The Hardness of Decision Tree Complexity

Bruno Loff, Alexey Milovanov

University of Lisbon

STACS 2025
6 March 2025

Formulation of a problem

- Let f be some Boolean function.
- The deterministic query complexity of f is the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. Denote it as $\mathrm{D}(f)$.
- How difficult is to find $\mathrm{D}(f)$?
- The answer depends on the way how f is given.
- **tt-DT** We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.
- **circuit-DT** We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

Formulation of a problem

- Let f be some Boolean function.
- The deterministic query complexity of f is the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. Denote it as $\mathrm{D}(f)$.
- How difficult is to find $\mathrm{D}(f)$?
- The answer depends on the way how f is given.
- **tt-DT** We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.
- **circuit-DT** We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

Formulation of a problem

- Let f be some Boolean function.
- The deterministic query complexity of f is the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. Denote it as $\mathrm{D}(f)$.
- How difficult is to find $\mathrm{D}(f)$?
- The answer depends on the way how f is given.
- **tt-DT** We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.
- **circuit-DT** We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

Formulation of a problem

- Let f be some Boolean function.
- The deterministic query complexity of f is the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. Denote it as $\mathrm{D}(f)$.
- How difficult is to find $\mathrm{D}(f)$?
- The answer depends on the way how f is given.
- **tt-DT** We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.
- **circuit-DT** We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

Formulation of a problem

- Let f be some Boolean function.
- The deterministic query complexity of f is the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. Denote it as $\mathrm{D}(f)$.
- How difficult is to find $\mathrm{D}(f)$?
- The answer depends on the way how f is given.
- **tt-DT** We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.
- **circuit-DT** We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

Formulation of a problem

- Let f be some Boolean function.
- The deterministic query complexity of f is the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. Denote it as $\mathrm{D}(f)$.
- How difficult is to find $\mathrm{D}(f)$?
- The answer depends on the way how f is given.
- **tt-DT** We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.
- **circuit-DT** We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

Formulation of a problem

- Let f be some Boolean function.
- The deterministic query complexity of f is the smallest depth of a deterministic decision tree that computes $f(x)$ by querying the bits of x. Denote it as $\mathrm{D}(f)$.
- How difficult is to find $\mathrm{D}(f)$?
- The answer depends on the way how f is given.
- **tt-DT** We are given f as a truth table, meaning a binary string of length $N = 2^n$ so that $f(x)$ appears at the x-th position.
- **circuit-DT** We are given f as a Boolean circuit, which potentially allows for a more succinct encoding of f.

Upper bounds: tt-DT belongs to P

Theorem

tt-DT belongs to P. More precisely, there is an algorithm that computes the DT-complexity of an n-ary Boolean function in time $O(3^n \cdot n) = O(N^{1.585\ldots} \log N)$, where $N = 2^n$.

Proof.

- For any partial assignment $\rho \in \{0,1,*\}^n$, $\mathrm{D}(f|_\rho) = 0$ if f is constant on ρ, and otherwise

$$\mathrm{D}(f|_\rho) = \min_{i \in \rho^{-1}(*)} \{1 + \max_{b \in \{0,1\}} \mathrm{D}(f|_{\rho \cdot [i \leftarrow b]})\}.$$

- This gives us a dynamic programming algorithm.
- There are 3^n partial assignments in total, and each computation $\mathrm{D}(f|_\rho)$ takes time $O(n)$ in a RAM.

□

Upper bounds: tt-DT belongs to P

Theorem

tt-DT belongs to P. More precisely, there is an algorithm that computes the DT-complexity of an n-ary Boolean function in time $O(3^n \cdot n) = O(N^{1.585\ldots} \log N)$*, where* $N = 2^n$*.*

Proof.

- For any partial assignment $\rho \in \{0, 1, *\}^n$, $\mathrm{D}(f|_\rho) = 0$ if f is constant on ρ, and otherwise

$$\mathrm{D}(f|_\rho) = \min_{i \in \rho^{-1}(*)} \{1 + \max_{b \in \{0,1\}} \mathrm{D}(f|_{\rho \cdot [i \leftarrow b]})\}.$$

- This gives us a dynamic programming algorithm.
- There are 3^n partial assignments in total, and each computation $\mathrm{D}(f|_\rho)$ takes time $O(n)$ in a RAM.

