eine Menge M von Klauseln über X gegeben. Das Problem besteht darin zu entscheiden, ob M erfüllbar ist. Mit k-SAT bezeichnen wir die Teilmenge der Instanzen von SAT, in der jede Klausel *genau* k Literale besitzt.

Das Problem SAT ist ein klassisches NP-vollständiges Problem. Bereits die Einschränkung 3-SAT ist NP-vollständig. Von 2-SAT weiß man hingegen, daß dieses Problem in Polynomialzeit gelöst werden kann.

Das Problem MAX SAT besteht darin, zu einer Menge X von Atomformeln und einer Menge M von Klauseln über X eine Belegung $\mathcal{B}$ zu finden, die die größtmögliche Anzahl von Klauseln erfüllt. MAX k-SAT besteht aus den Instanzen von MAX SAT, bei denen jede Klausel genau k Literale enthält.

(a) Ist die Menge $M = \{C_1, C_2, C_3, C_4\}$ mit $C_1 = \{A_1, \neg A_1, \neg A_2\}$, $C_2 = \{A_3, A_2, A_4\}$, $C_3 = \{\neg A_1, \neg A_3, \neg A_4\}$, $C_4 = \{A_2, \neg A_1, \neg A_3\}$ erfüllbar? **(2 Punkte)**

(b) Sei (X, M) eine beliebige Instanz von MAX k-SAT mit $M = \{C_1, \ldots, C_m\}$. Wir betrachten eine zufällige Belegung $\mathcal{B}\colon X \to \{0,1\}$ der Variablen aus X. Wie groß ist die Wahrscheinlichkeit, daß $\mathcal{B}$ die Klausel C_i erfüllt? Wie groß ist der Erwartungswert der Anzahl der erfüllten Klauseln? **(3 Punkte)**

(c) Zeigen Sie, daß es zu jeder Klauselmenge $M = \{C_1, \ldots, C_m\}$ mit m Klauseln eine Belegung $\mathcal{B}$ gibt, die mindestens $m/2$ Klauseln erfüllt. Geben Sie einen Algorithmus an, der in $\mathcal{O}(n + \sum_{i=1}^{m} |C_i|)$ Zeit eine solche Belegung findet. **(2 Punkte)**

Aufgabe 3: (Hillclimbing für MAX CUT)

Eine Instanz des Problems MAX CUT ist durch einen endlichen ungerichteten Graphen $G = (V, E)$ gegeben. Das Problem besteht darin, die Eckenmenge V in zwei Mengen S und $V \setminus S$ zu partitionieren, so daß die Anzahl der Kanten, die einen Endpunkt in S und den anderen in $V \setminus S$ haben, maximiert wird.

Wir betrachten nun das Hillclimbing-Verfahren aus der Vorlesung zur approximativen Lösung von MAX CUT. Als Zustandsmenge Z wählen wir

$$Z := \{ S : S \subseteq V \}.$$

Für einen Zustand $z = S$ besteht die Menge seiner Nachfolger aus alle denjenigen Zuständen, die durch Hinzufügen oder Entfernen einer einzelnen Ecke aus S entstehen:

$$N(z) := \{ S \cup \{v\} : v \in V \setminus S \} \cup \{ S \setminus \{v\} : v \in S \}.$$

Die Bewertung $\varphi(z)$ eines Zustandes $z = S$ ist die Anzahl der Kanten zwischen S und $V \setminus S$.

Der Hillclimbing-Algorithmus werde für diese Aufgabe so modifiziert, daß er vom aktuellen Zustand z nur dann in einen Folgezustand $z' \in N(z)$ übergeht, wenn $\varphi(z') > \varphi(z)$ gilt.

(a) Zeigen Sie, daß der (modifizierte) Hillclimbing-Algorithmus nach endlich vielen Schritten terminiert, aber nicht notwendigerweise eine optimale Lösung findet. **(2 Punkte)**

(b) Sei OPT der Wert der optimalen Lösung $(S^*, V \setminus S^*)$. Beweisen Sie, daß für den Zustand z_t, der vom (modifizierten) Hillclimbing-Algorithmus bei Abbruch geliefert wird, gilt: $\varphi(z_t) \geq \mathrm{OPT}/2$. **(4 Punkte)**

