Computational complexity and the Tarski search problem

Master's thesis

Kristóf Zólomy

MSc Matemathics

Supervisor:

Dömötör Pálvölgyi
Department of Computer Science
Eötvös Loránd University, Faculty of Science

Eötvös Loránd University
Faculty of Science
2023

Contents

Acknowledgements

I would like to thank my supervisor, Dömötör Pálvölgyi, for introducing me to the topic of computational complexities, and guiding me through the completion of this thesis. His answers, ideas, insightful remarks and reviews of my work were invaluable for my progress and development.

Introduction

The complexity of a computational problem is the amount of resources an optimal algorithm needs to use in order to solve it. The classical model for determining complexity defines it with Turing machines: assuming we have a Boolean function $f : \{0,1\}^n \to \{0,1\}$, the task is to construct a Turing machine, which given input $x \in \{0,1\}^n$ outputs $f(x)$, and the amount of steps it takes for the worst case input is as low as possible. This is the time complexity of the function f. If the critical resource is memory space rather than time, then the corresponding definition yields the notion of space complexity.

We may classify computational problems based on their complexities: e.g. **P** is the set of problems which have a polynomial time complexity. Based on time and space complexity, we may also define well-known complexity classes such as **EXP** and **PSPACE**, and by modifying the notion of Turing machines, we can get new complexity classes based on these resources, for instance **co-NP** and **NPSPACE**.

A different model for defining the difficulty of a problem is through *decision tree complexity*, also known as *query complexity*: in this model, we assume that reading an input bit requires a large amount of resources compared to making computations with information already available to us. Therefore in this model, the price of an execution of an algorithm is the amount of input-bits it queried while running.

We may think of decision tree complexity as the analogue of time complexity in this model. There are other notions similar to their counterparts in the original model: we can define complexities for randomized algorithms, as well as *certificate complexity*, which is the smallest amount of bits of an input x an oracle has to reveal in order to prove that the value of the function must be $f(x)$, for the worst case input.

We will later see that the decision tree complexity of a function can be bounded from below by its certificate complexity, but also from above by the square of its certificate complexity. However, the corresponding claim is not known to be true in the Turing machine model: if there was always an algorithm with time complexity not greater than the square of the complexity of a certificate, it would imply $\mathbf{P} = \mathbf{NP}$, which is a famous unsolved problem.

This observation shows that these two models are significantly different. In the query model, it is typical that for two notions of complexity, both can be approximated by above with a polynomial of the other, which means they are *polynomially equivalent.*

We may also define complexities of functions based on different properties: the degree of a Boolean function is the smallest possible degree of a polynomial representing it, while *sensitivity* examines the effect of small changes in the input on the output. Surprisingly, these notions are also polynomially equivalent to the ones described above.

A different type of computational problems is the set of search problems. In this model, the set of possible outputs (called *solutions*) is a set Q instead of $\{0, 1\}$, and for each input, $Q_x \subset Q$ is the set of *feasible solutions.* An algorithm solving search problem S, given input x, must output an element of Q_x.

Some complexity notions in the query model have natural variants for search problems, e.g. query complexity, while others need some adjustment, such is *critical block sensitivity*, replacing block sensitivity and sensitivity.

In my thesis, I will introduce the most common notions of complexity, and examine their interrelationships. Section 2 contains upper bounds between them, establishing their polynomial equivalence. The most notable of these inequalities is the result of Huang [1], who showed that the sensitivity of a function is polynomially equivalent to its degree, thus solving a 27-year-old conjecture.

When polynomial equivalence is already established, we may examine the largest possible gap between two notions of complexity. This is an actively researched area, I will present some recent results in section 3.

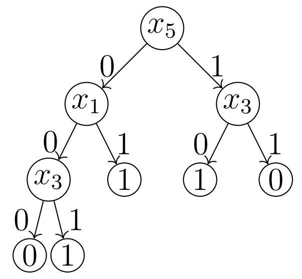
Section 4 contains the analysis of the Tarski search problem. I show current results about its query complexity and suggest a new method of tackling it, using critical block sensitivity.

1 Computational complexites

Let there be a Boolean function $f : \{0,1\}^n \to \{0,1\}$.

1.1 Decision tree complexity

A decision tree is a directed binary tree in which every non-leaf vertex has an out-edge labeled 0 and the other out-edge labeled 1. Each inner (non-leaf) vertex v also has a coordinate $i_v \in [n]$ assigned to it, while every leaf has an output-label 0 or 1. This defines an evaluation of the function for an arbitrary input x: the algorithm starts at the root-vertex v_0 and queries the coordinate i_{v_0} of x. Then it follows the out-edge with the label $x_{i_{v_0}}$ and arrives at the next vertex. It repeats this until it reaches a leaf, in which case it outputs the label of the leaf.



Example of a decision tree: Each inner vertex is labeled with its assigned coordinate of x, while each leaf is labeled with the relevant output.

We say that this tree calculates f if it gives the correct output for all possible inputs. The depth of the decision tree is the maximal amount of queries made for any input.

Definition 1.1. The decision tree complexity of f is the minimal depth over all decision trees that calculate f. Let us denote this by $\mathrm{D}^{\mathrm{dt}}(f)$.

1.2 Communication complexity

In this case the function is a bit different: let $\mathcal{X} = P(\{0,1\}^k)$ and $\mathcal{Y} = P(\{0,1\}^m)$ be two sets, and $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ a Boolean function.

There are two parties: Alice, who knows $x \in \mathcal{X}$ and Bob, who has $y \in \mathcal{Y}$. They don't have information about each-other's inputs, and their goal is to calculate $f(x,y)$ with as little communication as possible.

Alice \ Bob	00	01	10	11
00	0	1	1	0
01	1	0	0	1
10	1	0	0	1
11	0	1	1	0

Example of a communication matrix for the function Parity(x,y) = the number of $1'$s in x and y modulo 2.

In a general step of a protocol, a party can send a bit to the other, based on which one of them sends another bit or they end the protocol. The progress can be tracked using subrectangles of the table, which tracks the possible input pairs.

For example, if Alice sends her first bit as 0 and then Bob sends his second bit as 1, the relevant subrectangle will be the following:

Bob / Alice	00	01	10	11
00	0	1	1	0
01	1	0	0	1
10	1	0	0	1
11	0	1	1	0

The protocol ends if the current subrectangle only contains $0'$s or only contains $1's$, since in that case both parties can be sure of the output. For a given protocol, the maximal amount bits communicated over all inputs is called the depth of the protocol.

Definition 1.2. The communicational complexity of f, denoted by $\mathrm{D}^{\mathrm{cc}}(f)$, is the minimal depth over all protocols that solve f.

1.3 Polynomial degree

Every $f : \{0,1\}^n \to \{0,1\}$ function can be represented as a multivariate polynomial. We may assume that it is linear in every variable, and we take the lowest degree representation, which is unique.

The degree of this $\mathbb{R} \to \mathbb{R}$ polynomial can also be viewed as a complexity measure, denoted by $\deg f$.

A general form of this polynomial is $f(x) = \sum\limits_{I \subset [n]} \alpha_I \prod\limits_{i \in I} x_i$.

1.4 Certificate complexity

The certificate complexity of a function is the minimal amount of bits an oracle has to reveal to determine the output of a function.

Let $S \subset [n]$ be a set of bits. We call a function $C_S : S \to \{0,1\}^{|S|}$ an assignment of values. For an input x, if $f(x) = b$, we call an assignment C_S a b-certificate if for each $i \in S$, $C_S(i) = x_i$ and for each input $y \in \{0,1\}^n$ such that $y|_S = x|_S$, $f(y) = b$. In other words, revealing the bits of x in S already ensures that the output of the function is b. Let us call the size of the smallest b-certificate of x the certificate complexity of f at x, let us denote it by $C(x, b)$.

Definition 1.3. The certificate complexity of f is $C(f) = \max\limits_{x \in \{0,1\}^n} C(x, f(x))$.

The 0-certificate complexity of f is $C^{(0)}(f) = \max\limits_{\substack{x \in \{0,1\}^n \\ f(x)=0}} C(x, 0)$.

The 1-certificate complexity of f is $C^{(1)}(f) = \max\limits_{\substack{x \in \{0,1\}^n \\ f(x)=1}} C(x, 1)$.

Observation. $C(f) = \max(C^{(0)}(f), C^{(1)}(f))$.

1.5 Randomized decision tree complexity

A randomized decision tree is a probability distribution μ over all decision trees. On an input x, it chooses a decision tree with respect to μ and evaluates the input with it. The cost of the tree on an input x, $C(\mu, x)$ is the expected amount of queries made with distribution μ for x. The cost of the distribution μ is the cost for the worst case input, $C(\mu) = \max\limits_{x \in \{0,1\}^n} C(\mu, x)$. There are three different models about the error allowed in the computation:

- In the zero-error (Las Vegas) model, every decision tree in the support of μ must correctly evaluate f.
- In the one-sided error model, positive inputs must be accepted with probability 1, while negative input must be rejected with probability at least 1/2.
- In the two-sided error model, positive inputs must be accepted with at least $\frac{2}{3}$ probability, while negative inputs must be rejected with at least $\frac{2}{3}$ probability.

