View metadata, citation and similar papers at core.ac.uk
brought to you by CORE

FACTA UNIVERSITATIS
Series: **Electronics and Energetics** Vol. 31, N° 2, June 2018, pp. 169 - 187
https://doi.org/10.2298/FUEE1802169S

GENETIC ALGORITHM FOR BINARY AND FUNCTIONAL DECISION DIAGRAMS OPTIMIZATION*

Suzana Stojković[1], Darko Veličković[1], Claudio Moraga[2]

[1]Faculty of Electronic Engineering, University of Niš, Niš, Serbia
[2]Department of Computer Science, TU Dortmund, Germany University, Dortmund, Germany

Abstract. *Decision diagrams (DD) are a widely used data structure for discrete functions representation. The major problem in DD-based applicationsis the DD size minimization (reduction of the number of nodes), because their size is dependent on the variables order. Genetic algorithms are often used in different optimization problems including the DD size optimization. In this paper, we apply the genetic algorithm to minimize the size of both Binary Decision Diagrams (BDDs) and Functional Decision Diagrams (FDDs). In both cases, in the proposed algorithm, a Bottom-Up Partially Matched Crossover (BU-PMX) is used as the crossover operator. In the case of BDDs, mutation is done in the standard way by variables exchanging. In the case of FDDs, the mutation by changing the polarity of variables is additionally used. Experimental results of optimization of the BDDs and FDDs of the set of benchmark functions are also presented.*

Key words: *Binary Decision Diagrams, Functional Decision Diagrams, Decision Diagrams oprimization, Genetic algorithm.*

1 Introduction

Decision diagrams (DDs) are a compact data structures for discrete functions representation. Bryant showed their canonicity in 1986 in [1] and after that they have been applied in many areas in which discrete functions are used: hardware design, hardware testing, signal processing, etc. Complexities of the designed hardware or of the computations that are done by decision diagrams are directly proportional to the size of the diagram. A main disadvantage of the decision diagrams is that their size is dependent on the order of the variables that are used in the diagram.

Received October 21, 2017; received in revised form January 30, 2018
Corresponding author: Suzana Stojkovic
Faculty of Electronic Engineering, University of Niš, Medevedeva 14, 18000 Niš, Serbia
(E-mail: suzana.stojkovic@elfak.ni.ac.rs)
*An earlier version of this paper was presented as an invited address at the Reed-Muller 2017 Workshop, Novi Sad, Serbia, May 24-25, 2017

Optimization of the DD size is a very often solved problem. Algorithms for DD optimization can be classified into two categories: exact algorithms and heuristic algorithms. The basic exact algorithm is a brute-force algorithm creating the diagrams for all possible orders of variables and choosing the best. A slightly improved exact algorithm is presented in [2]. But, all exact algorithms are very slow and inapplicable for functions with a large number of variables.

The most widely used heuristic algorithm for DD optimization is Rudells sifting algorithm that was proposed in [3]. The main idea in that algorithm is the sifting of each variable through all levels in the diagram and choose the optimal position.

A genetic algorithm is a heuristic algorithm that can be applied in solving different optimization problems. Using genetic algorithm in DD optimization was first discussed in [4]. After that, genetic algorithms for DD optimization were improved in many papers ([5–13]). Some of them optimize the DD size ([4–11]). In [12] the 1-paths number is optimized, and in [13] a method for these two optimization is proposed.

In this paper, we present a genetic algorithm for optimization of BDDs and FDDs. Our main goal was minimization of FDD size because we use FDD in reversible synthesis (see for example [14], [15]). For comparison we also include results on the minimization of the size of the BDDs. In the applications for FDD-based reversible synthesis, the complexity of the generated network is directly dependent of the FDD size. Additional problem in FDD usage is that the size is dependent of the decomposition rules that are used in the nodes. In FDD in each a node positive or negative Davio decomposition can be used. Usually, the same decomposition is used in all nodes from the same level. It follows that for one variable order, 2^n different FDDs can be created. An exact algorithm in that case should check $2^n \cdot n!$ cases, which is impossible for large number of variables. Another group of minimization algorithms, sifting algorithms, analyze only variables order. Because of that we choose a side to try by applying the genetic algorithm. In the presented algorithm, for both BDD and FDD, a modified PMX crossover operator (BU-PMX Bottom-Up Partially Matched Crossover) and mutation by variable exchange are used. In the case of FDD optimization, mutation by polarity change is additionally used.

