SAT Solving and Computational Complexity

Ryan Williams, Stanford

(with help from Mohan Paturi, UCSD)

SAT Solving and Computational Complexity

~ 1%

Ryan Williams Stanford

The Points of This Talk

Discuss the worst-case complexity of SAT and the complexity theory surrounding it

Why?

1. I want to know how the techniques and ideas of practical solvers impact what is possible in theory.
2. I believe SAT can probably be solved even faster, and we're currently exploring a "local optimum"
3. Solving the worst-case implies *lower bounds*

Complexity Theory Refresher

P and NP (by request ☺)

Def. A computational problem π is a set of strings

Def. Algorithm A solves π in polynomial time if there is a polynomial p(n) such that on all inputs x,

1. A(x) runs in at most $p(|x|)$ steps, and
2. A(x) outputs 1 exactly when $x \in \pi$

Def. A computational problem π is in P if there is an algorithm A solving π in polynomial time.

$(\exists$ polytime algorithm A$)$ $(\forall$ strings x$)$ $[x \in \pi \text{ iff } A(x) = 1]$

Def. A computational problem π is in NP if

$$(\exists\, A, p)\ (\forall\, x)\ [x \in \pi \text{ iff } \underbrace{(\exists\, y_x : |y_x| \le p(|x|))}_{\text{Nifty Proof of x}}\underbrace{[A(x, y_x) = 1]}_{\text{verifier}}]$$

$\boxed{\approx 2^{p(|x|)} \text{ time to find } y_x}$

NP-completeness

- NP-complete problems are the "hardest" in NP
- Cook-Levin Theorem (1971): SAT is NP-complete

For every π in NP,
there is a polynomial-time reduction *A* from π to SAT.

$x \in \pi \iff A(x)$ is a satisfiable CNF formula

Every NP problem can be efficiently encoded in the "programming language" SAT

- The triumph of SAT solving in practice is also a triumph of the Cook-Levin Theorem!

Flavors of SAT

Circuit-SAT = { satisfiable Boolean circuits }

Formula-SAT = { satisfiable propositional formulas }

CNF-SAT = { satisfiable boolean formulas in CNF }

k-SAT = { SAT where all clauses have at most k literals}

Measures of SAT instances:

- n = number of variables
- m = number of clauses/connectives/gates

All the above are solvable in $\mathbf{2^n \cdot poly(m)}$ time.

All are **NP-complete**, and hence are ***equivalent*** in the theory of NP-completeness.

Faster Algorithms for NP Problems

Have: Verifier algorithm A(x, y) that reads **w(|x|)** bits of the nifty proof **y**, and runs in **poly(|x|)** time.

Goal: Deterministic/randomized algorithm which on all **x**:

1. Runs in ***much less than*** $2^{w(|x|)}$ **poly(|x|)** time
2. Finds a witness **y** such that **A(x, y)** accepts (or correctly concludes there is none)

***Many* known algorithms of this form:**

- **3-SAT: $O(1.31^n)$ time, n = number of variables**
- **k-SAT: $O(2^{n - n/k})$ time**
- **Hamiltonian Path: $O(1.66^n)$ time, n = number of vertices**
- **Max Indep. Set: $O(1.21^n)$, on degree-3 graphs: $O(1.09^n)$**
- **3-Coloring: $O(1.33^n)$ (improves on 3^n)**
- **Chromatic Number: $O(2^n)$ (improves on $(\Delta+1)^n$)**

Time Complexity of k-SAT (with respect to n)

[MS 85] **1.62^n for 3-SAT, 1.84^n for 4-SAT, etc.** **(DPLL branching)**

[Schiermeyer 93, Kullmann 96, …]

[PPZ 97] **$2^{n(1-1/k)}$** for k-SAT **(randomized branching w/ restart)**

(e.g. 1.59^n for 3-SAT)

[PPSZ 98] **1.31^n** for Unique 3-SAT **(randomized w/ resolution)**

[Sch 99] **$2^{n(1-1/\Theta(k))}$** for k-SAT, **1.34^n** for **3-SAT** **(rand. local search)**

[... combinations of the above...]

[Hertli 11] **1.31^n** for 3-SAT and **1.47^n** for 4-SAT

(PPSZ with better analysis)

ALL of these take $2^{n(1-1/\Theta(k))}$ time to solve k-SAT for constant k.

