COS 521: Homework 4

Due on November 14, 2022

Professor Matt Weinberg

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

Problem 1

Part A

Proof. We wish to solve the following LP

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} w_j(a_j^T x - b_j) \\ \text{subject to} \quad & x_i \geq 0 \quad \forall i \in [n] \\ & \sum_{i=1}^{n} x_i = 1 \end{aligned}$$

We start by noting that we can reformulate the objective as

$$\sum_{j=1}^{m} w_j(a_j^T x - b_j) = \sum_{j=1}^{m} \left(-w_j b_j + w_j \sum_{i=1}^{n} a_{ji} x_i \right) = -\left(\sum_{j=1}^{m} w_j b_j \right) + \sum_{i=1}^{n} \left(x_i \sum_{j=1}^{m} a_{ji} w_j \right)$$

Note that the left term is fixed, as it does not depend on our choice of $\vec{x}$. So, we wish to select x_i's that maximize $\sum_{i=1}^{n} \left(x_i \sum_{j=1}^{m} a_{ji} w_j \right)$, are nonnegative, and sum to 1. The allocation that maximizes this is the one that places $x_{i^*} = 1$ at the component $i^* = argmax_i \left\{ \sum_{j=1}^{m} a_{ji} w_j \right\}$ and $x_{\neq i^*} = 0$ elsewhere. We can show this by contradiction: if there were another optimal allocation that had $x_k = z > 0$ for some $k \neq i^*$, we can get a better value by setting $x_k \leftarrow 0$ and setting $x_{i^*} \leftarrow x_{i^*} + z$. Removing z from x_k will lose $z \sum_{j=1}^{m} a_{jk} w_j$ and adding z to x_{i^*} will gain $z \sum_{j=1}^{m} a_{ji^*} w_j$; since we selected i^* as the maximizing component, the gain will be larger than the loss, and the value of the objective will increase. This reallocation will still satisfy that the x_i's are nonnegative and still sum to 1, and so it forms a feasible solution with a better value. This contradicts the optimality of the $x_k = z > 0$ allocation, and so we see that the optimal allocation must have that $x_k = 0$ for all $k \neq i^*$ and therefore that $x_{i^*} = 1$. So, the solution to the LP is to set $x_{i^*} = 1$ and all other components to 0, where $i^* = argmax_i \left\{ \sum_{j=1}^{m} a_{ji} w_j \right\}$. ■

Part B

Proof. Suppose that there exist non-negative weights $w_1, ..., w_m$ such that the optimal value of the program from part (a) is negative. Suppose, by way of contradiction, that LP 1 is feasible. Then, there exist $x_1, ..., x_n$ such that $x_i \geq 0$ for all i, $\sum_{i=1}^{n} x_i = 1$, and $a_j^T x \geq b_j \iff a_j^T x - b_j \geq 0$ for all j (these are the feasibility conditions for LP 1). Then, since w_j and $a_j^T x - b_j$ are nonnegative for all $j \in [m]$, we see that $w_j(a_j^T x - b_j) \geq 0$ for all j, and so $\sum_{j=1}^{m} w_j(a_j^T x - b_j) \geq 0$. Note that this is the exact form of the objective from the program in part (a); this means that the $x_1, ..., x_n$ that satisfy LP 1 would produce a feasible solution with nonnegative value for the program from part (a). This is a contradiction, since we suppose that the optimal value of part (a)'s program is negative. Therefore, there cannot be a feasible solution to LP 1, and we arrive at the result. ■

Problem 1 continued on next page...

