Institut für Theoretische Informatik
Lehrstuhl Prof. Dr. D. Wagner

Übungsblatt 5

Vorlesung Theoretische Grundlagen der Informatik im WS 19/20

Ausgabe: 17. Dezember 2019
Abgabe: 14. Januar 2020, 11:00 Uhr (im Kasten im UG von Gebäude 50.34)

Aufgabe 1

(1 + 2 = 3 Punkte)

(a) Sei Π ein $\mathcal{NP}$-vollständiges Problem, zu dem ein pseudopolynomialer Algorithmus existiert. Warum impliziert dies nicht die Existenz eines pseudopolynomialen Algorithmus für jedes $\mathcal{NP}$-vollständige Problem?

Lösung:
Bei der polynomialen Transformation einer Eingabeinstanz I in eine Instanz I' beschränken wir die Länge $|I'|$ polynomial in der Länge von I. Dies impliziert allerdings nicht, dass auch die größte Zahl $\max(I')$ aus I' polynomial durch I' beschränkt ist. Dies führt unmittelbar dazu, dass die Unärkodierung der Instanz I' nicht polynomial beschränkt ist in der Unärkodierung von I.

(b) Zeigen Sie, dass ein stark $\mathcal{NP}$-vollständiges Problem genau dann von einem pseudopolynomialen Algorithmus entschieden wird, wenn $\mathcal{P} = \mathcal{NP}$ gilt.

Lösung:
Angenommen, es existiert ein pseudopolynomialer Algorithmus $\mathcal{A}$ für ein stark $\mathcal{NP}$-vollständiges Problem. Dann ist die Anzahl der Bits in der Unärkodierung durch ein Polynom beschränkt, da $\max(I)$ polynomial beschränkt ist. Mit Hilfe von $\mathcal{A}$ kann somit jedes Problem in $\mathcal{NP}$ in Polynomialzeit entschieden werden.

Angenommen, es gilt $\mathcal{P} = \mathcal{NP}$. Die stark $\mathcal{NP}$-vollständigen Probleme sind eine Teilmenge der $\mathcal{NP}$-vollständigen Probleme. Es existiert also ein Algorithmus $\mathcal{A}$, der polynomiale Laufzeit in n benötigt. Es bleibt zu zeigen, dass $\mathcal{A}$ auch pseudopolynomial ist. Sei die Zahl n die Anzahl der Bits in der Binärkodierung (oder ein Kodierungsschema, das polynomial transformierbar in die Binärkodierung ist). Die Anzahl der Bits N einer Unärkodierung ist somit $N \in \Theta(2^n)$. Angenommen, der Algorithmus ist nicht polynomial in N, dann folgt sofort ein Widerspruch zur polynomialen Laufzeit in n. Jeder Polynomialzeitalgorithmus ist also auch ein pseudopolynomialer Algorithmus.

Aufgabe 2

(2 + 2 = 4 Punkte)

Gegeben sei ein ungewichteter, ungerichteter Graph $G = (V, E)$. Für zwei Knoten $u, v \in V$ bezeichnen wir mit $\texttt{dist}(u, v)$ die Länge des kürzesten Pfads zwischen u und v in G. Der *Durchmesser* von G ist definiert als $D(G) = \max_{u,v \in V} \texttt{dist}(u, v)$. Also ist der Durchmesser die Länge des längsten kürzesten Pfads in G.

(a) Beschreiben Sie, wie der Durchmesser eines Graphen in Polynomialzeit berechnet werden kann.

(b) Zeigen Sie, dass in Zeit $\mathcal{O}(|V|+|E|)$ eine 2-Approximation für den Durchmesser eines Graphen berechnet werden kann.

Lösung:

(a) Für einen Knoten $u \in V$ kann mit einer Breitensuche $\texttt{dist}(u, v)$ für jeden anderen Knoten $v \in V$ berechnet werden. Dies benötigt Zeit $\mathcal{O}(|V| + |E|)$. Führe dies für jeden Knoten in V und bilde das Maximum über alle gefundenen Distanzen. Dies benötigt insgesamt Zeit $\mathcal{O}(|V|^2 + |V||E|)$.

