Is the Island Model Fault Tolerant?

J.I. Hidalgo
Complutense University
Facultad de Informática
28040 Madrid, Spain
+34913947537
hidalgo@dacya.ucm.es

F. Fernández-de Vega
University of Extremadura
Centro Universitario Mérida
06800 Badajoz, Spain
+34924387068
fcofdez@unex.es

J. Lanchares
Complutense University
Facultad de Informática
28040 Madrid, Spain
+34913947542
julandan@fis.ucm.es

D. Lombraña-González
University of Extremadura
Centro Universitario Mérida
06800 Badajoz, Spain
+34924387068
daniellg@unex.es

ABSTRACT

In this paper, we present a study on the fault tolerance nature of the island model when applied to Genetic Algorithms. Parallel and distributed models have been extensively applied to GAs when researchers tackle hard problems. The idea is both to reduce computing time while also improving diversity of populations and therefore quality of solutions. Nevertheless, there are few works dealing with the problem of faults that are usually present when a distributed infrastructure is employed for running the parallel algorithm. This paper studies the behavior of the Island Model when faults appear on a parallel computer or a network of computers. Two benchmark problems have been employed, and good results obtained for each of them allow us to reliably consider Island Model as a fault tolerant parallel algorithm.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search; Heuristic Methods.

General Terms: Management, Measurement, Experimentation, Performance, Reliability.

Keywords: Distributed GAs. Fault tolerance.

1. INTRODUCTION

During the last few years, several researchers have extensively studied and applied parallel versions of the Genetic Algorithms (GAs). The idea behind the parallelization of any Evolutionary Algorithm is both to improve the quality of solutions that are to be found and also reduce the time required for obtaining those solutions.

Greffensttete was the first to show the convenience of parallelizing a GA [11], and since then many researchers have find the idea of interest. Remarkably, Cantu-Paz presented a complete study showing the advantage of the parallel model, and describing how the main parameters of the algorithm can largely influence the results obtained [10,16].

GAs can be easily parallelized at two different levels: parallelizing individuals' evaluation or employing islands for evolving semi-isolated populations. At the individual level, each candidate solution is usually evaluated in a different worker node while for the island model each worker node evaluates a whole population –the island: a subpopulation of the original one. These two models can nowadays be applied with little effort by using existing GA tools.

Yet, few researchers have taken into account an important feature of many parallel and distributed systems: the presence of processing or communication faults that may affect –or even break - the execution of an algorithm. Although massively parallel computers are trusty enough for running applications without considering the emergence of faults, things change significantly when distributed systems are taken into account. When a large scale application is to be deployed harnessing volunteers PCs on wide area networks, for instance, one of the major concerns is Fault Tolerance. Faults are a key feature of network of computing resources. Faults might also appear on parallel computers, and algorithms must be ready for handling them.

Fault tolerance is the ability of a system to continue working in a well defined way despite of the errors/faults that can occur in hardware or software [13]. When we are talking about fault tolerance in parallel or distributed computing we are defining the reliability that we will have in a system when handling faults.

There are different approaches for providing fault tolerance in a distributed system: by using check points, n-version algorithms, rejuvenation, epidemic algorithms and redundancy [3, 4, 5, 6, 15]. Fortunately, there are different general purpose frameworks available that embody some of these techniques: Calypso [7], Condor [3], Starfish [12] and Boinc [5], to name but a few. Moreover, there have been developed some specific Parallel EAs frameworks that can be employed by researchers interested in Evolutionary Computation, such as Beagle [8] and Dream [9, 14, 15] which also include some of the above referred techniques, aimed at avoiding faults on parallel or distributed systems. Nevertheless, although those frameworks can recover from failures, the complexity of these tools has largely increased, as well as the computing time required for running the algorithms, due to the overhead of the techniques included. Table 1 shows a list of frameworks, both generic and EA specific, together with the fault-tolerant technique included.

In this paper we present a study devoted to test the fault tolerance of the island model on GAs when executing them on a distributed system. We have analyzed the quality of the solutions with and without considering processor faults - simulated by stoping some of the processors available. In this way we try to know under what conditions would not be necessary to apply any of the above referred techniques, thus obtaining a significant reduction in the execution time.

Therefore, in this paper we propose a non-conventional approach for managing faults when the island model is employed: instead of trying to handle the failures, we will ignore them and continue the execution of the algorithm, and we will try to see whether results are of similar quality as when faults are not present. If that is the case, we will conclude that the time required for controlling faults by means of specific techniques could be saved. To our best knowledge, this kind of solution has been never considered for the Island Model when using EAs. A similar idea considered for a parallelization at individual level has only been described before for GP & GAs (see for instance PGA at individual level [17], and PGP [2 ,19]). This paper thus studies the fault-tolerant nature of Parallel GAs when the Island Model is employed.

