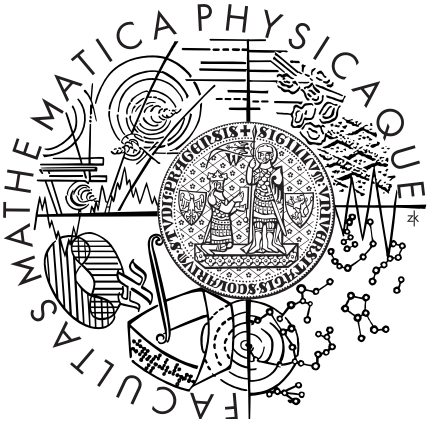
Charles University in Prague
Faculty of Mathematics and Physics
Department of Theoretical Computer Science

DOCTORAL THESIS

David Kronus

Interval Representations of Boolean Functions



Prague, 2007

Supervisor : doc. RNDr. Ondřej Čepek, Ph.D.
Branch : i1 – Theoretical Computer Science

I would like to thank my supervisor, Ondřej Čepek. He has provided motivation and support for my research and his help with the preparation of this thesis is invaluable. I also gratefully acknowledge the support of the Czech Science Foundation under the contract no. 201/05/H014.

Abstract

This thesis is dedicated to a research concerning representations of Boolean functions. We present the concept of a representation using intervals of integers. Boolean function f is represented by set I of intervals, if it is *true* just on those input vectors, which correspond to integers belonging to intervals in I, where the correspondence between vectors and integers depends on the ordering of bits determining their significancies. We define the classes of k-interval functions, which can be represented by at most k intervals with respect to a suitable ordering of variables, and we provide a full description of inclusion relations among the classes of threshold, 2-monotonic and k-interval Boolean functions (for various values of k).

The possibility to recognize in polynomial time, whether a given function belongs to a specified class of Boolean functions, is another fundamental and practically important property of any class of functions. Our results concerning interval functions recognition include a proof of co-NP-hardness of the general problem and polynomial-time algorithms for several restricted variants, such as recognition of 1-interval and 2-interval positive functions. We also present an algorithm recognizing general 1-interval functions provided that their DNF representation satisfies several (quite strong) conditions. For 2-monotonic or threshold functions we costruct an algorithm finding an interval representation with the minimum number of intervals.

The extension problem for interval functions involves deciding, whether a given partially defined function can be extended to a k-interval function, and we present polynomial-time algorithms solving it for the classes of positive 1-interval, general 1-interval and renamable 1-interval functions. In the best-fit extension problem, which generalizes the extension problem, we are given a partially defined function and our task is to find a totally defined k-interval function, which disagrees with the input function on the minimum number of inputs (or, in the weighted case, on a set of inputs with the minimum weight). By proving that it is NP-hard for positive 1-interval and general 1-interval functions we show, that the best-fit extension is substantially harder than the extension problem. We only manage to construct a polynomial-time algorithm for the 1-interval best-fit extension, when the ordering of variables determining their significance is fixed.

Contents

List of Figures

1 Introduction

The concept of Boolean functions is being used in many branches of theoretical computer science, mathematics and logic. In many more areas of research (e.g. relational databases, knowledge bases and neural networks) this concept can be used as a useful abstraction to formalize problems, describe their properties and find their solutions. Therefore the problems studied in relation to Boolean functions can have practical applications both in other areas of research and in technical practice.

One of the fundamental areas of research concerning Boolean functions studies their representations. While propositional logic formulae constitute probably the most widely used and best understood representation of Boolean functions, there exist other representation methods such as various kinds of circuits, decision trees, OBBDs and many more. Many problems can be studied for every such representation, e.g., the optimalization problem of finding the minimum (i.e., "shortest") representation of a chosen type.

In this thesis we show how to represent a Boolean function by intervals of integer numbers and we study the properties of this representation. This concept has been introduced in [24]. Disjoint intervals $I_1, \ldots, I_k$ of n-bit integers represent Boolean function f on n input variables, if the following condition is satisfied: f is *true* on input vector $\boldsymbol{x}$, if and only if $\boldsymbol{x}$ viewed as an n-bit integer belongs to one of the intervals $I_1, \ldots, I_k$. The correspondence of Boolean vectors of length n and n-bit integers is not unique, we have to specify the ordering determining the significancies of the bits. The number of intervals needed to represent Boolean function f may depend on the ordering of variables of f.

We use interval representation to define new classes of Boolean functions. In particular, the functions, which can be represented by at most k intervals (when using a suitable ordering of variables), constitute the class of k-interval functions. We show inclusion relations of these classes to some other well-known classes of Boolean functions. This provides some insight on the position of k-interval functions in the hierarchy of classes of Boolean functions.

Two fundamental problems, which can be studied for any class $\mathcal{C}$ of Boolean functions, are:

1. **Recognition problem:** Given a totally defined Boolean function, does it belong to $\mathcal{C}$?

2. **Extension problem:** Given a partially defined Boolean function, can it be extended to a total Boolean function, which belongs to $\mathcal{C}$?

For the classes of k-interval functions, the two above mentioned problems mean, that for a total Boolean function, given by some traditional representation, we want to decide, whether there is an ordering of its variables, with respect to which the function can be represented by at most k intervals, and, in case of the extension problem, whether a partial Boolean function, given by some traditional representation, can be extended to a total Boolean function, which can be represented by at most k intervals. We study both of these problems here.

In case of real-world experimental data originating from various sources and maybe even measured with varying reliabilities, an extending total Boolean function with the required properties may not exist, possibly due to errors or inaccuracies in some measurements. In this situation we may relax our requirement, that the

extension must perfectly agree with the given data, and we can try to search for a function, which belongs to the specified class of Boolean functions and which has the minimum number of conflicts with the input data. In case of variably reliable measurements, this minimum can be weighted, thus an optimal extending function should prefer violating unreliable data to violating reliable data. This problem, called **best-fit extension**, is also a subject of our research.

1.1 Outline

This dissertation is structured as follows:

In Section 2 we introduce the basic definitions and notation. In Section 3 we summarize some known properties of Boolean function representations and present several minor new results describing basic properties of interval representation. In Section 4 we show relations of some known classes of Boolean functions to Boolean functions, which have an interval representation with a bounded number of intervals. In particular, we study interval representations of threshold and 2-monotonic functions.

Section 5 is dedicated to the recognition of k-interval functions. First, we show that in general this problem is hard to solve even when the ordering of variables is fixed in advance. Therefore, we then study somehow restricted versions of the recognition problem, for example when the input function is from a specified class of Boolean functions, such as the classes of positive or regular functions. We present several polynomial-time algorithms for various modifications of the recognition problem.

Then in Section 6 we study the extension problem of partial Boolean functions for the class of Boolean functions, which can be represented by one interval. We present polynomial-time algorithm for this problem and we also generalize this result to the "renamable" variant of 1-interval functions. Our results concerning the best-fit extension problem of 1-interval functions are presented in Section 7. We show, that this problem is hard to solve in general, and present a polynomial-time algorithm for the case, when the ordering of variables is fixed.

We give some final remarks and conclusion in Section 8.

2 Notation and Definitions

In this section we summarize all the notation and definitions, that will be used throughout this thesis. We start with Boolean functions in general and then proceed to their representation by logic formulae. We define the notion of partially defined Boolean function and then we give a list of the classes of Boolean functions, that we study in this thesis. We also introduce the formal definition of the interval representation of Boolean functions. At the end of this section we summarize a list of the hard problems (both NP-hard and co-NP-hard) that we refer to in this thesis.

2.1 Boolean functions

A **Boolean function**, or a **function** in short, on n propositional variables is a mapping $f : \{0,1\}^n \mapsto \{0,1\}$. Usually, the **input variables** of a function will be denoted by $x_1, x_2, \ldots, x_n$. A **Boolean vector of length** n is an n-tuple of **Boolean**

values 1 and 0 (usually denoted by **true** and **false**). Boolean vectors (or **vectors** for short) will be denoted by $\boldsymbol{x}, \boldsymbol{y} \ldots$. The **bits** of vector $\boldsymbol{x} \in \{0,1\}^n$ will be denoted by $x_1, \ldots, x_n$. If $f(\boldsymbol{x}) = 1$ (0, resp.), then $\boldsymbol{x}$ is called a **true** (**false**, resp.) vector of f (sometimes called **truepoint** and **falsepoint**, resp.). The set of all true vectors (false vectors, resp.) is denoted by $T(f)$ ($F(f)$, resp.). For function f on n variables and $v \in \{0,1\}$ we denote by $f[x_i := v]$ the function on $(n-1)$ variables, which is formed from f by fixing the value of the i-th variable to v.

We will also use some well-known Boolean functions. The first one is the PARITY_n function on n variables, which is 1, if and only if the sum of all its input variables is odd. The second one is the MAJORITY_n function on n variables, which is 1, if and only if there are more input variables with value 1 than those with value 0.

For a vector $\boldsymbol{v} \in \{0,1\}^n$, let $ON(\boldsymbol{v}) = \{j \mid v_j = 1, j = 1,2,\ldots,n\}$ and let $OFF(\boldsymbol{v}) = \{j \mid v_j = 0, j = 1,2,\ldots,n\}$. By $\boldsymbol{x}^A$, where $A \subseteq \{1,\ldots,n\}$ for some $n \in \mathbf{N}$, we denote the Boolean vector of length n, which has ones exactly on those indices i, which are in A, i.e., $ON(\boldsymbol{x}^A) = A$. For vectors $\boldsymbol{u}, \boldsymbol{v}$ of the same length n we denote by $\boldsymbol{u} \geq \boldsymbol{v}$, that $\boldsymbol{u}$ is componentwise greater than or equal to $\boldsymbol{v}$, i.e., $\forall i \in \{1,\ldots,n\} : u_i \geq v_i$. We use vector relations $=, \neq, <, >, \leq$ in a similar manner.

Propositional variables $x_1, \ldots, x_n$ and their negations $\overline{x}_1, \ldots, \overline{x}_n$ are called **literals** (**positive** and **negative literals**, respectively). An elementary conjunction of literals

$$t = \bigwedge_{i \in I} x_i \wedge \bigwedge_{j \in J} \overline{x}_j \tag{2.1}$$

is called a **term**, if every propositional variable appears in it at most once, i.e., if $I \cap J = \emptyset$. A **disjunctive normal form** (or DNF) is a disjunction of terms. It is a well known fact (see e.g. [14]), that every Boolean function can be represented by a DNF. For DNF $\mathcal{F}$ and term t we denote by $t \in \mathcal{F}$ the fact, that t is contained in $\mathcal{F}$. For Boolean vector $\boldsymbol{v}$ and term t we say, that t **satisfies** (**falsifies**, resp.) $\boldsymbol{v}$, if t evaluates to 1 (0, resp.) on $\boldsymbol{v}$. In the subsequent text the "$\wedge$" sign for conjunction will be frequently omitted. The number of literals of DNF $\mathcal{F}$ will be denoted by $|\mathcal{F}|_l$ and the number of terms by $|\mathcal{F}|_t$.

The DNF version of the **satisfiability problem** (sometimes called the **falsifiability problem**) is defined as follows:

FALSIFIABILITY($\mathcal{F}$)	
Instance :	A DNF $\mathcal{F}$.
Question :	Is there an assignment of truth values to the variables which makes $\mathcal{F}$ evaluate to 0?

It is a well-known fact that this problem is NP-complete ([13]).

Given Boolean functions f and g on the same set of variables, we denote by $f \leq g$ the fact, that g is satisfied for any assignment of values to the variables, for which f is satisfied. We call term t an **implicant** of DNF $\mathcal{F}$, if $t \leq \mathcal{F}$. We call t a **prime implicant**, if t is an implicant of $\mathcal{F}$ and there is no implicant $t' \neq t$ of $\mathcal{F}$, for which $t \leq t' \leq \mathcal{F}$. We call DNF $\mathcal{F}$ **prime**, if it consists only of prime implicants. We call DNF $\mathcal{F}$ **irredundant**, if for any term $t \in \mathcal{F}$, DNF $\mathcal{F}'$ produced from $\mathcal{F}$ by deleting t does not represent the same function as $\mathcal{F}$. It is a well known fact, that if $\mathcal{F}$ belongs to some class $\mathcal{C}$ of DNFs, for which we can solve the falsifiability

problem in polynomial time and which is closed under partial assignment, then we can test in polynomial time for term t and DNF $\mathcal{F}$, whether t is an implicant of $\mathcal{F}$. To see this, observe that given a term $t = x_1 \dots x_{l_p} \overline{y}_1 \dots \overline{y}_{l_n}$, t is an implicant of f, if and only if $\mathcal{F}[x_1 := 1, \dots, x_{l_p} := 1, y_1 := 0, \dots, y_{l_n} := 0]$ is not falsifiable (there is no assignment to the remaining variables, which makes the DNF evaluate to 0). Therefore, for any such class $\mathcal{C}$ of DNFs, it is possible to transform a given input DNF from $\mathcal{C}$ to a logically equivalent DNF, which is prime and irredundant. The time requirements of this transformation can be expressed as $\Theta(|F|_l \cdot f_{\mathcal{C}}(|F|_l, |F|_t))$, where $f_{\mathcal{C}}(l, t)$ is a function describing the time needed for deciding falsifiability for a DNF from class $\mathcal{C}$, which has l literals and t terms.

We say, that two terms t_1 and t_2 **conflict in a variable** x, if t_1 contains literal x and t_2 contains literal $\overline{x}$. Two terms t_1 and t_2 have a **consensus**, if they conflict in exactly one variable. If $t_1 = Ax$ and $t_2 = B\overline{x}$, where A, B are two sets of literals and x is the only variable, in which t_1 and t_2 have conflict, we call a term $t = AB$ a **consensus of** terms t_1 and t_2. It is a well known fact (see [22]), that any prime implicant of a DNF $\mathcal{F}$ can be generated from $\mathcal{F}$ by a series of consensuses.

2.2 Partially defined Boolean functions

A **partially defined Boolean function (pdBf)** (see e.g. [5]) is defined by a pair of sets (T, F) of Boolean vectors on n variables, where T denotes a set of true vectors (or positive samples) and F denotes a set of false vectors (or negative samples). A Boolean function f is called an **extension** (or a theory) of the pdBf (T, F) if $T \subseteq T(f)$ and $F \subseteq F(f)$. We shall also say in this case that function f **correctly classifies** all vectors $\boldsymbol{a} \in T$ and $\boldsymbol{b} \in F$.

Clearly, the disjointness of the sets T and F is a necessary and sufficient condition for the existence of an extension. It is not clear, however, how difficult it is to find out, whether a given pdBf has an extension in $\mathcal{C}$, where $\mathcal{C}$ denotes a class of Boolean functions, such as the class of positive functions, k-DNF functions, etc. Therefore, we consider the problem of deciding the existence of an extension f in the specified class $\mathcal{C}$ of a given pdBf (T, F). (See also [5].)

EXTENSION($\mathcal{C}$)	
Instance :	A pdBf (T, F), where $T, F \subseteq \{0,1\}^n$.
Question :	Is there an extension $f \in \mathcal{C}$ of (T, F)?

Let us add that, in case of the affirmative answer, we expect to be able to specify a Boolean function $f \in \mathcal{C}$, for which $T \subseteq T(f)$ and $F \subseteq F(f)$, as a justification. If EXTENSION($\mathcal{C}$) is solvable in polynomial time, such a function should be defined in a polynomial way, so that for any vector $\boldsymbol{x} \in \{0,1\}^n$ one can compute its value $f(\boldsymbol{x})$ in time polynomial in the size of (T, F).

When there is no extension for a given pdBf in $\mathcal{C}$, we can try to find such a function from $\mathcal{C}$, which disagrees with the given pdBf on a minimum number of vectors. If the vectors of the input pdBf are weighted, we can search for a weighted minimum. This brings us to the best-fit problem.

Let ω be a positive function specifying weight $\omega(\boldsymbol{x})$ for every $\boldsymbol{x} \in T \cup F$ and let $\omega(S) = \sum_{\boldsymbol{x} \in S} \omega(\boldsymbol{x})$ for a subset $S \subseteq T \cup F$. Furthermore, let us define the **error**

size of a function f by

$$\epsilon(f) = \omega(T \cap F(f)) + \omega(F \cap T(f)) \ . \tag{2.2}$$

Using the measure of error defined by (2.2) we shall consider the following problem:

BEST-FIT($\mathcal{C}$)	
Instance :	A pdBf $(T,\ F)$, where $T, F \subseteq \{0,1\}^n$, and a positive weight function $\omega : T \cup F \mapsto \mathbf{R}_+$.
Goal :	Sets T^*, F^*, such that $T^* \cap F^* = \emptyset$, $T^* \cup F^* = T \cup F$, for which the pdBf (T^*, F^*) has an extension from $\mathcal{C}$, and $\omega(T^* \cap F) + \omega(T \cap F^*)$ is minimum.

Let us remark that, by the definition of T^* and F^*, any extension $f \in \mathcal{C}$ of the pdBf (T^*, F^*) satisfies

$$\epsilon(f) = \omega(T^* \cap F) + \omega(F^* \cap T)(= \min_{f' \in \mathcal{C}} \epsilon(f')) \ . \tag{2.3}$$

In other words, a pair of sets T^* and F^* attaining the minimality in the above best-fit problem could equivalently be defined as $T^* = (T \setminus F(f)) \cup (F \cap T(f))$ and $F^* = (F \setminus T(f)) \cup (T \cap F(f))$ for any function $f \in C$ having the minimum error size.

Again, when best-fit for a class $\mathcal{C}$ is solvable in polynomial time, we expect to be able to specify (in a polynomial way as well) a function $f \in \mathcal{C}$ that extends pdBf (T^*, F^*).

It is not necessary to allow also negative weights for vectors in $T \cup F$, because having $\boldsymbol{u} \in T$ with $\omega(\boldsymbol{u}) = p < 0$ is equivalent to having $\boldsymbol{u} \in F$ with $\omega(\boldsymbol{u}) = -p > 0$. Also, it is obvious that vectors $\boldsymbol{x} \in T \cup F$ with $\omega(\boldsymbol{x}) = 0$ could not play any role in the best-fit problem. Hence the requirement of positivity of the weights is not a restriction of generality.

Clearly, the extension problem is a special case of the best-fit problem, since EXTENSION has a solution if and only if BEST-FIT has a solution f with $\epsilon(f) = 0$. This means, that if BEST-FIT($\mathcal{C}$) is solvable in polynomial time (i.e., polynomial in n, $|T|$ and $|F|$), for some class $\mathcal{C}$, then EXTENSION($\mathcal{C}$) is also polynomially solvable. Conversely, if EXTENSION($\mathcal{C}$) is NP-hard, then so is BEST-FIT($\mathcal{C}$).

2.3 Studied classes of Boolean functions

We shall proceed with the definitions of the classes of Boolean functions that are studied in this thesis.

The class of all Boolean functions is in this dissertation denoted by $\mathcal{C}_{\text{all}}$.

A term t defined by (2.1) is called a **k-term**, if it contains exactly k literals (i.e. if $|I \cup J| = k$). A 1-term is also called a **linear term**. A DNF is called a **k-DNF**, if it consists only of k-terms. Boolean function belongs to the class of **k-DNF** functions, if it can be represented by a k-DNF. The class of k-DNF functions will be denoted by $\mathcal{C}_{k-\text{DNF}}$.

A term t defined by (2.1) is called **positive**, if it contains no negative literals (i.e., if $J = \emptyset$), and it is called **negative**, if it contains no positive literals (i.e., if $I = \emptyset$).

A DNF is called **positive**, if it contains only positive terms. Finally, a Boolean function is called **positive**, if it has at least one representation by a positive DNF. **Negative** DNFs and functions are defined similarly. The class of positive Boolean functions will be denoted by $\mathcal{C}^+$, the class of negative Boolean functions by $\mathcal{C}^-$.

The vector $\boldsymbol{x}$ corresponds to an integer number x with binary representation equal to $\boldsymbol{x}$. In this case x_1 is the most significant bit of x and x_n the least significant bit. Hence $x = \sum_{i=1}^{n} x_i 2^{n-i}$. Also, for any integer x we denote by $\boldsymbol{x}$ the vector corresponding to the binary representation of x. In case integer x is specified by an expression e, we denote the vector corresponding to the binary representation of its value by $\overrightarrow{e}$.

Definition 2.1. *For vector $\boldsymbol{x} \in \{0,1\}^n$ and for permutation $\pi : \{1,\ldots,n\} \mapsto \{1,\ldots,n\}$ we denote by $\boldsymbol{x}^\pi$ the vector of n bits formed by permuting bits of $\boldsymbol{x}$ by π. That means $x_i^\pi = x_{\pi(i)}$. By x^π we denote the number with binary representation $\boldsymbol{x}^\pi$.*

We denote the identical permutation by *id* (the number of its variables will always follow from the context).

Using the just introduced notation, we can easily define the class of Boolean functions, which can be represented by one interval.

Definition 2.2. *Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called a* **1-interval function** *(or simply* **interval function***), if there exist two n-bit integers a, b and permutation π of $\{1,\ldots,n\}$, such that for every n-bit vector $\boldsymbol{x} \in \{0,1\}^n$ we get $f(\boldsymbol{x}) = 1$, if and only if $x^\pi \in [a,b]$. The class of interval functions will be denoted by $\mathcal{C}_{1-\mathrm{int}}$. The class of positive (negative, resp.) interval functions will be denoted by $\mathcal{C}^+_{1-\mathrm{int}}$ ($\mathcal{C}^-_{1-\mathrm{int}}$, resp.).*

We can generalize this definition to any number of representing intervals. We present here the inductive definition of k-interval functions.

Definition 2.3. *Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called a* **0-interval function***, if it is identically equal to zero, i.e., $f(\boldsymbol{x}) = 0$ holds for all $\boldsymbol{x} \in \{0,1\}^n$.*

Definition 2.4. *Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called a k-***interval function***, for $k \geq 1$, if it is a $(k-1)$-interval function or there exist k pairs of n-bit integers $a^1, b^1, \ldots, a^k, b^k$, $a^1 \leq b^1 < a^2 \leq b^2 < \ldots < a^k \leq b^k$, and permutation π of $\{1,\ldots,n\}$, such that for every n-bit vector $\boldsymbol{x} \in \{0,1\}^n$ we get $f(\boldsymbol{x}) = 1$, if and only if $x^\pi \in \cup_{i=1}^{k}[a^i, b^i]$. The class of k-interval functions will be denoted by $\mathcal{C}_{k-\mathrm{int}}$. The class of positive (negative, resp.) k-interval functions will be denoted by $\mathcal{C}^+_{k-\mathrm{int}}$ ($\mathcal{C}^-_{k-\mathrm{int}}$, resp.).*

Note that the preceding definition ensures that for any k, l such that $l < k$, the class of k-interval functions (properly) includes the class of l-interval functions.

Definition 2.5. *Let f be a positive Boolean function on n variables $x_1, \ldots x_n$ and let x_1 and x_2 be two of its variables. We say that x_1 has* **equal or greater strength** *than x_2 in f, if for any evaluation e of variables $x_3, \ldots x_n$ it holds, that $f(0,1,e(x_3),\ldots,e(x_n)) \leq f(1,0,e(x_3),\ldots,e(x_n))$. We denote it by $x_1 \succeq x_2$. If $x_1 \succeq x_2$ and $x_2 \succeq x_1$ we say that x_1 and x_2 have* **equal strength** *in f and denote it by $x_1 \sim x_2$. If $x_1 \succeq x_2$ and not $x_2 \succeq x_1$ we say that x_1 is* **stronger** *than x_2 in f and denote it by $x_1 \succ x_2$.*

Positive Boolean function f on n variables is called a **regular function**, if it holds that $x_1 \succeq x_2 \succeq \ldots \succeq x_n$ in f. The class of regular functions will be denoted by $\mathcal{C}_{\text{reg}}$. Positive Boolean function f on n variables is called a **2-monotonic function**, if there exists a linear ordering of its variables, which respects their strength, i.e., it is possible to rename variables in f to get a regular function. The class of 2-monotonic functions will be denoted by $\mathcal{C}_{2\text{m}}$.

Boolean function $f : \{0,1\}^n \mapsto \{0,1\}$ on variables $x_1, \ldots x_n$ is called a **threshold function**, if there exist weight function $\omega : \{x_1, \ldots, x_n\} \to \mathbf{R}$ and real number t, such that for any vector $\boldsymbol{x} \in \{0,1\}^n$ it holds, that $f(\boldsymbol{x}) = 1$, if and only if $\omega(\boldsymbol{x}) \geq t$, where $\omega(\boldsymbol{x}) = \sum_{i=1}^{n} x_i \omega(x_i)$. A pair (ω, t) satisfying such equivalence for a threshold function f is then called a **threshold structure** for f. A threshold structure (ω, t) is positive, if ω assigns only non-negative weights. The class of threshold functions will be denoted by $\mathcal{C}_{\text{th}}$, the class of positive threshold functions by $\mathcal{C}_{\text{th}}^{+}$. It is easy to observe, that threshold function f is positive, if and only if there is a positive threshold structure for f. Negative threshold functions and negative threshold structures are defined similarly.

2.4 List of hard problems

In this part we present the chosen NP-hard and co-NP-hard problems, that will be used in this thesis for proving that other problems are NP-hard or co-NP-hard. The proofs of NP-hardness or co-NP-hardness of the chosen problems, as well as many other hard problems, can be found in [13].

We have already defined in Subsection 2.1 the FALSIFIABILITY problem, which is NP-complete.

Another problem considers finding a **minimum vertex cover** in a given graph. We recall that a vertex cover of graph $G = (V, E)$ is a subset C of its vertices such that for every edge $e = (u, v)$ in E at least one of its vertices is in C, that is $u \in C \vee v \in C$. The size of minimum vertex cover of graph G is usually denoted by $\tau(G)$.

MIN-VERTEX-COVER	
Instance :	An undirected graph G and $k \in \mathbf{N}$.
Question :	Is there a vertex cover C of graph G such that $\lvert C \rvert \leq k$?

This is a decision version of the problem and it is NP-complete. It is known (see e.g. [2]) that when the decision version of a problem is NP-hard then so are also the evaluation version (we are looking for the size of the minimum vertex cover) and the optimization version (we are looking for the minimum cover itself).

The problem of deciding, whether a given DNF $\mathcal{F}$ represents a **tautology**, that is, whether it is 1 for every evaluation of its variables, is co-NP-hard (this is a complementary problem to FALSIFIABILITY).

TAUTOLOGY	
Instance :	A DNF $\mathcal{F}$ representing Boolean function f on n variables.
Question :	Does for every Boolean vector $\boldsymbol{x}$ of length n hold that $f(\boldsymbol{x}) = 1$?

3 Representations of Boolean Functions and Their Properties

In this section we present several known results, which capture important properties of Boolean functions and their traditional representations. These properties will be used later. We also show here basic properties of interval representations of Boolean functions including several new results.

3.1 DNF representation

In this dissertation we study many problems concerning interval representations of Boolean functions first for the classes of positive (and negative) functions. We will need several known facts about positive functions and DNFs representing them. All of them can be easily transformed into facts about negative functions.

Lemma 3.1. *Every prime implicant of a positive function is a positive term.*

Proof : Let f be a positive function and t its prime implicant. Let $\mathcal{F}$ be a positive DNF representing f. Suppose for a contradiction that t contains negative literal $\overline{x}$, i.e. $t = t' \wedge \overline{x}$, where t' is a term. Let $\boldsymbol{u}$ be the vector, which satisfies t, i.e., $t(\boldsymbol{u}) = 1$, and which has zeros in every bit corresponding to the variables not present in t. It follows that the bit of $\boldsymbol{u}$ corresponding to the variable x is set to 0. This vector $\boldsymbol{u}$ must also satisfy $\mathcal{F}$. But $\mathcal{F}$ is a positive DNF, therefore $\mathcal{F}$ must also be satisfied by vector $\boldsymbol{v}$ formed from $\boldsymbol{u}$ by switching the bit corresponding to the variable x from 0 to 1. Hence also $t' \wedge x$ is an implicant of f, because any vector satisfying term $t' \wedge x$ is a trueipoint of f. Hence we have $t = t' \wedge \overline{x} \leq f$ and $t' \wedge x \leq f$, which implies that also $(t' \wedge \overline{x}) \wedge (t' \wedge x) = t' \leq f$, i.e. t' is an implicant of f. This contradicts the assumption that t is a prime term. ■

Lemma 3.2. *Every positive function f has a unique prime and irredundant DNF representation.*

Proof : Since there is no consensus possible among positive terms, it follows using Quine's theorem ([22]), that the DNF formed by all prime implicants of f is irredundant and also that any other prime DNF does not represent f, because from any subset of all prime implicants, which are positive according to Lemma 3.1, we cannot generate any other prime implicant. ■

We can observe from the definition of positive functions, that their class is closed under partial assignment, because by such assignment into the positive DNF representation we again get a positive DNF. Thus the resulting function is also positive.

