

Article

Optimization of Binary Decision Diagrams by Single-Grid Cellular Genetic Algorithm

Iulian Furdu ¹, Cosmin Tomozei ^{1,*} and Bogdan Pătruț ² ¹ Department of Mathematics and Informatics, Faculty of Sciences, Vasile Alecsandri University of Bacău, 600115 Bacău, Romania; ifurdu@ub.ro² Department of Computer Science, Faculty of Computer Science, Alexandru Ioan Cuza University of Iași, 700506 Iași, Romania; bogdan.patrut@info.uaic.ro

* Correspondence: cosmin.tomozei@ub.ro

Abstract

This paper presents a new approach based on cellular genetic algorithms for the problem of optimizing the variable ordering of binary decision diagrams. The initial population is placed on a bidimensional grid by means of a Kohonen self-organizing map, which acts on the associated feature vectors of chromosomes. The offspring are placed on the grid in the proximity of their parents at a distance that reflects the structural difference from them. Devouring techniques that uses the organizing of the population into clusters based on proximity ensure the selection accomplishes the survival selection. The experimental evaluation demonstrates reliable behavior across the evaluated benchmarks and represents a promising alternative heuristic for BDD minimization.

Keywords: binary decision diagram—BDD; variable ordering optimization; cellular genetic algorithm

MSC: 68T20; 90C35; 68U99

1. Introduction

Binary decision diagrams (BDDs) are the most powerful data structure for Boolean function manipulation. Among the many areas of application are hardware verification, model checking, and analysis of sequential systems. Additionally, BDDs have been effectively exploited to solve problems involving many variables in decision-making, reliability analysis, and performance analysis. They offer not only a concise representation for the switching functions but also provide fast manipulation. The reduced ordered binary decision diagrams (ROBDDs) introduced by Bryant [1] provide a canonical representation for the Boolean functions. In the sequel, according to the general use in the literature, we refer to a reduced OBDD as a BDD.

The number of non-terminal nodes in the graph represents the size of the OBDD and this is the value that must be minimized. An important problem is the fact that the BDD's size is very sensitive to the choice of ordering of the input variables. It is proven that optimizing the order of variables is NP-complete and finding the best order is NP-hard [2]. This paper addresses this problem, known as the variable ordering problem for BDDs. The next section describes both classic and recent techniques and algorithms for solving this problem and synthesizing new trends. Section 3 formulates the problem. Section 4 describes the principles of cellular genetic algorithms (GAs) and defines the main components of the



Academic Editors: Zdeněk Kala,
Jurgita Antucheviciene and
Janez Žerovnik

Received: 26 May 2026

Revised: 17 July 2026

Accepted: 23 July 2026

Published: 4 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

cellular GA for optimizing BDDs. Section 5 presents the experimental results, and the last section summarizes the work.

2. Background and Recent Trends in BDD Optimization

In the last decades, many methods have been proposed to solve the variable ordering problem for BDDs. The methods for finding the best variable ordering—the ordering which gives the smallest number of nodes for the BDD representation of a certain Boolean function—can be roughly classified in static and dynamic techniques. Static heuristics try to find a good initial order before starting the construction of the BDD. While most of the classic static techniques [3] use specific problem information to find good starting variable orders, the dynamic techniques [4,5] are based on a process of dynamically improving the size of an already built BDD. One of the most successful dynamic reordering strategies is Rudell’s sifting algorithm [5]. This algorithm tries to determine a good variable ordering of a BDD starting from the natural ordering by successively analyzing each variable. The investigated variable is moved through the whole ordering and put in its optimal position, i.e., the position for which the minimum of BDD size is achieved. The searching routine for a variable is stopped if a growing factor c is exceeded. Other sifting algorithm variants include block sifting, symmetric sifting and linear sifting [6].

From the perspective of evolutionary computation, some of the classic methods for optimizing the variable ordering based on genetic algorithms are described in [7,8] and the experimental results [9] have proven that they yield better results than other heuristic techniques.

Actual trends in BDD optimization are still focused on minimizing BDD node count while maintaining computational efficiency. Given the versatility of BDD as a data structure in many areas, new research on BDD optimization covers a wide field of problems from feature analysis to quantum computing. Advancing BDD compilation capabilities on real-world feature models is an important research direction [10] since BDD compilation has gained much traction within the context of feature model analysis. New tools have been created (DIMAGIC2.0) alongside existing ones (FORCE, ReMINCE, and ImplicPBDD) [10,11]. Machine learning with feature engineering or end-to-end deep learning are used as node-selection heuristics to construct restricted decision diagrams in the context of multi-objective integer linear programming. These data structures produce high-quality approximations of the Pareto frontier when decision diagram size exceeds memory and must be restricted to a subset of its states [12]. The same authors developed LEO [13], a supervised learning algorithm 30–300% faster at finding efficient variable orderings than common ordering strategies in the context of reducing the Pareto frontier enumeration time. Smaller BDDs are resolved faster by a novel divide-and-conquer based strategy for efficient evaluation of variable ordering [14].

Another research direction for BDD optimization is based on evolutionary computing. Experimental results show that hybrid methods outperform standalone heuristics in terms of BDD node minimization, particularly for large Boolean benchmarks. Recent work proposes a BDD optimizer driven by either genetic algorithm (GA) or swarm engines to maintain the equilibrium between exploration and exploitation during reordering [15]. In [16], this equilibrium is obtained by an enhanced GA reordering algorithm which chooses adequate variation operators based on a diversity measure calculated per generation. A quantum algorithm is used to produce a minimum BDD [16] in time/space complexity $O(2.77286^n)$ with an exponentially small error probability improving the result of $O(3^n)$ of the Friedman and Supowit deterministic algorithm [17].

While deterministic heuristics often get trapped in sub-optimal ordering regions and sifting algorithms still remain a standard, for artificial intelligence-based heuristics,

problems like high computational cost, parameter sensitivity or required training data have to be surpassed.

Modern research increasingly employs deep learning to predict high-quality initial variable orders or to guide dynamic reordering heuristics. In the context of BDD minimization, the authors of [18] present the first study applying neural networks to predict near-optimal variable orders for previously unseen Boolean formulas represented as hypergraphs. Their implementation consistently identifies solutions close to the optimal ordering while achieving substantially shorter computation times, even for computationally challenging instances.

Recent methodological advances—such as spatial–channel collaborative multi-scale graph–interaction deep transfer learning, multi-modal reversible-guidance frameworks, and cyclical knowledge-distillation strategies—have demonstrated strong capabilities in extracting deep structural and functional relationships from complex datasets. In parallel, the integration of robust fault-tolerant control mechanisms, including fixed-time adaptive fault-tolerant architectures with variable prescribed-performance bounds, has become essential for ensuring reliability under random disturbances and model uncertainties [19].

In a related direction, the authors of [20] propose a quantitative and qualitative reliability analysis of wind turbines using BDD-based modeling. These developments indicate that future breakthroughs will likely arise from learning-guided hybrid optimization frameworks that combine structural learning with heuristic or evolutionary search.

The next section defines the variable ordering problem in BDD minimization.

