RECOGNIZING DECOMPOSITION OF A PARTIAL BOOLEAN FUNCTION

Arkadij Zakrevskij

Abstract: A hard combinatorial problem is investigated which has useful application in design of discrete devices: the two-block decomposition of a partial Boolean function. The key task is regarded: finding such a weak partition on the set of arguments, at which the considered function can be decomposed. Solving that task is essentially speeded up by the way of preliminary discovering traces of the sought-for partition. Efficient combinatorial operations are used by that, based on parallel execution of operations above adjacent units in the Boolean space

Keywords: Partial Boolean function, non-disjunctive decomposition, weak partition, search by traces, recognition of solution.

Setting the problem

One of the major problems of the theory of Boolean functions is the problem of functional decomposition, which has useful application in design of logic circuits. Enough to say, that its positive solution can significantly simplify the logic circuits implementation of regarded Boolean functions (for example, at logical synthesis in the basis of LUTs, look up tables). It was set originally in papers [Povarov, 1954], [Ashenhurst, 1959] and [Curtis,1962] as the following problem of *disjoint two-block sequential decomposition*. Suppose a Boolean function $f(x) = f(x_1, x_2, \dots, x_n)$ is given, and it is required to represent it as a composition $f(x) = g(h(u), v)$ of two functions g and h of smaller number of variables constituting subsets u and v. By that $x = u \cup v$ and $u \cap v = \varnothing$.

A necessary and sufficient condition of existing of such a composition was found. Let $F(u \times v)$ be the Boolean matrix presenting all values of function f for different values of vector x in such a way, that its rows correspond to values of vector u, and its column correspond to values of vector v. The condition is formulated as follows: the rows of that matrix can have not more than two different values.

The more general, non-disjoint decomposition was investigated afterwards, at which the given function $f(x)$ should be represented as a composition

$$f(x) = g(h(u, w), w, v),$$

where three subsets of arguments are connected with the relations $x = u \cup w \cup v$, $u \cap w = u \cap v = w \cap v = \varnothing$, and the couple of sets u and v is regarded as a *weak partition* on set x and is designated u/v. Both types of decomposition are illustrated by Fig. 1.

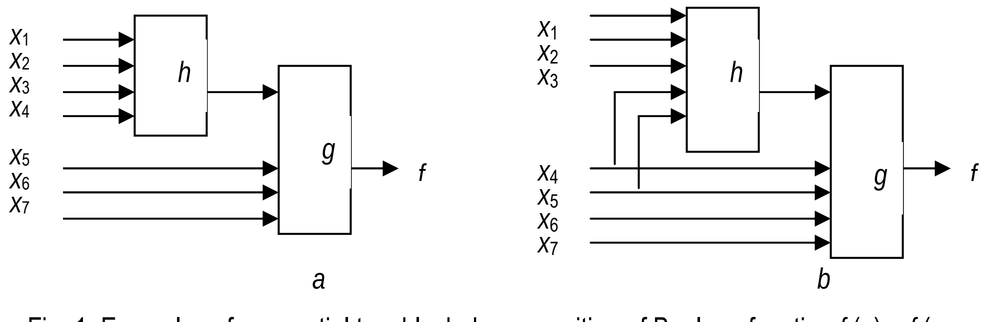

Fig 1. Examples of sequential two-block decomposition of Boolean function $f(x) = f(x_1, x_2, \dots, x_7)$: a) disjoint, b) non-disjoint ($u = (x_1, x_2, x_3)$, $w = (x_4, x_5)$ and $v = (x_6, x_7)$).

To solve the formulated task it is necessary, first of all, to find such a weak partition u/v, at which the variables of set u enter in number of arguments of function h only, and variables of v – only in number of arguments of function g. The conditions $|u| > 1$ and $|v| > 0$ should be fulfilled also, otherwise the composition will appear trivial (exists always). Let's name this partition *appropriate*, and the function $f(x)$ – *separable*, or *decomposable* at this partition.

The finding of appropriate partition is a difficult task, for which solution an effective combinatorial algorithm was offered [Zakrevskij, 2007a], in case of a completely specified Boolean function. This task becomes even more complicated, when the function f (x) appears to be partial, being defined not on all sets of values of variables from set x. Just this case is considered below.