Running time converges to 2^n as $k \rightarrow \infty$

For CNF SAT, best known algs take $O(2^{n - n/(\log(m/n))})$ time [CIP'06]

Why do k-SAT algorithms get worse, as k increases?

3-SAT, 4-SAT, CNF-SAT are all NP-complete…

Answer: The reduction from 4-SAT to 3-SAT blows up the number of variables!

You have to introduce a new variable for each clause.

A 3-SAT algorithm using c^n **time** would only imply a $\approx c^{n+m}$ **time** algorithm for 4-SAT, using the typical polynomial-time reduction

Generally speaking, **m >> n** …

Moral: The representation of the formula matters.

Refinements of P $\neq$ NP

There is a dramatic difference in how well various NP-complete problems can be solved in practice.

A formal understanding of 'what is possible' requires more 'fine-grained' assumptions than P $\neq$ NP.

We need lower bound hypotheses that guide us to a better understanding of the frontier: what we can and cannot expect to solve much faster than brute force.

Ultimately, we want to find better algorithms, even if they take super-polynomial time. (And if we can't, we want to know what the consequences would be.)

Hardness Hypotheses for NP

- ***Is 3-SAT in $2^{\epsilon n}$ time, for every $\epsilon > 0$?***

  Exponential Time Hypothesis [IPZ'01]: **Conjectures "no"**

  ETH: "3-SAT is not in $(1+\epsilon)^n$ time for some $\epsilon > 0$"

- ***Is k-SAT in $2^{\delta n}$ poly(m) time for some universal $\delta < 1$?***

  Strong ETH [IP'99,CIP'09]: **Conjectures "no"**

  SETH: "For all $\delta < 1$ there is a k such that k-SAT is not in $(2-\delta)^n$ poly(m) time"

Theorem: **SETH implies ETH**

Useful: These hypotheses entail many complexity predictions, and (so far) they are nicely consistent with general wisdom

Evidence for SETH and ETH?

All known algorithms are consistent with SETH and ETH

[Beck Impagliazzo '13]

1. There are unsatisfiable k-CNFs which *require* regular resolution proofs of size $2^{n-\frac{n}{k^{0.25}}}$ (Strong ETH holds for Reg. Res.)

2. There are unsatisfiable k-CNFs which require general resolution proofs of size $(1.5)^{n-\frac{n}{k^{0.25}}}$ (ETH holds for Gen. Res.)

[Chen Scheder Talebanfard Tang '13]

3. There are satisfiable k-CNFs for which the PPSZ algorithm takes at least $2^{n\left(1-\frac{\log k}{k}\right)}$ steps to solve, in expectation.

So, if SETH/ETH are false, we probably need radically new algs.

(On the other hand, all three of the above constructions are based on encodings of linear systems modulo 2...)

Key Tool: Sparsification Lemma

Lemma [IPZ 01] For every k and $\varepsilon > 0$, there is a $c > 1$ such that *every* k-CNF on n variables can be expressed (in $2^{\varepsilon n}$ time) as the **OR** of $2^{\varepsilon n}$ k-CNFs with $c\,n$ clauses.

Proof Idea: Pick a 'sunflower' of shortest clauses and 'branch' on whether the 'core' is true or false. Stop whenever the number of clauses is $O(n)$.

Corollary If 3-SAT is in time $2^{\varepsilon m}$ for all $\varepsilon > 0$ where m is the number of clauses, then ETH is false: 3-SAT is in time $2^{\varepsilon n}$ for all $\varepsilon > 0$

Corollary: SETH implies ETH

If k-SAT needs $2^{\varepsilon n}$ for some $\varepsilon > 0$, then so does 3-SAT

Sparsification and SETH

SETH: For every $\delta > 0$ there is a k such that k-SAT is not in $(2-\delta)^n$ poly(m) time.

Sparsification implies that SETH is equivalent to:

For every $\delta > 0$, there's a k, c such that k-SAT with ***c n clauses*** is not in $(2-\delta)^n$ time.

Tight lower bounds under ETH

Assuming ETH the problems Independent Set, Clique, Vertex Cover, Dominating Set, Graph Coloring, Max Cut, Set Splitting, Hitting Set, Min Bisection, Feedback Vertex Set, Hamiltonian Path, Max Leaf Spanning Tree, Subset Sum, Knapsack, 3-Dimensional Matching, Cluster Editing, Treewidth ***and many others*** do not have $2^{o(n)}$ time algorithms.

