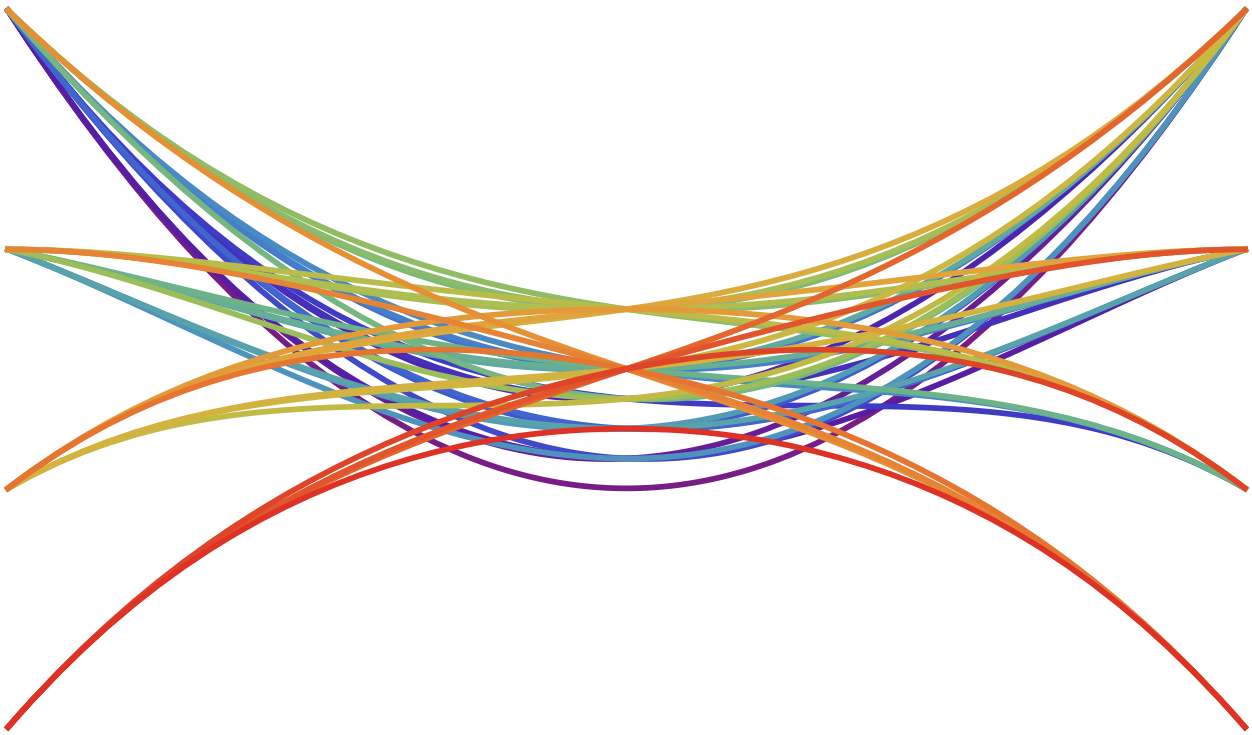


Level-p-complexity for Boolean Functions

Master's thesis in Engineering Mathematics and Computational Science

JULIA JANSSON

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Level-p-complexity for Boolean Functions

JULIA JANSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Analysis and Probability Theory
Chalmers University of Technology
Gothenburg, Sweden 2022

Level-p-complexity for Boolean Functions

JULIA JANSSON

Supervisor and examiner: Jeffrey Steif, Mathematical Sciences

Master's Thesis 2022
Department of Mathematical Sciences
Division of Analysis of Probability Theory
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Expected costs as a function of the probability p for the 39 different algorithms determining the output of the Boolean function f_{AC} described in Chapter 6.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Level-p-complexity for Boolean Functions

JULIA JANSSON
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

This thesis concerns characteristics of complexity, specifically level-p-complexity, for various Boolean functions. Boolean functions are functions f from n bits to one bit, and they can describe circuits built from logic gates, voting systems as well as graph properties. An example of a Boolean function is majority, which returns the value that has majority among the n input bits. Complexity for a Boolean function f can be seen intuitively as how much information is needed about the input bits until the result of f is certain. Some well-known notions of complexity for Boolean functions are deterministic and randomized complexity but in this thesis we focus on level-p-complexity. Level-p-complexity is defined as the minimum expected cost over algorithms determining the output, where the input bits are independent and identically distributed with Bernoulli(p) distribution. The level-p-complexity for a Boolean function f, is a function of p, so some interesting properties are explored, such as if it is differentiable, what the maximum is, and if it has more than one maximum.

First, we calculate the level-p-complexity for the Boolean functions "all" and "tribes". Next, we compute the level-p-complexity of majority specifically for three and five bits and we move on to iterated three bit majority on two levels. Then we apply Boolean functions to graphs, and we calculate the level-p-complexity for connectivity of graphs with three or four nodes. All of these examples have in common that their level-p-complexity is continuous and differentiable and has a unique maximum, even though a closed form for the maxima of the level-p-complexity of the tribes function was not found. Finally, we construct a Boolean function whose level-p-complexity has two maxima. In this case the optimal algorithm depends on p and the level-p-complexity is piecewise polynomial and thus continuous but not differentiable in the intersection points.

Future work could include calculating the level-p-complexity for other Boolean functions or constructing new Boolean functions, and explore properties of their level-p-complexity.

Keywords: Boolean functions, level-p-complexity, randomized complexity, deterministic complexity, evasiveness, influence, graph connectivity, iterated majority, Mathematica, Haskell

Acknowledgements

I am extremely grateful to my supervisor Jeffrey Steif for the continuous support and mathematical advice. Thank you for always being just an email or a Zoom call away, and for showing me how to think and write like a mathematician. I also thank everyone who has taken the time to read through my thesis for helpful feedback. Lastly, I am grateful to everyone on the Effective Altruism coworking discord server and gather town for keeping me company in remote work.

Julia Jansson, Gothenburg, May 2022

Contents

Nomenclature

f	Boolean function of type $\{0,1\}^n \to \{0,1\}$
x	Vector of input bits $x_1, x_2, \ldots, x_n \in \{0,1\}$
A	Algorithm
$c_f(A,x)$	Cost of algorithm A and input x for Boolean function f
$D(f)$	Deterministic complexity of f, $D(f) = \min_A \max_x c_f(A,x)$.
$D_p(f)$	Level-p-complexity of f, $D_p(f) = \min_A \mathbb{E}_x^{\pi_p}[c_f(A,x)]$,
π_p	Input distribution where the bits are i.i.d. Bernoulli(p)-distributed
$\mathbb{E}_{\text{variable}}^{\text{distribution}}$	Expectation over the variable that has a given distribution
$R(f)$	Randomized complexity of f, $R(f) = \min_D \max_x \mathbb{E}_A^D[c(A,x)]$
$I_i^p(f)$	Influence of bit i for f where the bits are π_p-distributed
π	Permutation acting on index i, input x, function f or algorithm A
DICT_i	The function that returns x_i, where the dictator is bit i
PAR_n	The function that is 1 iff there is an odd number of 1's on n bits
ALL_n^1	The function that is 1 iff all inputs are 1 on n bits
ANY_n^1	The function that is 1 iff any input is 1 on n bits
SAME_n	The function that is 1 iff all n bits have the same value
$\mathrm{TRI}_{m,k}$	The tribes function with m blocks of size k
MAJ_n	The majority function on n bits
MAJ_n^k	Iterated majority on k levels with n bits each
f_{AC}	The constructed function $f_{AC}(x) = ANY_2^1(f_A(x_A), f_C(x_C))$, where $(x_A, x_C) = \mathrm{split}_{A,C}(x)$, with sets A and C where $f_A = \neg\mathrm{SAME}_a$ and $f_C = \mathrm{SAME}_c$.

List of Definitions and Results

List of Figures

List of Tables

"Begin at the beginning", the King said, very gravely, "and go on till you come to the end: then stop."

Lewis Carroll, Alice in Wonderland

1 Introduction

Imagine a voter system with yes/no options, for example democracy or dictatorship. How much information of the votes do we need until we can conclude the outcome of the election? How much power does a single voter have to change the result? For dictatorship, we only need the information of the dictator as he or she has all the power, but for a democratic majority we need at least half the votes. Depending on the order in which we find out what the votes are we might need all of them before we can conclude the result.

A voter system can be described as a Boolean function, which is a function of n bits to a single bit, so that $f : \{0,1\}^n \to \{0,1\}$ [Yam12]. We choose the bit values to be $\{0,1\}$, but it can also be represented by some other two-element set such as {no, yes}, {False, True} or $\{-1,1\}$ [CH11]. Besides voting systems, Boolean functions can describe graph properties and circuits built from logic gates [ODo14].

A logic gate is a simple device that implements a Boolean function like AND or XOR. Circuits consist of several logic gates and are the fundamental building blocks of all digital systems. Questions to consider are: how many of the original values of the input must be known to determine the output of the circuit? Which of the input bits matter the most for the result?

Complexity measures of Boolean functions answer the above question of how many bits of the input an algorithm must check before it can determine the value of the function [GS14]. The question about "power" is closely related, and corresponds to the concept of influence. The aim of the project is to study the complexity properties for various Boolean functions.

Some well-known notions of complexity for Boolean functions are deterministic, randomized, nondeterministic and quantum complexity. There is research concerning all of these topics and quantum complexity is a hot research topic now as quantum computers are on the rise [BD02]. Deterministic complexity of monotone Boolean functions and graph properties have been studied in [Ros73] which resulted in the Aanderaa-Rosenberg-Conjecture. The lower and upper bounds of the randomized complexity of the Boolean function called iterated majority has been studied in [Lan+06], [Leo13] and [Mag+16].

The focus of this thesis, however, lies on none of these concepts but on a less well-known one, called level-p-complexity. This is to limit the scope of the thesis, and because level-p-complexity has some interesting connections and differences to the other notions of complexity. The level-p-complexity depends on a probability p, so some interesting properties are explored, such as if it is differentiable, and what the maximum is or if it has more than one maximum.

1.1 Background

In this section we will introduce deterministic and level-p-complexity for Boolean functions, but first the concept of algorithms for Boolean functions will be explained.

Consider a decision tree that queries the n bits of a Boolean function f in a deterministic way depending on previous values revealed. From now on, we will refer to decision trees as algorithms as it is the convention in the literature, even if algorithms are not limited to decision trees. It is assumed that any algorithm stops as soon as the output of the function f can be determined from the given information. An example of an algorithm for the majority function on three bits (which is 1 if there are two or more 1's in the input and 0 otherwise) can be seen in Figure 1.1 where we see that the algorithm stops once the output is known.

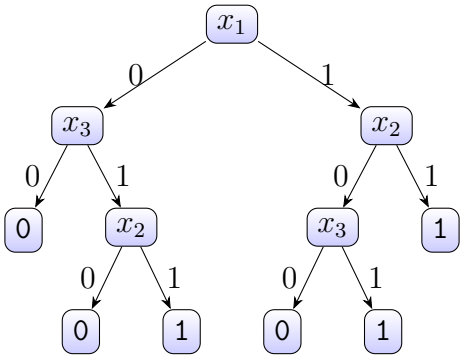


Figure 1.1: An example of an algorithm for 3-bit majority, represented as a decision tree. The algorithm first asks bit 1. Then, if the outcome is 1, it queries 2, while if the outcome is 0, it queries 3. It then queries the last bit if necessary. Note that this algorithm is equally good as the algorithms A_1 and A_2 seen in Figure 1.2.

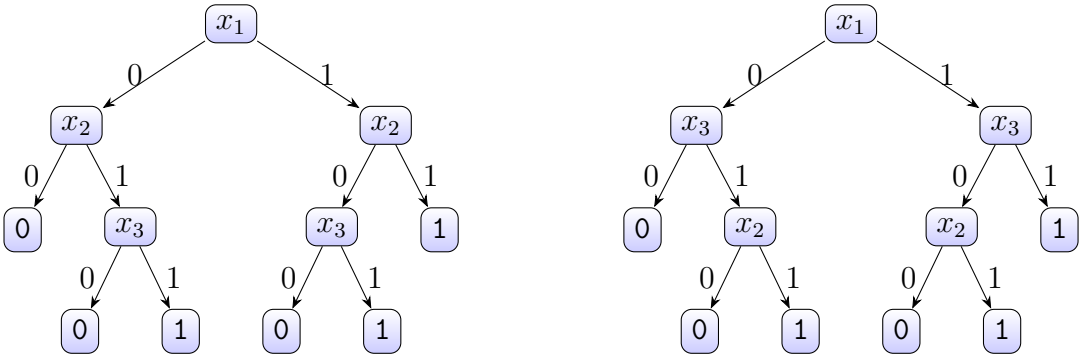
For $x \in \{0,1\}^n$ we let $c_f(A,x)$ be the number of queries that the algorithm A asks when the input to f is x. The subscript f is often left out when the context of the given Boolean function is known. Let $c_f(A) := \max_x c_f(A,x)$ be the maximum number of queries that A makes which we view as the cost of A on f [GS14]. This is the same as the depth of the decision tree: the maximum distance of a leaf from the root.

Definition 1.1 (Deterministic complexity [GS14])**.** The deterministic complexity of a Boolean function f, denoted by $D(f)$, is the minimum of $c_f(A)$ over all deterministic algorithms A for f, so that

$$D(f) = \min_A \max_x c_f(A,x).$$

Two further examples of algorithms for majority on three bits are seen below. The decision trees in Figure 1.2 describe how algorithms A_1 and A_2 behave depending on a particular input. Consider the input $x = (1,0,1)$ which the algorithm does not know. Depending on the order in which the bits are queried the result is attained in 2 or 3 questions, see Figure 1.2. Consider A_1 that first asks bit x_1 then x_2 and lastly x_3, so that $c(A_1,x) = 3$ seen in Figure 1.2a. Now A_2 first asks bit x_1

then x_3, seen in Figure 1.2b. In this case we already know that the majority is 1, without asking x_2, so $c(A_2, x) = 2$, and no other algorithm can do better for this input.



(a) Algorithm A_1. **(b)** Algorithm A_2.

Figure 1.2: Decision trees representing algorithms A_1 and A_2 for 3 bit majority.

If we instead consider a given algorithm, then it might be the case that the input can be chosen so that the algorithm is forced to ask as many questions as possible. As we see in Figure 1.2, we ask all three bits if and only if the first two bits asked are opposite, so the best strategy is to answer 0 and then 1 or the other way around. As we will see later, the deterministic complexity for majority on three bits is 3.

Now, let the bits be independent and identically distributed. We use the distribution π_p for the input bits which means that they are i.i.d. with Bernoulli distribution with parameter $p \in [0, 1]$ [GS14].

Definition 1.2 (Level-p-complexity [GS14])**.** The level-p-complexity of a Boolean function f, denoted by $D_p(f)$, is the minimum over all (deterministic) algorithms A for f of the expected number of questions that are asked when the input has distribution π_p, so that

$$D_p(f) = \min_A \mathbb{E}_x^{\pi_p}[c_f(A, x)],$$

where the notation $\mathbb{E}_{\text{variable}}^{\text{distribution}}$ stands for the expectation over the variable that has a given distribution. Thus, $\mathbb{E}_x^{\pi_p}[c_f(A, x)]$ denotes the expectation of $c_f(A, x)$ when $x \sim \pi_p$.

1.2 Aim

The aim of the project is to study the characteristics of level-p-complexity for various Boolean functions. We will consider questions such as

1. Is $D_p(f)$ continuous and differentiable?
2. What is the maximum of $D_p(f)$ and where is it attained?
3. Is there a Boolean function so that $D_p(f)$ has two maxima?
4. Does the optimal algorithm depend on p?

We will also compare $D_p(f)$ for monotone functions with the lower bound yielded from Theorem 2.3.

1.3 Overview

In Chapter 2, randomized complexity and its connection to game theory are introduced. Moreover, warm-up examples of easy Boolean functions are analyzed and some general properties of Boolean functions are discussed. Next, Chapter 3 explains how the computations were done using Mathematica and Haskell. In Chapter 4, we calculate the level-p-complexity for the Boolean functions all, tribes, majority and iterated majority in some special cases. Chapter 5 concerns the interplay between Boolean functions and graphs, and we calculate the level-p-complexity for connectivity of graphs with three or four nodes. Next, in Chapter 6, we construct a Boolean function so that the level-p-complexity has two maxima. Lastly, Chapter 7 concludes with answers to questions 1-4 in Section 1.2 for the Boolean functions and presents open questions and future work.

As far as I can see, there could be no theory of games ... without that theorem ... I thought there was nothing worth publishing until the Minimax Theorem was proved.

John von Neumann

2 Theory

In this chapter, basic game theory and the concept of randomized complexity is explained to give a broader understanding of the research field. Next, we show some examples of the complexity of Boolean functions. Lastly, we go through theory required for the thesis like graph theory and some general properties of Boolean functions.

2.1 Game theory

A two-player zero-sum game is a mathematical representation in game theory of a situation which involves two sides, where the result is a gain for one side and a loss for the other. For a zero-sum game, adding the total gains and losses corresponds to a sum of zero [KP17]. A simple example of a zero-sum game is matching pennies seen in Figure 2.1. Both players say either 1 or 0 and if they say the same, player 1 gives player 2 a penny and if they say different, player 2 gives player 1 a penny. In matching pennies the pure strategies only consist of one move, either 1 or 0. Moreover, the game is played simultaneously so that the players cannot base their move on the other player's move.

		Player 2	
		0	1
Player 1	0	−1	1
	1	1	−1

Figure 2.1: The payoff matrix for the zero-sum game of matching pennies. For player 2 the payoff matrix has opposite values.

Allowing mixed strategies widens the playing field and Theorem 2.1 can be used to define the value of the game [KP17]. A mixed strategy is of the form $\sum_{i=1}^{m} p_i e_i$ for $p_i \in [0,1]$ and $\sum_{i=1}^{m} p_i = 1$ where $e_i, i = 1 \ldots, m$ are the m possible pure strategies [KP17]. Returning to the example of matching pennies, a good mixed strategy for player 1 would be to pick 0 and 1 uniformly at random, so that $x = [1/2, 1/2]^T$. Then

$$x^T My = [1/2, 1/2] \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} y = [0,0]y = 0,$$

which means that $x^T My$ is 0 regardless of the strategy y of player 2. Since the

game is symmetric the same holds for player 1. The value of the game for matching pennies is 0 as described in Theorem 2.1.

In general for two-player zero-sum games with $m \times l$ payoff matrix M we know that

$$\forall x_0, y_0, \max_x x^T M y_0 \geq x_0^T M y_0 \geq \min_y x_0^T M y.$$

which gives the inequality

$$\min_y \max_x x^T M y \geq \max_x \min_y x^T M y \tag{2.1}$$

Theorem 2.1 states that (2.1) is an equality, yielding the value of the game.

Theorem 2.1 (von Neumann's Minimax theorem [Neu28])**.** For any two-person zero-sum game with $m \times l$ payoff matrix M, there is a number V, called the value of the game, satisfying:

$$\max_x \min_y x^T M y = V = \min_y \max_x x^T M y,$$

where x and y are mixed strategies.

Note that x and y range over all probability distributions of columns and rows, respectively. If we restrict the domain to only pure strategies, Theorem 2.1 does not hold. One counterexample is the following for matching pennies

$$\begin{aligned}\min_{y \text{ pure}} \max_{x \text{ pure}} x^T M y &= [1,0] \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 \geq -1 = \\ &= [1,0] \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \max_{x \text{ pure}} \min_{y \text{ pure}} x^T M y\end{aligned}$$

In this case inequality (2.1) holds strictly.