The paper is organized in the following way: Section 2 contains most important definitions related to the decision diagrams. Section 3 presents the general idea of genetic algorithms and their specifications in the case of applying in DD optimization. Section 4 describes the algorithm for BDD and

FDD optimization and the genetic operations that are used in it. Section 5 discuses experimental results and in Section 6 some concluding remarks are given.

2 Decision diagrams

Definition 1 (Binary Decision Tree) *A Binary Decision Tree (BDT) representing a Boolean function f is the binary tree created by the recursive application of the Shannon decomposition rule:*

$$f = \overline{x_k} \cdot f(x_k = 0) \oplus x_k \cdot f(x_k = 1) \tag{1}$$

Definition 2 (Terminal and Nonterminal nodes) *A BDT contains two types of nodes: nonterminal and terminal. A nonterminal node represents one decomposition and it has a joint decision variable. A terminal node contains the value of the function.*

Definition 3 (Level in BDT) *A level in the BDT is a set of nonterminal nodes with the same decision variable, or the set of terminal nodes.*

Definition 4 (Functional Decision Tree) *A Functional Decision Tree (FDT) representing a Boolean function f is the binary tree created by the recursive application of the positive (2) or negative (3) Davio decomposition rule:*

$$f = f(x_k = 0) \oplus x_k \cdot (f(x_k = 0) \oplus f(x_k = 1)) \tag{2}$$

$$f = \overline{x_k} \cdot (f(x_k = 0) \oplus f(x_k = 1)) \oplus f(x_k = 1) \tag{3}$$

Definition 5 (Fixed Polarity Functional Decision Tree) *A Functional Decision Tree in which the same decomposition is used in each node from the same level is called a Fixed Polarity Functional Decision Tree (FPFDT).*

Definition 6 (Polarity-vector) *The polarity-vector of the FPFDT is a bit vector which defines the types of the decompositions that are used in the levels. 0 denotes that the positive Davio decomposition is used, 1 denotes the negative.*

Definition 7 (Positive-polarity FDT) *A FDT in which positive Davio decomposition is used at all levels is a positive-polarity FDT.*

Definition 8 (Binary Decision Diagram) *A BDT is transformed into a Binary Decision Diagram (BDD) by using the following reduction rules:*

1. *Share the isomorphic sub-trees: if there are two terminal nodes with the same value, or two non-terminal nodes with isomorphic sub-trees, one of them is deleted. Its incoming edges are directed to the remaining node.*
2. *Eliminate the redundant nodes: if both outgoing edges from a non-terminal node point to the same sub-tree, this node is redundant and it is deleted. Its incoming edges are directed to the common sub-tree.*

Definition 9 (Functional Decision Diagram) *An FDT is transformed into an FDD by using the reduction rule 1 above and the following 0-suppress reduction rules:*

2.1 If the right outgoing edge from a positive Davio node points to the 0, the node is deleted. The edges pointing to the deleted node are directed to its left sub-tree.

2.2 If the left outgoing edge from a negative Davio node points to the 0, the node is deleted. The edges pointing to the deleted node are directed to its right sub-tree.

Example 1 *Figure 2 shows the BDD (a) and the positive-polarity FDD (b) of the function* $f(x_1, x_2, x_3, x_4) = \overline{x_1} \cdot \overline{x_2} + x_1 \cdot x_2 + x_3 + \overline{x_4}$.

Definition 10 (DD size) *DD size is equal to the number of the nonterminal nodes.*

Example 2 *Figure 1 shows the BDDs of the function* $f(x_1, \ldots, x_6) = x_1x_2 + x_3x_4 + x_5x_6$ *for variables orders (a)* $(x_1, x_2, x_3, x_4, x_5, x_6)$ *and (b)* $(x_1, x_4, x_2, x_5, x_3, x_6)$. *The size of the first BDD is 6, but the size of the second is 14.*

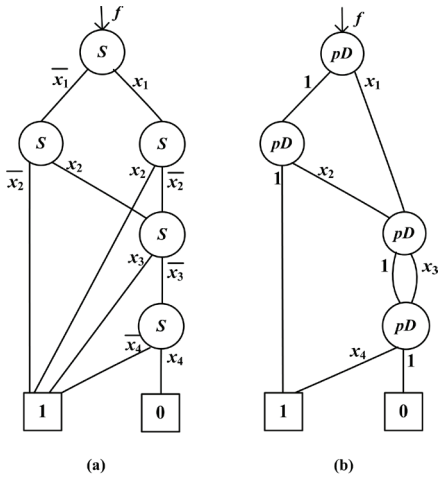

Fig. 1: BDD (a) and FDD (b) of the function from Example 1.

