NP-Hardness of Constant-round Communication Complexity

Shuichi Hirahara
NII, Tokyo

Rahul Ilango
MIT

Bruno Loff
INESC-TEC and
University of Porto

Oxford-Warwick Complexity Seminar, May 6 2021

My plan for this talk

My plan for this talk

- I will spend a significant fraction of the talk giving a broad, historical introduction to the family of problems our problem is inscribed in, because I think it is an important subject, which could use more publicity.

My plan for this talk

- I will spend a significant fraction of the talk giving a broad, historical introduction to the family of problems our problem is inscribed in, because I think it is an important subject, which could use more publicity.
- I will then explain which problem in this family we solve, and sketch how we solve it.

My plan for this talk

- I will spend a significant fraction of the talk giving a broad, historical introduction to the family of problems our problem is inscribed in, because I think it is an important subject, which could use more publicity.
- I will then explain which problem in this family we solve, and sketch how we solve it.
- Detailed proofs will be for another day, sorry, but I *will* give a decent sketch of the proof for a toy version of our problem, and explain how one can build upon that result to get the full thing.

My plan for this talk

- I will spend a significant fraction of the talk giving a broad, historical introduction to the family of problems our problem is inscribed in, because I think it is an important subject, which could use more publicity.
- I will then explain which problem in this family we solve, and sketch how we solve it.
- Detailed proofs will be for another day, sorry, but I *will* give a decent sketch of the proof for a toy version of our problem, and explain how one can build upon that result to get the full thing.
- I predict 1h15m will be enough, maybe a little more or a little less depending on questions.

Part 1

A fundamental problem

What is the complexity of finding
the optimal algorithm for a given computational problem?

A fundamental problem

What is the complexity of finding
the optimal algorithm for a given computational problem?

A fundamental problem

What is the complexity of finding
the optimal algorithm for a given computational problem?

This question can be asked in many different settings:

A fundamental problem

What is the complexity of finding
the optimal **algorithm** for a given computational problem?

This question can be asked in many different settings:

- Each computational model gives us a notion of **algorithm**.
  For example, circuits, formulas, protocols, low-depth circuits, decision trees, RAM programs, *etc*.

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given computational problem?

This question can be asked in many different settings:

- Each computational model gives us a notion of **algorithm**. For example, circuits, formulas, protocols, low-depth circuits, decision trees, RAM programs, *etc*.
- Each complexity measure on this model gives us a notion of **optimal**. For example we might want circuits of small size or small depth, small and/or fast programs, or a protocol with little communication, a RAM program using little memory, *etc*.

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

This question can be asked in many different settings:

- Each computational model gives us a notion of **algorithm**. For example, circuits, formulas, protocols, low-depth circuits, decision trees, RAM programs, *etc*.
- Each complexity measure on this model gives us a notion of **optimal**. For example we might want circuits of small size or small depth, small and/or fast programs, or a protocol with little communication, a RAM program using little memory, *etc*.
- The question also depends on the kind of **computational problem** for which we want to find optimal algorithms. For example computing functions, solving search problems, sampling problems, streaming problems, *etc*.

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

The canonical example is the following:

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

The canonical example is the following:

- What is the complexity of finding the **smallest** **Boolean** **circuit** that **computes a function** $f : \{0,1\}^n \to \{0,1\}$, which is specified by its truth-table.

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

The canonical example is the following:

- What is the complexity of finding the **smallest** **Boolean** **circuit** that **computes a function** $f : \{0,1\}^n \to \{0,1\}$, which is specified by its truth-table.
- Many other settings can be considered (give examples).

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

The canonical example is the following:

- What is the complexity of finding the **smallest** **Boolean** **circuit** that **computes a function** $f : \{0,1\}^n \to \{0,1\}$, which is specified by its truth-table.
- Many other settings can be considered (give examples).
- Many variants of this question can (and should) also be considered, for example:

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

The canonical example is the following:

- What is the complexity of finding the **smallest** **Boolean** **circuit** that **computes a function** $f : \{0,1\}^n \to \{0,1\}$, which is specified by its truth-table.
- Many other settings can be considered (give examples).
- Many variants of this question can (and should) also be considered, for example:
  - We may consider the complexity of the (possibly) easier **decision** problem of *distinguishing problems with high vs. low complexity*: $\mathrm{MCSP} = \{\langle f, s\rangle \mid \mathrm{SIZE}(f) \leq s\}$.

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

The canonical example is the following:

- What is the complexity of finding the **smallest** **Boolean** **circuit** that **computes a function** $f : \{0,1\}^n \to \{0,1\}$, which is specified by its truth-table.
- Many other settings can be considered (give examples).
- Many variants of this question can (and should) also be considered, for example:
  - We may consider the complexity of the (possibly) easier **decision** problem of *distinguishing problems with high vs. low complexity*: $\mathrm{MCSP} = \{\langle f, s\rangle \mid \mathrm{SIZE}(f) \leq s\}$.
  - We may consider the complexity of the (possibly) easier **approximation** problem of *roughly distinguishing high vs. low complexity*. E.g. distinguish $\mathrm{SIZE}(f) \leq s^{1/10}$ versus $\mathrm{SIZE}(f) > s^{10}$.

A fundamental problem

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

The canonical example is the following:

- What is the complexity of finding the **smallest** **Boolean** **circuit** that **computes a function** $f : \{0,1\}^n \to \{0,1\}$, which is specified by its truth-table.
- Many other settings can be considered (give examples).
- Many variants of this question can (and should) also be considered, for example:
  - We may consider the complexity of the (possibly) easier **decision** problem of *distinguishing problems with high vs. low complexity*: $\mathrm{MCSP} = \{\langle f, s\rangle \mid \mathrm{SIZE}(f) \leq s\}$.
  - We may consider the complexity of the (possibly) easier **approximation** problem of *roughly distinguishing high vs. low complexity*. E.g. distinguish $\mathrm{SIZE}(f) \leq s^{1/10}$ versus $\mathrm{SIZE}(f) > s^{10}$.
  - Maybe we only care to have an **average case** solution, e.g. solve the problem efficiently for *most* given functions.

A fundamental problem: origins

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

A fundamental problem: origins

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- This problem has been studied since the 1950s (e.g. Karnaugh maps, automata minimization). Back then it was studied for the practical purpose of designing efficient circuits.

A fundamental problem: origins

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- This problem has been studied since the 1950s (e.g. Karnaugh maps, automata minimization). Back then it was studied for the practical purpose of designing efficient circuits.
- Stephen Cook, in his NP-completeness paper, asked whether it was NP-hard to find the size of the smallest DNF for a given Boolean function.

A fundamental problem: origins

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- This problem has been studied since the 1950s (e.g. Karnaugh maps, automata minimization). Back then it was studied for the practical purpose of designing efficient circuits.
- Stephen Cook, in his NP-completeness paper, asked whether it was NP-hard to find the size of the smallest DNF for a given Boolean function. Solved by Masek in 1979, improved to HoA in 2008.

A fundamental problem: origins

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- This problem has been studied since the 1950s (e.g. Karnaugh maps, automata minimization). Back then it was studied for the practical purpose of designing efficient circuits.
- Stephen Cook, in his NP-completeness paper, asked whether it was NP-hard to find the size of the smallest DNF for a given Boolean function. Solved by Masek in 1979, improved to HoA in 2008.
- Leonard Levin is said to have delayed the publication of his own NP-completeness paper, because he could not prove that MCSP is NP-hard.

