COS 521: Homework 2

Due on October 11, 2022

Professor Matt Weinberg

Nameless Author :)
Collaborators: I can't tell you :)

Problem 1

Solution

Proof. Recall from lecture that a linear optimization oracle over a convex region $\mathcal{K}$ is poly-time reducible to a separation oracle for $\mathcal{K}$ (we showed this by writing an LP over $\mathcal{K}$ to find the "most violated" hyperplane, if one exists). So, we can claim that given $\mathcal{A}_P$ and $\mathcal{A}_Q$, we are given poly-time separation oracles for P and Q. From this, we simply need to devise a poly-time separation oracle for $P \cap Q$, since we can then apply the ellipsoid algorithm to optimize linear functions over $P \cap Q$ in poly-time. Recall that a separation oracle for $P \cap Q$ takes as input a vector $\vec{x}$ and outputs true if $\vec{x} \in P \cap Q$ and, if not, outputs a separating hyperplane $\vec{w} \in \mathbb{R}^n$ s.t. $\vec{x} \cdot \vec{w} > max_{\vec{y} \in P \cap Q} \{\vec{y} \cdot \vec{w}\}$. We can construct one using the oracles for P and Q that we make from $\mathcal{A}_P$ and $\mathcal{A}_Q$. For an input $\vec{x}$, query the separation oracles for P and Q. We split the algorithm for our oracle for$P \cap Q$ into three cases:

- **Both return contained.** If both return that $\vec{x}$ is contained in the regions P and Q, then we can immediately return that $\vec{x} \in P \cap Q$.

- **One returns contained and one returns a separating hyperplane.** If we see that $\vec{x}$ is contained in one of the regions, say P, but we get a separating hyperplane for Q, then we know that $\vec{x} \notin P \cap Q$ and that we should return that separating hyperplane for Q. To see this, note that a separating hyperplane for Q takes the form of a
$$\vec{w}_Q \in \mathbb{R}^n \text{ s.t. } \vec{x} \cdot \vec{w}_q > max_{\vec{y} \in Q} \{\vec{y} \cdot \vec{w}\}$$
Since $P \cap Q \subset Q$, we see that
$$\vec{x} \cdot \vec{w}_q > max_{\vec{y} \in Q} \{\vec{y} \cdot \vec{w}\} \geq max_{\vec{y} \in P \cap Q} \{\vec{y} \cdot \vec{w}\}$$
So, $\vec{w}_Q$ is a separating hyperplane for $P \cap Q$ as well, which means that returning $\vec{w}_Q$ suffices for the operation of a separation oracle for $P \cap Q$. Symmetric reasoning holds if the query to the P and Q oracles yields a separating hyperplane $\vec{w}_P$ for P and that $\vec{x} \in Q$; in this case we would return $\vec{w}_P$

- **Both return separating hyperplanes.** If we see that $\vec{x}$ is not contained in either P or Q and instead receive two separating hyperplanes $\vec{w}_P, \vec{w}_Q$, we can return either one. This is because, as before, a separating hyperplane for Q yields a seperating hyperplane for $P \cap Q$ via
$$\vec{w}_Q \in \mathbb{R}^n \text{ s.t. } \vec{x} \cdot \vec{w}_q > max_{\vec{y} \in Q} \{\vec{y} \cdot \vec{w}\} \implies \vec{x} \cdot \vec{w}_q > max_{\vec{y} \in Q} \{\vec{y} \cdot \vec{w}\} \geq max_{\vec{y} \in P \cap Q} \{\vec{y} \cdot \vec{w}\},$$
and the same logic applies for $\vec{w}_P$. So, we can return either $\vec{w}_P$ or $\vec{w}_Q$ as a separating hyperplane for $P \cap Q$.

