- Start as early as possible, and contact the instructor if you get stuck.
- See the course outline for details about the course's grading policy and rules on collaboration.
- Submit your completed solutions to **Crowdmark**.

1. **Self-Reducibility** [**8=4+4 marks**]
   Suppose you are given ϕ, a 3-CNF on n variables $x_1, \ldots, x_n$.
   (a) If you have an oracle that solves 3SAT, show how to construct a satisfying assignment for the variables in polynomial time.

Solution

Let ϕ be the formula. First, run the oracle on ϕ itself. If there is no solution, then report "no solution" and halt. Otherwise, there is either a solution with x_n set to 0 or with x_n set to 1. Notice that if we substitute a literal in a clause with a constant value, it simplifies the formula. Substituting a 1 satisfies the clause,

$$(x \vee y \vee z)|_{x=1} = 1 \vee y \vee z = 1,$$

which allows us to remove it entirely. Substituting a 0 removes the literal, which simplifies the clause:

$$(\overline{x} \vee y \vee z)|_{x=0} = 0 \vee y \vee z = y \vee z.$$

Either way, we can clearly substitute $x_n = 0$ into ϕ, generate the simplified 3-CNF[a], and the substitution is clearly polynomial time.
If $\phi|_{x_n=0}$ has a satisfying assignment, then recurse to find the rest of the assignment. If not, then $\phi|_{x_n=1}$ must have a satisfying assignment instead, so we construct it (again in polynomial time) and recursively find the rest of the assignment. Since we do polynomial work and one oracle call (excluding the very first one) per recursive call, we get polynomial time overall.

[a] Per a Piazza post, we allow formulas with ≤ 3 variables per clause. Exercise: show that you can do this problem even when exactly three variables are used

   (b) If you have an oracle that decides whether there are more satisfying or unsatisfying assignments (ties go to satisfying assignments) to a SAT problem, then show how to use this to count how many satisfying assignments ϕ has in polynomial time.

Solution

The oracle tells us when the number of satisfying assignments exceed a threshold (half). If only we could *control* the threshold, we could easily binary search for the answer.
First, observe that for any k and $0 \leq t < 2^k$, there is a formula $\psi_t(x_1, \ldots, x_k)$ which accepts if $x_k \cdots x_1$, read as a binary number, is $\leq t$. We construct

this recursively as follows.

$$\psi_t(x_1, \ldots, x_k) = \begin{cases} 1 & \text{if } k = 0 \\ \psi_t(x_1, \ldots, x_{k-1}) & \text{if } t < 2^{k-1}, \\ \overline{x_k} \vee \psi_{t-2^{k-1}}(x_1, \ldots, x_{k-1}) & \text{if } t \geq 2^{k-1}. \end{cases}$$

Now suppose we are given a formula $\phi(x_1, \ldots, x_n)$ to count the number of satisfying assignments, and the answer (unknown to us) is s. For any $0 \leq t \leq 2^n$, we can construct the corresponding $\psi_t(x_1, \ldots, x_n)$ and know that it has t satisfying assignments. Then we introduce a variable u and claim the formula

$$(u \wedge \phi(x_1, \ldots, x_n)) \vee (\overline{u} \wedge \psi_t(x_1, \ldots, x_n))$$

has exactly $s + t$ satisfying assignments out of a possible 2^{n+1}; either $u = 1$ and the formula is satisfied if and only if $\psi_t(x_1, \ldots, x_n)$ is (t possible assignments), or $u = 0$ and the formula is satisfied if and only if $\phi(x_1, \ldots, x_n)$. We can compare whether this is $\leq 2^n$ (half of all), and therefore we can compare whether $s < 2^n - t$ for all $0 \leq t < 2^n$. With this, we can binary search for s in $\mathcal{O}(\log 2^n) = \mathcal{O}(n)$ steps with 1 oracle call and $\mathcal{O}(\mathsf{poly}(n))$ additional work per step.

