Universität Paderborn
Prof. Dr. Johannes Blömer

Gruppe 27:
Eli Kogan-Wang (7251030)
David Noah Stamm (7249709)
Bogdan Rerich (7248483)
Jan Schreiber(7253698)

Berechenbarkeit und Komplexität – WS 2022/2023

Heimübung 11
Abgabe: 23. Januar 2023 – 13:00 Uhr
(Dieser Übungszettel besteht aus 4 Aufgaben mit insgesamt 24 Punkten)

Aufgabe 1 (P vs NP) (6 Punkte)
Zeigen Sie, dass die folgende Variante von SubsetSum namens BoundedSubsetSum in P liegt:

$$\left\{ \langle (S,W),t\rangle \;\middle|\; \begin{array}{l} S = \{s_1,\ldots,s_m\} \subseteq \mathbb{N},\ \text{mit } s_i \leq 10 \text{ für alle } i, \text{ Funktion } W: S \to \mathbb{N} \text{ und} \\ \text{es existiert Teilmenge } T \subseteq S \text{ und } (x_s)_{s\in T} \in \mathbb{N}^{|T|} \text{ mit } x_s \leq W(s), \\ \text{sodass } \sum_{s\in T} x_s \cdot s = t \end{array} \right\}$$

Hinweis: Um formal korrekt abzubilden, dass Elemente aus S mehrfach enthalten sein können, wurde das Problem im Stile einer Multimenge modelliert. Die Basismenge ist S und die Funktion W bildet ab, wie oft jedes Element $s \in S$ in der Multimenge enthalten ist. Die Bedingung über die Faktoren x fordert, dass ein Element $s \in T$ maximal $W(s)$ mal benutzt wird um das Ergebnis t zu erhalten.

Lösung: Es wird angenommen, dass die Summe $W(s_1) + \cdots + W(s_m)$ polynomiell in der Kodierung von (S, W) ist.
Wir beschreiben einen Algorithmus, welcher BoundedSubsetSum in P löst.
Seien $S = \{s_1,\ldots,s_m\} \subseteq \mathbb{N}$ und $W: S \to \mathbb{N}$ und $t \in \mathbb{N}$ gegeben.
Wir definieren einen Zustand (i,j) mit $i \in \{0,\ldots,m\}$ und $j \in \mathbb{N}$.
Dieser Zustand beschreibt, dass eine Teilmenge von $S' = \{s_1,\ldots,s_i\}$ und $(x_s)_{s\in S'} \in \mathbb{N}^{|S'|}$ mit $x_s \leq W(s)$ existiert, sodass $\sum_{s\in S'} x_s \cdot s = j$.
Jeder Zustand (i,j) zeigt auf weitere Zustände wie folgt:

- $(i+1, j)$, also man kann (mit beachtung von s_{i+1}) die gleiche Summe erreichen
- $(i+1, j + k \cdot s_{i+1})$, für alle $k \in \{1,\ldots,W(s_{i+1})\}$

Vom Startzustand aus verwenden wir eine Breitensuche, um alle Zustände zu erreichen. Wenn wir den Zielzustand (m,t) erreichen, dann ist die Lösung gefunden.
Die Komplexität der Breitensuche (worst-case) ist $O(|V| + |E|) = O(l + l^2) = O(l^2)$, wobei l die Anzahl der Zustände ist.
Die erreichbaren Summen t, sind teil der Zahlen von 0 bis $10 \cdot (W(s_1) + \cdots + W(s_m))$.
Also ist die Anzahl der Zustände $l = O(m \cdot t) = O(m \cdot 10 \cdot (W(s_1) + \cdots + W(s_m)))$.
Nun hängt die Komplexität von den Werten $W(s_i)$ ab.
Da die Summe $W(s_1) + \cdots + W(s_m)$ polynomiell in der Kodierung von (S, W) ist, dann ist die Komplexität des Algorithmus quadratisch in der Kodierung von (S, W), also BoundedSubsetSum ist in P.

