Static segregative genetic algorithm for optimizing variable ordering of ROBDDs

Octav Brudaru, Cristian Rotaru, Iulian Furdu

Abstract — This paper presents a segregative genetic algorithm for optimizing the variable order in Reduced Ordered Binary Decision Diagrams. The main components are a hybrid basic genetic algorithm and two feature functions used to measure the similarity between chromosomes. The clustering of the feature space induces subpopulations in the search space. The initial population is partitioned by a simple algorithm. The basic genetic algorithm explores each subpopulation. Depending on the available computational resources a number of copies of the basic genetic algorithm run in parallel. A communication protocol preserves the similarity within each subpopulation during the evolution process. The approach introduces an associative tabu search memory in order to avoid reexploration of the search space. High quality individuals from unexplored parts of search space are included in the clusters where they fit best. The segregative genetic algorithm offers a better trade-off between exploration, achieved by evolving many clusters, and exploitation, done by the focusing on each subpopulation.

Extensive experimental evaluation proves the efficiency and stability of the segregative approach, which systematically produces better results than the basic genetic algorithm. The efficiency of the distributed implementation in terms of resource usage and many aspects regarding the communication protocol between different components are thoroughly described. The experiments used classical benchmarks known as very difficult and show that the segregative variant is better than the monopopulation algorithm and the approach using the island model.

Keywords — segregative genetic algorithm, associative tabu search, similarity preserving communication, distributed implementation, extensive exploration, intensive exploitation.

I. INTRODUCTION

This paper proposes a new method used to optimize the variable order in Reduced Ordered Binary Decision Diagrams (ROBDDs). Boolean functions are represented as ROBDDs: a directed acyclic graph with respect to an order of the input variables and satisfying a set of properties. The goal is to minimize the size of the ROBDD, which is defined as the number of non-terminal nodes. Some of the applications of ROBDDs can be mentioned in: digital circuit design – for multiplexer based design styles such as Pass Transistor Logic (PTL), a smaller size directly transfers to a smaller chip area; formal verification of combinational circuits; analysis of sequential systems; symbolic model checking.

As the size of a ROBDD strongly depends on the chosen variable order, a difficult problem is to build good variable orders. Existing heuristic methods can be categorized in static and dynamic techniques. Static methods use application specific information (structure of the evaluated combinational circuit, for example) to build good orders before constructing the ROBDD. Examples of such methods can be found in [1]. Dynamic approaches improve the size of an already build ROBDD by modifying the used variable order. Examples of simple methods of this type are described in [2] and [3]. [4] introduces the sifting heuristic, a local search method based on the hill-climbing strategy. The simulated annealing heuristic is used by Bollig, Löbbing and Wegener in [5]. In [6] Drechsler, Becker and Göckel introduce the use of genetic algorithms to optimize the variable order. Following GA based approaches can be found in [7]. Exact methods used to tackle the problem mainly consist of branch-and-bound techniques that use the lemma proved in [8]. Hybrid approaches can be mentioned in [9], where an ordered best-first search with branch-and-bound is presented, and in [10], where a genetic algorithm is hybridized with the sifting heuristic and branch-and-bound techniques.

Papers [11] and [12] report better results by using parallel and distributed genetic algorithms. [13] introduces the idea of segregative algorithm by dividing the whole population into many subpopulations of similar individuals at the beginning of the evolutionary process. The existing subpopulations evolve until their fitness stagnates and in the end a reunification is made. The approach is often superior to other ways to organize the search space, like island model, but its success highly depends on the available similarity measure.

In this paper, a method to optimize the variable order for ROBDDs based on the segregative approach is proposed. Two feature vectors are used to define the similarity between chromosomes and compared between them. The obtained feature space is used to organize the search space in subpopulations. A simple clustering algorithm makes a partitioning of the initial population in an adequate number of clusters. The exploration of each subpopulation is achieved by using the basic genetic algorithm described in [14]. This algorithm is hybridized with the sifting heuristic. Depending on the available computational resources a number of copies
of the basic GA run in parallel. A communication protocol preserves the similarity within each subpopulation. The approach introduces an associative tabu search memory in order to avoid reexploration of the search space. This version is called static because the algorithm explores exclusively the initially created subpopulations and new genetic material is distributed in the clusters where it best fits, contrary to a dynamic approach, which explicitly manages zones of the search space that are not yet explored.

The paper is organized as follows: Section II formally describes the problem of optimizing variable order for ROBDDs; Section III shortly describes the basic genetic algorithm; Section IV details the mapping from genotype to phenotype, the control components, the communication within the system and the activity of the BGA and SGA; Section V presents the experimental evaluation of the proposed algorithm, the activity of the main components and the performance of a distributed implementation; the last section summarizes the work and presents further research directions.

II. THE PROBLEM OF OPTIMIZING THE VARIABLE ORDER FOR ROBDDs

This section formally describes the problem of optimizing the variable order for ROBDDs and shortly underlines the importance the variable ordering.