□

Upper bounds: tt-DT belongs to P

Theorem

tt-DT belongs to P. More precisely, there is an algorithm that computes the DT-complexity of an n-ary Boolean function in time $O(3^n \cdot n) = O(N^{1.585\ldots} \log N)$, where $N = 2^n$.

Proof.

- For any partial assignment $\rho \in \{0, 1, *\}^n$, $\mathrm{D}(f|_\rho) = 0$ if f is constant on ρ, and otherwise

$$\mathrm{D}(f|_\rho) = \min_{i \in \rho^{-1}(*)} \{1 + \max_{b \in \{0,1\}} \mathrm{D}(f|_{\rho \cdot [i \leftarrow b]})\}.$$

- This gives us a dynamic programming algorithm.
- There are 3^n partial assignments in total, and each computation $\mathrm{D}(f|_\rho)$ takes time $O(n)$ in a RAM.

□

Upper bounds: tt-DT belongs to P

Theorem

tt-DT belongs to P. More precisely, there is an algorithm that computes the DT-complexity of an n-ary Boolean function in time $O(3^n \cdot n) = O(N^{1.585\ldots} \log N)$*, where* $N = 2^n$*.*

Proof.

- For any partial assignment $\rho \in \{0, 1, *\}^n$, $\mathrm{D}(f|_\rho) = 0$ if f is constant on ρ, and otherwise

$$\mathrm{D}(f|_\rho) = \min_{i \in \rho^{-1}(*)} \{1 + \max_{b \in \{0,1\}} \mathrm{D}(f|_{\rho \cdot [i \leftarrow b]})\}.$$

- This gives us a dynamic programming algorithm.
- There are 3^n partial assignments in total, and each computation $\mathrm{D}(f|_\rho)$ takes time $O(n)$ in a RAM.

□

Game reformulation of $\mathrm{D}(f) \leq k$

- Consider the following game between Alice and Bob.
- The game lasts for k steps.
- At every step, Alice chooses a variable x_i, and Bob sets $x_i = 0$ or $x_i = 1$.
- After k steps, Alice wins if $f|_\rho$ is constant on the partial assignment corresponding to Alice and Bob's moves; otherwise, Bob wins.
- Alice has a winning strategy in this game iff $\mathrm{D}(f) \leq k$.
- Indeed, if $D(f) \leq k$, then Alice can make moves according to the corresponding tree.
- If $D(f) > k$ then Bob's strategy is to repeatedly choose the value $b \in \{0, 1\}$ that maximizes $\mathrm{D}(f|_{[i \leftarrow b]})$.

Game reformulation of $\mathrm{D}(f) \leq k$

- Consider the following game between Alice and Bob.
- The game lasts for k steps.
- At every step, Alice chooses a variable x_i, and Bob sets $x_i = 0$ or $x_i = 1$.
- After k steps, Alice wins if $f|_\rho$ is constant on the partial assignment corresponding to Alice and Bob's moves; otherwise, Bob wins.
- Alice has a winning strategy in this game iff $\mathrm{D}(f) \leq k$.
- Indeed, if $D(f) \leq k$, then Alice can make moves according to the corresponding tree.
- If $D(f) > k$ then Bob's strategy is to repeatedly choose the value $b \in \{0, 1\}$ that maximizes $\mathrm{D}(f|_{[i \leftarrow b]})$.

Game reformulation of $\mathrm{D}(f) \leq k$

- Consider the following game between Alice and Bob.
- The game lasts for k steps.
- At every step, Alice chooses a variable x_i, and Bob sets $x_i = 0$ or $x_i = 1$.
- After k steps, Alice wins if $f|_\rho$ is constant on the partial assignment corresponding to Alice and Bob's moves; otherwise, Bob wins.
- Alice has a winning strategy in this game iff $\mathrm{D}(f) \leq k$.
- Indeed, if $D(f) \leq k$, then Alice can make moves according to the corresponding tree.
- If $D(f) > k$ then Bob's strategy is to repeatedly choose the value $b \in \{0, 1\}$ that maximizes $\mathrm{D}(f|_{[i \leftarrow b]})$.

Game reformulation of $\mathrm{D}(f) \leq k$

- Consider the following game between Alice and Bob.
- The game lasts for k steps.
- At every step, Alice chooses a variable x_i, and Bob sets $x_i = 0$ or $x_i = 1$.
- After k steps, Alice wins if $f|_\rho$ is constant on the partial assignment corresponding to Alice and Bob's moves; otherwise, Bob wins.
- Alice has a winning strategy in this game iff $\mathrm{D}(f) \leq k$.
- Indeed, if $D(f) \leq k$, then Alice can make moves according to the corresponding tree.
- If $D(f) > k$ then Bob's strategy is to repeatedly choose the value $b \in \{0, 1\}$ that maximizes $\mathrm{D}(f|_{[i \leftarrow b]})$.

