An Hierarchical Fair Competition Genetic Algorithm for Numerical Optimization

Alexandre C. M. Oliveira
DEINF/UFMA
S. Luís MA.
CAP/INPE
S. José dos Campos SP
Brasil.
acmo@deinf.ufma.br

Luiz A. N. Lorena
LAC/INPE
S. José dos Campos SP
Brasil.
lorena@lac.inpe.br

Stephan Stephani
LAC/INPE
S. José dos Campos SP
Brasil.
stephan@lac.inpe.br

Airam J. Preto
LAC/INPE
S. José dos Campos SP
Brasil.
airam@lac.inpe.br

Abstract.

The philosophy of evolution of the species, which is copied by evolutionary algorithms, is to be unfair. The best individuals compete with the worst and generally supplant it. The competition is unfair because is put in the same scenario individuals with different skills being evaluated in a rudimentary way: only the best will survive. However, the population diversity is important to preserve important features that not always make the individual strong competitor. The population diversity could be reached keeping the competition among individuals fairer. In this work, an adaptive hierarchical fair competition genetic algorithm is presented and is applied to numerical optimization problems taken from the literature.

1. Introduction

The basic idea of genetic algorithms is to store a population of individuals (or chromosomes), representing candidate solutions for concrete problems, that evolves along the time (or generations) through a competition process, where the most adapted (better fitness) have better survival and reproduction possibilities. The evolutionary process is based on individuals' selection and modification of the solutions that they represent through genetic operators as crossover and mutation.

In function parameter optimization, real-coded genetic algorithms have been successively applied and authors have reported the easiness and flexibility of its implementation [1]. On the other hand, with fixed-length binary coding, there exists a fixed amount of precision that an algorithm can be hoped to achieve, and some precision matters have to be considered before an application.

Sequential GAs have been shown to be very successful in many applications and in very different domains. However some problems could be better addressed with some form of Parallel GA (PGA). For some kind of problems, the population needs to be very large and the memory required to store each individual might be considerable. In some cases this makes it impossible to run an application efficiently using a single processor. Besides, sequential GAs may get trapped in a sub-optimal region of the search space thus becoming unable to find better quality solutions, especially for very large search space. PGAs can search in parallel different search subspaces or in different visions of such subspaces, thus making it less likely to become trapped by low-quality subspaces.

However, the most important advantage of PGAs is that in many cases they provide better performance than single population-based algorithms. The reason is that multiple populations permit speciation, a process by which different populations evolve in different directions. Thus, Parallel GAs are not only extensions of the traditional GA sequential model, but they can represent even a new paradigm that searches the space of solutions differently. The only problem that one has to face to use PGAs is to choose the parallel model to be adopted.

A recent parallel model, the Hierarchical Fair Competition (HFC), was proposed by (Hu et al., 2002) and is inspired by stratified competition often observed in society[2]. Subpopulations are stratified by fitness in castes or classes of individuals with different skills. Individuals move from low-fitness to higher-fitness subpopulations if and only if they exceed the fitness-based admission threshold of the receiving subpopulations.

HFC was applied to real-world analog circuit synthesis problem using a genetic programming (HFC-GP)[2]. This work describes the preliminary results of the HFC using a real-coded genetic algorithm (HFC-GA) applied to numerical optimization. The HFC-GA was paralleled using the Message-Passing Interface (MPI).

This paper is organized as follows. Section 2 presents some parallel evolutionary models. Section 3 presents the baseline of the HFC model. An adaptive hierarchical fair competition genetic algorithm and the obtained results for numerical optimization are presented in Section 4. The main conclusions are presents at the end of this paper.

2. Parallel Evolutionary Models

The way in which GAs can be paralleled depends upon elements such as: a) how individual's evaluation and genetic operators are performed; b) whether single or multiple (how many?) subpopulations (demes) are used; c) how individuals are exchanged; etc. Some of the models found in the literature can be classified into classes[3]:

a) Master-Slave (global) parallelization;
b) Subpopulations with migration;
c) Subpopulations with static overlapping;
d) Subpopulations with dynamic overlapping;
e) Massively parallelization.

In Master-Slave model, only evaluation of individuals and genetic operators are paralleled and such parallel processes are all dependents of the master process. This kind of global parallelization simply shows how easy it can be to transpose any genetic algorithm onto a parallel processor, without changing anything of the nature of the algorithm.

