CS532 Homework 3

Qilin Ye

October 22, 2024

In this problem set I have discussed some high-level ideas with Rico Zhu. Problems include Q1.(ii), Q2.(ii), and Q4.(i).

Solution to problem 1. (i) Let S be a set of constraints and define $f(S)$ to be the feasible region under constraints in S. Asume that $V \neq \varnothing$ and $V \cap B = \varnothing$. This means x^* violates some constraints in $H \backslash B$ (and $H \backslash B$ only). Observe that by definition, x^* is the unique point lying in the intersections of the hyperplanes defined by constraints in B. Therefore, the feasible regions defined by B and $H \backslash B$ are disjoint, for

$$f(B) \cap f(H \backslash B) \subset f(B) = \{x^*\}$$

but we also know that x^* violates something in $H \backslash B$, i.e., $x^* \notin f(H \backslash B)$. Therefore, $f(B) \cap f(H \backslash B) = \varnothing$. But this contradicts the assumption that H is feasible. Therefore, if $V \neq \varnothing$ then $V \cap B \neq \varnothing$.

(ii) Let $H^{(t)}$, $V^{(i)}$, $B^{(i)}$ be the sets after iteration i. Initially $w(H^{(0)}) = n$, and the recurrence relation is given by

$$w(H^{(t)}) = w(H^{(t-1)}) + w(V^{(t-1)}) \leqslant w(H^{(t-1)}) + \frac{2w(H^{(t-1)})}{9d} = w(H^{(t-1)})(1 + 2/(9d)).$$

Therefore, applying the inequality $1 + x \leqslant e^x$, we see

$$w(H^{(i)}) \leqslant n(1 + 2/(9d))^i \leqslant n \exp(2i/9d).$$

As for lower bounding $w(B^{(t)})$, we observe from part (i) that $V^{(t)} \cap B^{(t)}$ is always nonempty whenever $V^{(t)}$ is. That is, at least one constraint in $B^{(t)}$ is doubled. Certainly the growth of $w(B^{(t)})$ is lower bounded by the round-robin case over B, whose size is bounded by d:

$$w(B^{(i)}) \geqslant d2^{i/d}.$$

(iii) Clearly, $w(B^{(t)}) \leqslant w(H^{(t)})$ for each t, but from (ii) we know that the LHS grows at rate $\exp(t/d \cdot \log 2)$ and the RHS at rate $\exp(t/d \cdot (2/9))$. Since $\log 2 > 2/9$, indeed LHS grows faster. Therefore, we simply need to solve for the inequality

$$d \exp(t/d \cdot \log 2) \leqslant n \exp(t/d \cdot 2/9) \implies t = \mathcal{O}(d \log n) = \mathcal{O}(\log n)$$

since d is a constant.

(iv) Also $\mathcal{O}(\log n)$, since with the assumption, each MWu is expected to take no more than 2 iterations to be triggered. Each iteration involves solving a LP defined by a random subset R of size $9d^2$, which is constant. Therefore the time to solve each LP is $\mathcal{O}(1)$, and the total runtime is still asymptotically bounded by the number of iterations, or $\mathcal{O}(\log n)$.

Solution to problem 2. (i) We consider a three-dimensional DP, where $f(i, c_1, c_2)$ represents the maximum profit achieved after using first i item, subject to knapsack 1 having remaining capacity c_1 and knapsack 2 with remaining capacity c_2. The initialization is given by $f(0, c_1, c_2) = 0$ for all $(c_1, c_2) \in [B_1] \times [B_2]$. The recurrence relation is given by the better of the three cases: (i) not pick the current item at all, (ii) let knapsack 1 pick it if budget allows, or (iii) let knapsack 2 pick it it budget allows. Put together,

$$f(i, c_1, c_2) = \max \begin{cases} f(i-1, c_1, c_2) \\ f(i-1, c_1 - s_i, c_2) + p_i & \text{if } c_1 \geqslant s_i \\ f(i-1, c_1, c_2 - s_i) + p_i & \text{if } c_2 \geqslant s_i. \end{cases}$$

With memoization, the final solution, $f(n, B_1, B_2)$ can be computed in $\mathcal{O}(n \cdot B_1 \cdot B_2)$ time, which is pseudopolynomial.