Recognizing solution

It] was shown [Zakrevskij, 2007a], that the probability of decomposability of a random completely defined Boolean function fast tends to zero with growth of number of variables n, so already at $n > 9$ such a function, most likely, is not decomposable. In case of partial functions this probability arises with growth of uncertainty, however even in this case it remains small enough, as will be shown below.

Taking into account the given remark, let's assume, that it is known beforehand, that the considered function f (x) is separable, being obtained as a result of composition g (h (u, w), w, v) of some two Boolean functions g and h on a weak partition u/v on the set of arguments x. It is required to detect (to recognize) this partition, after which the obtaining of functions g and h is not a difficult task.

A method of checking a partial Boolean function for decomposability at some given weak partition was offered in [Zakrevskij, 2007b]. An arbitrary Boolean function $f(x) = f(x_1, x_2, \ldots, x_n)$ was represented there by a "long" Boolean vector $\boldsymbol{f} = (f_0, f_1, \ldots, f_{2^n-1})$, which 2^n components present the values of the function corresponding to values of vector x enumerated in conventional order. For example, vector $\boldsymbol{f}$ = 10011011 represents the function $f(x_1, x_2, x_3)$ taking value 1 on values 000, 011, 100, 110, 111 of vector x and value 0 on values 001, 010, 101. The more convenient for visual perception matrix form of vector $\boldsymbol{f}$ is used below in examples: vector $\boldsymbol{f}$ is divided into parts corresponding to different sets of values of several left components of vector x, and these parts play the role of matrix rows.

For example, presenting a Boolean function $f(x_1, x_2, x_3, x_4, x_5, x_6)$ vector

f = **0110110101011110001001000001011011001010011100010100010111010011**

accepts the following matrix shape, where by thick lines are marked columns and rows where corresponding arguments of the function take value 1:

```
         --------     3
     ----    ----    4
   --  --  --  --   5
  - - - - - - - -  6
 0110110101011110
| 0010010000010110
|  1100101001110001
|| 0100010111010011
1 2
```

In the case, when the appropriate partition is not known a priori, it is possible to organize its search, sorting out different weak partitions and checking the function on decomposability at them. However, such a way is rather labor-consuming, as the number of different weak partitions on the set of variables is approximated from above by value 3^n, fast growing with increase of number of variables n.

In the present paper the method of search for appropriate partition u/v by its traces is suggested, which sufficiently cuts down the number of analyzed partitions. Originally, it was designed for completely specified Boolean functions [Zakrevskij, 2007a], but here it is extended on the case of partial Boolean functions.

Search by traces

The method of decomposition suggested below is based on the following reasons which key moments are given in the form of assertions. They were formulated before for the case of completely specified Boolean functions [Zakrevskij, 2007a], but remain valid when partial Boolean functions are considered.

Suppose two partitions u/v and u^*/v^* are given, such that $u^* \subseteq u$ and $v^* \subseteq v$. Let's speak, that partition u^*/v^* *submits* to partition u/v, and call it a *trace* of u/v, Assertion 1. If a partial Boolean function $f(x)$ is decomposable at partition u/v, it is decomposable as well at partition u^*/v^*.

Corollary. If the function $f(x)$ is not decomposable at partition u^*/v^*, it is not decomposable also at partition u/v.

Let's assume $|u| = k$ and $|v| = m$. Partition with $k = 2$ and $m = 1$ we shall term as a *triad*. It is the simplest of partitions, at which some nontrivial decomposition can take place.

Assertion 2. A partial Boolean function is decomposable, if and only if it is decomposable if only at one of triads.

Therefore the search for the partition u/v may be started with the search of its traces on the set of triads, i.e. with looking for an appropriate triad. The needed checking of triads can be fulfilled fast enough, as their number is not large, being significantly less than the number of all weak partitions.

Assertion 3. The number of triads is equal to $C_n^2\,(n-2) = \dfrac{n(n-1)(n-2)}{2}$.

Suppose that some appropriate triad $(x_p, x_q)/x_r$. is detected. If it submits to the required partition u/v, we can find the latter, having put for the beginning $u = (x_p, x_q)$ and $v = (x_r)$, and then sequentially expanding these two sets, sorting out remaining variables and testing them on possibility of inclusion into set u or v.

