CS 151 Complexity Theory	**Spring 2023**

Solution Set 7

Posted: June 1 — *Chris Umans*

Obviously, if you have not yet turned in Problem Set 7, you shouldn't consult these solutions.

1. (a) Let $p_i = \Pr_y[f(x+y) - f(y) = i]$. The probability two random voters disagree is $2p_0p_1$. If $p_0 \geq 1/2$ (so the majority is 0), then the probability a random voter disagrees with the majority is $p_1 \leq 2p_0p_1$; similarly if $p_1 \geq 1/2$ (so the majority is 1), then the probability a random voter disagrees with the majority is $p_0 \leq 2p_0p_1$. So we have

$$\Pr_y[f(x+y) - f(y) \neq \tilde{f}(x)] \leq 2 \Pr_{y,z}[f(x+y) - f(y) \neq f(x+z) - f(z)].$$

Now, if both (1) $f(x+y) + f(z) = f(x+y+z)$ and (2) $f(x+z) + f(y) = f(x+y+z)$ hold, then $f(x+y) - f(y) = f(x+z) - f(z)$. So at least one of (1) and (2) must fail to hold for the event on the right-hand-side to hold. Thus by a union bound

$$\begin{aligned} \Pr_{y,z}[f(x+y) - f(y) \neq f(x+z) - f(z)] &\leq \Pr_{y,z}[f(x+y) + f(z) = f(x+y+z)] \\ &\quad + \Pr_{y,z}[f(x+z) + f(y) = f(x+y+z)] \end{aligned}$$

and then by Eq. (7.1), the right-hand-side is bounded by 2δ. It follows that

$$\Pr_y[f(x+y) - f(y) = \tilde{f}(x)] \geq 1 - 4\delta.$$

(b) We know that $\Pr_{x,y}[f(x+y) - f(y) = f(x)] \geq 1 - \delta$ by Eq. (7.1). Now for those x such that $f(x) \neq \tilde{f}(x)$, we have

$$\Pr_y[f(x+y) - f(y) = f(x)] = \Pr_y[f(x+y) - f(y) \neq \tilde{f}(x)] \leq 1/2,$$

by the definition of $\tilde{f}$. Thus if $p = \Pr[f(x) \neq \tilde{f}(x)]$

$$\Pr_{x,y}[f(x+y) - f(y) = f(x)] \leq p/2 + (1-p) = 1 - p/2,$$

from which we conclude $1 - \delta \leq 1 - p/2$, and thus $p \leq 2\delta$.

(c) Fix x, y. Following the hint, we have

$$\begin{aligned} \Pr_{w,z}[f(w) + f(z) = f(w+z)] &\geq 1 - \delta \\ \Pr_{w,z}[f(x+w) + f(y+z) = f(x+y+w+z)] &\geq 1 - \delta \\ \Pr_{w,z}[f(x+w) - f(w) = \tilde{f}(x)] &\geq 1 - 4\delta \\ \Pr_{w,z}[f(y+z) - f(z) = \tilde{f}(y)] &\geq 1 - 4\delta \\ \Pr_{w,z}[f(x+y+w+z) - f(w+z) = \tilde{f}(x+y)] &\geq 1 - 4\delta. \end{aligned}$$

So with all but 14δ probability, *all* of the equations in the above probabilities hold, in which case

$$\tilde{f}(x+y) = f(x+y+w+z) - f(w+z) = f(x+w) + f(y+z) - f(w) - f(z) = \tilde{f}(x) + \tilde{f}(y).$$

Thus

$$\Pr_{w,z}[\tilde{f}(x) + \tilde{f}(y) = \tilde{f}(x+y)] \geq 1 - 14\delta > 0$$

(using the assumption that $\delta > 1/14$). But the event in the probability does not depend on w, z so it must hold (and the probability must be 1). This is true for all x, y.