We denote by $\mathrm{R}_0^{\mathrm{dt}}(f)$, $\mathrm{R}_1^{\mathrm{dt}}(f)$ and $\mathrm{R}_2^{\mathrm{dt}}(f)$ the lowest possible cost of a distribution for the zero-error, one-sided error, and two-sided error models respectively.

1.6 Sensitivity and block sensitivity

Definition 1.4. The sensitivity of f at an input x, $s(f, x)$ is the number of i coordinates for which flipping the x_i bit (while the other coordinates stay the same) changes the output.

The sensitivity of f is the maximal sensitivity over all inputs, $s(f) = \max\limits_{x \in \{0,1\}^n} s(f, x)$.

Let us call a set of indices a block, and flipping a block means changing the input on each index of the block.

Definition 1.5. The block sensitivity of f at an input x, $bs(f,x)$ is the maximal amount of disjoint blocks, such that by flipping any of the blocks, the output changes.

The block sensitivity of f is the maximal block sensitivity over all inputs, $bs(f) = \max\limits_{x\in\{0,1\}^n} bs(f,x)$.

We may choose the blocks to be of size 1, which yields the following inequality.

Observation. $bs(f) \geq s(f)$.

It has been known for a long time that block sensitivity is polynomially related to the other complexity measures, but whether sensitivity also falls into this equivalence was unsolved until very recently. In 2019, Huang [1] proved that $bs(f) \leq s(f)^4$ for any Boolean function, thus establishing the polynomial equivalence. The best known separation is quadratic, i.e. there exists an f for which $bs(f) \geq \Omega(s(f)^2)$, which can be found in example 3.1.

Huang's solution can be found at the end of subsection 2.2.

2 Polynomial equivalence of complexities

In this section I will show some of the polynomial relations between notions of complexity, establishing their polynomial equivalence. As before, let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function.

2.1 Linear upper bounds

$$\begin{tikzcd}
s(f) \arrow[r] & bs(f) \arrow[r] \arrow[dl] & C(f) \arrow[dr] & \\
\mathrm{R}_2^{\mathrm{dt}}(f) \arrow[r] & \mathrm{R}_1^{\mathrm{dt}}(f) \arrow[r] & \mathrm{R}_0^{\mathrm{dt}}(f) \arrow[r] & \mathrm{D}^{\mathrm{dt}}(f) \\
 & & \deg(f) \arrow[ur] &
\end{tikzcd}$$

Linear relations between complexities

Proposition 2.1. $\mathrm{D}^{\mathrm{dt}}(f) \geq \mathrm{R}_0^{\mathrm{dt}}(f) \geq \mathrm{R}_1^{\mathrm{dt}}(f) \geq \frac{1}{2}\mathrm{R}_2^{\mathrm{dt}}(f)$.

Proof. The explanations of the three inequalities from left to right:

- A zero-error protocol can simulate a decision tree.
- Any zero-error distribution also satisfies the one-sided error model's requirements.
- If we repeat a one-sided error protocol twice, and only accept the input if both executions accepted it, we get a protocol which always accepts positive inputs and rejects negative inputs with at least 3/4 probability, this satisfies the requirements of the two-sided error model.

□

Notation. We denote the input given by flipping the bits of a block $B \subset [n]$ in an input x by x^B.

Theorem 2.2. $bs(f) \leq C(f)$.

Proof. Let x be an input and $B_1, B_2, \ldots, B_k$ be blocks such that flipping any of them changes the output. Then $C(f) \geq k$, since if the input is x, a certificate must show at least one bit from each block: If a block B_i is still completely unknown, the input could still be x or x^{B_i}, which give different outputs, so more bits need to be revealed. □

Theorem 2.3. $C(f) \leq \mathrm{D}^{\mathrm{dt}}(f)$.

Proof. Let T be an optimal decision tree. For an input x, the inputs queried during the execution of T for x give a certificate for output being $f(x)$ since the tree calculates f. Therefore $C(x, f(x))) \leq \mathrm{depth}(T) = \mathrm{D}^{\mathrm{dt}}(f)$ for each x. □

Theorem 2.4. $\deg(f) \leq \mathrm{D}^{\mathrm{dt}}(f)$.

Proof. Let us take an optimal decision tree T calculating f, and identify each non-leaf vertex with its corresponding variable. For each leaf l labeled "1", let $P_l = \{x_0^l, x_1^l, \ldots, x_{d(l)-1}^l\}$ be the path from the root to l, where $d(l)$ is the depth of leaf l. Let us define $\overline{x_i}^l$ the following way: if the out-edge going from x_i^l to x_{i+1}^l is labeled "1", then $\overline{x_i}^l = x_i^l$, otherwise $\overline{x_i}^l = 1 - x_i^l$. Define the monomial $M_l(x) = \prod_{i=i}^{d(l)-1} \overline{x_i}^l$. Notice that $M_l(x) = 1$ if and only if for input x, the execution of the decision tree follows along the path P_l. Since T calculates f, $f(x) = 1$ if and only if the execution follows along a path P_l for a leaf l labeled "1". Therefore the following multinomial of degree $\mathrm{depth}(T) = \mathrm{D}^{\mathrm{dt}}(f)$ calculates f: $\sum_{\substack{l \text{ is a leaf,} \\ label(l)=1}} M_l(x)$. □

Theorem 2.5. $bs(f) \leq 3\mathrm{R}_2^{\mathrm{dt}}(f)$.

Proof. Take input x and blocks $B_1, B_2, \ldots, B_k$ with $k = bs(f)$ such that $f(x) \neq f(x^{B_i})$ for each block, and fix an optimal two-sided error randomized decision tree distribution μ.

Let p_i be the probability that the algorithm queries at least one bit in B_i.

Lemma 2.6. *For each i, $p_i \geq \frac{1}{3}$.*

Proof. Suppose that $p_i < \frac{1}{3}$ for some i. Then with more than $\frac{2}{3}$ probability, μ makes the same execution, and gives the same output for x and x^{B_i}. But only one of these can be correct, therefore

$$P(\mu \text{ gives the correct output for } x) + P(\mu \text{ gives the correct output for } x^{B_i}) \leq$$
$$\leq p_i + p_i + (1 - p_i) < \frac{4}{3},$$

which implies that for either x or x^{B_i}, the probability of an error is more than $\frac{1}{3}$, which is a contradiction. □

Let us denote the expected amount of queries by $\mathbf{E}()$. Then we have the following inequality:

$$\mathrm{R}_2^{\mathrm{dt}}(f) \;=\; \mathbf{E}(\text{queries}) \;=\; \sum_{i=1}^{k} \mathbf{E}(\text{ queries in block } B_i) \;\geq\; \sum_{i=1}^{k} p_i \;\geq\; \frac{k}{3} \;=\; \frac{bs(f)}{3}.$$

□

2.2 Non-linear upper bounds

In this subsection I will show some of the upper bounds between complexities which prove their polynomial equivalence. Most of these are not known to be sharp, the maximal separation is an open problem.

$$\begin{tikzcd}
s(f) \arrow[r] & bs(f) \arrow[r] \arrow[dl] \arrow[l, bend right] \arrow[ddr, bend left] & C(f) \arrow[dr] \arrow[l, bend right] & \\
R_2(f) \arrow[r] & R_1(f) \arrow[r] & R_0(f) \arrow[r] & \mathrm{D}^{\mathrm{dt}}(f) \arrow[ul, bend right] \arrow[ull, bend right] \\
 & & \deg(f) \arrow[ur] \arrow[uull, bend left] &
\end{tikzcd}$$

Polynomial relations between complexities: A black arrow implies a linear, a green arrow implies a quadratic, a blue arrow implies a cubic, while a red arrow implies a quartic upper bound.

Theorem 2.7 (Nisan [5]). *$C(f) \leq s(f)bs(f)$.*

Proof. Let $x \in \{0,1\}^n$ be an arbitrary input, we want to show that $C(x, f(x)) \leq s(f)bs(f,x) \leq s(f)bs(f)$. Let $B_1, B_2, \ldots, B_k$ be minimal blocks exhibiting $k = bs(f,x)$, $f(x^{B_i}) \neq f(x)$.

Lemma 2.8. *For each i, $|B_i| \leq s(f)$.*

Proof. For each $j \in B_i$, $f(x^{B_i - \{j\}}) = f(x) \neq f(x^{B_i})$, otherwise exchanging B_i to $B_i' = B_i - \{j\}$ would give a set of disjoint sensitive blocks whose union is smaller, contradicting the minimality of $(B_1, \ldots, B_k)$. This implies $s(f) \geq s(f, x^{B_i}) \geq |B_i|$. □

Let the certificate consist of the bits in $B_1 \cup B_2 \cup \cdots \cup B_k$. This will have size $\sum_{i=1}^{k} |B_i| \leq s(f)bs(f)$ and proves the output $f(x)$: Suppose that $f(x') \neq f(x)$ but x and x' are the same on the bits of $B_1 \cup B_2 \cup \cdots \cup B_k$. Let B_{k+1} be the set of indices on which they differ. Then we have $k+1$ disjoint blocks such that flipping any of them changes the output for input x, which is a contradiction, since $k = bs(f, x)$.

□

Corollary 2.9. $C(f) \leq bs(f)^2$.

