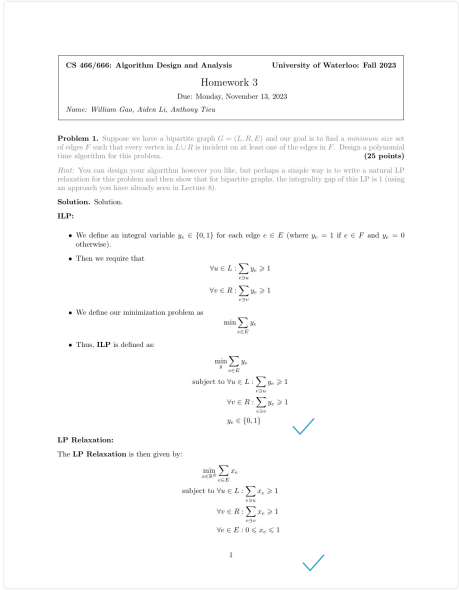
My grades for Homework 3

Q1 25 / 25

Refer to the assignment PDF.



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

Homework 3

Due: Monday, November 13, 2023

Name: William Gao, Aiden Li, Anthony Tieu

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)**

Hint: You can design your algorithm however you like, but perhaps a simple way is to write a natural LP relaxation for this problem and then show that for bipartite graphs, the integrality gap of this LP is 1 (using an approach you have already seen in Lecture 8).

Solution. Solution.

ILP:

- We define an integral variable $y_e \in \{0, 1\}$ for each edge $e \in E$ (where $y_e = 1$ if $e \in F$ and $y_e = 0$ otherwise).
- Then we require that

$$\forall u \in L : \sum_{e \ni u} y_e \geqslant 1$$

$$\forall v \in R : \sum_{e \ni v} y_e \geqslant 1$$

- We define our minimization problem as

$$\min \sum_{e \in E} y_e$$

- Thus, **ILP** is defined as

$$\min_{\mathbf{y}} \sum_{e \in E} y_e$$

$$\text{subject to } \forall u \in L : \sum_{e \ni u} y_e \geqslant 1$$

$$\forall v \in R : \sum_{e \ni v} y_e \geqslant 1$$

$$y_e \in \{0, 1\}$$

LP Relaxation:

The **LP Relaxation** is then given by:

$$\min_{x \in \mathbb{R}^E} \sum_{e \in E} x_e$$

$$\text{subject to } \forall u \in L : \sum_{e \ni u} x_e \geqslant 1$$

$$\forall v \in R : \sum_{e \ni v} x_e \geqslant 1$$

$$\forall e \in E : 0 \leq x_e \leqslant 1$$

1

The objective value of the **LP Relaxation** serves as a lower bound to the **ILP** (given that any solution to the **ILP** is also a solution to the **LP Relaxation**).

Proposition 1:

*Any feasible and optimal solution $x \in \mathbb{R}^E$ to the **LP Relaxation** for bipartite edge cover can be rounded to a feasible integral solution $y \in \{0,1\}^E$ without increasing the objective value.*

*Proof of **Proposition 1**.* Fix the feasible and optimal solution $x \in \mathbb{R}^E$. If x is already integral, we are done by taking $y = x$. Otherwise, perform the following steps.

We define the support of x as $\text{supp}(x) := \{e \in E : x_e > 0\}$.

Claim 1: Let a be any leaf-node in $\text{supp}(x)$ (if a leaf-node exists) and let i be the edge incident on a. By feasibility, $\sum_{e \ni a} x_e \geqslant 1$, as such, by optimality, $x_i = 1$.

Step 1: Cycle Canceling.

