Advanced Algorithms 2024 23/09, 2024

Sample Solutions 01

Lecturer: Johannes Lengler *Teaching Assistant: Andor Vari-Kakas*

1 Monotone Submodular Maximization

Let $\mathcal{U} = \{e_1, e_2, \dots, e_n\}$. Our goal is to find a set $S \subseteq \mathcal{U}$ with $|S| \leq k$ which maximizes $f(S)$. It is sufficient to consider sets of size exactly k, as f is monotone. A trivial idea is to compare the profits of all sets whose size is k, but this takes super-polynomial time (because we have $\binom{n}{k}$ sets of size k to check), for $k = \omega(1)$.

A simple polynomial-time algorithm is the greedy algorithm. It starts with $S_0 = \emptyset$. At each step i, it adds to S_{i-1} the element $e_{t(i)}$ with the largest marginal gain. That is, it chooses $e_{t(i)}$ which maximizes $f(S_{i-1} \cup \{e_{t(i)}\}) - f(S_{i-1})$. At last, it outputs S_k.

Next, let's analyze how it approximates the optimal solution. Denote the optimal set as $OPT = \{y_1, y_2, \dots, y_k\}$. Our goal is to show that in each step there is an element, which decreases the difference to the optimum solution.

$$\begin{aligned}
f(\mathbf{OPT}) &\leq f(\mathbf{OPT} \cup S_i) && \text{monotone}\\
&= f(S_i) + [f(S_i \cup \{y_1\}) - f(S_i)]\\
&\quad + f(S_i \cup \{y_1, y_2\}) - f(S_i \cup \{y_1\})\\
&\quad + \cdots\\
&\quad + f(S_i \cup \{y_1, \dots, y_k\}) - f(S_i \cup \{y_1, \dots, y_{k-1}\})\\
&\leq f(S_i) + [f(S_i \cup \{y_1\}) - f(S_i)] + \cdots + [f(S_i \cup \{y_k\}) - f(S_i)] && \text{submodular}\\
&= f(S_i) + \sum_{t=1}^{k} f(S_i \cup \{y_t\}) - f(S_i)
\end{aligned}$$

The first inequality is obtained by just adding each element of OPT to the set S_i, one at a time (note that some of these terms can be zero, e.g. if some y_t is already contained in S_i. For the second to last line, we used fact that f is submodular, allowing us to bound the marginal gain in each step, by the marginal gain of adding the element to S_i. Rearranging, we get

$$f(\mathbf{OPT}) - f(S_i) \leq \sum_{t=1}^{k} f(S_i \cup \{y_t\}) - f(S_i).$$

We know that there must be an term in the sum whose value is above the average $\frac{1}{k}\sum_{t=1}^{k} f(S_i \cup \{y_t\}) - f(S_i)$, which means there exists $j \in [k]$, s.t.

$$\begin{aligned}
f(S_i \cup \{y_j\}) - f(S_i) &\geq \frac{1}{k}(f(\mathbf{OPT}) - f(S_i))\\
f(S_{i+1}) - f(S_i) &\geq \frac{1}{k}(f(\mathbf{OPT}) - f(S_i)) && \text{greedy}\\
f(\mathbf{OPT}) - f(S_{i+1}) &\leq (1 - \frac{1}{k})(f(\mathbf{OPT}) - f(S_i))\\
f(\mathbf{OPT}) - f(S_k) &\leq (1 - \frac{1}{k})^k f(\mathbf{OPT}) \leq \frac{1}{e} f(\mathbf{OPT})\\
f(S_k) &\geq (1 - \frac{1}{e}) f(\mathbf{OPT}).
\end{aligned}$$

The second line is because Greedy always adds the element with the largest maximal gain, $f(S_{i+1}) \geq f(S_i \cup \{y_j\})$. Finally, we reach the conclusion that Greedy gives a solution whose value is better than $(1 - \frac{1}{e})$ times of the optimal profit.

2 2-Approximation for Knapsack (Vazirani 8.2)

Let O be a set containing the elements of an optimal solution. Define $K^- := \{a_1, a_2, \ldots, a_{k-1}\}$ and $K := \{a_1, a_2, \ldots, a_k\}$ with $\text{size}(K) =: C$.

Proof by contradiction: Assume the given algorithm is not a 2-approximation. Then, $\text{profit}(K^-) < \frac{\text{profit}(O)}{2}$ and $\text{profit}(\{a_k\}) < \frac{\text{profit}(O)}{2}$. Thus:

$$\text{profit}(K^-) + \text{profit}(\{a_k\}) = \text{profit}(K) < \text{profit}(O).$$

As K does not fit into the given budget B, we have $C > B$. In addition, K is an optimal solution to another knapsack problem of budget C (with the same item set), as we have the highest profit to size ratio and the highest possible size for this knapsack. But then we have a contradiction, as a bigger knapsack (budget C) cannot have an optimal solution with less profit than the optimal solution of a smaller knapsack (budget B) on the same item set. Therefore the given algorithm is a 2-approximation.

