Problem Set 2, Petar Maymounkov
6.841J – Advanced Complexity with Prof. Madhu Sudan
Collaborators: Oren Weimann, Alex Andoni, Krzysztof Onak

1. Circuit-size Hierarchy:

Let F_n be the set of binary functions on n variables, and $C_n(t(n))$ be the set of binary functions on n variables computable by circuits of size at most $t(n)$.

We use that every $f \in F_n$ can be computed by some $c_f \in C_n(O(2^n/n))$, and therefore $F_n \subseteq C_n(O(2^n/n))$. Furthermore, we use that $|C_n(t(n))| = 2^{O(t(n)\ln t(n))}$.

Set $x(n) = \ln\big(f(n)\ln f(n)\big)$. Note that $x(n) \leq n$. Set $A = C_n\big(o(f(n))\big)$, $B = F_x$ and $C = C_n\big(O(2^x/x)\big)$. We will show that $A \subsetneq B \subseteq C$ which would imply that there exists $g \in F_x \subseteq F_n$ computable by a $\big(f(n)\ln f(n)\big)$-size circuit, and not computable by a $o\big(f(n)\big)$-size circuit. $B \subseteq C$ follows immediately from the facts above. For $A \subsetneq B$ we use that $|A| \leq 2^{o(f(n)\ln f(n))} \ll f(n)^{f(n)} = |B|$.

2. Poly-size Circuits:

If NP $\subseteq$ P/poly then Ladner's theorem provides the desired language. Otherwise, NP $\not\subset$ P/poly and hence no NP-hard language is in P/poly. On the other hand, the unary halting problem (in P/1) is not in NP. And we are done.

TIME$(2^{n^{\log n}}) \not\subset$ P/poly: We would like to build a machine M such that for all input lengths $n \in \mathbb{N}$, and all circuits $C \in$ NC of size at most $g = n^{\sqrt{\log n}}$ (super-polynomial), there exists $x \in \{0,1\}^n$ with $M(x) \neq C(x)$. Since $g(n)$ is super-polynomial, eventually M will differ from all polynomial-size circuits.

Let $C = c_1, \ldots, c_m$ be an enumeration of all circuits on n inputs of size at most $g = n^{\sqrt{\log n}}$. By counting, $m \leq 3^g(g+n)^{2g} = 2^{O(g\log g)}$. Let $\alpha_1, \ldots, \alpha_{2^n}$ be all possible values of the input. $M(x)$ is computed as follows:

1. Set $i \leftarrow 1$ and $R \leftarrow C$
2. While $R \neq \varnothing$, repeat:
   i. $M(\alpha_i) \leftarrow \neg\mathrm{Maj}(R)$
   ii. $R \leftarrow \{c \in C \mid c(\alpha_i) \neq M(\alpha_i)\}$
   iii. $i \leftarrow i + 1$
3. For $j \geq i$ set $M(\alpha_j) \leftarrow 0$

We are thus left to show that M runs in time $2^{n^{\log n}}$. It is easy to see that M simulates at most $2m$ circuits, each requiring $n^{\sqrt{\log n}}$ steps, for a total:

$$2^{O\left(n^{\sqrt{\log n}}\cdot\log^{3/2} n\right)} \ll 2^{O(n^{\log n})}$$

3. CNF, DNF, and Branching Programs:

Let k-DNF formula Φ_{DNF} and an l-CNF formula Φ_{CNF} be given for a boolean function $f : \{0,1\}^n \to \{0,1\}$. It then immediately follows that the following hold:

$$\bigwedge_{C \in \Phi_{\text{CNF}}} \bigwedge_{T \in \Phi_{\text{DNF}}} \bigvee_{l \in T} C \ni l \tag{†}$$

$$\bigwedge_{T \in \Phi_{\text{DNF}}} \bigwedge_{C \in \Phi_{\text{CNF}}} \bigvee_{l \in C} T \ni l \tag{‡}$$

Where T is a term, C is a clause and l is a literal.

Let $A_{k,l}$ be the set of binary functions on n variables that have a k-DNF and an l-CNF formulas. For a fixed function $\Phi \in A_{k,l}$ we describe a depth-k branching program (BP) that either evaluates Φ on an input $x \in \{0,1\}^n$ or makes a recursive call to a BP evaluator for $\Phi' \in A_{k,l-1}$ on $x' \in \{0,1\}^{n-k}$, where $x' \subset x$.

