Algorithms as Lower Bounds

Lecture 3

Part 1: Solving QBF and NC1 Lower Bounds

(Joint work with Rahul Santhanam, U. Edinburgh)

Part 2: Time-Space Tradeoffs for SAT

Ryan Williams
Stanford University

Quantified Boolean Formulas (QBF)

A Quantified Boolean Formula has the form:

$$\phi = (Q_1\, x_1) \cdots (Q_n\, x_n)\psi(x_1, \dots, x_n)$$

where $Q_i \in \{\exists, \forall\}$, $x_1, \dots, x_n$ are Boolean variables, and ψ is a Boolean formula over $x_1, \dots, x_n$

Typically ψ is a CNF, i.e., an AND of ORs of literals

The QBF Problem

Given: Quantified Boolean Formula ϕ

Determine: Is ϕ true?

Canonical PSPACE-complete problem

Simple Example

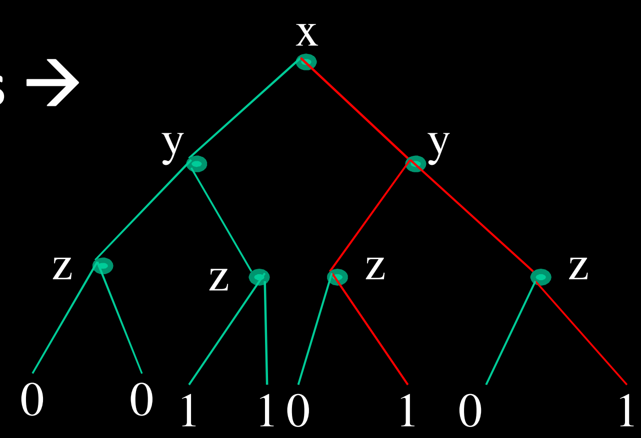
The QBF Problem

Given: Quantified Boolean Formula ϕ

Determine: Is ϕ true?

$$\boldsymbol{\phi} = \exists \boldsymbol{x} \, \forall \boldsymbol{y} \, \exists \boldsymbol{z} \, (\boldsymbol{x} \vee \boldsymbol{y}) \wedge (\neg \boldsymbol{x} \vee \boldsymbol{z})$$

Tree of variable assignments →



ϕ is a YES instance

Quantifier Blocks

Given a quantified Boolean formula:

$$\phi = (Q_1\ x_1) \cdots (Q_n\ x_n) \psi(x_1, \dots, x_n)$$

where $Q_i \in \{\exists, \forall\}$, $x_1, \dots, x_n$ are Boolean variables,

*The **number of quantifier blocks** in ϕ*
= 1 + (number of alternations in ϕ)
= 1 + (number of i such that $Q_i \neq Q_{i+1}$)

Examples:
$(\exists\ x_1) \cdots (\exists\ x_n) \psi(x_1, \dots, x_n)$ *has 1 quantifier block*
$(\exists\ x_1)(\forall\ x_2)(\forall\ x_3)(\exists\ x_4) \psi(x_1, \dots, x_4)$ *has 3 quantifier blocks*

QBF with k quantifier blocks
= Canonical $\Sigma_k P$-complete problem

(Worst-Case) Algorithms for QBF

For QBFs of size m with n Boolean variables:
Exhaustive search takes $2^n \cdot poly(m)$ time

Can we do better than this? If so, how much better?
Which cases of QBF are easy wrt *time complexity*?

[W'02] QB CNF formulas with m clauses in $O(1.71^m)$ time
When $m \gg n$, this yields no improvement over exhaustive search

[S'10] QB CNF formulas with n variables in $2^{n-\Omega\left(\frac{n}{\log n}\right)}$ time
when each variable appears O(1) times in the CNF
Again only useful when $m = O(n)$

[CIP'10] **Strong ETH $\Longrightarrow$ QB 3-CNF formulas with *two* quantifier blocks cannot be solved in $O(2^{\delta n})$ time, for all $\delta < 1$**

OPEN: QBFs over 3-CNFs with *two* quantifier blocks in $o(2^n)$ time??

QBF Algorithms Beating Brute Force

Thm 1 [SW'15] Quantified CNFs with $poly(n)$ clauses, n variables, and q quantifier blocks are solvable with zero error in

$$2^{n-n^{1/(q+1)}} \cdot poly(n) \text{ time}$$

Beats exhaustive search when q << log n/(loglog n)

Thm 2 [SW'15] QBFs of $poly(n)$ size, n variables, and q quantifier blocks can be solved in $2^{n-\Omega(q)} \cdot poly(n)$ time
Beats exhaustive search when q is large, e.g. q >> log n
Counterintuitive!
Problem gets *easier as quantifier blocks increase!*

What about when $q = \Theta(\log n)$?

"The Log-Quantifier Barrier"

Thm 4 **Suppose QBF on CNFs with n variables, $poly(n)$ size and $O(\log n)$ quantifier blocks is solvable in $2^n/n^{\log n}$ time …**

… Then, NEXP does not have non-uniform $poly(n)$-size $O(\log n)$-depth circuits!

Proof Sketch

1. Give a very tight reduction from:

SAT for *arbitrary* Boolean formulas to

QBF on CNF formulas with $O(\log n)$ quantifiers

2. Appeal to the fact that faster Formula-SAT algorithms imply circuit lower bounds.

Why is QBF hard to solve?

Compare with the two major approaches to SAT solving.

- **DPLL/Branching Algorithms**
  Explore tree of possible variable assignments by cleverly choosing variables to assign values to, "prune" the tree aggressively

Power of these algorithms comes from being able to choose any variable to branch on. For QBFs, this choice is much more restricted, due to the quantifiers

- **Local Search**
  Perform local search of solution space for a satisfying assignment

For QBFs, the "solutions" are not polynomial-size any more (unless PSPACE=NP), so local search of solution space becomes infeasible

This holds in practice as well – QBF is still considered intractable

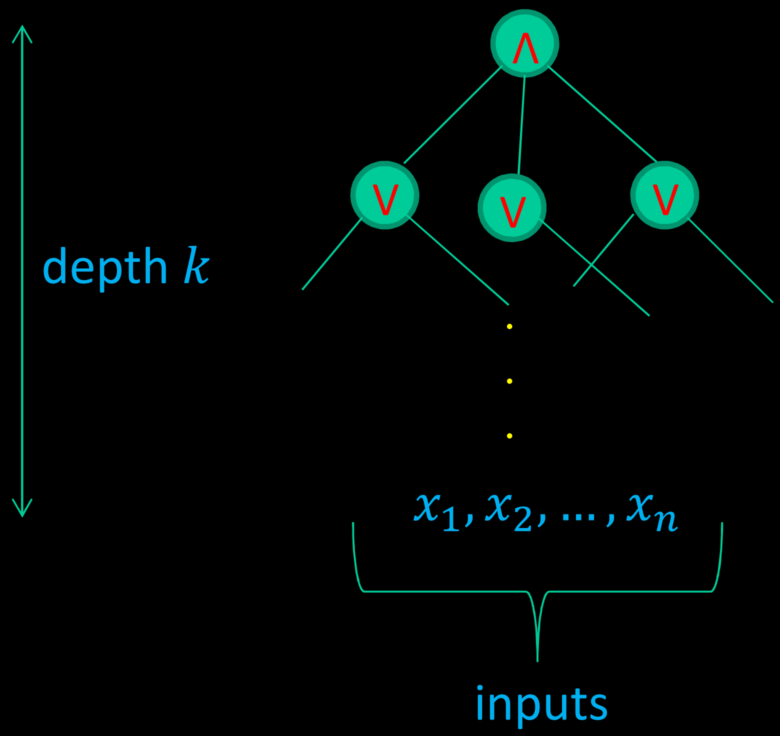
Reminder of Theorem 2

Thm 2. QBF on CNFs with $poly(n)$ clauses, n variables, and q quantifier blocks are solvable with a randomized algorithm in $2^{n-n^{1/(q+1)}} \cdot poly(n)$ time

Proof Idea: Think in terms of circuit complexity!