Multiple-deme (subpopulations) GAs are the most popular parallelization model. Depending upon number and the size of the demes, they can be called coarse or fine-grained models.

Coarse-grained algorithms are a general term for a subpopulation model with a relatively small number of demes with many individuals. These models are characterized by the relatively long time they require for processing a generation within each deme, and by their occasional communication for exchanging individuals (migration operator).

In fine-grained models, on the other hand, the population is divided into a large number of small demes. Inter-deme communication is realized either by using a migration operator, or by using overlapping demes. If the number of demes in fine-grained models are increased and the number of individuals in each deme are decreased, a massively parallelization is obtained.

The communication between demes can be performed, basically, by migration or overlapping (static or dynamic). The migration of individuals is controlled by several parameters, such as:

a) The topology that defines the connections between demes;
b) The frequency of migration and the amount of individuals exchanged;
c) The strategy of migration, i.e., the profile of the individuals to be exchanged.

In overlapping schema, overlapping areas are defined between demes, following some kind of topology, where some individuals belong to more than one deme. The improvement obtained by one deme is propagated through all demes by these areas.

In coarse-grained models, many topologies can be defined to connect the demes, but the most common models are the island model and the stepping-stones model. In the basic island model, migration can occur between any subpopulations, whereas in the stepping stone demes are disposed on a ring and migration is restricted to neighboring demes.

Choosing the frequency for migration and which individuals should migrate appears to be more complicated than the choice of the topology. Migrations should occur after a time long enough for allowing the development of goods characteristics in each subpopulation.

The greater advantage of the coarse-grained and fine-grained models is the possibility of better exploring the parallel potentialities. However, the coarse-grained model, in general, has smaller cost to be implemented (minor complexity of hardware and software).

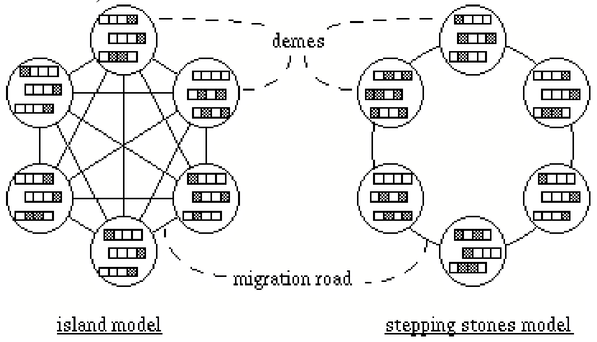

Figure 1 – Island and stepping-stones models.

3. Hierarchical Fair Competition model

The premature convergence of genetic algorithms is a problem to be overcome. The convergence is desirable, but must be controlled in order that the population does not get trapped in local optima. Even in dynamic-sized populations, the high-fitness individuals supplant the low-fitness or are favorites to be selected, dominating the evolutionary process. The philosophy of evolution of the species, which is copied by evolutionary algorithms, is unfair because is put in the same scenario individuals with different skills being evaluated in a rudimentary way: only the best will survive.

On the other hand, the population diversity is important to keep solution samples dispersed in search space, increasing the probability of finding out the global optima in multi-modal optimization problems. The population diversity could be reached keeping the competition among individuals fairer.

The Hierarchical Fair Competition (HFC) model is originated from an effort to avoid the premature convergence in traditional evolutionary algorithms[2]. The fair competition is obtained in HFC model by dividing the individuals in independent castes or classes according with their skills.

Such model is frequently observed in several advanced societies. In human society, competitions are often organized in to hierarchy of levels. None of them will allow unfair competition. For example, a primary student will not normally compete with graduate students in academic system. Even in cruel ecological systems, one can observe mechanisms of parental care to protect the young and allowing them to grow up and develop their full potentials.

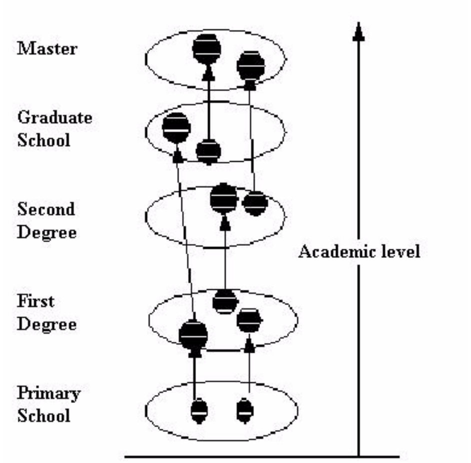

Figure 2 – Fair competition in educational system.

