A Massive Parallel Processing Genetic Algorithm for ROBDD Construction

Josep M. Garrell i Guiu, Department of Computer Science, Enginyeria La Salle, Universitat Ramon Llull, Barcelona, Spain , and *Miquel Bertran i Salvans,* Department of Computer Science, Enginyeria La Salle, Universitat Ramon Llull, Barcelona, Spain

ABSTRACT: *A Reduced Ordered Binary Decision Diagram (ROBDD) is a data structure that has been widely used in CAD tools for VLSI logic synthesis design. ROBDDs have some good properties and some drawbacks. The most important drawback is that the size of a ROBDD depends on input variable ordering.*

The problem of finding the best input variable ordering can be modeled as a Travelling Salesman Problem (TSP). In order to find such optimum ordering, Genetic Algorithms (GAs) are used. Due to the high computational cost, a parallel approach to GAs on a Massive Parallel Processing (MPP) system is followed.

1 Introduction

In most CAD tools for synthesis and verification of VLSI systems, one finds a common problem: the high computational complexity when manipulating logic functions. For this reason, it is necessary to find a way of simplifying the representation of logic expressions. Recently, the scientific community began to use a data structure known as *Binary Decision Diagram* (BDD). The main idea is relatively old, and could be traced back in 1959 with the work of Lee [11]. After that, in 1978, BDDs are found in the work of Akers [1]. Since the publication of Bryant's work in 1986 [3] [2] the scientific comunity began to use BDDs widely.

Reduced Ordered BDDs (ROBDDs) are a kind of BDDs that will be covered later on. Representation of logic functions with ROBDDs has many advantages, but there are also some drawbacks. The most important one is that the number of nodes of a ROBDD (and its size) depends on logic function input variable ordering. Obviosly, we are interested in finding the ROBDD with minimum size that represents a given logic expression. So, we are interested in finding a permutation of the input variables that minimizes ROBDD size. From this point of view, the problem can be modeled as a Travelling Salesman Problem (TSP).

One of the methodologies that can solve the TSP is Genetic Algorithms (GAs). Since the time for building a ROBDD can be relatively high, a parallel implementation of GAs has been used. In such a way, and thanks to a specific MPP developement system, execution time has been reduced considerably.

This paper is organized as follows. First, the ROBDD data structure will be introduced, together with its advantages and drawbacks. Next, a brief introduction to GAs will be given. After that, some different versions of GAs will be discussed. Next, the chromosomic representation and the genetic operators we chose will be explained. Finally, some different results obtained from our optimizations will be presented.

2 Reduced Ordered Binary Decision Diagrams

Let us introduce ROBDD through examples. Supose we have a logic function represented by the following sum-of-products:

$$f = abc + b'd + c'd$$

In figure 1 a posible BDD representing this logic function can be seen.

In order to evaluate this logic function for a set of values of its input variables a, b, c and d, it is necessary to follow a path from the initial node (the square one labeled f), into another square node, the *zero*[1] or the *one*[2] node. For instance, supose we want the evaluation of function f at $a = 1$, $b = 0$, $c = 1$ and $d = 0$. We will begin the path at the root node (the initial one). After that, we will reach a node with the a label. This means that we must *explore* the value of variable a. Depending on this value, we must choose one of its two output edges. Since $a = 1$, we choose the edge with label T ($True$). In this way, we can continue creating a path until we reach a terminal node. If this

[1] False.
[2] True.

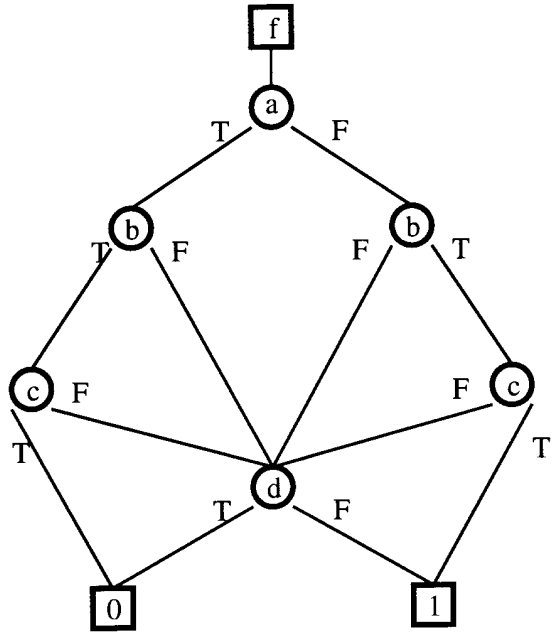

Figure 1: BDD corresponding to logic function $f = abc + b'd + c'd$. The input variable ordering is $a \leq b \leq c \leq d$.

terminal node is labeled with a *zero*, the value of the logic function will be *False*. In the other case, the terminal node is labeled with a *one*, and the value of the function will be *True*.