By reviewing some concrete triad u/v the Boolean space $M = \{0, 1\}^n$, where the partial Boolean function $f(x)$ is presented, is divided into 2^{n-3} intervals corresponding to different values of vector $w = x / (u \cup v)$. On each of them the corresponding coefficient f_i of disjunctive decomposition of the function by variables of set w is given. It represents some partial Boolean function of variables x_p, x_q, x_r. As a matter of convenience of subsequent reasoning we shall present each of these coefficients by a ternary matrix size 4×2, which rows correspond to values of the two-component vector u, and columns – to values of the one-component vector v. Let's designate this matrix T_i and name it a *fragment*. Thus, the 2^n-element ternary matrix representing function $f(x)$, is decomposed into 2^{n-3} eight-element fragments specifying functions $f_i\,(x_p, x_q, x_r)$.

A concrete example of such splitting into eight fragments for a partial Boolean function $f(x_1, x_2, x_3, x_4, x_5, x_6)$ and triad $(x_1, x_2)/\,x_6$ is shown below.

```
                  -----------    3
            -----       -----   4
         --    --    --    --  5
       -  -  -  -  -  -  -  - 6
    10 0- 1- 11 0- 10 -1 10
 |  -1 00 -1 1- 00 -1 1- -0
|   0- 1- -0 10 10 0- 11 -1
||  1- 01 11 -1 00 10 -0 11
1 2
```

Assertion 4. The function $f(x)$ can be decomposed at triad $(x_p, x_q)/x_r$, if and only if each of the coefficients $f_i\,(x_p, x_q, x_r)$ is decomposable also at the same triad.

It follows from here, that the probability of decomposability of function $f(x)$ at a concrete triad is equal to $\gamma^{\,k}$, where k is the number of coefficients equal 2^{n-3} and γ – the probability of decomposability of one coefficient. In the case of a completely specified Boolean function the last probability is approximated by the value 1/3, and with growth of uncertainty decreases. Nevertheless, the probability of decomposability of the function $f(x)$ quickly decreases with growth of the number of its arguments.

Checking triads for fitness

So, a triad is appropriate, if each fragment of the corresponding splitting of the ternary matrix is suitable. That means, the partial function $f(x)$ can be completely defined in such a way, that each fragment will contain no more than two types of Boolean rows (each having equal rows). In other words, a fragment is suitable, if it contains no more than two classes of compatible rows. Remind that two ternary rows are compatible, i.e. they could become equal by changing values "–"of some components for 1 or 0, if they are not orthogonal. It follows from here, that the fragment is suitable, if the graph of orthogonality of its rows is bichromatic [Harary, 1969].

Let us offer the following way of checking fragments with the purpose of detection of suitable ones among them. Any fragment contains four rows, therefore the graph of orthogonality has four vertices. It is bichromatic, if it has no cycle of length three. Let's select arbitrary two different vertices. If such a cycle exists, then one of the selected vertices will belong to it. Therefore, it is enough to test each of these two vertices on belonging to a cycle of length three. If such belonging will not be revealed, graph is bichromatic, and the triad is suitable.

Necessary and sufficient condition of entering a vertex, i.e. corresponding row, in a cycle of length three could be formulated as follows: among rows orthogonal to the given one, there exist mutually orthogonal rows.

For example, the left of the shown below fragments appears to be suitable, and the right – no, as there is a cycle of length three, composed by three last rows: each of them is orthogonal to the other two (look at Fig. 2).

```
10    -1   1
-1    00   2
0-    1-   3
01    01   4
```

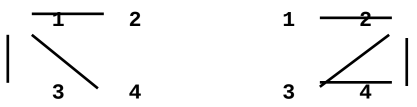

Fig. 2. Graphs of orthogonality of rows of fragments

It is not difficult to check any fragment separately, but the problem is how to check all $2^{|w|}$ fragments for the examined triad and how to find an appropriate partition fast enough. The next part of the paper is devoted to that problem.

Basic operations in Boolean space

A compact and effective set of basic combinatorial operations, which can greatly facilitate the program implementation of the regarded method of search for appropriate partitions, is described below. The parallelism of efficient operations over long Boolean vectors is laid into its foundation, and that essentially accelerates the fulfilled calculations.