In HFC model, multiple demes are organized in a hierarchy, in which each deme can only accommodate individuals within a specified range of fitness. The universe of fitness values must have a deme correspondence. Each deme has an admission threshold that determines the profile of the fitnesses in each deme. Individuals are moved from low-fitness to higher-fitness subpopulations if and only if they exceed the fitness-based admission threshold of the receiving subpopulations. Thus, one can note that HFC model adopts a unidirectional migration operator, where individuals can move to superior levels, but not to inferior ones.

The following figure illustrates the topology of HFC model. The arrows indicate the moving direction possibilities. The access deme (primary level) can send individuals to all other demes and the elite deme only can receive individuals from the others. One can note that, with respect to topology, HFC model is a specific case of island model, where only some moves are allowed.

Considering, the frequency of migration, authors have proposed that individuals must be moved away in regular intervals, using admission buffers to collect qualified candidates from other populations. The strategy to determine what individuals must be exchanged is not flexible. All individuals with fitness outside the fitness range of their deme (superior individuals) must be exported to admission buffer of the appropriate subpopulation. The amount of individuals to be exchanged cannot be determined and will depend upon the number of superior individuals of each level at each exchange time.

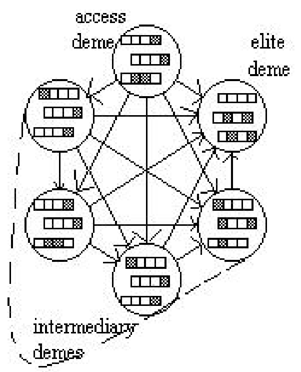

Figure 3 - HFC topology

An important feature can be incorporated to the HFC model: to work with a heterogeneous evolutionary environment. Once each subpopulation evolves individuals with different profiles, subpopulations can have different sizes, evolutionary operators and other parameters[2]. The HFC model allows employing different strategies of exploration and exploitation in each subpopulation.

The fitness range is determined considering the problem at hand and the number of demes. The authors suggest an adaptive schema using a **calibration stage** with few generations to capture an initial range of fitness values, based upon average $f\mu$, standard deviation σ_f, and the best fitness f_{min}. Afterwards, the initial population is divided into subpopulations, according to the individuals' fitness and moved to their respective demes. In regular generation intervals, the admission thresholds of all demes (but the access deme) are updated (**update stage**), evaluating $f\mu$, σ_f, and f_{min} again. The calibration and update stages together are needed to avoid the previous knowledge about the problem at hand, incorporating an adaptive feature to the algorithm.

For maximization problems, the admission threshold A_{Li} of each level can be calculated by the following formula:

Access level 0: $A_{L0} = +\infty$ (1)

Elite level N-1: $A_{LN-1} = f_{min} - \sigma_f$ (2)

Admission threshold of the other intermediary levels L_i (i=1,…,N-2), where N is the number of demes, is given by:

$A_{Li} = f\mu + (L_i-1) \times (f_{min} - \sigma_f - f\mu)/(N - 2),$ (3)

4. Computational tests

An Adaptive Hierarchical Fair Competition Genetic Algorithm (AHFCGA) was implemented using Message-Passing Interface (MPI) and was applied to numerical optimization problems. In this section, the novelties of this implementation and the computational tests are presented.

4.1 The AHFCGA

Some new features were implemented in this version of AHFCGA that was relevant to reach a greater fidelity to the HFC model. Differently that has been done by the authors, in previous works, the moving of the individuals is fully asynchronous and individuals belonging to superior levels do not participate anyway of the evolutionary process in an inferior level. In other words, the superior individuals are put on output buffer to be exported as soon they are generated or selected.

At beginning, all processors in parallel do the calibration stage. The average of $f\mu$, σ_f, and f_{min} are considered in calculation of the admission thresholds (see (1), (2) and (3)). In the following update stages the same process is done, but $f\mu$, σ_f, and f_{min} of the access level is not considered, due to the greater instability that can be inserted in the system. The authors suggest somewhat similar, except by considering only elite deme for calculations [2].