Part C

Proof. Let us first note that for any $\vec{x}$ feasible for LP 2 (meaning it has unit L1 norm and nonnegative components), since $|a_{ij}|, |b_j| \leq 1$ for all i, j, we know by the Triangle Inequality that

$$|a_j^T x - b_j| \leq |b_j| + \sum_{i=1}^{n} |a_{ji}||x_i| \leq 1 + ||\vec{x}||_1 \leq 2 \implies \left|\frac{a_j^T x - b_j}{2}\right| \leq 1$$

We will follow the hint that is given, in which we consider m "experts" to correspond to each of the m inequality constraints in LP 1. We want to use a "cost" for each expert related to how well it satisfies the inequality (i.e. cost is related to $a_j^T x - b_j$), but in order to ensure that each cost lies in the interval $[-1, 1]$, we must rescale by 2. The logic above guarantees that these rescaled costs are in the desired range. We can then use the multiplicative weights algorithm as it is taught in lecture to update the weights w_j in LP 2, hopefully finding a set of nonnegative weights $\vec{w}$ such that either LP 2 has a negative optimal value or we approximately satisfy LP 1.

Precisely put, we run multiplicative weights the following way:

- Fix $\eta \in \left[0, \frac{1}{2}\right]$.
- Initialize each weight $w_j^{(1)} = 1$ for $j \in [m]$.
- In each round t of the algorithm, we solve LP 2 with each weight w_j of the LP being set to a normalized $w_j = p_j^{(t)} = \frac{1}{\sum_l w_l^{(t)}} \cdot w_j^{(t)}$. Let $\vec{x}^{(t)} = x_1^{(t)}, ..., x_n^{(t)}$ be the optimal solution of LP 2 with the weights specified, and let the value it achieves be

  $$OPT(t) = \sum_{j=1}^{m} w_j (a_j^T \vec{x}^{(t)} - b_j) = \sum_{j=1}^{m} p_j^{(t)} (a_j^T \vec{x}^{(t)} - b_j)$$

- If $OPT(t) < 0$, we declare LP 1 to be infeasible and are done.
- Otherwise, observe costs for each expert $j \in [m]$ of

  $$m_j^{(t)} = \frac{a_j^T \vec{x}^{(t)} - b_j}{2}.$$

  Update each expert's weight for the next round to be

  $$w_j^{(t+1)} \leftarrow w_j^{(t)} \left(1 - \eta m_j^{(t)}\right)$$

We can observe the following things: firstly, we have already seen that $\left|\frac{a_j^T \vec{x}^{(t)} - b_j}{2}\right| \leq 1 \implies m_j^{(t)} \in [-1, 1]$ for all times t and all experts j. Also, for all t we can prove that $w_j^{(t)}$ is nonnegative for all $j \in [m]$ by induction: since they initialize at $w_j^{(1)} = 1 \geq 0$ we have a base case. To see the inductive step, $\eta \in [0, 1/2]$ and $m_j^{(t)} \leq 1 \implies (1 - \eta m_j^{(t)}) \geq \frac{1}{2} > 0$ together with $w_j^{(t)} \geq 0$ imply that $w_j^{(t+1)} = w_j^{(t)}(1 - \eta m_j^{(t)}) \geq 0$; so by induction on t, $w_j^{(t)} \geq 0$ for all experts j, for all timesteps t. Therefore, for every expert j and every timestep t, we see $p_j^{(t)} = \frac{1}{\sum_l w_l^{(t)}} \cdot w_j^{(t)} \geq 0$ as well. So, if we were to terminate early at any timestep t (because $OPT(t) < 0$), we would therefore have a set of nonnegative weights $\{p_j^{(t)}\}$ that produce a negative optimal value of LP 2. Part (b) would therefore confirm that LP 1 is infeasible. So, our observations culminate in two things: (1) $m_j^{(t)} \in [-1, 1]$ for all j, t, and (2) if we ever terminate early, we can declare LP 1 to be infeasible.

Problem 1 continued on next page...

