Automation and Remote Control, Vol. 65, No. 6, 2004, pp. 938–954. Translated from Avtomatika i Telemekhanika, No. 6, 2004, pp. 112–128.
Original Russian Text Copyright © 2004 by Kolpakov, Latypov.

Approximate Algorithms for Minimization of Binary Decision Diagrams on the Basis of Linear Transformations of Variables

A. Kolpakov and R. Kh. Latypov

Kazan University, Kazan, Russia

Received December 17, 2003

Abstract—Algorithms for an approximate minimization of binary decision diagrams (BDD) on the basis of linear transformations of variables are proposed. The algorithms rely on the transformations of only adjacent variables and have a polynomial complexity relative to the size of the table that lists values of the function involved.

1. INTRODUCTION

One of the main problems in the design of the equipment involving digital devices is the problem of optimizing the sizes of circuits. A decrease in the sizes of circuits at the stage of the design of devices makes it possible, on the one hand, to save means at the stage of production of circuits and can lead, on the other hand, to an appreciably increased rate of the computation process on account of a decrease in the delay time.

The idea of the linear transformation of input variables in the problem of minimization of circuits, which is put forward in [1], lies in the fact that the calculation of the initial output function is broken up into two steps. First, instead of the initial output function, the circuit calculates a certain "changed" function and then the second step is taken to carry out the linear transformation of variables so as to derive a "correct" output value. Thus, the value of the initial function is equal to $f(x_1, \ldots, x_n) = L(\widetilde{f}(x_1, \ldots, x_n))$, where L is the linear transformation of input variables and $\widetilde{f}$ is the function calculated by the circuit. Various aspects of the linear transformations of variables are set out in [2, 4]. Another approach based on the linear presentation of the structure of data is evolved in [5–8].

The practical implementation of a linear transformation makes sense where the linear transformation L and the function $\widetilde{f}$ are performed in a simpler way than the initial function; the minimization problem consists in the search for the simplest function $\widetilde{f}$ in the sense of its implementation. Whereas the effective implementation of the linear transformation on the basis of binary adders has a satisfactory solution [9], the exact minimization of circuits on the basis of linear transformations is unfortunately made difficult in view of a large number of possible transformations, which grows fast with an increase in the sizes of circuits. For this reason, the problem of the search for approximate minimization algorithms is urgent.

The binary decision diagrams (BDD) offer an effective method of handling large data structures in an effort to optimize and verify them and calculate various characteristics [10–14]. The BDD structure directly maps into the architecture of hardware tools, for example, programmable logic units, and finds use at the stage of synthesis of these units [14–16].

The linear transformations for the BDD minimization represent a reassuring concept because data structures proper remain invariable and only input variables undergo changes (recoding).

Instead of the operations performed on the table of the function, only transformations of input variables are produced, for which reason use is made of $n \times n$ matrices rather than matrices of dimension $2^n \times 2^n$, where n is the number of variables. The simplest linear transformation is the renaming or the permutation of variables. The complexity of the search for the optimal permutation of variables is displayed in [17, 18]. It amounts to $O(n^2 * 3^n)$, where n is the number of variables. The use of this method reduces the time it takes to seek the optimal result, on the average, by 50%. But in the sense of complexity, $O(3^n)$ is a very large value. The algorithm is set out in [19] that serves to find an optimal linear transformation of variables of the Boolean function with the aim to minimize exactly a requisite BDD. It is shown that the BDD can markedly be cut down using the linear transformations. In view of a large computation complexity, this algorithm is suitable only for functions that have no more than 6 incoming variables. In [20], the BDDs of incompletely prescribed functions are minimized with the use of a strict symmetry: the symmetric blocks are fixed and the common permutations are used for the remaining portion. The method is sufficiently effective, but it is proved that the results will be worse than in the case of common permutations, although the processing time decreases.

The analysis of the works reveals that the effective minimization algorithms are set up only for particular cases of linear transformations. An exact minimization is laborious. In this work, we suggest algorithms for an approximate minimization of BDDs. As the basis of the algorithms, we use transformations from a more common class in comparison with the linear transformation class, namely, the class of affine transformations of adjacent variables. Experimental results are laid down that confirm the effectiveness of the suggested algorithms.

The work presents the issues under discussion in the following sequence. First, we describe basic definitions of the linear and affine transformations and the binary decision diagrams. Then, we investigate the effect of linear and affine transformations on BDDs and single out the idea of minimization algorithms. Further, we present algorithms for an approximate minimization of BDDs and some experimental results.

2. BINARY DECISION DIAGRAMS

We will consider a Boolean function f: $B^n \Rightarrow B$ of variables $x_1, \ldots, x_n$. It is possible to correlate to any Boolean function a binary tree (Fig. 1) in which each nonterminal vertex is denoted by a variable x_i and two branches emerging from it identify the decomposition of the function into two cofunctions $f = \overline{x}_i f_{x_i=0} \vee x_i f_{x_i=1}$, $1 \leq i \leq n$.

In developing BDDs from a binary tree, use is made of the reduction of excessive subgraphs. We will consider the basic types of applied reductions.

Type I. *Reduction of isomorphic subgraphs.* It is only one of all the isomorphic subgraphs that is left intact. The input of the cancelable subgraph is redirected to the input of the isomorphic subgraph (the example of this reduction is shown in Fig. 2a).

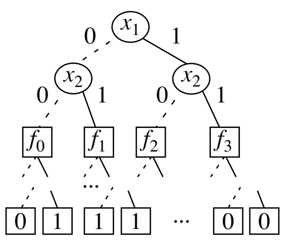

Fig. 1. Example of a binary tree (the expansion in two variables is shown in detail).

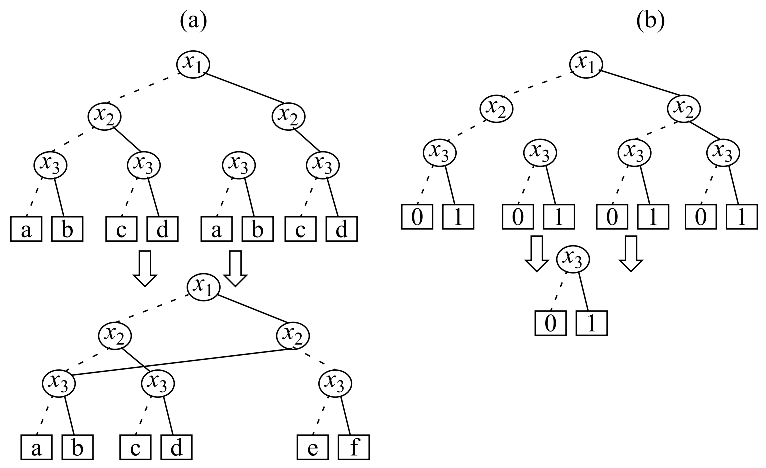

Fig. 2. Operations on the binary tree for development of BDDs.