- Note: if x is the optimal solution to the **LP Relaxation**, then $\text{supp}(x)$ cannot contain an integral cycle (i.e. a cycle such that all edges are assigned a value $x_e = 1$). If such a cycle did exist, then by removing one edge from it, all vertices within the cycle would still be incident on at least one edge and we would have found a new feasible set of edges with size smaller than that of x, a contradiction. Thus, any cycle in $\text{supp}(x)$ must be fractional (at least, in part).
- We perform **Cycle Canceling** as defined in Lec8 section 2.3:
  - Find a cycle in the support of x that contains a fractional value. If no cycle can be found, then proceed to *Step 2*.
  - Let $C = e_1, e_2, e_3, ..., e_l$ be our cycle where $e_1 \in C$ is the edge with the smallest fractional value in the cycle. Let $\delta = x_{e_1}$.
  - Update x as follows:
    $$x_{e_1} \leftarrow x_{e_1} - \delta, x_{e_2} \leftarrow x_{e_2} + \delta, x_{e_3} \leftarrow x_{e_3} - \delta, ...$$
- This guarantees:
  - At least one edge is removed from $\text{supp}(x)$ (since $x_{e_1} = 0$).
  - Since G is bipartite, its cycles must be even, as such $\sum_{e \in L} x_e$ remains the same for all $b \in L \cup R$ (we are increasing the value of one edge of b by δ and decreasing the value of another edge of b by the same value). Feasibility is retained.
  - Since G is bipartite, its cycles must be even, as such $\sum_{e \in R} x_e$ remains the same. Optimality is retained.
  - **Claim 2:** Let a be any leaf-node produced by **Cycle Canceling** and let i be the edge incident on a. Then $x_i = 1$.
    *Proof of **Claim 2**.* Since feasibility is retained, then $\sum_{e \ni a} x_e \geqslant 1$, thus $x_i \geqslant 1$. Since optimality is retained, then $x_i = 1$ (otherwise, if $x_i > 1$, then we could reduce $x_i = 1$ and obtain a solution that is smaller than the optimal solution, a contradiction).
- We repeat **Cycle Canceling** until no cycles remain in the support of x.
- Note that each iteration of **Cycle Canceling** is $O(E)$ (a cycle contains a maximum of $|E|$ edges) and since we remove at least one edge per iteration, the number of iterations of **Cycle Canceling** is also $O(E)$. As such, **Cycle Canceling** is $O(E^2)$ (polynomial time).

Step 2: Rounding Forests.

- No cycles remain in the support of x, so $\text{supp}(x)$ is a forest. There must always be a leaf-node (unless the forest is empty).

- Let u be a leaf-node in $\text{supp}(x)$, let v be its parent, and let $i \in \{u, v\}$ be the edge between them.
- By **Claim 1** and **Claim 2**, $x_i = 1$ (edge i is integral).
- Suppose that v has child-nodes (other than u). Let w be an arbitrary child-node of v (where $w \neq u$) and let $j \in \{v, w\}$ be the edge between them. Since there are no cycles in the support of x, then each child of v can be handled independently:
  - If w is not a leaf-node, then we distribute the x-value of j amongst the other edges incident on w and we set $x_j = 0$. We note that this does not affect $\sum_{e \ni u} x_e$ and it can only serve to possibly increase the value of $\sum_{e \ni a} x_e$ for any neighbours a of w (with the exception of v).
  - This guarantees:
    - $\sum_{e \in L} x_e$ increases or remains the same for all $b \in L \cup R$ (with the exception of v, but we already have that $\sum_{e \ni v} x_e \geqslant 1$ given that $x_i = 1$). Feasibility is retained.
    - $\sum_{e \in E} x_e$ remains the same. Optimality is retained.
    - **Claim 3:** Let a be any leaf-node produced by this process and let i be the edge incident on a. Then $x_i = 1$.
      *Proof of **Claim 3**.* Since feasibility is retained, then $\sum_{e \ni a} x_e \geqslant 1$, thus $x_i \geqslant 1$. Since optimality is retained, then there must exist some method of distributing the x-value of j amongst the other edges incident on w such that $x_i = 1$ (otherwise, if such a method did not exist and $x_i > 1$, then we could reduce $x_i = 1$ and obtain a solution that is smaller than the optimal solution, a contradiction).
  - Repeat for all possible child-nodes w of v.
- Ignore u, v, and i in subsequent iterations of **Rounding Forests**.
- Repeat until no leaf-nodes remain.
- Note that each iteration of **Rounding Forests** is $O(V)$ (v has less than $|V|$ child-nodes) and since we remove/ignore two vertices per iteration, the number of iterations of **Rounding Forests** is also $O(V)$. As such, **Rounding Forests** is $O(V^2)$ (polynomial time).

At this point, all x-values are integral. The solution remains feasible and retains the optimality of the original **LP Relaxation** solution. Since the **LP Relaxation** provides a lower bound to the **ILP**, then x is an optimal solution to the **ILP**, as desired. ■

Q2 25 / 25

Refer to the assignment PDF.