After the calibration stage, each deme has three kinds of individuals: the inferior, the belonging and the superior ones. The access deme has no inferior individual and the elite deme has no superior one. In previous works, the superior individuals are kept in deme a number of generations afterwards they are exported to their respective demes. This feature allows the synchronous exchange of individuals, once all demes can run a fixed number of generation and stopping for receiving individuals. In this work, an asynchronous exchange is implemented. Whenever a superior individual is selected, it moved to output buffer and it is not matched to crossover operation, not participating of the evolutionary process in that deme. Another individual is then selected out in his place to crossover. In the same way, whenever a superior individual is generated by crossover, it is moved to output buffer immediately.

When output buffer is full or after a maximum number of iterations, the evolutionary process is stopped and all superior individuals are exported, emptying the buffer. At follow, incoming individuals (from inferior levels) are received (if there exist) and moved from admission buffer to regular subpopulation. The sending and receiving movements are made asynchronously. There is no exact time to exchange.

The pseudo-code of the AHFCGA is presented in following. The procedure `EvolveDeme(P,MAX_GEN)` runs a real-coded GA until the output buffer is full or a maximum number of iterations. The worst individual is always chosen to be replaced by the incoming individual (steady-state updating)[4]. The stop criteria can be a specific number maximum of generations or the success to find the best-known solution. The procedure `Complement_Population(P)` is run only in access level, filling the free space caused by exported individuals. One can say that access level would work as permanent generator of individuals, feeding the whole population with new genetic material during all time life[2].

```
Initialize_Population(P);
Calibration_stage;
do
  if (UPDATING_TIME) then
      Compute_Local ($f\mu$, $\sigma_f$, $f_{min}$);
      Compute_Global($f\mu$, $\sigma_f$, $f_{min}$)
      Send (Admission_Thresholds, ALL_DEMES);
  end if
  EvolveDeme(P,MAX_ITERATIONS);
  while (OUTPUT_BUFFER NOT EMPTY)
      Send (Individual, RESPECTIVE_DEME);
  end while
  if (ACCESS_DEME)  then
      Complement_Population(P);
  else
      while (THERE_EXIST_INCOMING_INDIVIDUAL)
          Receive (Individual, ANY_WHERE);
          Update_Population(P, Individual);
      end while
  endif
while(NOT STOP_CRITERIA);
```

4.2 The Test functions

There exist many applications related to function parameter optimization (numerical optimization) well suitable to evolutionary algorithm application, such as neural network training, fuzzy set optimization, inverse problems, and others.

Several test functions can be found in literature and, frequently, many researchers have used them to study the performance of optimization algorithms. In this work, a small set of well-known test-functions was used as benchmark. The test functions *rosenbrock, schwefel, griewangk, and rastringin* are specially interesting due to be generalized functions, i.e., they can be set with n-dimensional variables. In the Figure 4, the test functions are shown.

$$f_{ROS} = \sum_{i=1}^{N-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \qquad -5{,}12 \leq x_i \leq 5{,}12$$

$$f_{SCH} = 10V + \sum_{i=1}^{N} (-x_i \sin(\sqrt{|x_i|})), \qquad -500 \leq x_i \leq 500$$

V=4189.829101

$$f_{RAS} = 20A + \sum_{i=1}^{N} (x_i^2 - 10\cos(2\pi x_i)), \qquad -5{,}12 \leq x_i \leq 5{,}12$$

A=10

$$f_{GRI} = 1 + \sum_{i=1}^{N} (\frac{x_i^2}{4000}) - \prod_{i=1}^{N} (\cos(\frac{x_i}{\sqrt{i}})) \qquad -600 \leq x_i \leq 600$$

Figure 4 – Test functions

The parameter N of the test functions is the number of variables to be optimized in order that the value of *f* is minimized. In the experiments, N was set with high values (above 50) to try the effectiveness of the implementation for large-scale problems. More information about the benchmark can be obtained in [5-6].

4.3 The experiments

For the computational tests, some parameters of AHFCGA were set. Following the suggestion found in [2], the calibration stage was run with 10 generations and the update stage procedure was called in intervals of 10 generations. The whole execution was limited to $20x10^6$ of objective function calls (reached by anyone of processors). The population and buffer sizes were set as 500 each. Other genetic parameter settings, as mutation rate, and evolutionary operators employed can be found in [4].