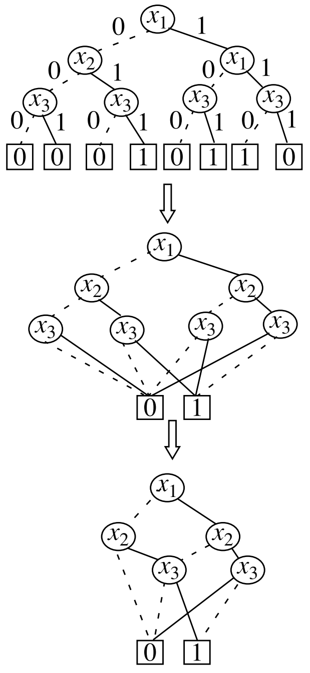

Fig. 3. Example of design of a BDD.

Type S. *Reduction of redundant vertices.* The vertex a is canceled, both outputs of which are the inputs of one and the same vertex b. The input of the canceled vertex a is redirected to the input of the vertex b (the example is shown in Fig. 2b).

The BDD of any kind must be ordered, i.e., any variable must occur merely once on any of the paths from a vertex to the root of the graph, and it must be reduced, i.e., more reductions than the possible ones applied to the diagrams of this type do not exist. The reduced and ordered BDDs (ROBDDs) prescribe a canonical presentation of a Boolean function.

Example 1. Figure 3 illustrates the process of developing a BDD for the function f of variables x_1, x_2, x_3, which is preset by the truth vector 00010110.

3. LINEAR AND AFFINE TRANSFORMATIONS OF VARIABLES

The linear transformation of variables means that each variable is replaced by a certain modulo 2 sum of variables, in which case some variables may not enter into the sum under consideration. In addition, the affine transformation [21] involves the negation assigned to some variables.

Example 2. The affine transformation specified by equalities for the input variables

$$\begin{gathered} x_1, x_2, x_3, x_4, \\ y_1 = \overline{x_1 \oplus x_3}, \\ y_2 = x_4, \\ y_3 = x_2 \oplus x_3, \\ y_4 = x_2 \end{gathered}$$

or in the matrix form

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 \oplus x_3 \oplus 1 \\ x_4 \\ x_2 \oplus x_3 \\ x_2 \\ 1 \end{pmatrix}, \tag{1}$$

means that $\overline{x_1 \oplus x_3}$ instead of x_1 is fed to the first input, x_4 instead of x_2 is fed to the second input, $x_2 \oplus x_3$ is fed to the third input, and x_2 is fed to the fourth input.

Let $X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$ be a prescribed vector of input variables, $Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}$ be a vector of transformed variables, and $D = \begin{bmatrix} d_1 \\ d_2 \\ \dots \\ d_n \end{bmatrix}$ be a certain binary vector. Then, the linear transformation of variables is preset by the relation $Y = TX$ and the affine transformation is preset by the expression $Y = TX \oplus D$ for a certain nonsingular matrix T, all operations being performed on the field $GF(2)$. The matrix T defines a method of the linear transformation. The nonsingularity of the matrix T implies that it is possible to restore uniquely the initial input vector X if the vector Y is known, i.e., there exists an inverse transformation that is also a linear one. The unit value of the component of the vector D signifies that the negation is assigned to an appropriate transformed variable. The affine transformation can be reduced to a linear one if we include an additional unit digit in the column of variables and border the transformation matrix:

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \\ 1 \end{bmatrix} = \begin{bmatrix} t_{11} t_{12} & \dots & t_{1n} d_1 \\ t_{21} t_{22} & \dots & t_{2n} d_2 \\ \dots\dots & \dots & \dots\dots \\ t_{n1} t_{n2} & \dots & t_{nn} d_n \\ 0 \quad 0 & \dots & 0 \quad 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \\ 1 \end{bmatrix}. \tag{2}$$

Linear transformations can lead to an appreciable decrease in the number of vertices (nodes) in a BDD.

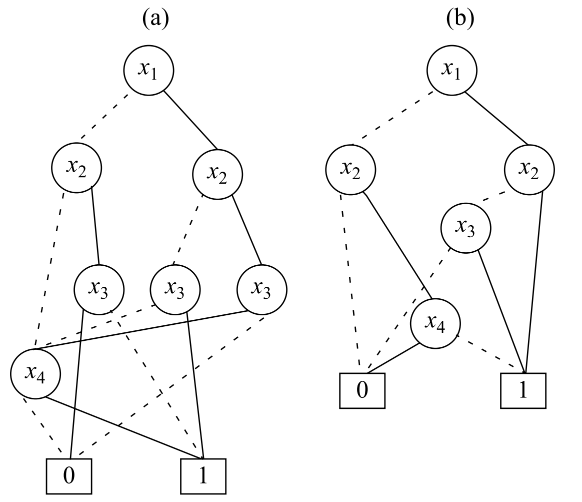

Fig. 4. Result of applying the affine transformation L^{-1} to the function $f(y_1, y_2, y_3, y_4) = 0101110001110001$0.

Example 3. Let us consider two Boolean functions prescribed by the truth vectors $f(y_1, y_2, y_3, y_4) = 0101110001110001$ and $\widetilde{f}(x_1, x_2, x_3, x_4) = 0000101000111111$. Figure 4a displays the BDD corresponding to the function f and Fig. 4b displays appropriate functions $\widetilde{f}$. We will denote by L the affine transformation (2) described in Example 2. It is easy to verify that $f(y_1, y_2, y_3, y_4) = L(\widetilde{f}(x_1, x_2, x_3, x_4))$. On the other hand, the function $\widetilde{f}$ is found from the function f by applying an affine transformation L^{-1} that is inverse to L, which is given by the equalities

$$\begin{gathered} x_1 = \overline{y_1 \oplus y_3 \oplus y_4}, \\ x_2 = y_4, \\ x_3 = y_3 \oplus y_4, \\ x_4 = y_2. \end{gathered}$$

The number of nodes in the BDD dropped from 7 to 5 as a result of this transformation.