- Convert part of the quantified CNF into a low-depth circuit
- Evaluate the low-depth circuit on all its inputs quickly
- Use brute-force to patch the results together

Conversion to AC0



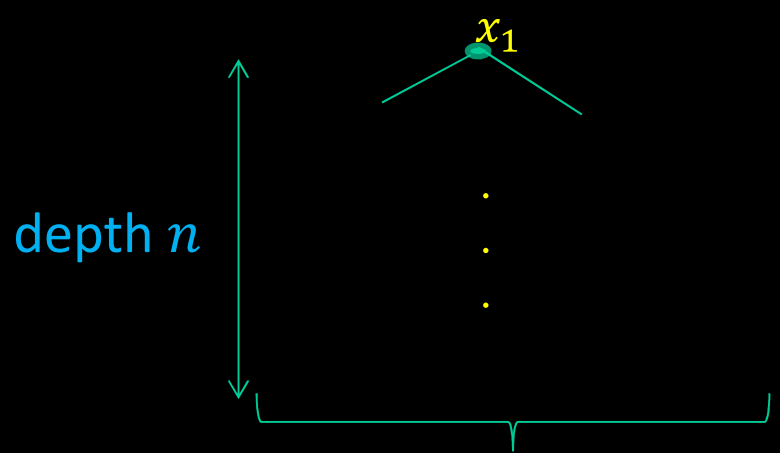
Simple Observation:
A quantified CNF with
m clauses, n' vars, q quantifier blocks
is equivalent to
evaluating an AC0 circuit of
depth $q + 2$ and size $2^{n'}m$

Low-depth AC circuits have many known limitations… can one algorithmically exploit this?

Sketch of QBF Algorithm

QBF $\phi = (Q_1\, x_1) \cdots (Q_n\, x_n) \psi(x_1, \ldots, x_n)$

Consider the tree of all possible assignments to $x_1, \ldots, x_n$



Leaves: Evaluations of CNF $\boldsymbol{\psi}$ over all 2^n variable assignments

Sketch of QBF Algorithm

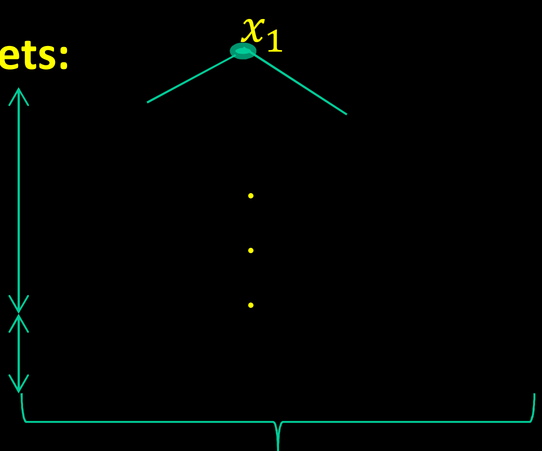
QBF $\phi = (Q_1\, x_1) \cdots (Q_n\, x_n)\psi(x_1, \dots, x_n)$

Consider the tree of all possible assignments to $x_1, \dots, x_n$

Divide vars into two sets:

|X'| = $\boldsymbol{n - \ell}$

|X''| = ℓ



Leaves: Evaluations of CNF $\boldsymbol{\psi}$ over all 2^n variable assignments

Define an AC0 circuit C(X')

$C(a_1, \dots, a_{n-\ell})$
$\equiv (Q_{n-\ell+1} x_{n-\ell+1}) \dots (Q_n x_n)$
$\psi(a_1, \dots, a_{n-\ell}, x_{n-\ell+1}, \dots, x_n)$

Number of inputs to $\boldsymbol{C}$ is $\boldsymbol{n - \ell}$

Depth of $\boldsymbol{C}$ $\leq \boldsymbol{q + 2}$

Size of $\boldsymbol{C}$ $\leq \boldsymbol{2^{\ell}\, poly(n)}$

Suppose we can evaluate C on all $2^{n-\ell}$ possible inputs, in $2^{n-\ell} \cdot poly(n)$ time.

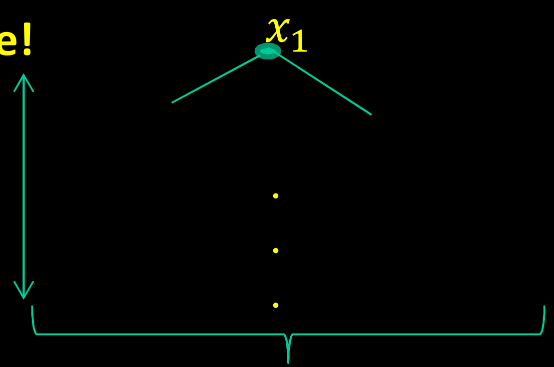
Sketch of QBF Algorithm

QBF $\phi = (Q_1\, x_1) \cdots (Q_n\, x_n) \psi(x_1, \ldots, x_n)$

Consider the tree of all possible assignments to $x_1, \ldots, x_n$

Make a shallower tree!

|X'| = $n - \ell$



Leaves: Evaluations of **circuit C** over all $2^{n-\ell}$ assignments

Goal: Pick parameter ℓ to minimize this runtime

Now solve the QBF ϕ by brute force over all $2^{n-\ell}$ assignments to X', using the computed truth table for C.

Takes $2^{n-\ell} \cdot poly(n)$ time.

Suppose we can evaluate C on all $2^{n-\ell}$ possible inputs, in $2^{n-\ell} \cdot poly(n)$ time.

Evaluating an AC0 Circuit On All Possible Inputs

Thm [IMP12] SAT of AC circuits of depth d and size m can be solved in $\mathbf{2}^{\boldsymbol{n}-\frac{\boldsymbol{n}}{\boldsymbol{log}^{\boldsymbol{d}-\mathbf{1}}(\boldsymbol{m})}}\boldsymbol{poly}(\boldsymbol{m})$ **time.**

In fact, we can evaluate depth-d circuits of size m on all 2^n possible inputs in the same runtime (with an additional $2^n \cdot poly(n)$ factor)

Use this algorithm and set $\ell = n^{\frac{1}{q+2}}$ in the previous slide.

Yields an $\mathbf{2}^{\boldsymbol{n}-\boldsymbol{n}^{\mathbf{1}/(\boldsymbol{q}+\mathbf{2})}} \cdot \boldsymbol{poly}(\boldsymbol{n})$ time algorithm!

