RWTH Aachen **SS 2010**
Lehrgebiet Theoretische Informatik Übungsblatt 9
Kneis—Langer—Rossmanith 30.06.2010

Übung zur Vorlesung Effiziente Algorithmen

Tutoraufgabe 21

Entwerfen Sie einen Approximationsalgorithmus mit konstanter Güte für Independent Set auf planaren Graphen.

Ansatz:

Zunächst einmal können wir zu der intuitiven Einsicht gelangen, daß ein planarer Graph nicht ausschließlich aus Knoten hohen Grades bestehen kann.

Tatsächlich folgt aus Eulers Formel $|V| - |E| + |F| = 2$ und einigen grundsätzlichen Überlegungen, daß ein endlicher ungerichteter Graph $G = (V, E)$ ohne Schleifen und ohne Multikanten höchstens $3|V| - 6$ Kanten haben kann, wenn $|V| \geq 3$ gilt.

Den Sonderfall, daß es kein inneres *face* gibt, möge man sich getrennt überlegen; man sieht leicht, daß die obige Schranke für alle derartigen Graphen mit mindestens drei Knoten gilt. Ansonsten verwenden wir die folgende Argumentation:

Jedes *face* berührt mindestens drei Kanten. Jede Kante berührt höchstens zwei *faces*. Deshalb gilt $|F| \leq \frac{2}{3}|E|$ und somit $|E| \leq 3|V| - 6$. Daraus folgt, daß jeder planare Graph einen Knoten v des Grades höchstens fünf hat.

Fügen wir v dem Independent Set hinzu, werden höchstens 5 Knoten aus einer optimalen Lösung blockiert. Da wir in jedem Schritt $N[v]$ aus dem Graph entfernen, ist der übrige Graph erneut planar und wir können das Verfahren wiederholen.

Die Laufzeit ist offensichtlich polynomiell, der Algorithmus also eine 5-Approximation.

Eine 4-Approximation erhält man, indem man den 4-Farben Satz anwendet und die Größte der Farbklassen als Independent Set wählt.

Tutoraufgabe 22

Entwerfen Sie ein PTAS für Independent Set auf Untergraphen eines $m \times n$-Gitters, bei denen einige Kanten entfernt wurden.

Ansatz:

Man zerlege das Gitter in Teilgraphen der Größe k. Dazu muss man höchstens $(n/\sqrt{k})m + (m/\sqrt{k})n$ Knoten entfernen. Dies ist leich einzusehen, da man das vollständige Gitter in Felder der Größe $\sqrt{k} \times \sqrt{k}$ zerlegen kann.

Berechnet man für die Teilinstanzen der Größe k eine optimale Lösung, so ergibt sich für die Güte $F^*(G) - F(G) \leq 2nm/\sqrt{k}$. Da eine optimale Lösung mindestens $(n \times m)/4$ Knotenenthält (folgt aus dem 4-Farben Satz), ergibt sich die Güte

$$\frac{2nm/\sqrt{k}}{nm/4} \leq \frac{8}{\sqrt{k}}.$$

Wählt man k groß genug, so ist diese Güte kleiner als ein beliebiges ϵ. Die Laufzeit des Algorithmusn ist offensichtlich $2^k p(n, m)$ für ein Polynom p.

Tutoraufgabe 23

Zeigen Sie, daß das folgende Problem NP-schwer ist, und finden Sie einen Approximationsalgorithmus mit konstanter Güte:

Am LuFGTI wird sehr oft (das heißt in einer unendlichen Folge von Runden) Tee gekocht und getrunken. Dabei gilt es, n Trinker zu bewirten, die jeweils Tassen der Größen $c_1, \ldots, c_n$ gefüllt haben möchten. In einer Runde wird jeweils eine Kanne der Größe C zubereitet, und es wäre äußerst unhöflich, jemandem nur eine halbe Tasse einzuschenken. Es gilt nun, ein Scheduling für das Nachfüllen zu finden, bei dem die längste Wartezeit minimiert wird.

Lösung:

Das Problem ist NP-schwer, da sich leicht eine Reduktion vom folgenden Rucksackproblem findet: Gegeben sind k Rucksäcke der Größe C und Gegenstände mit Gewicht $c_1, \ldots, c_m$. Falls mit diesen Werten ein Scheduling eine Wartezeit von höchstens k möglich ist, so können alle Gegenstände in die k Rucksäcke gepackt werden.

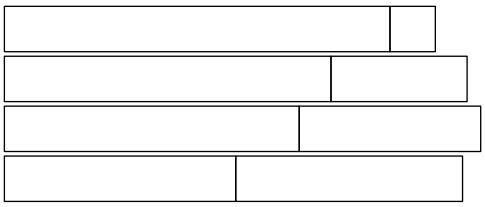
Ein Approximationsalgorithmus kann wie folgt vorgehen: Fülle die Tassen der Größe nach, solange bis die Kanne leer ist. Fülle danach dort weiter, wo in der letzten Runde abgebrochen wurde. Sobald die kleinste Tasse abgearbeitet ist, wird der Rest Tee verschüttet und in der nächsten Runde von vorne begonnen.

Um die Approximationsgüte zu ermitteln, reicht es aus, die Zeit zu vergleichen, in der jeweils alle zum ersten Mal mit Tee versorgt werden. Für unseren Algorithmus entspricht dies gerade der globalen maximalen Wartezeit, da der Schedule immer wiederholt wird. Für die optimale Lösung ist dieser Wert eine untere Schranke der Wartezeit.

Da in jedem Schritt die Kanne mindestens zur Hälfte geleert wird, eine optimale Lösung aber höchstens die ganze Kanne nutzt, ergibt sich eine Approximationsgüte von 2.

Hausaufgabe 18 (10 Punkte)

Beweisen Sie folgende Behauptung aus der Vorlesung: Falls ein optimaler Plan für I jedem Prozessor höchstens zwei Aufgaben zuordnet, dann ist ein LPT-Schedule für I ebenfalls optimal.



Lösung:

Zu zeigen ist: Wenn eine optimale Lösung jedem Prozessor genau zwei Jobs zuordnet, dann liefert auch LPT eine optimale Lösung.

Seien $s_1, \ldots, s_k, t_1, \ldots, t_k$ die zu verteilenden Jobs mit $s_i \geq t_j$, $s_i \geq s_{i+1}$ und $t_i \geq t_{i+1}$. Sei L die von LPT gelieferte Lösung und Z eine weitere Lösung, bei der dem i-ten Prozessor jeweils die Jobs s_i und t_{k-i+1} zugewiesen werden. Wir zeigen zunächst, daß L nicht teurer als Z ist:

Wenn in L jeder Prozessor zwei Aufgaben hat, dann ist $L = Z$. Andernfalls gibt es in L einen Prozessor A, der genau einen Job s_i hat. Den Prozessor, auf dem das Gegenstück

t_{k-i+1} liegt, nennen wir B. Wenn wir t_{k-i+1} aus B entfernen, dann ist die Laufzeit von A nicht kleiner als die von B (wegen LPT). Wird diese Aufgabe dann stattdessen dem Prozessor A zugewiesen, so kann die so entstandene neue Lösung L' nicht billiger als L geworden sein. Dieses Vorgehen kann iterativ angewandt werden, um eine Lösung mit zwei Jobs pro Prozessor zu konstruieren, die immer noch mindestens so teuer wie L ist. Insbesondere ergibt sich Z!

Sei OPT eine optimale Lösung für die gegebene Instanz. Es bleibt zu beweisen, daß Z nicht teurer als OPT ist. Falls ein Prozessor A die Jobs s_i und s_j verarbeitet, dann ist ein anderer Prozessor B mit t_m und t_n belegt. Da $s_i + t_n \leq s_i + s_j$ und $s_j + t_m \leq s_i + s_j$ gilt, können wir eine neue optimale Lösung bilden, bei der A die Jobs s_i und t_n verarbeitet, B hingegen s_j und t_m.

Wir erhalten also eine optimale Lösung, bei der jeder Prozessor ein s_i und ein t_n verarbeitet. Sei i der kleinste Index, so daß ein Prozessor A zwar s_i verarbeitet, aber nicht t_{k-i+1}, sondern t_{k-l+1} mit $l \neq i$. Sei B der Prozessor, der mit s_j und t_{k-i+1} belegt ist. Wenn A nun s_i und t_{k-i+1} verarbeitet, B dagegen s_j und t_{k-l+1}, so ist auch dies eine optimale Lösung.

Durch iteratives Anwenden dieser Vertauschoperationen erhalten wir genau Z. Damit muß auch L optimal sein, denn $L \leq Z \leq OPT$.

Hausaufgabe 19 (10 Punkte)

Gegeben ist folgendes Optimierungsproblem:

Eingabe:	Ein Graph $G = (V, E)$
Lösung:	Eine Partition $V = V_1 \cup \ldots \cup V_7$
Ziel:	Maximiere $\lvert\{\{v_i, v_j\} \in E \mid v_i \in V_s, v_j \in V_t, s \neq t\}\rvert$

Geben Sie eine 3/2-Approximation an (Sie sollen also mindestens 2/3 der Kanten einer optimalen Lösung schneiden). Beweisen Sie Korrektheit und Güte Ihres Verfahrens.

Lösung

Wir beginnen mit einer beliebigen Partition von V. Falls es einen Knoten $v \in V_i$ gibt, der mehr Nachbarn in V_i als in V_j hat, dann verschieben wir v nach V_j. Sobald es keinen solchen Knoten mehr gibt, haben wir eine 7/6 Approximation gefunden.

Beweis: Zuerst zeigen wir, daß der Algorithmus in polynomieller Zeit terminiert. Offensichtlich kann v in polynomieller Zeit gefunden werden. Das Verschieben ist ebenso in polynomieller Zeit möglich. Dabei erhöht sich aber die Anzahl der geschnittenen Kanten mindestens um 1, es gibt also höchgstens $|E|$ Vertauschungsoperationen.

Güte: $\deg_i(v)$ die Anzahl der Kanten von v zu Knoten in V_i. Falls $v \in V_s$, so gilt offensichtlich $\deg_i(v) \geq \deg_s(v)$ für alle $1 \leq i \leq 7$. Das bedeutet daß mindestens $6/7 \deg(v)$ Kanten adjazent zu v geschnitten werden. Da in einer optimalen Lösung höchstens alle Kanten geschnitten werden, ergibt sich eine Güte von

$$\frac{|E|}{1/2 \sum_{v \in V} 6/7 \deg(v)} \leq \frac{7|E|}{6|E|}.$$