Hardness magnification

Sanjana Das

18.405

May 14, 2024

What is hardness magnification?

Hardness magnification is a phenomenon where weak-looking lower bounds on certain problems imply much stronger lower bounds.

What is hardness magnification?

Hardness magnification is a phenomenon where weak-looking lower bounds on certain problems imply much stronger lower bounds.

Theorem (McKay–Murray–Williams 2019)

If there is $\varepsilon > 0$ and arbitrarily small $\beta > 0$ such that

- ⋆ MCSP[n^β] doesn't have circuits of size $n^{1+\varepsilon}$ (on inputs of length $n = 2^m$),
- ⋆ then NP $\not\subseteq$ Circuit[poly].

We believe MCSP[n^β] is 'hard,' so a lower bound of $n^{1+\varepsilon}$ looks very weak — but it would be enough to imply NP $\not\subseteq$ Circuit[poly]!

The high-level idea

We argue by contrapositive: e.g., we assume $\mathsf{NP} \subseteq \mathsf{Circuit}[\mathsf{poly}]$ and use this to get super efficient circuits for $\mathsf{MCSP}[n^{\beta}]$.

Main idea

Reduce to a problem on much smaller input size (e.g., n^{β} instead of n).

The high-level idea

We argue by contrapositive: e.g., we assume NP $\subseteq$ Circuit[poly] and use this to get super efficient circuits for MCSP[n^β].

Main idea

Reduce to a problem on much smaller input size (e.g., n^β instead of n).

- Given an oracle $\mathcal{O}$ for a specific problem, we can solve MCSP[n^β] with a circuit of size $n^{1+\beta}$ and oracle calls of size n^β.
- Under the assumption NP $\subseteq$ Circuit[poly], we can solve $\mathcal{O}$ with a decently efficient circuit (e.g., polynomial in its input length).
- The input length to $\mathcal{O}$ is so tiny that even a decently efficient circuit for $\mathcal{O}$ in terms of its input length is super efficient in terms of n.

Magnification for Search-MCSP

Definition (Search-MCSP[s])

- **Input**: $f\colon \{0,1\}^m \to \{0,1\}$, as a truth table of length $n = 2^m$.
- **Output**: a circuit of size at most s that computes f, or the all-0's string if no such circuit exists.

Magnification for Search-MCSP

Definition (Search-MCSP[s])

- **Input**: $f\colon \{0,1\}^m \to \{0,1\}$, as a truth table of length $n = 2^m$.
- **Output**: a circuit of size at most s that computes f, or the all-0's string if no such circuit exists.

Theorem (McKay–Murray–Williams 2019)

If there is $\varepsilon > 0$ and arbitrarily small $\beta > 0$ such that Search-MCSP[n^β] doesn't have circuits of size $n^{1+\varepsilon}$ and depth n^ε, then NP $\not\subseteq$ Circuit[poly].

Magnification for Search-MCSP

Definition (Search-MCSP[s])

- **Input**: $f\colon \{0,1\}^m \to \{0,1\}$, as a truth table of length $n = 2^m$.
- **Output**: a circuit of size at most s that computes f, or the all-0's string if no such circuit exists.

Theorem (McKay–Murray–Williams 2019)

If there is $\varepsilon > 0$ and arbitrarily small $\beta > 0$ such that Search-MCSP[n^β] doesn't have circuits of size $n^{1+\varepsilon}$ and depth n^ε, then NP $\not\subseteq$ Circuit[poly].

Theorem ('Contrapositive' of MMW19)

If NP $\subseteq$ Circuit[poly], then Search-MCSP[s] has circuits of size $n \cdot \mathrm{poly}(s)$ and depth poly(s). (Think of s as n^β.)

Main idea — compression using small circuits

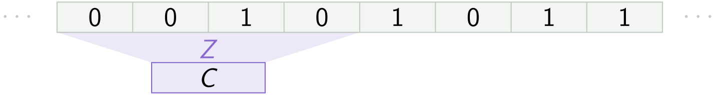
Main Idea

If our input f can be computed by a small circuit, then we can 'compress' chunks of it by representing the chunk with a small circuit.

Main idea — compression using small circuits

Main Idea

If our input f can be computed by a small circuit, then we can ‘compress’ chunks of it by representing the chunk with a small circuit.