Remark 2.10. If $C^{(1)}(f) = m$ for a function f, then for each input x for which $f(x) = 1$, there exists a 1-certificate of size at most m.

Theorem 2.11 (Beals et al. [6])**.** $\mathrm{D}^{\mathrm{dt}}(f) \leq C^{(1)}(f)bs(f)$.

Proof. We describe a decision tree which uses at most $C^{(1)}bs(f)$ steps to calculate $f(x)$.

1. Repeat the following $k = bs(f)$ times:
   Choose an input y with $f(y) = 1$, for which there exists a 1-certificate C_i of size at most $C^{(1)}(f)$ whose bits agree with our input on the coordinates queried so far. Query all its yet unqueried coordinates. If all of them agree with the certificate, stop the algorithm with output 1. If there is no such certificate, stop the algorithm with output 0.

2. Let x be an arbitrary input consistent with the queries made so far. Output $f(x)$.

This always correctly evaluates f: If the algorithm stops in step 1 with output 1, then it has verified a 1-certificate, ensuring a correct output. If the value of the function on our input x is 1, there must exist such a certificate of size at most $C^{(1)}(f, x) \leq C^{(1)}(f)$.

Consequently, if it stops in step 1 with output 0, then there are no 1-certificates for the input of size at most $C^{(1)}(f)$, therefore the value of the function must be 0.

We need to show that if step 2 is reached, all remaining possible inputs give the same output.

Suppose that there is a y and a y' both consistent with the queries made in step 1, with $f(y) = 0$ and $f(y') = 1$. Since $f(y') = 1$, there must be a 1-certificate of size at most $C^{(1)}(f)$ which is consistent with y', denote this by C_{k+1}. Since $f(y) = 0$, for each $1 \leq i \leq k+1$ there must be coordinates for which C_i and y don't agree. Let us denote this set with B_i.

Lemma 2.12. *For $i \neq j$, B_i and B_j are disjoint.*

Proof. We may assume $i < j$. Suppose that there is a coordinate $h \in B_i \cap B_j$. By definition, $C_j(h) \neq y(h)$. At the jth iteration, certificate C_j was chosen with $y(h)$ already queried (in the ith iteration or earlier), therefore $C_j(h) = y(h)$ must hold, which is a contradiction. □

Since $f(y) = 0$ and y^{B_i} is consistent with C_i, y_i is sensitive for each block B_i, this implies $bs(f) \geq k + 1$, which is a contradiction.

□

Corollary 2.13. $\mathrm{D}^{\mathrm{dt}}(f) \leq C(f)^2$, $\mathrm{D}^{\mathrm{dt}}(f) \leq s(f)bs^2(f) \leq bs(f)^3$.

For a quadratic bound between $bs(f)$ and $\deg(f)$, we need the following theorem, proven by Ehlich and Zeller [9]:

Theorem 2.14 (Ehlich and Zeller)**.** *Let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial such that $b_1 \leqslant p(i) \leqslant b_2$ for every integer $0 \leqslant i \leqslant n$, and $|p'(x)| \geqslant c$ for some real $0 \leqslant x \leqslant n$.*
Then $\deg(p) \geqslant \sqrt{cn/\left(c + b_2 - b_1\right)}$.

It also uses the method of symmetrization, introduced by Minsky and Papert [8]. If p is a polynomial on n variables, and S_n is the set of all possible permutations on n elements, we define $p^{\mathrm{sym}}(x) = \frac{\sum\limits_{\pi \in S_n} p(\pi(x))}{n!}$, a polynomial symmetric in the coordinates of the input, with a degree not higher than the original.

Lemma 2.15 (Minsky, Papert)**.** *If $p : \mathbb{R}^n \to \mathbb{R}$ is a multilinear polynomial, then there exists a single-variate polynomial $q : \mathbb{R} \to \mathbb{R}$, of degree at most the degree of p, such that $p^{sym}(x) = q(|x|)$ for all $x \in \{0,1\}^n$.*

Proof. Since p^{sym} is a symmetric polynomial, it can be written as $p(x_1, \ldots, x_n) = c_0V_0 + c_1V_1 + \ldots c_nV_n$, where V_i is the ith elementary basic polynomial, $V_i(x) = \sum\limits_{j_1<j_2<\cdots<j_i} x_{j_1} \cdot x_{j_2} \cdot \cdots \cdot x_{j_i}$, and c_i is a constant. Therefore it is enough to show that $V_i(x)$ can be written as such a single-variate polynomial on binary inputs.

For this we only need to observe that for $x \in \{0,1\}^n$, $V_i(x) = \binom{|x|}{i}$. Since the degree of this expression is the same as the degree of $V_i(x)$, the degree of $q(|x|)$ will be at most the largest i for which $c_i \neq 0$, which is the degree of p. □

Theorem 2.16 (Nisan and Szegedy [2])**.** $bs(f) \leq 2\deg^2(f)$.

Proof. Let a be an input and $B_1, B_2, \ldots, B_k$ blocks for which f is sensitive in a, where $k = bs(f)$. We may assume without loss of generality that $f(a) = 0$, since otherwise we could look the function $1 - f$, which has the same degree and block sensitivity as f. Let $p(x_1, x_2, \ldots, x_n)$ be a polynomial of degree $\deg(f)$ representing f. Let us define a polynomial $q(y_1, y_2, \ldots, y_k)$ the following way: replace every x_i in p with a variable y_j or a constant:

- If $i \in B_j$ and $a_i = 0$, replace x_i with y_j.
- If $i \in B_j$ and $a_i = 1$, replace x_i with $(1 - y_j)$.
- If i is not in a block, replace it with a constant a_i.

Note that q is a multilinear polynomial of degree at most $\deg(f)$, for which

- $q(0, 0, \ldots, 0) = f(a) = 0$
- $q(e_i) = f(a^{B_i}) = 1$, where e_i is the ith unit vector.

Let r be the single-variate polynomial of degree at most $\deg(f)$ lemma 2.15 yields for q.

Since every value of q on binary inputs is either 0 or 1, $0 \leq r(t) \leq 1$ for each integer $0 \leq t \leq k$, and $r(0) = f(0) = 0$, $r(1) = (q(e_1) + q(e_2) + \cdots + q(e_k))/k = 1$ together imply that there exists an $x \in (0, 1)$ such that $r'(x) = 1$.

Applying theorem 2.14 implies that $\deg(f) \geq \deg(r) \geq \sqrt{k/2}$.

□

Sensitivity vs. Block Sensitivity

The question about their polynomial equivalence was first proposed by Nisan and Szegedy [2]. Gotsman and Linial [3] reduced it to a graph theory problem: Let Q be the graph of the n-dimensional hypercube, i.e. the vertices of the graph correspond to the vertices of the hypercube, and the edges correspond to the edges of the hypercube, so two vertices $x, y \in \{0, 1\}^n$ span an edge if and only if they differ in exactly one coordinate.

Conjecture 2.17 (Gotsman and Linial). *Let H be an induced graph of Q with at least $2^{n-1} + 1$ vertices. Then there exists a vertex of H with degree at least $\sqrt{n}$.*

This statement was shown to be sharp, i.e. there exists a subgraph with $2^{n-1}+1$ vertices with maximal degree $\sqrt{n}$. The best known lower bound was logarithmic until recently, which was improved by Huang.

The reduction was done using polynomial representations. We have seen previously that $\deg(f)$ is polynomially equivalent to $bs(f)$, therefore it is enough to show the equivalence of $\deg(f)$ and $s(f)$. Theorem 2.16 together with $s(f) \leq bs(f)$ imply $s(f) \leq 2\deg^2(f)$, therefore it was enough to show that $d(f)$ can be approximated by a polynomial of $s(f)$ by above. Gotsman and Linial proved that this is equivalent to Proposition 2.17 using the following theorem:

Notation. For a subgraph H of Q , $\Gamma(H) := \max(\Delta(H), \Delta(Q\backslash H))$, the maximum of the maximal degree of H and the maximal degree of the subgraph induced by $V(Q)\backslash V(H)$.

Theorem 2.18. *Let $h : \mathbb{N} \to \mathbb{R}$ be a monotonically increasing function. The following two statements are equivalent:*

(1) For any induced subgraph G of Q such that $|V(G)| \neq 2^{n-1}$, $\Gamma(G) \geq h(n)$

(2) For any Boolean function f, $\deg(f) < h^{-1}(s(f))$.

Substituting $h(n) = \sqrt{n}$ shows that proving Proposition 2.17 is enough to prove the sensitivity conjecture, since if $V(G) \neq 2^{n-1}$, then either G or $Q\backslash G$ has at least $2^{n-1}+1$ vertices, therefore Proposition 2.17 implies (1), proving $\deg(f) < s(f)^2$ for any Boolean function f.

Proof of Conjecture 2.17 (Huang)

Considering this problem (and its equivalent version) was unsolved for decades, his proof is surprisingly simple and elegant.

He used Cauchy's interlace theorem, as well as two other lemmas for symmetric matrices.

A principal $m \times m$ submatrix of an $n \times n$ matrix is submatrix for which the set of indices for the rows and columns chosen is the same.