Let $T \in \Phi_{\text{DNF}}$ be a DNF term and assume $T = x_1 \dots x_k$. If T is true in x then halt and output "1". Otherwise, according to (†) we can remove at least one literal (in T) from each clause in Φ_{CNF} thus obtaining Φ', which we evaluate on $x_{k+1}, \dots, x_n$ recursively.

Similarly, for $\Phi \in A_{k,l}$ we have a depth-l BP that either evaluates Φ on x, or makes a recursive call to a BP evaluator for $\Phi' \in A_{k-1,l}$ on $x' \in \{0,1\}^{n-l}$, where $x' \subset x$.

Using these two constructions, it is straightforward that $f(k,l) \le kl + \min\{k,l\}$. (You can also do a little better and get kl.)

4. Finding a satisfying assignment when there are many:

(This solution is adapted from Hirsch'98.) For a formula f on n variables $X = \{x_1, \dots, x_n\}$ we let $f[l_1, \dots, l_k]$, where $L = \{l_1, \dots, l_k\}$ are literals in X, be the formula obtained by restricting f's inputs correspondingly.

For a k-CNF f consider the following algorithm:

1. $i \leftarrow 0$, $\Phi_0 = \{f\}$ and $\Phi_1 = \cdots = \Phi_n = \varnothing$
2. For all $g \in \Phi_i$, do:
   i. Let $(l_1 \vee \cdots \vee l_r)$ be the shortest clause in g
   ii. Consider the restrictions $g[l_1], g[\overline{l_1}, l_2], \dots, g[\overline{l_1}, \dots, \overline{l_{r-1}}, l_r]$. If any one of them is $\equiv 1$ then halt the algorithm and output a satisfying assignment for f. Otherwise, set $\Phi_{i+1} \leftarrow \Phi_{i+1} \cup \{g[l_1], g[\overline{l_1}, l_2], \dots, g[\overline{l_1}, \dots, \overline{l_{r-1}}, l_r]\}$
3. $i \leftarrow i+1$. If $i > d$, where d is a threshold to be specified later, halt the algorithm and output "f has less than $\epsilon 2^n$ satisfying assignments"
4. Go to step 2

Let T be the abstract tree induced by this algorithm, where each node v corresponds to a restriction $v = f[\dots]$ and a node u is a parent of v iff v is a restriction of u created in step 2.ii. of the algorithm. Let T be a subtree and v be a node in it. We define the *floor* of v with respect to T, denoted $\varphi_T(v)$, to be the number of variables in T's root that are restricted in v. Furthermore, we refine our notation by $\Phi_i^T := \{v \in T \mid \varphi_T(v) = i\}$, and thus $\Phi_i = \Phi_i^R$ where R is the whole tree.

Let's make an argument for soundness first. Assume the algorithm has reached to the point when Φ_i is complete, but it hasn't been processed yet (step 2). This implies that there are no satisfying partial assignments of at most i variables and furthermore any potential satisfying assignment must also be satisfying for some $v \in \Phi_i \cup \dots \cup \Phi_{i+k-1}$. Every $v \in \Phi_{i+j}$, where $0 \le j < k$, can have at most 2^{n-i-j} satisfying assignments. Therefore, at this point of the execution we have certified that f has at most $M_i = \sum_{j=0}^{k-1} |\Phi_{i+j}| \cdot 2^{n-i-j}$ satisfying assignments in total. The threshold d is chosen so that the algorithm stops as soon as $M_d/2^n < \epsilon$. This completes soundness. We now analyze the running time.

We begin by deriving that $|\Phi_i| \le \lambda_k^i$ where λ_k is the unique positive solution of $h_k(x) = 1 - x^{-1} - \dots - x^{-k}$. Induct on the size of T. In the base, $|T| = 1$ and we check that $\Phi_0 = 1 \le \lambda_k^0 = 1$ and $\Phi_i = 0 < \lambda_k^i = 1$ for $i > 0$. For the step, we let R be the root of the tree and $T_1, \dots, T_l$, where $0 \le l \le k$, be the subtrees of its children:

$$|\Phi_i^R| = \sum_{j=1}^{l} |\Phi_{i-j}^{T_j}| \le \sum_{j=1}^{l} \lambda_k^{i-j} = \lambda_k^i \sum_{j=1}^{l} \lambda_k^{-j} \le \lambda_k^i \sum_{j=1}^{k} \lambda_k^{-j} = \lambda_k^i \cdot \big(1 - h_k(\lambda_k)\big) \le \lambda_k^i$$

Using this bound we can now derive $d \ge \frac{\log 2/\epsilon}{\log 2/\lambda_k}$ using $M_d/2^n < \epsilon$ and:

$$\sum_{j=0}^{k-1} |\Phi_{d+j}| \cdot 2^{n-d-j} \le \sum_{j=0}^{k-1} \lambda_k^d \cdot 2^{n-d-j} \le 2 \cdot \lambda_k^d \cdot 2^{n-d}$$

We have thus far shown that if a f has "many" satisfying assignments, then it has a partial satisfying assignment on $\frac{\log 2/\epsilon}{\log 2/\lambda_k} + k - 1$ variables.

