

Aufgabe 1

(a)

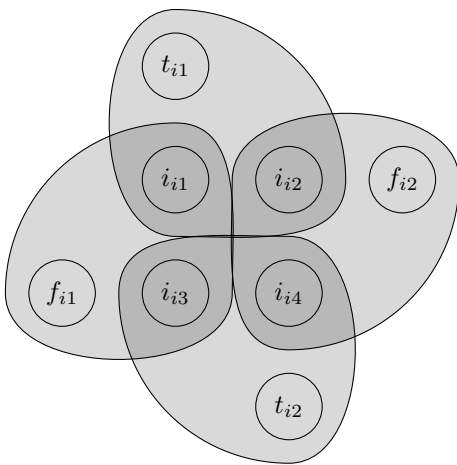
Wir abstrahieren zunächst ein wenig und modellieren einen Hypergraphen, in dem jede Kante zu genau drei Knoten adjazent ist. Anschließend werden wir die Knoten so auf drei Teilmengen verteilen, dass sich Mengen wie für ein 3D-Matching gefordert ergeben und jede Kante einen Knoten in der Teilmenge hat.

Sei eine 3-SAT-Formel φ mit n Variablen x_i und k Klauseln gegeben. $N_+(x_i)$ bezeichne die Anzahl der Vorkommen von x_i in φ , $N_-(x_i)$ entsprechend die Anzahl der Vorkommen von $\neg x_i$. Zudem sei $N(x_i) = \max\{N_+(x_i), N_-(x_i)\}$.

Der gesamte Graph hat zunächst die folgenden Knoten:

$$V = \{t_{i1}, t_{i2}, f_{i1}, f_{i2}, i_{i1}, \dots, i_{i4} \mid i = 1, \dots, n\} \cup \{c_i, c'_i \mid i = 1, \dots, k\} \cup \{d_{i1}, d'_{i1}, d_{i2}, d'_{i2} \mid i = 1, \dots, n\}$$

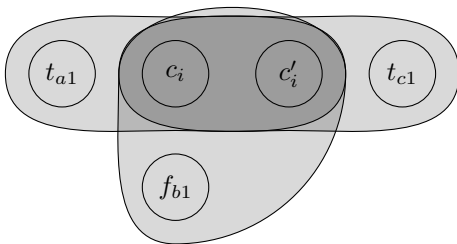
Wir modellieren nun einen Teilgraphen, der für eine Variable x_i aus der 3-SAT-Formel erzwingt, dass x_i entweder wahr oder falsch ist. Hier ist der Graph für $N(x_i) = 2$ gegeben.



Für $N(x_i) = j$ werden allgemein $2 \cdot j$ solcher Dreiecke in einem Ring verbunden, wobei jedes zweite einen t -Knoten (true), die anderen entsprechend f -Knoten (false) erhält.

In einem Maximum Matching können nun entweder alle Kanten mit t -Knoten oder alle Kanten mit f -Kanten gewählt werden. Im ersten Fall ist der Wert der Variablen i falsch im zweiten wahr.

Nun betrachten wir die Klauseln. Sei $c_i = \{x_a, \neg x_b, x_c\}$ eine Klausel ($x_a \vee \neg x_b \vee x_c$). Wir modellieren einen Teilgraphen, der im Falle eines Maximum Matchings erzwingt, dass eine Klausel erfüllt ist.



Die beiden Klauselknoten c_i und c'_i können nur dann abgedeckt werden, wenn eine der drei Kanten gewählt wird, gleichzeitig kann nur eine der drei gewählt werden. Wird eine der drei gewählt, so ist damit gleichzeitig einer der äußeren Knoten des Teilgraphen für die Variable abgedeckt, so dass dort nur die Kanten mit f -Knoten (für a und c) oder t -Knoten (für b) gewählt werden können.

Es bleibt noch zu zeigen, dass die Knoten auf drei disjunkte Teilmengen aufgeteilt werden können, so dass alle Kanten Knoten aus jeder Teilmenge enthalten.

Sei die folgende Aufteilung gegeben:

$$\begin{aligned} X &= \{c_i \mid i = 1, \dots, k\} \cup \{i_{ij} \mid i = 1, \dots, n, j = 1, 3, \dots, 2 \cdot N(x_i) - 1\} \\ Y &= \{c'_i \mid i = 1, \dots, k\} \cup \{i_{ij} \mid i = 1, \dots, n, j = 2, 4, \dots, 2 \cdot N(x_i)\} \\ Z &= \{t_{ij}, f_{ij} \mid i = 1, \dots, n, j = 1, \dots, N(x_i)\} \end{aligned}$$

X, Y, Z bilden eine Partition aller Knoten und jede der oben angegebenen Kanten hat je genau einen Knoten in jeder Teilmenge.

Durch diese Konstruktion können nun genau $k + \sum_{i=1}^n N(x_i)$ Kanten für das Matching gewählt werden.

Ein Maximum Matching auf dem Graphen definiert somit, welche Variable welchen Wahrheitswert hat und definiert für jede Klausel genau ein Literal, das wahr ist.