(ii) Let $\mathrm{OPT}, \mathrm{ALG}$ denote the total optimal profit and the profit returned by the proposed algorithm. Let $\mathrm{OPT}_i, \mathrm{ALG}_i$ denote the profit gained in iteration i, respectively.

Further, after iteration i, define R_i to be the remaining profit after packing the first i knapsacks via OPT. It follows that $R_1 = \mathrm{OPT} - \mathrm{OPT}_1$ and similarly $R_i = R_{i-1} - \mathrm{OPT}_i$. At each iteration i, the remaining optimal profit is distributed over the remaining knapsacks, so the maximum profit that can be packed into the i^{th} knapsack is at least $R_{i-1}/(k-i+1)$, and since ALG is α-approximate,

$$\mathrm{ALG}_i \geqslant \alpha \cdot \frac{R_{i-1}}{k-i+1}.$$

Therefore, the recurrence relation is given by

$$R_i = R_{i-1}\left(1 - \frac{\alpha}{k-i+1}\right),$$

and

$$R_k = \mathrm{OPT} \cdot \prod_{i=1}^{k}\left(1 - \frac{\alpha}{k-i+1}\right) \leqslant \mathrm{OPT} \cdot \exp\left(-\alpha \sum_{i=1}^{k} \frac{1}{k-i+1}\right) \leqslant \mathrm{OPT} \cdot e^{-\alpha},$$

which means that the total profit of $\mathrm{ALG} \geqslant \mathrm{OPT}(1 - e^{-\alpha})$, as claimed.

Solution to problem 3. (i) We first show the first claim, that there exists an optimal schedule where all on-time jobs are completed before late jobs. To this end consider any optimal schedule in which some late job L is schedule (and completed) before some on-time job O. An exchange argument shows that if we were to swap the execution order of L and O, then O is guaranteed to still be on-time while L can either remain late or become on-time also. In either cases, swapping does not make performance deteriorate, which shows the claim.

Second, we show that on-time jobs can be scheduled in an earliest due date order. Suppose that there exists an optimal schedule ALG that schedules all on-time jobs first, but not in the order of earliest due first. Then in the execution orders of the on-time jobs, there exists an adjacent pair such that the first one has a later due date. Suppose these jobs are i, j with $d_i \geqslant d_j$. Further, let T denote the time when job i starts to be processed in ALG. We use another exchange argument and consider ALG' whose execution is identical to ALG except that now j immediately precedes i, not the other way around.

Certainly, j is executed even earlier so it will finish earlier and remain as on-time. The new finish time for i will be $T + p_j + p_i$. Observe this is precisely the old finish time of job j (since i is being run first), so by assumption $T + p_j + p_i \leqslant d_j \leqslant d_i$. This means even though job i is delayed, it is still finished by its deadline and hence an on-time job. Therefore ALG' is not worse than ALG w.r.t. the objective, and the proof is complete.

(ii) By (i), the job scheduling problem essentially reduces to a variant of Knapsack, where we want to maximize the total weight of on-time jobs, where these selected jobs are also sorted in earliest due dates, and ignore the rest (since late jobs do not contribute to our objective). The first step is therefore to sort jobs from earliest to latest due dates. Then, define a DP function $f(i, w)$ the minimum completion time required to achieve total weight w by scheduling a subset of the first i (sorted) jobs. The base case is given by $f(0, 0) = 0$. For each job $i = 1, \cdots, n$:

(1) We can either not include job i, giving the same objective value $f(i-1, w)$, or

(2) If the last job in $f(i-1, w)$ is finished before $d_i - p_i$, then there is enough time to execute job i, and in this case we have an objective value of $f(i-1, w-w_i) + p_i$.