Later on, under the minimization of a BDD will be understood a decrease in the number of nodes in the BDD. In the course of the minimization of a BDD with the aid of linear transformations, the main problem lies in a large number of the versions of transformations: in all, there exist $\prod\limits_{i=0}^{n-1}(2^n - 2^i)$ nonsingular linear transformations of the variables $x_1, \ldots, x_n$ and $2^n \prod\limits_{i=0}^{n-1}(2^n - 2^i)$ affine transformations. These numbers are very large, and so even at the available powers of computers, it is possible to solve the minimization problem by exhaustion only up to 6 variables inclusive.

4. EFFECT OF AFFINE TRANSFORMATIONS ON BDDs

Let the initial function be expanded in the two variables

$$f = \overline{x}_i\overline{x}_j f_{x_i=0,x_j=0} \vee \overline{x}_i x_j f_{x_i=0,x_j=1} \vee x_i\overline{x}_j f_{x_i=1,x_j=0} \vee x_i x_j f_{x_i=1,x_j=1}$$

or

$$f = \overline{x}_i\overline{x}_j f_0 \vee \overline{x}_i x_j f_1 \vee x_i\overline{x}_j f_2 \vee x_i x_j f_3.$$

We will consider the case of $i = 1, j = 2$ and the effect of the linear nonsingular transformations of input variables x_1 and x_2 on the function (there is a total of five nonidentical linear transformations).

The linear transformation $x_1 = x_1 \oplus x_2, x_2 = x_2$ results in the permutation of f_0, f_3, f_2, f_1 cofunctions; the transformation $x_1 = x_1$, $x_2 = x_1 \oplus x_2$ results in the permutation of f_0, f_1, f_3, f_2; the transformation $x_1 = x_1 \oplus x_2, x_2 = x_1$ results in the permutation of f_0, f_2, f_3, f_1; the transformation $x_1 = x_2, x_2 = x_1 \oplus x_2$ results in the permutation of f_0, f_3, f_1, f_2; and transformation $x_1 = x_2, x_2 = x_1$ results in the permutation of f_0, f_2, f_1, f_3.

The considered transformations do not afford the permutations of functions f_0 and f_k, $k = 1, 2, 3$. To avoid this restriction, we will consider the vector $(x_1, x_2, 1)$ instead of the vector (x_1, x_2). This enables us introduce the negation of variables, i.e., to extend the transformation to the affine one and reduce the affine transformation $x_1 = a_{11}x_1 \oplus a_{12}x_2 \oplus b_1$, $x_2 = a_{21}x_1 \oplus a_{22}x_2 \oplus b_2$ to the linear transformation represented by the matrix (1) of dimension 3.

Let us note that the number of possible permutations of four cofunctions is equal to $4! = 24$. This number coincides with the number $2^n \prod\limits_{i=0}^{n-1} (2^n - 2^i) = 4(4-1)(4-2) = 24$ of possible affine transformations of two elements. Thus, to any permutation of cofunctions in the expansion in two variables there corresponds uniquely an affine transformation of two variables, and vice versa.

We will consider the affine transformations of a large number of variables. For $n > 2$, the number $2^n(2^n-1)(2^n-2)(2^n-3)\ldots 2 \times 1$ of the permutations of functions is higher than the number $2^n(2^n-1)(2^n-2)(2^n-4)\ldots(2^n-2^{(n-1)})$ of possible affine transformations. Thus, for the transformations applied to three variables and more, there is an insufficient number of affine transformations for prescribing all permutations of cofunctions in the Shannon expansion.

The main idea of the minimization algorithm involved lies in the fact that at each step, the manipulations are performed only at adjacent levels, which corresponds to the expansion in two adjacent variables. In this case, instead of affine transformations, use is made of various permutations of cofunctions.

5. DEVELOPMENT OF THE ALGORITHM

In the minimization of a BDD, the basic problem is the search for the criterion that enables us to define at each step of the operation which of the possible transformations of adjacent variables will lead to the BDD minimization. In this case, the complexity of operation of the algorithm must be markedly lower than that of search algorithms.

As shown above, in the case of the expansion of the function in the first two variables, all values of the cofunctions can be rearranged between one another with the aid of affine transformations. We will now consider the expansion in three variables (Fig. 5).

In applying similar transformations to the variables x_2 and x_3, instead of the considered variables x_1 and x_2, the permutable values of the function will become the nodes of level 3, which are shown in Fig. 5 in the form of functions f_{ji}, where $i = 0, 1; j = 0, 1, 2, 3$. Here, these transformations rearrange the functions within the rectangles denoted in Fig. 6 in synchronism with the rest of the rectangles. In other words, in the permutation of f_{j0} and f_{k0}, the functions f_{j1} and f_{k1} also undergo the permutation. In the general case, the process will occur in a similar way, and the number of rectangles for the level L, $L > 1$, will be equal to 2^{L-2}.

In the course of the minimization of Boolean functions, it is revealed that the least number of BDD nodes proves to be in a function of such a form that in the binary tree corresponding to this function, the end nodes (the resultant values of the function) have packages of identical numbers of zeros or units, which rather large in length and lie in one subtree. Next, the linear transformations are used for the ordering of the tree in the sense of the location of zeros and units.

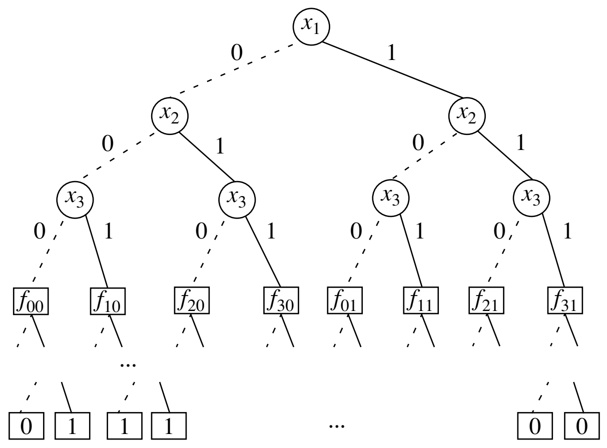

Fig. 5. Example of a binary tree (the expansion of the function in three variables is shown in detail).