In the general case, for the function of $2n$ *variables* $f(x_1, x_2, \ldots x_{2n-1}, x_{2n}) = x_1x_2 + \cdots + x_{2n-1}x_{2n}$*, the size of the BDD with variables order* $(x_1, x_2, \ldots, x_{2n-1}, x_{2n})$ *is* $2n$*, and with variables order* $(x_1, x_{n+1}, \ldots, x_n, x_{2n})$ *it is* $O(2^{n-1})$*.*

Besides the variable order, the size of fixed polarity FDDs is dependent also on the polarities for the variables.

Example 3 *Figure 3 shows the FDDs of the function in Example 1 for polarity vectors (a)* $\mathbf{F} = [\mathbf{1\ 1\ 1\ 1}]^{\mathbf{T}}$ *and (b)* $\mathbf{F} = [\mathbf{0\ 1\ 0\ 1}]^{\mathbf{T}}$*. The size of the diagram if the first case is 4, and in the second case is 6.*

3 Genetic algorithm

A genetic algorithm is a method for solving different optimization problems based on an analogy to the natural selection process. In this algorithm, the solution of a problem is presented as an array that is named chromosome. An element of the chromosome is a gene.

In general, the initial set of chromosomes are generated randomly, and then, the new generation is created by using two genetic operations: crossover and mutation. The crossover operator defines the way for creating the child chromosomes by combination of the genes from parent chromosomes. In practice, one point crossover (fig. 4(a)) and two-point crossover (fig. 4(b))

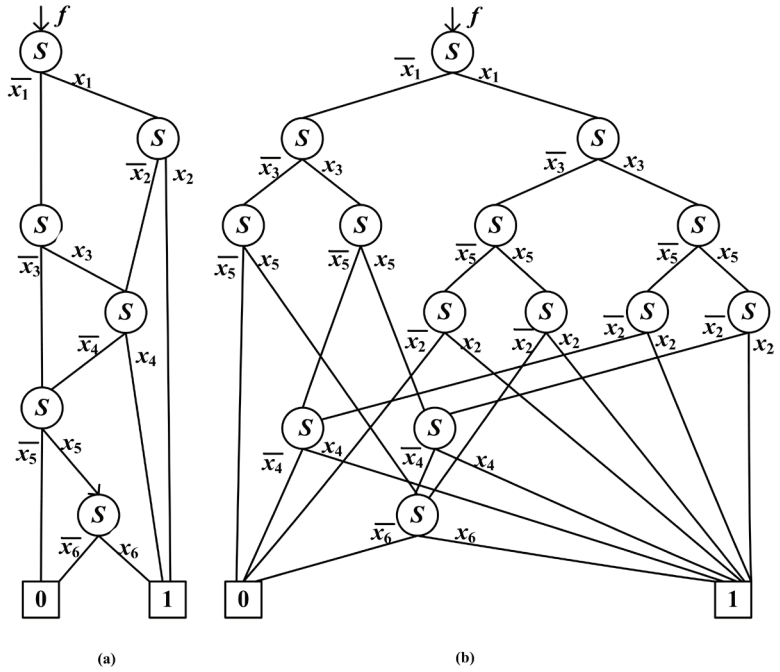

Fig. 2: BDDs of the function from Example 2 for two different variable orders.

are usually chosen. The mutation is often realized by changing the value of the gene at a selected position.

The measure of the quality of a solution (chromosome) is named fitness score. Fitness scores are used to compute the possibilities for selecting the chromosomes for parents for the next generation, and for selecting the chromosomes that will die after an iteration.

To define the genetic algorithm for a concrete optimization problem means to define: the type of genes, the fitness function and the genetic operations.

3.1 Genetic algorithm for BDD size optimization

One chromosome in a BDD optimization problem is one order of input variables, i.e. one permutation of the integer numbers from interval $[1, n]$. It follows that standard genetic operators cannot be used. Because of that, for a BDD optimization, special genetic operators are defined. Crossover oper-

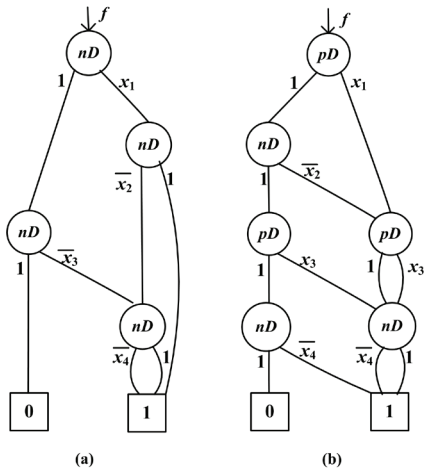

Fig. 3: FDDs of the function from Example 1 for two different polarity vectors.