A fundamental problem: origins

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- This problem has been studied since the 1950s (e.g. Karnaugh maps, automata minimization). Back then it was studied for the practical purpose of designing efficient circuits.
- Stephen Cook, in his NP-completeness paper, asked whether it was NP-hard to find the size of the smallest DNF for a given Boolean function. Solved by Masek in 1979, improved to HoA in 2008.
- Leonard Levin is said to have delayed the publication of his own NP-completeness paper, because he could not prove that MCSP is NP-hard. The problem remains unsolved to this day.

A fundamental problem: Positive vs Negative results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

A fundamental problem: Positive vs Negative results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- The known positive results give us methods to actually *find* the optimal (or near-optimal) algorithm for solving the given problem.

A fundamental problem: Positive vs Negative results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- The known positive results give us methods to actually *find* the optimal (or near-optimal) algorithm for solving the given problem.
- The known negative results show that it is hard, not just *finding* the optimal or near optimal algorithm, even just the problem of distinguishing a given high-complexity problem from a given low-complexity problem (sometimes even with a large promised gap), is already hard.

A fundamental problem: Positive vs Negative results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- The known positive results give us methods to actually *find* the optimal (or near-optimal) algorithm for solving the given problem.
- The known negative results show that it is hard, not just *finding* the optimal or near optimal algorithm, even just the problem of distinguishing a given high-complexity problem from a given low-complexity problem (sometimes even with a large promised gap), is already hard.
- I.e., known positive results show that the search version (find the algorithm) can be solved efficiently, whereas all the known negative results show that the *decision version* (distinguish complex from simple computational problems) is already hard.

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)
- Deterministic and randomized finite automata (Moore 1956)

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)
- Deterministic and randomized finite automata (Moore 1956)
- read-once formulas and read-once branching programs (Raviv)

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)
- Deterministic and randomized finite automata (Moore 1956)
- read-once formulas and read-once branching programs (Raviv)
- approximate and threshold degree (folklore LP, see Sherstov)

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)
- Deterministic and randomized finite automata (Moore 1956)
- read-once formulas and read-once branching programs (Raviv)
- approximate and threshold degree (folklore LP, see Sherstov)
- quantum query complexity (using semidefinite programming, by many people in early 2000s, Reichardt 2009)

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)
- Deterministic and randomized finite automata (Moore 1956)
- read-once formulas and read-once branching programs (Raviv)
- approximate and threshold degree (folklore LP, see Sherstov)
- quantum query complexity (using semidefinite programming, by many people in early 2000s, Reichardt 2009)
- "PP" communication protocols, aka discrepancy (using semidefinite programming, Linial, Lee, Shraibman and others, circa 2008)

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)
- Deterministic and randomized finite automata (Moore 1956)
- read-once formulas and read-once branching programs (Raviv)
- approximate and threshold degree (folklore LP, see Sherstov)
- quantum query complexity (using semidefinite programming, by many people in early 2000s, Reichardt 2009)
- "PP" communication protocols, aka discrepancy (using semidefinite programming, Linial, Lee, Shraibman and others, circa 2008)
- Many of these were found while trying to prove *lower bounds*!

A fundamental problem: positive results

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

There are some settings where we know how to efficiently find an optimal or near-optimal algorithm:

- Various kinds of decision-trees (folklore, see MSc thesis by Raviv)
- Deterministic and randomized finite automata (Moore 1956)
- read-once formulas and read-once branching programs (Raviv)
- approximate and threshold degree (folklore LP, see Sherstov)
- quantum query complexity (using semidefinite programming, by many people in early 2000s, Reichardt 2009)
- "PP" communication protocols, aka discrepancy (using semidefinite programming, Linial, Lee, Shraibman and others, circa 2008)
- Many of these were found while trying to prove *lower bounds*!
- One might optimistically ask: maybe it is *always* possible to find the best algorithm efficiently?

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For many settings, we know how to prove lower-bounds on this problem, if we assume that a certain cryptographic primitive is hard to break.

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For example, if we assume that factoring is hard, then it follows that it is hard to approximate:

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For example, if we assume that factoring is hard, then it follows that it is hard to approximate:
  - the minimum circuit size (Kabanets and Cai, 2000),

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For example, if we assume that factoring is hard, then it follows that it is hard to approximate:
  - the minimum circuit size (Kabanets and Cai, 2000),
  - the minimum formula size (Allender, Koucký, Ronneburger, and Roy 2003),

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For example, if we assume that factoring is hard, then it follows that it is hard to approximate:
  - the minimum circuit size (Kabanets and Cai, 2000),
  - the minimum formula size (Allender, Koucký, Ronneburger, and Roy 2003),
  - deterministic communication complexity (Kushilevitz and Weinreb 2009),

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For example, if we assume that factoring is hard, then it follows that it is hard to approximate:
  - the minimum circuit size (Kabanets and Cai, 2000),
  - the minimum formula size (Allender, Koucký, Ronneburger, and Roy 2003),
  - deterministic communication complexity (Kushilevitz and Weinreb 2009),
  - the size of the smallest AC_0 circuit (Allender, Hellerstein, McCabe, Pitassi, and Saks, 2008), or

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For example, if we assume that factoring is hard, then it follows that it is hard to approximate:
  - the minimum circuit size (Kabanets and Cai, 2000),
  - the minimum formula size (Allender, Koucký, Ronneburger, and Roy 2003),
  - deterministic communication complexity (Kushilevitz and Weinreb 2009),
  - the size of the smallest AC_0 circuit (Allender, Hellerstein, McCabe, Pitassi, and Saks, 2008), or
  - the size of the smallest branching program (Raviv, 2013)

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For example, if we assume that factoring is hard, then it follows that it is hard to approximate:
  - the minimum circuit size (Kabanets and Cai, 2000),
  - the minimum formula size (Allender, Koucký, Ronneburger, and Roy 2003),
  - deterministic communication complexity (Kushilevitz and Weinreb 2009),
  - the size of the smallest AC_0 circuit (Allender, Hellerstein, McCabe, Pitassi, and Saks, 2008), or
  - the size of the smallest branching program (Raviv, 2013)
  - (for computing a given total Boolean function).

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For many settings, we know how to prove lower-bounds on this problem, if we assume that a certain cryptographic primitive is hard to break.

Negative results: Cryptographic hardness

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For many settings, we know how to prove lower-bounds on this problem, if we assume that a certain cryptographic primitive is hard to break.
- In fact, if there existed a polytime algorithm for MCSP, then there would exist polytime algorithms for any problem in the class SZK of problems having statistically zero-knowledge proofs (Allender and Das, 2014).