3.2 Partially defined Boolean functions

The concept of a partially defined Boolean function (pdBf) [10, 5] is a natural generalization of the concept of a totally defined Boolean function, by allowing that the function values on some input vectors are unknown. PdBfs have many applications, in particular in computer science and knowledge engineering. For example, a classical application of pdBfs is in the design of switching circuits. A customary method in that field is to specify the inputs, on which the circuit must output 1, and the inputs, on which it must output 0. The output on the remaining inputs remains unspecified and is considered as "don't care".

Another application of pdBfs is the representation of incomplete information about cause-effect relationships [10]. Several problems involving pdBfs also originated from the area of logical analysis of data (LAD). The usefulness of Boolean techniques in this field has been demonstrated in many settings (e.g., [23, 1, 11]). Typically, we are in a situation, when some collection of data has been measured with relation to the occurence of a certain phenomenon. This means, that we are given a set of measurements capturing the conditions, under which the studied phenomenon did occur (the set of positive samples), and another set of measurements, when the phenomenon did not occur (the set of negative samples). Using this data, we should analyze, under which conditions does this phenomenon occur in general. The output of our analysis should be a function, which based on given conditions predicts, whether the phenomenon will occur or not also for vectors, which were not in the "training sample". The basic requirement on such a function is that it should agree with the measured data.

Usually, in such experiments we expect the explaining function to have some properties. For example, in case we measure the occurence of a disease based on patient's blood pressure and patient's temperature, we can expect that with a rising temperature the disease tends to occur more often, so the function should be positive in the input variable corresponding to temperature.

The sets of positive and negative samples can be binarized (see [4]) and then we can view these measurements as a pdBf (T, F), where T represents the binarized positive samples and F represents the binarized negative ones. The explaining function is then an extension of this pdBf. If we require the extension to have some properties (such as positivity), we search for an extension in a given class of Boolean functions corresponding to these properties.

As we have already mentioned before, an extension of pdBf (T, F) exists, if and only if $T \cap F = \emptyset$. This can be decided in asymptotically optimal time $O(n \cdot (|T| + |F|))$, where n is the length of vectors in $T \cup F$, for example by the following algorithm based on the *RadixSort* procedure:

Algorithm 3.3. Extension($\mathcal{C}_{\text{all}}$)

Input: *A pdBf (T, F) on n variables.*
Output: *DNF $\mathcal{F}$ representing an extension of pdBf (T, F), if one exists.* **NO** *otherwise.*

1: let V be the sequence of all vectors from T and F
2: let $k := |V|$
3: sort sequence V lexicographically using *RadixSort*, let the resulting ordering be $\boldsymbol{v}^1, \ldots, \boldsymbol{v}^k$

4: **for** $i := 1$ **to** $k-1$ **do**
5: **if** $\boldsymbol{v}^i = \boldsymbol{v}^{i+1}$ **then output NO endif**
6: **enddo**
7: **output** DNF $\mathcal{F}$ formed by $|T|$ terms of length n, where each term is satisfied exactly by one truepoint, i.e., $\boldsymbol{u} \in T$

Since *RadixSort* can be implemented to run in $O(n \cdot (|T| + |F|))$ time, the entire algorithm also runs in this time. Moreover, Algorithm 3.3 is asymptotically optimal, because the size of the input pdBf is $n \cdot (|T| + |F|)$ and every correct deterministic algorithm must at least read its whole input. We note, that the extension function represented by the output of Algorithm 3.3 is the one with the minimum set of truepoints among all extensions of the input pdBf.

Using the same idea we can also solve the best-fit problem for the class of all Boolean functions. In this case we are forced to make error only in case of vectors, which are contained both in T and in F. Hence, for all such pairs we classify incorrectly the vector, which is less significant (with respect to the weight function). Again, we employ *RadixSort*.

Algorithm 3.4. Best-Fit($\mathcal{C}_{\text{all}}$)

Input: *A pdBf (T, F) on n variables and weight function $\omega : T \cup F \mapsto \mathbf{R}_+$.*
Output: *DNF $\mathcal{F}$ representing a Boolean function, for which the weight of incorrectly classified vectors is minimum.*

1: let V be the sequence of all vectors from T and F
2: let $k := |V|$
3: let $T^* := T$ and $F^* := F$.
4: sort sequence V lexicographically using *RadixSort*, let the resulting ordering be $\boldsymbol{v}^1, \ldots, \boldsymbol{v}^k$
5: **for** $i := 1$ **to** $k-1$ **do**
6: **if** $\boldsymbol{v}^i = \boldsymbol{v}^{i+1}$ **then**
7: **if** $\omega(\boldsymbol{v}^i) \geq \omega(\boldsymbol{v}^{i+1})$ **then** remove $\boldsymbol{v}^{i+1}$ from T^* or F^* **endif**
8: **if** $\omega(\boldsymbol{v}^i) < \omega(\boldsymbol{v}^{i+1})$ **then** remove $\boldsymbol{v}^i$ from T^* or F^* **endif**
9: $i := i + 1$.
10: **endif**
11: **enddo**
12: **output** DNF $\mathcal{F}$ formed by $|T^*|$ terms of length n, where each term is satisfied exactly by one truepoint, i.e., $\boldsymbol{u} \in T^*$

This algorithm can again be implemented to run in $O(n \cdot (|T| + |F|))$ time which is asymptotically optimal.

Many results concerning the extension and best-fit problems were solved in [5]. We cite here explicitly the following results, which are related to the classes of Boolean functions, which we study in this dissertation (other results can be found e.g., in [10, 3, 20]):

Theorem 3.5. *The* EXTENSION *problem can be solved in polynomial time for the classes $\mathcal{C}^+$, $\mathcal{C}^-$, $\mathcal{C}_{\text{reg}}$, $\mathcal{C}_{2\text{m}}$, $\mathcal{C}_{\text{th}}$ and $\mathcal{C}_{k-\text{DNF}}$ for fixed k. It is NP-complete for the class $\mathcal{C}_{k-\text{DNF}}$, when k is a part of input.*

Theorem 3.6. *The* BEST-FIT *problem can be solved in polynomial time for the classes $\mathcal{C}^+$, $\mathcal{C}^-$ and $\mathcal{C}_{\text{reg}}$ and it is NP-hard for the classes $\mathcal{C}_{2\text{m}}, \mathcal{C}_{\text{th}}$ and $\mathcal{C}_{k-\text{DNF}}$.*

It is also an interesting problem to search for an extension, which is optimal with respect to a certain criteria. For instance, we studied the problem of finding the extension with the shortest DNF representation among all the extensions of the input pdBf. A similar problem of finding the shortest DNF representation of a given function is known as Boolean function minimization, which is NP-hard, since it can be easily observed, that FALSIFIABILITY is its special case. Moreover, in [16] it was proved, that minimization is NP-hard also for the class of Horn functions, for which FALSIFIABILITY can be solved in polynomial time. In agreement with these results the following theorem shows, that also our problem is probably intractable for all but trivial classes of Boolean functions. The proof of this theorem is a slight generalization of the same result shown in [20] for the class of Horn functions.

SHORTEST-EXTENSION(C)	
Instance :	A pdBf (T, F) and $k \in \mathbf{N}$.
Question :	Is there an extension of (T, F) from class C of Boolean functions which has a DNF representation $\mathcal{F}$ such that $\|\mathcal{F}\|_l \leq k$?

Theorem 3.7. *It is NP-hard to decide the* SHORTEST-EXTENSION*(C) problem, whenever C contains the class of positive or negative 1-DNF functions, that is functions that can be represented by a positive or negative DNF with only linear terms.*

Proof : We prove the hardness by reducing from MIN-VERTEX-COVER problem defined in Subsection 2.4. We assume that $\mathcal{C}$ contains all positive 1-DNF functions, the case, when $\mathcal{C}$ contains all negative 1-DNF functions, is similar. Let $G = (V, E)$ be an undirected graph where $V = \{1, \ldots, n\}$. Let us define $T, F \subseteq \{0,1\}^n$ as follows:

$$T = \{\boldsymbol{x}^A | A = \{i, j\}, \{i, j\} \in E\}$$
$$F = \{\boldsymbol{e} = (0 \ldots 0)\} \ .$$

We claim that there is a DNF $\mathcal{F}$ representing a function f from class C such that f extends (T, F) and $|\mathcal{F}|_l \leq k$, if and only if the size of a minimal vertex cover $\tau(G) \leq k$. This will complete the proof.

Let $\mathcal{F} = \bigvee_{l \in L} t_l$, where $t_l = \bigwedge_{i \in P_l} x_i \bigwedge_{j \in N_l} \overline{x}_j$, be a DNF representing function f from C and extending (T, F) such that $|\mathcal{F}|_l \leq k$. Then for every $l \in L$ the following conditions hold, or $\mathcal{F}$ can be transformed (as described below) to a shorter DNF $\mathcal{F}'$, which also represents an extension of (T, F) and for which these conditions hold:

(a) $P_l \neq \emptyset$, otherwise $t_l(\boldsymbol{e}) = 1$.

(b) $|P_l| = 1$, otherwise we can remove all but one (arbitrary) positive literal from t_l and the resulting DNF $\mathcal{F}'$ still represents an extension of (T, F). This is true, because $|P_l| \leq 2$, otherwise no vector $\boldsymbol{a} \in T$ satisfies t_l, and after removing one of the two positive literals from term t_l the resulting term t'_l is still not satisfied by falsepoint $\boldsymbol{e}$ and $t_l \leq t'_l$ holds.

(c) $N_l = \emptyset$, since replacing N_l by $\emptyset$ produces a shorter term t'_l, it follows that $t_l \leq t'_l$ holds, hence $t'_l(\boldsymbol{a}) = 1$ for all $\boldsymbol{a} \in T$ for which $t_l(\boldsymbol{a}) = 1$, and still $t'_l(\boldsymbol{e}) = 0$.

These conditions imply, that $\mathcal{F}$ is, or can be transformed into, a positive 1-DNF representing an extension of (T, F). Furthermore, since $\mathcal{F}(\boldsymbol{a}) = 1$ for every $\boldsymbol{a} \in T$, for every such $\boldsymbol{a}$ there must exist $l \in L$, such that $t_l(\boldsymbol{a}) = 1$ (i.e., $P_l \subset ON(\boldsymbol{a}) = \{i, j\}$ for the corresponding edge $\{i, j\} \in E$). Hence $C = \cup_{l \in L} P_l$ is a vertex cover of G such that $|C| = |\mathcal{F}|_l = |\mathcal{F}|_t \leq k$.

Conversely, if C is a vertex cover with $|C| \leq k$, then $\mathcal{F}_C = \bigvee_{i \in C} x_i$ is a positive 1-DNF extending (T, F), such that $|\mathcal{F}_C|_l = |\mathcal{F}_C|_t \leq k$. ■

From the proof of Threorem 3.7 we can also see, that the SHORTEST-EXTENSION problem does not become any easier, if we modify it to search for a function represented by a DNF with the minimum number of terms.

3.3 Interval representation

The motivation to study interval representations of Boolean functions comes from the research area of hardware verification ([8], [18]) and software testing ([12]). In a situation, when we are given a source code of some program, we want to test that its outputs are correct. There are many problems, that must be solved, to allow efficient automated testing, e.g., we must define what the correct output for a given input is. Another problem is how to generate inputs, for which we should execute the program and verify its output. Usually, we cannot generate all possible inputs, so we want to generate such a set of inputs, which verifies a substantial part of the program being tested. One of possible approaches is to generate a set of inputs, which ensures, that every part of the source code is executed on at least one the generated inputs.

More powerful, but also more time-consuming, is to generate a set of inputs, such that every path in a program is verified, where a path corresponds to a path in a directed graph called flowgraph constructed from the source code of the program as follows: the vertices of flowgraph are the blocks of code, which are always executed as a whole from its beginning to its end in the same way. That means, that such a block does not contain any cycles, conditions and jump statements, and also, it is not possible to jump in the middle of it from somewhere else. Edges of flowgraph correspond to possible transitions between blocks of code and are labeled with conditions, which must be satisfied to allow the corresponding transitions to occur.

A typical generator of test data then operates in the following way: its input is a source code, which is first analyzed and there are two outputs of this analysis - a flowgraph and data dependencies, which are used to keep track of what data is contained in which variable and which specify the connection between actual value of a variable and the values in an input of the program. The flowgraph is then used by path selector to generate the set of paths, which should be tested. For every such path a predicate is constructed by combining (using logical AND) all the conditions associated with the flowgraph edges on that path. This predicate then specifies the conditions on input data, which must be met to force the program in test to follow the chosen path.

The last step in test data generating is solving the predicate to get the actual input data. Because there can be various and very complex conditions involved, solvers usually require some canonical representation of the predicate. One of possible representations is a DNF. Therefore, we must be able to convert all possible kinds of conditions to a DNF representation. One type of a condition, which typically occurs, is that value of some input variable must belong to an interval. Of course, by producing shorter DNF representation of conditions, the predicate can be solved more efficiently. In [24] the problem of minimal DNF representations of 1-interval functions has been solved. This motivated our research of interval representations.

When Boolean function f has an interval representation consisting of a small number of intervals, then the value of f on any input can be very efficiently computed, e.g., using interval trees (see [9]). Also, this representation is compact and hence the construction of minimal interval representations can be considered a variant of Boolean function minimization and can be used for knowledge compression. Therefore we consider interesting the problem of finding minimal interval representations and our research effort has been aimed at the classes of Boolean functions, for which it is possible to solve this problem in polynomial time.

When studying interval representations of Boolean functions it is very helpful to view the situation using graphical information. A suitable tool for this is a **branching tree**, which is a complete binary tree with edges evaluated with values 0 and 1. Such tree displays all bit vectors of the length equal to the depth of the tree. Every path from the root to a leaf corresponds to one vector and we can identify each leaf with the integer value corresponding to this vector, where the significance of the bits decreases from the root to the leaf. An example of such tree for the case of three bits is on Figure 1.

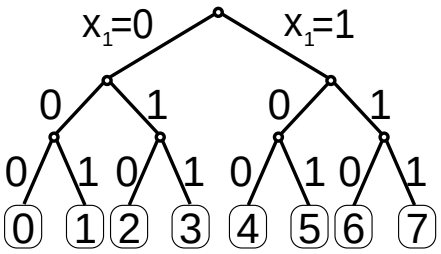


Figure 1: *A branching tree on 3 variables.*

When f is a Boolean function on n variables, which can be represented by interval $[a, b]$ with respect to ordering π of variables, then it means, that if we consider a branching tree of depth n and evaluate the edges incident to the root by the most significant variable $x_{\pi^{-1}(1)}$, the edges adjacent to these edges by the second most significant variable $x_{\pi^{-1}(2)}$ and so on and the edges adjacent to leaves by the least significant variable $x_{\pi^{-1}(n)}$, then f is 1 exactly on those bit vectors, which correspond to the leaves representing numbers from $[a, b]$.

Example 3.8. *On Figure 2 we have a function on three variables, which can be represented by interval* $[3, 7]$ *with respect to the ordering* x_1, x_2, x_3 *of its variables. This function can also be represented by DNF* $x_1 \vee x_2x_3$.

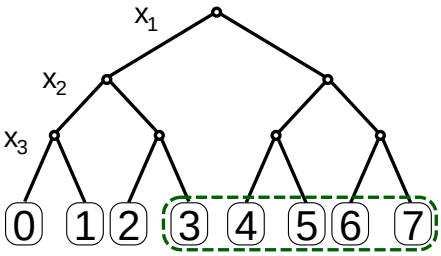


Figure 2: *A branching tree of the function represented by the DNF $x_1 \vee x_2x_3$, the interval of truepoints is circled by a dashed line.*

Example 3.9. *As the next example we consider function f represented by DNF $x_1x_2 \vee x_2x_3 \vee x_1x_3$. This function cannot be represented by one interval with respect to any ordering of its variables. The reason is that in every ordering (this function is symmetrical with respect to its variables, therefore we can consider just one ordering and it can be an arbitrary one) f has value 1 on vectors 011 and 101, but its value is 0 on vector 100. These three vectors prevent the function from being 1-interval. The situation is visualized using a branching tree on Figure 3.*

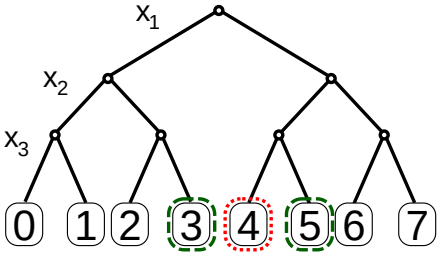


Figure 3: *A branching tree of the function represented by the DNF $x_1x_2 \vee x_2x_3 \vee x_1x_3$, two of the truepoints are circled by a dashed line and one of the falsepoints by a dotted line.*

A basic property of the interval representation of a function is the number of intervals. The following theorem shows the upper bound:

Theorem 3.10. *Any Boolean function on n variables has at most 2^{n-1} intervals. Moreover, this bound is tight.*

Proof : It is obvious that we get the most intervals by alternating zeros and ones on all neighbouring vectors in the given ordering. In $[0, 2^n - 1]$ this means we can create 2^{n-1} intervals. We can easily observe, that the PARITY$_n$ function does exactly this kind of alternating, starting with 0 in $00\ldots0$. Also, PARITY is symmetrical with respect to the ordering of its variables, hence PARITY$_n \in \mathcal{C}_{2^{n-1}-int} \setminus \mathcal{C}_{(2^{n-1}-1)-int}$. ■

However, the PARITY function does not have a short DNF representation either, the shortest DNF representing PARITY$_n$ has 2^{n-1} terms. The reason for this is, that every term in any DNF representation of PARITY$_n$ must have length n, because the value of PARITY$_n$ on any input depends on all variables (by changing the value

of any input bit we change the function value). Hence every term is satisfied by exactly one truepoint of PARITY_n. And since PARITY_n has 2^{n-1} truepoints, every DNF representation must have 2^{n-1} terms.

This motivates an interesting question, whether there is a function which has a short DNF representation and can only be represented by an exponential number of intervals (for every ordering of variables) with respect to the size of the DNF. The following example proves an affirmative answer:

$$\mathcal{F} = x_1 x_2 \dots x_{\frac{n}{2}} \vee x_{\frac{n}{2}+1} \dots x_n \ . \tag{3.1}$$

It has two terms, which will be denoted by t_1 and t_2, and n literals. We will show that it has an exponential number of intervals with respect to any ordering of variables.

Theorem 3.11. *The function f specified by DNF (3.1) is represented by $\Omega(2^{\frac{n}{2}})$ intervals in any ordering of its variables.*

Proof : Let permutation $\pi : \{1 \dots n\} \to \{1 \dots n\}$ be given specifying the ordering of variables, with respect to which we consider the number of intervals. We will show that there are at least $2^{\frac{n}{2}} - n - 1$ truepoints, which (with respect to ordering π) represent integers, which both increased and decreased by one (i.e., to form the neighbours in a branching tree) correspond to falsepoints of f. This will prove our bound.

We will assume that $\pi(n) = n$. This is just a technical detail, if π did not have this property, we would be able to rename variables of DNF (3.1) and possibly also terms t_1 and t_2 to achieve this. Let us define the set of vectors we shall count:

$$X_\pi = \{\boldsymbol{x} \mid 0 < x^\pi < 2^n - 1 \ \&\ f(\boldsymbol{x}) = 1 \ \&\ f(\overrightarrow{(x^\pi - 1)^{\pi^{-1}}}) = f(\overrightarrow{(x^\pi + 1)^{\pi^{-1}}}) = 0\}$$

We shall denote $|X_\pi|$ by N_π. Let us show that $\boldsymbol{x}$ belongs to X_π, if $t_1(\boldsymbol{x}) = 0$ and $t_2(\boldsymbol{x}) = 1$ holds and $\boldsymbol{x}$ has at least two zero bits. Let us fix one such $\boldsymbol{x}$. Its neighbours in a branching tree are

$$\boldsymbol{y} = \overrightarrow{(x^\pi - 1)^{\pi^{-1}}} \text{ and } \boldsymbol{z} = \overrightarrow{(x^\pi + 1)^{\pi^{-1}}} \ .$$

Because in $\boldsymbol{x}$ bit x_n must be 1 (because $t_2(\boldsymbol{x}) = 1$), vector $\boldsymbol{y}$ is formed from $\boldsymbol{x}$ by switching x_n to 0. Hence $f(\boldsymbol{y}) = 0$. Vector $\boldsymbol{z}$ is formed from $\boldsymbol{x}$ by switching some least significant bits from 1 to 0 (at least x_n is switched) and by switching one zero bit to 1. Because $\boldsymbol{x}$ has at least two zero bits, vector $\boldsymbol{z}$ has at least one more zero bit among variables of t_1. Hence we have also $f(\boldsymbol{z}) = 0$.

The number of vectors satisfying the conditions of the equivalent definition of X_π is $2^{\frac{n}{2}} - n - 1$, because the values of variables $x_{\frac{n}{2}+1}, \dots, x_n$ are fixed to 1 and the values of the variables $x_1, \dots, x_{\frac{n}{2}}$ can be chosen arbitrarily except the case, when there is no or only one value 0 among them. Hence $N_\pi = \Omega(2^{\frac{n}{2}})$. ∎

The following lemma states a basic property of positive k-interval functions and we shall use it implicitly throughout this whole thesis:

Lemma 3.12. *Let f be a positive k-interval function, where $k \geq 1$, on n variables, which is not a $(k-1)$-interval function and which is represented by intervals $[a^1, b^1] < \dots < [a^k, b^k]$. Then $b^k = 2^n - 1$.*

Proof : Clearly, if f is positive (and not identically equal to 0, which is implied by the assumption, that f is not a $(k-1)$-interval function), $f(11\ldots1) = 1$ must hold. ■

As any other result related to positive k-interval functions, there is a "mirror" version of Lemma 3.12 for negative functions stating, that $a^1 = 0$. We shall frequently state only the positive version of such lemmas and suppose implicitly, that it can be transformed into the negative version.

Another property of interval representation, which we will use later, is that by a partial assignment to any function we cannot increase the number of intervals, which it can be represented by in any ordering.

Lemma 3.13. *Let f be a Boolean function on n variables and let π be an ordering of its variables, with respect to which f can be represented by k intervals. Then the function $f' = f[x_i := c]$, where $i \in \{1,\ldots,n\}$ and $c \in \{0,1\}$, can be represented by k or less intervals with respect to ordering π' formed from π by restriction on $\{1,\ldots,i-1,i+1,\ldots,n\}$.*

Proof : The high-level idea of this lemma is, that when we fix the value of variable x_i to c, then the branching tree corresponding to f and ordering π loses one level and the intervals of zeros and ones "shrink" and maybe merge, because some of these values are left out. However, no new interval can be introduced in this process.

Formally, we will proceed by a contradiction. Suppose that f' cannot be represented by k or less intervals with respect to ordering π'. Then there are $2k+1$ vectors $\boldsymbol{a}_1,\ldots,\boldsymbol{a}_{k+1}$,$\boldsymbol{b}_1,\ldots,\boldsymbol{b}_k$ of $(n-1)$ bits with the following properties:

1. $\forall j \in \{1,\ldots,k+1\} : f'(\boldsymbol{a}_j) = 1$
2. $\forall j \in \{1,\ldots,k\} : f'(\boldsymbol{b}_j) = 0$
3. $\forall j \in \{1,\ldots,k\} : a_j^{\pi'} < b_j^{\pi'} < a_{j+1}^{\pi'}$.

These vectors prove that f' cannot be represented by k or less intervals with respect to ordering π'. However, if we insert the value c as the i-th bit in every vector $\boldsymbol{a}_j$ and $\boldsymbol{b}_l$, we get $2k+1$ vectors of n bits proving that f cannot be represented by k or less intervals with respect to the ordering π. This is the desired contradiction. ■

From Lemma 3.13 and from the fact (we have mentioned before), that positive and negative functions are closed under partial assignment, we have the following corollary:

Corollary 3.14. *For any $k \in \{0,\ldots\}$ the classes $\mathcal{C}_{k-\mathrm{int}}$, $\mathcal{C}^+_{k-\mathrm{int}}$ and $\mathcal{C}^-_{k-\mathrm{int}}$ are closed under partial assignment.*

4 Relations of Threshold and k-Interval Boolean Functions

In this section we study relations between the classes of positive functions, which can be represented by a limited number of intervals, and the class of positive threshold functions. This research has quite a natural motivation, because it can be easily seen,

that positive 1-interval functions constitute a proper subclass of positive threshold functions. Therefore we tried to find out, how this relation evolves with an increasing number of representing intervals.

We start by presenting the mentioned result, that positive 1-interval functions constitute a proper subclass of positive threshold functions. Then we generalize a certain idea of its proof to show, that positive threshold functions do not constitute a subclass of (positive) k-interval function, for arbitrary k. In the following subsection we show, that every positive 2-interval function is also threshold. We complete the description of the relations between the classes of positive threshold, positive k-interval and 2-monotonic functions by proving, that there is a positive 3-interval function, which is not 2-monotonic and hence also not threshold (see [21]).

4.1 Relations of positive threshold and positive 1-interval functions

We start with an easy result, which will serve as a motivation for further results of this section.

Lemma 4.1. $\mathcal{C}^+_{1-\text{int}} \subset \mathcal{C}^+_{\text{th}}$

Proof : First, we show that every positive interval function f is a positive threshold function. Let π be the permutation of variables, with respect to which f can be represented by interval $[a, 2^n - 1]$. We define the weight $\omega(x_i)$ of variable x_i to be $2^{n-\pi(i)}$ and set the threshold value to a. Then for every truepoint $\boldsymbol{x} \in \{0,1\}^n$ we have $\sum_{i=1}^n x_i\omega(x_i) = x^\pi < a$ and for every falsepoint $\boldsymbol{x}$ we have $\sum_{i=1}^n x_i\omega(x_i) \geq a$. Therefore (ω, a) is a positive threshold structure for f and hence f is a positive threshold function.

To show, that not all positive threshold functions are positive interval functions, we can take function f represented by positive DNF $x_1x_2 \vee x_2x_3 \vee x_1x_3$. By setting $\omega(x_1) = \omega(x_2) = \omega(x_3) = 1$ and $t = 2$, we obtain a positive threshold structure for f. However, we have already shown in Example 3.9, that f is not an interval function. ■

Since every positive threshold function is also a 2-monotonic function and, moreover, this is not true conversely (see [21]), we now also know, that positive interval functions constitute a proper subclass of 2-monotonic functions.

Corollary 4.2. $\mathcal{C}^+_{1-\text{int}} \subset \mathcal{C}^+_{\text{th}} \subset \mathcal{C}_{2\text{m}}$

4.2 Relations of positive threshold and positive k-interval functions

In the proof of Lemma 4.1 we used the function represented by DNF $x_1x_2 \vee x_2x_3 \vee x_1x_3$ to show, that there is a positive threshold function, which cannot be represented by one interval. This function is exactly MAJORITY_3. In the next theorem we generalize this approach and use MAJORITY_n as an example of a positive threshold function, which cannot be represented by a small number of intervals. This result will then be used to easily derive a corollary stating, that positive threshold functions do not constitute a subclass of k-interval functions for any k.

Theorem 4.3. *The* MAJORITY$_n$ *function on n variables is a positive threshold function and cannot be represented by* $2^{\lfloor n/2 \rfloor} - 2$ *or less intervals in any ordering of its variables.*

Proof : A threshold structure with all weights of variables set to 1 and the threshold set to $\lfloor n/2 \rfloor + 1$ proves that MAJORITY$_n$ is a positive threshold function.

Now we will prove the second part of the theorem. First of all, the MAJORITY$_n$ function is symmetrical with respect to its variables, hence we only need to consider one (arbitrary) ordering of its variables. Let us take the natural ordering $x_1, x_2, \ldots, x_n$.