(b) Betrachte folgenden Algorithmus $\mathcal{A}$: Wähle einen beliebigen Knoten $u \in V$ aus und führe von diesem eine Breitensuche aus. Gebe dann $\mathcal{A}(G) = \max_{v \in V} \texttt{dist}(u, v)$ aus.

Seien $x, y \in V$ die Knoten, für die $\texttt{dist}(x, y) = D(G)$ gilt. Wegen der Dreiecksungleichung gilt: $D(G) = \texttt{dist}(x, y) \leq \texttt{dist}(x, u) + \texttt{dist}(u, y) = \texttt{dist}(u, x) + \texttt{dist}(u, y) \leq 2 \cdot \mathcal{A}(G)$. Also berechnet $\mathcal{A}$ eine 2-Approximation. Die Breitensuche benötigt genau die geforderte Zeit.

Aufgabe 3

(1 + 1 + 2 + 2 + 2 + 1 = 9 Punkte)

In der Übung wurde das $\mathcal{NP}$-vollständige Entscheidungsproblem SetCover (deutsch *Mengenüberdeckung*) vorgestellt. Hier betrachten wir das entsprechende Optimierungsproblem min-SetCover:

Gegeben: Universum $\mathcal{U} = \{u_1, \ldots, u_m\}$, Menge $\mathcal{S} = \{S_1, \ldots, S_n\}$ von Teilmengen $S_i \subseteq \mathcal{U}$
Gesucht: Menge $C \subseteq \{1, \ldots, n\}$ mit $|C|$ minimal, sodass $\bigcup_{i \in C} S_i = \mathcal{U}$ gilt.

In der Vorlesung haben wir gesehen, dass sich jedes $\mathcal{NP}$-vollständige Problem als *ganzzahliges Programm* (engl. *Integer Program*, IP) darstellen lässt. Im Folgenden ist ein noch unvollständiges IP für min-SetCover gegeben, das wir mit SC-$\mathbb{N}$ bezeichnen:

Gegeben:
min-SetCover-Instanz $I = (\mathcal{U}, \mathcal{S})$
Variablen:
Für jede Teilmenge $S_i \in \mathcal{S}$ eine Variable x_i
Nebenbedingungen:

(I) Für jede Teilmenge $S_i \in \mathcal{S}$: $x_i \in \{0, 1\}$

(II) Für jedes Element $u_i \in \mathcal{U}$: ☐

Zielfunktion:
Minimiere $f_{\mathbb{N}}(I) = \sum_{i=1}^{n} x_i$

Aus einer Lösung für SC-$\mathbb{N}$ lässt sich eine Lösung für min-SetCover konstruieren, indem man $C = \{i \in \{1, \ldots, n\} \mid x_i = 1\}$ setzt. Der Wert von x_i gibt also an, ob S_i in der Mengenüberdeckung enthalten ist oder nicht.

(a) Ergänzen Sie den fehlenden Teil von Nebenbedingung (II), sodass eine Lösung von SC-$\mathbb{N}$ einer Lösung von min-SetCover entspricht.

Eine gängige Methode, IPs zu approximieren, ist die *LP-Relaxierung*. Dabei wird die Beschränkung aufgehoben, dass die Variablen ganzzahlige Werte haben müssen. Das dadurch entstehende *lineare Programm* (LP) lässt sich in Polynomialzeit lösen. Im Fall von SetCover erhalten wir das Problem SC-$\mathbb{R}$. Der einzige Unterschied gegenüber SC-$\mathbb{N}$ ist in Nebenbedingung (I): Die x_i dürfen nun beliebige reelle Zahlen aus dem Intervall $[0, 1]$ sein. Dementsprechend ist auch der Wert der Zielfunktion $f_{\mathbb{R}}(I) = \sum_{i=1}^{n} x_i$ reellwertig.

