Dynamic Segregative Genetic Algorithm for Optimizing the Variable Ordering of ROBDDs

Cristian Rotaru
”Al. I. Cuza” University of Iaşi, Faculty of Computer Science
Gen. Berthelot 5, Iaşi, Romania
cristian.rotaru@infoiasi.ro

Octav Brudaru
“Gh. Asachi” Technical University of Iaşi
D. Mangeron 53, Iaşi, Romania
brudaru@tuiasi.ro

ABSTRACT

In this paper an efficient dynamic segregative genetic algorithm for optimizing variable order in Reduced Ordered Binary Decision Diagrams is presented. The approach integrates a basic genetic algorithm and uses a feature function in order to define a similarity measure between chromosomes. Subpopulations of individuals, formed by applying a clustering procedure in the feature space, are explored in parallel by multiple copies of the basic genetic algorithm. A communication protocol preserves the similarity inside each subpopulation during the evolution process. The redundant exploration of the search space is avoided by using a tabu search associative memory. Genetic material from yet unexplored regions of the search space is managed and organized in order to explicitly guide the search process to yet undiscovered local optima. The experimental evaluation of the algorithm uses classical benchmark problems, known to be very difficult. Experiments suggest that our approach has a better performance in terms of stability and quality of the solution, when compared to other heuristics, such as local search methods, basic genetic algorithms, a cellular genetic algorithm and even the static segregative genetic algorithm that was the starting point of this work. The quality of the distributed implementation and the communication protocol are thoroughly analyzed.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search – *heuristic methods.*

General Terms

Algorithms

Keywords

segregative genetic algorithm, feature space, associative tabu search memory, search space management, similarity preserving communication protocol, extensive exploration, intensive exploitation, distributed implementation.

1. INTRODUCTION

We propose a new method to optimize the variable order in Reduced Ordered Binary Decision Diagrams (ROBDDs). ROBDDs are data structures that represent canonical forms for Boolean functions. An ROBDD is an acyclic graph with respect to some order of the input variables and satisfying a set of properties. The size of the ROBDD is the number of non-terminal nodes and it is proven that it strongly depends on the order of the variables ([1]). The goal is to minimize this size. One important application of ROBDDs is in digital circuit design – a smaller size of the ROBDD that represents a Boolean function describing a circuit transfers directly to a smaller chip size. Other applications include: symbolic model checking; formal verification of combinatorial circuits; analysis of sequential systems.

The methods used to tackle the problem are both exact and approximate. The most successful exact approaches are based on the branch-and-bound paradigm and use the lemma proven in [2]. The heuristic approaches are categorized in two types: static and dynamic methods. Static methods use application specific information, for example the structure of the evaluated combinatorial circuit, to build good variable orders before constructing the ROBDD ([3]). Dynamic methods improve the size of an ROBDD by modifying the order of variables. Basic approaches are described in [4] and [5]. [6] presents “the sifting heuristic”, a specialized local search technique based on hill-climbing. [7] describes a method based on simulated annealing and [8] introduces the use of genetic algorithms, further developed in [9]. Hybrid approaches are described in [10] and [11]. [12] introduces a novel method for optimizing the variable order, a cellular genetic algorithm with parallel evolving and communicating grids. This method employs multiple populations to explore the search space and uses a similarity preserving communication protocol to transfer high quality genetic material between subpopulations, in order to boost the quality of the solution.

In [13] the authors describe an initial variant of a segregative genetic algorithm for the problem of optimizing the variable order of ROBDDs. The segregative approach defines a similarity measure between chromosomes. The measure is computed in a feature space obtained by attaching feature vectors to chromosomes. Large numbers of individuals are organized in subpopulations, by applying a clustering procedure in the feature space. The search space is explored by means of a basic genetic algorithm (BGA) on each subpopulation. Depending on computational resources, multiple subpopulations can be explored in parallel. A communication protocol is used to maintain the similarity inside each subpopulation during the evolution. A tabu search associative memory is introduced in order to avoid redundant exploration of the search space.

In this paper an improved version of the algorithm described in [13] is proposed. Some of the components of the approach are revised to deliver better performance. The main improvement is the introduction of the management of yet unexplored regions of the search space. The algorithm in [13], called Static Segregative Genetic Algorithm (SSGA), generates a large initial population, divides it in a fixed number of subpopulations and explores these subpopulations. Chromosomes that occur during the evolution process and are not similar with any subpopulation are included in some subpopulation, but usually they do not have the power to guide the search process towards the newly discovered region of the search space. The new approach explicitly manages chromosomes from these regions, by dynamically creating subpopulations that will be later explored by the BGA. The approach is called Dynamic Segregative Genetic Algorithm (DSGA). Some of the ideas used by the approach can be also found in [14].

The structure of the paper is the following: Section 2 describes the problem; Section 3 briefly describes the BGA; Section 4 presents the translation from genotype to phenotype; Section 5 details all aspects of the segregative approach; Section 6 presents the extensive experimental evaluation of the method; Section 7 summarizes some conclusions and future work directions.

2. OPTIMIZING THE ORDER OF VARIABLES FOR ROBDDS

This section formally describes the ROBDD data structure and the problem of optimizing the variable order. The importance of variable ordering is briefly underlined.

Let *BF*: $B^n \rightarrow B^m$, $B = \{0,1\}$ be a Boolean function that outputs *m* values. Let π be a permutation of the *n* input variables, $\pi = (v_1,v_2,\ldots,v_n)$, $v_i \in B$, $i = 1,\ldots,n$.

An Ordered BDD (OBDD) for *BF* with respect to π is a directed acyclic graph with the following properties ([15]): 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 any path in the graph respects the order π. The number of nonterminal nodes in the graph represents the size of the OBDD and is the value that must be minimized.