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. Let's first consider an **Invariant 1** in the provided algorithm: for any G and G' where $G' := G - e$ with the updates on all the edges that are incident on e. Suppose the optimal maximum weight matching for G is Opt_G, and the optimal maximum weight matching for G' is $Opt_{G'}$. Let M and M' denote the maximum weight matching for G and G' returned by the algorithm respectively, then:

- If $e \in Opt_{G'}$:
  since removing an edge $e \in Opt_G$ will only reduce the weight of the edges incident on e, this won't reduce the weight of the other edges in Opt_G, so Opt_G is a matching. Hence, $w'(Opt_{G'}) \geq w(Opt_G) - w(e)$.
- If $e \notin Opt_{G'}$:
  Since there can be at most 2 edges from Opt_G that are incident on e, $w'(Opt_{G'}) \geq w(Opt_G) - 2w(e)$.

So, we have **Invariant 1: For any G' retrieved from G by step 2,** $w'(Opt_{G'}) \geq w(Opt_G) - 2w(e)$

Now, let's prove that this algorithm outputs a $(1/2)$-approximation to the maximum weight matching problem after backtracking by an induction.

Base case: If all edges in G have a non-positive weight, M will be empty. So, $w(M) = 0$. It's clear that $w(Opt_G) = 0$, so $w(M) \geq \frac{1}{2}w(Opt_G)$. □

Then, we can define our **inductive hypothesis**: Assume that for any intermediate G_k, $w_k(M_k) \geq \frac{1}{2}w_k(Opt_{G_k})$. The remained is to prove that for the original graph G, $w(M) \geq \frac{1}{2}w(Opt_G)$.

By step 3. we know that $M := M' \cup e$ or $M := M'$ depending on whether both endpoints of the edge e are unmatched in M':

(1)If $M := M' \cup e$, both endpoints of e are not in M'. So, we don't need to adjust the weight of the f in M of G since none of them can be incident on e. And the only adjustment required is to add $w(e)$ to $w'(M')$:

$$
\begin{aligned}
w(M) &= w'(M') + w(e) \\
&\geq \frac{1}{2}w'(Opt_{G'}) + w(e) && \text{(By IH)} \\
&\geq \frac{1}{2}[w(Opt_G) - 2w(e)] + w(e) && \text{(By Invariant 1)} \\
&\geq \frac{1}{2}w(Opt_G) - w(e) + w(e) \\
&= \frac{1}{2}w(Opt_G)
\end{aligned}
$$

(2)If $M := M'$, then at least one of the endpoints of e is matched in M'. So, when we backtrack from G' to G, we'll need to update the weight of those $f \in M'$ that are incident on e by adding back the reduced portion of the weight $w(e)$ in Step2. It's clear that the $\frac{w}{2}$ of f whose weight needs to be incremented by $w(e)$ is at least 1 and at most 2. Therefore,

$$w'(M') + 2w(e) \geq w(M) \geq w'(M') + w(e)$$

So,

$$
\begin{aligned}
w(M) &\geq w'(M') + w(e) \\
&\geq \frac{1}{2}w'(Opt_{G'}) + w(e) && \text{(By IH)} \\
&\geq \frac{1}{2}[w(Opt_G - 2w(e)) + w(e) && \text{(By Invariant 1)} \\
&\geq \frac{1}{2}w(Opt_G) - w(e) + w(e) \\
&= \frac{1}{2}w(Opt_G)
\end{aligned}
$$

Hence, by induction, we have the induction conclusion that this algorithm outputs a $(1/2)$-approximation to the maximum weight matching problem. ■

Solution. We first prove the following claim:

Claim 1. *Consider a recursive step running on graph G' with weights w' as defined in the algorithm. The matching M' returned by the algorithm in this step is a (1/2)-approximation to the maximum weight matching problem on G' with weights w'.*

It follows from the claim that the algorithm outputs a (1/2)-approximation by applying the claim on the original graph G with weights w.

Proof. Base case: All edges have non-positive weight. The maximum matching has weight 0 and the algorithm returns $\emptyset$ which also has weight 0.

Inductive step: Suppose our algorithm is running on G' with weights w'. Let e be an edge with positive weight ($w'(e) > 0$). Let $\overline{G} = G' - e$ be the new graph generated by step 2 of the algorithm with weights $\overline{w}(f) = w'(f)$ for every edge f not incident on e and $\overline{w}(f) = w'(f) - w'(e)$ for every edge f incident on e.