Sei im Folgenden $\mathrm{OPT}_{\mathbb{R}}(I)$ der Wert von $f_{\mathbb{R}}(I)$ für eine optimale Lösung von SC-$\mathbb{R}$ und $\mathrm{OPT}_{\mathbb{N}}(I)$ der Wert von $f_{\mathbb{N}}(I)$ einer optimalen Lösung von SC-$\mathbb{N}$.

(b) Zeigen Sie, dass für jede min-SetCover-Instanz $I = (\mathcal{U}, \mathcal{S})$ gilt: $\mathrm{OPT}_{\mathbb{R}}(I) \leq \mathrm{OPT}_{\mathbb{N}}(I)$.

(c) Geben Sie eine min-SetCover-Instanz $I = (\mathcal{U}, \mathcal{S})$ an, für die $\mathrm{OPT}_{\mathbb{R}}(I) < \mathrm{OPT}_{\mathbb{N}}(I)$ gilt. Geben Sie für beide Probleme eine optimale Lösung an.

Wir betrachten nun folgenden Algorithmus $\mathcal{A}$, der eine (nicht notwendigerweise optimale) Lösung für SC-$\mathbb{N}$ berechnet:

1. Berechne eine optimale Lösung $X = (x_1, \ldots, x_n)$ mit $x_i \in [0, 1]$ für SC-$\mathbb{R}$.
2. Sei f die maximale Anzahl von Mengen S_i, in denen irgendein Element u_i vorkommt.
3. Generiere eine Lösung $X' = (x'_1, \ldots, x'_n)$ mit $x'_i \in \{0, 1\}$ für SC-$\mathbb{N}$, indem wir jedes x'_i wie folgt wählen:

$$x'_i = \begin{cases} 1 & \text{falls } x_i \geq \frac{1}{f} \\ 0 & \text{sonst.} \end{cases}$$

(d) Zeigen Sie, dass Algorithmus $\mathcal{A}$ eine Lösung für SC-$\mathbb{N}$ berechnet, also dass die Nebenbedingungen erfüllt sind.

(e) Zeigen Sie, dass für beliebige min-SetCover-Instanzen I gilt: $\mathcal{A}(I) \leq f \cdot \mathrm{OPT}_{\mathbb{N}}(I)$

(f) Ist $\mathcal{A}$ ein Approximationsalgorithmus für SC-$\mathbb{N}$ mit relativer Gütegarantie? Begründen Sie!

Lösung:

(a)

$$\sum_{j : u_i \in S_j} x_j \geq 1$$

(b) Jede Lösung von SC-$\mathbb{N}$ erfüllt die Nebenbedingungen von SC-$\mathbb{R}$ und ist somit auch eine Lösung von SC-$\mathbb{R}$. Es kann also nicht $\mathrm{OPT}_{\mathbb{R}}(I) > \mathrm{OPT}_{\mathbb{N}}(I)$ gelten, weil $\mathrm{OPT}_{\mathbb{R}}(I)$ dann nicht optimal wäre.

(c) $I = (\mathcal{U}, \mathcal{S})$ mit $\mathcal{U} = \{u_1, u_2, u_3\}$ und $\mathcal{S} = \{\{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_3\}\}$.
Eine optimale Lösung für SC-$\mathbb{R}$ ist $X_{\mathbb{R}} = \{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\}$ mit $f_{\mathbb{R}}(I) = \frac{3}{2}$.
Eine optimale Lösung für SC-$\mathbb{N}$ ist $X_{\mathbb{N}} = \{1, 1, 0\}$ mit $f_{\mathbb{N}}(I) = 2$.

(d) Bedingung (I) ist offensichtlich erfüllt.