The structure called Reduced OBDD (ROBDD) is obtained by applying reduction rules to an OBDD with a fixed variable order, further reducing the size of the graph ([15]). A ROBDD is a canonical form for *BF*. The size of the structure strongly depends on the fixed order. In [1] it is proven that optimizing the order of variables is NP-complete and finding the best order is NP-hard.

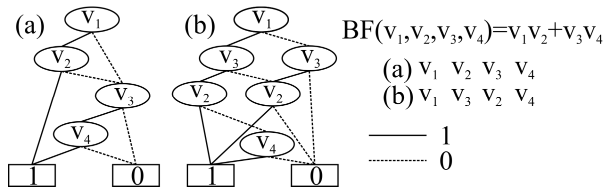
(a)
(b)
$BF(v_1,v_2,v_3,v_4)=v_1v_2+v_3v_4$
(a) v_1 v_2 v_3 v_4
(b) v_1 v_3 v_2 v_4
1
0

Figure 1. Dependence of the size of the ROBDD on the order of variables

Figure 1 illustrates the effect of using different orders of variables when building ROBDDs for a simple Boolean function. For two slightly different orders the sizes of the corresponding ROBDDs differ substantially.

3. BASIC GENETIC ALGORITHM

The method used to explore subpopulations in the context of the segregative approach can be any population based method for which a similarity measure between individuals can be defined. The method used in this paper is the genetic algorithm described thoroughly in [16].

A chromosome represents a permutation of the input variables of *BF*, denoted $x = (x_1,x_2,\ldots,x_n)$, $x_i \in \{v_1,\ldots,v_n\}$, i=1,…,*n*. The fitness of a chromosome, denoted $f(x)$, is the size of the ROBDD obtained by using *x* as the order of variables. The size of the initial population, denoted *pop_sz*, is directly proportional to the number of variables, *n*, as a consequence of an empirically determined rule. The initial population is generated randomly.

The algorithm applies an elitist selection for survival. The offspring compete with the current generation. The best *pop_sz* individuals with respect to fitness value are kept in the next generation.

The method uses two crossover operators, described in [9], and three mutation operators, described in [17]. [16] shows that using several operators yields better results than applying only one operator for crossover and one for mutation. The common characteristic of all genetic operators is that the similarity between parents and offspring grows with the number of generations, assuring the convergence of the algorithm. The probabilities to apply crossover and mutation, denoted p_{CX} and p_M respectively, are empirically initialized and dynamically controlled during the evolution process, by means of a measure of the variability of the population. The variability is computed each generation as the distribution of fitness values over the entire population. The probabilities are adjusted in order for the variability to match a predefined objective function. This function ultimately defines the balance between exploration and exploitation. In [16] specific equations are presented and the effects of several objective functions are thoroughly examined.

The algorithm stops when the variation of the average fitness of the population stagnates: $|avg\_fit(k) - avg\_fit(k-1)|/avg\_fit(k) < T_{stop}$ holds true for N_{stop} consecutive generations, where $avg\_fit(k)$ is the average fitness of the population at generation *k*, and T_{stop} a small threshold value.

4. FEATURE SPACE

The capacity of the segregative algorithm to manage individuals during the evolution process depends on the measure of similarity defined between individuals. As mentioned before, a chromosome represents a permutation of the input variables. This representation does not support a straightforward definition of a measure of similarity. In order to define the similarity between two individuals, a real valued feature vector is computed for each chromosome. The vector, denoted $\varphi(x)$, is computed when the chromosome *x* is evaluated. The measure of similarity between two chromosomes *x* and *x*′ is defined as the Euclidean distance in $\mathbb{R}^n$ between the corresponding feature vectors, $\| \varphi(x) - \varphi(x') \|$.

Two methods to compute the feature vector are described below, both used to define the similarity between chromosomes in [12] and [13]. Let *x* be a chromosome. One option is to use the structural information of *x*, as permutation of the input variables. The obtained feature vector, denoted $\varphi_1(x)$, is defined as: $\varphi_1(x) = (\alpha_1,\ldots,\alpha_n)$, where $\alpha_i = x_i/[n(n+1)/2]$, $i = 1,\ldots,n$. $\varphi_1(x) \in \mathbb{R}^n$ and $\Sigma\alpha_i = 1$. Another option is to capture the cost based characteristics of *x*. The second feature vector, denoted $\varphi_2(x)$, is defined as: $\varphi_2(x) =$

$(\lambda_1,\ldots,\lambda_n) \in \mathbb{N}^n$, where λ_i represents the number of nodes on the i-th level of the graph corresponding to the ROBDD.

The extensive experimental evaluations of the algorithms described in [12] and [13] show that using the feature vector φ_1 yields more stable but poorer quality results than using φ_2. In all cases, by using φ_2, the stability of the results decreases slightly, while the quality improves substantially. All experimental results further presented in this paper are obtained by using φ_2, denoted simply φ from now on.

5. THE SEGREGATIVE ALGORITHM

This section describes the elements of the segregative approach and how they work.

5.1 Search space exploration by subpopulations

The segregative algorithm employs multiple copies of the BGA to explore the search space. At any particular moment, each copy evolves a given subpopulation. Subpopulations are clusters of similar chromosomes, with regard to the feature vector φ, and are representative for some region of the search space. Depending on available computational resources, a number p of subpopulations are explored in parallel. During the evolution process, the segregative approach discovers and explores various well defined regions of the search space and intensifies the search in promising regions by applying the BGA. A communication protocol preserves the similarity inside the evolved subpopulations. The approach avoids redundant exploration of regions of the search space and manages yet unexplored regions of the search space. This is done by means of the control components of the algorithm, which will be later detailed. The segregative algorithm acts as a collector of various local optima from different regions of the search space.