Anhand einer Lösung für 3-SAT wiederum können sofort die entsprechenden Kanten gewählt werden: Ist eine Variable wahr, werden die f -Knoten gewählt und andersrum, für jede Klausel wird eine der Literale mit dem Wahrheitswert true ausgewählt.

(b)

Aufgabe 2

(a)

Sei φ eine Formel in 3-KNF mit Literalen L und k Klauseln und habe die Form

$$\varphi = (x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \dots$$

Dabei sind $x_{ij} \in \{l, \neg l \mid l \in L\}$.

Sei nun $G(\varphi) = (V, E, K = 2k + |L|)$ ein Graph mit

$$V = \{l, \neg l \mid L \in L\} \cup \{x_{ij}\}$$

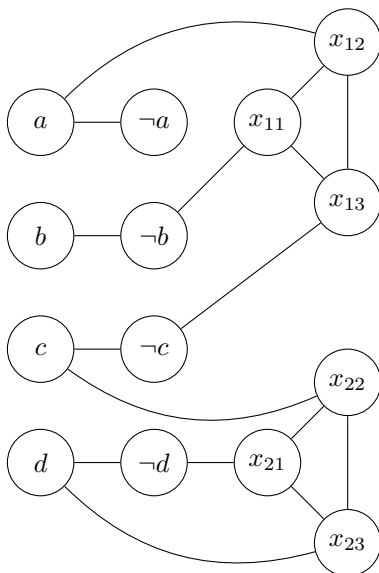
– wobei die Knoten aus L *Literalknoten* und die Knoten x_{ij} *Klauselknoten* heißen – und

$$E = \{(x_{i1}, x_{i2}), (x_{i1}, x_{i3}), (x_{i2}, x_{i3}) \mid i = 1 \dots k\} \cup \{(L(x_{ij}), x_{ij})\}$$

wobei $L(x_{ij})$ das Literal (oder die Negation des Literals) von x_{ij} darstellt. Die i Dreiecke aus Klauselknoten nennen wir *Cluster*.

Auf $G(\varphi)$ muss ein VertexCover der Größe K gefunden werden.

Für die Formel $\varphi = (\neg b \vee a \vee \neg c) \wedge (\neg d \vee c \vee d)$ ergibt sich also der Graph



Behauptung: **VertexCover** hat auf $G(\varphi)$ genau dann eine Lösung, wenn φ erfüllbar ist (3-SAT hat eine Lösung).

Beweis:

\Leftarrow :

Wir konstruieren ein Vertex Cover $W \subseteq V$ der Größe K . Seien L_1 die Literale, denen in der Lösung von 3-Sat der Wahrheitswert wahr zugewiesen wird. Dann sind alle $l \in L_1$ auch in W . Wähle nun aus jeder 3-Clique, die eine Klausel beschreibt, zwei beliebige Knoten, so dass der verbleibende mit einem bereits markierten Literal-Knoten verbunden ist. So ein Knoten muss existieren, da mindestens ein Literal (oder eine Negation eines Literals) in einer Klausel wahr sein muss, ein Knoten also mit einem bereits markierten Knoten verbunden sein muss.

\Rightarrow :

Beobachtung: Jedes Vertex Cover in $G(\varphi)$ muss mindestens einen Literalknoten jedes Literals sowie mindestens zwei Klauselknoten enthalten. Ein Vertex Cover der Größe $K = 2k + |L|$ enthält somit genau einen Literalknoten jedes Literals sowie genau zwei Klauselknoten jedes Dreiecks. Seien nun L_1 die Literalknoten des Vertex Covers und wie in \Leftarrow die Menge der Literale mit dem Wahrheitswert wahr. Die entsprechende Formel φ ist dann erfüllbar, da jedem Literal ein Wahrheitswert zugewiesen ist und jede Klausel erfüllbar ist: Jede Klausel enthält genau einen Knoten, der nicht im Vertex Cover enthalten ist, dessen Kante mit einem Literalknoten aber abgedeckt ist. Der inzidente Literalknoten muss daher im Vertex Cover enthalten sein, das zum Klauselknoten gehörende Literal also wahr sein. \square

(b)

Behauptung: $C \subseteq V$ ist genau dann ein **VertexCover**, wenn $I = V \setminus C$ ein **IndependentSet** ist.

Aus der Behauptung folgt sofort die Äquivalenz der Problem hinsichtlich NP-Vollständigkeit.

Beweis:

\Rightarrow :

Ist $C \subseteq V$ ein Vertex Cover, so sind alle Kanten des Graphen zu mindestens einem Knoten in C adjazent. Somit gibt es keine Kante, die zu zwei Knoten aus $I = V \setminus C$ adjazent ist. Also gibt es keine zwei inzidenten Knoten in I .

\Leftarrow :

Ist $I \subseteq V$ ein Independent Set, so existiert keine Kante im Graphen zwischen zwei Knoten in I . Jede Kante ist somit zu mindestens einem Knoten in $C = V \setminus I$ adjazent. Also deckt C alle Kanten ab. \square

Aufgabe 3

Betrachte den Lauf für eine beliebige Eingabe der Länge n . Sei α_i mit $i \in \mathbb{Z}$ das Zeichen an der i ten Bandposition. Sei δ_k die k te Transitionsregel. Der Unterstrich markiert die aktuelle Bandposition.