To determine the number of intervals needed to represent MAJORITY$_n$, we shall count the right boundaries of these intervals. One of them is, naturally, the vector of all ones. The other ones are identified by n-bit numbers x, binary representations of which have at least $\lfloor n/2 \rfloor + 1$ bits with value 1 and their successors, $x + 1$, have binary binary representation with at most $\lfloor n/2 \rfloor$ bits with value 1. The set

$$X_n(l) = \{x \mid 0 < x < 2^n - 1 \ \& \ |ON(\boldsymbol{x})| = l \ \& \ |ON(\overrightarrow{x+1})| \leq \lfloor n/2 \rfloor\}$$

contains all such numbers having exactly l bits with value 1 in their binary representation. Let us denote $|X_n(l)|$ by $x_n(l)$.

Now we shall count $x_n(l)$. Adding 1 to some n-bit number $x < 2^n - 1$ replaces the last (least significant) bit with value 0 in its binary representation by 1 and all subsequent (less significant) bits are switched from 1 to 0. Hence the numbers x, that we count in $x_n(l)$, must have binary representations such that the last $l - \lfloor n/2 \rfloor + 1$ bits have value 1 and the other bits can be an arbitrary combination of $\lfloor n/2 \rfloor - 1$ ones and $n - l$ zeros. This implies that for any such x, integer $x + 1$ has at most $l - (l - \lfloor n/2 \rfloor + 1) + 1 = \lfloor n/2 \rfloor$ bits with value 1. Therefore

$$x_n(l) = \binom{n - (l - \lfloor n/2 \rfloor + 1)}{\lfloor n/2 \rfloor - 1} = \binom{n - l + \lfloor n/2 \rfloor - 1}{\lfloor n/2 \rfloor - 1}$$

To count the number of right boundaries of intervals representing MAJORITY$_n$ we need to sum $x_n(l)$ for l ranging from $\lfloor n/2 \rfloor + 1$ (because there have to be enough bits with value 1) to $n - 1$ (because we are not counting $2^n - 1$). Therefore m_n, the number of the right boundaries of inner intervals (i.e., not having their right boundary in $2^n - 1$), is

$$m_n = \sum_{l=\lfloor n/2 \rfloor + 1}^{n-1} x_n(l) = \sum_{l=\lfloor n/2 \rfloor + 1}^{n-1} \binom{n - l + \lfloor n/2 \rfloor - 1}{\lfloor n/2 \rfloor - 1}$$

This can be simplified using a substitution $i = n - 1 - l$ as follows (note, that we sum in the reverse order):

$$m_n = \sum_{i=0}^{\lceil n/2 \rceil - 2} \binom{\lfloor n/2 \rfloor + i}{\lfloor n/2 \rfloor - 1} = \sum_{i=0}^{\lceil n/2 \rceil - 2} \binom{\lfloor n/2 \rfloor + i}{i + 1}$$

For our purpose we shall use a very simple approximation as follows:

$$m_n \geq \sum_{i=0}^{\lfloor n/2 \rfloor - 2} \binom{\lfloor n/2 \rfloor}{i + 1} = \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{\lfloor n/2 \rfloor}{j} - \binom{\lfloor n/2 \rfloor}{0} - \binom{\lfloor n/2 \rfloor}{\lfloor n/2 \rfloor} = 2^{\lfloor n/2 \rfloor} - 2 \quad (4.1)$$

The first inequality follows from the obvious fact, that $\binom{p+q}{r} \geq \binom{p}{r}$ for any non-negative integers, and the last equality is derived using the binomical theorem.

Due to (4.1), the MAJORITY$_n$ function cannot be represented by $2^{\lfloor n/2 \rfloor} - 2$ intervals or less. ■

Corollary 4.4.

(a) $\mathcal{C}^+_{\text{th}} \not\subseteq \mathcal{C}^+_{k-\text{int}}$ *for any* k.

(b) $\mathcal{C}^+_{\text{th}} \not\subseteq \mathcal{C}_{k-\text{int}}$ *for any* k.

4.3 Relation of positive threshold and positive 2-interval functions

In this subsection we shall gradually construct the proof that positive 2-interval functions constitute a subclass of positive threshold functions. Due to Corollary 4.4, this inclusion is proper.

We start by stating a sequence of rather technical lemmas, which describe basic properties of positive k-interval functions. We formulate these lemmas for general k, although we will need them later only for the case $k = 2$. A corollary of these properties will show, that in order to prove the desired inclusion between positive threshold and positive 2-interval functions, we can consider only those positive 2-interval functions, for which there is no index i satisfying at least one of the following conditions:

$$\forall \boldsymbol{x}(f(\boldsymbol{x}) = 1 \quad \Rightarrow \quad x_i = 1) \tag{4.2}$$

$$\forall \boldsymbol{x}(f(\boldsymbol{x}) = 0 \quad \Rightarrow \quad x_i = 0) \tag{4.3}$$

i.e., all truepoints or all falsepoints have the same value of i-th bit. We note, that for any positive function f it is not possible for all truepoints to contain 0 in i-th bit, for any i, because when $\boldsymbol{x}$ is a truepoint of f such that $x_i = 0$, then $\boldsymbol{x}'$, formed from $\boldsymbol{x}$ by switching bit x_i to 1, is also a truepoint of f, because f is positive. Similarly, it is not possible for all falsepoints of f to contain 1 in i-th bit, for any i.

Lemma 4.5. *Let f be a positive k-interval function on n variables. Let i be an index $1 \leq i \leq n$, such that at least one of the conditions (4.2), (4.3) is satisfied. Then there is ordering π, with respect to which f can be represented by l intervals $[a^1, b^1] < \ldots < [a^l, 2^n - 1]$, where $l \leq k$, and such that $\pi(i) = 1$, i.e., x_i is the most significant variable with respect to π. Moreover, for any such ordering π it holds, that $a^1_1 = a^2_1 = \ldots = a^l_1$, and $a^1_1 = 1$ holds, if i satisfies condition (4.2), otherwise $a^1_1 = 0$.*

Proof : It is best to view the situation on a branching tree. Figure 4 captures the branching trees of a function, in which index i satisfies the condition (4.2) or (4.3). We can see, that if we take x_i as the most significant variable, then one of the root subtrees contains either only truepoints or only falsepoints. Moreover, since the function considered is positive, the other subtree of the root has the nearest vector also truepoint or falsepoint (same as all the vectors in the other subtree), or it is identically equal to zero or one. Hence the interval representation of this possibly non-constant subtree can be easily extended to an interval representation of the whole tree, which has the same number of intervals.

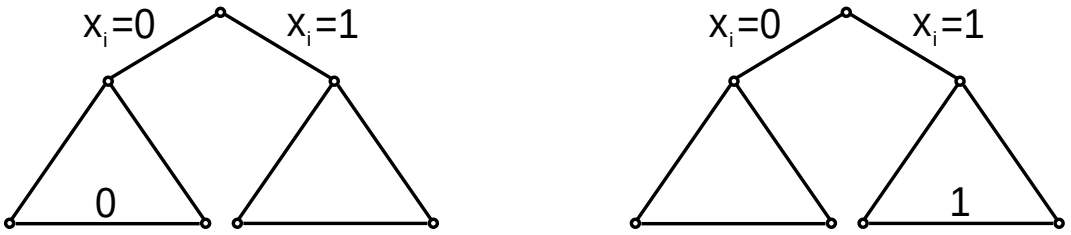


Figure 4: *Condition (4.2) on the left and condition (4.3) on the right in a branching tree.*

Now we proceed with the formal proof. First, if both conditions (4.2) and (4.3) are satisfied by index i, then f does not depend on values of other input variables but x_i, because $f(\boldsymbol{x}) = 1$ for any vector $\boldsymbol{x}$, such that $x_i = 1$, and $f(\boldsymbol{x}) = 0$ for any vector $\boldsymbol{x}$, such that $x_i = 0$. In this case $l = 1$, because f can be represented by interval $[2^{n-1}, 2^n - 1]$ for any ordering having x_i as the most significant variable. Therefore $a_1^1 = 1$.

In the rest of the proof we assume, that i satisfies exactly one of condiditions (4.2), (4.3). We shall define function g on $n-1$ variables. We set $g = f[x_i := 1]$, if i satisfies the condition (4.2), or $g = f[x_i := 0]$ otherwise. In either case g is a k-interval function according to Lemma 3.13. Let π' be an ordering of variables $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$, with respect to which g can be represented by $l \leq k$ intervals. We construct the desired ordering π from π' in the following way. We set $\pi(i) = 1$ and for all $j \neq i$ we set $\pi(i) = \pi'(i) + 1$, i.e., we add x_i as the most significant variable to π'. We claim that f is represented by l intervals with respect to π'. This is true, because if the condition (4.2) is satisfied, then all vectors $\boldsymbol{x}$ with $x_i = 0$ are falsepoints, every truepoint $\boldsymbol{u}$ has $u_i = 1$, hence $u^\pi > x^\pi$, and $f(\boldsymbol{y}) = 1$ holds for any vector $\boldsymbol{y}$ with $y_i = 1$, if and only if $g(\boldsymbol{y}') = 1$, where $\boldsymbol{y}'$ is formed from $\boldsymbol{y}$ by removing the i-th bit. If the condition (4.3) is satisfied, then all vectors $\boldsymbol{x}$ with $x_i = 1$ are truepoints, every falsepoint $\boldsymbol{v}$ has $v_i = 0$, hence $v^\pi < x^\pi$, and $f(\boldsymbol{y}) = 0$ holds for any vector $\boldsymbol{y}$ with $y_i = 0$, if and only if $g(\boldsymbol{y}') = 0$, where $\boldsymbol{y}'$ is again formed from $\boldsymbol{y}$ by removing the i-th bit. Thus if $[a^1, b^1] < \ldots < [a^l, 2^{n-1} - 1]$ are the intervals representing g with respect to π', then we can form from them an interval representation of f with respect to π by adding 1 (in case the condition (4.2) is satified), or 0 (in case the condition (4.3) is satisfied) as the most significant bit to all interval boundaries except $2^{n-1} - 1$, which is always transformed into $2^n - 1$. Therefore $a_1^1 = 1$, if i satisfies the condition (4.2). This completes the proof. ■

Lemma 4.6. *Let f be a positive k-interval function on n variables and let i be an index $1 \leq i \leq n$, which satisfies exactly one of the conditions (4.2), (4.3). Let us set $c = 1$, if i satisfies the condition (4.2), or $c = 0$, if i satisfies the condition (4.3). Then f is a positive threshold function, if and only if $f' = f[x_i := c]$ is a positive threshold function.*

Proof : To explain the idea of the proof, we shall again refer to Figure 4. In case of the condition (4.2) function f has positive threshold structure (ω, t), if and only if function f', which corresponds to the right subtree of the root, has positive threshold structure $(\omega, t - \omega(x_i))$. Similarly, in case of the condition (4.3) function

f has positive threshold structure (ω, t), if and only if f', which corresponds to the left subtree of the root, has positive threshold structure (ω, t).

Now we show the formal proof. Let f be a positive threshold function and let (ω, t) be a positive threshold structure for f. Let ω' be a restriction of ω on $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ and let $t' = t-(c\cdot\omega(x_i))$. Then (ω', t') is a positive threshold structure for f', because for any vector $\boldsymbol{x}'$ of length $n-1$ it holds, that $f'(\boldsymbol{x}') = 1$, if and only if $f(\boldsymbol{x}) = 1$, where the vector $\boldsymbol{x}$ is formed from $\boldsymbol{x}'$ by inserting c as the i-th bit. Moreover, $f(\boldsymbol{x}) = 1$ holds, if and only if $\omega(\boldsymbol{x}) \geq t$, and this holds, if and only if $\omega'(\boldsymbol{x}') \geq t'$. Hence f' is a positive threshold function.

Conversely, let f' be a positive threshold function on $n-1$ variables and (ω', t') be a positive threshold structure for f'. Let Ω be the sum of the weights of all variables in f' (let $\Omega = 0$ if f' has no variables). We set $\omega(x_i) = \Omega + 1$ and $\omega(x_j) = \omega'(x_j)$ for any $x_j \neq x_i$. Also, we set $t = t' + (c \cdot \omega(x_i))$. Then (ω, t) is a positive threshold structure for f, because

- for any vector $\boldsymbol{x}$, such that $x_i > c$, we have $\omega(\boldsymbol{x}) > t$, and such $\boldsymbol{x}$ is a truepoint, because $c = 0$ and hence all falsepoints have 0 in the i-th bit,
- for any vector $\boldsymbol{x}$, such that $x_i < c$, we have $\omega(\boldsymbol{x}) < t$, and such $\boldsymbol{x}$ is a falsepoint, because $c = 1$ and hence all truepoints have 1 in the i-th bit, and
- for any vector $\boldsymbol{x}$, such that $x_i = c$, we have $\omega(\boldsymbol{x}) = (c \cdot \omega(x_i)) + \omega'(\boldsymbol{x}')$, where $\boldsymbol{x}'$ is formed from $\boldsymbol{x}$ by removing the i-th bit, and thus $\omega(\boldsymbol{x}) \geq t$, if and only if $\omega'(\boldsymbol{x}') \geq t'$.

Therefore f is a positive threshold function. ■

In the proof of Lemma 4.6 it is also shown, how to construct a threshold structure for a positive k-interval function f, for which there is an index satisfying one of the conditions (4.2), (4.3), provided, that we have a threshold structure for f'.

Corollary 4.7. *Let $k \geq 2$ be given. Then $\mathcal{C}^+_{k-\text{int}} \subseteq \mathcal{C}^+_{\text{th}}$, if and only if every positive k-interval function, for which there is no index i satisfying one of the conditions (4.2) and (4.3), is a positive threshold function.*

Proof : While there is some index satisfying one of the conditions (4.2) or (4.3), we can use Lemma 4.5 (and also its proof) to fix its value and we again get a positive k-interval function, which, according to Lemma 4.6, is a positive threshold functions, if and only if f is a positive threshold function. ■

Using this corollary we shall show that positive 2-interval functions constitute a subclass of positive threshold functions. We shall proceed with some more properties of k-interval functions, that we need to prove this inclusion.

Lemma 4.8. *Let f be a positive function. Let i be an index not satisfying either of the conditions (4.2),(4.3). Then $f(\boldsymbol{x}) = 0$ holds for the vector $\boldsymbol{x}$ with bit x_i equal to 1 and with all other bits equal to 0, and $f(\boldsymbol{y}) = 1$ holds for the vector $\boldsymbol{y}$ with bit y_i equal to 0 and with all other bits equal to 1.*

Proof : We proceed by contradiction. If $f(\boldsymbol{y})$ is 0, then all vectors with the i-th bit equal to 0 must be mapped to 0 by f as well (because f is a positive function).

In that case all truepoints would have 1 as the i-th bit and hence i would satisfy the condition (4.2), contradiction. Therefore $f(\boldsymbol{y}) = 1$.

In case $f(\boldsymbol{x})$ is 1, then all vectors with the i-th bit equal to 1 must be truepoints and hence all falsepoints have the i-th bit equal to 0, i.e., i would satisfy the condition (4.3), contradiction. Therefore $f(\boldsymbol{x}) = 0$. ■

Corollary 4.9. *Let f be a positive 2-interval function, which is not a 1-interval function, and let π be an ordering, with respect to which f can be represented by intervals $[a^1, b^1] < [a^2, 2^n - 1]$ and such that i, for which $\pi(i) = 1$ (i.e., the index of the most significant variable with respect to π), does not satisfy either of the conditions (4.2), (4.3). Then $b^1 = y^\pi$, where $\boldsymbol{y}$ is the vector with $y_i = 0$ and all other bits* 1.

This corollary shows, that if there is no index satisfying either of the conditions (4.2), (4.3), then each of the two intervals representing a positive function spans the numbers corresponding to truepoints with the same value of the most significant bit. It should be also clear, that by fixing the value (either to 1 or 0) of the variable corresponding to the most significant bit, we get a (positive) 1-interval function.

The following lemma establishes a relation between the conditions (4.2), (4.3) and properties of prime DNF representations of positive functions.

Lemma 4.10. *Let $\mathcal{F}$ be a prime DNF representing positive function f. Then*

(a) every term of $\mathcal{F}$ contains variable x_i, if and only if i satisfies the condition (4.2),

(b) variable x_i forms a linear term in $\mathcal{F}$, if and only if i satisfies the condition (4.3).

Proof : We shall start with the first implications of the equivalences. In (a), if every term of $\mathcal{F}$ contains x_i, then any vector $\boldsymbol{x}$ having $x_i = 0$ is a falsepoint, hence every truepoint must have the i-th bit equal to 1. Similarly, if in (b) variable x_i forms a linear term in $\mathcal{F}$, then any vector $\boldsymbol{x}$ having $x_i = 1$ is a truepoint, hence every falsepoint must have the i-th bit equal to 0.

Now we shall prove the second implications. If i satisfies the condition (4.2), then by fixing x_i to 0 in $\mathcal{F}$ we must get an empty DNF, because it represents the function identically equal to 0. Hence x_i must occur in every term of $\mathcal{F}$. Similarly, if i satisfies the condition (4.3), then by fixing x_i to 1 in $\mathcal{F}$, we must get a DNF representing the function, which is identically equal to 1. Hence x_i must form a linear term in $\mathcal{F}$. ■

Corollary 4.11. *Let f be a positive 1-interval function on n variables, which is not identically equal to zero. Let π be an ordering, with respect to which f can be represented by one interval, and let x_i be the most significant variable, i.e., $\pi(i) = 1$, and let $\mathcal{F}$ be the prime DNF representing f. Then $\mathcal{F}$ contains the linear term x_i or the variable x_i is contained in every term of $\mathcal{F}$.*

Proof : Follows from Lemma 4.10, because i must satisfy at least one of the conditions (4.2), (4.3), otherwise f is not a 1-interval function due to Corollary 4.9. ■

Lemma 4.12. *Let $\mathcal{F}$ be the prime DNF representing positive 2-interval function f, which is not a 1-interval function, and such that there is no index i satisfying*

either of the conditions (4.2), (4.3). Let π be an ordering of variables, with respect to which f can be represented by 2 intervals. Then $\mathcal{F}$ has the form

$$\mathcal{F} = x_i x_j \vee x_i \mathcal{G} \vee x_j \mathcal{H},$$

where x_i, x_j are the first and the second most significant variables in the ordering π, and $\mathcal{G}$ and $\mathcal{H}$ are positive DNFs not containing variables x_i and x_j. Moreover, $\mathcal{G}$ and $\mathcal{H}$ represent positive functions, which can be represented by one interval with respect to the same ordering of variables.

Proof : Let π be an ordering, with respect to which f can be represented by two intervals $[a^1, b^1] < [a^2, 2^n - 1]$. According to Corollary 4.9, $b = 2^{n-1} - 1$. Therefore, both $f_0 = f[x_i := 0]$ and $f_1 = f[x_i := 1]$ are positive functions, which can be represented by one interval, moreover, with respect to the same ordering π', formed from π by restriction on variables $\{x_1, \ldots, x_{i-1}, x_{i+1}, x_n\}$.

Then x_j is the most significant variable with respect to π'. According to Corollary 4.11, the prime DNF representations $\mathcal{F}_0$ and $\mathcal{F}_1$ of f_0 and f_1 contain a linear term formed by x_j, or contain x_j in every term. But x_j cannot form a linear term in $\mathcal{F}_0$, because otherwise this term is also present in $\mathcal{F}$ and that is not possible according to Lemma 4.10 and the assumptions of the lemma, that we are proving. Therefore x_j has to be contained in every term of $\mathcal{F}_0$. If x_j was also contained in every term of $\mathcal{F}_1$, then x_j would be present in every term of $\mathcal{F}$, which is again not possible due to our assumptions. Therefore, x_j must form a linear term in $\mathcal{F}_1$. However, since $\mathcal{F}$ does not have linear terms, it is only possible for this term to occur in $\mathcal{F}_1$, if there is the term $x_i x_j$ in $\mathcal{F}$.

Putting this together with the fact, that $\mathcal{F}$ is prime, we get that

1. $x_i x_j$ is a term in $\mathcal{F}$ and this implies that no other term contains both varibles x_i and x_j
2. in every term t of $\mathcal{F}$ not containing x_j there is variable x_i (because none of these terms is in $\mathcal{F}_0$)

These conditions imply the desired form of $\mathcal{F}$. Also, $\mathcal{G} = \mathcal{F}_1[x_j := 0]$ and $\mathcal{H} = \mathcal{F}_0[x_j := 1]$, hence they both represent functions, which can be represented by one interval with respect to the ordering formed from π' by restriction on the remaining variables. ■

Lemma 4.13. *Let f be a positive 2-interval function on n variables, which is not a 1-interval function. Moreover, let us suppose, that there is no index satisfying either of the conditions (4.2), (4.3). Let π be an ordering of its variables, with respect to which f can be represented by intervals $[a, 2^n - 1], [c, 2^n - 1]$. Then*

(a) the most significant bit of a is 0 and the second most significant bit of a is 1,

(b) the most significant bit of c is 1 and the second most significant bit of c is 0.

Proof : Let us start with (a). Using Lemma 4.12 we get, that any vector, which has the first two most significant bits 0, is a falsepoint of f. This, together with Corollary 4.9, proves the first part of the lemma.

We proceed with (b). First, according to Corollary 4.9, the most significant bit of c is 1. Using Lemma 4.12, we know that $\mathcal{F} = x_i x_j \vee x_i \mathcal{G} \vee x_j \mathcal{H}$, where $\mathcal{F}$ is the

prime DNF representing f, x_i is the most significant variable and x_j is the second most significant variable with respect to π. It follows, that any vector, which has the most significant bit 1 and the second most significant bit 0, is a truepoint of f, if and only if $\mathcal{G}$ is satisfied by the values of the remaining bits. The second most significant bit of c is 1, if and only if all vectors with the first two most significant bits $1, 0$ are falsepoints and this holds, if and only if $\mathcal{G}$ represents the function, which is identically equal to zero. However, this is not possible, because it means, that the prime representation of f contains the second most significant variable in every term, which contradicts the assumptions of this lemma according to Lemma 4.10. ■

Lemma 4.13 together with Corollary 4.9 imply, that the branching tree of a positive 2-interval function, in which no index of a variable satisfies either of the conditions (4.2), (4.3), has the form visualized on Figure 5. Now we are ready to prove the main result of this subsection.

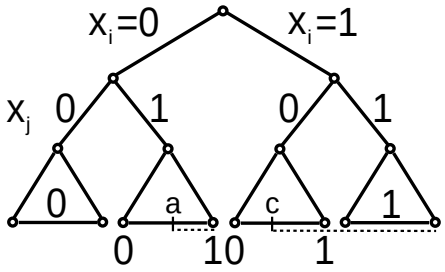


Figure 5: *A branching tree of a positive 2-interval function, which is represented by intervals* $[a, 2^{n-1}-1], [c, 2^n-1]$ *and in which no index of a variable satisfies conditions (4.2) and (4.3). The pair of variables* x_i, x_j *satisfies Lemma 4.12.*

Theorem 4.14. *Let f be a positive 2-interval function, which is not a 1-interval function and for which there is no index satisfying either of the conditions (4.2), (4.3). Then f is a threshold function.*

Proof : Again, we first explain the idea of the proof. We refer to Figure 5. We will assign the weights to variables x_i and x_j in such way, that the vectors corresponding to the interval boundaries a and c will exactly reach the threshold. This is possible due to Lemma 4.13.

Now let us prove the lemma formally. Let f be represented by $[a, 2^{n-1}-1] < [c, 2^n-1]$ with respect to some ordering π. First we note, that according to Lemma 4.13, the two most significant bits of a are $0, 1$ and the two most significant bits of c are $1, 0$. Let $\boldsymbol{b}$ be the vector of n bits, such that $b^\pi = a$, i.e., $\boldsymbol{b}$ is the vector corresponding to the boundary a (with respect to π). Hence $f(\boldsymbol{b}) = 1$ and

$$f(\overrightarrow{(b^\pi - 1)^{\pi^{-1}}}) = 0 \ .$$

Similarly, let $\boldsymbol{d}$ be the vector, such that $d^\pi = c$.

We shall construct a positive threshold structure (ω, t) for f. We set weight $\omega(x_i)$ of variable x_i to $2^{n-\pi(i)}$ with the only exception of variable x_j, such that $\pi(j) = 2$,

which will have weight $\omega(x_j) = \omega(\boldsymbol{d}) - \omega(\boldsymbol{b}')$, where $\boldsymbol{b}'$ is the vector of length $n-2$ formed from $\boldsymbol{b}$ by removing the two most significant bits with respect to π. This definition of $\omega(x_j)$ is correct, because c (and consequently also $\boldsymbol{d}$) has 0 in the bit corresponding to variable x_j. Threshold t will be set to $\omega(\boldsymbol{d}) = \omega(\boldsymbol{b}') + \omega(x_j) = \omega(\boldsymbol{b})$. The result is, that the weights are assigned in such way, that $\omega(\boldsymbol{b}) = \omega(\boldsymbol{d})$ and the threshold is set to this value as well. Hence, for any vector $\boldsymbol{x}$, such that $x^\pi < a$, we have $f(\boldsymbol{x}) = 0$ and $\omega(\boldsymbol{x}) < \omega(\boldsymbol{b}) = t$, because $\omega(x_j) > \omega(x_{\pi^{-1}(1)}) - \sum_{i=0}^{n-3} 2^i = 2^{n-1} - 2^{n-2} - 1 = 2^{n-2} - 1$ and hence no vector having two zeros in the two most significant bits can reach the threshold. Also, for any vector $\boldsymbol{y}$, such that $\boldsymbol{y}^\pi \geq 100\ldots0$ and $y^\pi < c$, we have $f(\boldsymbol{y}) = 0$ and $\omega(\boldsymbol{y}) < \omega(\boldsymbol{d}) = t$. This completes our proof. ■

Corollary 4.15. $\mathcal{C}^+_{2-\text{int}} \subset \mathcal{C}^+_{\text{th}}$.

Proof : Follows from Corollary 4.7 and Theorem 4.14. ■

4.4 Relation of positive threshold and positive 3-interval functions

In the previous subsection we proved, that positive 2-interval functions constitute a proper subclass of positive threshold functions. Also, some of the lemmas leading to this result were formulated for general k, hence it could be quite natural to expect, that also positive 3-interval functions constitute a subclass of positive threshold functions. However, in this subsection we prove the opposite result.

Theorem 4.16. $\mathcal{C}^+_{3-\text{int}} \not\subseteq \mathcal{C}_{2\text{m}}$.

Proof : We shall show an example of a positive 3-interval function and prove, that it is not a 2-monotonic function. Let f be the function on 4 variables represented by prime positive DNF

$$x_2x_4 \vee x_1x_3 \vee x_1x_2.$$

Function f can be represented by three intervals $[5,5], [7,7], [10,15]$ with respect to the natural ordering x_1, x_2, x_3, x_4. Hence it is a positive 3-interval function. However, f is not a 2-monotonic function, because the truepoint 1010 and the falsepoint 1001 require $x_3 \succ x_4$ and the truepoint 0101 and the falsepoint 0110 require $x_3 \prec x_4$. Hence x_3 and x_4 are not of comparable strength in f. ■

Corollary 4.17. $\mathcal{C}^+_{3-\text{int}} \not\subseteq \mathcal{C}^+_{\text{th}}$.

Proof : Follows from Theorem 4.16, because every positive threshold function is a 2-monotonic function as well (see [21]). ■

Corollary 4.18. $\mathcal{C}^+_{k-\text{int}} \not\subseteq \mathcal{C}^+_{\text{th}}$ *for any* $k \geq 4$.

Proof : Follows from Corollary 4.17 and the fact, that $\mathcal{C}^+_{3-\text{int}} \subset \mathcal{C}^+_{k-\text{int}}$ for any $k \geq 4$. ■

5 Interval Boolean Function Recognition

In this section we study the problem, when we are given a Boolean function represented by a DNF and we want to decide, whether it can be represented by at most k

intervals. We will call this problem INTERVAL-RECOGNITION and we shall consider two variants of it. One of them has k as a part of its input, the other variant has k as a constant (it is, in fact, a different problem for each value of k). We describe these variants by the following definitions.

