University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger, P. Schneider

Algorithms Theory
Sample Solution Exercise Sheet 12

Due: Tuesday, 2nd of February 2021, 4 pm

Exercise 1: Load Balancing *(10 Points)*

Consider the load balancing problem as introduced in the lecture with m machines $M_1, \dots, M_m$ and n jobs with processing times $t_1, \dots, t_n > 0$. Let $L := \sum_{i=1}^{n} t_i$ be the total processing time.

(a) Let $m = 2$. Show how to compute a scheduling whose makespan is a $(1 + \varepsilon)$-approximation of that of an optimal scheduling in time $O(n^3/\varepsilon)$. *(3 Points)*

Hint: Try utilize the PTAS for the Knapsack problem.

(b) Let m be arbitrary. Assume for now that all processing times t_i are integer. Show that the load balancing problem can be solved *exactly* in time $O(nmL^m)$. *(4 Points)*

Hint: You can use Dynamic Programming.

(c) Use the algorithm from part b) to develop a PTAS for the load balancing problem in case the number of machines is a constant (i.e., $m = O(1)$). More specifically, show that one can compute a $(1 + \varepsilon)$-approximation in time $O\left(\left(\frac{n}{\varepsilon}\right)^c\right)$ for some constant c (c can depend on m). *(3 Points)*

Hint: You can use the same scaling and rounding technique from the PTAS for Knapsack.

Sample Solution

(a) We reduce the 2-machine scheduling problem to an instance of the Knapsack problem. The jobs $1, \dots, n$ correspond to the items, where each item has weight and value $w_i = v_i = t_i$. We define the maximum weight of the Knapsack as $W := L/2$.

Let $S \subseteq \{1, \dots, n\}$ be a set of selected items. This corresponds to a solution for the 2-machine scheduling problem by scheduling all jobs in S on M_1 and the rest on M_2. By construction, M_2 always has more load: $v(S) \leq L - v(S)$. Therefore the makespan is $T(S) = L - v(S)$. So the quality of the obtained scheduling depends on the quality of the Knapsack solution S.

If S^* is an optimal selection of items (i.e., one that maximizes $v(S^*)$, s.t., $v(S^*) = w(S^*) \leq L/2$), then this corresponds to an optimal machine scheduling on M_1. This is due to the fact that maximizing the load of M_1 without going over $L/2$, minimizes the load of M_2 and therefore the optimal makespan is $T(S^*) = L - v(S^*)$.

As computing S^* is too costly (super-polynomial, unless $P = NP$), we run the Knapsack-PTAS on the instance constructed above, instead. This takes at most the time stated in the exercise. Let S be the solution from the PTAS. We have the guarantee $v(S) \geq (1 - \varepsilon)v(S^*)$. Then

$$\begin{aligned} T(S) &= L - v(S) \leq L - (1 - \varepsilon)v(S^*) = L - v(S^*) + \varepsilon v(S^*) \\ &\leq L - v(S^*) + \varepsilon(L - v(S^*)) = (1 + \varepsilon)(L - v(S^*)) = (1 + \varepsilon)T(S^*). \end{aligned}$$

(b) As hinted, we construct a solution using dynamic programming. The underlying concept is covered in prior lectures so we will restrict ourselves to defining the table which we want to fill, what the recursion and the base cases look like, how long it takes to fill the table and how we extract solution from it.

Let $\ell_1, \dots, \ell_m$ be *upper bounds* for the loads of $M_1, \dots, M_m$. Assume that we have at least $n > 1$ jobs and $m > 1$ machines, otherwise the problem is trivial. In that case it is never optimal to assign all jobs to one machine, thus we require $\forall i :\ 0 \le \ell_i < L = \sum_{i=1}^{m} \ell_i$. Let $A[i, \ell_1, \dots, \ell_m]$ be the table which we want to fill, which should contain the value 1 if there is a valid assignment of jobs $1, \dots, i$ to machines $M_1, \dots, M_m$ such that each machine has load at most $\ell_1, \dots, \ell_m$, else 0.

The base cases $i = 1$ are easy to compute. To fill out $A[1, \ell_1, \dots, \ell_m]$ we check if we can assign the first job to some machine M_j such that $t_i \le \ell_j$. This can all be done in time $\mathrm{O}(m)$. The cases $i > 1$ are traced back to the table entries with one job less, i.e., $A[i-1, \dots]$. The idea to compute $A[i, \ell_1, \dots, \ell_m]$ is to check if we can put job i on Machine M_j for each $j = 1, \dots, m$ and still find a valid scheduling.

