The hardness of decision-tree complexity

FLAT Seminar

Bruno Loff
University of Lisbon

The complexity of complexity

Suppose you are given a description of a computational problem P

and you wish to find the "best" algorithm for solving P

in a certain computational model M .

What is the computational complexity of this "meta" task?

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- For example, a regular language P is given to you by way of a DFA D for deciding it, and you wish to find a smallest-possible DFA for deciding the same language. So the description D is a DFA, the model M is DFAs.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- For example, a regular language P is given to you by way of a DFA D for deciding it, and you wish to find a smallest-possible DFA for deciding the same language. So the description D is a DFA, the model M is DFAs.
- This problem can be solved in polynomial time, e.g. by using Moore's algorithm for automata minimization.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- For example, a regular language P is given to you by way of a DFA D for deciding it, and you wish to find a smallest-possible DFA for deciding the same language. So the description D is a DFA, the model M is DFAs.
- This problem can be solved in polynomial time, e.g. by using Moore's algorithm for automata minimization.
- However, if the language P is described to you by way of a *non-deterministic* finite automaton (NFA), finding the smallest NFA for computing P is PSPACE-hard (Jiang&Ravikumar, 1993).

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- For example, a regular language P is given to you by way of a DFA D for deciding it, and you wish to find a smallest-possible DFA for deciding the same language. So the description D is a DFA, the model M is DFAs.
- This problem can be solved in polynomial time, e.g. by using Moore's algorithm for automata minimization.
- However, if the language P is described to you by way of a *non-deterministic* finite automaton (NFA), finding the smallest NFA for computing P is PSPACE-hard (Jiang&Ravikumar, 1993).
- So this problem can be sometimes easy, sometimes hard, and there are many cases where the complexity of the above problem is unknown.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- The difficulty of the above problem depends not only on the model. Even for the same model M, the difficulty of the above problem will depend on how P is described.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- The difficulty of the above problem depends not only on the model. Even for the same model M, the difficulty of the above problem will depend on how P is described.
- The above problem does not necessarily become harder as the model M becomes more powerful. Soon I will give examples of computational models M_1 and M_2, where M_2 is more powerful than M_1, but finding optimal algorithms is possible for M_2 but (e.g.) NP-hard for M_1.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- The difficulty of the above problem depends not only on the model. Even for the same model M, the difficulty of the above problem will depend on how P is described.
- The above problem does not necessarily become harder as the model M becomes more powerful. Soon I will give examples of computational models M_1 and M_2, where M_2 is more powerful than M_1, but finding optimal algorithms is possible for M_2 but (e.g.) NP-hard for M_1.
- The answer also depends on the kind of problems we wish to understand. Soon I will give examples where we can answer the above question efficiently, e.g., for total functions, but the meta-problem becomes NP-hard for partial functions.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- So we have a fundamental theoretical question, and the answer to it depends on:

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- So we have a fundamental theoretical question, and the answer to it depends on:
  - The computational model M being looked at.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- So we have a fundamental theoretical question, and the answer to it depends on:
  - The computational model M being looked at.
  - The measure of complexity being used (what is "best").

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the “best” algorithm for solving P in a certain computational model M

- So we have a fundamental theoretical question, and the answer to it depends on:
  - The computational model M being looked at.
  - The measure of complexity being used (what is “best”).
  - The kind of problem P for which we wish to find good algorithms.

The complexity of complexity

Suppose you are given a description D of a computational problem P and you wish to find the "best" algorithm for solving P in a certain computational model M

- So we have a fundamental theoretical question, and the answer to it depends on:
  - The computational model M being looked at.
  - The measure of complexity being used (what is "best").
  - The kind of problem P for which we wish to find good algorithms.
  - How the problem P is described to us.