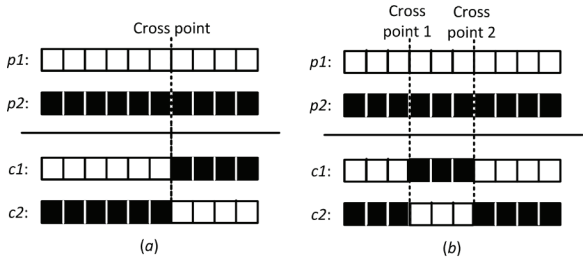

Fig. 4: One-point (a) and two-point (b) crossover operators.

ators that will be discussed in this section are: Order Crossover ([10], [11]), Cyclic Crossover ([10], [11]), Partially Matched Crossover ([4], [10], [11]) and Alternating Crossover ([7]).

Algorithm 1 (Cyclic Crossover Operator - CX) *:*

Step 1. Create a cycle of the genes defined by corresponding positions in the parent chromosomes starting from first unused gene in the first parent.

Step 2. Copy the genes from the cycle from one parent in the first child and from other parent in the second child.

Step 3. Repeat steps 1 and 2 by alternating change the target child in which the genes from one parent is copied.

Example 4 *Let we see the following parents:*

$$p_1 = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]$$

$$p_2 = [5\ 4\ 6\ 9\ 2\ 8\ 3\ 7\ 1\ 10]$$

The first cycle of the genes is created starting from the gene 1 from the first parent. On the corresponding position in the second parent is the gene 5. Then, we find the gene 5 in the first parent and in the corresponding position in the second parent is the gene 2. Process is continued until the cycle is closed. The created cycle is $1 \to 5 \to 2 \to 4 \to 9 \to 1$. *The child chromosomes after putting first cycle are:*

$$c_1 = [1\ 2\ \_\ 4\ 5\ \_\ \_\ \_\ 9\ \_]$$

$$c_2 = [5\ 4\ \_\ 9\ 2\ \_\ \_\ \_\ 1\ \_]$$

Second cycle is created starting from the gene 3: $3 \to 6 \to 8 \to 7 \to 3$

Child chromosomes after putting second cycle in the child chromosomes are:

$$c_1 = [1\ 2\ 6\ 4\ 5\ 8\ 3\ 7\ 9\ \_]$$

$$c_2 = [5\ 4\ 3\ 9\ 2\ 6\ 7\ 8\ 1\ \_]$$

The last cycle contains only gene 10, and, finally, child chromosomes are:

$$c_1 = [1\ 2\ 6\ 4\ 5\ 8\ 3\ 7\ 9\ 10]$$

$$c_2 = [5\ 4\ 3\ 9\ 2\ 6\ 7\ 8\ 1\ 10]$$

Algorithm 2 (Order Crossover Operator - OX) *:*

Step 1. Randomly select two crossover points.

Step 2. Copy in the child chromosome the genes from the first parent between crossover points.

Step 3. Delete from second parent the genes which are already in the child.

Step 4. Place the genes from the second parent into unfilled positions in child chromosome from left to right.

Example 5 *Let the parent variable orders be given by arrays:*

$$p_1 = [1\ 2\ 3\ |4\ 5\ 6\ |7\ 8\ 9\ 10]$$

$$p_2 = [6\ 7\ 4\ 2\ 3\ 10\ 9\ 5\ 1\ 8]$$

The crossover points are marked in the first parent. After step one, the child chromosome is:

$$c = [\_\ \_\ \_\ |4\ 5\ 6\ |\_\ \_\ \_\ \_]$$

After deleting corresponding genes, second parent is:

$$p_2 = [\not{6}\ 7\ \not{4}\ 2\ 3\ 10\ 9\ \not{5}\ 1\ 8]$$

Finally, after putting the genes from second parent, the generated child is:

$$c = [7\ 2\ 3\ |4\ 5\ 6\ |10\ 9\ 1\ 8]$$

Algorithm 3 (Partially Matched Crossover Operator - PMX) *:*

Step 1. Perform a two-point crossover.

Step 2. Create the mapping table of the genes from the central part of one parent that do not appear in the central part of the second parent. The mapping pair of a gene from position i of the first parent ($p_1[i]$) is the gene at the same position in the other parent ($p_2[i]$) if the gene $p_2[i]$ is not in the central part of the first parent, otherwise, if the $p_2[i] = p_1[j]$ the mapping pair of $p_1[i]$ is equal to the mapping pair of the gene $p_1[j]$.