BDDs used up to now are *Ordered* BDDs (OBDDs). The reason for this name is that if one follows any path from the root to a terminal node, the input variables are found in the same order. In our example the order is

$$a \leq b \leq c \leq d$$

The shape and size of a BDD (in the sense of number of nodes), depends on the exploration order of the input variables strongly. Supose, for example, that we build the BDD corresponding to the same logic function but using the following exploration order: $b \leq c \leq a \leq d$. Figure 2 shows the result. The resulting BDD is as small as it can be for that logic function.

Due to all that, it is absolutely necessary to find a method for searching the best input variable ordering. In this paper we present the results of aplying Genetic Algorithms to solve that problem.

In order to construct the BDD of a logic function, we will use the Shannon Expansion[3]. The Shannon Expansion of

[3]Sometimes this expansion is named *Boole Expansion.*

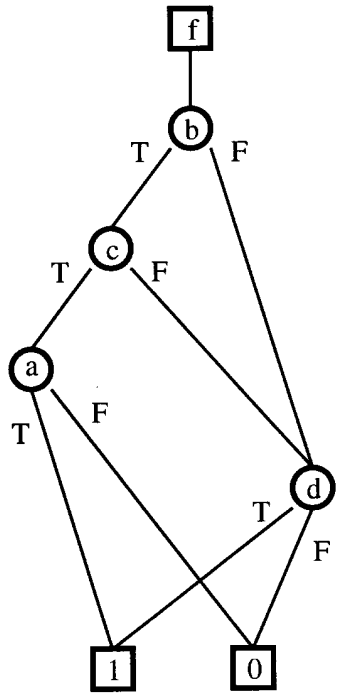

Figure 2: BDD corresponding to logic function $f = abc + b'd + c'd$. The input variable ordering is $b \leq c \leq a \leq d$.

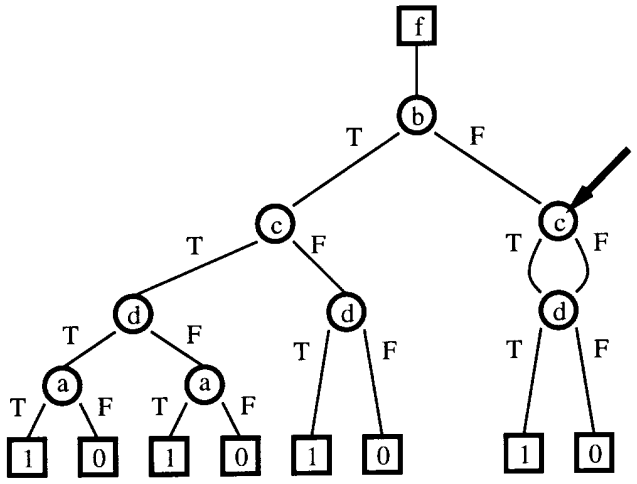

Figure 3: Non-reduced BDD. The presence of some redundant nodes and some isomorfic subgraphs can observed.

a logic function is:

$$f = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$$

for any input variable x_i.

A recursive method for constructing BDDs can be found directly from this expansion. The BDD of a logic function contains a node labeled with the first variable we explore. This node has two output edges. Each edge going to the BDD corresponding to its cofactors in the above expansion.

We are interested in a special kind of OBDD, the *Reduced* OBDD (ROBDD). This kind of BDD represents a logic function canonically. This is the main advantage of using ROBDDs for this representation. A OBDD is *Reduced*, if it has neither isomorfic subgraphs nor redundant nodes. A non-reduced BDD can be seen in figure 3. Some isomorfic subgraphs and some redundant nodes may be seen. Our construction method guarantees that the resulting BDD will be *Reduced*.

