The Residual Variable in Decision Diagrams

Jan Lucansky, Peter Pistek, Marian Maruniak

Institute of Computer Engineering and Applied Informatics, Slovak University of Technology in Bratislava, Ilkovičova 2, 842 16 Bratislava 4, Slovak Republic, jan.lucansky@stuba.sk, peter.pistek@stuba.sk, maruniak09@student.fiit.stuba.sk

Abstract: We propose a novel method for binary-based decision diagrams (DD) which uses a residual variable. A new type of DD – Residual Variable in decision diagram (RViDD) allows to work without the use of the lowest and the most numerous level of nodes but at the same time preserves all the fundamental characteristics of DD. Thanks to these properties, it allows the use of all existing algorithms for optimization with only slight or no modification. In this paper we present the required characteristics for providing compatibility between DD using different decompositions and RViDD as well as the exchange of residual variable without the necessity of constructing a new RViDD. Proposed method was experimentally validated on benchmark circuits with various types of experiments. We use exhaustive (all input variable combination) comparative between reduced and ordered DD and RViDD with the average improvement up to 17.55%. Since optimization of DD is a NP-complete problem, we also include the usage of evolutionary algorithm for RViDD in comparison to more effective algorithms.

Keywords: BDD; KFDD; decision diagrams; residual variable

1 Introduction

The idea to use Decision Diagrams (DDs) in computer science is a longstanding one. The widespread use of DDs started with [1] when a set of algorithms for constructing and operating on DDs as data structures was introduced. Since then, DDs have come a long way and numerous applications have been found with the byproduct of numerous forms of DDs [2]. Some of these applications, nowadays, include, but are not limited to, formal verification, logic synthesis, test generation, classification techniques or network security. One of the areas that has greatly benefited from the use of DDs is circuit design. With conventional technologies slowly approaching their physical limits and with continuous emphasis on high speed and low power requirements, the need for optimizing circuits at design level becomes more and more important and DDs are a natural way to achieve that. A good example can be found in [3] where Binary DDs (BDD) have been used to optimize logical circuits based on multiplexers (multiplexer trees) or in [4], where BDDs are used for synthesis of optical circuits. Another example of their ongoing

importance can be found in [5], where DDs are used for the design of reversible and quantum circuits. The problem of power consumption increased in importance in the last years and is usually highlighted on system level. Today, as IoT devices use many sensors, it is essential to lower energy consumption as much as possible at any level of abstraction [6]. It is worth mentioning that DDs are not the only alternative and several types of graphs can be used as SAT solvers [7].

An extensive research regarding the use of various types of DDs has already been done. Among the most popular ones are BDDs, which are based on Shannon decomposition of Boolean function. As mentioned in [8], only 2 types of decomposition have an impact on DD area reduction – Shannon and Reed-Muller (sometimes called Davio). DDs that make use of Davio decomposition, either its positive or negative form, are called Functional DDs (FDD) and a combination of both types of decomposition results in Kronecker Functional DD (KFDD). Applying reduction techniques and respecting a certain order of variables for input functions results in Reduced Ordered DD, which based on the decomposition used can result in either ROBDD, ROFDD or ROKFDD. Keeping track of these abbreviations becomes hard to remember and sometimes counterintuitive (accurately called "alphabet soup" in [2]). The need for various types of DD prevails as each type and mainly the decomposition used has its own advantages with regards to the input function. Experimental results suggest that BDDs are more suitable for reduction of control functions and KFDDs perform better with symmetric data functions. Moreover, there are many types of logic gates represented by DD, e.g. BDD nodes can be directly transformed into 2:1 multiplexers and used to construct a multiplexer tree. Several advanced techniques with various different gates can also be found in [9] and [10]. For the purpose of this paper, suppose the Reduced and Ordered notification to be implicit.

Scaling remains a major problem for all types of DD. The ordering complexity for an input function with n variables is factorial, the input Boolean function contains 2^n bits and was proven to be a NP-complete problem [11]. To be absolutely sure that the chosen order is the most suitable one for reduction, all order combinations should be tested. Several methods and heuristics from static variable ordering to complex ordering algorithms were combined with DD in order to address scaling problems. Promising results were achieved with Evolutionary Algorithms (EA), which can be used not only for area reduction, but to simultaneously focus on optimization of multiple parameters, such as Average Path Length (APL) or power consumption. APL is critical for circuit's delay when the majority of paths have equal probability of being traversed. The delay can have the highest priority among given parameters at all levels of abstraction [12].

To further increase the number of types of DD, we propose yet another type of DD with yet another abbreviation – RViDD. This type of DD exploits the advantages of Residual variable used at the lowest level of DD and effectively decreases the complexity of $n+1$ variable input to that of n variable input by replacing one variable with an exact logic value representation. We present the experimental results achieved with a combination of Residual variable and EA.

2 Preliminaries

The input for DD is a Boolean function (B-function) $f:B_n \rightarrow B$ over a set of variables X_n denoted as $f(x_0,x_1,\ldots,x_{n-1}) = y$. Variables are ordered based on significance from left to right, denoted by their index, where the order of input variables corresponds to the decreasing weights assigned to the variables from left to right, starting from the weight of 2^{n-1} for variable x_0 down to the weight of 2^0 for x_{n-1}. Any B-function f specified by its binary vector y and a fixed variable ordering can be easily expressed as a DD [3]. For easier transformation to residual function, the given B-function can be represented by modified truth table shown in Table 1.