Step 3. Eliminate duplicated genes in child chromosomes so that the central part of the chromosomes remains unchanged. If some gene from the central part appears again in other parts, replace it by the corresponding mapping pair.

Example 6 *Let the parent variable orders be given by arrays:*

$$p_1 = [9\ 8\ 4\ |5\ 2\ 7\ |1\ 3\ 6\ 10]$$

$$p_2 = [8\ 7\ 1\ |2\ 3\ 10\ |9\ 5\ 4\ 6]$$

Let the two-point crossover operator be performed with the crossover points 3 and 6. The resulting child chromosomes are:

$$c_1' = [9\ 8\ 4\ |2\ 3\ 10\ |1\ 3\ 6\ 10]$$

$$c_2' = [8\ 7\ 1\ |5\ 2\ 7\ |9\ 5\ 4\ 6]$$

Let us create a mapping table:
$p_2[4] = 2$ *exists in the central part of* p_1*, and it is not mapped.*
$p_2[5] = 3 \to 2 \to 5$. *Pair* $(3, 5)$ *is added into the mapping table.*
$p_2[6] = 10 \to 7$. *Pair* $(10, 7)$ *is added into the mapping table.*
Resulting child chromosomes after duplicate elimination are:

$$c_1 = [9\ 8\ 4\ |2\ 3\ 10\ |1\ 5\ 6\ 7]$$

$$c_2 = [8\ 10\ 1\ |5\ 2\ 7\ |9\ 3\ 4\ 6]$$

Algorithm 4 (Alternating Crossover Operator - AX) *Create the child chromosome by taking alternatively the genes from the first and the second parent. Before storing the gene into a child chromosome check whether it already exists there.*

Example 7 *Let the alternating crossover be performed over the same parents as in the previous example. The resulting child chromosome is:*

$$c = [9\ 8\ 7\ 4\ 1\ 5\ 2\ 3\ 10\ 6]$$

Mutation cannot be realized as it is shown in the previous section, too. In the literature, three ways for the mutation operation are suggested:

Algorithm 5 (Mutation by one variables exchange) *Randomly select two positions in a chromosome and exchange the variables from the selected positions.*

Algorithm 6 (Mutation by two variables exchanges) *Apply two-times mutation defined in the Algorithm 5.*

Algorithm 7 (Mutation by neighbor exchange) *Randomly select one position* i*. Exchange the variables from positions* i *and* $i + 1$*.*

The fitness function in a BDD optimization problem is the size of the BDD.

4 Genetic algorithm for BDD and FDD size optimization

In the original PMX algorithm, the central part of the chromosome is transferred into the child chromosome unchanged. But, the possibility of deleting a DD node in the reduction phase is greater if the node is at the bottom levels. It follows that good properties of the parents will be inherited if the order of the variables on the last levels is not changed. Because of that,we used a modified PMX algorithm in which the right part of the genes from parent chromosomes are directly transferred to the child chromosomes. This operator is named as the *Bottom-up PMX*, because the genes are written into the child chromosome from the right to the left, i.e. from the bottom levels up. The second reason why the part of the unchanged genes is shifted to the end of the chromosome is that in that case the DD corresponding to the child chromosome contains an identical set of nodes in the last levels as the DD corresponding to the parent chromosome and calculation time of the fitness function is shortened.

Algorithm 8 (Bottom-up PMX Operator - BU-PMX) *:*

Step 1. Perform an one-point crossover.

Step 2. Create the PMX mapping table for the right part of chromosomes.

Step 3. Eliminate duplicate genes from the left part of child chromosomes by using the PMX mapping table.

Example 8 *Let the Bottom-up PMX operator be performed over parents:*

$$p_1 = [1\ 2\ 3\ 4\ 5\ 6\ |7\ 8\ 9\ 10]$$

$$p_2 = [7\ 4\ 1\ 2\ 5\ 6\ |9\ 3\ 8\ 10]$$

After performing one-point crossover the generated children are:

$$c_1' = [7\ 4\ 1\ 2\ 5\ 6\ |7\ 8\ 9\ 10]$$

$$c_2' = [1\ 2\ 3\ 4\ 5\ 6\ |9\ 3\ 8\ 10]$$

The mapping table contains only the pair (7, 3). After duplicates elimination, the resulting child chromosomes are:

$$c_1 = [3\ 4\ 1\ 2\ 5\ 6\ |7\ 8\ 9\ 10]$$

$$c_2 = [1\ 2\ 7\ 4\ 5\ 6\ |9\ 3\ 8\ 10]$$

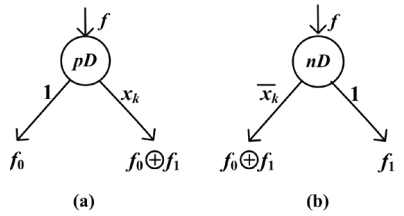

Fig. 5: Positive (a) and negative (b) Davio nodes.