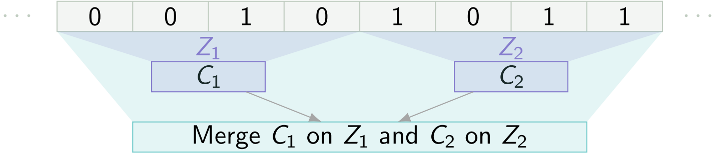


- Imagine we have some interval $Z = [z_1, z_2] \subseteq \{0,1\}^m$. Then we can store the values of f on Z by instead storing a small circuit C that computes f on Z — if no such circuit exists, we know f can't be computed by a small circuit.

Main idea — compression using small circuits

Main Idea

If our input f can be computed by a small circuit, then we can 'compress' chunks of it by representing the chunk with a small circuit.



- Imagine we have some interval $Z = [z_1, z_2] \subseteq \{0,1\}^m$. Then we can store the values of f on Z by instead storing a small circuit C that computes f on Z — if no such circuit exists, we know f can't be computed by a small circuit.
- If we've got two such circuits, to 'merge' them we only need their descriptions and the endpoints of their intervals — this is a problem with input length $\mathrm{poly}(s)$.

The smaller problem

Definition (Circuit-Merge)

- **Input**: $\langle 1^s, C_1, C_2, Z_1, Z_2, j\rangle$ where $j \in \mathbb{N}$, C_1 and C_2 are circuits of size at most s, and $Z_1, Z_2 \subseteq \{0,1\}^m$ are adjacent intervals.
- **Output**: $\langle C\rangle_j$ where C is the lexicographically first circuit C of size at most s such that $C(z) = C_1(z)$ for all $z \in Z_1$ and $C(z) = C_2(z)$ for all $z \in Z_2$, or 0 if no such circuit exists.

The smaller problem

Definition (Circuit-Merge)

- **Input**: $\langle 1^s, C_1, C_2, Z_1, Z_2, j\rangle$ where $j \in \mathbb{N}$, C_1 and C_2 are circuits of size at most s, and $Z_1, Z_2 \subseteq \{0,1\}^m$ are adjacent intervals.
- **Output**: $\langle C\rangle_j$ where C is the lexicographically first circuit C of size at most s such that $C(z) = C_1(z)$ for all $z \in Z_1$ and $C(z) = C_2(z)$ for all $z \in Z_2$, or 0 if no such circuit exists.

Claim

Circuit-Merge $\in \Sigma_3\mathrm{P}$.

Proof.

Existentially guess C. To check it works, universally guess z and check $C(z)$ is correct. To check it's lexicographically first, universally guess $C' \prec C$, then existentially guess z' and check $C'(z')$ is *incorrect*. □

An oracle circuit

Lemma

There is a Circuit-Merge-oracle circuit for Search-MCSP[s] with size $n \cdot \mathrm{poly}(s)$, queries of length $\mathrm{poly}(s)$, and depth $\log n$.

An oracle circuit

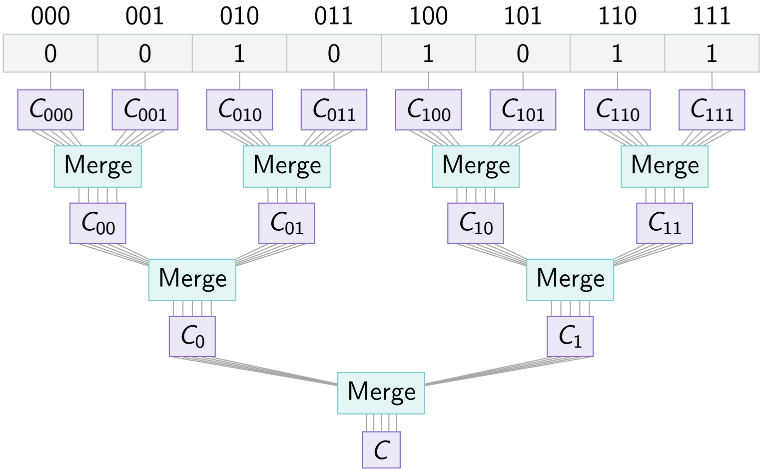
Lemma

There is a Circuit-Merge-oracle circuit for Search-MCSP[s] with size $n \cdot \mathrm{poly}(s)$, queries of length $\mathrm{poly}(s)$, and depth $\log n$.

Proof.