Once adjusted, 5 trials were performed with each one the 4 test functions. The best-found solution (FS), number of function calls (FC) and execution time (ET) were considered to evaluate the algorithm performance. FS is the average of all best solutions found in all trials; ET is the average of all worst execution times in all trials and FC is increased whenever an individual is evaluated with respect to objective function (fitness). The access deme has the feature of complementing the population and thus it has always a greater number of function calls. The following equations show how FS, FC and ET are calculated:

$$FS = average^T(\min^D(FS_{td})) \tag{4}$$

$$FC = average^T(average^D(FC_{td})) \tag{5}$$

$$ET = average^T(\max^D(ET_{td})) \tag{6}$$

where $<operation>^T$ means an operation over all trials (5 trials), $<operation>^D$ means the operation over all demes (from 4 up to 8 demes).

In order to better evaluate the AHFCGA performance, was employed a Sequential Real-coded GA (SRGA) similar to it with respect to parameters and genetic operators. A first attempt of exploring a heterogeneous evolutionary environment was made in this work. Different set of evolutionary operators was used in access and elite demes. In the access deme, a more aggressive blind mutation was employed to cause population instability (no convergence). Whereas, in the elite deme, a local search operator had the objective of accelerating the convergence rate.

In the first set of trials, it was explored variations in the number of demes without local search in both algorithms (AHFCGA and SRGA). Both algorithms were stopped either by getting the expected solution or when the maximum of $20x10^6$ objective function calls.

In the second set of trials, it was explored the local search operator in both algorithms: with certain probability (about 1%), the local search is applied to the offsprings.

Observing the results of the Table I (first set of trials), one can see that both overall performances were quite similar in terms of FS. None of them found the expected solution, but AHFCGA has reached good FS in Gri(300) and Gri(500), whereas SRGA in Ras(300). Considering FC, AHFCGA seems to perform better, but they are also similar because both have reached the maximum top of $20x10^6$ function calls. The FC, which is showed in AHFCGA's column, is just the average of all four demes. At last, with respect to ET is correct to suppose that AHFCGA have presented greater execution times probably due to the overhead inherent to the algorithm complexity. SRGA is free of exchange of individuals between population and buffers. Besides, the communication time between processors always affects the execution time of parallel algorithms and in this work the best configuration that minimizes its cost were not found.

The Table I shows set of trials with different number of processors. Other tests were made and, for while, it was not found indications that the number of processors meaningfully affects the algorithm performance. The table shows only the best set of trials.

The Table II (second set of trials) shows set of trials with 50 up to 200 dimension problems (number of variables N) running in 5 processors. Both algorithms apply local search to 1% of the offsprings. Both overall performance seems similar, except by functions *Sch(N) and Ras(N)*, where SRGA has reached all expected solutions in all trials.

Table I – Comparison between AHFCGA and a SRGA without local search

Problem		AHFCGA				SRGA		
Function(N)	Expected solution	Found solution	Function calls	Time(s)	Processors	Found solution	Function calls	Time(s)
Gri(300)	0,001	0,003	18.148.679,200	9.621,071	5	0,024	20.000.400,232	2.291,350
Gri(500)	0,001	0,008	14.652.728,600	16.404,450	5	2,735	20.000.941,554	2.724,560
Ros(300)	0,001	492,450	15.134.845,666	27.682,733	6	292,324	20.000.327,320	3.853,870
Ros(500)	0,001	293,881	14.906.582,000	13.900,931	8	508,006	20.000.743,199	4.929,760
Sch(300)	-125.694,872	-112.666,556	15.423.303,500	9.489,012	6	-119.015,114	20.000.569,543	1.631,580
Sch(500)	-209.491,454	-168.550,906	15.075.617,166	14.537,990	8	-152.370,806	20.000.998,011	2.139,420
Ras(300)	0,001	16,798	13.962.687,000	7.941,095	5	0,014	20.000.353,176	2.105,270
Ras(500)	0,001	47,210	19.265.249,200	13.875,082	5	56,252	20.000.513,101	2.767,930

Table II – Comparison between AHFCGA and a SRGA running in 5 processors with local search

