RWTH Aachen
Lehrgebiet Theoretische Informatik
Kuinke–Rossmanith–Sanchez

WS 2016
Lösungsvorschlag 11
17.1.2017

Übung zur Vorlesung Effiziente Algorithmen

Tutoraufgabe 28

a)+b) Wir können einfach ein inklusionsmaximales Matching berechnen. Wenn der Graph – wie üblich – durch Adjazenzlisten repräsentiert wird, dann ist das in linearer Zeit möglich: Wir gehen einfach alle Kanten e durch und fügen e der Lösung zu, falls e nicht inzident zu einem Knoten ist, welcher bereits der Lösung angehört. Diese verbotenen Knoten können z.B. in einem Bitarray gespeichert werden, so daß jede Operation nur konstante Zeit braucht (außer dem einmaligen Initialisieren).

Der Approximationsfaktor ist zwei: Sei M die Lösung, welche der Algorithmus berechnet und OPT eine optimale Lösung. Sagen wir eine Kante $e \in M$ *stört* eine Kante $e' \in \text{OPT}$, wenn e und e' inzident sind oder wenn $e = e'$. Es ist klar, daß jede Kante in OPT gestört wird und daß jede Kante in M höchstens zwei Kanten in OPT stören kann. Also ist $2|M| \leq |\text{OPT}|$.

c) Natürlich sind alle Knoten in einem inklusionsmaximalem Matching ein Vertex Cover. Da jede Kante im Matching in jedem Vertex Cover abgedeckt werden kann, ist unser Vertex Cover höchstens doppelt so groß wie jedes andere, auch wie jedes optimale. Der Approximationsfaktor ist also zwei.

Tutoraufgabe 29

a) Solange noch Variablen übrig sind, wählen wir eine Variable x. Nun setzen wir x auf *wahr*, wenn x in mehr Klauseln vorkommt als $\bar{x}$, ansonsten auf *falsch*. Danach vereinfachen wir die Formel entsprechend. Danach belegen wir die übriggebliebenen Variablen irgendwie.

Welchen Approximationsfaktor erhalten wir? Nehmen wir eine optimale Belegung ϕ. Diese erfüllt eine gewisse Menge an Klauseln. TODO

b) Wir können einfach einen optimalen Spannbaum berechen und den Reisenden um den Baum herumlaufen lassen (in einer Eulertour). Dabei wir aber jede Kante zweimal besucht, was nicht erlaubt. Sobald er also eine Kante zum zweiten Mal rückwärts besuchen will, nimmt er eine Abkürzung und geht gleich direkt zum nächsten Knoten auf der Euler-Tour, der noch nicht besucht ist. Wegen der Dreiecksungleichung ist die entstehende Tour nicht teurer als zweimal die Kosten des Spannbaums. Eine TSP-Tour kann nicht billiger als ein optimaler Spannbaum sein, da sie einen Spannbaum enthält (wenn man eine ihrer Kanten entfernt). Daher haben wir einen Approximationsfaktor von 2.

Hausaufgabe 19 (10 Punkte)

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 $L \leq Z \leq OPT$.