5.1.1 Initializing the subpopulations

Initially, a large population, denoted P_{init}, is randomly generated. All chromosomes are evaluated and the feature vectors are computed. A clustering algorithm is used to split the set of feature vectors in S similar groups, $Cl_1,\ldots,Cl_S \subset \mathbb{R}^n$, with the centroids $\phi_1,\ldots,\phi_S \in \mathbb{R}^n$. The initial subpopulations are represented by the preimages of the clusters from the feature space, in the representation space, $P_i = \varphi^{-1}(Cl_i)$, $i = 1,\ldots,S$. The subpopulations are sent to the component of the algorithm Q_{NEW} that will distribute them to the copies of BGA, for exploration. The activity of Q_{NEW} will be later detailed.

Experiments show that the *c-means* clustering algorithm offers good results in terms of speed and accuracy. The Davies-Bouldin index for the quality of the clustering solution was used to determine a good value for the number of clusters, S.

5.1.2 Control components

The control components of the algorithm are used to define the behavior of the method. The following subsections will detail: the tabu search associative memory, the management of yet unexplored and unknown regions of the search space, and the archive that stores promising individuals occurred during the evolution process. Several focused scenarios involving communications between components of the algorithm and subpopulations will be presented. A later subsection will detail all aspects regarding the communication protocol.

5.1.2.1 Tabu search associative memory

The tabu search associative memory, denoted TS_{AM}, is a control component introduced in order to avoid the redundant exploration of regions of the search space. As previously mentioned, the BGAs evolve subpopulations of individuals that induce clusters of feature vectors in the feature space. A subpopulation SP has a corresponding centroid in the feature space, $\phi_{SP} \in \mathbb{R}^n$, and a cluster radius, $\rho_{SP} \in \mathbb{R}$, which are computed when the clustering algorithm is applied. The centroid captures the features off all individuals in the cluster and is computed by the clustering algorithm. The radius is set as the maximum distance between any feature vector in the cluster and the centroid. When the evolution process on SP is finished, the pair (ϕ_{SP},ρ_{SP}) is sent to TS_{AM}. At some moment in the execution of the segregative algorithm, TS_{AM} stores s pairs (ϕ_{sp},ρ_{sp}), where ϕ_{sp} is the centroid of some explored subpopulation sp and ρ_{sp} is the radius of the cluster. The associative feature of this memory comes from the fact that it does not store chromosomes from different regions of the search space; it only stores centroids which capture the characteristics of many individuals from the corresponding regions.

Let x be a chromosome received by TS_{AM} from a copy of the BGA. The communication protocol between subpopulations currently explored by copies of BGA ensures that x does not belong to any of those subpopulations. The function of TS_{AM} is to decide whether x belongs to an already explored region of the search space or not. If for some $j \in \{1,\ldots,s\}$, $\| \varphi(x) - \phi_j \| \leq \rho_j$ holds, then x belongs to an already explored region and is discarded, otherwise x belongs to an yet unexplored or unknown region and it is sent to Q_{NEW}.

5.1.2.2 Management of the search space

The component that manages yet unexplored and unknown regions of the search space, denoted Q_{NEW}, contains two collections of individuals: PQ_{NEW} and U_{NEW}.

PQ_{NEW} is a priority queue of yet unexplored clusters of individuals. PQ_{NEW} stores the centroids and radiuses of the clusters and also a small number of representative chromosomes that will be used to start the evolution process on the subpopulation. The subpopulations are scored with the average fitness of these individuals. When a copy of the BGA finishes the evolution process on the assigned subpopulation, it will receive from PQ_{NEW} the best subpopulation to explore. PQ_{NEW} is initialized with the initial subpopulations, $P_1,\ldots,P_S$.

U_{NEW} is a large collection of chromosomes from unknown regions of the search space. Let x be a chromosome received by Q_{NEW} from TS_{AM}. x does not belong to any subpopulation that is currently or was already explored. If x is similar enough with some cluster from PQ_{NEW} it will be stored in the small population of that cluster if it is good enough, or it will be discarded. If not, x is stored in the unstructured collection U_{NEW}. Initially, U_{NEW} is empty. When the size of U_{NEW} reaches a high enough value, 1000–2000 chromosomes, denoted M_{UNEW}, the same clustering procedure described in the initialization phase is applied, U_{NEW} is emptied, and the resulting subpopulations are appended to PQ_{NEW}.

Q_{NEW} is the component that ensures the dynamic character of the approach by managing and organizing new genetic material produced during the evolution process on subpopulations. The segregative algorithm stops when PQ_{NEW} and U_{NEW} are empty and all copies of the BGA finish the execution on the assigned subpopulation.

5.1.2.3 Archive

The last component of the segregative approach is also a collection of chromosomes. The archive stores the elite of individuals from explored subpopulations, meaning 5−10% of the individuals from the last generation. The selection of the BGA being elitist, these are the best chromosomes produced by the evolution process on those subpopulations.

The main benefit of the archive is an alternative method to initialize the segregative method: it offers the means for efficiently restarting the algorithm, by including the high quality chromosomes from the archive in the primordial large population. This procedure can be correlated with keeping the information stored by TS_{AM} in consecutive runs of the algorithm, which can accelerate the discovery and exploration of new regions of the search space.

5.1.3 Communication protocol

Data transfer occurs between currently explored subpopulations, subpopulations and control components and between control components of the segregative algorithm. The communication between different entities defines their behavior and ensures the functionalities of the segregative method.

Each data transfer has a source and a destination. The source entity places data in a buffer at the destination. In order not to overwhelm the communication with processing times, data accumulates in buffers and all entities involved process their buffers every R evolution stages, counting all copies of the BGA. Effects of varying this value will be presented in the experimental evaluation section.