Another way to solve this problem would be to argue about the fractional version of Knapsack, where we are allowed to add fractional values of elements.

3 Bin Covering (Vazirani 9.7)

Rounding Without loss of generality, assume the items are ordered by size $a_1 \geq a_2 \geq \cdots \geq a_n$. We partition the items into groups of size at most $Q := \lfloor \lfloor n\varepsilon \rfloor \varepsilon \rfloor$ as follows: group G_1 contains the items $a_1, \ldots a_Q$, group G_2 contains $a_{Q+1}, \ldots, a_{2Q}$, and so on. Note, that the number of groups $k_\varepsilon := \lceil \frac{n}{Q} \rceil$ is a constant only depending on ε. Next we *round-down* the size of each item a_j in group G_l to $a'_j := a_{l \cdot Q}$, the size of the smallest element in G_l. We refer to the original problem instance as $\mathcal{I}$ and to the problem instance with *rounded-down* item sizes a' as $\mathcal{D}$. We can now run the following brute-force algorithm on $\mathcal{D}$.

Brute-force algorithm for bin-covering We first enumerate all possible ways to cover a bin. Because all items have size $\geq c$ it suffices to try out all combinations of up to $M := \lceil \frac{1}{c} \rceil$ items. Because there are only k_ε many different item sizes the number of valid combinations is at most $R_\varepsilon := \binom{M+k_\varepsilon}{M}$, which is a constant independent of n. Note that with n items we can cover at most n bins. Thus we can next enumerate all possible ways to cover up to n bins, each with any of the R_ε possible item combinations from before. There are $\binom{n+R_\varepsilon}{n} \leq (n+R_\varepsilon)^{R_\varepsilon} = \mathcal{O}(n^{R_\varepsilon})$ such combinations thus our algorithm terminates in time polynomial in n.

Analysis Note that the solution returned by the brute-force algorithm is optimal for the rounded problem instance $\mathcal{D}$, thus we refer to it as $OPT(\mathcal{D})$. Because we rounded all item sizes *down*, each set of items in $OPT(\mathcal{D})$ that covers a bin in $\mathcal{D}$ still covers a bin in $\mathcal{I}$. We claim that $OPT(\mathcal{D})$ is a $(1 - \varepsilon)$ approximation of the optimal solution to the original problem $OPT(\mathcal{I})$.

To show this we consider another problem instance $\mathcal{U}$ where we *round-up* all item sizes, i.e. similar to before we assign sizes a'' such that each element gets the size of the largest element in its group. Because we only increase item sizes in $\mathcal{U}$ with respect to the original problem $\mathcal{I}$, it is clear that $|OPT(\mathcal{U})| \geq |OPT(\mathcal{I})|$. We now argue that we can cover all up to Q bins of $OPT(\mathcal{U})$ using the *rounded-down* sizes a'. Consider the solution $OPT(U) =: \{S_1, \ldots, S_m\}$ (where each S_i covers one bin) and any element $a_j \in S_i$. If $a_j \in G_l$ for some $l > 1$ we can replace it with

any *rounded-down* element in G_{l-1}, because a''_j is at most a'_i for any $a_i \in G_{l-1}$. Because all groups except G_k have size Q and G_k has size $\leq Q$, we can do this for all elements in groups with index > 1. There are Q items in G_1, these items are distributed over at most Q bins. If we simply declare these bins as not covered, we get that $|OPT(\mathcal{D})| \geq |OPT(\mathcal{U})| - Q$.

Now let $\varepsilon \leq \frac{1}{\lceil \frac{1}{c} \rceil}$. Then it holds $\lfloor \varepsilon n \rfloor \leq \left\lfloor \frac{n}{\lceil \frac{1}{c} \rceil} \right\rfloor \leq |OPT(\mathcal{I})|$ where the second inequality holds because any $\left\lceil \frac{1}{c} \right\rceil$ elements cover a bin. Moreover, we get

$$Q = \lfloor \lfloor n\varepsilon \rfloor \varepsilon \rfloor \leq \varepsilon \lfloor n\varepsilon \rfloor \leq \varepsilon \cdot |OPT(\mathcal{I})|.$$

And thus, as desired

$$\begin{aligned} |OPT(\mathcal{D})| &\geq |OPT(\mathcal{U})| - Q \\ &\geq |OPT(\mathcal{I})| - Q \\ &\geq (1-\varepsilon)|OPT(\mathcal{I})|. \end{aligned}$$