2. **Balancing formulas** [**4 marks**]
We've seen in class that $x_1 \wedge x_2 \wedge \cdots \wedge x_n$ and $x_1 \oplus x_2 \oplus \cdots \oplus x_n$ have Boolean formulas of depth $\mathcal{O}(\log n)$. Surprisingly, *any* polynomial-size Boolean formula ϕ has an equivalent polynomial-size Boolean formula *with* $\mathcal{O}(\log n)$ *depth*.
The proof is by divide and conquer. For sufficiently large binary trees, there exists a node u such that at most $\frac{2n}{3}$ elements are in the subtree rooted at u, and at most $\frac{2n}{3}$ elements are *outside* the subtree—an approximately equal division. For our formula ϕ, this means we can divide it into the subtree, represented by a formula $h(x_1, \ldots, x_n)$ and the context, represented by a formula $g(x_1, \ldots, x_n, u)$ which uses variable u exactly once. Thus,

$$\phi(x_1, \ldots, x_n) = g(x_1, \ldots, x_n, h(x_1, \ldots, x_n)).$$

To finish the inductive argument, we need to **fill in the following step**:
Given formulas G and H equivalent[1] to g and h, construct a formula F equivalent to ϕ such that

$$L(F) \leq k \cdot (L(G) + L(H)) + \mathcal{O}(1),$$
$$\text{depth}(F) \leq \max\{\text{depth}(G), \text{depth}(H)\} + \mathcal{O}(1).$$

Explain why F is equivalent to ϕ.

[1] Here "equivalent" means they compute the same Boolean function.

Solution

The problem is the depth, and the fact that we need to know $h(x_1, \ldots, x_n)$ to feed it into $g(x_1, \ldots, x_n, u)$ as the value of u. Except there are only two possible outcomes for $h(x_1, \ldots, x_n)$, 0 and 1, so instead of waiting, we can compute both possibilities. One way to express ϕ is

$$g(x_1, \ldots, x_n, h(x_1, \ldots, x_n)) = (g(x_1, \ldots, x_n, 1) \wedge h(x_1, \ldots, x_n)) \\ \vee (g(x_1, \ldots, x_n, 0) \wedge \neg h(x_1, \ldots, x_n))$$

In other words, our formula will be $F := (G|_{u=1} \wedge H) \vee (G|_{u=0} \wedge \neg H)$. If $h(x_1, \ldots, x_n)$ evaluates to true then the formula easily simplifies to $G|_{u=1}$, and thus evaluates to

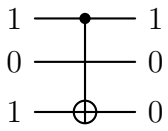
$$g(x_1, \ldots, x_n, 1) = g(x_1, \ldots, x_n, h(x_1, \ldots, x_n)) = \phi(x_1, \ldots, x_n).$$

Similarly, if $h(x_1, \ldots, x_n) = 0$ then the formula simplifies to $G|_{u=0}$ and thus to $\phi(x_1, \ldots, x_n)$.

It is not hard to see that $L(F) \leq 2(L(G) + L(H)) + \mathcal{O}(1)$, since H and G appear twice, substitution can only decrease the size of a formula (since parts may simplify), and there is a constant amount of "glue" to combine the pieces. Similarly, the depth is dominated by the depth of G or H (whichever is deeper) plus two extra levels ($\vee$ and $\wedge$). This meets the requirements for the problem, and leads to a polynomial-size Boolean formula of $\mathcal{O}(\log n)$ depth.

3. **Computation in just three bits [8=1+2+5 marks]**

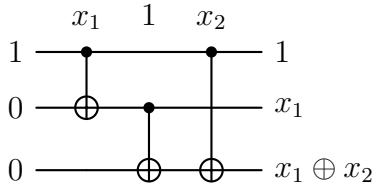
We consider a non-uniform space-limited model of computation on n-bit strings $x_1 x_2 \cdots x_n \in \{0,1\}^*$. In this model, we have three single-bit registers and a *gate* which adds bits—pick one register and add it into another one. In quantum computing, we draw such an operation like this:



The horizontal lines represent the three registers, and the gate is the vertical line. The $\oplus$ of the gate is on the *target* register (the one that is added to) and the $\bullet$ is on the *control* register.