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*
DFA	size	language	DFA	in P	Moore ’53

Some known results

Model	Measure	Problem	Description	Hardness	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93
DNF	size	total f	truth-table	NP-complete	Masek '79

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93
DNF	size	total f	truth-table	NP-complete	Masek '79
DNF	log-approx size	total f	truth-table	in P	Masek '79

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93
DNF	size	total f	truth-table	NP-complete	Masek '79
DNF	log-approx size	total f	truth-table	in P	Masek '79
DNF	$o(\log)$-approx	total f	truth-table	NP-complete	AHMPS,KS'08

Some known results

Model	Measure	Problem	Description	Hardness	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93
DNF	size	total f	truth-table	NP-complete	Masek '79
DNF	log-approx size	total f	truth-table	in P	Masek '79
DNF	$o(\log)$-approx	total f	truth-table	NP-complete	AHMPS,KS'08
DNF	size	total f	DNF	σ_2^p-complete	Umans'01

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93
DNF	size	total f	truth-table	NP-complete	Masek '79
DNF	log-approx size	total f	truth-table	in P	Masek '79
DNF	$o(\log)$-approx	total f	truth-table	NP-complete	AHMPS,KS'08
DNF	size	total f	DNF	σ_2^p-complete	Umans'01
Det-CC	depth	partial f	matrix	NP-hard	LY'94

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93
DNF	size	total f	truth-table	NP-complete	Masek '79
DNF	log-approx size	total f	truth-table	in P	Masek '79
DNF	$o(\log)$-approx	total f	truth-table	NP-complete	AHMPS,KS'08
DNF	size	total f	DNF	σ_2^p-complete	Umans'01
Det-CC	depth	partial f	matrix	NP-hard	LY'94
Det-CC	depth	total f	matrix	NP-hard	HIL'24

Some known results

Model	**Measure**	**Problem**	**Description**	**Hardness**	*reference*
DFA	size	language	DFA	in P	Moore '53
NFA	size	language	NFA	PSPACE-complete	JR'93
DNF	size	total f	truth-table	NP-complete	Masek '79
DNF	log-approx size	total f	truth-table	in P	Masek '79
DNF	$o(\log)$-approx	total f	truth-table	NP-complete	AHMPS,KS'08
DNF	size	total f	DNF	σ_2^p-complete	Umans'01
Det-CC	depth	partial f	matrix	NP-hard	LY'94
Det-CC	depth	total f	matrix	NP-hard	HIL'24
PP-CC	depth	total f	matrix	in P	LS'09

Our results

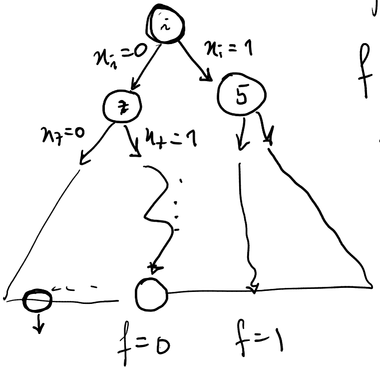
We understand the complexity of decision-tree complexity.

Decision trees

(explain what a decision-tree is)

input $x_1, \dots, x_n$

output $f(x_1 \dots x_n)$

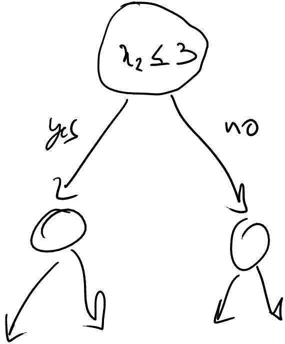


a decision-tree computes f if, for every leaf, $f(x)$ is the same for every input that would follow the path from root to that leaf

Previous results: the learning problem

(data space, queries, access to samples $(x, f(x))$, the goal is to produce a classifier)

x_1	x_2	x_3		obs
7	5	−2	...	0
				1
				⋮
				0
				0
				1
				⋮



Our setting: the algorithmic problem

Model	**Measure**	**Problem**	**Description**	**Hardness**
Det-DTs	depth	total f	circuit	?
Det-DTs	depth	total f	truth-table	?