Table 1
Modified truth table of B-function

x_1	x_2	...	x_{n-2}	x_{n-1}	y	
					x_0	$\overline{x_0}$
0	0	...	0	0	$f(2^{n-1})$	$f(0)$
0	0	...	0	1	$f(2^{n-1}+1)$	$f(1)$
0	0	...	1	0	$f(2^{n-1}+2)$	$f(2)$
.	.	...	.	.	.	
.	.	...	.	.	.	
.	.	...	.	.	.	
1	1	...	1	0	$f(2^n-2)$	$f(2^{n-1}-2)$
1	1	...	1	1	$f(2^n-1)$	$f(2^{n-1}-1)$

The modification of given function to a function of residual variables can be done by representing the binary vector of function as a canonical matrix (1). Such matrix [13] contains $2^{(n-1)}$ columns of two rows. Each column represents a pair of values x_0 (bottom row) and $\overline{x_0}$ (top row) of the function.

$$(B(x_0, x_1,..., x_{n-1})) = \begin{Vmatrix} f(0) & f(1) & \ldots & f(2^{n-1}-2) & f(2^{n-1}-1) \\ f(2^{n-1}) & f(2^{n-1}+1) & \ldots & f(2^n-2) & f(2^n-1) \end{Vmatrix} \quad (1)$$

Decomposition of B-function takes the input binary vector y of length 2^n and produces an output of 2 vectors of length $2^n/2$. Shannon decomposition for variable x_i effectively splits the vector in half based on the value of variable, the true half for $f_{x_i}(x)$ and the false half for $f_{\overline{x_i}}(x)$. Similarly, positive Davio decomposition outputs 2 vectors consisting of the false half and an eXclusive OR (XOR) of both halves, and negative Davio decomposes the vector into the true half and a logical XOR of both halves. Decomposition functions are shown in formula (2), (3) and Figure 1.

Variable x_i is a chosen variable with decomposition applied to it.

$$f_0 = f_{\overline{x_i}}(x); \qquad f_1 = f_{x_i}(x); \qquad f_2 = (f_0 \oplus f_1) \tag{2}$$

$$S = f_0 . f_1; \quad pD = f_0 \oplus x_i . f_2; \quad nD = f_1 \oplus \overline{x_i} . f_2 \tag{3}$$

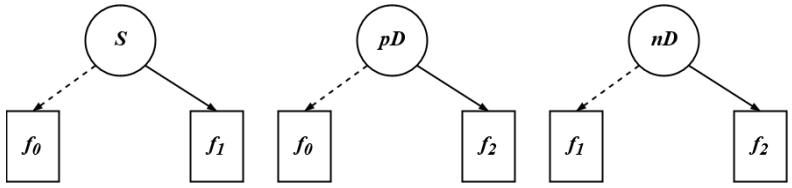

Figure 1
Three types of decompositions – Shannon (S), positive Davio (pD) and negative Davio (nD) from Formula 2 and 3

DD is a directed acyclic graph consisting of one root node, several intermediate nodes and up to two terminal nodes representing logical values true and false, usually labeled 1 and 0 respectively. Each non-terminal node is labeled by a Boolean variable x_i, depending on its position in the input order. Non-terminal nodes in unoptimized DD have one ingoing edge (with the exception of root node) and two outgoing edges labeled low and high representing the value of function f_i according to decomposition of the parent node. Each non-root non-terminal node is a root to a separate diagram called subdiagram.

DD optimization methods can be divided into two categories [14]:

1. DD ordering – results in Ordered Decision Diagram (ODD), which respects a given order of input variables. Variable ordering has a major impact on effectiveness of DD reduction.
2. DD reduction – when applied on ODD, results in Reduced ODD (ROBDD/ROKFDD) which has a lower node count than unreduced DD. Completely reduced DD represents a canonical form of DD. RODD respects three rules:
    a. Uniqueness (Type I) – no two distinct nodes *u* and *v* represent the same variable, have the same decomposition and have the same left and right successor. Reduction is applied when *var(u) = var(v), dec(u) = dec(v), left(u) = left(v), right(u) = right(v)* which implies *u = v*.
    b. Non-redundancy for Shannon node (Type S) – no variable node *u* has identical left and right successor. Reduction is applied when *left(u) = right(u)*.
    c. Non-redundancy for Davio node (Type D) – no variable node *u* has the right successor equal to terminal node 0. Reduction is applied when *right(u) = 0*.

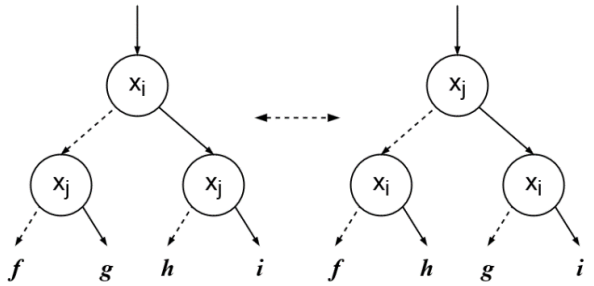

Figure 2
Exchange of adjacent variables in DD

As was already mentioned, the creation of an optimal DD has exponential increase in complexity. To further optimize a DD, an optimal input variable ordering has to be found. For circuits consisting of larger number input variables, searching the entire space of possibilities becomes unrealistic in acceptable time. Several methods were explored in ordering complexity mitigation with various degrees of reduction in final node count (size) of RODD. These methods can be separated into categories based on the complexity of underlying algorithm.

Basic methods are simple modifications in variable ordering. One such example is the exchange of adjacent variables on the same level in DD. The goal of exchanging variables at k and $k+1$ level for the DD G of function f is to transform G to DD G' of function f. The only difference between variable ordering π and π' is only at levels k and $k+1$ and can be expressed as $\pi(k) = \pi'(k+1)$ and $\pi(k+1) = \pi'(k)$. Exchange of adjacent variables does not affect the upper and lower levels in DD and is illustrated in Figure 2 [15].

