PARALLEL GENETIC ALGORITHMS: A FEASIBLE DISTRIBUTED IMPLEMENTATION

OCHOA C.*, GALLARD R.**

Grupo de Interés en Sistemas de Computación***
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis
Argentina
E-mail: gallardr@unslfm.edu.ar
Phone: 54++ 652 20823
Fax : 54++ 652 30224

Abstract

Parallel genetic algorithms, models and implementations, attempts to exploit the intrinsically parallel nature of genetic algorithms. By distributing the total population, these models reflects a behaviour nearer to that of natural systems. A variety of parallel computer systems architectures can offer distinct support features for their implementation.
This paper shows some remarkable characteristics of parallel genetic algorithms, details of a feasible design and their implementation. Also some results related to the island model are shown.

Keywords: Parallel genetic algorithms, performance, granularity, island model, migration schemes, interconnectivity, system architecture.

---

* Member of the UNSL 338403 project.

** Full Professor at the Informatics Department. Head of the Research Group.

*** The Research Group is supported by the Universidad Nacional de San Luis, the CONICET (Science and Technology Research Council) and the SSID (Subsecretary of Informatics and Development).

1. Introduction

Genetic algorithms (GAs) are stochastic and adaptive search algorithms that implement an elementary form of the natural selection mechanism to explore a problem space using the Darwinian principle of natural selection and survival of the fittest. They were first devised by John Holland [6] and his coworkers at the University of Michigan in the 1970s and have been studied by other research groups since. GAs are today considered as a robust technique, effective across a spectrum of problems even in the presence of difficulties such as noise, multimodality, high-dimensionality and discontinuity (De Jong [2]).
Basically, GAs simulate the evolution (natural adaptation) of a population of solutions for a given problem. After the initial population is created, the evolution loop of a genetic algorithm consists of (1) evaluating all individuals in the population, (2) selecting a new, intermediate population (better individuals have better chances to be selected), and (3) alternating their genetic code. These three steps are repeated until some termination condition is satisfied.
They can be used to find approximate solutions to numerical optimisation problems in cases where finding the exact optimum is prohibitively expensive, or where no algorithm is known.
As GAs are inherently parallel (Goldberg [3]), because many "individuals" are evolved in the search, this work shows diverse approaches of parallel computing platforms to implement GAs and a particular selected implementation.

2. Parallel Genetic Algorithms: A Survey

In his first work Holland [6] recognised the parallel nature of the reproductive paradigm and the intrinsic efficiency of parallel processing.
In his taxonomy of Parallel Genetic Algorithms (PGA), Levine [7] observes the importance of discerning between a PGA as a *model* of an specific GA and a PGA as a *means to implement* a (sequential or parallel model of) GA.
In a *PGA model* the total population is distributed in any of the following ways:

- A set of independent subpopulations of certain size.
- A single population where each member interacts with a limited number of neighbours.

A PGA *model* have some advantages when contrasted with the conventional GA model. Let us see some facts:

- In PGAs premature convergence risk is decreased by maintaining dissimilar subpopulations which interchange genetic material between them. As a consequence of this, a better exploration of the searching space is done.
- The expected number of descendants of a certain string depends on the relative value of its fitness within the population. Thus, in conventional GAs, this implies a global ranking which do not properly reflects the way in which natural selection works.
- A PGA model is more realistic model of natural behaviour; a subpopulation is typically compounded of many independent subpopulations which occasionally interact.

About *implementation*, it is possible to parallelise the traditional sequential GA. One of the simplest methods consists in parallelising the loops which creates the next generation from the current one. Many procedures in this loop, such as evaluation and application of genetic operators, could be executed in parallel.
If distributed approaches are chosen to parallelise a sequential GA the computing overhead due to distribution of data structures and synchronisation could decrease further improvements due to multiprocessing. For this reason tightly-coupled multiprocessors are the preferred platforms to parallelise this loop.
Other parallel approaches are devised for distributed processing in multicomputer systems and we will see them now.
According to the size of the components of the subdivided population (granularity) GAs can be classified as *Coarse Grained* GAs and *Fine Grained* GAs (Levine [7]). Some main characteristics of PGA models will be now briefly introduced.