- In our setting, we assume that we are given the *full description* of a total function $f : \{0,1\}^n \to \{0,1\}$, either as a truth-table, or succinctly as a Boolean circuit, and we wish to find a decision-tree for computing f that makes as few queries as possible. I.e.:

Our setting: the algorithmic problem

Model	Measure	Problem	Description	Hardness
Det-DTs	depth	total f	circuit	?
Det-DTs	depth	total f	truth-table	?

- In our setting, we assume that we are given the *full description* of a total function $f : \{0,1\}^n \to \{0,1\}$, either as a truth-table, or succinctly as a Boolean circuit, and we wish to find a decision-tree for computing f that makes as few queries as possible. I.e.:
  - Our model is decision-trees.

Our setting: the algorithmic problem

Model	Measure	Problem	Description	Hardness
Det-DTs	depth	total f	circuit	?
Det-DTs	depth	total f	truth-table	?

- In our setting, we assume that we are given the *full description* of a total function $f : \{0,1\}^n \to \{0,1\}$, either as a truth-table, or succinctly as a Boolean circuit, and we wish to find a decision-tree for computing f that makes as few queries as possible. I.e.:
  - Our model is decision-trees.
  - Our complexity measure is decision-tree depth (also known as *query complexity*).

Our setting: the algorithmic problem

Model	Measure	Problem	Description	Hardness
Det-DTs	depth	total f	circuit	?
Det-DTs	depth	total f	truth-table	?

- In our setting, we assume that we are given the *full description* of a total function $f : \{0,1\}^n \to \{0,1\}$, either as a truth-table, or succinctly as a Boolean circuit, and we wish to find a decision-tree for computing f that makes as few queries as possible. I.e.:
  - Our model is decision-trees.
  - Our complexity measure is decision-tree depth (also known as *query complexity*).
  - The given computational problem is a total Boolean function f,

Our setting: the algorithmic problem

Model	**Measure**	**Problem**	**Description**	**Hardness**
Det-DTs	depth	total f	circuit	?
Det-DTs	depth	total f	truth-table	?

- In our setting, we assume that we are given the *full description* of a total function $f : \{0,1\}^n \to \{0,1\}$, either as a truth-table, or succinctly as a Boolean circuit, and we wish to find a decision-tree for computing f that makes as few queries as possible. I.e.:
    - Our model is decision-trees.
    - Our complexity measure is decision-tree depth (also known as *query complexity*).
    - The given computational problem is a total Boolean function f,
    - which is either given as a truth-table, or as a circuit (we wish to understand both scenarios).

Our results

Theorem

The problem of computing the query complexity of f, when given the truth-table of f, is NC_1*-hard, and it can be computed by circuits of depth* $O(\log n \log\log n)$.

Our results

Theorem

The problem of computing the query complexity of f, when given the truth-table of f, is NC_1*-hard, and it can be computed by circuits of depth* $O(\log n \log\log n)$*.*

Theorem

The problem of computing the query complexity of f, when given a Boolean circuit for computing f, is PSPACE*-complete.*

Our results

Theorem

The problem of computing the query complexity of f, when given the truth-table of f, is NC_1-hard, and it can be computed by circuits of depth $O(\log n \log\log n)$.

Theorem

The problem of computing the query complexity of f, when given a Boolean circuit for computing f, is PSPACE-complete.

In this talk I will focus on the second result.

Our results

Theorem

The problem of computing the query complexity of f, when given the truth-table of f, is NC_1-hard, and it can be computed by circuits of depth $O(\log n \log \log n)$.

Theorem

The problem of computing the query complexity of f, when given a Boolean circuit for computing f, is PSPACE-complete.

In this talk I will focus on the second result.
The first result is proven by the same technique.