5.1.3.1 Data transfer at subpopulation level

Data transfer that originates in a subpopulation is intended to preserve the similarity inside that subpopulation. Offspring created by means of genetic operators are either kept in the subpopulation, or sent to other subpopulations or control components, if they are not similar enough with the current subpopulation.

Let BGA$_i$ be the copy of the BGA, which currently evolves the subpopulation SP_i. Let ϕ_i be the centroid of the cluster induced by SP_i in the feature space and ρ_i the radius of this cluster. Let x be a chromosome generated during the evolution process on SP_i.

If $\| \phi_i - \varphi(x) \| \le \rho_i$, then x is similar enough to the individuals in SP_i and is included in the evolution process performed by BGA$_i$.

If $\| \phi_j - \varphi(x) \| \le \rho_j$ holds for some $j \in \{1,\ldots,p\}, j \neq i$, then x is similar with the currently explored subpopulation SP_j and it will be included in the evolution process performed by BGA$_j$. BGA$_j$ receives such chromosomes in a buffer, denoted Buf_j.

If none of the above cases hold, then x is not similar with any currently explored subpopulation and it will be sent to TS_{AM}.

At the end of the evolution process on SP_i, the pair (ϕ_i,ρ_i) is sent to TS_{AM} and the elite of the current population is sent to the archive. The copy of the BGA that performed the exploration of SP_i will receive another subpopulation from Q_{NEW}, if available.

5.1.3.2 Data transfer at control component level

TS_{AM} receives chromosomes from the copies of the BGA in the buffer denoted B_{TSAM}. Let x be such an individual. If TS_{AM} decides that x belongs to an already explored region of the search space, then x is discarded, otherwise it is sent to Q_{NEW}.

Q_{NEW} receives individuals from TS_{AM} in the buffer denoted B_{QNEW}. The chromosomes are either discarded, or stored in the initial population of some cluster in PQ_{NEW}, or in U_{NEW}. Whenever a copy of the BGA finishes the exploration on some subpopulation, Q_{NEW} extracts the best subpopulation from PQ_{NEW} and sends it to the BGA for exploration.

5.2 Execution of the algorithm

This section summarizes the execution steps of the copies of BGA and of the control components of the segregative algorithm. The procedures include the functionalities of all entities and the data flows previously detailed.

5.2.1 Execution of the BGA

```
1. receive subpopulation $SP$ from $Q_{NEW}$: individuals to initialize
   the population, centroid and radius of the cluster induced in
   the feature space, $(\phi_{SP},\rho_{SP})$.
2. initialize the evolution stage, $st = 1$; initialize the current
   population $pop(1)$; empty the buffer, $Buf = \emptyset$.
3. while not stopped do:
   3a. $st = st + 1$;
       if $st \% R = 0$ then $pop(s) = pop(s-1) \cup Buf$, $Buf = \emptyset$;
       else $pop(s) = pop(s-1)$.
   3b. apply genetic operators, evaluate offspring, and
       compute feature vectors; for each new individual $x$:
       • $pop(s) = pop(s) \cup \{x\}$ or
       • send $x$ to another currently explored subpopulation or
       • send $x$ to $TS_{AM}$.
   3c. apply selection on $pop(s)$; update centroid and radius.
```

5.2.2 Execution of the segregative algorithm

```
1. initialize $P_{init}$; obtain $P_1,\ldots,P_S$ by clustering;
   $PQ_{NEW} = \{P_1,\ldots,P_S\}$; $U_{NEW} = \emptyset$.
2. $TS_{AM} = \emptyset$.
3. while $PQ_{NEW}$ and $U_{NEW}$ are not empty:
  3a. if a copy of the BGA is free and $PQ_{NEW}$ is not empty
      • choose the best subpopulation from $PQ_{NEW}$ and send
        it to the BGA;
      • the copy of the BGA starts the evolution process.
  3b. when a copy of the BGA finishes the exploration of a
      subpopulation:
      • send the centroid and radius of the completed cluster
        to $TS_{AM}$ ;
      • update the archive.
  3c. every $R$ generations, $TS_{AM}$ checks the chromosomes in
      $B_{TSAM}$ and either discards them or sends them to $Q_{NEW}$.
  3d. every $R$ generations, $Q_{NEW}$ processes the chromosomes in
      $B_{QNEW}$ and discards them, or stores them in some cluster in
      $PQ_{NEW}$, or stores them in $U_{NEW}$.
  3e. when $U_{NEW}$ reaches the size $M_{UNEW}$, or $U_{NEW}$ is not empty
      and $PQ_{NEW}$ is empty, apply the clustering procedure on the
      feature vectors of the individuals in $U_{NEW}$, obtain the
      similar subpopulations in the representation space and
      append them to $PQ_{NEW}$, empty $U_{NEW}$.
```

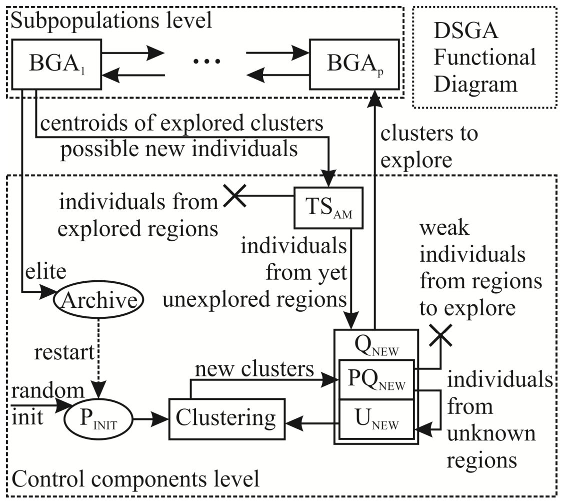

Figure 2. Functional diagram of the segregative algorithm

6. EXPERIMENTAL EVALUATION

The experiments are designed to study the stability and quality of the solution obtained by the segregative approach. Comparisons with the single population genetic algorithm, the cellular genetic algorithm from [12], the static segregative genetic algorithm from [13] and other heuristic approaches are extensively described below. Other experiments study the quality of the distributed implementation and the efficiency of the communication protocol.

