# LECTURE 22

One of motivations for proof complexity (from Lecture 1): proof systems $\mathcal{P}$ can be basis for algorithm deciding satisfiability.

~~Show~~ formula $F$ unsatisfiable by finding $\mathcal{P}$-refutation of $F$.

Two questions

1. Which formulas have efficient $\mathcal{P}$-refutations?

2. How can we efficiently <u>find</u> $\mathcal{P}$-refutations?

We say that a proof system $\mathcal{P}$ is AUTOMATIZABLE if there are efficient algorithms to search for $\mathcal{P}$-refutations.

How to define?

<u>Suggestion 1</u> Say that $\mathcal{P}$ is automatizable if there is an algorithm $A_{\mathcal{P}}$ such that given $F$, $A_{\mathcal{P}}$ runs in time polynomial in $S(F)$ [= total # literals] and outputs a $\mathcal{P}$-refutation or says "SAT".

Problematic... For this algorithm to exist, $P$ must be polynomially bounded.

This would imply $NP = co\text{-}NP$, which we don't believe.

And even if there would be such a $P$, for most proof systems we know we believe (and sometimes can prove) they are not polynomially bounded.

More reasonable to say that $A_P$ should be efficient compared to best possible $P$-refutation

Suggestion 2 Say that $P$ is automatizable if there is an algorithm $A_P$ such that given an unsatisfiable CNF formula $F$, $A_P$ runs in time $poly(S_P(F \vdash \bot))$ and outputs a $P$-refutation of $F$.

Much better... Almost the right definition. Except for the following stupid problem:

- Let $F_1(n)$ be a superhard unsat formula over $n$ variables
- Let $F_2$ be a small, constant-sized randomly sampled (or adversarially chosen) unsatisfiable formula
- Let $F_n$ be some random permutation of clauses in $F_1(n) \cup F_2$.

$F_n$ has (at least linear size)

$S_P(F_n \vdash \bot) = O(1)$ (since we can refute $F_2$)

But any algorithm would have to at least read the formula to have a chance to find $F_2$, which takes at least linear time.

However $\text{poly}(S_P(F_n \vdash \bot)) = \text{poly}(O(1)) = O(1)$.

So we need to give $A_P$ at least a chance to look at the formula...

<u>DEFINITION</u> A proof system $P$ is AUTOMATIZABLE if there is an algorithm $A_P$ such that given an unsatisfiable CNF formula $F$ it runs in time $\text{poly}(S(F) + S_P(F \vdash \bot))$ and outputs a $P$-refutation

Definition by [Bonet, Pitassi, & Raz '00]

Some other flavours of this definition:

- <u>Quasiautomatizable</u>: the running time is quasipolynomial

  [i.e., $\exp((\log n)^k)$ for some (constant) $k \in \mathbb{N}^+$]

  <u>Example</u> The "typical" quasipolynomial function is $n^{\log n}$

"Not quite polynomial, but almost"

- <u>Weakly automatizable</u>: there is some proof system $Q$ s.t. $A_P$ outputs a $Q$-refutation in time poly$(S(F) + S_P(F \vdash \bot))$ [Atserias & Bonet '04]

And a proof system can also be weakly quasiautomatizable.

Why study automatizability?

- Connection to proof search and satisfiability algorithms (where we started)
- Connections to feasible interpolation for proof systems (recall lectures 4-6 and Pavel Pudlák's guest lecture)

Note that for the definition we finally settled on, automatizability makes perfect sense even for weak proof systems where we know strong lower bounds.

Morally: The weaker the proof system

$\Downarrow$

The more likely the proof system is to be automatizable

Why?

Possible explanation by connection between automatizability and feasible interpolation

Known: $P$ automatizable $\Rightarrow$ $P$ has feasible interpolation

And feasible interpolation is antimonotone w.r.t. proof system strength

[Krajíček & Pudlák '98]
[Bonet, Pitassi, & Raz '00]
Extended Frege not automatizable under widely believed cryptographic assumptions

Also $TC^0$-Frege (Frege of bounded depth but with threshold gates - linear inequalities)

[Bonet, Domingo, Gavaldà, Maciel, & Pitassi '04]
Bounded-depth Frege not automatizable under stronger assumptions (about the hardness of factoring products of certain primes)

But we will focus on <u>resolution</u>

- Start with clauses $C \in F$
- Iteratively derive new clauses by resolution rule $\dfrac{B \vee x \quad C \vee \bar{x}}{B \vee C}$
- Stop when empty clause $\bot$ derived

Can view resolution refutation $\pi = (C_1, C_2, \ldots, C_{L-1}, C_L = \bot)$ as DAG $G_\pi$

Vertices $v_1, v_2, \ldots$ labelled by clauses
Sources are axioms $C \in F$
For non-sources, edges from clauses resolved.

$$S(\pi) = L(\pi) = \#\text{ vertices in } G_\pi$$
$$= \#\text{ clauses in } \pi$$

<u>Tree-like resolution</u>: Require that $G_\pi$ is a tree. Clauses in $\pi$ can (and generally will) repeat. Different vertices labelled by same clause. Count vertices = clauses with repetitions.