The case of many quantifiers

Thm 2. QBFs of $poly(n)$ size, n variables, and q quantifier blocks can be solved with in $2^{n-\Omega(q)} \cdot poly(n)$ time

Beats exhaustive search when q is LARGE, e.g. q >> log n

Proof Idea: Random exhaustive search!

- "Game tree evaluation" of **[Snir85]** and **[Saks-Wigderson86]**
- *Randomly* choose 0-1 values for variables in their quantifier order, plug in these values, recurse.
  On an **existential** variable, if first choice returns *true* then can return *true* – already found a "good" choice!
  On a **universal** variable, if first choice returns *false* then can return *false* – already found a "bad" choice!
- **Saves Ω(1) bits of guessing on average,
  for every two consecutive quantifier blocks!**

Some questions to think about

QBF is easier when:
- there are << $\log n$ / $\log \log n$ quantifier blocks
- there are >> $\log n$ quantifier blocks

How much easier can it get?
(Strong ETH only rules out 1.999^n time algorithms)

Solve QBF faster than brute force when the number of quantifier blocks is $o(\log n)$?

Evaluate large Boolean formulas on all possible inputs, like we can for AC circuits?

Time-Space Lower Bounds for SAT

A Crash Course

Introduction

How efficiently can one solve
NP complete problems?

P vs NP **is currently *far* out of reach**

But important and related questions may not be

Progress on Weaker Questions

There *has* been progress on the problem:

Is LOGSPACE = NP?

Are there algorithms for SAT that treat the input as read-only and use only O(log n) additional workspace?

We believe the answer is NO!

LOGSPACE $\subseteq$ P $\subseteq$ NP
so LOGSPACE $\neq$ NP is necessary for P $\neq$ NP

LOGSPACE vs NP

LOGSPACE $\neq$ NP
$\Longleftrightarrow$ $\forall$ k, SAT cannot be solved by an algorithm using n^k time and O(log n) space

Theorem [W' 07]
SAT can't be solved by an algorithm using $n^{2\cos(\pi/7) - o(1)}$ time and $n^{o(1)}$ space
1.801

Theorem holds for robust computational models (Pointer machines, Random access machines, etc.)

Builds on work of Fortnow, Lipton, Viglas, Van Melkebeek

Some Time-Space Lower Bounds

[W'07] SAT can't be in $n^{2\cos(\pi/7)}$ time and $n^{o(1)}$ space

[W'10] SAT can't be in $n^{1.3}$ time on offline one-tape TMs

[W'10] QBF_k can't be in $n^{k+1-\varepsilon_k}$ time and $n^{o(1)}$ space

[DvMW'09]
Tautologies can't be solved with nondeterminism in $n^{4^{1/3}}$ time and $n^{o(1)}$ space

Above results hold for other NP-hard problems as well

Making a Proof System

- All above lower bounds (and more) can be unified under a common formal framework that we call "alternation trading proofs"

- A search for alternation-trading proofs can be implemented on a computer by solving LPs (Leads to proofs of new lower bounds!)

- This reduction to LP can also be used to show limitations on the proof method

Outline

- Background
  - Alternating Algorithms
  - Alternation-Trading Proofs
- Examples of Time Lower Bounds
- Automating The Process

Alternating Algorithms

Deterministic Algorithms:

Exactly one possible step at any point.

x is accepted $\Leftrightarrow$ on input x, an accept state is reached

Nondeterministic Algorithms:

Multiple possible steps at any point.

x is accepted $\Leftrightarrow$ on input x, *some* sequence of steps reaches an accept state

Alternating algorithms:

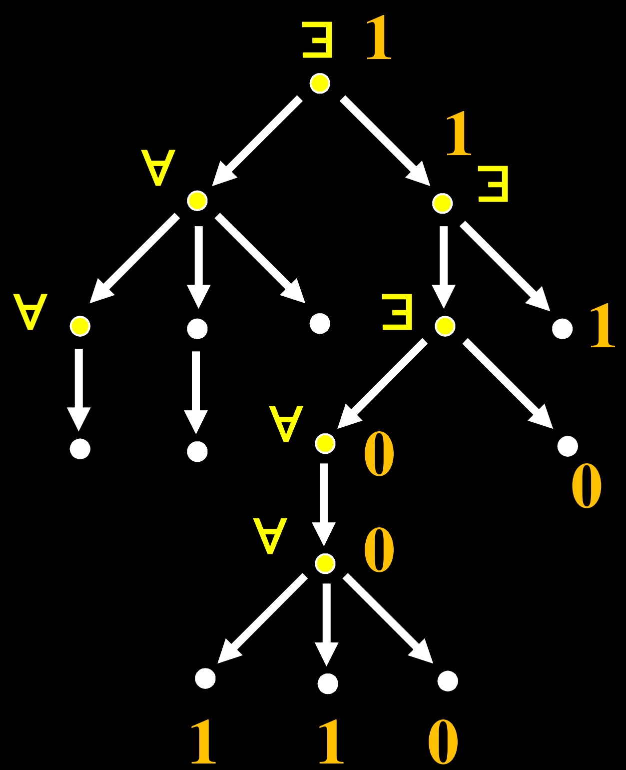
Massively parallel algorithms. Extension of nondeterminism.

Each state is classified as one of two modes:

EXISTENTIAL (*Nondeterministic*) and

UNIVERSAL (*Co-nondeterministic*)

Alternating Algorithms



Picture an alternating algorithm's computation on input **x** as a tree.

Nodes = Configurations

Edges = Steps

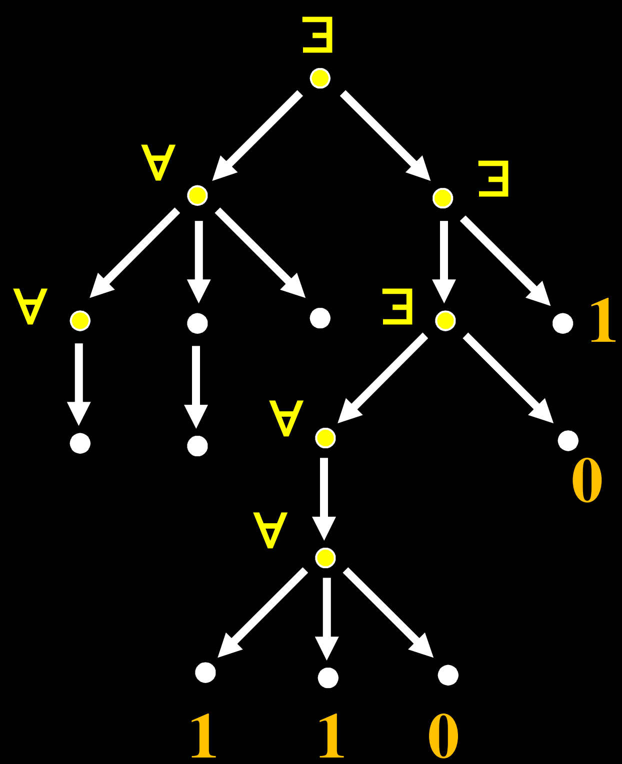
Leaves = Deterministic computations that accept (1) or reject (0)

Value of ∃-node **v** is:
1 iff *some* child of **v** is **1**.

Value of a ∀-node **v** is:
1 iff *all* children of **v** are **1**.

Input x is accepted ⇔ on input x, the *value of the root* is **1**.

Alternating Algorithms

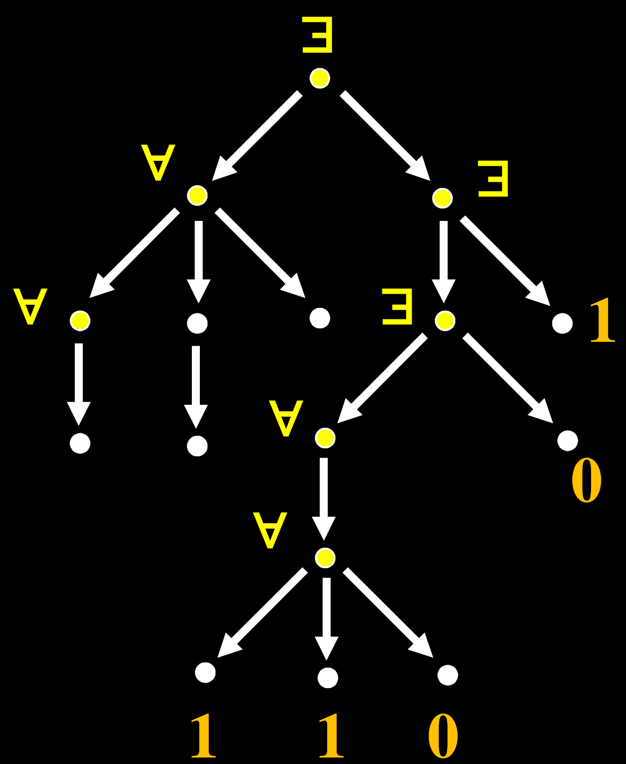


An alternating algorithm is said to run in **time t** if the **depth of the tree is t**

(Its runtime is limited only by the longest path in the tree)

Note: Completely unrealistic model of computation!
Still useful for classifying the complexity of problems.
We can use this unrealistic model as a gateway to lower bounds for **realistic** models!

Alternating Algorithms



Algorithm **makes k alternations** if the maximum number of times the mode switches on any path (from ∃ to ∀, or ∀ to ∃) is **k**

We'll call these
k-alternating algorithms

(The example is 1-alternating.)

Number of quantifier blocks = 1 + (Number of alternations)

Some Complexity Class Notation

$DTIME[n^k]$

Problems solved by **deterministic algorithms** in **n^k time.**

$DTS[n^k] = DTISP[n^k, n^{o(1)}]$

Solved by **deterministic algorithms** in **n^k time and $n^{o(1)}$ space.**

Let **C** be a complexity class. Define the **alternating complexity classes**:

($\exists$ t(n)) C

Tree has **existential paths** of length **O(t(n))** from the root, then the subtrees represent computations from class **C**

($\forall$ t(n))C

Tree has **universal paths** of length **O(t(n))** from the root, then the subtrees represent computations from class **C**

Examples:

($\exists$ n)DTIME[n] = NTIME[n], ($\forall$ n)DTIME[n] = coNTIME[n]

($\exists$ n)($\forall$ n)DTIME[n] = Σ_2 TIME[n]

Outline

- Background
  - Alternating Algorithms
  - Alternation-Trading Proofs
- Examples of Time Lower Bounds
- Automating The Process

Alternation-Trading Proofs

An *alternation-trading proof* of SAT $\notin$ DTS[n^c] works by:

1. Proving NTIME[n] $\not\subset$ DTS[$n^{c+o(1)}$]
2. Proving SAT $\in$ DTS[n^c] $\Longrightarrow$ NTIME[n] $\subseteq$ DTS[$n^{c+o(1)}$]

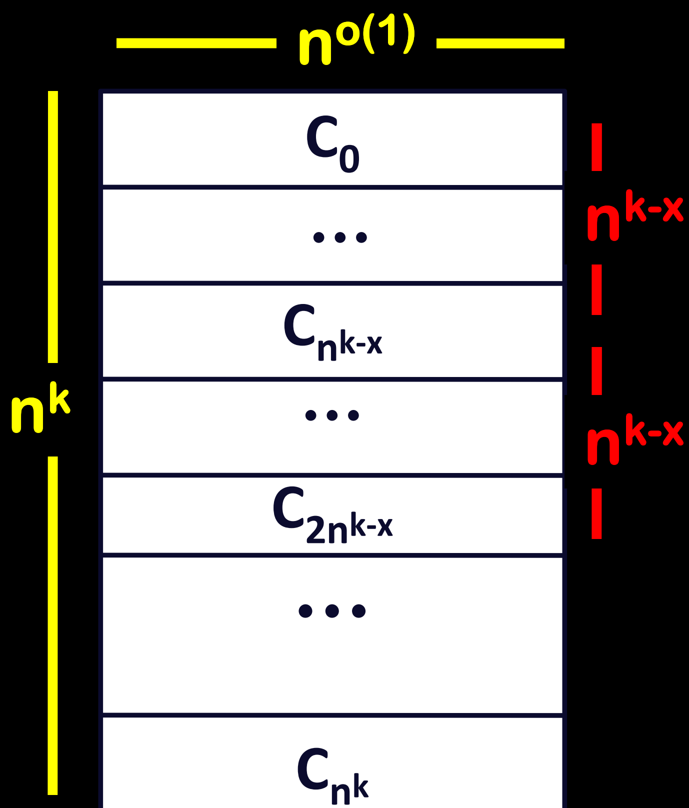
1. Prove NTIME[n] $\not\subset$ DTS[$n^{c+o(1)}$] by assuming the opposite, and apply three rules to derive a contradiction (e.g. prove NTIME[t] $\subseteq$ NTIME[$t^{1-\varepsilon}$], contradict NTIME hierarchy)

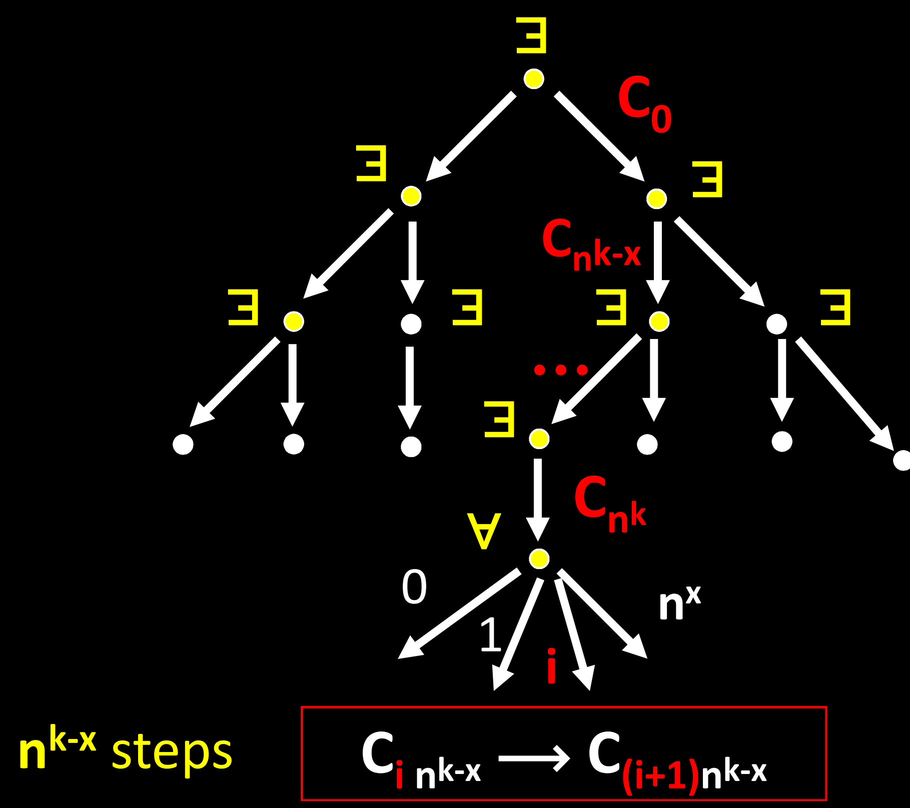
- **Speedup Lemma [Kannan, Fortnow-van Melkebeek]:**
  $\mathrm{DTS}[n^k] \subseteq (\exists\, n^{x+o(1)})\, (\forall \log n)\, \mathrm{DTS}[n^{k-x}]$ $\quad 1 \le x \le k$
  $\mathrm{DTS}[n^k] \subseteq (\forall\, n^{x+o(1)})\, (\exists \log n)\, \mathrm{DTS}[n^{k-x}]$
- **Slowdown Lemma:** If NTIME[n] $\subseteq$ DTS[$n^{c+o(1)}$] , then
  $\ldots (\exists\, n^{a1})\, (\forall\, n^{a2})\, \mathrm{DTS}[n^{a3}] \subseteq \ldots (\exists\, n^{a1})\, \mathrm{DTS}[n^{\max\{c\,a2,\, c\,a3\}}]$
  $\ldots (\forall\, n^{a1})\, (\exists\, n^{a2})\, \mathrm{DTS}[n^{a3}] \subseteq \ldots (\forall\, n^{a1})\, \mathrm{DTS}[n^{\max\{c\,a2,\, c\,a3\}}]$
- **Combination:**
  $(\exists\, n^a)(\exists\, n^b)\mathrm{DTS}[n^d] \subseteq (\exists\, n^{\max\{a,b\}})\, \mathrm{DTS}[n^d]$
  $(\forall\, n^a)(\forall\, n^b)\mathrm{DTS}[n^d] \subseteq (\forall\, n^{\max\{a,b\}})\, \mathrm{DTS}[n^d]$

Speedup Lemma

For all $1 \leq x \leq k$, $DTS[n^k] \subseteq (\exists\ n^{x+o(1)})\ (\forall \log n)\ DTS[n^{k-x}]$

"Every n^k time, $n^{o(1)}$ space algorithm can be simulated by a 1-alternating algorithm that $\exists$-guesses $n^{x+o(1)}$ bits, $\forall$-guesses $O(\log n)$ bits, then runs in n^{k-x} time, $n^{o(1)}$ space"





Slowdown Lemma

If NTIME[n] $\subseteq$ DTS[n^c], then

… ($\exists$ n^{a1}) ($\forall$ n^{a2}) DTS[n^{a3}] $\subseteq$ … ($\exists$ n^{a1}) DTS[$n^{c \max\{a2, a3\}}$]

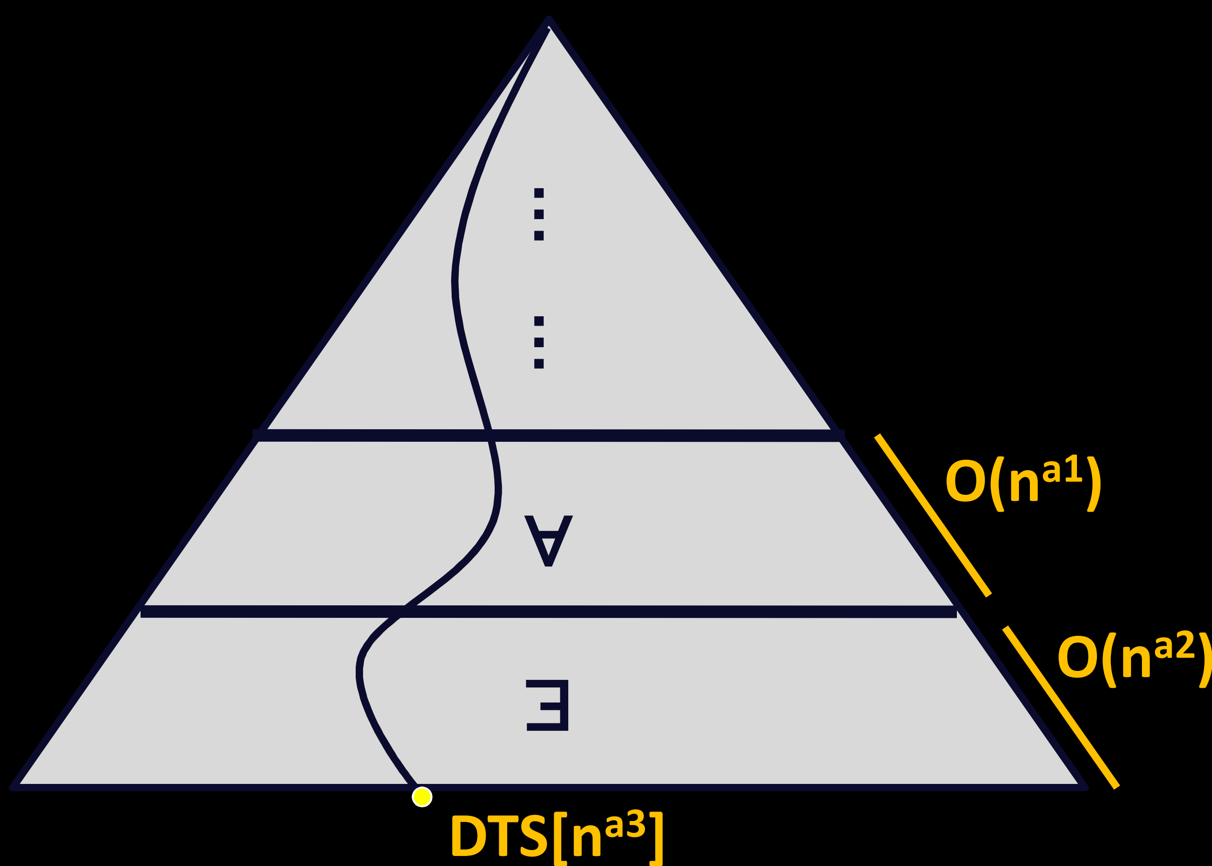
"If SAT has an n^c time algorithm, then can remove an alternation from any alternating algorithm, at a time cost of c."

Slowdown Lemma

If NTIME[n] ⊆ DTS[n^c], then

… (∃ n^{a1}) (∀ n^{a2}) DTS[n^{a3}] ⊆ … (∃ n^{a1}) DTS[$n^{c \max\{a2, a3\}}$]

"If SAT has an n^c time algorithm, then can remove an alternation from any alternating algorithm, at a time cost of c."

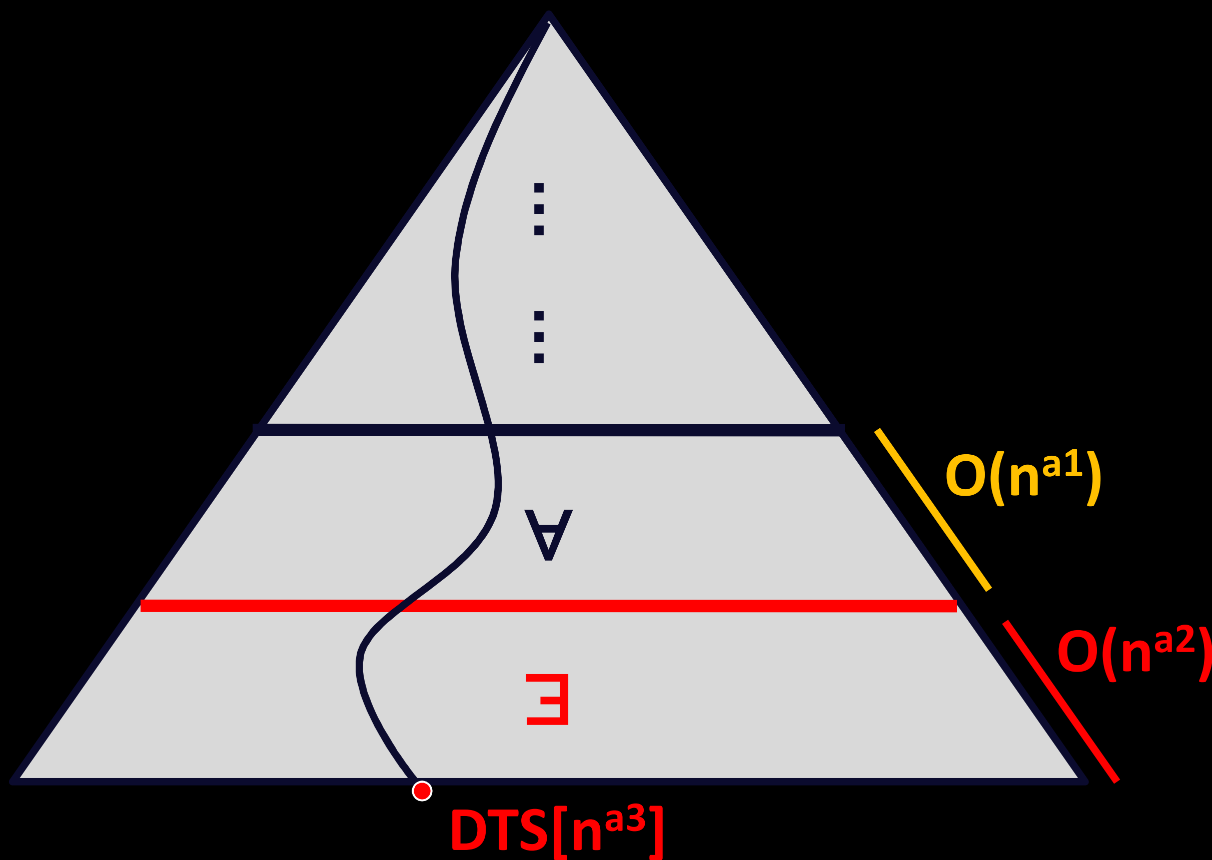


Slowdown Lemma

If NTIME[n] $\subseteq$ DTS[n^c], then

… ($\exists$ n^{a1}) ($\forall$ n^{a2}) DTS[n^{a3}] $\subseteq$ … ($\exists$ n^{a1}) DTS[$n^{c\max\{a2, a3\}}$]

"If SAT has an n^c time algorithm, then can remove an alternation from any alternating algorithm, at a time cost of c."



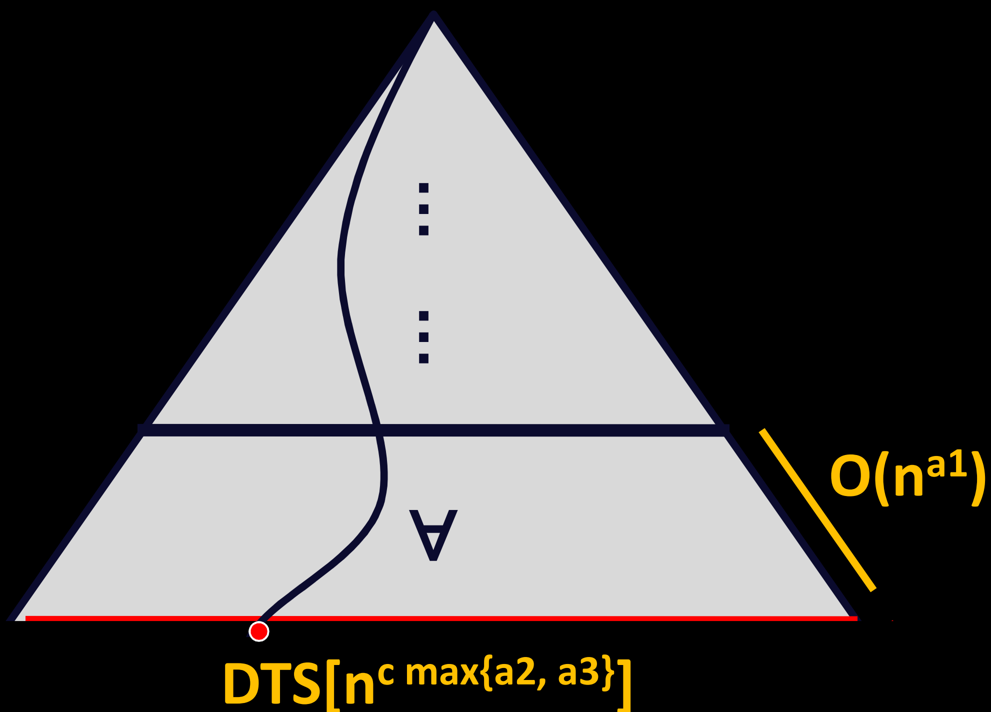
Non-det. computation running in $n^{\max\{a2, a3\}}$ time

Slowdown Lemma

If NTIME[n] $\subseteq$ DTS[n^c], then

… ($\exists$ n^{a1}) ($\forall$ n^{a2}) DTS[n^{a3}] $\subseteq$ … ($\exists$ n^{a1}) DTS[$n^{c \max\{a2, a3\}}$]

"If SAT has an n^c time algorithm, then can remove an alternation from any alternating algorithm, at a time cost of c."



By hypothesis, can replace nondet. $n^{\max\{a2, a3\}}$ time w/ DTS[$n^{c \max\{a2, a3\}}$]

Outline

- Background
  - Alternating Algorithms
  - Alternation-Trading Proofs
- Examples of Time Lower Bounds
- Automating The Process

Example 1

Time-Space Lower Bound for SAT

Alternation-Trading Proof that SAT $\notin$ DTS[$n^{2^{1/2}-\varepsilon}$]
[Lipton-Viglas'99]

Suppose **SAT $\in$ DTS[n^c]**. Then **NTIME[n] $\subseteq$ DTS[$n^{c+o(1)}$]**, and

NTIME[n^2]	$\subseteq$ DTS[$n^{2c+o(1)}$]	**(Slowdown)**
	$\subseteq$ ($\exists$ $n^{c+o(1)}$)($\forall$ log n) DTS[n^c]	**(Speedup, x=c)**
	$\subseteq$ ($\exists$ $n^{c+o(1)}$)DTS[$n^{c^2+o(1)}$]	**(Slowdown)**
	$\subseteq$ NTIME[$n^{c^2+o(1)}$]	

Contradiction to nondeterministic time hierarchy, when **$c < 2^{1/2}$**

Each class in the above list of inclusions is a ***line*** in the proof

Example 2

Alternation-Trading Proof that SAT $\notin$ DTS$[n^{1.6}]$

Suppose SAT $\in$ DTS$[n^c]$. Then NTIME$[n] \subseteq$ DTS$[n^{c+o(1)}]$, and

NTIME$[n^{c/2+2/c}] \subseteq$ DTS$[n^{c^2/2+2}]$	(Slowdown)
$\subseteq (\exists\ n^{c^2/2})(\forall\ \log n)DTS[n^2]$	(Speedup, $x=c^2/2$)
$\subseteq (\exists\ n^{c^2/2})(\forall\ \log n)(\forall\ n)(E\ \log n)DTS[n]$	(Speedup, $x=1$)
$\subseteq (\exists\ n^{c^2/2})(\forall\ n)(E\ \log n)DTS[n]$	(Combination)
$\subseteq (\exists\ n^{c^2/2})(\forall\ n)DTS[n^c]$	(Slowdown)
$\subseteq (\exists\ n^{c^2/2})DTS[n^{c^2}]$	(Slowdown)
$\subseteq (\exists\ n^{c^2/2})(\exists\ n^{c^2/2})(\forall\ \log n)DTS[n^{c^2/2}]$	(Speedup, $x=c^2/2$)
$\subseteq (\exists\ n^{c^2/2})\ (\forall\ \log n)DTS[n^{c^2/2}]$	(Combination)
$\subseteq (\exists\ n^{c^2/2})DTS[n^{c^3/2}]$	(Slowdown)
$\subseteq$ NTIME$[n^{c^3/2}]$	

Contradiction when $c^3/2 < c/2 + 2/c$.

Outline

- Background
  - Alternating Algorithms
  - Alternation-Trading Proofs
- Examples of Time Lower Bounds
- Automating The Process

Automating The Search for Proofs

Alternation-Trading Proofs apply simple **rules** at every step.

Define a **proof annotation** to be a **sequence of proof rules**.
E.g.: "Slowdown Speedup Slowdown"

Question: Suppose we fix a proof annotation. What can we say about the best lower bound proofs that follow the rules of this annotation?

THEOREM: Given a proof annotation of r rules, the best possible lower bound proof following that annotation can be determined (up to d digits of precision) in $poly(r, d)$ time.

Automating the Process

THEOREM: Given a proof annotation of **r** rules, the best possible lower bound proof following the annotation can be determined (up to **d** digits of precision) in ***poly(r,d)*** time.

STAGES OF THE PROOF:

1. **“Normalize” proofs so they have a common format**
   (E.g. Proof begins with **NTIME[n^k]** and ends with **NTIME[$n^{k-\varepsilon}$]** where **k** is a parameter to be determined… technical reduction)

2. **Create a linear programming instance with variables $a_{i, j}$**
   The $a_{i, j}$’s encode the complexity class on the **i**th line
   ($a_{1, j}$’s encode **NTIME[n^k]**, $a_{L, j}$’s encode **NTIME[$n^{k-\varepsilon}$]**)
   For the **i**th rule in the annotation, have linear constraints between $a_{i-1, j}$’s and $a_{i, j}$’s which encode an application of that rule.

3. **Repeatedly solve the LP for *fixed* lower bound exponents.**

Example of Linear Programming Reduction

Alternation-Trading Proof that $SAT \notin DTS[n^{2^{1/2}-\varepsilon}]$

$$\begin{aligned} NTIME[n^2] &\subseteq DTS[n^{2c}] \\ &\subseteq (\exists\ n^{c+o(1)})(\forall\ \log n)\ DTS[n^c] \\ &\subseteq (\exists\ n^{c+o(1)}) DTS[n^{c^2}] \\ &\subseteq NTIME[n^{c^2}] \end{aligned}$$

Example of Linear Programming Reduction

Alternation-Trading Proof that $SAT \notin DTS[n^{2^{1/2}-\varepsilon}]$

$$NTIME[n^a] \subseteq DTS[n^{ca}]$$
$$\subseteq (\exists\, n^x)(\forall \log n)\, DTS[n^{ca - x}]$$
$$\subseteq (\exists\, n^x) DTS[n^{\max\{c(ca - x),\, cx\}}]$$
$$\subseteq NTIME[n^{\max\{c(ca - x),\, cx\}}]$$

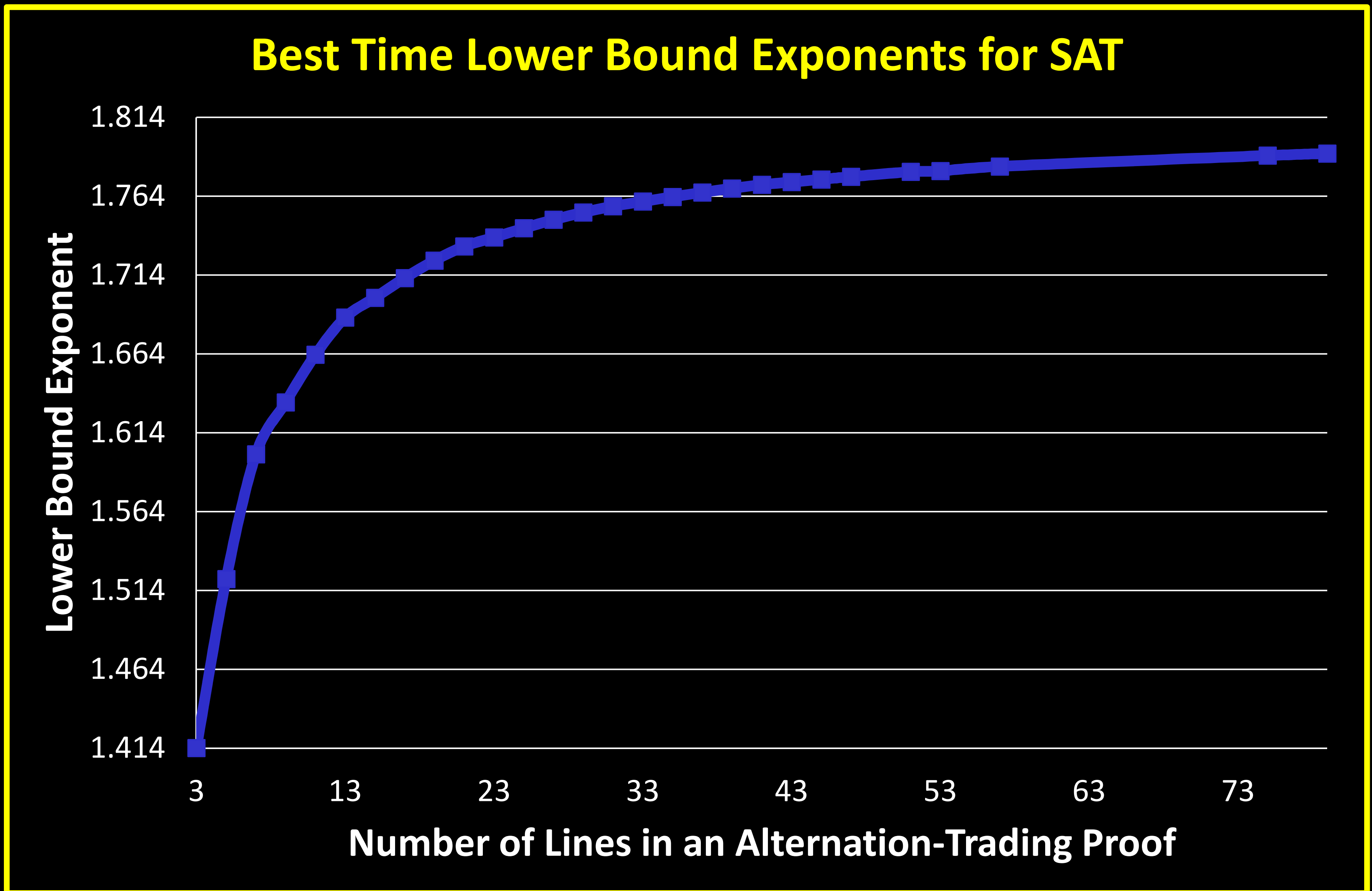
LP Constraints:

a	$>$	$c(ca - x)$
a	$>$	cx
a	$\geq$	1
x	$\geq$	1
$ca - x$	$\geq$	1

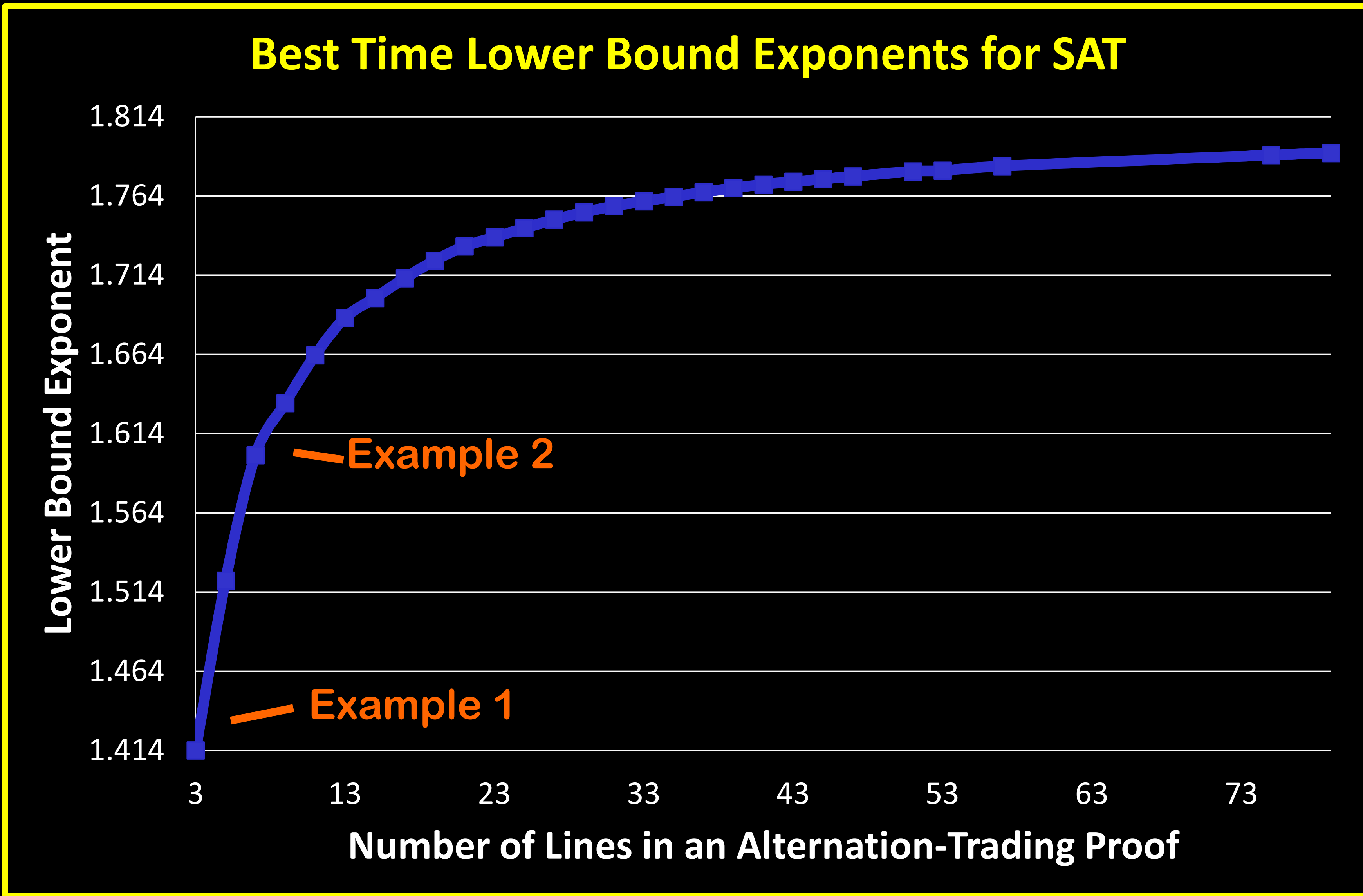
When $c < 2^{1/2}$, LP is feasible

When $c \geq 2^{1/2}$ LP is not feasible

Some Experimental Results



Some Experimental Results



Observations on the Experiments

- **The best proof annotations follow a regular pattern**
- **Following the pattern, a 424-line proof annotation yielded:**
  $n^{1.8017}$ time, $n^{o(1)}$ space lower bound

This very nearly matches the $2\cos(\pi/7) \sim 1.8019$ lower bound, and the proofs are virtually identical.

Conjecture: The $n^{2\cos(\pi/7)}$ time lower bound for SAT is the best possible with alternation-trading proofs.

Similar experiments performed in other settings
New proofs discovered where progress had stalled!

THEOREM: An n^2 time lower bound for SAT is not possible with alternation-trading proofs.

Idea: Observe in every such proof, at some point we have a speedup followed by a slowdown

Let P be a minimum length proof of n^2.

Show that we can remove this "speedup slowdown" from P, and the resulting proof P' is still valid (with possibly different parameter choices).

That is, we reduce the LP to a smaller one and argue that the new one is still feasible if the old one was.

Contradiction!

A New Understanding [Buss-W'12]

Theorem [Buss-W'12]:
The best known time-space lower bounds cannot be improved further within this proof system!
$2\cos(\pi/7) = 1.801\ldots$ is "optimal"

We need new ideas to push these lower bounds further!

Thank you!