

## Solution Set 5

Posted: May 14

Chris Umans

1. (a) Suppose we have a language  $L \in \mathbf{BPP}$  with associated probabilistic TM  $M$  that runs in time polynomial time. Given an input  $x$ , we can simulate  $M$  deterministically using an (NW-style) PRG with seed length  $O(\log |x|)$ , output length  $m = |x|^k$ , and “fooling size”  $s = |x|^k$ , for some constant  $k$  related to the running time of  $M$ .

We know from class that if  $E$  contains a function family  $f = \{f_n\}$  is  $2^{\delta n}$ -unapproximable (for some constant  $\delta > 0$ ) then we can build a PRG with seed length  $O(\log n)$ , output length  $m = n^{\delta/3}$  and fooling size  $s = n^{\delta/3}$ . The actual function from this family that is used is  $f_{\log n}$ , whose truth table has size  $n$ . Thus setting  $n = |x|^{3k/\delta}$ , we obtain the PRG necessary to deterministically decide whether  $x \in L$ .

We also know that we can obtain a function  $f : \{0, 1\}^{\log n} \rightarrow \{0, 1\}$  that is  $n^\delta$ -unapproximable from a function  $f' : \{0, 1\}^{\gamma \log n} \rightarrow \{0, 1\}$  that requires size  $n^{\delta'}$  circuits to compute exactly, for some constants  $\gamma < 1$  and  $1 > \delta' > \delta$ .

Thus we can *guess* the truth table of such a function  $f'$ , which has length  $|x|^{(3k\gamma)/\delta} = \text{poly}(|x|)$ , and decide in  $\text{poly}(|x|)$  time whether it has a size  $n^{\delta'}$  circuit (since by assumption  $\text{MTTC} \in \mathbf{P}$ ). Shannon’s counting argument says that with extremely high probability (certainly greater than  $1/2$ ) the guessed truth table will *not* be computable by a size  $n^{\delta'}$  circuit, and then we have the function required to build a PRG and determine if  $x \in L$ . If the guessed function has a small circuit, we output “fail”. We have described a **ZPP** algorithm for deciding  $L$ , and we know  $\mathbf{ZPP} \subseteq \mathbf{BPP}$ , so we have  $\mathbf{ZPP} = \mathbf{BPP}$ .

- (b) We use the same ideas as in part (a). We have an nondeterministic oracle machine  $M$  that decides language  $L \in \mathbf{NP}^{\mathbf{BPP}}$ . Let  $k$  be such that all of the **BPP** queries  $M$  makes on inputs of length  $n$  require no more than  $n^k$  random bits.

On input  $x$ , we guess the truth table of a function  $f$  of length  $\text{poly}(|x|)$ . As noted,  $\text{MTTC} \in \mathbf{NP}$ , and so with a single query to the **NP** oracle we can determine if  $f$  has the desired hardness. If it does not we output “fail”; by Shannon’s argument we know that this happens with very small probability (certainly less than  $1/2$ ). Now we can build a PRG that permits us to decide each of the **BPP** queries  $M$  makes on input  $x$  in deterministic polynomial time. Finally, we use the **NP** oracle one more time to simulate  $M$  replacing each of  $M$ ’s oracle queries with deterministic simulation using the PRG. This shows that  $L \in \mathbf{ZPP}^{\mathbf{NP}}$  as desired.

2. We are given a Boolean circuit  $C$  on  $n$  variables  $x_1, x_2, \dots, x_n$  with  $m$   $\wedge, \vee$  and  $\neg$  gates. Our 3-CNF formula will have  $m$  auxiliary variables  $z_1, z_2, \dots, z_m$  in addition to the  $x$  variables, and we associate each  $z$  variable with one of the  $m$  gates. We want to enforce constraints so that any satisfying assignment to all of the variables will have the  $z$  variables taking on the

value that the associated gates would output given the assignment to the  $x$  variables. We do this as follows:

- for a  $\neg$  gate associated with  $z_i$ , and with input  $w$  (which may be a  $z$  variable or an  $x$  variable), we enforce  $\neg w \Leftrightarrow z_i$  by including the clauses  $(w \vee z_i)$  and  $(\neg w \vee \neg z_i)$ .
- for an  $\wedge$  gate associated with  $z_i$ , and with inputs  $w$  and  $y$  (each of which may be a  $z$  variable or an  $x$  variable), we enforce  $(w \wedge y) \Leftrightarrow z_i$  by including the clauses  $(\neg w \vee \neg y \vee z_i)$ ,  $(\neg z_i \vee w)$  and  $(\neg z_i \vee y)$ .
- for an  $\vee$  gate associated with  $z_i$ , and with inputs  $w$  and  $y$  (each of which may be a  $z$  variable or an  $x$  variable), we enforce  $(w \vee y) \Leftrightarrow z_i$  by including the clauses  $(\neg w \vee z_i)$ ,  $(\neg y \vee z_i)$  and  $(\neg z_i \vee w \vee y)$ .

