Iterative Method of Minimization of Arbitrary Boolean Functions of Many Variables

Arkadij Zakrevskij

Abstract **— An iterative algorithm of minimization of Boolean functions of many variables based on usage of parallel operations above adjacent elements in Boolean space of arguments is offered. It includes the operation of fast finding of elements of characteristic set with small number of neighbors and creation of implicants defined by them. The iterative procedure of application of this operation to sequentially reduced characteristic set and operation of simplification of the obtained conjuncts lead to a correct solution.**

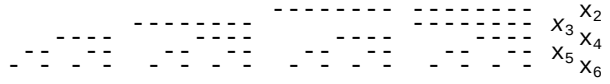
Index Terms—**Boolean function minimization, iterative algorithm, prime implicants**, **computer ex[eriment**

I. Introduction

The classical problem of minimization of Boolean functions in the class of disjunctive normal forms (DNF) can be formulated as follows.

Let us define an arbitrary Boolean function $f(\boldsymbol{x}) \equiv f(x_1, x_2, \ldots, x_n)$ in the standard way by a Boolean vector $\boldsymbol{f}$ with 2^n components numbered from left to right, starting with zero. Here the component number k sets the value of the function f on the set of values of arguments representing the generally accepted binary code $\boldsymbol{b}^k$ of number k.

For example, the following vector $\boldsymbol{f}$ represents the Boolean function f of six variables $x_1, x_2, x_3, x_4, x_5, x_6$, receiving value 1 on 27 collections of values of arguments constituting the characteristic set M^1 = {000000, 000011, 000101, …, 111110} of function f. The ordinal numbers of the appropriate components of vector $\boldsymbol{f}$ are 0, 3, 5, … 62.



10010101 00100110 00101101 10110010
| 00010010 01010100 10001001 00111010 x_1

Note, that such a vector can be interpreted as a perfect normal form of a Boolean function, which terms are represented by single components of the vector and their codes. For example, the code $\boldsymbol{b}^{10}$ = 001010 of element f_{10} defines the complete elementary conjunction

$$\bar{x}_1\ \bar{x}_2\ x_3\ \bar{x}_4\ x_5\ \bar{x}_6 .$$

The problem consists in finding a DNF, minimized as possible, for the function f presented by a given vector $\boldsymbol{f}$. Let us remark, that the traditional methods of solving this task need run-time fast growing with growth of the number of variables n, and practically become unacceptable at $n > 20$, when the number of terms in the perfect DNF is measured in millions [1]. In this connection an original method is offered in the given paper, for obtaining DNF of Boolean functions the number of which arguments can reach 24. The method is based on application of efficient parallel operations above Boolean 2^n-vectors offered in papers [2, 3].

One of such operations is the *operation of conjunctive symmetrizing* the vector $\boldsymbol{f}$ by the variable x_i, denoted below as $S\boldsymbol{f} \wedge i$. At its execution the vector $\boldsymbol{f}$ is divided into 2^{n-1} couples of components adjacent by the variable x_i and both units of each couple gain the value equal to conjunction of the values of these units. Let us remind that adjacent such components of the vector $\boldsymbol{f}$ are named, which correspond to sets of values of arguments distinguishing exactly in one argument.

When the number of variables n exceeds 5, it is convenient to represent vector $\boldsymbol{f}$ as a Boolean matrix of the size $2^5 \times 2^{n-5}$, presenting its 32-component rows by words in computer memory (that is adequate for the majority of modern computers). Then any two units of vector $\boldsymbol{f}$ adjacent by the variable x_i will belong to the same word if $i < 6$ and to different words otherwise, that is possible to use for acceleration of calculations.