(d) Completeness is obvious. If f passes the test with probability $1 - \delta$, then by definition

$$\Pr_{x,y}[f(x) + f(y) = f(x+y)] \geq 1 - \delta.$$

We can then say that there exists a *linear* function $\tilde{f}$ satisfying $\Pr_x[f(x) = \tilde{f}(x)] \geq 1 - 14\delta$, because if $\delta \geq 1/14$, this is trivially true, and otherwise by part (b) we get that the function $\tilde{f}$ defined using the majority function agrees with f on all but a $2\delta < 14\delta$ fraction of the x, and by part (c) we get that $\tilde{f}$ is linear.

2. (a) The probability that A satisfies a given ϕ_i is at most

$$(1 - \epsilon)^{\log_2 n} \leq e^{-\epsilon \log_2 n} = n^{-\epsilon \log_2 n / \ln n} = n^{-\epsilon \log_2 e} \leq n^{-\epsilon}/2.$$

Define the indicator random variable X_i to be 1 if A satisfies ϕ_i and zero otherwise. Notice that $\mathrm{E}[X_i] \leq n^{-\epsilon}/2$. Define $X = \sum_i X_i$, and notice that $\mathrm{E}[X] \leq n^{3-\epsilon}/2$ by linearity of expectations. Applying the Chernoff bound, we find that

$$\Pr[X > n^{3-\epsilon}] < e^{-n^{3-\epsilon}/6} \leq e^{-n^2}$$

as desired.

(b) It is clear that if ϕ is a YES instance, then every one of the ϕ_i is simultaneously satisfied by some assignment – namely, the one that satisfies all of the clauses of ϕ.

If ϕ is a NO instance, then taking the union bound over all 2^n possible assignments A, we find that

$$\Pr[\exists A \text{ that satisfies more than } n^{3-\epsilon} \text{ of the } \phi_i] \leq 2^n e^{-n^2} < 1/2,$$

as desired.

(c) We produce a graph with n^3 sets of nodes. Each node in set i corresponds to one of the possible satisfying assignments to ϕ_i. Since ϕ_i consists of $\log_2 n$ clauses with at most 3 variables each, there are at most n^3 nodes in each set, for a total of n^6 nodes in the graph. Now, we connect a node in set i to a node in set j (for $i \neq j$) iff the assignments they represent are consistent.

Now, in the positive case, it is clear that G has a clique of size n^3, consisting of the nodes representing assignment A to each of the ϕ_i.

In the negative case, we observe that a clique in G can have at most 1 node from each set (since there are no edges within the sets), so a clique of size greater than $n^{3-\epsilon}$ must imply an assignment that is simultaneously consistent with more than $n^{3-\epsilon}$ of the ϕ_i, a contradiction.

(d) Note that $N = n^c$ for some constant c. Set $\delta = \epsilon/(c+1)$. Given any language $L \in$ **NP** and an input x we obtain ϕ using the PCP theorem, and then use randomness to construct G from ϕ as described above. We then run the N^δ-approximation algorithm on the instance $(G, k = n^3)$. If it returns a clique of size at least $k/N^\delta > n^{3-\epsilon}$, then we accept; otherwise we reject.

Now, if $x \in L$, then our construction will always produce a graph G with a clique of size n^3, and our approximation algorithm is guaranteed to return a clique of size at least k/N^δ, and we will accept.

If $x \notin L$, then our construction will produce a graph with no clique larger than $n^{3-\epsilon}$ with probability $1/2$ and in this case we will reject (because no clique returned by the approximation algorithm will be large enough).

Thus we have a randomized algorithm that always accepts if $x \in L$, and rejects with probability at least $1/2$ if $x \notin L$. We conclude that $L \in$ **coRP**, and therefore **NP** $\subseteq$ **coRP**. Now, we know from the midterm that **NP** $\subseteq$ **coRP** $\subseteq$ **BPP** implies that **NP** $\subseteq$ **RP**. We conclude that **NP** $\subseteq$ (**RP** $\cap$ **coRP**) $=$ **ZPP** as required.