INTERVAL-RECOGNITION	
Instance :	A DNF $\mathcal{F}$ and a number $k \in \mathbf{N}$.
Question :	Can the function specified by $\mathcal{F}$ be represented by at most k intervals with respect to some ordering of its variables?

k-INTERVAL-RECOGNITION	
Instance :	A DNF $\mathcal{F}$.
Question :	Can the function specified by $\mathcal{F}$ be represented by at most k intervals with respect to some ordering of its variables?

In both variants we expect to be able to also output an interval representation of size at most k in case of the affirmative answer. For each variant of the interval-recognition problem we shall also study its modification, when the ordering of the variables is given in advance (as a part of the input) and we shall consider the question of the problem only with respect to this one ordering. We denote these modifications by INTERVAL-RECOGNITION-FIXED and k-INTERVAL-RECOGNITION-FIXED. We also suppose (without loss of generality and unless otherwise stated), that the only ordering we should consider is specified by the identical permutation (i.e., the ordering $x_1, x_2, \ldots, x_n$).

We shall study all of these problems in the following subsections. In the cases, when k is a constant, we consider several values of k. We shall also consider restricted versions of the problems, when we have an additional requirement on the input DNF, such as to represent a function from a specified class of functions (such as positive or negative), and to be prime and irredundant. First we prove, that the recognition of k-interval functions is co-NP-hard in general (even in the variant with a fixed ordering) for every value of $k \geq 1$. In the following subsection we study the recognition of positive and negative 1-interval functions and we construct a polynomial-time algorithm solving it. Then we generalize this result to general 1-interval functions, which have DNF representations satisfying additional requirements. The next step is the recognition of renamable 1-interval functions. Then we come back to positive and negative functions, present an algorithm solving the recognition problem for positive and negative 2-interval functions and a general framework yielding a polynomially incremental algorithm for solving the recognition problem with fixed ordering for positive and negative k-interval functions. In the last subsection we apply this framework to 2-monotonic functions and present an asymptotically optimal algorithm for threshold functions represented by a threshold structure.

5.1 Hardness of the general case

In this subsection we study the unrestricted versions of the recognition problems for k-interval functions, i.e., we do not have any additional requirements on the input DNF. By the following theorem we show, that it is unlikely that a polynomial algorithm can be constructed in any of these cases for any value of k.

Theorem 5.1. *The problem* k-INTERVAL-RECOGNITION *(for any* $k \in \mathbf{N}, k > 0$*) is co-NP-hard. The problem* k-INTERVAL-RECOGNITION-FIXED *(for any* $k \in \mathbf{N}, k > 0$*) is co-NP-complete.*

Proof : Let DNF $\mathcal{F}$ representing function f be given. We will take an instance of the problem TAUTOLOGY defined in Subsection 2.4. It is easy to observe, that f is a tautology, if and only if it can be represented by one interval and $f(00\ldots0) = f(11\ldots1) = 1$ holds at the same time. This proves the co-NP-hardness of 1-INTERVAL-RECOGNITION. Also, f is a tautology, if and only if f can be represented by one interval $[0, 2^n - 1]$ in every ordering of variables, so we can use this simple transformation from TAUTOLOGY also to prove the co-NP-hardness of 1-INTERVAL-RECOGNITION-FIXED.

To see that this version of our problem is in co-NP, we only need to observe, that when we are given three vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$, such that $x < y < z$ and $f(\boldsymbol{x}) = f(\boldsymbol{z}) = 1$ and $f(\boldsymbol{y}) = 0$, we have a certificate of the negative answer, which can be verified in polynomial time.

Now let us prove by induction on k, that k-INTERVAL-RECOGNITION and k-INTERVAL-RECOGNITION-FIXED are co-NP-hard. We show it by a polynomial reduction from $(k-1)$-INTERVAL-RECOGNITION ($(k-1)$-INTERVAL-RECOGNITION-FIXED, resp.). Let DNF $\mathcal{F}$ representing function f on n variables be given and we want to decide, whether it can be represented by $(k-1)$ intervals in a suitable (the given, resp.) ordering of variables. Let x_{n+1}, x_{n+2} be two new variables not occuring in $\mathcal{F}$. We construct DNF

$$\mathcal{F}' = \overline{x}_{n+1}\overline{x}_{n+2}\mathcal{F} \vee \wedge_{i=1}^{n+2} x_i \ ,$$

i.e., $\mathcal{F}'$ is formed from $\mathcal{F}$ by adding $\overline{x}_{n+1}$ and $\overline{x}_{n+2}$ to every term of $\mathcal{F}$ and by adding one new positive term containing all variables. Let f' be the function represented by $\mathcal{F}'$. In case of $(k-1)$-INTERVAL-RECOGNITION-FIXED, when we consider only the natural ordering $x_1, x_2, \ldots, x_n$ of variables, we add x_{n+1} and x_{n+2} as the two most significant variables to this ordering and consider the problem only with respect to the resulting ordering. The exact order of the two new variables does not matter, because they are symmetrical with respect to F' (and hence also to f'). We claim that $\mathcal{F}$ can represented by at most $(k-1)$ intervals, if and only if $\mathcal{F}'$ can be represented by at most k intervals.

First, if f can be represented by $l \leq k-1$ intervals $[a^1, b^1] < [a^2, b^2] \ldots [a^l, b^l]$, where all a^i, b^i are n-bit numbers, with respect to ordering π of variables, then f' can be represented by $l+1 \leq k$ intervals $[a^1, b^1] < [a^2, b^2] \ldots [a^l, b^l] < [2^{n+2} - 1, 2^{n+2} - 1]$, where all a^i, b^i are $(n+2)$-bit numbers, with respect to ordering π' formed from π by adding x_{n+1} and x_{n+2} as the two most significant variables. This should be easily observable from Figure 6, because the leftest subtree corresponding to f can be represented by l intervals and we only need one more interval to span vector $(11\ldots1)$. Hence if f can be represented by at most $k-1$ intervals with respect

to a suitable (the given, resp.) ordering, then f' can be represented by at most k intervals with respect to a suitable (the given, resp.) ordering.

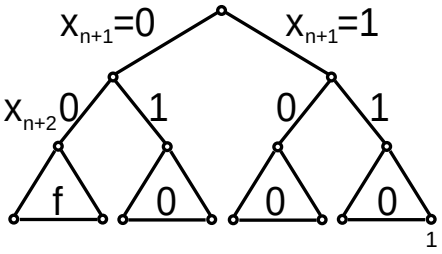


Figure 6: *A branching tree of function f' represented by DNF $\mathcal{F}'$ with respect to ordering π'. The leftest subtree on figure corresponds to the branching tree of function f with respect to ordering π. All other subtrees on figure contain only falsepoints with the exception in vector* $(11\ldots1)$.

Conversely, let us suppose that f' can be represented by $l \leq k$ intervals $[a^1, b^1] < [a^2, b^2] \ldots [a^l, b^l]$, where all a^i, b^i are $(n+2)$-bit numbers, with respect to ordering π'. Because $f = f'[x_{n+1} = 0][x_{n+2} = 0]$, we know, according to Lemma 3.13, that f can also be represented by at most l intervals, moreover, with respect to the ordering π formed from π' by restriction on $\{1, \ldots, n\}$. We show that at least one interval becomes unnecessary after such restriction. Let $\pi'(n+1) < \pi'(n+2)$. According to the definition of $\mathcal{F}'$, $f'(\boldsymbol{x}) = 0$ holds for all vectors $\boldsymbol{x}$, such that $x_i = 1$ for all i, for which $\pi'(i) < \pi'(n+1)$, and $x_{n+1} = 1$, except for the vector of all ones. Because we suppose $\pi'(n+1) < \pi'(n+2)$, there is at least one such vector $\boldsymbol{x}$ (i.e., $(11\ldots10)^{\pi'}$). Therefore, we have $a^l = b^l = 2^{n+2} - 1$. However, in f we do not have any of these vectors $\boldsymbol{x}$, including the vector of all ones, because $x_{n+1} = 1$. Therefore, we can construct the representation of f by $l - 1 \leq k - 1$ intervals by removing the $\pi'(n+1)$-th and $\pi'(n+2)$-th bits of $a^1, b^1, \ldots, a^{l-1}, b^{l-1}$. Hence if f' can be represented by at most k intervals with respect to a suitable (the given, resp.) ordering, then f can be represented by at most $k-1$ intervals with respect to a suitable (the given, resp.) ordering. Thus by transforming $\mathcal{F}$ to $\mathcal{F}'$ we reduced $(k-1)$-INTERVAL-RECOGNITION ($(k-1)$-INTERVAL-RECOGNITION-FIXED, resp.) to k-INTERVAL-RECOGNITION (k-INTERVAL-RECOGNITION-FIXED, resp.).

Again, to prove, that k-INTERVAL-RECOGNITION-FIXED is in co-NP, we can take truepoints $\boldsymbol{u}^1, \ldots, \boldsymbol{u}^{k+1}$ and falsepoints $\boldsymbol{v}^1, \boldsymbol{v}^k$, such that $u^1 < v^1 < u^2 < v^2 \ldots u^{k+1}$ as a polynomial certificate of the negative answer. ■

Corollary 5.2. *The problems* INTERVAL-RECOGNITION *and* INTERVAL-RECOGNITION-FIXED *are co-NP-hard.*

Proof : It suffices to realize, that these problems can be used to solve the problems k-INTERVAL-RECOGNITION and k-INTERVAL-RECOGNITION-FIXED for any k. ■

Due to the co-NP-hardness results, that we have just presented, we are forced to study somehow restricted versions of the interval recognition problems in order to be able to construct polynomial algorithms solving them.

5.2 Positive and negative 1-interval function recognition

In this section we solve the problem of recognizing, whether a given prime DNF, which is positive or negative, represents an interval function. We shall at first develop an algorithm recognizing positive interval functions and then we will explain, how to use it to recognize negative interval functions.

According to Lemma 3.12 we know, that any positive interval function is defined by interval $[a, 2^n - 1]$ for some a (or it is indentically equal to zero). Also any interval function defined by interval of type $[a, 2^n - 1]$ is positive. This is because by flipping some zeros to ones in any $x \in [a, 2^n - 1]$ the resulting integer remains in this interval, therefore it again corresponds to a truepoint. The following two lemmas state the conditions, that make it possible to effectively decide, whether a given positive function can be represented by one interval. The first one is formulated for a general k.

Lemma 5.3. *Let $\mathcal{F}$ be a positive DNF representing $f \in \mathcal{C}^+_{k-\text{int}} \setminus \mathcal{C}^+_{(k-1)-\text{int}}$, where $k \geq 1$. Let x_i be a variable forming a linear term in $\mathcal{F}$ (let $c := 0$ in this case), or contained in every term of $\mathcal{F}$ (let $c := 1$ in this case). Then there exists ordering π, with respect to which f can be represented by k intervals and such that $\pi(i) = 1$, i.e., x_i is the most significant variable with respect to π. Moreover, for every ordering π of variables of f, such that $\pi(i) = 1$, f can be represented by k intervals $[a^1, b^1] < \ldots < [a^k, 2^n - 1]$ with respect to π, if and only if $f[x_i := c]$ can be represented by k intervals $[a^{1\prime}, b^{1\prime}] < \ldots < [a^{k\prime}, 2^{n-1} - 1]$ with respect to ordering π', where $a^{i\prime}, b^{i\prime}$ are $(n-1)$-bit integers formed from n-bit integers a^i, b^i by removing the most significant bit and π' is a restriction of π to the remaining variables.*

Proof : According to Lemma 4.10 index i satisfies one of the conditions (4.2), (4.3). The existence of ordering π with the specified properties then follows from Lemma 4.5.

The same lemma implies, that with respect to this ordering π function f can be represented by k intervals $[a^1, b^1] < \ldots < [a^k, 2^n - 1]$, such that $a^1_1 = a^2_1 = \ldots = a^k_1$. Any n-bit vector $\boldsymbol{x}$, such that $x_i = a^1_1$, is a truepoint of f, if and only if $(n-1)$-bit vector $\boldsymbol{y}$ formed from $\boldsymbol{x}$ by removing the i-th bit is a truepoint of f'. Therefore, $\boldsymbol{x}^\pi \in \cup_{i=1}^k [a^i, b^i]$, where $b^k = 2^n - 1$, if and only if $\boldsymbol{y}^{\pi'} \in \cup_{i=1}^k [a^{i\prime}, b^{i\prime}]$, where π' is a restriction of π to $\{1, \ldots, i-1, i+1, \ldots, n\}$ and $b^{k\prime} = 2^{n-1} - 1$. The other vectors $\boldsymbol{x}$ of f, having $x_i \neq a^1_1$, are either all falsepoints such that $x^\pi < a^1$ (in case x_i is contained in every term of $\mathcal{F}$ and hence $a^1_1 = 1$), or all truepoints such that $x^\pi > a^k$ (in case x_i is a linear term in $\mathcal{F}$ and hence $a^1_1 = 0$), and therefore f' can be represented by k intervals, if and only if f can be represented by k intervals. ■

Lemma 5.4. *Let $\mathcal{F}$ be a nonempty prime DNF representing a positive interval function f. Then $\mathcal{F}$ contains a linear term or there is a variable contained in every term of $\mathcal{F}$.*

Proof : Follows from Corollary 4.11. ■

The previous lemmas suggest an algorithm for deciding whether a given positive DNF represents an interval function. In every iteration of the algorithm we will look for one variable satisfying the properties of Lemma 5.4 (i.e., a variable appearing as a linear term or contained in every term) and remove it by fixing its value to a

suitable constant. Due to Lemma 5.3 the resulting function is then interval, if and only if the original function is interval. This is the idea of the recognition algorithm, that we are now ready to formulate.

Algorithm 5.5. 1-Interval-Recognition($\mathcal{C}^+$)

Input: *A nonempty prime and irredundant positive DNF $\mathcal{F}$ on n variables representing function f.*

Output: **0** *if $\mathcal{F}$ represents a 0-interval function. Ordering π of variables and n-bit number a, if $\mathcal{F}$ represents a positive interval function, which can be defined by interval $[a, 2^n - 1]$ with respect to ordering π.* **NO** *otherwise.*

1: **if** $\mathcal{F}(11\ldots1) = 0$ **then** $\mathcal{F}$ is constant 0, **output 0**, **halt endif**
2: **if** $\mathcal{F}(00\ldots0) = 1$ **then** $\mathcal{F}$ is constant 1, **output** integer 0, **halt endif**
3: $a := 0$ # *a is an n-bit integer with all bits initialized to* 0
4: $m := 0$
5: **while** in $\mathcal{F}$ there is a variable x_i constituting a linear term or contained in every term **do**
6: $m := m + 1$, $\pi(i) := m$
7: **if** variable x_i is contained in every term of $\mathcal{F}$ **then**
8: $a_m := 1$
9: $\mathcal{F} := \mathcal{F}[x_i := 1]$
10: **else if** variable x_i constitutes a linear term of $\mathcal{F}$ **then**
11: $\mathcal{F} := \mathcal{F}[x_i := 0]$
12: **endif**
13: **enddo**
14: **while** $m < n$ **do**
15: $m := m + 1$ # *corresponding bits of a are set correctly by initialization*
16: $\pi(i) := m$, for some $i \leq n$, which is not mapped yet
17: **enddo**
18: **if** $\mathcal{F} = \emptyset$ **then**
19: **output** the ordering π and the integer a
20: **else**
21: **output NO**.
22: **endif**

Theorem 5.6. *Algorithm 5.5 correctly recognizes positive prime DNFs representing positive interval functions and for each such function outputs the order of variables and the interval represented by this interval function.*

Proof : The correctness follows from Lemma 5.3 and Lemma 5.4. When we process one variable x_i and fix its value, we get another DNF, which is again prime and irredundant, because we only remove a linear term x_i (hence x_i does not occur in any other term), or remove variable x_i from every term. The resulting DNF represents a positive interval function, if and only if the original function is a positive interval function (due to Lemma 5.3). We also set the most significant bit of the boundary a according to the condition that x_i fulfilled (following Lemma 4.5). Now we can iterate the process, because all preconditions of our algorithm are satisfied.

If we cannot process all variables of DNF $\mathcal{F}$ before halting, we know the condition of Lemma 5.4 is not satisfied and thus $\mathcal{F}$ does not represent an interval function.

If we can process all variables of $\mathcal{F}$, then a sufficient condition for $\mathcal{F}$ to represent an interval function is satisfied, but we still need to set the ordering mapping π for every variable of f, which is not present in $\mathcal{F}$. All such variables are treated as the least significant ones.

We also note that the order of steps 7 and 10 must be exactly as in our algorithm, because in the last iteration of the main loop 5 – 13, when we always have a DNF consisting of just one variable, it is necessary to treat it like a variable occuring in every term and set the corresponding bit of a to 1 (on line 8) to get the right value of a. ■

Next we will show how to implement this algorithm effectively (and asymptotically optimally).

Theorem 5.7. *It is possible to implement Algorithm 5.5 to run in $\Theta(l)$ time, where l is the number of literals in the input positive DNF $\mathcal{F}$.*

Proof : If we analyze the algorithm, we see, that it involves at most n main iterations. The complexity of every iteration depends on the implementation of the following operations (remaining operations are of constant time complexity):

1. "Find and remove from $\mathcal{F}$ (some) variable that occurs in every term of $\mathcal{F}$"
2. "Find and remove from $\mathcal{F}$ (some) linear term y"

We need to implement these operations in such a way, that their running time over all iterations of the algorithm is $O(l)$. We introduce three types of data structures to achieve this. We define a structure `T(t)` corresponding to every term t, a structure `V(x)` for every variable x, and also a structure `L(r)` for every literal r of DNF $\mathcal{F}$. For every term t we will also remember in `n(t)` the number of variables in t and for every variable x we will store in `t(x)` the number of literals formed by x (this equals to the number of terms x is in).

In the initialization step we sort term structures `T(t)` by `n(t)` in ascending order and variable structures `V(x)` by `t(x)` in descending order. If we employ *RadixSort*, we can achieve this in $O(l)$ time, because we are sorting integers from the range $[1, l]$. In these orders we put terms in double-linked list `terms` and variables in double-linked list `vars`. Then for every variable x we create a double-linked list of all structures corresponding to the literals formed by x. And for every term t we also create a double-linked list of the structures corresponding to the literals occuring in t. Every structure `L(r)` will also have a pointer to the structure corresponding to the variable forming r and to the structure of the term containing r. It is quite clear that we can perform this initialization of the data structures in $O(l)$ time.

Then when we are asked to find some variable x with occurences in all terms, we just need to look at the head `x` of the list `vars` and check whether the number `t(x)` equals the number of terms of $\mathcal{F}$. If it is equal, then we can remove `x` from `vars` in constant time and then we go through the list of literals formed by variable x and for every such literal r we decrease by one the number `n(t)` of term t containing r and we also remove r from the list of literals of t. During the whole execution of the algorithm we will deal with every literal just once, hence we can perform all these operations in $O(l)$ time.

In case we need to find a linear term, we just look at the first term `t` of list `terms` and check whether `n(t)` equals one. If it does, we can remove `t` from `terms`

in constant time and we also need to remove from `vars` the variable forming the (sole) literal of t. By using the pointer from the literal to its variable we can do this in constant time. This operation hence takes just constant time and as it is called at most n times, it fits in $O(l)$ time.

Our last note on the implementation is, that by removing linear term or variable contained in every term our data structure retains all its properties, that it has after initialization, namely the ascending order of terms with respect to `n(t)` is preserved, because we remove either the first term with `n(t)` equal to one or a variable with occurences in all terms and that means we decrease `n(t)` of all terms. Also the descending order of variables with respect to `t(x)` is preserved, because we only remove the first variable or a variable, which has `t(x)` equal to one and hence it is among the last variables in the list. Hence we can iterate these operations and our algorithm with this implementation has $\Theta(l)$ time complexity. ■

Now we will show how to use Algorithm 5.5 for the recognition of negative interval functions. Although we could reformulate all the ideas of this section for negative functions, the easiest way to explain the idea is to note, that by switching all literals of a positive prime DNF $\mathcal{F}$ representing positive interval function f from the positive form to the negative one, we get a DNF representation of a negative interval function f'. And if f is represented by interval $[a, 2^n - 1]$, then f' is represented by the "mirrored" interval $[0, \bar{a}]$, where $\bar{a}$ is formed from n-bit integer a by exchanging zeros and ones. Hence when we get the task to recognize, whether given negative DNF $\mathcal{F}$ represents a negative interval function, we can just switch all literals of $\mathcal{F}$ to the positive form to get positive DNF $\mathcal{F}'$ and then execute Algorithm 5.5 with input $\mathcal{F}'$. If its answer is `NO`, we also output `NO`. If the answer is, that $\mathcal{F}'$ represents a positive interval function, we also get a permutation and an interval and we output the same permutation and the "mirrored" version of the interval.

Corollary 5.8. *It is possible to recognize in $\Theta(l)$ time, whether a given prime negative DNF represents a negative interval function.*

5.3 General 1-interval function recognition

In this subsection, we shall show our results concerning recognition of general 1-interval functions. Due to Theorem 5.1 we know, that unless $P = NP$, there is no chance for a polynomial algorithm to exist for this problem in its whole generality. We present a solution of this problem for the case, when we restrict the accepted inputs to a class of DNFs with some special properties. The following theorem captures our result.

Theorem 5.9. *Let $\mathcal{C}$ be a class of DNFs which satisfies all of the following conditions:*

(a) It is closed under partial assignment.

(b) It is possible to decide the falsifiability problem in polynomial time for every DNF from $\mathcal{C}$.

(c) It is closed under deletion of redundant terms and replacement of a nonprime implicant t by an implicant t' with $t' \geq t$.

Let $\mathcal{F} \in \mathcal{C}$ be a DNF representing Boolean function f. Then we can test in polynomial time, whether $f \in \mathcal{C}_{1-\text{int}}$.

We shall at first show, that the properties (a) and (b) of the class $\mathcal{C}$ are sufficient to allow a polynomial time transformation of any given DNF from $\mathcal{C}$ into an equivalent prime and irredundant DNF, and that the property (c) guarantees, that the resulting DNF also belongs to $\mathcal{C}$.

Lemma 5.10. *Let $\mathcal{C}$ be a class of DNFs satisfying conditions stated in Theorem 5.9 and let $\mathcal{F}$ be an arbitrary DNF from $\mathcal{C}$. Then it is possible to transform $\mathcal{F}$ in polynomial time into a prime and irredundant DNF $\mathcal{F}' \in \mathcal{C}$, which represents the same function as $\mathcal{F}$.*

Proof : Since $\mathcal{C}$ is a class, for which we can decide the falsifiability problem in polynomial time and which is closed under partial assignment, we are able to check for every term $t \in \mathcal{F}$, whether t is an implicant of $\mathcal{F}$. This is achieved by assigning all literals in the given term t to the value 1 (which is the only assignment, which satisfies t), substituting these values into $\mathcal{F}$ and checking, whether the resulting DNF is falsifiable (term t is not an implicant), or is identically equal to 1 (term t is an implicant). Hence we can transform $\mathcal{F}$ into a prime and irredundant DNF $\mathcal{F}'$ representing f in polynomial time by deleting redundant terms and "shortening" the remaining terms into prime implicants (we refer the reader to [15], where this easy fact was shown for the special case of Horn functions). The time complexity of this procedure is $\Theta(l \cdot p(l))$, where l is the length (number of literals) of the input DNF, and $p(l)$ is the time required for testing the falsifiability of a DNF of length (at most) l. ■

Let us consider a DNF $\mathcal{F}$ representing interval function f on n variables defined by interval $[a, b] \subseteq [0, 2^n - 1]$ with respect to ordering $x_1, x_2, \ldots, x_n$. Due to Corollary 3.14, both $\mathcal{F}_0 = \mathcal{F}[x_1 := 0]$ and $\mathcal{F}_1 = \mathcal{F}[x_1 := 1]$ represent interval functions f_0, f_1, respectively, on $n-1$ variables. If $\mathcal{F}_0$ is empty, i.e., f_0 is identically equal to zero, then it means $[a, b] \subseteq [2^{n-1}, 2^n - 1]$ and we can get the values of a and b by finding a', b', such that the interval $[a', b']$ represents f_1 with respect to some ordering of $x_2, x_3, \ldots, x_n$, and setting $a := a' + 2^{n-1}$ and $b := b' + 2^{n-1}$. If, on the other hand, $\mathcal{F}_1$ is empty, i.e., f_1 is identically equal to zero, then $[a, b] \subseteq [0, 2^{n-1} - 1]$ and it is exactly the interval representing f_0 (with respect to some ordering of $x_2, x_3, \ldots, x_n$). Hence in both these cases, we can proceed by recursion on a shorter DNF ($\mathcal{F}_0$ or $\mathcal{F}_1$) with a smaller number of variables.

When none of $\mathcal{F}_0$ and $\mathcal{F}_1$ is empty, then $a < 2^{n-1} \leq b$. In this case f_0 can be represented by the interval $[a, 2^{n-1}]$ and f_1 can be represented by the interval $[0, b - 2^{n-1}]$, both with respect to the same ordering of variables $x_2, x_3, \ldots, x_n$. It follows, that f_0 is a positive interval function, f_1 is a negative interval function and we already have Algorithm 5.5, which can be used to recognize positive and negative (due to Corollary 5.8) interval functions and determine their representing intervals.

In our situation, when we are given a DNF and we do not know neither the correct ordering of variables nor the the fact whether it represents an interval function, we have to be more careful and try to uncover a suitable ordering step by step. Nevertheless, the ideas mentioned above are the cornerstone of our algorithm, which we are now ready to present.

Algorithm 5.11. 1-Interval-Recognition($\mathcal{C}$)

Input: *Nonempty DNF $\mathcal{F} \in \mathcal{C}$ on n variables representing function f (the class $\mathcal{C}$ satisfies the conditions of Theorem 5.9).*

Output: **0** *if $\mathcal{F}$ represents a 0-interval function. Ordering π of variables and n-bit numbers a, b, if $\mathcal{F}$ represents an interval function defined by interval $[a, b]$ with respect to the ordering π.* **NO** *otherwise.*

1: transform the input DNF $\mathcal{F}$ to a prime and irredundant DNF $\mathcal{F}$
2: **if** $\mathcal{F} = \emptyset$ **then output 0**, **halt endif**
3: $i := 1$
4: $\mathcal{F}_1 := \mathcal{F}$
5: **while** $\mathcal{F}_i \neq \emptyset$ **and** $i \leq n$ **and** $\exists j$ $(\mathcal{F}_i[x_j := 0] = \emptyset \vee \mathcal{F}_i[x_j := 1] = \emptyset)$ **do**
6: **if** $\mathcal{F}_i[x_j := 0] = \emptyset$ **then** # *x_j occurs in every term*
7: $a_i := 1$
8: $b_i := 1$
9: $\mathcal{F}_{i+1} := \mathcal{F}_i[x_j := 1]$
10: **else** # *$\overline{x}_j$ occurs in every term*
11: $a_i := 0$
12: $b_i := 0$
13: $\mathcal{F}_{i+1} := \mathcal{F}_i[x_j := 0]$
14: **endif**
15: $\pi(j) := i$
16: $i := i + 1$
17: **done**
18: **for every** variable x_j in $\mathcal{F}_i$ **do**
19: transform $\mathcal{F}_i[x_j := 0]$ into a prime and irredundant DNF $\mathcal{F}_i^0$
20: transform $\mathcal{F}_i[x_j := 1]$ into a prime and irredundant DNF $\mathcal{F}_i^1$
21: **if** $\mathcal{F}_i^0 \in \mathcal{C}_{1-\text{int}}^+ \wedge \mathcal{F}_i^1 \in \mathcal{C}_{1-\text{int}}^-$ and both are interval with respect to the same ordering π' of the remaining variables **then**
22: $\pi(j) := i$
23: $a_i := 0$
24: $b_i := 1$
25: Find the interval $[a', 2^{n-i} - 1]$ representing $\mathcal{F}_i^0$ with respect to the ordering π' (see Algorithm 5.5)
26: Find the interval $[0, b']$ representing $\mathcal{F}_i^1$ with respect to the ordering π' (see Algorithm 5.5 and Corollary 5.8)
27: $a := a_1 \ldots a_i a'$
28: $b := b_1 \ldots b_i b'$
29: **for all** $k \leq n$ unmapped by π **do**
30: $\pi(k) := \pi'(k) + i$ # *such k must be mapped by π'*
31: **done**
32: **output** the ordering π of variables and the values of a, b, **halt**
33: **done**
34: **output NO**

Before we prove the correctness of Algorithm 5.11, we shall give some insight in how the algorithm works. The `while` loop (steps 5 – 17) seeks at each of its tests a variable x_j, which would make either $\mathcal{F}[x_j := 1]$ or $\mathcal{F}[x_j := 0]$ empty (i.e., literal

x_j is in all terms or literal $\overline{x}_j$ is in all terms). This variable can serve us as the next one in the ordering we are searching for (according to the ideas presented before Algorithm 5.11). If no such variable exists, then we know, that if we choose any variable x_j as the next one in the ordering being constructed and if we have chosen correctly, then $\mathcal{F}[x_j := 0]$ must be a positive interval DNF and $\mathcal{F}[x_j := 1]$ must be a negative interval DNF. Since we do not know, which variable should be chosen as the next one, we simply try all the remaining variables one by one. We only have to ensure, that both $\mathcal{F}[x_j := 0]$ and $\mathcal{F}[x_j := 1]$ represent interval functions with respect to the same ordering of variables. To test this, we simply run two instances of Algorithm 5.5 in paralell and check, whether we can choose the same variable in both of these instances at each step.