ETH ⇒ Complexity of k-SAT Must Grow

Theorem [IP 99]

ETH ⇒ **There are infinitely many k such that (k+1)-SAT requires exponentially more time to solve than k-SAT.**

ETH predicts that k-SAT algorithms will perform worse as k increases.

Proof Idea: **Reduce k-SAT on n vars to K'-SAT on n' vars, where K' >> K, but n' << n.**

Given a k-CNF, *sparsify* it. WLOG, each var now has < c occurrences. Pick $n/(ck)$ "independent" vars and "resolve them out." This increases clause width by a constant, and removes a constant fraction of vars.

More Predictions Beyond NP!

For **many fundamental *polynomial-time* problems**, improving the best known algorithms, even slightly, implies **¬SETH** or **¬ETH**

Thus SETH and ETH also predict the optimality of many known polynomial-time algorithms.

At least one prediction of SETH was later proved true! (3-Party Set Disjointness)

A few results [PW'10]

- ***k-Dominating Set*:** Given a graph (V,E), find a k-set of nodes S such that $S \cup N(S) = V$.

  Solvable in $n^{k+o(1)}$ time [EG'04]

  If solvable in $O(n^{k-\epsilon})$ time for some $k > 2$, $\epsilon > 0$ $\Longrightarrow$ $\neg$**SETH**

- ***2SAT2*:** Given a 2CNF on $n^{1+o(1)}$ clauses with two extra clauses of arbitrary length, is it satisfiable? Solvable in $n^{2+o(1)}$ time

  If solvable in $O(n^{2-\epsilon})$ time for some $\epsilon > 0$ $\Longrightarrow$ $\neg$**SETH**

- **d-SUM:** Given n numbers, are there d that sum to zero?

  ETH $\Longrightarrow$ d-SUM requires $n^{\Omega(d)}$ time

- ***O.V.:*** Given a set of n binary d-dimensional vectors, are there two with inner product equal to zero?

  If solvable in $n^{2-\epsilon} 2^{o(d)}$ time for some $\epsilon > 0$ $\Longrightarrow$ $\neg$**SETH**

Faster K-Dominating Set $\Rightarrow$ $\neg$**SETH**

Theorem. k-Dominating Set in **$O(n^{k-\epsilon})$** time
$\Rightarrow$ SAT in **$2^{(1-\epsilon/k)n}$ poly(m)** time

Proof. Given F with n variables and m clauses, we construct a graph G on **$O(k\ 2^{n/k} + m)$** nodes, where

G has a dominating set of size k $\Leftrightarrow$ F is satisfiable

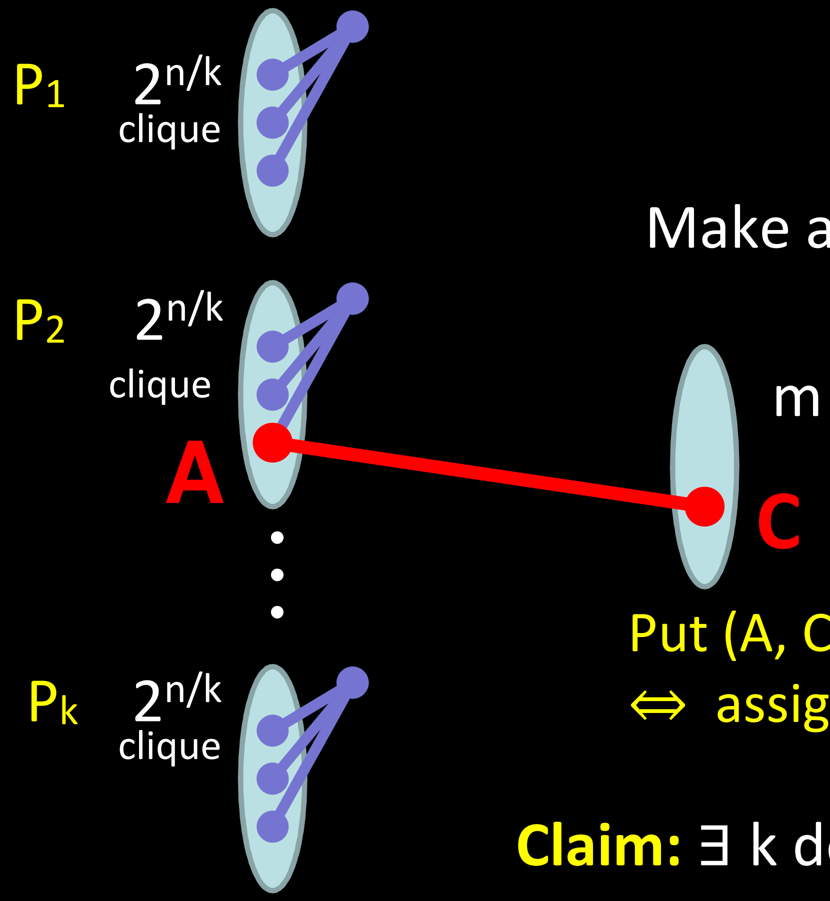
Note: Theorem shows that even tiny improvements in solving k-DS imply tiny SAT improvements

