CS 40: Computational Complexity

Sair Shaikh

October 20, 2025

Problem 18. In the proof of the Valiant-Vazirani theorem from the lecture, the CNF formula produced by the randomized mapping reduction is rather non-explicit: we invoked the Cook–Levin reduction to argue that the condition

$$h(x_1, \dots, x_n) = 0^k$$

(where h is a function from a suitable 2-universal hash family), being polynomial-time checkable, admits a polynomial-sized CNF formula using some auxiliary variables; these auxiliary variables are the ones introduced by Cook–Levin, to encode the computational tableau. This non-explicitness is unsatisfactory. (Ha ha!)
Remedy this by choosing the specific hash family from Problem 14 (the $Ax+b$ family) to make the reduction explicit. That is, write out the predicate

$$\psi(x_1, \dots, x_n) := \varphi(x_1, \dots, x_n) \wedge (h(x_1, \dots, x_n) = 0^k)$$

as a CNF formula $\tilde{\psi}(x_1, \dots, x_n, y_1, \dots, y_m)$ for some $m = \text{poly}(n)$, with the property that if ψ has a unique satisfying assignment, so does $\tilde{\psi}$. Prove this property of $\tilde{\psi}$.

Solution. Note that since $\varphi(x_1, \cdots, x_n)$ is a SAT instance, it is already a CNF. Thus, as ψ is an and of two clauses, it suffices to express the second, i.e. $h(x_1, \cdots, x_n) = 0^k$ as a polynomial sized CNF for a given h.

Write $x = [x_1, \cdots, x_n]^T$ as a vector. Using the 2-Universal hash family from before, there exists a matrix $A \in \{0,1\}^{k \times n}$ and vector $b \in \{0,1\}^k$, with:

$$h(x) = Ax + b$$

Note that:

$$Ax + b = 0^k \iff \bigwedge_{i=1}^{k} (Ax+b)_i = 0$$

Thus, as $k \in \text{poly}(n)$ in the original proof, it suffices to express $(Ax+b)_i = 0$ as a polynomial sized CNF as this gives a CNF for $h(x) = 0^k$. Thus, that's what we will show in the rest of the proof.

Noting the matrix product expansion, this means:

$$(Ax+b)_i = b_i + \sum_{j=1}^{n} A_{ij}x_j = 0 \iff \sum_{j=1}^{n} A_{ij}x_j = b_i$$

Let I be the set of indices for which the entry in the ith row of A is 1, i.e.:

$$I := \{j \in [n] : A_{ij} = 1\}$$

Then, by the definition of products mod 2, the previous equation reduces to:

$$\sum_{j=1}^{n} A_{ij}x_j = b_i \iff \sum_{j\in I} x_j = b_i$$

By relabelling the variables, we will write $\sum_{j\in I} x_j$ as $\sum_{j=1}^{t} x_j$ for $t = |I|$. Next, we introduce additional variables $y_1, \cdots, y_t$ and claim the following:

$$\sum_{j=1}^{t} x_j = b_i \iff \exists y_1, \cdots, y_t : (y_1 = x_1) \wedge (y_2 = y_1 + x_2) \wedge \cdots \wedge (y_t = y_{t-1} + x_t) \wedge (y_t = b_i)$$

Clearly, if $\sum_{j=1}^{t} x_j = b_i$ then there exist y_is, representing the partial sums of x_is, that satisfy the clauses on the left, with the full sum, $y_t = b_i$. Moreover, if such an assignment of y_is exists, then expanding out the partial sums that hold by the truth of the clauses, we get:

$$\begin{aligned} b_1 &= y_t \\ &= y_{t-1} + x_t \\ &= y_{t-2} + x_{t-1} + x_t \\ &= \cdots \\ &= y_1 + x_2 + \cdots + x_t \\ &= x_1 + \cdots + x_t \end{aligned}$$

Thus, we only need to write the expression on the RHS as a CNF. We do this as follows:

1. $(y_1 = x_1)$ can be written as $(y_1 \vee \overline{x_1}) \wedge (\overline{y_1} \vee x_1)$. Note that if $x_1 = 1$ (second clause already satisfied), then, $y_1 = 1$ is forced to satisfy the first clause. Instead, if $x_1 = 0$ (first clause already satisfied), $\overline{y_1} = 1$ is forced to satisfy the second clause, thus, $y_1 = 0$. In both cases, the only satisfying assignment forces $x_1 = y_1$.

2. We claim that we can write $y_r = y_{r-1} + x_r$ as the following CNF:

$$(x_r \vee y_{r-1} \vee \overline{y_r}) \wedge (x_r \vee \overline{y_{r-1}} \vee y_r) \wedge (\overline{x_r} \vee \overline{y_{r-1}} \vee \overline{y_r}) \wedge (\overline{x_r} \vee y_{r-1} \vee y_r)$$

   Note that if $x_r = 0$, then the last two clauses are already satisfied, and the first two clauses, after substituting $x_r = 0$, reduce to $y_r = y_{r-1}$ (notice similarity to (1)).

Instead, if $x_r = 1$, then the first two clauses are already satisfied, and the last two clauses, after substituting $x_r = 1$, reduce to $y_r = \overline{y_{r-1}}$.
Thus, in both cases, the only satisfying assignments (two, for a given value of x_r) satisfy $y_r = y_{r-1} + x_r$.