Betrachte für Bedingung (II) ein beliebiges Element $u_i \in \mathcal{U}$. Sei $\mathcal{S}' = \{S_j \in \mathcal{S} \mid u_i \in S_j\}$. Da X eine Lösung von SC-$\mathbb{R}$ ist, gilt:

$$\sum_{S_j \in \mathcal{S}'} x_j \geq 1$$

Es gilt $|\mathcal{S}'| \leq f$. Damit sich die Werte zu mindestens 1 summieren, muss es mindestens ein $S_j \in \mathcal{S}'$ mit $x_j \geq 1/f$ geben. Für dieses gilt $x'_j = 1$, also gilt auch:

$$\sum_{S_j \in \mathcal{S}'} x'_j \geq 1$$

(e) Es gilt für jedes x'_i:

$$x'_i = \left\{ \begin{array}{ll} 1 & \text{falls } x_i \geq \frac{1}{f} \\ 0 & \text{sonst} \end{array} \right\} \leq \left\{ \begin{array}{ll} fx_i & \text{falls } x_i \geq \frac{1}{f} \\ 0 & \text{sonst} \end{array} \right\} \leq fx_i$$

Damit gilt:

$$\mathcal{A}(I) = \sum_{i=1}^{n} x'_i \leq \sum_{i=1}^{n} fx_i = f \cdot \mathrm{OPT}_{\mathbb{R}}(I)$$

Wegen Aufgabenteil (b) gilt also $\mathcal{A}(I) \leq f \cdot \mathrm{OPT}_{\mathbb{N}}(I)$.

(f) Nein, denn f ist keine Konstante, sondern hängt von der Eingabeinstanz ab.

Aufgabe 4

(2 + 3 = 5 Punkte)

Das Optimierungsproblem BinPacking ist wie folgt definiert:

Gegeben: Endliche Menge $M = \{a_1, \ldots, a_n\}$ mit Gewichtungsfunktion $s : M \to (0, 1]$.
Gesucht: Zuweisung der Elemente in M zu einer minimalen Anzahl an Bins m, sodass für jeden Bin B_i mit $i = 1, \ldots, m$ gilt:

$$\sum_{a \in B_i} s(a) \leq 1$$

In der Übung wurde der Algorithmus NextFit vorgestellt. Dabei werden die Elemente in der Reihenfolge $a_1, a_2, \ldots, a_n$ bearbeitet. Passt das Element a_i noch in den aktuellen Bin, so wird es in diesen eingefügt. Ansonsten wird ein neuer Bin hinzugefügt und a_i in diesen eingefügt. In der Übung wurde gezeigt, dass NextFit ein Approximationsalgorithmus ist und eine obere Schranke von 2 für dessen relative Güte bewiesen.

(a) Zeigen Sie, dass diese obere Schranke gewissermaßen optimal ist. Geben Sie dazu für jedes $k > 0$ eine Folge von Elementen an, sodass NextFit bei Abarbeitung dieser Folge $2k$ Bins benötigt, ein optimaler Algorithmus dagegen nur $k + 1$. Daraus folgt, dass die relative Gütegarantie mindestens $\lim_{k \to \infty} \frac{2k}{k+1} = 2$ sein muss.

(b) Die Strategie, um Elemente in die Bins einzufügen, wird nun verändert. Statt nur den aktuellen Bin zu betrachten, wird jetzt ein Element in den ersten Bin eingefügt, in dem noch ausreichend Platz ist. Diese Strategie wird FirstFit genannt. Zeigen Sie, dass für diesen neuen Approximationsalgorithmus $\mathcal{A}$ gilt: $\mathcal{R}_{\mathcal{A}} \geq \frac{5}{3}$.

Hinweis: Finden Sie drei Mengen von jeweils 6 *Elementen gleicher Größe, sodass der Algorithmus die ersten* 6 *Elemente in einen Bin, die nächsten* 6 *Elemente in* 3 *Bins und die letzten* 6 *Elemente in* 6 *Bins einfügt. In einer optimalen Lösung sollten alle Elemente in* 6 *Bins eingefügt werden können.*

Lösung:

(a) Wähle eine Folge der Länge $4k$, bei der die Elemente abwechselnd die Größe $\frac{1}{2}$ und $\frac{1}{2k}$ haben. Dann packt NextFit in jeden Bin ein Element der Größe $\frac{1}{2}$ und ein Element der Größe $\frac{1}{2k}$, benötigt also insgesamt $2k$ Bins. Ein optimaler Algorithmus kann jedoch alle $2k$ Elemente der Größe $\frac{1}{2k}$ in einen Bin packen und die $2k$ Elemente der Größe $\frac{1}{2}$ in k Bins, benötigt also nur $k + 1$ Bins.