However, not both x and y need to be mixed. As we noted earlier in the example of mixed pennies, when player 1 has the optimal mixed strategy, all strategies for player 2 give the same value of the game, including the pure strategies. In fact, considering the expression $\min_y x^T M y$, we are minimizing a linear function ($x^T M y$ is linear in y) over a convex set (the set of mixed strategies $\{\sum_{i=1}^m p_i e_i | p_i \in [0,1]\}$). From linear programming it follows that the minimum is found at a corner point of the convex set i.e. a pure strategy e_i. Thus for $\max_x \min_y x^T M y$, y will be pure and x mixed. With a similar argument, x is pure and y is mixed in the expression $\min_y \max_x x^T M y$. So Theorem 2.1 holds when x is mixed and y is pure in the LHS and when x is pure and y is mixed in the RHS.

2.2 Randomized complexity

Now that we see the advantages of mixed strategies we introduce a new notion of complexity, which is defined similarly to the deterministic complexity but with an essential difference. This represents a zero-sum game where one player chooses an algorithm and the other chooses input. Mixed strategies correspond to randomizing

the algorithm or the input. Given an algorithm A and input x the cost $c(A, x)$ can be computed and is an entry in the payoff matrix at the row corresponding to A and column corresponding to x. The two players are the one that chooses algorithm A and the one that chooses input x, and $c(A, x)$ is seen as a loss for the player choosing A, since he or she wants to minimize it, and a gain for the player choosing x, since he or she wants to maximize it [KP17].

Definition 2.1 (Randomized complexity [GS14])**.** The randomized complexity of a Boolean function f, denoted by $R(f)$, is the minimum of the maximum expected cost $\max_x \mathbb{E}_A^D[c(A, x)]$ where we use a distribution D over deterministic algorithms, so that

$$R(f) = \min_D \max_x \mathbb{E}_A^D[c(A, x)].$$

This means we should choose probabilities p_i for each deterministic algorithm A_i so that

$$\max_x \sum_{k=1}^{m} p_k c(A_k, x)$$

is minimized, where m is the number of deterministic algorithms. From Theorem 2.1,

$$\max_P \min_A \mathbb{E}_x^P[c(A, x)] = \min_D \max_x \mathbb{E}_A^D[c(A, x)]$$

which means that if the adversary chooses an input distribution, we can do our best by subsequently choosing the best deterministic algorithm, rather than a randomized algorithm, for this input distribution [LY02].

A lower bound for $R(f)$ can be found by

$$\max_{p\in(0,1)} D_p(f) = \max_{p\in(0,1)} \min_A \mathbb{E}_x^{\pi_p}[c(A, x)] \leq \max_P \min_A \mathbb{E}_x^P[c(A, x)] = R(f)$$

since there might be a general distribution P that is better for x than π_p, even for maximal p, and the last equality holds due to Theorem 2.1. Thus, $\max_{p\in(0,1)} D_p(f)$ gives a lower bound for $R(f)$ which can be of help if the full randomized complexity is hard to determine. However, to find for which p that D_p is maximized is still non-trivial.

2.3 Some warm-up examples

The Boolean functions that are constantly 0 or 1 independent of input have $D(f) = D_p(f) = R(f) = 0$ since no questions need to be asked to know the result.

The dictator function is defined as $\mathrm{DICT}_i(x_1, \ldots, x_n) = x_i$ where the dictator is bit i [GS14]. Then there is only one minimizing algorithm irrespective of input: the one that queries bit i first. After asking the ith bit the function is reduced to the constant function, and $D(\mathrm{DICT}_i) = \min_A \max_x c(A, x) = 1$. Similarly, $D_p(\mathrm{DICT}_i) = 1$, since changing the input does not affect the choice of algorithm (there is only one best choice). Lastly $R(\mathrm{DICT}_i) = 1$ since randomizing the algorithm could not make it better than the deterministic choice for this case.

The parity function is the one where

$$\mathrm{PAR}_n(x_1, \ldots, x_n) = \begin{cases} 0, & \text{if there is an even number of 1's} \\ 1, & \text{if there is an odd number of 1's.} \end{cases}$$

In this case all questions have to be asked to determine the parity, regardless of input. Thus, $D(\mathrm{PAR}_n) = D_p(\mathrm{PAR}_n) = R(\mathrm{PAR}_n) = n$.

2.4 Properties of Boolean functions

The first property we introduce is evasiveness, which is useful due to Theorem 2.2.

Definition 2.2 (Evasive [GS14])**.** A Boolean function f on n variables is called evasive if there is an adversary who can answer each bit query so that all bits have to be queried.

Theorem 2.2 (Evasive)**.** A Boolean function f is evasive if and only if $D(f) = n$.

The $\implies$ direction is rather straightforward to prove, but the converse is more tricky. However, we will mostly use the $\implies$ direction, since it is often easier to prove that a function is evasive than $D(f) = n$.

Definition 2.3 (Monotonicity [OS07])**.** A function f is monotone if $x \leq y$ (meaning $x_i \leq y_i$ for each i) implies that $f(x) \leq f(y)$.

The intuition for monotonicity of Boolean functions is that for any bit that we switch from 0 to 1 in the input, the result can either stay the same or be switched up from 0 to 1. Some simple examples of monotone Boolean functions are the dictator function and majority. Parity does not satisfy monotonicity, since changing a bit from 0 to 1 could change the output from 1 to 0.

Definition 2.4 (Dual)**.** Let the line over a boolean variable denote flipping the bits value as

$$\overline{x_i} = \begin{cases} 1 \text{ if } x_i = 0, \\ 0 \text{ if } x_i = 1. \end{cases}$$

Then f is dual to g if

$$f(x_1, \ldots, x_n) = \overline{g(\overline{x_1}, \ldots, \overline{x_n})}.$$

If $f(x_1, \ldots, x_n) = \overline{f(\overline{x_1}, \ldots, \overline{x_n})}$ we say that f is self-dual, or odd [ODo14]. Functions that are self-dual are for example parity for odd n, majority and dictator.

A permutation $\pi \in S_n$ is a bijection from a set S to itself, reordering the elements. Here S_n denotes the full symmetry group on n elements, also known as the set of permutations of size n. For our case, let S be the set of indices $\{1, \ldots, n\}$. Permutations $\pi \in S_n$ act on strings $x \in \{0,1\}^n$ as: $(x^\pi)_i = x_{\pi^{-1}(i)}$. They also act on functions $f : \{0,1\}^n \to \{0,1\}$ via $f^\pi(x) = f(x^\pi)$ for all $x \in \{0,1\}^n$ [ODo14]. Further, permutations can also act on algorithms, as can be seen in Equation 3.1.

Definition 2.5 (Symmetric [ODo14])**.** A Boolean function f is symmetric if $f^\pi = f$ for all permutations $\pi \in S_n$.

This means that the output of f should be the same no matter how we reorder the input, so that $f(x)$ only depends on the number of 1's and 0's in x. This holds for the parity, majority and constant function, but not for the dictator function. It is easier to calculate complexity for symmetric functions since many of the algorithms will be equivalent up to a permutation. The symmetry property is thus highly desirable but rarely holds. However, there is a weaker definition of symmetry that we can use.

Definition 2.6 (Transitive [ODo14])**.** A function $f : \{0,1\}^n \to \{0,1\}$ is transitive if for all $i, j \in \{1, \ldots, n\}$ there exists a permutation $\pi \in S_n$ taking i to j such $f(x^\pi) = f(x)$ for all $x \in \{0,1\}^n$.

It is the case that all symmetric functions are transitive but not vice versa. Let

$$\Gamma(f) = \{\pi \in S_n : f^\pi = f\}$$

be the automorphism group of f which is the set of permutations of the variables that leave f unchanged. For symmetric functions all permutations leave f unchanged so $\Gamma(f) = S_n$.

Now we introduce pivotality and influence, that come from political science and is called the Banzhaf power index, named after John F. Banzhaf III [Ban64] (originally invented by Lionel Penrose in 1946 [Pen46]). It is defined by the probability of changing an outcome of a vote by a single voter changing their vote.

Definition 2.7 (Pivotality [GS14])**.** Given a Boolean function f, an input x, and an index $i \in \{1, \ldots, n\}$, we say that i is pivotal for f for x if $f(x_1, \ldots, x_i, \ldots, x_n) \neq f(x_1, \ldots, \bar{x_i}, \ldots, x_n)$.

Note that being pivotal does not depend on the value of x_i, but it is measurable with respect to the other variables.

Definition 2.8 (Pivotal set [GS14])**.** The pivotal set, $\mathcal{P}$, for f and an input x, is the set of $\{1, \ldots, n\}$ given by $\mathcal{P}(x) = \mathcal{P}_f(x) := \{i \in \{1, \ldots, n\} : i \text{ is pivotal for } f \text{ for } x\}$.

In words, the pivotal set is for each x, the set of bits with the property that if you flip the bit, then the function output changes.

Definition 2.9 (Influence [GS14])**.** The influence vector at level p, $I_i^p(f), i \in \{1, \ldots, n\}$, is defined by

$$\begin{aligned} I_i^p(f) :=& \mathbb{P}_p(i \text{ is pivotal for } f) \\ =& \mathbb{P}_p(f(x_1, \ldots, x_i, \ldots, x_n) \neq f(x_1, \ldots, \bar{x_i}, \ldots, x_n)) \\ =& \mathbb{P}_p(i \in \mathcal{P}(x)), \end{aligned}$$

where $\mathbb{P}_p$ is the probability measure of the distribution π_p.

Definition 2.10 (Total influence [GS14])**.** The total influence at level $p, \mathbf{I}^p(f)$, is defined by $\mathbf{I}^p(f) := \sum_i I_i^p(f) = \mathbb{E}_x^{\pi_p}[|\mathcal{P}(x)|]$.

The only bit in the dictator function that is pivotal for any x is bit i, which has influence 1, so the total influence 1. For the parity function for all x all bits change the result if their value is switched, so they are pivotal for all x with influence 1, so the total influence is n. Influence yields a lower bound for the level-p-complexity.

Theorem 2.3 (Lower bound [OS07])**.** For monotone Boolean functions f we have

$$D_p(f) \geq 4p(1-p)(\mathbf{I}^p(f))^2.$$

We now verify that the lower bound holds for some simple examples. It holds for the dictator function since $D_p(f) = 1 \geq 4p(1-p)1^2 = 4p(1-p)(\sum_i I_i^p(f))^2$ and $p \in [0,1]$. Equality holds if and only if $p = 1/2$. For the parity function the theorem cannot be applied since it is non-monotone.

2.5 Graph theory

A graph G consists of a set of nodes $\mathcal{N}$ and a set of edges $\mathcal{E}$. In this thesis we will consider graphs with $\mathcal{N} = \{1, \ldots, n\}$ and $E \subseteq \{\{i,j\} \text{ for } i \in \mathcal{N}, j \in \mathcal{N}, i \neq j\}$. Take an edge $e = \{u, v\}$; we then say that e is incident to u and v, which are themselves incident to e. The nodes u and v are said to be adjacent or neighbours; likewise two edges are adjacent if they have a node in common. The degree of a node v, denoted $deg(v)$, is the number of edges incident to v [BM+76].

Definition 2.11 (Path)**.** A path P of a graph G is a sequence of distinct nodes in G, in which every node is adjacent to the next.

Definition 2.12 (Connectivity [BM+76])**.** A graph G is connected if, for every pair of nodes u and v in G, there exists a path from u to v; otherwise it is disconnected.

Definition 2.13 (Isomorphy [Knu12])**.** The graphs G and H are isomorphic if there is a bijection f from the nodes of G to H such that any two nodes u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H.

Graph properties can be represented as Boolean functions, which we will see in Section 5 where we consider the graph property of connectivity.

Nowadays we can do computer experiments using `Mathematica`*, and even solve a system of 42 equations. This offers another route to knowledge, rather than mere ideas.*

John Nash

3 Method

This chapter explains the method for computing the level-p-complexity using helpful software such as `Haskell` and `Mathematica`[1]. First we explain the method for generating algorithms for a Boolean function and computing their expected cost by hand or using `Haskell`. Note that most calculations are still done by hand but `Haskell` is used when calculations by hand were too complicated, and for checking that the results are correct. Given expected costs of all algorithms for a Boolean function, either calculated by hand or with `Haskell`, we show how we can find $D_p(f)$ using `Mathematica`.

3.1 Calculation of expected cost

One way to calculate expected costs of algorithms is by introducing the indicator function $\mathbb{I}_i^A(x)$ as 1 if, given an input x, bit i is asked in algorithm A, and 0 otherwise. The interpretation of bit i is different depending on context, either it is a natural order of bits in the input or the order that the algorithm asks the bits. Then $c(A, x) = \sum_{i=1}^{n} \mathbb{I}_i^A(x)$ is the number of questions asked. Then

$$\mathbb{E}_x^{\pi_p}[c(A,x)] = \mathbb{E}_x^{\pi_p}\left[\sum_{i=1}^{n} \mathbb{I}_i^A(x)\right] = \sum_{i=1}^{n} \mathbb{E}_x^{\pi_p}[\mathbb{I}_i^A(x)] = \sum_{i=1}^{n} \mathbb{P}_p(\text{bit } i \text{ is asked for alg. } A)$$

Another way is to calculate the expected cost recursively. Recall the definition of an algorithm as a decision tree, where each node is either a bit query with two subalgorithms or a leaf containing the result. The bit query is represented as Pick i t_0 t_1 which means that we pick a bit i to query, and if $x_i = 0$ we go to subtree t_0 and otherwise we go to subtree t_1. When the algorithm stops we end up at a leaf in the decision tree, either Res 0 or Res 1. Using this syntax, the algorithm in Figure 3.1 can be described as

```
Pick 1 (Pick 3 (Res 0) (Pick 2 (Res 0) (Res 1)))
       (Pick 2 (Pick 3 (Res 0) (Res 1)) (Res 1))
```

Given an algorithm, expected cost is calculated by a recursive method. The expected cost of the base case is 0, since we already know the result and don't have to ask any more questions. Furthermore, for Pick i t_0 t_1 we first have to ask one question and then with probability $\mathbb{P}(x_i = 0) = (1 - p)$ we ask as many questions

[1]The code is found in the `git` repository
`https://github.com/juliajansson/ComplexityOfBooleanFunctions`

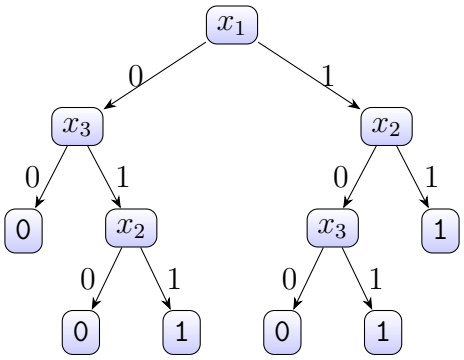


Figure 3.1: An example of an algorithm for 3-bit majority, represented as a decision tree. The algorithm first asks bit 1. Then, if the outcome is 1, it queries 2, while if the outcome is 0, it queries 3. It then queries the last bit if necessary.

that are needed for t_0. Added to this, with probability $\mathbb{P}(x_i = 1) = p$ we ask as many questions that are needed for t_1. We get the following formula:

$$\begin{aligned} \text{Ex}[\text{Res } b] &= 0 \\ \text{Ex}[\text{Pick } i\ t_0\ t_1] &= 1 + (1-p) \cdot \text{Ex}[t_0] + p \cdot \text{Ex}[t_1] \end{aligned}$$

More information can be found in the file `LevelpComplexity.hs`. This recursive method of calculating expected cost is sometimes used in manual calculations as well, for example in Chapter 5. The expectation of the algorithm in Figure 1.1 is

$$\text{Ex}[\text{Pick } 1\ t_0\ t_1] = 1 + (1-p) \cdot (1+p) + p(1 + (1-p)) = 2 + 2p - 2p^2.$$

We will see in Section 4.3 that this formula is in fact the result for $D_p(\text{MAJ}_3)$.

Note that the expected cost of any algorithm for a Boolean function of n bits will always be a polynomial. Thus the minimum over algorithms will yield a function that is piecewise polynomial, and thus continuous. This means that the level-p-complexity will always be continuous which answers the first part of question 3 in Section 1.2.

3.2 Polynomial representation

The result of the expectation calculation is a polynomial $\sum_{i=0}^{n} a_i p^i$, which can also be represented with a symmetric base as $\sum_{i=0}^{n} b_i \binom{n}{i} p^i (1-p)^{n-i}$. The base is called symmetric since for $i \in \{0, \ldots, n\}$ it holds that $b_i = b_{n-i}$ if and only if $f(p) = f(1-p)$ for $p \in [0, 1]$. This symmetric representation also allows us to see the number of questions b_i on average for all the different cases where we have i 1's and $n - i$ 0's. Since all $p^i(1-p)^{n-i}$ are ≥ 0 for $p \in [0, 1]$ we can thus compare different expected costs in their symmetric representation to see how many more questions they would ask in any particular case. This is an advantage compared to the regular polynomial representation, where the coefficients might also be negative, and thus hard to interpret in the case of questions asked. Conversion between these representations can be seen in `Symbase.hs` and an illustration is seen in `symbase.nb`. The polynomial implementation relies heavily on material from [JIB22].

3.3 Generating algorithms

When we query a bit i for a Boolean function f we get two reduced subfunctions, $f^{x_i=0}$ when $x_i = 0$ and $f^{x_i=1}$ when $x_i = 1$. Algorithms are generated given the Boolean function f recursively by either asking each bit i, and generating the algorithms for the subfunctions $f^{x_i=0}$ and $f^{x_i=1}$, or stopping if we know the result already which is the case if f is constant. The recursive step is shown in the formula below:

$$\begin{aligned}\text{GenAlg}_n(f) = \{\text{Pick } i\ t_0\ t_1 \mid i \in \{1,\ldots,n\}, \quad & t_0 \in \text{GenAlg}_{n-1}(f^{x_i=0}), \\ & t_1 \in \text{GenAlg}_{n-1}(f^{x_i=1})\}\end{aligned}$$

This method does not take any symmetry into account, however, so for the majority function on three bits we have 12 different algorithms that are generated. But due to symmetry, only one of them is relevant, the other ones are permutations. We define a permutation acting on an algorithm as

$$(\text{Res } b)^\pi = \text{Res } b \tag{3.1a}$$

$$(\text{Pick } i\ l\ r)^\pi = \text{Pick } \pi(i)\ (l)^\pi\ (r)^\pi \tag{3.1b}$$

To get remove some of the equivalent permutations we filter with the following method. For $i \in \{1,\ldots,n\}$ we filter out the indices $j \in \{1,\ldots,n\}$ where $f^{x_j=0} = f^{x_i=0}$ and $f^{x_j=1} = f^{x_i=1}$. This is because if the subfunctions are equal for i and j it does not matter which index of i and j we choose, and it is enough to only consider index i. The code is found in the file `GenAlg.hs`.

3.4 Computing the level-p-complexity

When we have all the expected costs of the algorithms we can write them in `Mathematica` and plot them to see what they look like. There we often see which of them are the lowest for a particular $p \in [0,1]$ and get a sense of how $D_p(f)$ looks. We can also check where some costs intersect by using the `NSolve` function. We can also evaluate the derivative of the costs in this intersection point by using symbolic derivative and then evaluating it in the given point. More formally, we get the answer by using the `Min` function of all the expected costs and then expand the expression for $p \in (0,1)$. This corresponds to $D_p(f)$ since we minimized over all algorithms. We can find the local maximum of $D_p(f)$ by using the `FindMaximum` function if we add a suitable starting point, which can be guessed by plotting the function.

Consider an example of a Boolean function f on five bits that has two algorithms A_1 and A_2 that ask all the bits in order or reversed[2]. In Listing 3.1 we see some `Mathematica` code for the expected cost A_1 and A_2. First we use `NSolve` to find where the polynomials intersect, namely at $p = p_\star = 0.356158$ and $p = 1 - p_\star$. Then we expand the minimum of these functions using `PiecewiseExpand` and `Min`

[2]This is a simplified example of the algorithms for f_{AC} in Chapter 6.