Negative results: Cryptographic hardness

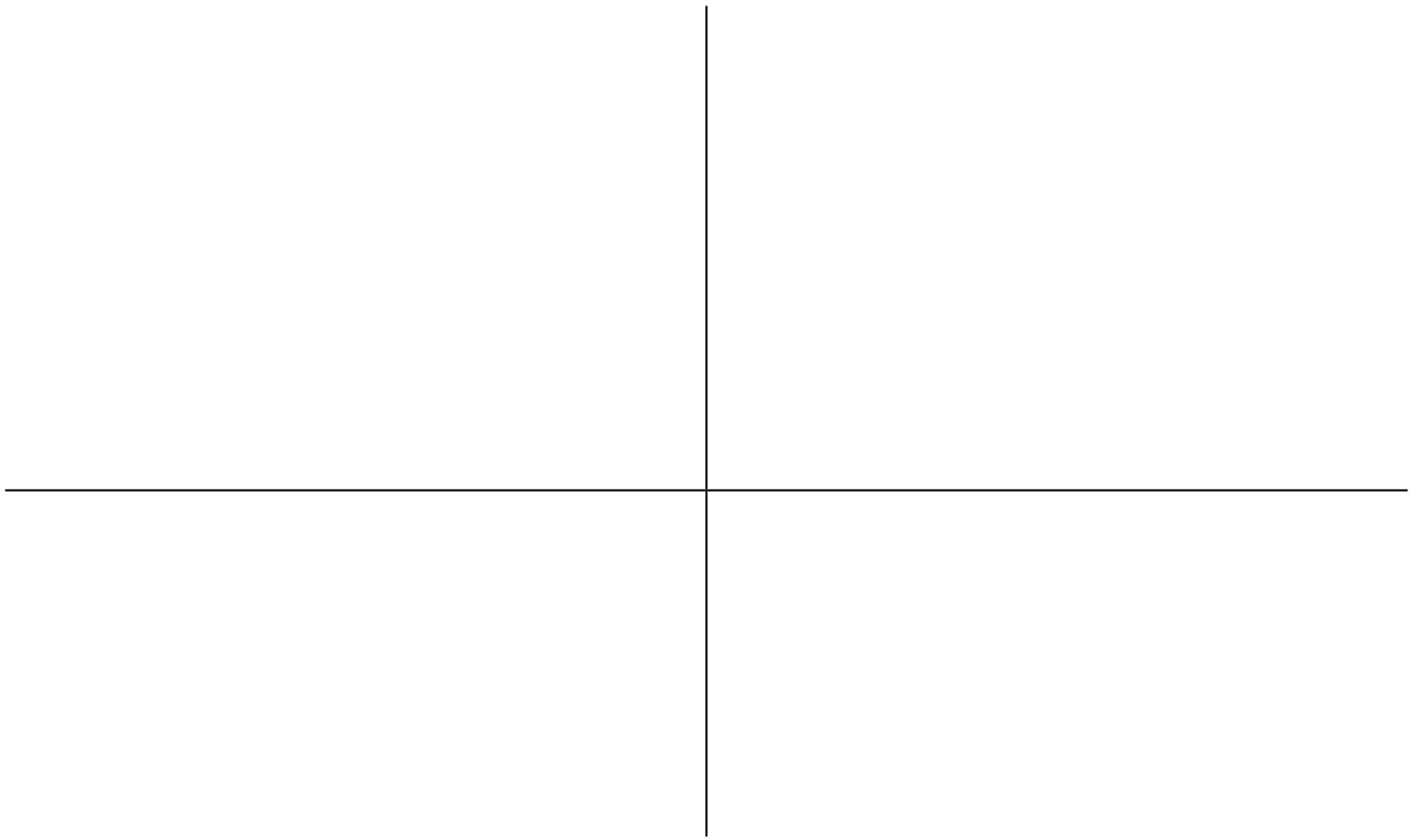
What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- Maybe it is *always* possible to find the best algorithm efficiently?
- There are very good reasons to believe this is not the case.
- For many settings, we know how to prove lower-bounds on this problem, if we assume that a certain cryptographic primitive is hard to break.
- In fact, if there existed a polytime algorithm for MCSP, then there would exist polytime algorithms for any problem in the class SZK of problems having statistically zero-knowledge proofs (Allender and Das, 2014).
- However, all of the above problems are in $\mathrm{NP} \cap \mathrm{coNP}$ (*), so one might still conjecture, for example, that MCSP is in $\mathrm{NP} \cap \mathrm{coNP}$.

The duality question

Is MCSP in NP $\cap$ coNP?

$$\mathrm{MCSP} = \{\langle f, s\rangle \mid \mathrm{SIZE}(f) \leq s\}.$$

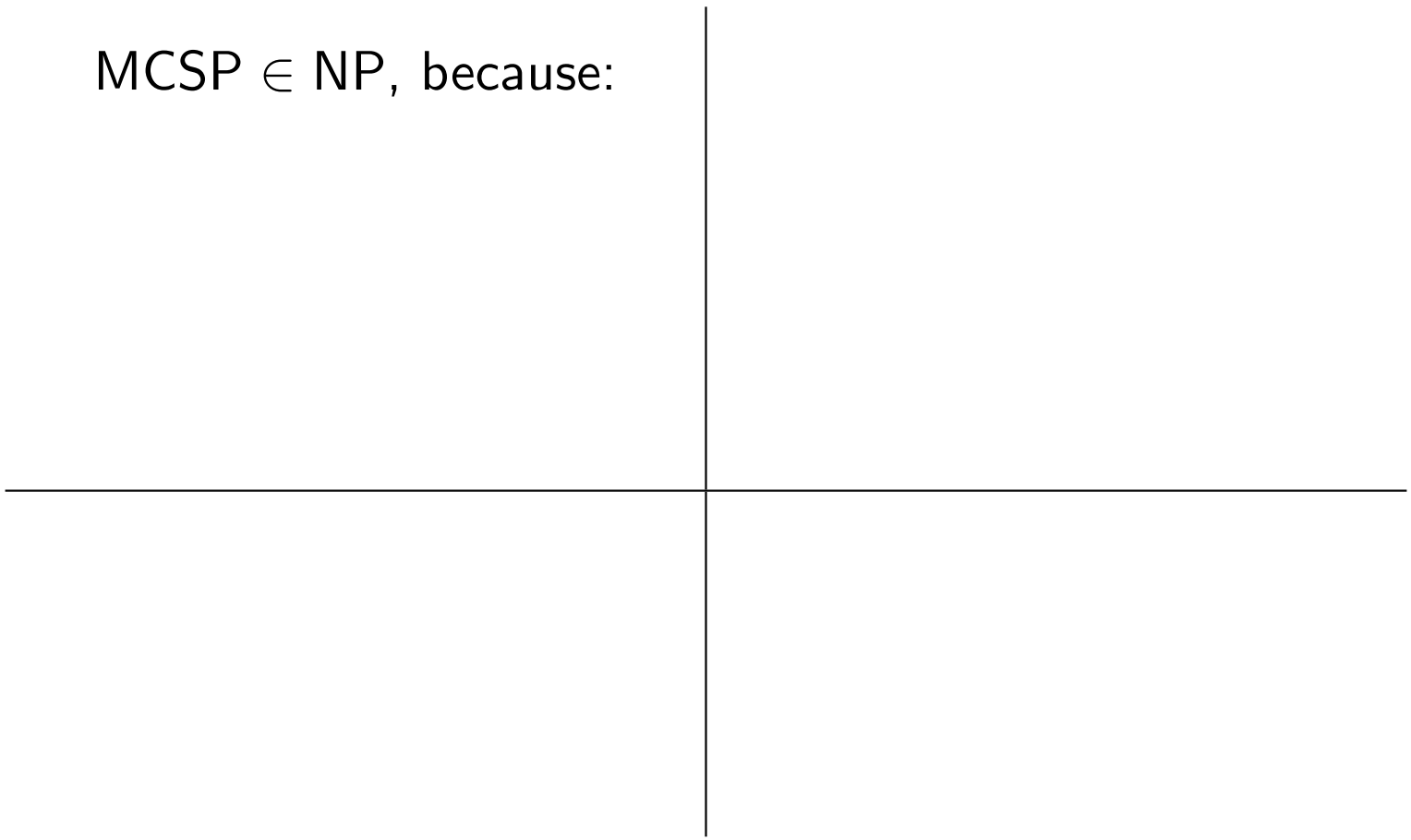


The duality question

Is MCSP in $\mathrm{NP} \cap \mathrm{coNP}$?

$$\mathrm{MCSP} = \{\langle f, s\rangle \mid \mathrm{SIZE}(f) \leq s\}.$$

MCSP $\in$ NP, because:



The duality question

Is MCSP in NP $\cap$ coNP?

$$\text{MCSP} = \{\langle f, s\rangle \mid \text{SIZE}(f) \leq s\}.$$

MCSP $\in$ NP, because:

$$\text{SIZE}(f) \leq s$$
$$\Longleftrightarrow$$
$$\exists C\ \varphi(f, s, C)$$

The duality question

Is MCSP in $\mathsf{NP} \cap \mathsf{coNP}$?

$$\mathrm{MCSP} = \{\langle f, s\rangle \mid \mathrm{SIZE}(f) \leq s\}.$$

MCSP $\in$ NP, because:

$$\mathrm{SIZE}(f) \leq s \iff \exists C\ \varphi(f, s, C)$$

What would it mean if MCSP $\in$ coNP?