(b) Wähle für die ersten 6 Elemente die Größe 0,15, für die nächsten 6 Elemente 0,34 und für die letzten 6 Elemente 0,51. Mit der vorgegebenen Strategie werden die ersten 6 Elemente in einen Bin mit Füllgrad 0,9 gepackt. Die nächsten 6 Elemente werden in 3 Bins mit Füllgrad 0,68 gepackt, und die letzten 6 Elemente werden in 6 Bins gepackt. Insgesamt werden so 10 Bins genutzt. Eine optimale Lösung würde jedoch nur 6 Bins benötigen, da die drei Elementgrößen zusammen genau 1 ergeben. Als relative Güte ergibt sich somit:

$$\mathcal{R}_{\mathcal{A}} \geq \frac{10}{6} = \frac{5}{3}$$

Aufgabe 5

(2 + 2 + 2 = 6 Punkte)

Betrachten Sie folgendes Job-Scheduling-Problem: Gegeben sind n Jobs $J = \{j_1, \ldots, j_n\}$, die auf einer Maschine bearbeitet werden sollen. Die Maschine kann immer nur einen Job gleichzeitig bearbeiten. Jeder Job j_i hat eine *Bearbeitungszeit* $p_i > 0$ und einen *Freigabezeitpunkt* $r_j \geq 0$, vor dem er nicht bearbeitet werden darf. Gesucht ist ein *Scheduling* S, das für jeden Zeitpunkt angibt, welcher Job gerade bearbeitet wird. Es soll

$$\sum_{i=1}^{n} C_i^S$$

minimiert werden, wobei wir mit C_i^S den Zeitpunkt bezeichnen, zu dem Job j_i im Scheduling S fertiggestellt wird.

Wir unterscheiden zwei Arten von Scheduling: Beim *nicht-präemptiven* Scheduling muss ein einmal angefangener Job solange bearbeitet werden, bis er fertiggestellt ist. Beim *präemptiven* Scheduling darf ein bereits angefangener Job unterbrochen und durch einen anderen ersetzt werden.

Betrachten Sie folgenden Algorithmus für die präemptive Variante des Problems:

Es wird also in jedem Zeitschritt unter allen verfügbaren Jobs derjenige mit der geringsten verbleibenden Bearbeitungszeit ausgewählt und bearbeitet. Falls gerade kein Job verfügbar ist, befindet sich die Maschine für diesen Schritt im Leerlauf.

Algorithmus 1: PreemptiveScheduling

```
for τ ← 0, 1, ... do
    if J = ∅ then break
    J_r ← {j_i ∈ J | r_i ≤ τ}
    if J_r = ∅ then
        S(τ) = ⊥
    else
        Wähle j_i ∈ J_r mit p_i minimal
        S(τ) ← j_i
        p_i ← p_i − 1
        if p_i = 0 then J ← J \ {j_i}
```

(a) Zeigen Sie, dass PreemptiveScheduling eine optimale Lösung berechnet.

Wir betrachten nun folgenden Algorithmus $\mathcal{A}$ für die nicht-präemptive Variante des Problems: Führe zunächst PreemptiveScheduling aus. Dies generiert ein präemptives Scheduling O. Sei $j_1, j_2, \ldots, j_n$ die Reihenfolge, in der die Jobs in O fertiggestellt werden, d.h. $C_1^O < C_2^O < \cdots < C_n^O$. Bearbeite die Jobs in dieser Reihenfolge, d.h. bearbeite Job j_i, sobald er freigegeben wurde und Job j_{i-1} fertiggestellt wurde.

(b) Sei S das von $\mathcal{A}$ erstellte Scheduling. Sei τ_i^S der Zeitpunkt, zu dem S mit der Bearbeitung von Job j_i beginnt. Zeigen Sie: Zu keinem Zeitpunkt zwischen C_i^O und τ_i^S ist die Maschine im Leerlauf.