We construct a graph G on **O(k $2^{n/k}$ + m)** nodes, where

G has a dominating set of size k $\Leftrightarrow$ F is satisfiable

Split n vars into k parts $P_1, \ldots, P_k$ with $\leq n/k+1$ variables each.

Make nodes for *all* assignments to the variables in a part.



Make a node for every clause in F

Put (A, C) as an edge
$\Leftrightarrow$ assignment A satisfies clause C

Claim: $\exists$ k dominating set in G $\Leftrightarrow$ F is SAT

New Complexity Theory Through SAT Solving

Faster Circuit-SAT algorithms?
This is open!

We know k-SAT can be solved in << 2^n time

3-SAT: **$O(1.31^n)$ time**

K-SAT: **$O(2^{n - n/k})$**

CNF SAT: **$O(2^{n - n/\log n})$**

Q: Why can't we solve Circuit-SAT faster than 2^n?

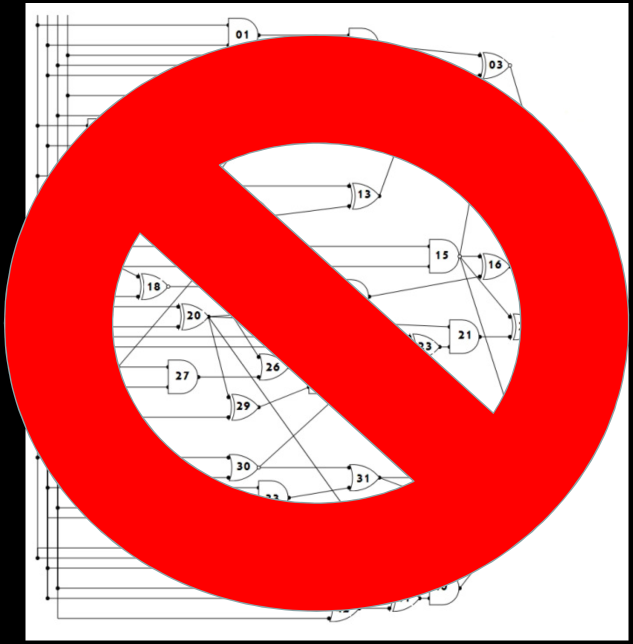
A: Again, the transformation from Circuit Satisfiability to CNF blows up the number of variables!