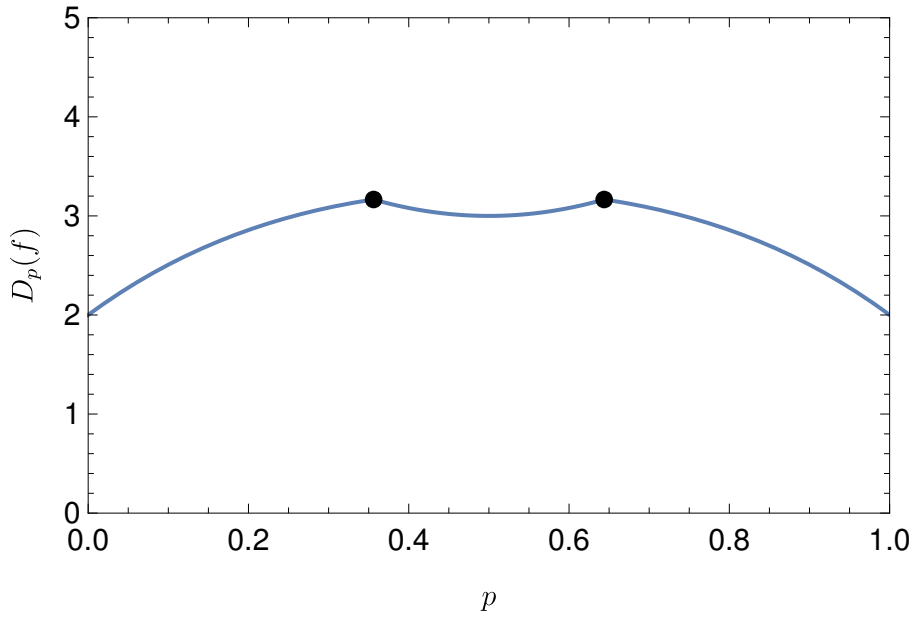


Figure 3.2: The minimum of the costs of A_1 and A_2 from the sample `Mathematica` code example plotted. The red dots note the intersections of costs of the algorithms.

to get the formula for $D_p(f)$ given a function f that only has the two algorithms A_1 and A_2. In Figure 3.2 we see the minimum of these functions plotted, with their intersection points as red dots. Further we find the maxima of $D_p(f)$ by using `FindMaximum` with guesses 0.4 and 0.6 from the graph respectively.

```
In: A1[p_] = 5 - 8 p + 8 p^2;
In: A2[p_] = 2 + 6 p - 10 p^2 + 8 p^3 - 4 p^4;
In: NSolve[A1[p] == A2[p] && p < 1 && p > 0]
Out: {{p -> 0.356158}, {p -> 0.643842}}
In: D[p_] = PiecewiseExpand[Min[A1[p],A2[p]], 0 < p < 1]]
Out: \[Piecewise] 5 - 8 p + 8 p^2,  0.356<=p<=0.644
 2 + 6 p - 10 p^2 + 8 p^3 - 4 p^4,  True
In: FindMaximum[D[p], {{p, 0.4}}]
Out: {3.16553, {p -> 0.356158}}
In: FindMaximum[D[p], {{p, 0.6}}]
Out: {3.16553, {p -> 0.643842}}
```

Listing 3.1: Sample `Mathematica` code.

There are 10 types of people in the world, those who know binary and those who don't.

Anonymous

4 Complexity of Boolean functions

In this chapter we will calculate the level-p-complexity for different Boolean functions like all, tribes, majority and iterated majority.

4.1 All

The Boolean function all is defined as

$$\mathrm{ALL}_n^1(x_1, \ldots, x_n) = \begin{cases} 1, \text{if all bits are 1's} \\ 0, \text{otherwise.} \end{cases}$$

This function is the negation of

$$\mathrm{ANY}_n^0(x_1, \ldots, x_n) = \begin{cases} 1, \text{if any bit is 0} \\ 0, \text{otherwise.} \end{cases}$$

which means $\mathrm{ALL}_n^1(x) = \overline{\mathrm{ANY}_n^0(x)}$. Since only the output is switched, they have the same complexity. The ALL_n^1 function is evasive because there is an adversary who can give answers to bit queries which always forces all the bits to be queried. The method is to answer 1 for all the bit queries, since the answer cannot be known until the last question is asked. This gives that $D(\mathrm{ALL}_n^1) = n$.

Proposition 4.1 ($D_p(\mathrm{ALL}_n^1)$)**.** For the ALL_n^1 function, the level-p-complexity is

$$D_p(\mathrm{ALL}_n^1) = \sum_{k=0}^{n-1} p^k.$$

Proof. Consider the algorithm that asks the bits in order from left to right, but stops if the bit last asked is 0. In this case we know that the output is 0, and otherwise we continue until we reach either output 0, or, if all bits are 1, output 1. Since the ALL_n^1 function is symmetric no algorithm will have a lower cost. The first bit has to be asked, so it has probability 1. We ask the next bit only if all previous bits are 1, so the probability of asking bit j is p^{j-1}. Consequently, we get

$$D_p(\mathrm{ALL}_n^1) = 1 + p + \ldots + p^{n-1} = \sum_{k=0}^{n-1} p^k$$

□

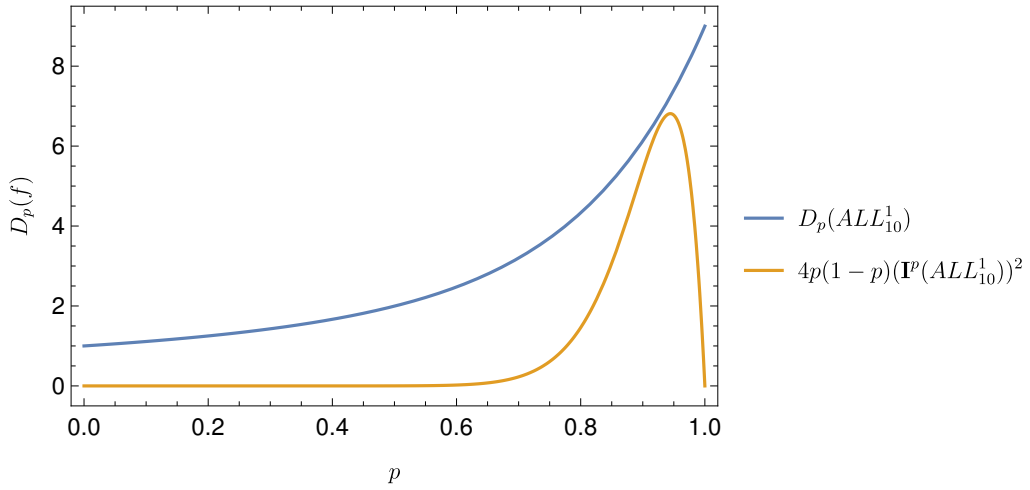


Figure 4.1: Level-p-complexity and lower bound for the ALL^1_{10} function. The code for the graph can be seen in `all.nb`.

The ALL^1_n function is monotone, since changing a value from 0 to 1 either increases the value or stays the same. It is also symmetric but not self-dual. The only case when bit i changing value leads to a change of result for the ALL^1_n function is when all other bits are 1. Thus, the influence is $I^p_i(f) = p^{n-1}$, and the lower bound in Theorem 2.3 is $4p(1-p)(np^{n-1})^2$. For the example of $n = 10$ we see the lower bound in Figure 4.1.

For the ALL^0_n function where we consider all 0's instead of all 1's we get the same deterministic complexity but $D_p(\mathrm{ALL}^0_n) = \sum_{k=0}^{n-1}(1-p)^k$. This function is the same as the negation of

$$\mathrm{ANY}^1_n(x_1, \ldots, x_n) = \begin{cases} 1, \text{if any bit is } 1 \\ 0, \text{otherwise.} \end{cases}$$

Later we will consider the function $\mathrm{SAME}_n = \mathrm{ALL}^0_n \vee \mathrm{ALL}^1_n$ which is 1 if all bits are the same.

4.2 Tribes

Partition $n = m \cdot k$ into m disjoint blocks of length k. Define $\mathrm{TRI}_{m,k}$ to be 1 if there exists at least one block of the form $1, 1, \ldots, 1$, and 0 otherwise. We know that if $k = 1$ the block size is one, so it corresponds to the ANY^1_m function and $D(\mathrm{TRI}_{m,1}) = m$ and $D_p(\mathrm{TRI}_{m,1}) = \sum_{i=0}^{m-1}(1-p)^i$. If $m = 1$ there is only one block, so we get the ALL^1_k function so that $D(\mathrm{TRI}_{1,k}) = k$ and $D_p(\mathrm{TRI}_{1,k}) = \sum_{i=0}^{k-1} p^i$.

The tribes function is transitive but not symmetric. We can make permutations in a block or move all bits in a block simultaneously, but we cannot move single bits between different blocks and get the same result.

Our algorithm asks the bits from left to right and for each bit we continue asking the next if the value is 1, and skip right to the next block if the value is 0. This is because if we get 0 we can immediately conclude that the block cannot contain

all 1's, so it is better to start asking the next block. If we get 1 we cannot rule out that the current block contains all 1's, so we should continue asking the bits in the block. If we find a block with all 1's we conclude that the result of the function is 1, and stop asking questions. Otherwise, if all blocks contain some 0s, we conclude that the result of the function is 0, and stop asking questions.

Now, this algorithm only has to ask until a block with all 1's is found, see example in Figure 4.2. However, if the input is such that that all blocks contain $k-1$ 1's and then a 0 in last position of the block, then the algorithm has to ask through all of a block for all blocks, see the longest path in the decision tree in Figure 4.2. To show that the function is evasive we have to show that all queries must be asked by giving suitable answers to each bit query. The method is to answer 1 unless the last unasked bit in a block is asked, in which case we answer 0. This accounts to $m \cdot k$ queries so that the function is evasive and thus $D(\mathrm{TRI}_{m,k}) = n$.

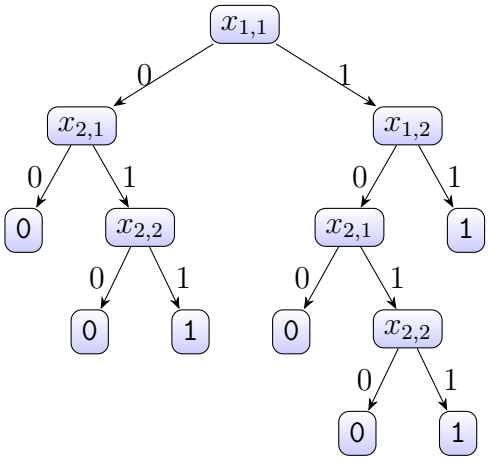


Figure 4.2: An example of the algorithm that asks in order for $\mathrm{TRI}_{2,2}$.

Next, we calculate level-p-complexity.

Proposition 4.2 ($D_p(\mathrm{TRI}_{m,k})$)**.** For the tribes function, the level-p-complexity is

$$D_p(\mathrm{TRI}_{m,k}) = \sum_{i=1}^{m}(1-p^k)^{i-1}\sum_{j=1}^{k}p^{j-1} = \frac{(1-(1-p^k)^m)(1-p^k)}{p^k(1-p)}.$$

Proof. The previous algorithm is not worse than any other considering that the order of asking the bits in the blocks does not matter, since the function is transitive. The order in which we ask the blocks does not make a difference either, and it clearly does not make sense to change blocks if you got all 1's so far. Thus calculating the cost for this algorithm will yield D_p. The expected number of queries asked is the same as the sum over j of the probability that the jth bit is asked. The probability that bit j is asked depends on which block we are in and where we are in the block. Let the pair (i,j) represent bit j in block i and $p_{(i,j)}$ the probability that this bit is asked. Then $p_{(1,j)} = p^{j-1}$ (the probability of all previous bits being 1), but $p_{(i,j)}$ depends on if we have previously had all 1's in another block. Since the probability of having all 1's in a block is p^k the probability of not having all 1's in a block

before block i is $(1-p^k)^{i-1}$. So $p_{(i,j)} = (1-p^k)^{i-1}p^{j-1}$. Altogether we get

$$\begin{aligned} D_p(\mathrm{TRI}_{m,k}) &= \sum_{i=1}^{m}\sum_{j=1}^{k}(1-p^k)^{i-1}p^{j-1} = \sum_{i=1}^{m}(1-p^k)^{i-1}\sum_{j=1}^{k}p^{j-1} \\ &= \frac{1-(1-p^k)^m}{1-(1-p^k)} \cdot \frac{1-p^k}{1-p} = \frac{(1-(1-p^k)^m)(1-p^k)}{p^k(1-p)} \end{aligned}$$

□

We know that the extreme cases for the level p complexity are $D_0(\mathrm{TRI}_{m,k}) = m$, since it is enough to check all m blocks once, when we get a zero, and $D_1(\mathrm{TRI}_{m,k}) = k$ since we are done after checking all the 1's in a block. We see that this formula holds in the extreme cases since

$$D_p(\mathrm{TRI}_{m,k}) = (1 + (1-p^k) + \ldots + (1-p^k)^{m-1})(1 + p + \ldots + p^{k-1}),$$

so that $D_0(\mathrm{TRI}_{m,k}) = m$ and $D_1(\mathrm{TRI}_{m,k}) = k$.

4.2.1 Maximizing the complexity of tribes

We start by studying a simple example with $m = k = 2$. Then

$$D_p(\mathrm{TRI}_{2,2}) = (1 + (1-p^2))(1+p) = 2 + 2p - p^2 - p^3.$$

We find the maximum by setting the derivative to zero

$$D'_p(\mathrm{TRI}_{2,2}) = 2 - 2p - 3p^2 = 0 \implies p = \frac{-1 \pm \sqrt{7}}{3},$$

and checking the second derivative at the relevant point ($p = \frac{\sqrt{7}-1}{3} \approx 0.55 \in [0,1]$)

$$D''_p(\mathrm{TRI}_{2,2}) = -2 - 6p = -2 - 6\frac{\sqrt{7}-1}{3} = -2 - 2(\sqrt{7}-1) = -2\sqrt{7} < 0.$$

So in the interval $[0,1]$ the maximum is $D_p(\mathrm{TRI}_{2,2}) \approx 2.63$ when $p \approx 0.55$.

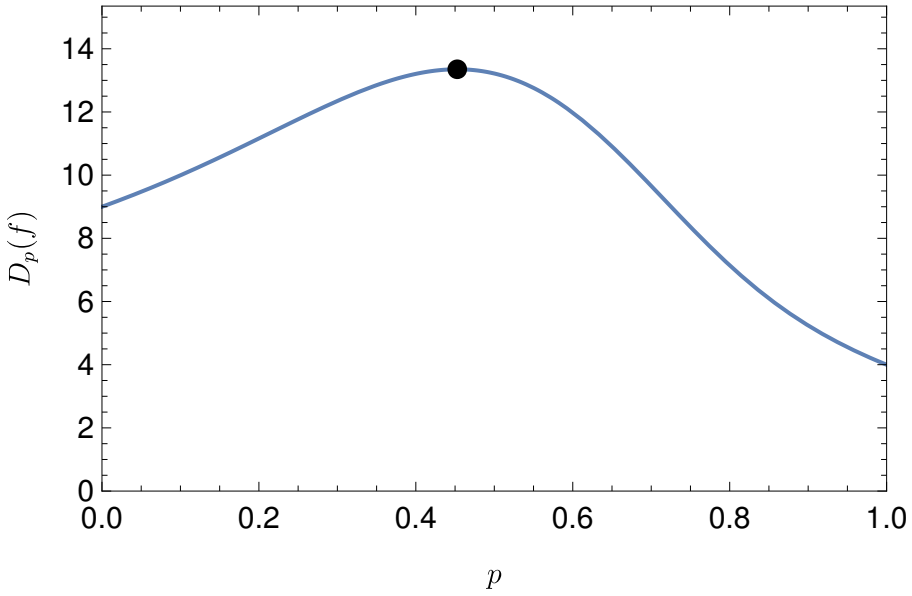
The level-p-complexity for the special case $k = 1$ and general m is

$$D_p(\mathrm{TRI}_{m,1}) = 1 + (1-p) + \ldots + (1-p)^{m-1}.$$

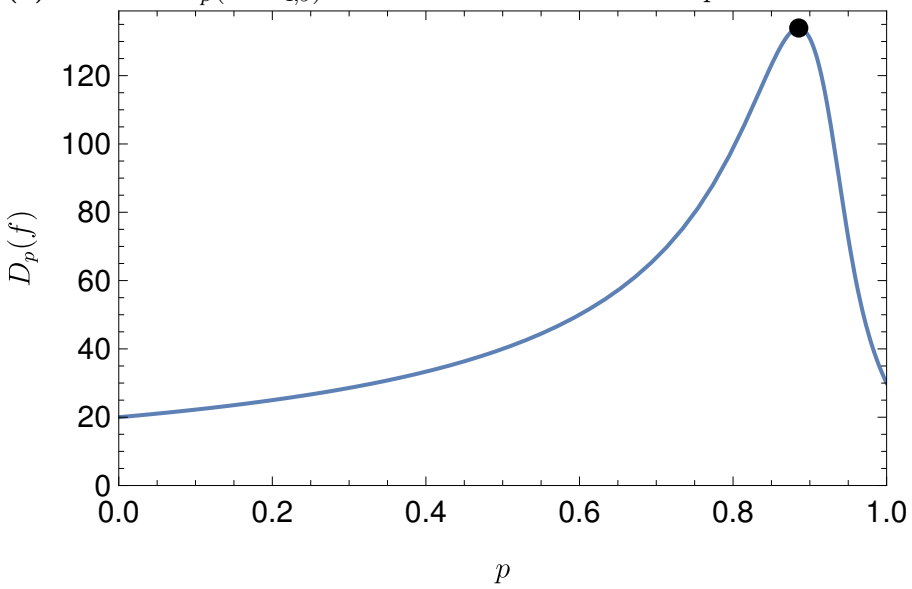
We see that this function is monotonically decreasing in the interval $[0,1]$ from m to 1 which means that the maximum is attained at $p = 0$ and the value is m. For $m = 1$ and general k, it is the case that

$$D_p(\mathrm{TRI}_{1,k}) = 1 + p + \ldots + p^{k-1}.$$

We see that this function is monotonically increasing in the interval $[0,1]$ from 1 to k which means that the maximum is attained at $p = 1$ and the value is k. In fact this case is reduced to the ALL function previously studied.



(a) Plot for $D_p(\mathrm{TRI}_{4,9})$ with maximum ≈ 13.35 at $p \approx 0.45$.



(b) Plot for $D_p(\mathrm{TRI}_{30,20})$ with maximum ≈ 133.94 at $p \approx 0.89$.

Figure 4.3: Plots of D_p for different tribes examples, the code can be found in the `Mathematica` script `TribesDpScript.wls`.

For more general m and k we want to show that there is a unique maximum in the interval $[0, 1]$, so we compute the general derivative. We get

$$D'_p(\mathrm{TRI}_{m,k}) = \frac{k((m+1)(1-p^k)^m - 1)p^k(1-p) - (1-(1-p^k)^m)(1-p^k)(k(1-p)-p)}{p^{k+1}(1-p)^2}$$

The formula for the derivative is quite complicated and we cannot find a solution analytically. Examples where we calculate the result numerically indicates that there is a unique maximum for the level-p-complexity, seen in Figure 4.3. An

alternate form of the derivative is

$$\begin{aligned} D'_p(\mathrm{TRI}_{m,k}) &= \left(\sum_{i=1}^{m}(1-p^k)^{i-1}\right)' \sum_{j=1}^{k} p^{j-1} + \sum_{i=1}^{m}(1-p^k)^{i-1}\left(\sum_{j=1}^{k} p^{j-1}\right)' \\ &= \sum_{i=2}^{m}(i-1)(1-p^k)^{i-2}(-kp^{k-1})\sum_{j=1}^{k} p^{j-1} + \sum_{i=1}^{m}(1-p^k)^{i-1}\sum_{j=2}^{k}(j-1)p^{j-2} \\ &= (-kp^{k-1})\sum_{i=2}^{m}(i-1)(1-p^k)^{i-2}\sum_{j=1}^{k} p^{j-1} + \sum_{i=1}^{m}(1-p^k)^{i-1}\sum_{j=2}^{k}(j-1)p^{j-2}. \end{aligned}$$

with $D'_0(\mathrm{TRI}_{m,k}) = m > 0$ and $D'_1(\mathrm{TRI}_{m,k}) = -k^2 + k(k-1)/2 = -k(k+1)/2 < 0$. Since the derivative changes sign from positive to negative, this means that there is at least one interior maximum in the $[0,1]$ interval, but we have not yet shown that it is unique or at what point it occurs.