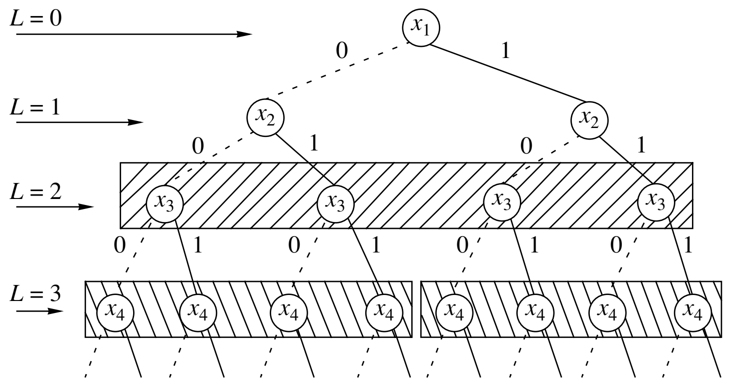

Fig. 6. Presentation of binary tree sections subjected to the linear transformation with respect to two preceding variables, depending an the level L.

Definition 1. The weight of a subtree with the vertex g will be called the weight $w(g)$, namely, the number of resultant units in the subtree with the vertex g (i.e., units at terminal vertices).

Definition 2. The balance of a binary tree will be called the difference between the weights of the right and the left branch, or the subtrees.

Definition 3. The specific balance $w(f, L)$ at the level L of a binary tree will be called the sum of balances $N = 2^L$ of the subtrees extending from the level L. In other words, $w(f, L) = \sum_{i=0}^{\frac{N}{2}-1} (w_{1i} - w_{0i} + w_{3i} - w_{2i})$, where w_{0i}, w_{1i}, w_{2i}, and w_{3i} are the weights of requisite subtrees.

Definition 4. An alternative specific balance at the level L of the binary tree will be called the balance $w_a(f, L) = \sum_{i=0}^{\frac{N}{2}-1} (w_{1i} - w_{0i} - w_{3i} + w_{2i})$.

We will consider the tree up to the second level (Fig. 6). Because there is only one action area, we will merely order 4 subtrees in their weight by rearranging the elements at level 2 with the aid of transformations of the elements at levels 0 and 1 (these are the transformations with respect to the variables x_1 and x_2).

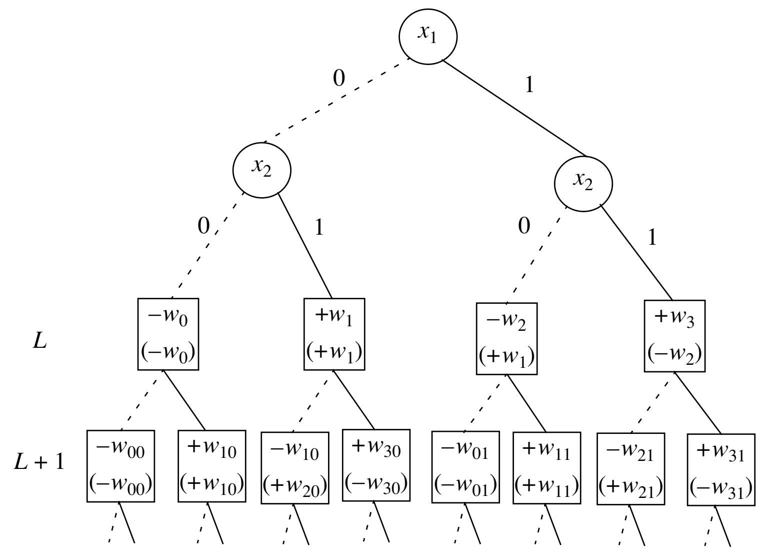

Fig. 7. Presentation of versions of the summation of weights.

At the next level (Fig. 6), we already have 2 areas of the synchronous action in using the transformations with respect to the variables at the preceding two levels (x_2 and x_3). We will consider the weights of 8 outgoing subtrees w_{ij}, $i = 0,1,2,3,\ j = 0,1$. Under the transformation that leads to the permutation of w_{00} and w_{10}, the permutation of w_{01} and w_{11} will also occur. This procedure will take place at any level, starting from the second one, and there will be 2^L quadruples of this type at the level L.

To each element at the level L of the binary tree there corresponds a pair of elements at the next level. In other words, to provide a way for a further maximization of the specific weight at a given level, it is necessary to change the weights of these pairs relative to one another at the level $L+1$. This will redistribute the weights of elements corresponding to them at the level L. Because to one quadruple at the level L there correspond two quadruples at the level $L+1$ and the two versions out of all the versions have no principal meaning, the versions of the change in the difference of weights become equal to 2: $-\ +\ -+$ or $-\ +\ +-$ (Fig. 7).

Let us examine a general case. We assume that there is a certain BDD developed for $L+1$ of incoming elements. We will consider the above—mentioned transformations for variables x_L and x_{L+1}. These transformations act synchronously in a similar way on the terminal quadruples of elements. Because to one quadruple at the level L there correspond two quadruples at the level $L+1$, in order to provide a way of increasing the difference between the weights of the left and the right portion of this quadruple, it is necessary to increase at the most the difference between the weights of elements in the left and the right pair. It is possible to reach this effect by way of the operation on the quadruples at the level $L+1$. Examining a greater number of elements than 8 at the level $L+1$ makes no sense because the cells each of 4 elements are maximized at the level L (because the preferable circuit is the circuit $-+-+$, etc., and the circuit $-++-$ is necessary in view of the limited area of the action of transformations—blocks each of four values, located in one subtree).

Next, we present the stages of developing an approximate algorithm for minimizing BDDs.

5.1. Algorithm for Ordering a Binary Tree

This algorithm preliminarily puts in order a binary tree, which leads to an appreciable minimization of a BDD. The algorithm is effective for the functions of a small number of variables.

The algorithm maximizes the balance of a tree by maximizing the specific balances or alternative specific balances of subtrees.

(1) $L = 1$. Order (rearrange) the functions f_0, f_1, f_2, f_3 in such a way that $w(f_0) \leq w(f_1) \leq w(f_2) \leq w(f_3)$.