The upper-bound

Definition (formal definition of the meta-problem)

The circuit-DT problem is as follows:

The upper-bound

Definition (formal definition of the meta-problem)

The circuit-DT problem is as follows:

- We are given as input a Boolean circuit C over n inputs $x_1, \ldots, x_n$, computing some function $f : \{0,1\}^n \to \{0,1\}$.

The upper-bound

Definition (formal definition of the meta-problem)

The circuit-DT problem is as follows:

- We are given as input a Boolean circuit C over n inputs $x_1, \ldots, x_n$, computing some function $f : \{0,1\}^n \to \{0,1\}$.
- We wish to output a decision tree for f of minimal depth.

The upper-bound

Definition (formal definition of the meta-problem)

The circuit-DT problem is as follows:

- We are given as input a Boolean circuit C over n inputs $x_1, \ldots, x_n$, computing some function $f : \{0,1\}^n \to \{0,1\}$.
- We wish to output a decision tree for f of minimal depth.

Our main result is that circuit-DT is PSPACE-complete.

The upper-bound

Definition (formal definition of the meta-problem)

The circuit-DT problem is as follows:

- We are given as input a Boolean circuit C over n inputs $x_1, \ldots, x_n$, computing some function $f : \{0,1\}^n \to \{0,1\}$.
- We wish to output a decision tree for f of minimal depth.

Our main result is that circuit-DT is PSPACE-complete. Let us start with the upper-bound:

The upper-bound

Definition (formal definition of the meta-problem)

The circuit-DT problem is as follows:

- We are given as input a Boolean circuit C over n inputs $x_1, \ldots, x_n$, computing some function $f : \{0,1\}^n \to \{0,1\}$.
- We wish to output a decision tree for f of minimal depth.

Our main result is that circuit-DT is PSPACE-complete. Let us start with the upper-bound:

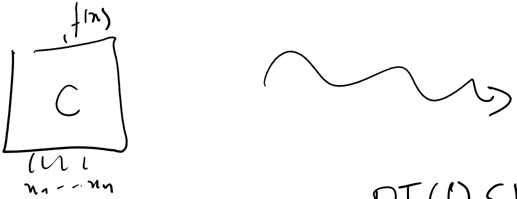
Theorem

circuit-DT $\in$ PSPACE.

Theorem

circuit-DT $\in$ PSPACE.

(DT$(f) \leq 0$ *iff f is constant, how about* DT$(f) \leq k$*?*)



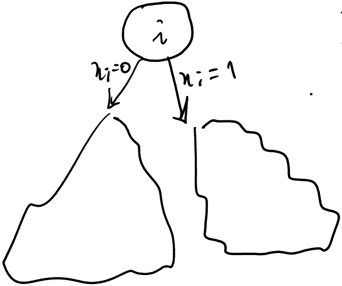
$\mathrm{DT}(f) = ?$

$$\mathrm{DT}(f) \leq k \iff$$

$$\exists_{i \in [n]} \; \mathrm{DT}(f|_{x_i=0}) \leq k-1$$

$$\wedge \; \mathrm{DT}(f|_{x_i=1}) \leq k-1$$

$$\iff \exists_{i \in [n]} \forall_{b \in \{0,1\}} \mathrm{DT}(f|_{x_i=b}) \leq k-1$$



Theorem

circuit-DT $\in$ PSPACE.

(DT$(f) \leq 0$ *iff* f *is constant, how about* DT$(f) \leq k$?)

The key difficulty: circuit-DT vs TQBF

(def. of TQBF, $\exists y_1 \forall x_1 \ldots$, TQBF as a two-player game vs circuit-DT as a two-player game)