Theorem 2.19 (Cauchy's Interlace Theorem for Symmetric Matrices)**.** *Let A be a symmetric $n \times n$ matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Let B be a symmetric $m \times m$ principal submatrix of A with eigenvalues $\mu_1, \mu_2, \ldots, \mu_m$ for some $m < n$. Then for each $1 \leq i \leq m$,*

$$\lambda_i \geq \mu_i \geq \lambda_{n-m+i}.$$

He iteratively defined a sequence of matrices, which can be "almost" considered the adjacency matrix of the given size cube graph Q: Let

$$A_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \text{ and } A_n = \begin{bmatrix} A_{n-1} & I \\ I & -A_{n-1} \end{bmatrix},$$

where I is the $2^{n-1} \times 2^{n-1}$ identity matrix.

Remark 2.20. If we take the absolute value of each element in A_n, we get the adjacency matrix of Q_n.

Proof. We can see this by induction, we get the graph of Q_n by placing two copies of Q_{n-1} under one another and connecting the corresponding vertex pairs, from a vertex in Q_{n-1} with index k the indices of the corresponding vertices in Q_n are of the form $k, 2^{n-1} + k$, this indexing gives the adjacency matrix A_n. □

Lemma 2.21. *The eigenvalues of A_n are $\sqrt{n}$ with multiplicity 2^{n-1}, and $-\sqrt{n}$ with multiplicity 2^{n-1}.*

Proof. It is easy to see by induction that $A_n^2 = n \cdot I_{2^n}$, whose eigenvalue is n with multiplicity n, therefore the eigenvalues of A_n can only be $\pm\sqrt{n}$. Since the sum of eigenvalues is the trace is matrix, which is 0 by induction, the multiplicities must both be 2^{n-1}. □

The next lemma connects the eigenvalues to the maximal degree of a graph.

Lemma 2.22. *Let H be an m-vertex undirected graph, and $A_n \in \{-1, 0, 1\}^{m \times m}$ a matrix such that its columns and rows are indexed with the vertices of H, and whenever two vertices $u, v \in V(H)$ are non-adjacent, $A_{u,v} = 0$. Then*

$$\Delta(H) \geq \lambda_1(A),$$

where λ_1 is the largest eigenvalue of A.

Proof. Let $v = (v_1, v_2, \dots, v_n)$ be the eigenvector corresponding to λ_1. Suppose that v_1 is the coordinate with the largest absolute value. Then $|\lambda_1 v_1| = \left|\sum\limits_{i=1}^{n} A_{1,i} v_i\right| \leq \sum\limits_{i=1}^{n} |A_{1,i}||v_1| \leq \Delta(H)|v_1|$, since the number of i indices where $|A_{1,i}|$ is 1 instead of 0 is at most $\Delta(H)$. Since $|v_1| \neq 0$, $|\lambda_1| \leq \Delta(H)$ follows. □

The proof of Proposition 2.17:

Remark 2.20 implies that A_n and Q_n satisfy the conditions of Lemma 2.22, and any $2^{n-1}+1$-vertex H induced subgraph and its associated subgraph A_H also satisfy these conditions, therefore $\Delta(H) \geq \mu_1(A_H)$. But Theorem 2.19 implies that $\mu_1(A_H) \geq \lambda_{2^n-2^{n-1}-1+1}(A_n) = \sqrt{n}$, since the eigenvalues of A_n were $\sqrt{n}$ with multiplicity 2^{n-1}, and $-\sqrt{n}$ with multiplicity 2^{n-1}, this concludes the proof.

3 Separations

As we have seen, most complexity measures are polynomially equivalent to each other, but we can examine the exact relationship between them, i.e. search for the functions exhibiting the largest separations between two given complexity measures.

3.1 Sensitivity vs. block sensitivity

As we have seen in the previous section, $bs(f) \leq s(f)^4$ is a polynomial upper bound, but the exact sharp upper bound is unknown. The best known separation is quadratic, which was first shown by Rubinstein [4]:

Theorem 3.1 (Rubinstein)**.** *There exists an f with $bs(f) \geq \Omega(s(f)^2)$.*

Proof. Let $n = k^2$ for some even integer k, and let us partition $[n]$ into $\sqrt{n}$ sets of size $\sqrt{n} = k$, with $S_i = \{x_{k\cdot i+1}, x_{k\cdot i+2}, \ldots, x_{k\cdot i+k-1}\}$. For an input $x = (x_1, x_2, \ldots, x_n)$, let $f(x) = 1$ if and only if there exists a set S_i such that exactly two adjacent coordinates in S_i are 1, the other $k-2$ are 0.

It is easy to see that $bs(f) \geq n/2$: let us partition $[n]$ into $n/2$ blocks of adjacent integers, with $B_i = \{x_{2i-1}, x_{2i}\}$. For the all-0 input, $bs(f,x) \geq n/2$ with these blocks since flipping any block gives two adjacent $1'$s in some set S_i (because k is even).

Now we need an upper bound on the sensitivity of f. Let x be an arbitrary input.

- **Case 1**: $f(x) = 1$. This means that there exists a set S_i with two adjacent $1's$. If these aren't changed, the value of the function remains one, therefore $s(f,x) \leq 2$.

- **Case 2**: $f(x) = 0$. Flipping a bit in a set S_i can only change the output if there were exactly one or three $1'$s in that set. If there was one, it had to be adjacent to the flipped bit. If there were three, then one of them had to be flipped to change the output, such that the remaining two $1's$ are adjacent (therefore the middle "1" bit couldn't be flipped). This means that there can be at most two bits in each set for which the function is sensitive at x. This implies $s(f,x) \leq 2k = 2\sqrt{n}$.

These bounds together imply $bs(f) \geq \frac{1}{8}s(f)^2$ for infinitely many n-s.

□

3.2 Unambiguous non-deterministic vs. deterministic complexities

Göös, Pitassi, and Watson [7] examined separations between unambiguous non-deterministic and deterministic models for both decision tree and communication complexity. This subsection contains their results for both cases.

They first found an f which has a large gap between its decision tree and unambiguous decision tree complexity.

An unambiguous decision tree is a non-deterministic decision tree which has a unique accepting computation for each correct input: formally, a non-deterministic decision tree is a collection of 1-certificates, in the unambiguous case, each input can only have 1 certificate verifying it. Non-deterministic decision tree complexity is denoted by $\mathrm{NP}^{\mathrm{dt}}(f)$, and its unambiguous variant by $\mathrm{UP}^{\mathrm{dt}}(f)$.

The following proof uses non-Boole functions to find separations. In general, if a function f is defined on strings of length n from an alphabet Σ, the decision tree complexities of $f : \Sigma^n \to \{0,1\}$ are defined the same way, querying a coordinate reveals $x_i \in \Sigma$.

Results using larger alphabets can be applied in our model for Boolean functions:

Let $h : \{0,1\}^{\lceil \log|\Sigma| \rceil} \to \Sigma$ be an arbitrary surjection. If a decision tree T_f for $f : \Sigma^n \to \{0,1\}$ solves f, we can construct a decision tree $T_{f \circ h^n}$ for $f \circ h^n : \{0,1\}^{n \cdot \lceil \log|\Sigma| \rceil} \to \{0,1\}$ which simulates T_f, such that if T_f queries a bit $x_i \in \Sigma$, $T_{f \circ h^n}$ queries all $\lceil \log|\Sigma| \rceil$ bits necessary to compute x_i, therefore $\mathrm{D}^{\mathrm{dt}}(f \circ h^n) \leq \mathrm{D}^{\mathrm{dt}}(f) \cdot \lceil \log|\Sigma| \rceil$.

In the other direction, it is easy to see that $\mathrm{D}^{\mathrm{dt}}(f \circ h^n) \geq \mathrm{D}^{\mathrm{dt}}(f)$, since a decision tree $T_{f \circ h^n}$ for $f \circ h^n$ can be simulated with no extra cost: If $T_{f \circ h^n}$ queries a bit b_j^i (where $i \in [n]$, $j \in [\lceil \log|\Sigma| \rceil]$), then T_f queries the corresponding coordinate x_i of its input. The value of bit b_j^i only holds information about x_i, therefore T_f has at least the same amount of information about the input as $T_{f \circ h^n}$ at any step. This argument holds true for the non-deterministic and unambiguous cases as well.

This implies that taking a larger alphabet only creates an error of size $\log|\Sigma| = O(\log(n))$ if $|\Sigma| = \mathit{poly}(n)$, which can often be neglected.

Theorem 3.2 (Göös et al.). *There is a function f for which $\mathrm{UP}^{\mathrm{dt}}(f) \leq 2k-1$, and $\mathrm{D}^{\mathrm{dt}}(f) \geq k^2$ for any $k \in \mathbb{N}$.*

Proof. Let us start with defining a function for which $\mathrm{NP}^{\mathrm{dt}}(f) \leq 2k-1$, and $\mathrm{D}^{\mathrm{dt}}(f) \geq k^2$, and then modifying it to allow unique certificates.

Let $f : \{0,1\}^{k \times k} \to \{0,1\}$ be a Boolean function, we interpret it as a $k \times k$ grid filled with 1's and 0's. On an arbitrary input, f gives output 1 if and only if there exists a unique column containing only 1's (all other columns contain at least one 0).