In heuristic methods, the ordering of variables will be determined according to the information available about the issue before the construction of DD itself. Force algorithm [15] belongs among the best known algorithms in this category. The idea behind Force is simple - the algorithm computes the forces acting upon each variable and displaces the variables in the direction of the forces acting upon them. In Force, a CNF formula is viewed as a hypergraph, where the formula's variables correspond to vertices and clauses correspond to hyperedges. The algorithm itself determines two values during execution and iteratively uses them to order the variables. Another heuristic method [16] is based on the proven assumption that the number of nodes in a particular level of DD depends only on the arrangement of variables at lower levels. The algorithm of this method sequentially places all the variables to the first level and determines to which of them it received the least number of nodes. This variable (or several variables) is saved for the chosen level and the remaining variables are tested at upper levels. This process is repeated until the final DD is obtained. Algorithm complexity remains exponential, but provides better parameters than iterating through all possible variable orderings.

The third and final category consists of alternative methods based on evolutionary algorithms (EA). EA belong to a state-of-art in optimization algorithms. The core term of EA is population, which represents a set of chromosomes. Chromosome can be either the input order of variables or decompositions represented as a vector of genes, such as $\{x_0,x_5,x_6,x_3,x_2,x_1,x_4,x_7\}$. In this case, each gene is a variable in particular order (chromosome) of given population of orders. A population of a specific stage of EA is called generation. Initial population, or the first generation, is created using randomly generated chromosomes, which are then sorted through based on their fitness value to form a new generation.

After creating a population, a fitness value is calculated for each chromosome. The underlying algorithm of fitness calculation depends on the targeted problem. An example of multi-parametric fitness formula can be seen in (4) where A_r, P_w and A_p are the coefficients setting percentage weights of optimized parameters and their sum $A_r + P_w + A_p$ should be equal to 1 at all the times. The formula also contains 3 parameters (area, power consumption and APL) that have their orderings normalized to range *(0,1)*.

$$fitness = A_r \times area\_normalized + P_w \times power\_normalized + A_p \times apl\_normalized \tag{4}$$

Most fit chromosomes, then have a certain probability of genetic operations being applied to them to increase the diversity of current generation and decrease the chance of getting stuck in a local optimum. Some forms of EA also introduce a technique called elitism which ensures the preservation of the best chromosomes across generations in order to make sure subsequent generations never provide worse, and therefore useless, results than the previous generations.

Chromosomes of the current generation are selected based on their fitness value. At this point, EA starts to populate a new generation by selecting chromosomes from current generation and applying genetic mutation and crossover. A genetic mutation in vector of variables (chromosome) is to invert a random gene in chromosome. This operation is implemented as a swap of the variable on a random position and the variable positioned on the complementary position. For example, a mutation in chromosome of length 8, at position 2, is illustrated in Figure 3.

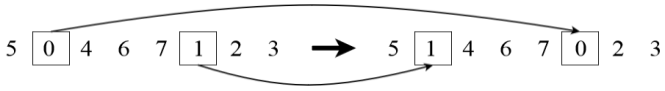

Figure 3
Mutation example

The second genetic operation called crossover causes two selected chromosomes to be cut at the same randomly chosen point and exchange their segments. In some variations, two points can be randomly selected in one chromosome and the

segment between these two points is then replaced with the segment on the same position in paired chromosome. Since a simple exchange of the parts of variable orderings can violate the uniqueness of each variable, this operation usually has to be corrected. Variables that are already present in the unchanged part of chromosome are replaced with unused variables. Crossover is illustrated in Figure 4, where variable 5 in the first chromosome and variable 6 in the second were corrected.

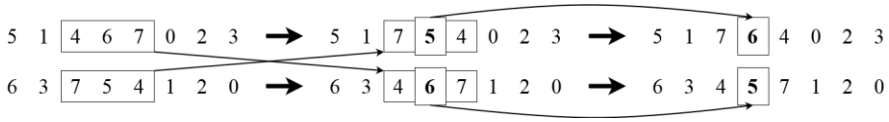

Figure 4
Crossover example

EA starts with the creation of a new population (in this work we also reuse previous generations several times) and keeps applying mutation and crossover on selected chromosomes with predefined probability. The size of population as well as the probability of EA operations are a subject to research themselves and can vary depending of the application of EA.

Several variations of EAs exist. For example, Particle Swarm Optimization (PSO) [17] is a population-based stochastic technique inspired by social behavior of bird flocking or fish schooling. In PSO, the potential solutions called particles fly through the problem space by following the current optimum particles. Particles learn from their past experiences, learn from experience of others and finally converge near the solution, which may be the best or a suitably good solution after satisfying a definite termination criterion. The particles sense their proximity to a good solution using a fitness function [18]. Another variation of EA with interesting results is modified memetic algorithm (MMA) [18]. The key feature of MMA is the use of various techniques of local search. While the gene that passes on the offspring cannot be changed (except for mutations) in the classic EA, memes transmit information among themselves so as to best suit the evaluation function (for example through local searches) in MMA which happens through knowledge of the solution's local space.

All the previously mentioned, algorithms share the same feature, which is that they primarily focus on size reduction. Number of nodes in DD is directly proportional to the size of represented circuit and has impact on other parameters as well, e.g. if the DD (and most notably BDD) is used as representation of a multiplexer tree, each node corresponds to a multiplexer and the dynamic power consumption of entire circuit can be easily estimated [19].

3 Residual Variable

If *n* variable B-function is to be implemented using the complexity of DD with *n-1* variables, one of the input variables must to be available in both direct and complemented form. The input variable order has to be modified in a way where this particular variable has the highest weight in binary vector.

Definition 1 – If the presences in a function of *n* variables are identified (e.g. by their order in truth table), it is possible to assign a weight to these variables. The variable with the highest weight can then be removed from the vector of variables and replaced with logic value in direct and complemented form. Such variable is called a *residual variable* (RV). An arbitrary variable may be a RV if it is available both in direct and complemented form, otherwise the transformation of RViDD to a specific circuit would require an additional NOT logic gate.