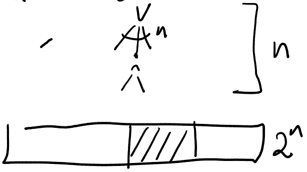
<u>TQBF</u>

input: $C(y_1, x_1, \ldots, y_n, x_n)$ a Boolean circuit

<u>output</u> whether or not

$$\exists y_1 \in \{0,1\} \; \forall x_1 \; \exists y_2 \; \forall x_2 \; \cdots \; \exists y_n \; \forall x_n \quad C(y_1, x_1, \ldots) = 1$$

vs Circuit-DT $(DT(f) \leq k)$



<u>input</u> $C(z_1, \ldots, z_n)$

<u>output</u>

$$\exists_{i_1 \in [n]} \; \forall z_{i_1} \; \exists_{i_2} \; \forall z_{i_2} \; \cdots \; \exists_{i_k} \; \forall z_k \quad C|_{z_{i_1} \cdots z_{i_k}} \text{ constant}$$

Theorem

TQBF $\leq_p$ circuit-DT, *hence* circuit-DT *is* PSPACE-*complete.*

Theorem

TQBF $\leq_p$ circuit-DT, *hence* circuit-DT *is* PSPACE-*complete.*

- I will not prove the full theorem, but I will prove an important auxiliary lemma.

Theorem

TQBF $\leq_p$ circuit-DT, *hence* circuit-DT *is* PSPACE-*complete.*

- I will not prove the full theorem, but I will prove an important auxiliary lemma.
- This will give some idea of how one forces a circuit-DT game to behave like a TQBF game.

Consider the following function:

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

where:

$$f_i = y_i \wedge y_i'$$

$$g_i = \begin{cases} x_i & \text{if } \overbrace{f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i} = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

Consider the following function:

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

where:

$$f_i = y_i \wedge y_i'$$
$$g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- So g_i depends on the variables $y_1, y_1', x_1, x_1', \dots y_i, y_i', x_i, x_i'$.

Consider the following function:

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

where:

$$f_i = y_i \wedge y_i'$$
$$g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- So g_i depends on the variables $y_1, y_1', x_1, x_1', \dots y_i, y_i', x_i, x_i'$.
- What is the decision-tree depth of F_n?

Consider the following function:

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

where:

$$f_i = y_i \wedge y_i'$$
$$g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- So g_i depends on the variables $y_1, y_1', x_1, x_1', \ldots y_i, y_i', x_i, x_i'$.
- What is the decision-tree depth of F_n?
- How might Alice and Bob play the circuit-DT game on F_n?

Consider the following function:

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

where:

$$f_i = y_i \wedge y_i'$$
$$g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- So g_i depends on the variables $y_1, y_1', x_1, x_1', \dots y_i, y_i', x_i, x_i'$.
- What is the decision-tree depth of F_n?
- How might Alice and Bob play the circuit-DT game on F_n?
- Remember, in the circuit-DT game, Alice chooses a variable, and Bob sets that variable to some value.

Consider the following function:

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

where:

$$f_i = y_i \wedge y_i'$$
$$g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- So g_i depends on the variables $y_1, y_1', x_1, x_1', \ldots y_i, y_i', x_i, x_i'$.
- What is the decision-tree depth of F_n?
- How might Alice and Bob play the circuit-DT game on F_n?
- Remember, in the circuit-DT game, Alice chooses a variable, and Bob sets that variable to some value.
- Bob's goal is to make the game last as long as possible before the function becomes constant.

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = \underbrace{y_i \wedge y_i'}_{} \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.
- If Bob sets the chosen y variable to $\cancel{1}$ 0, then there is no need to check the other variable to know that $f_1 = 0$.

$$F_n(y_1, y'_1, x_1, x'_1, \dots y_n, y'_n, x_n, x'_n) := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y'_i \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x'_i & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y'_1.
- If Bob sets the chosen y variable to 1, then there is no need to check the other variable to know that $f_1 = 0$. So Alice didn't need to ask about the other y variable, and she has saved one query.

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.
- If Bob sets the chosen y variable to 1, then there is no need to check the other variable to know that $f_1 = 0$. So Alice didn't need to ask about the other y variable, and she has saved one query.
- This forces Bob to set the chosen variable to 1.