Table 1. Fault-tolerant frameworks.

Framework	Techniques
Beagle	Redundancy
Java	Check points
Calypso	Specific techniques
Boinc	Redundancy
Dream	Epidemic Algorithms
Starfish	Checkpoint

The rest of the paper is structured in the following way: Section 2 presents a brief summary of the Island Model. Section 3 describes the methodology and experiments performed. Section 4 analyses the experimental results and finally some conclusions and future research lines are depicted.

2. THE ISLAND MODEL

The Island Model is a structured model inspired by nature, which tend to spatially organize populations [1, 2]. This model includes different sub-populations or demes geographically separated. The idea behind this model is to explore different areas of the search space and at the same time it tries to maintain a higher diversity within each sub-population. For achieving the expected behavior individuals are exchanged between sub-populations with a given frequency. A rate of migration defines how many individuals will be exchanged, with which frequency. A migration topology is also defined for establishing destinations of migrating individuals.

Figure 1 shows the island model considering a ring topology; the demes will exchange a number “n” of individuals each “N” generations.

3. METHODS AND EXPERIMENTS

This section explains the methods and experiments performed. First, we describe the way we simulate a failure on a processor, and then we explain the two kinds of problems analyzed: Fmodal and Schwefel functions. Both are multimodal functions that search for global optima (maximum or minimum) on a search space with several local optima. Typically, they are described by trigonometric equations [20].

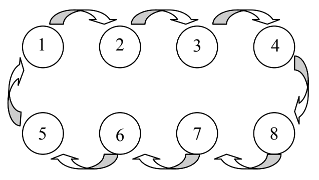


Figure 1: Island Model for Parallel GA

Even when we focus on only a kind of problem, the “no free lunch" theorem [21, 22] tell us that, if we compare two searching algorithms with all possible functions –problems-, the performance of any two algorithms will be, on average, the same. So if we try to design a perfect test set, where all the possible functions are present in order to determine whether an algorithm works or not for every function, we surely make a mistake. Therefore we have set out our experiments focused only on a kind of problems for this step of the research.

We have thus selected a couple of separable multimodal functions, which are well known in GA literature [23]. We will thus obtain conclusions for this kind of problems, although results may be extended to a wider set of functions and problems later. A function is multimodal if it has two or more local optima. A function of variables is separable if it can be rewritten as an addition of functions of just one variable

3.1 Faults simulation

In our experiments we have employed a multiprocessor computer (Proliant ML570 G2, HP) with 8 core processors (4 Intel ® Xeon™ MP CPU 2.80 GHz, stepping 05, dual core processors). We have therefore employed for experiments up to 8 subpopulations within the Island Model.

We implemented a coarse grained parallel GA, with the same structure as described in section 2. The population is divided into a number of subpopulations or demes, and each of these relatively large demes evolves separately on different processors. Exchange between subpopulations is possible via a migration operator. Although many topologies can be defined to connect the demes we have employed the well known ring topology, and migration is restricted to neighbouring demes.

Given the main goal of this research, and given that the multiprocessor system is reliable enough to neglect the presence of random failures on processors during experiments, we had to simulate faults on processor: we did it by closing the communication between a given processor and their neighbours. Therefore that processor abandons the experiment, and the remaining ones redefine their communicating neighbourhood reducing the ring consequently. Figure 2 shows an example of a fault on processor number 4.

As we will see below, we have performed a series of experiments considering different number of processor faults: the larger number of processors failing, the larger number of islands that disappear from the experiment, and consequently the smaller number of individuals that keeps working and helping to find good solutions. Furthermore: every time a processor fails, the best individual may be lost. In the following section we show results obtained when 1, 2 or 4 processors fails (we have consider up to 50% of processors failing, which is an extremely an abnormal fault rate for a parallel or distributed system).