For this f, $\mathrm{D}^{\mathrm{dt}}(f) \geq k^2$, since the algorithm cannot always be sure of the answer after $k^2 - 1$ queries:

In each column, we answer queries with "1", unless it is the last query in that column, in which case we answer "0". This makes it so the algorithm cannot be sure whether there exists an all-1 column until it has queried all columns fully.

However, there exists a non-deterministic decision tree, proving the existence using $2k - 1$ queries: It can guess the unique all-1 column, query each cell in it, and query (guess) a 0-cell in each other column, this is enough to prove that the output is 1 on the given input.

Notice however that this collection of certificates is not unambiguous, because there may be multiple zeroes in the other columns, so there may be multiple ways to prove the positivity for one input. To solve this, they modified the function:

Instead of $\{0, 1\}$, let us take a bigger alphabet: $\Sigma = \{0, 1\} \times ([k] \times [k] \cup \{\perp\})$, and work with functions $f : \Sigma^{k \times k} \to \{0, 1\}$.

Intuitively this means that in the $k \times k$ $0-1$ table we also assign to each element $(i_0, j_0) \in [k] \times [k]$ a pointer to another element (i_1, j_1) in the table, or we assign to it the sign $\perp$, called the zero pointer.

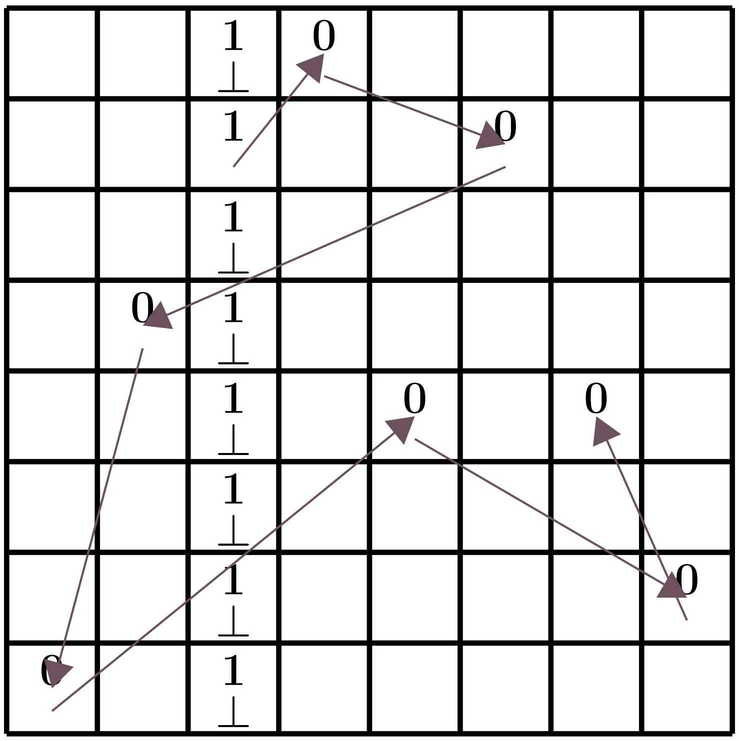


Figure 1: Example of an accepted input. The undefined values can be arbitrary.

For a given input x, let $f(x) = 1$ if and only if there exists a unique all-1 column (as before), while also requiring that in the unique column, each element has the zero pointer except for one, which has a pointer (i_1, j_1). If we iteratively follow these pointers, letting (i_{v+1}, j_{v+1}) be the element to which the pointer of (i_v, j_v) leads, the elements $(i_0, j_0), (i_1, j_1), \ldots, (i_{k-1}, j_{k-1})$ should be in pairwise different columns, and $(i_1, j_1), \ldots, (i_{k-1}, j_{k-1})$ should all contain 0's.

This is a stronger condition than the previous example, but now we can efficiently give unambiguous certificates: for each accepting input, querying the $2k-1$ elements mentioned in the definition ensures that the output would be 1. These are also unambiguous, since for each such input there is a unique 1-column, a unique non-$\perp$ element in that column, and a unique path following the pointers from that element. Therefore $\mathrm{UP}^{\mathrm{dt}}(f) \le 2k - 1$.

We also prove that $\mathrm{D}^{\mathrm{dt}}(f) \ge k^2$, by giving an answer strategy for any series of queries such that even after k^2-1 queries, the algorithm cannot decide the value of f.

In each column, let us call the last unqueried element "critical". For each non-critical query, the answer should be $(1, \perp)$. For the first critical query, let us answer with $(0, \perp)$, and recursively, for the mth critical query let us answer with $(0, (i_{m-1}, j_{m-1}))$, where (i_{m-1}, j_{m-1}) are the coordinates of the previous critical query.

With this strategy, regardless of the query algorithm, after $k^2 - 1$ moves the output can still be 1, if we set the last element to be $(1, (i_{k-1}, j_{k-1}))$, or 0, if we set the last element to be any other value, therefore $\mathrm{D}^{\mathrm{dt}}(f) \ge k^2$. $\square$

Let us call f the Göös-Pitassi function.

Notation. We use the $\tilde{\Omega}$ notation as an Ω which hides polylogarithmic factors, so $g(n) \ge \tilde{\Omega}(h(n))$ means $\exists C, k \in \mathbb{R}^+ : g(n) \ge C \cdot h(n) \cdot \frac{1}{\log(n)^k}$ for every $n \in \mathbb{N}$.

We get the following separation as a direct consequence:

Corollary 3.3. *There exists a Boolean function f such that* $\mathrm{D}^{\mathrm{dt}}(f) \ge \tilde{\Omega}\left(\mathrm{UP}^{\mathrm{dt}}(f)^2\right)$.

Using this result as a black box, the authors proved another separation theorem in communication complexity:

Definition 3.4. For a communication problem with a matrix M, $\chi_1(f)$ is the least number of pairwise disjoint rectangles needed to cover the 1's of M that no 0 is covered by any rectangle.

Definition 3.5. For a function $F : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$, $\mathrm{UP}^{\mathrm{cc}}(F) \overset{\mathrm{def}}{=} \lceil \log \chi_1(F) \rceil$ is the unambiguous nondeterministic communication complexity of F.

This is an analogous version of $\mathrm{UP}^{\mathrm{dt}}$ to communication protocols, since a 1-certificate in a communication problem is a collection of 1-rectangles which cover all the $1'$s in the matrix.

Theorem 3.6. *There is a function F with deterministic communication complexity $\tilde{\Omega}(\log^2 \chi_1(F)) = \tilde{\Omega}(\mathrm{UP}^{\mathrm{cc}}(F)^2)$.*

Remark 3.7. It was known for a long time that $\log \chi_1 \leq \mathrm{D}^{\mathrm{cc}}(F) \leq O(\log^2 \chi_1)$, but the maximal separation was unknown, this theorem proves that the maximum allowed by this bound is possible.

To prove Theorem 3.6, they used a simulation theorem. Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function, and $\mathcal{X}$,$\mathcal{Y}$ be the set of possible inputs of Alice and Bob respectively. The size of Alice's input is $\log(|\mathcal{X}|) = \Theta(\log n)$, while the size of Bob's input is $\Theta(\mathrm{poly}(n))$.

Theorem 3.8 (Simulation theorem)**.** *There is a gadget $g : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ such that for all $f : \{0,1\}^n \to \{0,1\}$, the following holds:*

$$\mathrm{D}^{\mathrm{cc}}(f \circ g^n) = \mathrm{D}^{\mathrm{dt}}(f) \cdot \Theta(\log(n)).$$

Let the gadget g be the following: let $m = \mathrm{poly}(n)$, $\mathcal{X} = [m]$, and $\mathcal{Y} = \{0,1\}^m$, $g(x,y) = y_x$ for $x \in \mathcal{X}$, $y \in \mathcal{Y}$. This way it is not easy for the two parties to gain meaningful information about an input bit of f without querying the relevant bits of x and y.

One side of this theorem is fairly straightforward, $\mathrm{D}^{\mathrm{cc}}(f \circ g^n) \leq \mathrm{D}^{\mathrm{dt}}(f) \cdot O(\log(n))$ is true because a communication protocol can follow along a decision tree: if the decision tree queries a bit, Alice can communicate the relevant parts of her input using $O(\log(m))$ bits, and this way they find out the queried bit, so they can simulate a query with a $O(\log(m))$ cost, therefore $\mathrm{D}^{\mathrm{cc}}(f \circ g^n) \leq \mathrm{D}^{\mathrm{dt}}(f) \cdot O(\log(m)) = \mathrm{D}^{\mathrm{dt}}(f) \cdot O(\log(n))$ holds.

For the other direction, we need to prove that there is no communication with a more efficient strategy than the one described above. This is significantly harder, we can prove it in a similar way: we take a communication protocol, and give a decision tree which makes queries using the protocol. In the following paragraphs, I provide a sketch of their proof.

The simulation starts at the beginning of the protocol, in each iteration, it either makes the protocol send a bit, or queries a bit. If we are currently at a node v in the protocol, let us denote by R_v the subrectangle corresponding to v, the set of possible input pairs at v. We also maintain a "cleaned up" subrectangle $A \times B \subset R_v$ such that all input pairs in $A \times B$ are still possible given the queries of our decision tree so far.