Definition 2 – If RV is identified in a function of *n* variables, the circuit representation of this function can be transformed into a circuit representation of function with *n-1* variables where the RV is connected to the data input.

Definition 3 – DD for the function *f* with *n* variables is called RViDD (Residual Variable in DD), if one variable is the residual variable and it is also a terminal node of RViDD.

To create a DD with *n-1* variables, the existing procedure repeats decomposition until it reaches the level defined by the formula 5 with chosen variable x_n

$$f_{x_{n-1}=c}(x_0,\ldots,x_{n-1}) := f((vector\ n-1),c) \tag{5}$$

In formula 5, *c* is a constant, $x_{n-1} = c$, $(c \in \{0, 1\})$ and *(vector n-1)* contains corresponding substitution of 0s and 1s according to the given order of variables $x_0,\ldots x_{n-1}$ in the upper levels of DD and its particular propagation path. for given variables $x_0,\ldots x_{n-1}$ is obtained.

In the case where the decomposition of input function is stopped one iteration earlier, formula 5 is transformed into formula 6.

$$f_{x_{n-2}=c}(x_0,\ldots,x_{n-2},x_{n-1}) := f((vector\ n-2),c,x_{n-1}) \tag{6}$$

In formula 6, *c* is a constant, $x_{n-2} = c$, $(c \in \{0, 1\})$ and *(vector n-2)* contains corresponding substitution of 0s and 1s according to the given order of variables $x_0,\ldots x_{n-2}$ in the upper levels of DD and its particular propagation path. It is possible by using this method to achieve up to four final states, or rather substitution rules, which depend on the value *c* and x_{n-1}. Substitution rules take 2 input values and provide 1 output value, which represents RV, as shown in Table 2, where v_1 represents $f((vector\ n-2),x_{n-1})$, v_2 represents $f((vector\ n-2),\overline{x_{n-1}})$ and u_i represents value of *i*-th leaf in RViDD.

Table 2
Substitution rules for residual functions

Rule	v_1	v_2	u_i
0	0	0	0
1	0	1	x_{n-1}
2	1	0	$\overline{x_{n-1}}$
3	1	1	1

Using formula (2), (3) and rules in Table 2, final states of RV obtained with every decomposition are shown in Table 3 for any chosen residual variable (x_i).

Table 3
Final states of residual variable x_i for every decomposition

f_0	f_1	f_2 $(f_0 \oplus f_1)$	Function value			Final state		
			S	pD	nD	S	pD	nD
0	0	0	0.0	$0 \oplus x_i.0$	$0 \oplus \overline{x_i}.0$	0	0	0
0	1	1	0.1	$0 \oplus x_i.1$	$1 \oplus \overline{x_i}.1$	x_i	x_i	x_i
1	0	1	1.0	$1 \oplus x_i.1$	$0 \oplus \overline{x_i}.1$	$\overline{x_i}$	$\overline{x_i}$	$\overline{x_i}$
1	1	0	1.1	$1 \oplus x_i.0$	$1 \oplus \overline{x_i}.0$	1	1	1

Example 1: Suppose the input B-function f_1 with 4 variables in given order $f_1(x_2,x_0,x_1,x_3)=1000010001011011$ where variable x_2 is chosen as RV. For simplicity, only Shannon decomposition is used. Truth table and its modified version for f_1 are shown in Table 4. Results in column y can be expressed as a canonical matrix B_1 of resulting values (7). The first row represents vector v_1 from Table 2 and the second row represents values from vector v_2.

$$(B_1(x_2, x_0, x_1, x_3)) = \left\| \begin{matrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \end{matrix} \right\| \tag{7}$$

Using substitution rules in Table 2, canonical matrix (7) can be expressed as a vector Z_1 of residual functions (8). Notice that the length of vector Z_1 is half the length of input vector f_1. Same steps apply for all decomposition types (Table 3).

$$(Z_1(x_2, x_0, x_1, x_3)) = (\overline{x_2}, x_2, 0, x_2, x_2, \overline{x_2}, x_2, x_2) \tag{8}$$

3.1 Replacement of Residual Variable

Since RViDD is a new type of DD, it is important to maintain properties for basic reduction rules in order to reuse existing reduction and optimization algorithms developed mainly for BDD or KFDD. As a consequence, main features of DD

with any type of decomposition are preserved and the DD can be directly compared to RViDD in terms of various factors, such as size (node count).

Table 4
Modified truth table for f_1 from Example 1

x_2	x_0	x_1	x_3	y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

➡

x_0	x_1	x_3	y	
			x_2	$\overline{x_2}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

Transformation of DD node at the first level to RViDD node is done according to rules in Table 3 where the residual functions are replaced with corresponding final state of RV (Figure 5). This transformation applies to all decompositions since their final states are equal for each input. Upper levels of RViDD remain unchanged and identical to their DD counterpart.

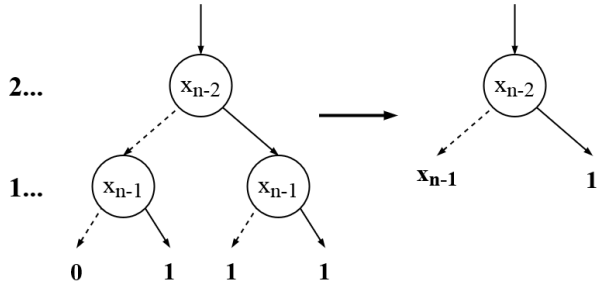

Figure 5
Example of transformation of DD node to RViDD node

An important feature in reduction, is the exchange of adjacent variables. Although the exchange itself is not very efficient, it serves as the basis for many algorithms. If adjacent variables are exchanged (levels *i* and *j*) then variable ordering of given function is modified from $f(x_0,\ldots,x_i,x_j,\ldots,x_{n-1})$ to $f(x_0,\ldots,x_j,x_i,\ldots,x_{n-1})$ without disrupting upper and lower levels. This approach remains unchanged if none of the variables is positioned at the first level, i.e. none of the variables is a RV.