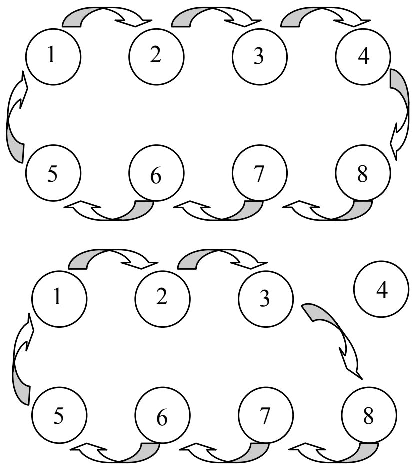


Figure 2: Fault simulation on processor #4

3.2 Fmodal Function

Fmodal function is a trigonometric function that is frequently employed as a benchmark for GAs [24]. *Fmodal* is defined by equation 1. It is a 10 variable function and each 50 bits of the chromosome give us the value of one variable. We have employed a 500-bit chromosome. Figure 3 represents Fmodal function for 2 and 3 variables.

$$F = -\sum_{i=1}^{10} x_i \cdot \sin(10 \cdot \pi \cdot x_i) + 1 \qquad (1)$$

3.3 Schwefel´s function

Schwefel's function is another multimodal function and is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction. We have also evaluated a 500-bit chromosome taking 50 bit for each variable, in order to evaluate a 10 variable function. Figure 4 represents Schwefel´s function for 2 and 3 variables.

$$F = -\sum_{i=1}^{10} x_i \cdot \sin\left(\sqrt{|x_i|}\right) \qquad (2)$$

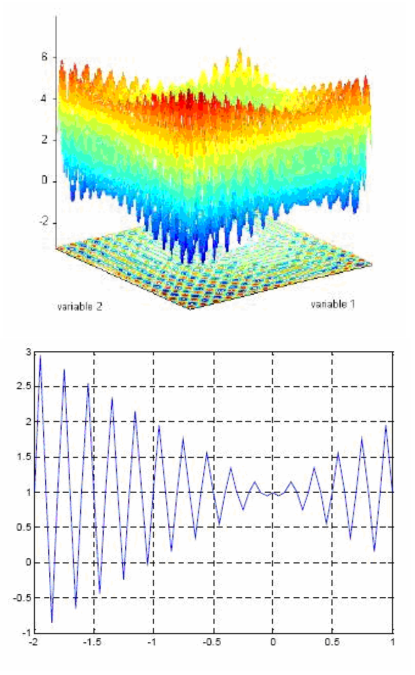


Figure 3: Fmodal Function

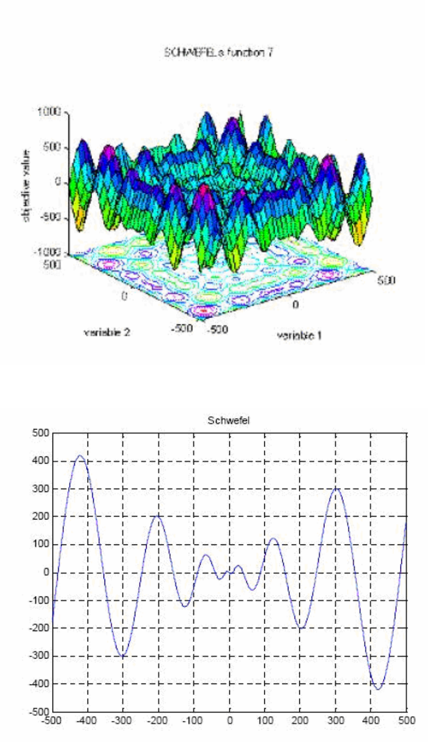


Figure 4: Schwefel function

4. EXPERIMENTAL RESULTS

In this section we describe the experimental results we have obtained when simulating processors' failures when the parallel GA is run. We have tested each of the experiments with several values of a set of important parameters:

- Number of processors in the experiment: The initial number corresponds to the number of islands at the beginning of the experiment. Experiments have considered values varying from 2 to 8 processors. As said in previous section, faults have been simulated by removing communication links between processors, and up to 50% of processors employed for a given experiment.
- Migration frequency: Several values have been tested: 10, 25, 50 or 100 generations computed before the migration step. Best individual from each of the processor is selected for migrating.
- Processor failures: We have simulated a processor failure every 50, 100 or 200 generations. Actually, the faults are simulated by affecting communications so that the isolated processor continues computing results independently. We don't care about the results obtained by isolated populations. Every experiment has also been computed without faults, and this latter case is employed for comparing results.