Alternatively, we could argue directly that **NP** $\subseteq$ **coRP** implies **coNP** $\subseteq$ **RP**, and therefore

$$\mathbf{NP} \subseteq \mathbf{coRP} \subseteq \mathbf{coNP} \subseteq \mathbf{RP} \subseteq \mathbf{NP}$$

and so **NP** $=$ **coRP** $=$ **RP** $=$ **ZPP**.

3. (a) We describe a recursive divide and conquer algorithm. As the base case, if $n = 1$ then it is easy to evaluate $f(0)$ and $f(1)$ with $O(1)$ operations. If $n > 1$, then write $f(x_1, \ldots, x_n) = g(x_2, \ldots, x_n) + x_1 h(x_2, \ldots, x_n)$, and recursively compute g and h at all of $\{0,1\}^{n-1}$. Note that $f(0, x_2, \ldots, x_n) = g(x_2, \ldots, x_n)$ while $f(1, x_2, \ldots, x_n) = g(x_2, \ldots, x_n) + h(x_2, \ldots, x_n)$. So we can obtain all of the required evaluations from the values returned by the recursive calls.

Preparing g and h for the recursive calls requires $O(2^n)$ operations (since we just need to go through the coefficients one by one), and computing the evaluations of f from the returned lists takes $O(2^n)$ operations (we need to copy one list of size 2^{n-1} and then output the element-wise sum of two lists of size 2^{n-1}).

Let $T(n)$ denote the number of operations when there are n variables. Then we have

$$T(n) \leq 2T(n-1) + O(2^n),$$

from which we conclude $T(n) = O(n2^n)$. We know that $f(x) \leq \text{quasipoly}(|C|)$, and we are always summing positive numbers, so the maximum magnitude of any integer in these operations is $\text{quasipoly}(|C|)$, and arithmetic operations on such integers take time $O(\text{poly}(\log |C|))$. The overall running time is $O(\text{quasipoly}(|C|) + 2^n \cdot \text{poly}(n)$ to obtain the representation in the theorem, plus $O(2^n \text{poly}(n, \log |C|))$ for evaluating $f(x)$ at all of $\{0,1\}^n$ plus the time to perform 2^n evaluations of T, each of which takes time $\text{poly}(\log \ell)) = \text{poly}(\log |C|))$.

(b) Plug in each of the $2^{n'}$ possible values, resulting in a new circuit, and let C' be the OR of these $2^{n'}$ circuits, which remains an **ACC**-type circuit, of size $\text{poly}(n) \cdot 2^{n'}$. Clearly C' is satisfiable iff C is. Applying the procedure in the previous part to C' takes time

$O(2^{n-n'}\text{poly}(n) + \text{quasipoly}(\text{poly}(n)\cdot 2^{n'}))$. By choosing $n' = n^\epsilon$ for ϵ a sufficiently small constant we can make $\text{quasipoly}(\text{poly}(n)2^{n'}) < O(2^{\sqrt{n}})$ (say), and the overall running time is thus $O(2^{n-n^\delta})$ for a constant $\delta < \epsilon$.

(c) Consider the language consisting of pairs (C, i) where C is a SUCCINCT 3-SAT instances, and the i-th bit of the lexicographically first satisfying assignment to the 3-SAT formula encoded by C is one. We claim this language is in $\mathbf{E}^{\mathbf{NP}}$. Indeed, in time at most $2^{|C|}\text{poly}(|C|)$, we can extract the 3-SAT formula encoded by C. Then using the NP oracle, we can perform a binary search to find the lexicographically first satisfying assignment, if there is one. Then it is easy to accept or reject based on the i-th bit of this assignment. Since we are assuming $\mathbf{E}^{\mathbf{NP}} \subset \mathbf{ACC}$, there exists an $\mathbf{ACC}$ circuit W_x as described in the problem by hardwiring C_x as part of the input to the $\mathbf{ACC}$ circuit decided this language (for inputs of the appropriate length).

(d) To begin, we perform the SUCCINCT 3-SAT reduction from language L, with input x, to obtain C_x. Set $n = |x|$. So far this takes polynomial time.