(c) Zeigen Sie, dass $\mathcal{A}$ ein Approximationsalgorithmus mit relativer Gütegarantie 2 ist. Zeigen Sie dafür, dass für jeden Job j_i gilt: $C_i^S \leq 2 \cdot C_i^O$.

Lösung:

(a) Sei S ein optimales Scheduling, die zu irgendeinem Zeitpunkt τ nicht den Job j_a mit der kleinsten verbleibenden Bearbeitungszeit unter allen verfügbaren bearbeitet, sondern einen anderen Job j_b. Bezeichne mit $p_i(\tau)$ die verbleibende Bearbeitungszeit eines Job j_i zum Zeitpunkt τ. Es gilt also $r_a, r_b \leq \tau$ und $p_a(\tau) < p_b(\tau)$. Es gibt nach τ insgesamt noch $p_a(\tau) + p_b(\tau)$ viele Schritte in S, in denen entweder j_a oder j_b bearbeitet wird. Modifiziere S wie folgt zu einem neuen Scheduling S': Bearbeite in den ersten $p_a(\tau)$ dieser Schritte j_a und in den restlichen $p_b(\tau)$ Schritten j_b.

Da S zum Zeitpunkt τ Job j_b bearbeitet, wird j_a in S' mindestens einen Schritt früher fertig als in S, und wegen $p_a(\tau) < p_b(\tau)$ auch früher als j_b in S. Job j_b wird in S' zum gleichen Zeitpunkt fertig wie der spätere der beiden Jobs in S. Insgesamt wird in S' also ein Job mindestens einen Schritt früher fertig als in S und die anderen Jobs nicht später. Also ist S nicht optimal, ein Widerspruch.

(b) Angenommen, die Maschine sei zu einem Zeitpunkt $\tau \in [C_i^O, \tau_i^S]$ im Leerlauf. Das muss daran liegen, dass sie auf einen Job j_k wartet, der noch nicht freigegeben wurde, d.h. $r_k > \tau$. Da S Job j_k vor j_i abarbeitet, gilt $C_k^O < C_i^O$. Da j_k zum Zeitpunkt C_k^O in O bereits fertiggestellt wurde, muss $r_k < C_k^O$ gelten. Daraus folgt aber $r_k < C_i^O \leq \tau$, ein Widerspruch.

(c) Sei $\texttt{idle}(\tau)$ die Zeit, die die Maschine vor Zeitpunkt τ im Leerlauf verbracht hat. Es gilt:

$$C_i^S = \sum_{k=1}^{i} p_k + \texttt{idle}(t_i^S)$$

Offensichtlich gilt:

$$\sum_{k=1}^{i} p_k \leq C_i^O$$

Außerdem gilt nach Aufgabenteil (b): $\texttt{idle}(t_i^S) = \texttt{idle}(C_i^O) \leq C_i^O$. Damit gilt insgesamt $C_i^S \leq 2 \cdot C_i^O$ und somit auch:

$$\sum_{i=1}^{n} C_i^S \leq 2 \cdot \sum_{i=1}^{n} C_i^O$$

Die Polynomialität von $\mathcal{A}$ folgt daraus, dass PREEMPTIVE SCHEDULING Zeit $\mathcal{O}(n \log n)$ benötigt. Der logarithmische Faktor kommt dadurch zustande, dass in jedem Schritt der Job mit der geringsten Bearbeitungszeit gefunden werden muss.

Aufgabe 6

(3 + 2 = 5 Punkte)

(a) Sei $k > 1$ beliebig. Zeigen Sie: Wenn es einen Approximationsalgorithmus mit relativer Gütegarantie k^2 für MAX-CLIQUE gibt, dann gibt es auch einen Approximationsalgorithmus mit relativer Gütegarantie k.