Assume that  $z_m$  is the variable associated with the output gate. By construction our 3-CNF so far has the property that any assignment that satisfies the above clauses must assign to  $z_m$  the value that  $C(x_1, x_2, \dots, x_n)$  takes given the assignment to the  $x$  variables. We add a final clause  $(z_m)$ . Now an assignment satisfies the formula if and only if the assignment sets the  $x$  variables in such a way that  $C(x_1, x_2, \dots, x_n) = 1$ . It is also easy to see that any assignment to the  $x$  variables for which  $C(x_1, x_2, \dots, x_n) = 1$  can be extended to an assignment to the  $x$  and  $z$  variables that satisfies all of the above clauses, by simply setting each  $z_i$  to the value the  $i$ -th gate is outputting in circuit  $C$ . Thus  $C$  is satisfiable if and only if the just-constructed 3-CNF formula is. If we call the 3-CNF formula  $\phi$ , then we have, as desired:

$$\exists z_1, z_2, \dots, z_m \quad \phi(x_1, x_2, \dots, x_n, z_1, z_2, \dots, z_m) = 1 \Leftrightarrow C(x_1, x_2, \dots, x_n) = 1.$$

For the second part, we first take  $C$  and add a  $\neg$  gate to its output; call this circuit  $C'$ . Now applying the above transformation to  $C'$  gives a 3-CNF formula  $\phi'$  with the property that:

$$\exists z_1, z_2, \dots, z_m \quad \phi'(x_1, x_2, \dots, x_n, z_1, z_2, \dots, z_m) = 1 \Leftrightarrow C'(x_1, x_2, \dots, x_n) = 1.$$

Equivalently,

$$\forall z_1, z_2, \dots, z_m \quad \phi'(x_1, x_2, \dots, x_n, z_1, z_2, \dots, z_m) = 0 \Leftrightarrow C'(x_1, x_2, \dots, x_n) = 0.$$

Let us define  $\phi$  to be  $\neg\phi'$ , and note that (if we distribute the  $\neg$ )  $\phi$  is a 3-DNF formula. We have:

$$\forall z_1, z_2, \dots, z_m \quad \phi(x_1, x_2, \dots, x_n, z_1, z_2, \dots, z_m) = 1 \Leftrightarrow C'(x_1, x_2, \dots, x_n) = 0.$$

Finally, by the definition of  $C'$  we have:

$$\forall z_1, z_2, \dots, z_m \quad \phi(x_1, x_2, \dots, x_n, z_1, z_2, \dots, z_m) = 1 \Leftrightarrow C(x_1, x_2, \dots, x_n) = 1$$

as desired.

3. (a) For each  $k = 1, 2, 3, \dots, n - 2$  we perform the following procedure:

- Pick a random  $h \in \mathcal{H}_{n,k}$ , and ask the **NP** oracle query: is there some  $y \in \{0, 1\}^k$  for which  $|\{x : x \in S, h(x) = y\}| > n^4$ ?

Now, when  $k$  is such that  $|S|/2^k > n^4$ , we know by the pigeonhole principle that the answer to this query will be “yes.” Now consider the *last*  $k$  for which this holds; i.e., the integer  $k$  for which

$$2^k < \frac{|S|}{n^4} \leq 2^{k+1}.$$

We claim that when we ask the query for  $k+2$ , with very high probability the answer will be “no.” Since  $2^{k+2} < 4|S|/n^4$ , the Lemma applies, and tells us that for every  $y \in \{0, 1\}^{k+2}$ ,

$$\Pr_{h \in \mathcal{H}_{n,k+2}} \left[ |\{x : x \in S \wedge h(x) = y\}| > 2 \cdot \frac{|S|}{2^{k+2}} \right] \leq 2^{-2n}.$$

Since  $n^4 \geq |S|/2^{k+1} = 2|S|/2^{k+2}$ , this implies:

$$\Pr_{h \in \mathcal{H}_{n,k+2}} \left[ |\{x : x \in S \wedge h(x) = y\}| > n^4 \right] \leq 2^{-2n}.$$

By a union bound,

$$\Pr_{h \in \mathcal{H}_{n,k+2}} \left[ \exists y \in \{0, 1\}^{k+2} \text{ for which } |\{x : x \in S \wedge h(x) = y\}| > n^4 \right] \leq 2^n \cdot 2^{-2n} = 2^{-n}.$$

So, if we set  $k^*$  to be the first value for which we receive a “no” answer from the oracle, we know with very high probability that  $k^*$  will be either  $k+1$  or  $k+2$  (because by the time we get to  $k+2$ , we have shown that with high probability the answer will be “no”). Therefore we can satisfy the problem demands by outputting  $k^* - 2$ .

(b) Using the procedure from the first part, and  $k^*$  as defined above, we obtain a function  $h : \{0, 1\}^n \rightarrow \{0, 1\}^{k^*}$  for which:

- $\forall y \in \{0, 1\}^{k^*}$ , the set  $\{x : x \in S \wedge h(x) = y\}$  has size at most  $n^4$  (because the **NP**-oracle query answer was “no” for  $k^*$ ).
- with probability at least  $7/8$ ,  $2^{k^*-2} < |S|/n^4$ , as required by (a).

We perform the following procedure: pick a random  $y \in \{0, 1\}^{k^*}$  and use the **NP**-oracle to enumerate the set  $T_y = \{x : x \in S \wedge h(x) = y\}$ . Choose a random number  $i$  between 1 and  $n^4$  and output the  $i$ -th element of  $T_y$ , or “fail” if there is no  $i$ -th element. To connect this with the hint: we are thinking of each  $y$  as specifying a page of a notebook which has  $n^4$  lines per page, and the elements  $T_y$  written on the first  $|T_y|$  lines of page  $y$ .

The probability we output any given  $x \in S$  is exactly  $2^{-k^*} \cdot (1/n^4)$  – so conditioned on not failing, the output is exactly uniformly distributed on  $S$ . The probability that we output “fail” is exactly  $1 - |S|/(2^{k^*}n^4)$ . With probability at most  $1/8$  the second item above fails to hold (and then we have no handle on the failure probability). Otherwise, we know that  $|S| > 2^{k^*-2}n^4$ , so the failure probability is at most

$$1 - \frac{|S|}{2^{k^*}n^4} < 1 - \frac{2^{k^*-2}n^4}{2^{k^*}n^4} = \frac{3}{4}.$$

Thus the overall failure probability is at most  $3/4 + 1/8 = 7/8$ , as required.