3. The Variable Ordering Problem

The role of reduced OBDDs as the underlying data structure and the critical importance of variable ordering in BDD construction are briefly reviewed. Let $f: B^n \rightarrow B^m$, $B = \{0, 1\}$ be a Boolean function. Let π be a total order of a fixed set of Boolean variables $x = (x_1, \dots, x_n)$, $x_i \in B$. An ordered BDD (OBDD) for f with respect to π is a single rooted direct acyclic graph that satisfies the following properties:

- (a) There are exactly two terminal nodes labeled by Boolean constants 0 and 1, respectively.
- (b) Each non-terminal node is labeled by a variable x_i and has two outgoing edges, called 0-edge and 1-edge, respectively; in each internal node k , the two subfunctions f_{x_k} and $f_{x_k^-}$ are computed according to Shannon decomposition [18]:

$$f = (x_1, \dots, x_k, \dots, x_n) = (x_k \cdot f_{x_k}) + (x_k^- \cdot f_{x_k^-})$$

- (c) The order in which the variables appear on a path is consistent with π ; that is, each variable is encountered at most once on each path from the root to a terminal node, and all paths respect the same variable order.
- (d) Further reduction rules could be applied [1] to obtain a canonical representation of f . An OBDD is called *reduced* (ROBDD) if it contains neither nodes whose subgraphs are isomorphic nor nodes whose two outgoing edges point to the same successor.

An ROBDD therefore constitutes a canonical form for a Boolean function. The size of the structure depends strongly on the chosen order. In this paper, we assume that all OBDDs are reduced, denoted simply as BDDs.

The permutation of the input variables' order often leads to different BDDs for the same Boolean function. Hence, in some cases the sizes of BDDs could be significantly different. A small example is presented in Figure 1, where the natural order (a, b, c, d, e, f) leads to four nodes, while the permutation (a, b, d, c, e, f) gives six nodes. Although many optimization methods and variants of the original BDD representation have been

introduced over the last decades, most of the existing techniques are limited by this memory explosion problem.

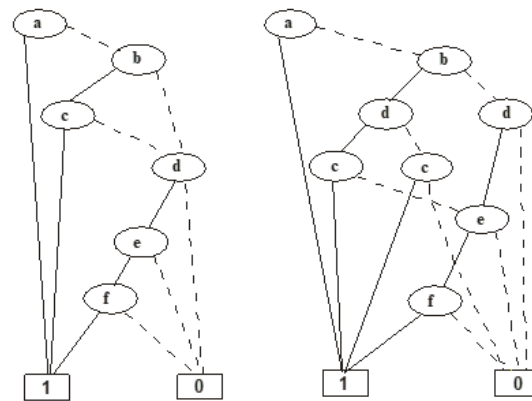


Figure 1. The influence of the variable ordering for $f(a, b, c, d, e, f) = a + bc + def$ with order a, b, c, d, e, f (left) and order a, b, d, c, e, f (right), (0—edges are dashed).

Formally, let $f: B^n \rightarrow B^m$, with $B = \{0, 1\}$ denote the Boolean function under optimization. A chromosome encodes, in binary form, a given permutation $\pi \in S_n$. The reduced OBDD is constructed using CUDD package (Section 5) and the objective function—i.e., the chromosome fitness—is defined as:

$$\text{fit}(\pi) = |\text{ROBDD}(f, \pi)|,$$

the number of non-terminal nodes in the corresponding OBDD. The optimization problem is therefore:

$$\min \text{fit}(\pi), \pi \in S_n,$$

which directly reflects the NP-complete nature of determining an optimal variable ordering. From a hardware perspective, the permutation that yields the smallest BDD—measured in terms of node count—corresponds to the representation requiring minimal chip area. This paper describes a new technique for optimizing variable ordering in BDDs based on cellular genetic algorithms. The next section presents the main features of the proposed algorithm.

4. Cellular Genetic Algorithms for Optimizing the Variable Ordering

The proposed method builds upon earlier hybrid and cellular GA frameworks; however, its main contribution lies in the unified grid-based integration of a self-organizing-map-driven, similarity-preserving population placement. This mechanism employs a distance-controlled spatial arrangement of offspring that reflects structural variation within the permutation space. In addition, a cluster-adaptive predator–prey devouring scheme is used, enabling dynamic adaptation to the evolving spatial distribution of candidate solutions. This integration provides effective management of structural diversity and high-quality solutions across all benchmark instances.

The cellular model of GAs represents a variant of GAs (Figure 2) in which the chromosomes (usually, called preys) are placed on a single bidimensional grid. The survival selection is accomplished by a set of mobile entities called predators that move on the same lattice. This matrix support allows miming of various reproduction, selection, and mutation processes, which constitutes an appropriate way to define the mutation and crossover experiments. It also tries to organize the solutions so that the natural implicit clustering of the preys is the expression of the structural or cost-based similarities.

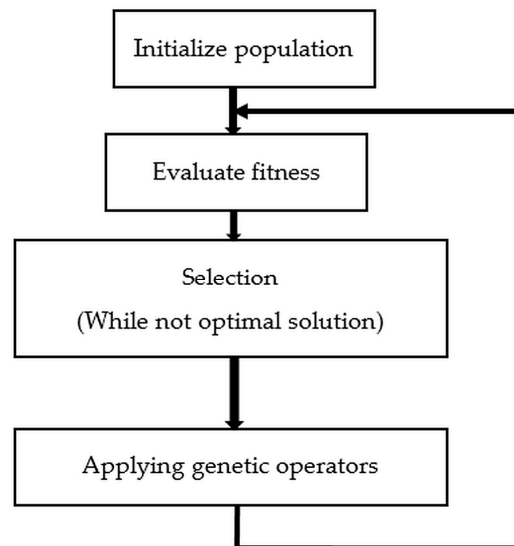


Figure 2. Basic GA for BDD optimization.

Moreover, the way to structure the population of candidate solutions facilitates the introduction of a communication scheme between different groups of chromosomes. In general, the object of communication in GAs working with many subpopulations is the change in chromosomes of good quality between different parts of the population under examination. In this way, a cellular genetic algorithm appears as an intermediate model between the classic GAs and the many-populations, similarity-based, structured, segregation type of algorithm.

Cellular genetic algorithms (CGAs) have a good parallel processing potential—both fine-grained, at the level of cells of the grid, and coarse-grained addressed to the simultaneous processing of multiple grids. At each level, the data communication flows can be controlled and calibrated for a good balance between communication, computing and the convergence speed of the algorithm. Examples of CGAs for various optimization problems can be found in [21–23].

In the next subsection, we employ the cellular genetic algorithm model introduced in [23] to address the optimization of variable ordering in BDDs. Its main components are summarized below.

4.1. Constructing the Feature Space

The feature space is constructed by associating a vector $\phi(x) \in \mathbb{R}^n$ with each chromosome, $x \in S$. The permutation space S is the set of permutations of n binary variables defining the binary function(s). The vector $\phi(x)$ is named the feature vector of x . This mapping $\phi: S \rightarrow \mathbb{R}^n$ should ensure that similar solutions x and x' are mapped in similar vectors $\phi(x)$ and $\phi(x')$ in $\mathbb{R}^n$, where the Euclidian distance is taken as a similarity measure in $\mathbb{R}^n$. The construction of ϕ is discussed below.