The duality question

Is MCSP in NP $\cap$ coNP?

$$\text{MCSP} = \{\langle f, s\rangle \mid \text{SIZE}(f) \leq s\}.$$

MCSP $\in$ NP, because:

$$\text{SIZE}(f) \leq s$$
$$\Longleftrightarrow$$
$$\exists C\ \varphi(f, s, C)$$

What would it mean if MCSP $\in$ coNP?

$$\text{SIZE}(f) > s$$
$$\Longleftrightarrow$$
$$\exists W\ \psi(f, s, W)$$

The duality question

Is MCSP in NP $\cap$ coNP?

$$\text{MCSP} = \{\langle f, s\rangle \mid \text{SIZE}(f) \leq s\}.$$

MCSP $\in$ NP, because:

$$\text{SIZE}(f) \leq s$$
$$\Longleftrightarrow$$
$$\exists C\ \varphi(f, s, C)$$

What would it mean if MCSP $\in$ coNP?

$$\text{SIZE}(f) > s$$
$$\Longleftrightarrow$$
$$\exists W\ \psi(f, s, W)$$

$$x \in P$$
$$\Longleftrightarrow$$
$$\exists (a_1, \ldots, a_n)\ x = \textstyle\sum_i a_i v_i$$

The duality question

Is MCSP in NP $\cap$ coNP?

$$\mathrm{MCSP} = \{\langle f, s\rangle \mid \mathrm{SIZE}(f) \leq s\}.$$

MCSP $\in$ NP, because:

$$\mathrm{SIZE}(f) \leq s \iff \exists C\ \varphi(f, s, C)$$

What would it mean if MCSP $\in$ coNP?

$$\mathrm{SIZE}(f) > s \iff \exists W\ \psi(f, s, W)$$

$$x \in P \iff \exists (a_1, \ldots, a_n)\ x = \textstyle\sum_i a_i v_i$$

$$x \notin P \iff \exists (w_1, \ldots, w_n)\ \langle x, w\rangle > 0 \quad \forall i \langle v_i, w\rangle < 0$$

The duality question

Is MCSP in coNP?

The duality question

Is MCSP in coNP?

- **if the answer is Yes:** then we have a duality for circuit size, and we know what a lower-bound proof "looks like".

The duality question

Is MCSP in coNP?

- **if the answer is Yes:** then we have a duality for circuit size, and we know what a lower-bound proof "looks like".

(I.e., a complexity theorist's dream)

The duality question

Is MCSP in coNP?

- **if the answer is Yes:** then we have a duality for circuit size, and we know what a lower-bound proof "looks like". 
- **if the answer is No:** Then there is no duality for circuit size, which roughly means that the generic tight lower-bound cannot be proven by a constructive proof. (We wouldn't be able to recognize a proof if we saw one.)

The duality question

Is MCSP in coNP?

- **if the answer is Yes:** then we have a duality for circuit size, and we know what a lower-bound proof "looks like". 
- **if the answer is No, and MCSP is** NP-**hard:** There is no exact duality unless NP $=$ coNP. But we may still have an *approximate* duality (for example, if $\mathrm{SIZE}(f) \geq s$ then there exists some efficiently-recognizable witness proving that $\mathrm{SIZE}(f) \geq s^{1/10}$).

The duality question

Is MCSP in coNP?

- **if the answer is Yes:** then we have a duality for circuit size, and we know what a lower-bound proof "looks like".

- **if the answer is No, and MCSP is** NP**-hard to approximate, even in the average case, even with a very high approximation ratio!:** This would give us a new barrier in complexity theory, similar to Razborov and Rudich's natural proofs barrier, but based on the assumption that NP $\neq$ coNP instead of based on cryptographic assumptions. It would tell us that for most functions, proving a circuit-size lower-bound, even a lower-bound which is only remotely close to optimum, cannot be done constructively (assuming NP $\neq$ coNP).

The duality question

Is MCSP in coNP?

- **if the answer is Yes:** then we have a duality for circuit size, and we know what a lower-bound proof "looks like". 
- **if the answer is No, and MCSP is** NP**-hard to approximate, even in the average case, even with a very high approximation ratio!:** This would give us a new barrier in complexity theory, similar to Razborov and Rudich's natural proofs barrier, but based on the assumption that NP $\neq$ coNP instead of based on cryptographic assumptions. It would tell us that for most functions, proving a circuit-size lower-bound, even a lower-bound which is only remotely close to optimum, cannot be done constructively (assuming NP $\neq$ coNP).

(I.e., a complexity theorist's nightmare)

The duality question

Is MCSP in coNP?

- **if the answer is Yes:** then we have a duality for circuit size, and we know what a lower-bound proof "looks like". 
- **if the answer is No, and MCSP is** NP**-hard to approximate, even in the average case, even with a very high approximation ratio!:** This would give us a new barrier in complexity theory, similar to Razborov and Rudich's natural proofs barrier, but based on the assumption that NP $\neq$ coNP instead of based on cryptographic assumptions. It would tell us that for most functions, proving a circuit-size lower-bound, even a lower-bound which is only remotely close to optimum, cannot be done constructively (assuming NP $\neq$ coNP).

(I.e., a complexity theorist's nightmare)
In this precise sense it could be said that NP $\neq$ coNP, if true, would "fight" against its own proof.

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)
Circuits with oracle gates	1	?	Ilango 2020

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)
Circuits with oracle gates	1	?	Ilango 2020
Circuits (multi-output f)	1	?	Ilango, Loff, Oliveira 2020

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)
Circuits with oracle gates	1	?	Ilango 2020
Circuits (multi-output f)	1	?	Ilango, Loff, Oliveira 2020
Circuits	?	$2^n/2^{o(n)}$	Hirahara and Allender 2017

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)
Circuits with oracle gates	1	?	Ilango 2020
Circuits (multi-output f)	1	?	Ilango, Loff, Oliveira 2020
Circuits	?	$2^n/2^{o(n)}$	Hirahara and Allender 2017
depth-d formulas	$1+\varepsilon_d$	?	Ilango 2020

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)
Circuits with oracle gates	1	?	Ilango 2020
Circuits (multi-output f)	1	?	Ilango, Loff, Oliveira 2020
Circuits	?	$2^n/2^{o(n)}$	Hirahara and Allender 2017
depth-d formulas	$1+\varepsilon_d$	?	Ilango 2020
depth-3 protocols	$2^{\frac{1}{16}n}$	$2^n/\sqrt{n}$	This work

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)
Circuits with oracle gates	1	?	Ilango 2020
Circuits (multi-output f)	1	?	Ilango, Loff, Oliveira 2020
Circuits	?	$2^n/2^{o(n)}$	Hirahara and Allender 2017
depth-d formulas	$1+\varepsilon_d$	?	Ilango 2020
depth-3 protocols	$2^{\frac{1}{16}n}$	$2^n/\sqrt{n}$	This work
depth-d protocols	$1+\varepsilon_d$	?	