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.
- If Bob sets the chosen y variable to 1, then there is no need to check the other variable to know that $f_1 = 0$. So Alice didn't need to ask about the other y variable, and she has saved one query.
- This forces Bob to set the chosen variable to 1.
- So f_1 is not yet fixed.

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.
- If Bob sets the chosen y variable to 1, then there is no need to check the other variable to know that $f_1 = 0$. So Alice didn't need to ask about the other y variable, and she has saved one query.
- This forces Bob to set the chosen variable to 1.
- So f_1 is not yet fixed. Alice then asks about the other variable.

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.
- If Bob sets the chosen y variable to 1, then there is no need to check the other variable to know that $f_1 = 0$. So Alice didn't need to ask about the other y variable, and she has saved one query.
- This forces Bob to set the chosen variable to 1.
- So f_1 is not yet fixed. Alice then asks about the other variable.
- By some other part of the construction which I will soon sketch, Bob will be forced to answer 0 to the second query.

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.
- If Bob sets the chosen y variable to 1, then there is no need to check the other variable to know that $f_1 = 0$. So Alice didn't need to ask about the other y variable, and she has saved one query.
- This forces Bob to set the chosen variable to 1.
- So f_1 is not yet fixed. Alice then asks about the other variable.
- By some other part of the construction which I will soon sketch, Bob will be forced to answer 0 to the second query.
- So by asking variables in the right order, Alice can force Bob to set y_1 to 1 or 0, as she desires.

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- Suppose Alice chooses either y_1 or y_1'.
- If Bob sets the chosen y variable to 1, then there is no need to check the other variable to know that $f_1 = 0$. So Alice didn't need to ask about the other y variable, and she has saved one query.
- This forces Bob to set the chosen variable to 1.
- So f_1 is not yet fixed. Alice then asks about the other variable.
- By some other part of the construction which I will soon sketch, Bob will be forced to answer 0 to the second query.
- So by asking variables in the right order, Alice can force Bob to set y_1 to 1 or 0, as she desires.
- This is exactly the power that Alice has in the TQBF game.

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- But why should Alice be forced to ask about y_1 and y_1', at all.

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- But why should Alice be forced to ask about y_1 and y_1', at all.
- Maybe she will do something else, something crazy.

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- But why should Alice be forced to ask about y_1 and y_1', at all.
- Maybe she will do something else, something crazy.
- However, if she did not ask about y_1 and y_1', she does not know the value of f_1, and so she does not know which variable x_1 or x_1' is relevant for g_1.

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- But why should Alice be forced to ask about y_1 and y_1', at all.
- Maybe she will do something else, something crazy.
- However, if she did not ask about y_1 and y_1', she does not know the value of f_1, and so she does not know which variable x_1 or x_1' is relevant for g_1. So she must ask both.

$$F_n(y_1, y_1', x_1, x_1', \dots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- But why should Alice be forced to ask about y_1 and y_1', at all.
- Maybe she will do something else, something crazy.
- However, if she did not ask about y_1 and y_1', she does not know the value of f_1, and so she does not know which variable x_1 or x_1' is relevant for g_1. So she must ask both.
- Using this kind of reasoning, it can be shown that an optimal Alice playing the DT-game must first ask y_1 and y_1' in any order, and then the relevant x (x_1 or x_1'), ...

$$F_n(y_1, y'_1, x_1, x'_1, \dots y_n, y'_n, x_n, x'_n) := f_1 \oplus g_1 \oplus \dots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y'_i \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \dots \oplus g_{i-1} \oplus f_i = 1 \\ x'_i & \text{otherwise.} \end{cases}$$

- But why should Alice be forced to ask about y_1 and y'_1, at all.
- Maybe she will do something else, something crazy.
- However, if she did not ask about y_1 and y'_1, she does not know the value of f_1, and so she does not know which variable x_1 or x'_1 is relevant for g_1. So she must ask both.
- Using this kind of reasoning, it can be shown that an optimal Alice playing the DT-game must first ask y_1 and y'_1 in any order, and then the relevant x (x_1 or x'_1), ...
- This kind of construction, with several more non-obvious tricks, eventually allows us to construct a DT-game where all optimal strategies of Alice and Bob behave like optimal strategies of a given TQBF instance.