2.1. Coarse Grained PGAs

In a Coarse Grained Parallel Genetic Algorithm (CGPGA), known as the *island model*, the entire population is divided in a number of subpopulations which are distributed among multiple processors. Each of these processors run a sequential GA on their own subpopulation. Occasionally these processors interact exchanging chromosomes.
Some open questions in CGPGA are; how frequently processors exchange strings ?, which other processors a processor exchange with ? and what scheme is used to select strings to exchange ?.
When compared with sequential GAs, the CGPGA model shows a behaviour nearer to that of biological systems by maintaining multiple separate subpopulations which evolve independently. In this manner, each subpopulation explores a different area of the searching space, each maintaining its high fitness individual (elitism) and controlling their migration to other subpopulations.
Figure 1 shows a CGPGA model with a *three-dimensional hypercube* interconnection topology. Here each processor has exactly three neighbours for exchanging strings.

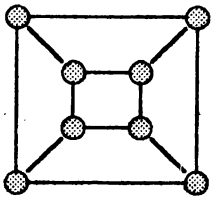


Fig. 1: A Coarse Grain Parallel Genetic Algorithm Model.

CGPGAs can be further classified according to the migration scheme, interconnection topology and homogeneity of processing nodes (Shyh-Chang Lin [9]).

2.1.1. Migration Schemes

String migration between subpopulations is a key characteristic of CGPGA that helps to maintain genetic diversity by inserting strings arriving from subpopulations separately evolved. Nevertheless, if many strings are frequently exchanged the algorithm is prone to premature convergence, because after some number of migrations all subpopulations will have copies of outstanding individuals which will be inclined to dominate those populations.
A migration scheme, determines how frequently and under which time constraints string exchange is done. Migration schemes provide an additional subclassification for CGPGAs: *isolated, synchronous* and *asynchronous.*
Isolated CGPGAs, also known as *partitioned* GAs are the simplest CGPGA model and do not allow migration between subpopulations.
Synchronous CGPGAs where the migration is synchronised, allow subpopulations to evolve in parallel until certain point before any exchange can occur. Synchronisation between processors can be controlled by the number of generations or any other measure of progress (convergence, mean population fitness, etc.). Dedicated hardware with similar processing power in each processor, can directly support such synchronisation. In distributed environments unbalanced workloads can arise, because different processing speeds can result in some idle nodes waiting for strings from slow processors. In such environment the global population evolves according to the slower processor speed.
Asynchronous CGPGAs, allow migration at any moment independently of the evolution state of subpopulations. This asynchronous behaviour reflects the kind of migration that, in fact, happens in nature where diverse populations have distinct evolution paces. This migration scheme is suitably fitted for distributed workstation environments where dissimilar computer architectures and workloads cause different evolution speeds in each node.

2.1.2. Interconnectivity

The processing nodes interconnectivity (topology and interconnection degree) affects the performance of a CGPGA. The type of topology (*lines, rings, n-cubes, torus, meshes*, etc.) determines which nodes are the neighbours of a certain one, for string exchanging.
Interconnection schemes can be subdivided into two main groups: *static* and *dynamic*.
In a *static interconnection scheme* the node connections are defined at the beginning of the run once for all. In a *dynamic interconnection scheme* the initial topology may vary during execution. In some cases, due to changes occurred in subpopulation evolution it is desirable to reconfigure the topology. A problem arise when the insertion of a new imported string is not effective because it is a superindividual (which can dominate the subpopulation) or it is a very low fitness individual (which can be rejected). In order to avoid these extreme situations, subpopulations can begin running without predefined neighbours. Then, when migrations occur the node chose their neighbours deciding by a criteria based on the degree of similarity (or non similarity) of their chromosomes.