3. Finally, the argument for $y_t = b_i$ is identical to (1).

Note that each of these required at most 4 clauses with at most 3 variables each, i.e. a constant number. Since we had $t+1$ of these expressions, and $t = |I| \leq n$, we only used a linear number of additional clauses and additional variables to express $(Ax+b)_i = 0$ as a CNF. Thus, we are done.

Problem 2. Let FIND-SAT be the following problem: given a CNF formula $\varphi(x_1, \ldots, x_n)$, either output any one Boolean assignment $(\alpha_1, \ldots, \alpha_n) \in \{0,1\}^n$ that satisfies φ, or else report that no such assignment exists.
Suppose you are given an oracle for SAT. As you know, you can invoke this oracle repeatedly to solve FIND-SAT in polynomial time. However, the standard way of doing this requires adaptive queries to the oracle, i.e., each question you pose to the oracle depends on its answers to your previous questions.
Prove that there exists a randomized polynomial-time algorithm for FIND-SAT that uses only non-adaptive queries to the oracle, i.e., the algorithm prepares a (possibly randomized) batch of questions to pose to the oracle, asks them all at once, and based on the answers (and perhaps using further computation of its own), produces an output that is correct with probability at least $\frac{2}{3}$.

Solution. Our algorithm, A, is as follows:

1. Use the Valiant-Vazirani reduction k times on $\varphi(x_1, \cdots, x_n)$ to get formulae $\psi_1, \cdots, \psi_k$ (in x_i's and some auxillary variables), where $k := \lceil 8 \ln(3) n \rceil$.
2. Use the oracle for SAT to find whether φ is satisfiable and the following are satisfiable, for each $i \in [k]$:
   (a) ψ_i
   (b) $\psi_i \wedge x_j$ for all $j \in [n]$
3. If φ is unsatisfiable, report no such assignment exists.
4. Iterate over the ψ_i. For each ψ_i that is satisfiable (i.e. the oracle accepted (a) for this formula), use the answers to all the queries in (b) to come up with a candidate assignment, i.e. set $x_j = 1$ if $\psi_i \wedge x_j$ was accepted, and 0 otherwise. Verify whether the candidate assignment satisfies φ. If the candidate assignment for any ψ_i works, return this assignment as the solution. Otherwise, report that no such assignment exists.

Since $k \in \text{poly}(n)$, we only query the oracle $O(kn) \subseteq \text{poly}(n)$ times in (2), and we derive and verify at most k assignments each of which takes polynomial time, the algorithm overall runs in polynomial time. Thus, we only need to prove that the algorithm returns a correct output with probability at least $\frac{2}{3}$.

First, we claim and prove the following lemma:

Lemma 0.1. *If $\psi_i \in USAT$ for any $i \in [k]$, we return correctly with certainity.*

Proof. Note that we always verify an assignment before returning it, thus, if an assignment is returned, it is correct with certainty. Thus, the only mistakes are made when we report that no assignments exist. Thus, it suffices to show that if any $\psi_i \in$ USAT, then an assignment is returned.

If $\psi_i \in$ USAT, the result of each query $\psi_i \wedge x_j$ uniquely determines the value of x_j in the unique satisfying assignment for ψ_i, which we pick for our candidate assignment in step (3). Then, this candidate assignment satisfies φ, assuming the particular form of the Valiant-Vazirani reduction presented in class and in the question above. Then, either we return a correct assignment before the loop reaches ψ_i, or we return this assignment. In particular, we would not return that there are no assignments to φ and thus would not make an error. That concludes the proof. $\square$

Using this lemma, A returning incorrectly implies that (i) φ is satisfiable (otherwise we'd catch this in Step (3)), and (ii) that none of the ψ_i were in USAT. That is,

$$\mathbf{Pr}[\text{A returns incorrectly}] \leq \mathbf{Pr}\left[\bigwedge_{i=1}^{k} \psi_i \notin \text{USAT}\right]$$

Since we run the Valiant-Vazirani reductions using independent random bits, we note that $\psi_i \notin$ USAT are all mutually independent. Thus, using the Valiant-Vazirani bounds (knowing that $\varphi \in$ SAT), we write:

$$\mathbf{Pr}\left[\bigwedge_{i=1}^{k} \psi_i \notin \text{USAT}\right] = \mathbf{Pr}\left[\psi_i \notin \text{USAT}\right]^k \leq \left(1 - \frac{1}{8n}\right)^k$$

We want this probability to be at most $\frac{1}{3}$. Thus, calculating, we get:

$$\begin{aligned}
&\left(1 - \frac{1}{8n}\right)^k \leq \frac{1}{3} \\
\implies &k \ln\left(1 - \frac{1}{8n}\right) \leq -\ln(3) \\
\implies &k \geq -\frac{\ln(3)}{\ln\left(1 - \frac{1}{8n}\right)} \qquad \left(\text{as } \ln\left(1 - \frac{1}{8n}\right) < 0\right)
\end{aligned}$$

Using the Taylor expansion for $\ln(1-x)$, we get:

$$\begin{aligned}
\ln(1-x) &= \sum_{t=1}^{\infty} -\frac{x^t}{t!} \\
&\implies \ln\left(1 - \frac{1}{8n}\right) \geq -\frac{1}{8n} \\
&\implies -\frac{1}{\ln\left(1 - \frac{1}{8n}\right)} \geq 8n
\end{aligned}$$

Thus, overall, we get:

$$k \geq -\frac{\ln(3)}{\ln\left(1 - \frac{1}{8n}\right)} \geq (8 \ln 3)n$$

Thus, our chosen k suffices to prove the required bounds.