<u>Lower bounds</u>

[Iwama '97] Finding a shortest resolution refutation is NP-hard.

[Alekhnovich, Buss, Moran, & Pitassi '01]
If $P \neq NP$, then length of shortest resolution refutation cannot be approximated to within constant factor (true for general and tree-like resolution)

Set-up: Input is resolution refutation.

<u>Upper bounds</u>

Tree-like resolution is weakly quasiautomatizable
[Beame & Pitassi '01] (observed in)
based on [Ben-Sasson & Wigderson '01]

OUR GOAL FOR FINAL LECTURES
~~following~~ [Alekhnovich & Razborov '01, '08])

Prove that resolution and tree-like resolution are not automatizable under some reasonable complexity assumption.

But this assumption has to be "fairly precise" since tree-like resolution weakly quasi-automatizable.

Therefore, we need to make a detour

CRASH COURSE ON PARAMETERIZED COMPLEXITY

Often NP-problems come with a natural second parameter in addition to size

- ~~CNF~~ CIRCUIT SAT : # variables
- VERTEX COVER : size of solution (# vertices)
- COLOURING : # colours
- CLIQUE : clique size

Classify hardness wrt multiple parameters

Maybe dependence on input size $n$ nice if other parameter bounded

Examples:

- SAT with $\leq k$ variables : $2^k \text{ poly}(n)$
- VERTEX COVER of size $\leq k$ $2^k \text{ poly}(n)$

Nice dependence on $n$, "isolated" bad dependence on $k$

- k-CLIQUE hard to do better than $n^k$
  not nice dependence — "k & n mix"

- k-COLOURING NP-hard already for $k=3$
  probably exponentially hard.
  Super-not-nice!

PARAMETERIZED PROBLEM

Language $\mathcal{L} \subseteq \Sigma^* \times \mathbb{N}$

$\Sigma$ finite alphabet

For $(x,k) \in \mathcal{L}$ think of

$x$ — input

$k$ — parameter

~~First~~ systematic work (and still standard reference) [Downey & Fellows '99]

$\mathcal{L}$ is FIXED-PARAMETER TRACTABLE (FPT) if the question "$(x,k) \in \mathcal{L}$?" can be decided in time $f(k)\,\mathrm{poly}(|x|)$ where $f$ arbitrary but depends only on $k$

[ Recall $\mathrm{poly}(n) = O(n^k)$ for some fixed $k \in \mathbb{N}^+$ ]

One can also define notions of reductions and complete problems.

We will skip the details (at least for now)

Instead of the P vs. NP question, parameterized complexity has a whole hierarchy of classes.

Difficult/involved to define, and indeed even in conference presentations most often "defined" by example. So this is what we will do ~~as~~ as well ... 8-)

inclusions believed to be strict.

$$FPT \stackrel{\triangle}{=} W[0] \stackrel{\triangle}{\subseteq} W[1] \stackrel{\triangle}{\subseteq} W[2] \subseteq \dots$$

<u>FPT</u>: Vertex cover

Find set of vertices covering (intersecting) all edges in graph)

<u>W[1]-complete</u>: k-clique

Does G contain k vertices that are all neighbours with each other

<u>W[2]-complete</u>: Dominating set

Does G contain k vertices such that all others are neighbours of these?

Then there is an even larger class of problems called W[P]

Again we will "define" the class by giving an example of a complete problem

For a vector $\vec{\alpha} \in \{0,1\}^n$, let its HAMMING WEIGHT be $wt(\vec{\alpha}) = |\{ i : \alpha_i = 1 \}|$

## MONOTONE MINIMUM CIRCUIT SATISFYING ASSIGNMENT (MMCSA)

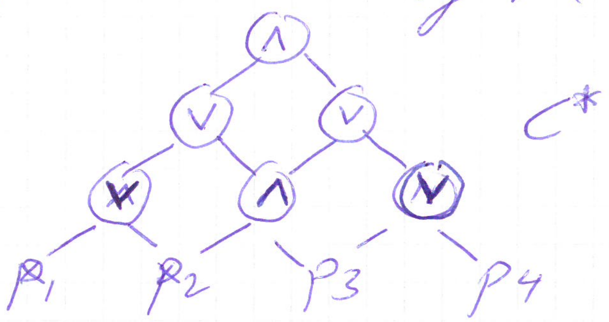
**Instance** Monotone Boolean circuit $C(p_1, \ldots, p_n)$ over $n$ variables with gates $\{\wedge, \vee\}$

**Solution** $\vec{\alpha} \in \{0,1\}^n$ s.t. $C(\vec{\alpha}) = 1$

**Objective function** $k(\vec{\alpha}) = wt(\vec{\alpha})$

Denote optimal value by $k(C)$.