The experimental evaluation uses well known benchmark problems from the LGSynth91 dataset, found at [21]. The instances are divided into three categories, depending on their size: small – *cm85a*, *cm163a*, *cu*, *alu4*, *s1494*, *vda*, *misex3*; medium – *apex2*, *apex7*, *cordic*, *ttt2*; and large instances – *i3*, *apex6*.

The optimal solution is known for most of the instances and is denoted *bk*. The reported indicators are the following: best found solution (*bf*), the average (*m*), standard deviation (σ) and unitized risk (σ/m) for the absolute error ($bf - bk$). The column titled In/Out presents the number of input and output variables of the Boolean function, respectively.

The CUDD library ([22]) was used in order to build the ROBDD associated with an order of variables and to compute the fitness and feature vector φ_2, which is used in all experiments.

6.1 Parameter tuning

The parameters of the basic genetic algorithm are the ones which gave the best results in [16]. The values are empirically set. The extensive experimental evaluation in [16] analyzes the effect of different choices for all parameters. The following values are used in [16] to produce the results reported in the performance evaluation and are also used in the segregative approach. The size of the population, *pop_sz,* is set to 50 for the small and medium problem instances, and to 50 and 100 for the large instances. p_{CX}, the probability of the crossover operator, is initialized to 0.6. p_M, the probability of the mutation operator, is initialized to 0.35. The objective function for the variability of the population, used to control p_{CX} and p_M, is an exponential periodic function. The parameters used to define the stopping condition are: $N_{stop} = 20$ and $T_{stop} = 0.001$.

The experiment designed to determine a suitable size for the primordial population, P_{init}, consists in running the algorithm on some of the test instances (*alu4*, *cordic* and *apex6*) with different values, ranging from 200 to 10000, and checking for statistically significant differences between the averages of the absolute error (*m*), obtained on each instance. Both Wilcoxon Signed-Rank non-parametric test and the two-tailed t-test with assumed unequal variances showed ($p < 0.01$) that values ranging from 2000 to 200 chromosomes produce a decrease in stability and quality of the solution. Values ranging from 2000 to 10000 revealed no significant difference. This proves the efficiency of the component Q_{NEW} that successfully manages the exploration of unknown regions of the search space, compensating for a poor initial genetic pool. The size P_{init} is set to 2000 individuals. As showed in [13], the static segregative genetic algorithm requires an initial population of size at least 5000 to deliver best performance. M_{UNEW}, the maximum size of the unstructured population U_{NEW}, is set to 1000 chromosomes ($|P_{init}| / 2$).

Various values for the communication period, *R*, are tested: 2, 4, 8, 10 and 12 evolution stages. Detailed results are presented in the section concerning the evaluation of the distributed implementation. The value used for generating the results in the performance evaluation is 8.

In conclusion, most of the parameters of the approach are set to known best suitable values, the rest are determined by means of a limited statistical analysis. As the results obtained by using these settings are very promising, a further in-depth analysis of the parameters and dependencies between them is not included in this work. Future research should devise experiments and procedures to determine if the whole ensemble of parameters, for the BGA and control components of the DSGA, can be fine-tuned against each other to deliver even better performance for the segregative approach.

6.2 Performance evaluation

For the performance evaluation, a computer with an AMD Athlon X4 630 processor was used. As this processor has 4 physical cores, 4 copies of the BGA ran in parallel. For each problem instance, the algorithm was executed 30 times.

6.2.1 Quality and stability

The reported indicators are presented in Table 1.

Table 1. Performance evaluation of the DSGA

Instance	In/Out	*bk*	*bf*	*m*	*σ*	*σ/m*
cm85a	11/3	28	28	0	0	-
cm163a	16/5	26	26	0	0	-
cu	14/11	32	32	0	0	-
alu4	14/8	350	350	0	0	-
s1494	14/25	369	369	0	0	-
vda	17/39	478	478	0	0	-
misex3	14/14	478	478	0	0	-
apex2	39/3	-	303	305.5	2.063	0.0067
apex7	49/37	-	214	215	1.114	0.0051
cordic	23/2	42	42	0	0	-
ttt2	24/21	107	107	0	0	-
i3	132/6	133	133	0	0	-
apex6	135/99	498	506	20.16	7.985	0.3959

The segregative method finds the optimal solution in all runs on all small and medium instances, for which the optimal value is known. This happens also for one of the large instances (*i3*). For the second large instance, *apex6*, which is the most difficult one, the results are very close to the optimal value. The standard deviation and unitized risk prove the high stability of the method which replicates the same or very close results in nearly all runs.

6.2.2 Comparison with other heuristic methods

The first comparison focuses on some simple search procedures and other more complex heuristics from the literature. As published results are not available in a systematic manner, only the best found solution is reported on some of the small and medium instances. Table 2 present the comparison with the following methods: the sifting heuristic ([6]); the genetic algorithm described in [8]; depth first search, breadth first search, fault simulation ([18]); level and fanin heuristics ([19]); fanout heuristic ([20]). The results show that the DSGA offers results at least as good as or better than these approaches.

Table 2. Comparison with other heuristics

Inst.	sift	GA[8]	DSGA	Inst.	sift	GA[8]	DSGA
cm85a	36	28	28	alu4	602	350	350
cm163a	28	26	26	s1494	388	370	**369**
cu	35	32	32				

Inst.	dfs	bfs	fsim	lvl	fin	fout	DSGA
vda	529	3647	533	2733	515	533	**478**
ttt2	190	171	234	163	170	186	**107**

The second comparison is a more detailed one with the following methods: the basic genetic algorithm used in the segregative approach (BGA), the static segregative genetic algorithm from [13] (SSGA) and the best variant of the cellular genetic algorithm from [12] (CGA). As all methods behave very well on the small instances, only the results for the medium and large instances are reported.