As it was shown in Example 3, the FDD size is dependent on the polarity vector. Because of that, in FDD optimization an additional mutation producing a polarity change is used. To specify the transformation that is done on the FDD when this mutation is performed, the positive Davio and negative Davio nodes are shown in Figure 5 ((a) and (b), respectively). In this figure $f_0 = f(x_k = 0)$ and $f_1 = f(x_k = 1)$. Let f_l and f_r be the left and right successors of the node. If the polarity is changed from positive to negative, the transformation that is done is:

$$\begin{aligned} f_{l\_new} &= f_{r\_old} \\ f_{r\_new} &= f_{l\_old} \oplus f_{r\_old} \end{aligned} \tag{4}$$

If the reverse polarity change is done, the applied transformation is:

$$\begin{aligned} f_{r\_new} &= f_{l\_old} \\ f_{l\_new} &= f_{l\_old} \oplus f_{r\_old} \end{aligned} \tag{5}$$

Algorithm 9 (Mutation by polarity change) *Randomly select a variable. Change the expansion rule in all nodes at the level corresponding to the selected variable.*

The complete genetic algorithm that is used for BDD and FDD optimization is shown in the Algorithm 10.

Algorithm 10 (Genetic algorithm for DD optimization) *:*

Step 1. Create initial population of chromosomes and compute the fitness score for each of them.

Step 2. Select pairs of parents for reproduction.

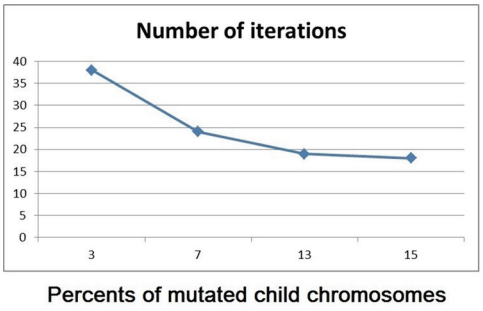

Fig. 6: Number of iterations needed to reach the minimum BDD size for the *bw* benchmark function as a function of a percents of the mutated child chromosomes.

Step 3. Create child chromosomes by BU-PMX.

Step 4. Mutate child chromosomes (by mutation probability).

Step 5. Do Darwins process - remove from population the worst chromosome or more bad chromosomes if the population is full.

Step 6. Repeat steps 2-5 until the goal is reached or the computing time is exhausted.

5 Experimental results

5.1 Results of BDD size optimization

At first, we tested how the mutation probability influences the convergence of the algorithm. Figure 6 shows the number of iterations that is needed to reach the minimum BDD size for the function *bw* for different percents of mutated chromosomes. Each experiment was repeated 100 times and in the figure the average values are shown. The number of needed iterations decreases when the percents of the mutated chromosomes increases. For percents greater than 15 the decreasing is very slow and 0.15 is chosen as an optimal mutation probability.

Then, we tested the convergence of the proposed algorithm on the set of a small benchmark functions for which we know the optimal size. We tested the number of iterations that is needed to reach the minimum BDD size. We compared these results with results obtained by using the Order Corossover (OX), Cyclic Crossover (CX), original PMX and Alternating Crossover (AX)

Table 1: Number of iterations needed to reach minimum BDD size by using different crossover operators

Function	In/Out	OX	CX	PMX	AX	BU-PMX
bw	5/25	5.5	7.4	5.9	7.9	5.2
5xp1	7/10	56.1	46.6	47.5	72.8	32.1
con1	7/2	97.9	76.8	87.2	86.7	71
misex1	8/7	152.5	93	67.7	84.2	107.8
sqrt8	8/4	135.4	159.3	116.8	371.1	113.7
clip	9/5	72.9	75	59.4	194.4	49.7

operators. The experiments were repeated 10 times and in Table 1 the average values are shown. Table 1 shows that for 5 out of 6 functions the smallest BDD size was obtained with the smallest number of iterations when the BU-PMX operator is used. In these 5 cases, alternating crossover was the worst. Only for the function $misex1$ the minimal BDD size was obtained with less number of iterations when PMX operator is used.