Replacing RV with another variable requires a few more steps. While terminal nodes in DD can only have 2 values *{0,1}*, which lead up to 4 possible states *{00,01,10,11}*, RViDD has 4 values $\{0,1,x_i,\overline{x_i}\}$, which lead up to 16 possible states. Exchanging residual variable therefore has to follow rules in Table 2 and the final state after exchange can be achieved by simply deconstructing the RV into B-functions, swap the variables as in DD and construct RViDD again by substituting the terminal level functions with newly chosen RV. Rules for exchanging RV are shown in Table 5, where only different outcomes are displayed, rules with identical outcomes or outcomes where RV is simply replaced by a new one (while maintaining position and negation of RV) are omitted.

Table 5
Rules for exchanging residual variable

Before exchange	After exchange	Before exchange	After exchange
Binary vector (v_1,v_2,v_3,v_4)	Binary vector (v_1,v_3,v_2,v_4)	Values (u_1,u_2)	Values (u_1',u_2')
00,10	01,00	$0,\overline{x_i}$	$x_j,0$
00,11	01,01	0,1	x_j,x_j
01,00	00,10	$x_i,0$	$0,\overline{x_j}$
01,01	00,11	x_i,x_i	0,1
10,10	11,00	$x_i,\overline{x_i}$	1,0
10,11	11,01	$\overline{x_i},1$	$1,x_j$
11,00	10,10	1,0	$\overline{x_j},\overline{x_j}$
11,01	10,11	$1,x_i$	$\overline{x_j},1$

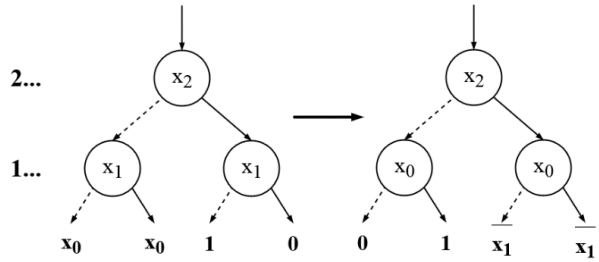

Figure 6
Replacement of residual variable in RViBDD

Replacing RV allows the modification of existing RViDD, which is more efficient than constructing a new one with a different RV. Since the replacement of RV needs to perform more operations than a common exchange of variables at adjacent levels in DD, it can be used in conjunction with basic methods to find the variable that appears to be most suitable for the position of RV. Although it requires more steps, this appears to be the only drawback in computation time of synthesis in RViDD. Example of RV replacement is shown in Figure 6.

3.2 RViDD Construction

Important advantage of RV is the fact that the final states (Table 3) are equal for every decomposition. This automatically preserves the ability to apply reduction rules to nodes in RViDD. Reduction rule I can still be applied to any 2 nodes with 2 identical successors. Reduction rule S can be used on any Shannon node that has 2 identical successors, even if both of them are RV, e.g. the right successor of x_2 node in RViBDD in Figure 6. Reduction rule D applies to any Davio node whose right successor is equal to terminal value 0.

Example 2: Suppose the input B-function f_2 with 4 variables $f_2(x_2,x_1,x_3,x_0)=0101011010011001$. Unreduced BDD for f_2 is shown in Figure 7.

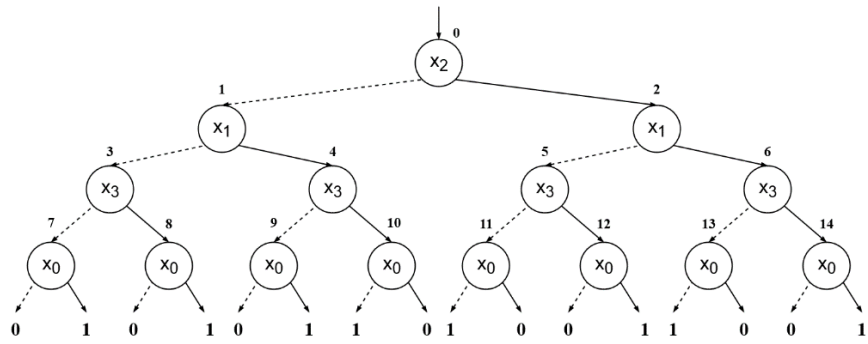

Figure 7
BDD for f_2 from Example 2

Variable x_0 is chosen as RV. Following the same steps presented in Section 2 - Example 1, a modified truth table is constructed with vector of residual functions Z_2 (9). For simplicity, only Shannon decomposition is used. Unreduced RViBDD for f_2 is shown in Figure 8. Recursively applying reduction rules results in reduced RViBDD shown in Figure 9.

$$(Z_2(x_2, x_1, x_3, x_0)) = (x_0, x_0, x_0, \overline{x_0}, \overline{x_0}, x_0, \overline{x_0}, x_0) \tag{9}$$

Figure 7 shows all nodes for the complete unreduced DD. With top-to-bottom construction approach (synthesis) of DD, reduction rule I can be applied on any node which shares characteristics with an already existing node – the same level, variable, decomposition and identical successors. This eliminates the need to synthesize entire subdiagram of the reduced node.

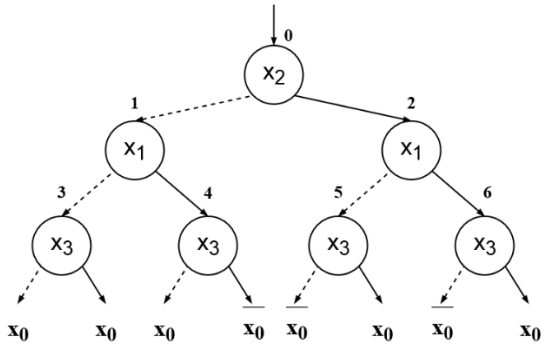

Figure 8
Unreduced RViBDD for f_2 from Example 2

From Figure 7 and 8 it can be seen that RViBDD has half the node count of BDD thanks to omitting the most numerous first level and replacing it with RV. Reducing RViBDD is subject to the same reduction approach as DD but with lower node count, therefore the time of reduction should be, in theory, halved.