Let \( F : B^n \rightarrow B^m, B = \{0, 1\} \) be a Boolean function that outputs \( m \) values, and \( \pi \) a permutation of the input variables, denoted by \( v_1, v_2, \ldots, v_n \), \( v_i \in B, i = 1, \ldots, n \). As described in [15], an Ordered BDD (OBDD) for \( F \) with respect to \( \pi \) is a directed acyclic graph with the following properties: i) it has exactly two terminal nodes, labeled with 0 and 1, respectively; ii) each nonterminal node is labeled by a variable \( v_i, 0 < i \leq n \) and has two out-edges labeled with 0 and 1, respectively; iii) the order in which the variables appear on a path in the graph respects the order \( \pi \). The size of the OBDD is defined as the number of nonterminal nodes in the corresponding graph and is the measure that must be minimized.

[15] defines the structure called Reduced OBDD (ROBDD), by applying reduction rules to a OBDD with a fixed variable ordering. In this way the size of the OBDD is further reduced and the obtained structure represents a canonical form for \( F \).

The size of the ROBDD strongly depends on the order of the input variables. [16] shows that optimizing the order is NP-complete and finding the best order is NP-hard.

III. BASIC GENETIC ALGORITHM

In order to better understand the segregative approach, the main features of the basic genetic algorithm proposed in [14] are shortly described in this section.

Each chromosome is a permutation of the input variables of the Boolean function \( F \) and represents a specific ROBDD variable order. A chromosome is denoted by \( x = (x_1, x_2, \ldots, x_n), \ x_i \in \{v_1, \ldots, v_n\}, \ i = 1, \ldots, n \). The fitness of a chromosome \( x \), denoted \( f(x) \), represents the size of the ROBDD with respect to order \( x \).

The initial population of the algorithm is randomly generated. The size of the population (\( pop\_size \)) is directly proportional with the number of variables by an empirically determined rule.

The selection for survival is elitist, meaning that the new individuals compete with the current generation and the best \( pop\_size \) chromosomes, with respect to fitness value, are kept in the next generation.

The crossover operator is a group version of the alternating crossover operator described in [7]. The algorithm employs three classic mutation operators from [17]: mutual exchange, group mutation and inversion. All operators have the property that as the number of generations grows the offspring are more similar to the parents (one of the parents in case of crossover), assuring the convergence of the algorithm. The probabilities to apply crossover and mutation, denoted \( p_{CX} \) and \( p_M \), respectively, are initialized with empirically determined values and are dynamically controlled via a mechanism based on the variability of the population. The variability of the population is computed by using the fitness values of the individuals and is required to match some predefined objective function as best as possible. The probabilities are automatically adjusted using the difference between the value computed for the current population and the value required at the current evolution stage. The used objective function defines the balance between exploration and exploitation.

The genetic algorithm stops when the average fitness of the population stagnates for a prescribed number of generations, denoted \( N_{stop} \). More precisely

\[
|\text{AvgFit}(k) - \text{AvgFit}(k-1)|/\text{AvgFit}(k) < \epsilon_{stop},
\]

where \( \text{AvgFit}(k) \) represents the average fitness of generation \( k \), holds true for \( N_{stop} \) consecutive generations.

IV. THE SEGREGATIVE GENETIC ALGORITHM: STRUCTURE AND ACTIVITY

In this section the main components of the SGA are described and the activities performed by the algorithm are detailed.

A. From genotype to phenotype

The idea behind the segregative approach requires the definition of a notion of similarity between two solution representations used by the basic genetic algorithm. As previously defined, a chromosome represents a permutation of the input variables of the Boolean function \( F \). Such a representation doesn’t support a straightforward definition of a measure of similarity. In order to define the similarity between two individuals, for each chromosome a real valued feature vector is computed.

In this paper two alternatives to compute the feature vector are proposed. Let \( x \) be a chromosome. The first type of feature vector, denoted \( \varphi_1(x) \), captures the structural information of \( x \), as permutation of the input variables.
\( \varphi_1(x) = (y_1, \ldots, y_n), \) where \( y_i = x_i / (n \cdot (n+1)/2) \), \( i = 1, \ldots, n \).
\( \varphi_1(x) \in \mathbb{R}^n \) and \( \sum_{i=1}^{n} y_i = 1 \). The second feature vector, denoted \( \varphi_2(x) \), contains the cost based characteristics of \( x \).
\( \varphi_2(x) = (l_1, \ldots, l_n) \in \mathbb{N}^n \), where \( l_i \) is the number of nodes on the level \( i \) of the graph that represents the ROBDD.

The Euclidean distance in \( \mathbb{R}^n \), \( \|\varphi(x) - \varphi(x')\| \), is used as a measure of similarity between \( x \) and \( x' \), where \( \varphi \in \{\varphi_1, \varphi_2\} \).