Hinweis: Betrachten Sie für einen gegebenen Graphen $G = (V, E)$ den Graphen $G^2 = (V \times V, E')$, wobei $\{(u, v), (w, x)\} \in E'$ genau gilt, wenn die folgenden beiden Bedingungen erfüllt sind:

- $\{u, w\} \in E$ *oder* $u = w$
- $\{v, x\} \in E$ *oder* $v = x$

(b) Angenommen, MAX-CLIQUE hätte einen Approximationsalgorithmus mit relativer Gütegarantie k für irgendein $k > 1$. Zeigen Sie, dass es dann ein PTAS für MAX-CLIQUE gäbe. Ist Ihr Algorithmus auch ein FPTAS?

Anmerkung: In der Tat lässt sich zeigen, dass es für MAX-CLIQUE *keinen relativen Approximationsalgorithmus gibt, und somit auch kein PTAS.*

Lösung:

(a) Angenommen, es gäbe einen Approximationsalgorithmus $\mathcal{A}$ mit relativer Gütegarantie k^2. Wir konstruieren daraus einen Approximationsalgorithmus $\mathcal{A}'$ mit relativer Gütegarantie k.

Gegeben sei ein Graph $G = (V, E)$. Konstruiere daraus den Graphen $G^2 = (V \times V, E')$ wie im Hinweis vorgegeben. Wir zeigen: G besitzt genau dann eine Clique der Größe ℓ, wenn G^2 eine Clique der Größe ℓ^2 besitzt.

Angenommen, G besitzt eine Clique C der Größe ℓ. Dann ist die Knotenmenge $C^2 = \{(u, v) \mid u, v \in C\}$ eine Clique in G^2. Seien (u, v) und (w, x) zwei Knoten aus C^2. Dann sind $u, v, w, x \in$

C. Also existiert entweder die Kante $\{u, w\} \in E$ oder $u = w$; analog für v und x. Also gilt $\{(u, v), (w, x)\} \in E'$. Somit ist C^2 eine Clique und es gilt offensichtlich $|C^2| = |C|^2$.

Angenommen, G^2 besitzt eine Clique C^2 der Größe ℓ^2. Betrachte die Menge $C = \{u \in V \mid \exists v \in V : (u, v) \in C^2\}$. Diese ist eine Clique, denn für $u, w \in C$ gibt es Knoten $(u, v), (w, x) \in C^2$. Nach Definition von G^2 gilt dann $\{u, w\} \in E$ oder $u = w$. Analog ist auch die Menge $C' = \{v \in V \mid \exists u \in V : (u, v) \in C^2\}$ eine Clique. Offensichtlich gilt $C^2 = C \times C'$. Da die Kantenbeziehung in G^2 symmetrisch ist, muss $C = C'$ gelten. Daraus folgt $|C| = \sqrt{|C^2|} = \ell$.

Wende $\mathcal{A}$ auf G^2 an. Dies liefert eine Clique C^2 der Größe $k^2 \operatorname{OPT}(G^2)$. Extrahiere daraus die entsprechende Clique C der Größe $k \operatorname{OPT}(G)$ in G. Dies liefert offensichtlich eine k-Approximation. Die Laufzeit ist ebenfalls offensichtlich polynomial.

(b) Durch Iterieren des Vorgehens aus Aufgabenteil (a) lässt sich aus einem Approximationsalgorithmus mit relativer Güte k ein Approximationsalgorithmus mit relativer Güte $k^{1/i}$ für jedes $i \in \mathbb{N}$ konstruieren. Wenn eine $1 + \varepsilon$-Approximation gewünscht ist, muss man also i so wählen, dass $k^{1/i} \leq 1 + \varepsilon$ gilt. Das ist der Fall für $i = \lceil 1/(\log_k(1 + \varepsilon)) \rceil$.

Dabei wird der Graph G^i der Größe n^i konstruiert. Dieser hat Größe $\mathcal{O}(n^{1/\log \varepsilon})$, also ist der Algorithmus kein FPTAS.

Aufgabe 7

(2 + 1 = 3 Punkte)

Sei p ein Polynom und Π ein $\mathcal{NP}$-schweres Minimierungsproblem, bei dem die Optimierungsfunktion f_Π des Problems ganzzahlig ist. Außerdem gelte für jede Instanz I, dass $\operatorname{OPT}_\Pi(I) < p(|I_u|)$, wobei I_u die unäre Kodierung von I bezeichnet.

