Problem Set 3, Answers

Kevin Lacker

June 2, 2020

Problem 1. This is a four-part problem.

1. Let L be the set of leaves of the minimum-height protocol tree for f that give output 1. For each $l \in L$, let M_l be the matrix which is 1 on the input pairs whose path through the protocol tree ends at the lead l. By the definition of a protocol tree, M_l can also be defined by a set X and Y with:
$$M_l[i,j] = 1 \iff i \in X, j \in Y \tag{1}$$
Thus, there are two possible rows in M_l, either all zeros or 1 precisely on indices in Y. Thus, the rank of M_l is 1, regardless of what field the rank is taken over.

   Note that:
$$M_f = \sum_{l \in L} M_l \tag{2}$$
Since rank is subadditive, we can conclude:
$$rank(M_f) \leq \sum_{l \in L} rank(M_l) = |L| \leq 2^{D(f)} \tag{3}$$
The last inequality holds since L is a subset of leaves of a tree of height $D(f)$. We can then conclude
$$log_2(rank(M_f)) \leq D(f) \tag{4}$$

2. There are 2^n rows of M_f. Consider them as vectors over the field $\mathbb{F}_2$, and say the rows span a subspace of dimension d, so $rank(M_f) = d$. Since the field has size 2, this subspace has exactly 2^d elements in it. Since the rows of M_f are 2^n distinct vectors in this subspace of size 2^d, we have $2^n \leq 2^d$. So $log_2(n) \leq log_s(d)$ and by the conclusion of the previous problem, $\log_2(n) \leq D(f)$.

3. First consider M_f as a matrix over $\mathbb{F}_2$. If the rank is r, we can write any row in the matrix as a linear combination of r basis vectors. Since the constant on each basis vector can only be 0 or 1, we can communicate our

entire row by sending r bits, and then the other player knows our row, so they can send a bit for the answer. Thus

$$D(f) \leq rank(M_f) + 1 \tag{5}$$

This rank is over $\mathbb{F}_2$, but the rank over the reals cannot be smaller than the rank over any other field, so the equation holds for rank over the reals as well.

4. TODO

Problem 2. Our randomized protocol for GT will use the randomized protocol for EQ, to do a binary search to find the first bit in which the two bit strings disagree. The algorithm works like:

- Find the middle of our bit string.
- Use the EQ protocol to determine whether the bits to the left of the midpoint are the same in the two strings.
- If they are the same, recurse on the right half of the string. If they are not the same, recurse on the left half of the string.
- Once we have found the first bit that disagrees, in constant communication we can figure out which of the strings is greater.

We have $O(logn)$ steps, and each step introduces a constant error. We need to reduce this error to $O(1/log(n))$ so that it is constant over the whole algorithm, so we need to repeat the protocol at each steg $O(log(log(n)))$ times. This gives a total communication complexity of $R^{pub}(GT) \leq O(log(n)log(log(n)))$.

Newman's theorem states that this can be converted into a private coin protocol with an additive log penalty, but this complexity is larger than the log penalty already, so the same asymptotic bound holds for $R(GT)$ as for $R^{pub}(GT)$.

Problem 3. Assuming f has a protocol tree T with l leaves, our task is to construct another protocol tree for f that has depth $O(l)$.

First, let's find a subtree of T that contains approximately half of its nodes. Start at the root and at each step recurse toward the larger subtree. The size of the subtree can drop by a factor of at most $1/2$ in each iteration. It starts at l and ends at 1. Therefore, it must pass through the range of $l/3$ to $2l/3$ at some point. We pick the subtree in that range and call it U.

So T can be seen as the combination of two trees, U and $T - U$. Each of these trees is at most $2/3$ the size of T. Each node in T corresponds to a rectangle, so there are two sets X and Y such that inputs (x, y) are determined by T iff $x \in X$ and $y \in Y$.

We can also use recursion to find two balanced trees, B_U and B_{T-U}, which are protocol trees for U and $T - U$ respectively, with depth is less than $c \cdot log(2l/3)$ for some constant c.

Now, we can construct a new protocol tree for f.

- Determine whether the inputs are in the (X, Y) rectangle. This just takes two bits of communication, one from each player whether their input is in the rectangle.
- If they are in the rectangle, use B_U.
- If they are not in the rectangle, use B_{T-U}.

The total depth of this tree is $2 + c \cdot log(2l/3)$, which for large enough c is less than $c \cdot log(l)$. So there is a protocol tree for f with $O(log(l))$ depth.