3 Fundamentals of Genetic Algorithms

Genetic Algorithms (GA) amount to an optimization technique introduced by John Holland [10] in 1975. It is based on the natural evolution of species and Mendel's genetic theory.

As Dorigo [4] said, it is possible to establish an analogy between GAs and natural evolution. It is posible to view an individual as a solution to one problem: the existence problem. From this point of view, when we work with GAs, the solution to a problem will be called an *individual* or a *chromosome*. A set of individuals forms a *population*.

Natural evolution may be viewed as a *selective force* that explores potential solutions. Through generations, the *selective force* will bring the population of individuals towards regions of the search space where the average *fitness* of the population is better. This will be possible because the *selective force* favours the survival of the fitest individuals, and penalizes the worst ones.

When talking about GAs, people use many words from biology, like *chromosomes*, *genes*, *individuals*, *crossover*, *mutation*, etc.

The typical structure of a GA is the following:

Genetic Algorithm

```
t:=0;
init_population_P(t);
evaluation_P(t);
WHILE (not end-condition) DO
     t:=t+1;
     select_P(t)_from_P(t-1);
     crossover_P(t);
     mutation_P(t);
     evaluation_P(t);
END_WHILE
```

It is easy to observe that the algorithm works with a population P that varies in time. This population is initialized and evaluated before entering the loop. The loop will be repeated until an *end-condition* is satisfied. Inside the loop, the process is very simple: a new population from the old one is selected (this process simulates the natural death of some individuals), the individuals will be crossed, then the mutation that takes place during the reproduction phase we will be simulated, and, finally, the new population obtained will be evaluated.

The *end-condition* of the loop, can be either a maximum number of iterations or a convergence measure. This process is represented graphically in figure 4.

4 Parallel Genetic Algorithms

Given their inspiring principle of parallel evolution of a population of individuals[4], GAs are good candidates for effective parallelization. However, one must be carefull with the traditional method (the Holland algorithm), because it needs a centralized control process. This centralization is particularly required during the selection procedure. For that reason several methods have been introduced, with the main goal of exploiting its intrinsic parallel nature. We will briefly review three possible ways of parallelization: the Holland Approach, the Island Approach and the Neighbourhood Model.

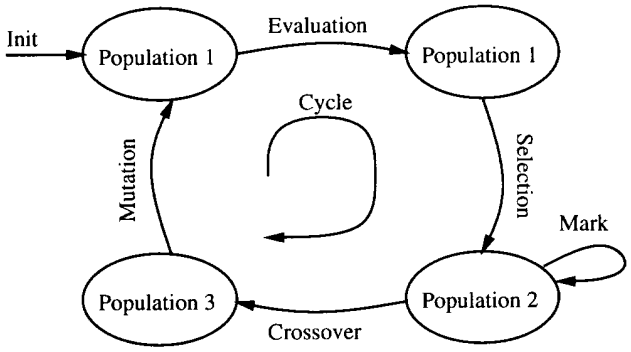

Figure 4: The Genetic Algorithm main cycle.

- Holland Approach

  The idea is very simple. The operations that can be carried out in parallel are: evaluation of individuals, crossover an mutation. We will only do in parallel the evaluation phase. The resulting parallel structure is the typical *leader-servers* structure. The *leader* process runs all the GA, while the *servers* only receive evaluation orders, execute them and, finally, return the result to the *leader*. If the number of *servers* is zero, the *leader* will behave as a *server*, and one has a sequential running. If the number of *servers* is greater than zero, the *leader* will order the different evaluations to them.

- Distributed models for genetic evolution: the Island Approach

  In this approach we will have several isolated subpopulations evolving in parallel. Periodically, the different isolated subpopulations interchange the best individuals using a process called *migration*.

  The idea comes from the observation, reported in biology, that isolated environtments, such as islands, often produce animal species that are more specifically adapted to the peculiarities of their environtments than to corresponding areas of wider surfaces. This observation has given rise to the so-called niching and specification theory [8] and has inspired the GA community with new operators and architectural issues.

- Distributed models for genetic evolution: the Neighbourhood Model

  In this model only one population evolves. Each individual of the population will be placed in a cell of a toroidal bidimensional grid. The genetic operators selection and crossover can be applied between individuals placed in neighbour cells of the grid only. A distributed model results, with no need of a centralized control process.

