Aufgabenblatt 7

Zeigen Sie: $SAT \in P$

Bemerkung. *Um zu zeigen, dass $2SAT \in P$ müssen wir in polynomieller Zeit entscheiden können ob der Ausdruck*

$$\phi = (a_1 \vee b_1) \wedge \cdots \wedge (a_k \vee b_k),$$

wahr ist.

Konstruktion eines Hilfsproblems

Dazu konstruieren wir einen Graphen $G = (V, E)$ mit $|V| = 2L$, wobei jeder Knoten eines der Literale x_i und dazugehörig $\neg x_i$ symbolisiert. (Daher existieren Knoten $x_1, \neg x_1, \ldots, x_L, \neg x_L$). Für jede Klausel $(a \vee b)$, wobei a, b Literale sind, erstellen wir nun eine gerichtete Kante von $\neg a$ zu b und von $\neg b$ zu a da, falls a nicht wahr ist, b wahr sein muss und umgekehrt. Das bedeutet unser Graph G enthält die Kante eine Kante (a, b) genau dann wenn eine Klausel $(\neg a \vee b)$ oder $(b \vee \neg a)$ in ϕ existiert.

Anmerkung. *Sei $a, b \in V(G)$. Falls G einen Weg von a nach b enthält, dann auch einen Weg von $\neg b$ nach $\neg a$.*

Beweis. Folgt sofort aus der Konstruktion von G, da falls es eine Kante $(e_1, e_2) \in E(G)$ existiert, dann ist auch $(\neg e_2, \neg e_1) \in E(G)$ □

Anmerkung. *Eine 2KNF Formel ϕ ist genau dann nicht erfüllbar, wenn eine Variable x existiert, sodass*

1. *ein Weg von x nach $\neg x$ im Graph G existiert.*
2. *ein Weg von $\neg x$ nach x im Graph G existiert.*

Beweis. Angenommen es existiere der Weg von einer Variable x nach $\neg x$ und der Weg von $\neg x$ nach x und eine Zuordnung $Z(x_1, x_2, \ldots, x_L)$ sodass ϕ erfüllbar ist. Wir unterscheiden zwei Fälle:

1. Sei $Z(x_1, \ldots x_L)$, sodass $x = TRUE$ und sei $P = (x, \ldots, y, z, \ldots, \neg x)$ der Weg von x nach $\neg x$.

   Da $x = TRUE$ sind alle Literale auf dem Weg P bis inklusive y $TRUE$, da durch Konstruktion der Kanten in G für den Nachbar $\hat{x}$ von x gelten muss, dass $\hat{x} = TRUE$, da ansonsten $(\neg x \vee \hat{x}) = FALSE$ (Gleiches Argument für Literale bis y). Analog gilt aber für Literale von z bis $\neg x$, dass diese $FALSE$ sein müssen. Das Bedeutet nun wir haben eine Kante $(y, z) \in E(G)$ mit $y = TRUE$ und $z = FALSE$ und somit ist die Klausel $(\neg y \vee z) = FLASE$ und dies ist ein Widerspruch zu unserer Annahme, dass ϕ mit $Z(x_1, \ldots, x_L)$ erfüllbar war.

2. Sei $Z(x_1, \ldots x_L)$, sodass $x = FALSE$ und sei $P = (x, \ldots, y, z, \ldots, \neg x)$ der Weg von x nach $\neg x$.

   Die Argumenation erfolgt hier Analog zu Fall 1 zu dem selben Widerspruch.

□

Conclusio

Wir wissen nun also, dass wir die Entscheidbarkeit einer 2KNF zurückführen darauf zurückführen können, einen Weg im oben Konstruierten Graphen von x nach $\neg x$ bzw umgekehrt zu finden. Der Graph kann offensichtlich in polynomieller Zeit konstruiert werden und die Suche eines solchen Weges kann mithilfe von

$REACH$, wie wir schon gesehen haben, in polynomieller Zeit gefunden werden (Im worst case wird man $REACH$ $2L$ mal ausführen müssen, also $\mathcal{O}(L)$).

Zusammengefasst bedeutet dies, dass $2SAT \in P$, da sich das Problem auf ein Graphenproblem, welches in polynomieller Zeit lösbar ist, in polynomieller Zeit transformieren lässt.