Introduce a new variable for each *gate* of the circuit

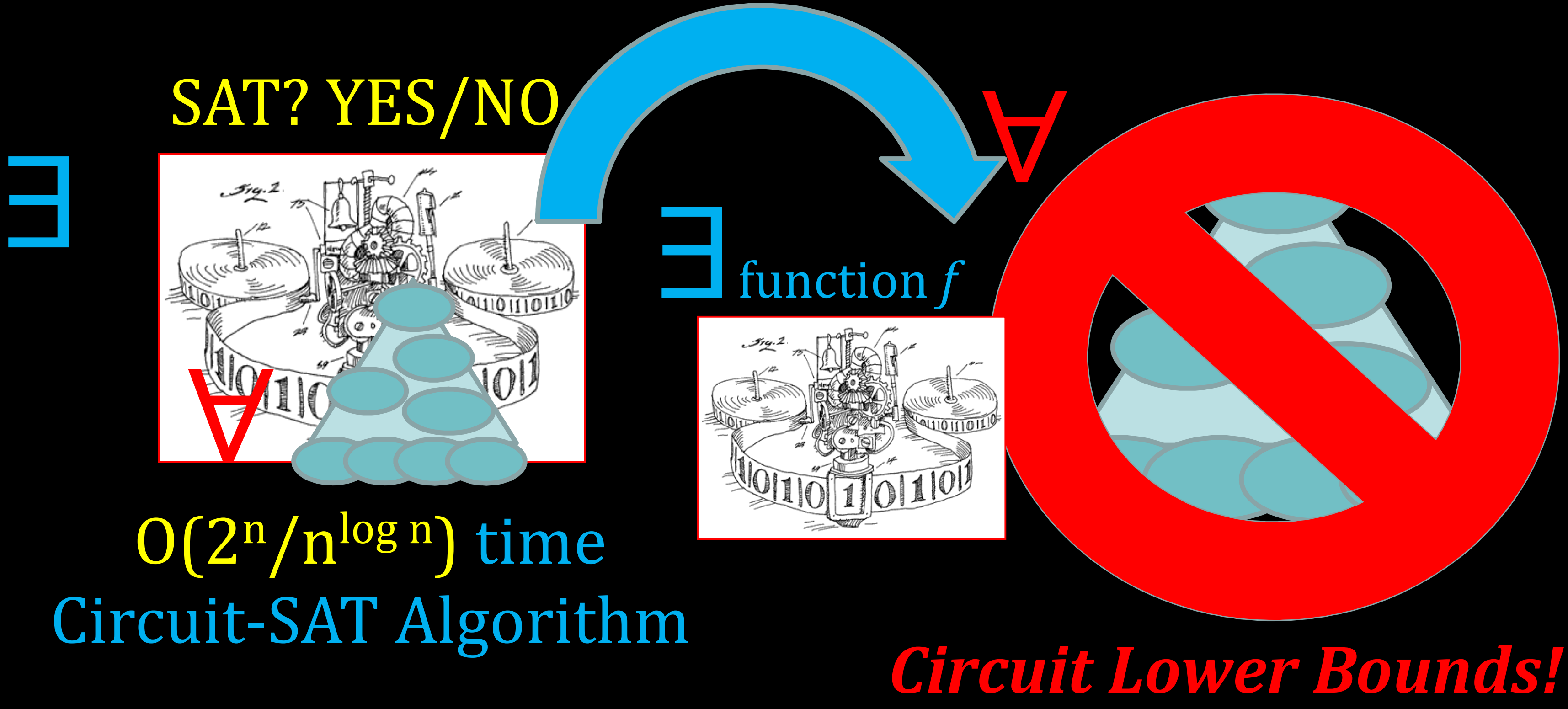
Circuit-SAT Algs Imply Lower Bounds!

If **Circuit-SAT** is in $\frac{2^n}{n^{\log n}}$ time for all poly(n)-size circuits…

Then size lower bounds on computing certain functions with **non-uniform circuits**

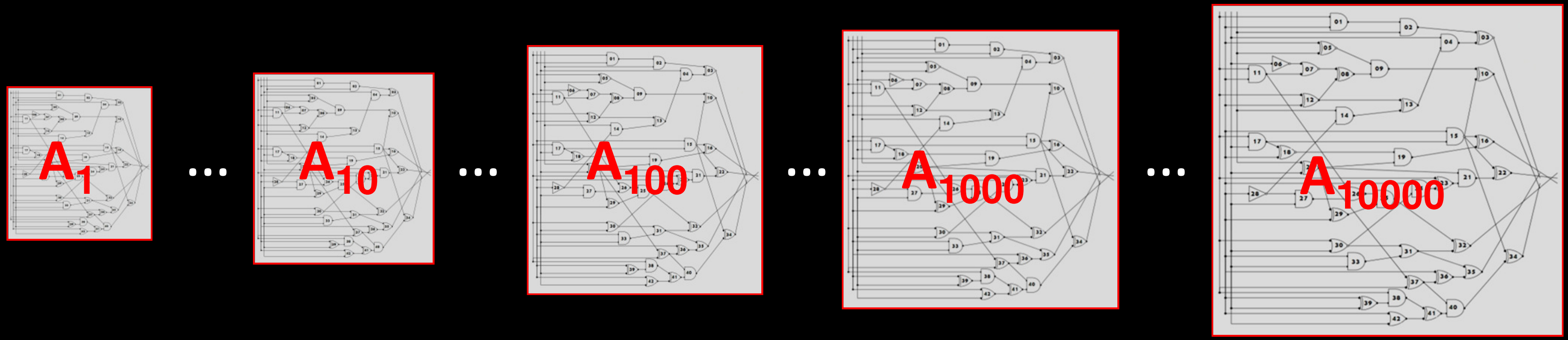


Circuit-SAT Algs Imply Lower Bounds!