(2) $L = 2$. Order $f_{00}, f_{10}, f_{20}, f_{30}, f_{01}, f_{11}, f_{21}, f_{31}$ by way of the permutations of pairs (f_{00}, f_{01}), (f_{10}, f_{11}), (f_{20}, f_{21}), (f_{30}, f_{31}) so that $w(f_{00}) + w(f_{01}) \leq w(f_{10}) + w(f_{11}) \leq w(f_{20}) + w(f_{21}) \leq w(f_{30}) + w(f_{31})$. In the case of the successful ordering (the value of $w(f, L)$) increased), return by one step or try to increase the value of $w_a(f, L)$ by way of the same permutations. If the ordering is successful (the value of $w_a(f, L) > w(f, L)$), return to $L = L - 1$ by one step.

(3) $L = L + 1, N = 2^L$. Order $f_{ij},\ i = 0, \ldots, 3,\ j = 0, \ldots, \dfrac{N}{2} - 1$ so that $\sum\limits_j w(f_{0j}) \leq \sum\limits_j w(f_{1j}) \leq \sum\limits_j w(f_{2j}) \leq \sum\limits_j w(f_{3j})$ by way of the permutations of four subsets: $(f_{00}, \ldots, f_{0,N/2-1})$, $(f_{10}, \ldots, f_{1,N/2-1})$, $(f_{20}, \ldots, f_{2,N/2-1})$, $(f_{30}, \ldots, f_{3,N/2-1})$. Further, act in the same way as at Step 2.

The algorithm terminates if L is equal to the number of levels in the tree, otherwise we return to Step 3.

The disadvantages of the designed algorithm are the following. The examined versions of the circuits $(- + + -$, and so on) are urgent only for binary trees of a small size, which depend merely on three variables. For the functions of a higher number, we obtain more blocks of the types

$$(- + + - - + + -) \quad (- + + - - + + -)$$

or

$$(- + - + - + - +) \quad (- + - + - + - +).$$

But because the equivalent versions are also versions of the types

$$(- + + - - + + -) \quad (+ - + + - +)$$

or

$$(- + - + - + - +) \quad (+ - + - + - + -),$$

it is necessary to consider the summands taken in absolute values. In addition, on account of the conjugation of $w_a(f, L)$ and $w(f, L)$, the recycling of the algorithm may occur.

5.2. Modified Ordering Algorithm (A1)

1. $L = 1,\ N = 2$. Order (rearrange) the functions $f_0, f_1,\ f_2, f_3$ in such a way that $w(f_0) \leq w(f_1) \leq w(f_2) \leq w(f_3)$.

2. $L = L + 1,\ N = 2N$. Maximize the quantity

$$\sum_{i=0}^{N/2-1} |w(f_{0i}) + w(f_{1i}) - w(f_{2i}) - w(f_{3i})|,$$

by rearranging the subsets $((f_{00}, \ldots, f_{0,N/2-1})$, $(f_{00}, \ldots, f_{0,N/2-1})$, $(f_{20}, \ldots, f_{2,N/2-1})$, $(f_{30}, \ldots,$ $f_{3,N/2-1})$.

If the permutation occurred successfully (the maximizable value changed), return by one level.

The cycle comes to an end if L is equal to the number of levels in the tree, otherwise return to Step 2.

This version enabled us to do away with the ambiguity of the criterion and a different interpretation of the sequences $+-$. In addition, in view of the simplification of the criterion, there is no recycling of the algorithm. However, the presented algorithm does not afford the uniqueness of the function because a few versions of the location of quadruples are possible.

To reduce the size of the class of functions obtained after the use of the modified algorithm, we can apply some transformations that do not worsen the described criterion, but prescribe more clearly the form of the Boolean function. As one of the versions, use can be made of the method involving the two-level minimization of the binary diagram, which also operates only on adjacent variables, in order to reduce the size of the binary diagram that selects possible transformations of adjacent variables. A moderate effect results from the use of the ordering of the value of the function after the application of the above-described algorithm before the two-level minimization of the binary diagram. Further, we design a few algorithms, the aim of which is either to reduce the function to a definite form or to minimize the size, or to attempt to form a binary tree of the kind that may afford performing the transformations more effectively.

5.3. Two-Level Search Method (A2)

We carry out the search using the transformations only with respect to adjacent variables.

(1) $L = 1,\ N = 2$. Minimize the number of nodes in a BDD by rearranging the functions f_0, f_1, f_2, f_3.

(2) $L = L + 1,\ N = 2N$. Minimize the number of nodes in a BDD by rearranging the subsets $(f_{00}, \dots, f_{0,N/2-1})$, $(f_{00}, \dots, f_{0,N/2-1})$, $(f_{20}, \dots, f_{2,N/2-1})$, $(f_{30}, \dots, f_{3,N/2-1})$.

If the permutation occurred successfully (the number of nodes decreased), return to the upper position by one level.

The cycle terminates if L is equal to the number of levels in the tree, otherwise return to Step 2.

5.4. Algorithm for Symmetry Revealing (A3)

(1) $L = 1,\ S = 0, N = 2$. Reveal the symmetry in f_0, f_1, f_2, f_3: if there is a pair of equal values, perform permutations and define the form XXYY or XXYZ or YZXX. In obtaining 2 pairs of equal values (the case (XXYY)), fix the first level, setting $S = 1$. This means that the first level does not undergo the transformation any more.

(2) $L = L + 1$, $N = 2N$. In a similar way, consider the subsets

$$(f_{00}, \dots, f_{0,N/2-1}), \quad (f_{00}, \dots, f_{0,N/2-1}), \quad (f_{20}, \dots, f_{2,N/2-1}), \quad (f_{30}, \dots, f_{3,N/2-1}).$$

If there are no element-wise equal subsets, then pass on to the next level. If there are element-wise equal subsets, perform permutations and define the form XXYY or XXYZ or YZXX. Here, if 2 equal pairs are set up (the case (XXYY)), then in the case of $S \geq L - 1$, return to the upper position by one level, otherwise fix the upper levels, setting $S = L$. If there are no two equal pairs, return to the upper position by one level.

(3) The cycle terminates if L is equal to the number of levels in the tree, otherwise return to Step 2.

The idea of this algorithm lies in the revealing of the cases of the symmetry and the "lifting" of it to higher levels where it would remain in the invariable state, because the symmetry is the most optimal version for the reduction of the number of nodes in the binary tree.