Aufgabe 2 (Approximationsalgorithmus) (6 Punkte)
Sei $G = (V, E)$ ein ungerichteter Graph. Eine Partition von V in drei Mengen S_1, S_2, S_3 nennen wir einen 3-Cut von G. Ferner bezeichnet $w(S_1, S_2, S_3)$ die Anzahl der Kanten von G, deren Endpunkte in unterschiedlichen Mengen S_i liegen. Wir nennen $w(S_1, S_2, S_3)$ das Gewicht des 3-Cut S_1, S_2, S_3.
Beim Max-3-Cut-Problem ist für einen gegebenen, ungerichteten Graphen $G = (V, E)$ ein 3-Cut gesucht, dessen Gewicht maximal unter allen 3-Cuts von G ist. Betrachten Sie den folgenden Approximationsalgorithmus für das Max-3-Cut-Problem.
Zeigen Sie, dass der Algorithmus APPROX3CUT einen Approximationsfaktor von $\frac{2}{3}$ erreicht. Das heiSSt, der Algorithmus berechnet für jeden Graphen einen 3-Cut, dessen Gewicht mindestens $\frac{2}{3}$ des Gewichts einer Partition mit maximalem Gewicht beträgt.

Algorithm 1 APPROX3CUT(G) mit Eingabe $G = (V, E)$

$S_1 \leftarrow V, S_2 \leftarrow \emptyset, S_3 \leftarrow \emptyset$;
while Es existiert Permutation $\pi : \{1, 2, 3\} \rightarrow \{1, 2, 3\}$ und ein $v \in S_{\pi(1)}$,
sodass $w(S_1, S_2, S_3) < w(S_{\pi(1)} \setminus \{v\}, S_{\pi(2)} \cup \{v\}, S_{\pi(3)})$ **do**
$S_{\pi(1)} \leftarrow S_{\pi(1)} \setminus \{v\}$;
$S_{\pi(2)} \leftarrow S_{\pi(2)} \cup \{v\}$;
end while
return S_1, S_2, S_3

Hinweis: Beweisen Sie, dass der Algorithmus nicht hält, solange es noch ein $i \in \{1, 2, 3\}$ und ein $v \in S_i$ gibt, sodass mehr als $\frac{1}{3}$ der Nachbarn von v ebenfalls in S_i enthalten sind. Beweisen Sie auch, dass der Algorithmus in Polynomialzeit terminiert.

Lösung:
Behauptung:

Approx3Cut(G) ist polynomiell bezüglich der Koodierung von G.

Die While Schleife terminiert bezüglich G in polynomiell Zeit, da in jedem Interationsschritt das Gewicht des 3 Cuts um 1 erhöht wird. Dieses kann maximal |E| betragen. Folglich muss die Schleife nach Endlich vielen Schritt (maximal |E|) terminieren. AuSSerdem gibt es insgesamt 3! Verschiedene Permutationen für π, da dieses aus S_3 kommt. Da diese Permututionen für jeden Knoten überprüft werden müssen, da die while schleife eine Laufzeit von 6*$|V| \cdot |E| \in \mathcal{O}(|V| \cdot |E|)$. Da dieses polynoiell bezüglich G ist und alle anderen Schritte des Algos eine Konstante Laufzeit haben, folgt die Polynomilität des Algo.

Behauptung: (1)
Der Algorithmus hält nicht, solange es noch ein $i \in \{1, 2, 3\}$ und ein $v \in S_i$ gibt, sodass mehr als $\frac{1}{3}$ der Nachbarn von v ebenfalls in S_i enthalten sind.

Beweis durch Widerspruch.

Angenommen der Algo. würde halten, obwohl eine solches v, wie in der Annahme existieren würde.

o.B.d.A Sei $i = 1$, v in der Menge S_1 enthalten und $N_1(v)$:= $\frac{\text{Anzahl der Nachbarn von v in } S_1}{\text{Anzahl der Nachbarn von v}}$.

Nach Voraussetzung gilt: $N_1(v) > \frac{1}{3}$.