2.1.3. Nodes Homogeneity

Homogeneity in PGAs is related to the similitude of the GAs running in different nodes. Therefore CGPGAs are roughly classified as *homogeneous* and *heterogeneous*. In an *homogeneous CGPGA* model the GA executed in each processor have the same parameters set (population size, crossover and mutation probabilities, migration intervals) , genetic operators, objective functions, etc. The advantage of this approach resides in its easy implementation. An *heterogeneous CGPGA* model allows the evolution of diverse subpopulation with different parameters, genetic operators and objective functions. This approach could be advantageous to find the best initial set of parameters for a GA.

2.2. Fine Grained PGAs

In a Fine Grained Parallel Genetic Algorithm (FGPGA) also known as *cellular* or *massively parallel* GA exactly one processor is allocated to each string.
In this model the population consists of strings which interact only within a local neighbourhood. In this way each string is part of multiple subpopulations and the membership is determined by the processor interconnection topology. Thus, the global population can be visualised as being composed of small overlapped subpopulations. The main problem in this model is the net topology design, because in this way the degree of individual isolation is predetermined and consequently the genetic diversity is also affected.
Depending on communication restrictions global and random mating could be a method to be used but seems inadequate. In alternative approaches, each processor searches for mating the best string residing in the nodes of its vicinity and produce a single offspring which remains in this processor.
After the first population evaluation, strings are randomly distributed among processors and after some generations groups of similar strings with similar fitness values are formed and continue growing (Withley [11]).
Figure 2 shows a FGPGA model with 64 processors storing a string each, represented by a shaded square. Different shaded zones depict the neighbourhoods built after some generations. In this case the interconnection topology is a torus where opposite extremes are connected.

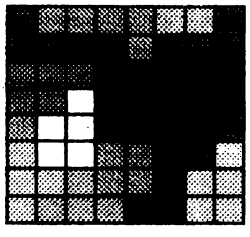


Fig. 2: A Fine Grained Parallel Genetic Algorithm Model.

2.3. Micrograined PGAs

In a Micrograined PGAs (MPGA) a single population is maintained, so the output is the same as in a conventional sequential GA. Parallelism consists in using multiple processors to evaluate individual fitness values (Punch [8]).
This scheme is particularly beneficial when computations for function evaluation, as in real time control, are costly when contrasted with other genetic operations.
In this model, a single master processor runs the GA and distributes strings to other slave processors for their evaluation. Other genetic operations take place in the master processor.
In this manner, if the evaluation of individuals dominates the remaining computations, a lineal speedup proportional to the number of slave processors can be expected by using this technique.
The optimum alternative is to allocate a slave processor per string in the population. In this instance the total population evaluation time would be equivalent to the worst time to evaluate an individual.
If the number of available processor is less than the number of strings in the population, then each node would be responsible to evaluate a given subset. This decision results in a total population evaluation time equivalent to the worst time required to evaluate a given population subset.
In Figure 3 a MPGA model is shown. The central node represents the master processor while the peripheral nodes stand for the slave processors in charge of strings evaluation.

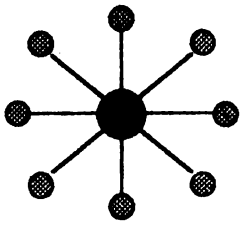


Fig. 3. Micrograined Parallel Genetic Algorithm Model.