Then, suppose that we continued the algorithm without early termination for T rounds. Since the costs are normalized to $[-1, 1]$, we can directly apply Theorem 3 from the Multiplicative Weights lecture to get the result that for any expert j,

$$\sum_{t=1}^{T} \vec{m}^{(t)} \cdot \vec{p}^{(t)} \leq \sum_{t=1}^{T} m_j^{(t)} + \eta \sum_{t=1}^{T} |m_j^{(t)}| + \frac{\ln m}{\eta} \leq \sum_{t=1}^{T} m_j^{(t)} + \eta T + \frac{\ln m}{\eta}$$

Let us inspect the leftmost side of this inequality. We can simplify

$$\vec{m}^{(t)} \cdot \vec{p}^{(t)} = \sum_{j=1}^{m} m_j^{(t)} p_j^{(t)} = \sum_{j=1}^{m} p_j^{(t)} \cdot \frac{a_j^T \vec{x}^{(t)} - b_j}{2} = \frac{OPT(t)}{2}$$

Note that we terminate the algorithm and declare LP 1 infeasible any time that $OPT(t) < 0$; so, if the algorithm has not yet terminated for T rounds, then $OPT(t) \geq 0$ for all $t \in [T]$, meaning that $\sum_{t=1}^{T} \vec{m}^{(t)} \cdot \vec{p}^{(t)} = \sum_{t=1}^{T} \frac{OPT(t)}{2} \geq 0$. So, for all experts j,

$$\sum_{t=1}^{T} m_j^{(t)} + \eta T + \frac{\ln m}{\eta} \geq 0 \implies \frac{1}{2} \sum_{t=1}^{T} (a_j^T \vec{x}^{(t)} - b_j) + \eta T + \frac{\ln m}{\eta} = \frac{a_j^T}{2} \left(\sum_{t=1}^{T} \vec{x}^{(t)} \right) - \frac{b_j T}{2} + \eta T + \frac{\ln m}{\eta} \geq 0$$

If we denote $\vec{x}^* = \frac{1}{T} \sum_{t=1}^{T} \vec{x}^{(t)}$ to be the average of our LP solutions over the T rounds, we can see that for all j,

$$\frac{T a_j^T}{2} \vec{x}^* - \frac{b_j T}{2} + \eta T + \frac{\ln m}{\eta} = \frac{T}{2} \cdot (a_j^T \vec{x}^* - b_j) + \eta T + \frac{\ln m}{\eta} \geq 0 \implies a_j^T \vec{x}^* - b_j \geq -\left(2\eta + \frac{2 \ln m}{\eta T} \right)$$

Let $\epsilon > 0$ be arbitrary. Let us set $T = \frac{16 \ln m}{\epsilon^2}$ and $\eta = \sqrt{\frac{\ln m}{T}} = \frac{\epsilon}{4}$. Then, we see that this inequality reduces to the fact that for all j,

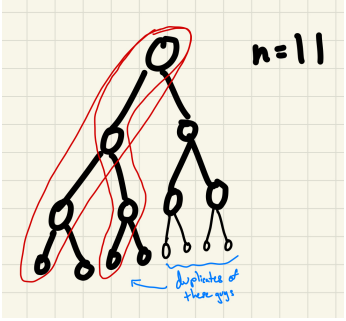
$$a_j^T \vec{x}^* - b_j \geq -\left(\frac{\epsilon}{2} + \frac{8}{\epsilon} \cdot \frac{\epsilon^2}{16} \right) = -\epsilon$$

So, with this selection of η (which is certainly less than $\frac{1}{2}$ and therefore valid for all interesting ϵ), after $T = \frac{16 \ln m}{\epsilon^2}$ rounds of this algorithm, either (a) the algorithm will terminate early and we will declare LP 1 infeasible or (b) we will obtain a solution $\vec{x}^*$ that approximately satisfies each constraint of LP 1 within ϵ. Since $\vec{x}^*$ is an average of $\vec{x}^{(t)}$'s that were all feasible solutions to LP 2 and each had nonnegative components that summed to 1, we know that $\vec{x}^*$ has also has nonnegative components that sum to 1, meaning it satisfies the last two constraints of LP 1. So, the above algorithm will certainly terminate within $O\left(\frac{\ln m}{\epsilon^2}\right)$ rounds and will either (a) declare LP 1 infeasible or (b) approximately solve it. Since each round solves precisely one LP of the form in LP 2, we have proven the claim. ■

