Thomas Rothvoß
Location: ELD120 Discussion: 24.03.10

Exercises

Approximation Algorithms

Spring 2010

Sheet 4

Note: This is just one way, a solution could look like. We do not guarantee correctness. It is your task to find and report mistakes.

Exercise 1

Consider again the MINCONGESTION problem, where a directed graph $G = (V, E)$ with demand pairs (s_i, t_i) for $i = 1, \dots, k$ is given and one aims at finding s_i-t_i paths P_i that minimize the congestion $\max_{e \in E} |\{i : e \in P_i\}|$. Show that the algorithm presented in the lecture gives a $O(1)$-approximation (with high probability) if $k \geq (\log_2 |E|) \cdot |E|$ (and $s_i \neq t_i$ for all $i = 1, \dots, k$).

Hint: Which lower bound on the fractional congestion from the linear program do you obtain using the additional assumption $k \geq (\log_2 |E|) \cdot |E|$?

Solution:

Let (f, C) be an optimum fractional solution from the LP from the lecture. Every s_i-t_i path must use at least one edge, hence $\sum_{e \in E} \sum_{i=1}^k f_i(e) \geq k \geq |E| \cdot \log_2 |E|$. By the pigeonhole principle there must be an edge e^* with $C \geq \sum_{i=1}^k f_i(e^*) \geq \log_2 |E|$.

Next, fix any edge $e \in E$. We again let $X_i^e \in \{0, 1\}$ be the random variable, saying whether the s_i-t_i path uses e and $X^e := \sum_{i=1}^k X_i^e$ denotes the resulting congestion on e. We apply the Chernov bound

$$\begin{aligned} \Pr\Big[X^e > (\overbrace{2}^{=:\delta} + 1) \overbrace{C}^{\geq E[X^e]}\Big] &\leq \left(\frac{e^2}{(1+2)^{1+2}}\right)^{\overbrace{C}^{\geq \log_2 |E|}} \\ &\leq \left(\frac{1}{4}\right)^{\log_2 |E|} \\ &= \frac{1}{|E|^2} \end{aligned}$$

Hence

$$\Pr\Big[\bigvee_{e \in E} (X^e > 3 \cdot C)\Big] \leq |E| \cdot \frac{1}{|E|^2} \leq \frac{1}{|E|}$$

In words, we obtain a 3-approximation with probability at least $1 - \frac{1}{|E|}$.

Exercise 2
Suppose we have 3 machines and n jobs. If we run job j on machine $i \in \{1,2,3\}$ this takes a *processing time* of $p_{ij} \in \mathbb{Q}_+$. The MINIMUMMAKESPAN problem is to find a way to assign the jobs to machines such that the load of the highest loaded machine (the *makespan*) is minimized. Formally

$$OPT = \min_{J_1 \dot\cup J_2 \dot\cup J_3 = \{1,\dots,n\}} \Big\{ \max_{i=1,\dots,3} \Big\{ \sum_{j \in J_i} p_{ij} \Big\} \Big\}$$

Design an FPTAS for this problem (and prove an approximation guarantee of $1+\varepsilon$).
Hint: Similar to the KNAPSACK FPTAS it is a good idea to first round the running times in a suitable way. Then apply dynamic programming.

Solution:
W.l.o.g. we scale the running times until $p_{\max} := \max_{j=1,\dots,n} \min_{i=1,\dots,3}\{p_{ij}\} = n/\varepsilon$. Clearly $OPT \geq p_{\max} = n/\varepsilon$. Furthermore we obtain the bound $OPT \leq n \cdot p_{\max} = n^2/\varepsilon$ (if we schedule all jobs on its fastest machine).

Let $p'_{ij} := \lfloor p_{ij} \rfloor$ be rounded running times. Let OPT' be the value of the optimum makespan for the rounded running times. Note that $OPT' \leq OPT$. We claim that we can compute OPT' in polynomial time by dynamic programming. Consider table entries

$$A(j,b_1,b_2,b_3) = \begin{cases} 1 & \text{One can distribute jobs } 1,\dots,j \text{ to the machines s.t.} \\ & \text{the load on machine } i \text{ is at most } b_i \\ 0 & \text{otherwise} \end{cases}$$

for $j = 1,\dots,n$, $b_1,b_2,b_3 \in \{0,\dots,n \cdot p'_{\max}\}$. We obtain the base cases $A(j,b_1,b_2,b_3) = 0$ if $b_i < 0$ for some $i = 1,2,3$ and $A(0,b_1,b_2,b_3) = 1$ for $b_1,b_2,b_3 \geq 0$. We compute the remaining entries using the recursion

$$A(j,b_1,b_2,b_3) = A(j-1,b_1-p_{1j},b_2,b_3) \vee A(j-1,b_1,b_2-p_{2j},b_3) \vee A(j-1,b_1,b_2,b_3-p_{3j})$$

Then
$$OPT' = \min\{b \in \{0,\dots,n \cdot p_{\max}\} \mid A(n,b,b,b) = 1\}$$

Let $J_1 \dot\cup J_2 \dot\cup J_3 = \{1,\dots,n\}$ be the reconstructed solution of value OPT'. If we use the same solution for the original running times this costs

$$\max_{i=1,\dots,3} \Big\{ \sum_{j \in J_i} p_{ij} \Big\} \leq \max_{i=1,\dots,3} \Big\{ \sum_{j \in J_i} (p'_{ij} + 1) \Big\} \leq OPT' + n \leq (1+\varepsilon) \cdot OPT$$

Each table entry can be computed in $O(1)$, hence the algorithm needs time $O(1) \cdot n \cdot (n^2/\varepsilon)^3 = O(n^7/\varepsilon^3)$.