B. Subpopulation-based exploration of the search space

In the segregative approach, the search space is explored in parallel by copies of the basic genetic algorithm. Each copy of the BGA evolves a given subpopulation, intensifying the search process in that area of the search space. Depending on available computational resources a given number of subpopulations, denoted \( p \), are explored in parallel at some moment. When a copy of the BGA finishes the exploration of a subpopulation it receives another one, if available. During the evolution process of all available subpopulations, the segregative algorithm collects many local minima from different regions of the search space.

1) Subpopulations initialization

The following procedure is used to generate the subpopulations that will be explored by the segregative algorithm. A (very) large initial population \( P \) is generated randomly and the feature vector \( \varphi(x) \in \{\varphi_1(x), \varphi_2(x)\} \) is computed for each chromosome. The set of images of all chromosomes in the feature space is clustered in order to organize the images of the individuals into a number of similar groups \( G_1, \ldots, G_S \subset \mathbb{R}^n \) with the centroids \( \omega_1, \ldots, \omega_S \in \mathbb{R}^n \).

The number \( S \) is determined by optimizing the Davies-Bouldin index associated with the clustering solution. The subpopulations of individuals are constructed as the preimages \( P_i = \varphi^{-1}(G_i) \subset P, \ i = 1, \ldots, S \), of the clusters of feature vectors in the representation space.

The subpopulations \( P_i, \ i = 1, \ldots, S \) are sent to the component \( Q_U \), defined in the following subsection (B.2.b).

In the evaluation experiments performed to obtain the results presented in this paper, the \( c \)-means clustering algorithm was used because it provides good results in terms of speed and accuracy.

2) Control components

The control components of the segregative algorithm execute different activities that define the behavior of the method. In this paper the following components are described: an associative tabu search memory, a priority queue for subpopulations of individuals, that are not yet explored, and a thesaurus for storing the promising individuals obtained during the evolution process. The communications between these components and the subpopulations under examination are analyzed in a next section.

a) Tabu search associative memory

The segregative algorithm employs a tabu search associative memory \( (TS_A) \), in order to avoid reexploration of regions of the search space.

As previously mentioned, the copies of the BGA explore clusters of individuals, formed over the feature space. These clusters have centroids, which capture the characteristics of the individuals in that cluster. Once a copy of the BGA finishes the exploration of a subpopulation, it sends its centroid to the buffer \( B_{TS}^{in} \) of \( TS_A \). \( TS_A \) receives only the centroid of the subpopulation, it doesn’t effectively store individuals belonging to explored subpopulations, hence the associative characteristic of this memory. \( TS_A \) contains \( \mathbf{S} \) pairs of the form \( (\omega_i, r_i) \), where \( \omega_i \) represents the centroid of some exhausted cluster \( i \) and \( r_i \) is the radius of this cluster.

Initially, the memory is empty, \( \mathbf{S} = 0 \); at the end of the execution \( \mathbf{S} \) will be equal to \( S \), the number of subpopulations obtained by clustering the initial large population \( P \).

The function performed by \( TS_A \) is to decide whether a given chromosome \( x \) belongs to an already explored region of the search space or not. If \( \|\omega_k - \varphi(x)\| \leq r_k \) for some \( k \in \{1, \ldots, S\} \) then it is considered that \( x \) is similar to some already evaluated solutions and it is consequently removed. If not \( x \) will be sent to \( Q_U \).

b) Unexplored subpopulations

In order to manage and prioritize the exploration of the search space, a second component, \( Q_U \), is introduced. \( Q_U \) is a priority queue that stores the not yet explored clusters of individuals. In the initialization phase of the algorithm, \( Q_U \) receives and stores all subpopulations, along with the corresponding cluster centroids and radiiuses, obtained by clustering the initial large population \( P \). Each subpopulation is scored with the average of the fitness values of its members.

The main function of \( Q_U \) is to provide a not yet explored subpopulation to a copy of BGA whenever one becomes free. The best score cluster is extracted from the queue and it is sent to the demanding BGA. Contrary to the \( TS_A \) that manages only centroids of feature vectors, \( Q_U \) also stores chromosomes, which will serve as initial population for the BGA.

Another function of \( Q_U \) is to manage the chromosomes received from \( TS_A \), which are individuals not similar with any subpopulation currently or previously explored. Let \( x \) be such an individual. \( Q_U \) will search the subpopulation \( P_k \), such that \( \|\omega_k - \varphi(x)\| \) is minimized. If the fitness of \( x \) is worse than that of the weakest chromosome in \( P_k \) then \( x \) is discarded. Otherwise \( P_k = P_k \cup \{x\} \).
c) Thesaurus

The last component used by the segregative algorithm is the thesaurus. The thesaurus is a memory, which stores only chromosomes. These chromosomes occur from two sources: local optima discovered by the algorithm during exploration and the elite of the last generation of the subpopulations explored by the copies of the genetic algorithm. Finally, the best solution is extracted from the thesaurus.

3) Communications protocol

This section details the data flows between the components of the segregative algorithm. The transfers are defined following the source and destination.

a) Data flows between subpopulations

Denote by \( P_k \) the subpopulation currently explored by the copy \( k \) of the BGA and by \((o_k, r_k)\) the corresponding cluster centroid and radius, \( k = 1, p \).