II. Build-up of the Boolean matrix of neighborhood N

At the first stage of the offered method the *Boolean matrix of neighborhood* $\boldsymbol{N}$ of the size $n \times 2^n$ is created by means of n-fold application of the operation $S\boldsymbol{f} \wedge i$. Each row $\boldsymbol{n}_i$ of this matrix represents the result of execution of the operation $S\boldsymbol{f} \wedge i$ at a concrete parameter value i (from 1 up to n). The matrix $\boldsymbol{N}$ represents the structure of the characteristic set M^1 of function $f(\boldsymbol{x})$, where this function receives value 1. The element n_i^k of the matrix $\boldsymbol{N}$ receives value 1 if and only if the element f_k of vector $\boldsymbol{f}$ equals 1 and has a neighbor by the variable x_i which also has value 1. Thus, the row $\boldsymbol{n}_i$ of this matrix represents the subset of elements from the set M^1, having in the same set neighbors by the variable x_i, and the column $\boldsymbol{n}^k$ displays, by which variables has neighbors the element from M^1 presented by the component f_k of vector $\boldsymbol{f}$.

Manuscript received 23 March, 2009.

Arkadij Zakrevskij is with the United Institute of Informatics Problems of the National Academy of Sciences of Belarus, 220023, Minsk; e-mail: zakr@tut. by.

Let us illustrate obtaining the matrix $\boldsymbol{N}$ with an example of the same vector $\boldsymbol{f}$, specifying a random Boolean function of six variables:

```
10010101 00100110 00101101 10110010
00010010 01010100 10001001 00111010  f

00010000 00000100 00001001 00110010
00010000 00000100 00001001 00110010  n1

00000101 00100010 00000101 00100010
00000000 00010000 00000000 00010000  n2

00000100 00000100 00100000 00100000
00010000 00010000 00001000 00001000  n3

00010001 00100010 00000000 00100010
00000000 01000100 10001000 00100010  n4

00000101 00000000 00000101 10100000
00000000 01010000 00000000 00001010  n5

00000000 00000000 00001100 00110000
00000000 00000000 00000000 00110000  n6
```

III. Matrix representation of DNF

In the similar form, the Boolean vector of solution $\boldsymbol{g}$ and the Boolean matrix of solution $\boldsymbol{D}$ of the same dimension as $\boldsymbol{f}$ and $\boldsymbol{N}$ may be used to represent the required DNF, considered as some collection of intervals of the Boolean space $M = \{0, 1\}^n$. Some elements of these intervals, by one for each interval, are marked with ones in vector $\boldsymbol{g}$, and the internal variables of these intervals are shown in columns of matrix $\boldsymbol{D}$.

Considering vectors $\boldsymbol{f}$ and $\boldsymbol{g}$ as sets of the elements of space M marked in them, and matrix $\boldsymbol{N}$ and $\boldsymbol{D}$ as subsets of Cartesian product of sets $\boldsymbol{x}$ and M, we shall formulate obvious

Affirmation 1. $\boldsymbol{g} \subseteq \boldsymbol{f}$ and $\boldsymbol{D} \subseteq \boldsymbol{N}$.

In other words, vector $\boldsymbol{g}$ and matrix $\boldsymbol{D}$ can be obtained accordingly from vector $\boldsymbol{f}$ and matrix $\boldsymbol{N}$ by replacement in them some ones with zeros, as it is done in the method circumscribed in this paper.

For example, according to that method, vector $\boldsymbol{f}$

```
10010101 00100110 00101101 10110010
00010010 01010100 10001001 00111010
```

is transformed to vector $\boldsymbol{g}$

```
10010100 00000110 00101000 10010000
00000010 01000000 10000001 00001000
```

and matrix $\boldsymbol{N}$ – to matrix $\boldsymbol{D}$ presenting DNF of function f (it will be shown later).

In the more known form this DNF is set by a ternary matrix, which columns represent elementary conjunctions ($\bar{x}_1\ \bar{x}_2\ \bar{x}_3\ \bar{x}_4\ \bar{x}_5\ \bar{x}_6$, $\bar{x}_2\ \bar{x}_3\ \bar{x}_4\ x_5\ \bar{x}_6$, etc.)

```
1  0-0-0000-111-1
2  00-0-111100111
3  00011-01101001
4  0011-010010-11
5  01-0110-11-01-
6  011100-0-01010
```

The number p of conjunctions in the obtained DNF is equal to the weight of the vector of solution $\boldsymbol{g}$, and the length of DNF (the total of conjunction ranks) is equal to $pn - q$, where q is the number of elements in matrix $\boldsymbol{D}$.

