Quiz 4 - COMS E6261: Advanced Cryptography

Question 1

Consider the following version of the discrete log search problem.

Given a circuit $C : [N] \times [N] \to [N]$ representing a binary operation on a set of size N, a set element $g \in [N]$ and target element $t \in [N]$, find a number $x \in [N]$ s.t. $g^x = t$ (where exponentiation means repeated application of C).

This problem is known to be in (check all that apply):

- ☐ FP
- ☑ FNP
- ☐ TFNP
- ☐ TFUP (TFNP search problems with unique solutions)

Explanation: This problem is in FNP because given a proposed solution x, there is an efficient algorithm to check it – this is the repeated squaring algorithm we saw in class. It is not known to be in FP (and if the discrete-log assumption in cryptography is true for any efficiently computable group, then this problem is hard / not in FP). The problem is not in TFNP since it is not total – if the input does not correspond to a valid cyclic group with generator g, there may not be any solution.

Question 2

Consider the following version of the discrete log search problem.

Suppose $\mathcal{G}$ is an efficient randomized algorithm that on input 1^n uses some fixed poly(n) number of random bits, and outputs (G, N, g) which is guaranteed to correspond to a cyclic group G with N elements and a generator g. Moreover, assume that from the group description G one can efficiently compute the group operation and check whether a given element is in G.

Given as input the randomness r for $\mathcal{G}$ (which gives rise to the group $\mathcal{G}(1^n; r) = (G, N, g)$) and a target $t \in G$, find $x \in [N]$ such that $g^x = t$.

This problem is known to be in (check all that apply):

- ☐ FP
- ☑ FNP
- ☑ TFNP
- ☑ TFUP (TFNP search problems with unique solutions)

Explanation: The first two items (in FNP, probably not in FP) are the same as in the previous question. Since the output of $\mathcal{G}$ is guaranteed to be a valid cyclic group and generator, this problem must be total, so it is also in TFNP. Moreover, since g is guaranteed to be a generator and $t \in G$ (which can be efficiently checked), there exists a unique solution x, so the problem is in TFUP.

We note that such a $\mathcal{G}$ is part of the usual setup for cryptographic discrete log assumption (one example is $\mathcal{G}$ that finds a prime p together with the factorization of $p-1$, uses this to find a generator g of $\mathbb{Z}_p^*$, and outputs $(p, p-1, g)$; there are other examples for other groups).

The Discrete Log Assumption with respect to $\mathcal{G}$ is that given such an output (G, N, g) and target t, it's hard to find x such that $g^x = t$ in the group.

A stronger cryptographic assumption is that this remains hard even if you're given the internal randomness r of $\mathcal{G}$ (and not just its output); this is also believed to be true in some cases (e.g., as mentioned in class, for the $\mathbb{Z}_p^*$ example). This stronger assumption corresponds to the hardness of the TFUP problem in this question.

Question 3

The rest of the questions concern the journey from NP hardness to TFNP hardness.

A very natural thing to try, is to prove that we can go from standard worst-case NP hardness to worst-case TFNP hardness. That is, to try proving that if P $\neq$ NP, then FP $\neq$ TFNP.

However, there are some known barriers to proving that. Here we will detail one such barrier – we will show that if you can prove it using a deterministic, many-one (aka mapping or Karp) reduction, then you would prove NP = coNP (which we believe is unlikely).

The proof is detailed below, with some steps that need to be filled in by you, and some follow up questions.

Question 3.1

Suppose we could prove the statement "if P $\neq$ NP, then FP $\neq$ TFNP" using a reduction as mentioned above. This means we have a reduction (efficient algorithm) that takes an instance A of

- ◉ SAT (or another NP-complete problem)
- ○ some TFNP problem

Question 3.2

and efficiently transform it to an instance B of

- ○ SAT (or another NP-complete problem)
- ◉ some TFNP problem

such that given any solution to B, the algorithm solves A.

Question 3.3

We now use this to prove that SAT is in co-NP. To show this, we need to show an efficient verifier that (given the right witness) can verify that an input formula is

- ○ in SAT
- ◉ not in SAT

Such a verifier indeed exists: just use the reduction, and since the problem in TFNP is total, it must have a solution. That solution can be used as a witness.

This proves that if we had such a reduction, we would have that an NP-complete problem is in co-NP, and thus NP=co-NP.

Question 3.4

Above we proved that that if P $\neq$ NP implies FP $\neq$ TFNP, then NP = co-NP

- ○ True
- ◉ False

Explanation: The proof above only applies when this implication can be proved via a reduction, and moreover it assumes that the reduction is deterministic. There could be other ways to prove this implication (and it is unknown whether this would imply NP=co-NP).

In the literature, there are some more sophisticated barriers, showing that other types of reductions would also yield other surprising consequences, or not be possible.

Question 4

The [HNY] paper we saw in class addresses the question of going from average-case NP hardness to average-case TFNP hardness.

In fact, their main result goes from average-case NP hardness to average-case TFNP/poly hardness: they show that if we assume NP is hard-on-average, we can construct a hard-on-average total search problem R, where instead of a normal NP-verifier for (instance, solution) pairs, there exists an efficient verifier which takes as input also an "advice" string s_n for each $n \in \mathbb{N}$. (R is hard for all poly-time adversaries whether s_n is provided to R or not).

This result is:

- ◉ Weaker than showing NP hardness-on-average implies a problem in TFNP is hard-on-average.
- ○ Stronger than showing NP hardness-on-average implies a problem in TFNP is hard-on-average.

Question 5

The [HNY] paper shows that, in fact, if one chooses the string s_n uniformly at random, then with prob. $\geq 3/4$, it is a good advice string for that n (the search problem will have a solution for each instance of length n).

True/False: then we can modify the search problem R to have as input (x, s_n) where x is an R instance and s_n is sampled uniformly at random. This gives us a hard-on-average problem in TFNP.

- ○ True
- ◉ False

Explanation: This might work if we had a way of verifying that a string s_n is good, i.e. that it is one of the 3/4 good strings in $\{0,1\}^n$ that makes the problem total for that input length. Since we have no way of verifying this, if we choose one of the bad strings, then there might not be a solution for x. Thus, the problem may not be total.