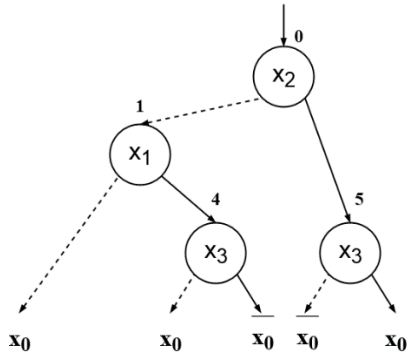

Figure 9
Reduced RViBDD for f_2 from Example 2

Constructing RViDD follows the same steps as a regular DD construction with any decomposition. Thanks to properties of RV, reduction rules remain unchanged and can be applied equally to DD and RViDD, which makes RV an important contributor to size reduction in binary-based DDs. For DDs using only Shannon decomposition, terminal values can be directly derived and the input B-function can be immediately replaced with vector of residual functions, even before DD construction. Thus the construction cost is minimized right after ordering phase. Davio decompositions have a slight disadvantage, since the terminal values cannot be directly estimated from the input function. The DD has to be constructed first and only once the state of the first level is known can the RV be applied. Although it has virtually no impact on construction time, it still lowers the number of nodes to check for reduction rules by removing the most numerous level.

4 Experimental Results

The advantages RV brings to DD optimization were verified on benchmark circuits LGSynth'93 [20]. The use of RV shows significant improvement in size reduction as well as the expected improvement in synthesis time. As was already mentioned in Section 3.2, RV effectively halves the size of DD and therefore lowers the number of nodes that need to be checked for reduction suitability in each iteration. Table 6 shows results achieved using only Shannon decomposition so the use of different decomposition methods does not obscure the actual impact of RV itself. Columns BDD and RViBDD show the number of nodes of reduced and ordered DD and columns Size imp. and Time imp. indicate the improvement in size and synthesis time respectively. Benchmarks marked with * used *n* orders instead of *n!*, where *n* is the number of variables.

The average improvement in time needed for synthesis of reduced and ordered RViBDD compared to BDD is 40.7% and in some cases rises up to 81.38%. The size reduction shows improvement between 6.06% and 33.33%. It is important to mention the possibility where the initial check on the most numerous level will not apply any RV and the computation time may slightly increase. As can be seen, this does not happen often and average computation time is lower in almost every case. Presented results were achieved using a new algorithm that combines RV, basic reduction rules and EA with following parameters:

- Check all possible orders and decomposition for functions with < 8 input variables
- Population size: 500 (< 12 input variables), 200 (>= 12 and < 21 input variables) or 100 (>= 21 input variables)
- Crossover probability: 80%
- Mutation probability: 20%

- Elitism: 1%
- Iterations (population count): 100

Table 6
Residual variable impact on BDD

Benchmark	BDD	RViBDD	Size imp. [%]	Time imp. [%]
parity*	31	29	6.45	-
cm151a*	44	30	31.82	-
cm152a*	21	14	33.33	-
sao2	103	96	6.8	45.13
9sym	33	31	6.06	41.28
sqrt8	35	32	8.57	40.45
rd84	71	64	9.86	43.13
misex1	62	49	20.97	20.41
Inc	96	81	15.63	43.15
5xp1	76	59	22.37	34.94
xor5	9	7	22.22	30.64
con1	15	12	20	81.38
squart5	47	34	27.66	31.64
rd53	29	24	17.24	39.41
majority	7	6	14.29	29.23
Average	45.27	37.87	17.55	32.05

The overall percentage improvement in size reduction in RViDD for chosen benchmarks is shown in Table 7. Columns *S*, *pD* and *nD* show the improvement of reduced and ordered RViDD against unreduced and unordered DD with the respective decomposition. Column *RViKFDD* shows the improvement when all 3 decompositions are combined with RV. It is obvious that the combination of all decomposition methods provides the best results with average of 87.24% in comparison to single decomposition used. RViKFDD takes advantage of EA not only for population of orders, but for vector of decompositions as well (each variable has exactly 1 decomposition assigned) using the same parameters mentioned above with 2 exceptions – no fitness function and no elitism.

Table 7
Residual variable impact on size reduction in [%] with various decompositions

Benchmark	S	pD	nD	RViKFDD
misex1	89.64	85.62	89.43	90.7
Inc	92.02	89.46	89.66	92.12
5xp1	90.13	87.58	87.58	90.92
xor5	77.42	80.65	80.65	87.1
con1	87.23	84.04	85.11	88.3
squart5	86.29	85.48	84.27	87.5

rd53	74.19	78.49	78.49	80.65
majority	80.65	77.42	77.42	80.65
Average	84.70	73.59	84.08	87.24

Table 8
Comparison of various algorithms in size reduction

Benchmark	BDD	RViBDD	RViKFDD	PSO	MMA	Sifting
cordic	209	144	102	105	-	93
cm150a	32	31	31	32	-	33
mux	32	31	31	32	-	33
cm151a	32	30	29	32	-	34
sao2	103	96	96	91	85	92
9sym	33	31	25	-	33	33
sqrt8	35	32	31	33	33	42
rd84	71	64	46	-	59	59
misex1	62	49	44	36	36	41
Inc	96	81	80	79	61	68
5xp1	76	59	56	68	68	82
con1	15	12	11	16	15	18
squar5	47	34	31	37	37	38
rd53	29	24	18	-	23	23

Subsequent generations of decomposition chromosomes were chosen randomly. Choosing the fitness of a certain decomposition is dependent on the input function and order of variables, which is not known during population creation.