Game reformulation of $\mathrm{D}(f) \leq k$

- Consider the following game between Alice and Bob.
- The game lasts for k steps.
- At every step, Alice chooses a variable x_i, and Bob sets $x_i = 0$ or $x_i = 1$.
- After k steps, Alice wins if $f|_\rho$ is constant on the partial assignment corresponding to Alice and Bob's moves; otherwise, Bob wins.
- Alice has a winning strategy in this game iff $\mathrm{D}(f) \leq k$.
- Indeed, if $D(f) \leq k$, then Alice can make moves according to the corresponding tree.
- If $D(f) > k$ then Bob's strategy is to repeatedly choose the value $b \in \{0, 1\}$ that maximizes $\mathrm{D}(f|_{[i \leftarrow b]})$.

Game reformulation of $\mathrm{D}(f) \leq k$

- Consider the following game between Alice and Bob.
- The game lasts for k steps.
- At every step, Alice chooses a variable x_i, and Bob sets $x_i = 0$ or $x_i = 1$.
- After k steps, Alice wins if $f|_\rho$ is constant on the partial assignment corresponding to Alice and Bob's moves; otherwise, Bob wins.
- Alice has a winning strategy in this game iff $\mathrm{D}(f) \leq k$.
- Indeed, if $D(f) \leq k$, then Alice can make moves according to the corresponding tree.
- If $D(f) > k$ then Bob's strategy is to repeatedly choose the value $b \in \{0, 1\}$ that maximizes $\mathrm{D}(f|_{[i \leftarrow b]})$.

Game reformulation of $\mathrm{D}(f) \leq k$

- Consider the following game between Alice and Bob.
- The game lasts for k steps.
- At every step, Alice chooses a variable x_i, and Bob sets $x_i = 0$ or $x_i = 1$.
- After k steps, Alice wins if $f|_\rho$ is constant on the partial assignment corresponding to Alice and Bob's moves; otherwise, Bob wins.
- Alice has a winning strategy in this game iff $\mathrm{D}(f) \leq k$.
- Indeed, if $D(f) \leq k$, then Alice can make moves according to the corresponding tree.
- If $D(f) > k$ then Bob's strategy is to repeatedly choose the value $b \in \{0, 1\}$ that maximizes $\mathrm{D}(f|_{[i \leftarrow b]})$.

Upper bounds: circuit-DT belongs to PSPACE

One can algorithmically find the winner in the game by a simple recursive algorithm.

Theorem

circuit-DT belongs to PSPACE.

Denote by $\widetilde{\mathrm{NC}^1}$ the class of functions $f : \{0,1\}^n \to \{0,1\}^m$ computable by Boolean circuits (with binary AND and OR gates, and unary NOT gates) in depth $O(\log n \cdot \log\log n)$ and size (n, m).

Theorem

tt-DT is in $\widetilde{NC^1}$.

Proof.

One can implement the algorithm for tt-DT is in P. □

Upper bounds: circuit-DT belongs to PSPACE

One can algorithmically find the winner in the game by a simple recursive algorithm.

Theorem

circuit-DT belongs to PSPACE.

Denote by $\widetilde{\mathrm{NC}^1}$ the class of functions $f : \{0,1\}^n \to \{0,1\}^m$ computable by Boolean circuits (with binary AND and OR gates, and unary NOT gates) in depth $O(\log n \cdot \log\log n)$ and size (n, m).

Theorem

tt-DT is in $\widetilde{NC^1}$.

Proof.

One can implement the algorithm for tt-DT is in P. □

Upper bounds: circuit-DT belongs to PSPACE

One can algorithmically find the winner in the game by a simple recursive algorithm.

Theorem

circuit-DT belongs to PSPACE.

Denote by $\widetilde{\mathrm{NC}^1}$ the class of functions $f : \{0,1\}^n \to \{0,1\}^m$ computable by Boolean circuits (with binary AND and OR gates, and unary NOT gates) in depth $O(\log n \cdot \log \log n)$ and size (n, m).

Theorem

tt-DT is in $\widetilde{NC^1}$.

Proof.

