# Problem Set 2, Answers

Kevin Lacker

May 20, 2020

**Problem 1.** The idea is to use MAJ circuits to calculate PARITY, and thus use our lower bound on PARITY circuits to provide a lower bound on MAJ circuits.

First, we can create a circuit for the function that counts whether precisely half of the bits of the input are 1. I.e., $HALF(x) = 1$ iff $|x| = n/2$, with two MAJ circuits:

$$HALF(x) = MAJ(x) \wedge MAJ(\neg x) \tag{1}$$

Here $\neg x$ represents the bitwise negation of $x$. Our circuit for for HALF uses two MAJ subcircuits and one more depth.

Now that we have HALF, we can construct a circuit to count whether precisely $k$ bits of the input are 1, $EXACT_k$, by padding the input with ones or zeros and using a single HALF circuit on at most twice the input size.

We can then take one $EXACT_K$ for each odd number less than or equal to $n$, and take their conjunction to create a PARITY circuit of depth $d + 2$. This PARITY circuit of depth $d+2$ is built of $O(n)$ depth-$d$ MAJ circuits, plus $O(n)$ extra gates, where each MAJ circuit has at most $2n$ inputs.

Let $H_{MAJ}(n, d)$ denote the minimum size of a circuit of depth $d$ calculating MAJ on a size-$n$ input, and $H_{PARITY}$ the same for PARITY. This construction demonstrates that

$$O(n) \cdot H_{MAJ}(n, d) \geq H_{PARITY}(n/2, d + 2) \tag{2}$$

But we know that

$$H_{PARITY}(n, d) \geq exp(\Omega(n^{2^{-d}})) \tag{3}$$

Substituting in and simplifying we get

$$H_{MAJ}(n, d) \geq exp(\Omega(n^{2^{-d-O(1)}})) \tag{4}$$

**Problem 2.** Let's say we have an estimate $v$ of $\Sigma_x f(x)$ and we want to improve that estimate. We define:

$$g(x) = \begin{cases} f(0) - v & \text{for } x = 0 \\ f(x) & \text{otherwise} \end{cases} \tag{5}$$

We can then use our estimation oracle to get an estimation for $\Sigma_x g(x) = \Sigma_x f(x) - E$. Basically, we are estimating the error in our original estimate. When we add this estimate to $E$, we are improving our estimate to $\Sigma_x f(x)$. If our estimation oracle returns a value within a factor of $1 \pm \epsilon$ of the correct value, then the size of our error shrinks by a factor of $\epsilon$ every iteration.

Recursing until our error is below 1 thus takes time that is linear in the size of the output value, so we can use this to get an exact answer to $\Sigma_x f(x)$ in polynomial time, assuming the answer has polynomial length.

We can use this to exactly solve problems in $\sharp SAT$. Let $x$ represent a possible solution to a $\sharp SAT$ problem, and $f(x) = 1$ when $x$ is a solution, 0 otherwise. Since $f(x)$ is a constant, the sum has polynomial length, and our algorithm takes polynomial time.

$\sharp SAT$ is $\sharp P$-complete so this solves any $\sharp P$ problem in polynomial time.