So, we have crafted a working poly-time separation oracle for $P \cap Q$ using the given optimization algorithms $\mathcal{A}_P$ and $\mathcal{A}_Q$. The only thing we need to do in order to apply the ellipsoid algorithm to optimize over $P \cap Q$ in poly-time is to construct a bounding box of $P \cap Q$; we can make a poor, but good enough bounding box by simply finding the maximum and minimum coordinates of P in each of the n dimensions. Each of these n different max and min functions can be framed as optimizing a linear function (namely, for each coordinate $i \in [n]$, we optimize the LP that extracts the max or min of the linear objective $\sum_{j=1}^{n} \mathbb{1}_{j=i} x_j$) over P. So, we can find a bounding box, and therefore a bounding ellipsoid, over P using $\mathcal{A}_P$. Since $P \cap Q \subset P$, this also yields a bounding ellipsoid for $P \cap Q$. Armed with a poly-time separation oracle and a bounding ellipsoid for $P \cap Q$, we can apply the ellipsoid algorithm to optimize linear functions over $P \cap Q$ in poly-time. ■

Problem 2

Solution

Proof. We will first formulate an integer program that exactly reproduces the MAX-3CUT problem. We will then devise an LP rounding scheme and then prove that, in expectation, the algorithm achieves a ???? approximation.

Integer Program. Consider the following integer program over n vectors $\vec{u}_1, ..., \vec{u}_n \in \mathbb{R}^n$, where n is the number of vertices in the graph. (Note that e_i is the standard basis vector of $\mathbb{R}^n$ with a 1 in the i^{th} coordinate and 0's elsewhere).

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{2}{3}(1 - \langle \vec{u}_i, \vec{u}_j \rangle) \\ \text{subject to} \quad & ||\vec{u}_i|| = 1 \quad \forall i \in [n] \\ & \vec{u}_i \in \left\{ e_1, \frac{\sqrt{3}}{2}e_2 - \frac{1}{2}e_1, \frac{-\sqrt{3}}{2}e_2 - \frac{1}{2}e_1 \right\} \quad \forall i \in [n] \end{aligned}$$

This is clearly an exact formulation of MAX-3CUT, since each vertex is assigned to exactly one set, corresponding to one of the three elements in the second constraint. Also, for each element of the sum in the objective (for each edge in the graph), the objective increases by 1 if $\langle \vec{u}_i, \vec{u}_j \rangle = -\frac{1}{2}$ (pairs of the three elements in the second constraint dot to $-\frac{1}{2}$), and doesn't increase if $\langle \vec{u}_i, \vec{u}_j \rangle = 1$. If we think of this objective as a counter, we see that the counter increases by 1 exactly when an edge is between two vertices assigned to different sets, and 0 otherwise. So, the objective function faithfully counts the number of edges crossing the 3-cut, and is therefore the value of the 3-cut, which we are maximizing. We find that this integer program is a formulation of the MAX-3CUT problem.

Rounding. To begin, we remove the integer constraint to get a program with constraints and objectives that are linear functions of inner products between our vectors; in other words, the poly-time solvable semi definite program:

$$\begin{aligned} \text{maximize} \quad & \sum_{(i,j)\in E} \frac{2}{3}(1 - \langle \vec{u}_i, \vec{u}_j \rangle) \\ \text{subject to} \quad & ||\vec{u}_i|| = 1 \quad \forall i \in [n] \end{aligned}$$

The solution to this SDP will yield unit vectors $\hat{u}_1, ..., \hat{u}_n$ that maximize the objective, say to some value OPT. Consider the following rounding scheme, inspired by the random hyperplane MAX-CUT rounding scheme from lecture:

- Choose two random vectors $\vec{c_1}, \vec{c_2} \sim \mathcal{N}(0,1)^n$ (component-wise drawn from $\mathcal{N}(0,1)$) and fix them for the rest of the algorithm. Observe that these two vectors have the property that they, when projected in any 2D plane of $\mathbb{R}^n$, will have angles above the horizontal that are uniformly distributed from 0 to 2π.
- For each vertex i, compute $sign\left(\langle \vec{c_1}, \hat{u}_i \rangle\right)$. If this sign is positive, assign vertex i to set X. Otherwise, continue to the next step.
- For each unassigned vertex i, compute $sign\left(\langle \vec{c_2}, \hat{u}_i \rangle\right)$. If this sign is positive, assign vertex i to set Y. Otherwise, assign vertex i to set Z.