5.5. Homogeneity Algorithm (A4)

In the algorithms described above, the way of estimating the number of steps is intricate. For this reason, new ideas appeared for the design of a clearer algorithm that would minimize the value of the common criterion at each level of operation. This will enable us to assert at once that there no infinite cycles in the operation of the program because at each stage an improvement of the common criterion occurs, such that it does not affect those components of the criterion that remain intact at a given level.

Definition 5. The homogeneity H of the function will be called the modulus of the difference between the number of units and zeros in a value of the function

$$H = |\text{ number of elements 1---number of elements 0}|.$$

We will consider the assignment of the function in the form of a binary truth vector of length $N = 2^n$. Let the number of units in the collection be equal to n_1 and the number of zero elements in the collection be equal to n_0, so that $N = n_1 + n_0$ and thus $H = |n_1 - n_0| = |N - 2n_0|$.

The homogeneity of the level L is given by the formula

$$H_L = \sum_{i=0}^{2^L-1} |N - 2n_{0i}|,$$

where n_{0i} is the number of zeros in an appropriate minorant function. Accordingly, the homogeneity of the function is given by formula

$$H_f = \sum_{L=1}^{n-1} H_L.$$

On the basis of the introduced definitions, we develop a new algorithm with the criterion of maximization of the function homogeneity.

(1) $L = 1$, $N = 2$. Order (rearrange) the functions $f_0, f_1,\ f_2, f_3$ so that $w(f_0) \leq w(f_1) \leq w(f_2) \leq w(f_3)$.

(2) $L = L+1$, $N = 2N$. Maximize the quantity H_L by rearranging the subsets $(f_{00}, \ldots, f_{0,N/2-1})$, $(f_{00}, \ldots, f_{0,N/2-1})$, $(f_{20}, \ldots, f_{2,N/2-1})$, $(f_{30}, \ldots, f_{3,N/2-1})$.

If the permutation occurred successfully (the maximizable value changed), return to the upper position by one level.

(3) The cycle terminates if L is equal to the number of levels in the tree, otherwise return to Step 2.

The new approach enabled us to simplify extremely the criterion and thus obtain new results. In addition, the possibility appeared that permits us to obtain theoretically the upper estimate of the complexity of operation of the algorithm.

It is known that in view of a selected criterion, it is possible to perform no more than one successful permutation at one level in succession, which satisfies the imposed constraints. Moreover, a specific feature of the algorithm is the sequence of its operation. At each level, two outcomes occur: the successful transformation and the unsuccessful one (i.e., transformations were not found, such as could improve the criterion under consideration). In view of the structure of the tree and the use of the transformations only with respect to adjacent variables, merely the preceding level is brought into use at the current permutations. This is attributable to the fact that the transformations of adjacent variables, which correspond to a given level of the tree, carry out the synchronous permutations of elements of the level that are broken up into quadruples. In view of the structure of the binary tree, to the quadruple of elements of the current level there corresponds a pair of the elements of the preceding level and just only one element of a yet higher level. Hence it follows that the permutation of elements in the quadruple can affect only the preceding level.

Let S be the number of successful transformations.

Lemma 1. *The number of steps of the algorithm is equal to* $2S + n - 1$.

We obtained the answer to the question as to how many steps the examined algorithms have depending on the number of successful transformations. For the final estimate, we need to know a value of S.

Theorem 1. *The value of the homogeneity of a certain level* L *cannot change under transformations of another level.*

If follows from Theorem 1 that from the viewpoint of the homogeneity criterion, all values of the homogeneity of various levels can be considered separately and the algorithm ensures the possibility to increase values of the homogeneity by performing limited manipulations at every instant.

Theorem 2. $S \le H \le (n-1)2^n$.

The number M of steps of the algorithm is equal to $M \le 2(n-1)2^n + n - 1$ or $M \le (n-1)2^{n+1} + n - 1$, so that in the worst case we have $M = O\left(N \log_N\right)$, where $N = 2^n$.

Example 4. We will consider in detail the operation of the presented algorithm on the basis of the example of the test circuit XOR5 from the collection LGSynth 93. The truth vector of the output function for this circuit is

$$f = 01101001100101101001011001101001.$$

The BDD for this circuit has 9 nodes. The initial value of the homogeneity is $H_f = 0$. We set the initial value of the matrix of the transformation of input variables, which is given as $T = I$, where I is the identity matrix of order 6. Let us consider the first stage of operation of the algorithm.

At the first three levels, we have the values of $H_1 = 0, H_2 = 0, H_3 = 0$, and the versions of the improvement of a value of H do not exist (any possible permutation of cofunctions does not change values of the homogeneity), for which reason the transformations in the binary tree do not occur.

At the fourth level, we also have $H_4 = 0$, but there is a version of the improvement, namely, the permutation of the elements 0 and 2 (the rearrangeable elements are shown in the form of heavy digits):

$$\underline{\mathbf{0}}1\underline{\mathbf{1}}0\underline{\mathbf{1}}0\underline{\mathbf{0}}1\underline{\mathbf{1}}0\underline{\mathbf{0}}1\underline{\mathbf{0}}1\underline{\mathbf{1}}0\underline{\mathbf{1}}0\underline{\mathbf{0}}1\underline{\mathbf{0}}1\underline{\mathbf{1}}0\underline{\mathbf{0}}1\underline{\mathbf{1}}0\underline{\mathbf{1}}0\underline{\mathbf{0}}1.$$

After the permutation, we obtain the truth vector

$$11000011001111000011110011000011$$

and the homogeneity value of $H_4 = 32$. To the permutation of the elements 0 and 2 at level 4 there correspond changes of the transformation matrix in the form

$$\begin{pmatrix} 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&1&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&0&1 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{pmatrix} = \begin{pmatrix} 1&0&0&0&0&0 \\ 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&1&1&1 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&1 \end{pmatrix}.$$

We return to the upper level $L = 3$. The homogeneity value is $H_3 = 0$, but there is a version of the improvement, namely, the permutation of the elements 0 and 2 at level 3 (the boldface elements):

$$\underline{\mathbf{11}}00\underline{\mathbf{00}}110011\underline{\mathbf{11}}00\underline{\mathbf{00}}11\underline{\mathbf{11}}00\underline{\mathbf{11}}00\underline{\mathbf{00}}11.$$