Let us remark, that the introduced above Boolean matrices and vectors at program implementation of the offered method are handled in internal cycles and, therefore, should be represented in RAM, that limits their allowable size. Taking into account parameters of mogern PCs, it is possible to affirm that the given method is implemented fast enough if the number of variables of the minimized Boolean function does not exceed 24.

IV. Selection of elements with small number of neighbors

The build-up of a DNF implementing function f is expedient to begin with the search for elements of the solution kernel – obligatory simple implicants of high rank. Let us term obligatory such a simple implicant of function f, which enters anyone shortest DNF of this function. The search is facilitated by preliminary allocation in vector $\boldsymbol{f}$ of elements of characteristic set M^1 with small number of neighbors, as such elements can determine implicants of high rank displayed by elementary conjunctions with many literals.

The numbers of neighbors for all elements of vector $\boldsymbol{f}$ with value 1 are presented by the finite-valued vector $\boldsymbol{w}$:

```
10010101 00100110 00101101 10110010
00010010 01010100 10001001 00111010 f

0..2.3.3 ..2..22. ..1.23.3 1.62..3.
...2..0. .2.3.2.. 1...3..1 ..332.3. w
```

However, it is more convenient in the long term for program implementation of the subsequent calculations, to sort elements by the number of neighbors and to present the result by a series of Boolean vectors $\boldsymbol{m}_i$ in which ones mark elements with i neighbors.

For the considered example, at $i < 4$, we shall receive:

```
10000000 00000000 00000000 00000000
00000010 00000000 00000000 00000000  m0

00000000 00000000 00100000 10000000
00000000 00000000 10000001 00000000  m1

00010000 00100110 00001000 00010000
00010000 01000100 00000000 00001000  m2

00000101 00000000 00000101 00000010
00000000 00010000 00001000 00110010  m3
```

It is easy to receive these vectors – rows of the Boolean matrix $\boldsymbol{M}$, by efficient component-wise operations above rows of matrix $\boldsymbol{N}$, that essentially accelerates the search for appropriate implicants.

V. Finding of prime implicants

Let us designate through $\boldsymbol{t}^k$ the ternary vector obtained from the Boolean vector $\boldsymbol{b}^k$ (the code of element f_i) by appropriation of value "–" to components marked with ones in column $\boldsymbol{n}^k$ of matrix $\boldsymbol{N}$. Vector $\boldsymbol{t}^k$ can be interpreted as some interval Int_k of the Boolean space $M = \{0, 1\}^n$, and also as

corrsponding elementary conjunction, which can appear a prime implicant of the function *f*.

A f f i r m a t i o n 2. The vector $\boldsymbol{t}^k$ represents an obligatory prime implicant of the function *f* if and only if $Int_k \subseteq M^1$.

A f f i r m a t i o n 3. For each obligatory prime implicant of the function *f* there exists in matrix $\boldsymbol{N}$ a column $\boldsymbol{n}^k$, appropriate to which ternary vector $\boldsymbol{t}^k$ represents this implicant.

Obligatory prime implicants of rank *n* are easily found – they are represented by elements of set M^1 not having neighbors. They are enumerated in vector $\boldsymbol{m}_0$ and represented in corresponding columns of matrix $\boldsymbol{N}$, not containing 1s. Similarly, all obligatory prime implicants of rank $n-1$ are enumerated in vector $\boldsymbol{m}_1$ and also represented in appropriate columns of matrix $\boldsymbol{N}$ – containing one 1.

The detection of obligatory prime implicants of smaller rank is a little bit more difficult. However, it is easy enough for ranks $n-2$ and $n-3$, being carried out by means of component-wise operations above some columns of the neighborhood matrix $\boldsymbol{N}$.

A f f i r m a t i o n 4. Assume, that element f_k of vector $\boldsymbol{f}$ has two neighbors. Then vector $\boldsymbol{t}^k$ represents an obligatory prime implicant of the function *f* if and only if the component f_j of vector $\boldsymbol{f}$ is equal to 1, where $\boldsymbol{b}^j = \boldsymbol{b}^k \oplus \boldsymbol{n}^k$.