By the inductive hypothesis, the algorithm will return a matching $\overline{M}$ such that $\overline{M}$ is a (1/2)-approximation of a maximum matching on $\overline{G}$ with weights $\overline{w}$.

Let M' be the matching constructed in step 3 of the algorithm where $M' := \overline{M} \cup \{e\}$ if both endpoints u, v of e are unmatched, otherwise $M' := \overline{M}$. We wish to prove that M' is a (1/2)-approximation on G', w'.

Since $\overline{M} \subseteq M'$, M' is a (1/2)-approximation for $\overline{G}$, $\overline{w}$. Now, examine the weight function

$$\hat{w} = \begin{cases} w'(e) & \text{if } f \text{ is incident on } e \\ 0 & \text{otherwise} \end{cases}$$

Then, since the algorithm in step 2, subtracts $w'(e)$ from $w'(f)$ if f is incident on e to get $\overline{w}$,

$$w' = \overline{w} + \hat{w} \tag{1}$$

Now consider a maximum matching on $\hat{w}$. Since there are only two vertices u, v that are incident to e and any edge that is not incident to e has weight 0 under $\hat{w}$, a max matching on $\hat{w}$ has weight at most $2w'(e)$. Furthermore, M' must have at least one edge incident to u or v, otherwise we would have added e to M'. So M' has weight at least $1 \cdot w'(e)$ under $\hat{w}$ and thus M' is a (1/2)-approximation under $\hat{w}$.

Now, to conclude the proof, we let M be a max matching under w' on G'. We show that $w'(M') \geq \frac{1}{2}w'(M)$:

$$\begin{aligned} w'(M') &= \overline{w}(M') + \hat{w}(M') && \text{(From (1))} \\ &\geq \frac{1}{2}\overline{w}(M) + \frac{1}{2}\hat{w}(M) && \text{(Since } M' \text{ is a (1/2)-approximation for both } \overline{w} \text{ and } \hat{w}) \\ &= \frac{1}{2}w'(M) \end{aligned}$$

as desired. □

Q3

21.5 / 25

Refer to the assignment PDF.

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.

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

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

(12.5 points)

Solution. Solution.

Deterministic Strategy:

(a) Pay L dollars for $\frac{G-L}{L}$ days.
(b) Then give up and pay G.

Proof of ***Deterministic Strategy.*** Let d denote the number of days our unreasonable friend persists with the bet.

Claim 1: The minimum amount we could possibly spend is either $L \cdot d$ or G. By the rules of the bet, we must either pay L dollars for d days; pay L dollars for $a \neq \{0, d\}$ days, then pay G dollars; or pay G dollars on the first day. Note that paying L dollars for $a \neq \{0, d\}$ days, then paying G dollars is never a minimum amount since we can simply pay G dollars on the first day.

We wish to show that our strategy works for all values of d:

- **Case 1:** $d \leqslant \frac{G-L}{L}$
  Total amount we pay: $L \cdot d$ dollars.
  We wish to show that paying $L \cdot d$ dollars total is optimal when $d \leqslant \frac{G-L}{L}$ (i.e., by **Claim 1**, we want to show that $L \cdot d \leqslant G$):

$$\begin{aligned} L \cdot d &\leqslant L \cdot \frac{G-L}{L} \\ &= G - L \\ &< G \end{aligned}$$

  Since it is already optimal, it is certainly within $(2 - \frac{L}{G})$ times the optimal value.

- **Case 2:** $d > \frac{G-L}{L}$ (or $d \geqslant \frac{G-L}{L} + 1$)
  Total amount we pay: $\frac{G-L}{L} \cdot L + G = G - L + G = 2G - L$ dollars.
  Observe that G becomes the optimal value when $L \cdot d \geqslant G$ (or when $d \geqslant \frac{G}{L}$). When $d \geqslant \frac{G-L}{L} + 1$, we have:

$$\begin{aligned} d &\geqslant \frac{G-L}{L} + 1 \\ d &\geqslant \frac{G-L+L}{L} \\ d &\geqslant \frac{G}{L} \end{aligned}$$

  That is, when $d \geqslant \frac{G-L}{L} + 1$, G is optimal. As such, we want to pay at most $(2 - \frac{L}{G})G = 2G - L$ dollars in total. This is precisely the amount we pay.

Thus, we spend at most $(2 - \frac{L}{G})$ times the minimum amount we would have had to spend if we knew which day the bet would end, as desired. ■

7

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

(12.5 points)

Solution. Let $\frac{L}{G} = \frac{1}{n}$. Consider any deterministic strategy:

- Say the deterministic strategy gives up and pays G immediately. Then imagine the friend stops the bet on day 1 in which case the optimal strategy was to just pay L on the first day. So the amount of money we spend in this case is $\frac{G}{L} = 2$ times more than the optimal amount.
- Say the deterministic strategy pays L until day $d \geq 1$ and then gives up and pays G on day $d+1$. Now suppose the friend never stops the bet in which case the optimal strategy was to just immediately pay G. So the amount we pay is $d \cdot L + G \geq L + G = \frac{1}{2}G$.

  So the amount we pay in this case is at least $\frac{\frac{1}{2}G}{G} = \frac{1}{2} = (2 - \frac{L}{G})$ times the optimal amount.

So in either case we cannot achieve a ratio better than $(2 - \frac{L}{G})$. ■

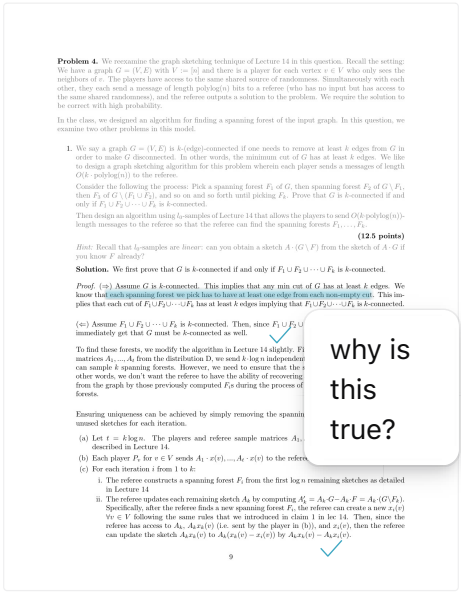
need to generalize this argument to all values of L and G.

8

Q4

17 / 25

Refer to the assignment PDF.



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.

1. 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 $F_1 \cup F_2 \cup \cdots \cup F_k$ 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)

   Hint: Recall that l_0-samples are *linear*: can you obtain a sketch $A \cdot (G \setminus F)$ from the sketch of $A \cdot G$ if you know F already?

   Solution. We first prove that G is k-connected if and only if $F_1 \cup F_2 \cup \cdots \cup F_k$ is k-connected.

   Proof. ($\Rightarrow$) Assume G is k-connected. This implies that any min cut of G has at least k edges. We know that each spanning forest we pick has to have at least one edge from each non-empty cut. This implies that each cut of $F_1 \cup F_2 \cup \cdots \cup F_k$ has at least k edges implying that $F_1 \cup F_2 \cup \cdots \cup F_k$ is k-connected.

   ($\Leftarrow$) Assume $F_1 \cup F_2 \cup \cdots \cup F_k$ is k-connected. Then, since $F_1 \cup F_2 \cup$ immediately get that G must be k-connected as well.

   To find these forests, we modify the algorithm in Lecture 14 slightly. Fi matrices $A_1, \ldots, A_t$ from the distribution D, we send $k \cdot \log n$ independen can sample k spanning forests. However, we need to ensure that the other words, we don't want the referee to have the ability of recovering from the graph by those previously computed F_is during the process of forests.

   Ensuring uniqueness can be achieved by simply removing the spanni unused sketches for each iteration.

   (a) Let $t = k \log n$. The players and referee sample matrices A_1, described in Lecture 14.
   (b) Each player P_v for $v \in V$ sends $A_1 \cdot x(v), \ldots, A_t \cdot x(v)$ to the refere
   (c) For each iteration i from 1 to k:
      i. The referee constructs a spanning forest F_i from the first $\log n$ remaining sketches as detailed in Lecture 14
      ii. The referee updates each remaining sketch A_h by computing $A'_h = A_h \cdot G - A_h \cdot F = A_h \cdot (G \setminus F_h)$. Specifically, after the referee finds a new spanning forest F_i, the referee can create a new $x_i(v)$ $\forall v \in V$ following the same rules that we introduced in claim 1 in lec 14. Then, since the referee has access to A_h, $A_h x_h(v)$ (i.e. sent by the player in (b)), and $x_i(v)$, then the referee can update the sketch $A_h x_h(v)$ to $A_h(x_h(v) - x_i(v))$ by $A_h x_h(v) - A_h x_i(v)$.