To sum up:

$$f(i,w) = \begin{cases} f(i-1,w) & \text{if } w \leqslant w_i \text{ or } f(i-1,w-w_i)+p_i > d_i \\ \min(f(i-1,w), f(i-1,w-w_i)+p_i) & \text{otherwise.} \end{cases}$$

The runtime is $\mathcal{O}(nW)$, as claimed.

(iii) *I'm not sure how previous parts help here, but I will use discretization to give a $1-\epsilon$ approximate solution of Knapsack. Reference: my own scribed notes for both 632 and 532.* Let $\epsilon > 0$ be given. Define $W = \max w_i$ and let $k = W\epsilon/n$. Then, for each job i, we round its weight down to the nearest multiple of k, i.e., $w'_j \leftarrow \lfloor w_j/k \rfloor \cdot k$. After rounding, each job loses weight of at most k. Since at most n jobs can be scheduled on time, the total loss is bounded by $nk = W\epsilon$. On the other hand, the optimal solution is certainly no worse than simply scheduling the one most valuable job, so $\mathrm{OPT} \geqslant W$. Chaining them together, the trade is $\leqslant \epsilon^{-1}$ of OPT and hence the overall approximation factor is $\geqslant 1-\epsilon$. The runtime of this algorithm is $\mathcal{O}(n^2 \cdot \sum w_i/k)$ as we only have $\sum w_i/k$ discrete thresholds. But this translates

$$\mathcal{O}(n^2 \sum w_i/k) = \mathcal{O}(n^2 \cdot nW/(W\epsilon/n)) = \mathcal{O}(n^3/\epsilon)$$

which is now strongly polynomial, since ϵ is prescribed.

Solution to problem 4. (i) It might be more intuitive to use the original definitions mentioned in the lecture, where instead of keeping track of various edge weights, we allow duplicate edges (hence multigraph) but set every edge to have weight 1. Suppose the min cut has value λ and that C is an α-approximate cut, so $|C| = c = \alpha\lambda$. Because the min cut has value λ, we know that every vertex must have degree $\geqslant \lambda$ for otherwise that singleton vertex would give a cut with value $< \lambda$. Therefore, the total number of edges is at least $|E| = n\lambda/2$.

Keeping the algorithm fixed, the probability that C survives is the product that C survives each iteration,

i.e., no edge crossing C is contracted. The first iteration has success probability

$$\mathbb{P}(C \text{ survives first iteration}) = 1 - \frac{c}{|E|} \geqslant 1 - \frac{c}{n\lambda/2} = 1 - \frac{\alpha\lambda}{n\lambda/2} = 1 - \frac{2\alpha}{n}.$$

Likewise, the second iteration has success probability $\geqslant 1 - (2\alpha)/(n-1)$ and so on, so (note that compared to the $\alpha = 1$ case, we stop early here, for we cannot continue contracting when there are only 2α edges left)

$$\begin{aligned}\mathbb{P}(C \text{ survives}) &\geqslant \prod_{2\alpha \leqslant j \leqslant n} \left(1 - \frac{2\alpha}{j}\right) = \left(1 - \frac{2\alpha}{n}\right)\left(1 - \frac{2\alpha}{n-1}\right)\cdots\left(1 - \frac{2\alpha}{\lceil 2\alpha \rceil + 1}\right) \\ &= \left(\frac{n-2\alpha}{n}\right)\left(\frac{n-2\alpha-1}{n-1}\right)\cdots\left(\frac{1}{\lceil 2\alpha \rceil + 1}\right) \\ &\geqslant \frac{\Gamma(n+1-2\alpha)}{\Gamma(n+1)} \approx n^{-2\alpha}.\end{aligned}$$

(ii) Since the algorithm has a probability $\geqslant n^{-2\alpha}$ of producing a single α-cut, the total number of such α-approximate cuts must be bounded by the reciprocal $n^{2\alpha}$.