We have used a Parallel GA library: PGAPack [25]. PGAPack is a parallel genetic algorithms library that provides most capabilities desired in a genetic algorithms package. The features of PGAPack that encouraged us to use it are: Parameterized population replacement, multiple choices for selection crossover and mutation operators, and large set of example problems. Some minor changes were required to fit our experiments. To build a parallel version you must have an implementation of the Message Passing Interface MPI for the parallel computer or workstation network you are running on. We have used the MPICH implementation of MPI which is freely available [26]. We have fixed a total population size of 512 and this number of individuals is distributed equally into the number of processors – demes- employed for each of the experiments: if we use 2 processors, we have two population of 512/2, 512/4 if we employ 4 processors instead, and so on. The rest of the parameters for the algorithm are the following ones:

- Crossover probability: 0.8
- Mutation probability: 0.002
- Initialization of the population: Random
- Elitism
- Stopping criteria: Solution is found or a maximum of 5000 generations are computed.
- Remaining parameters: Tournament Selection, Two-Point Crossover, Fitness Raw (see PGAPack standard parameters [25]).

Every experiment has been repeated 100 times employing a different random seed. In order to obtain good random values, the random device (/dev/random) from the Linux OS has been employed. This device retrieves environmental noise from device drivers storing it on an entropy file. This file is then employed for generating random numbers. Processor #0 is the only one in charge of accessing the file, thus avoiding waiting times for other processors trying to access the same shared file.

Results shown below are always the average over the 100 runs computed for every experiment, so that they are statistically significant.

When experimenting with regular GA, without faults, the experiment finishes when results have converged, after a number *n* of generations, or 5000 generations otherwise (therefore *n=5000)*. On the other hand, when experiments undergoing faults are considered, the experiment finishes after the same number of *n* generations computed for the experiment without faults. We can thus fairly compare fitness values for both experiments.

For a better understanding of results summarized on Tables 1 and 2, we first describe information included for each of the columns:

- **Experiment:** A summary of the experiments, including some important parameters:
    - Initial number of populations – processors.
    - Number of generations computed before every simulated failure
- **Interchange:** Period employed for migrating individuals.
- **Inheritance:** Indicates whether all subsisting processors at the end of the run have received genetic material belonging to all of the processors from the experiment.
- **Generation:** Total number of generations for the experiment.
- **Max with faults:** Best fitness obtained for the experiment considering faults.
- **Max w/o faults:** Best fitness obtained for the same experiments when no faults are present.
- **% Difference:** Difference between results obtained with and without faults.
- **Min with faults:** Worst fitness obtained for the experiment considering faults.
- **Min w/o faults:** Worst fitness obtained for the same experiments when no faults are present.

Experimental results included also the minimum values obtained with and without simulation of errors. The aim of this figure is to check that we have reached the convergence of the populations. The first columns on tables name the experiments, for example, 50#P8 stands for simulating a fault every 50 generations on an implementation with 8 processors.

We have included information concerning inheritance, for making a first analysis about the relationship between propagation of information and fault tolerance.

Experiment	interchange	Inheritance	Number of gener.	MAX with fault	MAX w/o fault	% Difference	MIN with fault	MIN w/o fault
50 #P=8	10	Yes	760	28,1	28,6	1,74825	28,1	28,6
	25	Yes	760	28,5	28,7	**0,69686**	28,5	28,3
	50	no	860	28,1	28,4	1,05634	27,7	26,8
	100	no	840	27	27,3	1,09890	25,6	24,5
50 #P=4	10	Yes	840	27,9	28,4	1,76056	27,9	28,4
	25	Yes	800	28	28,7	2,43902	28	28,5
	50	no	820	27,6	28,7	3,83275	27,5	28,5
	100	no	780	27,2	28,8	5,55556	27,1	27,9
50 #P=2	10	Yes	800	27,5	28,2	2,48227	27,5	28,2
	25	Yes	840	27	28,2	4,25532	27	28
	50	no	740	26,6	28,3	6,00707	26,2	28,1
	100	no	700	26,6	28,6	6,99301	26,6	28,4
100 #P=8	10	Yes	760	28,4	28,6	**0,69930**	28,4	28,6
	25	Yes	760	28,7	28,7	**0,00000**	28,7	28,3
	50	Yes	1020	28,3	28,4	**0,35211**	27,5	26,8
	100	no	960	27,3	27,3	**0,00000**	25,4	24,5
100 #P=4	10	Yes	560	28,1	28,4	1,05634	28,1	28,4
	25	Yes	740	28,4	28,7	1,04530	28,4	28,5
	50	Yes	620	28,3	28,7	1,39373	28,2	28,5
	100	no	880	27,7	28,8	3,81944	27,5	27,9
100 #P=2	10	Yes	840	28,1	28,2	**0,35461**	28,1	28,2
	25	Yes	820	27,7	28,2	1,77305	27,7	28
	50	Yes	840	27,3	28,3	3,53357	27,2	28,1
	100	no	780	26,7	28,6	6,64336	26,6	28,4
200 #P=8	10	Yes	700	28,7	28,6	**0,34965**	28,6	28,6
	25	Yes	760	28,8	28,7	**0,34843**	28,5	28,3
	50	Yes	800	28,4	28,4	**0,00000**	27,2	26,8
	100	Yes	860	27,5	27,3	**0,73260**	25	24,5
200 #P=4	10	Yes	660	28,2	28,4	**0,70423**	28,2	28,4
	25	Yes	700	28,6	28,7	**0,34843**	28,5	28,5
	50	Yes	700	28,6	28,7	**0,34843**	28,6	28,5
	100	Yes	880	28,5	28,8	1,04167	28,3	27,9
200 #P=2	10	Yes	700	28,2	28,2	**0,00000**	28,2	28,2
	25	Yes	760	28,1	28,2	**0,35461**	28,1	28
	50	Yes	760	28,1	28,3	**0,70671**	28,1	28,1
	100	Yes	820	28,1	28,6	1,74825	27,8	28,4