Negative results: NP-hardness

We could be living in the nightmare scenario, and we already have NP-hardness results in several settings, albeit for exact computation or for small approximation factors. For example, if we are given the truth-table of a function f on n-bit inputs, then we have the following NP-hardness:

Model	**HoA** factor	not NPH	
DNFs	$n^{1-\varepsilon}$	$O(n)$	Khot and Saket, 2008
$\mathrm{DNF} \circ \mathrm{MOD}_p$	$\Omega(\log n)$	$O(n)$	(Hirahara, Oliveira, Santhanam 2018)
Circuits with oracle gates	1	?	Ilango 2020
Circuits (multi-output f)	1	?	Ilango, Loff, Oliveira 2020
Circuits	?	$2^n/2^{o(n)}$	Hirahara and Allender 2017
depth-d formulas	$1+\varepsilon_d$	?	Ilango 2020
depth-3 protocols	$2^{\frac{1}{16}n}$	$2^n/\sqrt{n}$	This work
depth-d protocols	$1+\varepsilon_d$	?	
polytime RAM programs	1	?	Liu and Pass two weeks ago

Proving NP-hardness

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).
- It just so happens, the outputs of these reductions are computational problems different to those studied so far.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).
- It just so happens, the outputs of these reductions are computational problems different to those studied so far.
- And so, in all the papers on this subject, new lower-bound techniques had to be developed in order to show this second part.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).
- It just so happens, the outputs of these reductions are computational problems different to those studied so far.
- And so, in all the papers on this subject, new lower-bound techniques had to be developed in order to show this second part. E.g., new lifting theorems, new chain rules, and (today) a new round-elimination lemma for deterministic CC.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).
- It just so happens, the outputs of these reductions are computational problems different to those studied so far.
- And so, in all the papers on this subject, new lower-bound techniques had to be developed in order to show this second part. E.g., new lifting theorems, new chain rules, and (today) a new round-elimination lemma for deterministic CC.
- This makes studying our research question extremely attractive, because it is not only interesting in itself, it seems likely to advance our knowledge of lower-bounds.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).
- It just so happens, the outputs of these reductions are computational problems different to those studied so far.
- And so, in all the papers on this subject, new lower-bound techniques had to be developed in order to show this second part. E.g., new lifting theorems, new chain rules, and (today) a new round-elimination lemma for deterministic CC.
- This makes studying our research question extremely attractive, because it is not only interesting in itself, it seems likely to advance our knowledge of lower-bounds.
- Sometimes this connection can be made formal.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).
- It just so happens, the outputs of these reductions are computational problems different to those studied so far.
- And so, in all the papers on this subject, new lower-bound techniques had to be developed in order to show this second part. E.g., new lifting theorems, new chain rules, and (today) a new round-elimination lemma for deterministic CC.
- This makes studying our research question extremely attractive, because it is not only interesting in itself, it seems likely to advance our knowledge of lower-bounds.
- Sometimes this connection can be made formal. There are several results showing that proving NP-hardness of MCSP under various types of reductions would require proving new separations, e.g. EXP $\not\subseteq$ ZPP.

Proving NP-hardness

- These proofs of NP-hardness are quite fun, as they mix reductions and lower-bound techniques. E.g. we must produce from a formula ψ a function f_ψ so that (1) ψ is satisfiable implies f_ψ is simple to compute, and (2) ψ is not satisfiable implies f_ψ is hard. Part (2) requires proving a lower-bound (*).
- It just so happens, the outputs of these reductions are computational problems different to those studied so far.
- And so, in all the papers on this subject, new lower-bound techniques had to be developed in order to show this second part. E.g., new lifting theorems, new chain rules, and (today) a new round-elimination lemma for deterministic CC.
- This makes studying our research question extremely attractive, because it is not only interesting in itself, it seems likely to advance our knowledge of lower-bounds.
- Sometimes this connection can be made formal. There are several results showing that proving NP-hardness of MCSP under various types of reductions would require proving new separations, e.g. EXP $\not\subseteq$ ZPP. (See surveys by Eric Allender)

Hardness Magnification

Hardness Magnification

- There are some unconditional hardness results for this family of problems, e.g., MCSP cannot be solved by subcubic formulas, by subexponential AC_0 circuits, or by polysize $AC_0[p]$ circuits (Allender, Buhrman, Koucký, Melkebeek, and Ronneburger, 2006, Hirahara and Santhanam, 2017, Cheraghchi, Kabanets, Lu, and Myrisiotis, 2020, Golovnev, Ilango, Impagliazzo, Kabanets, Kolokolova, and Tal, 2019).

Hardness Magnification

- There are some unconditional hardness results for this family of problems, e.g., MCSP cannot be solved by subcubic formulas, by subexponential AC_0 circuits, or by polysize $AC_0[p]$ circuits (Allender, Buhrman, Koucký, Melkebeek, and Ronneburger, 2006, Hirahara and Santhanam, 2017, Cheraghchi, Kabanets, Lu, and Myrisiotis, 2020, Golovnev, Ilango, Impagliazzo, Kabanets, Kolokolova, and Tal, 2019).
- Remarkably, it can be formally shown that proving lower-bounds for certain variants of our problem, in some cases *even lower-bounds that we already know how to prove for other problems*, will lead to breakthrough results in computational complexity.

Hardness Magnification

- There are some unconditional hardness results for this family of problems, e.g., MCSP cannot be solved by subcubic formulas, by subexponential AC_0 circuits, or by polysize $AC_0[p]$ circuits (Allender, Buhrman, Koucký, Melkebeek, and Ronneburger, 2006, Hirahara and Santhanam, 2017, Cheraghchi, Kabanets, Lu, and Myrisiotis, 2020, Golovnev, Ilango, Impagliazzo, Kabanets, Kolokolova, and Tal, 2019).
- Remarkably, it can be formally shown that proving lower-bounds for certain variants of our problem, in some cases *even lower-bounds that we already know how to prove for other problems*, will lead to breakthrough results in computational complexity.
- Meaning, even weak lower-bounds on our family of problems can sometimes be converted into strong lower-bounds which we do not know how to prove.

Hardness Magnification

Hardness Magnification

- To give an example of such a result: if one can prove that a certain average-case approximation version of MCSP cannot be computed by formulas of size $n^{1+\varepsilon}$, then NP does not have polynomial-size formulas (i.e., NP $\not\subseteq \mathsf{NC}_1$) (Oliveira and Santhanam, 2018).

Hardness Magnification

- To give an example of such a result: if one can prove that a certain average-case approximation version of MCSP cannot be computed by formulas of size $n^{1+\varepsilon}$, then NP does not have polynomial-size formulas (i.e., NP $\not\subseteq$ NC_1) (Oliveira and Santhanam, 2018).
- It is worth noting that we know how to prove formula lower-bounds of $n^{3-\varepsilon}$ for some problems (Hästad, 1999), and

Hardness Magnification

- To give an example of such a result: if one can prove that a certain average-case approximation version of MCSP cannot be computed by formulas of size $n^{1+\varepsilon}$, then NP does not have polynomial-size formulas (i.e., NP $\not\subseteq \mathrm{NC}_1$) (Oliveira and Santhanam, 2018).
- It is worth noting that we know how to prove formula lower-bounds of $n^{3-\varepsilon}$ for some problems (Hästad, 1999), and
- we know how to prove formula lower-bounds of $n^{2-\varepsilon}$ for the *worst-case* version of the same MCSP variant (Hirahara and Santhanam, 2017), and we even have worst-case to average-case reductions for other MCSP variants (Hirahara, 2018).