4.3 Majority

The majority function is defined for odd numbers $n = 2k-1$ where $k \in \{1, 2, \ldots\}$ so that

$$\mathrm{MAJ}_n(x) = \begin{cases} 1, x_1 + \cdots + x_n \geq k \\ 0, \text{otherwise.} \end{cases}$$

A more general version is the threshold function, that determines whether a weighted sum of its inputs exceeds a certain threshold. It is defined as 1 if $w_1x_1 + w_2x_2 + ... + w_nx_n \geq t$ and 0 otherwise where t is a real number called the threshold and $w_1, w_2, \ldots, w_n$ are real-numbered weights. [Hu65]

$$\mathrm{THR}_n^w(x_1, x_2, ..., x_n) = \begin{cases} 1, & w_1x_1 + w_2x_2 + ... + w_nx_n \geq t \\ 0, & \text{otherwise} \end{cases}$$

Setting $w_1 = \cdots = w_n = 1$ and $t = k = (n+1)/2$ we get the majority function, and the dictator function is a special case when $t = 1$ and all weights are 0 except for w_i when x_i is the dictator bit.

The majority function can be shown to be evasive, which means that $D(\mathrm{MAJ}_n) = n$. The method is to answer 0 and 1 in an alternating pattern until all bits have been queried, regardless of order. Furthermore, the majority function is symmetric, monotone and odd. In addition, the influence for a bit i is $I_i^p(f) = \binom{n-1}{k-1}p^{k-1}(1-p)^{k-1}$. This is because bit i is only pivotal when we have $k-1$ bits with value 0 and $k-1$ with value 1 among the other bits, so that if i changes value, the output is changed. Furthermore, there are $\binom{n-1}{k-1}$ ways to place $k-1$ bits on $n-1$ positions. This gives us a formula for the lower bound for majority. When calculating the level-p-complexity for the majority function, we use an algorithm querying the bits in order from left to right, and stopping if the output is known. There is no better algorithm, since the function is symmetric and all orders of asking the bits are equal.

4.3.1 Majority on three bits

Proposition 4.3 ($D_p(\mathrm{MAJ}_3)$)**.** The level-p-complexity of 3-bit majority is $D_p(\mathrm{MAJ}_3) = 2 + 2p - 2p^2$ with maximum $5/2 = 2.5$ attained at $p = 1/2$.

Proof. The three bits are independent and each has Bernoulli(p) distribution so that

$$x_i = \begin{cases} 0, & \text{with probability } 1-p \\ 1, & \text{with probability } p \end{cases}, i \in \{1,2,3\}.$$

The level-p-complexity is given by $p_1 + p_2 + p_3$ where p_i is probability that the ith bit queried by the algorithm is asked. Since the first two questions must be asked we have $p_1 + p_2 = 2$. Now p_3 is the probability that the third bit is asked which only happens if the first two bits are 01 or 10 so that $p_3 = p(1-p) + (1-p)p = 2p - 2p^2$. Then we get

$$D_p(\mathrm{MAJ}_3) = 2 + 2p - 2p^2.$$

The function $D_p(\mathrm{MAJ}_3)$ is symmetric around $p = 1/2$ as can be seen in Figure 4.4. We can find the maximum by setting the derivative to zero:

$$D_p'(\mathrm{MAJ}_3) = 2 - 4p = 0 \implies p = 1/2$$

and also $D_p''(\mathrm{MAJ}_3) = -4 < 0$. So the maximum is attained at $p = 1/2$ with $D_p(\mathrm{MAJ}_3) = 5/2 = 2.5$. □

The level-p-complexity and the lower bound in Theorem 2.3 can be seen in Figure 4.4.

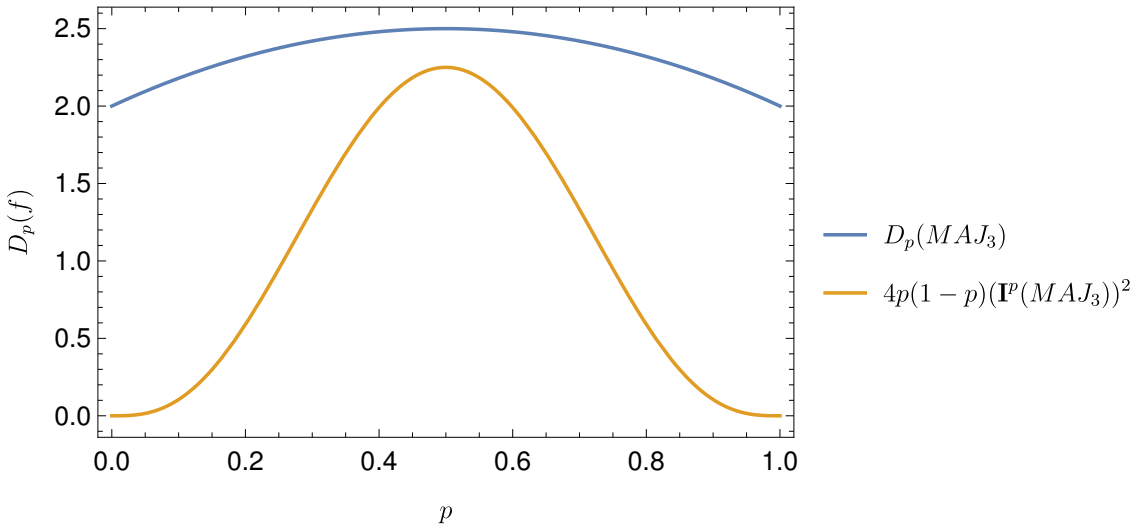


Figure 4.4: Level-p-complexity and lower bound for MAJ_3 using Theorem 2.3. The code for generating this figure is found in `maj2.nb`.

4.3.2 Majority on five bits

Proposition 4.4 ($D_p(\mathrm{MAJ}_5)$)**.** The level-p-complexity of 5-bit majority is

$$D_p(\mathrm{MAJ}_5) = 3 + 3p + 3p^2 - 12p^3 + 6p^4$$

with maximum 4.125 attained at $p = 1/2$.

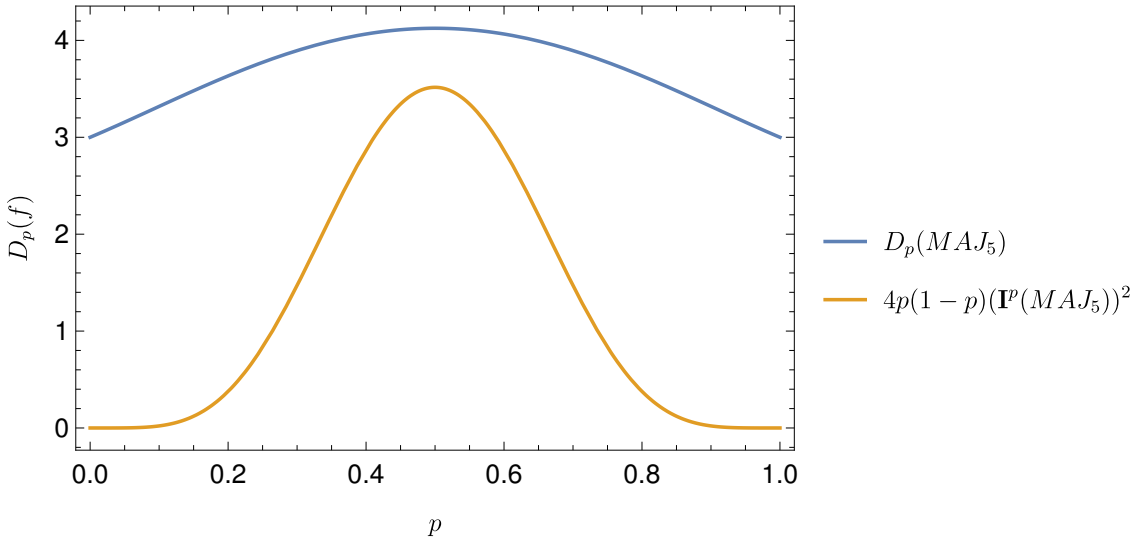


Figure 4.5: Level-p-complexity and lower bound for MAJ_5 using Theorem 2.3. The code for generating this figure is found in `maj2.nb`.

Proof. The level-p-complexity is given by $p_1 + \ldots + p_5$ where p_i is probability that the ith bit queried by the algorithm is asked. Since the first three questions must be asked we have $p_1 + p_2 + p_3 = 3$. Now p_4 is the probability that the fourth bit is asked which is the complementary probability to stopping to ask after 3 questions. This only happens if we get 3 consecutive 0's or 1's, so $p_4 = 1 - p^3 - (1-p)^3$. Lastly asking the fifth question only happens if we get exactly two 1's and two 0's in some order in the first 4 positions. Thus, $p_5 = \binom{4}{2}p^2(1-p)^2$, and consequently,

$$\begin{aligned} D_p(\mathrm{MAJ}_5) &= 3 + 1 - p^3 - (1-p)^3 + \binom{4}{2}p^2(1-p)^2 \\ &= 3 + 1 - p^3 - 1 + 3p - 3p^2 + p^3 + 6p^2 - 12p^3 + 6p^4 \\ &= 3 + 3p + 3p^2 - 12p^3 + 6p^4 \end{aligned}$$

or alternatively

$$D_p(\mathrm{MAJ}_5) = 3(1-p)^4 + 15p(1-p)^3 + 30p^2(1-p)^2 + 15p^3(1-p) + 3p^4.$$

This representation shows that $D_p(\mathrm{MAJ}_5)$ is symmetric around $p = 1/2$, as is seen in Figure 4.5.

We can find the maximum by setting the derivative to zero and solving for roots in Mathematica:

$$D'_p(\mathrm{MAJ}_5) = 3 + 6p - 36p^2 + 24p^3 = 0 \implies p_{\text{root}} \in \left\{ \frac{1}{2} - \frac{\sqrt{2}}{2}, \frac{1}{2}, \frac{1}{2} + \frac{\sqrt{2}}{2} \right\}$$

and also

$$D''_p(\mathrm{MAJ}_5) = 6 - 72p + 72p^2 = 6 - 72/2 + 72/4 = -12 < 0$$

for the root $p = 1/2$ which is in the interval $[0, 1]$. So the maximum is attained at $p = 1/2$ with value $D_p(\mathrm{MAJ}_5) = 33/8 = 4.125$. □

The level-p-complexity and the lower bound in Theorem 2.3 can be seen in Figure 4.5.

4.4 Iterated majority

Majority can be extended to iterated majority, where we combine the majority functions recursively in n levels as

$$\mathrm{MAJ}_k^{n+1}(\boldsymbol{x_1}, \ldots, \boldsymbol{x_k}) = \mathrm{MAJ}_k(\mathrm{MAJ}_k^n(\boldsymbol{x_1}), \ldots, \mathrm{MAJ}_k^n(\boldsymbol{x_k}))$$

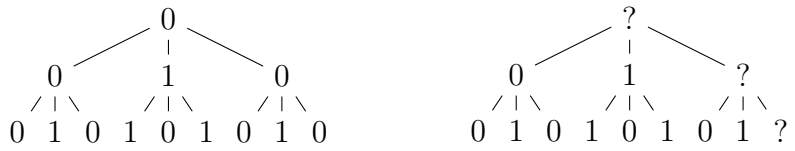
where MAJ_k is the majority function on k bits and MAJ_k^n is the iterated majority function on n levels taking k^n inputs. Here each $\boldsymbol{x_i}$ is a block of k^n input bits. In the base case we have $k^0 = 1$ bit in which case we just return the value: $\mathrm{MAJ}_k^0(x) = x$. Each level corresponds to an iteration of the majority function. Iterated majority is transitive but not symmetric, similarly to tribes, which makes the analysis of D_p more difficult than for regular majority. In general, it seems difficult to compute the level-p-complexity of iterated majority, but we can compute it for a special case and develop upper and lower bounds in the general case. We will thus calculate the level-p-complexity for MAJ_3^2. Lastly we will study bounds on the complexity for n levels.

4.4.1 Complexity of two-level three-bit majority

Let (i, j) represent bit j in block i, $x_{(i,j)}$ the value of this bit, and m_i the majority of block i. We have a Boolean function using 9 bits to determine the value, by first considering the majority in each of the subtrees and then the majority of the resulting three bits as shown in

$$\underbrace{\underbrace{x_{(1,1)}, x_{(1,2)}, x_{(1,3)}}_{m_1=\mathrm{MAJ}_3}, \underbrace{x_{(2,1)}, x_{(2,2)}, x_{(2,3)}}_{m_2=\mathrm{MAJ}_3}, \underbrace{x_{(3,1)}, x_{(3,2)}, x_{(3,3)}}_{m_3=\mathrm{MAJ}_3}}_{\mathrm{MAJ}_3(m_1,m_2,m_3)}$$

We illustrate an example by the decision tree in Figure 4.6a. On the lowest level we have $\mathrm{MAJ}_3(0, 1, 0) = 0$, then $\mathrm{MAJ}_3(1, 0, 1) = 1$, and lastly $\mathrm{MAJ}_3(0, 1, 0) = 0$. Then on the middle level we have again $\mathrm{MAJ}_3(0, 1, 0) = 0$ which is our final output.



(a) Example of iterated majority. **(b)** The unknown bit is pivotal.

Figure 4.6: Input with 3×3 bits illustrating an iterated majority example.

In the case seen in Figure 4.6a, an algorithm seeking to determine the value of the function by querying the bits left to right is forced to ask all the questions. Figure 4.6b shows that even when asked 8 questions we cannot be sure of the output: it could still be 1, therefore the algorithm has to ask the 9th question. In general, we can answer the queries in such a way that all questions must be asked. The method of answering queries is

- Always answer 1 and then 0 for the first two bits asked in a block. This forces the algorithm to query the third bit in the block.
- When we are asked for the third bit of a block for the first time, set it to 1.
- Next time we ask for a third bit in a block, set it to 0. Then the majority of this block is 0, and since the majority of the other block is 1, the end result is not yet known.

This forces all questions to be asked, and the function is thus evasive.

We move on to compute the level-p-complexity, seen in Theorem 4.1. To do this we consider the expected number of queries when the input has distribution π_p for some different types of algorithms, seen in Theorem 4.5. The first algorithm A_1 is the one that we always used so far, the one that asks the bits left to right, and skips to the next block if we already know the majority of the current block. Since the order in each block and between the blocks do not matter, many algorithms are equivalent to this one as long as they go through one block at a time.

However, this algorithm will be shown to be suboptimal relative to the last class of "smarter" algorithms. First, we ask 2 or 3 bits in the first block, yielding a value $m_1 = \mathrm{MAJ}_3(x_{(1,1)}, x_{(1,2)}, x_{(1,3)})$ for the majority in the first block. Then we ask $(2,1)$; if $x_{(2,1)} = m_1$, we continue asking the second block, otherwise we save the second block for later and continue with the third block. Due to this choice depending on $x_{(2,1)}$ there are two options for this algorithm, seen in Table 4.1. Since the function is transitive, we do not have to consider other orders than these 2 before or after asking $(2,1)$. Note that this is the order we ask the questions if we have to ask all the questions, but otherwise the algorithms stop when the result is known, for example we just ask $(2,3)$ if $x_{(2,1)} \neq x_{(2,2)}$.

Depending on m_1 and $x_{(2,1)}$ when $m_1 \neq x_{(2,1)}$ we see the 4 different ways an algorithm asks questions in Order 1 or Order 2 in Table 4.2. Algorithms that ask in Order 2 when $m_1 = x_{(2,1)}$ will be worse than asking in Order 1 since it is more likely that $m_1 = m_2$ than $m_1 = m_3$ given $m_1 = x_{(2,1)}$, which means it is better to ask block 2 before block 3.

Question-order	1	2	3	4	5	6	7	8	9
Order 1	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)	(3,1)	(3,2)	(3,3)
Order 2	(1,1)	(1,2)	(1,3)	(2,1)	(3,1)	(3,2)	(3,3)	(2,2)	(2,3)

Table 4.1: Two options of the order asked by the smart algorithms.

m_1	$x_{(2,1)}$	A_1	A_2	A_3	A_4
0	0	Order 1	Order 1	Order 1	Order 1
0	1	Order 1	Order 2	Order 1	Order 2
1	0	Order 1	Order 1	Order 2	Order 2
1	1	Order 1	Order 1	Order 1	Order 1

Table 4.2: The algorithms ask according to Order 1 or Order 2 depending on m_1 and $x_{(2,1)}$.

Now onto the expected number of queries for the different cases. They are described in Proposition 4.5 and illustrated in Figure 4.7.

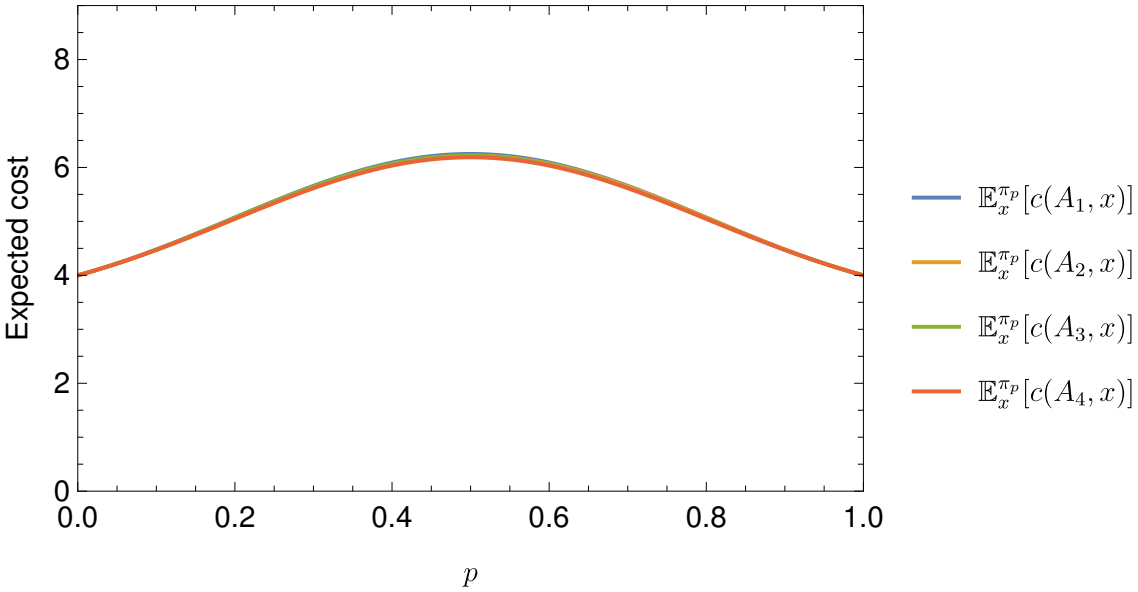


Figure 4.7: Expected costs of the four different algorithms. Since it is hard to differentiate between them their differences can be seen in Figure 4.8.