For example, $\boldsymbol{b}^{27} \oplus \boldsymbol{n}^{27} = 011011 \oplus 100001 = 111010$, $j = 58$ and $f_{58} = 1$; therefore, ternary vector $\boldsymbol{t}^{27} = -1101-$ represents an obligatory prime implicant.

A f f i r m a t i o n 5. Let us assume, that element f_k of vector $\boldsymbol{f}$ has three neighbors. Then vector $\boldsymbol{t}^k$ represents an obligatory prime implicant of the function *f* if and only $\boldsymbol{n}^k \le \boldsymbol{n}^j$ (the vector $\boldsymbol{n}^k$ is covered with vector $\boldsymbol{n}^j$), where $\boldsymbol{b}^j = \boldsymbol{b}^k \oplus \boldsymbol{n}^k$.

The proof of this assertion is based on the fact, that the vectors $\boldsymbol{b}^k$ and $\boldsymbol{b}^j$ are opposite elements of the interval Int_k of rank 3 and together with their neighbors they cover all eight elements of this interval (see fig. 1).

There is a problem of practical expediency of checking the components of vector $\boldsymbol{f}$ for satisfying the conditions formulated in the assertions 4 and 5. Let us assume that a Boolean function *f* of *n* variables is preset with probability ½ of having value 1 in its components, independently of one another. Consider now some component with value 1 which has *k* neighbors. How probable it is, that this component determines an appropriate prime implicant of rank $n-k$?

A f f i r m a t i o n 6. The probability of such an event is equal to $1/2^t$, where $t = 2^k - (k+1)$.

This probability fast tends to zero. For example, at $k = 2$, 3, 4 and 5 it equals 1/2, 1/16, 1/2048 and 1/67 208 864, accordingly, being independent on *n*.

Therefore, in the offered heuristic algorithm only such single components of vector $\boldsymbol{f}$ are exposed to analysis, the number of the neighbors for which does not exceed three.

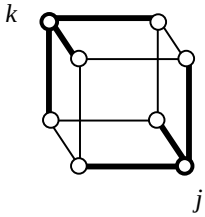


Fig. 1

VI. Obtaining a partial solution

In the offered algorithm implicants of high ranks are sequentially found. The result is fixed by introduction of ones into the vector of solution $\boldsymbol{g}$ (at first $\boldsymbol{g} = \boldsymbol{0}$), definition of corresponding to them columns of the solution matrix $\boldsymbol{D}$, and correction of vector $\boldsymbol{f}^*$, representing the set of yet uncovered elements of the characteristic set M^1.

1. $\boldsymbol{g} := \boldsymbol{m}_0$, $\boldsymbol{f}^* := \boldsymbol{f} \setminus \boldsymbol{m}_0$. So the implicants of rank *n* are found (in the given example there exist two such implicants presented by vectors 000000 and 100110 and defined by elements of vector $\boldsymbol{f}$ with numbers 0 and 38).

2. Here are considered sequentially the elements of vector $\boldsymbol{f}$, marked both in vectors $\boldsymbol{m}_1$ and $\boldsymbol{f}^*$. For a current element $\boldsymbol{f}_k$ the corresponding to it vector $\boldsymbol{b}^k$ is taken, which component marked by one in column $\boldsymbol{n}^k$ of the neighborhood matrix $\boldsymbol{N}$, is changed for symbol «–». The result represents a prime implicant of rank $n-1$. The vectors $\boldsymbol{g}$ and $\boldsymbol{f}^*$ are accordingly corrected: in the first vector one 1 is added, and from the second two 1s are deleted.

So in the given example elements with numbers $k = 18$, 24, 48 and 55 are sequentially considered, to which sets 010010, 011000, 110000 and 110111 correspond and which determine prime implicants of rank $n-1$: 01-010, 0110-0, 110-00, -10111. The vector of solution $\boldsymbol{g}$ receives the value

10000000 00000000 00100000 10000000
00000010 00000000 10000001 00000000 *g*

and accordingly (as a result of deleting covered elements – they are marked by underline) the value of the vector $\boldsymbol{f}^*$ varies:

00010101 00100110 00001100 00010010
00010000 01010100 00000000 00111010 *f**

3. At this stage the elements f_k with two neighbors marked simultaneously in vectors $\boldsymbol{m}_2$ and $\boldsymbol{f}^*$ are considered sequentially and implicants of rank $n-2$ are created.