Let \( x \) be a new individual, produced by \( BGA_i \) during the exploration of its own subpopulation \( P_j \), by applying genetic operators. If \( \| o_i - \varphi(x) \| \leq r_i \), then \( x \) takes further part in the evolution process performed by \( BGA_i \).

If there exists some \( j \in \{1, \ldots, p\} \setminus \{i\} \), such that \( \| o_j - \varphi(x) \| \leq r_j \), then \( x \) is similar with the subpopulation \( P_j \), currently explored by \( BGA_j \). In this case \( B_j^m = B_j^m \cup \{x\} \), where \( B_j^m \) is a buffer in which \( BGA_j \) receives chromosomes that are similar to the individuals in its current subpopulation. Periodically, each copy of BGA empties its buffer and includes the chromosomes in the selection process of the current generation.

If \( \| o_j - \varphi(x) \| > r_j \), for \( j = 1, \ldots, p \) then \( x, \varphi(x) \) and its fitness are sent to the buffer \( B_j^m \), because \( x \) is not enough similar with any of the current subpopulations.

Whenever new chromosomes enter in the next generation in the evolution process performed on \( P_i \), \( BGA_i \) updates the centroid \( o_i \) and radius \( r_i \) and sends them to all other \( BGA_j \), with \( j \neq i \).

When a copy \( BGA_i \) finishes the exploration on \( P_i \), the pair \((o_\star, r_\star) = (o_i, r_i)\), containing the centroid and radius of the exhausted population is sent to \( TS_A \). The best found chromosome \( x^\star \) is sent to the thesaurus.

In order to efficiently use the computational resources, the traffic between different copies of BGA and the traffic from the copies of BGA towards \( TS_A \), occurs once at every \( r \) generations. This means that the individuals received by a copy \( BGA_i \) in the buffer \( B_i^m \) are introduced in the own evolution process at every \( r \) evolution stages of the population \( P_i \). Also \( TS_A \) processes the individuals sent by all copies of BGA at every \( r \) generations.

b) Data flows involving control components

Every \( r \) generations, counting all \( BGA \) copies, \( TS_A \) processes the chromosomes from \( B_j^m \). Let \( x \) be such a chromosome. If \( TS_A \) doesn’t discard \( x \) due to similarity with already explored solution, \( x \) is sent to \( Q_U \).

Whenever a copy of BGA finishes the exploration on a subpopulation, the component \( Q_U \) sends a new cluster to be explored to that copy of BGA. The sent cluster is removed from \( Q_U \).

c) Stopping condition

The stopping condition for each copy of BGA is that the best fitness of the evolved population stagnates for a prescribed number of stages. After finishing the evolution process a copy of BGA requests a new subpopulation from \( Q_U \). If \( Q_U \) meets the demand the activity is restarted on the newly received subpopulation. The entire SGA stops when all copies of BGA finish their tasks and \( Q_U \) becomes empty.

C. Execution of the algorithm

This section presents the execution steps of the segregative algorithm. The steps of the BGA include the normal execution and the aspects related to the communication protocol. After that the execution steps regarding the activities of the control components are detailed.

Basic genetic algorithm -BGA

i. Receive a subpopulation \( P_k \), \( k \in \{1, \ldots, S\} \), to be explored from \( Q_U \) with its associated individuals, feature vectors, centroid \( o_k \), and radius \( r_k \).

ii. Denote by \( g \) the current generation, initially \( g = 1 \); initialize the evolved population \( pop(1) \) with \( P_k \); empty the input buffer \( B^m \).

iii. While stopping condition is false do:

   iii.a. \( g = g + 1 \);

   iii.b. \( pop(g) = pop(g - 1) \);

   iii.c. Adjust genetic operators probabilities, apply mutation and crossover, evaluate new chromosomes, compute feature vectors; for each new chromosome \( x \) either:

   - \( pop(g) = pop(g) \cup \{x\} \);
   - Send \( x \) to another explored subpopulation;
   - Send \( x \) to \( TS_A \), \( B_j^m = B_j^m \cup \{x\} \);

   iii.d. If \( g \% r = 0 \) then \( pop(g) = pop(g) \cup B^m \), \( B^m = \emptyset \);

   iii.e. Apply selection on \( pop(g) \)

   iii.f. Update centroid \( o_k \) and radius \( r_k \) and send them to other BGA copies.

The steps of the SGA control components are further
detailed. They mostly correspond to the data flows described in the previous sections.

**Segregative genetic algorithm - SGA**

i. Randomly initialize the primordial population $P$;

ii. Obtain the initial subpopulations $P_i, i=1,...,S$, by clustering $P$, initialize $Q_U$ with these subpopulations;

iii. $TS_A = \emptyset$, $B_{TS}^0 = \emptyset$;

iv. While $Q_U$ is not empty do:

   iv.a. If there is a free BGA then send the most promising unexplored cluster from $Q_U$ to BGA, activate BGA to begin the evolution process on the given subpopulation;

   iv.b. When a BGA finishes the exploration send the centroid and radius of the explored cluster to $TS_A$, update the thesaurus and go to step iv.a;

   iv.c. At every each $r$ iterations counting all copies of BGA, for each chromosome $x$ in $B_{TS}^n$:

      - Send $x$ to $TS_A$;

      - $TS_A$ will either discard $x$ as already explored or send $x$ to $Q_U$ otherwise;

      - If it receives $x$, $Q_U$ searches the most similar not yet explored cluster and either discards $x$ if its fitness is worse than the lowest fitness of any individual in that cluster, or stores $x$ in the initial population of that cluster, otherwise.

The components of SGA and their interactions are shown in Fig. 1.

![Fig. 1. Functional diagram of the segregative genetic algorithm.](image)

**V. EXPERIMENTAL RESULTS**

This section presents the experimental evaluation of the segregative algorithm. The adopted methodology is designed to accomplish the following goals: setting the parameters of BGA and SGA; reporting the performance for difficult ROBDD instances; analyzing the efficiency of the distributed implementation and communication protocol. Another set of experiments is focused on comparing the segregative approach with a single population algorithm.

Some classic benchmark problems from the LGSynth91 dataset [18] were used in all experiments. These problem instances are divided into three categories, depending on their size – small: cm85a, cm163a, cu, alu4, sl494, vda, misex3; medium: apex2, apex7, cordic, ttt2; and large instances: i3, apex6. The optimal solution values are known for almost all instances.

The program was tested on a computer with an AMD Athlon II X4 630 processor. This processor has 4 cores, so a maximum number of 4 copies of the BGA can be evolved in parallel.

In order to build the ROBDD corresponding to some variable ordering (fitness evaluation), the CUDD library was used [19].

**A. Parameters setting**

The parameters used for each copy of the BGA in all experiments are the ones thoroughly described in [10] and [14]. The parameters are the following: $pop\_size = 50$ for small and medium instances, 50 and 100 for large instances; $p_{CX}$, the probability to apply the crossover operator, is initially set to 0.6; $p_M$, the probability to apply mutation, is initially set to 0.35; the probabilities to choose between the three mutation operators are fixed – simple mutation: 0.25, inversion: 0.3, group mutation: 0.45; the objective function for the population variability is a exponential periodic function; $N_{stop}$ is set to 20 and $e_{stop}$ to 0.001.

The parameters for the SGA are determined empirically by running the algorithm a small number of times with different settings on different test instances. The initial large population size is set to 5000 chromosomes. Higher values don’t produce any improvement, while smaller values seem to gradually diminish the stability of the algorithm. Values smaller than 500 decrease the efficiency of the algorithm, bringing the performance closer to that of the single population approach.

The tested values for the communication period, $r$, are 2, 4, 8 and 12. The value used in the performance evaluation is 8. Other conclusions regarding this measure are detailed in a next subsection, when the quality of the distributed implementation is analyzed.

**B. Performance evaluation**

1) Best feature function

Both described feature vectors, $\varphi_1$ and $\varphi_2$, were used to produce results. For each benchmark instance 30 runs of the algorithm were made with each of the two proposed feature vectors. Tables I and II report the evaluation of the algorithm in terms of quality and stability of the solution and computational effort, using feature vectors $\varphi_1$ and $\varphi_2$, respectively. For each test instance the following indicators are reported: best found solution in all 30 runs, denoted $best\_f$; the average ($m$), standard deviation ($\sigma$) and unitized risk ($\sigma/m$) for the absolute error ($best\_f - best\_k$), where $best\_k$ denotes the best known solution;
average number of iterations (it) and average number of objective function evaluations (fit) rounded to the nearest integer.

### Table I

**Performance Evaluation Using Feature Vector \( \phi_1 \)**

<table>
<thead>
<tr>
<th>Instance</th>
<th>best_k</th>
<th>best_f</th>
<th>m</th>
<th>( \sigma )</th>
<th>( \sigma/m )</th>
<th>it</th>
<th>fit</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm85a</td>
<td>28</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20.23</td>
<td>1283</td>
</tr>
<tr>
<td>cm163a</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21.32</td>
<td>648</td>
</tr>
<tr>
<td>cu</td>
<td>32</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20.13</td>
<td>1178</td>
</tr>
<tr>
<td>alu4</td>
<td>350</td>
<td>350</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>58.82</td>
<td>3056</td>
</tr>
<tr>
<td>s1494</td>
<td>369</td>
<td>369</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>162.2</td>
<td>7529</td>
</tr>
<tr>
<td>vda</td>
<td>478</td>
<td>478</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>355.4</td>
<td>12472</td>
</tr>
<tr>
<td>misex3</td>
<td>478</td>
<td>478</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>66.32</td>
<td>2993</td>
</tr>
<tr>
<td>apex2</td>
<td>-</td>
<td>316</td>
<td>319.63</td>
<td>2.235</td>
<td>0.0069</td>
<td>1462.32</td>
<td>225180</td>
</tr>
<tr>
<td>apex7</td>
<td>-</td>
<td>226</td>
<td>230.03</td>
<td>2.173</td>
<td>0.0094</td>
<td>1236</td>
<td>207021</td>
</tr>
<tr>
<td>cordic</td>
<td>42</td>
<td>42</td>
<td>1.36</td>
<td>0.889</td>
<td>0.6511</td>
<td>201.82</td>
<td>9172</td>
</tr>
<tr>
<td>tt2</td>
<td>107</td>
<td>107</td>
<td>1.63</td>
<td>0.668</td>
<td>0.4093</td>
<td>227.94</td>
<td>9205</td>
</tr>
<tr>
<td>i3</td>
<td>133</td>
<td>143</td>
<td>11</td>
<td>0.787</td>
<td>0.0716</td>
<td>954.11</td>
<td>75271</td>
</tr>
<tr>
<td>apex6</td>
<td>498</td>
<td>484</td>
<td>64.9</td>
<td>7.901</td>
<td>0.1217</td>
<td>6891.8</td>
<td>437910</td>
</tr>
</tbody>
</table>

### Table II

**Performance Evaluation Using Feature Vector \( \phi_2 \)**

<table>
<thead>
<tr>
<th>Instance</th>
<th>best_k</th>
<th>best_f</th>
<th>m</th>
<th>( \sigma )</th>
<th>( \sigma/m )</th>
<th>it</th>
<th>fit</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm85a</td>
<td>28</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21.26</td>
<td>1445</td>
</tr>
<tr>
<td>cm163a</td>
<td>26</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22.26</td>
<td>803</td>
</tr>
<tr>
<td>cu</td>
<td>32</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>21.43</td>
<td>1438</td>
</tr>
<tr>
<td>alu4</td>
<td>350</td>
<td>350</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>66.53</td>
<td>3321</td>
</tr>
<tr>
<td>s1494</td>
<td>369</td>
<td>369</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>184.6</td>
<td>7997</td>
</tr>
<tr>
<td>vda</td>
<td>478</td>
<td>478</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>377.3</td>
<td>15434</td>
</tr>
<tr>
<td>misex3</td>
<td>478</td>
<td>478</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>72.36</td>
<td>3453</td>
</tr>
<tr>
<td>apex2</td>
<td>-</td>
<td>304</td>
<td>315.16</td>
<td>3.553</td>
<td>0.0112</td>
<td>1668.73</td>
<td>279530</td>
</tr>
<tr>
<td>apex7</td>
<td>-</td>
<td>214</td>
<td>227</td>
<td>3.513</td>
<td>0.0154</td>
<td>1516</td>
<td>254095</td>
</tr>
<tr>
<td>cordic</td>
<td>42</td>
<td>42</td>
<td>1.3</td>
<td>1.022</td>
<td>0.7862</td>
<td>253.73</td>
<td>11140</td>
</tr>
<tr>
<td>tt2</td>
<td>107</td>
<td>107</td>
<td>1.5</td>
<td>1.106</td>
<td>0.7376</td>
<td>274.56</td>
<td>11606</td>
</tr>
<tr>
<td>i3</td>
<td>133</td>
<td>138</td>
<td>9.7</td>
<td>3.142</td>
<td>0.3239</td>
<td>1056.13</td>
<td>84898</td>
</tr>
<tr>
<td>apex6</td>
<td>498</td>
<td>536</td>
<td>55.93</td>
<td>9.655</td>
<td>0.1726</td>
<td>7883.5</td>
<td>484306</td>
</tr>
</tbody>
</table>

### The Pattern

The pattern that can be discovered in these experimental results is that the usage of feature vector \( \phi_1 \), which captures the structural information of chromosomes, provides better results than \( \phi_2 \), which captures cost-based information of individuals. The algorithm reaches better local optima, but does it at a little less stable. The usage of \( \phi_2 \) still produces better average results.

The small values of the standard deviation and unitized risk indicate the high stability of the method, its capacity to produce the best or very close to best results in almost each run. In case of both feature vectors the global optimum is reached in all runs of the algorithm on all small instances. On medium and large instances involved in the experiment, by using \( \phi_2 \), the segregative approach reaches or gets (very) close to the global optimum.

Further, \( \phi_2 \) is adopted as a feature function.

2) **SGA versus BGA**

Table III presents the same indicators evaluated for the basic genetic algorithm, **BGA**, used in the segregative approach. All experiments were made in the same environment. The algorithm was randomly restarted in order to obtain a number of iterations / objective function evaluations per run as similar as possible to the values obtained in the case of the segregative algorithm (in average).

### Table III

**Performance Evaluation for the Basic Genetic Algorithm**

<table>
<thead>
<tr>
<th>Instance</th>
<th>best_k</th>
<th>best_f</th>
<th>m</th>
<th>( \sigma )</th>
<th>( \sigma/m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm85a</td>
<td>28</td>
<td>28</td>
<td>0.30</td>
<td>0.90</td>
<td>3.00</td>
</tr>
<tr>
<td>cm163a</td>
<td>26</td>
<td>26</td>
<td>0.20</td>
<td>0.63</td>
<td>3.16</td>
</tr>
<tr>
<td>cu</td>
<td>32</td>
<td>32</td>
<td>0.60</td>
<td>0.51</td>
<td>0.86</td>
</tr>
<tr>
<td>alu4</td>
<td>350</td>
<td>350</td>
<td>2.00</td>
<td>2.16</td>
<td>1.00</td>
</tr>
<tr>
<td>s1494</td>
<td>369</td>
<td>369</td>
<td>4.9</td>
<td>5.04</td>
<td>1.02</td>
</tr>
<tr>
<td>vda</td>
<td>478</td>
<td>478</td>
<td>6.40</td>
<td>6.44</td>
<td>1.00</td>
</tr>
<tr>
<td>misex3</td>
<td>478</td>
<td>478</td>
<td>13.10</td>
<td>20.64</td>
<td>1.57</td>
</tr>
<tr>
<td>apex2</td>
<td>-</td>
<td>308</td>
<td>352.39</td>
<td>54.28</td>
<td>0.15</td>
</tr>
<tr>
<td>apex7</td>
<td>-</td>
<td>215</td>
<td>240.40</td>
<td>11.44</td>
<td>0.04</td>
</tr>
<tr>
<td>cordic</td>
<td>42</td>
<td>42</td>
<td>5.30</td>
<td>3.59</td>
<td>0.67</td>
</tr>
<tr>
<td>tt2</td>
<td>107</td>
<td>107</td>
<td>4.10</td>
<td>5.44</td>
<td>1.32</td>
</tr>
<tr>
<td>i3</td>
<td>133</td>
<td>157</td>
<td>35.39</td>
<td>11.92</td>
<td>0.33</td>
</tr>
<tr>
<td>apex6</td>
<td>498</td>
<td>561</td>
<td>84.8</td>
<td>22.26</td>
<td>0.26</td>
</tr>
</tbody>
</table>

On all test instances the results clearly indicate the significantly better performance of the segregative approach in terms of solution quality and especially of solution stability. Greater differences in favor of the segregative approach can be seen on the large and hard instances of the problem, i3 and apex6.

The following paragraphs summarize and underline some important characteristics and advantages of the segregative algorithm, compared to its single population counterpart:

a. **SGA** obtains significantly more stable results. The variance of the solution for the most difficult problem instances takes values like 0.787 (i3), 7.901 (apex6), for the segregative approach, while for the **BGA** the corresponding values are 11.92 and 22.26, respectively.

b. **SGA** compensates for weak performances of the basic population based heuristic. This can be seen in the case of best found solution and absolute error average, especially for difficult instances: **apex6** – **SGA**: 536 (55.93) / **BGA**: 561 (84.8); i3 – **SGA**: 138 (9.7) / **BGA**: 157 (35.39).

c. The exploration of the search space is accomplished in a more systematical manner, as it is guided by well-defined characteristics of the search space. The exploitation is more efficient, as it is focused on clearly defined regions of the search space.

d. **SGA** systematically avoids being trapped in local optima, by using the tabu memory.

e. **SGA** has very good potential for a distributed implementation, which makes it scalable to very large instances of the problem, given the needed computation resources.

Finally, we mention that in this case the segregative approach dominates the island model of **GA**.

C. **Distributed implementation evaluation**

The following section describes the experiments designed to evaluate the distributed implementation and the communication protocol.

As previously mentioned, the program was tested on a 4 cores processor, so 4 copies of the **BGA** ran in parallel. All described data is taken from executions of the algorithm on the **cordic** problem instance.
1) Computation efficiency

The quality measures used to evaluate the distributed implementation are: the parallel computing time - maximum execution time for any processor, denoted by $T_p$, and the efficiency of processors utilization $E_c = \frac{T_s}{(p \cdot T_p)}$, where $p$ is the number of processors and $T_s$ is the sequential computing time.

The average values for the described measures for all problem instances are given in Table IV. The instances are grouped depending on their size. The parallel computing time is quite small taking into account the complexity of the fitness evaluation (building the ROBDD) and that the algorithm explores many subpopulations. $E_c$ is very close to 1, meaning that almost the entire processing power is used, and proving the efficient design of the communication protocol.

<table>
<thead>
<tr>
<th>Instance class</th>
<th>$T_p$ (sec)</th>
<th>$E_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>4.152</td>
<td>0.91</td>
</tr>
<tr>
<td>medium</td>
<td>16.76</td>
<td>0.95</td>
</tr>
<tr>
<td>large</td>
<td>41.64</td>
<td>0.975</td>
</tr>
</tbody>
</table>

These results were obtained using a communication period, $r$, equal to 8. Decreasing this value to 4 or 2 produces a smaller efficiency of the processor usage (~0.8) on small instances, because the communication starts to overwhelm the computations regarding the evolution process performed by the copies of $BGA$, which are very fast in this case. On medium and especially large instances the difference is almost insignificant. Increasing the value of $r$ to 12 slightly increased $E_c$ on small and medium instances.

2) Communication protocol

This subsection presents some aspects regarding the flows of data at different levels of the segregative approach.

Fig. 2 represents the traffic of individuals from three of the currently explored subpopulations towards the fourth one. The communication moments represent evolution stages multiple of the communication period $r$. Experiments haven’t revealed any special pattern in the communication between subpopulations, like having a more intense communication in the beginning of the exploration and a decreasing trend as the populations evolve and finally converge, as happens with other models of GA with many populations. One reason for this type of unpredictable behavior is the dynamic adjustment of the genetic operator probabilities, $p_{cx}$ and $p_m$, which imposes a periodic variability target in the evolved population.

In general, the experiments show that the volume of transferred individuals is small compared with the volume of kept chromosomes. This fact could suggest that even if $\varphi_2$ is better than $\varphi_1$, it has a discriminating power that should be improved.

Fig. 3 depicts the traffic that occurs at the control component level of the segregative algorithm. The values in both graphs represent the total amount of transferred chromosomes at some generation in the whole evolution process.

These moments are considered over all subpopulations that evolve in parallel in the order in that they occur in real time. The dotted line represents the number of chromosomes transferred from the subpopulations under exploration towards $B_{TS}^{m}$ and immediately from $B_{TS}^{m}$ to $TS_A$. These are chromosomes which are discarded by $TS_A$ as they belong to already explored regions of the search space. The initial value is 0, because initially $TS_A$ is empty. The solid line represents the amount of individuals redirected by $TS_A$ to $QU$. This line starts from 5000 because this is the size of the initial large population, which is clustered in order to obtain the subpopulation to be explored.

In the beginning of the evolution process and until the copies of $BGA$ finish the exploration of a few number (4-6) of subpopulations, many individuals sent to $TS_A$ will be redirected to $QU$. This can be seen in Fig. 3 as the ascent of the solid line, correlated with the low amounts of individuals represented by the dotted line. As the explore experience of $TS_A$ grows, by storing more and more centroids of exhausted subpopulations, many individuals are discarded and the flow towards $QU$ stagnates. The experiments have revealed that the quantity of individuals that enter $QU$ is quite small and stagnates very fast. This indicates again that the feature vector should have better abilities to discriminate between similar chromosomes.

Fig. 4 represents the evolution of the mean fitness in the thesaurus. It is not monotone because the values that enter the thesaurus are taken from all explored subpopulations, including weak quality ones. Toward the end of the execution the fitness values that enter the thesaurus are very weak,
proving that the estimated priority of the subpopulations in $Q_U$ is not totally inaccurate. This experiment suggests that the exploration could be stopped earlier, before $Q_U$ becomes empty.

![Evolution ofthesaurus](image)

**Fig. 4.** Evolution of thesaurus.

Fig. 5 represents the evolution of the best-found solution, taking in considerations all copies of the BGA. The initial value is very high, due to random initialization. The algorithm produces very fast much better solutions. The constant improvement denotes the capability of the approach to better explore the search space and to avoid being trapped in local optima.

![Best found solution](image)

**Fig. 5.** Evolution of best-found solution.

VI. CONCLUSIONS AND FURTHER RESEARCH DIRECTIONS

In this paper an efficient segregative genetic algorithm with statically initialized subpopulations for optimizing variable order for ROBDDs was presented. The approach explores many subpopulations, acts in both representation and feature spaces and uses an associative tabu search mechanism to avoid being trapped in local optima. The method was tested on classical benchmark problems available in the literature, and gives very good results, especially in terms of stability, compared to the single population algorithms. The scheme of the segregative genetic algorithm is general, can incorporate different single population algorithms, and can be used to solve other difficult problems as long as a similarity measure between solution representations can be devised.

The presented method is the first segregative approach working in feature space designed for ROBDD variable order optimization. Further research is envisioned to the extending of the algorithm to manage new genetic material in a dynamic manner. In order to achieve this goal and also better performance for the static version, the feature vectors must be modified, as momentarily their ability to discriminate between similar structures of chromosomes seems to be quite poor.

ACKNOWLEDGEMENT

Special thanks to Dr. Rüdiger Ebendt, Institute of Transportation Systems, Berlin, German Aerospace Center, for his support and for providing us his ROBDD parser, based on CUDD package. All the experiments included in this paper are made with this parser.

The second author acknowledges that this work was partially supported by the European Social Fund in Romania, under the responsibility of the Managing Authority for the Sectorial Operational Programme for Human Resources Development 2007-2013, Grant POSDRU/88/1.5/S/47646.

REFERENCES

[10] synasc
[19] CUDD package url: vlsi.colorado.edu/~fabio/ CUDD.