CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2023**

Homework 3 Solutions

December 13, 2023

Name: Sepehr Assadi

Problem 1. Suppose we have a bipartite graph $G = (L, R, E)$ and our goal is to find a *minimum size* set of edges F such that every vertex in $L \cup R$ is incident on at least one of the edges in F. Design a polynomial time algorithm for this problem. **(25 points)**

Solution. Consider the following linear program:

$$\begin{aligned} \min_{x \in [0,1]^E} \quad & \sum_{e \in E} x_e \\ \text{subject to} \quad & \sum_{e \ni v} x_e \geqslant 1 \quad \forall v \in V. \end{aligned}$$

This is a fractional relaxation of the original problem because *if* we take x to be in $\{0,1\}^n$ and interpret $x_e = 1 \equiv e \in F$, then each vertex has at least one edge inside F and we are minimizing the size of F.

We can solve the above problem in polynomial time using any LP solver, then round the resulting fractional solution to an integral solution of the same value. This solves the original problem.

Given $x \in [0,1]^n$, we first apply **cycle canceling** done for bipartite matching in Lecture 8 – pick any cycle, and alternatively increase and decrease the values of x over the cycle so that this cycle breaks. Since the input is a bipartite graph, all cycles are of even length and thus this operation does not change the feasibility nor the value of x. At this point, x is supported only on a forest with no cycles. We then run a *different* **forest rounding** approach as the one for bipartite matching.

For each tree T in this forest, we root the tree arbitrarily at some degree one vertex. Any edge e incident on leaf-nodes of this tree clearly has $x_e = 1$. Let f be a parent edge e and z be a parent of f (if these edges do not exist, e belongs to a star and we are already done). Update $x_z = x_z + x_f$ and $x_f = 0$. This preserves both the value of x and its feasibility, and further partition this tree into a forest and a star. By repeatedly applying this argument, we can decompose every forest into a collection of stars. A star however is fully integral and thus we are done.

Problem 2. Let $G = (V, E)$ be any graph with integer weights $w(e)$ on edges $e \in E$. For this question, we allow the edges to have negative weight. Consider the following approximation algorithm for the maximum weight matching problem, namely, finding a matching $M \subseteq E$ that maximizes $\sum_{e \in M} w(e)$.

1. If all edges in G have a non-positive weight, return $M = \emptyset$ and terminate.
2. Pick an arbitrary edge $e \in E$ with $w(e) > 0$. Create the new graph $G' := G - e$ with weights $w'(f) = w(f)$ for every edge f not incident on e and $w'(f) = w(f) - w(e)$ for every edge f incident on e (basically, we are subtracting $w(e)$ from the weights of all edges incident on e in G').
3. Run the algorithm recursively on G' to obtain a matching M'. If both endpoints of the edge e are unmatched, return $M := M' \cup \{e\}$, otherwise, return $M = M'$ as the final answer.

Prove that this algorithm outputs a (1/2)-approximation to the maximum weight matching problem.

(25 points)

Solution. We start with the following claim:

Claim 1. *Let $G = (V, E)$ be any graph with two weight functions $w_1 : E \to \mathbb{R}$ and $w_2 : E \to \mathbb{R}$ over its edges. Suppose M is* simultaneously *an α-approximate matching for both (G, w_1) and (G, w_2) for some $\alpha \in [0, 1]$. Then, M is also an α-approximate matching for $(G, w_1 + w_2)$.*

Proof. Define $w := w_1 + w_2$ and M^* be the maximum weight matching in G. We have,

$$w(M) = w_1(M) + w_2(M) \geqslant \alpha \cdot w_1(M^*) + \alpha \cdot w_2(M^*) = \alpha \cdot (w(M^*)),$$

where the inequality is because $w_1(M)$ and $w_2(M)$ are α-approximation of maximum weight matchings in (G, w_1) and (G, w_2), each of which are at least as large as $w_1(M^*)$ and $w_2(M^*)$, respectively. □

Consider any edge e in the algorithm and the weight functions $w_1 = w'$ and $w_2 = w - w'$. The algorithm recursively computes a matching M' with respect to the weight function $w_1 = w'$. Assume *inductively* (with base case being empty graph) that M' is a $(1/2)$-approximation to the maximum weight matching in $(G - e, w')$. Since e has weight zero in w', either of M' or M is also a $(1/2)$-approximation in (G, w'). On the other hand, all edges in w_2 have the same weight of $w(e)$ and they are all incident on the edge e; so, the maximum weight matching in (G, w_2) has weight $2 \cdot w(e)$. On the other hand, either M always contain at least one edge from (G, w_2) incident on e and thus it is a $(1/2)$-approximation to (G, w_2).