  For each cell of the grid, the algorithm will apply the selection process to the individuals assigned to this grid plus the individuals assigned to the neighbour cells. The crossover operation will create couples of individuals from the individuals of the cell and its neighbours. Finally, the mutation process will take place locally as it happens in any other model.

We have tested the Holland and Island approaches.

5 Chromosomic Representation and Genetic Operators

Since our problem can be modelled as a *Travelling Salesman Problem* (TSP), we must apply a modified GA for solving TSP problems. This means that the chromosomic representation and the genetic operators will be special.

The chromosomic representation we use is the *Path Representation* [12]. This representation is the easiest one we can choose. For instance, the path:

$$5 - 1 - 7 - 8 - 9 - 4 - 6 - 2 - 3$$

will be represented by the list (5 1 7 8 9 4 6 2 3).

There are three posible standard crossover operators for that chromosomic representation[12], although it is possible to find a bigger set in Syswerda[16], Goldberg[7], Fox[5], Oliver[13], Starkweather[14] and Syswerda[15]. We have chosen the PMX (*Partially-Maped Crossover*)[9]. This crossover builds an offspring by choosing a subsequence of a tour from one parent and preserving the order and position of as many cities as possible from the other parent.

6 Simulation Results

In the problem of finding the best input variable ordering for a ROBDD, the evaluation of the individual's fitness reduces to constructing the ROBDD corresponding to a logic function whose input variable ordering is represented by the individual. It is easy to observe that the problem has a high computational cost. Due to this fact, we solve the problem using a parallel approach to GAs on a Massive Parallel Processing (MPP) system. In particular we have been using the Cray T3D MPP system from the Pittsburgh Supercomputing Center (Carnegie Mellon University, University of Pittsburgh). The code is written in ANSI C using standard and portable PVM. The work at the Pittsburgh Supercomputing Center was published at [6].

PEs	Execution Time	Speedup	Efficiency
1	440.765	1	
2	244.835	1.8	90%
4	143.033	3.081	77%
8	96.001	4.591	57.38%

Table 1: Execution time (in seconds),speedup and efficiency corresponding to the Holland version of a GA with 50 individuals in the population.

PEs	Execution Time	Speedup	Efficiency
1	563.688	1	
2	293.601	1.919	95.95%
4	158.674	3.552	88.8%
8	87.708	6.426	80.32%

Table 2: Execution time (in seconds) speedup and efficiency corresponding to the Holland version of a GA with 400 individuals in the population.

Next the results obtained with several executions is going to be presented, together with the speedup and convergence of the algorithms. We will also present some results of tunning the convergence of the Holland approach.

6.1 Speedup

We will begin analysing the results of the Holland approach. Tables 1 and 2 show the execution time, speedup and efficiency obtained solving a problem with 15 input variables. Table 1 corresponds to a execution with 50 individuals in the population, while table 2 corresponds to an execution with 400 individuals in the population. It is important to observe that the algorithm behaves as expected: the speedup improves as the population grows. This happens because, as the population grows, the percentatge of parallelizable code also grows, so that there is more work for the *servers*. Figure 5 shows the speedup from tables 1 and 2.

Table 3 shows the execution time, speedup and efficiency obtained solving a problem with 15 input variables and using the Island approach of a GA. Due to the high number of PEs used (up to 128), and with respect the results of the Holland approach, the results obtained here can be considered as MPP. Figure 6 shows the speedup from table 3. Obviosly, the algorithm is a good candidate for runing in a MPP environtment. It shows a high efficiency for very different numbers of PEs. It is important to observe that the efficiency is, more or less, constant from 16 to 128 PEs.

Finally, table 4 and figure 7, shows the best speedups

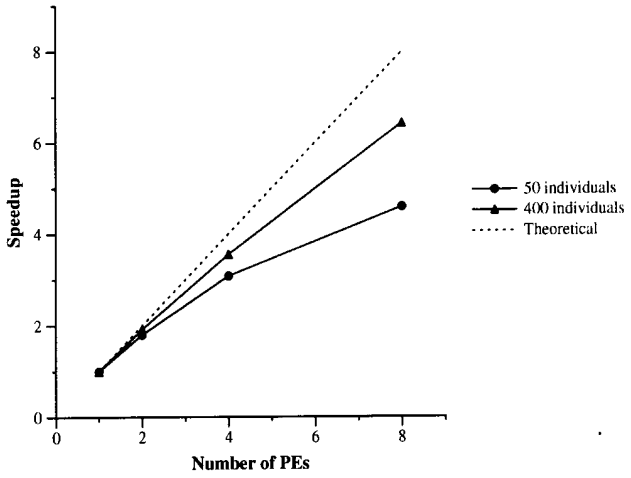

Figure 5: Speedup from the Holland version of a GA.

