Cheating on Circuit Sat is hard

Rohan Jhunjunwala
University of California, Berkeley
Electrical Engineering and Computer Science
rjhunjhunwala80@berkeley.edu

Abstract

This work came out of an independent exploration of a humorous project proposal for SIGBOVIK (http://www.sigbovik.org/) involving "pessimal proofs" aka proof, that can be arbitrarily long yet still provide exponential advantage over brute force. This is not official academic work, and this was not submitted to any conference.

We consider a model of computation for CSP's on n boolean values for which the solver is allowed to request partial solutions. More precisely, the algorithm may optionally request the solution for $\leq pn$ variables ($p \in (0, 1)$) For each such variable, the solver receives an assignment of the variable along with a promise that their exists a feasible solution consistent with this partial assignment. We show a relatively intuitive result: Conditioned on ETH: there exists no polynomial time solver for circuit SAT even if the solver is granted a partial assignment to p * n variables of its choice. We then suggest a number of open related problems, and suggest practical applications for our result.

1 Formalisms

For this paper, we consider the search problem circuit sat, that given an explicit $f(\overrightarrow{x}) : \{0,1\}^n \longrightarrow \{0,1\}$ find any $\overrightarrow{x}^*$ s.t $f(\overrightarrow{x}^*) = 1$ During the process the solver may make $\leq p * n$ queries to some oracle. $g(i), i \in \{0..n-1\}$ such that:

$$\exists \overrightarrow{x}^* \quad \forall i \quad f(\overrightarrow{x}^*) = 1 \wedge \overrightarrow{x}^*_i = g(i)$$

Effectively, g allows partial access to some feasible solution.

There is an obvious SETH based bound that suggests, that, for a carefully chosen $f(x)$ there does not exist some algorithm to find a satisfying $\overrightarrow{x}^*$ given access to such an oracle. Because such an algorithm naturally leads to a $O(2^{p*n} * n^k)$ algorithm simply by enumerating over possible responses of g. For p bounded away from 1 by a constant epsilon: this gives an algorithm for k-sat with runtime

$$O(2^{(1-\epsilon)*n})$$

contradicting the Strong Exponential time hypothesis.
Our main result is stronger, needing only to be conditioned on the ETH.

2 Main Result

We will now appeal only to the exponential time hypothesis to show the nonexistence of an algorithm that is able to successfully use $p * n$ hints to solve circuit sat.

Recall that the exponential time hypothesis suggests that: k-sat for $k \geq 3$ has runtime

$$O(2^{a_k * n}) \wedge \forall k \geq 3, a_k > 0$$

The strong exponential time hypothesis suggests that the limit of a_k is 1. Neither of these hypothesis are proven, however, it seems more easy to believe that all a_k are nonzero then, they necessarily approach exactly one.

The key idea to our proof is introspection. The algorithm can recurse and ask itself, "what series p * n hints would result in the outer algorithm as a whole terminating in polynomial time with a correct assignment."

We use a nondeterminsitic turing machine to model a potential algorithm that can solve circuit sat in polynomial time without requiring a long witness. The nondeterministic machine receives it's p * n nondeterministic choices as an explicit witness string of $p * n$ bits, and this can be considered a deterministic circuit on $p * n$ inputs.

Algorithm 1 Calculate $\overrightarrow{x}^* \longrightarrow f(\overrightarrow{x}^*) = 1$

Require: $n \geq 0, p \in (0,1), h \in \{0,1\}^{p*n}, \overrightarrow{x}^* \in \{0,1\}^n$
Ensure: H chosen nondeterministically $f(\overrightarrow{x}^*) = 1$
Use the $p*n$ hints from H to return a feasible assignment in polymomial time, return any assignemnt (feasible or otherwise if the witness string was invalid.

Let's consider an algorithm Y.

Algorithm 2 Calculate $\overrightarrow{x}^* \longrightarrow f(\overrightarrow{x}^*) = 1$

Require: $n \geq 0, p \in (0,1), \overrightarrow{x}^* \in \{0,1\}^n$
Ensure: $f(\overrightarrow{x}^*) = 1$
if recursion depth is less than d **then**
Construct a function g(h), which is whether or not algorithm X succeeds using h as a hint vector. Invoke Algorithm Y on g in order to find the correct vector of hints, and then later the correct vector of assignment to f(x).
else
Give up, and solve the problem with brute force
end if

Now, algorithm Y feels a little too powerful, but let's formalize this idea. We can consider it's run time consider it's recurrence relation. Let's say that f is a circuit that has size $C(n)$ and algorithm x takes time $n^b * C(n)^a$

$$X(n, C(n)) = X(p * n, n^b * C(n)^a) + n^b * C(n)^a$$

We know C(n) is at least linear, so we can replace define $k = a + b$

$$X(n, C(n)) \leq X(p * n, C(n)^k) + C(n)^k$$

If we unwind this recursion to some depth d and use brute force to solve the last layer...
We get an algorithm with runtime dominated by

$$f(n) \in O(2^{(p^d)*n} * C(n)^{k^d})$$
$$\forall \epsilon > 0 \exists d, f(n) \in O(2^{\epsilon * n})$$

This get's an algorithm with arbitrarily good exponent, this contradicting the exponential time hypothesis.

3 Further Work

One obvious extension of this work worth considering, is if algorithm 1 also received a hint as to which subset of variables are most tricky to solve, and then could use those to request hints. A lengthy calculation shows that our conditional nonexistence result still only holds for $p < .22$ If Algorithm Y simply needs to recurse on not just a hint

but also a description of the subset of variables to provide. This bitstring remains short iff: p ¡ .22. Effectively ...

$$p + log_2(nCR(n, p * n))/n \leq 1 - \epsilon$$

For $p > .22$ there is an eplicit way to construct algorithm X where it receives a hint of which of $nCR(n, p * n)$ variables to solve, uses that description string to produce a bit string of length $> (1 - p) * n$and uses the concatenation of the variables provided with this other bit string to offer a satisfying assignnment.

A second obvious extension of our work is to (conditionally prove) prove nonexistence of algorithm X remains a dream for restricted f(x) in some way, one natural restriction is in the form of 3sat. Such conditional nonexistence results might also require bounds on p.

4 Applications

One natural application to this, is to side channel attacks to crypto. Some side channel attacks partially leak a key. if a crypto problem can be cast as a circuit for wich the cheating CSP remains hard for all p, it can be seen as a mark of resistance against side channel attacks.