If there is a coordinate of our input which depends too much on other unqueried bits then we query that coordinate, if there isn't, we move in the communication protocol, always choosing the node which leaves more options, i.e. for which the updated cleaned up subrectangle will be larger. The algorithm maintains an invariant which ensures that there are at most $\frac{1}{\Theta(\log(m))}$ times as many queries as communication steps, which is sufficient as there can be at most $\mathrm{D}^{\mathrm{cc}}(f)$-many communication steps in any execution of the protocol.

3.3 Deterministic decision tree complexity vs. randomized query complexity

Ambainis et al. [10] used the Göös-Pitassi function to find other separations:

Theorem 3.9 (Ambainis et al.). *There exists an f such that* $\mathrm{D}^{\mathrm{dt}}(f) \geq \left(\tilde{\Omega}\left(R_0(f)\right)\right)^2$.

They modified the function to exhibit this separation. Let M be a grid of $2n$ rows and n columns, filled with binary entries. Each cell also has 3 pointers.

- Left and right pointers point to another cell in the grid, or take the value "$\perp$".
- Back pointers point to a column or take the value "$\perp$".

So each cell v has 4 parameters, its value, right pointer, left pointer, and back pointer, denoted by $val(v)$, $rpoint(v)$, $lpoint(v)$ and $bpoint(v)$ respectively.

Let us fix a balanced binary tree T on n leaves and $n-1$ inner vertices: if $2^p \leq n < 2^{p+1}$, every vertex will be at a distance p or $p+1$ from the root, let us add the vertices on level $p+1$ from left to right (see figure 2). Label the leaves from left to right with the elements of $[n]$.

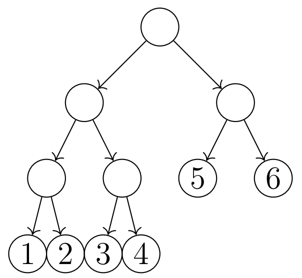


Figure 2: Example of a balanced binary tree with $p = 2$ levels and 6 leaves

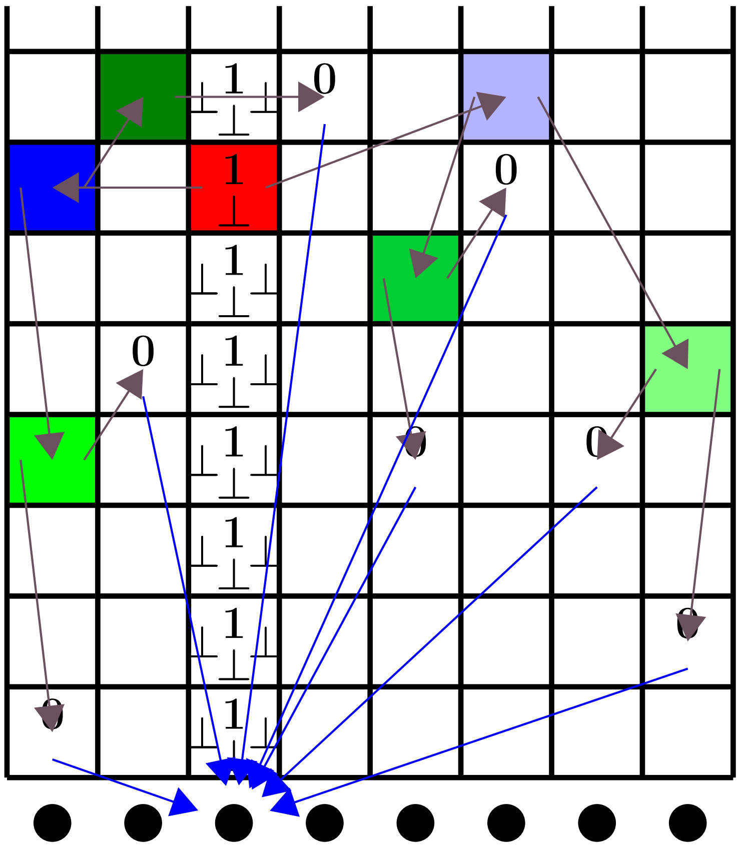
Let us define $f_{2n,n} : M \times (M \cup \{\perp\}) \times (M \cup \{\perp\}) \times ([n] \cup \{\perp\}) \to \{0,1\}$ the following way:

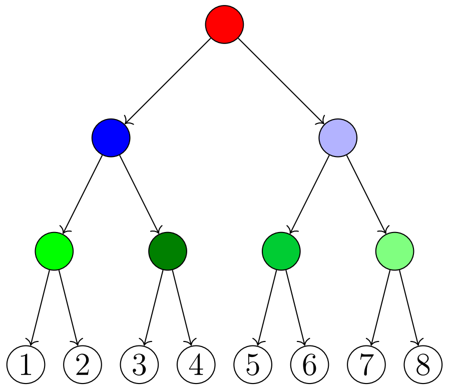
For the output to be 1, there needs to be a unique all-1 column, let us call this the *marked column* with index $b \in [n]$. In the marked column, each input must be $(1, \perp, \perp, \perp)$ except for one, which we call the *special element*.

We also need to assign a cell to each vertex of T such that it satisfies the following criteria:

The root must be the special element a, and for every inner vertex y of T, if the cell v is assigned to it, the left child of y must be $lpoint(v)$, and the right child must be $rpoint(v)$.

For each unmarked column $j \in [n] \backslash \{b\}$, there is a unique left-right sequence $T(j)$ going from the root of T to the leaf labeled j.





An accepted input of the function. The undefined inputs can be arbitrary. The 3rd leaf of T is inconsequential since there are no conditions between the marked column and the binary tree.
The top 8 rows have been omitted, there is no condition on their inputs except for the cells 3rd column, which have the same input as the other non-red cells in that column.

We define a sequence of cells with the help of this sequence: let us start in the special element a and in each step, we take either the left or right pointer of the current cell depending on the corresponding element of the sequence $T(j)$, and we move on to the cell it points to. When we reach the end of the sequence, we arrive at a cell l_j. We require that this cell exists (that is, no pointer up to this point was $\perp$), that l_j is in column j, $bpoint(l_j) = b$, and $val(l_j) = 0$.

The value of the function is 1 if and only if it satisfies all these conditions.

The following theorem states that any decision tree needs to query at least half the cells in M in the worst case.

Theorem 3.10. *If n is large enough, $\mathrm{D}^{\mathrm{dt}}(f) \geq n^2$.*

Proof. We describe an adversary strategy for which the algorithm cannot be sure of the output after n^2 steps.

Assume it queries the cell (i, j), and it is the kth queried cell in the column. If $k \leq n$, answer with $(1, \perp, \perp, \perp)$, otherwise answer with $(0, \perp, \perp, k - n)$

Lemma 3.11. *If there is a column $b \in [n]$ with at least n unqueried cells and there are at least $4n$ unqueried cells in M, then the value of the function is still undetermined.*

Proof. If each unqueried cell v had $val(v) = 0$, then the output would be 0, since the adversary strategy didn't make an all-1 column at any point.

The output can also be 1:

The marked column will be b. Let all unqueried cells there have value $(1, \perp, \perp, \perp)$ except for one arbitrary cell, which will be the special element a, this will be assigned to the root of T.

At a column $j \in [n] \backslash \{b\}$, if there is an unqueried cell, let its value be $(0, \perp, \perp, b)$, this cell will be l_j. If all cells are queried, then there is a cell with value $(0, \perp, \perp, b)$, let this be l_j.

After these assignments there are still at least $4n - n - n = 2n$ unqueried cells, we use $n - 2$ of these as the inner vertices of T. We can use any assignment, but we need to set the left and right pointers such that l_j is assigned to the jth leaf of T. This makes sure that the value of the function is 1. $\square$

If there are at least n^2 unqueried cells then there must be a column with at least n unqueried cells and if $n \geq 4$, there are at least $4n \leq n^2$ unqueried cells, so the conditions of the lemma are satisfied. $\square$

Theorem 3.12. *The function f has Las Vegas randomized complexity $\tilde{O}(n)$.*

Proof. Most of the algorithm is deterministic, it uses random queries in only one part of it.

It uses two subroutines:

- **VerifyColumn**(j) checks whether column j is marked:

1. It first queries each cell in column j, checking that each input is $(1, \perp, \perp, \perp)$ except for one, which is the special element a.
2. Then following the left and right pointers from a, it checks the second condition for the output to be one, namely that for each column i the left-right sequence defined by $T(i)$ leads to a cell l_i in column i, whose input is $(0, \perp, \perp, j)$.

- **Testcolumn**(c, k) always returns 1 if there are no zeroes in column c. If there are at least $k/2$ zeroes, it returns 0 with at least $1 - \frac{1}{n^4}$ probability. It returns anything in intermediate cases.

  1. Query $\lceil 8\frac{n}{k}\log(n)\rceil$ random elements from column c. If all of them have value 1, output 1, otherwise output 0.

  This yields the desired results: the probability of a false positive is at most $\left(\frac{n-k/2}{n}\right)^{4\frac{n}{k}2\log(n)} = \left(1 - \frac{k}{2n}\right)^{\frac{2n}{k}\cdot 4\log(n)} \leq \sup_{x\in(0,1)} (1-x)^{\frac{1}{x}\cdot 4\log(n)} = e^{-4\log(n)} = \frac{1}{n^4}$.