Table 3 presents the comparison with the BGA. In order to achieve a fair comparison, an execution of the BGA that contributes to the computation of the reported parameters consists actually of multiple executions followed by an aggregation of the results. The number of executions depends on the available computation time, which is set for each instance as the average execution time of the DSGA on that instance.

Table 3. Comparison with BGA

Method	**BGA**			**DSGA**		
Instance	***bf***	***m***	***σ***	***bf***	***m***	***σ***
apex2	312	352	54	303	305.5	2.063
apex7	215	240	11	214	215	1.114
cordic	42	5.3	3.6	42	0	0
ttt2	107	4.1	5.4	107	0	0
i3	157	35.4	12	**133**	**0**	**0**
apex6	561	84.8	22	**506**	**20.16**	**7.985**

Table 3 shows that both quality and stability of the solution are significantly higher in the case of the DSGA, especially on the most difficult instances. This fact proves some of the qualities of the segregative approach compared to the single population genetic algorithm. In the case of the DSGA, the stability of the solution is an explicit goal of the approach, achieved by a systematic exploration of the search space. The search procedure is guided by well-defined characteristics of the search space and the intensification of the search is more focused on clearly defined regions. The method avoids being trapped in local optima by using the tabu search memory. The quality of the solution, best found and average, shows that the efficient search process achieved by the DSGA compensates for the weak performance of the BGA.

Table 4 presents the comparison with the two other multi population approaches: SSGA and CGA.

Table 4. Comparison with SSGA and CGA

Meth.	**SSGA**			**CGA**			**DSGA**		
Inst.	***bf***	***m***	***σ***	***bf***	***m***	***σ***	***bf***	***m***	***σ***
apex2	304	315	3.6	309	318.6	7.2	303	305.5	2.06
apex7	214	227	3.5	214	225.3	11	214	215	1.12
cordic	42	1.3	1.0	42	0.1	0.3	42	0	0
ttt2	107	1.5	1.1	107	0.4	0.8	107	0	0
i3	138	9.7	3.1	133	0	0	**133**	**0**	**0**
apex6	536	55.9	9.6	508	38.6	13	**506**	**20.16**	**7.98**

The SSGA is a primitive version of the DSGA, having the same main purpose: a systematic exploration of the search space that focusses on solution stability and aims at improving the quality by exploring various local optima. The CGA is an intermediate design between the single population genetic algorithm and the segregative approach. The merit of the approach is the communication between parallel evolving populations that considerably boosts the quality of the solution. The results suggest that the DSGA performs better than both other approaches in terms of stability and quality of the solution.

All reported results are obtained in similar conditions: each algorithm was executed 30 times for each instance, on the same processor type. The parameters for the components of the BGA that are incorporated in all approaches are the same. Both Wilcoxon Signed-Rank non-parametric test and the two-tailed t-test with assumed unequal variances proved ($p < 0.01$) the statistical significance of the differences between the averages of the absolute error (*m*), on each problem instance.

The results of the t-test for the comparison with the CGA are the following: *apex2* – t(34) = −9.623, $p < 0.001$; *apex7* – t(30) = −5.321, $p < 0.001$; *cordic* – t(29) = −1.795, p = 0.083; *ttt2* – t(29) = −3.067, $p < 0.005$; *apex6* – t(45) = −6.623, $p < 0.001$. The statistical significance of the differences proves the quality of the main components introduced by the segregative design: the exploration of the search space by subpopulations, the communication protocol that preserves the similarity of the explored subpopulations and the tabu search associative memory.

The results for the comparison with the SSGA are the following: *apex2* – t(47) = −12.885, $p < 0.001$; *apex7* – t(35) = −17.832, $p < 0.001$; *cordic* – t(29) = −6.966, $p < 0.001$; *ttt2* – t(29) = −7.426, $p < 0.001$; *i3* – t(29) = −16.909, $p < 0.001$; *apex6* – t(54) = −16.091, $p < 0.001$. The tests confirm that the newly introduced component Q_{NEW}, that represents the main difference between DSGA and SSGA, produces significant differences in the quality of the solution obtained by the algorithm.

6.3 Evaluation of the distributed implementation

This section presents results regarding the efficiency of the distributed implementation and details several aspects regarding the communication protocol at different levels of the segregative approach.

6.3.1 Efficiency of computations

Four copies of the BGA were run in parallel on a four-core processor. The measures used to evaluate the quality of the distributed implementation are the following: the parallel computing time, denoted T_p – the maximum execution time for any processor; the efficiency of processors utilization, denoted E_c – $E_c = T_s / (p \times T_p)$, where T_s is the sequential execution time and p is the number of processors, $p = 4$.

The values are computed as averages over all executions of the algorithm that produced the results reported in the performance evaluation section and are grouped according to the size of the problem instances: small – T_p = 5.2 seconds, $E_c = 0.9$; medium – T_p = 22.5 seconds, $E_c = 0.94$; large – T_p = 70.3 seconds, $E_c = 0.97$.

The parallel time, in each processor, mainly consists of the fitness evaluation. The reported times are quite small taking into account the size of the problems and the complexity of building the ROBDD for a chromosome. Also, the algorithm explores many subpopulations during the process. An immediate advantage of the segregative approach is that the parallel time can be reduced by running more copies of the BGA in parallel, without any further changes.

As comparison, [10] reports the following execution times for three different exact methods, for some of the test instances, in seconds: *cordic* – 1.93 / 1.35 / 1.29; *ttt2* – 435 / 345 / 362. Results for execution times of exact methods on the large instances are not available.