3. The System Supporting PGA Execution

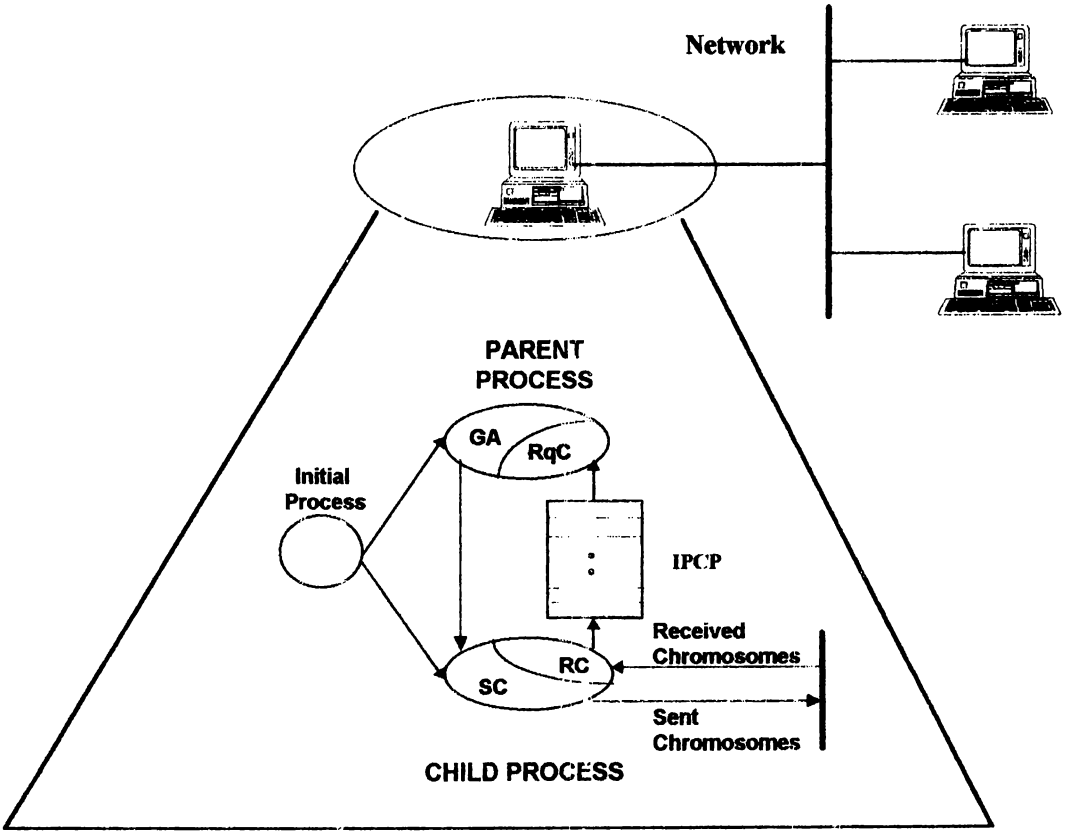
The main decision made was the kind of PGA model to use. In our case due to the lack of massively parallel processors we implemented a CGPGA model for which our network of workstations resulted suitable enough.

3.1. System Architecture

When a distributed application for a PGA begins execution it is necessary to specify a number of input parameters. Then the initial process in each processor forks once. The parent process will be responsible of GA execution and requesting of migration services while the child process will be in charge of managing arrival of chromosomes from remote processors, forwarding them to the parent process and sending local migrating chromosomes throughout the net to other populations. This is done by using Interprocess Communications Primitives and Application Program Interfaces (Arredondo et al[1]). The corresponding architecture is shown in figure 4.
The processes *send_chromosomes* and *receive_chromosomes* are the main components implementing the interaction with other processes of the distributed system. Because the local area network is reliable and small amount of data will be transmitted each time, these routines use a socket interface and a connectionless protocol (UDP) to minimise communication overhead.
Sending and receiving chromosomes need synchronisation. A class of non blocking IPCP was used in order to allow progressing the GA even if chromosomes did not yet arrived from remote processors.

3.2. Some Details on Implementation

A large set of runs were done for our first PGA implementation. Initially, to study subpopulations interactions, it was decided to implement the PGA model on only two workstations of different characteristics and run it on a set of well known test functions solving optimisation problems (De Jong [2], Schaffer [4] and other functions). See Table 1.



GA : Genetic Algorithm
SC : send_chromosomes
RqC : Request Chromosomes
RC : receive_chromosomes
IPCP: Interprocess Communication Primitives

Fig. 4. System Architecture to Support Migration

The following schemes were chosen for gathering of preliminary results:

Homogeneity

In this first experience we only work with homogeneous nodes: same population size, crossover method and probabilities were used in both workstations.

Migration Decisions

In order to compare effectiveness of each method, isolated, synchronous and asynchronous schemes were implemented. Also the expected improvements of migration were contrasted against its cost by comparing the isolated (no migration) method with the other two.
About the number of strings to be migrated, we decided to migrate only one (the best) string each time. In this way we tried to guarantee an existent but no major influence of external evolution on local evolution.