Hardness Magnification

- To give an example of such a result: if one can prove that a certain average-case approximation version of MCSP cannot be computed by formulas of size $n^{1+\varepsilon}$, then NP does not have polynomial-size formulas (i.e., NP $\not\subseteq$ NC_1) (Oliveira and Santhanam, 2018).
- It is worth noting that we know how to prove formula lower-bounds of $n^{3-\varepsilon}$ for some problems (Hästad, 1999), and
- we know how to prove formula lower-bounds of $n^{2-\varepsilon}$ for the *worst-case* version of the same MCSP variant (Hirahara and Santhanam, 2017), and we even have worst-case to average-case reductions for other MCSP variants (Hirahara, 2018).
- Almost there?

Hardness Magnification

- To give an example of such a result: if one can prove that a certain average-case approximation version of MCSP cannot be computed by formulas of size $n^{1+\varepsilon}$, then NP does not have polynomial-size formulas (i.e., NP $\not\subseteq \mathrm{NC}_1$) (Oliveira and Santhanam, 2018).
- It is worth noting that we know how to prove formula lower-bounds of $n^{3-\varepsilon}$ for some problems (Hästad, 1999), and
- we know how to prove formula lower-bounds of $n^{2-\varepsilon}$ for the *worst-case* version of the same MCSP variant (Hirahara and Santhanam, 2017), and we even have worst-case to average-case reductions for other MCSP variants (Hirahara, 2018).
- Almost there? These problems seem to require more refined lower-bound techniques than the ones we currently have (see *Hardness Magnification and Locality*, by Chen, Hirahara, Oliveira, Pich, Rajgopal, and Santhanam).

Other results

There are other good reasons to study our problem.

Other results

There are other good reasons to study our problem.

- There exists a surprising worst-case to average-case reduction for a certain variant of MCSP (Hirahara 2018), so if this variant of MCSP is NP-hard this would make it the only known NP-complete problem with such worst-case to average-case reductions, and indeed the only such problem not in $\mathrm{NP} \cap \mathrm{coNP}$ (assuming $\mathrm{NP} \neq \mathrm{coNP}$).

Other results

There are other good reasons to study our problem.

- There exists a surprising worst-case to average-case reduction for a certain variant of MCSP (Hirahara 2018), so if this variant of MCSP is NP-hard this would make it the only known NP-complete problem with such worst-case to average-case reductions, and indeed the only such problem not in $\mathsf{NP} \cap \mathsf{coNP}$ (assuming $\mathsf{NP} \neq \mathsf{coNP}$).
- There is a ubiquitous connection to cryptography.

Other results

There are other good reasons to study our problem.

- There exists a surprising worst-case to average-case reduction for a certain variant of MCSP (Hirahara 2018), so if this variant of MCSP is NP-hard this would make it the only known NP-complete problem with such worst-case to average-case reductions, and indeed the only such problem not in $\mathsf{NP} \cap \mathsf{coNP}$ (assuming $\mathsf{NP} \neq \mathsf{coNP}$).
- There is a ubiquitous connection to cryptography. Existence of one-way functions is equivalent to hardness of certain average-case variants of our problem (Liu and Pass 2020).

Other results

There are other good reasons to study our problem.

- There exists a surprising worst-case to average-case reduction for a certain variant of MCSP (Hirahara 2018), so if this variant of MCSP is NP-hard this would make it the only known NP-complete problem with such worst-case to average-case reductions, and indeed the only such problem not in $\mathsf{NP} \cap \mathsf{coNP}$ (assuming $\mathsf{NP} \neq \mathsf{coNP}$).
- There is a ubiquitous connection to cryptography. Existence of one-way functions is equivalent to hardness of certain average-case variants of our problem (Liu and Pass 2020).
- There is a connection to learning.

Other results

There are other good reasons to study our problem.

- There exists a surprising worst-case to average-case reduction for a certain variant of MCSP (Hirahara 2018), so if this variant of MCSP is NP-hard this would make it the only known NP-complete problem with such worst-case to average-case reductions, and indeed the only such problem not in $\mathsf{NP} \cap \mathsf{coNP}$ (assuming $\mathsf{NP} \neq \mathsf{coNP}$).
- There is a ubiquitous connection to cryptography. Existence of one-way functions is equivalent to hardness of certain average-case variants of our problem (Liu and Pass 2020).
- There is a connection to learning. The existence of certain learning algorithms can also be shown to be (roughly) equivalent to the hardness of certain average-case variants of our problem (Carmosino, Impagliazzo, Kabanets, and Kolokolova, 2016, Ilango, Loff and Oliveira 2020).

Conclusion

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

Conclusion

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- We have a natural and fundamental question, that has been asked since the beginning of our discipline.

Conclusion

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- We have a natural and fundamental question, that has been asked since the beginning of our discipline.
- It is deeply connected to several other fundamental questions in computational complexity, including lower-bounds, cryptography, worst-case to average-case reductions, and learning theory.

Conclusion

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- We have a natural and fundamental question, that has been asked since the beginning of our discipline.
- It is deeply connected to several other fundamental questions in computational complexity, including lower-bounds, cryptography, worst-case to average-case reductions, and learning theory.
- History suggests, and it can sometimes be formally shown, that any progress in this question will lead to important contributions to our field.

Conclusion

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- We have a natural and fundamental question, that has been asked since the beginning of our discipline.
- It is deeply connected to several other fundamental questions in computational complexity, including lower-bounds, cryptography, worst-case to average-case reductions, and learning theory.
- History suggests, and it can sometimes be formally shown, that any progress in this question will lead to important contributions to our field.
- **Nonetheless, the question remains unanswered in most settings!** (for most choices of **optimal**, **algorithm**, and **computational problem**)

Conclusion

What is the complexity of finding
the **optimal** **algorithm** for a given **computational problem**?

- We have a natural and fundamental question, that has been asked since the beginning of our discipline.
- It is deeply connected to several other fundamental questions in computational complexity, including lower-bounds, cryptography, worst-case to average-case reductions, and learning theory.
- History suggests, and it can sometimes be formally shown, that any progress in this question will lead to important contributions to our field.
- **Nonetheless, the question remains unanswered in most settings!** (for most choices of **optimal**, **algorithm**, and **computational problem**)

End of Part 1

Part 2

The complexity of constant-round communication complexity

What is the complexity of finding
the **smallest constant-round two-party protocol**
for **computing a given function**?

Part 2

The complexity of constant-round communication complexity

What is the complexity of finding
the **smallest constant-round two-party protocol**
for **computing a given function**?

Answer: **It's** NP-**hard!**

Part 2

The complexity of constant-round communication complexity

What is the complexity of finding
the **smallest constant-round two-party protocol**
for **computing a given function**?

Answer: **It's** NP-**hard!**
(To a not-completely-satisfactory approximation factor)

A two-party communication problem

$$A, B \in \mathbb{N}$$

A two-party communication problem

$$A, B \in \mathbb{N}$$
$$f : [A] \times [B] \to \{0, 1\}$$

A two-party communication problem

$$A, B \in \mathbb{N}$$
$$f : [A] \times [B] \to \{0, 1\}$$

Alice gets $x \in [A]$, Bob gets $y \in [B]$,
and they wish to learn $f(x, y)$.

A two-party communication problem

$$A, B \in \mathbb{N}$$
$$f : [A] \times [B] \to \{0, 1\}$$

Alice gets $x \in [A]$, Bob gets $y \in [B]$,
and they wish to learn $f(x, y)$.

(draw f as a matrix, explain what happens in a protocol, and how one counts "size" and "length" of the protocol)

A two-party communication problem

$$A, B \in \mathbb{N}$$
$$f : [A] \times [B] \to \{0, 1\}$$

Alice gets $x \in [A]$, Bob gets $y \in [B]$,
and they wish to learn $f(x, y)$.