For this we first check if $A[i-1, \ell_1, \dots, \ell_j - t_i, \dots, \ell_m] = 1$ for some $j \in \{1, \dots, m\}$. If yes, we set $A[i, \ell_1, \dots, \ell_m] := 1$. To catch marginal cases, we assume $A[i-1, \ell_1, \dots, \ell_j - t_i, \dots, \ell_m] = 0$ if $\ell_j – t_i < 0$ (i.e., we always return 0 when we query a table entry where one dimension is negative). If none of these conditions apply the required scheduling is impossible and we set $A[i, \ell_1, \dots, \ell_m] := 0$. Overall, we have to check m entries to compute $A[i, \ell_1, \dots, \ell_m]$ which takes $\mathrm{O}(m)$ timesteps.

Using dynamic programming, each table entry has to be computed at most once. The question is how many table entries there are. The number of possibilities of the first dimension of the table A is n. Since $0 \le \ell_i < L$, the number of combinations for $(\ell_1, \dots, \ell_m)$ is L^m. So the overall the number of table cells is $\mathrm{O}(nL^m)$ and given a running time of $\mathrm{O}(m)$ for the computation of one entry, the overall time is $\mathrm{O}(nmL^m)$.

To get the minimum makespan from the table, one has to compute from the last “hyper-row” $A[n, \dots]$ all the entries which are 1, and from those we have to choose the entry which has the smallest makespan. Formally:

$$\text{makespan} = \min_{\substack{\ell_1, \dots, \ell_m \text{ with} \\ A[n, \ell_1, \dots, \ell_m] = 1}} \max\bigl(\ell_1, \dots, \ell_m\bigr).$$

Evaluating the n-th “hyper-row” takes $\mathrm{O}\bigl(mL^m\bigr)$ time. If we want to extract a schedule from A, we first have to find the entry which defines the makespan as described above, and then backtrack the recursive path that leads to the computation of that entry, which lets us decide where each job is scheduled. The time required for that is less than computing the table in the first place.

(c) *Overview:* As suggested by the hint, we repeat the approach used for the PTAS of Knapsck. The idea is to first scale all processing times appropriately and then round them up to obtain integer processing times $\widehat{t}_1, \dots, \widehat{t}_n > 0$. Then we can use the procedure from part (b) to solve the integer instance of the problem. We show that the procedure has polynomial runtime in $\frac{n}{\varepsilon}$ (provided $m \in \mathrm{O}(1)$) and that the makespan of the obtained scheduling is at most $(1 + \epsilon)$ times larger than that of the optimal scheduling.

Detailed Solution: We define the scaled and rounded weights as $\widehat{t}_i := \left\lceil \frac{nt_i}{\varepsilon t} \right\rceil$, where $t := \max_{i=1}^{n} t_i$. We observe that $\widehat{t}_i \le \frac{nt_i}{\varepsilon t} + 1$. Then the total processing time $\widehat{t}$ of the transformed problem is

$$\widehat{L} := \sum_{i=1}^{n} \widehat{t}_i \le \sum_{i=1}^{n} \left(\frac{nt_i}{\varepsilon t} + 1 \right) = n + \frac{n}{\varepsilon} \sum_{i=1}^{n} \frac{t_i}{t} \le n + \frac{n^2}{\varepsilon} = \mathrm{O}(n^2/\varepsilon)$$

Now we run the algorithm from part (b) on this problem instance. The runtime is

$$\mathrm{O}\bigl(nm\widehat{L}^m\bigr) = \mathrm{O}\bigl(m \cdot n^{2m+1}/\varepsilon^m\bigr) = \mathrm{O}\bigl(n^{2m+1}/\varepsilon^m\bigr).$$

So the runtime requirements of this task are fulfilled if we choose $c = 2m + 1 \in \mathrm{O}(1)$.

Let S be the schedule we get from this procedure. Note that S is *optimal* with respect to the $\widehat{t_i}$ but not with respect to the t_i. Let S^* be an optimal scheduling with respect to the t_i. Let $T(S)$ and $T(S^*)$ be the makespan of both schedules with respect to the unmodified processing times t_i and let $\widehat{T}(S)$ and $\widehat{T}(S^*)$ be the according values with respect to the modified $\widehat{t_i}$.