$F1$: $f(x_1, x_2) = 21.5 + x_1 \cdot sin(4\pi x_1) + x_2 \cdot sin(20\pi x_2)$, for;
$-3.0 \leq x_1 \leq 12.1, \quad 4.1 \leq x_2 \leq 5.8$
estimated maximum value: 38.850292

$F2$: $f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$, for;
$-5.12 \leq x_i \leq 5.12$, $i = 1,2,3$ (*De Jong Function F1*)
minimum global value: 0

$F3$: $f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$, for;
$-2.048 \leq x_i \leq 2.048$ $i = 1,2$ (*De Jong Function F2*)
minimum global value: 0

$F4$: $f(x_1) = 2.0 + x_1 \cdot sin(10\pi x_1)$, for;
$-1 \leq x_1 \leq 2$
estimated maximum value: 3.850273

$F5$: $f(x_1, x_2) = 0.5 + \dfrac{sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{1.0 + 0.0001(x_1^2 + x_2^2)^2}$, for;
$-100 \leq x_i \leq 100$, $i = 1,2$ (*Schaffer Function F6*)
minimum global value: 0

$F6$: $f(x_1, x_2) = (x_1 \cdot \mathrm{sgn}(x_1)) \cdot (x_2 \cdot \mathrm{sgn}(x_2))$, for;
$-1 \leq x_i \leq 2$, $i = 1,2$
estimated maximum value: 4

$F7$: $f(x_1, x_2, x_3, x_4, x_5) = \sum_{i=1}^{5} \mathrm{int}(x_i)$, for;
$-5.12 \leq x_i \leq 5.12$ $i = 1,2,3,4,5$ (*De Jong Function F3*)
minimum global value: -25

Table 1. Set of Testing Functions

In order to favour genetic diversity we resolved not to reject any incoming string. Thus, as we are working with fixed population size, a policy for choosing a "victim" string for replacement was used. Some authors advocate for the replacement of the worst individual, but this can result also in a loss of genetic diversity. So we implemented a policy consisting in a random selection of two candidate strings which then were subjected to a probabilistic tournament for ultimate decision [7].
In order to obtain results to be used as milestones for further research, we begin choosing workstations of quite dissimilar processing speeds (W_1 was 4 times faster than W_2). In this particular environment subpopulations evolve at different speed, and then some problems arouse:

- The incoming individual was originated in a low evolved population (compared with local population). Depending on the population size, this fact does not influence too much on global fitness and can help to maintain or increase genetic diversity. This contributes to explore another searching areas.
- The incoming string arrives from a high evolved population. In this case if the gap between the fitness of the new individual and that of the best local individual is large enough then a risk of premature convergence can arise.

To avoid falling towards a local optimum, by introduction of high performers, another additional strategy was devised: A parameter Γ, called *maximum gap allowed*, was defined as the maximum difference accepted between the best local individual fitness and the incoming string fitness. Therefore, if the external string is superior than the best local individual beyond a certain threshold it will be rejected otherwise will be inserted into the population. In other words, if the condition:

$$\text{Fitness}_{ext} - (1+\Gamma)\,\text{Fitness}_{local} \leq 0 \quad (0 \leq \Gamma \leq 1)$$

holds, then accept insertion of the incoming string, otherwise reject string.
By using this strategy the influence of high evolved external strings was decreased and, consequently, the risk of falling into local optimum also declined.

4. Comments on Results

Before tuning the island model to be run in two distinct-feature workstations, the GA was run on each workstation and on each of the selected test functions. As a result it was observed that in similar environmental conditions workstation W_1 was nearly four times faster than workstation W_2. Therefore, migration intervals were defined every 1000 generations in W_1 and every 250 generations in W_2. Consequently, in each run a total number of 16000 generations and 4000 generations were respectively defined for W_1 and W_2.
The next step was addressed to select the most interesting functions for the island model. So, a set of runs determined that functions F1 and F5 were the most attracting functions because the remaining functions reach very fast their optimal (or near optimal) values. In all cases a typical one-point crossover algorithm with population size of 50, probability of crossover 0.65 and probability of mutation 0.001 was run.
After that, the island model was thoroughly tested. In this stage interaction of subpopulations was done via the migration of the best individual. Incidentally, running the model on F1, it was detected that a high performer migrating string from W_2 had a fitness value of 38.19 while the corresponding value for the best individual of the supposedly more evolved W_1 population was 35.95. This was the appealing fact to insert the new parameter Γ.
Subsequently, a number of runs for the island model under each of the migration schemes (isolated, synchronous and asynchronous) were completed with the same above indicated parameter set[1]. Mean values for the following relevant performance variables were determined:

Optimal Hits is the hit ratio to find the optimal solution, all over the total number of runs.
Ebest = (*opt_val* - best value/*opt_val*)100
It is the percentual error of the best found individual when compared with *opt_val*[2]. It give us a measure of how far we are from that *opt_val*.
Ebest reduction = $(\text{Ebest}_i - \text{Ebest}_{s|a})/\text{Ebest}_i)100$, where
Ebest_i is the Ebest value for the isolated method, and
$\text{Ebest}_{s|a}$ is the Ebest value for the synchronous or asynchronous methods.
The above expression shows the percentual of error reduction for the best found individual when contrasting either synchronous or asynchronous methods against the isolated method.

---

[1] In number of generations a variant was introduced for the synchronous migration case where the same number of 4000 generations was selected.
[2] *opt_val* is the known, or estimated, optimum value.

Epop = ((*opt_val* - mean pop value)/*opt_val*)100
It is the percentual error of the population mean when compared with *opt_val*. It tell us how far the mean fitness is from that *opt_val*.
Gbest is the mean of the number of generations where the best individual was found.
Time is the mean elapsed time, in seconds, for a single run of the algorithm.
Dtime = $((T_{s|a} - T_i)/T_i)100$, where
$T_{s|a}$ is the running time of synchronous (or asynchronous) run, and
T_i is the running time of isolated run.
The above expression shows the percentual of time increase when contrasting either synchronous or asynchronous methods against the isolated method.

The following tables and graphs show a report of experimental results on function F1. All the values in the tables are mean values obtained from the multiple run series.

FUNCTION F1

Migration Scheme	Optimal Hits	Ebest	Ebest reduction	Epop	Gbest	Time	Dtime
Isolated	34.78	2.39	0.00	2.88	3993	133.46	0.00
Synchronous	52.17	0.46	80.75	1.03	1497	142.61	6.85
Asynchronous	65.21	0.15	93.72	0.70	3607	134.24	0.58

Tabla 2: Performance values on workstation W1.

Migration Scheme	Optimal Hits	Ebest	Ebest reduction	Epop	Gbest	Time	Dtime
Isolated	13.04	1.62	0.00	1.86	1203	142.22	0.00
Synchronous	52.17	0.57	64.81	1.13	1254	142.57	0.24
Asynchronous	60.86	0.29	82.09	0.63	1198	143.01	0.55

Tabla 3: Performance values on workstation W2.