Die TM startet in q_0 auf einer beliebigen Eingabe

$q_0 _ \$ \text{INPUT} \$ _$

Solange das aktuelle Zeichen kein Blank-Symbol ist, geht die TM nach rechts, ohne etwas zu verändern (δ_1, δ_2).

Hat die TM das Ende des Wortes erreicht, so erscheint das erste Blanksymbol:

$_ \$ \text{INPUT} \$ q_0 _$

Die TM wechselt nun nach q_1 , schreibt ein # und geht nach rechts:

$_ \$ \text{INPUT} \$ \# q_1 _$

Ein Blank: Sie wechselt nach q_2 , schreibt eine 1 und geht nach rechts:

$_ \$ \text{INPUT} \$ \# 1 q_2 _$

Ein Blank: Sie bleibt in q_2 und schreibt eine 0:

$_ \$ \text{INPUT} \$ \# 1 q_2 0$

Eine 0: Sie wechselt nach q_3 und geht nach rechts:

⌊\$INPUT\$#10 q_3 ⌊

Ein Blank: Sie schreibt eine 1, wechselt nach q_4 und geht nach rechts:

⌊\$INPUT\$#101 q_4 ⌊

Ein Blank: Sie wechselt nach q_1 , schreibt eine 0 und geht nach links:

⌊\$INPUT\$#10 q_1 10⌊

Eine 1: Sie geht nach rechts:

⌊\$INPUT\$#101 q_1 0⌊

Eine 0: Sie geht nach rechts und wechselt nach q_5 :

⌊\$INPUT\$#1010 q_5 ⌊

Ein Blank: Sie schreibt eine 1:

⌊\$INPUT\$#1010 q_5 1

Eine 1: Sie wechselt nach q_6 und geht nach rechts:

⌊\$INPUT\$#10101 q_6 ⌊

Ein Blank: Sie wechselt nach q_3 und schreibt eine 0:

⌊\$INPUT\$#10101 q_3 0

Eine 0: Sie wechselt nach q_F und geht nach rechts:

⌊\$INPUT\$#101010 q_F ⌊

Die Endkonfiguration der TM ist also für jedes Eingabewort die folgende: Sie endet im Zustand q_F , an die Eingabe wurden die Zeichen #101010 angehängt und der Lesekopf befindet sich beim ersten Blank hinter diesen Zeichen.

Aufgabe 4

Sei $G = (V, E)$ der Graph mit $V = \{v_i \mid i = 1 \dots n\}$ und $W = \sum_{i=1}^n w_i$.

Beobachtungen:

Ist G ein Pfad mit n Knoten, so gibt es nur $n - 2$ (sinnvolle) Knoten, die in V' enthalten sein können: Alle bis auf die Endknoten.

Das Partitionieren entspricht dem Aufteilen des Pfades in $(k + 1)$ Abschnitte.

Weiterhin kann ein zusätzlicher Knoten in V' nie von Nachteil sein, o.B.d.A. wird V' also stets k Knoten enthalten.

Algorithmus: Seien $x_j \in \{1, \dots, n\}$ die k Positionen der Knoten in V' und bezeichne $sum(i)$ die Summe der Kantengewichte aller Kanten im i ten Abschnitt. Ein Abschnitt (oder seine Summe) heißt *maximal*, falls kein Abschnitt Kanten mit größerem Gesamtgewicht hat.

```

1  $x_i = i$  # fuer alle  $i$ 
2 while  $sum(1)$  nicht maximal:
3      $j =$  maximaler Abschnitt
4     if  $x_j - x_{j-1} == 1$ :
5         return  $x_i$ 
6      $x_{j-1} = x_{j-1} + 1$ 
7     compare( $x_i$ )

```

Bei der ersten Partitionierung sind die ersten k Abschnitt minimal (je nur eine Kante), der Rest ist im letzten Abschnitt enthalten.

Nun wird der maximale Abschnitt verkleinert, indem die erste Kante dieses Abschnitts in den vorherigen Abschnitt verschoben wird. Hat der maximale Abschnitt nur eine Kante, so ist die aktuelle Partitionierung optimal: Kleiner als die größte Kante kann es nicht werden.

Sobald der erste Abschnitt maximal ist, kann die Partitionierung nicht mehr besser werden. Die beste Partitionierung wurde somit bereits betrachtet und ist unter den mittels **compare** miteinander verglichenen.

Da jeder der $k < n$ Schnitte maximal $n - k$ mal verschoben werden kann, werden maximal $k \cdot (n - k) < n^2$ Schritte durchgeführt.

Die optimale Partitionierung muss betrachtet worden sein, da in jedem Schritt versucht wird, die Zielfunktion – den Wert des maximalen Abschnitts – zu verkleinern und am Ende eine besserer Zielwert nicht mehr erreicht werden kann.