First elements f_k satisfying the condition of the assertion 4 are found. Corresponding components of vector $\boldsymbol{g}$ receive value 1, the columns $\boldsymbol{d}^k$ of matrix $\boldsymbol{D}$ remain equal to columns $\boldsymbol{n}^k$ of matrix $\boldsymbol{N}$, and elements covered with intervals Int_k are deleted from vector $\boldsymbol{f}^*$.

If the condition of the assertion 4 is not satisfied, one of two neighbors of element f_k is selected. (desirably yet not covered). In the column $\boldsymbol{d}^k$ only one corresponding 1 remains and the operation foreseen for an element with one neighbor is fulfilled.

In the given example (it appears rather simple) that leads to the final solution – covering all elements of the characteristic set M^1, obtaining of the couple of vectors

10010100 00000110 00101000 10010000
00000010 01000000 10000001 00001000 *g*

00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 *f**

build-up of the matrix of solution $\boldsymbol{D}$, obtained from $\boldsymbol{N}$ by deleting of one unity in every column marked in vector

00010000 00100110 00001000 00010000
00010000 01000100 00000000 00001000 $\boldsymbol{m}_2$

and deleting of all unities in the columns which have not been marked in vector $\boldsymbol{g}$:

00010000 00000100 00000000 00010000
00000000 00000000 00000001 00000000 $\boldsymbol{d}_1$

00000100 00000010 00000000 00000000
00000000 00000000 00000000 00000000 $\boldsymbol{d}_2$

00000000 00000000 00100000 00000000
00000000 00000000 00000000 00000000 $\boldsymbol{d}_3$

00000000 00000010 00000000 00000000
00000000 00000000 10000000 00000000 $\boldsymbol{d}_4$

00000100 00000000 00000000 10000000
00000000 01000000 00000000 00001000 $\boldsymbol{d}_5$

00000000 00000000 00001000 00010000
00000000 00000000 00000000 00000000 $\boldsymbol{d}_6$

4. If the vector $\boldsymbol{f}^*$ remains nonzero, the elements with three neighbors marked simultaneously in vectors $\boldsymbol{m}_3$ and $\boldsymbol{f}^*$ are considered. The elements satisfying the condition of the assertions 5 are found, and the corresponding implicants of rank $n - 3$ are entered into solution. At omission of the given condition the implicants of higher rank ($n - 1$ and $n - 2$) are found, defined by these elements.

As a result of the circumscribed procedure of processing of elements of vector $\boldsymbol{f}$, having no more than three neighbors, we obtain a set of implicants constituting a *partial solution S*, and vector $\boldsymbol{f}^*$, representing *residual* – the collection of uncovered by this solution elements of set M^1.

VII. ITERATIVE ALGORITHM

The idea of this heuristic algorithm consists in the following. At first it discovers a partial solution for the vector $\boldsymbol{f}$, then fulfils the same operation for the residual $\boldsymbol{f}^*$, supplementing the set of obtained implicants and accordingly simplifying vector $\boldsymbol{f}^*$ (deleting some ones from it). If after simplification vector $\boldsymbol{f}^*$ will contain some ones, the following iterations are fulfilled up to that moment, when $\boldsymbol{f}^*$ becomes equal to zero.

The implicants obtained at that can be not obligatory and even not prime. Therefore in conclusion they are reduced to prime ones (by lowering the rank), and also the obtained doubles are eliminated.

Let us demonstrate the algorithm on a concrete example, Let $n = 19$ and each element of vector $\boldsymbol{f}$ receives value 1 with probability $p = 1/4$. At this supposition a random vector $\boldsymbol{f}$ with 147 232 elements was generated and the neighborhood matrix $\boldsymbol{N}$ with $2^{19} = 534\,288$ columns was constructed with following distribution of columns on number of ones in them ($N(i)$ columns contain i ones each):