PEs	Execution Time	Speedup	Efficiency
1	538.790	1	
2	257.543	2.09	104.5%
4	128.453	4.19	104.75%
8	71.912	7.49	93.6%
16	45.096	11.94	74.6%
32	24.843	21.68	67.75%
64	11.087	48.59	75.92%
128	5.766	93.44	73%

Table 3: Execution time (in seconds), speedup and efficiency corresponding to the Island version of a GA.

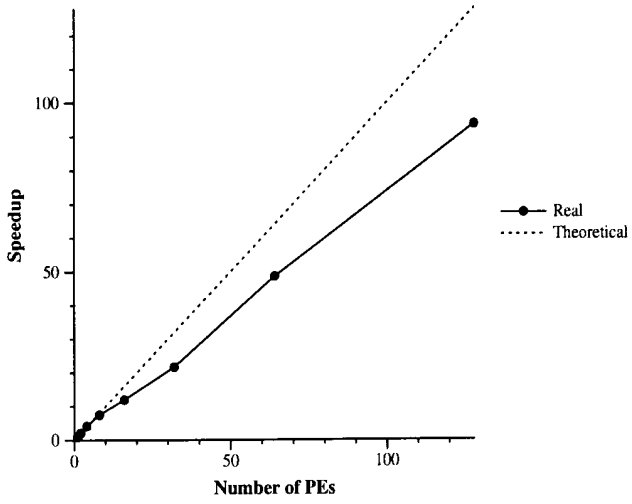

Figure 6: Speedup from the Island version of a GA.

PEs	Holland's Speedup	Island Speedup
1	1	1
2	1.919	2.09
4	3.552	4.19
8	6.426	7.49

Table 4: Comparison between the speedup of the Holland version and the Island version of a GA.

of the Holland and Island approaches. We have used the same scale in order to compare both approaches.

6.2 Convergence

In order to show the convergence of the algorithm, we will use the same logic function we have been using for the speedup. It is a logic function with 15 input variables. Figure 8 shows the number of nodes needed for the logic function using the Holland approach of a GA. The convergence of the algorithm is better as we use populations with a higher number of individuals.

Figure 9 shows the behaivor of the Island approach to GAs. It can be seen that, normally, as the number of islands grows (and also the number of PEs), convergence improves.

6.3 Tunning

One of the first things that GA users realize is that the behaivour of the algorithms strictly depends on its control parameters. The highest dependence is on crossover

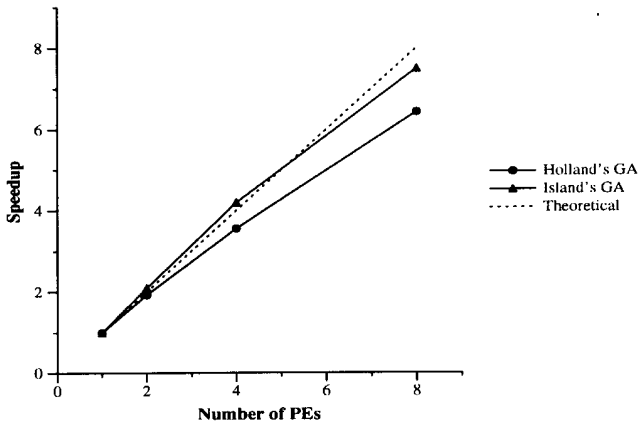

Figure 7: Comparison between the speedup of the Holland version and the Island version of a GA.