First, let us include into our set the two-place Boolean operations $f \vee g$, $f \wedge g$, $f \oplus g$, $f \sim g$, $f \rightarrow g$ which are easily implemented as parallel component-wise operations over corresponding Boolean vectors. They are designated $\mathbf{f} \vee \mathbf{g}$, $\mathbf{f} \wedge \mathbf{g}$ (or, simpler, $\mathbf{f}\,\mathbf{g}$), $\mathbf{f} \oplus \mathbf{g}$, $\mathbf{f} \sim \mathbf{g}$, $\mathbf{f} \rightarrow \mathbf{g}$.

Second, we shall supplement them by some useful operations of interaction between adjacent (neighboring) components within the framework of one Boolean vector [Zakrevskij, 2007c].

Let's remind, that the function $f(x)$ can be represented as Shannon disjunctive decomposition by an arbitrary variable x_i – $f(x) = \overline{x_i}\, f_{i\,0} \vee x_i\, f_{i\,1}$, which coefficients $f_{i\,0}$ and $f_{i\,1}$ are Boolean functions obtained as a result of substitution of values 0 or 1 for variable x_i

Using vector representation of the function, we shall designate these operations of substitution accordingly through **f** - *i* and **f** + *i*. They are easily implemented in the Boolean space on couples of elements adjacent by the variable x_i. When executing the operation **f** - *i* both elements of the couple gain the value of the element defined by the condition $x_i = 0$, at execution of the operation **f** + *i* – gain the value of the other element corresponding to value 1 of variable x_i.

Let's show examples of such operations, and also of their compositions:

```
          f                  f-5              f-5-2
           --------           --------           --------    3
       ----    ----       ----    ----       ----    ----   4
     --  --  --  --     --  --  --  --     --  --  --  --  5
    - - - - - - - -    - - - - - - - -    - - - - - - - - 6
   0110110101011110   0101111101011111   0101111101011111
 | 0010010000010110   0000010100000101   0101111101011111
|  1100101001110001   1111101001010000   1111101001010000
|| 0100010111010011   0101010111110000   1111101001010000

   1101110111101110   1010101000010001   0000000000110011
 | 0100010001100110   0101010100110011   0000000000110011
|  1010101000010001   1010101000010001   1100110011111111
|| 0101010100110011   0101010100110011   1100110011111111
1 2        f+4               f+4+1              f+6-4+2
```

By interaction of adjacent units there are implemented also the operation $\mathrm{Inv}_i\, \boldsymbol{f}$ of inverting the function f at the variable x_i (adjacent elements interchange their values), and so-called operations of symmetrization $\mathrm{S}_i{}^{*} f$, in which both elements get value defined by the two-place operation $* \in \{ \vee , \wedge , \oplus , \sim g, \rightarrow \}$ above their initial values [Zakrevskij, 1963]. As a result of these operations the function $f(x) = \overline{x_i}\, f_{i0} \vee x_i\, f_{i1}$ is transformed correspondingly into functions $\overline{x_i}\, f_{i1} \vee x_i\, f_{i0}$, $\overline{x_i}\, (f_{i0} * f_{i1}) \vee x_i\, (f_{i0} * f_{i1})$.
Examples of these operations are shown below.

```
           f                Inv_3 f              S_3^∨ f

            --------           --------           --------    3
        ----    ----       ----    ----       ----    ----   4
      --  --  --  --     --  --  --  --     --  --  --  --  5
     - - - - - - - -    - - - - - - - -    - - - - - - - - 6
    0110110101011110   0101111001101101   0111111101111111
  | 0010010000010110   0001011000100100   0011011000110110
 |  1100101001110001   0111000111001010   1111101111111011
 || 0100010111010011   1101001101000101   1101011111010111

    0000110000001100   0011001100110011   0111111101111111
  | 0000000000000000   0011001000110010   0111111101111111
 |  1100000000110000   1011101110111011   1111111111111111
 || 0000000011000011   1001011010010110   1111111111111111
 1 2
         S_6^∧ f            S_3^⊕ f            S_3,2^∨ f
```

Here $\mathrm{S}_{3,2}{}^{\vee} f$ means composition $\mathrm{S}_3{}^{\vee} (\mathrm{S}_2{}^{\vee} f)$.

Algorithm of checking triads

The suggested way of checking triads is implemented by the following algorithm, which is remarkable by that it checks on fitness simultaneously all 2^{n-3} fragments generated by the given triad, and finds out by that if the triad is appropriate.