Zeigen Sie:

(a) Falls es für Π ein FPTAS gibt, so gibt es auch einen pseudopolynomialen Algorithmus für Π.

(b) Falls Π stark $\mathcal{NP}$-vollständig ist und $\mathcal{P} \neq \mathcal{NP}$ gilt, gibt es für Π kein FPTAS.

Lösung:

(a) Sei $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ ein FPTAS für Π, d.h., die Laufzeit von $\mathcal{A}_\varepsilon$ ist polynomial in $|I|$ und $\frac{1}{\varepsilon}$. Formal: Die Laufzeit von $\mathcal{A}_\varepsilon$ ist $q(|I|, \frac{1}{\varepsilon})$, wobei q ein Polynom ist.

Gegeben eine Instanz I berechnet der pseudopolynomiale Algorithmus $\varepsilon = \frac{1}{p(|I_u|)}$ und wendet $\mathcal{A}_\varepsilon$ auf I an.[1] Für die Lösung $\mathcal{A}_\varepsilon(I)$ gilt:

$$\mathcal{R}_{\mathcal{A}_\varepsilon} = \frac{\mathcal{A}_\varepsilon(I)}{\operatorname{OPT}_\Pi(I)} \leq (1 + \varepsilon)$$

[1] Dazu muss zunächst eine Beschreibung von $\mathcal{A}_\varepsilon$ generiert werden. Dies muss in polynomieller Zeit in $|I|$ und $\frac{1}{\varepsilon} = p(|I_u|)$ (also polynomiell in $|I_u|$) geschehen, damit der Algorithmus pseudopolynomial bleibt. In der Vorlesung wurde ein Approximationsschema als eine Familie von Algorithmen $\{\mathcal{A}_\varepsilon \mid \varepsilon > 0\}$ definiert. Diese Definition erfordert nicht, dass sich für ein bestimmtes ε eine Beschreibung des dazugehörigen Algorithmus $\mathcal{A}_\varepsilon$ in polynomieller Zeit generieren lässt. In der Fachliteratur ist jedoch folgende Definition gängiger: Ein Approximationsschema ist ein einzelner Algorithmus $\mathcal{A}(\varepsilon)$, bei dem der Approximationsgrad $\varepsilon > 0$ eine zusätzliche Eingabe ist. Bei einem FPTAS muss dementsprechend die Laufzeit dieses einzelnen Algorithmus polynomiell in $|I|$ und $\frac{1}{\varepsilon}$ sein. Hier muss also nicht mehr das passende $\mathcal{A}_\varepsilon$ ausgewählt werden, sondern ε wird einfach als Eingabe an $\mathcal{A}(\varepsilon)$ weitergereicht.

Wir erhalten also folgende Abschätzung:

$$\mathcal{A}_\varepsilon(I) \leq (1+\varepsilon)\,\mathrm{OPT}_\Pi(I) = \mathrm{OPT}_\Pi(I) + \varepsilon\,\mathrm{OPT}_\Pi(I) < \mathrm{OPT}_\pi(I) + \varepsilon p(|I_u|) = \mathrm{OPT}_\Pi(I) + 1$$

Da Π ein Minimierungsproblem ist und die Optimierungsfunktion von Π ganzzahlig ist, gilt somit $\mathcal{A}_\varepsilon(I) = \mathrm{OPT}_\Pi(I)$. $\mathcal{A}_\varepsilon(I)$ mit $\varepsilon = \frac{1}{p(|I_u|)}$ ist also ein exakter Algorithmus für Π.

Eine analoge Aussage kann auch für Maximierungsprobleme gezeigt werden.

(b) Nach Aufgabenteil (a) gilt: Wenn Π ein FPTAS besitzt, dann auch einen pseudopolynomialen Algorithmus. Π kann somit nicht stark $\mathcal{NP}$-vollständig sein.