Hinweis: Sei $z_t = S$ die vom Algorithmus gelieferte Lösung. Betrachten sie die Partitionierung von V in $V = V_1 \cup V_2 \cup V_3 \cup V_4$, so daß $S = V_1 \cup V_2$, $V \setminus S = V_3 \cup V_4$ und $(S^*, V \setminus S^*) = (V_1 \cup V_3, V_2 \cup V_4)$ gilt. Seien e_{ij} die Anzahl der Kanten mit einem Endpunkt in V_i und dem anderen in V_j. Es gilt dann

$$\begin{aligned} \mathrm{opt} &= e_{12} + e_{34} + e_{14} + e_{23} \\ \varphi(z_t) &= e_{13} + e_{14} + e_{23} + e_{24}. \end{aligned}$$

Zeigen Sie zunächst die Ungleichung $2e_{11} + e_{12} \leq e_{13} + e_{14}$ und folgern sie daraus $e_{12} \leq e_{13} + e_{14}$. Beweisen Sie dann analoge Ungleichungen für e_{34}, e_{14} und e_{23}.

Abgabetermin: Dienstag, 18. November 1997, bis 10^{00} Uhr in den dafür vorgesehenen Kästen vor der Teilbibliothek Mathematik/Informatik

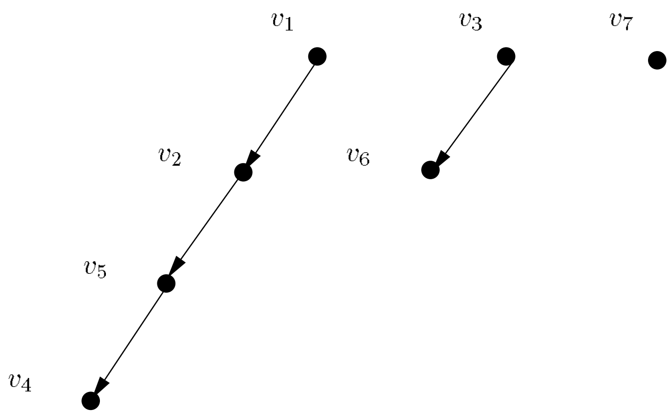
Hinweise: Bis zu zwei Übungsteilnehmer dürfen gemeinsam ein Übungsblatt abgeben.

Die erste Übung findet am Freitag, den 21. November, statt. Die Gruppeneinteilung entnehmen Sie bitte den Aushängen.

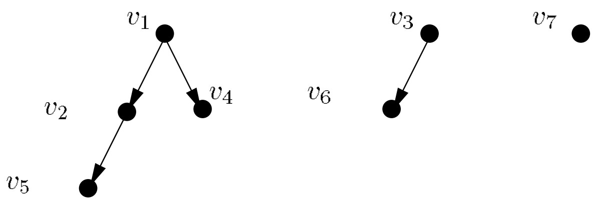
Lösungen

Aufgabe 1: (Grundlegende graphentheoretische Konzepte)

(a) Der DFS-Algorithmus (Tiefensuche) liefert folgenden Baum:



(b) Der BFS-Algorithmus (Breitensuche) liefert folgenden Baum:



(c) Es gibt folgende 5 starke Zusammenhangskomponenten:

$$ZK_1 = \{v_1\}$$
$$ZK_2 = \{v_2, v_4, v_5\}$$
$$ZK_3 = \{v_3\}$$
$$ZK_4 = \{v_6\}$$
$$ZK_5 = \{v_7\}$$

(d) Es gibt folgende 2 schwache Zusammenhangskomponenten:

$$ZK_1 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$
$$ZK_2 = \{v_7\}$$

Aufgabe 2: (Erfüllbarkeitsprobleme)

(a) Durch scharfes Hinschauen erkennen wir zuerst, daß die Klausel $C_1 = \{A_1, \neg A_1, \neg A_2\}$ *gültig* (d.h. unter allen Belegungen erfüllt) ist, da A_1 oder $\neg A_1$ wahr ist. Für die anderen Klauseln betrachten wir folgende Wahrheitswertetabelle:

A_1	A_2	A_3	A_4	$\mathcal{B}(C_2)$	$\mathcal{B}(C_3)$	$\mathcal{B}(C_4)$	$\mathcal{B}(M)$
0	0	0	0	0	1	1	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	1
0	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1
0	1	0	1	1	1	1	1
0	1	1	0	1	1	1	1
0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	1	0	1	0

Man sieht also, daß es für M erfüllende wie nicht erfüllende Belegungen gibt. M ist also erfüllbar aber nicht gültig. Ein alternativer Lösungsweg wäre gewesen, zu zeigen daß man aus der Klauselmenge nicht die leere Menge resolvieren kann (Vorlesung Grundstudium: Logik für Informatiker).

(b) Die Wahrscheinlichkeit, daß eine zufällige Belegung $\mathcal{B}$ die Klausel C_i erfüllt, ist gleich der Wahrscheinlichkeit P_i, daß ein oder mehr Literale aus C_i erfüllt sind. Dies ist gleich der Gegenwahrscheinlichkeit $1-P_i$, daß kein Literal erfüllt ist. Die Wahrscheinlichkeit p_a, daß eine atomare Formel A_j den Wert 1 hat ist 1/2.
Es gilt also

$$\begin{aligned} P(\mathcal{B} \models C_i) &= 1 - p_a^k \\ &= 1 - (1/2)^k \\ &= 1 - 2^{-k} \end{aligned}$$

Sei die Zufallsvariable X die Anzahl der Klauseln $\in M$ mit $\mathcal{B} \models$ diese Klauseln. Es ist:

$$\begin{aligned} E(X) &= \sum_{l=1}^{m} P(\mathcal{B} \models C_l) \\ &= \sum_{l=1}^{m} 1 - 2^{-k} \\ &= m(1 - 2^{-k}) \end{aligned}$$

(c) Wir zeigen, daß jede Belegung $\mathcal{B}$ die Klauselmenge in zwei Teile partitioniert, so daß entweder $\mathcal{B}$ oder die komplementäre Belegung mindestens $m/2$ Klauseln erfüllt.

Definition: $\forall A \in X : \overline{\mathcal{B}}(A) =_{def} 1 - \mathcal{B}(A)$

(A) Sei $\mathcal{B}$ eine zur Klausel C passende Belegung so gilt $\mathcal{B} \not\models C \Rightarrow \overline{\mathcal{B}} \models C$

(B) Also gilt für jede Klausel $C : \mathcal{B} \models C$ oder $\overline{\mathcal{B}} \models C$. Falls $\mathcal{B}$ r Klausen erfüllt und $r < \frac{m}{2}$ so erfüllt $\overline{\mathcal{B}}$ aber mindestens $m - r > \frac{m}{2}$ Klauseln. Damit ist entweder $\mathcal{B}$ oder $\overline{\mathcal{B}}$ eine Belegung mit der gesucheten Eigenschaft. $\square$

Algorithmus:

- Array B[] der Größe n entspreche der Belegung der atomaren Formeln also
  B[i]=x $\Rightarrow$ $\mathcal{B}(A_i) = x$
- C sei Array einer Klauselklasse so daß
  C[i].laenge $=_{def}$ $|C_i|$
  C[i].element(j) $=_{def}$ L_{ij}
- L sei eine Literalklasse, so daß
  L.vorzeichen=NEG|POS wenn NEG falls L$=_{def}$ $\neg A_i$ sonst POS
  L.atom=i wenn L $=_{def}$ $\neg A_i$ oder A_i

Algorithmus 1 Algorithmus für passende Belegung

Input: Klauselmenge C

```
for (x=1; x<n+1; x++)
    B[x]=1;
erfuellt=0;
for (x=1; x<m+1; x++)
    for (y=1; y<C[x].laenge; y++)
        {
        L=C[x].element(y);
        if (L.vorzeichen ==POS)
            {erfuellt++; break}
        }
if (erfuellt < (m/2))
    for (x=1; x<n+1; x++)
        B[x]=0
```

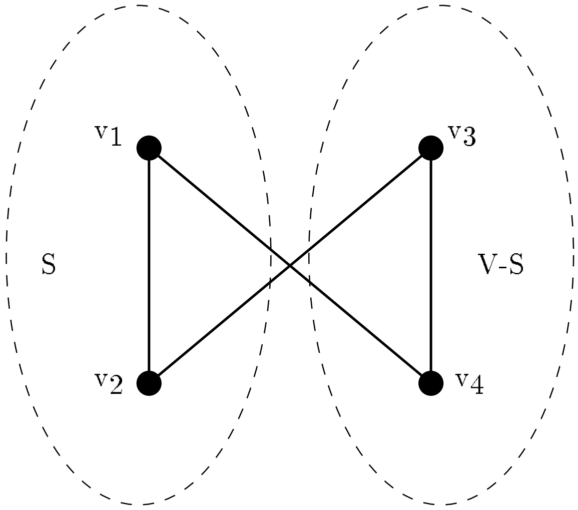
Dieser Algorithmus ist wie leicht zu erkennen (siehe FOR-Schleifen) in Laufzeit $\mathcal{O}(n + \sum_{i=1}^{m} |C_i|)$ zu berrechnen.

Aufgabe 3: (Hillclimbing)

(a) Da der modifizierte Hillclimbing-Algorithmus von einem Zustand z nur dann zu einem neuen Zustand z' übergeht, wenn $\varphi(z') > \varphi(z)$ gilt, ist eine spätere Rückkehr zu z ausgeschlossen, weil sonst $\varphi(z) > \varphi(z)$ gelten müsste.
Gleichzeitig ist die Menge Z der Zustände endlich, da G ein endlicher Graph ist, und somit auch nur endlich viele Partitionen auf G existieren. Da Z endlich ist, und kein Zustand mehrmals angenommen wird, können nur endlich viele Zustandsübergänge erfolgen. Der Algorithmus terminiert

also nach endlich vielen Schritten.

Dass der modifizierte Hillclimbing-Algorithmus nicht notwendigerweise eine optimale Lösung findet, erkennt man schnell an einem Gegenbeispiel. Betrachten wir etwa den folgenden Graphen:



Angenommen der Algorithmus befindet sich im Startzustand z mit der gezeigten Partition, also $S = \{v_1, v_2\}$. Die Menge $N(z)$ der Nachbarzustände hat genau 4 Elemente, nämlich $N(z) = \{\{v_1\}, \{v_2\}, \{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}\}$, wobei für alle $z' \in N(z)$ gilt: $\varphi(z') = 2 = \varphi(z)$
Da keiner der Nachbarzustände eine bessere Bewertung hat als der Startzustand, terminiert der Algorithmus sofort mit der Lösung $z = \{v_1, v_2\}$ und $\varphi(z) = 2$.
Man sieht aber leicht, dass mit $z^* := \{v_1, v_3\}$ und $\varphi(z^*) = 4$ eine bessere Lösung existiert. □

(b) Sei $V = V_1 \cup V_2 \cup V_3 \cup V_4$ eine Partition von V gemäß dem angegebenen Hinweis. Wir betrachten nun einen Knoten $v \in V_1$ und seine adjazenten Kanten etwas näher.
Kanten, die von v aus zu einem Knoten in V_1 oder V_2 führen, liegen vollständig in S und tragen somit nichts zum Schnitt $(S, V \setminus S)$ bei, ganz im Gegensatz zu den Kanten, die von v aus zu einem Knoten in V_3 oder V_4 verlaufen. Da der modifizierte Hillclimbing-Algorithmus mit der Partition $(S, V \setminus S)$ abbricht, kann es höchsten so viele mit v adjazente Kanten in S geben wie es Kanten im Schnitt gibt, die mit v adjazent sind. Denn sonst hätte der HC-Algorithmus den Schnitt durch ein „Hinüberziehen“ von v nach $V \setminus S$ vergrößern können.
Weil dieser Sachverhalt für alle Ecken $v \in V_1$ gilt, ergibt sich durch Summation über alle Ecken in V_1, dass

$$2e_{11} + e_{12} \leq e_{13} + e_{14}$$

Der Faktor 2 bei e_{11} resultiert daraus, dass Kanten, die zwischen zwei Ecken aus V_1 verlaufen, bei der Summation doppelt gezählt werden. Da die

Zahl e_{11} der Ecken ≥ 0 ist, können wir sie aus der Ungleichung streichen ohne die Aussage zu verfälschen und man kann folgern:

$$e_{12} \leq e_{13} + e_{14}$$

Durch Anwendung der obigen Argumentation gelangen wir bei der Betrachtung der Ecken in V_3 zu der analogen Aussage:

$$e_{34} \leq e_{13} + e_{23}$$

Die Addition beider Ungleichungen liefert:

$$e_{12} + e_{34} \leq e_{13} + e_{14} + e_{13} + e_{23}$$

Wir addieren nun auf beiden Seiten $e_{14} + e_{23}$:

$$\begin{aligned} e_{12} + e_{34} + e_{14} + e_{23} &\leq e_{13} + e_{14} + e_{13} + e_{23} + e_{14} + e_{23} \\ &\Rightarrow \mathrm{OPT} \leq 2e_{13} + 2e_{14} + 2e_{23} \\ &\Rightarrow \mathrm{OPT} \leq 2e_{13} + 2e_{14} + 2e_{23} \leq 2(e_{13} + e_{14} + e_{23} + e_{24}) = 2\varphi(z_t) \end{aligned}$$

Und es folgt $\varphi(z_t) \geq \mathrm{OPT}/2$ $\square$.