Homework 2

Due: 18 October, 2024

Each problem is graded on the coarse scale of $\checkmark^+$, $\checkmark$, $\checkmark^-$ and no $\checkmark$. It is also assigned a multiplier, denoting the relative importance of the problem. Both correctness and presentation are grading criteria.

Please read and make sure you understand the collaboration policy on the course missive. Extra credit problems are clearly marked below (see course missive for the details of grade calculations).

Remember to prove all your (non-elementary and not shown in class) mathematical claims, unless stated otherwise.

Each pair of students should submit only 1 pdf to the corresponding Canvas assignment.

Problem 1

Two short questions:

1. (1 $\checkmark$) Recall that the (relative) Hamming distance between two strings (of the same length) is the fraction of indices on which they differ. Give an algorithm for estimating the Hamming distance d to within additive error ϵ, succeeding with probability at least 2/3, with query complexity $O\left(\frac{d}{\epsilon^2}\right)$. Concretely, if your algorithm is given a query budget of q, the additive error of your estimate should be at most $O\left(\sqrt{\frac{d}{q}}\right)$ with probability at least 2/3.

2. Recall the framework for property testing as follows. There is a class $\mathcal{C}$ of objects (say, graphs), and there is a property $\mathcal{P} \subseteq \mathcal{C}$ that is a subset of these objects that we wish to test for. For example, $\mathcal{P}$ might be the set of connected graphs. Recall that a two-sided ϵ-tester for $\mathcal{P}$ should, with probability at least 2/3

   - accept all objects in $\mathcal{P}$
   - reject objects that are ϵ-far from $\mathcal{P}$ (under a suitable definition of distance between objects)

   Suppose there are properties $\mathcal{P}_1$ and $\mathcal{P}_2$ such that $\mathcal{P}_1 \subseteq \mathcal{P}_2$. Are the following statements true? Prove or disprove:

   (a) (1 $\checkmark$) If $\mathcal{P}_1$ has an ϵ-tester with query complexity $q(\epsilon)$, then $\mathcal{P}_2$ has an ϵ-tester with query complexity $O(q(\epsilon))$.

   (b) (1 $\checkmark$) If $\mathcal{P}_2$ has an ϵ-tester with query complexity $q(\epsilon)$, then $\mathcal{P}_1$ has an ϵ-tester with query complexity $O(q(\epsilon))$.

   (c) (1 $\checkmark$) If $\mathcal{P}_1$ has an ϵ-tester with query complexity $q(\epsilon)$, then $\overline{\mathcal{P}_1}$ has an ϵ-tester with query complexity $O(q(\epsilon))$.

Problem 2

(3 ✓s)

In this problem, we will use what we learnt about the connectedness property to design an efficient *distributed* property testing algorithm in the bounded degree graph setting.

Consider the following setting. We have n machines connected over a communication network that is the complete graph, that is, each machine can directly communicate with any other machine. On top of the communication network, the machines are also connected in a graph structure G with degree upper bounded by some constant d. Each machine is uniquely named (say, with the numbers $[1..n]$), and initially knows only the neighbours it is connected to in G. The goal is to test whether G is connected, versus if it is ϵ-far from connected, using a small number of *communication rounds*. The success condition is as follows:

- If the graph G is connected, then all machines know that G is connected with probability at least $2/3$.
- If the graph G is ϵ-far from connected, then with probability at least $2/3$, at least one machine will know (correctly) that G is disconnected.

The machines in the network have only $O(\text{polylog}(n))$ memory, so they can only each store poly-logarithmically many machine names (each of which take $\log n$ bits). In particular, no machine can hold (even close to) the entire graph G in memory.

In each synchronous round, each machine first computes arbitrary amount of computation as it needs to (a slightly unrealistic assumption but unimportant for this problem), and then it can simultaneously send as many messages as it wishes (to any other machine, including non-neighbours in G, and to at most $\text{polylog}(n)$ many machines, but simplifying you may assume that the machines can broadcast completely). However, the messages sent in a single synchronous round must be committed to at the same time, meaning that the messages sent by one machine cannot depend adaptively on the messages sent by other machines. Such adaptivity will need to happen across multiple synchronous rounds.

Leader election is an important primitive that you will need to use, which you may assume is implementable without proof. Running the leader election protocol for t rounds, over the graph G, gives the following guarantees:

- The machines are separated into partitions where each subset corresponds to machines having the same "leader" (a machine, although the leader may not be in the subset itself). Each of the subsets is guaranteed to have diameter at most $2t$ (where the diameter is defined in terms of G).
- If a connected component (in G) has diameter at most $c \cdot t$ for some universal (small) constant c, then the entire component will have the same leader from the component itself.

Using the structural result from class on ϵ-farness from connectedness, as well as the leader election primitive, design a distributed algorithm in the above setting that tests the connectedness of the graph G using $O\left(\frac{1}{\epsilon d}\right)$ synchronous rounds. You may assume d and ϵ to be "constants" that are independent of n as n grows.

Problem 3

In this problem, consider the special case of the list monotonicity property testing problem (in Hamming distance) mentioned in class, where the list is a bit list containing only 0s and 1s. Concretely, given a bit list of length n, test whether it is monotonic or ϵ-far in Hamming distance from being monotonic, succeeding with probability at least 2/3.

1. (1 ✓) Give an efficient testing algorithm, with time and query complexities depending only on poly$(1/\epsilon)$ and independent of n.
2. (2 ✓s, **Extra credit**) Prove the correctness and efficiency of your algorithm.