Table 1: Experimental Results for FModal. Bold figures indicates differences < 1%

4.1 Fmodal

Table 1 shows results obtained for Fmodal problem. If we focus on **%Difference** column, we notice that differences are usually very small. They only grow (up to a 6%) when failures happen very frequently or a small number of processors or both (see for instance Experiments simulating failures every 50 generations or considering 2 processors –islands- or 100 generations between failures and 2 processors, or 50 generations and 4 processors). When a larger number of generations are computed before every new fault and a larger number of processors -subpopulations, differences are very narrow (1%). Therefore, the first conclusion is that processor faults do not affect significantly the quality of obtained results.

If we continue analyzing results, we see that when 200 generations are computed before every new fault, differences obtained are extremely narrow (below 1%). When 100 generations are computed instead, similar results are obtained if we use a number of processors larger than 2. Only with 2 processors results deteriorates. But if we remember how the island model work, we see that if a fault occur in one out of the two processor, no island model exist any more, given that the two processors will remain isolated.

In both of the above referred cases, differences of fitness quality with and without failures are below 2%.

Summarizing, we have seen for this problem that the island model can provide results of equivalent quality in the presence of processor faults (when compared with the model in the absence of faults, and for a number of island greater than 2).

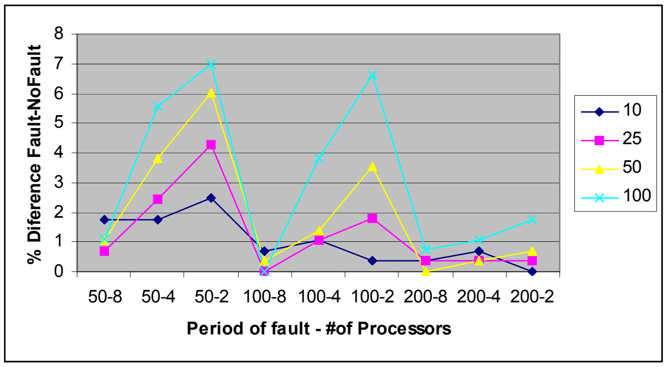


Figure 5: % Differences between Max averaged values for exchange periods of 10, 25, 50 generations. Fmodal function

If we look to inheritance column within the table, we see that when the exchange period is smaller (10 or 25 generations), processors have enough time to distribute their genetic material to the remaining processors before faults occur. This helps to obtain smaller differences between executions with and without processor failures.

The reason for this behaviour is clear: When the exchange period is larger than faults period, a population might disappear before sending its best genetic material to another population, thus keeping it from participating in the generation of even better solutions. Therefore, on a distributed environment with a given frequency of faults, the exchange rate should be tuned appropriately for obtaining results of quality.

Those results can also been observed on Figure 5. This graph represents the differences on maximum values for different interchanges period 10, 25, 50 and 100 generations. Confirming the commented results, when the genetic information is transmitted at the beginning of the execution (10 and 25), the GA suffers less the effect of a fault.

4.2 Schwefel

Similar experiments have been conducted for schwefel function, and similar conclusions can be drawn from table 2:

- Results are on average of similar quality when faults are present when compared with a fault-free run on the Island model.
- Better to exchange more frequently individuals, so that island has time to distribute their genetic material before a fault can appear (better results with a period of 10 or 25).
- In the presence of processors' failures, it is better to employ a larger number of islands for the experiment (better results with 8 processors).

