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

Übungsblatt 5

Vorlesung Theoretische Grundlagen der Informatik im WS 18/19

Ausgabe 18. Dezember 2018
Abgabe 15. Januar 2019, 11:00 Uhr (im Kasten im UG von Gebäude 50.34)

Aufgabe 1

(3 + 2 + 2 = 7 Punkte)

(a) Beweisen Sie, dass es unter der Annahme $\mathcal{P} \neq \mathcal{NP}$ keinen polynomialen Approximationsalgorithmus mit relativer Gütegarantie für das TravelingSalesman-Problem gibt. Zeigen Sie dazu, dass für jedes $k \geq 1$ gilt: Wenn es einen polynomialen Approximationsalgorithmus mit relativer Gütegarantie k für das TSP gäbe, so ließe sich daraus ein Polynomialzeitalgorithmus für das $\mathcal{NP}$-vollständige HamiltonianCycle-Problem konstruieren.

(b) Wir betrachten nun das metrische TSP, bei dem die Gewichtsfunktion die Dreiecksungleichung erfüllt, d.h. für alle $u, v, w \in V$ gilt: $c(u, w) \leq c(u, v) + c(v, w)$. Zeigen Sie, dass das metrische TSP $\mathcal{NP}$-schwer ist, indem Sie von TSP reduzieren.

(c) In der Vorlesung wurde gezeigt, dass für das metrische TSP polynomiale Approximationsalgorithmen mit relativer Gütegarantie existieren. Warum lässt sich mit der Reduktion aus Aufgabenteil (b) kein polynomialer Approximationsalgorithmus mit relativer Gütegarantie für das TSP konstruieren?

Lösung:

(a) Sei als HamiltonianCycle-Instanz ein Graph $G = (V, E)$ gegeben. Konstruiere dann den Graphen $G' = (V, E')$ mit $E' = V \times V$ und definiere für $e \in E'$:

$$c(e) = \begin{cases} 1 & \text{falls } e \in E \\ k|V| + 1 & \text{sonst} \end{cases}$$

Offensichtlich lässt sich diese Konstruktion in Polynomialzeit durchführen.

Wenn G einen Hamiltonkreis enthält, so enthält (G', c) eine optimale Traveling-Salesman-Tour der Länge $|V|$. Ein k-Approximationsalgorithmus würde also eine Tour mit Länge $\leq k|V|$ finden. Wenn G keinen Hamiltonkreis enthält, enthält jede Traveling-Salesman-Tour in (G', c) mindestens eine Kante mit Gewicht $k|V|+1$, d.h. alle Touren sind länger als $k|V|$. Also enthält G genau dann einen Hamiltonkreis, wenn ein k-Approximationsalgorithmus auf (G', c) eine Tour mit Länge $\leq k|V|$ findet. Hätte ein solcher Algorithmus polynomielle Laufzeit, wäre also auch das HamiltonianCycle-Problem in Polynomialzeit lösbar.

(b) Gegeben sei eine TSP-Instanz $(G = (V, E), c)$ mit Gewichtsfunktion c, die nicht die Dreiecksungleichung erfüllt. Sei $c_{\max} := \max_{e \in E} c(e)$. Wir konstruieren die metrische TSP-Instanz (G, c') mit $c'(e) = c(e) + c_{\max}$, was offensichtlich in Polynomialzeit geht. Da sich dadurch der Wert jeder Tour um genau $|V|c_{\max}$ erhöht, bleibt die Menge der optimalen Touren gleich. Außerdem ist die Dreiecksungleichung erfüllt, da für alle $u, v, w \in V$ gilt:

$$c'(u, w) = c(u, w) + c_{\max} \leq 2c_{\max} \leq 2c_{\max} + c(u, v) + c(v, w) = c'(u, v) + c'(v, w)$$

(c) Betrachte eine TSP-Instanz $(G = (V, E), c)$, in der die Länge einer optimalen Tour W ist. Die in der Reduktion aus Aufgabenteil (b) konstruierte Instanz für das metrische TSP hat dann eine optimale Tourlänge von $W + |V|c_{\max}$. Ein polynomialer Approximationsalgorithmus mit relativer Gütegarantie $k > 1$ gibt also eine Tour mit Länge höchstens $k(W + |V|c_{\max}) = kW + k|V|c_{\max}$ aus. Die entsprechende Tour in G hat eine Länge von $kW + (k-1)|V|c_{\max} > kW$, ist also keine k-Approximation.