Consider a prefix $x = (x_1, \dots, x_k)$ of an ordering of n variables. Let $x' = (x_1, \dots, x_k, x'_{k+1}, \dots, x'_n)$ be an extension of x to a complete ordering. Let $L_j(x')$ be the level j in the OBDD built on x' . If x'' is another extension of x to a complete ordering, then from [16] it results that $L_j(x') = L_j(x'')$, $j = 1, \dots, k$. Based on this result, the mapping $\phi: S \rightarrow \mathbb{R}^n$ is defined as follows:

If $k = n$ then $\phi(x) = (\tau_1, \dots, \tau_n)$, with $\tau_j = \text{card}(L_j(x))$, $j = 1, \dots, n$.

If $k < n$ then $\phi(x) = (\tau_1, \dots, \tau_n)$, with $\tau_j = \text{card}(L_j(x))$, $j = 1, \dots, k$, and $\tau_j = 1$, $j = k + 1, \dots, n$.

The mapping $\phi: S \rightarrow R^n$ is not injective, since distinct permutations may yield identical level-cardinality vectors, particularly for functions exhibiting structural symmetries. Likewise, the Euclidean distance $\|\phi(x) - \phi(x')\|^2$ does not preserve permutation metrics such as Kendall's tau. Instead, ϕ serves as a stability-preserving surrogate representation: small changes in the prefix of a permutation typically induce proportionally small variations in the early components of $\phi(x)$, consistent with the prefix-invariance property of OBDD levels. Consequently, the mapping provides an embedding that captures coarse structural similarity between orderings, yielding a feature space sufficiently smooth for topology-preserving clustering methods such as Kohonen self-organizing maps, which rely on local continuity rather than strict metric preservation. This phenomenological grouping influences the behavior of the fitness landscape across the cellular genetic algorithm grid, rather than enforcing strict sequence-based isomorphism.

4.2. Population Management

The chromosome is an ordering $x = (x_1, \dots, x_n)$ of the n variables of the BDD. The support of the chromosomes and predators is a bidimensional grid. This grid is represented by a matrix $\Gamma = \gamma_{ij}$ of $N \times N$ locations. Each location (i, j) can be the place either for a chromosome (prey) or for a predator. Initially, all positions in the grid are empty. The grid is sized so that $ps < 0.6 N^2$, where ps denotes the population size.

As mentioned above, for each chromosome $x = (x_1, \dots, x_n)$, $\phi(x)$ is its feature vector defined in Section 4.1.

The procedure for mapping of the initial population onto Γ is described below.

Mapping the initial population

1. An initial population $|P|$ of ps embryos is randomly generated.
2. Compute the image $F = \phi(P)$.
3. Apply the Kohonen algorithm to map F onto the current grid Γ , with an 8-connected mesh of $N \times N$ neurons, using $c_0 = ps/20$ clusters. Let $c \leq c_0$ be the resulting number of nonempty clusters $K_1, \dots, K_c$.
4. Consider the clusters in descending order of their cardinality. For each cluster K_j do:
 - 4.1 Sort its members in increasing order of their distance from the centroid of the cluster.
 - 4.2 Place them starting from the position (x_j, y_j) associated with the centroid and extending to new positions using the *burning prairie* rule based on the 8-connected connectivity pattern of Γ . If a position is not free then this position becomes free by putting the occupant chromosome in a waiting list L for later placement.
5. The chromosomes in L are randomly placed on the remaining free positions in Γ .

This procedure favors the creation of similarity-based clusters in contiguous areas of Γ . Another way to populate a grid is to randomly assign positions for each member of the initial population. These two procedures can be used in prescribed proportions to populate the grid.

Further, if x is the chromosome, then $\text{pos}(x)$ is the pair of coordinates (i, j) where x is placed on Γ and γ_{ij} is the chromosome itself or its address. In order to simplify the management of the grid space all coordinates are taken as modulo N .

The grid Γ is also populated with predators that eliminate poor performance solutions. If R denotes the number of predators, then $R \approx 0.25 \cdot ps$. They are randomly placed into unoccupied cells following the initialization of the population. The case $N = 9$ is illustrated in Figure 3.

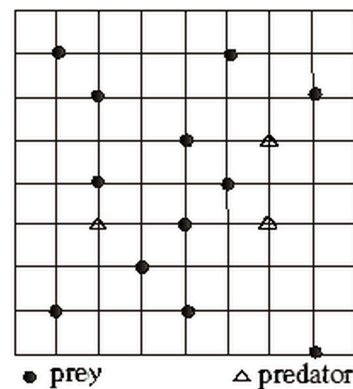


Figure 3. Two-dimensional grid with 11 preys (●) and 3 predators (Δ).

The fitness value $fit(x)$ associated with a given chromosome x represents the number of nodes in the OBDD built on the ordering given by x . The evolution is organized into stages. During each stage, the current population supports mutation, hypermutation, crossover, and devouring.

The parameter $R \approx 0.25 p_s$ regulates the selective pressure exerted by the predators and governs the exploration–exploitation balance. We have $R = \delta p_s$, where R is the number of predators, p_s is the population size, and δ is the predator ratio; its calibration is discussed in Section 5.

4.3. Genetic Operators on the Grid

The genetic operators used by CGA are exactly those which gave the best results from [8]. The parameters values are set empirically. In [8], an extensive experimental evaluation of the parameters is conducted to analyze the impact of different parameter configurations, and the values reported in performance evaluation were also adopted in [7], which constitutes the baseline method used for comparison. For the embryonic GA, only two crossover operators are employed: the alternate crossover from [8] and a restricted variant of itself. The specific difference is the manner of placing the resulted chromosomes on the grid. The probabilities to apply all these operators follow exactly the same rules used for the hybridized algorithm described in [24].

Mutation and hypermutation.

Three mutation operators are used: mutual exchange, group mutation, and inversion. Regardless of which mutation operator μ is chosen, the distance d between the cutting points linearly decreases with the number of generations. If mutation acts on a grid Γ , the distance between x and $y = \mu(x)$ is proportional to d . At the beginning of the evolution d is large, μ produces important disruption in x , and y is placed far from x because it is expected to be less similar to x . To the end of evolution, the disruption produced by mutation is small so a natural similarity-based aggregation of chromosomes is produced.

The shifting heuristic [5] incorporated by hypermutation uses a window $[x_r, \dots, x_s]$, $1 \leq r < s \leq n$, in the chromosome $x = (x_1, \dots, x_n)$ whose width $s - r$ is used to determine the place in which to put the result $h(x)$ on the grid. Namely, if x is the argument and $(i, j) = \text{pos}(x)$ is its position, then one starts from (i, j) and an attempt is made to place $h(x)$ using no more than $d = q(s - r)$ steps using enlarging squares centered in (i, j) , i.e., the first 8 steps could put $h(x)$ in some free node in $V((i, j), 1)$, the next 16 steps in $V((i, j), 2)$, the next 24 steps in $V((i, j), 3)$, and so on as shown in Figure 4. The parameter q is empirically set to a value that guarantees the success in $q(s - r)$ attempts and it is as small as possible.