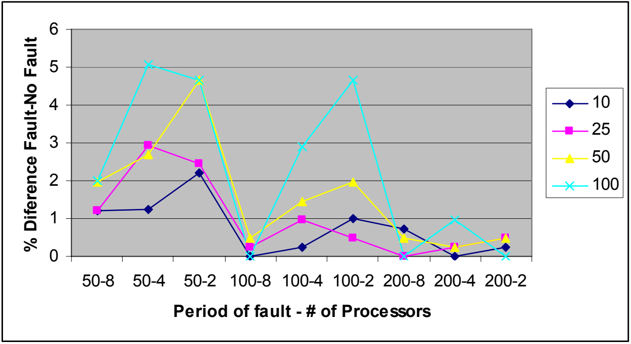


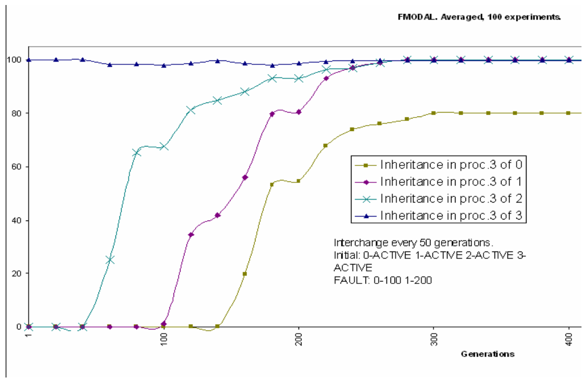
Figure 6: % Difference between Max averaged values for 10,25,50 and 100 interchange period. Schwefel function

Those results can also been observed on Figure 6. Again it represents the differences on maximum values for different interchanges period 10, 25, 50 and 100 generations. Let us just comment one point on the figure. It could be seen that the behaviour is not clear. That is simply a consequence of the selected representation. Actually, point at 100-8 and 200-8 x-axis signs indicated a very good fault tolerance of the configurations with 8 processors.

Finally, Figure 7 shows for the previous Fmodal function, a graph including information of inherited material. The figure shows for the experiment which employes 4 islands, how material from each of those 4 islands have reached one of the islands along the experiment. The graph shows the percentage of individuals from population #3, that includes material comming from populations #0 ,#1 and #2. We see that material inherited grows within the population under study, when exchange periods are taking place. Given that population #2 is the one closer to population #3, its material is the one that arrives first. This graphs shows the importance of the migration periods, and how material from all the population is present in the remaining ones, which helps to understand why when a population fails, results are not greatly affected. Similar results have been obtained for other number of islands and also for the schwefel problem.

Experiment	Interchange Period	Inheritance	Number of gener	MAX with fault	MAX no fault	% Diference	MIN with fault	MIN no fault
50 #P=8	10	yes	700	4050	4100	1,21951	4050	4100
	25	yes	820	4080	4130	1,21065	4070	4100
	50	no	840	4020	4100	1,95122	3990	3950
	100	no	860	3950	4030	1,98511	3790	3620
50 #P=4	10	yes	680	4010	4060	1,23153	4010	4050
	25	yes	760	3980	4100	2,92683	3980	4090
	50	no	780	3990	4100	2,68293	3990	4090
	100	no	1000	3920	4130	5,08475	3920	4070
50 #P=2	10	yes	760	3970	4060	2,21675	3970	4050
	25	yes	760	3970	4070	2,45700	3960	4050
	50	no	820	3890	4080	4,65686	3890	4060
	100	no	780	3890	4080	4,65686	3880	4080
100 #P=8	10	yes	740	4100	4100	**0,00000**	4100	4100
	25	yes	860	4120	4130	**0,24213**	4110	4100
	50	yes	940	4080	4100	**0,48780**	4030	3950
	100	no	1000	4030	4030	**0,00000**	3860	3620
100 #P=4	10	yes	720	4050	4060	**0,24631**	4050	4050
	25	yes	840	4060	4100	**0,97561**	4050	4090
	50	yes	880	4040	4100	1,46341	4020	4090
	100	no	860	4010	4130	2,90557	3980	4070
100 #P=2	10	yes	760	4020	4060	**0,98522**	4010	4050
	25	yes	800	4050	4070	**0,49140**	4050	4050
	50	yes	780	4000	4080	1,96078	4000	4060
	100	no	780	3890	4080	4,65686	3880	4080
200 #P=8	10	yes	660	4070	4100	**0,73171**	4070	4100
	25	yes	820	4130	4130	**0,00000**	4110	4100
	50	yes	940	4120	4100	**0,48780**	4030	3950
	100	yes	820	4030	4030	**0,00000**	3720	3620
200 #P=4	10	yes	660	4060	4060	**0,00000**	4060	4050
	25	yes	820	4090	4100	**0,24390**	4090	4090
	50	yes	840	4090	4100	**0,24390**	4090	4090
	100	yes	960	4090	4130	**0,96852**	4060	4070
200 #P=2	10	yes	780	4050	4060	**0,24631**	4050	4050
	25	yes	780	4090	4070	**0,49140**	4080	4050
	50	yes	800	4100	4080	**0,49020**	4100	4060
	100	yes	920	4080	4080	**0,00000**	4030	4080