Finally, we optimized the BDD size for benchmark functions of a larger number of variables. Table 2 compares the sizes of the BDDs with initial order of variables and with optimal order generated by the genetic algorithm. In each experiment, the initial population contains $2n$ chromosomes (permutations) and maximum population size is $10n$, where n is the number of input variables. The table shows that the proposed algorithms reduced the size of the BDD, on the average, by 46.375%.

5.2 Comparison BDD optimization by proposed genetic algorithm and by other heuristic algorithms

The paper [11] compares the sizes of BDDs optimized by different heuristic algorithms and with genetic algorithm with 3 types of crossover operators (OX, CX and PMX). The paper shows that results that were produced by genetic algorithms are better than results of the other heuristic algorithms. Table 3 compares the sizes of DDs generated by the genetic algorithm presented in the paper [11] and by the genetic algorithm that is proposed in this paper. Table 3 shows that, for the functions with small number of variables, all algorithms found absolute minimum. For the functions with large number of variables algorithms that used PMX or BU-PMX operator produced better results. The algorithm that is proposed in this paper produced the smallest BDD for 13 out of 15 functions.

Table 2: BDD size for initial variable order and for optimal order generated by the proposed genetic algorithm

Function	In/Out	INIT	OPTIMAL	Iterations	Red. ratio [%]
alu4	14/8	1352	701	300	48
cu	14/11	65	37	300	43
misex3	14/14	1301	544	300	58
misex3c	14/14	810	443	300	45
table3	14/14	941	752	300	20
b12	15/9	91	60	300	34
table5	17/15	873	683	300	22
cc	21/20	105	49	400	53
dike2	22/29	976	373	400	62
i1	25/16	58	43	500	26
misex2	25/18	140	86	500	39
vg2	25/8	1059	84	500	92
frg1	28/3	203	89	600	56
c8	28/18	145	93	600	36
in4	32/20	1109	410	600	63
unreg	36/16	146	81	600	45
Average					46.375

5.3 Results of FDD size optimization

As was shown above, the FDD size is dependent on the variable order and the polarity. To determine the mutation that should be used in FDD optimization, a genetic algorithm with different mutation operators is performed on the set of function of a small number of variables (less than 10). Table 4 shows sizes of FDDs when:

- the initial order of variables and positive-polarity is used (INIT),
- the genetic algorithm with mutation by variables exchange is used (GA,VE),
- the genetic algorithm with mutation by polarity change is used (GA, PC), and
- the genetic algorithm with both mutation operators (with probabilities 0.5) are used (GA,VE+PC).

Table 3: Comparision of sizes of BDDs produced by the proposed algorithm and by the existing genetic algorithm with OX, CX and PMX crossover operators

Function	In/Out	OX	CX	PMX	BU-PMX
squar5	5/8	**37**	**37**	**37**	**37**
bw	5/28	**106**	**106**	**106**	**106**
5xp1	7/10	**68**	69	**68**	**68**
con1	7/2	16	**15**	**15**	**15**
inc	7/9	**72**	**72**	**72**	**72**
misex1	8/7	**36**	**36**	**36**	**36**
sqrt8	8/4	**33**	**33**	**33**	**33**
clip	9/5	102	109	**93**	**93**
sao2	10/4	92	90	**85**	**85**
alu4	14/8	891	939	734	**701**
b12	15/9	70	68	**50**	60
t481	16/1	85	78	**30**	38
duke2	22/9	506	512	390	**373**
misex2	25/18	100	102	87	**86**
vg2	25/8	339	301	148	**84**

It can be seen from the table that the FDDs with minimal sizes are generated when both mutations are used in the genetic algorithm. Because of that, in the experiments for optimization of FDDs of the functions of a larger number of variables (greater than 10), the approach with both mutation operators is used. Results of these experiments are shown in Table 5. As it can be seen from this table, FDDs are reduced by the proposed genetic algorithm, on the average, by 48.875%.

These experiments are done with the functions up to 25 variables. It is applicable on the functions with large number of variables, because the number of cases that are checked in the algorithm is determined by three parameters:

- number of crossover operations that is done in one iteration (CX),
- possibility of applying of mutation operator (p_m), and
- maximal number of iterations (IT).