Circuit Complexity

For all n, imagine a logical circuit A_n with n bits of input



P/poly = Problems solvable with a **circuit family** **$\{A_n\}$**
where the ***number of gates*** **of A_n** is ≤ **poly(n)**

Conjecture: NP ⊄ P/poly

This is an ***infinite computational model!***

$\{0^n$ | the nth Turing machine halts on blank tape$\}$ ∈ P/poly

The usual techniques of computability theory are essentially powerless for understanding **P/poly**

Non-Uniform Computation

NP $\not\subset$ P/poly would imply **P $\neq$ NP**

But non-uniformity can be **very** powerful!

Still Open: Is **NEXP $\subset$ P/poly?**

Can all problems with ***exponentially long*** solutions be solved with ***polynomial size*** circuit families?

If I were to allow you "infinite" preprocessing time, could you set up small size circuits that can solve any NEXP problem?

Generic Circuit Satisfiability

Let $\mathcal{C}$ be a class of Boolean circuits

$\mathcal{C}$ = {formulas}, $\mathcal{C}$ = {arbitrary circuits}, $\mathcal{C}$ = {CNF formulas}

The $\mathcal{C}$-SAT Problem:
Given a circuit $K(x_1,\ldots,x_n) \in \mathcal{C}$, is there an assignment $(a_1, \ldots, a_n) \in \{0,1\}^n$ such that $K(a_1,\ldots,a_n) = 1$?

$\mathcal{C}$**-SAT** is NP-complete, for essentially all interesting $\mathcal{C}$

$\mathcal{C}$**-SAT** is solvable in $O(2^n |K|)$ time
where **$|K|$ is the size of the circuit K**

SAT Solving Can Help Complexity!

Faster $\mathcal{C}$-SAT Algorithms

$\Rightarrow$ Limitations on Circuits from Class $\mathcal{C}$

Informal Theorem

If $\mathcal{C}$-SAT with n inputs and $2^{n^{\varepsilon}}$ size can be solved in $O(2^n/n^{\log n})$ time,
Then we can find functions in NEXP that cannot be computed with $\mathcal{C}$-circuits of polynomial size.

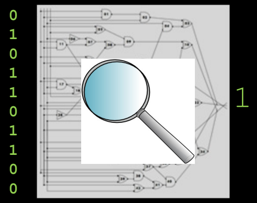
The problem of circuit lower bounds is turned into an algorithm design problem!

SAT Solving Can Help Complexity!

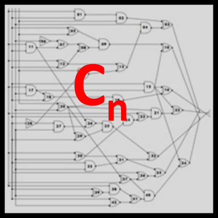
Theorem: For many natural circuit classes $\mathcal{C}$,
IF $\mathcal{C}$-SAT on poly-size circuits is in $O(2^n / n^{\log n})$ time,
THEN NEXP can't be simulated by polysize $\mathcal{C}$-circuits

Proof Plan: Assume we have two kinds of "good algorithms"

1. Slightly faster $\mathcal{C}$-SAT algorithm



2. Every problem in NEXP has a small $\mathcal{C}$-circuit family

$\Pi \in \text{NEXP} \Rightarrow$ Π is solved by a family {  }

Use both to simulate every 2^n time algorithm in << 2^n time

False by the **time hierarchy theorem**!