$$F_n(y_1, y_1', x_1, x_1', \ldots y_n, y_n', x_n, x_n') := f_1 \oplus g_1 \oplus \ldots \oplus f_n \oplus g_n,$$

$$f_i = y_i \wedge y_i' \qquad g_i = \begin{cases} x_i & \text{if } f_1 \oplus g_1 \oplus \ldots \oplus g_{i-1} \oplus f_i = 1 \\ x_i' & \text{otherwise.} \end{cases}$$

- But why should Alice be forced to ask about y_1 and y_1', at all.
- Maybe she will do something else, something crazy.
- However, if she did not ask about y_1 and y_1', she does not know the value of f_1, and so she does not know which variable x_1 or x_1' is relevant for g_1. So she must ask both.
- Using this kind of reasoning, it can be shown that an optimal Alice playing the DT-game must first ask y_1 and y_1' in any order, and then the relevant x (x_1 or x_1'), ...
- This kind of construction, with several more non-obvious tricks, eventually allows us to construct a DT-game where all optimal strategies of Alice and Bob behave like optimal strategies of a given TQBF instance.
- Let me briefly show you the final construction.

Conclusion

Conclusion

- The fundamental meta-complexity problem — *how hard is it to find an optimal (or near-optimal) algorithm?* is wide open in many setting, including many computational models on which we have a good chance at solving the problem.

Conclusion

- The fundamental meta-complexity problem — *how hard is it to find an optimal (or near-optimal) algorithm?* is wide open in many setting, including many computational models on which we have a good chance at solving the problem.
- E.g. *size* of deterministic decision trees, size/depth of randomized decision trees, parity decision-trees, approximating communication complexity of total functions, randomized communication complexity, ...

Conclusion

- The fundamental meta-complexity problem — *how hard is it to find an optimal (or near-optimal) algorithm?* is wide open in many setting, including many computational models on which we have a good chance at solving the problem.
- E.g. *size* of deterministic decision trees, size/depth of randomized decision trees, parity decision-trees, approximating communication complexity of total functions, randomized communication complexity, ...
- It is also open for various computational models, such as Boolean formula depth, Boolean circuit size, *etc* where a proof of NP-hardness would have very dramatic consequences.

Conclusion

- The fundamental meta-complexity problem — *how hard is it to find an optimal (or near-optimal) algorithm?* is wide open in many setting, including many computational models on which we have a good chance at solving the problem.
- E.g. *size* of deterministic decision trees, size/depth of randomized decision trees, parity decision-trees, approximating communication complexity of total functions, randomized communication complexity, . . .
- It is also open for various computational models, such as Boolean formula depth, Boolean circuit size, *etc* where a proof of NP-hardness would have very dramatic consequences.
- Ultimately I am interested even in the weaker question: If a computational problem is hard, does there exist a short *proof* that it is hard? (I.e. is our problem in coNP).

Conclusion

- The fundamental meta-complexity problem — *how hard is it to find an optimal (or near-optimal) algorithm?* is wide open in many setting, including many computational models on which we have a good chance at solving the problem.
- E.g. *size* of deterministic decision trees, size/depth of randomized decision trees, parity decision-trees, approximating communication complexity of total functions, randomized communication complexity, ...
- It is also open for various computational models, such as Boolean formula depth, Boolean circuit size, *etc* where a proof of NP-hardness would have very dramatic consequences.
- Ultimately I am interested even in the weaker question: If a computational problem is hard, does there exist a short *proof* that it is hard? (I.e. is our problem in coNP).

For the full text:

`https://eccc.weizmann.ac.il/report/2024/034/`

Thank you!