The proof of the correctness of Algorithm 5.11 is split into two lemmas. Lemma 5.12 shows, that the `while` loop in steps 5 – 17 is correct. The correctness of the `if` condition in step 21 is shown in Lemma 5.13.

We present the following lemma in a form generalized for k-interval functions.

Lemma 5.12. *Let f be a Boolean function on n variables and let $\mathcal{F}$ be its prime and irredundant DNF representation. Let x_j be a variable, such that $\mathcal{F}[x_j := 0] = \emptyset$ ($\mathcal{F}[x_j := 1] = \emptyset$, resp.). Then f is a k-interval function, where $k > 1$, if and only if $\mathcal{F}[x_j := 1]$ ($\mathcal{F}[x_j := 0]$, resp.) represents a k-interval function. Moreover, if $\mathcal{F}[x_j := 1]$ ($\mathcal{F}[x_j := 0]$, resp.) can be represented by k intervals with respect to ordering π' of variables, then $\mathcal{F}$ can be represented by k intervals with respect to ordering π, such that $\pi(j) = 1$ and $\forall i \neq j$ $(\pi(i) = \pi'(i) + 1)$.*

Proof : Let $u \in \{0, 1\}$ be such that $\mathcal{F}' = \mathcal{F}[x_j := u] = \emptyset$, let $v = 1 - u$ and let f' denote the function represented by $\mathcal{F}'$.

(*if* part) Let us assume that f' can be represented by $l \leq k$ intervals $[a^{1\prime}, b^{1\prime}] < \ldots < [a^{l\prime}, b^{l\prime}]$ with respect to ordering π' of its variables. We claim that f can be represented by l intervals $[a^1, b^1] < \ldots < [a^l, b^l]$ with respect to ordering π, where $a^i = va^{i\prime}$, $b^i = vb^{i\prime}$ and ordering π is formed from π' as in the proposition of the lemma. Let $\boldsymbol{x}$ be a vector of length n, such that the bit x_j has value v. Then $f(\boldsymbol{x}) = 1$, if and only if $f'(\boldsymbol{x}') = 1$, where $\boldsymbol{x}'$ is formed from $\boldsymbol{x}$ by removing its j-th bit, and this holds, if and only if $\boldsymbol{x}' \in \cup_{i=1}^{l}[a^{i\prime}, b^{i\prime}]$. This condition is equivalent to $\boldsymbol{x} \in \cup_{i=1}^{l}[a^i, b^i]$. When the j-th bit of $\boldsymbol{x}$ has value u, it is $f(\boldsymbol{x}) = 0$ according to the assumptions, hence the proof of the first implication is complete.

(*only if* part) Now let us suppose that f is a k-interval function. According to Corollary 3.14, f' is also a k-interval function. The same arguments as in the *if* part show, that also in this case f can be represented by $l \leq k$ intervals with respect to ordering π defined as in the proposition of the lemma. ■

Lemma 5.13. *Let f be a Boolean function on n variables and let $\mathcal{F}$ be a DNF, which represents f. If $\mathcal{F}[x_j := 0] \neq \emptyset$ and $\mathcal{F}[x_j := 1] \neq \emptyset$ holds for every $j \in \{1, \ldots, n\}$, then f is an interval function, if and only if there is a variable x_j, such that the function f_0 represented by $\mathcal{F}_0 = \mathcal{F}[x_j := 0]$ is a positive interval function and the function f_1 represented by $\mathcal{F}_1 = \mathcal{F}[x_j := 1]$ is a negative interval function and both these functions are interval with respect to the same ordering π' of variables $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$.*

Proof : (*only if* part) Let us at first assume that f is interval with respect to some ordering π. Let x_j be the most significant variable with respect to π, i.e.,

$\pi(j) = 1$. The fact that $\mathcal{F}[x_j := 0] \neq \emptyset$ implies $f_0 = f[x_j := 0] \not\equiv 0$. Similarly, $f_1 = f[x_j := 1] \not\equiv 0$. Since f is an interval function with respect to π and there is a truepoint with the most significant bit having the value 0 as well as a truepoint with the most significant bit having the value 1, it must be the case that $f(\boldsymbol{x}) = 1$, where $\boldsymbol{x}$ has the most significant bit 0 and all other bits 1, and similarly $f(\boldsymbol{y}) = 1$, where y has the most significant bit 1 and all other bits 0. Moreover, according to Lemma 3.13, both f_0 and f_1 are interval functions with respect to ordering π' formed from π by restriction on variables other than x_j. Hence f_0 must be a positive interval function with respect to the ordering π' and similarly f_1 must be a negative interval function with respect to π'. Clearly, x_j is the variable, the existence of which we claim in the proposition of the lemma.

(*if* part) Now let us suppose, that there is a variable x_j and ordering π' of the remaining variables, such that $\mathcal{F}[x_j := 0]$ represents a positive interval function defined by interval $[a', 2^{n-1} - 1]$ with respect to the ordering π', and that $\mathcal{F}[x_j := 1]$ represents a negative interval function defined by interval $[0, b']$ with respect to π'. Then clearly $\mathcal{F}$ represents an interval function defined by the interval $[a = 0a', b = 1b']$ with respect to the ordering π formed from π' by adding x_j as the most significant variable. ■

Theorem 5.14. *Algorithm 5.11 works correctly and takes $O(l \cdot n \cdot p(l))$ time for transformations into prime and irredundant representations and $O(l \cdot n)$ time for the remaining work, where l is the length (the number of literals) of the input formula and $p(l)$ is the time necessary for testing falsifiability of a DNF of length (at most) l.*

Proof : The correctness of the algorithm follows using Lemma 5.12, Lemma 5.13, and the induction on the number of steps of Algorithm 5.11. It is, of course, based on the correctness of Algorithm 5.5.

We shall proceed by induction on i. Let V_i denote the set of variables of the DNF $\mathcal{F}_i$ and let f_i denote the function represented by $\mathcal{F}_i$. We shall show, that before we pass the condition in step 21, f_i is an interval function with respect to some ordering π_i of V_i, if and only if f is an interval function with respect to ordering π, which has already set by the algorithm the $i-1$ most significant variables from $\{1, \ldots, n\} \setminus V_i$ and which agrees with π_i on the variables from V_i, i.e., $\pi(j) = \pi_i(j) + i - 1$ for $x_j \in V_i$. The proposition holds for $i = 0$, since $\mathcal{F}_i = \mathcal{F}$. Let us suppose, that it is true for $1, \ldots, i$. By the induction hypothesis, f_i is an interval function with respect to some ordering π_i, if and only if f is an interval function with respect to π, which extends π_i. Let x_j be the variable, which has been chosen in i-st step, i.e., $\pi(j) = i$. If x_j has been chosen in step 5, then it satisfies the conditions of Lemma 5.12 and hence f_i is an interval function with respect to some ordering π_i, where $\pi_i(j) = 1$, if and only if f_{i+1} is an interval function with respect to π_{i+1}, which is formed from π_i by restriction on the remaining variables $V_{i+1} = V_i \setminus \{x_j\}$. It follows, that when the algorithm leaves the `while` cycle (steps 5 – 17), f_i is an interval function, if and only if f is an interval function. If x_j has been chosen in step 21, then the proposition of the induction follows from Lemma 5.13.

The time requirements of the algorithm are quite straightforward. The complexity of the steps transforming a DNF into a prime and irredundant form follows from the proof of Lemma 5.10. The test in step 5 can be implemented to take only $O(n)$ time, since e.g. the test whether $\mathcal{F}[x_j := 0]$ only means, whether x_j is present

as a positive literal in every term of $\mathcal{F}$. If we have an array in which we store for each variable the number of terms, in which it appears positively, and another array counting for each variable the number of terms, in which it appears negatively, then the appropriate x_j can be found in $O(n)$ time. These arrays can be updated during partial assignment without changing its asymptotic time requirement $O(l)$. Hence the `while` cycle (steps 5 – 17) requires together time $O(n \cdot l)$.

It only remains to describe, how to perform the `if` test in step 21. For each x_j we run two instances of Algorithm 5.5 in parallel for $\mathcal{F}[x_j := 0]$ and $\mathcal{F}[x_j := 1]$. When we are choosing the next variable in the ordering, we ensure, that the same variable can be chosen in both instances. It should be clear from the fact that Algorithm 5.5 is correct (Theorem 5.6) and the way how it works, that this approach is correct as well. In order to achieve the desired time requirements, we can proceed as follows. Let $\mathcal{F}_0$ denote $\mathcal{F}[y := 0]$, similarly let $\mathcal{F}_1$ denote $\mathcal{F}[y := 1]$. Let I_0 denote the set of variables, which appear in every term of $\mathcal{F}_0$ or which appear as a linear term in $\mathcal{F}_0$. Let I_1 denote the similar set for $\mathcal{F}_1$. Note, that the sets I_0 and I_1 together with DNFs $\mathcal{F}_0$ and $\mathcal{F}_1$ may change in every step of Algorithm 5.5. Let us denote $I = I_0 \cap I_1$. This set then consists of exactly those variables, which can be chosen as the next ones in the ordering of variables, with respect to which f can be represented by a single interval.

Now we shall describe, how to maintain the set I throughout the parallel running of two instances of Algorithm 5.5. At each step when we choose a variable from I and perform appropriate partial assignments in $\mathcal{F}_0$ and $\mathcal{F}_1$, some new variables may be added into I_0 and I_1. We note, that the instance of Algorithm 5.5 for $\mathcal{F}_0$ implemented as described in the proof of Theorem 5.7 in fact already implicitly maintains the set I_0, since this set consists of the appropriate prefix of the list `vars` and of the appropriate prefix of the list `terms`, hence it suffices to remember indices of the first nonlinear term in `terms` and of the first variable, which does not appear in all terms of $\mathcal{F}_0$, in `vars`. If we perform a partial assignment in $\mathcal{F}_0$, these indices are incremented so that they point to the appropriate elements. For each variable newly added to I_0 we test, whether it is also present in I_1, which can be done in constant time using the data structures of `vars` and `terms`. The same holds for $\mathcal{F}_1$ and I_1. Hence, for each such variable we can test, whether it should be added into I in constant time. It should be clear that the maintenance of I can be implemented to include only two $O(1)$ tests per variable, once it is added into I_0 and once it is added into I_1, since the variable is removed from I_1 or I_0 only, when it is used as the next one in the ordering being constructed. Summing over all variables, the maintenance of I takes $O(n)$ time over the entire execution of two instances of Algorithm 5.5. According to Theorem 5.7, the runs of these two parallel instances itself take $O(l)$ time, if appropriately implemented. The fact, that we are not generally using the first elements of the lists `vars` and `terms` does not imply any additional time requirements, because `vars` and `terms` are implemented as double linked lists. Since we run them for each possible x_j in the worst case, the test in step 21 takes $O(n \cdot l)$ time. Each of the remaining steps takes linear time, hence the time requirements follow. ■

5.4 Renamable 1-interval funtion recognition

It is not hard to see, that interval functions are not closed under variable complementation. Consider e.g., the function defined by the following DNF:

$$\mathcal{F} = b\overline{a} \vee c\overline{a} \vee a\overline{b} \vee c\overline{b} = (a \vee b \vee c)(\overline{a} \vee \overline{b}) \ .$$

This DNF represents the interval function defined by interval $[1, 5]$ with respect to the alphabetical ordering of variables. However, if we complement variable b, we get

$$\mathcal{F}' = \overline{b}\overline{a} \vee c\overline{a} \vee ab \vee cb \ ,$$

which does not represent an interval function with respect to any ordering of variables. This can be observed e.g., using the fact, that vectors (000) and (111) are true-points of the function represented by $\mathcal{F}'$ and vector (101) (i.e., $a = 1, b = 0, c = 1$) is its falsepoint and obviously for any ordering π it is $(000)^\pi < (101)^\pi < (111)^\pi$.

Hence it is natural to ask, if it is possible to recognize, whether a given prime and irredundant DNF can be switched to represent an interval function by changing the polarity of some of its variables. First, we shall formalize the notion of renaming.

Definition 5.15. *Let $\mathcal{F}$ be a DNF on n variables and let $S \subseteq \{1, \ldots, n\}$ be a set indexing a subset of variables. Let us define $\mathcal{F}^S$ to be a DNF, which is produced from $\mathcal{F}$ by replacing all occurences of x_i by $\overline{x}_i$ and all occurences of $\overline{x}_i$ by x_i for every $i \in S$, and by leaving all other literals (corresponding to variables x_i, $i \notin S$) unchanged. Boolean function f on n variables is called a* **renamable (positive) interval function***, if there exists DNF $\mathcal{F}$ representing f and index set $S \subseteq \{1, \ldots, n\}$, such that $\mathcal{F}^S$ represents a (positive) interval function. We denote by $\mathcal{C}^R_{\text{int}}$ the class of renamable interval functions, by $\mathcal{C}^{R+}_{\text{int}}$ the class of renamable positive interval functions and by $\mathcal{C}^{R-}_{\text{int}}$ the class of renamable negative interval functions.*

Let us at first observe, that recognition of renamable positive (or negative) interval functions can be done easily using the algorithms we have already constructed.

Lemma 5.16. *Let $\mathcal{F}$ be a prime DNF on n variables and let S be an index set, which consists of indices of all variables, which constitute negative literals in $\mathcal{F}$. Then $\mathcal{F}$ represents a renamable positive interval function, if and only if $\mathcal{F}^S$ represents a positive interval function.*

Proof : The proof rests on the fact, that a prime DNF of a positive function contains only positive literals. Thus a prime DNF $\mathcal{F}$ represents a renamable positive function, if and only if every variable of $\mathcal{F}$ constitutes only positive or only negative literals. Variables that fulfil the latter constitute the set S. ■

Proposition of Lemma 5.16 can be easily reformulated for negative functions.

Corollary 5.17. *Let $\mathcal{F}$ be a prime DNF on n variables and let S be an index set, which consists of indices of all variables, which constitute positive literals in $\mathcal{F}$. Then $\mathcal{F}$ represents a renamable negative interval function, if and only if $\mathcal{F}^S$ represents a negative interval function.*

Therefore, it is possible to use Algorithm 5.5 to recognize positive and negative renamable 1-interval functions. We just need to preprocess the input DNF

by complementing all positive or negative literals. If it is not possible to rename the input DNF to a positive or negative DNF, then we output `NO`. Otherwise, we run Algorithm 5.5 and output its answer possibly together with the set of switched variables.

Moreover, the properties of positive and negative prime DNFs yield the following statement.

Lemma 5.18. *Let $\mathcal{F}$ be a DNF on n variables representing function f. Then $f \in \mathcal{C}^{R+}_{\text{int}}$, if and only if $f \in \mathcal{C}^{R-}_{\text{int}}$.*

Proof : Clearly, $\mathcal{F}^S$ represents a positive interval function (implying $f \in \mathcal{C}^{R+}_{\text{int}}$), if and only if $\mathcal{F}^T$, where $T = \{1, ..., n\} \setminus S$, represents a negative interval function (implying $f \in \mathcal{C}^{R-}_{\text{int}}$). ■

Now we shall show, that the recognition of general renamable interval functions can be done using only a slightly modified Algorithm 5.11. The `while` cycle in steps 5 – 17 does not have to be changed at all, since it does not matter, whether $\mathcal{F}_i[x_j := 0] = \emptyset$ or $\mathcal{F}_i[x_j := 1] = \emptyset$, these cases are completely symmetrical. Step 21 will now test, whether $\mathcal{F}^0_i \in \mathcal{C}^{R+}_{\text{int}} \wedge \mathcal{F}^1_i \in \mathcal{C}^{R-}_{\text{int}}$ and whether both are renamable interval functions with respect to the same renaming and ordering of variables. Due to Lemma 5.18 it is not necessary to test the case, when $\mathcal{F}^1_i \in \mathcal{C}^{R+}_{\text{int}} \wedge \mathcal{F}^0_i \in \mathcal{C}^{R-}_{\text{int}}$. Thus, with the above described small modification of step 21, Algorithm 5.11 correctly tests, whether given DNF $\mathcal{F}$ represents a function belonging to $\mathcal{C}^R_{\text{int}}$. Hence, we have shown the following:

Theorem 5.19. *Let $\mathcal{F}$ be a DNF on n variables representing function f. If $\mathcal{F}$ belongs to a class of DNFs satisfying conditions of Theorem 5.9, then it is possible to check in polynomial time, whether $f \in \mathcal{C}^R_{\text{int}}$.*

5.5 Positive and negative 2-interval function recognition

In this subsection we will consider a generalization of the problem of recognizing a 1-interval function. We will focus on positive functions, which can be represented by (at most) two intervals and present a polynomial algorithm recognizing positive 2-interval functions. Since 1-interval functions constitute a (proper) subclass of 2-interval functions and the algorithm presented here has the same asymptotic time complexity as Algorithm 5.5 recognizing positive 1-interval functions, our new algorithm extends the previous result.

The algorithm will solve the following problem: given a positive prime DNF $\mathcal{F}$ (which is therefore also irredundant, because duplicate terms could be easily eliminated) representing function f decide, whether f is a 2-interval function, and in the affirmative case also output an ordering π of its variables and (at most) 2 intervals, which represent f with respect to π.

Let us start with the high-level idea of the algorithm. It constructs step by step the ordering of variables, with respect to which f may, eventually, be represented by two (or less) intervals. It always looks for a suitable candidate for the most significant variable among the remaining variables. If it finds one, it fixes its value (to a suitable constant) in the input DNF and continues with the resulting DNF. If there is no suitable candidate, then it outputs `NO` and stops.

The algorithm has two parts. In the first part, it looks for the candidate for the most significant variable among the variables forming linear terms in $\mathcal{F}$ and variables contained in every term of $\mathcal{F}$. The idea is, that if there is a linear term x_i, then all vectors having the i-th bit 1 are truepoints, hence we can fix the value of x_i in $\mathcal{F}$ to 0, because the input DNF represents a positive 2-interval function, if and only if the DNF $\mathcal{F}[x_i := 0]$ represents a positive 2-interval function (this will be proved later). Similarly, when there is a variable x_i in every term of $\mathcal{F}$, then all vectors having the i-th bit 0 are falsepoints, hence we can proceed with $\mathcal{F}[x_i := 1]$.

After this process terminates and if the resulting DNF $\mathcal{F}'$ representing function f' is not empty, the algorithm looks for a variable x_j, such that both $f'_0 = f'[x_j := 0]$ and $f'_1 = f'[x_j := 1]$ are positive 1-interval functions with respect to the same ordering of variables. In that case, we can use the two intervals representing f'_0 and f'_1 to represent f'. We prove later, that if there is no such variable x_j, then f' (and consequently also the function f represented by the input DNF $\mathcal{F}$) cannot be represented by 2 intervals.

Now we are ready to formulate the algorithm recognizing positive 2-interval functions.

Algorithm 5.20. 2-Interval-Recognition($\mathcal{C}^+$)

Input: *A nonempty positive prime DNF $\mathcal{F}$ on n variables representing a function f.*

Output: **0** *if f is identically equal to zero. The ordering π of variables and either interval $[a, 2^n - 1]$, where a is an n-bit integer, if f is a positive 1-interval function represented by such interval with respect to π, or intervals $[a^1, b^1]$ and $[a^2, 2^n - 1]$, where a^1, b^1, a^2 are n-bit integers, if f is a positive 2-interval function represented by such intervals with respect to π.* **NO** *otherwise (f cannot be represented by 2 or less intervals with respect to any ordering).*

1: **if** $\mathcal{F}(11\ldots1) = 0$ **then** f is constant 0, **output 0**, **halt endif**
2: **if** $\mathcal{F}(00\ldots0) = 1$ **then** f is constant 1, **output** $[0, 2^n-1]$ and any ordering, **halt endif**
3: $a^1 := 0$ *# a^1 is an n-bit integer with all bits initialized to 0*
4: $i := 1$
5: $\mathcal{F}_1 := \mathcal{F}$
6: **while** $\mathcal{F}_i \neq \emptyset$ **and** $i \leq n$ **and** $\exists j$ $(\mathcal{F}_i[x_j := 0] = 0 \vee \mathcal{F}_i[x_j := 1] = 1)$ **do**
7: **if** $\mathcal{F}_i[x_j := 0] = 0$ **then** *# x_j appears in every term*
8: $a^1_i := 1$
9: $\mathcal{F}_{i+1} := \mathcal{F}_i[x_j := 1]$
10: **else** *# x_j forms a linear term*
11: $\mathcal{F}_{i+1} := \mathcal{F}_i[x_j := 0]$
12: **endif**
13: $\pi(i) := j$
14: $i := i + 1$
15: **done**
16: **if** $\mathcal{F}_i = \emptyset$ **then** *# in this case f can be represented by one interval*
17: **while** $i \leq n$ **do** *# make ordering π complete*
18: $\pi(j) := i$, for some $j \leq n$, which is not mapped yet
19: $i := i + 1$

20: **done**
21: **output** ordering π and interval $[a^1, 2^n - 1]$
22: **halt**
23: **endif**
24: $b^1 := a^2 := a^1$ # *b^1 and a^2 are n-bit integers*
25: **if** $\exists j$ $(\mathcal{F}_i[x_j := 0] \in \mathcal{C}^+_{1-\text{int}} \wedge \mathcal{F}_i[x_j := 1] \in \mathcal{C}^+_{1-\text{int}}$ and both are 1-interval with respect to the same ordering π') **then**
26: $\pi(i) := j$
27: $a^1_i := 0$
28: $b^1_i := 0$
29: $a^2_i := 1$
30: Find interval $[c^1, 2^{n-i} - 1]$ representing $\mathcal{F}_i[x_j := 0]$ with respect to ordering π'
31: Find interval $[c^2, 2^{n-i} - 1]$ representing $\mathcal{F}_i[x_j := 1]$ with respect to ordering π'
32: $a^1 := a^1_1 \ldots a^1_i c^1$
33: $b_1 := b^1_1 \ldots b^1_i 11 \ldots 1$
34: $a^2 := a^2_1 \ldots a^2_i c^2$
35: **for all** $k \leq n$ unmapped by π **do** # *make ordering π complete using π'*
36: $\pi(k) := \pi'(k - i)$ # *such k must be mapped by π'*
37: **done**
38: **output** ordering π and intervals $[a^1, b^1]$ and $[a^2, 2^n - 1]$.
39: **else**
40: **output NO**
41: **endif**

Before we prove the correctness of Algorithm 5.20, we prove several lemmas, which will be used to describe the asymptotically optimal implementation. We start by stating a corollary, which is a slightly modified Lemma 4.12. We reformulate it here mainly for the reason of a different notation used in (5.1), which will be more suitable for its purpose here.

Corollary 5.21. *Let $\mathcal{F}$ be a positive prime DNF representing positive 2-interval function f, such that it does not contain any linear term and there is no variable contained in all terms of $\mathcal{F}$. Let π be any ordering, with respect to which f can be represented by 2 intervals and let y, z be the two most significant variables with respect to π. Then the DNF $\mathcal{F}$ has the following form:*

$$\mathcal{F} = yz \vee y\mathcal{F}_y \vee z\mathcal{F}_z, \tag{5.1}$$

where $\mathcal{F}_y$ and $\mathcal{F}_z$ are positive DNFs not containing variables y and z. Moreover, $\mathcal{F}_y$ and $\mathcal{F}_z$ represent positive functions, which can be represented by one interval with respect to the same ordering of variables.

Proof : Follows from Lemma 4.12 using Lemma 4.10. ■

The next lemma will help us understand the structure of a prime DNF representing a positive 2-interval function.

Lemma 5.22. *Let $\mathcal{F}$ be a positive prime DNF, which represents positive 2-interval function f and which does not contain a linear term or a variable occurring in all terms of $\mathcal{F}$. Then all the pairs of variables, for which $\mathcal{F}$ has the form (5.1), have one variable shared.*

Proof : We will observe, that it is not possible to have two disjoint pairs (x, y) and (w, z) of such variables. According to (5.1), there has to be the quadratic term xy and all other terms have to contain one of the variables x and y. However, this means, that $\mathcal{F}$ cannot contain the quadratic term wz. ■

Let us now analyze the structure of positive prime DNF $\mathcal{F}$ (with no linear terms and no variables contained in every term) representing a 2-interval function. Putting together Corollary 5.21 and Lemma 5.22 we get, that

$$\mathcal{F} = y(\bigvee_{i=1}^{k} z_i \vee \mathcal{H}) \vee \bigwedge_{i=1}^{k} z_i \mathcal{G} \tag{5.2}$$

where $\{z_1, y\}, \ldots \{z_k, y\}$ are all the pairs of variables, for which $\mathcal{F}$ has the form (5.1), and $\mathcal{G}$ and $\mathcal{H}$ are positive DNFs not containing any of the variables z_i and y. This is because for every pair $\{z_i, y\}$ all of the following conditions hold:

1. there has to be the quadratic term $z_i y$,
2. in every term there is z_i or y,
3. no variable forms a linear term and no variable is contained in every term.

Corollary 5.21 gives us a hint on how to find a candidate for the variable x_j sought in step 25 of Algorithm 5.20. We could take one of the variables from an arbitrary pair y, z, for which the DNF $\mathcal{F}_i$ has the form (5.1). However, we do not know yet, if for any variable x_j selected in the above described way there is an ordering, which has x_j as the most significant variable and with respect to which function f_i defined by $\mathcal{F}_i$ can be represented by 2 intervals (when f_i is a 2-interval function). It is clear that the pair of variables in (5.1) is symmetrical, hence we can choose any variable from such pair. The next lemma answers the question asked above. It is in fact the converse implication from Corollary 5.21.

Lemma 5.23. *Let $\mathcal{F}$ be a nonempty positive prime DNF representing a positive 2-interval function f, such that it does not contain any linear term and there is no variable contained in all terms of $\mathcal{F}$. Let $\{y, z\}$ be a pair of variables, for which $\mathcal{F}$ has the form (5.1). Then there is an ordering π, with respect to which f can be represented by two intervals and such that y, z are the two most significant variables with respect to π.*

Proof : Due to Lemma 5.22, if $\mathcal{F}$ has the form (5.1), then it can also be written in the form given by (5.2). First, we shall prove, that $\mathcal{G}$ and $\mathcal{H}$ (from (5.2)) represent positive functions g, h, which can be represented by one interval with respect to the same ordering of variables.

Because f is a 2-interval function, there is an ordering ρ, with respect to which f can be represented by two intervals. Let $\{y', z'\}$ be the pair of variables, which are the two most significant ones with respect to ρ. If the pairs $\{y, z\}$ and $\{y', z'\}$

are equal, then the proof is complete. Hence in the following we assume that these pairs differ. However, due to Corollary 5.21 f can be represented by a DNF in the form (5.1) and by Lemma 5.22 the pairs $\{y, z\}$ and $\{y', z'\}$ must intersect, so w.l.o.g. let us assume that $y = y'$. From (5.2) we have, that both z and z' are members of $\{z_1, \ldots, z_k\}$. Let us assume w.l.o.g. that $z_1 = z'$ and $z_k = z$.