Problem 2

Solution

Proof. Consider a universe $U = \{1, ..., n\}$ of n elements. We will construct our family of sets $\mathcal{S}$ by constructing a complete binary tree of the n elements in any arbitrary configuration where each element corresponds to exactly one node. If the binary tree is not full (i.e. if n is not a sum of powers of 2), we duplicate leaf nodes to make new leaf nodes until it is full. Note that this tree must have $\lfloor \log_2 n \rfloor + 1 = \Omega(\log n)$ levels. Also, since we assured that it is full, we will have $\Theta(n)$ leaf nodes. The idea is to let each leaf node, which comes equipped with a unique simple path from the root to the leaf, define a subset of U consisting of the elements on this path. In other words, each set in $\mathcal{S}$ will correspond to the elements connected in this binary tree by a simple path from the root to a certain leaf. Since there are no duplicates anywhere except potentially in the last level (we may have filled the last level), all of these sets are distinct and therefore $m = |\mathcal{S}| = \Theta(n)$ is the number of leaf nodes. A drawing of this construction is displayed below for $n = 11$. I circle in red two potential sets in $\mathcal{S}$; there are 8 total, one for each leaf node. The 4 rightmost leaf nodes were made by duplication to ensure that the tree is full: this doesn't break anything.



Equipped with this construction, we can fool any deterministic online set cover algorithm. Each of the possible sets that the algorithm can choose from is a set of elements of U corresponding to a simple path from the root to a leaf. We start by revealing the element $r \in U$ corresponding to the root. The algorithm must select a set in $\mathcal{S}$ containing r. This set will either contain the element $l \in U$ corresponding to the left child of the root or the element $r \in U$ corresponding to the right child of the root; there is no set in $\mathcal{S}$ containing both l and r (since this wouldn't correspond to a simple path from root to leaf). If the selected set contains l we choose to reveal r next; otherwise we reveal l. By construction, the element we reveal hasn't yet been covered by sets selected by the algorithm, and so it must select another. Put precisely, suppose by induction that we have just revealed a certain element that the algorithm hasn't yet covered, corresponding to a certain node. The algorithm will select a set to cover this node, but the set it selects will leave exactly one child uncovered. In the next step we reveal the uncovered child, ensuring that it is an element the algorithm hasn't yet covered and allowing this scheme to continue. Since the algorithm starts by not having covered the root (base case), induction shows that each iteration of this method reveals an element the algorithm hasn't yet covered, forcing the online algorithm to select a new set.

In this way, we can reveal $\Omega(\log n)$ elements of U (one for each level of the tree, since each iteration moves us down one level in the tree) and every iteration the online algorithm must add a new set to their cover. Therefore, the cost of an online algorithm in this construction must be $\Omega(\log n)$. Since $m = |\mathcal{S}| = \Theta(n)$, the online cost can also be written as $\Omega(\log(mn))$. However, note that throughout this process we have revealed elements corresponding to a simple path from root to a leaf of the tree, since we always followed links from parent to child. By construction, there exists a single set in $\mathcal{S}$ corresponding to this path. So, all the elements we revealed can actually be covered by a single set, and therefore the offline optimal solution has cost of 1. This implies that no deterministic online algorithm can have $o(\log(mn))$ competitive ratio. ■