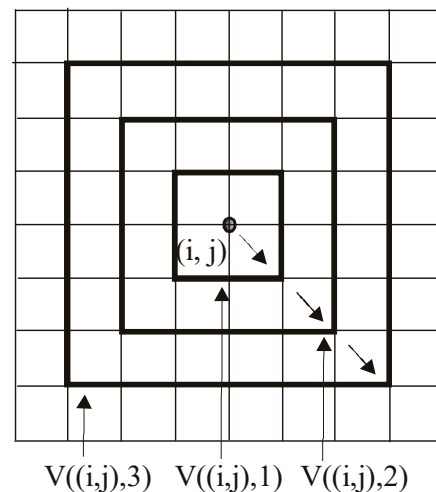


Figure 4. Enlarged squares showing successive wavefronts for finding a free position for the hypermutation results.

Crossover. Let x and x' be the parents involved in crossover operation, and $(i, j) = \text{pos}(x)$, $(i', j') = \text{pos}(x')$ their positions, respectively. Let x^- and x'^- be the offspring that got the prefix delimited by the first cutting point from x and x' , respectively, while the remaining positions come from the other parent. Then one needs to place x^- and x'^- in two free positions, so that they are close to $1/4\text{pos}(x) + 3/4\text{pos}(x')$ and $1/4\text{pos}(x') + 3/4\text{pos}(x)$, respectively. The technique of enlarging squared used for (hyper)mutation is applied for this. Figure 5 illustrates the placing of the offspring on the grid.

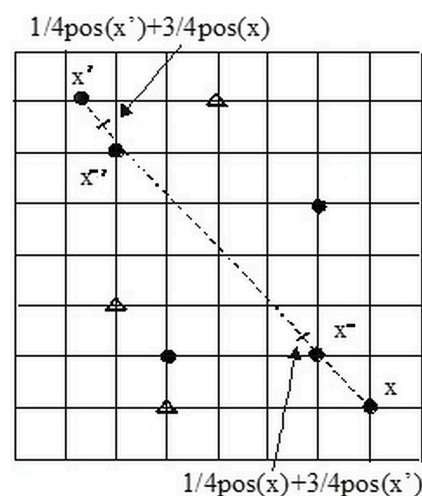


Figure 5. Positioning the offspring x^- and x'^- near their parents.

4.4. Survival Selection

The survival selection is done through devouring that is accomplished by the predators. At the end of each evolution stage, they move on the grid in order to eliminate the poor performance chromosomes met on their route. Different movement rules for the predators have been proposed and evaluated in [23]. This approach uses the following scheme.

Devouring rule.

1. The set of all prey are organized in K groups $C_1, \dots, C_K$ (for example, with the K-means algorithm) based on the set of bidimensional vectors of their positions. Let $x_1^-, \dots, x_K^-$ be the resulting centroids that are not necessarily points on Γ .
2. For each predator, consider its position (i, j) and try to find preys in its smallest vicinity $V((i, j), 1)$. If they exist, the predator devours the poorest one and takes its position

(local devouring). If no prey exists near (i, j) , it moves toward the closest centroid x_h^- using the edges of the grid. Whenever the predator reaches a new position (i', j') on such a path, it repeats the local devouring attempt. This moving continues while no prey is found along the path from (i, j) to x_h^- . If the movement ends with no devouring than the target becomes an arbitrary prey belonging to the cluster C_h and that is closest to x_h^- .

This type of devouring has the advantage of reducing the number of devouring attempts and realizing a balance between the survival selection applied to concentrated preys and that used for relatively isolated preys. This devouring scheme is illustrated in Figure 6 for $K = 2$.

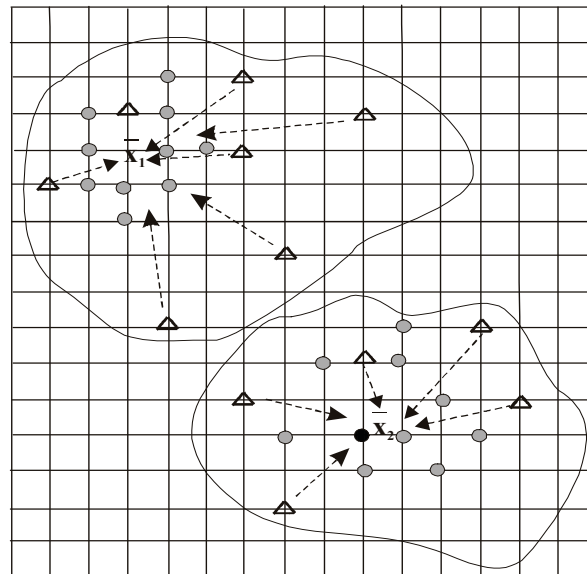


Figure 6. Driving the predators among groups of preys.

The clustering algorithm is applied to each r evolution stage. Between two successive applications, the current clustering solution is maintained by putting the results of mutation and hypermutation in the cluster to which the parents belong, while for crossover each offspring goes to the cluster of its dominant parent, because the parents could be from different clusters. Any change to the content of a cluster produces the updating of its centroid.

4.5. Saving the Best Temporary Solutions

In order to ensure that the best found solution is stored for final use, a thesaurus is included. At the end of each evaluation stage, the best found adult solution that is devoured is stored in the thesaurus. This prevents the loss of a best found solution when it is the unique chromosome in the closest vicinity of a predator. This component is not used for embryos. The thesaurus can be set to keep a prescribed number of different best found solutions.

4.6. Stopping Condition

The run of the basic CGA acting on the grid is stopped when the average of the fitness of chromosomes in the current population stagnates for a given number of successive evolution stages. The best results can be extracted from the grid using its final content and the attached thesaurus.

4.7. Formal Complexity Analysis

Introducing K-means clustering and predator movement requires evaluation of the computational overhead of these mechanisms relative to the time required for BDD construction. We show that the cost of the additional components is negligible compared to the construction of OBDDs.

In each generation, K-means clustering is applied to a population of size ps , where the individual is represented by an n -dimensional feature vector. For G generations, K clusters and maximum of I iterations, the computational complexity is $O(G \cdot I \cdot K \cdot ps \cdot n)$. Because both the population size ps and the number of variables n for LGSynth benchmarks are relatively small, and because I typically converges in fewer than 10–15 iterations, this step is highly efficient, requiring less than 1 ms per generation. The movement of a predator towards a cluster centroid on the 2D cellular grid requires computing the Euclidean distance to the nearest target. For P predators, this results in a simple coordinate update of complexity $O(P)$ per generation, yielding a total lifetime complexity of $O(G \cdot P)$.

In every generation, however, the fitness of the population must be evaluated, which requires execution of BDD operations for the chromosomes. The total cost of these evaluations scales as $O(G \cdot ps \cdot \text{Cost}_{\text{OBDD}})$. Since BDD manipulation may scale exponentially with circuit complexity, we formally have:

$$\text{Cost}_{\text{OBDD}} \gg I \cdot K \cdot n$$

5. Experimental Evaluation of Performance

In this section, the performance of the single-grid cellular genetic algorithm is experimentally investigated on a subset of the well-known LGSynth91 benchmarks [25] in .blif format which represents Boolean functions using sum-of-products expressions. The LGSynth91 instances are grouped into three categories, according to their size: small—cm85a, cm163a, cu, alu4, s1494, vda, misex3; medium—apex2, apex7, cordic, tt2; and large—i3, apex6. The optimal solution values are known for almost all instances. BDD manipulation was performed using the CUDD 3.0 package [26] and C language (GCC compiler). The tests were executed on an Intel Core i7 dual core, 16 GB RAM Ubuntu Linux machine with 1 TB SSD NVMe and 24 GB swap. For each circuit, 30 runs were conducted.