Newly proposed algorithm with RV was compared in matter of size with PSO, MMA and Sifting, a method presented in [21]. This comparison is shown in Table 8 (using the same EA parameters as in Table 6). Column BDD shows reduced and ordered BDD and is a clear indication that on its own, the Shannon decomposition is not sufficient enough to achieve optimal results. Columns RViBDD and RViKFDD show reduced and ordered results for their respective DD with RV applied. It is again proved that a combination of all decompositions provides better results than using a single decomposition. Applying RV provides improvement not only in comparison to BDD, but also to other current methods in field. Combination of RV and several decompositions shows better results in 10 out of 14 cases.

RV improvement in multi-parametric optimization of underlying circuits is shown in Table 9. Since the number of nodes has usually the highest priority among all parameters, input values for fitness functions were chosen accordingly:

- Number of nodes: 98%
- Average Path Length (APL): 1%
- Power Consumption (PC): 1%

Achieved results could be optimized in other ways choosing different parameter, e.g. if power consumption holds a higher level of importance in comparison to area of circuit, its impact value could be higher and that of a size parameter could be lower (the sum always has to add up to 100%).

Table 9
RV impact on Power Consumption and Average Path Length using Shannon decomposition

Benchmark	In	Out	Size	PC min.	PC max.	PC diff.	APL min.	APL max.	APL diff.
cordic	23	2	144	28.36	30.38	6.66	11.75	14.09	16.63
cm150a	21	1	31	11.63	15.38	24.39	3.06	4.13	25.85
mux	21	1	31	15.38	15.38	0	3.06	3.06	0
t481	16	1	36	14.86	14.86	0	4.15	4.15	0
parity	16	1	29	14.5	14.5	0	15	15	0
cm151a	12	2	30	11.25	14.75	23.73	5.25	6.5	19.24
cm152a	11	1	14	7	7	0	3.25	3.25	0
sao2	10	4	96	23.52	28.88	18.54	10.33	12.36	16.44
9sym	9	1	31	10.22	11.22	8.92	7.13	8.13	12.31
sqrt8	8	4	32	13.43	14.23	5.65	9.94	11.31	12.15
rd84	8	4	64	24.52	24.52	0	22.36	22.36	0
misex1	7	7	49	18.91	20.44	7.49	16.75	19.06	12.13
inc	7	9	81	30.06	33.27	9.63	20.75	25.16	17.52
5xp1	7	10	59	27.82	29.5	5.69	21.66	23.34	7.23
con1	6	2	12	5.26	5.26	0	4.25	4.56	6.85
xor5	5	1	7	3.5	3.5	0	4	4	0
squar5	5	8	34	13.7	14.65	6.5	15.38	17	9.56
rd53	5	3	24	10.1	10.1	0	11.25	11.25	0

Columns In, Out and Size represent the number of inputs and outputs for each benchmark and the size of RViBDD (only Shannon decomposition was used in this table). Columns PC min., PC max. and PC diff. display the minimal and maximal achievable power consumption (PC) and their difference in [%]. PC is shown in relative value independent of technology used in circuit synthesis and is directly proportional to the number of switches performed in circuit. This value was estimated based on formulas presented in [19]. The final trinity of columns marked as APL min., APL max. and APL diff. display the minimal and maximal achievable values for APL and their difference in [%]. The highest achieved improvement was 24.39% in PC and 25.58% in APL, while the average improvement for all tested benchmarks comes down to 6.24% in PC and 9.47% in APL. It can be observed that symmetric functions have little to no improvement in both parameters due to limited changes in structure of DD during variable reordering.

Conclusions

We have proposed a novel solution for the optimization of binary-based Decision Diagrams (DD), by introducing new type of DD. Our RViDD (Residual Variable in Decision Diagrams) uses one input variable as a residual variable which can be utilized as another type of a terminal node. Thanks to this modification, the optimized RViDD has almost half the nodes, compared to unoptimized DD with the same variable ordering.

We proved that the same basic reduction rules can be used as well for RViDD as for DD without any modification. Another well-known rule – exchange of adjacent variables – remains also applicable with only a small modification in the level of terminal nodes. We called this procedure "replacement of residual variable" where residual variable (RV) can be replaced by any other input variable which might prove to be necessary during optimization phase. Replacement of RV leads up to 16 different possible states (compared to 4 in DD) out of which only 8 lead to other than simple 1:1 swapping of the old RV for a new one.

During experimental phase we chose several known types of DD (BDD, FDD, KFDD) and created their equivalents with residual variable (RViBDD, RViFDD, RViKFDD). Our focus was on comparison of three parameters, primarily on number of nodes, but also on energy consumption and average path length (APL). In average our solution (residual variable in DD) has 17.55% less nodes than the solution without residual variable. Because RV can change the most suitable decomposition for given benchmark circuit, a test comparing its impact on various decompositions is presented. It is not a surprise that the best solution is based on RViKFDD which uses all types of decompositions. Based on the previous experiment, we were able to tell that residual variable is suitable for all types of decomposition, because there were no exceptions (results with smaller improvement).

We also tried to use more complicated optimization process by involving evolutionary algorithm. We compared it with Sifting method (as reference), Particle swarm optimization (PSO) and Modified memetic algorithm (MMA). Even though evolutionary algorithm is not superior for PSO and MMA in every case, our solution (RViBDD) was better in 8 cases (10 for RViKFDD) out of 14 benchmarks with improvement up to 22.51%. This proves the impact residual variable has in optimization process.

The Residual variable also shows the positive impact on multi-parametric optimization (number of nodes, power consumption and Average Path Length (APL)). In average, dynamic power consumption of any underlying circuit should be decreased by 6.24%. This is achieved by lowering the number of switches performed in the circuit when traversing the diagram. APL improvement moves around 9.47% for all tested benchmarks, which also shows RV can (positively) affect the symmetry of the circuit, should it be one of the desired attributes.

