1 Example Class (23rd May 2019, 16.15-17.30)

Question 1. We consider the KNAPSACK-problem, where we are given n items each of which comes with an integral weight $w_i > 0$ and integral value $v_i > 0$. The knapsack has capacity C and the goal is to fill the knapsack so as to maximise its total value. Further, we denote by $OPT \leq \max\{C, \sum_{i=1}^{n} v_i\}$ the value obtained by an optimal solution. As a side remark, we may assume that for all items $1 \leq i \leq n$, $w_i \leq C$.

1. Design a simple ("the arguably most natural") greedy algorithm and analyse its approximation ratio.
2. Consider a modified greedy algorithm, which takes the better solution of the algorithm from Part 1 and item with the largest value. Prove that the approximation ratio of this new algorithm is two.
   Hint: One way of establishing this approximation ratio involves the following steps:
   (a) First define a LP relaxation of the knapsack problem.
   (b) Find the optimum solution of the LP relaxation.
   (c) Use the result from (b) to argue that the solution of the algorithm is within a factor of two of the optimum LP solution.
3. Consider the dynamic programming technique. Derive two algorithms based on this technique that achieve a runtime of $O(n \cdot C)$ and $O(n \cdot OPT)$, respectively.
   (Question: Why are both of these algorithms *not* polynomial-time?)
4. Design a FPTAS based on the second dynamic programming algorithm with runtime $O(n^3/\epsilon)$.
   Hint: Round down all values so that they will lie in a suitable range (depending of course on $\epsilon > 0$!).

Answer 1. 1. The most natural greedy algorithm is to sort all items non-increasingly according to their value/weight ratio:

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n},$$

and then greedily taking as many items as possible as long as we are not exceeding the capacity C.

Unfortunately the approximation ratio can be arbitrarily bad. Consider, for example, the following instance: $w_1 = 1, v_1 = 2, w_2 = C, v_2 = C$. The greedy algorithm would only return a solution with value 2, whereas the optimum solution would achieve a value of C.

2. (a) We will be working with the same ordering of the n items according to their value/weight ratio as in the previous part. With this, the LP-relaxation looks

as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} v_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i y_i \le C \\ & 0 \le y_i \le 1 \qquad \text{for all } 1 \le i \le n \end{aligned}$$

(b) The optimal solution of this LP will assign as much value as possible to the items with largest value/weight ratio and fill the knapsack exactly up to C (this is a.k.a. *fractional* knapsack problem). Therefore the optimum y^* satisfies, $y_1^* = 1, y_2^* = 1, \ldots, y_{k-1}^* = 1, y_k^* = \frac{C-(w_1+w_2+\cdots+w_{k-1})}{w_k}$, where k is the largest integer such that $w_1 + w_2 + \cdots + w_{k-1} \le C$.

(c) Note that the *unmodified* greedy algorithm would yield a profit of $v_1 + v_2 + \cdots + v_{k-1}$. Further, the *modified* greedy algorithm would yield a profit of $\max\{v_1 + v_2 + \cdots + v_{k-1}, v_{\max}\}$, where $v_{\max}$ is the value of the most valuable item (recall that we have made the assumption that $v_{\max} \le C$, so taking the most valuable item is always feasible!). Since $v_{\max} \ge v_k$, the profit of the *modified* greedy algorithm is a least

$$\max\{v_1 + v_2 + \cdots + v_{k-1}, v_k\} \ge \frac{1}{2} \cdot (v_1 + v_2 + \cdots + v_k)\,.$$

On the other hand, the objective value of the optimal LP is

$$v_1 + v_2 + \cdots + v_{k-1} + \underbrace{\frac{C - (w_1 + w_2 + \cdots + w_{k-1})}{w_k}}_{\le 1} \cdot v_k \le v_1 + v_2 + \cdots + v_k.$$

Hence the approximation ratio of the *modified* greedy algorithm is at most 2.