According to Corollary 5.21 DNFs $\mathcal{G}_1 = \mathcal{F}[z_1 := 1][y := 0]$ and $\mathcal{H}_1 = \mathcal{F}[z_1 := 0][y := 1]$ represent positive functions g_1 and h_1, which can be represented by one interval with respect to the same ordering of variables. Let π_k be any such ordering. We have (from (5.2))

$$\begin{aligned} \mathcal{G}_1 &= \bigwedge_{i=2}^{k} z_i \mathcal{G} \\ \mathcal{H}_1 &= \bigvee_{i=2}^{k} z_i \vee \mathcal{H} \ . \end{aligned}$$

Due to Lemma 3.13 we know, that both $g = g_1[z_2 := 1][z_3 := 1] \ldots [z_k := 1]$ and $h = h_1[z_2 := 0][z_3 := 0] \ldots [z_k := 0]$ are positive functions, which can be represented by one interval with respect to the same ordering π_0 of the remaining variables, where π_0 is a restriction of π_1 to the remaining variables.

Let $\mathcal{G}_k = \mathcal{F}[z_k := 1][y := 0]$ and $\mathcal{H}_k = \mathcal{F}[z_k := 0][y := 1]$. Since these DNFs can also be written as

$$\begin{aligned} \mathcal{G}_k &= \bigwedge_{i=1}^{k-1} z_i \mathcal{G} \\ \mathcal{H}_k &= \bigvee_{i=1}^{k-1} z_i \vee \mathcal{H} \ , \end{aligned}$$

we have from Lemma 5.3 that both $\mathcal{G}_k$ and $\mathcal{H}_k$ represent positive functions, which can be represented by one interval with respect to the same ordering π_k formed from π_0 by adding $z_1, \ldots, z_{k-1}$ as the most significant variables (in any ordering).

Now, it is not hard to see, that f can be represented by two intervals with respect to the ordering π formed from π_k by adding $z_k = z, y$ as the most significant variables (in any ordering). ■

Now, we are ready to prove the correctness of Algorithm 5.20.

Theorem 5.24. *Algorithm 5.20 correctly recognizes positive 2-interval functions and can be implemented to run in $O(l)$ time.*

The first part (steps 6 – 23) of Algorithm 5.20 is in fact Algorithm 5.5 recognizing positive 1-interval functions. Therefore, if that part outputs `0` or an ordering and one interval, then the correctness of this output follows from Theorem 5.6. If this part terminates with a nonempty DNF, then due to Lemma 5.4 the input function is not a 1-interval function and the parts of π and a^1, which has been constructed so far, are correct in the sense, that if there is a 2-interval representation of the input function f, then it can be constructed by extending this π and a^1. This is justified by Lemma 5.3, because in the first part of the algorithm only variables forming linear terms or occuring in every term are taken as the most significant ones.

If the resulting DNF $\mathcal{F}_i$ is not empty, then (according to Corollary 5.21) it represents a 2-interval function, if and only if there is a variable x_j, such that both $\mathcal{F}_{i,0} = \mathcal{F}_i[x_j := 0]$ and $\mathcal{F}_{i,1} = \mathcal{F}_i[x_j := 1]$ represent 1-interval functions, moreover, with respect to the same ordering of the remaining variables. We can use two instances of the algorithm recognizing positive 1-interval functions running in parallel to check this for any variable x_j.

It remains to show how to implement Algorithm 5.20 to achieve $O(l)$ running time. Due to Theorem 5.7, which describes an $O(l)$ time implementation of positive 1-interval function recognition, it should be clear that all steps except step 25 can be easily implemented to fit in $O(l)$ time. According to Lemma 5.23, in step 25 we only need to find one variable (if it exists), for which (in pair with another variable) $\mathcal{F}_i$ has the form (5.1). To identify such pair of variables in $O(l)$ time we need to go through the DNF once, identify quadratic terms and count occurences of every variable. Then we go through all the pairs of variables forming quadratic terms and choose the first pair, for which the sum of the occurences of both variables is equal to the number of terms in the DNF $\mathcal{F}_i$ plus 1 (because of the occurences in the quadratic term formed by such pair). If there is no such pair, we conclude, that there exists no variable x_j required in step 25 and output `NO`. Otherwise, we use one variable of such pair as x_j in step 25 and test the condition of this step only for it. This is correct according to Lemma 5.23.

To see how to implement two parallel invocations of algorithm recognizing positive 1-interval functions with respect to the same ordering of variables, we refer to Theorem 5.14, where in the implementation of the algorithm recognizing general 1-interval functions we solved the same problem (although in that case we needed to run in parallel one instance of the algorithm recognizing positive 1-interval functions and one instance of the algorithm recognizing negative 1-interval functions, but these two algorithms differ only by the preprocessing and use the same data structures). ■

The recognition of negative 2-interval functions can be performed using Algorithm 5.20 in the similar way as Algorithm 5.5 was used for recognition of negative 1-interval functions. The idea of this is explained before Corollary 5.8.

5.6 Recognition of positive and negative k-interval functions with respect to a fixed ordering of variables

We will show, that for positive or negative function f it is possible to construct the interval representation with respect to a given ordering in time, which is proportional to the number of intervals in this representation, and only polynomial time (with respect to the size of the input representation of f) is needed to output each interval. In other words, we shall formulate a polynomially incremental algorithm solving the problem k-INTERVAL-RECOGNITION-FIXED for the classes of positive and negative functions.

Throughout this section, we will assume, that the fixed ordering of variables is $x_1, x_2, \ldots, x_n$, where x_1 is the most significant variable. The formulation of the recognition algorithm will abstract from the way the input function is represented and its time complexity will depend on time requirements of some basic operations on the input function, which may, naturally, depend on the function representation

used.

We shall start by explaining the idea of the algorithm. It will start with variable x_1 and inspect, whether index 1 satisfies at least one of the conditions (4.2), (4.3) defined in Section 4. The implementation of this test depends on the way the input function f is represented. Now we will distinguish the four cases, which may occur:

(a) Index 1 satisfies both of the conditions. In this case the input function f depends only on variable x_1 ($\boldsymbol{x}$ is a truepoint, if and only if $x_1 = 1$), hence f can be represented by interval $[2^{n-1}, 2^n - 1]$ with respect to the fixed ordering $x_1, \ldots, x_n$.

(b) Index 1 satisfies the condition (4.2). Then according to Lemma 5.3 the interval representation of f can be constructed from an interval representation of $f' = f[x_1 := 1]$ by adding 1 as the first bit to every integer specifying a boundary of a representing interval. Hence we can iterate with f'.

(c) Index 1 satisfies the condition (4.3). Then again according to Lemma 5.3 the interval representation of f can be constructed from an interval representation of $f' = f[x_1 := 0]$ by adding 0 as the first bit to every integer specifying a boundary of a representing interval and by extending the interval $[a, 2^{n-1} - 1]$ to $[a, 2^n - 1]$. Hence we can again iterate with f'.

(d) Index 1 does not satisfy any of the conditions. Then according to Lemma 4.8, the vector $(011\ldots1)$ constitutes a boundary of an interval, hence we can form a minimal interval representation of f by concatenating a minimal interval representation of $f_0 = f[x_1 := 0]$ and a modified minimal interval representation of $f_1 = [x_1 := 1]$ with bit 1 added as the most significant to every interval boundary. Therefore we first recursively call our algorithm on f_0 and then on f_1.

Now we can formulate the algorithm.

Algorithm 5.25. k**-Interval-Recognition-Fixed(**$\mathcal{C}^+$**)**

Input: *A positive function f on n variables in a suitable representation.*
Output: **0** *if f is identically equal to* 0*. Otherwise, intervals* $[a^1, b^1] < \ldots < [a^k, 2^n - 1]$ *constituting the interval representation of f with respect to ordering $x_1, \ldots, x_n$.*

1: **if** f(11...1)=0 **then output 0**, **halt** # *f is constant* 0 **endif**
2: **if** f(00...0)=1 **then output** $[0, 2^n - 1]$, **halt** # *f is constant* 1 **endif**
3: $p := ()$ # *empty interval*
4: **call** `recognition`$(f, n, 1, p, 2^n - 1)$
5: **halt**
6: *# recursive procedure follows, f is the input function, m number of its variables, i is the index of the most significant variable, p is a vector specifying the prefix, which should be added to every interval boundary and r is the boundary of the rightmost interval*
7: **procedure** `recognition`(function f, integer m, index i, prefix p, integer r)
8: **if** i satisfies condition (4.2) **then** $cnd_1 := true$ **else** $cnd_1 := false$ **endif**
9: **if** i satisfies condition (4.3) **then** $cnd_2 := true$ **else** $cnd_2 := false$ **endif**

```
10:    if cnd_1 and cnd_2 then
11:      a := p100...0          # m-bit integer, there may be no trailing zeros
12:      output [a, r]
13:    else if cnd_1 then
14:      p' := p1                                   # 1 added as the last bit
15:      f' := f[x_i := 1]
16:      call recognition(f', m - 1, i + 1, p', r)
17:    else if cnd_2 then
18:      p' := p0                                   # 0 added as the last bit
19:      f' := f[x_i := 0]
20:      call recognition(f', m - 1, i + 1, p', r)
21:    else
22:      f_0 := f[x_i := 0]
23:      f_1 := f[x_i := 1]
24:      p_0 := p0
25:      p_1 := p1
26:      call recognition(f_0, m - 1, i + 1, p_0, 2^(m-1) - 1)
27:      call recognition(f_1, m - 1, i + 1, p_1, r)
28:    endif
29:  end of procedure
```

Algorithm 5.25 is a quite straightforward implementation of the idea described above. The main part of the algorithm only checks, whether f is constant, all other work is done in the recursive procedure `recognition`. There the algorithm first checks, whether the index of the most significant variable satisfies conditions (4.2) and (4.3). Then it distinguishes the four cases (a)-(d) described before. In case both of the conditions are satisfied, it outputs the correct interval, which spans half of the vectors, i.e., the vectors having 1 as the most significant bit. We note, that in the case, when there is only one remaining variable, both of the conditions (4.2), (4.3) must be satisfied, hence we do not need to explicitly check `m` to end the recursion. In the cases (b)-(d) the algorithm behaves according to the description presented above. The correctness of Algorithm 5.25 should follow from the description as well.

The time requirements depend on the implementation of steps 8, 9 and the time complexity of fixing the value of a variable in a function, which is used in steps 15, 19, 22 and 23. If the complexity of testing the conditions in steps 8 and 9 is bounded by $p(l)$ and the complexity of fixing the value of a variable in such representation is bounded by $q(l)$, where l is the size of the reprentation of f, then the overall time requirements of Algorithm 5.25 are $O(kn \cdot (p(l) + q(l)))$, where k is the number of intervals output by the algorithm. Moreover, the intervals are output ordered from the interval spanning the least integers to the interval spanning the largest integers and the time needed to output the next interval is always bounded by $O(n \cdot (p(l) + q(l)))$. Therefore, if p and q are polynoms, then Algorithm 5.25 is polynomially incremental.

We shall describe the implementation of Algorithm 5.25 in case the input function is represented by a prime DNF. In this case we can again employ the data structures used in the optimal implementation of Algorithm 5.5 (see the proof of Theorem 5.7). These structures can be set up in $O(l)$ time, where l is the number of literals of the input function and it is possible to check, whether arbitrary variable x_i forms a

linear term or occurs in every term of the prime DNF, and all these checks for all variables take alltogether only $O(l)$ time. Also these data structures can be easily updated, when fixing of the value of a variable occurs, and for all variables alltogether these operations take only $O(l)$ time. In case that during the execution of the algorithm none of the conditions in steps 8, 9 is satisfied for some variable, we have to copy these data structures to be able to recursively invoke two instances of the `recognition` procedure in steps 26, 27. But this happens only $(k-1)$ times and in each case it takes $O(l)$ time. Hence we have proved the following theorem.

Theorem 5.26. *If the input function is represented by a prime DNF, then Algorithm 5.25 is a polynomially incremental algorithm, which can be implemented to run in $O(kl)$ time, where k is the minimum number of intervals needed to represent the input function (with respect to the fixed ordering) and l is the number of literals of the input DNF. The time needed to output the next interval is $O(l)$.*

Corollary 5.27. *The problems* INTERVAL-RECOGNITION-FIXED$(\mathcal{C}^+)$ *and* INTERVAL-RECOGNITION-FIXED$(\mathcal{C}^-)$ *can be solved in $O(kl)$ time. The problems k-*INTERVAL-RECOGNITION-FIXED$(\mathcal{C}^+)$ *and k-*INTERVAL-RECOGNITION-FIXED$(\mathcal{C}^-)$ *can be solved in $O(l)$ time.*

Proof : Algorithm 5.25 solves the problem INTERVAL-RECOGNITION-FIXED$(\mathcal{C}^+)$ with the specified time complexity. It can also be used to solve the problem k-INTERVAL-RECOGNITION-FIXED$(\mathcal{C}^+)$ by adding a counter of the output intervals. If it is about to output the $(k+1)$-st interval, it can output `NO` to signal, that the input function cannot be represented by k intervals with respect to the fixed ordering.

To solve the problems for negative functions, we again only need to complement all literals of the input negative DNF, then run Algorithm 5.25 on the resulting positive DNF and complement all output intervals, i.e., complement the bits of the n-bit integers defining boundaries of every output interval. ■

5.7 Interval representations of 2-monotonic and threshold Boolean functions

Variables of every 2-monotonic function f are linearly ordered by their strength in f (this ordering may not be unique). We can ask, if this ordering has any relation to the ordering, with respect to which f can be represented by the minimum number of intervals. In this subsection we show, that any 2-monotonic function can be represented by the minimum number of intervals with respect to any ordering of variables respecting their strengths.

We shall need the following property of variable strengths.

Lemma 5.28. *Let f be a 2-monotonic function on n variables and let x_i, x_j be two of its variables, such that $x_i \succeq x_j$ in f. Let x_k be a variable of f other than x_i, x_j and let $c \in \{0,1\}$ be chosen arbitrarily. Then $x_i \succeq x_j$ holds also in $f' = f[x_k := c]$.*

Proof : We proceed by a contradiction. Let $\boldsymbol{x}', \boldsymbol{y}'$ be vectors of length $n-1$ proving that $x_j \succ x_i$ in f', i.e., $x'_i = y'_j = 1$, $x'_j = y'_i = 0$, $x'_l = y'_l$ for $l \neq i, j$, $f'(\boldsymbol{x}') = 0$ and $f'(\boldsymbol{y}') = 1$. Then by adding c as the k-th bit to $\boldsymbol{x}'$ and $\boldsymbol{y}'$ we form vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, such that $f(\boldsymbol{x}) = f'(\boldsymbol{x}') = 0$, $f(\boldsymbol{y}) = f'(\boldsymbol{y}') = 1$, $x_i = y_j = 1$ and $x_j = y_i = 0$, i.e., $\boldsymbol{x}, \boldsymbol{y}$ prove that $x_j \succ x_i$ in f, which contradicts our assumptions. ■

Now let $\mathcal{F}$ be a prime DNF representation of 2-monotonic function f. According to Lemma 5.3 we know, that if variable x_i forms a linear term in $\mathcal{F}$ or occurs in every term of $\mathcal{F}$, then there is an ordering π, such that $\pi(i) = 1$, with respect to which f can be represented by the minimum number of intervals. The following lemma shows, that the position of x_i in the ordering π corresponds to the strength of x_i in f.

Lemma 5.29. *Let $\mathcal{F}$ be a prime DNF representing 2-monotonic function f. Let x_i be a variable forming a linear linear term in $\mathcal{F}$ or occuring in every term of $\mathcal{F}$. Then $x_i \succeq x_j$ holds for any other variable x_j from $\mathcal{F}$. Moreover, if x_j does not form a linear term nor occurs in every term of $\mathcal{F}$, then $x_i \succ x_j$ holds.*

Proof : When x_i forms a linear term in $\mathcal{F}$, then $f(\boldsymbol{x}) = 1$ holds for any vector $\boldsymbol{x}$ having $x_i = 1$. Therefore there can be no stronger variable x_j in $\mathcal{F}$ and hence $x_i \succeq x_j$, because f is 2-monotonic. If $x_j \neq x_i$ is a variable, which does not form a linear term in $\mathcal{F}$ (and it also cannot occur in every term of $\mathcal{F}$, since there is a linear term x_i), then vector $\boldsymbol{x}$, which has all bits zero except the i-th bit, together with vector $\boldsymbol{y}$, which has all bits zero except the j-th bit, prove, that $x_i \succ x_j$.

Similarly, when x_i occurs in every term of $\mathcal{F}$, then $f(\boldsymbol{x}) = 0$ for every vector having $x_i = 0$, and hence $x_i \succeq x_j$ for any other variable in $\mathcal{F}$. If x_j does not occur in every term of $\mathcal{F}$, then let $t = \wedge_{k \in T} x_k$ be a term in $\mathcal{F}$ not containing x_j. Let $\boldsymbol{x}, \boldsymbol{y}$ be vectors, such that $ON(\boldsymbol{x}) = T$ and $ON(\boldsymbol{y}) = T \setminus \{i\} \cup \{j\}$. Then $\boldsymbol{x}$ is a truepoint of f, since it satisfies t, and $\boldsymbol{y}$ is a falsepoint of f, because $y_i = 0$ and thus $\boldsymbol{y}$ does not satisfy any term in $\mathcal{F}$. Hence, vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ show, that $x_i \succ x_j$. ■

Now we can present the main result of this subsection.

Theorem 5.30. *Let f be a positive k-interval function, which is not a $(k-1)$-interval function, and let us assume that f is 2-monotonic. Let π be an ordering, which orders variables with respect to their strength in f, i.e., $x_{\pi^{-1}(1)} \succeq x_{\pi^{-1}(2)} \ldots x_{\pi^{-1}(n)}$. Then f can be represented by k intervals with respect to π.*

Proof : We shall proceed by induction on k. Using Corollary 4.11 and Lemma 5.29 we get, that the claim holds for positive 1-interval functions.

Now let $k > 1$ and suppose the claim holds for all $l < k$. Let $\mathcal{F}$ be a prime DNF representing f. We can assume, that in $\mathcal{F}$ there is no variable forming a linear term or occuring in every term, because Due to Lemma 5.29 such variables satisfy the proposition of the lemma we are proving and can be easily eliminated. Let σ be an ordering, with respect to which f can be represented by k intervals. If σ respects strengths of variables in f, then there is nothing to prove, because σ either directly equals π or it can be transformed to π by permuting variables of equal strength, which does not modify the corresponding branching tree at all, hence the same interval representation could be used also with respect to π. So let us assume the opposite is true, i.e., σ does not respect the strength of variables in f. In the following we will show, how to transform the interval representation with respect to σ to an interval representation with respect to π without increasing the number of intervals needed to represent f.

Let x_i be the most significant variable with respect to σ. Let $f_0 = f[x_i := 0], f_1 = f[x_i := 1]$ and let k_0, k_1 be the minimum number of intervals representing f_0 and f_1, respectively. Then due to Lemma 4.8 we know, that $k_0 < k$, $k_1 < k$

and $k_0 + k_1 \leq k$. Let π_i be an ordering of variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ formed from π by restriction on these variables. According to Lemma 5.28 π_i respects the strength of the variables in f_0 and f_1 (the strenghts are the same in f_0 and f_1 according to the same lemma). Due to the induction presumption f_0 and f_1 can be represented by the minimum number of intervals (i.e., k_0 and k_1) with respect to the ordering π_i. Let x_j be the most significant variable with respect to π_i (it is also the strongest variable in f_0 and f_1). If $x_i \succeq x_j$, then the proof is finished, because by adding x_i as the most significant variable to π_i we get ordering ρ, which is either directly equal to π, or we just need to permute some variables of equal strength to form π from ρ. Hence in the following we assume that $x_j \succ x_i$ holds. Now we know, that f can be represented by $k_0 + k_1$ intervals with respect to the ordering ρ and hence $k_0 + k_1 = k$, because $f \notin \mathcal{C}^+_{(k-1)-\text{int}}$. Next, we will show, that f can also be represented by k intervals with respect to the ordering ρ' formed from ρ by swapping x_i and x_j and then we will finish the proof by using the induction presumption once again.

Let us explain what happens, when we swap x_i and x_j in the ordering ρ. We can divide vectors from $\{0,1\}^n$ into four sets X, Y, Z, W according to the values of i-th and j-th bit:

- $\boldsymbol{x} \in X \leftrightarrow x_i = x_j = 0$,
- $\boldsymbol{x} \in Y \leftrightarrow x_i = 0, x_j = 1$,
- $\boldsymbol{x} \in Z \leftrightarrow x_i = 1, x_j = 0$,
- $\boldsymbol{x} \in W \leftrightarrow x_i = x_j = 1$.

Let us denote

- $\boldsymbol{y}^0 = \arg\min\{y^\rho \mid \boldsymbol{y} \in Y\}$,
- $\boldsymbol{y}^1 = \arg\max\{y^\rho \mid \boldsymbol{y} \in Y\}$,
- $\boldsymbol{z}^0 = \arg\min\{z^\rho \mid \boldsymbol{z} \in Z\}$,
- $\boldsymbol{z}^1 = \arg\max\{z^\rho \mid \boldsymbol{z} \in Z\}$.

The branching tree with variables ordered by ρ is on Figure 7. Clearly, the same

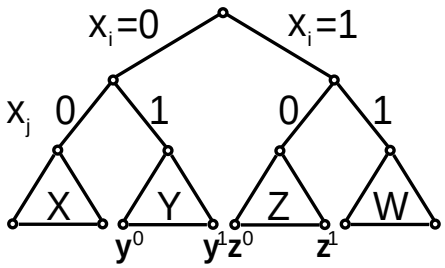


Figure 7: *Sets X, Y, Z, W and vectors $\boldsymbol{y}^0, \boldsymbol{y}^1, \boldsymbol{z}^0$ and $\boldsymbol{z}^1$ in a branching tree with variables ordered by ρ.*

vectors are minimal (maximal, resp.) within these sets with respect to the ordering

ρ'. Also, it is easy to see, that the vectors $\boldsymbol{y}^0, \boldsymbol{z}^0$ contain only one bit with value 1, hence they are both falsepoints of f, because there are no linear terms in $\mathcal{F}$. Likewise, the vectors $\boldsymbol{y}^1, \boldsymbol{z}^1$ contain only one bit with value 0, hence they are both truepoints of f, because there is no variable occuring in all terms of $\mathcal{F}$. Therefore in the interval representation of f with respect to the ordering ρ all intervals, which span at least one vector from X, Y or Z, must represent only vectors from one of the sets X, Y and Z, except the interval spanning vector $\boldsymbol{z}^1$, which may either have its right bound in $\boldsymbol{z}^1$, or may span also all vectors from W (because f is a positive function and hence if the vector representing the minimum integer among all vectors in W is a truepoint, then all vectors in W are truepoints).

For any vectors $\boldsymbol{x} \in X, \boldsymbol{y} \in Y, \boldsymbol{z} \in Z, \boldsymbol{w} \in W$ it holds that $x^\rho < y^\rho < z^\rho < w^\rho$ and $x^{\rho'} < z^{\rho'} < y^{\rho'} < w^{\rho'}$, which obviously follows from Figure 7. Hence the intervals used to span vectors from X and W with respect to the ordering ρ can be used to span the same vectors with respect to the ordering ρ'. The intervals used to span vectors from Y (Z, resp.) with respect to the ordering ρ, can be modified by exchanging the values of the i-th and j-th bits of their bounds to span the same vectors with respect to ρ'. The only exception may be the interval mentioned above, which with respect to ρ spans some vectors from Z and also all vectors from W. If there is such interval, we exchange the values of the i-th and j-th bit in its left boundary and set its right boundary to $(z^1)^{\rho'}$ and we also set the right boundary of the interval spanning $\boldsymbol{y}^1$ to $2^n - 1$. Hence f can also be represented by k intervals with respect to the ordering ρ'.

Now, to finish the proof, we consider functions $f'_0 = f[x_j := 0]$ and $f'_1 = f[x_j := 1]$. Due to Lemma 4.8 both f'_0 and f'_1 can be represented by less than k intervals, hence due to the induction presumption they can both be represented by the minimum number of intervals with respect to the ordering π_j formed from π by restriction on $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$. Hence by adding x_j as the most significant variable to π_j we form an ordering, with respect to which f can be represented by k intervals. Since x_j has the biggest strength in f among all variables, this is exactly the ordering π and the proof is complete. ■

Thus we know, that any 2-monotonic function can be represented by the minimum number of intervals with respect to any ordering of variables respecting their strength. Moreover, for positive 1-interval functions also the converse implication is true. We prove it in the following lemma.

Lemma 5.31. *Let f be a positive 1-interval function with respect to ordering π of its variables and let us assume that f is 2-monotonic. Then it holds that $x_{\pi^{-1}(1)} \succeq x_{\pi^{-1}(2)} \ldots x_{\pi^{-1}(n)}$, i.e., π is an ordering, which respects the strength of variables in f.*

Proof : Let us proceed by contradiction. We will assume that π is not an ordering with respect to the strength of variables in f. That means that there are i, j such that $\pi(i) < \pi(j)$ and $x_j \succ x_i$. Therefore there exists falsepoint $\boldsymbol{x}$, such that $x_i = 1$ and $x_j = 0$, and truepoint $\boldsymbol{y}$ formed from $\boldsymbol{x}$ by swapping i-th and j-th bit. However, in ordering π variable x_i is more significant than x_j (because $\pi(i) < \pi(j)$) and hence $x^\pi > y^\pi$. This is the desired contradiction, because $f(\boldsymbol{x}) < f(\boldsymbol{y})$ and thus positive function f cannot be interval with respect to π, since a positive interval function is always represented by an interval of the type $[a, 2^n - 1]$. ■

This converse implication does not hold for positive k-interval functions for any $k \geq 2$. We prove it by the following example of a positive 2-interval function, which can be represented by 2 intervals with respect to ordering not respecting the strenght of variables.

Example 5.32. *The following prime DNF $\mathcal{F}$ represents positive 2-interval function f, which is therefore also 2-monotonic (according to Corollary 4.15):*

$$\mathcal{F} = x_1x_2 \vee x_1x_3 \vee x_2x_3x_4 \ .$$

Function f can be represented by intervals $[6,7],[11,15]$ with respect to ordering x_2, x_1, x_3, x_4. The pair of vectors $\boldsymbol{x} = (0110), \boldsymbol{y} = (1010)$, shows that $x_1 \succ x_2$, because falsepoint $\boldsymbol{x}$ and truepoint $\boldsymbol{y}$ differ only by swapped values of bits corresponding to variables x_1, x_2. Since it can be easily observed, that $f \notin \mathcal{C}^+_{1-\text{int}}$ (due to Corollary 4.11), we have shown, that f can be represented by the minimum number of intervals with respect to an ordering not respecting the strength of variables.

When we are given a prime DNF $\mathcal{F}$ on n variables representing a 2-monotonic function, we are able to determine a linear ordering of variables with respect to their strength in $\mathcal{F}$ in $O(nl)$ (see [19]), where l is the number of literals of $\mathcal{F}$. Therefore, we have the following corollary from Theorem 5.26.

Corollary 5.33. *The problem* INTERVAL-RECOGNITION$(\mathcal{C}_{2\text{m}})$ *can be solved in $O(l(k+n))$ time and the problem k-*INTERVAL-RECOGNITION$(\mathcal{C}_{2\text{m}})$ *can be solved in $O(nl)$ time.*

Since every positive threshold function is also a 2-monotonic function, we now also know how to solve the recognition problem for the classes of positive and negative threshold functions. However, we will now describe an implementation of Algorithm 5.25, which achieves better time complexity for positive threshold functions. By the usual argument (complementing input, using algorithm for positive functions and complementing its outputs) we could show, that it can be also applied to negative threshold functions.

We use a positive threshold structure (ω, t) to represent the input positive threshold function f. Then index i satisfies condition (4.2), if and only if $\sum_{j \neq i} \omega(x_j) < t$, because this means that every vector $\boldsymbol{x}$ having $x_i = 0$ is a falsepoint. If we precompute the sum of variable weights in the beginning of the algorithm, we can perform this test in $O(\log \omega_{\max})$ time, where $\omega_{\max}$ is the maximum weight of a variable in f. Similarly, index i satisfies condition (4.3), if and only if $\omega(x_i) \geq t$, because this means that every vector $\boldsymbol{x}$ having $x_i = 1$ is a truepoint. Thus this test can be also perfomed in $O(\log \omega_{\max})$ time. Fixing the value of a variable x_i to $c \in \{0,1\}$ in f can be achieved by decreasing the threshold by $c \cdot \omega(x_i)$. Again this operation can be done in $O(\log \omega_{\max})$.