Proposition 4.5 (Expected cost for MAJ_3^2)**.** The expected costs for the four algorithms are

$$\begin{aligned}
\mathbb{E}_x^{\pi_p}[c(A_1,x)] &= 4+4p+8p^2+4p^3-56p^4+20p^5+68p^6-64p^7+16p^8,\\
\mathbb{E}_x^{\pi_p}[c(A_2,x)] &= 4+4p+7p^2+6p^3-54p^4+12p^5+75p^6-66p^7+16p^8,\\
\mathbb{E}_x^{\pi_p}[c(A_3,x)] &= 4+4p+8p^2+4p^3-59p^4+28p^5+61p^6-62p^7+16p^8,\\
\mathbb{E}_x^{\pi_p}[c(A_4,x)] &= 4+4p+7p^2+6p^3-57p^4+20p^5+68p^6-64p^7+16p^8.
\end{aligned}$$

The maxima of the expected costs are

$$\begin{aligned}
\max_p \mathbb{E}_x^{\pi_p}[c(A_1, x)] &= 6.25 \text{ at } p = 0.5, \\
\max_p \mathbb{E}_x^{\pi_p}[c(A_2, x)] &\approx 6.2189 \text{ at } p \approx 0.5029, \\
\max_p \mathbb{E}_x^{\pi_p}[c(A_3, x)] &\approx 6.2189 \text{ at } p \approx 0.4971, \\
\max_p \mathbb{E}_x^{\pi_p}[c(A_4, x)] &= 6.1875 \text{ at } p = 0.5.
\end{aligned}$$

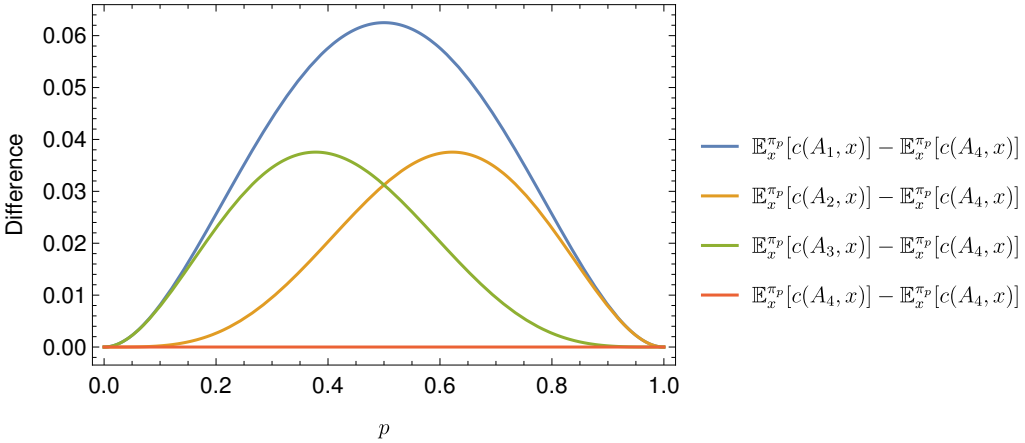


Figure 4.8: Difference between the expected costs of A_1, A_2, A_3, A_4 and A_4.

Proof. Let $p_{(i,j)}$ the probability that bit (i, j) is asked. The easiest case is for the algorithm that asks the bits in order. We ask $(1, 1), (1, 2), (2, 1), (2, 2)$ with probability 1 and $(1, 3), (2, 3)$ with p_{diff}, which is the probability that the first two bits in this block are alternating. We know from results of the MAJ_3 function that $p_{\text{diff}} = 2p(1 - p)$. The third block is only asked if $m_1 \neq m_2$, with probability p_{alt}. This probability is harder to compute, so first we compute $P_1 = \mathbb{P}(\text{MAJ}_3(x) = 1)$ when $x \sim \pi_p$. It is the probability of getting two 1's and a 0, or three 1's, so $P_1 = 3p^2(1 - p) + p^3$. Then we get

$$p_{\text{alt}} = 2\mathbb{P}(\text{MAJ}_3(x) = 1) \cdot \mathbb{P}(\text{MAJ}_3(x) = 0) = 2P_1 \cdot (1 - P_1).$$

We get

$$\mathbb{E}_x^{\pi_p}[c(A_1, x)] = (2+p_{\text{alt}})(2+p_{\text{diff}}) = 4+4p+8p^2+4p^3-56p^4+20p^5+68p^6-64p^7+16p^8$$

where the expansion of the expression is done by Mathematica. The function $\mathbb{E}_x^{\pi_p}[c(A_1, x)]$ has minimum value of 4 at $p = 0$ and $p = 1$ and maximum value of $25/4 = 6.25$ at $p = 1/2$.

We now calculate the expected cost for algorithm A_2. The expected cost of the algorithm before switching blocks is always the same since we first ask bits $(1, 1), (1, 2)$ and $(1, 3)$ if $x_{(1,1)} \neq x_{(1,2)}$, and then ask $(2, 1)$. The expected cost of

this first part is $(2 + p_{\text{diff}}) + 1$, and then we have four subalgorithms with different expected costs.

First, consider the case when $m_1 = x_{2,1} = 0$, which has probability $(1-P_1)(1-p)$. We ask the questions in Order 1 seen in Table 4.1. First, we ask $(2, 2)$, and we only ask $(2, 3)$ if $x_{2,2} = 1$ since the bits are alternating in this case. If $m_2 = 0$ we are done asking questions, else we continue in the third block. The probability of $m_2 = 1$ given $x_{2,1} = 0$ is p^2 since we need $x_{2,2} = x_{2,3} = 1$. The expected cost for the subalgorithm A_{00} when $m_1 = x_{2,1} = 0$ is thus

$$\mathbb{E}_x^{\pi_p}[c(A_{00}, x)] = (1 + p) + p^2(2 + p_{\text{diff}}).$$

Next, consider the case when $m_1 = 0$ but $x_{2,1} = 1$, which has probability $(1 - P_1)p$. From the specification in Table 4.2, we now ask the questions in Order 2 seen in Table 4.1. First, we ask $(3, 1), (3, 2)$ and $(3, 3)$ if needed. If $m_3 = 1$ we have to go back and ask the bits in block 2, which happens with probability P_1. Then we ask $(2, 2)$ and $(2, 3)$ only if $x_{2,2} = 0$ since the bits are alternating in this case. The expected cost for the subalgorithm A_{01} when $m_1 = 0$ and $x_{2,1} = 1$ is thus

$$\mathbb{E}_x^{\pi_p}[c(A_{01}, x)] = (2 + p_{\text{diff}}) + P_1(1 + (1 - p)).$$

Now we consider the case when $m_1 = 1$ and $x_{2,1} = 0$, which has probability $P_1(1 - p)$. From the specification in Table 4.2, we now ask the questions in Order 1 seen in Table 4.1. First, we ask $(2, 2)$, and we only ask $(2, 3)$ if $x_{2,2} = 1$ as in A_{00}. If $m_2 = 1$ we are done asking questions, else we continue in the third block. The probability of $m_2 = 0$ given $x_{2,1} = 0$ is $1-p^2$. The expected cost for the subalgorithm A_{10} when $m_1 = 1$ and $x_{2,1} = 0$ is thus

$$\mathbb{E}_x^{\pi_p}[c(A_{10}, x)] = (1 + p) + (1 - p^2)(2 + p_{\text{diff}}).$$

Lastly we have the case when $m_1 = x_{2,1} = 1$, which has probability $P_1 p$. We ask the questions in Order 1 seen in Table 4.1. First, we ask $(2, 2)$, and we only ask $(2, 3)$ if $x_{2,2} = 0$. If $m_2 = 1$ we are done asking questions, else we continue in the third block. The probability of $m_2 = 0$ given $x_{2,1} = 1$ is $(1 - p)^2$. The expected cost for the subalgorithm A_{11} when $m_1 = x_{2,1} = 1$ is thus

$$\mathbb{E}_x^{\pi_p}[c(A_{11}, x)] = (1 + (1 - p)) + (1 - p)^2(2 + p_{\text{diff}}).$$

Combining these four cases to the expected cost of A_2 yields

$$\begin{aligned}\mathbb{E}_x^{\pi_p}[c(A_2, x)] = (2 + p_{\text{diff}}) + 1 &+ (1 - P_1)(1 - p) \cdot \mathbb{E}_x^{\pi_p}[c(A_{00}, x)] \\ &+ (1 - P_1)p \cdot \mathbb{E}_x^{\pi_p}[c(A_{01}, x)] \\ &+ P_1(1 - p) \cdot \mathbb{E}_x^{\pi_p}[c(A_{10}, x)] \\ &+ P_1 p \cdot \mathbb{E}_x^{\pi_p}[c(A_{11}, x)]\end{aligned}$$

This can be expanded using Mathematica[1]:

$$\mathbb{E}_x^{\pi_p}[c(A_2, x)] = 4 + 4p + 7p^2 + 6p^3 - 54p^4 + 12p^5 + 75p^6 - 66p^7 + 16p^8.$$

[1] See file `iterated_maj.nb`.

The calculation of expected cost for A_3 is similar, but we change A_{01} and A_{10} to A'_{01} and A'_{10}. For A'_{01} we see in the specification in Table 4.2 that we now ask the questions in Order 1 seen in Table 4.1. First, we ask $(2,2)$, and we only ask $(2,3)$ if $x_{2,2} = 0$. If $m_2 = 0$ we are done asking questions, else we continue in the third block. The probability of $m_2 = 1$ given $x_{2,1} = 1$ is $(1-(1-p)^2)$. The expected cost for the subalgorithm A'_{01} is thus

$$\mathbb{E}_x^{\pi_p}[c(A'_{01}, x)] = (1 + (1-p)) + (1-(1-p)^2)(2 + p_{\text{diff}}).$$

For A'_{10} we see in the specification in Table 4.2 that we now ask the questions in Order 2 seen in Table 4.1. First, we ask $(3,1), (3,2)$ and $(3,3)$ if needed. If $m_3 = 0$ we have to go back and ask the bits in block 2, which happens with probability $1 - P_1$. Then we ask $(2,2)$ and $(2,3)$ only if $x_{2,2} = 1$ since the bits are alternating in this case. The expected cost for the subalgorithm A'_{10} is thus

$$\mathbb{E}_x^{\pi_p}[c(A'_{10}, x)] = (2 + p_{\text{diff}}) + (1 - P_1)(1 + p).$$

We get

$$\begin{aligned}\mathbb{E}_x^{\pi_p}[c(A_3, x)] = (2 + p_{\text{diff}}) + 1 &+ (1 - P_1)(1-p) \cdot \mathbb{E}_x^{\pi_p}[c(A_{00}, x)] \\ &+ (1 - P_1)p \cdot \mathbb{E}_x^{\pi_p}[c(A'_{01}, x)] \\ &+ P_1(1-p) \cdot \mathbb{E}_x^{\pi_p}[c(A'_{10}, x)] \\ &+ P_1 p \cdot \mathbb{E}_x^{\pi_p}[c(A_{11}, x)]\end{aligned}$$

This can be expanded using Mathematica:

$$\mathbb{E}_x^{\pi_p}[c(A_3, x)] = 4 + 4p + 8p^2 + 4p^3 - 59p^4 + 28p^5 + 61p^6 - 62p^7 + 16p^8.$$

Lastly we have the expected cost of A_4 which is similar to that of A_2 and A_3 and uses subalgorithms, A_{00}, A'_{01}, A_{10} and A_{11}. Thus the cost is

$$\begin{aligned}\mathbb{E}_x^{\pi_p}[c(A_4, x)] = (2 + p_{\text{diff}}) + 1 &+ (1 - P_1)(1-p) \cdot \mathbb{E}_x^{\pi_p}[c(A_{00}, x)] \\ &+ (1 - P_1)p \cdot \mathbb{E}_x^{\pi_p}[c(A'_{01}, x)] \\ &+ P_1(1-p) \cdot \mathbb{E}_x^{\pi_p}[c(A_{10}, x)] \\ &+ P_1 p \cdot \mathbb{E}_x^{\pi_p}[c(A_{11}, x)]\end{aligned}$$

This can be expanded using Mathematica:

$$\mathbb{E}_x^{\pi_p}[c(A_4, x)] = 4 + 4p + 7p^2 + 6p^3 - 57p^4 + 20p^5 + 68p^6 - 64p^7 + 16p^8.$$

□

Conjecture 4.1 ($D_p(\text{MAJ}_3^2)$)**.** For $p \in [0,1]$ algorithm A_4 has the lowest expected cost among the algorithms. Thus, the level-p-complexity for iterated 3-bit majority on two levels is

$$D_p(\text{MAJ}_3^2) = 4 + 4p + 7p^2 + 6p^3 - 57p^4 + 20p^5 + 68p^6 - 64p^7 + 16p^8.$$

The maximum of $D_p(\text{MAJ}_3^2)$ is $99/16 = 6.1875$ at $p = 1/2$.

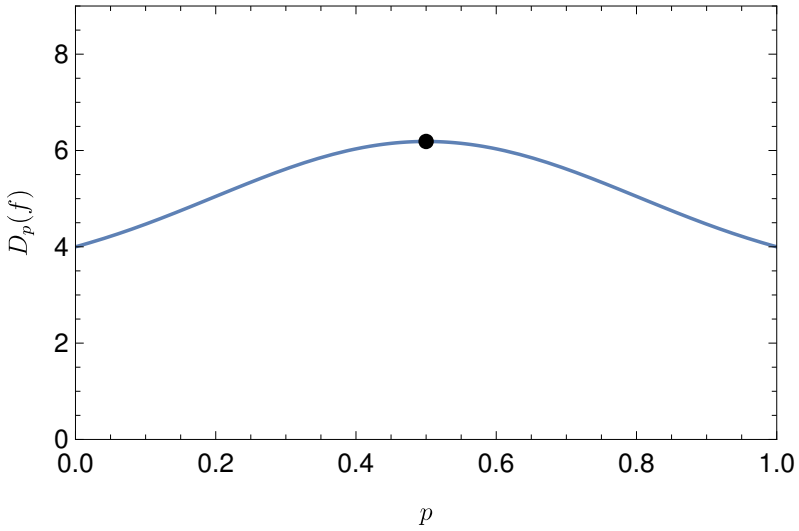


Figure 4.9: Level-p-complexity of iterated majority.

The method of generating algorithms as proposed in Chapter 3 proved unfeasible for 9 bits. We can not prove this as there might be other algorithms than $A_1, \ldots, A_4$ that are better. But as we have not found any other better algorithms we provide strong evidence for Conjecture 4.1. The proposed formula is seen in Figure 4.9.

Proof sketch of Conjecture 4.1. We use the representation $\sum_{i=0}^{n} b_i \binom{n}{i} p^i (1-p)^{n-i}$, which allows us to see the number of questions in total for all the different cases where we have i 1's and $n-i$ 0's. In Table 4.3 we see the coefficients for the expected costs of the four algorithms.

	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9
$\mathbb{E}_x^{\pi_p}[c(A_1,x)]$	$4/1$	$40/9$	$184/36$	$508/84$	$864/126$	$864/126$	$508/84$	$184/36$	$40/9$	$4/1$
$\mathbb{E}_x^{\pi_p}[c(A_2,x)]$	$4/1$	$40/9$	$183/36$	$503/84$	$857/126$	$861/126$	$508/84$	$184/36$	$40/9$	$4/1$
$\mathbb{E}_x^{\pi_p}[c(A_3,x)]$	$4/1$	$40/9$	$184/36$	$508/84$	$861/126$	$857/126$	$503/84$	$183/36$	$40/9$	$4/1$
$\mathbb{E}_x^{\pi_p}[c(A_4,x)]$	$4/1$	$40/9$	$183/36$	$503/84$	$854/126$	$854/126$	$503/84$	$183/36$	$40/9$	$4/1$

Table 4.3: The coefficients for the expected costs of algorithms A_1, A_2, A_3 and A_4 when using the symmetric base representation $\sum_{i=0}^{n} b_i \binom{n}{i} p^i (1-p)^{n-i}$.

We observe that the costs of A_1 and A_4 are symmetric around $p = 1/2$ since $b_i = b_{n-i}$ for $i \in \{1, \ldots, n\}$. Now, the expected costs of A_2 and A_3 are not symmetric around $p = 1/2$, but they are similar to each other, just flipping the coefficients. From this representation we immediately see that A_4 is lower than A_1, A_2 and A_3 for $p \in [0, 1]$ since all $p^i(1-p)^j$ terms are ≥ 0 in this interval and the coefficients in A_4 are less than for the others. We can also see in Figure 4.8 that the expected cost of A_4 is the lowest of the four algorithms. So among these algorithms, A_4 is the best, and if there are no other better algorithms we have $D_p(f) = \mathbb{E}_x^{\pi_p}[c(A_4, x)]$ since D_p is defined to be the minimum over algorithms. □

It is interesting to note that if we subtract the costs of the algorithms A_1 and A_4 we can interpret the coefficients as

- there is the same amount of questions asked on average for A_1 and A_4 for the cases when we have nine 0's or nine 1's.
- there is the same amount of questions asked on average for A_1 and A_4 for the cases when we have one 1 and eight 0's or one 0 and eight 1's.
- there are 1/36 more questions asked on average for A_1 than A_4 for the cases when we have two 1's and seven 0's or two 0's and seven 1's.
- there are 5/84 more questions asked on average for A_1 than A_4 for the cases when we have three 1's and six 0's or three 0's and six 1's.
- there are 10/126 more questions asked on average for A_1 than A_4 for the cases when we have four 1's and five 0's or four 0's and five 1's.

4.4.2 General complexity bounds

Randomized complexity of iterated three bit majority has been studied in [Lan+06], [Leo13] and [Mag+16], as they improve the lower bound for $R(\mathrm{MAJ}_3^n)$ in terms of the level n.

Proposition 4.6 (Bounds for $D_p(\mathrm{MAJ}_3^n)$)**.** A lower bound for the level-p-complexity for n levels of iterated majority on three bits is 2^n, and an upper bound is 2.5^n so $2^n \leq D_p(\mathrm{MAJ}_3^n) \leq 2.5^n$.

Proof. The lower bound is 2^n since the least amount of questions asked for MAJ_3^n is 2^n. This can be seen if we consider all bits to be 0, in which case all blocks can be determined in 2 questions asked, and we only have to ask 2^{n-1} blocks to know the result.We prove the upper bound by induction. The base case is 1 level which is regular majority, and we know that $D_p(\mathrm{MAJ}_3) = 2.5$ so that $D_p(\mathrm{MAJ}_3^1) \leq 2.5$. Assuming $D_p(\mathrm{MAJ}_3^k) \leq 2.5^k$ we find that an algorithm for querying bits for iterated majority with $k+1$ can be combined as the algorithm for 1 level and for k levels, since we consider each of the subtrees separately. Then we get $D_p(\mathrm{MAJ}_3^{k+1}) \leq 2.5^{k+1}$. □

This means that using the algorithm for level 1 we can construct an algorithm for level n inductively. However, there might be better algorithms, so the upper bound for D_p might not be tight. From the base case we already know that the lower bound is not tight. Furthermore, we know $D_p(\mathrm{MAJ}_3^2) \leq 6.1875 < 6.25 = 2.5^2$ since $\max_{p\in[0,1]} D_p(\mathrm{MAJ}_3^2) = 6.1875$, so the upper bound is not tight either. This means that better bounds could be achieved.

We use the multiplicative form of Fekete's Lemma. [Fek23]

Lemma 4.1 (Fekete's Lemma [Fek23])**.** Let the sequence $\{c_n \,|\, n = 1, 2, \ldots\}$ of positive numbers be submultiplicative in the sense that $c_{n+m} \leq c_n c_m$ for $n, m = 1, 2, \ldots$ (so that the sequence $\log(c_n)$ is subadditive). Then

$$\lim_{n\to\infty} c_n^{1/n} = \inf_n c_n^{1/n}$$

Since $D_p(\mathrm{MAJ}_3^n)$ is submultiplicative we get that there exists a number

$$l = \lim_{n\to\infty} D_p(\mathrm{MAJ}_3^n)^{1/n} = \inf_n D_p(\mathrm{MAJ}_3^n)^{1/n}$$

and we know $2 \leq l \leq 2.5$.