A *program* in this model is a sequence of gates labeled by an input bit x_i, the negation of an input bit, $\overline{x_i}$, or the constant 1. We *run* a program on an input $x_1 \cdots x_n$ by discarding the gates not labelled 1 (i.e., drop gates labelled x_i when $x_i = 0$ or labelled $\overline{x_i}$ if $x_i = 1$), then apply the gates to sequentially to bits 100 and accept if the last bit

ends as a 1. For example, the program

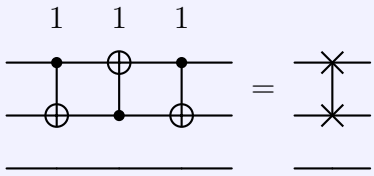


computes $x_1 \oplus x_2$ in an unnecessarily convoluted way.

(a) Give a program that swaps register 1 with register 2.

Solution

Simple:

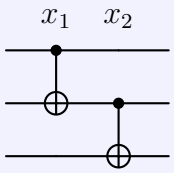


The standard notation for a swap is a line between a pair of X's as shown above. Historically, this trick was also used to swap registers, although it is not such a good idea on modern machines.

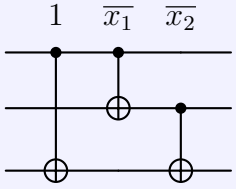
(b) Give a program that computes $x_1 \wedge x_2$, and a program that computes $x_1 \vee x_2$.

Solution

The following circuit computes $x_1 \wedge x_2$.



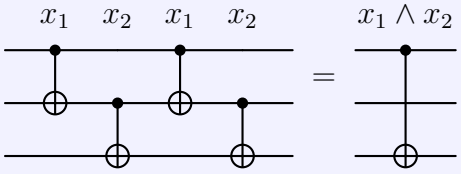
After the first gate, the bit x_1 is on register 2. From there, the second gate adds $x_1 \wedge x_2$ to register 3, since the gate only activates when $x_2 = 1$, and then changes register 3 only if $x_1 = 1$.
We can get OR by De Morgan's law: $x_1 \vee x_2 = \overline{\overline{x_1} \wedge \overline{x_2}}$. We achieve the outermost NOT by having an initial (or final) CNOT from 1 to 3. Then we have the AND gate where the gates are labeled by the negated variables.



(c) Prove that any polynomial-size formula F (where AND and OR have fan-in 2) can be computed by a polynomial length program.
Hint: You may apply the result claimed in the previous problem. I.e., you can assume WLOG that F is $\mathcal{O}(\log n)$ depth.

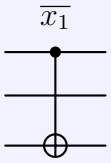
Solution

We need an upgrade of part (b) first. The following circuit computes $x_1 \wedge x_2$ in the stronger sense that it acts exactly like a gate from register 1 to register 3 that is applied if and only if $x_1 \wedge x_2$.

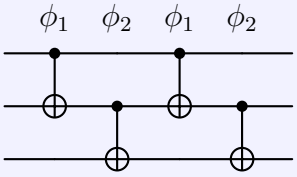


If we prefer the gate to act between a different pair of registers (e.g., from 1 to 2, or 2 to 3), we can simply permute them into positions 1 and 3 with swaps, apply the gate, and then undo the permutation.
Given an arbitrary formula ϕ, it is either

- a literal, which we can implement with a single gate, e.g.,



- an AND, $\phi_1 \wedge \phi_2$, in which case we recursively implement ϕ_1 and ϕ_2 and do



- an OR, $\phi_1 \vee \phi_2$, which we turn into an AND by De Morgan's law, and solve similarly to the AND case above.