Now, we need to argue that D, G, V exist. For this we note that the following are functions in $\mathbf{P}$:

- given a circuit C and an input x, output $C(x)$
- given a circuit C and an input i, output the gate information for gate i of circuit C
- given a circuit C, an input i, and an input x, output the value of gate i when evaluating C on input x.

Since we are assuming $\mathbf{E}^{\mathbf{NP}} \subseteq \mathbf{ACC}$, we certainly have $P \subseteq \mathbf{ACC}$. Thus there are polynomial-size families of $\mathbf{ACC}$ circuits computing each of these functions. Hardwiring C_x as the circuit C in the $\mathbf{ACC}$ circuit of the appropriate size yield the $\mathbf{ACC}$ circuits D, G, and V, respectively. As usual, it is more challenging to actually get our hands on these circuits, and for this we use the ability to guess and verify as suggested in the hint.

We now nondeterministically guess D, G, V, W_x. Given guessed $\mathbf{ACC}$ circuits D, G, V, we note the following:

- in polynomial time, we can verify that G is correct, by running through all of its inputs (there are at most polynomially many) and consulting C_x,
- V is correct iff the following holds for all x and all i: evaluate $G(i)$ to determine the gate type of gate i, and its at most 2 input gates j, k; check that $V(x, i)$, $V(x, j)$ and $V(x, k)$ are consistent (e.g., if the gate type of gate i is OR, and the two input gates are gate j and gate k, then we check that $V(x, i) = V(x, j) \vee V(x, k)$), and
- D is correct iff for all x: $D(x) = V(x, i^*)$ where i^* is the index of the output gate (we can standardize our gate numbering so this is always gate 0, for example).

Observe that after the universal quantification of x, i, the checks in the last two bullets can be expressed as an $\mathbf{ACC}$ circuit with $\ell = |(x, i)| = n + O(\log n)$ inputs, because in both cases we are performing a constant number of evaluations of $\mathbf{ACC}$ circuits and using those values on a computation involving at most $O(\log n)$ bits (which we could even afford to write out as a CNF). Therefore, we can use part (b) to perform these checks in $O(2^{\ell-\ell^\delta})$ time.

If D, G, V pass these checks, then we are left checking whether W_x encodes a satisfying assignment to ϕ_x (the 3-SAT instance succinctly encoded by C_x – and now D as well). Recall that C_x (and D) have at most $m = n + 5\log n$ inputs. Thus there are at most 2^m clauses in ϕ_x, and ϕ_x involves at most 2^m variables. Thus, given a clause number i, it takes $\text{poly}(m) = \text{poly}(n)$ many evaluations of D to extract a description of clause i. This consists of the names of the three variables j_1, j_2, j_3 appearing in the clause, and whether or not they are negated. We can then check whether $W_x(j_1), W_x(j_2), W_x(j_3)$ satisfy the clause. Again, after the universal quantification of the clause number i, this check can be expressed as an **ACC** circuit with m inputs, as we are just plugging a sequence of evaluations of D into W_x, three times, and possibly negating the results before taking their OR. Therefore, we can use part (b) to perform these checks in $O(2^{m-m^\delta})$ time.

Altogether, on input x (an instance of L, an arbitrary language in $\mathbf{NTIME}(2^n)$), we guess $\text{poly}(n)$ bits (to describe D, G, V, W_x), and perform $\text{poly}(n)$ deterministic computation (to produce C_x, to check the correctness of G, and to set up the **ACC** circuits to be used in the two invocations of part (b)), followed by $O(2^{\ell-\ell^\delta}) + O(2^{m-m^\delta})$ steps to invoke part (b) twice. Since $\ell, m \leq n + O(\log n)$, this last quantity plus the various $\text{poly}(n)$ quantities is at most $O(2^{n-n^{\delta'}})$ for some constant $\delta' > 0$. We accept iff D, G, V pass their checks, and W_x indeed encodes a satisfying for ϕ_x, which happens iff $x \in L$.