While all the presented results display a positive influence of residual variable on DD optimization, there is still room for improvement. One such case would be to introduce the logic of Free DDs [22] where the rule that each path from root to terminal nodes has to follow the same order of variables is relaxed. Although this greatly increases the complexity of the used algorithm, it is expected to bring an even further decrease, in all observed parameters.

Acknowledgement

This work was supported by the Cultural and Educational Grant Agency of the Slovak Republic (KEGA 011STU-4/2017) and ITMS 26240220084.

References

[1] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," in IEEE Transactions on Computers, Vol. C-35, No. 8, pp. 677-691, Aug. 1986, doi:10.1109/TC.1986.1676819

[2] R. E. Bryant, "Binary decision diagrams and beyond: enabling technologies for formal verification," Proceedings of IEEE International Conference on Computer Aided Design (ICCAD), San Jose, CA, USA, 1995, pp. 236-243, doi:10.1109/ICCAD.1995.480018

[3] P. Pistek, “New multiplexer-based switching circuits synthesis methods.” in Information Sciences and Technologies, Vol. 7, No. 1, pp 19-27, 2015

[4] A. Deb, et al, “Synthesis of Optical Circuits Using Binary Decision Diagrams.” Integration, Vol. 59, pp. 42-51, May 2017, doi:https://doi.org/10.1016/j.vlsi.2017.05.001

[5] R. Wille, P. Niemann, A. Zulehner and R. Drechsler, "Decision diagrams for the design of reversible and quantum circuits," 2018 International Symposium on Devices, Circuits and Systems (ISDCS), Howrah, 2018, pp. 1-6, doi:10.1109/ISDCS.2018.8379626

[6] Technology Roadmap for Semiconductors: Design. 2015: https://www.semiconductors.org/resources/2015-international-technology-roadmap-for-semiconductors-itrs/ (accessed June 1, 2016)

[7] B. Nagy, “Optimal Boolean Programming with Graphs.” in Acta Polytechnica Hungarica, Vol. 16, No. 4, 2019, doi:10.12700/APH.16.4.2019.4.12

[8] B. Becker and R. Drechsler, "How many decomposition types do we need? [decision diagrams]," Proceedings the European Design and Test Conference. ED&TC 1995, Paris, France, 1995, pp. 438-443, doi: 10.1109/EDTC.1995.470359

[9] L. Amarú, P. Gaillardon and G. De Micheli, "Majority-Inverter Graph: A New Paradigm for Logic Optimization," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 35, No. 5, pp. 806-819, May 2016, doi: 10.1109/TCAD.2015.2488484

[10] L. Amarú, P. Gaillardon and G. De Micheli, "BDS-MAJ: A BDD-based logic synthesis tool exploiting majority logic decomposition," 2013 50th

ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, 2013, pp. 1-6, doi: 10.1145/2463209.2488792

[11] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," in IEEE Transactions on Computers, Vol. 45, No. 9, pp. 993-1002, Sept. 1996, doi: 10.1109/12.537122

[12] R. Gerov, Z. Jovanovic, “Parameter Estimation Method for the Unstable Time Delay Process.” In Acta Polytechnica Hungarica, Vol. 16, No. 3, 2019, doi: 10.12700/APH.16.3.2019.3.6

[13] P. Pistek, M. Kolesar and K. Jelemenska, "Optimization of multiplexer trees using modified truth table," 2010 International Conference on Applied Electronics, Pilsen, 2010, pp. 1-4

[14] R. Ebendt, G. Fey, R. Drechsler, ”Advanced BDD Optimization” (1. ed.). Netherlands: Springer, 2005, 222p, ISBN 978-0-387-25453-, doi: 10.1007/b107399

[15] M. Rice, S. Kulhari, “A Survey of Static Variable Ordering Heuristics for Efficient BDD/MDD Construction”. University of California, CA, USA, Tech. Rep., 2008 [Online] Available: shorturl.at/cCGMR

[16] S. J. Friedman and K. J. Supowit, "Finding the optimal variable ordering for binary decision diagrams," in IEEE Transactions on Computers, Vol. 39, No. 5, pp. 710-713, May 1990, doi: 10.1109/12.53586

[17] A. Mitra and S. Chattopadhyay, "Variable ordering for shared binary decision diagrams targeting node count and path length optimisation using particle swarm technique," in IET Computers & Digital Techniques, Vol. 6, No. 6, pp. 353-361, November 2012, doi: 10.1049/iet-cdt.2011.0051

[18] S. Rehan, M. Bansal, “Performance Comparison among Different Evolutionary Algorithms in terms of Node Count Reduction in BDDs” in International Journal of VLSI and Embedded Systems, Vol. 4, pp. 491-496, July 2013

[19] R. Marcolescu, R. Marculescu, M. Pedram, "Efficient Power Estimation for Highly Correlated Input Streams," 32nd Design Automation Conference, San Francisco, CA, 1995, pp. 628-634, doi: 10.1145/217474.217601

[20] K. McElvain, “LGSynth93 Benchmark Set: Version 4.0”, 1993. Distributed by NC State University. Available: https://ddd.fit.cvut.cz/prj/Benchmarks/IWLS93.7z

[21] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," Proceedings of 1993 International Conference on Computer Aided Design (ICCAD), Santa Clara, CA, USA, 1993, pp. 42-47, doi: 10.1109/ICCAD.1993.580029

[22] J. Bern, J. Gergov, C. Meinel and A. Slobodova, "Boolean manipulation with free BDD's. First experimental results," Proceedings of European Design and Test Conference EDAC-ETC-EUROASIC, Paris, France, 1994, pp. 200-207, doi: 10.1109/EDTC.1994.326915