- We start out with one circuit representing each bit of the input (e.g., the all-0's or all-1's circuit).
- Then we repeatedly merge two consecutive circuits (in a binary tree) to get circuits representing larger and larger chunks of the input.
- If we ever get 'stuck' (i.e., there's no small circuit merging C_1 and C_2), then we know f itself doesn't have a small circuit.
- Otherwise, we eventually get a small circuit representing f. $\square$

An oracle circuit



(We hardcode 1^s and all the values of Z_1 and Z_2.)

Finishing the proof

Theorem ('Contrapositive' of MMW19)

If NP $\subseteq$ Circuit[poly], then Search-MCSP[s] has circuits of size $n \cdot \mathrm{poly}(s)$ and depth $\mathrm{poly}(s)$.

Proof.

- We've made a Circuit-Merge-oracle circuit of size $n \cdot \mathrm{poly}(s)$ and depth $\log n$, making queries of length $\ell = \mathrm{poly}(s)$.
- If NP $\subseteq$ Circuit[poly], then Σ_3P $\subseteq$ Circuit[poly].
- So we can implement all the oracle queries with circuits of size $\mathrm{poly}(\ell) = \mathrm{poly}(s)$ to get an actual circuit for Search-MCSP[s] of size $n \cdot \mathrm{poly}(s)$ and depth $\mathrm{poly}(s)$. □

Magnification for more general languages

Question

What's special about MCSP[n^β]?

Magnification for more general languages

Question

What's special about MCSP[n^β]?

- One property of MCSP[n^β] is that it's sparse — MCSP[s] only has $2^{s \log s}$ **YES** instances (out of 2^n possible inputs).
- Surprisingly, this is enough to get a hardness magnification result!

Magnification for more general languages

Question

What's special about MCSP$[n^\beta]$?

- One property of MCSP$[n^\beta]$ is that it's sparse — MCSP$[s]$ only has $2^{s \log s}$ **YES** instances (out of 2^n possible inputs).
- Surprisingly, this is enough to get a hardness magnification result!

Theorem (Chen–Jin–Williams 2019)

Suppose there is some $\varepsilon > 0$ and a family of languages $\{L_\beta\} \subseteq$ NP (with arbitrarily small values of β) such that each L_β is 2^{n^β}-sparse and doesn't have circuits of size $n^{1+\varepsilon}$. Then NP $\not\subseteq$ Circuit$[n^k]$ for all k.

Magnification for more general languages

Question

What's special about MCSP[n^β]?

- One property of MCSP[n^β] is that it's sparse — MCSP[s] only has $2^{s \log s}$ **YES** instances (out of 2^n possible inputs).
- Surprisingly, this is enough to get a hardness magnification result!

Theorem (Chen–Jin–Williams 2019)

Suppose there is some $\varepsilon > 0$ and a family of languages $\{L_\beta\} \subseteq \mathrm{NP}$ (with arbitrarily small values of β) such that each L_β is 2^{n^β}-sparse and doesn't have circuits of size $n^{1+\varepsilon}$. Then $\mathrm{NP} \not\subseteq \mathrm{Circuit}[n^k]$ for all k.

Theorem (Contrapositive of CJW19)

Suppose that $\mathrm{NP} \subseteq \mathrm{Circuit}[n^k]$. Then every 2^{n^β}-sparse language $L_\beta \in \mathrm{NP}$ has circuits of size $O(n^{1+k\beta})$ (for all β).

Main idea — hashing

Lemma

Given β, there is a hash family $\{h_v \colon \{0,1\}^n \to \{0,1\}^t\}$ such that:

- For any $S \subseteq \{0,1\}^n$ of size $|S| \le 2^{n^\beta}$, there is some 'good seed' v such that h_v hashes all $x \in S$ to different values.
- The output length t and seed lengths $|v|$ are both $O(n^\beta)$.
- Given n, β, v, and x, we can very efficiently compute $h_v(x)$.

Main idea — hashing

Lemma

Given β, there is a hash family $\{h_v : \{0,1\}^n \to \{0,1\}^t\}$ such that:

- For any $S \subseteq \{0,1\}^n$ of size $|S| \leq 2^{n^\beta}$, there is some 'good seed' v such that h_v hashes all $x \in S$ to different values.
- The output length t and seed lengths $|v|$ are both $O(n^\beta)$.
- Given n, β, v, and x, we can very efficiently compute $h_v(x)$.