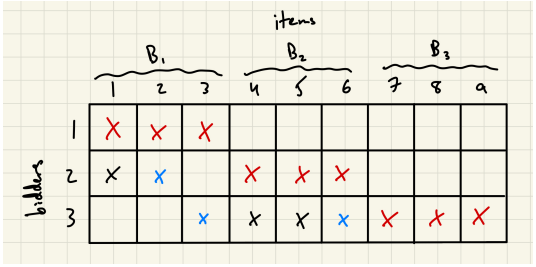
Problem 3

Solution

Proof. Let the number of items m up for auction be arbitrary. Suppose for simplicity that it is a perfect square (if not, we could treat everything with the floor or ceiling of the square root of m and get the same asymptotic behavior, but that would be very annoying :)). Let there be $n = \sqrt{m}$ bidders. We will design the valuation functions such that each bidder has a set of items they desperately want: if they get every item in this set (and potentially other items too) they have a value of 1, and otherwise a value of 0. Firstly, split the items $\{1, ..., m\}$ into $\sqrt{m}$ many blocks B_j, each of size $|B_j| = \sqrt{m}$ such that the B_j's are disjoint and each item is in a block (basically, partition the items into $\sqrt{m}$ blocks of size $\sqrt{m}$ and label each block B_j for $j \in \{1, ..., \sqrt{m}\}$). We will assign the sets in the following way:

- Initialize for each bidder $i \in \{1, ..., \sqrt{m}\}$ a desired set $D_i = \emptyset$.
- For each block $j \in \{1, ..., \sqrt{m}\}$ do the following:
  - (1) Update $D_j \leftarrow D_j \cup B_j$.
  - If $j < \sqrt{m}$, for each bidder $i \in \{\sqrt{m}, ..., j+1\}$ do the following:
    - Remove the last element of B_j from B_j, and call it x.
    - (2) Update $D_i \leftarrow D_i \cup \{x\}$.
  - If $j < \sqrt{m}$, after this there will be j elements left in B_j since we started with $\sqrt{m}$ elements and popped $\sqrt{m} - j$ of them. (3) Append all these remaining elements to D_{j+1}.

I am really sorry for the goofy and algorithmic presentation of this construction, I wanted to feel thorough and this is the result. I attach below a picture of an example construction for the case where $m = 9$ for clarity. In this picture, the columns correspond to items and the rows correspond to bidders. There is an X in an entry of the table if that item is an element of that bidder's desired set. I wrote in red the X's that came from updates of type (1), in blue the X's that came from updates of type (2), and in black the X's that came from updates of type (3) to try and make this more transparent. In this example, bidder 1 has a desired set $D_1 = \{1, 2, 3\}$, bidder 2 has $D_2 = \{1, 2, 4, 5, 6\}$, and bidder 3 has $D_3 = \{3, 4, 5, 6, 7, 8, 9\}$.



Now that we have seen the construction, let's look at some properties. The valuation functions are defined for each bidder $i \in \{1, ..., \sqrt{m}\}$ to be $v_i(S) = \begin{cases} 1 & D_i \subset S \\ 0 & else \end{cases}$.

Lemma 1. *The welfare of the optimal deterministic allocation is 1.*

Proof of Lemma 1. We can see in this construction that each bidder i will want at least one item from every block B_j such that $j < i$ because of updates of type (2). Also, each bidder i will want all the items from the block B_i because of updates of type (1). In other words, for each i the desired set D_i contains at least one element from each of the blocks $B_1, ..., B_{i-1}$ and it contains all the elements from block B_i. In particular, this means that for any two distinct bidders $i < j$, $D_i \cap D_j \neq \emptyset$: this is because $B_i \subset D_i$ and at

least one element of B_i is also in D_j. This important property means that for any deterministic allocation, if one bidder receives all the elements in their desired set then no other bidder can also be satisfied (since any pair of bidders must be in contention for at least one item). So, any deterministic allocation of items can only create nonzero value from at most one bidder: therefore, the welfare of the optimal deterministic allocation is 1. ■

Lemma 2. *There exists a feasible fractional allocation for the configuration LP with value at least* $\frac{\sqrt{m}}{2}$.