This concludes the rounding scheme.

Problem 2 continued on next page...

Analysis. In order to perform the analysis, we want to compute the probability that vertices i and j get assigned to different sets, as this will help us find the expected value of our rounded solution. Consider any arbitrary edge $(i, j) \in E$. We can compute the probability that i and j get assigned to different sets in the following way:

- We know that i and j get assigned to X if and only if the projection of $\vec{c}_1$ onto the plane spanned by $\hat{u}_i$ and $\hat{u}_j$ lies between $\hat{u}_i$ and $\hat{u}_j$. Since the angle of this projected decision boundary above the horizontal is distributed from $\mathcal{U}[0, 2\pi]$, we see that the probability it lies between $\hat{u}_i$ and $\hat{u}_j$ is precisely
$$\frac{\theta_{ij}}{\pi},$$
where θ_{ij} is the angle between these two vectors in the plane. This is the probability that one gets assigned to X and the other doesn't.
- The probability that both i and j do not get assigned to X is also $\frac{\pi-\theta_{ij}}{2\pi}$. In this case, whether i and j end up in different sets depends only on whether $\vec{c}_2$ lands between the two, which once again happens with probability $\frac{\theta_{ij}}{\pi}$. Since the drawing of $\vec{c}_1$ and $\vec{c}_2$ are independent, we can multiply these probabilities. So, the probability that one gets assigned to Y and the other to Z is
$$\frac{\pi - \theta_{ij}}{2\pi} \cdot \frac{\theta_{ij}}{\pi} = \frac{\pi\theta_{ij} - \theta_{ij}^2}{2\pi^2}$$
- The above two cases totally account for all the ways that i and j could be assigned to different sets. So, we sum the probabilities to see that the total probability of i and j being assigned to different sets is
$$\frac{\theta_{ij}}{\pi} + \frac{\pi\theta_{ij} - \theta_{ij}^2}{2\pi^2} = \frac{3\pi\theta_{ij} - \theta_{ij}^2}{2\pi^2}$$

Since this holds for every edge in the graph, we can compute the expected value of the rounded 3-cut to see that it equals
$$\mathbb{E}\left[\text{rounded value}\right] = \sum_{(i,j)\in E} \frac{3\pi\theta_{ij} - \theta_{ij}^2}{2\pi^2}$$

We can compare this to each element of the sum that OPT (note that $\langle \hat{u}_i, \hat{u}_j \rangle = cos(\theta_{ij})$ since they are unit vectors) takes to see that we have a ratio of
$$\frac{\frac{3\pi\theta_{ij} - \theta_{ij}^2}{2\pi^2}}{\frac{2}{3}(1 - cos(\theta_{ij}))}$$
where the θ_{ij}'s are the same since the rounded solution uses the same vectors as the optimal solution to the SDP. We can see that this ratio is always greater than 0.725 analytically (Desmos). So, our rounded solution obtains in expectation at least 0.725 of the value of the fractional solution OPT. Since OPT is clearly better than the optimal solution to the integer program for MAX-3CUT (since it is relaxed), this means that our rounded algorithm achieves a 0.725-approximation for MAX-3CUT. ■

Problem 3

Part A

Firstly, note that by ED post **#59**, if we know that a feasible solution exists, then we know that a basic feasible solution exists. We will show that a feasible solution exists in every step by induction on step n. Note that at step $n = 0$, we clearly have a feasible LP (since none of the x_t's are fixed to anything yet, we can set them all to be 0 trivially).