The only question is how big the resulting formula will be. Observe that for a formula of depth $d \geq 1$, we may get four recursions on formulas ϕ_1 and ϕ_2 of depth $d-1$. In the worst case, this could lead to a formula of length 4^d.
Fortunately, we recall problem 2. For any polynomial-size formula F, there is a log-depth polynomial-size formula F'. If the depth is $d = \mathcal{O}(\log n)$, then $4^{\mathcal{O}(\log n)}$ becomes $n^{\mathcal{O}(4)} = \mathsf{poly}(n)$, and therefore we have a polynomial length formula.

4. **Median of means [12=2+3+7 marks]** Randomization is useful not just for decision problems. It is common to have a randomized algorithm $\mathcal{A}$ which generates samples such that the mean μ is a number we want to compute. For example, suppose I sample $x, y \in [0,1]$ uniformly at random, and output 1 if $x^2 + y^2 \leq 1$ and 0 otherwise. The mean is $\mu = \pi/4$, and the mean of many samples will approach this value. Similar strategies can be used to estimate area or volume of much more complicated shapes.

   Suppose we want to estimate a number $x^* \in \mathbb{R}$, and we have an algorithm that produces a random estimate x with mean $\mu = x^*$ and known variance σ^2. This question walks through the "mean of medians" method for turning this algorithm into an estimate $\hat{x}$ with the rigorous guarantee that

$$\Pr[|\hat{x} - x^*| \leq \epsilon] \geq 1 - \delta, \tag{1}$$

   or equivalently

$$\Pr[|\hat{x} - x^*| > \epsilon] \leq \delta. \tag{2}$$

   (a) Suppose we use one sample x from algorithm $\mathcal{A}$. Use Chebyshev's inequality to show that we have (2) with $\delta = \mathcal{O}(\sigma^2/\varepsilon^2)$.
   Reminder: Chebyshev's inequality is as follows. For any $k > 0$,

$$\Pr[|x - x^*| \geq k\sigma] \leq \frac{1}{k^2}.$$

   Solution

   This is pretty direct. With $k = \frac{\epsilon}{\sigma}$ we have

$$\begin{aligned} \Pr[|x - x^*| > \epsilon] &= \Pr[|x - x^*| > k\sigma] \\ &\leq \frac{1}{k^2} && \text{by Chebyshev's inequality} \\ &= \mathcal{O}\left(\frac{\sigma^2}{\epsilon^2}\right). \end{aligned}$$

   (b) Now suppose we take n independent samples $x_1, \ldots, x_n$ from algorithm $\mathcal{A}$ and take their empirical mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Show that we can achieve (2) with any desired accuracy $\epsilon > 0$ and confidence $\delta > 0$ by taking a mean of

$$n = \mathcal{O}\left(\frac{\sigma^2}{\delta\epsilon^2}\right)$$

   samples.

   Solution

   You should know from your statistics classes that taking an average of n

independent samples reduces the variance by a factor of n. To remind you,

$$\mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\mathrm{Var}(x_i) = \frac{\sigma^2}{n}.$$

If we then apply Chebyshev's inequality to the mean, we get

$$\Pr[|x - x^*| \geq \epsilon] = \mathcal{O}\left(\frac{\sigma^2}{n\epsilon^2}\right).$$

We want this to be less than δ, so the estimate is correct with high probability. Rearranging $\frac{C\sigma^2}{n\epsilon^2} < \delta$ for n gives $n \geq \frac{C\sigma^2}{\epsilon^2\delta}$, and the claimed result follows.