You can't connect the dots looking forward; you can only connect them looking backwards.

Steve Jobs

5 Boolean functions on graphs

This chapter concerns graph properties, which can be encoded as Boolean functions. Consider a graph G with nodes $N = \{1, \ldots, n\}$ and edges $E \subseteq \{\{i, j\} \text{ for } i, j \in \mathcal{N}\}$. We do not allow self-loops and all the edges are undirected. Let $v_{i,j} = 1$ if the edge between i and j exists and 0 otherwise. Given the all the input bits $v_{i,j}$, the Boolean function f is 1 if G fulfills the given graph property and 0 otherwise.

5.1 Previous work

There has been a lot of research on the deterministic and randomized complexity for graph properties and more generally for non-constant transitive monotone Boolean functions. For example, [RV75] proves that for any non-constant transitive monotone Boolean function f of n bits, $D(f) = n$ if n is a prime power. This theorem was derived in order to then prove (in the same paper) the Aanderaa-Rosenberg Conjecture which states that the deterministic complexity for any non-constant monotone graph property on n nodes is $\Omega(n^2)$ [Ros73]. There is also a variant of the Aanderaa-Rosenberg Conjecture for randomized complexity, called the Aanderaa-Karp-Rosenberg Conjecture which has not yet been proved. However the lower bound of $n^{4/3}$ was proved in [Haj92].

5.2 Connectivity

We now consider the graph property of connectivity, and first show that it is evasive for any connected graph, and then compute the level-p-complexity for some specific graphs. More details about the calculations for the graphs can be found in the Mathematica notebook `graph.nb`. Originating from any connected graph $G = (N, E)$ we get a subgraph with the same set of nodes N but a subset of the edges, namely the ones that satisfy $v_{i,j} = 1$. This subgraph is the input to the Boolean function f that determines if the subgraph is connected or not.

5.2.1 Evasiveness

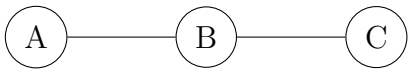
Proposition 5.1 (Evasiveness of graph connectivity [LY02])**.** For a connected graph G with n edges we have $D(f) = n$, where f represents connectivity of G.

Proof. We prove the proposition by showing that f is evasive. We let an adversary answer 0 unless that answer would imply the graph was disconnected, in which he or

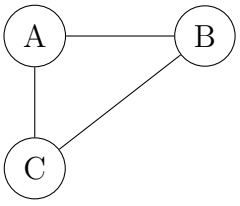
she answers 1. We have to show that it is not possible for f to be determined before we have queried all the edges. Assume the contrary, so that the answer is known without having queried $\{i, j\}$. For this to hold, this graph has to be connected already, not including the edge $\{i, j\}$, which means that there must be a path that connects nodes i and j that does not contain edge $\{i, j\}$. Of the edges on this path from i to j, suppose the last edge queried is $\{u, v\}$. But since $\{i, j\}$ is not yet queried, answering 0 for $\{u, v\}$ does not imply that the graph is disconnected so setting $\{u, v\}$ to 1 would go against the strategy described above. The adversary should instead answer 0 for $\{u, v\}$ and then 1 for $\{i, j\}$ as the last question. Thus, by using this strategy, the answer cannot be known until all the edges are queried. □

5.2.2 Small graphs on three or four nodes

Now we move over to calculating the level-p-complexity for different examples of graphs. First we consider a simple example of a graph with three nodes. Either we have two edges or three edges, seen in Figure 5.1. Let f_{32} be the Boolean function determining if the graph with 3 nodes and 2 edges (seen in Figure 5.1a) is connected. Next, f_{33} determines if the graph with 3 nodes and 3 edges (seen in Figure 5.1b) is connected or not.



(a) A path between the 3 nodes with $\deg(A) = \deg(C) = 1$ and $\deg(B) = 2$, corresponding to Boolean function f_{32}.



(b) A fully connected graph with 3 nodes where all nodes have degree 2, corresponding to Boolean function f_{33}.

Figure 5.1: The possible graphs with 3 nodes.

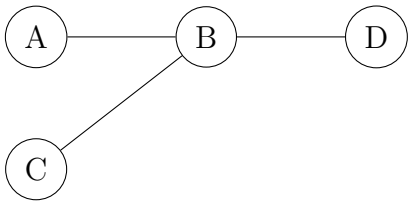
The next cases are graphs with four nodes and three edges. There are two possibilities, seen in Figure 5.2a and Figure 5.2b, with two different Boolean functions: $f_{43\text{class1}}$ and $f_{43\text{class2}}$.

Proposition 5.2 ($D_p(f)$ for small graphs)**.** The level-p-complexities of the Boolean functions for the graphs seen in Figure 5.1 and Figure 5.2 are

$$D_p(f_{32}) = 1 + p, \qquad D_p(f_{33}) = 2 + 2p - 2p^2,$$
$$D_p(f_{43\text{class1}}) = 1 + p + p^2, \qquad D_p(f_{43\text{class2}}) = 1 + p + p^2.$$

Proof. For f_{32}, both edges have to exist for the graph to be connected which corresponds to the ALL_2^1 function so $D_p(f_{32}) = 1 + p$ with maximum 2 at $p = 1$. For f_{33} we need the majority of the edges to be there, so it is the same as MAJ_3 so $D_p(f_{33}) = 2 + 2p(1 - p)$ with maximum $D_p(f_{33}) = 2.5$ at $p = 1/2$. Now, consider $f_{43\text{class1}}$ and $f_{43\text{class2}}$. Since all edges are needed for the graphs to be connected we

(a) A path between the 4 nodes with $\deg(A) = \deg(D) = 1$ and $\deg(B) = \deg(C) = 2$, corresponding to Boolean function $f_{43\text{class}1}$.

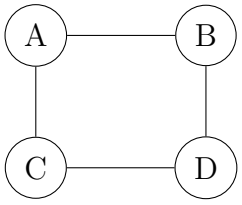


(b) A graph with 4 nodes and 3 edges with $\deg(B) = 3$ and $\deg(A) = \deg(C) = \deg(D) = 1$, corresponding to Boolean function $f_{43\text{class}2}$.

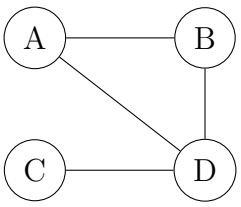
Figure 5.2: The graphs with 4 nodes and 3 edges.

have the ALL_3^1 function for both cases so $D_p(f_{43\text{class}1}) = D_p(f_{43\text{class}2}) = 1 + p + p^2$ with maximum 3 at $p = 1$. □

Next, we consider the possible graphs with four nodes and four edges. There are two important differences regarding degree of the nodes, which splits up the graph into two isomorphism classes. In the first version, shown in Figure 5.3a all nodes have degree 2, and the graph is symmetric. In the other case, seen in Figure 5.3b two nodes have degree 2, one degree 3 and one degree 1. The corresponding Boolean functions are $f_{44\text{class}1}$ and $f_{44\text{class}2}$ respectively.



(a) The graph where all nodes have degree 2, corresponding to Boolean function $f_{44\text{class}1}$.



(b) The graph with $\deg(D) = 3$ and $\deg(C) = 1$, corresponding to Boolean function $f_{44\text{class}2}$.

Figure 5.3: The graphs with 4 nodes and 4 edges.

Proposition 5.3 ($D_p(f_{44})$)**.** The level-p-complexities of the Boolean functions for the graphs seen in Figure 5.3 are

$$D_p(f_{44\text{class}1}) = 2 + 2p + 2p^2 - 3p^3,$$
$$D_p(f_{44\text{class}2}) = 1 + 2p + 2p^2 - 2p^3,$$

with maxima ≈ 3.36 at $p \approx 0.74$ for $D_p(f_{44\text{class}1})$ and 3 at $p = 1$ for $D_p(f_{44\text{class}2})$.

Proof. For class 1 there is no fundamental difference between the edges, and we know that the graph is connected if there are three 1's or more, and that it is disconnected if we have two 0's or more. Thus, $D_0(f_{44\text{class}1}) = 2$ and $D_1(f_{44\text{class}1}) = 3$. We must ask the first two queries, then we ask the third only if the previous ones were not both zero. Then we only ask the fourth if we have two 1's and one 0,

since this is the only case where we cannot yet conclude if the graph is connected or not. We get

$$D_p(f_{44\text{class1}}) = 2 + (1 - (1-p)^2) + 3p^2(1-p) = 2 + 2p + 2p^2 - 3p^3.$$

From Mathematica, we find the maximum ≈ 3.36 at $p \approx 0.74$ numerically.

For class 2 it does matter in which order we ask the edges; it is better to start with the one connected to the node with degree 1 (edge (C, D) in Figure 5.3b), because if there is no edge to this node we immediately conclude that the subgraph is disconnected. Given that the edge exists, the order of asking the other edges does not matter (at least 2 of the edges $(A, B), (A, D)$ or (B, D) in Figure 5.3b are needed for the graph to be connected). This gives

$$D_p(f_{44\text{class2}}) = 1 + p(2 + 2p(1-p)) = 1 + 2p + 2p^2 - 2p^3,$$

with maximum 3 at $p = 1$. □

5.2.3 Graphs with four nodes and five edges

Now we consider the graph with 4 nodes and 5 edges, seen in Figure 5.4, with Boolean function f_{45}. For 5 edges all the six possibilities of graphs are isomorphic. They consist of one edge where both nodes have degree 3 (in Figure 5.4 this edge is x_5), and the other four edges that all have one side with degree 2 and the other with degree 3. Before computing $D_p(f_{45})$ we start with computing the influences

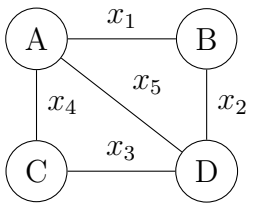


Figure 5.4: The graph with 4 nodes and 5 edges, corresponding to f_{45}.

for each of the edges.

Proposition 5.4 ($I_i^p(f_{45})$)**.** The influences for the edges in the graph seen in Figure 5.4 are

$$I_i^p(f_{45}) = 5p^2(1-p)^2 + p^3(1-p), i \in \{1, 2, 3, 4\}$$
$$I_5^p(f_{45}) = 4p^2(1-p)^2.$$

Proof. The influence for edge x_1 will be the same as for x_2, x_3 and x_4, so it is enough to calculate one of them. The edge x_1 is pivotal when it is not yet known if the graph is connected given the other edges (x_2, x_3, x_4, x_5). There are six cases in which the connectivity depends on x_1: when (x_2, x_3, x_4, x_5) is $(1, 1, 0, 0)$, $(1, 0, 1, 0)$, $(0, 1, 1, 0)$, $(0, 1, 0, 1)$, $(0, 0, 1, 1)$ or $(0, 1, 1, 1)$. Thus, we get for $i \in \{1, 2, 3, 4\}$:

$$I_i^p(f_{45}) = 5p^2(1-p)^2 + p^3(1-p).$$

The influence for edge x_5 is calculated similarly. There are four cases when it is not yet known if the graph is connected given the other edges: when (x_1, x_2, x_3, x_4) is $(1,0,0,1)$, $(1,0,1,0)$, $(0,1,0,1)$, or $(0,1,1,0)$. Thus, we get

$$I_5^p(f_{45}) = 4p^2(1-p)^2$$

□

We observe that $I_5^p(f_{45})$ is smaller than $I_1^p(f_{45})$ for $p \in [0,1]$. This is an indication that the algorithm should start with x_1 rather than x_5, which is stated in Proposition 5.5. The influences also give us the lower bound

$$4p(1-p)(\mathbf{I}^p(f))^2 = 2304p^5 - 10752p^6 + 20032p^7 - 18624p^8 + 8640p^9 - 1600p^{10}$$

seen in Figure 5.7.

(a) Reduced graph for $x_1 = 1$. **(b)** Reduced graph for $x_5 = 1$.

Figure 5.5: Reduced graphs for the graph in Figure 5.4.

Proposition 5.5 ($D_p(f_{45})$)**.** The algorithm A_1 starting with querying x_1 has lower expected cost for $p \in [0,1]$ than the algorithm A_5 that starts with querying x_5. Thus, the level-p-complexity of connectivity on the graph with 4 nodes and 5 edges is the expected cost of A_1 and has formula

$$D_p(f_{45}) = 2 + 3p + 4p^2 - 10p^3 + 4p^4$$

which has maximum at ≈ 3.61 at $p \approx 0.64$

Proof. We first compute the cost of an algorithm A_1 starting with asking x_1 (which is the same as starting with x_2, x_3 or x_4). If $x_1 = 0$, the Boolean function is reduced to $f_{44\text{class2}}$, since setting $x_1 = 0$ in Figure 5.4 gives the graph seen in Figure 5.3b. If $x_1 = 1$, the Boolean function is reduced to the connectivity of the graph seen in Figure 5.5a. This graph is similar to the graph in Figure 5.1b and the Boolean function is $\text{MAJ}_3(x_3, x_4, \text{ANY}_2^1(x_2, x_5))$. The algorithm would then ask x_3, x_4 in order as in regular majority, stop if done, otherwise ask x_2 and if needed x_5. The expected cost of A_1 is then

$$\begin{aligned}\mathbb{E}_x^{\pi_p}[c(A_1, x)] &= 1 + (1-p)(1 + 2p + 2p^2 - 2p^3) + p(2 + 2p(1-p)(1 + (1-p))) \\ &= 2 + 3p + 4p^2 - 10p^3 + 4p^4,\end{aligned}$$

and can be seen in Figure 5.7.

Now, we compute the cost of algorithm A_5 that starts with asking x_5. If $x_5 = 0$ then the Boolean function is reduced to $f_{44\text{class2}}$ since since setting $x_5 = 0$ in Figure 5.4 gives the graph seen in Figure 5.3a. If $x_5 = 1$ the reduced function is more complicated, see the corresponding graph in Figure 5.5b. This is similar to the case in 5.1a but with four edges instead of two. The Boolean function is thus $\text{ALL}_2^1(\text{ANY}_2^1(x_1, x_2), \text{ANY}_2^1(x_3, x_4))$. Due to symmetry it does not matter if we ask x_1, x_2 or x_3, x_4, and also the order between x_1 and x_2 and x_3 and x_4 does not matter. WLOG we ask the bits in order, this algorithm can be seen in Figure 5.6, and the

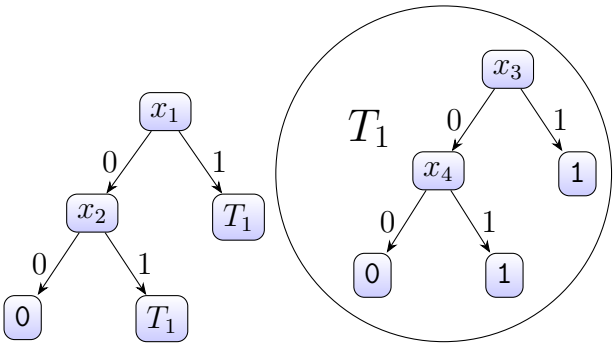


Figure 5.6: One of the possible algorithms for Figure 5.5b after $x_5 = 1$.

expected cost of it is

$$1 + p(1 + (1 - p)) + (1 - p)(1 + p(1 + (1 - p))) = 2 + 3p - 4p^2 + p^3$$

This expression comes from splitting up in two cases, first we ask one question, and then we calculate the expected cost of the reduced function when we get a 0 with probability $1 - p$ or 1 with probability p. These subexpressions are calculated similarly and the expected cost for T_1 is $1 + (1 - p)$ since we first ask one question and then ask the next with probability $1 - p$. Combining these results the expected cost of A_5 is

$$\begin{aligned}\mathbb{E}_x^{\pi_p}[c(A_5, x)] &= 1 + (1 - p)(2 + 2p + 2p^2 - 3p^3) + p(2 + 3p - 4p^2 + p^3) \\ &= 3 + 2p + 3p^2 - 9p^3 + 4p^4.\end{aligned}$$

As seen in Figure 5.7 the expected cost of A_1 is lower than that of A_5 in the interval and the intersection point between them is at $p = 1$. We can also see this algebraically by using the $p^i(1 - p)^j$ representation

$$\mathbb{E}_x^{\pi_p}[c(A_5, x)] - \mathbb{E}_x^{\pi_p}[c(A_1, x)] = (1 - p)^4 + 3(1 - p)^3 p + 2(1 - p)^2 p^2$$

which is always ≥ 0 for $p \in (0, 1)$, since all $p^i(1 - p)^j$ terms are positive in the interval. This means that the level-p-complexity is the expected cost of A_1 in the whole interval so that

$$D_p(f_{45}) = \mathbb{E}_x^{\pi_p}[c(A_1, x)] = 2 + 3p + 4p^2 - 10p^3 + 4p^4.$$

With Mathematica, we find the maximum ≈ 3.61 at $p \approx 0.64$ numerically. □

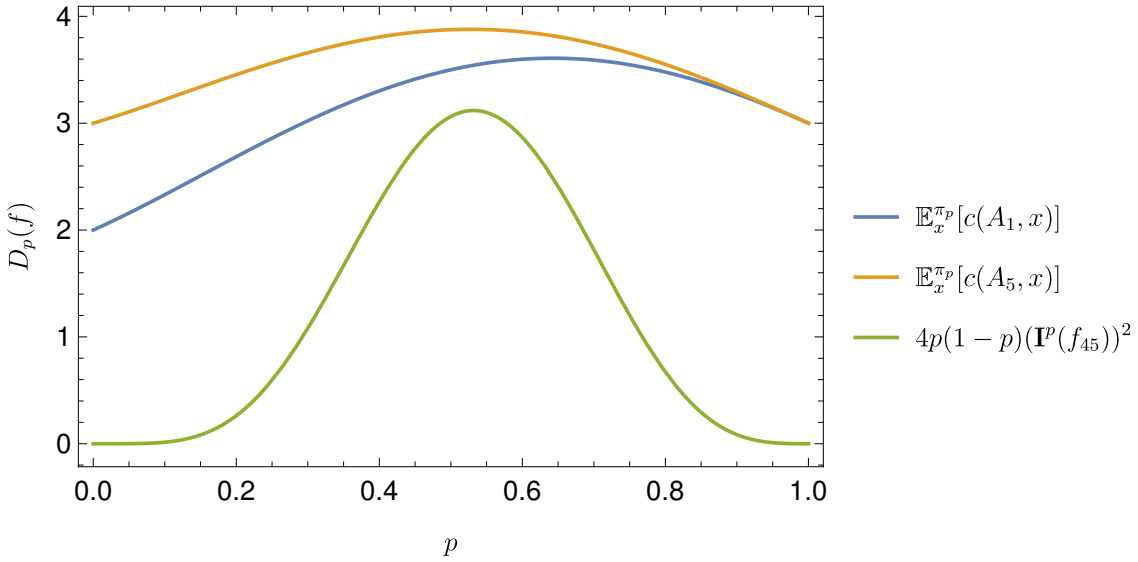
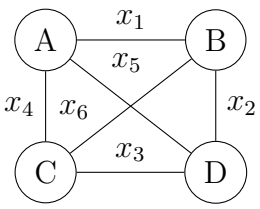


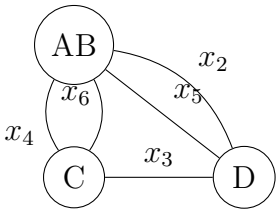
Figure 5.7: The expected costs for algorithms A_1 and A_5 for f_{45}.

5.2.4 Graphs with four nodes and six edges

Now we consider the case with 4 nodes and 6 edges seen in Figure 5.8a, which is the complete graph and corresponds to Boolean function f_{46}.