The regarded function $f(x)$ is represented by a couple of Boolean vectors $\boldsymbol{f}^0$ and $\boldsymbol{f}^1$, in first of which by 1s are marked the values 0 of the function and in the second - values 1. The splitting of space into fragments is fulfilled by the triad $(x_p, x_q) / x_r$.

To begin with, first rows of fragments are selected, which form together the initial coefficient f^- of decomposition of the function $f(x)$ by variables x_p and x_q (that coefficient corresponds to values $x_p = 0, x_q = 0$). The rows orthogonal to this row, are marked by value 1 in the corresponding parts of computed vector $\boldsymbol{g}$, and their values are fixed by the couple of vectors h^0 and h^1, checked up further for orthogonality. Alike vectors $\boldsymbol{f}^0$ и $\boldsymbol{f}^1$, they are Boolean vectors with 2^n components.

$h^0 := (\boldsymbol{f}^0 - p) - q$	Getting the initial coefficient f^-
$h^1 := (\boldsymbol{f}^1 - p) - q$	
$g := \mathrm{S}_r{}^{\vee} (h^0 \boldsymbol{f}^1 \vee h^1 \boldsymbol{f}^0)$	Finding coefficients orthogonal to f^-
$h^0 := \mathrm{S}_u{}^{\vee} (\boldsymbol{f}^0 g)$	Computing their intersection
$h^1 := \mathrm{S}_u{}^{\vee} (\boldsymbol{f}^1 g)$	

If it turns out that $h^0 h^1 \neq 0$, the triad is accepted as not appropriate. In case if $h^0 h^1 = 0$ the final rows of fragments are checked, which constitute the final coefficient f^+ (corresponding to values $x_p = 1, x_q = 1$).

$h^0 := (\boldsymbol{f}^0 + p) + q$	Getting the final coefficient f^+
$h^1 := (\boldsymbol{f}^1 + p) + q$	
$g := \mathrm{S}_r{}^{\vee} (h^0 \boldsymbol{f}^1 \vee h^1 \boldsymbol{f}^0)$	Finding coefficients orthogonal to f^+
$h^0 := \mathrm{S}_u{}^{\vee} (\boldsymbol{f}^0 g)$	Computing their intersection
$h^1 := \mathrm{S}_u{}^{\vee} (\boldsymbol{f}^1 g)$	

If $h^0 h^1 \neq 0$, then the triad is not appropriate. On the other hand, if $h^0 h^1 = 0$, the triad is accepted as appropriate.

Example. Let’s return to regarding the partial Boolean function $f(x_1, x_2, x_3, x_4, x_5, x_6)$, representing it by a couple of Boolean vectors (rolled up into matrices) f^0 and f^1:

```
                           -----------    3
                     -----       -----   4
                  --    --    --    --  5
                -  -  -  -  -  -  -  - 6
           10 0- 0- 11 0- 10 -1 10
        |  -1 10 -1 1- 00 -1 0- -0
       |   0- -1 -0 00 10 0- 11 -1
       ||  1- -0 11 -1 -0 10 -0 11
      1 2
         01 10 10 00 10 01 00 01   f^0
         00 01 00 00 10 00 10 01
         10 00 01 11 00 10 00 00
         00 01 00 00 00 01 01 00

         10 00 00 11 00 10 01 00   f^1
         01 10 01 10 00 01 00 00
         00 01 00 00 10 00 11 01
         10 00 11 01 00 10 00 01
```

The check of the function for decomposability at triad $(x_1, x_2) / x_6$ is reduced to testing in parallel all of fragments for fitness. In the given example all of eight fragments are suitable, therefore the function can be decomposed at that triad.

We shall illustrate that operation by the case of testing one of the fragments, third at the left, demonstrating initial values of appropriate components of the ternary vector *f* and of vectors obtained sequentially by the algorithm:

$$f^0, f^1, h^0, h^1, h^0f^1, h^1f^0, g, f^0g, f^1g, h^0 \text{ and } h^1.$$

The check is carried out first on initial coefficient f^-, and then on finite coefficient f^+.

```
f   f^0  f^1  h^0  h^1 h^0f^1 h^1f^0 g  f^0g f^1g h^0  h^1
0-  10   00   10   00   00   00   00   00   00   00   11   Initial
-1  00   01   10   00   00   00   00   00   00   00   11  coefficient
-0  01   00   10   00   00   00   00   00   00   00   11     f^-
11  00   11   10   00   10   00   11   00   11   00   11
                                                           h^0h^1 = 0
              00   11   00   10   11   10   00   11   00   Final
              00   11   00   00   00   00   00   11   00  coefficient
              00   11   00   01   11   01   00   11   00     f^+
              00   11   00   00   00   00   00   11   00
                                                           h^0h^1 = 0
The triad is appropriate
```