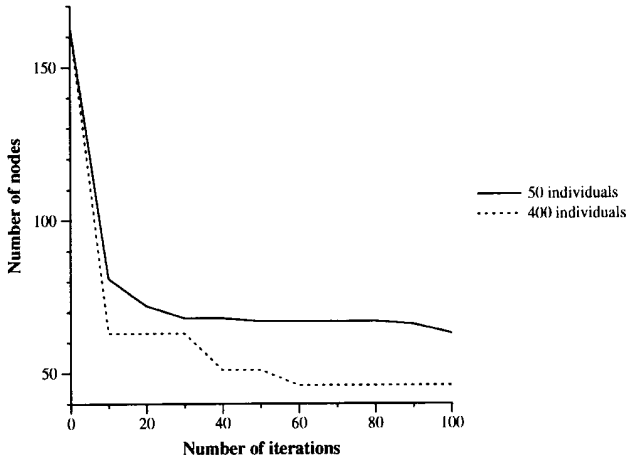

Figure 8: Evolution of the number of nodes needed for synthesizing a logic function, versus population size. Using a Holland version of GAs.

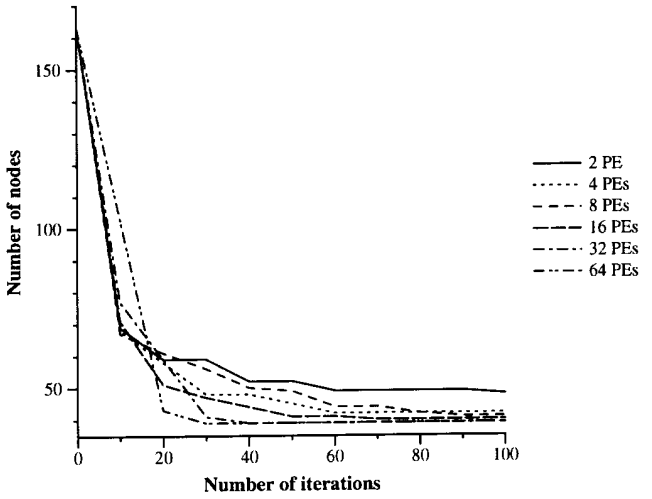

Figure 9: Evolution of the number of nodes needed for synthesizing a logic function, versus population size. Using an Island version of a GA.

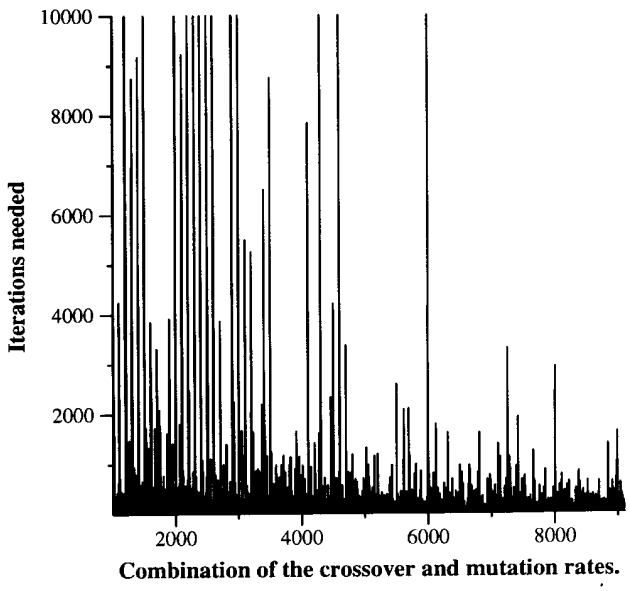

Figure 10: Convergence of the Holland version of a GA depending on the crossover and mutation rates.

and mutation rate values. For that reason we decided to test the convergence of the algorithm with different values of crossover and mutation rates. We chose a very simple problem for doing the optimization, and we chose the Holland approach of the GA. The crossover rate values are taken from the interval $[0.1, 0.9]$, and the mutation rate values from the interval $[0.001, 0.05]$. For each couple of values we ran the algorithms until a solution 80% off the best one was reached. If the algorithm did not find the solution, it was halted at 10.000 iterations.

Figures 10 and 11, show the obtained results. The x-axis represents a combination of crossover and mutation rates. The first two digits are the crossover rate, and the last two ones the mutation rates. The y-axis reprents the number of iterations needed for reaching the solution. If the number of iterations needed reach 10.000, it means that the algorithm does not converge and the runing has been stoped. The second figure is a zoom of the first one, corresponding to the values of crossover rate from the interval $[0.25, 0.26]$.