(draw f as a matrix, explain what happens in a protocol, and how one counts "size" and "length" of the protocol)

(define $\mathrm{L}_r^{\mathrm{Alice}}(f)$ and $\mathrm{L}_r^{\mathrm{Bob}}(f)$)

A two-party communication problem

$$A, B \in \mathbb{N}$$
$$f : [A] \times [B] \to \{0, 1\}$$

Alice gets $x \in [A]$, Bob gets $y \in [B]$,
and they wish to learn $f(x, y)$.

(draw f as a matrix, explain what happens in a protocol, and how one counts "size" and "length" of the protocol)

(define $\mathrm{L}_r^{\mathrm{Alice}}(f)$ and $\mathrm{L}_r^{\mathrm{Bob}}(f)$)

Main question:
Suppose we are given the communication matrix of f as input.
What is the complexity of computing $\mathrm{L}_r^{\mathrm{Alice}}(f)$?

Graph Coloring and non-binary 3-rounds

Definition

Given an undirected graph $G = ([n], E)$ over n vertices, a *coloring* is an assignment of colors to the vertices so that no edge exists in G between vertices having the same color.
$\chi(G)$ is the minimum number of colors needed to color the graph.

Graph Coloring and non-binary 3-rounds

Definition

Given an undirected graph $G = ([n], E)$ over n vertices, a *coloring* is an assignment of colors to the vertices so that no edge exists in G between vertices having the same color.
$\chi(G)$ is the minimum number of colors needed to color the graph.

Theorem (HoA for $\chi(G)$, Zuckerman and others)

For every $\varepsilon > 0$, there is a deterministic polynomial time algorithm that on an input $x \in \{0,1\}^*$, outputs a graph G on n vertices such that

- *if x is a YES instance of SAT, then $\chi(G) \leq n^{\varepsilon}$, and*
- *if x is a NO instance of SAT, then $\chi(G) \geq n^{1-\varepsilon}$.*

Graph Coloring and non-binary 3-rounds

Definition

Given an undirected graph $G = ([n], E)$ over n vertices, a *coloring* is an assignment of colors to the vertices so that no edge exists in G between vertices having the same color.
$\chi(G)$ is the minimum number of colors needed to color the graph.

Theorem (HoA for $\chi(G)$, Zuckerman and others)

For every $\varepsilon > 0$, there is a deterministic polynomial time algorithm that on an input $x \in \{0,1\}^*$, outputs a graph G on n vertices such that

- *if x is a YES instance of SAT, then $\chi(G) \leq n^{\varepsilon}$, and*
- *if x is a NO instance of SAT, then $\chi(G) \geq n^{1-\varepsilon}$.*

$$A, B, K \in \mathbb{N}$$

Graph Coloring and non-binary 3-rounds

Definition

Given an undirected graph $G = ([n], E)$ over n vertices, a *coloring* is an assignment of colors to the vertices so that no edge exists in G between vertices having the same color.
$\chi(G)$ is the minimum number of colors needed to color the graph.

Theorem (HoA for $\chi(G)$, Zuckerman and others)

For every $\varepsilon > 0$, there is a deterministic polynomial time algorithm that on an input $x \in \{0,1\}^*$, outputs a graph G on n vertices such that

- *if x is a YES instance of SAT, then $\chi(G) \leq n^{\varepsilon}$, and*
- *if x is a NO instance of SAT, then $\chi(G) \geq n^{1-\varepsilon}$.*

$$A, B, K \in \mathbb{N}$$
$$f : [A] \times [B] \to \{0, 1, \ldots, K\}$$

Graph Coloring and non-binary 3-rounds

Definition

Given an undirected graph $G = ([n], E)$ over n vertices, a *coloring* is an assignment of colors to the vertices so that no edge exists in G between vertices having the same color.
$\chi(G)$ is the minimum number of colors needed to color the graph.

Theorem (HoA for $\chi(G)$, Zuckerman and others)

For every $\varepsilon > 0$, there is a deterministic polynomial time algorithm that on an input $x \in \{0,1\}^*$, outputs a graph G on n vertices such that

- *if x is a YES instance of SAT, then $\chi(G) \leq n^{\varepsilon}$, and*
- *if x is a NO instance of SAT, then $\chi(G) \geq n^{1-\varepsilon}$.*

$$A, B, K \in \mathbb{N}$$
$$f : [A] \times [B] \to \{0, 1, \ldots, K\}$$

Let us try to understand $\mathrm{L}_1^{\mathrm{Bob}}$, $\mathrm{L}_2^{\mathrm{Alice}}$ and $\mathrm{L}_3^{\mathrm{Bob}}(f)$.

How to get the 3-round binary case?

(An idealized gadget. Won't work because Bob can break up the columns. But should work with a random matrix, and ultimately can be made to work with pseudorandom matrix.)

How to get the 3-round binary case?

(An idealized gadget. Won't work because Bob can break up the columns. But should work with a random matrix, and ultimately can be made to work with pseudorandom matrix.)

Theorem

It is NP*-hard to distinguish, for a given* $f : [N] \times [N] \to \{0, 1\}$*, whether* $L_3^A(f) \leq N^{\varepsilon}$ *or whether* $L_3^A(f) \geq N^{\frac{1}{8}-\varepsilon}$.

How to get the general k-round case?

How to get the general k-round case?

We have shown that $L_3(f)$ is hard, so now we show a way of estimating $L_r(f)$ using an oracle for $L_{r+1}(F)$.

How to get the general k-round case?

We have shown that $L_3(f)$ is hard, so now we show a way of estimating $L_r(f)$ using an oracle for $L_{r+1}(F)$.

Natural thing to try: round-elimination.

How to get the general k-round case?

We have shown that $L_3(f)$ is hard, so now we show a way of estimating $L_r(f)$ using an oracle for $L_{r+1}(F)$.

Natural thing to try: round-elimination. But it is probabilistic.

How to get the general k-round case?

We have shown that $L_3(f)$ is hard, so now we show a way of estimating $L_r(f)$ using an oracle for $L_{r+1}(F)$.

Natural thing to try: round-elimination. But it is probabilistic. So we show a new deterministic round-elimination theorem:

How to get the general k-round case?

We have shown that $L_3(f)$ is hard, so now we show a way of estimating $L_r(f)$ using an oracle for $L_{r+1}(F)$.
Natural thing to try: round-elimination. But it is probabilistic. So we show a new deterministic round-elimination theorem:

Definition

Let $d \geq 3$. Let $f : [a] \times [b] \to \{0,1\}$. Let
$F : ([k] \times [a]) \times ([k] \times [b] \times \{0,1\} \times \{0,1\}) \to \{0,1\}$ be given by

$$F(x_0, x_1; y_0, y_1, z, i) = \begin{cases} z & , \text{ if } i = 0,\ x_0 \neq y_0 \\ 1 - z & , \text{ if } i = 0,\ x_0 = y_0 \\ f(x_1; y_1) & , \text{ if } i = 1. \end{cases}$$

How to get the general k-round case?

We have shown that $L_3(f)$ is hard, so now we show a way of estimating $L_r(f)$ using an oracle for $L_{r+1}(F)$.
Natural thing to try: round-elimination. But it is probabilistic. So we show a new deterministic round-elimination theorem:

Definition

Let $d \geq 3$. Let $f : [a] \times [b] \to \{0,1\}$. Let
$F : ([k] \times [a]) \times ([k] \times [b] \times \{0,1\} \times \{0,1\}) \to \{0,1\}$ be given by

$$F(x_0, x_1; y_0, y_1, z, i) = \begin{cases} z & , \text{ if } i = 0,\ x_0 \neq y_0 \\ 1 - z & , \text{ if } i = 0,\ x_0 = y_0 \\ f(x_1; y_1) & , \text{ if } i = 1. \end{cases}$$

Theorem

We always have

$$\min\{2k, \mathrm{L}_d^{\mathrm{Bob}}(f) - O(1)\} \leq \mathrm{L}_{d+1}^{\mathrm{Alice}}(F) - 2k \leq \mathrm{L}_d^{\mathrm{Bob}}(f).$$

How to get the general k-round case?