Now, suppose by way of induction that we have a feasible LP at step n for some arbitrary $n \in \mathbb{N}$. Let $\vec{x}$ be a basic feasible solution of the LP at step n. We know that for all t, one of the following cases holds:

1. If x_t was already fixed prior to step n, it is still fixed to the same ϵ_t.
2. If x_t was not already fixed prior to step n and $x_t \in \{-1, 1\}$ in the basic feasible solution, we will fix x_t to some ϵ_t that is its value.
3. Otherwise, we do not fix x_t at all.

So, we find that when we create the LP in step $n + 1$, x_t's that were in case 1 will stay fixed to the same value as they had in $\vec{x}$. Also, x_t's that were in case 2 will become fixed to the same value as they had in $\vec{x}$. Lastly, x_t's that were in case 3 will be free to take any values; in particular, they can take the same values as they had in $\vec{x}$. This yields that we could use the same vector $\vec{x}$ as a feasible solution in step $n+1$; therefore, the LP is feasible in step $n + 1$. By induction, we can say that the claim is therefore true for all n. So, the LP is feasible at every step.

Part B

Suppose that at a particular step in the process, there are k colored columns left in the LP (i.e. there are k constraints of the form $x_t = \epsilon_t$). (Note that by part **(a)**, there exists a basic feasible solution for this step). Then, there are $n - k$ unfixed variables with constraints $x_t \in [-1, 1]$. So, there can only be $s(n - k)$ uncolored ones in the worst case (s per uncolored column), which means that there can be at most $\frac{s(n-k)}{2s} = \frac{n-k}{2}$ rows with at least $2s$ uncolored ones. This means that there can be at most $\frac{n-k}{2}$ constraints of the form $\sum_t A_{it}x_t = 0$ (i.e. the number of row constraints is at most half the number of uncolored columns). There are also exactly k constraints of the form $x_t = \epsilon_t$ since there are k colored columns. So, since a basic feasible solution satisfies n constraints with equality we find that there are at least

$$n - \left(\frac{n-k}{2} + k\right) = \frac{n+k}{2} - k = \frac{n-k}{2}$$

constraints of the form $x_t \in [-1, 1]$ that a basic feasible solution will satisfy with equality. Satisfying this constraint with equality corresponds to $x_t \in \{-1, 1\}$, which will result in coloring column t. So, we arrive at the result that at least $\frac{n-k}{2}$ columns will be colored in this step. Therefore, at least half of the uncolored columns will be colored every step.

Part C

We see from the results in parts **(a-b)** that the iteration will terminate when we have at most $2s$ uncolored columns. Also, when it terminates, it will have ended on a basic feasible solution. Suppose that upon termination of the iteration, there were no rows containing more than $2s$ colored ones. Then, we can simply color the remaining $2s$ columns however we would like and achieve a worst case difference between the number of 1's and the number of -1's in each row of $O(s)$. Now, suppose that upon termination there were some rows containing more than $2s$ colored ones. Clearly, none of these rows contain more than $2s$ uncolored ones,

Problem 3 continued on next page. . .

since the termination ended with at most $2s$ uncolored columns. So, we can imagine the step in the iteration at which there were more than $2s$ uncolored ones in each of these rows (this will be $O(s)$ steps before the termination). For each of these rows, at this step the row constraint guaranteed that the sum amounted to 0. The number of ones that were added to that row since then must be $O(s)$, and so the difference between the number of 1's and number of -1's is still bounded by $O(s)$.

Problem 4

Part A

Proof. We wish to show that no algorithm with $m = o(n)$ samples can always predict the median within a factor of 1.1. To show this, we will construct two sequences of length n from which sets of m samples will look the same with high probability. Then, if n is odd, consider the two sequences of odd length

$$X^{(-)} = \{0, ..., 0, 0, 1, ..., 1\} \qquad X^{(+)} = \{0, ..., 0, 1, 1, ..., 1\}$$

where $X^{(-)}$ has $\frac{n+1}{2}$ 0's and $\frac{n-1}{2}$ 1's, and $X^{(+)}$ has $\frac{n-1}{2}$ 0's and $\frac{n+1}{2}$ 1's. If n is even, consider the two sequences of even length