Aufgabe 2

(1 + 1 = 2 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 3

(1 + 4 + 3 + 2 = 10 Punkte)

Das *k-Zentren-Problem* ist wie folgt definiert: Gegeben seien eine Menge von Punkten $P \subseteq \mathbb{R}^2$ im zweidimensionalen Raum, die euklidische Distanzmetrik $\text{dist}(\cdot, \cdot) \colon P^2 \to \mathbb{R}$ und ein fester Parameter

k. Gesucht ist eine Teilmenge $C \subseteq P$ mit $|C| = k$ von *Zentren*, sodass die maximale Distanz eines Punktes $p \in P$ zu seinem nächsten Zentrum in C minimiert wird. Wir bezeichnen das nächste Zentrum eines Punktes p in C mit $C(p)$. Wir wollen also $\max_{p \in P} \text{dist}(p, C(p))$ minimieren.

Betrachten Sie folgenden Algorithmus für das k-Zentren-Problem:

Algorithmus 1: k-CenterApprox

```
Wähle beliebigen Punkt p ∈ P
C_1 ← {p}
Für i = 2 ... k
    Für p ∈ P \ C_{i-1}
        dist_i[p] ← min_{c ∈ C_{i-1}} dist(p, c)
    Wähle c_i ∈ P \ C_{i-1} mit dist_i[c_i] maximal
    C_i ← C_{i-1} ∪ {c_i}
return C_k
```

Es wird also in jedem Schritt der Punkt als neues Zentrum ausgewählt, der die größte Distanz zu den bisher ausgewählten Zentren hat.

(a) Sei $C_{\mathcal{A}}$ die Menge der Zentren, die von k-CenterApprox berechnet wurden. Sei p der Punkt, für den $\text{dist}(p, C_{\mathcal{A}}(p))$ maximal ist. Zeigen Sie: Wenn $\text{dist}(p, C_{\mathcal{A}}(p)) > x$ gilt, dann haben alle Zentren in $C_{\mathcal{A}}$ paarweise untereinander Distanz $> x$.

(b) Zeigen Sie, dass k-CenterApprox eine 2-Approximation für das k-Zentren-Problem berechnet.

Wir betrachten nun als Variante das *k-Produzenten-Problem*: Hier wird die Menge P der Punkte unterteilt in eine Menge D von Konsumenten und eine Menge S von Produzenten. Als Zentren dürfen jetzt nur Produzenten gewählt werden, also $C \subseteq S$. Minimiert werden soll nun $\max_{d \in D} \text{dist}(d, C(d))$, d.h. uns interessiert nur noch die Distanz der Konsumenten zum nächsten Zentrum.

(c) Geben Sie einen Algorithmus an, der eine 3-Approximation für das k-Produzenten-Problem berechnet.

(d) Beweisen Sie, dass Ihr Algorithmus eine 3-Approximation berechnet.

Lösung:

(a) Betrachte die i-te Iteration des Algorithmus. Es wird ein Zentrum c_i ausgewählt und zu C_{i-1} hinzugefügt, um C_i zu bilden. Offensichtlich gilt für alle $c \in C_{i-1}$, dass $\text{dist}(c, p) > x$, also gilt $\text{dist}_i[p] > x$. Nach Konstruktionsvorschrift des Algorithmus wird c_i so gewählt, dass $\text{dist}_i[c_i]$ maximiert wird. Also muss $\text{dist}_i[c_i] \geq \text{dist}_i[p] > x$ gelten, d.h. c_i hat zu allen zuvor ausgewählten Zentren Distanz $> x$. Da sich dieses Argument auf alle Iterationen des Algorithmus anwenden lässt, haben alle Zentren paarweise Distanz $> x$.

(b) Bezeichne wie üblich für eine Probleminstanz I den von k-CenterApprox berechneten Wert mit $\mathcal{A}(I)$ und den Optimalwert mit $\text{OPT}(I)$. Die von k-CenterApprox berechneten Zentren bezeichnen wir mit $C_{\mathcal{A}}$ und die optimalen Zentren mit C_{OPT}. Angenommen, der Algorithmus berechnet keine 2-Approximation. Dann gibt es eine Probleminstanz I, für die

$\mathcal{A}(I) > 2 \cdot \mathrm{OPT}(I)$ gilt. Betrachte den Punkt p, für den $\mathrm{dist}(p, C_{\mathcal{A}}(p))$ maximal ist. Es gilt $\mathrm{dist}(p, C_{\mathcal{A}}(p)) > 2 \cdot \mathrm{OPT}(I)$, d.h. nach Aufgabenteil (a) haben auch alle Zentren in $C_{\mathcal{A}}$ paarweise Distanz $> 2 \cdot \mathrm{OPT}(I)$. Zusammen mit p finden wir also $k+1$ Punkte, die paarweise Distanz $> 2 \cdot \mathrm{OPT}(I)$ haben.

Betrachte für jeden Punkt $q \in C_{\mathcal{A}} \cup \{p\}$ das nächste Zentrum $C_{\mathrm{OPT}}(q)$ in der optimalen Lösung. Da wir $k+1$ Punkte haben, aber nur k Zentren, müssen mindestens zwei Punkte q_1, q_2 dasselbe Zentrum haben, also $C_{\mathrm{OPT}}(q_1) = C_{\mathrm{OPT}}(q_2)$. Der Abstand beider Punkte zum Zentrum beträgt jeweils höchstens $\mathrm{OPT}(I)$. Nach der Dreiecksungleichung gilt $\mathrm{dist}(q_1, q_2) \leq \mathrm{dist}(q_1, C_{\mathrm{OPT}}(q_1)) + \mathrm{dist}(C_{\mathrm{OPT}}(q_2), q_2) \leq 2 \cdot \mathrm{OPT}(I)$. Das ist ein Widerspruch zur Annahme, dass der Algorithmus keine 2-Approximation berechnet.

(c) Wende zunächst k-CenterApprox auf die Menge D der Konsumenten an. Das Ergebnis ist eine Menge $C_D = \{d_1, \ldots, d_k\}$ von k Konsumenten-„Zentren“. Berechne nun für jedes $d_i \in C_D$ den Produzenten s_i, der die geringste Distanz zu d hat, und füge s_i zur Lösung hinzu. Es ist möglich, dass einige Produzenten dabei mehrfach ausgewählt werden. In diesem Fall können wir die Lösung einfach mit beliebigen weiteren Produzenten auffüllen.

(d) Betrachte einen beliebigen Konsumenten $d \in D$. Die Ausführung von k-CenterApprox weist d einen anderen Konsumenten d_i mit $\mathrm{dist}(d, d_i) \leq 2 \cdot \mathrm{OPT}(I)$ hinzu. Als Zentrum $C(d)$ wird der von d_i nächste Produzent ausgewählt, also gilt $\mathrm{dist}(d_i, C(d)) \leq \mathrm{OPT}(I)$. Mit der Dreiecksungleichung gilt dann $\mathrm{dist}(d, C(d)) \leq 2 \cdot \mathrm{OPT}(I) + \mathrm{OPT}(I) = 3 \cdot \mathrm{OPT}(I)$.

Aufgabe 4 (3 Punkte)

Geben Sie einen pseudopolynomialen Algorithmus für das Partition-Problem an. Zeigen Sie, dass Ihr Algorithmus polynomielle Laufzeit in der Eingabegröße und der größten in der Eingabe vorkommenden Zahl hat.

Hinweis: Verwenden Sie dynamische Programmierung.

Lösung:
Sei $M = \{m_1, \ldots, m_n\}$ die gegebene Menge und $w\colon M \to \mathbb{N}_0$ die Gewichtsfunktion. Sei $W := \sum_{m \in M} w(m)$ das Gesamtgewicht aller Elemente. Die Partition-Instanz ist genau dann lösbar, wenn es eine Teilmenge $M' \subseteq M$ gibt, für die $\sum_{m \in M'} w(m) = W/2$ gilt.

Wir definieren für $0 \leq i \leq n$ und $X \in \mathbb{Z}$ die boolesche Variable $q(i, X)$. Diese soll genau dann wahr sein, wenn es eine Teilmenge $M' \subseteq \{m_1, \ldots, m_i\}$ gibt, für die $\sum_{m \in M'} w(m) = X$ gilt. Um die Partition-Instanz zu lösen, muss also $q(n, W/2)$ berechnet werden.

Die $q(i, X)$ lassen sich rekursiv mit folgender Formel berechnen:

$$q(i, X) = \begin{cases} \text{true} & \text{, falls } X = 0 \\ \text{false} & \text{, falls } X < 0 \\ \text{false} & \text{, falls } i = 0 \text{ und } X > 0 \\ q(i-1, X) \vee q(i-1, X - w(m_i)) & \text{sonst} \end{cases}$$

Wir können $q(n, W/2)$ also mit einem dynamischen Programm berechnen, das in aufsteigender Reihenfolge alle $q(i, X)$ für $1 \leq i \leq n$ und $1 \leq X \leq W/2$ berechnet und dabei auf die bereits berechneten Werte zurückgreift. Dieses dynamische Programm hat Laufzeit $\mathcal{O}(nW)$. Die größte in der Eingabe vorkommende Zahl ist $w_{\max} := \max_{m \in M} w(m)$. Offensichtlich gilt $W \leq n w_{\max}$, also

hat der Algorithmus Laufzeit $\mathcal{O}(n^2 w_{\max})$, was wie gefordert polynomial in der Eingabegröße und $w_{\max}$ ist.

Aufgabe 5

(3 + 1 = 4 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 $\mathrm{OPT}_\Pi(I) < p(|I_u|)$, 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.[1]

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.[2] Für die Lösung $\mathcal{A}_\varepsilon(I)$ gilt:

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

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.

Aufgabe 6

(3 + 1 + 1 = 5 Punkte)

[1] In der ursprünglichen Version des Übungsblatts hieß es fälschlicherweise: „Unter der Annahme $\mathcal{P} \neq \mathcal{NP}$ gibt es für ein stark $\mathcal{NP}$-vollständiges Problem kein FPTAS.“ Die Einschränkungen für Π aus dem vorigen Aufgabenteil müssen aber auch hier gelten.

[2] 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.

Für einen Punkt p in der Ebene und eine positive reelle Zahl $r > 0$ ist eine *Doppelkreisbeschriftung* durch zwei Punkte q_1, q_2 gegeben, die folgende Bedingungen erfüllen:

$$|pq_1| = |pq_2| = r \qquad \text{und} \qquad |q_1q_2| = 2r.$$

Anschaulich bedeutet dies, dass der Punkt p der Berührungspunkt zweier Kreise mit Radius r und mit den Mittelpunkten q_1 und q_2 ist.

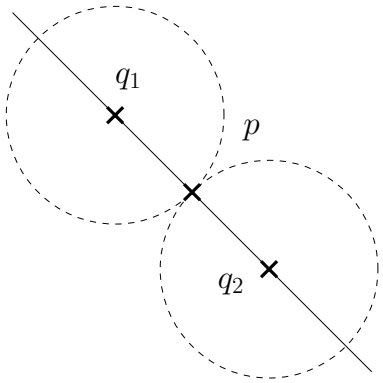

Abbildung 1: Beispiel einer Doppelkreisbeschriftung.

Bei dem Optimierungsproblem DoubleCircleLabel wird zu einer Menge von n Punkten $p_1, \ldots, p_n$ in der Ebene die größte (positive) reelle Zahl r gesucht, so dass jeder Punkt p_i eine r-Doppelkreisbeschriftung hat und die Doppelkreisbeschriftungen verschiedener Punkte schnittfrei sind.

(a) Geben Sie einen Approximationsalgorithmus mit relativer Gütegarantie 2 und polynomialer Laufzeit an und zeigen Sie ebendiese Eigenschaften.