Zeigen Sie: $HORN - SAT \in P$

Bemerkung. *Um zu zeigen, dass $HORN-SAT \in P$, werden wir einen polynomiellen Algorithmus angeben, der für eine Instanz des $HORN - SAT$ Problems also eine Horn-Formel, eine Zuordnung findet, die der Formel den Wert $TRUE$ zuordnet oder falls keine solche Zuordnung existiert, den Wert $FALSE$.*

Anmerkung. *Sei ϕ eine Horn-Formel, die in jeder Klausel zumindest einen nichtnegativen Literal enthält. Dann ist ϕ erfüllbar indem wir jeder Variable den Wert $FALSE$ zuordnen.*

Beweis. Da wir alle Variablen den Wert $FALSE$ zuordenen und in einer Horn-Formel maximal ein positiver Literal enthalten ist, ist jede Klausel erfüllt und damit die gesamte Horn-Formel auch. □

Sei K die Menge aller Klauseln in ϕ. Wir wissen aus der obigen Annahme, dass wenn in jeder Klausel mindestens ein nichtnegativer Literal ist, die gesuchte Zuordnung jene ist, die allen Variablen den Wert $FALSE$ zuweist, ansonsten muss K eine Klausel enthalten, die einzig aus einem positiven Literal p besteht. Eine Zuordnung die nun ϕ erfüllen soll, muss p den Wert $TRUE$ zuweisen und wir können nun alle Klauseln vereinfachen, indem wir jene Klauseln die p enthalten reduzieren und p aus der jeweiligen Klausel entfernen (da wir ja $p = TRUE$ gesetzt haben). Falls eine Klausel $\neg p$ enthält können wir $\neg p$ auch entfernen. So verringert sich die Anzahl der Klauseln in um mindestens eine und wir können rekursiv erneut suchen. Diese Überlegungen legen den folgenden Algorithmus nahe:

```
1:  Z ← ∅
2:  while K nicht leer do
3:      if Jede Klausel in K enthält mindestens einen negativen Literal then
4:          Setze Z sodass jeder Variable in K FALSE zugeordnet wird
5:          return Z
6:      else
7:          if Es existiert eine leere Klausel in K then return Nein
8:          else
9:              Sei k eine Klausel in K, die nur aus einem positiven Literal besteht
10:             Z ← Z ∪ {p = TRUE}
11:             Reduziere K, wie oben besprochen
12: return Z
```

Algorithm 1: HORN-SAT

Wir beweisen nun die Korrektheit des Algorithmus:
Zunächst zeigen wir, dass falls eine Zuordnung existiert, die alle Klauseln erfüllt, der Algorithmus eine von diesen retouniert. Dazu führen wir eine vollständige Induktion über der Größe von K. Die Induktionsbasis ist klar. Im Induktionsschritt unterscheiden wir zwei Fälle:

1. Alle Klauseln in K enthalten mindestens einen negativen Literal. Dann ist laut der Annahme oben die Formel erfüllbar und der Algorithmus retourniert eine korrekte Zuordnung.

2. Es gibt eine Klausel in K, die genau aus einem positiven Literal besteht. Der Algorithmus reduziert in diesem Fall um mindestens eine Klausel und wir können die Induktionsannahme anwenden.

Nun zeigen wir, dass falls ϕ unerfüllbar ist, der Algorithmus 'Nein' zurückliefert - Daher nehmen wir an ϕ sei unerfüllbar. Falls wir eine leere Klausel besitzen liefert der Algorithmus in Zeile 7. ein 'Nein' zurück, ansonsten reduzieren wir unsere Klauseln und erhalten wieder entweder eine leere Klausel oder eine Klausel mit einzig einem positiven Literal. Dieser Prozess terminiert durch die endliche Inputmenge und wir müssen bei einer Menge von Klauseln angelangen, die eine leere Klausel enthält.

Dieser Algorithmus hat polynomielle Laufzeit, da das Suchen nach einer Klausel mit einem negativen Literal und das Suchen nach einer leeren Klausel in polynomieller Zeit möglich ist. Wir rufen weiteres die while-Schleife $\mathcal{O}(|K|)$ mal, also auch polynomiell oft, auf. Insgesamt haben wir einen Algorithmus der $HORN-SAT$ in polynomieller Zeit entscheiden kann und somit folgt $HORN - SAT \in P$

Zeigen Sie: $HORN - SAT \in P - complete$

Aufgabenblatt 8

Zeigen Sie: $\neq SAT$ ist NP-vollständig.