(c) Now let us take $n = ms$ independent samples from $\mathcal{A}$, divided into m groups of size s. Let the estimate be $\hat{x} := \mathrm{median}_{i\in[m]}\left(\mathrm{mean}(x_{i1}, \ldots, x_{is})\right)$. Show that with the right balance of m and s, we can achieve accuracy $\epsilon > 0$ and confidence $\delta > 0$ with

$$n = \mathcal{O}\left(\frac{\sigma^2 \log(1/\delta)}{\epsilon^2}\right)$$

samples. *Hint: A Chernoff bound may be useful. For example, for independent Bernoulli*[2] *random variables* $X_1, \ldots, X_n$*, the sum* $X = \sum_{i=1}^{n} X_i$ *has the property*

$$\Pr\left[X \geq (1+a)\mathbb{E}[X]\right] \leq \exp\left(-\frac{a^2\mathbb{E}[X]}{2+a}\right) \tag{3}$$

for all $a > 0$.

Solution

Let $\mu = \mathbb{E}[X]$. Let $X_1, \ldots, X_m$ be Bernoulli random variables that take the value 1 if the ith group mean $x_i := \frac{1}{s}\sum_j x_{ij}$ differs from μ by more than ϵ. Let $p := \Pr[X_i = 1]$ be the probability that a group is wrong; since all groups are the same size and independent, p does not depend on i. In part (b) we showed that

$$p = \Pr[X_i = 1] = \Pr[|x_i - \mu| > \epsilon] = \mathcal{O}\left(\frac{\sigma^2}{s\epsilon^2}\right).$$

The final estimate is

$$\hat{x} := \operatorname*{median}_{i\in[m]} x_i.$$

In order for $|\hat{x} - \mu|$ to be larger than ϵ, either at least half of the x_i are bigger than $\mu + \epsilon$, or at least half are smaller than $\mu - \epsilon$. Either way, at least half of the X_i are 1, and thus the total is $X = \sum_{i=1}^{m} X_i > \frac{m}{2}$. This

[2] I.e., on $\{0, 1\}$

looks like the Chernoff bound, which we now apply:

$$\begin{aligned}\Pr[X \geq \tfrac{m}{2}] &= \Pr[X \geq (1+a)\mathbb{E}[X]] \\ &\leq \exp\left(-\frac{a^2\mathbb{E}[X]}{2+a}\right),\end{aligned}$$

where we choose a so that $\frac{m}{2} = (1+a)mp$, or $a = 1 - \frac{1}{2p}$.

$$\begin{aligned}\exp\left(-\frac{a^2\mathbb{E}[X]}{2+a}\right) &= \exp\left(-\frac{(1-\frac{1}{2p})^2 mp}{3+\frac{1}{2p}}\right) \\ &= \exp\left(-\frac{(2p-1)^2}{2(6p+1)}m\right).\end{aligned}$$

Notice that $\frac{(2p-1)^2}{2(6p+1)}$ is positive for all probabilities $0 \leq p < \frac{1}{2}$. The precise value of p does not matter; any constant will lead to $\exp(-\mathcal{O}(m))$ probability of failure. Recall that $p = \mathcal{O}(\sigma^2/s\epsilon^2)$, so it suffices to choose $s = \mathcal{O}\left(\frac{\sigma^2}{\epsilon^2}\right)$ to ensure p is less than $\frac{1}{4}$. For $p \geq \frac{1}{4}$, we have

$$\exp\left(-\frac{(2p-1)^2}{2(6p+1)}\right) \leq \exp(-1/20),$$

and thus

$$\begin{aligned}\Pr[|\hat{x} - \mu| > \epsilon] &\leq 2\Pr[X \geq \tfrac{m}{2}] \\ &\leq 2\exp\left(-\frac{(2p-1)^2}{2(6p+1)}m\right) \\ &\leq 2\exp\left(-m/20\right).\end{aligned}$$

In order to guarantee $\Pr[|\hat{x} - \mu| > \epsilon] \leq \delta$, it is enough to take $m = \mathcal{O}(\log(1/\delta))$.
We have shown that $m = \mathcal{O}(\log(1/\delta))$ groups of $s = \mathcal{O}\left(\frac{\sigma^2}{\epsilon^2}\right)$ samples suffice, for a total of

$$n = \mathcal{O}\left(\frac{\sigma^2}{\epsilon^2}\log(1/\delta)\right).$$