Problem Set 2

CS6230: Topics in Information Security
Due on: Sep 18, 2021
Total Points: 100

- Collaboration on the problem set is encouraged. Write up your submission on your own, though, and list the names of all of your collaborators.
- Solutions to some of the problems in this problem set are easy to find on the internet or in textbooks. Please do not look them up.
- Formal mathematical proofs (e.g., of completeness and soundness) are required in support of your answer to each of the problems.

---

1 Errors in Interactive Proofs

We will prove here some results about interactive proofs that were mentioned in class. Below, unless otherwise specified, IPs have completeness and soundness errors 1/3. Also, assume that at the end of any public-coin protocol, after all messages have been sent and received, the verifier makes its decision to accept or reject deterministically – this may be arranged simply by adding an extra dummy round of interaction if necessary, and is often how constant-round public-coin protocols are defined.

1.1 Parallel Repetition

In class, we saw that repeating an IP sequentially and check whether a majority of the repetitions are accepted exponentially decreases the completeness and soundness errors. This effect can also be achieved by repeating the IP in parallel, which has the added benefit of not increasing the number of rounds of communication. For clarity, suppose an IP with $2k$ messages looks as follows.

1. Given input x, the verifier uses random string r to generate its first message $\alpha_1 \leftarrow V(x; r)$ and sends it to the prover.
2. The (honest) prover responds with its first message $\beta_1 \leftarrow P(x, \alpha_1)$.
3. Then the messages alternate as $\alpha_2 \leftarrow V(x, \alpha_1, \beta_1; r)$, $\beta_2 \leftarrow P(x, \alpha_1, \beta_1, \alpha_2)$, and so on.
4. At the end, the verifier outputs accept or reject computed as $V(x, \alpha_1, \beta_1, \ldots, \alpha_k, \beta_k; r)$.

In an m-fold sequential repetition, m instances of the above protocol are run one after the other, with the verifier picking an independent random string r each time. An m-fold parallel repetition, on the other hand, looks like this:

1. Given input x the verifier samples m random strings $r^1, \ldots, r^m$. For each $j \in [m]$, it computes the message $\alpha_1^j \leftarrow V(x; r^j)$, and sends all the α_1^j's to the prover.
2. The (honest) prover similarly computes, for each $j \in [m]$, a response $\beta_1^j \leftarrow P(x, \alpha_1^j)$, and sends all of the β_1^j's.
3. Then they alternate sending $\left\{\alpha_2^j\right\}$, $\left\{\beta_2^j\right\}$, and so on.
4. At the end, the verifier accepts iff, for a majority of $j \in [m]$, the $V(x, \alpha_1^j, \ldots, \beta_k^j)$'s accept.

The fact that such parallel repetition improves errors is harder to prove than the same for sequential repetition, but is true. Think about why the proof for sequential repetition we saw in class does not carry over here. We will concern ourselves now with the special case of perfectly complete IPs, for which amplification is easier to show than in general. In this case, instead of the verifier seeing if a majority of the repetitions accept, it checks whether all of them accept.

Problem 1.1 (20 points)**.** *Show that when a public-coin IP with perfect completeness and soundness error* 1/3 *is repeated* m *times in parallel, with the verifier accepting iff all the repetitions accept, the resulting IP is still perfectly complete and its soundness error is at most* $2^{-\Omega(m)}$*.*

Hint 1.1. *It turns out that here too it is true that the best prover strategy is to operate independently on each of the repetitions, though this is harder to prove directly here than it was in the case of sequential repetition. (What goes wrong if you try to apply the argument we saw in class?) You may try to prove this directly, but the solution I have in mind only does so implicitly, and instead goes through the following slightly opaque function. For any verifier messages $\alpha_1, \ldots, \alpha_i$ and prover messages $\beta_1, \ldots, \beta_i$,*

$$p(\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i) = \max_{P^*} \Pr\left[(P^*, V) \text{ accepts} \mid \text{the first } 2i \text{ messages were } (\alpha_1, \beta_1, \ldots, \alpha_i, \beta_i)\right]$$

Do you see how to extend this proof to IPs with imperfect completeness?

1.2 Perfect protocols

Problem 1.2 (10 points)**.** *Show that if a language L has a 2-message public-coin IP where the verifier speaks first (that is, an AM proof), then it also has a 3-message public-coin IP with perfect completeness.*

Hint 1.2. *Look at the ideas in the proof of the Sipser-Lautemann Theorem on Wikipedia. This theorem that states that* BPP *is contained in* PH*. How can you use the ideas from that proof here?*

Think about how you would extend this transformation to work for protocols with more rounds. Due to the existence of the above transformation, sometimes interactive proofs are defined as already having perfect completeness. Perfect soundness, on the other hand, turns out to be much more restrictive.

Problem 1.3 (5 points)**.** *Show that if a language L has an IP with perfect soundness, then $L \in$ NP.*

2 Derandomising Balls and Bins

Recall the process of tossing N balls into N bins from the previous problem set. There, for each ball, a bin was chosen at random, and the ball was tossed into it. We were interested, then, in bounding the maximum number of balls in any bin at the end of this process. This may be framed in the following equivalent manner. Consider the function h that maps each ball to the bin it is tossed into. That is, h is a mapping from $[N]$ to $[N]$, and for each $i \in [N]$, the bin that the i^{th} ball is tossed into is $h(i)$. What we proved last time, then, was that if h is chosen to be a uniformly random function, then, with high probability, the ball-tossing strategy represented by h results in at most $O(\log N)$ in any bin.