(a) Graph with 4 nodes and 6 edges, corresponding to f_{46}.



(b) Reduced graph for the graph in Figure 5.8a when $x_1 = 0$.

Figure 5.8: The complete graph on 4 nodes and its reduced graph.

Proposition 5.6 ($D_p(f_{46})$)**.** Connectivity on the graph with 4 nodes and 6 edges has level-p-complexity

$$D_p(f_{46}) = 3 + 4p + 6p^2 - 27p^3 + 23p^4 - 6p^5$$

which has maximum ≈ 4.39 at $p \approx 0.46$ and minimum 3 at $p = 0 = 1$.

Proof. Now all nodes have degree 3 and due to this symmetry we may WLOG start asking x_1. If $x_1 = 0$ the Boolean function is reduced to f_{45} since the subgraph is isomorphic to the graph seen in Figure 5.4. If instead $x_1 = 1$ resulting subgraph is seen in Figure 5.8b, which gives Boolean function $\mathrm{MAJ}_3(x_3, \mathrm{ANY}_2^1(x_2, x_5), \mathrm{ANY}_2^1(x_4, x_6))$. Due to this it is best to first ask x_3, but then it does not matter if we ask x_2, x_5 or x_4, x_6 first due to the majority characteristics, and also the order between x_2 and

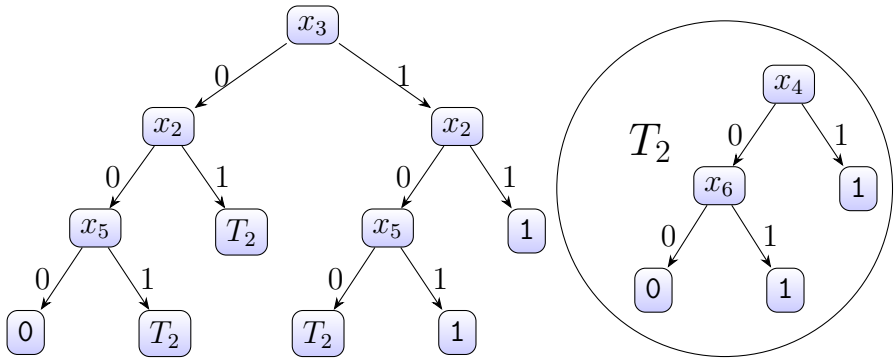


Figure 5.9: One of the possible algorithms for connectivity on the graph in Figure 5.8 after $x_1 = 1$.

x_5 and x_4 and x_6 does not matter. So WLOG we can use the algorithm seen in Figure 5.9 which has expected cost

$$\begin{aligned}\mathbb{E}_x^{\pi_p}[c(A_{\text{reduced}}, x)] &= 1 + (1-p)(1 + (1-p)(1 + p(2-p)) + p(2-p)) \\ &\quad + p(1 + (1-p)(1 + (1-p)(2-p))) \\ &= 3 + 5p - 13p^2 + 9p^3 - 2p^4.\end{aligned}$$

This expression comes from splitting up in two cases, first we ask one question, and then we calculate the expected cost when we get a 0 with probability $1-p$ or 1 with probability p. These subexpressions are calculated similarly and the expected cost for T_2 is $1 + (1-p) = 2 - p$ since we first ask one question and then ask the next with probability $1 - p$. This gives level-p-complexity

$$\begin{aligned}D_p(f_{46}) &= 1 + (1-p)D_p(f_{45}) + p\mathbb{E}_x^{\pi_p}[c(A_{\text{reduced}}, x)] \\ &= 3 + 4p + 6p^2 - 27p^3 + 23p^4 - 6p^5\end{aligned}$$

which has maximum ≈ 4.39 at $p \approx 0.46$ and minimum 3 at $p = 0 = 1$. □

Nothing takes place in the world whose meaning is not that of some maximum or minimum.

Leonhard Euler

6

Level-p-complexity with two maxima

In this section we are going to construct a Boolean function so that D_p has interesting properties. All the examples we have seen so far have one unique maximum. Is there a Boolean function so that $D_p(f)$ has two maxima? Moreover, are these maxima at the endpoints or somewhere inbetween?

Theorem 6.1 ($D_p(f)$ with two maxima)**.** There is a Boolean function f so that $D_p(f)$ has two maxima that are not at the endpoints.

Proof. We do this proof in four steps, first we construct a suitable function, then we describe the possible algorithms for querying the bits, then we describe the calculations of the expected cost, and finally we show the level-p-complexity has two maxima.

Construction of the function Let a and c be natural numbers, and A and C be two disjoint sets partitioning $\{1,\dots,n\}$ so that the sizes of A and C are a and c respectively. We let $f_A = \neg\text{SAME}_a$ and $f_C = \text{SAME}_c$. This means that f_A is 1 only if not all bits in A are the same, and f_C is 1 if all bits in C are the same. In general if we have a function that depends on two disjoint sets of the bits we use the $\text{split}_{A,C} : \{0,1\}^n \to \{0,1\}^a \times \{0,1\}^c$ function to split the bits into two sets: $\text{split}_{A,C}(x) = (x_A, x_C)$. We then define $f(x) = ANY_2^1(f_A(x_A), f_C(x_C))$.

In our example we fix $a = 3$ and $c = 2$ and function f_{AC} can be illustrated as

$$\overbrace{\underbrace{x_1, x_2, x_3}_{\neg\text{SAME}_3}, \underbrace{x_4, x_5}_{\text{SAME}_2}}^{\neg\text{SAME}_3 \vee \text{SAME}_2}$$

In fact $a = 3$ and $c = 2$ are the smallest parameters for which the level-p-complexity of the function has two maxima. For the case of $a = 2$ and $c = 2$ we have functions $f_A = \neg\text{SAME}_2$ and $f_C = \text{SAME}_2$. But

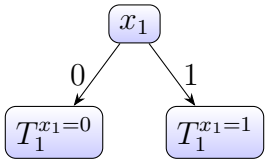
$$\mathbb{P}_p(f_C = 1) = p^2 + (1-p)^2 \geq 2p(1-p) = \mathbb{P}_p(f_A = 1)$$

when $p \in [0,1]$, so it will always be better to ask C before A. In this case we will not get two maxima in $D_p(f_{AC})$. For larger parameters a and c we could get other interesting behaviour but we limit the analysis to the smallest parameters for which $D_p(f_{AC})$ has two maxima, since this is the point of Theorem 6.1 which we want to prove.

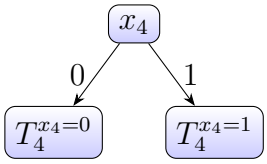
Describing the possible algorithms First we consider algorithms for determining f_A and f_C and then we combine them in different ways to determine f_{AC}. Both f_A and f_C are symmetric, so the order of the input bits do not matter for the output. No matter how we reorder the bits we still have the same result for the property if the bits are same or not. Thus the cost of the algorithm does not depend on which order we ask the bits in sets A and B separately, but it could matter if we are switching between the two sets. We determine f_A by asking the first two bits, and then if they are different, we conclude $f_A = 1$, otherwise we check the third bit as well. Determining f_C is similar; we ask the two bits and if they are the same we conclude $f_C = 1$.

When we combine f_A and f_C, it is enough if we know either is 1, as the result of f_{AC} is 1 as well in this case. Two possible algorithms for f_{AC} is just asking the bits in set A as described above and stopping if $f_A = 1$, otherwise continuing to ask the bits in set C. Or vice versa, asking the bits in set C and stopping if $f_C = 1$, otherwise continuing to ask the bits in set A. We call these algorithms T_{AC} and T_{CA} respectively, and they can be seen in Figure 6.1.

There are also algorithms for f_{AC} that switch between the blocks instead of asking all bits needed to determine f_A in A or all bits in C separately. First, asking one of the bits in A, WLOG x_1, we are left with two reduced Boolean functions on the four remaining bits, $f_{AC}^{x_1=0}$ and $f_{AC}^{x_1=1}$. We have six subalgorithms for $f_{AC}^{x_1=0}$, which can be seen in the Appendix in Figure A.3. We also have six subalgorithms for $f_{AC}^{x_1=1}$, which can be seen in Figure A.4. Since there are two subtrees that could each take 6 subalgorithms, we get $6 \cdot 6$ options in total by combining all the possible subtrees, if we first ask x_1. Figure 6.1a shows one of the 36 possible combinations of the subtrees.



(a) Example of an algorithm where we ask x_1 first, which corresponds to T_{AC}. Subtrees $T_1^{x_1=0}$ and $T_1^{x_1=1}$ can be seen in Figure A.3 and Figure A.4.



(b) Example of an algorithm where we ask x_4 first, which corresponds to T_{CA}. Subtrees $T_1^{x_4=0}$ and $T_1^{x_4=1}$ can be seen in Figure A.1 and Figure A.2.

Figure 6.1: Two of the 52 generated algorithms.

If we first ask a bit in C, WLOG x_4, we have a similar argument, with 4 subalgorithms for $f_{AC}^{x_4=0}$, seen in Figure A.1, and 4 subalgorithms for $f_{AC}^{x_4=1}$, seen in Figure A.2. Figure 6.1b shows one of the 16 possible combinations of the subtrees. In total we have $6 \cdot 6 + 4 \cdot 4 = 52$ algorithms. In Haskell[1] all the 52 possible algorithms and their expected costs were generated as described in Chapter 3. Some of them had equivalent cost, so in total there are 39 unique costs, which can be seen in Figure 6.2.

[1]The code for this can be found in the file `GenAlg.hs`.

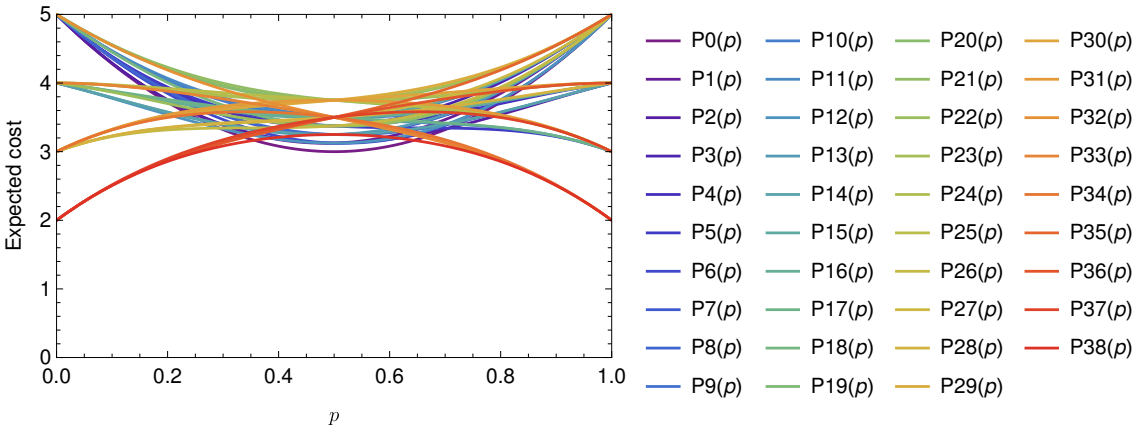


Figure 6.2: Costs of all costs of all the 39 possible algorithms for f_{AC}

Level-p-complexity results In Figure 6.2 we see that there are a lot of lines that overlap and have intersections, however we are only interested in the bottom area, as we are minimizing over algorithms. The lowest lines are P_0 (the purple convex line) and P_{38} (the red concave line) and they intersect at two points around ≈ 0.35 and ≈ 0.65.

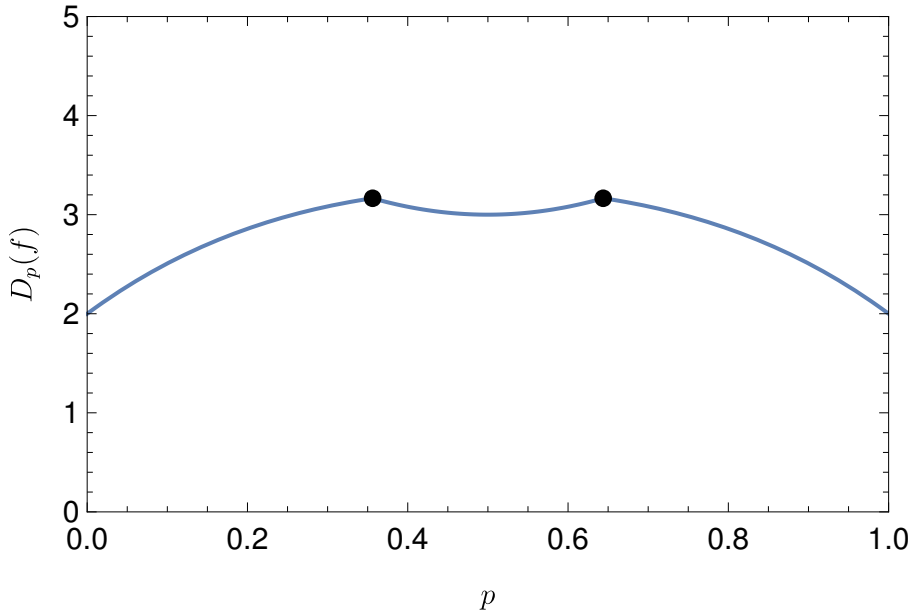


Figure 6.3: Level-p-complexity of f_{AC}, where the red points note the intersections of the costs of the algorithms.

By minimizing over all 39 expected costs in Mathematica[2] for $p \in [0, 1]$ we get

$$D_p(f) = \begin{cases} P_{38}(p) = 2 + 6p - 10p^2 + 8p^3 - 4p^4, & p \in [0, 0.356] \\ P_0(p) = 5 - 8p + 8p^2, & p \in [0.356, 0.644] \\ P_{38}(p) = 2 + 6p - 10p^2 + 8p^3 - 4p^4, & p \in [0.644, 1] \end{cases}$$

[2]See the Mathematica notebook `bimodal.nb`.

and can be seen in Figure 6.3. The level-p-complexity is continuous but not differentiable in the intersection points $p \approx 0.356$ and $p \approx 0.644$ where the derivative changes between ≈ 1.198 and ≈ -2.301. We see clearly that it has two maxima in the intersection points, at $p \approx 0.356$ and $p \approx 0.644$ with value ≈ 3.17.

In fact $P_{38}(p) = \mathbb{E}_x^{\pi_p}[c_f(T_{CA}, x)]$ and $P_0(p) = \mathbb{E}_x^{\pi_p}[c_f(T_{AC}, x)]$, so the level-$p$-complexity can be interpreted in terms of T_{AC} and T_{CA}. For p around $1/2$ it is best to first ask A and then, if needed, C, since the probability that the bits in A are not the same (which is $1 - p^3 - (1-p)^3$) is higher than that of the bits in C being the same (which is $p^2 + (1-p)^2$). On the other hand, for p close to 0 there is a overwhelming probability of both bits in C being 0 and thus the same, while there is less chance of the bits in A being different. This means that we should start with asking C first and then, if needed, A. The same goes for p close to 1 with the bits likely being 1.

Note that both algorithms that have the lowest cost are "naive": they ask all of A and C separately and don't switch between the blocks. So in the end there was no advantage of switching between the blocks for these parameters, but for different parameters it might differ. □

Out of complexity, find simplicity!

Albert Einstein

7

Conclusion

This thesis concerns characteristics of complexity, specifically level-p-complexity, for various Boolean functions. The formulas for the level-p-complexities are seen in Table 7.1. In Chapter 4 we calculated the level-p-complexity for Boolean functions all, tribes, majority and iterated majority. For the ALL_n^1 function (see Section 4.1), $D_p(f)$ is continuous, differentiable and increasing in p on the whole interval.

f	$D_p(f)$	$p_{\max}$	$D_{p_{\max}}(f)$
Const	0	—	—
Dictator	1	—	—
Parity	n	—	—
ALL_n^1	$\frac{1-p^n}{1-p}$	1	n
Tribes, $\mathrm{TRI}_{m,k}$	$\frac{(1-(1-p^k)^m)(1-p^k)}{p^k(1-p)}$	—	—
MAJ_3	$2+2p-2p^2$	$1/2$	$5/2$
MAJ_5	$3+3p+3p^2-12p^3+6p^4$	$1/2$	4.125

Table 7.1: Level-p-complexity for some Boolean functions.

The level-p-complexity of the tribes function (see Section 4.2) can be seen in Table 7.1, and its values at the endpoints are $D_0(\mathrm{TRI}_{m,k}) = m$ and $D_1(\mathrm{TRI}_{m,k}) = k$. The function is continuous and differentiable and the maximum can be found numerically depending on m and k. The analytical formula for the maximum was not found, and therefore it is also hard to reason about if the maxima is unique even if it is unique for all m and k parameters tried numerically.

Furthermore, the formulas for the majority functions on 3 and 5 bits (see Section 4.3) can be seen in Table 7.1. They are symmetric around $p = 1/2$, which can be seen by writing them in the symmetric polynomial representation. The proposed formula for iterated majority was too complicated to fit in the table, but it can be seen in Conjecture 4.1 in Section 4.4. It is also symmetric around $p = 1/2$ and has maxima at $p = 1/2$ of 6.1875.

Further, Chapter 5 explores level-p-complexity for the graph property of connectivity, and we can see the results for graphs with 3 or 4 nodes in Table 7.2. The graphs with 3 nodes have very simple formulas, and the same goes for the graph with 4 nodes and 3 edges. For 4 nodes and 4,5 or 6 edges we get polynomials of higher order, but still the same algorithm is optimal for all p so the functions are continuous and differentiable.

In Chapter 6 a function f_{AC} is constructed such that the level-p-complexity has two maxima. There are a lot of different relevant algorithms for f_{AC} and they

f	$D_p(f)$	$p_{\max}$	$D_{p_{\max}}(f)$
f_{32}	$1+p$	1	2
f_{33}	$2+2p-2p^2$	1/2	5/2
$f_{43\text{class1}}$	$1+p+p^2$	1	3
$f_{43\text{class2}}$	$1+p+p^2$	1	3
$f_{44\text{class1}}$	$2+2p+2p^2-3p^3$	≈ 0.74	≈ 3.36
$f_{44\text{class2}}$	$1+2p+2p^2-2p^3$	1	3
f_{45}	$2+3p+4p^2-10p^3+4p^4$	≈ 0.64	≈ 3.61
f_{46}	$3+4p+6p^2-27p^3+23p^4-6p^5$	≈ 0.46	≈ 4.39

Table 7.2: Level-p-complexity for connectivity of graphs with 3 nodes with 2 or 3 edges, and 4 nodes with 3, 4, 5 or 6 edges.

intersect many times, yielding the level-p-complexity

$$D_p(f_{AC}) = \begin{cases} \mathbb{E}_x^{\pi_p}[c_f(T_{CA}, x)] = 2+6p-10p^2+8p^3-4p^4, & p \in [0, 0.356] \\ \mathbb{E}_x^{\pi_p}[c_f(T_{AC}, x)] = 5-8p+8p^2, & p \in [0.356, 0.644] \\ \mathbb{E}_x^{\pi_p}[c_f(T_{CA}, x)] = 2+6p-10p^2+8p^3-4p^4, & p \in [0.644, 1] \end{cases}$$

This is the first function for which the optimal algorithm depends on p, as the expected cost of the different algorithms intersect, so the level-p-complexity is piecewise differentiable, with different polynomial form in the different intervals. The two maxima are at $p \approx 0.356$ and $p \approx 0.644$ with value ≈ 3.17.

To summarize, we have calculated the level-p-complexity for Boolean functions such as const, dictator, all, tribes, majority, graph connectivity on graphs with 3 or 4 nodes. But this is only part of all the possible Boolean functions, so it could be interesting to calculate the level-p-complexity for other Boolean functions[1]. Some examples are extensions of Boolean functions whose complexity we have calculated and will likely have similar results, like the threshold function which is described in Section 4.3, but others are entirely different, for example percolation [GS14].