$$X^{(-)} = \{0, ..., 0, 0, 1, ..., 1\} \qquad X^{(+)} = \{0, ..., 0, 1, 1, ..., 1\}$$

where $X^{(-)}$ has $\frac{n+2}{2}$ 0's and $\frac{n-2}{2}$ 1's, and $X^{(+)}$ has $\frac{n-2}{2}$ 0's and $\frac{n+2}{2}$ 1's. Clearly, $median(X^{(-)}) = 0$ and $median(X^{(+)}) = 1$. Let us draw $m = o(n)$ samples, where for each draw in the sample we select an index from $\{1, ..., n\}$ and grab the i^{th} elements from $X^{(-)}$ and $X^{(+)}$. Note that each individual draw has a probability of $\begin{cases} \frac{1}{n} & n \text{ odd} \\ \frac{2}{n} & n \text{ even} \end{cases}$ of yielding two different elements; otherwise, the draw looks identical between $X^{(-)}$ and $X^{(+)}$. We can apply the union bound to see that the probability that **any** of the m samples look different is at most $\begin{cases} \frac{m}{n} & n \text{ odd} \\ \frac{2m}{n} & n \text{ even} \end{cases}$. Since $m = o(n)$, we can take $m < \frac{1}{3}n$ for large enough n to see that the probability that two samples of size m look different is, in the worst case, at most $\frac{2m}{n} < \frac{2}{3}$. So, the probability that two samples from these sequences with different medians look identical is at least $\frac{1}{3}$. Any algorithm for finding the median, when met with these two identical samples, will incorrectly (worse than factor of 1.1) predict the median of one of them (this is because the "correctness" intervals that the factor of 1.1 yields don't overlap, so any prediction will be outside of one of the intervals). So, we see that for the two sequences defined above, any algorithm will be off by a factor of more than 1.1 with probability of at least $\frac{1}{3}$. ■

Part B

Proof. Firstly, let

$$X^{(+)} = \left\{ x_i \quad | \quad rank(x_i) \geq \frac{n}{2} + t \right\}$$

and

$$X^{(-)} = \left\{ x_i \quad | \quad rank(x_i) \leq \frac{n}{2} - t \right\}$$

be the upper and lower tails of the sequence X, defined by parameter t. We want to show that if we take enough samples then, with high probability, less than half of our samples land in $X^{(+)}$ **and** less than half of our samples land in $X^{(-)}$. This will ensure that the median of our samples lands within the desired range.

Suppose that we take m samples from a sequence X. For each element s_i of the sample, we see that because it is drawn uniformly from X, the probability it lands in $X^{(+)}$ is

$$\mathbb{P}\left[s_i \in X^{(+)} \right] = \frac{\frac{n}{2} - t}{n} = \frac{n - 2t}{2n}$$

Similarly,

$$\mathbb{P}\left[s_i \in X^{(-)} \right] = \frac{\frac{n}{2} - t}{n} = \frac{n - 2t}{2n}$$

Problem 4 continued on next page...

Let $N^{(+)}$ and $N^{(-)}$ be random variables for the number of elements of our sample that are in $X^{(+)}$ and $X^{(-)}$, respectively; we want both of these to be less than $\frac{m}{2}$ with high probability. Since all s_i's in our sample are drawn independently, we can compute

$$\mathbb{E}\left[N^{(+)}\right] = \mathbb{E}\left[N^{(-)}\right] = \frac{m(n-2t)}{2n}$$