It is possible to see that for finding a good convergence, we must chose a value of crossover rate relatively high. Also, the mutation rate must not be too small.

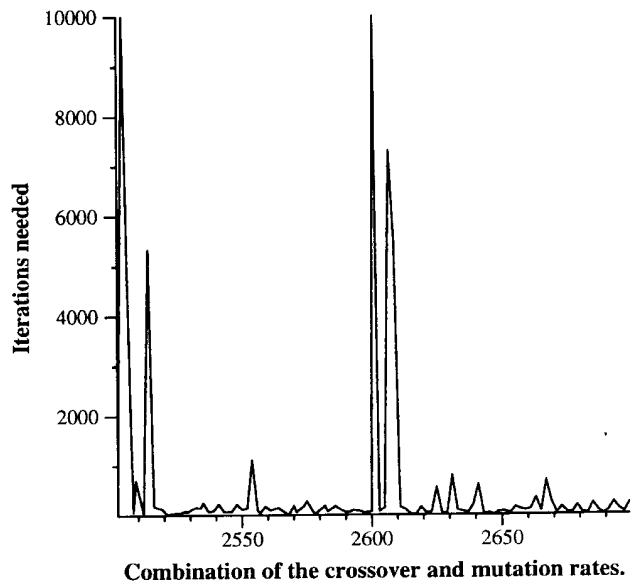

Figure 11: Convergence of the Holland version of a GA depending on crossover and mutation rates.

7 Conclusions

In this paper we have presented a study of the efficiency and adaptation of different versions of GAs in a MPP environtment optimizing the size of ROBDD. We showed how the Island approach to GAs is better than the Holland approach when using a MPP system. This version attains

efficiencies of 70% for different and high number of PEs.

From the convergence point of view, we showed that if the number of individuals and islands grows, the convergence improves. We also have shown that the convergence will be better if we chose some relatively high value of the crossover and mutation rates.

Finally, we would like to point out the high decrease in the execution time we observe using parallel algorithms instead of sequential ones. For instance, in some cases we presented, the execution time goes from 8 minutes and 59 seconds to 5.7 seconds.

8 Acknowledgements

We would like to thank Cray, Inc., Fundació Catalana per la Recerca and Enginyeria La Salle (Universitat Ramon Llull) for all their support.

References

[1] S.B. Akers. Binary decisions diagrams. *IEEE Transactions on Computers*, 27:509–516, 6 1978.

[2] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th. Design Automation Conference*, pages 40–45, 1990.

[3] R.E. Bryant. Binary decisions diagrams. *IEEE Transactions on Computers*, 35:677–691, 8 1986.

[4] M. Dorigo and V. Maniezzo. *Parallel Genetic Algorithms*, chapter Parallel Genetic Algorithms: Introduction and Overview of Current Research, pages 5–42. IOS Press, 1993.

[5] B.R. Fox and M.B. McMahon. Genetic operators for sequencing problems. In *Proceedings of the First Workshop on the Foundations of genetic Algorithms and Classifier Systems*, pages 284–300, 1991.

[6] J.M. Garrell. Microelectronic circuit synthesis tool for VLSI technology using switch-level logic synthesis. In *Directory of Cray Sponsored University Research and Development Grants*, page 97, 1 1995.

[7] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, 1989.

[8] D.E. Goldberg. A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 6:445–460, 1990.

[9] D.E. Goldberg and R. Lingle. Alleles, Loci and the TSP. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159. Lawrence Erlbaum Associates, 1985.

[10] John H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, 1992.

[11] C.Y. Lee. Binary decisions diagrams. *Bell Systems Technical Journal*, 38:985–999, 4 1959.

[12] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1992.

[13] I.M. Oliver, D.J. Smith, and J.R.C. Holland. A study of permutation crossover operations on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 224–230. Lawrence Erlbaum Associates, 1987.

[14] T. Starkweather, S. McDaniel, K. Mathias, C. Whitley, and D. Whitley. A comparision on genetic sequencing operators. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236. Morgan Kaufmann Publishers, 1991.

[15] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann Publishers, Inc., 1989.

[16] G. Syswerda. *Handbook of Genetic Algorithms*, chapter Schedule Optimization Using Genetic Algorithms, pages 332–349. Van Nostrand Reinhold, 1991.