One can implement the algorithm for tt-DT is in P. □

Upper bounds: circuit-DT belongs to PSPACE

One can algorithmically find the winner in the game by a simple recursive algorithm.

Theorem

circuit-DT belongs to PSPACE.

Denote by $\widetilde{\mathrm{NC}^1}$ the class of functions $f : \{0,1\}^n \to \{0,1\}^m$ computable by Boolean circuits (with binary AND and OR gates, and unary NOT gates) in depth $O(\log n \cdot \log\log n)$ and size (n, m).

Theorem

tt-DT is in $\widetilde{NC^1}$.

Proof.

One can implement the algorithm for tt-DT is in P. □

Upper bounds: circuit-DT belongs to PSPACE

One can algorithmically find the winner in the game by a simple recursive algorithm.

Theorem

circuit-DT belongs to PSPACE.

Denote by $\widetilde{\mathrm{NC}}^1$ the class of functions $f : \{0,1\}^n \to \{0,1\}^m$ computable by Boolean circuits (with binary AND and OR gates, and unary NOT gates) in depth $O(\log n \cdot \log\log n)$ and size (n, m).

Theorem

tt-DT is in $\widetilde{NC}^1$.

Proof.

One can implement the algorithm for tt-DT is in P. □

Our main results are the following.

Theorem

circuit-DT is PSPACE-hard under polynomial-time reductions.

Theorem

tt-DT is NC^1-hard under NC^0-reduction.

We say that $A \leq_{\mathrm{NC}^0} B$, if there is a simply (namely, DLOGTIME-uniform) family of NC^0-circuits C_n such that, for every $x \in \{0,1\}^n$, $x \in A$ iff $C_n(x) \in B$.

Our main results are the following.

Theorem

circuit-DT is PSPACE-hard under polynomial-time reductions.

Theorem

tt-DT is NC^1*-hard under* NC^0*-reduction.*

We say that $A \leq_{\mathrm{NC}^0} B$, if there is a simply (namely, DLOGTIME-uniform) family of NC^0-circuits C_n such that, for every $x \in \{0, 1\}^n$, $x \in A$ iff $C_n(x) \in B$.

Lower bounds

Our main results are the following.

Theorem

circuit-DT is PSPACE-hard under polynomial-time reductions.

Theorem

tt-DT is NC^1*-hard under* NC^0*-reduction.*

We say that $A \leq_{\mathrm{NC}^0} B$, if there is a simply (namely, DLOGTIME-uniform) family of NC^0-circuits C_n such that, for every $x \in \{0,1\}^n$, $x \in A$ iff $C_n(x) \in B$.

Other NC^1-hard problems

The S_5 *identity problem*, S_5IP, is the problem of deciding if the product of given permutations from S_5 is equal to the identity.

Theorem (Barrington)

Then S_5IP is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.

tt-TQBF We are given as input a Boolean function $h : \{0,1\}^{2n} \to \{0,1\}$ as a truth table, and wish to know whether it holds:

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \ldots \exists y_n \forall x_n h(y_1, x_1, y_2, x_2, \ldots y_n, x_n),$$

Theorem

tt-TQBF *is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.*

We reduce TQBF (tt-TQBF) to circuit-DT (tt-DT) to prove our main result.

The S_5 *identity problem*, S_5IP, is the problem of deciding if the product of given permutations from S_5 is equal to the identity.

Theorem (Barrington)

Then S_5IP is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.

tt-TQBF We are given as input a Boolean function $h : \{0,1\}^{2n} \to \{0,1\}$ as a truth table, and wish to know whether it holds:

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \dots \exists y_n \forall x_n h(y_1, x_1, y_2, x_2, \dots y_n, x_n),$$

Theorem

tt-TQBF *is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.*

We reduce TQBF (tt-TQBF) to circuit-DT (tt-DT) to prove our main result.

Other NC^1-hard problems

The S_5 *identity problem*, S_5IP, is the problem of deciding if the product of given permutations from S_5 is equal to the identity.

Theorem (Barrington)

Then S_5IP is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.

tt-TQBF We are given as input a Boolean function $h : \{0,1\}^{2n} \to \{0,1\}$ as a truth table, and wish to know whether it holds:

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \ldots \exists y_n \forall x_n h(y_1, x_1, y_2, x_2, \ldots y_n, x_n),$$

Theorem

tt-TQBF *is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.*

We reduce TQBF (tt-TQBF) to circuit-DT (tt-DT) to prove our main result.

