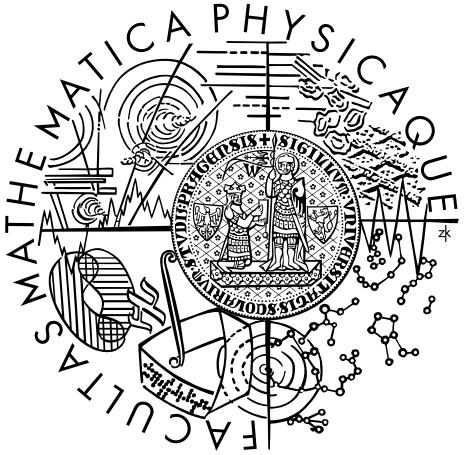
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Filip Bártek

Minimum representations of boolean functions defined by multiple intervals

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	RNDr. Petr Kučera, Ph.D.
Study programme:	Informatics
Study branch:	Theoretical Computer Science

Prague 2015

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature

Title: Minimum representations of boolean functions defined by multiple intervals

Author: Filip Bártek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Petr Kučera, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract:

When we interpret the input vector of a Boolean function as a binary number, we define interval Boolean function $f^n_{[a,b]}$ so that $f^n_{[a,b]}(x) = 1$ if and only if $a \leq x \leq b$. Disjunctive normal form is a common way of representing Boolean functions. Minimizing DNF representation of an interval Boolean function can be performed in linear time.[2] The natural generalization to k-interval functions seems to be significantly harder to tackle. In this thesis, I discuss the difficulties with finding an optimal solution and introduce a $2k$-approximation algorithm.

Keywords: Boolean minimization, disjunctive normal form, interval functions

Contents

1. Introduction

1.1 Definitions

The basic terminology used in this thesis was introduced by Schieber et al.[2] and Dubovský.[1]

Definition 1.1.1 (Binary vector)**.** In this thesis, we'll use the term *binary vector* to denote a vector over the Boolean domain. That is, x is a binary vector if and only if $\exists n.x \in \{0,1\}^n$. We call such x an *n-bit binary vector*.

Note that each binary vector represents a natural number. More specifically, n-bit binary vectors represent numbers between 0 and $2^n - 1$. We won't differentiate between a number and its binary representation. Especially, we'll use a lexicographic linear ordering on binary vectors – for n-bit vectors a and b, $a \leq b$ if and only if $a = b$ or the most significant bit in which a and b differ is 0 in a and 1 in b. Clearly, this ordering corresponds to the natural ordering on the represented numbers.

Definition 1.1.2 (Symbol extraction)**.** Let v be a vector of length n and $1 \leq i \leq n$. Then $v^{[i]}$ is the i-th component of v.

Let $1 \leq a \leq b \leq n$. Then $v^{[a,b]}$ is the subvector of v that starts at a-th position and ends at b-th position ($v^{[a,b]} = (v^{[a]}, \ldots, v^{[b]})$).

Notably, $v = v^{[1]}, \ldots, v^{[n]} = v^{[1,n]}$.

Definition 1.1.3 (Symbol repetition)**.** Let x be a symbol (for example 0 or 1) and $0 \leq n$. Then $x^{\{n\}}$ is the vector of length n each component of which is equal to x.

Definition 1.1.4 (Boolean function)**.** $f : \{0,1\}^n \to \{0,1\}$ is an *n-ary Boolean function*.

Since we won't be dealing with any non-Boolean functions, in the remainder of the text, I will use the terms ,,function" and ,,Boolean function" interchangeably.

Every n-ary Boolean function f can be expressed as a propositional formula $\mathcal{F}$ in n variables $x_1, \ldots, x_n$. There's a natural bijection between binary vectors of length n and valuations of n variables (1-bits in the binary vector correspond exactly to 1-valued variables in the valuation).

Definition 1.1.5 (Disjunctive normal form)**.** A propositional logic formula $\mathcal{F}$ is in *disjunctive normal form (DNF)* if and only if $\mathcal{F}$ is a disjunction of conjuncts, where *conjuncts* are conjunctions of literals.

We'll often view conjuncts as sets of literals and DNF formulas as sets of conjuncts.

A DNF formula is a *DNF representation* of a Boolean function if it expresses the same function, that is $\forall x \in \{0,1\}^n : f(x) = 1 \iff [[\mathcal{F}]]_x = 1$.

To make our work easier, we'll define a vector representation of a conjunct:

Definition 1.1.6 (Ternary vector)**.** A *ternary vector* is a vector over the set $\{0, 1, \phi\}$.

There's a natural bijection between ternary vectors and conjuncts that don't mix polarities of variables[1]. Each of the n components of a ternary vector T corresponds to the single occurrence (or its absence) of one of the n variables of conjunct C:

$$T \equiv C \iff \forall i \in \{1, \ldots, n\}. \begin{cases} T^{[i]} = \phi & \text{iff} \quad C \cap \{x_i, \overline{x_i}\} = \emptyset \\ T^{[i]} = 1 & \text{iff} \quad C \cap \{x_i, \overline{x_i}\} = \{x_i\} \\ T^{[i]} = 0 & \text{iff} \quad C \cap \{x_i, \overline{x_i}\} = \{\overline{x_i}\} \end{cases}$$

Namely, a *binary* vector corresponds to a full conjunct (that is one that contains every variable), and the ternary vector $\phi^{\{n\}}$ corresponds to the empty conjunct (tautology).

It's easy to see that sets of ternary vectors of length n correspond to DNF *formulas* on n variables.

Definition 1.1.7 (Spanning)**.** A ternary vector T of length n *spans* a binary vector x of length n if and only if $\forall i \in \{1, \ldots, n\}. T^{[i]} = \phi$ or $T^{[i]} = x^{[i]}$.

The i-th symbol of a ternary vector constrains the i-th symbol of the spanned binary vector (or, equivalently, the valuation of the i-th variable). If $T^{[i]}$ is a ,,fixed bit" ($T^{[i]} \in \{0, 1\}$), the spanned binary vector x must have the same value in its i-th position, that is $x^{[i]} = T^{[i]}$. If $T^{[i]}$ is the ,,don't care symbol" ϕ, the spanned binary vector may have any value in its i-th position.

Definition 1.1.8 (Spanned set)**.** Let T be a ternary vector of length n. Set of points spanned by T, denoted $span(T)$, is defined in the following way:

$$span(T) = \{x \in \{0, 1\}^n | T \text{ spans } x\}$$

Note that a ternary vector with m ϕ-positions spans 2^m binary vectors.

We'll also use a natural extension of spanning to sets of ternary vectors – if $\mathcal{T}$ is a set of ternary vectors, then

$$span(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} span(T)$$

Definition 1.1.9 (True point)**.** Let f be an n-ary Boolean function. An n-bit binary vector x is a *true point* of f if and only if $f(x) = 1$.

Equivalently, we define a *false point* of f as any point x such that $f(x) = 0$.

We'll denote the set of all true points of f as $TP(f)$ and the set of all false points as $FP(f)$.

Definition 1.1.10 (Minimum DNF representation)**.** A DNF representation of a function is *minimum* if and only if there's no representation with fewer conjuncts.

The focus of this thesis is minimizing the number of conjuncts of DNF representations of certain Boolean functions.

Note that a function may have more than one minimum DNF representation.

[1] A conjunct that mixes polarities of a variable, for example $x_1\overline{x_1}$, is trivially false. Note that leaving out such conjunct from a formula always preserves the represented function.

Definition 1.1.11 (Spanning set)**.** Let f be an n-ary Boolean function. Set $\mathcal{T}$ of n-bit ternary vectors is a *spanning set* of f if and only if $span(\mathcal{T}) = TP(f)$.

In other words, a spanning set of a function spans all of its true points and none of its false points.

Note that spanning sets of a function correspond to DNF representations of the function. The correspondence preserves size (number of ternary vectors and conjuncts, respectively), so a minimum spanning set corresponds to a minimum DNF representation.

Definition 1.1.12 (Disjoint spanning set)**.** A spanning set $\mathcal{T}$ is *disjoint* if and only if the spanned sets of its vectors don't overlap, that is

$$\forall T_i, T_j \in \mathcal{T}.T_i = T_j \text{ or } span(T_i) \cap span(T_j) = \emptyset$$

In terms of DNF, disjoint spanning sets correspond to DNF formulas such that every valuation satisfies at most one conjunct.

We have introduced two equivalent representations of Boolean functions – one based on ternary vectors and the other being the DNF representation. Table 1.1 shows the corresponding terms side by side.

ternary vector	conjunct
ternary vector set	DNF formula
binary vector	variable valuation
ternary vector *spans* a binary vector	valuation *satisfies* a conjunct
ternary vector set *spans* a binary vector	valuation *satisfies* a DNF formula

Table 1.1: Corresponding terms used in Boolean function representations

Definition 1.1.13 (k-interval Boolean function)**.** Let $a_1, b_1, \ldots, a_k, b_k$ be n-bit numbers such that $0^{\{n\}} \leq a_1$, $a_1 \leq b_1$, $b_1 \leq a_2 - 2$, …, $a_i \leq b_i$, $b_i \leq a_{i+1} - 2$, …, $b_k \leq 1^{\{n\}}$. Then $f^n_{[a_1,b_1],\ldots,[a_k,b_k]} : \{0,1\}^n \to \{0,1\}$ is a function defined as follows:

$$f^n_{[a_1,b_1],\ldots,[a_k,b_k]}(x) = \begin{cases} 1 & \text{if } x \in [a_i, b_i] \text{ for some } i \\ 0 & \text{otherwise} \end{cases}$$

Note that adjacent intervals are required to be separated by at least one false point.

Especially, $f^n_{[a,b]}$ is a (1-interval) function whose true points form the interval $[a, b]$.

2. Single interval functions

In this chapter, I will introduce an efficient algorithm for optimal spanning of single interval Boolean functions, originally shown by Schieber et al.[2]

In the chapters that follow, I will use this algorithm as a procedure in order to span multi-interval functions.

In the remainder of this chapter, a and b ($a \leq b$) will denote the endpoints of the spanned interval and n the number of input bits. Thus, we'll be spanning the function $f^n_{[a,b]}$.

The algorithm is recursive.

2.1 Trivial cases

First we'll deal with the trivial interval functions.

2.1.1 $a = b$

We span the interval $[a, a]$ with the single ternary vector a.

From now on, let $a < b$.

2.1.2 $a = 0^{\{n\}}$ and $b = 1^{\{n\}}$

We span the interval $[0^{\{n\}}, 1^{\{n\}}]$ with the single ternary vector $\phi^{\{n\}}$.

From now on, let $a > 0^{\{n\}}$ or $b < 1^{\{n\}}$.

2.2 Prefix and suffix case

Since we have dealt with the trivial cases, we are now left with the situation $0^{\{n\}} \leq a < b \leq 1^{\{n\}}$ and $a > 0^{\{n\}}$ or $b < 1^{\{n\}}$.

Let's consider the prefix case, that is $[0^{\{n\}}, b]$ ($a = 0^{\{n\}}$), and the suffix case, that is $[a, 1^{\{n\}}]$ ($b = 1^{\{n\}}$).

Definition 2.2.1 (Complement)**.** The complement of a binary symbol x ($x \in \{0, 1\}$), denoted $\overline{x}$, is $1 - x$.

We get the complement of a binary vector by flipping all of its bits, that is $\overline{x} = (\overline{x_1}, \ldots, \overline{x_n})$.

The complement of the ϕ symbol is ϕ ($\overline{\phi} = \phi$).

To obtain the complement of a ternary vector, we flip all its fixed bits.

Note that prefix and suffix cases are complementary – we may transform a suffix instance $[a, 1^{\{n\}}]$ to a prefix instance $[0^{\{n\}}, \overline{a}]$. Flipping the polarity of the resulting ternary vectors yields a solution of the initial suffix instance $[a, 1^{\{n\}}]$. This means it suffices to solve the prefix case, which is what we'll do.

Let $a = 0^{\{n\}}$ and $b < 1^{\{n\}}$. Let c be the n-bit number $b + 1$. Since $b < 1^{\{n\}}$, we don't need more than n bits to encode c.

The algorithm produces one ternary vector for each 1-bit in c. If o is a position of a 1-bit in c ($c^{[o]} = 1$), then the corresponding ternary vector is $c^{[1,o-1]}0\phi^{\{n-o\}}$. Thus we get the following spanning set:

$$\mathcal{T} = \{c^{[1,o-1]}0\phi^{\{n-o\}} | c^{[o]} = 1\}$$

Theorem 2.2.0.1. *$\mathcal{T}$ spans exactly the interval $[0^{\{n\}}, b]$ (feasibility).*

Proof. To see that every number spanned by $\mathcal{T}$ is in $[0^{\{n\}}, b]$, note that all the numbers spanned by $\mathcal{T}$ are smaller than c, since their most significant bit different from c must be a 0-bit and they must differ from c.

On the other hand, every number x smaller than c must differ from c, and the leftmost different bit must be 0 in x and 1 in c. Let o be its position. Then $c^{[1,o-1]}0\phi^{\{n-o\}} \in \mathcal{T}$ spans x. □

Theorem 2.2.0.2. *$\mathcal{T}$ is minimal in size (optimality).*

Proof. We'll construct a set V of $|\mathcal{T}|$ true points no pair of which can be spanned by a single ternary vector. Dubovský[1] calls such sets ,,orthogonal". Doing so, we'll show a lower bound $|\mathcal{T}|$ on the size of feasible solutions, proving optimality of $\mathcal{T}$.

Similarly to the spanning vectors, the orthogonal true points correspond to 1-bits of c:

$$V = \{c^{[1,o-1]}0c^{[o+1,n]} | c^{[o]} = 1\} \tag{2.1}$$

Clearly $|V| = |\mathcal{T}|$.

It's easy to see that any ternary vector that spans two different points in V must also span the false point c, so it can't be a part of the solution. Also note that all points in V are smaller than c, so they are true points.

If there was a feasible spanning set of size smaller than $|V|$, at least one of its vectors would need to span at least two points in V. As we have shown, such ternary vector would necessarily also span the false point c, leading to contradiction with the set's feasibility. □

2.3 General case

Having solved the trivial and prefix and suffix cases, we are left with the situation $0^{\{n\}} < a < b < 1^{\{n\}}$.

If a and b have the same MSB, we recursively span $[a^{[2,n]}, b^{[2,n]}]$ and prepend the MSB to the solution.

Let $a^{[1]} \neq b^{[1]}$. Since $a \leq b$, necessarily $a^{[1]} = 0$ and $b^{[1]} = 1$. This restriction leaves us with four possible combinations of pairs of MSBs:

1. $a^{[1,2]} = 01$, $b^{[1,2]} = 10$
2. $a^{[1,2]} = 00$, $b^{[1,2]} = 10$
3. $a^{[1,2]} = 01$, $b^{[1,2]} = 11$
4. $a^{[1,2]} = 00$, $b^{[1,2]} = 11$

Following Schieber et al.,[2] we'll deal with each of these cases separately. We won't prove feasibility nor optimality in this text. Please refer to the original article for the proofs.

2.3.1 $a^{[1,2]} = 01$ and $b^{[1,2]} = 10$

In this case, we'll span the two sub-intervals $[a, 01^{\{n-1\}}]$ and $[10^{\{n-1\}}, b]$ separately. Note that after leaving out the initial shared bit, they are prefix and suffix interval, respectively, so the algorithm from section 2.2 is used to span each of the sub-intervals.

2.3.2 $a^{[1,2]} = 00$ and $b^{[1,2]} = 10$

In this case, we divide the interval in three sub-intervals:

- $[a, 001^{\{n-2\}}]$
- $[010^{\{n-2\}}, 011^{\{n-2\}}]$
- $[100^{\{n-2\}}, b]$

The subinterval $[010^{\{n-2\}}, 011^{\{n-2\}}]$ is spanned by the single ternary vector $01\phi^{\{n-2\}}$.

The subintervals $[a, 001^{\{n-2\}}]$ and $[100^{\{n-2\}}, b]$ are spanned together as follows:

- Recursively solve the $(n-1)$-bit instance $[0a^{[3,n]}, 1b^{[3,n]}] = [a^{[1]}a^{[3,n]}, b^{[1]}b^{[3,n]}]$
- Insert a 0-bit in the second position of the vectors from the resulting spanning set

2.3.3 $a^{[1,2]} = 01$ and $b^{[1,2]} = 11$

This case is complementary to case 2.3.2. As in suffix case, note that flipping the bits in the endpoints transforms this case to case 2.3.2 and flipping the fixed bits in the resulting spanning set preserves correctness.

2.3.4 $a^{[1,2]} = 00$ and $b^{[1,2]} = 11$

Let j be maximal such that $a^{[1,j]} = 0^{\{j\}}$ and $b^{[1,j]} = 1^{\{j\}}$. Since $a > 0^{\{n\}}$ (and $b < 1^{\{n\}}$), necessarily $j < n$.

j gives us three sub-intervals to span:

- $[a, 0^{\{j\}}1^{\{n-j\}}]$
- $[0^{\{j-1\}}10^{\{n-j\}}, 1^{\{j-1\}}01^{\{n-j\}}]$
- $[1^{\{j\}}0^{\{n-j\}}, b]$

Note that the second sub-interval contains exactly the numbers that don't start with either j zeros or j ones, and as such can be spanned by j ternary vectors.

The other two sub-intervals are spanned recursively.

3. $2k$-approximation algorithm for minimizing DNF representation of k-interval Boolean functions

3.1 Introduction

In this chapter, an algorithm will be shown that computes a small DNF representation of a Boolean function given as a set of k intervals. The input intervals are represented by pairs of endpoints (n-bit numbers). An approximation ratio of $2k$ will be proved.

3.2 Algorithm

3.2.1 Description

Input Numbers $a_1, b_1, \ldots, a_k, b_k$ that satisfy the inequalities in definition 1.1.13

Output A set of ternary vectors

Procedure The algorithm goes through all the intervals $[a_i, b_i]$. For each i, the longest common prefix of a_i and b_i is computed. Let j be its length. Note that $a^{[j+1]} = 0$ and $b^{[j+1]} = 1$. Now let $a'' = a_i^{[j+2,n]}$ and $b'' = b_i^{[j+2,n]}$. Optimally span the suffix interval $[a'', 1^{n-j-1}]$ and the prefix interval $[0^{n-j-1}, b'']$ using the (linear time) algorithm introduced in [2]. Prepend $a_i^{[1,j+1]}$ and $b_i^{[1,j+1]}$ to the respective ternary vectors and add them to the output spanning set.

3.2.2 Correctness

Theorem 3.2.2.1. *The algorithm spans exactly* $f^n_{[a_1,b_1],\ldots,[a_k,b_k]}$.

Proof. This is easy to see from the fact that the subintervals form a partition of the multi-interval (i.e. the set of all true points) and that each of them is spanned exactly by the suffix or prefix procedure. □

3.2.3 Approximation ratio

Theorem 3.2.3.1. *Let* $\mathcal{T}_{opt}$ *be an optimal spanning set of* $f^n_{[a_1,b_1],\ldots,[a_k,b_k]}$ *and let* $\mathcal{T}_{approx}$ *be the spanning set returned by the algorithm. We claim that:*

$$|\mathcal{T}_{approx}| \leq 2k|\mathcal{T}_{opt}| \tag{3.1}$$

Proof. Let $\mathcal{T}_x$ be the largest (n-bit) spanning set of a "suffix" or "prefix" subinterval added in the algorithm. Without loss of generality, let the respective

subinterval be "prefix" $[b_i^{[1,j+1]}, b_i]$. From [1, p. 36] we know that there is an orthogonal set of $[b_i^{[1,j+1]}, b_i]$ of size $|\mathcal{T}_x|$, and moreover that its orthogonality only depends on the false point $b+1$. Note, however, that $b+1$ is also a false point in $f^n_{[a_1,b_1],\ldots,[a_k,b_k]}$. Thus we obtain an orthogonal set of size $|\mathcal{T}_x|$ for the k-interval function, limiting the size of its optimal spanning set $|\mathcal{T}_{opt}| \geq |\mathcal{T}_x|$.

Since $\mathcal{T}_x$ is the largest of the $2k$ partial sets used to span the function in the approximation algorithm, we know that $|\mathcal{T}_{approx}| \leq 2k|\mathcal{T}_x|$.

Joining the inequalities together we conclude: $|\mathcal{T}_{approx}| \leq 2k|\mathcal{T}_x| \leq 2k|\mathcal{T}_{opt}|$. □

Theorem 3.2.3.2. *The approximation ratio of* $2k$ *is tight.*

Proof. For every k that is a power of 2 we'll show a k-interval function such that $|\mathcal{T}_{approx}| = 2k|\mathcal{T}_{opt}|$ (following the notation from Theorem 3.2.3.1.

Let $k = 2^{n_k}$. Let P be all the n_k-bit numbers, that is $P = \{0,1\}^{n_k}$. Note that $|P| = k$.

For each $p \in P$, we define the interval $[a_p, b_p]$ by appending 2-bit suffixes to p:

- $a_p = p00$
- $b_p = p01$

The first appended bit (0 for both a_p and b_p) ensures that there is at least one false point between any pair of intervals defined this way. The second appended bit (0 for a_p and 1 for b_p) ensures that the interval has two points, so the approximation algorithm will use two vectors to span it. Thus we have k (n_k+2)-bit intervals, none of which intersect or touch.

Since $P = \{0,1\}^{n_k}$, we can span all the intervals by the single ternary vector $\phi^{\{n_k\}}0\phi$. Clearly $|\mathcal{T}_{opt}| = 1$.

However, the approximation algorithm uses $2k$ vectors to span the intervals, since it spans each of the intervals separately and uses two vectors for each interval.

We get $|\mathcal{T}_{approx}| = 2k|\mathcal{T}_{opt}|$.

Note that the optimal spanning set is disjoint, so the ratio is tight in disjoint case as well. □

Conclusion

Bibliography

[1] Jakub Dubovský. A construction of minimum DNF representations of 2-interval functions. Master's thesis, Charles University in Prague, 2012.

[2] Baruch Schieber, Daniel Geist, and Ayal Zaks. Computing the minimum DNF representation of boolean functions defined by intervals. *Discrete Applied Mathematics*, 149(1–3):154 – 173, 2005. Boolean and Pseudo-Boolean Functions Boolean and Pseudo-Boolean Functions.

Glossary

DNF disjunctive normal form. ii, 1, 5

MSB most significant bit. 3