$\implies N_2(v) + N_3(v) < \frac{2}{3} \implies N_2(v) < \frac{1}{3} \vee N_3(v) < \frac{1}{3}$.

Gelte dies o.B.d.A. für S_2.

$\implies w(S_1, S_2, S_3) < w(S_1 \setminus \{v\}, S_2 \cup \{v\}, S_3) = w(S_1, S_2, S_3) + N_1(v) - N_2(v)$.

Folglich kann der Algorithmus noch nicht gehalten haben, da eine weitere Iteration der While-Schleife ausgeführt werden kann. HIER BLITZ EINFÜGEN. $\square$

Also gilt $\forall v \in V$, dass $N_i(v) \leq \frac{1}{3} \iff w(S_1, S_2, S_3) \geq \frac{2}{3}$.
Behauptung:
Der Approx3Cut hat eine Approximationsfaktor von $\frac{2}{3}$

Aus (1) folgt:

Approx3Cut(G)= w(S) $\geq \frac{2 \cdot |E|}{3} \geq \frac{2 \cdot \text{opt(G)}}{3}$

$\iff \frac{\text{Approx3Cut(G)}}{\text{opt(G)}} \geq \frac{2}{3}$

Aufgabe 3 (Rekursive Aufzählbarkeit und Entscheidbarkeit – Klausuraufgabe 18/19 + 20/21) (6 Punkte)

a) *Beweisen* Sie, ob folgende Sprache rekursiv aufzählbar oder, durch geeignete Reduktion, nicht rekursiv aufzählbar ist

$L_a = \{\langle M \rangle x \mid \text{DTM } M \text{ gestartet mit } x \text{ erreicht eine Konfiguration mit leerem Band.}\}$

b) *Beweisen* Sie, ob folgende Sprache entscheidbar oder, durch geeignete Reduktion, nicht entscheidbar ist

$$L_b = \left\{\langle M \rangle x \;\middle|\; \text{DTM } M \text{ berechnet bei Eingabe } x \text{ die Binärdarstellung von } |x|^2.\right\}$$

Lösung:
a)

Sei im folgenden M_x eine Turingmaschine:

M_x bei Eingabe w $\in \{0,1\}^*$

1. (Formcheck) Prüfe ob w = $\langle M \rangle$x, wobei M die Gödelisierung einer Turingmaschine ist und x $\in \{0,1\}^*$.

2. Widerhole für i=1,2,3,....
   a) Simuliere M bei Eingabe x für i Schritte. Prüfe nach jedem Schritt, ob das Band leer ist.
   b) Falls dies in einem Schritt der Fall ist, so akzeptiere.

Behauptung: L(M_x)=L_a.

Angenommen w $\in L_a$

$\implies$ w = $\langle M \rangle$ x und die Turingmaschine erreicht eine Konfiguration mit einem leeren Band.

$\implies$ w wird in (1) nicht abgelehnt und da in 2 a) durch die Simulation alle möglichen Konfigurationen irgendwann erreicht folglich auch die mit einem leeren Band und die Turingmaschine hält nach diesem.
$\implies$ w $\in$ L(M)
Angenommen w $\notin L_a$.
$\implies$ Entweder w $\neq \langle M \rangle$x, wird also in (1) abgelehnt oder w = $\langle M \rangle$x, aber in keiner Konfiguration wird ein leeres Band erreicht.

Folglich wird w in (1) nicht angelehnt, aber es kann in 2a) keine Konfiguration gefunden werden, in der ein leeres Band erreicht wird.

$\implies$ w wird in jedem Iterationsschritt abgelehnt.
$\implies$ w $\notin$ L(M)
$\implies$ L(M)=L_a $\square$.
Da M_x offensichtlicher Weise rekursiv aufzählbar ist, da falls M bei Eingabe x in eine Endlosschleife kommt, auch M_x in eine Endlosschleife kommt.
$\implies$ L_a ist rekursiv aufzählbar.