Hinweis: Sie dürfen davon ausgehen, dass zwei mit Radius r schnittfrei doppelkreisbeschriftete Punkte p_1, p_2 mindestens Abstand $2r$ haben.

(b) Konstruieren Sie ein Beispiel, bei dem Ihr Approximationsalgorithmus nicht den optimalen Wert ausgibt. Erklären Sie!

(c) Das Problem DoubleCircleLabel ist $\mathcal{NP}$-vollständig. Geben Sie eine Intuition dafür, inwiefern die Entscheidungen, die Ihr Approximationsalgorithmus aus Aufgabenteil (a) trifft, „leichter“ sind als diejenigen, die ein optimaler Algorithmus für DoubleCircleLabel treffen muss.

Lösung:

(a) Der Approximationsalgorithmus gibt ein Viertel der kleinsten Distanz zweier verschiedener Punkte der Eingabemenge aus. Dies ist in quadratischer Zeit möglich.

Sei r der ausgegebene Wert. Dann ist $r' = 2r$ der maximale Wert, so dass sich um jeden Punkt der Eingabe ein Kreis mit Radius r' zeichnen lässt, ohne dass sich Kreise (bis auf ihren Rand) schneiden. In jeden dieser Kreise lässt sich dann eine Doppelkreisbeschriftung mit Radius r schnittfrei einzeichnen. Der Algorithmus gibt also immer einen Wert aus, zu dem es auch tatsächlich eine Lösung gibt.

Es bleibt zu zeigen, dass der Wert des Approximationsalgorithmus mindestens halb so groß ist wie der optimale Wert. Nehme dazu an, dass es eine Doppelkreisbeschriftung mit Radius

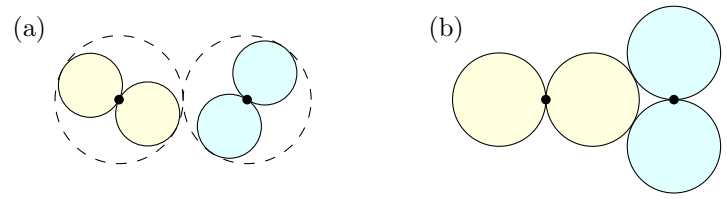

Abbildung 2: Eine approximierte Lösung (a) und eine bessere und optimale Lösung (b).

$r'' > 2r$ gäbe. Bezeichne mit $p_1 \neq p_2$ die beiden Punkte der Eingabe mit geringstem Abstand. Es gilt also $|p_1p_2| = 4r$. Da die Doppelkreisbeschriftung schnittfrei ist, würde gemäß dem Hinweis $|p_1p_2| \geq 2r'' > 4r$ gelten, ein Widerspruch.

(b) Siehe Abbildung 2.

(c) Der Approximationsalgorithmus kann die Winkel, in denen die Doppelkreisbeschriftungen gezeichnet werden, beliebig wählen, da diese in jedem Fall innerhalb der schnittfreien umschriebenen Kreise liegen und damit ebenfalls schnittfrei sind. Ein optimaler Algorithmus muss jedoch Winkel wählen, die gut zueinander passen, damit die Beschriftung schnittfrei bleibt. In der Lösung in Abbildung 2 (b) könnten etwa die blauen Kreise nicht rotiert werden, ohne einen Schnitt zu erzeugen.

Aufgabe 7

(3 + 2 + 1 = 6 Punkte)

Beim Entscheidungsproblem k-GraphPartition gilt es zu entscheiden, ob die Knoten eines Graphen $G = (V, E)$ so in k Mengen partitioniert werden können, dass alle Mengen der Partition exakt gleich groß sind und die Anzahl an Kanten, die nicht zwischen Knoten derselben Menge verlaufen, höchstens m ist (wobei m auch Teil der Eingabe ist).