Another extension would be to properly prove Conjecture 4.1 and not just provide strong evidence. It would also be interesting to go further in the iterated majority function MAJ_k^n and explore different cases with different n and k.

In my opinion, the most interesting direction would be to continue constructing new Boolean functions using easy building blocks like in Chapter 6, and explore properties of their level-p-complexity. For example it would be interesting if we could find a monotone function for which the level-p-complexity has two maxima, since f_{AC} is not monotone.

[1]See a list of Boolean functions at the wiki page: `https://booleanzoo.weizmann.ac.il/index.php/Main_Page#Boolean_functions`

Bibliography

[Ban64] John F Banzhaf III. "Weighted voting doesn't work: A mathematical analysis". In: *Rutgers L. Rev.* 19 (1964), p. 317.

[BM+76] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*. Vol. 290. Macmillan London, 1976.

[BD02] Harry Buhrman and Ronald De Wolf. "Complexity measures and decision tree complexity: a survey". In: *Theoretical Computer Science* 288.1 (2002), pp. 21–43.

[CH11] Yves Crama and Peter L Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

[Fek23] Michael Fekete. "Über die Verteilung der Wurzeln bei gewissen algebraischen Gleichungen mit ganzzahligen Koeffizienten". In: *Mathematische Zeitschrift* 17.1 (1923), pp. 228–249.

[GS14] Christophe Garban and Jeffrey E Steif. *Noise sensitivity of Boolean functions and percolation*. Vol. 5. Cambridge University Press, 2014.

[Haj92] Péter Hajnal. *Decision tree complexity of Boolean functions*. North-Holland Publishing Co.; János Bolyai Mathematical Society, 1992.

[Hu65] Sze-Tsen Hu. *Threshold logic*. University of California Press, 1965.

[JIB22] Patrik Jansson, Cezar Ionescu, and Jean-Philippe Bernardy. *Domain-Specific Languages of Mathematics*. English. Vol. 24. Texts in Computing. College Publications, Jan. 2022, p. 268. ISBN: 978-1-84890-388-3.

[KP17] Anna R Karlin and Yuval Peres. *Game theory, alive*. Vol. 101. American Mathematical Soc., 2017.

[Knu12] Donald E Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2012.

[Lan+06] Itamar Landau et al. "The lower bound for evaluating a recursive ternary majority function: an entropy-free proof". In: *Undergraduate Research Reports, Department of Statistics, University of California, Berkeley* (2006).

[Leo13] Nikos Leonardos. "An improved lower bound for the randomized decision tree complexity of recursive majority". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2013, pp. 696–708.

[LY02] László Lovász and Neal E Young. "Lecture notes on evasiveness of graph properties". In: *arXiv preprint cs/0205031* (2002).

[Mag+16] Frédéric Magniez et al. "Improved bounds for the randomized decision tree complexity of recursive majority". In: *Random Structures & Algorithms* 48.3 (2016), pp. 612–638.

[Neu28] John von Neumann. "Zur theorie der gesellschaftsspiele". In: *Mathematische annalen* 100.1 (1928), pp. 295–320.

[ODo14] Ryan O'Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

[OS07] Ryan O'Donnell and Rocco A Servedio. "Learning monotone decision trees in polynomial time". In: *SIAM Journal on Computing* 37.3 (2007), pp. 827–844.

[Pen46] Lionel S Penrose. "The elementary statistics of majority voting". In: *Journal of the Royal Statistical Society* 109.1 (1946), pp. 53–57.

[RV75] Ronald L Rivest and Jean Vuillemin. "A generalization and proof of the Aanderaa-Rosenberg conjecture". In: *Proceedings of the seventh annual ACM symposium on Theory of computing*. 1975, pp. 6–11.

[Ros73] Arnold L Rosenberg. "On the time required to recognize properties of graphs: A problem". In: *ACM SIGACT News* 5.4 (1973), pp. 15–16.

[Yam12] Yoshinori Yamamoto. "Banzhaf index and Boolean difference". In: *2012 IEEE 42nd International Symposium on Multiple-Valued Logic*. IEEE. 2012, pp. 191–196.

A
Appendix

This Appendix mainly consists of the 10 different possible algorithms for the reduced functions $f_{AC}^{x_1=0}$, $f_{AC}^{x_1=1}$, $f_{AC}^{x_4=0}$ and $f_{AC}^{x_4=1}$, seen in Figures A.3, A.4, A.1 and A.2 respectively.

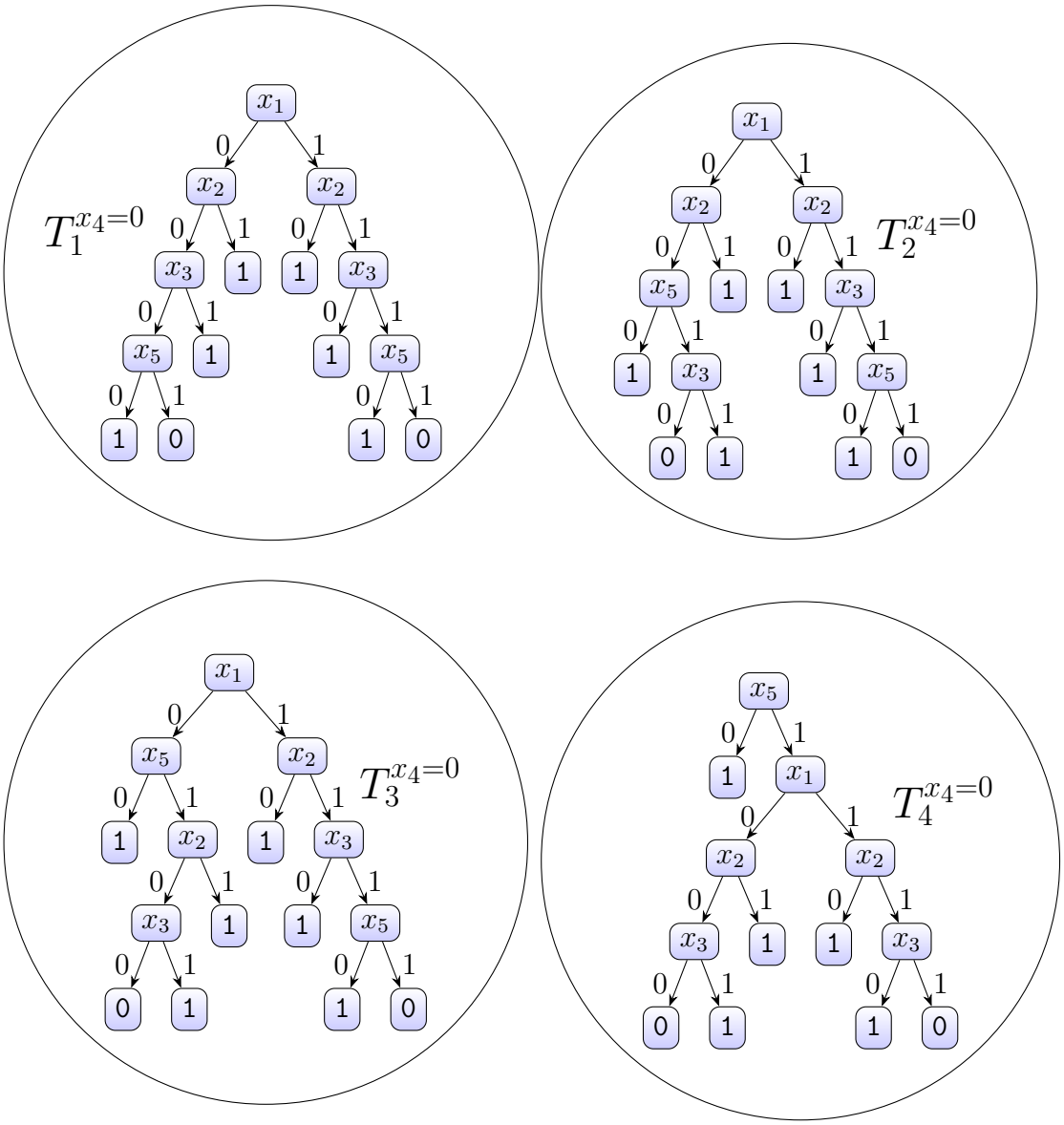


Figure A.1: The four subalgorithms for the reduced function $f^{x_4=0}$.

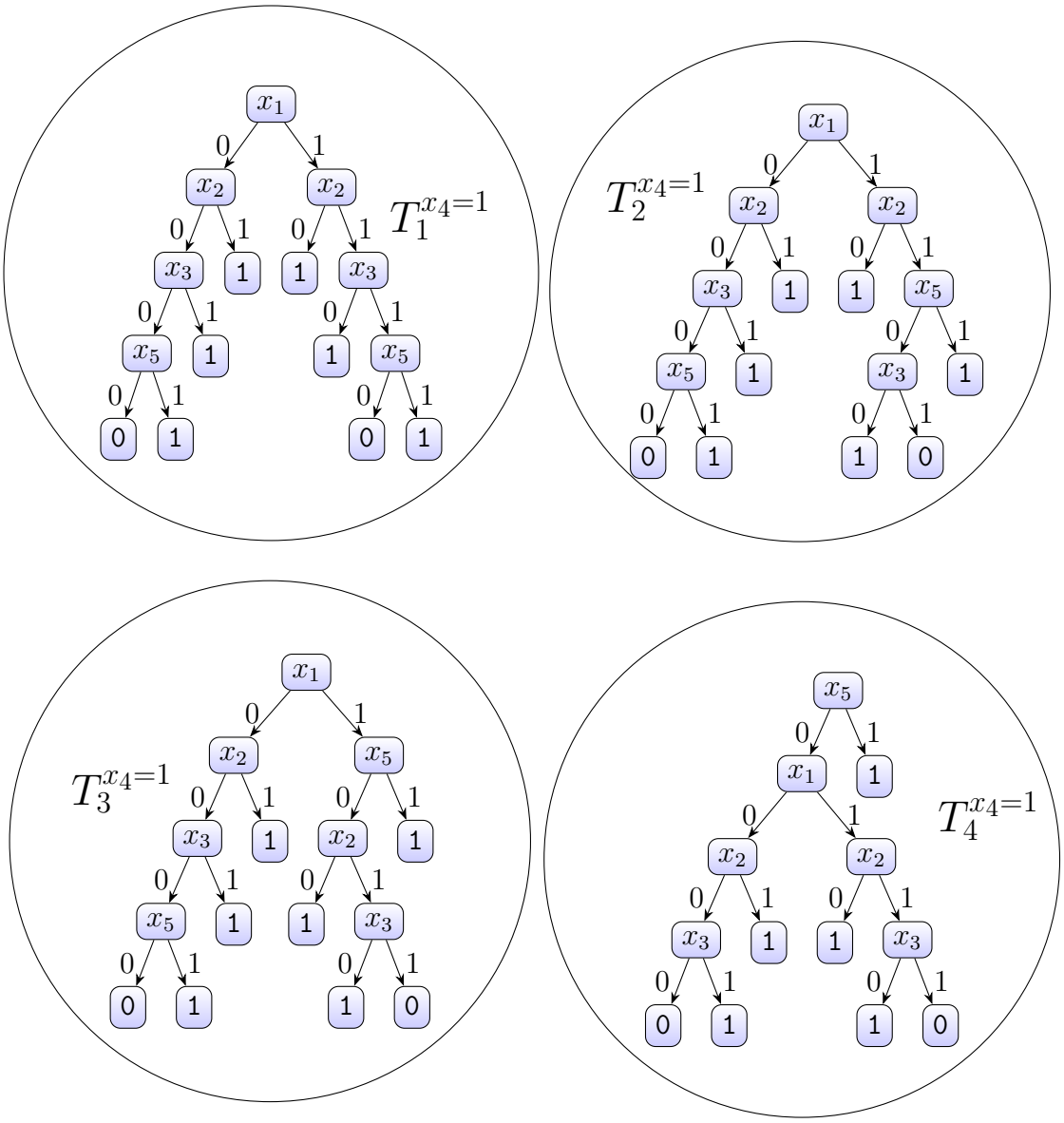


Figure A.2: The four subalgorithms for the reduced function $f^{x_4=1}$.

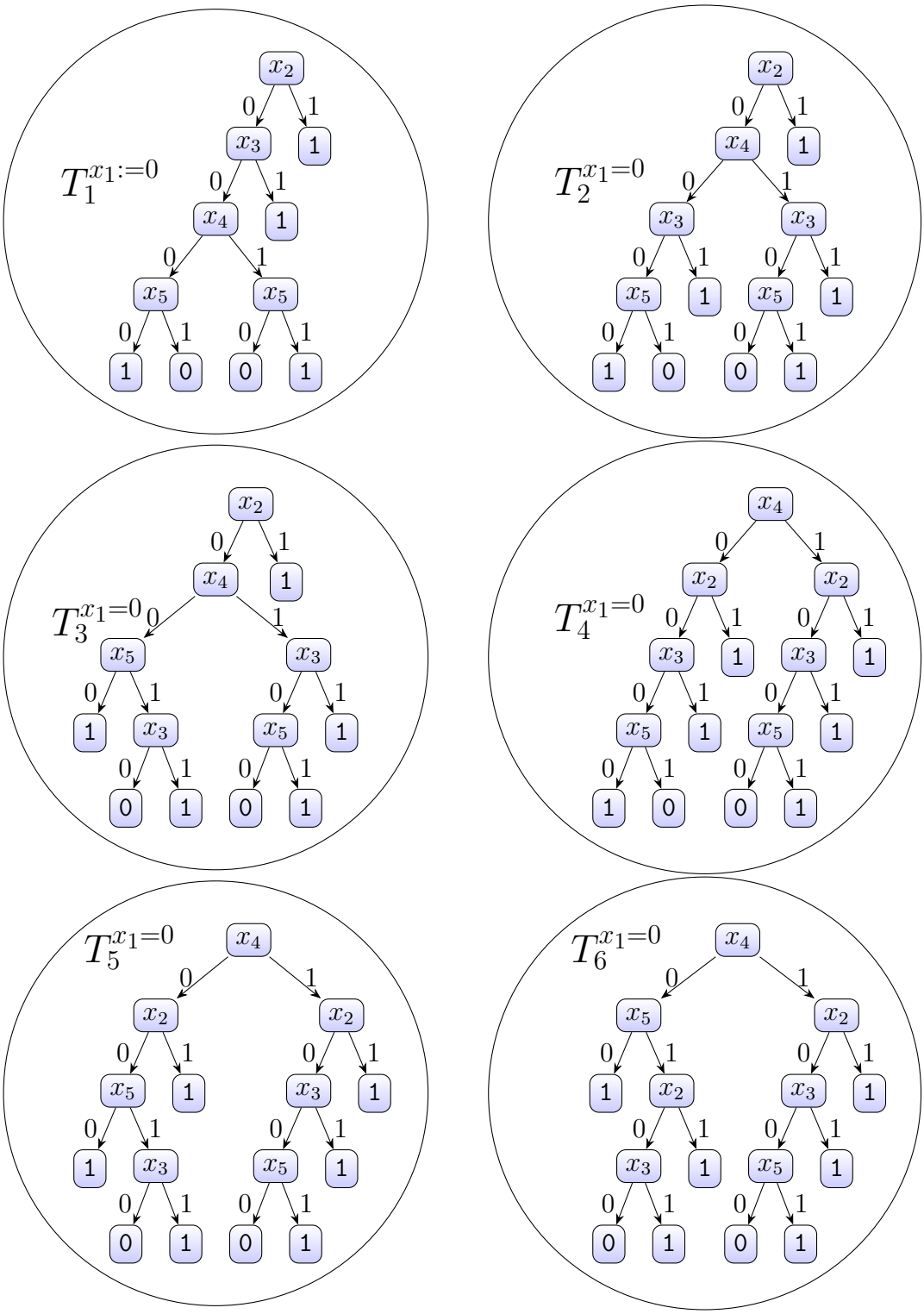


Figure A.3: The six subalgorithms for the reduced function $f^{x_1=0}$.

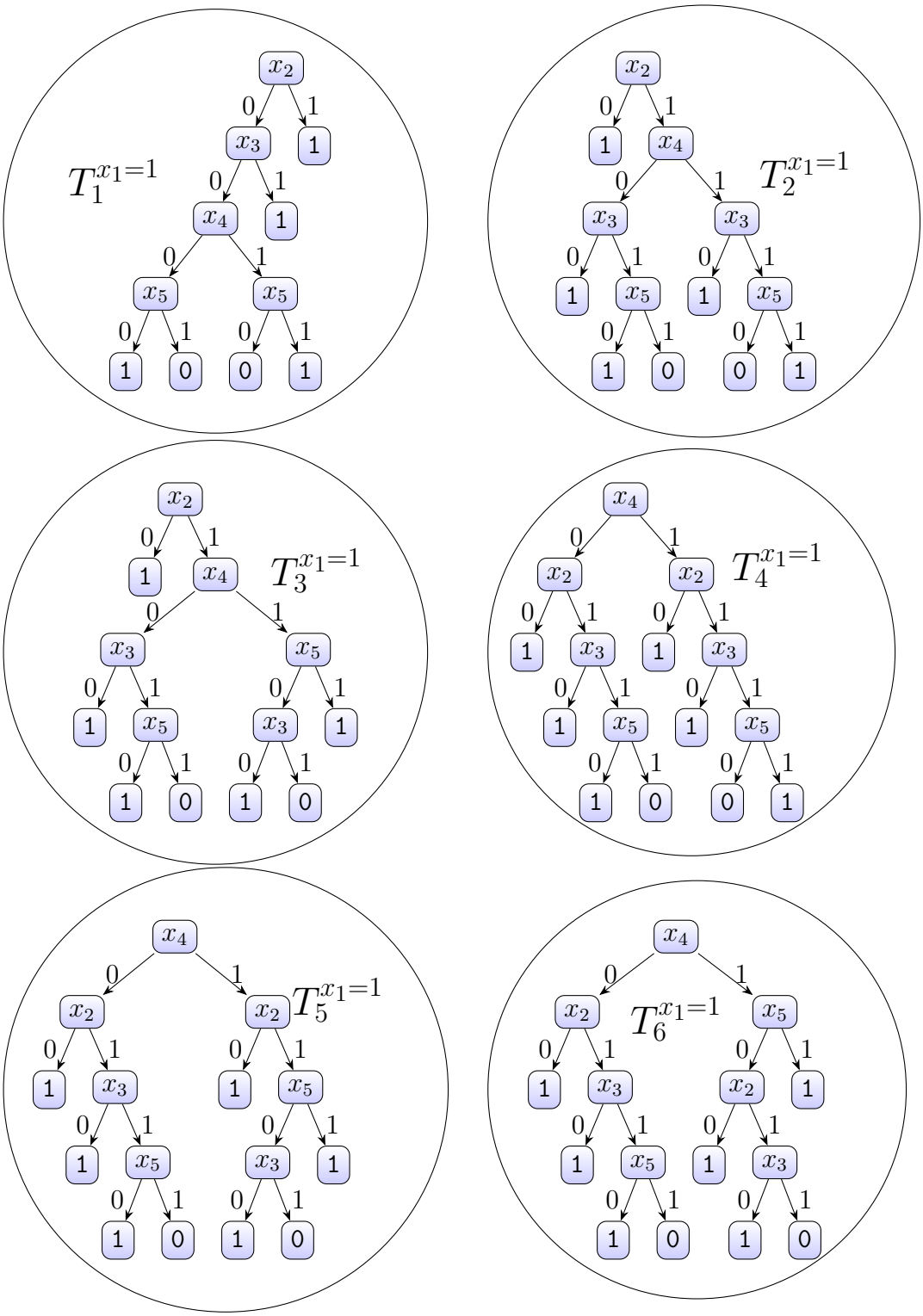


Figure A.4: The six subalgorithms for the reduced function $f^{x_1=1}$.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY