Interval Extensions of Partially Defined Boolean Functions

ONDŘEJ ČEPEK*
Charles University
Dep. of Theoret. Comp. Science
Malostran. nám. 25, 118 00 Prague 1
CZECH REPUBLIC

DAVID KRONUS
Charles University
Dep. of Theoret. Comp. Science
Malostran. nám. 25, 118 00 Prague 1
CZECH REPUBLIC

PETR KUČERA
Charles University
Dep. of Theoret. Comp. Science
Malostran. nám. 25, 118 00 Prague 1
CZECH REPUBLIC

Abstract: Interval functions constitute quite a special class of Boolean functions for which it is very easy and fast to determine their functional value on a specified input vector. The value of an n-variable interval function specified by interval $[a, b]$ (where a and b are n-bit binary numbers) is $true$ if and only if the input vector viewed as an n-bit number belongs to the interval $[a, b]$. Partially defined Boolean function (pdBf) is a pair (T, F) of sets of vectors representing truepoints and falsepoints respectively. In this paper we study the problem of finding an interval extension of given pdBf, that is a Boolean function f which respects truepoints and falsepoints of the input pdBf and can be represented by an interval. We present a polynomial-time algorithm which solves this problem.

Key–Words: partially defined Boolean function, interval function

1 Introduction

The class of interval functions was introduced in [6], where the following problem was presented: Given two n-bit numbers a, b, find a minimum (shortest) DNF representing a Boolean function f on n variables, which is true exactly on numbers from the interval $[a, b]$. This problem originated from the field of automatic generation of test patterns for hardware verification, see e.g. [5, 4]. In [6] the authors also studied the problem in which they are looking for a shortest DNF $\mathcal{F}$ representing f (given by an interval [a,b]) such that for any vector x, for which $f(x) = 1$ there is exactly one term t in $\mathcal{F}$ for which $t(x) = 1$. In other words, there is an additional requirement that the sets on which the terms of $\mathcal{F}$ are satisfied must be pairwise disjoint.

The representation of an interval Boolean function using only the two n-bit numbers a, b can be rather useful (and in particular it is very short). In [7] we studied the reverse problem which also appears to be interesting and practically important: Given a DNF $\mathcal{F}$, can we recognize whether it represents an interval function? This problem is co-NP hard in general, as we have shown in [7], but can be solved in linear time with some additional requirements on the input DNF (the Boolean function represented by the input DNF belongs to some class of Boolean functions for which we are able to solve the satisfiability problem in polynomial time). The most trivial example is the class of monotone functions.

Another widely studied and practically important problem is finding an extension of a pdBf in a specified class. This problem can be defined as follows: Given partially defined Boolean function (T, F) find a function from a specified class which is true on all vectors from T and false on all vectors from F, or prove that such a function does not exist. In [1] this problem was studied for a variety of classes of Boolean functions. For some of the classes the problem is solvable in polynomial time, for others it is NP-hard. In this paper we show, that given a partially defined Boolean function, we can find its interval extension, if it exists, in polynomial time.

1.1 Outline

The paper is structured as follows. In Section 2 we introduce the necessary notation and give basic definitions. The notions of an interval function and a partially defined Boolean function is also made precise here. In Section 3 we show, that it is possible to find a positive interval extension of a pdBf, if it exists, and we also generalize this result to general interval functions later in Section 4. We close our paper with few concluding remarks in Section 5.

2 Notation and definitions

A *Boolean function*, or a *function* in short, is a mapping $f : \{0,1\}^n \mapsto \{0,1\}$, where $x \in$

*also teaching at the Institute of Finance and Administration (VŠFS), Estonská 500, Prague 10, Czech republic

$\{0,1\}^n$ is called a *Boolean vector* (a *vector* in short). 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

$$t = \bigwedge_{i \in I} x_i \wedge \bigwedge_{j \in J} \overline{x}_j \qquad (1)$$

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. [3].), that every Boolean function can be represented by a DNF. For a DNF $\mathcal{F}$ and a term t we denote by $t \in \mathcal{F}$ the fact, that t is contained in $\mathcal{F}$.

The DNF version of the *satisfiability problem* (sometimes called the *falsifiability problem*) is defined as follows: given a DNF $\mathcal{F}$, does there exist an assignment of truth values to the variables which makes $\mathcal{F}$ evaluate to 0?

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 a term t an *implicant* of a 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 $\mathcal{F}$.

A term is called *positive* if it consists only of positive literals, it is called *negative* if it consists only of negative literals. A DNF $\mathcal{F}$ is called *positive* (*negative* resp.) if it consists only of positive (negative resp.) terms. A function f is called *positive* (*negative* resp.) if it admits a positive (negative resp.) representation. The class of positive functions will be denoted by $\mathcal{C}^+$, the class of negative functions will be denoted by $\mathcal{C}^-$.

We will denote binary vectors by $\vec{x}, \vec{y}, \ldots$. The bits of vector $\vec{x} \in \{0,1\}^n$ will be denoted by $x_1, \ldots, x_n$. The vector $\vec{x}$ also corresponds to an integer number x with binary representation equal to $\vec{x}$. In this case x_1 is the most significant bit of x and x_n the least significant bit. Hence $x = \sum_{i=1}^{n} x_i 2^{n-i}$.

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

Definition 2. *Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called an* 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 $\vec{x} \in \{0,1\}^n$ we get $f(\vec{x}) = 1$ if and only if $x^\pi \in [a, b]$. The class of interval functions will be denoted by $\mathcal{C}_{int}$. The class of positive (negative resp.) interval functions will be denoted by C_{int}^+ (C_{int}^- resp.).*

In this paper we shall need the fact that interval functions are closed under partial assignment.

Lemma 3. *The class of (positive) interval functions is closed under partial assignment.*

Proof : Let f be an interval function with respect to the order of variables $x_1, \ldots, x_n$, representing the interval $[a, b]$. Let $f' = f[x_i := v]$, where $1 \leq i \leq n$ and $v \in \{0,1\}$ are taken arbitrarily. Let us assume by a contradiction that f' is not interval with respect to the order of variables $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. Then there are three $(n-1)$-bit numbers $u'_1 < u'_2 < u'_3$, such that $f'[u'_1] = f'[u'_3] = 1$ while $f'[u'_2] = 0$. But that would mean that there would also be n-bit numbers $u_1 < u_2 < u_3$ originating from u'_1, u'_2, u'_3 by inserting the bit of value v at the i-th position, for which $f[u_1] = f[u_3] = 1$ while $f[u_2] = 0$, which is a contradiction to the fact that f is an interval function. Since the class of positive functions is closed under partial assignment, the class of positive interval functions is closed under partial assignment as well. □

Definition 4. *Let T and F be two sets of n-bit Boolean vectors such that $T \cap F = \emptyset$. Then we call the pair (T, F) a* partially defined Boolean function (pdBf). *A Boolean function f defined on n variables* extends *a pdBf (T, F) if it is true on all vectors from T and false on all vectors from F.*

3 Positive Interval Extensions of pdBfs

Given a class $\mathcal{C}$ of Boolean functions, an *extension problem* for $\mathcal{C}$ can be stated as follows: given an arbitrary pdBf (T, F), does there exist a function $f \in \mathcal{C}$ such that f extends (T, F)? The following algorithm solves the extension problem for the class of positive interval functions.

Algorithm 5. EXTENSION($\mathcal{C}^+_{int}$)

Input: *A pdBf* (T, F)
Output: *Order* $x_{\pi(1)}, \ldots, x_{\pi(n)}$ *of variables, and* n*-bit number* a*, if there is a positive interval function which extends* (T, F) *and represents an interval* $[a, 2^n)$ *with respect to the order* $x_{\pi(1)}, \ldots, x_{\pi(n)}$. **NO** *otherwise.*

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

Algorithm 5 is based on similar ideas as Algorithm 4.2 in [7]. 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 5 looks at every step for an index j, which would satisfy conditions in steps 4 or 10. If the condition in step 4 is satisfied, it means that each truepoint has the j-th bit equal to 1. In this case we assume, that a positive interval extension f which is being constructed satisfies that if $f(x) = 1$ for some vector x, then $x_j = 1$, and hence conversely if $x_j = 0$ then $f(x) = 0$. Due to this assumption we can discard all the falsepoints v for which $v_j = 0$, which we do in step 8. The similar situation is, when each falsepoint v has $v_j = 0$, in this case we assume, that the extension has the similar property, in particular, that if $f(x) = 0$ for some vector x, then $x_j = 0$, and hence if $x_j = 1$, then $f(x) = 1$. Due to this assumption we can discard all the truepoints u for which $u_j = 1$ (step 14), since these are not interesting any more.

The proof of the correctness of Algorithm 5 rests on lemmas 6, 7, and 8. The first one shows that the actions which follow a successful test in step 4 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.

Lemma 6. *Let* (T, F) *be a pdBf and let us assume that* $u_1 = 1$ *holds for all* $u \in T$*. Let* T' *consist of all vectors from* T *with the first bit omitted, let* F' *consist of all vectors* 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 order of variables* $x_2, \ldots, x_n$*, then function* $f(x) = x_1 \wedge f'(x')$ *(where* x' *is the vector* x *with the first bit omitted) is a positive interval extension of* (T, F) *with respect to the order of variables* $x_1, x_2, \ldots, x_n$*.*

Proof :(*only if* part) Let us at first assume, that (T, F) has a positive interval extension f, the function $f' = f[x_1 := 1]$ clearly extends (T', F'), moreover f' is an interval function according to Lemma 3.

(*if* part) Now, let us suppose, that (T', F') has an extension f' which is a positive interval function with respect to some order $x_2, \ldots, x_n$. We shall show, that $f(x) = x_1 \wedge f'(x')$ is a positive interval extension of (T, F). To see, that f is a positive interval function it suffices to take a positive interval DNF $\mathcal{F}'$ representing f', now DNF $\mathcal{F} = x_1 \wedge \mathcal{F}'(x')$ is a DNF representing f, such that x_1 is a variable which appears in every term of $\mathcal{F}'$. $\mathcal{F}$ is a positive interval DNF because Algorithm 4.2 in [7] for positive interval function recognition would recognize it so, as can be easily observed. Hence it remains to show, that f is an extension of (T, F), which is quite clear. Given a vector $u \in F$, either $u_1 = 0$, in which case $f(u) = 0$, or $u_1 = 1$, in which case $u' \in F'$, $f(u) = f'(u')$, and $f'(u')$ extends (T', F'), hence also in this case $f(u) = 0$. Given $v \in T$, surely $v_1 = 1$ holds, and thus $f(v) = f'(v')$. Since f' extends (T', F') and $v' \in T'$, we have $f(v) = f'(v') = 1$, which completes the proof. □

Lemma 7. *Let* (T, F) *be a pdBf and let us assume that* $v_1 = 0$ *holds for all* $v \in F$*. 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 order of variables* $x_2, \ldots, x_n$*, then function* $f(x) = x_1 \vee f'(x')$ *(where* x' *is the vector* x *with the first bit omitted) is a positive interval extension of* (T, F) *with respect to the order of variables* $x_1, x_2, \ldots, x_n$*.*

Proof : Since the statement and also its proof are "mirror images" of Lemma 6 and of its proof, we leave the

details to the reader. □

Lemma 8. *Let (T, F) be a pdBf and let us assume that there exists no index j such that $u_j = 1$ holds for all $u \in T$ or $v_j = 0$ holds for all $v \in F$. 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 (numbers) x and y starting with value zero in bit j (i.e. $x_j = y_j = 0$) such that $x \in T$ and $y \in F$, and similarly there exist two vectors (numbers) u and v starting with value one in bit j (i.e. $u_j = v_j = 1$) such that $u \in T$ and $v \in F$. However, these four vectors prevent the function from being a positive interval function with respect to any order of variables in which bit j is the most significant one. □

Theorem 9. *Algorithm 5 works correctly and can be implemented so that it requires $O(n \cdot (n + |T| + |F|))$ time.*

Proof : The correctness of the algorithm follows using a simple induction on i. Lemma 6, Lemma 7, and Lemma 8 show, that each step is correct. Note, that each time the algorithm chooses an index j as the next element in the order, variable x_j plays the role of x_1 in lemmas 6 and 7. 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 $O(n \cdot (n + |T| + |F|))$. 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 4.2 in [7].

- Array A of length n, each element $A[i]$ will contain a pointer to a double-linked list of structures representing true points $u \in T$ for which $u_i = 1$. 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 false points $v \in F$ for which $v_i = 0$. 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 $u \in T \cup F$ will be represented by a structure S_u. This structure will contain an array P_u of length n. If $u \in T$ and $u_i = 1$, then $P_u[i]$ points to the element of the list representing u in $A[i]$. If $u \in F$ and $u_i = 0$, then $P_u[i]$ points to the element of the list representing u in $B[i]$. In another case $P_u[i]$ contains a nil value. Moreover each element of list in $A[i]$, $B[i]$ representing a vector 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 4 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 for 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 14. 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. □

4 Interval Extensions of pdBfs

The following algorithm solves the extension problem for the class of interval functions. 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 $S|_{j=e} = \{u \in S \mid u_j = e\}$, where $e \in \{0, 1\}$, and by $S^I = \{u \text{ restricted to bits from } I \mid u \in S\}$.

Algorithm 10. EXTENSION($\mathcal{C}_{int}$)

Input: *A pdBf (T, F)*

Output: *Order $x_{\pi(1)}, \ldots, x_{\pi(n)}$ of variables, and n-bit numbers a, b, if there is an interval function which extends (T, F) and represents an interval $[a, b]$ with respect to the order $x_{\pi(1)}, \ldots, x_{\pi(n)}$.* **NO** *otherwise.*

1: $i := 1$
2: $I := \{1, \ldots, n\}$
3: **while** $(\exists j \in I)\ [(\forall u \in T)\ (u_j = 1) \vee (\forall u \in T)\ (u_j = 0)]$
4: **do**
5: **if** $(\forall u \in T)\ (u_j = 1)$
6: **then**
7: $a_i := 1$
8: $b_i := 1$
9: $F := F \setminus \{v \in F \mid v_j = 0\}$
10: **else**
11: $a_i := 0$
12: $b_i := 0$
13: $F := F \setminus \{v \in F \mid v_j = 1\}$
14: **endif**
15: $\pi(i) := j$
16: $I := I \setminus \{j\}$
17: $i := i + 1$
18: **done**
19: **for** $j \in I$
20: **do**
21: $T_0 := (T|_{j=0})^{I \setminus \{j\}}$
22: $F_0 := (F|_{j=0})^{I \setminus \{j\}}$
23: $T_1 := (T|_{j=1})^{I \setminus \{j\}}$
24: $F_1 := (F|_{j=1})^{I \setminus \{j\}}$
25: **if** (T_0, F_0) has a positive interval extension f_0 representing interval $[a', 2^{n-i})$ **and** (T_1, F_1) has a negative interval extension f_1 representing interval $[0, b']$ **and** both extensions are interval with respect to the same order $x_{i+1}, \ldots, x_n$.
26: **then**
27: $\pi(i) := j$
28: $a_i := 0$
29: $b_i := 1$
30: $a := a_1 \ldots a_i a'$
31: $b := b_1 \ldots b_i b'$
32: **return** order $x_{\pi(1)}, \ldots, x_{\pi(n)}$ and the numbers a, b.
33: **endif**
34: **done**
35: **return NO**

Now we will prove the correctness of Algorithm 10 again by using series of lemmas.

Lemma 11. *Let (T, F) be a pdBf on n variables and let us assume that $u_1 = 1$ ($u_1 = 0$ resp.) holds for all $u \in T$. Let T' consist of all vectors from T with the first bit omitted, let F' consist of all vectors 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 order of variables $x_2, \ldots, x_n$, then function $f(x) = x_1 \wedge f'(x_2, \ldots, x_n)$ ($f(x) = \overline{x}_1 \wedge f'(x_2, \ldots, x_n)$ resp.) is a interval extension of (T, F) with respect to the order of variables $x_1, x_2, \ldots, x_n$.*

Proof : We shall only show the case when all vectors 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

(*only if part*) Let us at first assume, that (T, F) has an interval extension f, the function $f' = f[x_1 := 1]$ clearly extends (T', F'), moreover f' is an interval function according to Lemma 3.

(*if part*) Now, let us suppose, that (T', F') has an extension f' which is an interval function with respect to some order $x_2, \ldots, x_n$. We shall show, that $f(x) = x_1 \wedge f'(x')$ is an interval extension of (T, F). Let us consider a prime and irredundant interval DNF $\mathcal{F}'$ representing f'. Then DNF $\mathcal{F} = x_1 \wedge \mathcal{F}'(x')$ is a prime and irredundant DNF representing f, and moreover it is an interval DNF, since (as can be easily observed), Algorithm 5.3 in [7] would recognize it so. Hence f is an interval function. Similarly as in the proof of Lemma 6 it can be observed that f extends (T, F), as well. □

Lemma 12. *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 $u, v \in T$ such that $u_i = 1$ and $v_i = 0$. Let T_0, F_0, T_1, F_1 be defined as in steps 21-24 of Algorithm 10. Then (T, F) has an interval extension if and only if there exists an index j (w.l.o.g. let $j = 1$ after renumbering) such that the following conditions hold:*

1. *(T_0, F_0) has a positive interval extension f_0, and*
2. *(T_1, F_1) has a negative interval extension f_1, and*
3. *f_0 and f_1 are both interval with respect to the same order of variables $x_2, \ldots, x_n$.*

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 $f(x) = [x_1 \wedge f_1(x_2, \ldots, x_n)] \vee [\overline{x}_1 \wedge f_0(x_2, \ldots, x_n)]$ is an interval extension of (T, F) with respect to order $x_1, x_2, \ldots, x_n$.

Proof :(*only if part*) Let us at first assume that (T, F) has an interval extension f with respect to some order $x_1, \ldots, x_n$. Set $j = 1$, $f_0 = f[x_1 := 0]$, and $f_1 = f[x_1 := 1]$. By Lemma 3 both f_0 and f_1 are interval functions. It should be clear, that both of them are interval with respect to order $x_2, \ldots, x_n$. Moreover, it is obvious that $f(x) = [x_1 \wedge f_1(x_2, \ldots, x_n)] \vee [\overline{x}_1 \wedge f_0(x_2, \ldots, x_n)]$.

(*if part*) Now, let us assume, that there is some j, for which the conditions 1-3 apply. We shall show, that $f(x) = [x_1 \wedge f_1(x_2, \ldots, x_n)] \vee [\overline{x}_1 \wedge f_0(x_2, \ldots, x_n)]$ is an interval extension of (T, F). Again to see, that f is an interval function, we shall recall Algorithm 5.3 from [7]. Given prime and irredundant DNFs $\mathcal{F}_0$ representing f_0 and $\mathcal{F}_1$ representing f_1, $\mathcal{F} = (x_1 \wedge \mathcal{F}_1) \vee (\overline{x}_1 \wedge \mathcal{F}_0)$ would be recognized by Algorithm 5.3 from [7] as an interval DNF. Since Algorithm 5.3 from [7] correctly recognizes interval functions, f is an interval function. It remains to show, that f extends (T, F), which is quite easy. Given $u \in T$, either $u_1 = 1$, in which case u restricted to bits $2, \ldots, n$ belongs to T_1 and hence $f_1(u_2, \ldots, u_n) = 1$ and also $f(u) = 1$. If $u_1 = 0$, similarly $f_0(u_2, \ldots, u_n) = 1$ and $f(u)$ is again 1. The case when $u \in F$ is similar and so we leave the details to the reader. □

Theorem 13. *Algorithm 10 is correct and works in time* $O(n \cdot (n + |T| + |F|))$.

Proof : The correctnes of Algorithm 10 is a corollary of Lemma 11, Lemma 12, and simple induction on i.

Time requirements are straightforward, the *while* cycle in (steps 3 – 18) repeats at most n times, all the steps can be performed in $O(n \cdot (n + |T| + |F|))$ time. The same holds for the *for* cycle in (steps 19 – 34), where we use Theorem 9 to show, that condition in the *if* statement (step 25) can be tested in $O(n \cdot (n + |T| + |F|))$ time. □

5 Conclusions

In Section 4 we have shown, that given a pdBf (T, F), we can find an (positive) interval extension of (T, F) in polynomial time, if it exists.

However many problems concerning interval functions and pdBfs still remain open. The following list contains some of them:

1. BEST-FIT interval extensions of pdBfs.
   This problem is a generalization of the interval extension problem of pdBf we studied in Section 4. We are looking for interval function f which for given pdBf (T, F) minimizes the number of vectors that are not correctly classified, that means that we want to minimize $|(T(f) \cap F) \cup (F(f) \cap T)|$.

2. Interval extensions of pBmds.
   This is another generalization of interval extension problem of pdBf. Here we want to decide whether a given pBmd (partially defined Boolean function with missing data) has interval extension. In pBmd (T, F) some vectors in $T \cup F$ can have in some positions '*' instead of 0 or 1, and this means that the value of this bit is not determined. There are several variants of this problem, see [2] for more information.

Acknowledgements: The work on this research was supported by the Czech Science Foundation (GACR), grants No. 201/04/1102 (first author) and No. 201/05/H014 (second author).

References:

[1] E.BOROS, T.IBARAKI AND K.MAKINO Error-free and best-fit extensions of partially defined boolean functions. *Information and Computation 140*, 1998, 254 – 283.

[2] E.BOROS, T.IBARAKI AND K.MAKINO Extensions of partially defined boolean functions with missing data. Tech. Rep. 6-96, RUTCOR Research Report RRR, Rutgers University, New Brunswick, NJ, 1996.

[3] M.GENESERETH AND N.NILSSON *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, Los Altos, CA, 1987.

[4] C.-Y.HUANG AND K.-T.CHENG Solving constraint satisfiability problem for automatic generation of design verification vectors. *Proceedings of the IEEE International High Level Design Validation and Test Workshop*, 1999.

[5] D.LEWIN, L.FOURNIER, M.LEVINGER, E.ROYTMAN, AND G.SHUREK Constraint satisfaction for test program generation, 1995.

[6] B.SCHIEBER, D.GEIST AND Z.AYAL Computing the minimum dnf representation of boolean functions defined by intervals. *Discrete Applied Mathematics 149*, 2005, 154 – 173.

[7] O.ČEPEK, D.KRONUS AND P.KUČERA Renamable interval boolean functions. *Proceedings of Czech-Japan Seminar 2006, Kitakyushu, Japan*, 2006, 232 – 241.