Corollary 5.34. *The problems* INTERVAL-RECOGNITION$(\mathcal{C}^+_{\text{th}})$ *and* INTERVAL-RECOGNITION$(\mathcal{C}^-_{\text{th}})$ *can be solved in $O(kn \cdot \log \omega_{\max})$ time. The problems k-*INTERVAL-RECOGNITION$(\mathcal{C}^+_{\text{th}})$ *and k-*INTERVAL-RECOGNITION$(\mathcal{C}^-_{\text{th}})$ *can be solved in $O(n \cdot \log \omega_{\max})$ time.*

6 Interval Extensions of pdBf

In this section we study the extension problem for several subclasses of interval functions. First, we present a polynomial-time algorithm deciding whether there is a positive 1-interval function extending a given pdBf. Then we generalize this result to all 1-interval functions and finally to renamable 1-interval functions. The ideas leeding to polynomial-time algorithms are usually similar to those used in the previous section to develop algorithms for the recognition problem.

6.1 Positive and negative 1-interval extensions of pdBf

We start with the extension problem for the class of positive 1-interval functions. We present a polynomial-time algorithm solving this problem. It is based on similar ideas as Algorithm 5.5, namely on Lemma 5.3 and Lemma 5.4. In every iteration, the algorithm will find out, whether it is possible to extend a given pdBf to a positive function, such that in its prime DNF there is a variable occuring in every term or forming a linear term. If it finds such variable, it fixes its value and iterates until it processes all variables. Otherwise there is no positive interval extension of the input pdBf.

Algorithm 6.1. Extension($\mathcal{C}^+_{1-\text{int}}$)

Input: *A pdBf (T, F) on n bits.*
Output: **0**, *if $T = \emptyset$. Ordering π of variables and n-bit number a, if there is a positive interval function, which extends (T, F) and is represented by interval $[a, 2^n - 1]$ with respect to ordering π.* **NO** *otherwise.*

1: **if** $T = \emptyset$ **then output 0**, **halt endif**
2: **if** $F = \emptyset$ **then output** integer 0, **halt endif**
3: $I := \{1, \ldots, n\}$
4: **for** $i := 1$ **to** n **do**
5: **if** $(\exists j \in I)\ (\forall \boldsymbol{u} \in T)\ (u_j = 1)$ **then**
6: $\pi(j) := i$
7: $a_i := 1$
8: $F := F \setminus \{\boldsymbol{v} \in F \mid v_j = 0\}$
9: $I := I \setminus \{j\}$
10: **else if** $(\exists j \in I)\ (\forall \boldsymbol{v} \in F)\ (v_j = 0)$ **then**
11: $\pi(j) := i$
12: $a_i := 0$
13: $T := T \setminus \{\boldsymbol{u} \in T \mid u_j = 1\}$
14: $I := I \setminus \{j\}$
15: **else**
16: **return NO**, **halt**
17: **endif**
18: **done**
19: **output** the ordering π and the integer a

The index set I contains at each time indices, which have not been considered yet and we restrict our attention only to these indices. Algorithm 6.1 looks at every

step for an index j, which satisfies the condition in step 5 or 10. If the condition in step 5 is satisfied, then each truepoint has the j-th bit equal to 1. In this case we assume, that the positive interval extension f, which is being constructed, satisfies that if $f(\boldsymbol{x}) = 1$ for some vector $\boldsymbol{x}$, then $x_j = 1$, and hence conversely if $x_j = 0$ then $f(\boldsymbol{x}) = 0$. Due to this assumption, we can discard all the falsepoints $\boldsymbol{v}$, for which $v_j = 0$, and we do that in step 8. A similar situation is, when each falsepoint $\boldsymbol{v}$ has $v_j = 0$. In this case we assume, that the extension has the similar property, in particular, that if $f(\boldsymbol{x}) = 0$ for some vector $\boldsymbol{x}$, then $x_j = 0$, and hence if $x_j = 1$, then $f(\boldsymbol{x}) = 1$. Due to this assumption we can discard all the truepoints $\boldsymbol{u}$, for which $u_j = 1$ (step 13), since these are not interesting anymore.

The proof of the correctness of Algorithm 6.1 rests on lemmas 6.2, 6.3, and 6.4. The first one shows that the actions, which follow a successful test in step 5, are correct. The second one shows that the actions, which follow a successful test in step 10, are correct and, finally, the third one proves, that if neither of the two tests succeeds, then there exists no positive interval extension of the given input pdBf.

Lemma 6.2. *Let (T, F) be a pdBf and let us assume, that $u_1 = 1$ holds for all $\boldsymbol{u} \in T$. Let T' consist of all vectors from T with the first bit omitted, let F' consist of all vectors $\boldsymbol{v}$ from F with $v_1 = 1$ and with the first bit omitted. Then (T, F) has a positive interval extension, if and only if (T', F') has a positive interval extension. Moreover, if (T', F') has a positive interval extension f' with respect to some ordering π' of variables $x_2, \ldots, x_n$, then function $f(x) = x_1 \wedge f'(\boldsymbol{x}')$, where $\boldsymbol{x}'$ is the vector $\boldsymbol{x}$ with the first bit omitted, is a positive interval extension of (T, F) with respect to the ordering of variables $x_1, x_2, \ldots, x_n$ formed from π' by adding x_1 as the most significant variable.*

Proof : (*only if* part) Let us at first assume, that (T, F) has a positive interval extension f. Then the function $f' = f[x_1 := 1]$ clearly extends (T', F'), moreover f' is an interval function according to Corollary 3.14.

(*if* part) Now, let us suppose, that (T', F') has an extension f', which is a positive interval function with respect to some ordering π' of variables $x_2, \ldots, x_n$. We shall show, that $f(\boldsymbol{x}) = x_1 \wedge f'(\boldsymbol{x}')$ is a positive interval extension of (T, F). To see, that f is a positive interval function, it suffices to take a prime DNF $\mathcal{F}'$ representing f'. Now DNF $\mathcal{F} = x_1 \wedge \mathcal{F}'(\boldsymbol{x}')$ is a DNF representing f, such that x_1 is a variable, which appears in every term of $\mathcal{F}$. According to Lemma 5.3 DNF $\mathcal{F}$ represents a positive interval function with respect to the ordering formed from π' by adding x_1 as the most significant variable.

Hence it remains to show, that f is an extension of (T, F), which is quite clear. Given a vector $\boldsymbol{u} \in F$, either $u_1 = 0$, in which case $f(\boldsymbol{u}) = 0$, or $u_1 = 1$, in which case $\boldsymbol{u}' \in F'$, $f(\boldsymbol{u}) = f'(\boldsymbol{u}')$, and $f'(u')$ extends (T', F'), hence also in this case $f(\boldsymbol{u}) = 0$. Given $v \in T$, surely $v_1 = 1$ holds, and thus $f(\boldsymbol{v}) = f'(\boldsymbol{v}')$. Since f' extends (T', F') and $\boldsymbol{v}' \in T'$, we have $f(\boldsymbol{v}) = f'(\boldsymbol{v}') = 1$, which completes the proof. ■

Lemma 6.3. *Let (T, F) be a pdBf and let us assume, that $v_1 = 0$ holds for all $\boldsymbol{v} \in F$. Let F' consist of all vectors from F with the first bit omitted, let T' consist of all vectors u from T with $u_1 = 0$ and with the first bit omitted. Then (T, F) has a positive interval extension if and only if (T', F') has a positive interval extension. Moreover if (T', F') has a positive interval extension f' with respect to some ordering*

π' of variables $x_2, \ldots, x_n$, then function $f(\boldsymbol{x}) = x_1 \vee f'(\boldsymbol{x}')$, where $\boldsymbol{x}'$ is the vector $\boldsymbol{x}$ with the first bit omitted, is a positive interval extension of (T, F) with respect to the ordering of variables $x_1, x_2, \ldots, x_n$ formed from π' by adding x_1 as the most significant variable.

Proof : Since the statement and also its proof are "mirror images" of Lemma 6.2 and of its proof, we leave the details to the reader. ■

Lemma 6.4. *Let (T, F) be a pdBf and let us assume that there exists no index j, such that $u_j = 1$ holds for all $\boldsymbol{u} \in T$ or $v_j = 0$ holds for all $\boldsymbol{v} \in F$. Then there exists no positive interval extension of (T, F).*

Proof : The assumption of the lemma implies, that no matter which bit j is selected to be the most significant one, there exist two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ having value 0 in the j-th bit (i.e., $x_j = y_j = 0$) and such that $\boldsymbol{x} \in T$ and $\boldsymbol{y} \in F$, and, similarly, there exist two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ having value 1 in the j-th bit (i.e., $u_j = v_j = 1$) and such that $\boldsymbol{u} \in T$ and $\boldsymbol{v} \in F$. However, these four vectors prevent the function from being a positive interval function with respect to any ordering of variables, with respect to which the j-th bit is the most significant one. ■

Theorem 6.5. *Algorithm 6.1 works correctly and can be implemented to run in $O(n \cdot (n + |T| + |F|))$ time.*

Proof : The correctness of the algorithm follows using a simple induction on i. Lemma 6.2, Lemma 6.3, and Lemma 6.4 show, that each step is correct. We note, that each time the algorithm chooses an index j as the next element in the ordering, variable x_j plays the role of x_1 in lemmas 6.2 and 6.3. Set I contains all the time those indices, which we still take into account, the remaining ones are omitted.

The time requirements depend on the implementation. We shall describe, how to achieve running time $O(n \cdot (n + |T| + |F|))$, which is asymptotically optimal. At the begining of the algorithm we shall construct the following data structures (which are similar to the ones used in the linear time implementation of Algorithm 5.5).

- Array `A` of length n, each element `A[i]` will contain a pointer to a double-linked list of structures representing truepoints $\boldsymbol{u} \in T$ with $u_i = 1$. The head of this list will contain the number of its elements. Each element of the list will contain a pointer to the head.

- Array `B` of length n, each element `B[i]` will contain a pointer to a double-linked list of structures representing the falsepoints $\boldsymbol{v} \in F$ with $v_i = 0$. The head of this list will contain the number of its elements. Each element of the list will contain a pointer to the head.

- Each vector $\boldsymbol{u} \in T \cup F$ will be represented by a structure `S(u)`. This structure will contain an array `P(u)` of length n. If $\boldsymbol{u} \in T$ and $u_i = 1$, then `P(u)[i]` points to the element of the list representing $\boldsymbol{u}$ in `A[i]`. If $\boldsymbol{u} \in F$ and $u_i = 0$, then `P(u)[i]` points to the element of the list representing $\boldsymbol{u}$ in `B[i]`. In other cases `P(u)[i]` contains a `null` value. Moreover, each element of the list in `A[i]`, `B[i]` representing a vector $\boldsymbol{u}$ will point to the structure `S(u)`.

It should be clear that these data structures can be set up in $O(n \cdot (|T| + |F|))$ time. Using these data structures, we can test the condition in step 5 in $O(n)$ time, since we simply look at each element in array `A` and ask, whether the number of elements of the appropriate list is $|I|$. Similarly, the condition in step 10 can be tested in $O(n)$ time. Both tests are performed at most n times and hence they together require $O(n^2)$ time.

In step 8 we look at `B[j]` and delete all structures of vectors, which are present in the list in `B[j]`, each structure is deleted from all lists, in which it is present. This can be done using the pointer to the `S(v)` structure. A similar observation can be made about step 13. Total time requirements of all deletes from our data structures can be at most as high as the total number of elements in all lists, which is $O(n \cdot (|T| + |F|))$, this time is independent on the `for` cycle.

The remaining steps take constant time, since they are repeated n times, they together take $O(n)$ time. ■

Algorithm 6.1 can be also used to solve the extension problem for the class of negative 1-interval functions. It suffices to exchange the sets of truepoints and falsepoints. Therefore, to solve EXTENSION($\mathcal{C}^-_{1-\text{int}}$) on a given pdBf (T, F), we can use Algorithm 6.1 on input (F, T) and if it outputs an interval $[a, 2^n - 1]$ and ordering π, our solution is $[0, \overline{a}]$ and π, where $\overline{a}$ is the n-bit integer formed from the n-bit integer a by complementing all its bits.

Corollary 6.6. *It is possible to solve* EXTENSION*($\mathcal{C}^-_{1-\text{int}}$) for an input pdBf (T, F) on n bits in $O(n \cdot (n + |T| + |F|))$ time.*

6.2 General 1-interval extensions of pdBf

Next, we use our result from the previous subsection to solve the extension problem for the class of 1-interval functions. Again, we will use ideas from the corresponding algorithm recognizing general 1-interval functions.

Let us first introduce additional notation used only for the purpose of the following algorithm. For a set S of n-bit vectors, index $j = 1, \ldots, n$, and the set of indices $I \subseteq \{1, \ldots, n\}$ let us denote by $S|_{j=e} = \{\boldsymbol{u} \in S \mid u_j = e\}$, where $e \in \{0, 1\}$, and by $S^I = \{\boldsymbol{u}$ restricted to bits from $I \mid \boldsymbol{u} \in S\}$.

Now we are ready to formulate the algorithm.

Algorithm 6.7. Extension($\mathcal{C}_{1-\text{int}}$)

Input: *A pdBf (T, F) on n bits.*
Output: **0**, *if $T = \emptyset$. Ordering π of variables and n-bit integers a, b, if there is an interval function, which extends (T, F) and which is represented by the interval $[a, b]$ with respect to the ordering π.* **NO** *otherwise.*

1: **if** $T = \emptyset$ **then output 0**, **halt endif**
2: **if** $F = \emptyset$ **then output** integer 0, **halt endif**
3: $i := 1$
4: $I := \{1, \ldots, n\}$
5: **while** $(\exists j \in I)\ [(\forall \boldsymbol{u} \in T)\ (u_j = 1) \vee (\forall \boldsymbol{u} \in T)\ (u_j = 0)]$ **do**
6: **if** $(\forall \boldsymbol{u} \in T)\ (u_j = 1)$ **then**
7: $a_i := 1$

8: $b_i := 1$
9: $F := F \setminus \{\boldsymbol{v} \in F \mid v_j = 0\}$
10: **else**
11: $a_i := 0$
12: $b_i := 0$
13: $F := F \setminus \{\boldsymbol{v} \in F \mid v_j = 1\}$
14: **endif**
15: $\pi(j) := i$
16: $I := I \setminus \{j\}$
17: $i := i + 1$
18: **done**
19: **for** $j \in I$ **do**
20: $T_0 := (T|_{j=0})^{I \setminus \{j\}}$
21: $F_0 := (F|_{j=0})^{I \setminus \{j\}}$
22: $T_1 := (T|_{j=1})^{I \setminus \{j\}}$
23: $F_1 := (F|_{j=1})^{I \setminus \{j\}}$
24: **if** (T_0, F_0) has a positive interval extension f_0 represented by interval $[a', 2^{n-i}-1]$ **and** (T_1, F_1) has a negative interval extension f_1 represented by interval $[0, b']$ **and** both extensions are interval with respect to the same ordering π' **then**
25: $\pi(j) := i$
26: $a_i := 0$
27: $b_i := 1$
28: $a := a_1 \ldots a_i a'$
29: $b := b_1 \ldots b_i b'$
30: **for each** $j \in I$ **do**
31: $\pi(j) = \pi'(j) + i$
32: **done**
33: **output** the ordering π and the integers a, b, **halt**
34: **endif**
35: **done**
36: **output NO**

The proof of the correctness of Algorithm 6.7 is very similar to the proof of the correctness of Algorithm 5.11, since both algorithms share the same ideas, which they are based on.

Lemma 6.8. *Let (T, F) be a pdBf on n variables and let us assume that $u_1 = 1$ ($u_1 = 0$, resp.) holds for all $\boldsymbol{u} \in T$. Let T' consist of all vectors from T with the first bit omitted, let F' consist of all vectors $\boldsymbol{v}$ from F with $v_1 = 1$ ($v_1 = 0$, resp.) and with the first bit omitted. Then (T, F) has an interval extension, if and only if (T', F') has an interval extension. Moreover, if (T', F') has an interval extension f' with respect to some ordering π' of variables $x_2, \ldots, x_n$, then function $f(\boldsymbol{x}) = x_1 \wedge f'(\boldsymbol{x}')$ ($f(\boldsymbol{x}) = \overline{x}_1 \wedge f'(\boldsymbol{x}')$, resp.), where $\boldsymbol{x}'$ is the vector $\boldsymbol{x}$ with the first bit omitted, is an interval extension of (T, F) with respect to the ordering of variables $x_1, x_2, \ldots, x_n$ formed from π' by adding x_1 as the most significant variable.*

Proof : We shall only show the case when all vectors $\boldsymbol{u}$ in T have the first bit equal to 1, the latter case is analogous. The proof of this lemma is just a modification of

the proof of Lemma 6.2.

(*only if* part) Let us at first assume, that (T, F) has an interval extension f. Then the function $f' = f[x_1 := 1]$ clearly extends (T', F'), moreover, f' is an interval function according to Corollary 3.14.

(*if* part) Now, let us suppose, that (T', F') has an extension f', which is an interval function with respect to some ordering π' of $x_2, \ldots, x_n$. We shall show, that $f(\boldsymbol{x}) = x_1 \wedge f'(\boldsymbol{x}')$ is an interval extension of (T, F). Let us consider a prime and irredundant DNF $\mathcal{F}'$ representing f'. Then DNF $\mathcal{F} = x_1 \wedge \mathcal{F}'(\boldsymbol{x}')$ is a prime and irredundant DNF representing f, and, moreover, it represents an interval function according to Lemma 5.12. Hence f is an interval function. Similarly as in the proof of Lemma 6.2 it can be observed, that f extends (T, F), as well. ■

Lemma 6.9. *Let (T, F) be a pdBf on n variables and let us assume, that for any index $i = 1, \ldots, n$ there are some vectors $\boldsymbol{u}, \boldsymbol{v} \in T$ such that $u_i = 1$ and $v_i = 0$. Let T_0, F_0, T_1, F_1 be defined as in steps 20 – 23 of Algorithm 6.7. Then (T, F) has an interval extension, if and only if there exists an index j such that the following conditions hold:*

(a) (T_0, F_0) has a positive interval extension f_0, and

(b) (T_1, F_1) has a negative interval extension f_1, and

(c) f_0 and f_1 are both interval with respect to the same ordering π' of their variables.

Moreover, if f_0 is a positive interval extension of (T_0, F_0) and f_1 is a negative interval extension of (T_1, F_1), then $f(\boldsymbol{x}) = [x_j \wedge f_1(\boldsymbol{x}')] \vee [\overline{x}_j \wedge f_0(\boldsymbol{x}')]$, where $\boldsymbol{x}'$ is the vector $\boldsymbol{x}$ with the j-th bit omitted, is an interval extension of (T, F) with respect to the ordering of $x_1, x_2, \ldots, x_n$ formed from π' by adding x_j as the most significant variable.

Proof : (*only if* part) Let us at first assume that (T, F) has an interval extension f with respect to some ordering π of $x_1, \ldots, x_n$. Let j be such that $\pi(j) = 1$, i.e., x_j is the most significant variable with respect to π. Let $f_0 = f[x_j := 0]$, and $f_1 = f[x_j := 1]$. According to Lemma 3.13 both f_0 and f_1 are interval functions with respect to the ordering formed from π by restriction on $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$. Moreover, it is obvious that $f(\boldsymbol{x}) = [x_j \wedge f_1(\boldsymbol{x}')] \vee [\overline{x}_j \wedge f_0(\boldsymbol{x}')]$.

(*if* part) Now, let us assume, that there is some j, for which the conditions (a)-(c) apply. We shall show, that $f(\boldsymbol{x}) = [x_j \wedge f_1(\boldsymbol{x}')] \vee [\overline{x}_j \wedge f_0(\boldsymbol{x}')]$ is an interval extension of (T, F). First, f is an interval function, because given prime and irredundant DNFs $\mathcal{F}_0$ representing f_0 and $\mathcal{F}_1$ representing f_1, the DNF $\mathcal{F} = (x_j \wedge \mathcal{F}_1) \vee (\overline{x}_j \wedge \mathcal{F}_0)$ represents an interval function according to Lemma 5.13. It remains to show, that f extends (T, F), which is quite easy. Given $\boldsymbol{u} \in T$, either $u_1 = 1$, in which case $\boldsymbol{u}'$ with the j-th bit omitted belongs to T_1 and hence $f_1(\boldsymbol{u}') = 1$ and also $f(\boldsymbol{u}) = 1$. If $u_1 = 0$, then similarly $f_0(\boldsymbol{u}') = 1$ and $f(\boldsymbol{u})$ is again 1. The case when $\boldsymbol{u} \in F$ is similar and therefore we leave the details to the reader. ■

Theorem 6.10. *Algorithm 6.7 is correct and works in time $O(n \cdot (n + |T| + |F|))$.*

Proof : The correctnes of Algorithm 6.7 is a corollary of Lemma 6.8, Lemma 6.9 and simple induction on i.

Time requirements are straightforward, the `while` cycle in steps 5 – 18 repeats at most n times and all the steps can be performed in $O(n \cdot (n+|T|+|F|))$ time. The same holds for the `for` cycle in steps 19 – 35, where we use Theorem 6.5 to show, that the condition in the `if` statement (step 24) can be tested in $O(n \cdot (n+|T|+|F|))$ time. ■

6.3 Renamable 1-interval extensions of pdBf

Now we show, how to generalize the previous results to solve the extension problem for the classes of renamable (positive) 1-interval functions defined in Subsection 5.4. We will first construct a modification of Algorithm 6.1, which solves the extension problem for the class of renamable positive (and also negative) interval functions and based on this, we will describe, how to modify Algorithm 6.7 to solve the extension problem for the class of renamable interval functions.

We start with the idea of our algorithm finding a renamable positive interval extension. We proceed as in Algorithm 6.1 as long as there is j satisfying one of the conditions in steps 5 and 10. If there is no such j (and $i \leq n$), we look for a variable k, that can be switched (which means exchanging 1 and 0 in the k-th bit of all vectors in (T, F)) to allow the algorithm to continue. This means, when there is no variable such that all (remaining) truepoints have the corresponding bit equal to 1 (in step 5) and also no variable such that all (remaining) falsepoints have the corresponding bit equal to 0 (in step 10), we look for a bit satisfying one of the "mirrored" conditions (i.e., a bit k, such that all truepoints have k-th bit 0 or all falsepoints have k-th bit 1), switch this bit in all remaining vectors (and also the corresponding variable) and continue.

The algorithm finding a renamable positive interval extension can be thus formulated as follows:

Algorithm 6.11. Extension($\mathcal{C}^{R+}_{\text{int}}$)

Input: *A pdBf (T, F) on n bits.*
Output: **0**, *if $T = \emptyset$. Ordering π, set S and n-bit integer a, if there is a renamable positive interval function f, which extends (T, F) and can be represented by interval $[a, 2^n - 1]$ with respect to ordering π with variables from S switched.* **NO** *otherwise.*

1: **if** $T = \emptyset$ **then output 0 endif**
2: **if** $F = \emptyset$ **then output** integer 0 **endif**
3: $I := \{1, \ldots, n\}$; $S = \emptyset$
4: **for** $i := 1$ **to** n **do**
5: **if** $(\exists j \in I)\ (\forall \boldsymbol{u} \in T)\ (u_j = 1)$ **then**
6: $\pi(j) := i$
7: $a_i := 1$
8: $F := F \setminus \{\boldsymbol{v} \in F \mid v_j = 0\}$
9: $I := I \setminus \{j\}$
10: **else if** $(\exists j \in I)\ (\forall \boldsymbol{v} \in F)\ (v_j = 0)$ **then**
11: $\pi(j) := i$
12: $a_i := 0$
13: $T := T \setminus \{\boldsymbol{u} \in T \mid u_j = 1\}$

14: $I := I \setminus \{j\}$
15: **else if** $(\exists j \in I)\ (\forall \boldsymbol{u} \in T)\ u_j = 0)$ **or** $(\forall \boldsymbol{v} \in F)\ v_j = 1)$ **then**
16: $S = S \cup \{j\}$
17: switch the j-th bit of all the remaining vectors in $T \cup F$
18: **goto** 5
19: **else**
20: **output NO**
21: **endif**
22: **done**
23: **output** π, a

To prove the correctness of Algorithm 6.11, we could proceed with variants of Lemma 6.2, Lemma 6.3 and Lemma 6.4. But the proofs would be very similar. So we will only discuss the differencies.

If Algorithm 6.11 finds a renamable positive extension, then if we first switch the variables from S (by switching corresponding bits in all vectors of the input pdBf), then Algorithm 6.1 successfully finds a positive interval extension of the resulting pdBf. Therefore in this case we know, that the input pdBf really has renamable positive extension.

If Algorithm 6.11 outputs `NO`, then at some point there is no bit, in which all vectors from T or F have the same value. But this means, that no matter which variable we choose as the most significant, there will always be two truepoints, one having the most significant bit 0 and the other having the most significant bit 1, and also two falsepoints, again one having the most significant bit 0 and the other having the most significant bit 1, and these four vectors prove, that there is no positive interval extension having the selected variable as the most significant one. Moreover, it does not help to switch some variables, because by switching the most significant variable we only exchange the sets of vectors starting with 0 and 1 and by switching some other variable we only reorganize vectors within these two sets. Therefore in this case the input pdBf has no renamable positive interval extension.

The only important thing to note here is, that in case we have to switch some variable (in step 16), we can choose any suitable variable, because once some variable is suitable for renaming it remains in this state no matter which other variables we swicth or fix to a particular value (because "suitable" means that all the remaining vectors from T or F have the same value in the corresponding bit and during the execution of our algorithm we only remove bits and remove vectors). Therefore this greedy approach works.

Theorem 6.12. *Algorithm 6.11 correctly decides, whether there is a renamable positive interval extension and can be implemented in* $O(n \cdot (n + |T| + |F|))$ *time.*

Proof : The correctness follows from the preceding discussion and from the correctness of Algorithm 6.1. The time requirements follow from the time requirements of Algorithm 6.1 (we need to modify a bit the data structures used in the implementation, in particular, we need to add for each bit a counter of the number of truepoints having 0 in that bit and a counter of the number of falsepoints having 1 in that bit). ■

To find out, how to solve the extension problem for the class of renamable negative interval functions, we only need to observe that pdBf (T, F) has a renamable

positive interval extension with set S of switched variables, if and only if (T, F) has a renamable negative interval extension with set $\{1, \ldots, n\} \setminus S$ of switched variables.

It is now easy to describe, how to modify Algorithm 6.7 to solve the extension problem for the class of renamable interval functions. The steps 5 to 18 do not need to change at all, because they are completely symmetrical. Then in step 24 we need to check whether (T_0, F_0) has a renamable positive interval extension and (T_1, F_1) has a renamable negative interval extension with the same sets of switched variables and with respect to the same ordering of variables.

Theorem 6.13. *Given a pdBf (T, F) it is possible to check in $O(n \cdot (n + |T| + |F|))$ time, whether there is a renamable interval extension of (T, F).*

7 Best-Fit Interval Extensions of pdBf

We will study 1-interval best-fit extensions in this section. We show, that this generalized problem of finding an extension which makes the minimal number of errors is NP-hard in the case when all orderings of variables should be considered. When a fixed ordering is given in advance, it is possible to solve this problem in polynomial time and we present here an asymptotically optimal algorithm.

We start by proving that it is NP-hard to find positive 1-interval best-fit extension of a given pdBf even in the case, when the vectors are not weighted, e.g. when the size of the error of an extension is equal to the number of vectors incorrectly classified. Then we use this result and extend it to general 1-interval best-fit extensions, although here we present only a proof using the possibility of non-uniformly weighted vectors of the input pdBf. Then we proceed by showing, that when the ordering of variables determining their significancies is fixed, it is possible to find (positive) 1-interval extension in polynomial time and we construct asymptotically optimal algorithms solving this problem for the classes of positive and general 1-interval functions. The complexity of the best-fit problems for (positive) k-interval functions, where $k > 2$, are (to the best of our knowledge) still open.

7.1 Hardness of the general 1-interval best-fit

We will first show that it is NP-hard to find the best-fit extension from the class of positive 1-interval functions, then we will generalize this result to all 1-interval functions.

Theorem 7.1. *Problem* BEST-FIT$(\mathcal{C}^+_{1-\text{int}})$ *is NP-hard, even if all vectors have unit weights.*

Proof : We will reduce the problem of finding a minimum vertex cover, which is well-known to be NP-hard (see [13]), to the BEST-FIT$(\mathcal{C}^+_{1-\text{int}})$ problem.