3. For the first dynamic programming solution, let $opt(j, c)$ be the optimal knapsack solution restricted to items $\{1, 2, \ldots, j\}$ and capacity $c \le C$. To find a recurrence formula for $opt(j, c)$, consider the j-th item and note that this item could be part of a optimal solution or not (or possibly both):

$$opt(j,c) = \begin{cases} 0 & \text{if } j = 0. \\ opt(j-1,c) & \text{if } w_j > c, \\ \max\{opt(j-1,c), v_j + opt(j-1, c-w_j)\} & \text{otherwise.} \end{cases}$$

This directly leads to a $O(n \cdot C)$ algorithm by filling values of a two-dimensional array with dimensions n and C.

The second dynamic programming approach will take a “dual” approach. We let $opt(j, v)$ be the minimum knapsack weight that yields a total value of exactly v using only the items in $\{1, 2, \ldots, j\}$. For the recurrence formula, the two cases are again whether a optimal solution includes item j or not:

$$opt(j, v) = \begin{cases} 0 & \text{if } j = 0. \\ opt(j-1, v) & \text{if } v_j > v, \\ \min\{opt(j-1, v), w_j + opt(j-1, v - v_j)\} & \text{otherwise.} \end{cases}$$

(Notice the switch of the roles of w_j and v_j compared to the first dynamic programming solution.) Again, using a bottom-up approach, all values for $opt(j, v)$ with $1 \leq j \leq n$ and $1 \leq v \leq OPT$ can be computed leading to an algorithm with running time $O(n \cdot OPT)$ (Although it is not strictly needed for the PTAS, this running time can be achieved by computing all values up until $opt(n, OPT + v_{\max})$ and stopping when $opt(n, .)$ does not change. Note that $OPT + v_{\max} \leq 2OPT$ thanks to the assumption on $v_{\max}$)

Both algorithms are not *polynomial-time*, since the running time is not polynomial in the input-size (for that, the dependence should be poly-logarithmic in C or OPT!).

We will now describe a FPTAS for the Knapsack Problem.

KNAPSACK-FPTAS$(\epsilon, n, C, v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_n)$

1: **For** each item $i = 1, 2, ..., n$ set $\overline{v}_i = \lfloor \frac{v_i}{\alpha} \rfloor$, where $\alpha = \frac{\epsilon \cdot v_{\max}}{n}$ is the scaling factor.
2: Run the exact dynamic programming algorithm on the rounded instance to obtain a subset $\overline{S}^*$
3: **Return** $\overline{S}^*$

Let us now analyse this algorithm, first the runtime and then the approximation ratio.

- **Running Time.** Recall that the exact dynamic programming algorithm has a runtime of $O(n \cdot \overline{OPT})$. The optimum solution of the rounded instance is at most $\overline{OPT} \leq n \cdot n \cdot \overline{v}_{\max} = n^2 \cdot \lfloor 1/\epsilon \cdot n \rfloor = O(n^3/\epsilon)$.

- **Approximation Ratio.** Let $S^* \subseteq \{1, \ldots, n\}$ be the optimal set of items in the original instance and $\overline{S}^* \subseteq \{1, \ldots, n\}$ be the optimal set of items in the rounded instance. Note that $\sum_{i \in \overline{S}^*} v_i$ is the value of the computed solution. Then,

$$\begin{aligned}
\sum_{i \in \overline{S}^*} v_i &\geq \sum_{i \in \overline{S}^*} \alpha \cdot \overline{v}_i && \text{(by definition of the rounded instance)} \\
&\geq \sum_{i \in S^*} \alpha \cdot \overline{v}_i && \text{(since } \overline{S}^* \text{ is the optimum for the rounded instance)} \\
&\geq \sum_{i \in S^*} (v_i - \alpha) && \text{(by the definition of scaling)} \\
&\geq \sum_{i \in S^*} v_i - \alpha \cdot n \\
&\geq OPT - \frac{\epsilon \cdot v_{\max}}{n} \cdot n \\
&\geq OPT - \frac{\epsilon \cdot OPT}{n} \cdot n && \text{(since } v_{\max} \leq OPT) \\
&= (1 - \epsilon) \cdot OPT,
\end{aligned}$$

where in the third inequality we have used the simple fact that $\lfloor x/\alpha \rfloor \geq x/\alpha - 1$ implies $\alpha \cdot \lfloor x/\alpha \rfloor \geq x - \alpha$.