We define M, M^* as the machines which have the *largest* load of $T(S), T(S^*)$ under the schedules S and S^* respectively, w.r.t. the processing times t_i. Analogously, let $\widehat{M}, \widehat{M}^*$ be the machines that have *largest* load of $\widehat{T}(S), \widehat{T}(S^*)$ under the schedules S and S^*, respectively, w.r.t. the $\widehat{t_i}$.

Let $J_{S'}(M') \subseteq \{1, \ldots, n\}$ be the set of jobs scheduled on machine M' under some schedule S'. For brevity, we define the jobs assigned to the above machines under the corresponding schedules as follows $J := J_S(M), \widehat{J} := J_S(\widehat{M}), K := J_{S^*}(M^*), \widehat{K} := J_{S^*}(\widehat{M}^*)$. Then we have

$$\begin{aligned}
T(S) &= \sum_{i \in J} t_i = \frac{\varepsilon t}{n} \sum_{i \in J} \frac{n t_i}{\varepsilon t} \leq \frac{\varepsilon t}{n} \sum_{i \in J} \left\lceil \frac{n t_i}{\varepsilon t} \right\rceil \\
&= \frac{\varepsilon t}{n} \sum_{i \in J} \widehat{t_i} \leq \frac{\varepsilon t}{n} \sum_{i \in \widehat{J}} \widehat{t_i} && \textit{M has load less equal as $\widehat{M}$ w.r.t. $\widehat{t_i}$} \\
&\leq \frac{\varepsilon t}{n} \sum_{i \in \widehat{K}} \widehat{t_i} = \frac{\varepsilon t}{n} \sum_{i \in \widehat{K}} \left\lceil \frac{n t_i}{\varepsilon t} \right\rceil && \textit{$\widehat{M}$ has load less equal as $\widehat{M}^*$ w.r.t. $\widehat{t_i}$} \\
&\leq \frac{\varepsilon t}{n} \sum_{i \in \widehat{K}} \left(\frac{n t_i}{\varepsilon t} + 1 \right) = \sum_{i \in \widehat{K}} t_i + \sum_{i \in \widehat{K}} \frac{\varepsilon t}{n} \\
&\leq \sum_{i \in K} t_i + \sum_{i \in \widehat{K}} \frac{\varepsilon t}{n} = T(S^*) + \sum_{i \in \widehat{K}} \frac{\varepsilon t}{n} && \textit{$\widehat{M}^*$ has load less equal as M^* w.r.t. t_i} \\
&\leq T(S^*) + \varepsilon \textstyle\sum_{i=1}^{n} \frac{t}{n} \leq T(S^*) + \varepsilon t = T(S^*) + \varepsilon t \leq (1+\varepsilon) T(S^*).
\end{aligned}$$

Exercise 2: Minimum Set Cover with Bounded Frequency *(10 Points)*

Consider a set cover instance with a set of elements X and a family of subsets $\mathcal{S} \subseteq 2^X$ of X. A set cover instance $(x, \mathcal{S})$ is said to have *frequency* f iff every element $x \in X$ is contained in at most f of the sets $S \in \mathcal{S}$.

(a) First consider the unweighted version of the set cover problem. Give a polynomial-time algorithm with approximation ratio at most f for the minimum set cover problem with frequency f. *(3 Points)*

Hint: The special case where $f = 2$ is equivalent to the minimum vertex cover problem.

(b) Now we consider the weighted set cover problem, where every set $S \in \mathcal{S}$ has a weight $w_S > 0$. We first consider an assignment of prices $p_x \geq 0$ to all elements $x \in X$. We call a price assignment *valid* if for every set $S \in \mathcal{S}$: $\sum_{x \in S} p_x \leq w_S$. Show that for any valid price assignment $p_x \geq 0$ for all $x \in X$, it holds that

$$\sum_{x \in X} p_x \leq \sum_{S \in \mathcal{C}} w_S \quad \text{for every set cover } \mathcal{C}. \qquad \textit{(2 points)}$$

(c) We say that a valid price assignment $p_x \geq 0$ for $x \in X$ is *maximal* if it is not possible to increase some price p_x without violating the validity condition of the price assignment. A maximal price assignment can be computed by a simple greedy algorithm. Show that a maximal price assignment can be used to derive a set cover algorithm with approximation ratio at most f. *(5 Points)*

Hint: Come up with a family of sets $\mathcal{C} \subseteq \mathcal{S}$ based on a maximal price assignment and show that it covers X. Then use the property from part b) to show the approximation ratio.