The truth vector after this permutation is equal to

$$00001111111100001111000000001111.$$

The homogeneity value at the third level becomes equal to $H_3 = 32$. The transformation matrix changes to the matrix

$$\begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&1&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&1\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&1&1\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} = \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&1&1&0\\ 0&0&0&1&1&1\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix}.$$

We return once again to the level $L = 2$. Here, we also have a version of the improvement, namely, the permutation of the elements 0 and 2 at level 2 (the boldface elements):

$$\underline{\mathbf{0000}\ }1111\underline{\mathbf{1111}\ }0000\underline{\mathbf{1111}}0000\underline{\mathbf{0000}\ }1111.$$

After the transformation, the truth vector is

$$11111111000000000000000011111111.$$

The homogeneity value is $H_2 = 32$ and the transformation matrix takes the form

$$\begin{pmatrix} 1&0&0&0&0&0\\ 0&1&1&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&1\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&1&1&0\\ 0&0&0&1&1&1\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} = \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&1&1&1&1\\ 0&0&1&1&1&0\\ 0&0&0&1&1&1\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix}.$$

We now return to the level $L = 1$. Here, the version of the improvement is the permutation of the elements 0 and 2 at level 1 (the boldface elements):

$$\underline{\mathbf{1111\ 1111}}00000000\underline{\mathbf{0000\ 0000}}11111111.$$

After the transformation, the truth vector is equal to

$$00000000000000011111111111111111.$$

The homogeneity level is $H_1 = 32$ and the transformation matrix is given as

$$\begin{pmatrix} 1&1&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&1\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} \times \begin{pmatrix} 1&0&0&0&0&0\\ 0&1&1&1&1&1\\ 0&0&1&1&1&0\\ 0&0&0&1&1&1\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&0\\ 0&1&1&1&1&1\\ 0&0&1&1&1&0\\ 0&0&0&1&1&1\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{pmatrix}.$$

We will perform one more passage forward.

$L = 1$, $H_1 = 32$, and there are no versions of the improvement;

$L = 2$, $H_2 = 32$, and there are no versions of the improvement;

$L = 3$, $H_3 = 32$, and there are no versions of the improvement;

$L = 4$, $H_4 = 32$, and there are no versions of the improvement;

$L = n = 5$, and the algorithm terminates,

$$H_f = \sum_{L=1}^{n-1} H_L = H_1 + H_2 + H_3 + H_4 = 128.$$

The transformed function is equal to $\widetilde{f} = x_1$, and so the BDD for this function has only one node. The matrix of the affine transformation of variables for deriving the initial function f from this function is given as

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

6. EXPERIMENTAL RESULTS

The results were obtained with the aid of our own program written on the Visual C on the computer-type PIII of 750 MHz. The algorithms were designed and worked out on the basis of all possible functions of the number of incoming variables up to four inclusive. This stems from the size of the set of all versions that are equal to 2^{2^n}. To test BDDs built up from 5 and more incoming variables, recourse was made to the generation of 200 random test sequences from the set $\{0, 1\}$ of dimension 2^n (i.e., random vectors relative to n variables), and the mean result was taken of the percentage reduction of the size of binary diagrams. The results are shown in Fig. 8. The results of operation of the algorithms described above are displayed in Fig. 8a. The partial search method with the use of merely adjacent variables (Algorithm 2) has the best indices of the percentage reduction of the size of a binary diagram, but Fig. 8b discloses that in their sequential

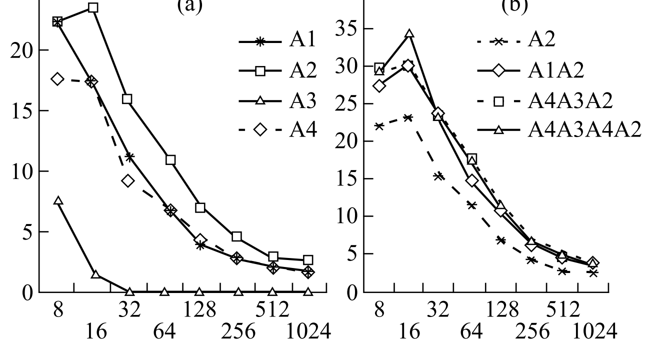

Fig. 8. Percentage reduction of the size of a BDD (the Y axis) depending on the number n of variables (the number 2^n laid off on the X axis defines the size of the truth table) with the use of the algorithms described above.

Results of testing of Algorithms A1 and A2 by the tests LGSynth 93. In the brackets are shown the results obtained with the use of the homogeneity method (A4, A2). The symbol $*$ indicates that the diagrams with a few outputs are considered as the aggregate of diagrams with one output. The symbol $**$ denotes that the diagrams with a few outputs subdivide isomorphic subgraphs (here, the methods act in the same way as in the case of $*$)

Type of circuit	Number of inputs	Number of outputs	Initial size*	Resultant size	Size with regard to isomorphism of outputs**		t, s
					initial	resultant	
Bw	5	28	253	170(158)	114	82(66)	1.60
C17	5	2	12	11	10	11	0.11
Rd53	5	3	29	15	23	15(14)	0.17
Xor5	5	1	9	1	9	1	0.07
9sym	9	1	33	38(24)	33	38(24)	0.59
Con1	7	2	18	15(16)	18	15(14)	0.18
Inc	7	9	115	96(100)	85	81(78)	0.72
Sqrt8	8	4	43	45(41)	42	44	0.61
Ex5p	8	63	793	672(639)	356	322(280)	9.80
Misex1	8	7	75	60(58)	47	41(45)	1.04
Apex4	9	19	1610	1563(1585)	1021	1034()	13.41
Clip	9	5	280	112(127)	254	97(87)	3.47
Sao2	10	4	182	103(108)	154	91(88)	11.11

joint use, the described algorithms complement one another, improving the result at the common high speed (the time it takes to process all 65 536 functions of four variables is equal to about 1 minute).