Search for appropriate partition

If the considered triad $(p, q) / r$ has appeared suitable, it is possible to assume, that it is a trace of the sought-for appropriate partition. In this case the latter can be found by moving along the track generated from the found trace. By that the value of vector *g* obtained at the previous stage is used, and sets *u* and *v* are sequentially expanding, beginning with initial values $u = (p, q)$ and $v = (r)$.

Expanding set v. Let's begin from set *v*. Sorting out sequentially all elements *s* from set $x \setminus (u \cup v)$, we shall discover among them such ones, at which inclusion in set *v* the partition *u*/*v* remains appropriate. With this purpose three operations are fulfilled for each element *s*:

$$e := S_s^{\vee} g$$
$$h^0 := S_u^{\vee}(f^0e)$$
$$h^1 := S_u^{\vee}(f^1e)$$

and if $h^0h^1 = 0$, then *s* is included into *v* by implementing operations

$$v := v \cup \{s\},\ g := e.$$

So the final value of set *v* is found.

Expanding set u. The maximum expansion of set *u* is found similarly. If it is known, that the required partition *u* / *v* is strict (i. e. w = ∅), it is possible to put $u = x / v$ and, probably, to test the function for decomposability, as the algorithm used is heuristic. Let's remark, however, that the probability of obtaining by this algorithm erroneous solution fast tends to zero with growth of the number of variables *n*.

If the required partition could be non-strict, it is necessary to test all elements from initial value of set $x \setminus (u \cup v)$ for the possibility of including them into set *u*.

Check of the immediate element *s* can be fulfilled by the following heuristic algorithm, which partly implements the procedure circumscribed in [Zakrevskij, 2007a]. The algorithm considers the initial coefficient f^- of the function *f* decomposition by the current value of set *u*, finds orthogonal to it coefficients, checks them for compatibility and, in case of compatibility, includes element *s* in set *u* without further check.

```
$e := u \cup \{s\}$
$h^0 := f^0 - e$
$h^1 := f^1 - e$
$g := S_v^{\vee}(h^0 f^1 \vee h^1 f^0)$
$h^0 := S_u^{\vee}(f^0 g)$
$h^1 := S_u^{\vee}(f^1 g)$
```

If $h^0 h^1 = 0$, then *s* is included into *u* by operation $u := e$.

In such a way the set *u* is found and, therefore, the whole partition *u*/*v*.

Note, that the operation of looking for coefficient f^- is presented in this algorithm in abbreviated form, by expressions $h^0 := f^0 - e$ and $h^1 := f^1 - e$, instead of more detailed

$$h^0 := (\ldots((f^0 - e_1) - e_2) - \ldots) - e_t,$$
$$h^1 := (\ldots((f^1 - e_1) - e_2) - \ldots) - e_t,$$

where $e = (e_1, e_2, \ldots, e_t)$.

Results of experiments

To estimate the efficiency of suggested methods and algorithms and determine the area of their practical application, an experimental system was used based on principles described in [Zakrevskij, 2006a]. It includes a generator of a flow of random examples of initial data, which essentially supplements the well known and widely spread mechanism of Benchmarks, because it enables statistical investigation of regarded algorithms.

The suggested heuristic algorithm was programmed in C++ and tested on computer (Pentium IV, 2.8 GHz) [Zakrevskij, 2006b]. In a series of experiments with completely defined Boolean functions the values *n*, $k = |u|$, $m = |w|$ were set, a random partition *u*/*v* on the set *x* and functions *g*, *h* were generated, then function *f*(*x*) was calculated. After that the given algorithm was fulfilled, which found partition *u*/*v* for the function *f*(*x*), the number *q* of triads scanned by search for traces was fixed, and the total time *t* (in seconds) spent during search for the partition was measured.

The obtained results are represented in the following table, which right part corresponds to splitting of the set of arguments *x* in three parts *u*, *w* and *v*, whenever possible the same size, and left part - in two: *u* and *v*.