Sample Solution

(a) *Intuition*: The special case $f = 2$ equals vertex cover problem where edges are the elements and the nodes correspond to sets which "contain" their incident edges. In the lecture this was solved by finding a maximal matching of the edges and then simply adding all the endpoints of the added edges to the vertex cover. For the f-frequency set cover problem we do something similar.

Approach: We choose a *maximal independent* set of elements $I \subseteq X$, that is no two elements $x, y \in I$ "share" a set S, i.e., $x \notin S$ or $y \notin S$ for all $S \in \mathcal{S}$[1] such that no element can be added anymore without violating this condition (maximality). A simple greedy approach gives such a maximal $I \subseteq X$ in polynomial time ($O(|X| \cdot |\mathcal{S}|)$ suffices).

Then we choose $\mathcal{C} \subseteq \mathcal{S}$ as all sets which contain at least one element of I. This corresponds to adding both endpoints of all edges in the matching to the vertex cover. Formally: $\mathcal{C} := \{S \in \mathcal{S} \mid \exists x \in I : x \in S\}$. We show that $\mathcal{C}$ is a set cover. Each element from X is either in I in which case it is covered by some $S \in \mathcal{C}$. Or otherwise, it must be in some set $S \in \mathcal{C}$ which contains an element $x \in I$. Because if not we could add x to I without violating the independence, thus I was not maximal, a contradiction.

Approximation ratio: We generalize the proof for vertex cover from the lecture to the maximal independent set $\mathcal{C} \subseteq \mathcal{S}$ to show an approximation ratio of f. Let $\mathcal{C}^* \subseteq \mathcal{S}$ be the minimum set cover. Due to the independence condition, there are no two elements in I that are contained in the same set $S \in \mathcal{S}$. For that reason, *any* set cover $\mathcal{C}' \subseteq \mathcal{S}$ must contain one distinct set for each $x \in I$ in order to cover I. Hence $|I| \leq |\mathcal{C}'|$ and in particular $|I| \leq |\mathcal{C}^*|$.

But since each element $x \in I$ is contained in at most f sets the size of the setcover $\mathcal{C}$ which we computed above is at most $|\mathcal{C}| \leq f \cdot |I|$ (recall that for each $x \in I$ we add all of the at most f sets containing x to $\mathcal{C}$). Overall we have $|\mathcal{C}| \leq f \cdot |I| \leq f \cdot |\mathcal{C}^*|$.

(b) By definition of a set cover $\mathcal{C}$ we have $\bigcup_{S \in \mathcal{C}} S = X$. Hence we can modify the sum as follows.

$$\sum_{x \in X} p_x \leq \sum_{S \in \mathcal{C}} \sum_{x \in S} p_x \leq \sum_{S \in \mathcal{C}} w_S.$$

(c) Let $p_x, x \in X$ be a maximal valid price assignment. There must be some $S \in \mathcal{S}$ with $\sum_{x \in S} p_x = w_S$ (otherwise the price assignment was not maximal). We suspect that these sets must be important, and define $\mathcal{C} := \{S \in \mathcal{S} \mid \sum_{x \in S} p_x = w_S\}$. First we prove that this is actually a set cover and second that it is at most f times heavier than the minimum weight set cover.

For a contradiction, assume that x is not covered, i.e., $x \notin \bigcup_{S \in \mathcal{C}} S$. Then for all S with $x \in S$ we have that $\sum_{x \in S} p_x < w_S$ by definition of $\mathcal{C}$. That means that we could slightly increase p_x without violating the validity of the price assignment. But then the price assignment was not maximal, a contradiction.

Let $\mathcal{C}^*$ be a minimum weighted set cover. For the approximation ratio we have to again utilize the fact that each element is contained in at most f sets of $\mathcal{S}$ in middle step of the following computation. Additionally, we use that part (b) holds for any set cover, in particular $\mathcal{C}^*$

$$\sum_{S \in \mathcal{C}} w_S = \sum_{S \in \mathcal{C}} \sum_{x \in S} p_x \leq f \cdot \sum_{x \in X} p_x \overset{(b)}{\leq} \sum_{S \in \mathcal{C}^*} w_S$$

[1]This corresponds to the matching condition where two edges (=elements) share a node (=set) as endpoint.