We can apply the Wikipedia Chernoff bound to bound the probabilities that these variables are larger than $\frac{m}{2}$ with:

$$\begin{aligned}\mathbb{P}\left[N^{(+)} \geq \frac{m}{2}\right] &= \mathbb{P}\left[N^{(+)} \geq \left(\frac{n}{n-2t}\right)\mathbb{E}\left[N^{(+)}\right]\right] = \mathbb{P}\left[N^{(+)} \geq \left(1+\frac{2t}{n-2t}\right)\mathbb{E}\left[N^{(+)}\right]\right] \\ &\leq e^{-\frac{\left(\frac{2t}{n-2t}\right)^2 \frac{m(n-2t)}{2n}}{2+\frac{2t}{n-2t}}} = e^{-\frac{\frac{4t^2m}{2n}}{2(n-2t)+2t}} = e^{-\frac{t^2m}{n^2-nt}} \leq e^{-\frac{t^2m}{n^2}}\end{aligned}$$

Let $\delta > 0$ be arbitrary. If we take $m = \left(\frac{n}{t}\right)^2 log(\frac{2}{\delta})$ samples, we see that the bound becomes precisely $\mathbb{P}\left[N^{(+)} \geq \frac{m}{2}\right] \leq \frac{\delta}{2}$. The exact same bound happens to show that $\mathbb{P}\left[N^{(-)} \geq \frac{m}{2}\right] \leq \frac{\delta}{2}$, as they have the same expectation. So, we can use a union bound to show that the probability that either of these two events happens is less than the sum of their probabilities. In other words, the probability that less than half of our samples land in $X^{(+)}$ **and** less than half of our samples land in $X^{(-)}$ is at least $1-\delta$. Therefore, the probability that the median of our sample is in the desired range is at least $1-\delta$. ■

Problem 5

Part A

Proof. Consider the event that grabbing all n coupons takes longer than $2n \cdot log\ (n)$ steps; call this event X. We wish to show that the event happens with probability at most $\frac{1}{n}$, i.e. that $\mathbb{P}[X] \le \frac{1}{n}$.

For each coupon i, define X_i to be the event that drawing coupon i takes longer than $2n \cdot log\ (n)$ steps. Since each coupon has a $1 - \frac{1}{n}$ chance of not being drawn during each step, and the steps happen independently, we see that the probability of a coupon not being drawn after precisely $2n \cdot log\ (n)$ steps is $\mathbb{P}[X_i] = \left(1 - \frac{1}{n}\right)^{2n \cdot log\ (n)}$ for all coupons i. Lastly, note that the event X is the union of all the events X_i, since event X happens if and only if *at least one* coupon takes longer than $2n \cdot log\ (n)$ steps to be drawn. We can now apply union bound to get

$$\mathbb{P}[X] = \mathbb{P}\left[\bigcup_{i=1}^{n} X_i\right] \le \sum_{i=1}^{n} \mathbb{P}[X_i] = n \cdot \left(1 - \frac{1}{n}\right)^{n \cdot 2log\ (n)} \le n \cdot \left(\frac{1}{e}\right)^{2log\ (n)} = n \cdot \frac{1}{n^2} = \frac{1}{n}$$

Therefore, the probability that grabbing all coupons happens within $2n \cdot log(n) = O(n \cdot log(n))$ steps is at least

$$1 - \mathbb{P}[X] \ge 1 - \frac{1}{n}$$

■

Part B

Lemma 1. *If there are ϵn jobs currently unmatched jobs ($0 < \epsilon < 1$), then there are in expectation at most $\epsilon^2 n$ jobs remaining after one round. Furthermore, for any $\delta > 0$ the probability that there are more than $\epsilon^2 n + \delta$ jobs remaining after one round is upper bounded by $e^{-\frac{\delta^2}{2\epsilon n}}$.*

Proof of Lemma 1. Suppose that there are ϵn unmatched jobs trying to assign themselves randomly to n processors. Let X_i be an indicator that takes value 1 if processor i gets precisely 1 job and 0 otherwise. For every processor there are ϵn choices of jobs, meaning that $\forall i$,

$$\mathbb{E}\{X_i\} = \mathbb{P}\{\text{processor } i \text{ gets exactly 1 job}\} = \epsilon n \cdot \left(\frac{1}{n}\right)\left(\frac{n-1}{n}\right)^{\epsilon n - 1} = \epsilon \left(\frac{n-1}{n}\right)^{\epsilon n - 1}$$