Other NC^1-hard problems

The S_5 *identity problem*, S_5IP, is the problem of deciding if the product of given permutations from S_5 is equal to the identity.

Theorem (Barrington)

Then S_5IP is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.

tt-TQBF We are given as input a Boolean function $h : \{0,1\}^{2n} \to \{0,1\}$ as a truth table, and wish to know whether it holds:

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \dots \exists y_n \forall x_n h(y_1, x_1, y_2, x_2, \dots y_n, x_n),$$

Theorem

tt-TQBF *is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.*

We reduce TQBF (tt-TQBF) to circuit-DT (tt-DT) to prove our main result.

Other NC^1-hard problems

The S_5 *identity problem*, S_5IP, is the problem of deciding if the product of given permutations from S_5 is equal to the identity.

Theorem (Barrington)

Then S_5IP is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.

tt-TQBF We are given as input a Boolean function $h : \{0,1\}^{2n} \to \{0,1\}$ as a truth table, and wish to know whether it holds:

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \ldots \exists y_n \forall x_n h(y_1, x_1, y_2, x_2, \ldots y_n, x_n),$$

Theorem

tt-TQBF *is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.*

We reduce TQBF (tt-TQBF) to circuit-DT (tt-DT) to prove our main result.

Other NC^1-hard problems

The S_5 *identity problem*, S_5IP, is the problem of deciding if the product of given permutations from S_5 is equal to the identity.

Theorem (Barrington)

Then S_5IP is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.

tt-TQBF We are given as input a Boolean function $h : \{0,1\}^{2n} \to \{0,1\}$ as a truth table, and wish to know whether it holds:

$$\exists y_1 \forall x_1 \exists y_2 \forall x_2 \ldots \exists y_n \forall x_n h(y_1, x_1, y_2, x_2, \ldots y_n, x_n),$$

Theorem

tt-TQBF *is NC^1-complete under $\leq_{\mathrm{NC}^0}$ reductions.*

We reduce TQBF (tt-TQBF) to circuit-DT (tt-DT) to prove our main result.

Open problems

1. What is the exact time-complexity of tt-DT? Is it possible to improve $O(3^n n)$-algorithm ? Is it possible to prove any non-trivial bounds (for example, under the Exponential Time Hypothesis)?
2. Is it possible to improve the $O(\log N \log \log N)$-depth bound of tt-DT?
3. What is the exact time, space, and circuit complexity of the problem of finding the minimum *size* of a decision tree that computes a given Boolean function?
4. What can we say about the problem of *approximating* DT complexity?

Open problems

1. What is the exact time-complexity of tt-DT? Is it possible to improve $O(3^n n)$-algorithm ? Is it possible to prove any non-trivial bounds (for example, under the Exponential Time Hypothesis)?
2. Is it possible to improve the $O(\log N \log\log N)$-depth bound of tt-DT?
3. What is the exact time, space, and circuit complexity of the problem of finding the minimum *size* of a decision tree that computes a given Boolean function?
4. What can we say about the problem of *approximating* DT complexity?

Open problems

1. What is the exact time-complexity of tt-DT? Is it possible to improve $O(3^n n)$-algorithm ? Is it possible to prove any non-trivial bounds (for example, under the Exponential Time Hypothesis)?
2. Is it possible to improve the $O(\log N \log\log N)$-depth bound of tt-DT?
3. What is the exact time, space, and circuit complexity of the problem of finding the minimum *size* of a decision tree that computes a given Boolean function?
4. What can we say about the problem of *approximating* DT complexity?

Open problems

1. What is the exact time-complexity of tt-DT? Is it possible to improve $O(3^n n)$-algorithm ? Is it possible to prove any non-trivial bounds (for example, under the Exponential Time Hypothesis)?
2. Is it possible to improve the $O(\log N \log\log N)$-depth bound of tt-DT?
3. What is the exact time, space, and circuit complexity of the problem of finding the minimum *size* of a decision tree that computes a given Boolean function?
4. What can we say about the problem of *approximating* DT complexity?

Open problems

1. What is the exact time-complexity of tt-DT? Is it possible to improve $O(3^n n)$-algorithm ? Is it possible to prove any non-trivial bounds (for example, under the Exponential Time Hypothesis)?
2. Is it possible to improve the $O(\log N \log\log N)$-depth bound of tt-DT?
3. What is the exact time, space, and circuit complexity of the problem of finding the minimum *size* of a decision tree that computes a given Boolean function?
4. What can we say about the problem of *approximating* DT complexity?

Thank you!