---

b)

Sei

$$L_H = \{\langle M \rangle x \mid M \text{ ist eine Turingmaschine die bei Eingabe } x \text{ hält } \}$$

Das Halteproblem, welches bekanntlich nicht entscheidbar ist.

Wir zeigen: $L_H \leq L_b$

Zuerst die Reduktionsfunktion $f : \{0,1\}^* \to \{0,1\}^*$.

Sei $w \in \{0,1\}^*$, aber keine Kodierung $\langle M \rangle x$, mit M eine Gödelisierung einer Turingmaschine und $x \in \{0,1\}^*$.

Dann ist $f(w) := w$, was auch keine Gödelisierung einer Turingmaschine ist.

Sei $w = \langle M \rangle x$ mit M eine Gödelisierung einer Turingmaschine und $x \in \{0,1\}^*$.

Nun ist $f(w)$ eine Turingmaschine M', die bei Eingabe y wie folgt verhält:

1. Simuliere M bei Eingabe x, bis zur Terminierung.
2. Schreibe die Binärdarstellung von $|y|^2$ auf das Band.

Wir zeigen: f ist eine Reduktion von L_H nach L_b.

Angenommen $w \in L_H$.

Also $w = \langle M \rangle x$ mit M eine Gödelisierung einer Turingmaschine und $x \in \{0,1\}^*$.

Da $w \in L_H$ hält M bei Eingabe x.

$\implies f(w) = \langle M' \rangle$.
Bei Eingabe y wird M', die Binärdarstellung von $|y|^2$ auf das Band schreiben, da M bei Eingabe x hält.

Also ist $f(w) \in L_b$.
Angebommen $w \notin L_H$.

Wenn $w \notin L_H$ ist w keine Kodierung $\langle M \rangle x$, mit M eine Gödelisierung einer Turingmaschine und $x \in \{0,1\}^*$.

$\implies f(w) = w$.
Also ist $f(w) \notin L_b$, da $f(w)$ keine Gödelisierung einer Turingmaschine ist.
Wenn $w \notin L_H$ ist w eine Kodierung $\langle M \rangle x$, mit M eine Gödelisierung einer Turingmaschine und $x \in \{0,1\}^*$.

Da $w \notin L_H$ hält M nicht bei Eingabe x.

$\implies f(w) = \langle M' \rangle$.
Bei Eingabe y wird M' nicht halten, da M nicht bei Eingabe x hält.

Also wird M' nicht die Binärdarstellung von $|y|^2$ auf das Band schreiben.

Also ist $f(w) \notin L_b$.
Damit ist f eine Reduktion von L_H nach L_b.

$\implies L_H \leq L_b$.
Da das Halteproblem nicht entscheidbar ist, ist L_b nicht entscheidbar.

Aufgabe 4 (NP-Vollständigkeit – Klausuraufgabe 20/21) (6 Punkte)

Betrachten Sie die folgende Sprache.

$$\textsc{Max-SAT} := \left\{ \langle \phi, k \rangle \;\middle|\; \begin{array}{l} \phi \text{ ist eine Boolsche Formel in KNF, } k \in \mathbb{N} \text{ und es} \\ \text{existiert eine Belegung der Variablen, sodass} \\ \text{mindestens } k \text{ Klauseln von } \phi \text{ erfüllt sind.} \end{array} \right\}$$

Zeigen Sie, dass die Sprache Max-SAT NP-vollständig ist. Nutzen Sie für Ihre Reduktion eine der drei folgenden NP-vollständigen Sprachen

a) SAT = $\{\langle \phi \rangle \mid \phi$ ist eine erfüllbare Boolesche Formel in KNF.$\}$

b) 3SAT = $\{\langle \phi \rangle \mid \phi$ ist eine erfüllbare Boolesche Formel in 3-KNF.$\}$

c) VertexCover = $\{\langle G, k \rangle \mid G$ ist ein ungerichteter Graph mit einer k-Knotenüberdeckung.$\}$

Lösung: Es ist folgendes zu Zeigen:

1. Max-Sat $\in$ NP
2. SAT $\leq_p$ Max-Sat

Zu (1):
Betrachte folgende Turingmaschine:
V bei Eingabe x $\in \{0,1\}^*$:

1. (Formcheck) Prüfe, ob x = $\langle \phi, k, B \rangle$ eine Kodierung ist, wobei ϕ eine boolesche Formel in KNF ist, k $\in \mathbb{N}$ und B eine Belegung von ϕ
2. Prüfe, ob B mindestens k Klauseln der KNF erfüllt. Falls dies der Fall ist so lehne x ab.
3. Akzeptiere.