**Example**




$k(C^*) =$

Minimal solution has weight $2$.

If output gate connected to at least some input, solution always has weight $\leq n$.

Since MMCSA is W[P]-complete, the intuition is that this problem should not be possible to solve efficiently with an algorithm running in FPT-style time $f(k)\, \mathrm{poly}(n)$

(Unless the parameterized complexity hierarchy is not a strict hierarchy after all.)

Yet, what [AR08] proves is the following.

THEOREM If either resolution or tree-like resolution is automatizable, then for any fixed $\varepsilon > 0$ there exists an algorithm $\Phi$ that
- takes monotone circuit $C$ as input
- runs in time $\exp(k(C)^{O(1)}) \cdot |C|^{O(1)}$
- approximates $k(C)$ to within factor $(1+\varepsilon)$

PLAN FOR REST OF TODAY

- Sketch reduction from MMCSA to resolution automatizability (simplified version)
- State some tentative lemmas
- Fail to prove them
( - So that in coming lectures we can start to patch the details to make things work)

REDUCTION LEMMA (PRELIMINARY)
There is a poly-time reduction $R$ that maps any monotone circuit $C$ and any $1^m$ to an unsatisfiable CNF formula $F(C,m)$ such that

$$\mathcal{L}_R(F(C,m) \vdash \bot) \approx m^{k(C)}$$

[Recall $1^m = \overbrace{111\ldots1}^{m}$ = $m$ encoded in unary]

So if you can estimate refutation length of $F(C,m)$ really well, then you get a good sense of what $k(C)$ is.

Then combine with the following

SELF-IMPROVEMENT PROPOSITION [ABMP'01]
For every fixed $d \in \mathbb{N}^+$ $\exists$ poly-time computable function $f_d$ which maps monotone circuits to monotone circuits in such a way that

$$k(f(C)) = (k(C))^d$$

for all $C$.

Proof Exercise.

Applying self-improvement and then the reduction should give an even better sense of what $k(C)$ is...

Let us describe the reduction
Let $C = C(p_1, \ldots, p_n)$ fixed monotone circuit in $n$ variables. (We don't get to choose it — it is fed into the reduction.

Let $A \subseteq \{0,1\}^m$ be a set of $m$-length bit vectors (which we will get to choose cleverly). Think of $\vec{a} \in A$ as column vectors.

Say that a 0/1 $m \times n$ - matrix $M$ is $\mathcal{A}$ - ADMISSIBLE if all columns of $M$ are vectors from $\mathcal{A}$ (such vectors $\vec{a} \in \mathcal{A}$ we also call admissible).

Consider the following combinatorial principle (which might be true or false depending on $C$ and $\mathcal{A}$)

<u>$(C, \mathcal{A})$ - $\exists$SAT</u>

For every 0/1 $m \times n$ $\mathcal{A}$-admissible matrix $M = (m_{ij})$ it holds that there is a row $i \in [m]$ such that
$C(m_{i,1}, \ldots, m_{i,n}) = 1$

<u>Example</u>

Consider our circuit $C^*$ above
Consider $\mathcal{A}^* = \left\{ \begin{pmatrix}1\\1\\1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\1\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\0\\0\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\\0\\0\\1\end{pmatrix} \right\}$

Then it is straightforward to verify that $(C^*, \mathcal{A}^*)$ - $\exists$SAT is true.

Let us see a more general argument why this is true.

**DEF** Let $d_1(\mathcal{A})$ be the maximal $d$ such that for every $d$ vectors $\vec{a}^{(1)}, \dots, \vec{a}^{(d)} \in \mathcal{A}$ there is a position $i \in [m]$ s.t. $\vec{a}_i^{(1)} = \dots = \vec{a}_i^{(d)} = 1$.

Alternative view:

Let $Z(\vec{a}) = \{ i \in [m] \mid \vec{a}_i = 0 \}$,

Then $d(\mathcal{A}) + 1$ is exactly the $\overset{\text{minimal}}{\text{number}}$ over such sets needed to cover $[m]$, i.e. s.t. $\bigcup_i Z(\vec{a}^{(i)}) = [m]$.