Proof of Lemma 2. We can see from the construction of the desired sets that each item $k \in [m]$ is placed in the desired set of at most 2 bidders, since it can only be desired because of an update of type (1), which happens once for the entire block, an update of type (2), which happens at most once, or an update of type (3), which can only happen for item k if an update of type (2) didn't. So, we can safely allocate $\frac{1}{2}$ of each item to every bidder that desires it, such that each bidder receives half of their desired set and outputs half of their value.

In the more precise language of the configuration LP, we can say that for any bidder i and any subset $S \subset [m]$, we set

$$x_{i,S} = \begin{cases} \frac{1}{2} & S = D_i \\ 0 & else \end{cases}$$

We can see that for any item $k \in [m]$, there are at most two desired sets that contain it. So, there are at most two subsets $S \subset [m]$ with nonzero $x_{\cdot,S}$ that contain it: one for each of the bidders that desires it (all other sets yield coefficients of 0). Therefore, we can say that for all items k,

$$\sum_i \sum_{S \ni k} x_{i,S} \leq \frac{1}{2} + \frac{1}{2} = 1,$$

where for items not in the last block $B_{\sqrt{m}}$ the inequality is tight. We can also reason that for every bidder i, the only set that gives a nonzero $x_{i,S}$ is exactly $S = D_i$, and so for all i

$$\sum_S x_{i,S} = x_{i,D_i} = \frac{1}{2} \leq 1$$

Lastly, each $x_{i,S} \geq 0$ by construction. These three inequalities show that this fractional allocation is feasible for the configuration LP. We can find the value of this allocation to be

$$\sum_{i,S} x_{i,S} v_i(S) \geq \sum_i x_{i,D_i} v_i(D_i) = \sum_i \frac{1}{2} \cdot 1 = \frac{\sqrt{m}}{2},$$

since there are $\sqrt{m}$ bidders. ■

Lemmas 1 and 2 yield that the optimal deterministic allocation has welfare 1, but there exist feasible fractional allocations for the configuration LP with value $\Omega(\sqrt{m})$. This shows that the integrality gap of the configuration LP is at least $\Omega(\sqrt{m})$. ■

Problem 4

Solution

Proof. Suppose that there are m items and n bidders, each with a monotone submodular valuation function $f_i(\cdot)$. Let $GREEDY(m)$ be the value of the welfare that the greedy algorithm achieves, and let $OPT(m)$ be the value of the welfare that the optimal allocation achieves. We want to show that $2 \cdot GREEDY(m) \geq OPT(m)$. We will do so by considering the problem where we have yet to allocate the first item 1. Let $g \in [m]$ be the bidder that the greedy algorithm allocates 1 to. Define $f_g^-(S) = f_g(S \cup \{1\}) - f_g(\{1\})$ to be the marginal value that g gets for any set, assuming bidder g already had item 1. For the other bidders $i \neq g$, define $f_i^-(S) = f_i(S)$. With this setup, we essentially get a subproblem where we wish to allocate $m-1$ items, and bidder g has the modified valuation function. Let $GREEDY(m-1)$ be the value of the welfare for this subproblem, and let $OPT(m-1)$ be the optimal welfare value for this subproblem. Clearly,

$$GREEDY(m) = GREEDY(m-1) + f_g(\{1\}),$$

since bidder g already gets 1 in the greedy solution to the problem with m items. Let bidder o be the bidder that gets item 1 in the optimal solution that achieves value $OPT(m)$. Suppose without loss of generality that $o \neq g$; if they are equal we can consider the subproblem where we ignore item 1 without anything changing, and start applying the above logic from there. Now, we can say that if $S_1, ..., S_n$ is the optimal allocation that achieves value $OPT(m)$, the allocation $S_1, ..., S_{o-1}, S_o \setminus \{1\}, S_{o+1}, ..., S_n$ obtained by removing 1 from S_o is a valid solution for the subproblem. Consider the value V gotten from this modified in the subproblem: we can see that