Main idea

On input x, we hash x; then instead of checking whether $x \in L_\beta$, we check whether there's some $y \in L_\beta$ with $h_v(y) = h_v(x)$.

This is a problem on much smaller input length — $h_v(x)$ only has length $O(n^\beta)$ (as opposed to x, which has length n).

The smaller problem

Definition (L_β-Hash-Match)

- **Input**: $\langle n, v, x^*, i, b\rangle$ (where $|x^*| = cn^\beta$, $i \in [n]$ and $b \in \{0,1\}$).
- **Decide**: is there some $y \in \{0,1\}^n$ such that $y \in L_\beta$, $h_v(y) = x^*$, and $y_i = b$?

The smaller problem

Definition (L_β-Hash-Match)

- **Input**: $\langle n, v, x^*, i, b\rangle$ (where $|x^*| = cn^\beta$, $i \in [n]$ and $b \in \{0,1\}$).
- **Decide**: is there some $y \in \{0,1\}^n$ such that $y \in L_\beta$, $h_v(y) = x^*$, and $y_i = b$?

Claim

L_β-Hash-Match $\in$ NP.

Proof.

- Guess y (of length n), check that $h_v(y) = x^*$ and $y_i = b$, and nondeterministically check $y \in L_\beta$.
- This takes poly(n) time; the input length is at least n^β, and n is a polynomial in n^β for fixed β. □

An oracle circuit

Lemma

There is a L_β-Hash-Match oracle circuit for L_β with size $O(n^{1+\beta})$ and queries of length $O(n^\beta)$.

An oracle circuit

Lemma

There is a L_β-Hash-Match oracle circuit for L_β with size $O(n^{1+\beta})$ and queries of length $O(n^\beta)$.

Construction

- Fix a good seed v for the **YES** instances of L_β (which we hardcode).
- Given x, compute $h_v(x)$ (which we can do with a $O(n)$-size circuit).
- Output $\bigwedge_{i \in [n]} L_\beta\text{-Hash-Match}(\langle n, v, h_v(x), i, x_i \rangle)$.

An oracle circuit

Lemma

There is a L_β-Hash-Match oracle circuit for L_β with size $O(n^{1+\beta})$ and queries of length $O(n^\beta)$.

Construction

- Fix a good seed v for the **YES** instances of L_β (which we hardcode).
- Given x, compute $h_v(x)$ (which we can do with a $O(n)$-size circuit).
- Output $\bigwedge_{i\in[n]} L_\beta\text{-Hash-Match}(\langle n, v, h_v(x), i, x_i\rangle)$.

Proof.

- For every i, is there some $y \in L_\beta$ with $h_v(y) = h_v(x)$ and $y_i = x_i$?
- If $x \in L_\beta$, then we can take $y = x$ for all i.
- If $x \notin L_\beta$, then there is at most one $y \in L_\beta$ with $h_v(y) = h_v(x)$, and this y must be wrong at some index i. $\square$

Finishing the proof

Theorem (Contrapositive of CJW19)

Suppose that NP $\subseteq$ Circuit$[n^k]$. Then every 2^{n^β}-sparse language $L_\beta \in$ NP has circuits of size $O(n^{1+k\beta})$ (for all β).

Proof.

- We take our oracle circuit and replace each call to L_β-Hatch-Match with an actual circuit.
- The assumption NP $\subseteq$ Circuit$[n^k]$ means that L_β-Hash-Match has circuits of size ℓ^k on inputs of length ℓ.
- Our oracle circuit made n queries of length $O(n^\beta)$, so now we can solve each query with a circuit of size $O(n^{\beta k})$; then our resulting circuit has size $O(n^{1+\beta k})$. $\square$

References

Thanks for listening!

- Lijie Chen, Ce Jin, and R. Ryan Williams. *Hardness magnification for all sparse* NP *languages*. 2019.
- Dylan M. McKay, Cody D. Murray, and R. Ryan Williams. *Weak lower bounds on resource-bounded compression imply strong separations of complexity classes*. 2019.
- Igor C. Oliveira, Ján Pich, and Rahul Santhanam. *Hardness magnification near state-of-the-art lower bounds*. 2019.
- Igor C. Oliveira and Rahul Santhanam. *Hardness magnification for natural problems*. 2018.