In diesem Beweis, ist zu zeigen, das $\neq SAT$ NP-vollständig ist. Wir wissen, dass $3SAT$ NP-vollständig ist und dürfen annehmen, dass $\neq SAT \in NP$. Daher wollen wir $3SAT$ auf $\neq SAT$ reduzieren und schreiben dafür die jede Klausel $(a_i \vee b_i \vee c_i)$ aus einer Instanz des $3SAT$ in zwei neue Klauseln um (Siehe Angabe):

$$(a_i \vee b_i \vee z_i) \quad \textit{und} \quad (\neg z_i \vee c_i \vee y),$$

wobei für jede Klausel ein eigenes z_i und das y nur einmal neu eingeführt wird.
Wir betrachten nun alle möglichen Fälle für eine Klausel einer $3SAT$ instanz und zeigen, dass es immer möglich ist sie auf eine $\neq SAT$ Instanz zu führen:

Tabelle 1: Fallunterscheidung

a		b		c		a		b		z		$\neg z$		c		y
(1	$\vee$	1	$\vee$	1)	$\rightarrow$	(1	$\vee$	1	$\vee$	0)	$\wedge$	(1	$\vee$	1	$\vee$	0)
(1	$\vee$	1	$\vee$	0)	$\rightarrow$	(1	$\vee$	1	$\vee$	0)	$\wedge$	(1	$\vee$	0	$\vee$	0)
(1	$\vee$	0	$\vee$	1)	$\rightarrow$	(1	$\vee$	0	$\vee$	X)	$\wedge$	(X	$\vee$	1	$\vee$	0)
(0	$\vee$	1	$\vee$	1)	$\rightarrow$	(0	$\vee$	1	$\vee$	X)	$\wedge$	(X	$\vee$	1	$\vee$	0)
(0	$\vee$	0	$\vee$	1)	$\rightarrow$	(0	$\vee$	0	$\vee$	1)	$\wedge$	(0	$\vee$	1	$\vee$	0)
(0	$\vee$	1	$\vee$	0)	$\rightarrow$	(0	$\vee$	1	$\vee$	0)	$\wedge$	(1	$\vee$	0	$\vee$	0)
(1	$\vee$	0	$\vee$	0)	$\rightarrow$	(1	$\vee$	0	$\vee$	0)	$\wedge$	(1	$\vee$	0	$\vee$	0)

Bemerkung.
1. *Der Fall* $(0 \vee 0 \vee 0)$ *liefert kein* $TRUE$ *und ist daher ausgeschlossen.*
2. $X \in \{0, 1\}$ *beliebig.*
3. *Wir sehen, dass* $y = 0$ *für alle Klauseln einheitlich gewählt werden kann.*
4. *In allen 7 Fällen erhielten wir durch die Transformation aus einer* $TRUE$*-Belegung für wieder* $TRUE$*-Belegung in der neuen Instanz.*
5. *Offensichtlich sind nach der Transformation in jeder Klausel mindestens zwei Literale mit unterschiedlicher Ausprägung.*

Die oben vorgeschlagene Transformation aus der Angabe transformiert also tatsächlich eine $3SAT$ Instanz zu einer $\neq SAT$ Instanz und damit kann das $3SAT$ Problem kann auf das $\neq SAT$ Problem reduziert werden, somit ist auch $\neq SAT$ NP-vollständig, da $3SAT \in$ NP-vollständig, $\neq SAT \in NP$ und diese Transformation offensichtlich in polynomieller Zeit möglich ist. □

Zeigen Sie: MAX-CUT ist NP-vollständig.

Wir reduzieren dafür das $\neq 3SAT$ auf $MAXCUT$. Sei eine $\neq 3SAT$ Instanz ϕ gegeben. Wir konstruieren nun einen Graphen (G, c) sodass für jede Variable x_i von ϕ zwei Knoten x_i und $\neg x_i$ eingeführt werden. Weiterhin verbinden wir diese 2 Knoten mit einer Kante und geben ihr die Kapazität $M = c \cdot m$, wobei $c > 3$ und m die Anzahl der Klauseln und n die Anzahl der Variablen in ϕ sei. Weiterhin werden wir für jede Klausel K in ϕ ein Dreieck in G einführen, welches die jeweiligen Variablen aus der Klausel K in G verbindet. Jede solche Kante erhält Kapazität 1.

Anmerkung. *G enthält einen Schnitt mit mindestens* $n \cdot M + 2 \cdot m$ *Kapazität genau dann wenn* ϕ *erfüllbar ist.*

Sei ϕ nun erfüllbar und wir betrachten eine Zuordnung Z die ϕ erfüllt. Teilen wir nun unsere Knotenmenge in zwei Partitionen, nämlich in die der Variablen, die in Z die Ausprägung $TRUE$ haben und den Rest (Also diejenigen Variablen, die den Wert $FALSE$ haben). Offensichtlich ist für alle $i = 1, \ldots, n$ nur entweder x_i oder $\neg x_i$ $TRUE$ und damit haben wir eine Kapazität von $2 \cdot M$. Da Z unser ϕ erfüllt verlaufen klarerweise genau zwei Kanten aus jedem Dreieck zwischen den Partitionen und haben somit eine Kapazität von $2 \cdot m$. Damit haben wir eine Gesamtkapazität des Schnittes von $n \cdot M + 2 \cdot m$.