Decreases in the value of E_c are obtained only by synchronizations during communication. The values, very close to 1, prove the efficiency of the communication protocol, which does not load the computations of the copies of BGA and control components with unnecessary overhead. The presented values are obtained with a communication rate, R, of 8 evolution stages, counting all copies of the BGA. Higher values for R, such as 10 and 12, produced almost the same results. Smaller values, such as 2 and 4, yielded worse processor utilization on small instances, where computations are very fast and a more intense communication overwhelms the system. In terms of quality and stability of the solution, different values for R do not produce significantly different results. As an empiric rule, R should be set to $p \times 2$ or $p \times 3$. Both Wilcoxon Signed-Rank non-parametric test and the two-tailed t-test with assumed unequal variances were applied to check the statistical significance of the differences between the averages of the absolute error (m) and E_c, obtained on each test instance, for the tested values for R. Only R was varied, all other parameters were the same as the ones used in the performance evaluation. All tests confirmed the presented conclusions with $p < 0.01$.

6.3.2 Communication issues

This subsection presents some issues regarding data flows at different levels of the segregative approach, in order to illustrate the effects of the communication protocol. All data corresponds to an execution on the *cordic* problem instance. The parameters are the same as the ones used in the performance evaluation.

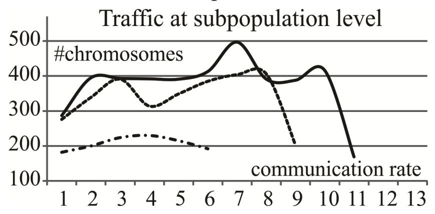

Figure 3. Data flows at subpopulation level

Figure 3 presents the number of chromosomes transferred by three parallel evolving subpopulations towards the fourth one. The moments of communication are multiples of the communication period, R. The dynamic adjustment of the genetic operator rates makes the communication at subpopulation level hard to predict. The data does not indicate a classical communication pattern for algorithms with multiple populations, such as an intense communication in the beginning of the evolution and then a decreasing trend towards the end.

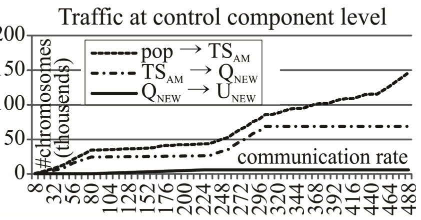

Figure 4. Data flows at control components level

Figure 4 presents the traffic towards and between the control components of the segregative algorithm. The values on each of the three graphs represent the total amount of transferred chromosomes until some evolution stage, counting all copies of the BGA. The dotted line represents the number of chromosomes transferred by the copies of the BGA towards TS_{AM} (chromosomes which are not similar to any of the four currently explored subpopulations. The intensity of the communication is correlated with the intensity of the traffic at subpopulation level. Lower data flows between currently explored subpopulations produce a more intense transfer towards TS_{AM}. The dash-dot line represents the amount of chromosomes that are transferred by TS_{AM} towards Q_{NEW} (chromosomes that do not belong to any explored region of the search space). The traffic is more intense in the beginning of the execution, which is to be expected, taking into account that the exploration history recorded by TS_{AM} is initially void and becomes larger and larger during the execution. As the algorithm converges, the transfer towards Q_{NEW} stagnates. The solid line represents the amount of chromosomes that are stored by Q_{NEW} in U_{NEW}, the large unstructured population with individuals from unexplored regions of the search space. The trend is similar to the previously described one: the communication is more intense in the beginning of the execution and stagnates as the algorithm converges.

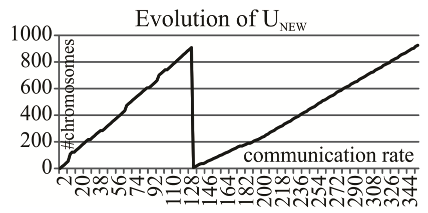

Figure 5. Amount of chromosomes entering U_{NEW}

Figure 5 depicts the evolution of the number of chromosomes in U_{NEW}. U_{NEW} is initially empty and it receives individuals until it reaches the maximum size, M_{UNEW}. After that the clustering procedure is applied and U_{NEW} is emptied. As the exploration performed by the algorithm becomes more extensive, U_{NEW} gets filled up more slowly.

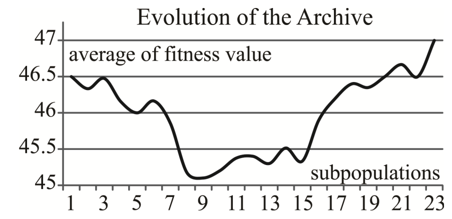

Figure 6. Average fitness in the Archive

Figure 6 presents the average fitness of the individuals in the archive, computed after the evolution of some subpopulation is finished and the archive is updated. Because of the random initialization of P_{init}, the initial values represent, in general, low fitness values. The values improve during the execution. The quality decreases towards the end because the algorithm also explores low quality subpopulations. The trend of the evolution suggests that PQ_{NEW} handles well the prioritization of subpopulations which are sent to the copies of BGA for exploration.

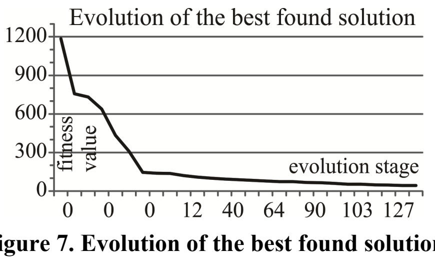

Figure 7. Evolution of the best found solution

Figure 7 presents the evolution of the best found solution during the execution of the algorithm. The starting value is of low quality, due to random initialization. The method greatly improves the solution, and very fast. Constant improvement towards the end of the evolution underlines the capacity of the algorithm to escape local optima.