Theorem

Given f and k, we can construct F such that

$$\min\{2k, \mathrm{L}_d^{\mathrm{Bob}}(f) - O(1)\} \leq \mathrm{L}_{d+1}^{\mathrm{Alice}}(F) - 2k \leq \mathrm{L}_d^{\mathrm{Bob}}(f).$$

How to get the general k-round case?

Theorem

Given f and k, we can construct F such that

$$\min\{2k, \mathrm{L}_d^{\mathrm{Bob}}(f) - O(1)\} \leq \mathrm{L}_{d+1}^{\mathrm{Alice}}(F) - 2k \leq \mathrm{L}_d^{\mathrm{Bob}}(f).$$

We then use an oracle to compute $\mathrm{L}_{d+1}^{\mathrm{Alice}}(F)$ for all suitable k, and from this we obtain an approximation for $\mathrm{L}_d^{\mathrm{Bob}}(f)$.

How to get the general k-round case?

Theorem

Given f and k, we can construct F such that

$$\min\{2k, \mathrm{L}_d^{\mathrm{Bob}}(f) - O(1)\} \leq \mathrm{L}_{d+1}^{\mathrm{Alice}}(F) - 2k \leq \mathrm{L}_d^{\mathrm{Bob}}(f).$$

We then use an oracle to compute $\mathrm{L}_{d+1}^{\mathrm{Alice}}(F)$ for all suitable k, and from this we obtain an approximation for $\mathrm{L}_d^{\mathrm{Bob}}(f)$.

Corollary

Let $0 < \epsilon < \frac{1}{8}$. Given an oracle computing a $(1+\epsilon)$-approximation of $\mathrm{L}_{d+1}^{\mathrm{Alice}}(\cdot)$ and a function $f : [a] \times [b] \to \{0,1\}$, one can deterministically compute a $(1+4\epsilon)$-approximation of $\mathrm{L}_d^{\mathrm{Bob}}(f)$ in time $(ab)^{O(1)}$

How to get the general k-round case?

Theorem

Given f and k, we can construct F such that

$$\min\{2k, \mathrm{L}_d^{\mathrm{Bob}}(f) - O(1)\} \leq \mathrm{L}_{d+1}^{\mathrm{Alice}}(F) - 2k \leq \mathrm{L}_d^{\mathrm{Bob}}(f).$$

We then use an oracle to compute $\mathrm{L}_{d+1}^{\mathrm{Alice}}(F)$ for all suitable k, and from this we obtain an approximation for $\mathrm{L}_d^{\mathrm{Bob}}(f)$.

Corollary

Let $0 < \epsilon < \frac{1}{8}$. Given an oracle computing a $(1+\epsilon)$-approximation of $\mathrm{L}_{d+1}^{\mathrm{Alice}}(\cdot)$ and a function $f : [a] \times [b] \to \{0,1\}$, one can deterministically compute a $(1+4\epsilon)$-approximation of $\mathrm{L}_d^{\mathrm{Bob}}(f)$ in time $(ab)^{O(1)}$

Corollary

$\mathrm{L}_d^{\mathrm{Alice}}(\cdot)$ *is hard to approx. for all $d \geq 4$, to within a factor $1 + \varepsilon_d$.*

Conclusion

- We have shown hardness of approximation for protocol size.

Conclusion

- We have shown hardness of approximation for protocol size. In the 3-round case, this is enough to get hardness of approximation for communication (which is roughly speaking log of the protocol size).

Conclusion

- We have shown hardness of approximation for protocol size. In the 3-round case, this is enough to get hardness of approximation for communication (which is roughly speaking log of the protocol size).
- Our hardness-of-approximation factor is much smaller for $r \geq 4$, which is not enough to get hardness for communication.

Conclusion

- We have shown hardness of approximation for protocol size. In the 3-round case, this is enough to get hardness of approximation for communication (which is roughly speaking log of the protocol size).
- Our hardness-of-approximation factor is much smaller for $r \geq 4$, which is not enough to get hardness for communication.
- What is the optimal HoA factor? We devised an algorithm which approximates $\mathsf{L}_3^{\mathsf{Alice}}(f)$ to within a factor of $N/\sqrt{\log n}$ (for a function $f : [N] \times [N] \to \{0, 1\}$). Nothing better is known.

Conclusion

- We have shown hardness of approximation for protocol size. In the 3-round case, this is enough to get hardness of approximation for communication (which is roughly speaking log of the protocol size).
- Our hardness-of-approximation factor is much smaller for $r \geq 4$, which is not enough to get hardness for communication.
- What is the optimal HoA factor? We devised an algorithm which approximates $\mathrm{L}_3^{\mathrm{Alice}}(f)$ to within a factor of $N/\sqrt{\log n}$ (for a function $f : [N] \times [N] \to \{0, 1\}$). Nothing better is known.
- In the unbounded-round case we have cryptographic hardness (Kushilevitz and Weinreb, 2009), but no NP-hardness.

Conclusion

- We have shown hardness of approximation for protocol size. In the 3-round case, this is enough to get hardness of approximation for communication (which is roughly speaking log of the protocol size).
- Our hardness-of-approximation factor is much smaller for $r \geq 4$, which is not enough to get hardness for communication.
- What is the optimal HoA factor? We devised an algorithm which approximates $\mathrm{L}_3^{\mathrm{Alice}}(f)$ to within a factor of $N/\sqrt{\log n}$ (for a function $f : [N] \times [N] \to \{0, 1\}$). Nothing better is known.
- In the unbounded-round case we have cryptographic hardness (Kushilevitz and Weinreb, 2009), but no NP-hardness.
- How about *randomized* communication complexity?

Conclusion

- We have shown hardness of approximation for protocol size. In the 3-round case, this is enough to get hardness of approximation for communication (which is roughly speaking log of the protocol size).
- Our hardness-of-approximation factor is much smaller for $r \geq 4$, which is not enough to get hardness for communication.
- What is the optimal HoA factor? We devised an algorithm which approximates $\mathrm{L}_3^{\mathrm{Alice}}(f)$ to within a factor of $N/\sqrt{\log n}$ (for a function $f : [N] \times [N] \to \{0,1\}$). Nothing better is known.
- In the unbounded-round case we have cryptographic hardness (Kushilevitz and Weinreb, 2009), but no NP-hardness.
- How about *randomized* communication complexity? Here we have some kind of convexity, can we leverage this to get decent approximation algorithms for $\mathrm{L}_{r,\varepsilon}^{\mathrm{Alice}}(f)$?

Conclusion

- We have shown hardness of approximation for protocol size. In the 3-round case, this is enough to get hardness of approximation for communication (which is roughly speaking log of the protocol size).
- Our hardness-of-approximation factor is much smaller for $r \geq 4$, which is not enough to get hardness for communication.
- What is the optimal HoA factor? We devised an algorithm which approximates $\mathrm{L}_3^{\mathrm{Alice}}(f)$ to within a factor of $N/\sqrt{\log n}$ (for a function $f : [N] \times [N] \to \{0,1\}$). Nothing better is known.
- In the unbounded-round case we have cryptographic hardness (Kushilevitz and Weinreb, 2009), but no NP-hardness.
- How about *randomized* communication complexity? Here we have some kind of convexity, can we leverage this to get decent approximation algorithms for $\mathrm{L}_{r,\varepsilon}^{\mathrm{Alice}}(f)$?

End of Part 2