Fig. 5: Function F1

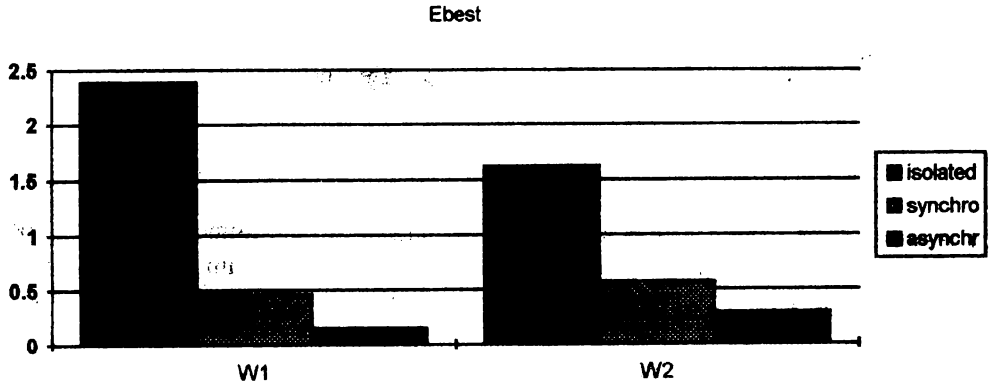


Fig. 6: Function F1

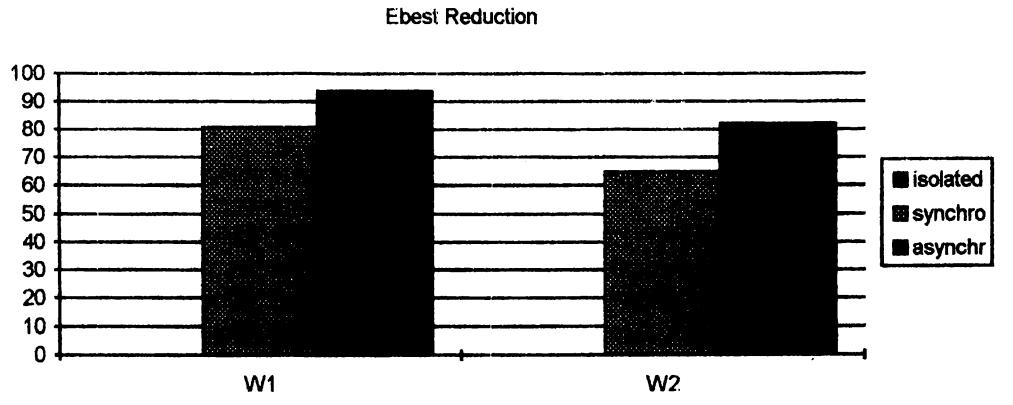


Fig. 7: Function F1

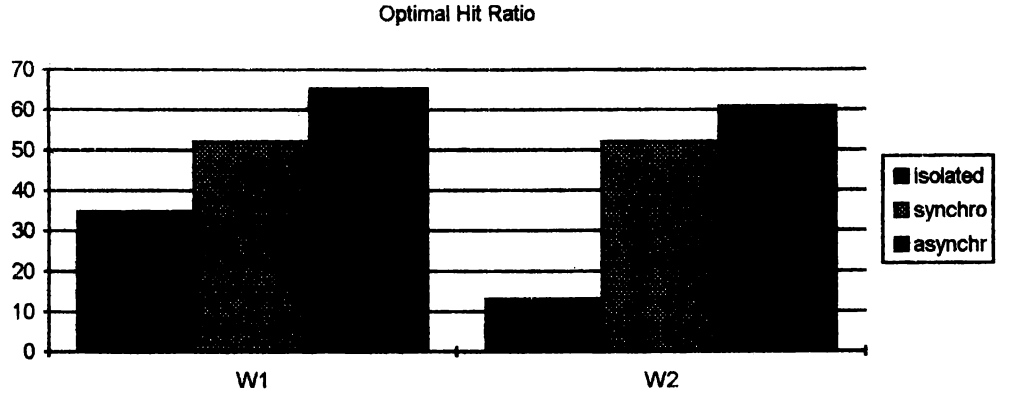
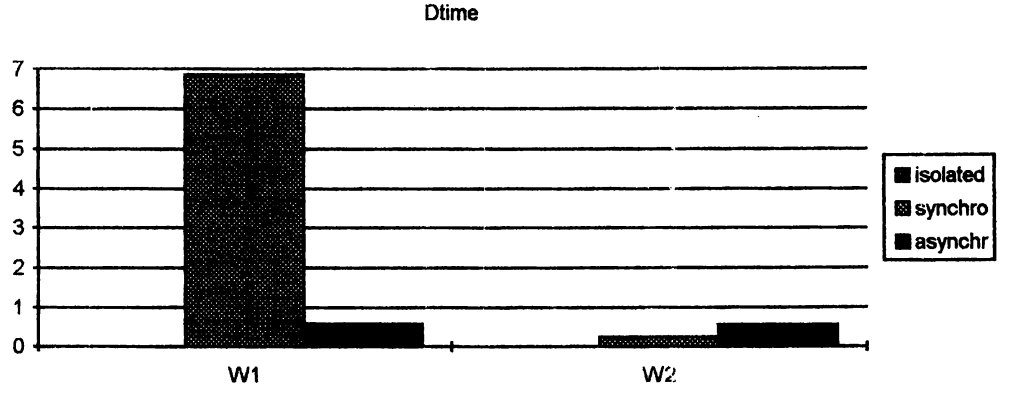