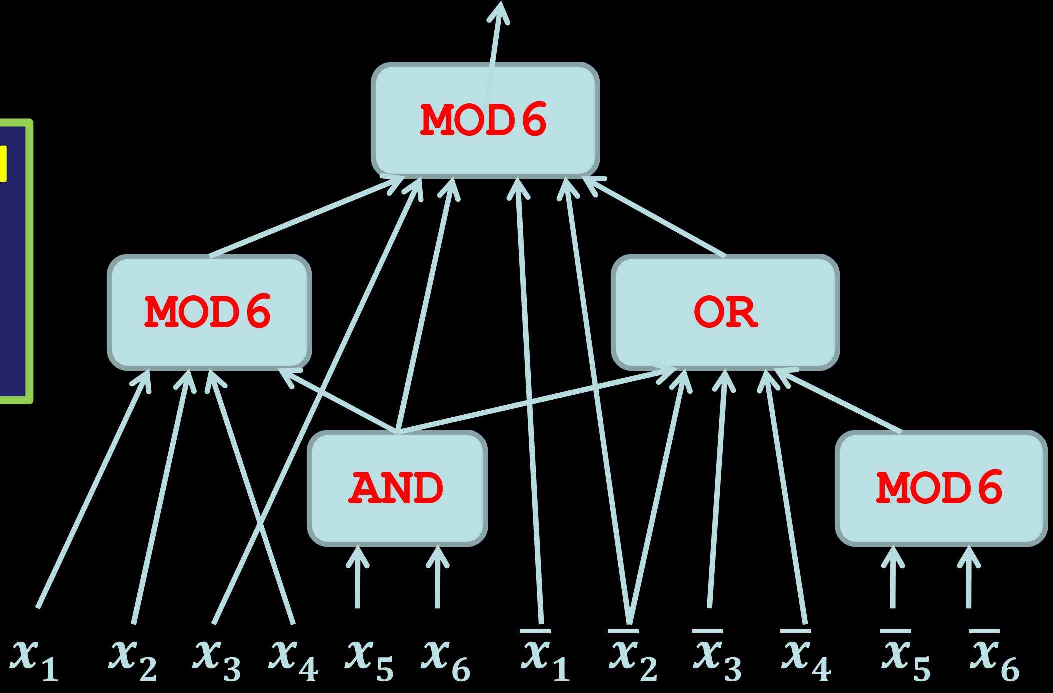
This Actually Led to a Lower Bound

Theorem [W 11] There is a function in NEXP that does not have polynomial-size ACC circuits

ACC circuit = Constant-depth with AND/OR/NOT/MODm gates

MODm($x_1, \dots, x_t$) = 1 iff $\sum_i x_i$ is divisible by m

No nontrivial limitations were known!


MOD6
MOD6
OR
AND
MOD6
x_1 x_2 x_3 x_4 x_5 x_6 $\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$ $\bar{x}_4$ $\bar{x}_5$ $\bar{x}_6$

[W '11] **ACC-SAT** is in $2^{n - n^{e}}$ time for circuits of polynomial size

A Question For The Audience

Consider a simple DPLL-style algorithm for Circuit SAT:

```
1. Simplify the circuit: Push values through NOT gates.
       – If an AND gate has a 0 input, replace AND with 0
       – If an OR gate has an 1 input, replace OR with 1
2. If the circuit = 1, then SAT; if the circuit = 0, return
3. Pick an input variable x with maximum fan-out
4. Recursively branch on (x=0) and (x=1)
```

Question **Does this algorithm run in $O(2^n/n^{10})$ time, for all circuits of size n^2?**

Are there infinitely many circuits on which it runs slowly?

A Question For The Audience

1. Simplify the circuit: Push values through NOT gates.
    – If an AND gate has a 0 input, replace AND with 0
    – If an OR gate has an 1 input, replace OR with 1
2. If the circuit = 1, then SAT; if the circuit = 0, return
3. Pick an input variable x with maximum fan-out
4. Recursively branch on (x=0) and (x=1)

Question: Maybe some modification will work?

If you can find a modification that works, you will have proved a major result in complexity theory!

My Personal Opinion

I believe that the Strong ETH is false.

(But ETH, I don't know…)

My belief is the minority opinion.

(But the chances I'll be proved wrong in my lifetime are nil!)

Even if SETH is true, my belief in the opposite led me to many ideas I'd have never found otherwise.

Maybe 30 years from now

they'll be useful in practice ☺

That's another talk…

Open Problems

- Give more evidence that **SETH** is true?
  - **Prove that ETH is equivalent to SETH?**

    [Cygan et al. CCC'12] Equivalences
  - **Prove that ¬SETH implies an unlikely collapse of complexity classes?**
- Give more evidence that **SETH** is false? ☺
- What is "really going on" between Circuit-SAT and circuit complexity?

(Make future talks more satisfying?)

Thank you!