Table 2: Experimental Results for Schwefel. Bold figures indicates differences < 1%



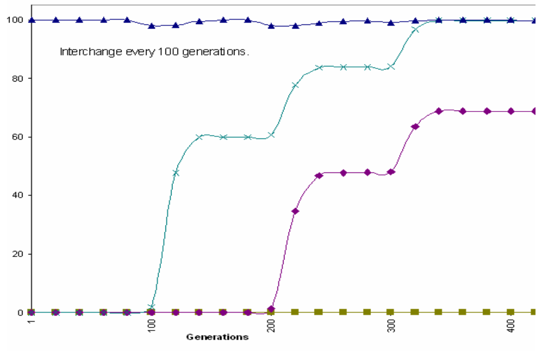


Figure 7: Percentage of individuals from population #3 that includes material inherited from the remaining populations when the frequency of exchange is 50 generations (above) or 100 generations (below), along generations.

5. CONCLUSIONS AND FUTURE WORK

This paper has shown that the coarse grain parallel GA structured by means of the Island Model achieves fault tolerance when applied to Multimodal Functions.

We have experimented on a multiprocessor computer running a GA library and employing MPI for migrating individuals among processors. Processor failures have been simulated when experimenting with Fmodal and Schwefel functions.

A different number of processors have been employed for each experiment and we have considered different failure rates for the available processors. For every experiment results obtained without any fault have been compared with those obtained when faults are present. For all experiments, differences have been quite small (below 7%), being the average difference always below 2% (1,81 for Fmodal function and 1.4 for the schwefel function). Especially good results were obtained when using more than 2 processors-islands for the experiment, so that after faults the island model is actually present.

For a number of 8 processors-islands and 200 generations computed before every fault, differences of results are below 1%. We have also seen that better results are obtained when exchange frequency is larger than faults rate. Although results have been obtained simulating faults and on a multiprocessor system, the high failure rate considered allows us to be confident on the extension of conclusions to real distributed networks of computers (we have considered up to 50% of processors failures, which is an extremely high number of faults).

Summarizing, researchers can trust on the island model when running experiments on a non-reliable parallel or distributed infrastructure: Despite processor failures the island model GA will still provide results of similar quality as when a reliable infrastructure is employed, and this without making the algorithm more complex by adding any special techniques required for dealing faults.

The Island Model has proved fault-tolerant by nature, at least on the problems faced within this research.

6. ACKNOWLEDGMENTS

This work has been partially supported by "OPLINK" Spanish Government Project and Project number TIN 2005-5619 also from the Spanish Goverment. We also want to thank to the students who help us with the simulations Amelia Barroso, David Boíllos y Ana Belén Jerónimo.

7. REFERENCES

[1] W. D. Whitley, S. B. Rana, and R. B. Heckendorn, "Island model genetic algorithms and linearly separable problems," in Selected Papers from AISB Workshop on Evolutionary Computing. London, UK: Springer-Verlag, 1997, pp. 109–125.

[2] M. Tomassini, "Spatially Structured Evolutionary Algorithms: Artificial Evolution In Space And Time", (Natural Computing Series), Springer-Verlag New York, Inc., Secaucus, NJ, 2005 Springer, 2005. ISBN:3540241930

[3] J. B. M. Litzkow, T. Tannenbaum and M. Livny, "Checkpoint and migration of unix processes in the condor distributed processing system," University of Wisconsin, Technical Report #1346, University of Wisconsin Madison Computer Sciences, April 1997.

[4] A. T. Tai and K. S. Tso, "A performability-oriented software rejuvenation framework for distributed applications," in DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05). Washington, DC, USA: IEEE Computer Society, 2005, pp. 570–579.

[5] D. Anderson, "BOINC: a system for public-resource computing and storage," in Grid Computing, 2004. Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on, pp 4- 10.