$$f_o(S_o) - f_o^-(S_o \setminus \{1\}) \leq f_o(\{1\}) - f_o(\emptyset) = f_o(\{1\}),$$

by the submodularity of f_o and the fact that $f_o^- = f_o$ since $o \neq g$. Also, we get that

$$f_g(S_g) - f_g^-(S_g) = f_g(S_g) - f_g(S_g \cup \{1\}) + f_g(\{1\}) \leq f_g(\{1\}),$$

by the monotonicity of f_g and plugging in our modified f_g^-. Lastly, we can note that since the greedy algorithm selected bidder g to get the first item, we must have that $f_o(\{1\}) \leq f_g(\{1\})$. These together yield that

$$V \geq OPT(m) - f_g(\{1\}) - f_o(\{1\}) \geq OPT(m) - 2f_g(\{1\})$$

Since V is a valid allocation for the subproblem, we also know that $V \leq OPT(m-1)$ since $OPT(m-1)$ is optimal. Putting all these togetther, we get that

$$OPT(m-1) \geq OPT(m) - 2f_g(\{1\})$$

We can handle the base case easily: when $m = 1$, the greedy algorithm certainly selects the maximal welfare allocation, and so we trivially have $2 \cdot GREEDY(1) \geq OPT(1)$. Suppose by way of induction that $2 \cdot GREEDY(m-1) \geq OPT(m-1)$. Then,

$$2 \cdot GREEDY(m-1) \geq OPT(m) - 2f_g(\{1\})$$

So, the first inequality we derived gives

$$2 \cdot GREEDY(m) = 2 \cdot GREEDY(m-1) + 2f_g(\{1\}) \geq OPT(m) - 2f_g(\{1\}) + 2f_g(\{1\}) = OPT(m)$$

So, by induction, the claim holds for all m. ■

Problem 5

Solution

Proof. Suppose that Alice and Bob are given two different bitstrings A and B, respectively. We make no assumptions of their lengths. Suppose that Alice and Bob have access to an infinite, shared random stream of bits. The communication protocol goes as follows:

- Alice appends a 1 to A and Bob appends a 1 to B, yielding A' and B' respectively. This is to avoid degeneracy with leading 0's.
- Let a be the decimal number that is represented by the bitstring A' (i.e. if $A' = 10010$ then $a = 18$) and let b be the decimal number that is represented by the bitstring B'.
- Alice inspects the shared random bit stream at the indices given by a and $a + 1$ - call them x_A, y_A - and communicates these two bits. Bob does the same, communicating the bits located at indices b and $b + 1$ in the shared random bit stream - call them x_B, y_B.
- If Alice's two bits match Bob's two bits ($x_A = x_B$ and $y_A = y_B$), everyone outputs "yes". Otherwise, everyone outputs "no".

Let's analyze this. Clearly, both Alice and Bob output only $O(1)$ bits, since they both output exactly 2 bits each. Also, if Alice and Bob had the same input bitstrings $A = B$, then it will be the case that $A' = B'$ and $a = b$, and the broadcasted pairs of bits (x_A, y_A) and (x_B, y_B) will be identical. So, if Alice and Bob have equal inputs, they will certainly output "yes". Suppose now that Alice and Bob have different inputs. Then, $a \neq b$. So, the probability that $x_A = x_B$ is equal to the probability that two different random bits in the stream are equal: this happens with probability $\frac{1}{2}$ since the bits in the shared random bitstream are distributed i.i.d. Similarly, the probability that $y_A = y_B$ is also $\frac{1}{2}$. Since these are two independent events (the bits in the shared random bitstream are distributed i.i.d.), the probability that both $x_A = x_B$ and $y_A = y_B$ is $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. So, the probability that Alice and Bob output "yes" when given two different inputs is $\frac{1}{4}$. This means that if Alice and Bob have different inputs, they will output "no" with probability $\frac{3}{4} > \frac{2}{3}$. The above protocol has all of the desired properties. ■