Für die Rückrichtung betrachten wir einen Schnitt mit einer Kapazität von mindestens $n \cdot M + 2 \cdot m$. Zunächst sehen wir, dass jede der Variablen-Kanten (Also die nicht Dreieck Kanten aus den Klauseln) zwischen dem Schnitt verlaufen muss (bzw. der Schnitt die Kantenmenge so partitioniert, dass die Menge der Variablen-Kanten eine Teilmenge der Menge aller Kanten die zwischen den zwei Partitionen verlaufen ist), da wenn dies nicht der Fall wäre, ein Schnitt dann maximal Kapazität $(n-1)M + 3m = nM + 3m - c \cdot m$ hätte und da $c > 3$ ist, ist dies ein Widerspruch zur Annahme, dass unser Schnitt mindestens Kapazität $n \cdot M + 2 \cdot m$ besitzt.

Als nächstes überlegen wir uns, dass genau zwei der Dreiecks-Kanten aus einer Klausel zwischen dem Schnitt verläuft. Ein Schnitt kann offensichtlich nicht alle 3 Kanten teilen und falls er weniger als 2 Kanten teilt so wird die Kapazität des Schnittes echt kleiner als $n \cdot M + 2 \cdot m$, was wieder einen Widerspruch zur Annahme bringt. Damit, da der Schnitt genau 2 Kanten in jedem Dreieck trennt, erfüllt die zugehörige Variablenbelegung die $\neq -$Bedingung und auch alle Klauseln aus ϕ

Dies bedeutet also wir reduzieren eine Instanz des $\neq SAT$ auf eine Instanz des etwas modifizierten $MAXCUT$ Problems, als in der Angabe, denn in der Angabe wird Schnittkapazität nicht über die Kapazität der Kanten, sondern über die Anzahl der zwischen den Partitionen verlaufenden Kanten definiert. Offensichtlich ist dies nur ein Spezialfall der allgemeineren Definition, in dem einfach jedes Gewicht gleich 1 gesetzt wird. Da wir also die Reduktion auf den allgemeineren Fall gezeigt haben, ist auch der Spezialfall inkludiert.
Offensichtlich ist das zugehörige Entscheidungsproblem von $MAXCUT \in NP$, da wir leicht einen polynomiellen Algorithmus stellen können, der für einen gegebenen Graphen für ein k und für einen Schnitt überprüfen kann, ob er ein Schnitt der Größe k ist. (Einen solchen Schnitt zu finden ist nicht in polynomieller Zeit möglich!)

Letztlich wissen wir also $MAXCUT \in NP$ und wir haben eine NP-Vollständiges Problem gefunden und dieses auf $MAXCUT$ reduziert - Also ist auch $MAXCUT$ ein NP-Vollständiges Problem.

Aufgabenblatt 9

Zeigen Sie, das SOLITAIRE in NP enthalten ist.

Laut dem Blatt 6 ist das Spielfeld für SOLITAIRE mit n Zeilen und m Spalten definiert.
Um zu zeigen, dass SOLITAIRE NP-Vollständig ist, werden wir das 3SAT Problem auf SOLITAIRE reduzieren. Es soll gelten:

$$3SAT \leq_P SOLITAIRE$$

Wir definieren nun ein effizientes Verfahren R, welches eine 3KNF Φ in in ein Solitaire-Spiel G überführt.
Als erstes wird festgelegt, dass jede Zelle vom Spielfeld einer Variable zugewiesen wird, wenn in der Zelle ein Stein vorhanden ist. Weiters legen wir fest, dass ein Literal, welcher dem Wert wahr entspricht, ein roter Stein und negativer Literal ein grüner Stein ist.
Wenn wir also $x_{i,j}$ gegeben hätten, dann wäre das ein roter Stein auf der Zeile i und Spalte j. Grüner Stein wäre dementsprechend $\neg x_{i,j}$ in der gleichen Zelle.
Um das ganze nun in $3SAT$ umzuwandeln definieren wir n Klauseln mit m Variablen für Anzahl von Spalten. Dies würde zur folgenden Form führen:

$$(x_{1,1} \vee x_{1,2} \vee \cdots \vee x_{1,m}) \wedge (x_{2,1} \vee x_{2,2} \vee \cdots \vee x_{2,m}) \wedge \cdots \wedge (x_{n,1} \vee x_{n,2} \vee \cdots \vee x_{n,m})$$

Natürlich muss jede Klausel in $3SAT$ umgeformt werden, um eine valide $3SAT$ zu erhalten.

Aus dem kann man schließen, dass es eine Lösung für $SOLITAIRE$ existiert nur genau dann wenn $3SAT$ eine Lösung hat.