The main procedure of the algorithm:

1. Begin with an arbitrary column $j \in [n]$ and $k = n$.

   Then repeat these steps until the algorithm stops:

2. Query all the elements in column j.

   (a) If all of them have value 1, **VerifyColumn**(j).

   (b) If there are more than k zeroes, query all the elements in M and output the value of the function.

   (c) In intermediate cases, let C be the set of back pointers from the zeroes of the column. For each $c \in C$, **TestColumn**(c, k). If all of them return 0, then reject the input. Otherwise, let $j' \in C$ be an arbitrary column for which it returned 1, and move on to column j' instead of j.

3. If $k = 0$, reject the input. Otherwise, $k \leftarrow \lfloor \frac{k}{2} \rfloor$.

Analysis of the algorithm

If we find an all-1 column, then it must be the marked column, which can be checked in $O(n)$ time in step 2.(a) .

If there is a small number of zeroes in column j, then if the input is to be accepted, the marked column can only be where one of the back pointers of the $0'$s in column j points. The column we move on to might be a false positive, but the probability of this is set such that this "error" only adds a logarithmic factor to the expected amount of queries.

Case 2.(b) adds a quadratic number of steps, but also has low probability of happening.

Let us check the correctness of the algorithm:

If the input should be accepted, then the algorithm never rejects it:

- In step 2.(c), the marked column will always be in C,
- In step 3., if $k = 0$ then either case 2.(a) or (b) holds, the former will give the correct output since there can be only one all-1 column, while the latter always gives the correct output. The algorithm never reaches step 3. if $k = 0$ and the input is correct.

The algorithm only accepts an input if it satisfies all the necessary criteria in **VerifyColumn**, therefore it always rejects any false input.

Now estimate the expected amount of queries for the worst case input:

Let us fix an arbitrary input x. Since the value of k is halved in each iteration, there are at most $O(\log n)$ loops.

Case 2.(a) can only hold once, in which case the algorithm queries $O(n)$ cells and gives the correct output.

Case 2.(b) cannot hold in the first iteration, and if it later holds, it means we got a false positive in the previous step. A given column has at most $\frac{1}{n^4}$ probability to be a false positive, therefore this case can only add $O\left(n^3 \cdot \log(n) \cdot \frac{1}{n^4}\right) = o(1)$ to the expected number of queries.

In case 2.(c), we need to check at most k columns using $O(k \cdot 8\frac{n}{k}\log(n))$ queries. This case can be invoked at most $O(\log n)$ times, therefore the expected number of queries added by this case can be at most $O(n \log^2 n) = \tilde{O}(n)$.

Each possible case adds at most $\tilde{O}(n)$ to the expected number of queries, the expected number of total queries is also $\tilde{O}(n)$.

□

This concludes the proof of the separation theorem 3.9.

4 Tarski search problem

In this section, I will examine the properties of the Tarski search problem.

In general, a search problem $S \subset \{0,1\}^n \times Q$ is a problem with a set of inputs $\{0,1\}^n$ and a set of possible solutions Q. The set of feasible input-solution pairs is denoted by S. We assume that every input has at least one feasible solution.

Given an input x, our goal is to find a solution $q \in Q$ for which $(x, q) \in S$. We say that function $s : \{0,1\}^n \to Q$ solves S if $(x, s(x)) \in S$ for each input x.

We may define complexities for search problems the same way as we did for Boolean functions, for example the deterministic decision tree complexity of S is the smallest possible cost of a decision tree which solves S.

The definitions of sensitivity and block sensitivity use the evaluating Boolean function f, which is not defined in this case. Therefore these notions are not well-defined for search problems, so we need to modify the definition to include the choice of s:

Definition 4.1. We call an input x *critical* if there is only one feasible solution for it.

Definition 4.2. The critical block sensitivity of S at s is the block sensitivity restricted to critical inputs, $cbs(S, s) = \max_{x \text{ critical}} bs(s, x)$.

The critical block sensitivity of S is the minimal critical block sensitivity over all $s \subset S$ functions which solve S, $cbs(S) = \min_{\substack{s:\{0,1\}^n \to Q \\ s \subset S}} cbs(S, s)$.

On the d-dimensional n-sized grid, $[n]^d$, there is a natural partial ordering: $(x_1, x_2, \dots, x_n) \leq (y_1, y_2, \dots, y_n)$ if and only if $x_i \leq y_i$ for all $1 \leq i \leq n$. Given a monotone function f with respect to this partial ordering on this grid, Tarski's fixed point theorem implies the existence of a fixpoint. In the d-dimensional Tarski search problem, denoted by $T_d = T_d(n)$, the inputs are the monotone functions while the possible solutions are vertices of the grid. Our goal is to find a fixpoint of the input function, therefore the set of feasible solutions is $S = \{(f, x) : f(x) = x\}$.

4.1 Decision tree complexity of the Tarski search problem

We may examine the decision tree complexity of this problem. Let us start with the $d = 1$ case.

Lemma 4.3. *If $f(i) \geq i$ for some $i \in [n]$, there exists a fixpoint in the subinterval $[i, n]$. If $f(i) \leq i$ for some i, there exists a fixpoint in the subinterval $[1, i]$.*

Proof. Monotonicity implies that $f(j) \geq i$ for each $j \geq i$, therefore the image of $f|_{[i,n]}$ is in $[i, n]$, this constrained function is also monotone, therefore the fixed point theorem for $f|_{[i,n]}$ implies the existence of a fixpoint in $[i, n]$.

The second statement can be proven similarly. $\square$

These two observations imply that we can find a fixpoint using binary search, in each step we move left or right depending on whether the value of the queried variable is less or more than its index. Therefore $\mathrm{D}^{\mathrm{dt}}(T_1) \leq \lceil \log(n) \rceil$.

For the lower bound, fix a set of possible inputs: let $i \in [n]$ be an arbitrary coordinate, and the only fixpoint of f. At all the other indices, let $f(j) = j + 1$ if $j < i$, and $f(j) = j - 1$ if $j > i$. If we only consider these n strings as possible inputs, finding a fixpoint is exactly as hard as a binary search problem, which is known to be of $\lceil \log(n) \rceil$ complexity.

Corollary 4.4. $\mathrm{D}^{\mathrm{dt}}(T_1) = \lceil \log(n) \rceil$.

The previous lemma has a generalization in d dimensions:

Lemma 4.5. *If $f(x) \geq x$ for some $x \in [n]^d$, then there is a fixpoint in the sublattice $\{y \in [n]^d : y \geq x\}$. Similarly, if $f(x) \leq x$ for some $x \in [n]^d$, then there is a fixpoint in the sublattice $\{y \in [n]^d : y \leq x\}$.*

Proof. If $f(x) \geq x$, then by monotonicity, $f(y) \geq x$ for all $y \geq x$, therefore f maps the sublattice $L' = \{y \in [n]^d : y \geq x\}$ into itself, thus we can apply Tarski's fixed point theorem for $f|_{L'}$, so there exists a fixpoint of f in L'.

The other case can be proven the same way. $\square$

In the d-dimensional case, $\log^d(n)$ is a not too difficult upper bound:

Proposition 4.6 (Dang et al. [14])**.** $\mathrm{D}^{\mathrm{dt}}(T_d) \leq \log^d(n)$.

Proof. The $d = 1$ is already proven, we will prove the rest using induction.

Let us fix the last coordinate to be $\lfloor n/2 \rfloor$, which gives us a sublattice H in a hyperplane. Let us define $f' : H \to H$ by modifying f on H such that $f'(x) = pr_H(f(x))$ for each $x \in H$, f' is the "projection of $f|_H$ to H". This gives us a monotone function on an $n-1$-dimensional grid, therefore we can find a fixpoint of f' in $O(\log^{d-1}(n))$ time by induction.

This means that we found an x with dth coordinate $\lfloor n/2 \rfloor$ such that $f(x)$ can only differ from x in the dth coordinate, therefore either $f(x) \leq x$ or $f(x) \geq x$ holds. Lemma 4.5 states that it is sufficient to consider points smaller or larger than x (depending on whether $f(x)$ or x is greater) for the rest of the algorithm, which halves the height of the grid we need to consider when looking for a fixpoint.

Applying this $\log(n)$ times will yield a fixpoint, thus the total runtime of the algorithm is $O(\log^{d-1}(n) \cdot \log(n))$.

$\square$

For a while, this upper bound was conjectured to be sharp, but Fearnley et al. [13] found an algorithm which finds the fixpoint in 3 dimensions in $\log^2(n)$ time (by optimizing the 2-dimensional recursive step to only require $\log(n)$ steps). Using this result, Chen and Li [11] gave an algorithm in d dimensions with a runtime of $\log^{\lceil \frac{d+1}{2} \rceil}(n)$.

In 2 dimensions, Etessami et al. [12] showed a lower bound of $\Omega(\log^2(n))$ by giving an oracle algorithm for answering queries.

This is the best known lower bound for all $d \geq 2$ dimensions, it gives a sharp bound for $d = 2, 3$, but the problem is still open for $d \geq 4$.