This abstraction of tossing balls into bins is quite common in the design of interactive protocols, and also more generally in cryptography. In many of these circumstances, however, we want this to be done effectively. That is, we would like the function h above to be efficiently computable given its input i as a $\log N$-bit string. For this purpose, random functions h are not good. So it is common to try to replace them with more efficient families of functions that still preserve properties of interest. The property of minimising the maximum number of balls in any bin is referred to as "load-balancing". We will be interested here in designing families $\{H : h[N] \to [N]\}$ of functions such that for random ball-tossing functions h sampled from these families, the maximum number of balls in any bin is as close as possible to that with completely random functions h.

Below, consider N to be quite large, say at least 100. By "high probability", we mean some large constant probability, say at least 0.9. Perhaps the most common kind of functions used to derandomise such task are pairwise independent functions.

Problem 2.1 (5 points)**.** *Prove that if H is a family of pairwise independent functions, then with high probability the maximum number of balls in any bin is at most $O(\sqrt{N})$.*

Here, however, they are not quite good enough. But it turns out this form of limited independence is not a bad direction to take.

Problem 2.2 (2 points)**.** *Along the lines of pairwise independence, for larger $t \in \mathbb{N}$, write down a definition of a notion of t-wise independence. It should capture the independence of the outputs on any fixed t input values, in the same way that pairwise independence captures the independence of the outputs on any two input values.*

Notice that in this case, we want our function h to have the same domain and range $[N]$. This makes constructions of such families quite simple. Let N be a prime number for our convenience. Consider the field $\mathbb{F}_N$, and the family of all linear functions $H_2 = \{h_{a,b} : \mathbb{F}_N \to \mathbb{F}_N\}$, where $a, b \in \mathbb{F}_N$, and $h_{a,b}(x) = ax + b$.

Problem 2.3 (3 points)**.** *Prove that the above family H_2 is pairwise independent.*

Note, further, that a random $h_{a,b} \in H_2$ is efficient to both sample and compute – both can be done in time polylog(N).

Problem 2.4 (10 points)**.** *For any $t < N$, construct a family of t-wise independent hash functions $H_t = \{h : \mathbb{F}_N \to \mathbb{F}_N\}$ such that a random h from the function can be sampled in $\mathsf{poly}(t, \log N)$ time, and for any $x \in \mathbb{F}_N$, the output $h(x)$ can be computed in $\mathsf{poly}(t, \log N)$ time.*

With t-wise independent families for large enough t, we can indeed replicate the load-balancing properties of random functions with efficient tossing functions.

Problem 2.5 (10 points)**.** *Prove that if the family of ball-tossing functions H is a family of t-wise independent functions for $t = \log N$, then with high probability the maximum number of balls in any bin is at most $\log N$.*

3 Doubly Efficient IPs

There are many interesting structured problems that have doubly efficient interactive proofs. Here we will see an example. Below, you can use any of the protocols we have seen in class as subprotocols, and make use of their known error and efficiency bounds. Assume the existence of an oracle that, given any n, outputs the smallest prime number larger than n – this will help set aside issues of the prover and verifier agreeing on which finite field to use, etc., which is something to care about otherwise with doubly efficient IPs.

3.1 Orthogonal Vectors

In the Orthogonal Vectors (OV) problem of dimension d, an instance consists of two sets U and V of n vectors each from $\{0,1\}^d$. That is, $U = \{u_i\}_{i \in [n]}$ and $V = \{v_i\}_{i \in [n]}$, where each u_i and v_i are from $\{0,1\}^d$. The problem is to decide whether there exist $i, j \in [n]$ such that u_i and v_j are disjoint – that is, the inner product $\langle u_i, v_j \rangle = 0$ (over $\mathbb{N}$). This problem can be solved trivially in time $O(n^2 d)$, and with some effort by marginally more efficient algorithms, but it is believed that, for $d = \omega(\log n)$, it takes at least $n^{2-o(1)}$ time to solve.

Problem 3.1 (20 points)**.** *Construct a IP for the OV problem of dimension $\log^2(n)$ where the prover runs in time $\widetilde{O}(n^2)$, and the verifier in time $\widetilde{O}(n)$.*

Hint 3.1. *Notice that the number of orthogonal vectors is given by the sum $\sum_{i,j \in [n]} (1 - u_{i1}v_{i1})(1 - u_{i2}v_{i2}) \cdots (1 - u_{id}v_{id})$. If only you could write the terms in the sum as evaluations of a polynomial in i and j somehow, you could use sumcheck. How would you do this? Note that evaluations of this polynomial should still be computable in $\widetilde{O}(n)$ time for the verifier to be efficient. Can you write the different u_{ik}'s and v_{ik}'s as evaluations of some polynomials?*

4 Set Upper Bounds

We saw that, given access to a membership oracle M_S for a set $S \subseteq \{0,1\}^m$, and a number $t \leq m$, there is an IP that accepts with high probability if $|S| \geq 2^t$, and rejects with high probability if $|S| \leq 2^t/10$. This protocol, in a sense, proves a lower bound of 2^t on the size of the set S. A protocol to prove an *upper bound* on the size of S, however, is not known in this setting. However, such a protocol is possible with one additional assumption – at the beginning of the protocol execution, the verifier is privately given a uniformly random element $x \leftarrow S$ that is not known to the prover. Assume you have access to any necessary efficient k-wise independent family of functions for any constant k.

Problem 4.1 (15 points)**.** *In the above setting, construct an interactive protocol where the verifier accepts with high probability if* $|S| \leq 2^t$*, and rejects with high probability if* $|S| \geq 8 \cdot 2^t$*.*

Hint 4.1. *Try constructing a protocol using random functions, then see how you can derandomise it.*