Thus, we have a matching M which is a $(1/2)$-approximation to both (G, w_1) and (G, w_2) and thus by the claim above, it is also a $(1/2)$-approximation to $(G, w_1 + w_2)$ which is the original weight function (G, w). This proves the induction step and the correctness of the algorithm. This concludes the proof.

Problem 3. Consider the following online problem. Unfortunately, you have lost a bet to your (most unreasonable) friend and now have to accommodate them: every day, you either have to buy lunch for your groups of friends at a cost of L dollars or on that day, you just give up and pay a total of G dollars to your friend to end your misery (at least in this bet). However, your friend can also decide at any day that they are done with this bet and no longer want to continue this and you have no idea on if or when they do this[1].

1. Design a *deterministic* strategy such that the total money you spend is at most $\left(2 - \frac{L}{G}\right)$ times the minimum amount you had to spend if you knew which day your friend decides to stop this bet.

   (12.5 points)

   Solution. Define $k := G/L$ (recall that we can assume L divides G). For the first $k - 1$ days, buy the lunch for the group for the cost of L each day and in the day k just pay the total cost of G and finish the game. Suppose your friend decides to finish the game at day $\ell \geqslant 1$:

   - If $\ell \leqslant k - 1$, then, the cost of your algorithm is $\ell \cdot L < k \cdot L = G$; thus, even if you knew your friend was going to stop at day ℓ, your cost could have not been better. So, in this case, your algorithm is 1-competitive.
   - If $\ell \geqslant k$, then, the cost of your algorithm is $(k - 1) \cdot L + G = (2 - (L/G)) \cdot G$; if you knew your friend was going to stop at day $\geqslant k$, your cost could have been G. So, in this case your algorithm is $(2 - (L/G))$-competitive, concluding the proof.

---

2. Prove that the above bound is optimal, i.e., *any* deterministic strategy that you choose cannot results in a ratio less than $\left(2 - \frac{L}{G}\right)$ in the above bound. **(12.5 points)**

   Proof. Any deterministic algorithm you use corresponds to paying cost of L for ℓ days and then paying the cost of G on the day $\ell + 1$. Your strategy is thus only to decide which ℓ to choose. Consider the adversarial strategy for your your friend to finish the game exactly the day after you have payed the cost of G. Define $k = G/L$ again.

[1] As I told you, they are particularly unreasonable ...

- If $\ell \geqslant k-1$, then, your cost will be $\ell \cdot L + G \geqslant (2 - L/G) \cdot G$ while the optimal cost is to just pay G in the first day, so at best this is $(2-(L/G))$-competitive.
- If $\ell < k-1$, then, your cost will be $\ell \cdot L + G \geqslant (2 - L/G) \cdot (\ell+1) \cdot L$, while the optimal cost is to pay $\ell + 1 \cdot L$ throughout

Thus, no algorithm can be better than $(2-(L/G))$-competitive. □

You may assume in this question that L divides G to simplify the math.

Problem 4. We reexamine the graph sketching technique of Lecture 14 in this question. Recall the setting: We have a graph $G = (V, E)$ with $V := [n]$ and there is a player for each vertex $v \in V$ who only sees the neighbors of v. The players have access to the same shared source of randomness. Simultaneously with each other, they each send a message of length polylog(n) bits to a referee (who has no input but has access to the same shared randomness), and the referee outputs a solution to the problem. We require the solution to be correct with high probability.

In the class, we designed an algorithm for finding a spanning forest of the input graph. In this question, we examine two other problems in this model.

(*a*) We say a graph $G = (V, E)$ is k-(edge)-connected if one needs to remove at least k edges from G in order to make G disconnected. In other words, the minimum cut of G has at least k edges. We like to design a graph sketching algorithm for this problem wherein each player sends a messages of length $O(k \cdot \text{polylog}(n))$ to the referee.

Consider the following the process: Pick a spanning forest F_1 of G, then spanning forest F_2 of $G \setminus F_1$, then F_3 of $G \setminus (F_1 \cup F_2)$, and so on and so forth until picking F_k. Prove that G is k-connected if and only if $F_1 \cup F_2 \cup \ldots \cup F_k$ is k-connected.