**Example** $d_1(\mathcal{A}^*) = 2$.

**LEMMA** If $k(C) \leq d_1(\mathcal{A})$, then $(C, \mathcal{A})$-$\exists$SAT is true.

**Proof** Fix a satisfying assignment $\alpha$ to $C$ of minimum weight. Consider the columns $j_1, \dots, j_{k(C)}$ s.t. $\alpha_{j_\ell} = 1$ in this assignment. Whatever vectors from $\mathcal{A}$ are chosen for these columns, there is some row $i$ s.t. $m_{i,j_1} = \dots = m_{i,j_{k(C)}} = 1$ since $k(C) \leq d_1(\mathcal{A})$. This row $i$ witnesses the truth of $(C, \mathcal{A})$-$\exists$SAT for this particular matrix $M$. $\square$

Intuitively, if $C$ and $\mathcal{A}$ are tricky enough, then we might need to check many of the $|\mathcal{A}|^{k(C)}$ possible choices of vectors to convince ourselves that we always get a row that works.

We want to code $(C, \mathcal{A})$-$\exists$SAT (~~for~~ $C, \mathcal{A}$ chosen so that the principle is true) as an unsatisfiable CNF formula, and then show that resolution has to do this exhaustive search.

GAME PLAN

1. Given $C$, find $\mathcal{A}$ so that $(C, \mathcal{A})$-$\exists$SAT is true.

2. Encode CNF formula $\tau^*(C, \mathcal{A})$ saying that $(C, \mathcal{A})$-$\exists$SAT is false.

3. Prove that refuting formulas of type $\tau^*(C, \mathcal{A})$ in resolution requires large width.

4. Hit $\tau^*(C, \mathcal{A})$ with random restriction $\rho$. If refutation $\pi : \tau^*(C, \mathcal{A})$ is short, then all wide clauses disappear in $\pi\restriction_\rho$

5. But $\tau^*(C, \mathcal{A})\restriction_\rho = \tau^*(C', \mathcal{A}')$ for some $C'$ and $\mathcal{A}'$ that require large width to refute according to 3 — contradiction.

# $\tau^*(C, \mathcal{A})$

Monotone circuit $C = C(p_1, \dots, p_n)$

- input nodes identified with $p_i$
- output node $v_{out}$

Set of vectors $\mathcal{A} \in \{0,1\}^m$

$\tau^*(C,\mathcal{A})$ should say: "there is an $\mathcal{A}$-admissible matrix $M$ for which $C(\alpha) = 0$ $\forall$ row $\alpha \in M$"

For every row $i \in [m]$, every node $v \in C$:

$z_{i,v}$ = "value of node $v$ when input is row $i$"

For every column $j \in [n]$, every $\vec{a} \in \mathcal{A}$:

$col_{j,\vec{a}}$ = "$j$th column of matrix is $\vec{a}$"

$\boxed{\text{recall "}(x \wedge y) \to z\text{" shorthand for } \bar{x} \vee \bar{y} \vee z}$

## Clauses

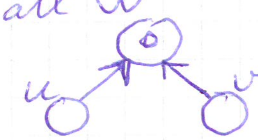
(i) $\bigvee_{\vec{a} \in \mathcal{A}} col_{j,\vec{a}}$ all $j \in [n]$

some $\vec{a}$ chosen for every column

(ii) $col_{j,\vec{a}} \to z_{i,p_j}$ all $\vec{a} \in \mathcal{A}$ and $i \in [m]$ such that $\vec{a}_i = 1$

values from row $i$ feed into circuit

(iii) $(z_{i,u} \circ z_{i,v}) \to z_{i,w}$ all $w$ $\circ \in \{\wedge, \vee\}$ $i \in [m]$




gates computed correctly on $i$th row

(iv) $\overline{z}_{i,v_{out}}$ $i \in [m]$

Output of $C$ when evaluated on $i$th row is zero

Only one problem ... XVII

Our game plan doesn't work

What should the random restriction be?

a) Hit $col_{j,\vec{a}}$ variables randomly?

Changes combinatorics of $A$ in strange ways

- Some vectors plugged into some columns
- Sum vectors forbidden in other columns

Not the same problem anymore ...

b) Hit $z_{i,v}$ variables randomly?

But if we set $z_{i,v}$ true for vertices close to $v_{out}$, circuit very likely satisfied

Plus anyway the circuit changes.

And changes to different circuits for different rows. Not the same problem anymore ...

We need a nicer encoding for $\tau^*(C, A)$ that plays better with random restrictions

(Actually it is going to be a ~~very~~ not nice encoding at all, but we shall see it next time)