i =	0	1	2	3	4	5	6
$N(i)$ =	296	2087	7180	16352	25357	29868	26913

i =	7	8	9	10	11	12	13	14	15	16
$N(i)$ =	19406	11379	5401	2087	690	172	35	8	1	0

At the first iteration of algorithm the elements of the set M^1 with number of neighbors $i = 0, 1, 2$ and 3 are handled, that is $296 + 2087 + 7180 + 16352 = 25915$ elements. The obligatory prime implicants defined by them are found, altogether $296 + 2082 + 1974 + 80 = 4432$. The total number of the prime implicants retrieved at the first iteration (together with non-obligatory ones) is equal to 24 379. They are marked in the vector of solution $\boldsymbol{g}$ and the elements covered with them are deleted from the variable residual vector $\boldsymbol{f}^*$ (in the beginning equal to $\boldsymbol{f}$). The number of ones in vector $\boldsymbol{f}^*$ becomes equal to 81 910.

At the second iteration a new matrix $\boldsymbol{N}$, representing the relation of neighborhood on elements of the set M^1 marked in the residual vector $\boldsymbol{f}^*$, is considered. In this vector the elements having no more than three neighbors in the same vector are discovered, and determined by them implicants are found. Vectors $\boldsymbol{g}$ and $\boldsymbol{f}^*$ gain new values. If after that $\boldsymbol{f}^* \neq \boldsymbol{0}$, the following iteration is fulfilled.

So for the considered example four iterations are fulfilled, before vector $\boldsymbol{f}^*$ becomes equal to $\boldsymbol{0}$ and all elements of the set M^1 appear covered with 61 477 implicants with the following distribution on ranks (for example, there exist 2 458 implicants of rank 19)):

19 – 2 458, 18 – 33 668, 17 – 25 237, 16 – 114.

As was already said, not all these implicants appear prime. Therefore two procedures of bringing the obtained solution to correct appearance are in conclusion fulfilled. First of them simplifies implicants, appealing to the initial neighborhood matrix $\boldsymbol{N}$ and deleting from implicants some literals if possible. It brings to a new distribution of implicants on ranks:

19 – 296, 18 – 20 933, 17 – 38 617, 16 – 1 631.

The second procedure eliminates doubles among the obtained implicants, therefore the number of latter's is reduced down to 60 972, and their disribution on ranks receives the following appearance:

19 – 296, 18 – 20 933, 17 – 38 353, 16 – 1 390.

VIII. COMPUTER EXPERIMENTS

The explained above iterated algorithm was software implemented in C ++ and tested on the computer (Pentium IV, 2.8 GHz). In order to make more convenient dealing with Boolean vectors, in which the neighbors for considered elements of vector $\boldsymbol{f}$ are enumerated, the neighborhood matrix $\boldsymbol{N}$ was transposed beforehand.

A series of experiments were carried out on a set of pseudo-random Boolean functions with two parameters: the number of variables n and the density of ones r, defining the expected number of ones q in vector $\boldsymbol{f}$ representing Boolean function f ($r = 32q / 2^n$). For example, at $r = 16$ an absolutely random Boolean function is considered, receiving value 1 on each element of Boolean space with probability 1/2.

First of all, the boundaries of practical applicability of the offered algorithm in the space of parameters n and r were defined. The point is, that at a large enough value of parameter r the algorithm can stop the operation after some

number of iterations, because the values $N(0)$, $N(1)$, $N(2)$ and $N(3)$ can appear equal to zero, while vector $\boldsymbol{f}^*$ will remain distinct from $\mathbf{0}$.

For example, if n=17 and r=15, the algorithm stops after eight iterations, finding only 636 implicants. But at n =17 and r=14 it discovers after 90 iterations 20 077 implicants covering the whole set M^1 (after consequent simplification of these implicants and eliminating doubles their number is reduced to 19 811). Therefore, the couple $(n, r) = (17, 14)$ is a unit of the upper bound of the algorithm usage.

In table I the basic characteristic of this boundary obtained experimentally is represented. The table strings corresponds to the numbers of variables n (from 4 up to 23) and appropriate to them maximum values of the parameter r, at which the program works correctly.