[6] K. Imamura and J. A. Foster, "Fault tolerant computing with N-version genetic programming," in GECCO 2001: Proceedings of Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, L. Spector, E. D. Goodman (eds.), Morgan Kaufmann: San Francisco, CA, USA, 2001, p. 178

[7] A. Baratloo and P. Dasgupta and Z.M. Kedem, " CALYPSO: a novel software system for fault-tolerant parallel processing on distributed platforms", Fourth IEEE International

Symposium on High Performance Distributed Computing (HPDC-4 '95) p. 122. 1995, IEEE Press.

[8] C. Gagné and M. Parizeau and M. Dubreuil, “ Distributed BEAGLE: An Environment for Parallel and Distributed Evolutionary Computations”, in 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS) 2003, pp 201 – 208.

[9] MG Arenas et al “A framework for distributed evolutionary algorithms Lecture Notes in Computer Science . Springer Berlin / Heidelberg . Volume 2439/2002 . Parallel Problem Solving from Nature - PPSN VII: 7th International Conference Proceedings, Granada, Spain, September 7-11, 2002. Pages 665-675.

[10] E. Cantú -Paz and D. E. Goldberg, “Efficient parallel genetic algorithms: Theory and practice,” Computer Methods in Applied Mechanics and Engineering, 2000. citeseer.ist.psu.edu/cantu-paz00efficient.html.

[11] C.B. Pettey, M. R. Leuze and J. J. Grefenstette, “A parallel genetic algorithm” in Proceedings of the Second International Conference on Genetic Algorithms and their application, J.J. Grefenstette (Editor).1987, pp 155-161

[12] A. Agbaria and R. Friedman, “Starfish: Fault-Tolerant Dynamic MPI Programs on Clusters of Workstations” in HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing, 1999, pp 31.

[13] W. Gropp and E. Lusk, “Fault Tolerance in MPI Programs”, in Proceedings of the Cluster Computing and Grid Systems Conference, December 2002.

[14] Paechter, B.; Back, T.; Schoenauer, M.; Sebag, M.; Eiben, A.E.; Merelo, J.J.; Fogarty, T.C., "A Distributed Resource Evolutionary Algorithm Machine (DREAM)," *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on* , vol.2, no.pp.951-958 vol.2, 2000.

[15] M. Jelasity, M. Preuss, M. van Steen, B. Paechter, “Maintaining Connectivity in a Scalable and Robust Distributed Environment” in Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002), 2002.

[16] E. Cantú-Paz “Efficient parallel genetic algorithms” Kluwer Academic Publishers, 2000. ISBN 0-7923-7221-2.

[17] F. Fernandez, G. Spezzano, M. Tomassini, and L. Vanneschi, “Parallel genetic programming,” in Parallel Metaheuristics, ser. Parallel and Distributed Computing, E. Alba, Ed. Hoboken, New Jersey, USA: Wiley-Interscience, 2005, ch. 6, pp. 127–153.

[18] M. Tomassini, Spatially Structured Evolutionary Algorithms, Springer, Ed. Springer, 2005.

[19] D. Lombraña González, F. Fernández de Vega, “On the intrinsic fault-tolerance nature of Parallel Genetic Programming”, in The Fifteen Euromicro Conference on Parallel, Distributed and Network-based Processing, 2007.

[20] H. P.Schwefel. *Numerical Optimization of Computer Models*. John Wiley & Sons, 1981.

[21] Wolpert, D. H, Macready, W. G., “No Free Lunch theorems for optimization”, IEEE Transactions on Evolutionary Computation, Vol 1, N 1, pp: 67-82. Apr. 1997.

[22] Ho, Y.C., Pepyne, D.L. (2002), Simple Explanation of the No-Free-Lunch Theorem and Its Implications, Journal of Optimization Theory and Applications 115, 549.

[23] Cantu-Paz, E., Goldberg, D. E.: Are multiple runs of genetic algorithms better than one? Proceedings of Gecco (2003). Chicago. Pp. 801-812. Springer Verlag.

[24] Z. Michaelewicz. Genetic Algorithms + Data Structures = Evolution Programs. Springer,3rd edition, 1996

[25] David Levine, PGA Pack. Available at: http://www-fp.mcs.anl.gov/CCST/research/reports_pre1998/comp_bio/stalk/pgapack.html

[26] W Gropp, E Lusk “User’s Guide for MPICH, a Portable Implementation of MPI” - Argonne National Laboratory, 1994. http://www-unix.mcs.anl.gov/~lusk/oldpapers/mpich-guide/paper.html