As mentioned in Section 4.3, we used the same parameter setting as in [7,8]. The probabilities for applying the genetic operators follow the rules described in [24].

For each test instance, the following indicators are reported in Table 1: the number of input and output variables of the Boolean function (*In/Out*), respectively; best known solution from the literature (*bk*)—taken from [7] which used the same subset of benchmarks, best found solution in all 30 runs (*bf*), obtained by SGCGA, the corresponding number of iterations (*iter_bf*) in the run which gave the best found solution (*bf*), and the time (*time_bf*) of that run.

SGCGA finds the optimal solution in all runs for every small instance, for which the optimal value is known. This also holds for one of the large instances (i3). For the second largest, and most challenging instance, apex6, the results remain very close to the optimal value. The DSGA method used in [7] gives 303 as the best found solution for apex 2, and 214 as the best solution for apex 7. The SGCGA result of 302 for apex2 represents a one-node improvement over the best result of 303 reported in [7].

Table 2 lists the average number of iterations (*iter_avg*) made until the algorithm stops and the average number of objective function evaluations (*f_avg*). The last three columns indicate m , σ , and σ/m , computed over the 30 runs, where:

$$\text{bf} = \min_i f_i,$$

$$m = (1/30) \sum_{i=1}^{30} f_i,$$

where f_i is the best node count obtained in run i , and

$$\sigma = \text{SD}(f_1, \dots, f_{30}),$$

where σ is the population standard deviation, and the coefficient of variation is (σ/m)

Table 1. Best known solutions (bk) [7], best found solutions (bf), and computational costs (iter_bf, time_bf) for the run where the best found solution for SGCGA was obtained.

Bench	In/Out	bk [7]	bf	Iter_bf	Time_bf
cm85a	11/3	28	28	42	8.2
cm163a	16/5	26	26	40	7.1
cu	14/11	32	32	41	7.9
alu4	14/8	350	350	87	320.8
s1494	14/25	369	369	51	247.0
vda	17/39	478	478	96	12.0
misex3	14/14	478	478	85	288.9
cordic	23/2	42	42	113	403.0
ttt2	24/21	107	107	72	508.3
apex2	39/3	-	302	250	1389.2
apex7	49/37	-	214	303	1590.2
apex6	135/99	498	504	524	2809.1
i3	132/6	133	133	463	4544.2

Table 2. Averaged performance and costs for single-grid SGCGA.

Bench	Iter_avg	f_avg	m	σ	σ/m
cm85a	52.6	1217.7	28	0	-
cm163a	51.6	1311.12	26	0	-
cu	50.0	1323.00	32	0	-
alu4	70.3	1847.90	350	0	-
s1494	61.2	1978.37	369	0	-
vda	98.5	1660.80	478	0	-
misex3	97.5	2312.40	478	0	-
cordic	114.1	2648.17	42.300	0.526	0.012
ttt2	96.3	2276.69	108.033	1.722	0.016
apex2	369.1	9613.63	303.800	2.638	0.087
apex7	276.2	7078.60	216.833	3.795	0.017
apex6	446.3	9502.34	505.700	3.532	0.007
i3	466.4	12,435.58	133	0	-

For many benchmarks (several small and medium instances), m is equal to bf and $\sigma = 0$ for multiple cases, indicating highly repeatable performance in those instances. For larger or harder instances, the method shows more variability (for example, apex7:

$m = 216.833$, $\sigma = 3.795$, apex 6: $m = 505.700$, $\sigma = 3.532$), so stability is instance-dependent rather than universal.

We applied the two-sided Wilcoxon signed-rank test to the 30 SGCGA node-count values for apex2, using the best result reported in [7] ($bk = 303$) as the reference. The null hypothesis stated that the median SGCGA node count equals this baseline value. The test yielded a signed-rank statistic of $W = 142.5$, with $Z = -0.84$ under the normal approximation and a two-sided p -value of 0.40. The effect size was $r = -0.165$, indicating a small deviation. Because the p -value is non-significant, the data do not provide sufficient evidence to reject the null hypothesis. Under the same conditions, for apex6 ($bk = 498$), the test yielded a signed-rank statistic of $W = 465$, corresponding to $Z = 4.78$ under the normal approximation and a two-sided p -value of 1.7×10^{-6} . The effect size was $r = 0.872$, representing a large deviation, and therefore the Wilcoxon test provides strong evidence that SGCGA performs worse than the best known solution on apex6.

Comparison with other heuristic methods.

Table 3 present a comparison with several classic approaches: the sifting heuristic [6]; the genetic algorithm described in [27]; fault-simulation-based ordering [28]; level and fan-in heuristics [29]; and the fan-out heuristic [30]. The results show that SGCGA performs at well as or better than these methods across the evaluated benchmarks.

Table 3. Comparison with other heuristics.

Bench	Sift [6]	GA [27]	SGCGA	Bench	Sift [6]	GA [27]	SGCGA
cm85a	36	28	28	alu4	602	350	350
cm163a	28	26	26	s1494	388	370	369
cu	35	32	32	apex7	320	214	214
bench	fsim [28]	lvl [29]	fin [30]	fout [30]	SGCGA		
vda	533	2733	515	533	478		
tt2	234	163	170	186	107		

Table 4 presents the comparison with the two other closely related multi-population approaches: CGA and SSGA. The CGA and SSGA methods were originally introduced in [31] and [32], respectively. The numerical comparison values reported in Table 4 are reproduced or converted from Table 4 of [7]. The authors [31,32] reported the following indicators: best found solution (bf), average (m), and standard deviation (σ).

Table 4. Comparison with similar heuristics: CGA and SSGA.

Method	CGA			SSGA			SGCGA		
bench	bf	m	σ	bf	m	σ	bf	m	σ
apex2	309	318.6	7.2	304	315	3.6	302	303.800	2.638
apex7	214	225.3	11	214	227	3.5	214	216.833	3.795
cordic	42	42.1	0.3	42	43.3	1.0	42	42.300	0.526
tt2	107	107.4	0.8	107	108.5	1.1	107	108.033	1.722
apex6	508	546.6	13	536	591.9	9.6	504	505.700	3.532
i3	133	133	0	138	147.7	3.1	133	133	0

The CGA represents an intermediate design between the single-population genetic algorithm and the segregative method. Its added value lies in the communication among parallel simultaneously evolving populations, which substantially enhances solution qual-

ity [7]. SSGA is an early version of DSGA, sharing the same primary objective: a systematic exploration of the search space that emphasizes solution stability and aims to improve quality by exploring multiple local optima. The results confirm that a simple single-grid cellular GA based on predator–prey techniques can produce solutions that match or even surpass those obtained by more complex computational methods. Compared with the selected heuristics in Tables 3 and 4, SGCGA matches or improves the best reported node counts on several benchmarks (notably apex2), and attains comparable best found values on many others. However, performance differences vary by instance: on apex6 SGCGA's best find (504) is worse than some baselines, and variability measures differ across methods. We therefore report both best found (bf) and per-run statistics (m , σ) to allow a balanced comparison.

Some qualitative aspects of the workings of SGCGA for optimizing the variable order of BDDs are illustrated in the sequel. For the cu.blif instance, the variation in the best chromosome fitness and the average population fitness during the evolution is shown in Figure 7.