Bei dem Problem 3-Partition erhält man eine Menge von $n = 3k$ Ganzzahlen $a_1, a_2, \ldots, a_n$ und einen Wert A mit

$$A/4 < a_i < A/2 \quad \text{und} \quad \sum_{i=1}^{n} a_i = kA.$$

Die Aufgabe ist es, zu entscheiden, ob die Zahlen so in Tripel partitioniert werden können, dass die Summe der Zahlen jedes Tripels gerade A ist. Dieses Problem ist stark $\mathcal{NP}$-vollständig. Also ist das Problem Restricted-3-Partition, bei dem alle a_i höchstens polynomielle Größe in n haben, immer noch $\mathcal{NP}$-vollständig.

(a) Geben Sie eine polynomiale Transformation von Restricted-3-Partition auf k-GraphPartition an. Beweisen Sie alle geforderten Eigenschaften.

(b) Beim Optimierungsproblem k-GraphPartition gilt es, den kleinsten Wert von m auszugeben, für den eine derartige Partition existiert. Zeigen Sie, dass es unter Annahme von $\mathcal{P} \neq \mathcal{NP}$ keinen Approximationsalgorithmus mit fester relativer Gütegarantie und polynomialer Laufzeit für das Optimierungsproblem k-GraphPartition gibt.

(c) Weshalb lässt sich ein analoger Beweis zu dem aus Teilaufgabe (b) nicht mit einer polynomiellen Transformation von Partition führen?

Lösung:

(a) Konstruiere einen Graph G, der für jede Zahl a_i eine Clique der Größe a_i enthält. Zwischen den Cliquen verlaufen keine Kanten. Da die a_i polynomiell beschränkt sind, ist die Konstruktion von G in polynomieller Zeit möglich.

Falls die Restricted-3-Partition-Instanz lösbar ist, lässt sich G so in k Teile partitionieren, dass keine Kante zwischen Knoten unterschiedlicher Mengen verläuft. Falls sich andererseits G so in k Teile partitionieren lässt, dass keine Kante zwischen Knoten unterschiedlicher Mengen verläuft, müssen in jeder Clique alle Knoten derselben Menge zugeteilt worden sein. Wegen der Einschränkung $A/4 < a_i < A/2$ muss jede Menge aus genau drei Cliquen bestehen, da die Mengen sonst nicht gleich groß wären. Also induziert die Partitionierung eine Lösung der Restricted-3-Partition-Instanz.

Mit $m = 0$ ergibt sich also eine k-GraphPartition-Instanz, die äquivalent zur Restricted-3-Partition-Instanz ist.

(b) Angenommen, es gäbe einen polynomiellen Approximationsalgorithmus mit relativer Gütegarantie für das k-GraphPartition-Problem. Konstruiere daraus einen polynomiellen Algorithmus, der das Restricted-3-Partition-Problem löst. Verwende die Transformation aus Teilaufgabe (a). Falls die Restricted-3-Partition-Instanz lösbar ist, lässt sich G so partitionieren, dass keine Kante zwischen Knoten unterschiedlicher Mengen verläuft. Der Optimalwert ist also 0, und da der Approximationsalgorithmus relative Güte garantiert, muss auch dieser 0 ausgeben. Falls die Restricted-3-Partition-Instanz nicht lösbar ist, verläuft in der optimalen Partitionierung von G mindestens eine Kante zwischen Knoten unterschiedlicher Mengen. Der Optimalwert ist also echt größer als 0, und somit muss auch der Approximationsalgorithmus einen Wert echt größer als 0 ausgeben. Die Restricted-3-Partition-Instanz ist also genau dann lösbar, wenn der Approximationsalgorithmus 0 ausgibt. Dies widerspricht der $\mathcal{NP}$-Schwere von Restricted-3-Partition, es kann also keinen polynomiellen Approximationsalgorithmus für k-GraphPartition mit relativer Gütegarantie geben.

(c) Das Problem Partition ist nicht stark $\mathcal{NP}$-vollständig. Dadurch benötigt entweder die Konstruktion von G exponentielle Zeit oder – wenn man sich auf polynomiell begrenzte Zahlen einschränkt – das Problem wäre in Polynomialzeit lösbar. Dann wäre insbesondere der exakte Lösungsalgorithmus auch ein Approximationsalgorithmus mit relativer Gütegarantie 1 und polynomialer Laufzeit.