TABLE I

n	r	N	C	S	It	T
4	16	5	3	10	1	0.00
5	16	16	8	27	1	0.00
6	16	31	14	62	1	0.00
7	16	62	31	169	1	0.00
8	16	138	58	360	2	0.00
9	16	274	107	744	3	0.00
10	16	556	194	1513	3	0.00
11	16	1083	379	3355	4	0.01
12	16	2203	735	7127	5	0.03
13	16	4337	1414	15057	7	0.09
14	16	8734	2780	32266	12	0.35
15	15	16404	5313	67324	13	1.01
16	14	31021	10181	139827	13	3.12
17	14	61150	19811	291507	90	24.37
18	13	114347	38007	500357	21	42.26
19	12	212620	72343	1220742	12	148
20	11	392995	137215	2462996	10	610
21	11	786345	270642	5121875	18	2499
22	10	1442274	512529	10255122	10	8858
23	9	2620069	966357	20386490	8	31064

The following values are shown in strings of this and other tables: N – the number of ones in vector $\boldsymbol{f}$ (the number of implicants in the perfect DNF of function f), C – the number of implicants in obtained DNF, S – the length of obtained DNF (the total of implicant ranks), Q_k – the number of implicants of the rank k in obtained DNF (at $k = n, n–1, n–2, n–3, n–4$), It – the number of iterations, T – the run-time in seconds.

The table II shows the behavior of the algorithm at the number of variables n = 17. It is evident, that the number of iterations It and the run-time T grow at increase of the density of ones r.

TABLE II

n	r	N	C	S	It	T
17	2	12285	7856	127689	2	0.27
17	4	20427	11117	177205	2	0.53
17	6	28594	13761	215604	3	1.90
17	8	36507	15621	240722	3	4.04
17	10	44756	17348	263178	4	6.49
17	12	52999	18691	279341	7	8.43
17	14	61150	19811	291507	90	24.52

The behavior of the algorithm at the greatest possible for it number of variables (n=24) is shown in table III. At increase of the density of ones r by one the run-time T grows more than twice, reaching 6.8 hours at $r = 4$.

TABLE III

n	r	N	C	S	It	T
24	1	1047350	685881	15982597	2	1693
24	2	1571532	919682	21231157	2	4068
24	3	2095590	1124293	25759214	3	10695
24	4	2619724	1297946	29532155	3	24348

Table IV shows some additional results of computer experiments at the number of arguments n = 17. Apparently, at increase of the density of ones r the distribution of the obtained implicants on ranks fast varies in favor of implicants of smaller ranks.

TABLE IV

r	Q_{17}	Q_{16}	Q_{15}	Q_{14}	Q_{13}	It	T
2	2283	5283	290	0	0	2	0.27
4	1147	8162	1802	6	0	2	0.53
6	417	8406	4887	51	0	3	1.90
8	135	6370	8883	233	0	3	4.04
10	41	3864	12456	986	1	4	6.49
12	6	1880	13899	2896	10	7	8.43
14	1	684	12842	6224	60	9	24.52
						0	

IX. CONCLUSION

In the offered algorithm of minimization of Boolean functions of many variables (up to 24) the following ideas are implemented:

1) The role of elementary operands is played by Boolean vectors with 2^n components representing arbitrary Boolean functions of n variables.

2) The Boolean matrix of neighborhood $\boldsymbol{N}$ by the size $n \times 2^n$ is created, mapping the structure of the characteristic set M^1.

3) With its help prime implicants of four higher ranks are quickly found, coating a part of the set M^1.

4) The structure of the residual is represented by a new matrix $\boldsymbol{N}$, the new implicants of high rank are found, etc. The iterations will stop, when the set M^1 becomes empty.

5) Finally, the obtained implicants are transformed to prime ones and any doubles are eliminated.

The algorithm is implemented by the efficient program, developed by Nikolaj Toropov from the UIIP of NAS of Belarus. It can appear useful at solution of systems of the Boolean equations, verification of logic circuits and solution of other labor-consuming tasks of the theory of Boolean functions.

REFERENCES

[1] Zakrevskij A. D. Logical design of cascade circuits. – Moscow, 1981 (in Russian).

[2] 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.

[3] Arkadij Zakrevskij. Programming Calculations in Many-Dimensional Boolean Space // Radioelectronics & Informatics, No 1(40), January-March 2008, pp. 19-25.