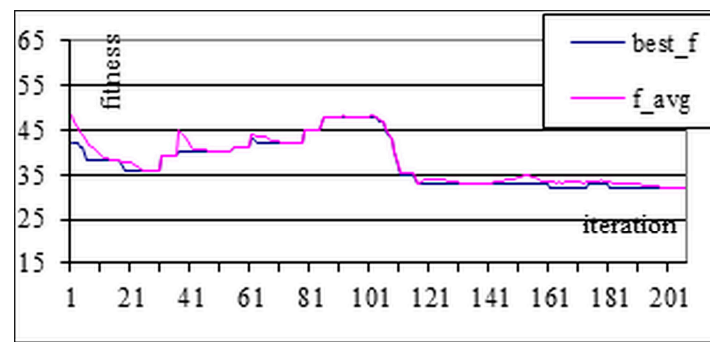


Figure 7. Best fitness (best_f) and average fitness (f_avg) evolution during an SGCGA run for the cu.blif benchmark.

In the first 100 stages, the fitness of the best embryo has a high variability. Starting from stage 110, the GA acts as a pure GA focusing on the exploitation stage and the fitness of the best chromosome is slowly improved until the best known number of nodes is obtained (32 for cu.blif). For the same benchmark, the fitness values of the solutions saved in the thesaurus, i.e., bf_i (inspected at every 10 stages), are shown in Figure 8a for successive stages sending something to this component, and Figure 8b including the inspected stages elapsed between two consecutive inputs in the thesaurus.

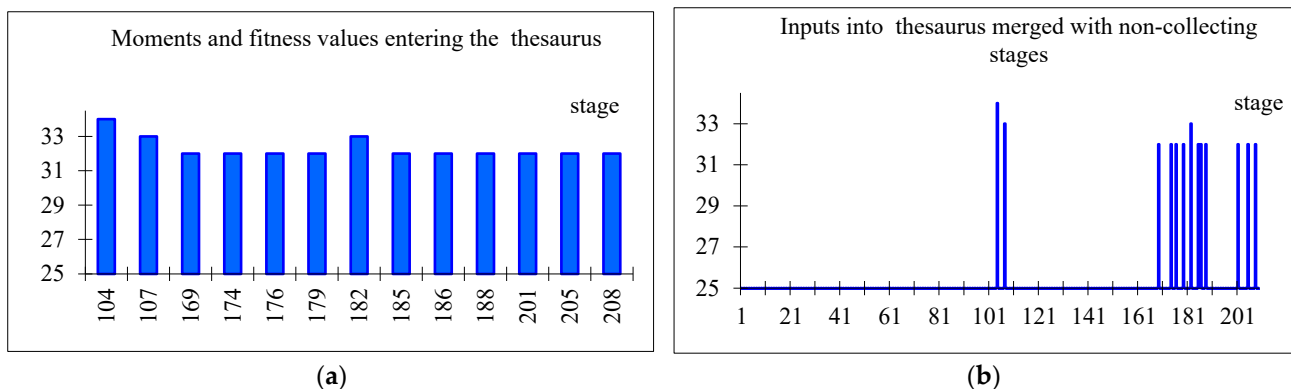


Figure 8. Fitness values in thesaurus for cu.blif instance, inspected at every 10 stages (a) and fitness of solutions entering thesaurus merged with non-collecting stages (b).

On the other hand, the number of clusters varies along the evolution as shown in Figure 9 for the cu.blif instance.

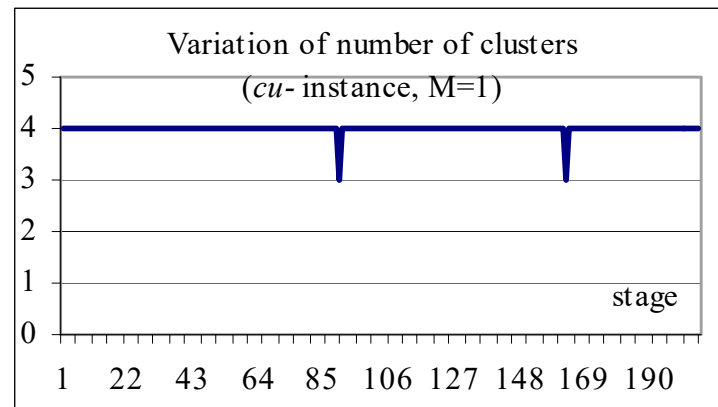


Figure 9. Dynamics of the number of clusters during the evolution stages for the cu.blif benchmark.

The grid contents corresponding to four different moments of the evolution for cu.blif are shown in Figure 10.

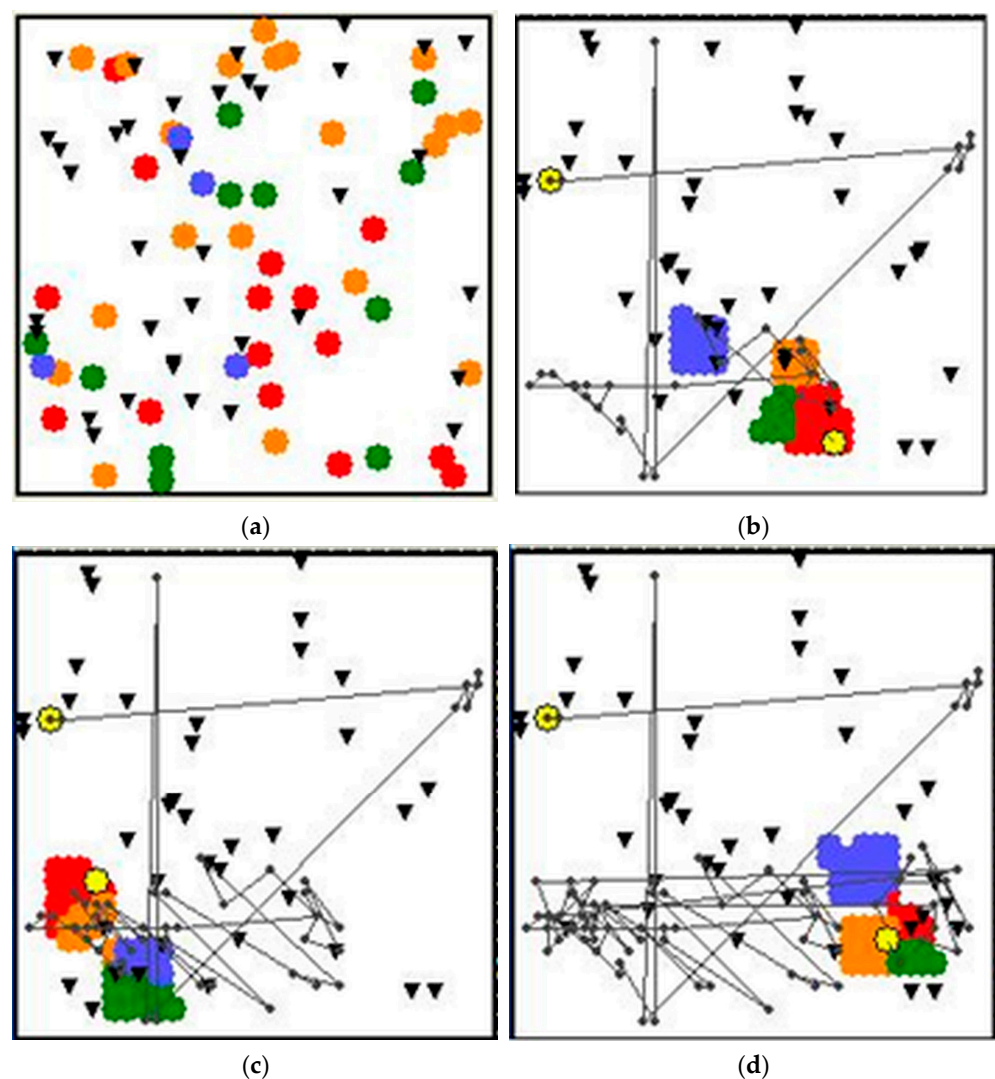


Figure 10. Four moments in the evolution of one grid: (a) beginning of the evolution, (b) third of the entire evolution time, (c) two thirds of the time, (d) final stage.

Preys having the same color belong to the same cluster. A triangle denotes a predator on that position. The gray polygonal line in Figure 10d illustrates successive positions of the best found solution on the grid. A yellow circle inside a gray circle labels the starting point, whilst the ending point is a yellow disc with no outgoing segment. The capture from Figure 10d shows a compact arrangement of preys, when the clusters are dense and the isolated individuals have disappeared. The predators are represented by black triangles.

Setting R-ratio. As noted in Section 4.2, the parameter $R \approx 0.25 p_s$, where p_s denotes population size, governs the extinction rate applied by the predators within the grid and directly controls the exploration–exploitation trade-off. The number of predators is computed as $R = \delta p_s$, where δ is the predator ratio. Table 5 presents its empirical tuning for apex7.

Table 5. Setting the predator ratio δ for apex7.

Predator Ratio δ	#Nodes	#Iter
0.05	258	545
0.10	241	480
0.15	230	412
0.20	222	356
0.25	215	303
0.30	216	362
0.35	228	340
0.40	239	315
0.45	251	280
0.50	274	242

When δ is set to a lower threshold ($\delta \leq 0.10$), the selective pressure becomes insufficient, leading to structural stagnation and a substantially higher number of generations (approximately 500 iterations). Conversely, if the ratio is overly aggressive ($\delta \geq 0.40$), the framework suffers from over-exploitation: elite variable orderings are eliminated prematurely, reducing the iteration count but producing sub-optimal graphs.

Additional tests show that for smaller circuits, $R \approx 0.25 p_s$ ensures fast convergence due to a limited search space. At larger circuit scales where BDD size explosions are severe, this ratio enables SGCGA to actively eliminate poor structural topologies while keeping 75% of the population grid available for further exploration.

6. Conclusions and Future Work

The paper presents a new cellular genetic algorithm designed for optimizing the variable ordering of BDDs. This tool is based on the components of the double hybridized genetic algorithm presented in [8]. The experimental evaluation shows that SGCGA produces solution quality comparable to prior GA-based methods on the tested LGSynth91 subset, with one node improvement (apex2) and mixed results on the largest instance (apex6). These outcomes indicate that SGCGA does not consistently outperform existing methods but provides a viable alternative heuristic with reliable behavior across multiple benchmarks. The method’s grid and predator–prey components appear promising for preserving diversity and guiding search, but further benchmarking against modern state-of-the-art solvers and additional statistical analysis are required before claiming any superiority. If an appropriate communication protocol is defined, the single-grid GA could be extended to multi-grids in an attempt to improve the quality of the solutions [7]. Concep-

tually, SGCGA is an intermediate design between (hybrid) GA with one population and the segregation type of GA. Contrary to a segregation-type GA that uses clustering algorithms based on similarity for multiple population management, the cellular variant manages different groups using positional information that does not depend on some computed features. The actual design of SGCGA could be improved using an embryonic representation of the solutions and different growing strategies. Moreover, the proposed SGCGA can easily incorporate different GAs. To further validate SGCGA's efficiency, it must be benchmarked against at least one modern SOTA solver or a deep-learning-based heuristic. This limitation, inherent to the current proof-of-concept stage of our method, will be tackled in future work through the development of a new dynamic cellular algorithm. An exciting and highly relevant direction for future research lies in scaling the proposed framework to meet the demands of next-generation communication paradigms like, for example, vehicular and aerial communications toward 6G. Although the current evaluation is restricted to standard 2D BDD optimization for Boolean benchmarks, future dynamic environments will require BDD-based hardware verification and resource allocation to operate within high-dimensional state spaces. A unified channel model-driven optimization framework becomes essential for maintaining system-wide stability. Our proposed SGCGA can be extended to a 3D cellular grid tensor. Within a unified channel model-driven optimization framework [33], this 3D expansion would enable the K-means clustering and predator–prey dynamics to operate across multidimensional state boundaries—for example, simultaneously accounting for spatial distribution, altitude variation, and channel fluctuations. This structural adaptation would allow the algorithm to maintain system-wide stability and prevent decision-diagram explosions when verifying massive, real-time routing logic in 6G vehicular and multi-mode unmanned aerial vehicle (UAV) networks [34,35].

Author Contributions: Conceptualization, I.F. and C.T.; methodology, I.F., C.T. and B.P.; software, I.F. and C.T.; validation I.F.; formal analysis, I.F. and B.P.; resources, C.T.; writing—review and editing, I.F. and B.P. All authors have read and agreed to the published version of the manuscript.

Funding: The research was funded by the Ministry of Education, through the National Council for the Financing of Higher Education, Romania, grant number CNFIS-FDI-2025-F-0483: Improving the quality of RDI activities at “Vasile Alecsandri” University of Bacău through national and international cooperation (acronym: Connect-UBc).

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bryant, R.E. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Comput.* **1986**, *35*, 667–691.
2. Bollig, B.; Wegener, I. Improving the Variable Ordering of OBDDs Is NP-Complete. *IEEE Trans. Comput.* **1996**, *45*, 993–1002. [\[CrossRef\]](#)
3. Fujii, H.; Ootomo, G.; Hori, C. Interleaving based variable ordering methods for OBDD. In *Proceedings of the International Conference on Computer-Aided Design, Santa Clara, CA, USA, 7–11 November 1993*; ACM: New York, NY, USA, 1993; pp. 38–41.
4. Meinel, C.; Slobodova, A. Speeding up variable reordering for OBDDs. In *Proceedings of the International Conference on Computer Design, Washington, DC, USA, 12–15 October 1997*; IEEE: New York, NY, USA, 1997; pp. 338–343.
5. Rudell, R. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the International Conference on Computer-Aided Design, Santa Clara, CA, USA, 7–11 November 1993*; IEEE: New York, NY, USA, 1993; pp. 42–47.
6. Meinel, C.; Somenzi, F.; Theobald, T. Linear sifting of decision diagrams. In *Proceedings of the 34th ACM/IEEE Design Automation Conference, Anaheim, CA, USA, 9–13 June 1997*; ACM: New York, NY, USA, 1997; pp. 202–207.
7. Rotaru, C.; Brudaru, O. Dynamic Segregative Genetic Algorithm for Optimizing the Variable Ordering of ROBDDs. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO'12)*; ACM: New York, NY, USA, 2012; pp. 1111–1118. [\[CrossRef\]](#)

8. Furdu, I.; Brudaru, O. New hybrid genetic algorithm with adaptive operators and variability target for optimizing variable order in OBDD. *Sci. Stud. Res.* **2009**, *19*, 156–172.
9. Lenders, W.; Baier, C. Genetic Algorithms for Variable Ordering Problem of Binary Decision Diagrams. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3469, pp. 1–20.
10. Dubsclaff, C.; Husung, N.; Käfer, N. Tailoring binary decision diagram compilation for feature models. *J. Syst. Softw.* **2026**, *231*, 112566. [\[CrossRef\]](#)
11. Santos, P.G.; Ruas, P.H.B.; Neves, J.C.V.; Silva, P.R.; Dias, S.M.; Zárate, L.E.; Song, M.A.J. ImplicPBDD: A New Approach to Extract Proper Implications Set from High-Dimension Formal Contexts Using a Binary Decision Diagram. *Information* **2018**, *9*, 266. [\[CrossRef\]](#)
12. Patel, R.; Khalil, E.; Bergman, D. Heuristic Multiobjective Discrete Optimization using Restricted Decision Diagrams. *arXiv* **2026**. [\[CrossRef\]](#)
13. Patel, R.; Khalil, E. LEO: Learning Efficient Orderings for Multiobjective BDDs. 2023. Available online: <https://www.catalyze.com/paper/leo-learning-efficient-orderings-for> (accessed on 25 June 2026).
14. Krauss, R.; Drechsler, R. Efficient Evolution of Variable Ordering for Binary Decision Diagram Optimization. *IEEE Trans. Evol. Comput.* **2025**, *30*, 762–772. [\[CrossRef\]](#)
15. Awad, A.; Hawash, A.; Abdalhaq, B. A Genetic Algorithm (GA) and Swarm-Based Binary Decision Diagram (BDD) Reordering Optimizer Reinforced With Recent Operators. *IEEE Trans. Evol. Comput.* **2023**, *27*, 535–549. [\[CrossRef\]](#)
16. Tani, S. Quantum algorithm for finding the optimal variable ordering for binary decision diagrams. *Theor. Comput. Sci.* **2025**, *1041*, 115230. [\[CrossRef\]](#)
17. Friedman, S.J.; Supowit, K.J. Finding the Optimal Variable Ordering for Binary Decision Diagrams. *IEEE Trans. Comput.* **1990**, *39*, 710–713. [\[CrossRef\]](#)
18. Xu, F.; He, F.; Xie, E.; Li, L. Fast OBDD Reordering using Neural Message Passing on Hypergraph. *arXiv* **2018**. [\[CrossRef\]](#)
19. Zhao, H.; Wang, B.; Fu, Y.; Li, N.; Gao, Z. Fixed-time adaptive fault-tolerant control of a multi-mode VTOL UAV with variable prescribed performance boundaries under random disturbances. *ISA Trans.* **2026**, *175*, 126–138. [\[CrossRef\]](#) [\[PubMed\]](#)
20. García Márquez, F.P.; Segovia Ramírez, I.; Mohammadi-Ivatloo, B.; Marugán, A.P. Reliability Dynamic Analysis by Fault Trees and Binary Decision Diagrams. *Information* **2020**, *11*, 324. [\[CrossRef\]](#)
21. Salto, C.; Alba, E. Cellular Genetic Algorithms: Understanding the Behavior of Using Neighborhoods. *Appl. Artif. Intell.* **2019**, *33*, 863–880. [\[CrossRef\]](#)
22. Dorronsoro, B.; Alba, E. *Cellular Genetic Algorithms*; Book series: Operations Research/Computer Science Interfaces Series; Springer: Berlin/Heidelberg, Germany, 2008.
23. Brudaru, O.; Popovici, D.; Copăceanu, C. Cellular genetic algorithm with communicating grids for assembly line balancing problems. *Adv. Electr. Comput. Eng.* **2010**, *10*, 87–93. [\[CrossRef\]](#)
24. Brudaru, O.; Furdu, I.; Ebendt, R. *Embryonic Genetic Algorithm with Random Generational Growing Strategy for Optimizing Variable Ordering of BDDs*; Series Mathematics and Informatics; Scientific Studies and Research: Bacău, Romania, 2010; Volume 20.
25. LGSynth91 Benchmarks. Available online: <https://ddd.fit.cvut.cz/www/prj/Benchmarks/> (accessed on 25 June 2026).
26. CUDD Package. Available online: https://github.com/davidkebo/cudd/tree/main/cudd_versions (accessed on 25 June 2026).
27. Drechsler, R.; Becker, B.; Göckel, N. A genetic algorithm for variable ordering of OBDDs. *IEEE Proc.* **1996**, *143*, 363–368. [\[CrossRef\]](#)
28. Butler, K.M.; Ross, D.; Kapur, R.; Mercer, M.R. Heuristics to compute variable orderings for efficient, manipulation of OBDDs. In *Proceedings of the 28th ACM/IEEE, Design Automation Conference*; Association for Computing Machinery: New York, NY, USA, 1991; pp. 417–420.
29. Malik, S.; Wang, A.R.; Brayton, R.K.; Sangiovanni-Vincentelli, A. Logic verification using BDDs in a logic synthesis environment. In *[1988] IEEE International Conference on Computer-Aided Design (ICCAD-89), Santa Clara, CA, USA, 7–10 November 1988*; IEEE: New York, NY, USA, 2002; pp. 6–9. [\[CrossRef\]](#)
30. Fujita, M.; Fujisawa, H.; Kawato, N. Evaluation and improvements of Boolean comparison method based on BDDs. In *Proceedings of the [1988] IEEE International Conference on Computer-Aided Design (ICCAD-89), Santa Clara, CA, USA, 7–10 November 1988*; IEEE: New York, NY, USA, 2002; pp. 2–5. [\[CrossRef\]](#)
31. Rotaru, C.; Brudaru, O. Multi-grid cellular genetic algorithm for optimizing variable ordering of ROBDDs. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation, Brisbane, Australia, 10–15 June 2012*; IEEE: New York, NY, USA, 2002; pp. 1–8. [\[CrossRef\]](#)
32. Brudaru, O.; Rotaru, C.; Furdu, I. Static segregative genetic algorithm for optimizing variable ordering of ROBDDs. In *Proceedings of the 13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*; IEEE: Piscataway, NJ, USA, 2011; pp. 222–229.
33. Ma, Z.; Lin, Y.; Hua, B.; Mao, K.; Zeng, L.; Lian, Z. *SIM-Empowered LAINs: A Unified Channel Model-Driven Optimization Framework*; IEEE Wireless Communications: Piscataway, NJ, USA, 2026. [\[CrossRef\]](#)

34. Jiang, H.; Pan, X.; Shi, W.; Zeng, L.; Chen, Z.; Shu, F.; Wang, J. Deep Reinforcement Learning Enabled UAV Trajectory Optimization for A2G Communication Systems. *IEEE Trans. Cogn. Commun. Netw.* **2026**, *12*, 3164–3178. [[CrossRef](#)]
35. Ma, Z.; Lu, S.; Peng, Y.; Zhou, J.; Xu, J.; Luo, G.; Luo, M. Geometrical-Based Modeling for Aerial Intelligent Reflecting Surface-Based MIMO Channels. *Electronics* **2026**, *15*, 875. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.