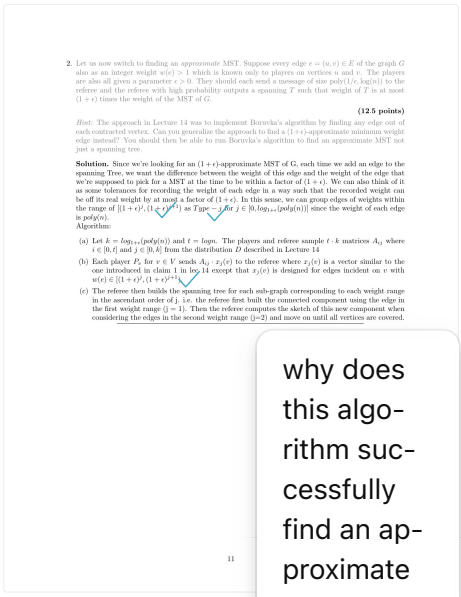
   9

why is this true?

      iii. Throw out the $\log n$ sketches used in this round.
   (d) At the end, output $F_1, \ldots F_k$

   Finally, in Lecture 14, we bounded the probability of failure of finding one spanning tree by $\frac{1}{\text{poly}(n)}$. Since we find $k \leq m \leq n^2$, union bounding over k still yields a $\frac{1}{\text{poly}(n)}$ chance of failure, as desired. Also, it's clear that now each player sends a message of size of $k \cdot \text{polylog}(n)$.

   10



2. 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 $w(e) > 1$ which is known only to players on vertices u and v. The players are also all given a parameter $\epsilon > 0$. They should each send a message of size $\text{poly}(1/\epsilon, \log(n))$ to the referee and the referee with high probability outputs a spanning T such that weight of T is at most $(1 + \epsilon)$ times the weight of the MST of G.

(12.5 points)

Hint: The approach in Lecture 14 was to implement Boruvka's algorithm by finding any edge out of each contracted vertex. Can you generalize the approach to find a $(1+\epsilon)$-approximate minimum weight edge instead? You should then be able to run Boruvka's algorithm to find an approximate MST not just a spanning tree.

Solution. Since we're looking for an $(1+\epsilon)$-approximate MST of G, each time we add an edge to the spanning Tree, we want the difference between the weight of this edge and the weight of the edge that we're supposed to pick for a MST at the time to be within a factor of $(1+\epsilon)$. We can also think of it as some tolerances for recording the weight of each edge in a way such that the recorded weight can be off its real weight by at most a factor of $(1+\epsilon)$. In this sense, we can group edges of weights within the range of $[(1+\epsilon)^j, (1+\epsilon)^{j+1})$ as $Type - j$ for $j \in [0, \log_{1+\epsilon}(\text{poly}(n))]$ since the weight of each edge is $\text{poly}(n)$.
Algorithm:

(a) Let $k = \log_{1+\epsilon}(\text{poly}(n))$ and $t = \log n$. The players and referee sample $t \cdot k$ matrices A_{ij} where $i \in [0, t]$ and $j \in [0, k]$ from the distribution D described in Lecture 14

(b) Each player P_v for $v \in V$ sends $A_{ij} \cdot x_j(e)$ to the referee where $x_j(e)$ is a vector similar to the one introduced in claim 1 in leq 14 except that $x_j(e)$ is designed for edges incident on v with $w(e) \in [(1+\epsilon)^j, (1+\epsilon)^{j+1}]$.

(c) The referee then builds the spanning tree for each sub-graph corresponding to each weight range in the ascendant order of j. i.e. the referee first built the connected component using the edge in the first weight range ($j = 1$). Then the referee computes the sketch of this new component when considering the edges in the second weight range (j=2) and move on until all vertices are covered.

11

why does this algorithm successfully find an approximate MST? what is the number of bits?

Q5 0 / 15

Refer to the assignment PDF.

This question wasn't answered