4.2 Critical block sensitivity of the Tarski search problem

We can define communication protocols for search problems the same way as for Boolean functions, only in this case the final answer of the parties must be a feasible solution instead of the value of the booelan function.

Similarly to randomized decision trees, a randomized communication protocol is a distribution over all possible deterministic communication protocols. We say that a randomized communication protocol *consistently solves* S, if for each input, there is a unique feasible solution $q \in Q$ which they answer with probability at least 3/4. Let us denote the minimal complexity of such a protocol with $R^{cc}_{cons}(S)$.

Recall the gadget g used in the proof of Theorem 3.8. Let us use the same gadget, modified for search problems: let $S \subseteq \{0,1\}^n \times Q$ be a search problem, and $l \in \mathbb{Z}^+$.

Let $\mathcal{X} = [l]^n$ and $\mathcal{Y} = \left(\{0,1\}^l\right)^n$, and $g : [l] \times \{0,1\}^l \to \{0,1\}$, $g(x,y) = y_x$.

This defines a communication search problem $S_g \subset (\mathcal{X} \times \mathcal{Y}) \times Q$, where
$\big((x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots, y_n), q\big) \in S_g \Leftrightarrow \big((g(x_1,y_1), g(x_2,y_2), \ldots, g(x_n,y_n)), q\big) \in S$.

Huynh and Nordstrom [15] showed a connection between critical block sensitivity and randomized consistent communication complexity:

Theorem 4.7 (Huynh, Nordström). *Let $S \subseteq \{0,1\}^m \times Q$ be a search problem. Then it holds for any $\ell \geq 3$ that any consistent randomized (and hence also any deterministic) two-party protocol solving S_g requires $\Omega\left(cbs(S)\right)$ bits of communication, where Alice receives the x-variables and Bob receives the y-variables in S_g.*

Theorem 4.8 (Theorem 4.7 rephrased). *For any search problem $S \subseteq \{0,1\}^m \times Q$ and $l \geq 3$ there exists a gadget $g : [l] \times \{0,1\}^l \to \{0,1\}$, for which $R^{cc}_{cons}(S_g) = \Omega(cbs(S))$, and therefore $\mathrm{D^{cc}}(S_g) = \Omega(cbs(S))$.*

Combined with the search problem variant of theorem 3.8 we get the following:

Corollary 4.9. *For any search problem $S \subseteq \{0,1\}^m \times Q$ and $l \geq 3$ there exists a gadget $g : [l] \times \{0,1\}^l \to \{0,1\}$, for which $\mathrm{D}^{\mathrm{dt}}(S) = \mathrm{D}^{\mathrm{cc}}(S_g)/\Theta(\log(n)) = \Omega(cbs(S)/\log(n))$.*

This suggests that we should look at the critical block sensitivity of the Tarski search problem, as it might give an improved lower bound for its deterministic decision tree complexity. If we could prove a critical block sensitivity of an order of magnitude $\omega(\log^3(n))$ for any dimension d, it would already be sufficient for an improvement.

First, let us look at the critical block sensitivity in 1-dimension.

The following proof constructs bicritical inputs to prove a lower bound for critical block sensitivity. We call an input x *bicritical* if it has exactly two feasible solutions.

Theorem 4.10. $cbs(T_1) = \Omega(\log(n))$

Proof. Let $i \in [n]$ be an arbitrary index. Define critical inputs t_i ($i \in [n]$) as in the proof of Corollary 4.4: let $t(i) = i$, $t(h) = h + 1$ if $h < i$, and $t(h) = h - 1$ if $h > i$. We will define $\Omega(\log n)$ pairwise disjoint blocks of indices B_k such that changing the input only in B_k creates a bicritical input, which will be useful in proving a lower bound for critical block sensitivity.

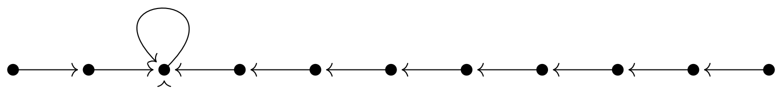


Figure 3: Input t_3 for $n = 11$.

Let $B_k^{\text{right}} = \{i + 2^k, i + 2^k + 1, \ldots, i + 2^{k+1} - 1\}$ for $k = 0, 1, \ldots, \lfloor \log(n - i) \rfloor$, and for $j \in B_k$, change the input such that instead of $t(j) = j - 1$, now $t(j) = j + 1$ for $j \neq i + 2^{k+1} - 1$, and $t(j) = i + 2^{k+1} - 1$ for $j = i + 2^{k+1} - 1$, creating a fixpoint (see figure 4). Let us denote this modified input by $t_{i,j}$. Define $B_k^{\text{left}} = \{i - 2^k, i - (2^k + 1), \ldots, i - (2^{k+1} - 1)\}$ for $k = 0, 1, \ldots, \lfloor \log(i) \rfloor$, and define $t_{j,i}$ similarly, by changing the direction of the arrows and creating a fixpoint at the end.

Note that $t_{i,j}$ is a monotone function which we can get using this method from both t_i and t_j, so for the sake of simplicity we may assume $t_{i,j} = t_{j,i}$ for all i, j. Fix an arbitrary function f which solves the Tarski-1 problem, that is, it assigns a fixpoint to every monotone function.

Specifically, for each $t_{i,j}$, it either outputs i or j. We can look at this as a directed graph G_f: let the vertices be the indices $1, 2, \ldots, n$, and two vertices indexed a, b are

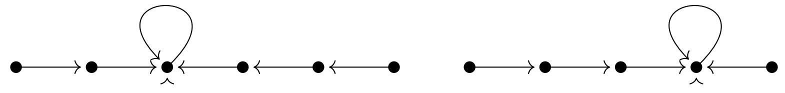


Figure 4: Input $t_{3,10}$ for $n = 11$.

connected by an edge if and only if $|a - b|$ is a power of two (which means there is a bicritical input $t_{a,b}$ for this indexpair). Direct this edge towards a if $f(t_{a,b}) = a$, direct it towards b if $f(t_{a,b}) = b$. There must be a vertex whose outdegree is at least the half of its degree, and all the vertices have degree $\Theta(\log(n))$, therefore there is an index l such that $f(t_{i,l}) = i$ for $\Omega(\log(n))$-many i's, therefore $cbs(T_1, f) = \Omega(\log(n))$ for all f solutions, which proves $cbs(T_1) = \Omega(\log(n))$.

□

We were unable to prove a higher lower bound in higher dimensions, our idea was to use the idea of Etessami et al. to construct a set of inputs which exhibit a high critical block sensitivity.

Conjecture 4.11. $cbs(T_2) \geq \Omega(\log^2(n))$.

As mentioned above, finding dimension d with $cbs(T_d) = \omega(\log^3(n))$ would suffice, but finding such a construction may not be easier than proving query complexity directly. For this approach, one may try to iterate the idea of Etessami et al., to hide a path on which the fixpoint is hidden, to reach higher query complexity.

References

[1] Huang, H., Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, **190** (3) 949-955 (2019).

[2] Nisan, N., Szegedy, M., On the degree of Boolean functions as real polynomials. *Comput. Complexity*, **4** 301–313 (1994).

[3] Gotsman, C., Linial, N., The equivalence of two problems on the cube. *Journal of Combinatorial Theory A*, **61** (1) 142-146 (1992).

[4] Rubinstein, D., Sensitivity vs. block sensitivity of Boolean functions. *Combinatorica 15*, 297–299 (1995).

[5] Nisan, N., CREW PRAMs and decision trees. *SIAM J. Comput.* **20** (6) 999–1007 (1991).

[6] Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R., Quantum lower bounds bypolynomials. *Proc. 39th FOCS*, 352–361 (1998).

[7] Göös, M., Pitassi, T., Watson, T., Deterministic Communication vs. Partition Number. *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, 1077-1088 (2015).

[8] Minsky, M., Papert, S., *Perceptrons*, 2nd expanded ed., MIT Press, Cambridge, MA, 1968, 1988.

[9] Ehlich, H., Zeller, K. Schwankung von Polynomen zwischen Gitterpunkten. *Math Z*, **86** 41–44 (1964).

[10] Ambainis, A., Balodis, K., Belovs, A., Lee, T., Santha, M., Smotrovs, J., Separations in query complexity based on pointer functions. *Journal of the ACM (JACM)*, **64.5** 1-24 (2017).

[11] Chen, X., Li, Y., Improved Upper Bounds for Finding Tarski Fixed Points. *arXiv preprint* arXiv:2202.05913 (2022).

[12] Etessami,K.,Papadimitriou, C., Rubinstein, A., Yannakakis, M., Tarski's theorem, supermodular games, and the complexity of equilibria. *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)* Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020).

[13] Fearnley, J., Pálvölgyi, D., Savani, R., A faster algorithm for finding Tarski fixed points. *ACM Transactions on Algorithms (TALG)*, **18.3** 1-23 (2022).

[14] Dang, C., Qi, Q., Ye, Y., Computational models and complexities of Tarski's fixed points. *Technical Report, Stanford University* (2012).

[15] Huynh, T., Nordstrom, J., On the virtue of succinct proofs: amplifying communication complexity hardness to time-space trade-offs in proof complexity. *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC '12)*, 233–248 (2012).