Problem 4. Rather than solve this problem once using $O(log^2(n))$ and again in $O(log(n))$, we'll just do it the harder way, which implies the easier way.

The first step is to use a log amount of communication to reduce the problem to one where the sets are either the same size or differ in size by 1 element. Alice and Bob can communicate the size of their sets with log communication. Without loss of generality, assume that Bob has more elements. Bob then calculates the interval b_{low} to b_{high} that contains his median elements that would be the same amount of elements, plus or minus 1, as Alice has. So the number of elements Bob has below b_{low} and the number of elements above b_{high} are equal. Bob then communicates these bounds to Alice.

If Alice's elements are either all above b_{low} or all above b_{high}, she communicates this fact to Bob. Bob is then able to calculate the median all by himself, because Alice's elements are all known to lie on one particular side of the median.

Otherwise, we know the median lies within the b_{low} and b_{high} interval. Bob can then discard his elements outside that interval without affecting the median, and the algorithm can continue.

Now we need a recursion phase. Alice and Bob have $O(n)$ elements (same or off by 1), and they know the median lies in a particular range of size $O(n)$. Alice and Bob pick the midpoint of the range and tell each other whether most of their items are above or below of the midpoint, or whether they are split evenly. There are a few possible outcomes.

1. If the majority of Alice's and Bob's items are on the same side of the midpoint, we know the median lies on that side as well. Recurse on the smaller range. This also works if one player has their items evenly split, and the other player has a majority.
2. If the majority of Alice's and Bob's items are on opposite sides of the midpoint, they can both discard half of their items without changing the midpoint. (If the amounts are off by 1, round so that each player discards the same amount.) Recurse on the same range, but with fewer elements.
3. If both Alice and Bob have their items evenly split on each side of the midpoint, we know the median is the average of the highest item below the midpoint and the lowest item above it. So Alice and Bob can each report their closest items to the midpoint, both above and below, and

this $O(log(n))$ communication is enough to calculate the median, without recursing.

Each phase of the recursion takes constant communication and cuts either the size of the sets or the size of the range by a constant fraction, so the total communication is $O(log(n))$.

Problem 5. 1. Here is a recursive algorithm for finding $|C \cap I|$. Note that it must be either 0 or 1 because only a zero-vertex or single-vertex subgraph can be both a clique and an independent set.

- Alice finds the vertex v_A in C with the smallest degree. If the degree is no more than $n/2$, communicate this vertex to Bob. If Bob has this vertex, we are done. If Bob does not have this vertex, we have at least communicated that the answer must be adjacent to v_A. Discarding the vertices that are not adjacent to v_A, we can recurse on the rest.
- Bob then finds the vertex v_B in I with the largest degree. If the degree is no less than $n/2$, communicate this vertex to Alice. If Alice has this vertex, we are done. If Alice does not have this vertex, we have at least communicated that the answer cannot be adjacent to v_B. Discarding the vertices that are adjacent to v_B, we can recurse on the rest.
- If neither Alice nor Bob has a vertex satisfying these conditions, we know none of their vertices can overlap, since the possible range of their degrees does not overlap. So we can return 0.

The recursion takes at most $O(log(n))$ steps, and each step has communication of size $O(log(n))$, so the total communication for this algorithm is $O(log^2(n))$.

2. We present a protocol for solving any function based on solving the CIS_G problem. First, divide all the ones in the communication matrix M_f into $C_1^D(f)$ disjoint rectangles. Make a graph G where each vertex in the graph corresponds to one of these rectangle. Connect two rectangles if they share a row.

Now, let's design a protocol to solve f. The question is equivalent to asking whether the point (x, y) lies in one of these rectangles in our graph. Let Alice be the player who knows x, ie which row the rectangle would have to overlap with, and Bob be the player that knows y, ie which column the rectangle would have to overlap with. Alice knows the row, which corresponds to a clique of rectangles. Bob knows the column, and since the rectangles are disjoint, no two rectangles can share both a column and a row, so Bob knows an independent set the rectangle must lie in. Thus, if we can solve the CIS_G problem in $O(g(n))$ communication, we can solve any problem in $O(g(C_1^D(f)))$ communication.

Substituting in $log^c(n)$ for $g(n)$ in this gives us the desired implication:

$$D(CIG_G) \leq O(log^c(n)) \implies D(f) \leq O(log(C_1^D(f))^c) \tag{6}$$