Unfortunately, with an increase in the number of variables, the results get worse in regard to a percentage decrease in the size of binary diagrams of a random function. This can readily be explained. The transformations are performed synchronously on the quadruples of elements. At small levels, each of the quadruples includes very many elements and they stochastically balance each other. At large levels, a similar problem takes place, but now on account of a large number of synchronously transformed quadruples. Therefore, there is a need for a more complex criterion compensating for these phenomena, which is evolved for the functions of a large number of variables. Apart from the tests on random variables, we carried out tests on the functions from LGSynth 93. The results for the jointly used Algorithms A1 and A2 are listed in table. The results of the joint use of Algorithms A4 and A2 are shown in brackets. It should be noted that the functions with a few outputs were processed as a few functions with one output. The results are presented in two types. First, the complexity of a function is defined as a sum of the complexities of all diagrams of the functions representing its outputs and then it is defined with due regard for the isomorphism (i.e., where the common portions of diagrams are reduced and have references to one another) in much the same way as the reduction of the diagrams of type I.

7. CONCLUSIONS

The new algorithms for minimizing BDDs are worked out on the basis of the affine transformations of variables. The use of the presented minimization criteria enables reducing the size of a BDD on account of the search for suitable transformations of input variables that involve a relatively small amount of computations.

APPENDIX

Proof of Lemma 1. The number of unsuccessful attempts cannot exceed the number of successful attempts including the number of levels in the tree. At the first level (the transformation of the first and the second variable), the cases of a success and a failure entail an identical action, namely, the descent, which is equivalent to a failure. But each case of the successful step at any other level is the ascent by one level. Considering that the tree has $n-1$ levels, we find that the number of unsuccessful attempts is equal to $S+n-1$, while the number of steps of the algorithm is equal to $S+S+n-1=2S+n-1$.

Proof of Theorem 1. If we consider the level that is larger than the current one, it will be seen that transformations at the former level will lie within the block of the current level (see Fig. 6) because the block is the pair of elements at the current level, which correspond to the quadruple at the next level. Therefore, the changes in the number of units with respect to the number of zeros in each individual block will not occur. If we examine the level that is less than the current one, it will be found that the transformations will be made with respect to elements each of which is a block at the requisite level. Thus, in none of these cases the value of the homogeneity of the current level will change, apart from the case in which transformations take place at the current level. This is what we set out to prove.

Proof of Theorem 2. The number of values of the function is equal to 2^n. The quantity H_L takes a maximum value in the case when only units or zeros lie within each of the blocks. The minimum value is equal to zero in the case when there is an equal number of zeros and units within each of the blocks. In other words,

$$0 \le H_L \le 2^n.$$

Under each successful transformation at the level L, the value increases:

$$H = \sum_{L=1}^{n-1} H_L.$$

We obtain

$$0 \le H \le (n-1)2^n.$$

The number S of successful transformations cannot be higher than the value of H. (The case is unreal if at each step this quantity will increase only by unity).

REFERENCES

1. Nechiporuk, E.I., On Synthesis of Circuits with the Aid of Linear Transformations of Variables, *Dokl. Akad. Nauk SSSR*, 1958, vol. 123, no. 4, pp. 610–612.
2. Rahardia, S. and Falkovski, B.J., Fast Linearly Independent Arithmetic Expansion, *IEEE Trans. Comput.*, 1999, vol. 48, no. 9, pp. 991–999.
3. Stankovic, R. and Astola, J., Some Remarks on Linear Transform of Variables in Adders, *Proc. 5 Reed–Muller Workshop*, Starkvill, USA, 2001, pp. 294–302.
4. Kolpakov, A. and Latypov, R., Minimization of BDDs Using Linear Transformation of Variables, *Proc. 5 Reed–Muller Workshop*, Starkvill, USA, 2001, pp. 57–65.
5. Karpovsky, M.G., *Finite Orthogonal Series in the Design of Digital Devices*, New York: Willey, 1976.

6. Malyugin, V.D., Implementation of Tuples of Boolean Functions by Means of Linear Arithmetic Polynomials, *Avtom. Telemekh.*, 1984, no. 2, pp. 114–121.

7. Dzyuzhan'ski, P., Malyugin, V., Shmerko, V., and Yanushkevich, S., Linear Models of Circuits Using Multivalued Elements, *Avtom. Telemekh.*, 2002, no. 9, pp. 99–119.

8. Shmerko, V., Malyugin's Theorems: A New Insight into Logic Control, Design of VLSICs, and Three-Dimensional Structures of Data for New Technologies, *Avtom. Telemekh.*, 2004, no. 6.

9. Detjens, E., FPGA Devices Require FPGA-Specific Synthesis Tools, *Computer Design*, 1990, vol. 124.

10. Akers, S.B., Binary Decision Diagrams, *IEEE Trans. Comput.*, 1978, vol. C-27, no. 6, pp. 509–516.

11. Bryant, R.E., Graph-Based Algorithms for Boolean Functions Manipulations, *IEEE Trans. Comput.*, 1986, vol. C-35, no. 8, pp. 667–691.

12. Drechsler, R., *Formal Verification of Circuits*, Boston: Kluwer Academic, 2000.

13. *Verification of Digital and Hibrid Systems*, Boston: Kluwer Academic, 2001.

14. Minato, S., *Binary Decision Diagrams and Applications in VLSI CAD*, Boston: Kluwer Academic, 1996.

15. Brown, S.D., Fracis, R.J., Rose, J., and Vranesic, Z.G., *Field-Programmable Gate Arrays*, Boston: Kluwer Academic, 1992.

16. Murgai, R., Brayton, R.K., and Sangiovanni-Vincentelli, A.L., *Logic Synthesis for Field-Programmable Gate Arrays*, Boston: Kluwer Academic, 1995.

17. Drechsler, R. and Gunter, W., Using Lower Bound During Dynamic BDD Minimization, *Proc. Design Automat. Conf.*, 1997, pp. 348–356.

18. Ruddel, R., Dynamic Variable Ordering for Ordered Binary Decision Diagrams, *Proc. Int. Conf. on CAD*, 1993, pp. 42–47.

19. Gunter, W. and Drechsler, R., Linear Transformations and Exact Minimization of BDDs, *Proc. IEEE Great Lakes Sympos., VLSI*, 1998, pp. 325–330.

20. Scholl, C., Melchior, S., Hotz, G., and Molitor, P., Minimizing ROBDD Sizes of Incompletely Specified Boolean Functions by Exploiting Strong Symmetries, *Proc. Eur. Design and Test Conf.*, Paris, 1997.

21. Thierauf, T., *The Computational Complexity of Equivalence and Isomorphism Problems*, New York: Springer, 2000.

This paper was recommended for publication by P.P. Parkhomenko, a member of the Editorial Board