Finally, we need to show that the size of the tree is small:

$$|T| = \sum_{i=0}^{i-1} |\Phi_i| + \sum_{j=0}^{k-1} |\Phi_{i+j}| \le \sum_{i=0}^{i-1} \lambda_k^i + k \cdot \lambda_k^i = \dots = O\left(k(2/\epsilon)^{\left(\log_{\lambda_k} 2 - 1\right)^{-1}}\right)$$

The algorithm spends L steps at each tree node, where L is the size of f. This concludes the proof that the algorithm runs in polynomial time.

5. Majority:

As seen from Smolensky's proof that $\oplus_2 \notin \mathrm{AC}^0$, it is the case that $\oplus_3 \notin \mathrm{AC}^0_{\oplus_2}$ (where $\mathrm{AC}_{\oplus_2}$ stands for AC with parity gates). Therefore, it would be sufficient to show that $\oplus_3 \le \mathrm{Maj}$ using a constant depth, polynomial size reduction.

For $x \in \{0,1\}^n$, let $\ge_{1,t}(x)$ be a circuit that determines if $\#_1(x) \ge t$, where $\#_1(x)$ is the number of 1's in x. To implement $\mathrm{GE}_{1,t}(x)$, assume w.l.o.g. $t \le n/2$ and verify that $\mathrm{GE}_{1,t}(x) := \mathrm{Maj}(1^{n-2t} \cdot x)$

works. We can now also implement $\mathrm{EQ}_{1,t}(x)$ which decides whether x has exactly t entries 1, as $\mathrm{EQ}_{1,t}(x) := \mathrm{GE}_{1,t}(x) \wedge \mathrm{GE}_{0,|x|-t}(x)$.

Then, set $k := \lceil n/3 \rceil$ and define:

$$\oplus_3(x) := \begin{cases} 0, & \text{if } \bigvee_{i=0}^{k} \mathrm{EQ}_{1,3k}(x) \\ 1, & \text{if } \bigvee_{i=0}^{k} \mathrm{EQ}_{1,3k+1}(x) \\ 2, & \text{otherwise} \end{cases}$$

This completes the reduction $\oplus_3 \leq \mathrm{Maj}$ (and the problem).

6. A Lower Bound via Communication Complexity:

We assume for contradiction that there exists a 1-tape Turing machine M that solves the PALINDROME language in $o(n^2)$ time. Using M we build a communication protocol for EQ which has worst-case complexity $o(n)$, leading to a contradiction.

Let Alice be given $x \in \{0,1\}^n$ and Bob be given $y \in \{0,1\}^n$. (Making sure that $|x| = |y|$ is trivial using two $\log n$-bit rounds.)

Alice runs M on input $w_x = x \cdot 0^n \cdot x^R$ and recognizes a location $i_x \in [n+1, 2n]$ on the tape such that the number of times M's pointer passes through i_x is $o(n)$. Let's see why such an i always exists. Let $c_i(w)$ be the number of times M's pointer passes through i during computation on w. Since M runs in $o(n^2)$, we have that $\sum_{n<i\leq 2n} c_i(w) = o(n^2)$, and thus by averaging there is an i_x for which $c_{i_x}(w) = o(n)$. Bob performs a similar computation with $w_y = y \cdot 0^n \cdot y^R$.

Next, Alice and Bob exchange the indices i_x and i_y in two rounds of $\log n$-bits each. If the indices differ, then $x \neq y$ and the protocol is over. Otherwise:

Alice and Bob simulate an imaginary run of M on tape $x \cdot 0^n \cdot y^R$, such that at any point either Alice or Bob is simulating and they alternate whenever M's cursor passes through the agreed upon location i. Control is transferred by sending M's state in $O(1)$-bits.

If $x = y$ the simulated computation will go exactly as Alice and Bob expect, in $o(n)$ rounds and bits and they will accept. If $x \neq y$, it must be the case that Alice or Bob halts M pre-maturely, or the state of M at hand-off is not what is expected. We have thus obtained a protocol for EQ which requires $o(n)$-bits communication complexity in the worst case – a contradiction.