So, by linearity of expectation, the expected number of jobs that got assigned to any processor that received precisely one job is

$$\sum_{i=1}^{n} \mathbb{E}\{X_i\} = \sum_{i=1}^{n} \epsilon \left(\frac{n-1}{n}\right)^{\epsilon n - 1} = \epsilon n \left(\frac{n-1}{n}\right)^{\epsilon n - 1}$$

All of the above jobs will be executed since they had no contention. Therefore, the expected number of jobs remaining after one round is

$$\epsilon n - \sum_{i=1}^{n} \mathbb{E}\{X_i\} = \epsilon n - \epsilon n \left(\frac{n-1}{n}\right)^{\epsilon n - 1} = \epsilon n \left(1 - \left(\frac{n-1}{n}\right)^{\epsilon n - 1}\right) = \epsilon n \left(1 - \left(\left(1 - \frac{1}{n}\right)^{n}\right)^{\frac{\epsilon n - 1}{n}}\right)$$

We know that $\left(1 - \frac{1}{n}\right)^n \le \frac{1}{e} < 1$. Since $\epsilon < 1 \implies \frac{\epsilon n - 1}{n} < 1$, we find

$$\left(\left(1 - \frac{1}{n}\right)^{n}\right)^{\frac{\epsilon n - 1}{n}} \ge \left(\frac{1}{e}\right)^{\frac{\epsilon n - 1}{n}} \ge \left(\frac{1}{e}\right)^{\frac{\epsilon n}{n}} = \left(\frac{1}{e}\right)^{\epsilon}$$

Problem 5 continued on next page. . .

So, the expected number of jobs remaining after one round is at most

$$\epsilon n - \sum_{i=1}^{n} \mathbb{E}\{X_i\} = \epsilon n \left(1 - \left(\left(1-\frac{1}{n}\right)^n\right)^{\frac{\epsilon n - 1}{n}}\right) \leq \epsilon n \left(1 - \left(\frac{1}{e}\right)^{\epsilon}\right)$$

We know that $1 + z \leq e^z$ for all real z, which implies $1 - \epsilon \leq e^{-\epsilon} = \left(\frac{1}{e}\right)^{\epsilon} \implies 1 - \left(\frac{1}{e}\right)^{\epsilon} \leq \epsilon$. Using this, we arrive at the result that the expected number of remaining jobs after one round is

$$\epsilon n - \sum_{i=1}^{n} \mathbb{E}\{X_i\} \leq \epsilon n \left(1 - \left(\frac{1}{e}\right)^{\epsilon}\right) \leq \epsilon n \cdot \epsilon = \epsilon^2 n$$

To provide a concentration bound on this, first let $X = \sum_{i=1}^{n} X_i$ be the random variable of the number of jobs that get processed in this round. Note that we would like to bound the probability that at least $\epsilon^2 n + \delta$ jobs remain unmatched for some $\delta > 0$; this is equivalent to $\mathbb{P}[X < \epsilon n - \epsilon^2 n - \delta]$. We can write

$$\epsilon n - \epsilon^2 n - \delta = \left(1 - \frac{\delta}{\epsilon n - \epsilon^2 n}\right)(\epsilon n - \epsilon^2 n)$$

As we have seen earlier,

$$\mathbb{E}[X] = \epsilon n \left(1 - \frac{1}{n}\right)^{\epsilon n - 1} \geq \epsilon n - \epsilon^2 n$$