Fig. 8: Function F1



This first set of results, clearly show that those models permitting chromosome migration produce a significant error reduction when contrasted with the isolated model. As an example, it is observed that this improvement ranges from 65% up to 94% at a cost of a slightly increased running time that ranges from 0.2% up to about 7%.

To this extent, the asynchronous model behaves better than the synchronous model. It is also observed that higher Dtime values are achieved on the faster machine (W1) under the synchronous model. This effect is a consequence of the synchronised interaction imposed on both workstations: W1 must wait for W2.

Similar results corroborating the above conclusions were found for function F5.

5. CONCLUSIONS

In this paper aspects related to parallel genetic algorithms models and implementations were discussed. Also, a feasible refined design to support an implementation for the island model in a distributed and some representative results were shown. Some points deserve special attention:

- The original design was slightly modified and exhibit to be suitable for implementing and testing diverse coarse grained PGAs. The running time for testing, even for a large number of generations, was moderate.

- Models permitting migration behave notably better than those which do not allow any chromosome exchange.

- When exchange is allowed then some strategy should be adopted in order to prevent premature convergence: controlled arrival of high performers showed to be beneficial (our Γ parameter).

- When relative speeds of computer supporting subpopulations differ markedly, then the asynchronous model show a better general behaviour than the synchronous model.

At the light of this results, which where obtained using the simplest GA model, new experiments for more advanced models ([5] Esquivel et al, [10] Syswerda) are being devised with larger networks and diverse topologies, migration schemes and heterogeneity.

6. Bibliography

[1] Arredondo D., Printista M., Gallard R. - Un Sistema Distribuido para el Procesamiento Paralelo de Algoritmos Genéticos. Proceedings del Primer Congreso Argentino de Ciencias de la Computación, pp 242-252, Universidad Nacional del Sur, Bahia Blanca, Octubre 1995.

[2] De Jong, K. A. - Analysis of the Behavior of a Class of Genetic Adaptive Systems - Ph.D. dissertation. University of Michigan. 1975.

[3] Goldberg, D. - Genetics Algorithms in Search, Optimization and Machine Learning. - Addison-Wesley, Reading, MA, 1989.

[4] Eshelman, L. J. and Schaffer, D. J. - Crossover Niche - Proceedings of the Fifth International Conference on Genetic Algorithms, Stephanie Forrest (Editor), Morgan Kaufmann Publishers, 9-14, 1993.

[5] Esquivel S., Gallard R., Michalewicz Z. - MCPC: Another Approach to Crossover in Genetic Algorithms - Proceedings of the Primer Congreso Argentino de Ciencias de la Computación , pp 141-150, Univ. Nacional del Sur, Bahia Blanca, Argentina, October 1995.

[6] Holland, J.H. - Adaptation in Natural and Artificial Systems. Ann Arbor, The University of Michigan Press. 1975.

[7] Levine, D. - A Parallel Genetic Algorithm for the Set Partitioning Problem. - Ph D Thesis, Illinois Institute of Technology and Argone National Lab. (ANL -94/23), 1994.

[8] Punch W., Goodman E., Min Pei, Lai Chia-Shun, Hovland P., Enbody R. - Further Research on Feature Selection and Classification Using Genetics Algorithms. -ICGA93, pp 557 - 564, Champaign III. 1993.

[9] Shyh-Chang Lin, Punch W., Goodman E. - Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach. Parallel & Distributed Processing, Dallas TX, Oct. 1994.

[10]Syswerda G. - Uniform Crossover in Genetic Algorithms - Proceeding of the Third International Conference on Genetic Algortihms- 2-9, 1989.

[11]Withley D. - A Genetic Algorithm Tutorial. - Computer Science Departament, Colorado State University.