7. CONCLUSIONS AND FUTURE WORK

In this paper a highly efficient dynamic segregative genetic algorithm for optimizing the order of variables in ROBDDs was presented. The approach deals with both representation and feature space. The exploration of the search space is achieved by evolving many subpopulations of individuals, which are induced by clustering in the feature space. The design of the algorithm is general and can incorporate any population based heuristic, as long as a similarity measure between individuals can be defined. An efficient communication protocol is introduced in order to maintain similarity inside the explored subpopulations and to focus the search on well-defined regions of the search space. The approach systematically avoids being trapped in local optima by using a tabu search associative memory. The exploration capabilities of the method are enhanced by a component that manages individuals from yet unexplored regions of the search space. Large quantities of promising individuals are managed and organized in subpopulations, in order to guide the search process towards newly discovered regions of the search space. The approach has a high potential for distributed computing and can be easily extended to solve problem instances in short time or to successfully solve very large instances that are intractable with exact methods.

Extensive evaluation on difficult benchmark problems shows a high level of performance of the algorithm in terms of quality and stability of the solution. Comparisons with single and multi-population heuristics are presented. The efficiency of the communication protocol and that of a distributed implementation are thoroughly analyzed.

Future research directions are focused on: devising a better feature vector, in order to better discriminate between highly similar individuals; integration of the branch-and-bound paradigm by evolving embryos instead of chromosomes; parameter tuning.

8. ACKNOWLEDGMENTS

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

Cristian Rotaru 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.

9. REFERENCES

[1] Bollig, B., Wegener, I. 1996. *Improving the variable ordering of OBDDs Is NP-complete*. IEEE Trans. on Computers, vol. 45, p. 993–1002.

[2] Friedman, S. J., Supowit, K. J. 1990. *Finding the optimal variable ordering for BDDs*. IEEE Trans. on Comp., vol. 39, p. 710–713.

[3] Fujii, H., Ootomo, G., Hori, C. 1993. *Interleaving based variable ordering methods for OBDD*. Int'l Conf. on CAD, p. 38–41.

[4] Meinel, C., Slobodova, A. 1997. *Speeding up variable reordering for OBDDs*. Int'l Conf. on Computer Design, p. 338–343.

[5] Panda, S., Somezi, F. 1995. *Who are the variables in your neighborhood*. Int'l Conf. of CAD, p. 74–77.

[6] Rudell, R. 1993. *Dynamic variable ordering for ordered binary decision diagrams*. Int'l Conf. of CAD, p. 42–47.

[7] Bollig, B., Lobbing, M., Wegener, I. 1995. *Simulated annealing to improve variable orderings for OBDDs*. Int'l Workshop on Logic Synth., p. 5b:5.1–5.10.

[8] Drechsler, R., Becker, B., Göckel, N. 1996. *A genetic algorithm for variable ordering of OBDDs*, IEEE Proceedings, 143(6), p. 363–368.

[9] Lenders, W., Baier, C. 2005. *Genetic algorithms for variable ordering problem of BDDs*. Lect. Notes in Comp. Sc., Springer, p. 1–20, vol. 3469/2005.

[10] Ebendt, R., Günther, W., Drechsler, R. 2005. *Combining ordered best-first search with branch and bound for exact BDD minimization.* IEEE Trans. on CAD of Integ. Circuits and Syst. 24(10), p. 1515–1529.

[11] Brudaru, O., Ebendt, R., Furdu, I. 2010. *Optimizing variable ordering of BDDs with double hybridized embryonic genetic algorithm*. Proc. of The 12th Int. Symposium on Symb. and Num. Algorithms for Sc. Comp., Synasc 2010, p. 167–173.

[12] Rotaru, C., Brudaru, O. 2012. *Multi-grid cellular genetic algorithm for optimizing variable ordering of ROBDDs*. Proc. of the 2012 IEEE Congress on Evolutionary Computation, IEEE CEC 2012, in press.

[13] Brudaru, O., Rotaru, C., Furdu, I. 2011. *Static segregative genetic algorithm for optimizing variable ordering of ROBDDs*. Proc. of The 13th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Synasc 2011, p. 222–229.

[14] Brudaru, O., Rotaru, C. 2010. *Dynamic segregative genetic algorithm for assembly lines balancing*. Proc. of The 12th Int. Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Synasc 2010, p. 229–236.

[15] Bryant, R.E. 1986. *Graph-based algorithms for Boolean function manipulation*. IEEE Trans on Comp. 35(8), p. 667–691.

[16] Furdu, I., Brudaru, O. 2009. *New hybrid genetic algorithm with adaptive operators and variability target for optimizing variable order in OBDD*. Proc. of "Gh. Vranceanu" Int'l Conference on Mathematics and Informatics ICMI 2, nr. 2 , p.156–172, vol. 19/2009.

[17] Goldberg, D.E. 1989. *Genetic algorithms in search, optimization and machine learning*, Addison Wesley.

[18] Butler, K.M., Ross, D., Kapur, R., Mercer, M.R. 1991. *Heuristics to compute variable orderings for efficient manipulation of OBDDs.* Proc. of the 28th ACM/IEEE Design Autom. Conf., p. 417–420.

[19] Malik, S., Wang, A.R., Brayton, R.K., Sangiovanni-Vincentelli, A. 1988. *Logic verification using BDDs in a logic synthesis environment.* Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design, p. 6–9.

[20] Fujita, M., Fujisawa, H., Kawato, N. 1988. *Evaluation and improvements of Boolean comparison method based on BDDs*. Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design, p. 2–5.

[21] http://cadlab.cs.ucla.edu/~kirill/, accessed June 2010.

[22] CUDD package URL: vlsi.colorado.edu/~fabio/CUDD.