Total number of created DDs is:

Table 4: Initial FDDs sizes and sizes of FDDs generated by genetic algorithms with different mutation operators

Function	In/Out	(INIT)	(GA,VE)	(GA,PC)	(GA, VE+PC)
add2	4/3	8	7	7	7
squar5	5/8	32	30	29	29
bw	5/28	144	97	93	93
inc	7/9	121	79	78	73
f51m	8/8	40	34	27	20
sqrt8	8/4	48	25	26	24

Table 5: FDD size for initial variable order and positive-polarity and for order and polarity generated by the proposed genetic algorithm

Function	In/Out	INIT	OPTIMAL	Iterations	Red. ratio [%]
alu4	14/8	840	541	300	36
cu	14/11	74	37	300	50
misex3	14/14	1024	764	300	25
misex3c	14/14	759	635	300	16
b12	15/9	116	62	300	47
cc	21/20	78	40	400	49
misex2	25/18	149	37	500	75
vg2	25/8	942	68	500	93
Average					48.875

$$N = CX \cdot (1 + p_m) \cdot IT$$

If we need smaller DD, the number of CX and IT should be greater. If the execution time is critical, CX and IT should be smaller. In our experiments:

$$CX = 2 \cdot n,$$

$$p_m = 0.15,$$

$$IT = [n/5] \cdot 100.$$

$$N = 2 \cdot n \cdot 1.15 \cdot [n/5] \cdot 100 \approx 46 \cdot n^2.$$

It is much smaller than when the brut-force exact algorithm in which $N = 2^n \cdot n!$.

6 Conclusion

In this paper, a genetic algorithm for BDD and FDD optimization is presented. In the algorithm a modification of the PMX operator is proposed: in the initial phase, instead of two-point crossover, one-point crossover is used. It follows that in the generated DD based on child permutation, part of the DD in the last levels is equal to the corresponding part of DD generated by the parent chromosome. In this way, the child chromosome inherits good properties of parent chromosome. In the case of FDD optimization, the proposed algorithm introduced mutation of polarity. Experiments show that when this mutation is used in combination with variable exchange, the genetic algorithm gives the best results. In the presented algorithm, sifting is not used as an additional method to improve the generated diagrams. Our goal was to show the performances of the genetic algorithm. In a real application of the algorithm, sifting can be included, too.

References

[1] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[2] S. J. Friedman and K. J. Supowit, "Finding the optimal variable ordering methods for binary decision diagrams," *IEEE Transactions on Computers*, vol. 39, no. 5, pp. 710–713, 1990.

[3] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proceedings of International Conference on CAD*, 1993, pp. 42–47.

[4] R. Drechsler, B. Becker, and N. Göckel, "A genetic algorithm for variable ordering of ob-dds," in *IEE Proceedings Computers and Digital Techniques*, vol. 143, no. 6, 1996, p. 363368.

[5] R. Drechsler and N. Göckel, "Minimization of bdds by evolutionary algorithms," in *International Workshop on Logic Synthesis (IWLS)*, 1997.

[6] R. Drechsler, B. Becker, and N. Göckel, "Learning heuristics for obdd minimization by evolutionary algorithms," in *Proceedings Parallel Problem Solving from Nature (PPSN), Lecture Notes in Computer Science*, vol. 1141, 1996, pp. 730–739.

[7] W. Lenders and C. Baier, "Genetic algorithms for the variable ordering problem of binary decision diagrams," *Lecture Notes in Computer Science*, vol. 3469, pp. 1–20, 2005.

[8] I. Furdu and B. Patrut, "Genetic algorithm for ordered decision diagrams optimization," in *Proceedings of ICMI 45*, 2006, pp. 437–444.

[9] I. Furdu and T. Socaciu, "Genetic algorithm for variable ordering of ordered binary decision diagrams," in *Proceedings of CNMI*, 2007, pp. 67–78.

[10] R. Kaur and M. Bansal, "Bdd ordering and minimization using variouscrossover operators in genetic algorithm," *Inernational Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering*, vol. 2, no. 3, pp. 1247–1250, 2014.

[11] S. Jindal and M. Bansal, "A novel and efficient variable ordering and minimization algorithm based on evolutionary computation," *Indian Journal of Science and Technology*, vol. 8, no. 48, pp. 1–10, 2016.

[12] M. Hilgemeier, N. Drechsler, and R. Drechsler, "Minimizing the number of one-paths in bdds by an evolutionary algorithm," 2003.

[13] S. Shirinzadeh, M. Soeken, and R. Drechsler, "Multi-objective bdd optimization with evolutionary algorithms," 2015, pp. 751–758.

[14] S. Stojković, M. Stanković, and C. Moraga, "Complexity reducton of toffoli networks besed on fdd," *Facta Universitatis, Ser. Electronics and Energetics*, vol. 28, no. 2, pp. 251–262, 2015.

[15] S. Stojković, M. Stanković, C. Moraga, and R. Stanković, "Procedure for fdd-based reversible synthesis by levels," 2016, pp. 1–6.