So, we apply a Chernoff bound with

$$\begin{aligned}
\mathbb{P}[X < \epsilon n - \epsilon^2 n - \delta] &= \mathbb{P}\left[X < \left(1 - \frac{\delta}{\epsilon n - \epsilon^2 n}\right)(\epsilon n - \epsilon^2 n)\right] \\
&\leq \mathbb{P}\left[X < \left(1 - \frac{\delta}{\epsilon n - \epsilon^2 n}\right)\mathbb{E}[X]\right] \\
&\leq e^{-\frac{\delta^2 \mathbb{E}[X]}{2(\epsilon n - \epsilon^2 n)^2}} \leq e^{-\frac{\delta^2 (\epsilon n - \epsilon^2 n)}{2(\epsilon n - \epsilon^2 n)^2}} = e^{-\frac{\delta^2}{2\epsilon n(1-\epsilon)}} \\
&\leq e^{-\frac{\delta^2}{2\epsilon n}},
\end{aligned}$$

where the last step comes from the fact that $1 - \epsilon < 1$. So, the number of remaining unmatched jobs concentrates well. ■

Main Proof. To prove the main result, we can repeatedly apply the result of Lemma 1 with $\delta = \sqrt{2\epsilon n \cdot log\ (n)}$ to see that we will have more than $\epsilon^2 n + \sqrt{2\epsilon n \cdot log\ (n)}$ jobs remaining with probability less than $\frac{1}{n}$. We can say that this deviation of δ is negligible. To see this, imagine repeatedly applying Lemma 1 until the case where ϵn is exceeded by $\sqrt{2\epsilon n \cdot log\ (n)}$ (i.e. the deviation is too big), which implies that $\sqrt{\epsilon n} < \sqrt{2log\ (n)} \implies \epsilon < \frac{2log(n)}{n}$. So, in the next step, there will be with high probability $\epsilon^2 n < \frac{4log(n)^2}{n}$ jobs left to process, which is less than 1 for all $n > 1$. Such a case is obviously trivial, and so we see that the concentration bound that Lemma 1 provides with $\delta = \sqrt{2\epsilon n \cdot log\ (n)}$ is a useful concentration for all the rounds that we care about. In other words, by the time that ϵn gets small enough that the deviation δ is meaningful, the process has already terminated.

Now, we can finally get to proving the main result. Firstly, we would like to determine how many unmatched jobs to expect after the first round, and how well this amount concentrates. We can follow the logic of the first part of the proof of Lemma 1 to see that the expected number of jobs that get executed is

$$n \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq n \cdot \left(\frac{1}{e}\right)^{\frac{n-1}{n}} \geq \frac{n}{e}$$

Note that this also concentrates reasonably well, as we can apply a similar Chernoff-style bound as before. So, we start in the second step with $n \cdot (1 - \frac{1}{e})$ jobs to be executed $\implies \epsilon_0 = 1 - \frac{1}{e}$, with high probability. Let

Problem 5 continued on next page...

r be the number of rounds that will execute until completion after this starting point. "Completion" occurs when $\epsilon = \frac{1}{n}$, since this corresponds to exactly 1 job remaining that will always successfully be executed in the subsequent round. Furthermore, by Lemma 1 we know that with high probability, ϵ is roughly squared each round. So. we seek r such that

$$(\epsilon_0)^{2^r} = \frac{1}{n} \iff \left(1 - \frac{1}{e}\right)^{2^r} = \frac{1}{n} \iff \left(\frac{e}{e-1}\right)^{2^r} = n \iff r = log\left(\frac{log(n)}{log\left(\frac{e}{e-1}\right)}\right)$$

This gives us the result that the process to terminate after $r = O(log(log(n)))$ rounds with high probability.

To definitively say that this happens with high probability, we can apply a union bound the rounds. From applying the Lemma with $\delta = \sqrt{2\epsilon n \cdot log\ (n)}$, we know that the probability of any individual round behaving worse than expected is less than $\frac{1}{n}$. So, the probability that any of the r rounds behaved worse than expected is at most $\sum_{i=1}^{r} \frac{1}{n} = \frac{O(log(log(n)))}{n}$. Therefore, the probability that things behave as expected (i.e. each application of the Lemma concentrates the way we desire) is at least $1 - \frac{O(log(log(n)))}{n} = 1 - o(1)$. This means that we can conclude that the process terminates after $O(log(log(n)))$ rounds with probability at least $1 - o(1)$, and we are done. ■