Then design an algorithm using ℓ_0-samples of Lecture 14 that allows the players to send $O(k \cdot \text{polylog}(n))$-length messages to the referee so that the referee can find the spanning forests $F_1, \ldots, F_k$.

(12.5 points)

Solution. We start with proving the claim in the problem statement.

Claim 2. *G is k-connected if and only if $F_1 \cup F_2 \cup \ldots \cup F_k$ is k-connected.*

Proof. Since $F_1, \ldots, F_k$ is a subgraph of G, if the former is k-connected, so is the latter. Now let us consider the converse and suppose towards a contradiction that G is k-connected by $F := F_1 \cup \ldots \cup F_k$ is not. Consider the cut $(S, V \setminus S)$ with less than k edges in F which should exist because F is not k-connected. Since G is k-connected, it means that there is an edge $e = (u, v)$ in G that crosses this cut but this edge is not part of F. But then this means that in each forest F_i for $i \in [k]$, we have picked another edge other than e from this cut otherwise u and v remain disconnected even though this edge exists in $G \setminus F$ and thus should be connected in all the spanning forests we picked. Hence, we have picked at least k edges from the cut, contradiction that the cut had less than k edges. □

Let A be the sketching matrix that can compute a spanning forest of the given input graph using the approach of Lecture 14. We sample $A_1, A_2, \ldots, A_k$ and compute $A_1 \cdot G$ and $A_2 \cdot G$, up to, $A_k \cdot G$ and send all of them to the referee. Recall that $A_1, \ldots, A_k$ are sampled using public randomness and the referee knows them as well. The referee can compute a spanning forest F_1 of G from $A_1 \cdot G$. Then, the referee computes $A_2 \cdot F_1$ on its own and compute $A_2 \cdot G - A_2 \cdot F_1 = A_2 \cdot (G - F_1)$ and obtain a spanning forest F_2 of $G \setminus F_1$. The referee continues this to compute all the k spanning forests.

Compared to the original spanning forest algorithm, the communication of each player increases by a factor of k in this new algorithm and the probability of success decreases by a factor of k (which is negligible given the high-probability bound of the algorithm). This concludes the proof.

(*b*) Let us now switch to finding an *approximate* MST. Suppose every edge $e = (u, v) \in E$ of the graph G also as an integer weight $w(e) \geqslant 1$ which is known only to players on vertices u and v. The players are also all given a parameter $\varepsilon > 0$. They should each send a message of size $\text{poly}(1/\varepsilon, \log(n))$ to the referee and the referee with high probability outputs a spanning T such that weight of T is at most $(1 + \varepsilon)$ times the weight of the MST of G. **(12.5 points)**

Solution. We round the weight of every edge to the nearest power of $(1 + \varepsilon/4)$. This can only change the weight of the MST by a factor of $(1 + \varepsilon)$ which is okay for our application. But now, the total number of different edge weights is $t := O(\log n/\varepsilon)$ using the fact that the maximum weight edge is also $\text{poly}(m, n)$. For partition the graph into t different graphs $G_1, G_2, \ldots, G_t$, where the graph G_i consists of all edges with weight $(1 + \varepsilon)^i$.

Let A be the sketching matrix that can compute a spanning forest of the given input graph using the approach of Lecture 14. We sample $A_1, A_2, \ldots, A_t$ and compute $A_i \cdot G_i$ for every $i \in [t]$ and send it to the referee. The referee computes a spanning forest $F_1, F_2, \ldots, F_t$ using these sketches. It then run Kruskal's algorithm over these edges by picking all edges in F_1, then picking edges from F_2 that do not create a cycle, then edges from F_3 that do not create a cycle and so on and so forth. This way, we obtain an MST of the graph $G_1 \cup G_2 \cup \ldots \cup G_t$ (the proof is identical to that of Kruskal's algorithm and we do not repeat it here). As argued earlier, this is also a $(1 + \varepsilon)$-approximate MST of the input.

Compared to the original spanning forest algorithm, the communication of each player increases by a factor of $t = O(\log n/\varepsilon)$ in this new algorithm and the probability of success decreases by a factor of t (which is negligible given the high-probability bound of the algorithm). This concludes the proof.