Given $G = (V, E)$, an instance of the minimum vertex cover problem, we construct a pdBf (T, F), an instance of the BEST-FIT$(\mathcal{C}^+_{1-\text{int}})$ problem, as follows. Vectors in T, F have $2n$ bits, where $n = |V|$, first n bits denoted by indices $i \in V$, second n bits by indices i' for $i \in V$. The sets T, F are then

$$\begin{aligned} T &= \{\boldsymbol{x}^{\{i,j\}}, \boldsymbol{x}^{\{i',j'\}} : \{i, j\} \in E\} \\ F &= \{\boldsymbol{x}^{\{i,i'\}}, i \in V\} \ . \end{aligned}$$

All vectors will be weighted uniformly to 1. We will show that the error $\epsilon(f)$ of the best-fit positive 1-interval extension is equal to the size $\tau(G)$ of a minimum vertex cover of graph G, which will finish our proof.

First, let us show that $\min \epsilon(f) \leq \tau(G)$. Let C be some (possibly a minimum) vertex cover of graph G. We shall define a positive 1-interval function f, for which $\epsilon(f) = |C|$. This function f is represented by interval $[a, 2^{2n}-1]$, where $a = 2^{2n-2|C|}$, with respect to ordering π defined as follows:

$$\begin{aligned} \pi(i) &\in \{1, 3, \ldots, 2|C|-1\} \text{ for } i \in C \\ \pi(i) &\in \{2|C|+1, 2|C|+3, \ldots, 2n-1\} \text{ for } i \notin C \\ \pi(i') &= \pi(i)+1 \end{aligned}$$

Values of $\pi(i)$ can be chosen arbitrary from the specified sets (but all values must be used). The function f now correctly classifies all vectors from T, because C is a vertex cover of G and therefore for every edge there is at least one of its end vertices in C. The variable corresponding to this vertex (and also its prime variant) thus has weight at least $2^{2n-2|C|}$ and thus all truepoints from T fit in the interval $[a, 2^{2n}-1]$.

When considering falsepoints, it is easy to observe, that all vectors $x^{\{i,i'\}}$ for $i \in C$ are mapped to 1 by function f, because the corresponding integer value of this vector is at least $2^{2n-2|C|+1} + 2^{2n-2|C|}$ and this fits in the interval $[a, 2^{2n}-1]$. Hence, these vectors are not classified correctly by f. The other falsepoints have corresponding integer values at most $2^{2n-2|C|-1} + 2^{2n-2|C|-2} < a$ and hence they are classified correctly by f. Therefore $\epsilon(f) = |C|$ and we have proved that $\min \epsilon(f) \leq \tau(G)$.

Now, we will prove that $\min \epsilon(f) \geq \tau(G)$. Let us take any positive 1-interval function f on $2n$ variables represented by interval $[a, 2^{2n}-1]$ with respect to ordering π of its variables. Then the significancy of variable i is $\omega(i) = 2^{2n-\pi(i)}$. Let us define the following subsets of V:

$$\begin{aligned} A &= \{i \mid \omega(i) < \frac{a}{2} \ \wedge \ \omega(i') \geq \frac{a}{2}\} \\ B &= \{i \mid \omega(i) \geq \frac{a}{2} \ \wedge \ \omega(i') < \frac{a}{2}\} \\ C &= \{i \mid \omega(i) < \frac{a}{2} \ \wedge \ \omega(i') < \frac{a}{2}\} \\ D &= \{i \mid \omega(i) \geq \frac{a}{2} \ \wedge \ \omega(i') \geq \frac{a}{2}\} \end{aligned}$$

Clearly, f makes at least the following errors:

$$f(\boldsymbol{x}^{\{i,j\}}) = 0 \text{ for any } \{i,j\} \in E(A \cup C) \tag{7.1}$$

$$f(\boldsymbol{x}^{\{i',j'\}}) = 0 \text{ for any } \{i,j\} \in E(B \cup C) \tag{7.2}$$

$$f(\boldsymbol{x}^{\{i,i'\}}) = 1 \text{ for any } i \in D \ , \tag{7.3}$$

where $E(S)$ denotes the set of edges $\{i,j\} \in E$ with $i,j \in S$. Let $P \subseteq V$ be a minimum vertex cover of all edges in $E(A \cup C)$ and $Q \subseteq V$ be a minimum vertex cover of all edges in $E(B \cup C)$. We have

$$|E(A \cup C)| \geq |P|, |E(B \cup C)| \geq |Q| \ .$$

Then the set $P \cup Q \cup D$ forms a vertex cover of all edges from $E \setminus E(A;B)$, where $E(A;B) = \{\{i,j\} \in E \mid i \in A, j \in B\}$, and the size of this vertex cover is at most equal to the number of errors of f counted in (7.1),(7.2) and (7.3).

It remains to count the errors, that f makes on vectors corresponding to edges in $E(A;B)$. We claim, that f makes at least one error in each of the sets $S(i,j) = \{\boldsymbol{x}^{\{i,j\}}, \boldsymbol{x}^{\{i',j'\}}, \boldsymbol{x}^{\{i,i'\}}, \boldsymbol{x}^{\{j,j'\}}\}$ corresponding to edges $\{i,j\} \in E(A;B)$. To prove this, let us assume that there is no such error for some edge $\{i,j\}$. Then $f(\boldsymbol{x}^{\{i,j\}}) = f(\boldsymbol{x}^{\{i',j'\}}) = 1$ implies $\omega(i) + \omega(j) \geq a$ and $\omega(i') + \omega(j') \geq a$, thus $\omega(i) + \omega(j) + \omega(i') + \omega(j') \geq 2a$ follows. On the other hand $f(\boldsymbol{x}^{\{i,i'\}}) = f(\boldsymbol{x}^{\{j,j'\}}) = 0$ implies $\omega(i)+\omega(i') < a$ and $\omega(j)+\omega(j') < a$, yielding the contradiction $\omega(i)+\omega(j)+\omega(i')+\omega(j') < 2a$.

In order to count the errors due to the above reason, we remark that $S(i,j) \cap S(k,l) \neq \emptyset$ holds, if $\{i,j\} \cap \{k,l\} \neq \emptyset$. Therefore, to find a family of pairwise disjoint subsets $S(i,j)$, we define $G' = (A \cup B, E(A;B))$, which is a bipartite graph and hence we have $\rho(G') = \tau(G')$ (see e.g., [17]), where $\rho(G')$ denotes the size of a maximum matching in G'. Thus there are edges $\{i_k, j_k\} \in E(A;B)$ for $k = 1, 2, \ldots, \tau(G')$ which are pairwise disjoint (i.e., they form a matching). Then the sets $S(i_k, j_k) = \{\boldsymbol{x}^{\{i_k,j_k\}}, \boldsymbol{x}^{\{i'_k,j'_k\}}, \boldsymbol{x}^{\{i_k,i'_k\}}, \boldsymbol{x}^{\{j_k,j'_k\}}\} (\subseteq T \cup F)$ are pairwise disjoint for $k = 1, 2, \ldots, \tau(G')$, showing that f makes at least $\tau(G')$ errors in addition to (7.1),(7.2) and (7.3). In total

$$\epsilon(f) \geq |D| + |E(A \cup C)| + |E(B \cup C)| + \tau(G') \ .$$

Let $R \subseteq V$ be a minimum vertex cover of all edges in $E(A;B)$ (i.e. $|R| = \tau(G')$). Then the vertex set $D \cup P \cup Q \cup R$ covers all edges of G. Therefore, $\epsilon(f) \geq |D| + |P| + |Q| + |R| \geq \tau(G)$. ■

Theorem 7.2. *Problem* BEST-FIT$(\mathcal{C}_{1-\mathrm{int}})$ *with arbitrarily weighted vectors is NP-hard.*

Proof : We will use the just-proved NP-hardness of BEST-FIT$(\mathcal{C}^+_{1-\mathrm{int}})$ for the polynomial reduction. Consider pdBf (T,F) on n bits with weight function ω given as an input to BEST-FIT$(\mathcal{C}^+_{1-\mathrm{int}})$. We will construct pdBf (T',F') on $n+1$ bits with weight function ω' as follows:

1. add 0 as $(n+1)$-st bit to all vectors in F to form F'
2. add 0 as $(n+1)$-st bit to all vectors in T to form T_0
3. add vector $\boldsymbol{e} = (11\ldots1)$ of $n+1$ bits to T_0 to form T'
4. for all transformed vectors $\boldsymbol{x}' \in T' \cup F'$ corresponding to some vector $\boldsymbol{x} \in T \cup F$ let $\omega'(\boldsymbol{x}') = \omega(\boldsymbol{x})$
5. let $\omega'(\boldsymbol{e}) = \sum_{\boldsymbol{x} \in T \cup F} \omega(\boldsymbol{x}) + 1$

We will show that the best-fit 1-interval extension of pdBf (T',F') can be transformed into the best-fit positive 1-interval extension of pdBf (T,F). This will finish the proof.

Let f' be the best-fit 1-interval extension of (T',F'). First of all, we remark that vector $\boldsymbol{e}$ must be classified correctly by f', because its weight is so large that it is better for the best-fit extension to make error even on all other vectors than misclassify $\boldsymbol{e}$. This implies that f' is a positive 1-interval function, because it is a 1-interval function and its representing interval contains the right-most point of the whole interval $[0, 2^{n+1} - 1]$.

Function f' is represented by an interval $[a', 2^{n+1} - 1]$ with respect to ordering π' of its $n+1$ variables. The artificial variable x_{n+1} added to vectors of pdBf (T, F) is the $\pi'(n+1)$-st most significant variable. We recall that all vectors except $\boldsymbol{e}$ have 0 in the bit corresponding to this variable.

Now, let a be the number which has the bit representation similar to a' with $\pi'(n+1)$-th bit left out (a' is an $(n+1)$-bit number, a is an n-bit number). Let π be the permutation of n numbers defined as follows:

1. $\pi(i) = \pi'(i)$ for i such that $\pi'(i) < \pi'(n+1)$
2. $\pi(i) = \pi'(i) - 1$ for i such that $\pi'(i) > \pi'(n+1)$.

Let f be the function on n variables represented by interval $[a, 2^n - 1]$ with respect to the ordering π. We claim that f is the best-fit positive 1-interval extension of pdBf (T, F). We will prove it by a contradiction. Let g be a positive 1-interval function such that its error on pdBf (T, F) is lower than the error of f, i.e., $\epsilon(g) < \epsilon(f)$. Then using the inverse of the transformation used to get f from f' we could get g' which has lower error on pdBf (T', F') than f', which is the desired contradiction, because f' is the best-fit extension of (T', F'). ■

7.2 The fixed-ordering case of the 1-interval best-fit

The situation is different when we have a fixed ordering of variables given in advance. The weighted best-fit problem is then solvable in polynomial time both for positive and general 1-interval functions.

In either case, we first use a preprocessing, which given a pdBf (T, F) and a weight function $\omega : T \cup F \mapsto \mathbf{R}^+$ constructs a pdBf (T', F') and a weight function $\omega' : T' \cup F' \mapsto \mathbf{R}^+$, such that for every interval of lexicographically following truepoints (falsepoints, resp.) from T (F, resp.) only one truepoint (falsepoint, resp.) remains in T' (F', resp.) with the new weight equal to the sum of the original vectors being replaced. The resulting pdBf (T', F') is then used as an input for the main parts of the algorithms solving the best-fit problems. This preprocessing can be done in $O((n + \log \omega_{\max}) \cdot (|T| + |F|))$ time, where $\omega_{\max} = \max\{\omega(\boldsymbol{u}) \mid \boldsymbol{u} \in T \cup F\}$, if we first lexicographically sort all vectors from $T \cup F$ (using *RadixSort* this sorting can be done in $O(n \cdot (|T| + |F|))$ time) and then simply step through this sorted sequence once, replace every interval of falsepoints or truepoints by one vector from this interval and count the new weight function. We also add $O(\log \omega_{\max} \cdot (|T| + |F|))$ time for the arithmetic operations involving values of ω, because we did not limit the sizes of weights in any way.

The use of the preprocessing is justified by the fact, that it can never happen, that the optimal interval includes only one of two truepoints or falsepoints, which are neighbours in lexicographical ordering of $T \cup F$. Therefore, we can assume that the input pdBf for our algorithms does not have any neighbouring falsepoints or truepoints. This makes our algorithms simpler to explain and does not increase their time requirements.

In case of the algorithm finding the best-fit extension from the class of positive 1-interval functions we also assume, that the input of the algorithm is a sequence of vectors $\boldsymbol{u}^1, \boldsymbol{v}^1, \ldots, \boldsymbol{u}^m, \boldsymbol{v}^m$, where $\boldsymbol{u}^i \in T$ and $\boldsymbol{v}^i \in F$ for all i, such that $u^1 < v^1 < u^2 < v^2 < \ldots < u^m < v^m$. The ordering is set up by the afore-mentioned

preprocessing and it is not necessary to consider falsepoints, which are lexicographically lower than all truepoints, because they could never be spanned by the best-fit interval, and also it is not necessary to consider truepoints lexicographically greater than all falsepoints, because these would always be included in the best-fit positive interval.

Let us proceed with the main part of the algorithm solving the best-fit problem for positive 1-interval functions:

Algorithm 7.3. Best-Fit-Fixed($\mathcal{C}^+_{1-\text{int}}$)

Input: *A sequence $\boldsymbol{u}^1, \boldsymbol{v}^1, \ldots, \boldsymbol{u}^m$, where $\boldsymbol{u}^i \in T$ and $\boldsymbol{v}^i \in F$, and a weight function $\omega : T \cup F \mapsto \mathbf{R}^+$.*

Output: **0**, *if the best-fit extension is the function identically equal to zero. Otherwise, an interval $[a, 2^n-1]$ representing a positive 1-interval function f with respect to ordering $x_1, \ldots, x_n$ with $\epsilon(f)$ being the minimum among all functions from the class $\mathcal{C}^+_{1-\text{int}}$.*

```
 1: err := Σ_{i=1}^{k} ω(u^i)
 2: minerr := err
 3: a := 2^n
 4: for i := m to 1 do
 5:    err := err − ω(u^i) + ω(v^i)
 6:    if err < minerr then
 7:       minerr := err
 8:       a := u^i
 9:    endif
10: done
11: if a ≠ 2^n then
12:    output [a, 2^n − 1]
13: else
14:    output 0
15: endif
```

The idea of this algorithm is very simple. Because we are searching for a positive 1-interval function, the vector $(11\ldots1)$ must be the truepoint in all cases but the function represented by the empty interval. So we consider all intervals with the right bound equal to $2^n - 1$ and the left bound equal to the number represented by some truepoint and we choose the one with the minimum error. For optimization, we precompute the sum of the weights of all truepoints in advance.

Theorem 7.4. *The problem* BEST-FIT*($\mathcal{C}^+_{1-\text{int}}$) with a fixed ordering of variables can be solved in $O((n+\log\omega_{\max})\cdot(|T|+|F|))$ time, which is the asymptotically best time possible.*

Proof : The correctness of Algorithm 7.3 should be obvious. The algorithm itself runs in $O(\log\omega_{\max}\cdot(|T|+|F|))$ time, hence the resulting complexity is determined by the time required by the preprocessing, which is $O((n+\log\omega_{\max})\cdot(|T|+|F|))$. Moreover, $n\cdot(|T|+|F|)$ is the size of the input pdBf (T, F) and $\log\omega_{\max}\cdot(|T|+|F|)$ is the time required to process all weights, hence the running time of Algorithm 7.3

is asymptotically optimal, since every correct deterministic algorithm must at least read its whole input. ■

A similar algorithm can be used for the BEST-FIT($\mathcal{C}_{1-\text{int}}$) problem with fixed ordering of variables. It runs in $O(|T| \cdot (n + \log \omega_{\max}) \cdot (|T| + |F|))$ time. In fact, it uses $|T|$ invocations of Algorithm 7.3 (for each possible choice of the right bound of the interval) and checks all intervals having both of their bounds in some truepoints.

However, we shall construct an asymptotically optimal algorithm based on a different idea. Let us assume that the lexicographical ordering of vectors from $T \cup F$ is $\boldsymbol{u}^1, \boldsymbol{v}^1, \ldots, \boldsymbol{u}^{m-1}, \boldsymbol{v}^{m-1}, \boldsymbol{u}^m$, where $\boldsymbol{u}^i \in T$ and $\boldsymbol{v}^i \in F$ for all i. If there are any falsepoints outside the range of the truepoints, we can omit them, because they can never occur in the interval representing the best-fit extension. By a weight of an interval $[u^j, u^i]$, where $j \leq i$, we denote the sum $\sum_{k=j}^{i} \omega(\boldsymbol{u}^k) - \sum_{k=j}^{i-1} \omega(\boldsymbol{v}^k)$, i.e., we sum the weights of all the truepoints in such interval positively and the weights of all the falsepoints in such interval negatively. By $\texttt{maxright}(\boldsymbol{u}^i)$ we shall denote the weight of the interval $[u^j, u^i]$, $j \leq i$, which has the maximum weight among all such intervals. We shall show, that it is possible to compute $\texttt{maxright}(\boldsymbol{u}^i)$ for all truepoints $\boldsymbol{u}^i$ in $O((n+\log \omega_{\max}) \cdot (|T|+|F|))$ time and use it to identify the interval representing the best-fit 1-interval extension.

We claim, that the interval $[u^j, u^i]$ is the best-fit interval extension, if and only if its weight equals $\texttt{maxright}(\boldsymbol{u}^i)$ and it is the maximum value of $\texttt{maxright}(\boldsymbol{u}^i)$ for all truepoints. This is true, because the error size of the function f defined by an interval $[u^i, u^j]$ is

$$\epsilon(f) = \Omega - \rho \ , \tag{7.4}$$

where $\Omega = \sum_{i=1}^{k} \omega(\boldsymbol{u}^i)$ and ρ is the weight of the interval $[u^i, u^j]$. Hence, by maximizing the weight of the interval we minimize the error.

Now we can present our algorithm solving the best-fit problem for 1-interval functions:

Algorithm 7.5. Best-Fit-Fixed($\mathcal{C}_{1-\text{int}}$)

Input: *A sequence $\boldsymbol{u}^1, \boldsymbol{v}^1, \ldots, \boldsymbol{u}^m$, where $\boldsymbol{u}^i \in T$ and $\boldsymbol{v}^i \in F$, and a weight function $\omega : T \cup F \mapsto \mathbf{R}^+$.*
Output: *An interval $[a, b]$ representing the best-fit 1-interval extension.*

1: $max := \sum_{i=2}^{m} \omega(\boldsymbol{u}^i)$
2: $a := b := u^1$
3: $\texttt{maxright}(\boldsymbol{u}^1) := \omega(\boldsymbol{u}^1)$
4: $a^1 := u^1$
5: **for** $i := 2$ **to** m **do**
6: $\quad d :=\texttt{maxright}(\boldsymbol{u}^{i-1}) - \omega(\boldsymbol{v}^{i-1})$
7: $\quad$ **if** $d > 0$ **then**
8: $\quad\quad \texttt{maxright}(\boldsymbol{u}^i) := d + \omega(\boldsymbol{u}^i)$
9: $\quad\quad a^i := a^{i-1}$
10: $\quad$ **else**
11: $\quad\quad \texttt{maxright}(\boldsymbol{u}^i) := \omega(\boldsymbol{u}^i)$
12: $\quad\quad a^i := u^i$
13: $\quad$ **endif**
14: $\quad$ **if** $\texttt{maxright}(\boldsymbol{u}^i) > max$ **then**

15: $a := a^i$
16: $b := u^i$
17: $max :=$`maxright`$(\boldsymbol{u}^i)$
18: **endif**
19: **done**
20: **output** $[a, b]$

Lemma 7.6. *Algorithm 7.5 correctly computes* `maxright`$(\boldsymbol{u}^i)$ *for all truepoints.*

Proof : We proceed by induction. First, the value of `maxright`$(\boldsymbol{u}^1)$ is computed correctly, because there is only one interval to consider in this case, i.e., $[u^1, u^1]$. Let us suppose, that the value of `maxright`$(\boldsymbol{u}^j)$ has been computed correctly for all $j = 1, \ldots, i$ and we want to prove it for $i + 1$. One candidate for the value of `maxright`$(\boldsymbol{u}^{i+1})$ is $\omega(\boldsymbol{u}^{i+1})$ itself. However, if $\omega(\boldsymbol{v}^i)$ is not greater than or equal to `maxright`$(\boldsymbol{u}^i)$, then it pays off to concatenate the interval $[a^i, u^i]$ (having the weight `maxright`$(\boldsymbol{u}^i)$) with $[u^{i+1}, u^{i+1}]$, because the resulting interval has greater weight than $\omega(\boldsymbol{u}^{i+1})$. And as `maxright`$(\boldsymbol{u}^i)$ has correct value by now, we cannot construct an interval with a greater weight by concatenating $[u^{i+1}, u^{i+1}]$ with any other interval having its right bound in u^i. ■

Using Lemma 7.6 it is now quite obvious to prove, that Algorithm 7.5 correctly finds the interval representing the best-fit extension. Because every interval must have one of the truepoints as its right bound, we only need to find the vector $\boldsymbol{u}^i$, for which the value of `maxright`$(\boldsymbol{u}^i)$ is maximum. The interval, weight of which the maximum `maxright`$(\boldsymbol{u}^i)$ denotes, represents the best-fit 1-interval extension of the input pdBf, because when the weight of the representing interval is maximum, the error made by the extension is minimum.

Theorem 7.7. *The problem* BEST-FIT$(\mathcal{C}_{1-\text{int}})$ *with a fixed ordering of variables can be solved in* $O((n + \log \omega_{\max}) \cdot (|T| + |F|))$ *time, which is the asymptotically best time possible.*

8 Conclusion

The presented results concerning interval representations of Boolean functions are certainly only an introductory contribution in this field of a research. There are many open problems stemming from the studied problems (often as their generalizations). We summarize some of them in the following list:

(a) Recognition of positive (and negative) k-interval functions for $k \geq 3$.

(b) Recognition of somehow restricted general k-interval functions for $k \geq 2$.

(c) k-interval extensions and best-fit extensions of pdBfs for $k \geq 2$.

(d) k-interval extensions and best-fit extensions of partially defined Boolean functions with missing data (pBmd).

Regarding the recognition of positive k-interval functions it seems to us, that it should be possible to construct an algorithm for the case of positive 3-interval functions, which is based on a similar idea as Algorithm 5.20 recognizing positive 2-interval functions. This should be possible, because when we have a positive 3-interval function, the prime DNF of which has no linear terms and no variables occuring in all terms, and we branch on a suitable variable, then one of the resulting functions must be positive 1-interval (due to Corollary 4.9) and hence the ordering of variables, with respect to which this function can be represented by one interval, is quite strictly determined by Corollary 4.11. On the other hand, starting with $k = 3$ the class of positive k-interval functions is no longer a subclass of the class of 2-monotonic functions, hence it may seem reasonable to expect some substantial change in properties of positive 3-interval functions, which may also influence the complexity of their recognition.

With $k > 3$ we do not have any similar kind of determination for the ordering, because we only know, that one of the branches must be represented by at most two intervals (in case $k = 4$), but it may be the case, that this branch can be represented even by one interval, but in the ordering allowing this representation the second branch has very large interval representation. Therefore, we cannot use the algorithms we have developed so far for 1-interval and 2-interval positive functions, because these are only capable of finding a minimal interval representation. A prerequisite for recognition of positive k-interval functions for $k > 3$ thus seems to be a research of other than minimal interval representations of 1-interval and 2-interval positive functions. The situation of general k-interval function recognition is even more complicated. While, according to our recent research, it seems, that the recognition of 2-interval functions can be solved using similar ideas and under similar restrictions as recognition of 1-interval functions, for higher values of k there are at least the same problems as with positive functions. These complications also mirror in k-interval extensions for $k > 2$. Also, we expect that the best-fit extension problem remains NP-hard for higher values of k.

Extensions of pBmds ([7, 6]) are another generalization of extensions of pdBfs, where in the given vectors specifying the truepoints and falsepoints some bits may be missing, for example as a result of unavailable data in the measurement, which produced the vector. These bits are marked by some value distinct from $0, 1$. There are several definitions of an extension of pBmd presented and studied in [6]. The two most common ones are:

- The extension function must agree with all positive and negative samples for some (at least one) choice of {0,1} values for the unknown bits.
- The extension function must agree with all positive and negative samples for all possible choices of {0,1} values for the unknown bits.

References

[1] Angluin, D.
Queries and concept learning.
Machine Learning 2, 4 (1987), 319–342.

[2] Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., and Kann, V.
Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties.
Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[3] Boros, E., Gurvich, V., Hammer, P. L., Ibaraki, T., and Kogan, A.
Decomposability of partially defined boolean functions.
Discrete Appl. Math. 62, 1-3 (1995), 51–75.

[4] Boros, E., Hammer, P. L., Ibaraki, T., and Kogan, A.
Logical analysis of numerical data.
Math. Program. 79, 1-3 (1997), 163–190.

[5] Boros, E., Ibaraki, T., and Makino, K.
Error-free and best-fit extensions of partially defined boolean functions.
Information and Computation 140 (1998), 254 – 283.

[6] Boros, E., Ibaraki, T., and Makino, K.
Fully consistent extensions of partially defined boolean functions with missing bits.
Tech. Rep. 99-30, RUTCOR Research Report RRR, 2 1999.

[7] Boros, E., T.Ibaraki, and Makino, K.
Extensions of partially defined boolean functions with missing data.
Tech. Rep. 6-96, RUTCOR Research Report RRR, Rutgers University, New Brunswick, NJ, 1996.

[8] Chandra, A.-K., and Iyengar, V.-S.
Constraint solving for test case generation a technique of high level design verification.
Proceedings of the IEEE International Conference on Computer Design (ICCD) (1992).

[9] Cormen, T., Leiserson, C., and Rivest, R.
Introduction to Algorithms, Second Edition.
MIT Press and McGraw-Hill, 2001.

[10] Crama, Y., Hammer, P. L., and Ibaraki, T.
Cause-effect relationships and partially defined boolean functions.
Ann. Oper. Res. 16, 1-4 (1988), 299–325.

[11] Dechter, R., and Pearl, J.
Structure identification in relational data.
Artificial Intelligence 58 (1992), 237 – 270.

[12] DeMillo, R.-A., and Offutt, A.-J.
Constraint-based automatic test data generation.
IEEE Trans. on Software Engng. 17 (1991), 900 – 910.

[13] Garey, M., and Johnson, D.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
W.H. Freeman and Company, San Francisco, 1979.

[14] Genesereth, M., and Nilsson, N.
Logical Foundations of Artificial Intelligence.
Morgan Kaufmann, Los Altos, CA, 1987.

[15] Hammer, P., and Kogan, A.
Horn functions and their dnfs.
Information Processing Letters 44 (1992), 23 – 29.

[16] Hammer, P., and Kogan, A.
Optimal compression of propositional horn knowledge bases: Complexity and approximation.
Artificial Intelligence 64 (1993), 131 – 145.

[17] Kuhn, H.-W.
The hungarian method for solving the assignment problem.
Naval Research Logistics Quarterly 2 (1955), 83 – 97.

[18] Lewin, D., Fournier, L., Roytman, E., and Shurek, G.
Constraint satisfaction for test program generation.
Proceedings of the 14th IEEE International Phoenix Conference on Computers and Communications (1995), 45 – 48.

[19] Makino, K.
A linear time algorithm for recognizing regular boolean functions.
J. Algorithms 43, 2 (2002), 155–176.

[20] Makino, K., Hatanaka, K.-I., and Ibaraki, T.
Horn extensions of partially defined boolean functions.
SIAM Journal on Computing 28 (1999), 2168 – 2186.

[21] Muroga, S.
Threshold Logic and Its Applications.
Wiley, 1971.

[22] Quine, W.
The problem of simplifying the truth functions.
Amer.Math.Monthly 59 (1952), 521 – 531.

[23] Quinlan, J. R.
Induction of decision trees.
Mach. Learn. 1, 1 (1986), 81–106.

[24] Schieber, B., Geist, D., and Ayal, Z.
Computing the minimum dnf representation of boolean functions defined by intervals.
Discrete Applied Mathematics 149 (2005), 154 – 173.