Formale Grundlagen der Informatik 1
Kapitel 23
NP-Vollständigkeit (Teil 2)

Frank Heitmann
heitmann@informatik.uni-hamburg.de

5. Juli 2016

Die Klassen P und NP

$$
\begin{aligned}
P &:= \{L \mid \text{es gibt ein Polynom } p \text{ und eine} \\
&\quad p(n)\text{-zeitbeschränkte DTM } A, \\
&\quad \text{die } L \text{ entscheidet}\} \\
NP &:= \{L \mid \text{es gibt ein Polynom } p \text{ und eine} \\
&\quad p(n)\text{-zeitbeschränkte NTM } A, \\
&\quad \text{die } L \text{ entscheidet}\}
\end{aligned}
$$

Eine Sprache L ist in P, wenn eine eine DTM A und ein Polynom p existiert, so dass A bei einer Eingabe w der Länge n nach $p(n)$ Schritten spätestens anhält und dann korrekt akzeptiert oder ablehnt.
Entsprechend ist L in NP, wenn eine NTM existiert, die analog zu obigem auf jeder Rechnung auf w nach $p(n)$ Schritten hält.

Reduktionen

Definition (Reduktion)

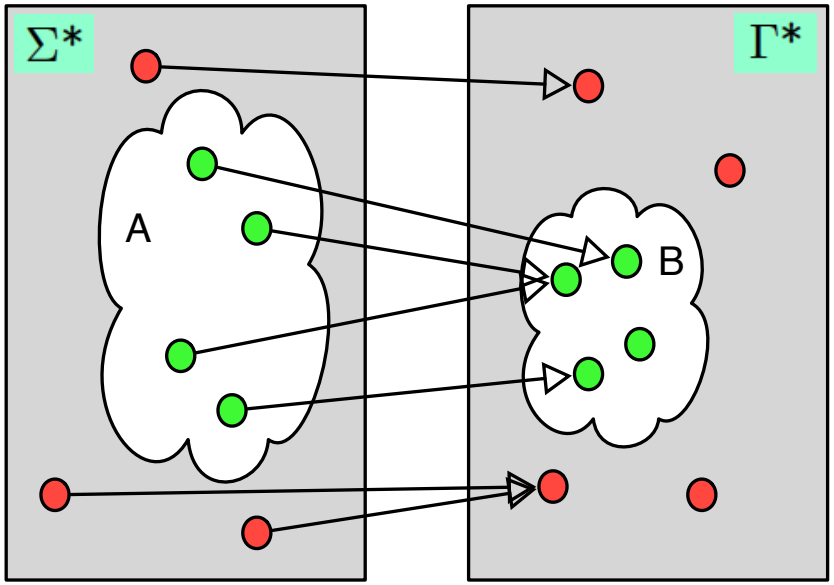
Seien $L_1, L_2 \subseteq \{0,1\}^*$ zwei Sprachen. Wir sagen, dass L_1 auf L_2 *in polynomialer Zeit reduziert wird*, wenn eine in Polynomialzeit berechenbare Funktion $f : \{0,1\}^* \to \{0,1\}^*$ existiert mit

$$x \in L_1 \text{ genau dann wenn } f(x) \in L_2$$

für alle $x \in \{0,1\}^*$ gilt. Hierfür schreiben wir dann $L_1 \leq_p L_2$. f wird als Reduktionsfunktion, ein Algorithmus der f berechnet als Reduktionsalgorithmus bezeichnet.

Eine Reduktion ist im Kern auch einfach ein Algorithmus. Es werden Probleminstanzen eines Problems in Probleminstanzen eines anderen Problems umgewandelt. Im Kern findet also eine (algorithmische) Konvertierung statt.

Reduktionen: Erläuterungen



- Ja-Instanzen ($x \in A$) auf Ja-Instanzen ($f(x) \in B$) abbilden,
- Nein-Instanzen ($x \notin A$) auf Nein-Instanzen ($f(x) \notin B$).
- Gleiche Antwort auf die Fragen '$x \in A$?' und '$f(x) \in B$?'
- Viele Ja-Instanzen können auf *eine* Ja-Instanz abgebildet werden. (Daher auch als 'many-one'-Reduktion bezeichnet.)

NP-vollständig

Definition

Eine Sprache $L \subseteq \{0,1\}^*$ wird als *NP*-vollständig bezeichnet, wenn

1. $L \in NP$ und
2. $L' \leq_p L$ für jedes $L' \in NP$ gilt.

Kann man für L zunächst nur die zweite Eigenschaft beweisen, so ist L *NP*-schwierig (-schwer/-hart).
Alle *NP*-vollständigen Probleme bilden die Komplexitätsklasse *NPC*.

Wichtige Sätze

Satz

Seien $L_1, L_2 \subseteq \{0,1\}^*$ mit $L_1 \leq_p L_2$, dann folgt aus $L_2 \in P$ auch $L_1 \in P$.

Theorem

Sei $L \in NPC$. Ist nun $L \in P$, so ist $NP = P$.

Lemma

Ist $L_1 \leq_p L_2$ und $L_2 \leq_p L_3$, so ist $L_1 \leq_p L_3$.

Theorem

Sei L eine Sprache und $L' \in NPC$. Gilt $L' \leq_p L$, so ist L NP-schwierig. Ist zusätzlich $L \in NP$, so ist L NP-vollständig.

Verfahren

Methode zum Beweis der NP-Vollständigkeit einer Sprache L:

1. Zeige $L \in NP$.
2. Wähle ein $L' \in NPC$ aus.
3. Gib einen Algorithmus an, der ein f berechnet, das jede Instanz $x \in \{0,1\}^*$ von L' auf eine Instanz $f(x)$ von L abbildet (also eine Reduktion).
4. Beweise, dass f die Eigenschaft $x \in L'$ gdw. $f(x) \in L$ für jedes $x \in \{0,1\}^*$ besitzt.
5. Beweise, dass f in Polynomialzeit berechnet werden kann.

Anmerkung

Die letzten drei Punkte zeigen $L' \leq_p L$. Mit dem vorherigen Satz folgt daraus und aus den ersten beiden Punkten $L \in NPC$.

NP-vollständige Probleme

Definition (SAT)

$\mathtt{SAT} = \{\langle\phi\rangle \mid \phi \text{ ist eine erfüllbare aussagenlogische Formel}\}$

Das *Erfüllbarkeitsproblem der Aussagenlogik*, SAT, ist historisch das erste *NP*-vollständige Problem. Weitere:

Definition (CNF)

$\mathtt{CNF} = \{\langle\phi\rangle \mid \phi \text{ ist eine erfüllbare aussagenlogische Formel in KNF}\}$

Definition (3CNF)

$\mathtt{3CNF} = \{\langle\phi\rangle \in \mathtt{CNF} \mid$
jede Klausel hat genau drei verschiedene Literale$\}$

NP-vollständige Probleme (2)

Definition (Clique)

$\texttt{CLIQUE} = \{\langle G, k\rangle \mid G \text{ enthält einen } K^k \text{ als Teilgraphen}\}$

Definition (Independent Set)

Gegeben ist ein ungerichteter Graph $G = (V, E)$ und ein $k \in \mathbb{N}$. Enthält G ein *Independent Set* der Größe k, d.h. k Knoten bei denen keine zwei miteinander verbunden sind?

Präsenzaufgabe Blatt 13

NP-vollständige Probleme (3)

Definition (Big-Clique)

$\texttt{Big-Clique} = \{\langle G \rangle \mid$
G enthält eine Clique aus mindestens $|V(G)|/2$ Knoten$\}$

Präsenzaufgabe Blatt 14

Es gibt viele weitere *NP*-vollständige Probleme und einige Probleme haben wir gesehen (ohne den Nachweis gemacht zu haben, dass sie in *NPC* sind), z.B. das Teilsummenproblem, das Mengenpartitionsproblem usw.

Ausblick: Wie löst man die Probleme dennoch?

Wir wissen jetzt, dass es Probleme in *NPC* gibt und dass es dort viele wichtige Probleme gibt. Wir wissen aber auch, dass man diese Probleme deterministisch nur sehr schlecht (in $2^{O(n^k)}$) lösen kann.

Dennoch gibt es Möglichkeiten diese Probleme zu attackieren:

- Einschränkung der Eingabe (z.B. Bäume statt Graphen)
- Approximationsalgorithmen (man kriegt nicht das Optimum)
- Randomisierte Algorithmen (z.B. manchmal kein Ergebnis)
- Heuristiken (Laufzeit und Güte des Ergebnisses meist unklar)
- ...

Bei neuen Problemen

Fragen/Aufgaben bei einem neuen Problem bzw. einer neuen Fragestellung:

- Ausdrücken / Modellieren / Formalisieren
- Entscheidbar (gut) oder unentscheidbar (schlecht)?
- In P (gut)?
- In NPC (schlecht)?
- Letzteres kann man mit obigen versuchen anzugehen!

Das ist nur ein erster grober Einstieg in das Wechselspiel von Algorithmenentwurf und Komplexitätstheorie. Es gibt dann neben P und NPC noch vieles weitere zu entdecken...

Fragen

Welche Aussage gilt unter der Annahme $P \neq NP$?

1. $P \subsetneq NP \subsetneq NPC$
2. $P \subsetneq NPC \subsetneq NP$
3. $P \subsetneq NPC$ und $NP \subsetneq NPC$
4. $P \subsetneq NP$ und $NPC \subsetneq NP$ und $P \cap NPC = \emptyset$
5. keins davon

Fragen

Sei *NPH* die Klasse der *NP*-schwierigen Probleme. Was gilt?

1. $NPC \cap NPH = \emptyset$
2. $NPC \cap NPH \neq \emptyset$ aber auch $NPC \setminus NPH \neq \emptyset$ und $NPH \setminus NPC \neq \emptyset$
3. $NPH \subseteq NPC$
4. $NPC \subseteq NPH$
5. keins davon

Fragen

Hat man $L_1 \leq_p L_2$, was stimmt?

1. Wenn L_1 unentscheidbar ist, dann auch L_2.
2. Wenn L_2 unentscheidbar ist, dann auch L_1.
3. Keines davon

Fragen

Sei A *NP*-schwierig. Was stimmt unter der Annahme $P \neq NP$?

1. $A \in P$
2. $A \in NP$
3. $A \in NP \setminus P$
4. $A \in P \setminus NP$
5. keins davon

Fragen

Sei $L_{NPC} \in NPC$ und die Komplexität von $L_?$ unbekannt. Welche Reduktion müssen Sie zeigen, um $L_?$ als *NP*-vollständig nachzuweisen?

1. $L_?$ auf L_{NPC} reduzieren
2. L_{NPC} auf $L_?$ reduzieren
3. Beide oben genannten Reduktionen
4. Welche Richtung ist wegen der Eigenschaft der Reduktion ($x \in L_1$ gdw. $f(x) \in L_2$) egal.

Fragen

Sei nochmal $L_{NPC} \in NPC$ und $L_?$ ein Problem mit unbekannter Komplexität. Was wissen Sie, wenn Sie doch $L_? \leq_p L_{NPC}$ zeigen?

1. Nichts! (Zumindest nichts hilfreiches!)
2. $L_? \in NPC$
3. $L_? \in NP$
4. L_{NPC} "erbt" die Komplexität von $L_?$, wenn wir diese ermittelt haben.

Fragen

Zur Nachbereitung

1. 4. ist richtig. Alle anderen entfallen, da sie $P \subset NPC$ enthalten, woraus $P = NP$ folgt.
2. 4. ist richtig. Jedes *NP*-vollständige Problem ist auch *NP*-schwierig, da für *NP*-schwierig weniger verlangt wird.
3. 1. ist richtig.
4. 5. ist richtig. Wenn A *NP*-schwierig ist, dann kann A auch außerhalb von *NP* liegen (es könnte aber auch noch $A \in NP$ und damit $A \in NPC$ gelten, dann ist $A \in NP \setminus P$).
5. 2. ist richtig. Zudem muss noch $L_? \in NP$ gezeigt werden.
6. 3. ist richtig. Man kann in Polynomialzeit mit der Reduktion $L_?$ auf L_{NPC} reduzieren und dann (in *NP*) L_{NPC} lösen. Damit hat man dann einen Algorithmus, der $L_? \in NP$ zeigt.

Weiteres Vorgehen

Zur Übung wollen wir nun noch weitere Probleme als *NP*-vollständig nachweisen.

2-SAT

Definition (2-SAT)

Sei 2-SAT die Menge aller (sinnvoll codierten) aussagenlogischen Formeln, die mindestens zwei erfüllende Belegungen haben. Zeigen Sie, dass 2-SAT *NP*-vollständig ist.

Ideen ?

2-SAT in NP

Beweis

Zunächst ist 2-SAT in *NP*. Als Zertifikat zu einer Formel F dienen zwei Belegungen. Der Verifikationsalgortihmus überprüft dann, ob die zwei Belegungen verschieden sind und ob jede die Formel F erfüllt. Dies ist in Polynomialzeit möglich und das Problem damit in *NP*. (Alternativ: Nichtdeterministisch werden zwei Belegungen geraten und dann wie oben deterministisch überprüft, ob diese verschieden sind und, falls ja, ob beide die Formel erfüllen.)

Reduktion auf 2-SAT

Beweis

Wir zeigen nun $\texttt{SAT} \leq_p \texttt{2-SAT}$, woraus wegen $\texttt{SAT} \in NPC$ folgt, dass 2-SAT *NP*-vollständig ist.
Sie F eine aussagenlogische Formel. Sei ferner X eine nicht in F auftretende atomare Formel. Die Reduktion macht aus F lediglich die Formel $G := F \vee X$ ...
Das wird nicht klappen! Ist F unerfüllbar, so hat G als erfüllende Belegungen all jene, die X wahr machen und mit den Variablen aus F etwas beliebiges machen. Das sind i.A. dann mehr als zwei Belegungen!

Reduktion auf 2-SAT

Beweis

Wir zeigen nun $\texttt{SAT} \leq_p \texttt{2-SAT}$, woraus wegen $\texttt{SAT} \in NPC$ folgt, dass 2-SAT *NP*-vollständig ist.
Sie F eine aussagenlogische Formel. Sei ferner X eine nicht in F auftretende atomare Formel. Die Reduktion macht aus F lediglich die Formel $G := (F \wedge X) \vee (F \wedge \neg X)$. Diese Reduktion geht ganz bestimmt in polynomieller Zeit in der Länge der Eingabe (also in der Länge von F). Es ist lediglich nötig einmal über F zu lesen, um die vorkommenden atomaren Formeln zu bestimmen und so eine neue auswählen zu können und dann die gewünschte Formel zu bilden, die von der Länge in $O(|F|)$ ist und daher auch in Polynomialzeit konstruiert werden kann.

Reduktion auf 2-SAT

Beweis

Es ist aber noch zu zeigen, dass $F \in \texttt{SAT}$ genau dann gilt, wenn $G \in \texttt{2-SAT}$ gilt. Sei dazu $F \in \texttt{SAT}$, also gibt es eine Belegung $\mathcal{A}$, die F wahr macht, d.h. $\mathcal{A}(F) = 1$ gilt. G wird dann von der Belegung $\mathcal{A}'$ mit $\mathcal{A}'(A) = \mathcal{A}(A)$ für jedes in F auftretende Aussagesymbol A und $\mathcal{A}'(X) = 1$ wahr gemacht und auch von der Belegung $\mathcal{A}''$, die wie $\mathcal{A}'$ definiert ist mit der Ausnahme von $\mathcal{A}''(X) = 0$. Es gibt also mindestens zwei erfüllende Belegungen für G und damit ist $G \in \texttt{2-SAT}$.

Reduktion auf 2-SAT

Beweis.

Sei andersherum $G \in \texttt{2-SAT}$ und sei $\mathcal{A}$ eine Belegung mit $\mathcal{A}(G) = 1$. $\mathcal{A}$ muss nun entweder $F \wedge X$ oder $F \wedge \neg X$ wahr machen und damit aber auf jeden Fall auch F, woraus $\mathcal{A}(F) = 1$ folgt und damit $F \in \texttt{SAT}$. (Tatsächlich sind F und G äquivalent, eine Belegung, die G wahr macht, muss daher zwangsläufig auch F wahr machen.) □

Verallgemeinertes Minesweeper

Verallgemeinertes Minesweeper

Die folgende Beschreibung verallgemeinert das Spiel *Minesweeper* auf einen ungerichteten Graphen: Sei G ein ungerichteter Graph. Jeder Knoten von G enthält entweder eine einzelne *Mine* oder ist leer. Der Spieler kann eine Knoten wählen. Ist es eine Mine, hat er verloren. Ist der Knoten leer, dann wird dieser mit der Anzahl der direkt benachbarten Knoten beschriftet, die eine Mine enthalten. Der Spieler gewinnt, wenn alle leeren Knoten gewählt wurden. Das Problem ist nun folgendes: Gegeben ein Graph zuzüglich einiger beschrifteter Knoten, ist es möglich Minen so auf die verbleibenden Knoten abzulegen, dass jeder Knoten v, der mit k beschriftet ist, genau k direkt benachbarte Knoten besitzt, die eine Mine enthalten?

Formalisierung

Formalisierung

Zunächst müssen wird das Problem formalisieren. Das Problem ist nur dann schwierig, wenn es mehr freie Felder gibt, als noch Minen zu platzieren sind (sonst ist es leicht in polynomieller Zeit zu lösen). Das Problem kann daher durch ein Tupel (G, f, k) formalisiert werden, wobei G ein ungerichteter Graph ist, $f : V \to \mathbb{N} \cup \{-1\}$ eine Funktion, die jedem Knoten $v \in V$ die Anzahl $f(v)$ der benachbarten Minen zuweist oder die Zahl -1, wenn der Knoten noch leer ist. Die Frage ist nun, ob k Minen so auf jene Knoten v mit $f(v) = -1$ plaziert werden können, dass für jeden Knoten v' mit $f(v') \neq -1$ im Anschluss gilt, dass er gerade $f(v')$ benachbarte Knoten mit einer Mine hat.

Mines ist in NP

Beweis

Man kann leicht sehen, dass dieses Problem in *NP* ist. Ein Zertifikat ist eine Liste der Knoten, auf denen eine Mine platziert werden soll. Ein Verifikationsalgorithmus überprüft dann, ob diese Liste k Knoten enthält. Im Anschluss wird geprüft, ob jeder dieser Knoten von f auf -1 abgebildet wird und zuletzt werden für jeden Knoten mit $f(v) \neq -1$ die direkten Nachbarn gezählt, die eine Mine enthalten, und geprüft, ob dies gerade $f(v)$ viele sind.

Reduktion

Beweis

Zu zeigen, dass das Problem (nachfolgend `Mines` genannt) *NP*-vollständig ist, ist schwieriger. Wir geben hier eine Reduktion von `3SAT` an. Sei dazu F eine Formel mit den atomaren Formeln $A_1, \ldots, A_n$ und den Klauseln $K_1, \ldots, K_m$. Wir konstruieren einen Graphen G und eine Funktion f wie folgt.
Für jedes A_i erstellen wir drei Knoten v_t^i, v_f^i, v^i mit den Kanten $\{v_t^i, v^i\}, \{v^i, v_f^i\}$. Zudem setzen wir $f(v^i) = 1$ und $f(v_t^i) = f(v_f^i) = -1$. (Der Sinn ist, dass genau eine Mine auf v_t^i oder v_f^i gesetzt werden kann und damit ausgesagt wird, ob A_i mit wahr oder falsch belegt wird.)

Reduktion

Beweis

Für jedes $K_j = (L_j^1 \vee L_j^2 \vee L_j^3)$ mit den Literalen L_j^1, L_j^2, L_j^3 erstellen wir weiterhin neun Knoten $u_1^j, u_2^j, u_3^j, \overline{u_1^j}, \overline{u_2^j}, \overline{u_3^j}, u^j, d_1^j, d_2^j$ und die Kanten $\{d_1^j, u^j\}, \{d_2^j, u^j\}, \{u^j, \overline{u_1^j}\}, \{u^j, \overline{u_2^j}\}, \{u^j, \overline{u_3^j}\}, \{\overline{u_1^j}, u_1^j\}, \{\overline{u_2^j}, u_2^j\}, \{\overline{u_3^j}, u_3^j\}$. Ferner wird u_1^j gerade mit v_t^i verbunden, wenn das Literal L_j^1 gerade die atomare Formel A_i ist. Ist hingegen $L_j^1 = \neg A_i$, dann wird u_1^j mit v_f^i verbunden. Entsprechend für u_2^j und u_3^j. Zuletzt werden die $u_1^j, u_2^j u_3^j$ von f mit 1 bewertet, der Knoten u^j mit 2 und alle anderen mit -1. Der Sinn hier ist, dass wenn ein Literal mit wahr belegt wird, dass die Klausel wahr macht gerade eine Mine wie oben beschrieben auf den zugehörigen Knoten gelegt wird. Dadurch bleibt dann mindestens eins der $\overline{u_1^i}$ frei wodurch, die Bedingung von u^j erfüllt werden kann. (Die Hilfskonten d_i^j und d_2^j werden benötigt, da auch mehr als ein Literal in einer Klausel wahr gemacht werden kann.) Als Zahl k für die Anzahl der zu platzierenden Minen setzen wir $k := n + 2m$.

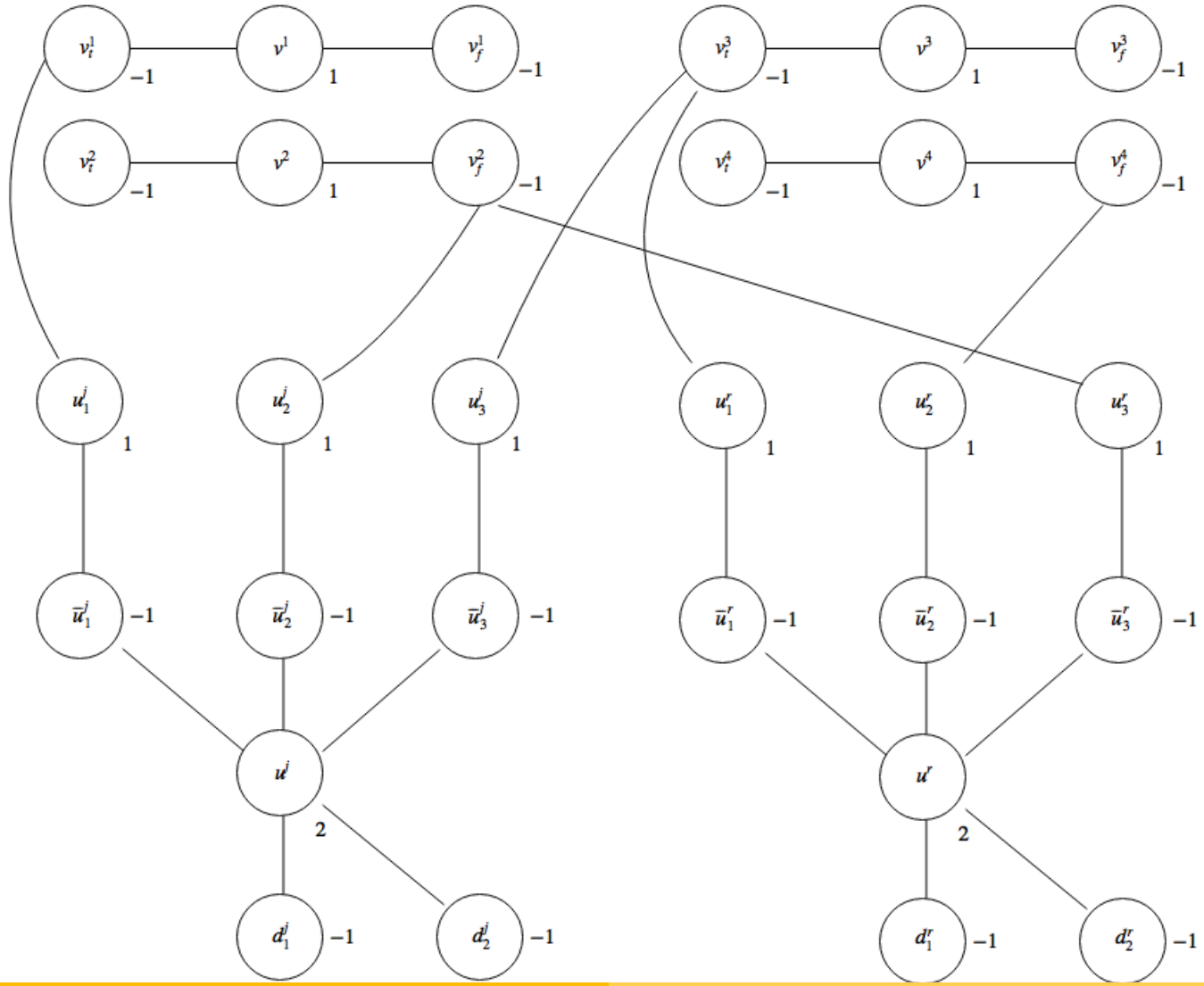
Reduktion

Beweis

Die folgende Abbildung zeigt ein Beispiel Die abgebildeten Knoten oben gehören zu den Variablen $A_1, \ldots, A_4$. Unten im Bild sind die beiden Konstrukte für die Klauseln $K_j = A_1 \vee \neg A_2 \vee A_3$ und $K_r = A_3 \vee \neg A_4 \vee \neg A_2$ zu sehen.

Reduktion auf Mines

Konstruktion für $(A_1 \vee \neg A_2 \vee A_3) \wedge (A_3 \vee \neg A_4 \vee \neg A_2)$



Reduktion

Beweis

Zur Laufzeit. Es werden für die atomaren Formeln $3 \cdot n$ Knoten und $2 \cdot n$ Kanten erzeugt und für die Klauseln $9 \cdot m$ Knoten und $11 \cdot m$ Kanten also insgesamt nur polynomiell in n und k viele Knoten und Kanten. Auch die Speicherung von f benötigt nur $O(n)$ viel Platz. Damit gelingt die Konstruktion in Polynomialzeit.
Es ist noch zu zeigen, dass $F \in \texttt{3SAT}$ genau dann gilt, wenn $(G, f, k) \in \texttt{Mines}$ gilt, wobei (G, f, k) wie oben beschrieben konstruiert werden.

Reduktion

Beweis

Sei zunächst $F \in \texttt{3SAT}$ und wieder mit den atomaren Formeln $A_1, \ldots, A_n$ und den Klauseln $K_1, \ldots, K_m$, wobei $K_j = (L_j^1 \vee L_j^2 \vee L_j^3)$ sei. Insb. ist F nun erfüllbar, d.h. es gibt eine Belegung $\mathcal{A}$, die in jeder Klausel K_j mindestens ein Literal wahr macht. In G können nun Minen wie folgt plaziert werden: Zunächst wird, wenn $\mathcal{A}(A_i) = 1$ ist auf v_t^i eine Mine gelegt, sonst auf v_f^i. Die Anforderung der Knoten v^i sind damit erfüllt und es sind n Minen platziert.
Für jedes Literal L_j^p, dass nun in einem K_j *nicht* wahr gemacht wird, wird zudem auf $\overline{u_p^j}$ eine Mine gelegt. Man beachte, dass von den drei möglichen $\overline{u_p^j}$ genau zwischen 0 und 2 nun so belegt werden, da mindestens ein Literal pro Klausel von $\mathcal{A}$ wahr gemacht werden muss. Nun werden noch die Hilfsplätze d_1^j und d_2^j so mit Minen belegt, dass die Anforderung von u^j erfüllt werden (dass in seiner Nachbarschaft zwei Minen liegen). Damit sind alle Anforderungen erfüllt und es liegen die gewünschten k Minen auf den Knoten von G, also ist $(G, f, k) \in \texttt{Mines}$.

Reduktion

Beweis.

Sei andersherum $(G, f, k) \in \texttt{Mines}$, dann können $k = n + 2m$ Minen in G platziert werden und die Anforderungen von f so erfüllt werden.Zunächst muss nach Konstruktion von G stets genau einer der Knoten v_t^i, v_f^i mit einer Mine belegt werden (da sonst die Anforderungen von v^i nicht erfüllt wären). Wir behaupten, dass die Belegung $\mathcal{A}$, die A_i genau dann zu wahr auswertet, wenn auf v_t^i eine Mine liegt, eine erfüllende Belegung von F ist. Ist es möglich, dass $\mathcal{A}$ eine Klausel K_j nicht erfüllt? Nein, denn das würde nach Konstruktion bedeuten, dass, um die Anforderungen der Knoten u_1^j, u_2^j, u_3^j zu erfüllen, alle Knoten $\overline{u_1^j}, \overline{u_2^j}, \overline{u_3^j}$ mit einer Mine belegt wären, was den Anforderungen von u_j widerspricht. Damit gilt $F \in \texttt{3SAT}$ und wir sind fertig. □

Ausklang

In den letzten Kapiteln und heute haben wir

- *P* eingeführt (effizient lösbare Probleme)
- *NP* eingeführt
- *NPC* eingeführt (sehr wahrscheinlich nicht effizient lösbare Probleme)
- *NPH* eingeführt
- Für *NPC* haben wir zudem Reduktionen eingeführt