BH.: V ist Polynomiell

1. Der Formcheck kann trivialerweise in polynomieller Zeit durchgeführt werden.
2. Da hierbei nur maximal die Anzahl der Klauseln in ϕ überprüft werden müssen, ist dies in poly. Zeit bezüglich ϕ möglich.
3. Trivialerweise in Polynomieller Zeit möglich

Da B eine Belegung von ϕ, welche jedem Literal aus diesem einen Wahrheitswert zuordnet, hat dieses bezüglich ϕ eine polynomielle Länge.

BH.: V ist Verifzierer von Max-SAT

Angenommen $\langle \phi, k \rangle \in$ Max-SAT. d.h. es existiert eine Belegung B von ϕ, welche k Klauseln erfüllt.

$\implies$ V akzeptiert x=$\langle \phi, k, B \rangle$ nach (3).

$\implies$ x $\in L(v)$

Angenommen $\langle \phi, k \rangle \notin$ Max-SAT

Beweis durch Widerspruch:

Angenommen es würde eine Belegung B von ϕ geben s.d.

x = $\langle \phi, k, B \rangle \in L(V)$
$\implies$ (Nach (3)) $\langle \phi, k \rangle \in$ Max-SAT WIEDERSPRUCHS BLITZ EINFÜGEN.
$\implies$ V ist ein Verifizierer von Max-SAT.
$\implies$ Max-SAT $\in NP$

Zu (2):

Behauptung: **SAT** $\leq_p$ **Max-Sat**.

Betrachte folgende Reduktionsfunktion:

$$f(w) = \begin{cases} \langle \phi, k \rangle & \text{wenn w} = \langle \phi \rangle\text{,wobei } \phi \text{ die Koodierung einer KNF ist} \\ & \text{und k die Anzahl ihrer Klauseln} \\ \delta & \text{wenn w} \neq \langle \phi \rangle \text{ sonst} \end{cases}$$

Wobei δ eine Formel ist, welche nicht in KNF ist.

Die Laufzeit von f(w) ist trivialerweise polynomiell und berechenbar,da diese nur die Anzahl der Klauseln von ϕ bestimmt.

Behauptung: w $\in$ SAT $\iff$ f(w) $\in$ Max-Sat

Angenommen w $\in$ SAT

$\implies$ w wird unter f auf $\langle \phi, k \rangle$ abgebildet und es existiert eine Erfüllende Belegung B von ϕ.

$\implies$ Folglich muss B jede Klauseln in ϕ erfüllen (,da KNF) und B erfüllt damit k Klauseln.
$\implies$ f(w) $\in$ Max-Sat.

Angenommen w $\notin$ Sat.

Also ist w entweder von der flasche Form, wird dann also auf δ abgeldet, welches nicht in Max-Sat ist, da nicht in KNF oder auf $\langle \phi, k \rangle$

Angommen f(w) $\in$ Max-Sat
$\implies$ Es existiert eine Belegung, welche K klauseln erfüllen würde. Nach wahl von k ist diese Belegung dann auch eine erfüllende Belegung von ϕ

$\implies$ w $\in$ SAT WIDERSPRUCHSPFEIL EINFÜGEN.
$\implies$ w $\in$ SAT $\iff$ f(w) $\in$ Max-Sat

$\implies$ **SAT** $\leq_p$ **Max-Sat**.
$\xrightarrow{1,2}$ Max Sat ist NP vollständig. $\square$