It should be noted that the table begins with $n = 14$, because $t \le 0.00$, if $n \le 14$.

Table 1. Results of experiments over 15 examples
with increasing number of variables *n* from 14 up to 28

n	k/m	q	t	k/m	q	t
14	7/7	3	0.00	5/5	39	0.00
15	8/7	2	0.00	5/5	78	0.02
16	8/8	1	0.00	6/5	18	0.01
17	9/8	11	0.02	6/6	34	0.05
18	9/9	4	0.02	6/6	42	0.09
19	10/9	3	0.03	7/6	39	0.17
20	10/10	9	0.11	7/7	3	0.16
21	11/10	1	0.11	7/7	3	0.39
22	11/11	3	0.48	8/7	6	2.41
23	12/11	9	2.17	8/8	16	7.05
24	12/12	3	2.48	8/8	26	18.17

25	13/12	4	5.61	9/8	19	33.16
26	13/13	4	11.64	9/9	3	52.11
27	14/13	19	60.13	9/9	36	187.55
28	14/14	19	1280.67	10/9	3	1585.03

As can be seen from the table, the regarded task of Boolean function decomposition is solved in less than one minute, when the number of arguments does not exceed 26. The amount of memory for representation of Boolean function *f*(*x*) grows quickly, reaching, for example, 2^{28} = 268 435 456 bits for representation of one long Boolean vector, when *n* = 28. Because of the restrictions on the used operation memory that leads to an essential decrease of the calculation speed.

Conclusion

In this paper, the heuristic algorithm is offered for finding such weak two-block partition on the set of variables of a partial Boolean function, on which the function can be decomposed. The algorithm is effective, if there exists a good solution "hidden" in vector representation of the function of many variables. In this case the search of the partition is reduced to recognition of the latter.

Acknowledgement

This work was supported by Belarusian Republican Fond of Fundamental Researches (Project Ф07MC-034).

Bibliography

[Ashenhurst, 1959] Ashenhurst R.L. The decomposition of switching functions. – Proc. International Symposium on the Theory of Switching, Part 1. – Harvard University Press, Cambridge, 1959, pp. 75-116.

[Curtis,1962] Curtis H.A. Design of switching circuits. – Van Nostrand, Princeton, N. J., 1962.

[Povarov, 1954] Povarov G.N. About functional decomposition of Boolean functions. – Reports of the AS of USSR, 1954. – V. 4, No 5 (in Russian).

[Harary, 1969] Frank Harary. Graph theory. – Addison-Wesley Publishing Company : Reading, Massachusetts; Menlo Park, California; London; Don Mills, Ontario. 1969.

[Zakrevskij, 1963] Zakrevskij A.D. Universal system for solving problems the type relay system synthesis. – Annals of Siberian Physical-Technical Institute, 1963. – Issue 42, pp. 9-37 (in Russian).

[Zakrevskij, 2006a] Arkadij Zakrevskij. Combinatorial methods of solving Boolean problems. – Boolean Problems. 7th International Workshop (ed. by B. Steinbach). – September 21-22, 2006, Freiberg, pp.1-10.

[Zakrevskij, 2006b] Arkadij Zakrevskij. A new heuristic algorithm for sequential two-block decomposition of Boolean functions. – Proceedings of 3rd IFAC Workshop on Discrete Event System Design DESDes'06. September 26-28, 2006, Rydzyna, Poland. University of Zielona Gora. – pp. 13-17.

[Zakrevskij, 2007a] Arkadij Zakrevskij. Decomposition of Boolean functions – recognizing a good solution by traces. – International Journal "Information Theories and Applications", vol. 14, 2007, pp. 359-365.

[Zakrevskij, 2007b] Zakrevskij A.D. Decomposition of partial Boolean functions – checking for decomposability at a given partition. – Informatics, 2007, 1(13), pp. 16-21 (in Russian).

[Zakrevskij, 2007c] Zakrevskij A.D. Parallel operations over neighbors in Boolean space. – Proceedings of the Sixth International Conference CAD DD-07, - Minsk 2007, vol. 2, pp. 6-13.

Author's Information

Arkadij Zakrevskij *- United Institute of Informatics Problems of the NAS of Belarus, Surganov Str. 6, 220012 Minsk, Belarus; e-mail: zakr@newman.bas-net.by*