Problem		AHFCGA			SRGA		
Function(N)	Expected solution	Found solution	Function calls	Time(s)	Found solution	Function calls	Time(s)
Gri(50)	0,001	0,002	6.760.905,000	609,786	0,015	20.000.733,453	401,980
Gri(100)	0,001	0,002	8.079.508,750	1.211,795	0,017	20.000.524,392	739,060
Gri(150)	0,001	0,003	9.670.970,750	1.945,880	0,003	20.001.981,220	1.072,310
Gri(200)	0,001	0,005	13.756.132,500	2.753,484	0,003	20.000.982,266	1.403,970
Ros(50)	0,001	0,027	9.470.336,750	814,264	0,003	20.000.577,098	490,120
Ros(100)	0,001	0,007	8.947.193,500	2.089,886	0,001	20.000.989,235	920,580
Ros(150)	0,001	0,014	10.846.004,250	2.425,631	0,053	20.001.928,033	1.350,070
Ros(200)	0,001	0,016	15.308.611,500	5.785,164	0,056	20.000.423,300	1.809,890
Sch(50)	-20.949,145	-20.949,144	6.659.924,000	555,838	-20.949,144	20.000.128,226	217,830
Sch(100)	-41.898,291	-40.003,274	7.704.175,000	1.577,982	-41.898,289	20.001.338,660	385,250
Sch(150)	-62.847,436	-56.017,438	9.200.378,750	1.344,362	-62.847,433	20.003.759,500	518,790
Sch(200)	-83.796,582	-71.696,182	10.634.924,250	2.823,487	-83.796,577	20.000.866,245	700,160
Ras(50)	0,001	0,001	1.287.368,000	120,828	0,000	201.691,235	2,970
Ras(100)	0,001	0,001	6.190.600,750	1.450,926	0,000	1.208.324,330	38,760
Ras(150)	0,001	1,048	6.569.484,250	1.745,440	0,000	3.930.745,200	149,340
Ras(200)	0,001	6,950	10.451.448,000	2.057,527	0,001	14.795.927,100	752,850

With respect to FC, AHFCGA has presented the less values, in average, however the elite deme has reached the maximum top of $20x10^6$ function calls, due to the local search operator. At last, SRGA has presented the less execution times in all set of trials.

5. Conclusion

This work describes the preliminary results of an Adaptive Hierarchical Fair Competition Genetic Algorithm (AHFCGA) based upon HFC model and applied to minimization of numerical functions. Some new features were implemented that was relevant to reach a greater fidelity to the HFC model as the fully asynchronous exchange of individuals and the fully stratified subpopulation. A heterogeneous evolutionary environment with local search operator also was implemented.

The potential of HFC model should taken advantage when applied to very large optimization problems. In numerical optimization fashion, large problems would have above 100 variables that is commonly seen in real-world problems in areas as engineering problems. But in first experiments done in this work, AHFCGA did not stand out when compared to a similar sequential algorithm. The probable cause of its poor performance is that AHFCGA needs careful adjustment of its parameters to avoid that each deme prematurely converges to a local optima, stopping the individuals' moving among demes and making the whole population to converge prematurely. Besides, the asynchronous AHFCGA's behavior can sometimes cause an unbalanced load among processors. Depending upon the problem in focus, some processors can be more required than others, overloading them.

The next step of this work will be to perform a consistent pool of tests aiming to find the optimal configuration, overcoming the convergence and overload troubles. Once surpassed the configuration step, the HFC model shall be compared to another parallel models, as island and stepping-stones.

Acknowledgements

The authors acknowledge FAPESP for support received in the research project "*Paralelização de Aplicações em Física dos Materiais num Ambiente de Memória Distribuída*"(proc. 01/03100-9) and CNPq for partial financial support (proc. 300837/89-5).

References

[1] Z. Michalewicz. *Genetic Algorithms + Data Structures =Evolution Programs.* Springer-Verlag, New York. 1996.

[2] J. Hu, E. D. Goodman, K. Seo, M. Pei, *Adaptive Hierarchical Fair Competition (AHFC) Model for Parallel Evolutionary Algorithms,* In Proc. of the Genetic and Evolutionary Computation Conference, GECCO-2002, New York, July, 2002, pp. 772-779.

[3] M. Nowostawski and R. Poll. *Parallel Genetic Algorithm Taxonomy*. In Proc. of the Third Intern. Conf. on Knowlege-basecl Intelligent Information Engineering Systems KES'99, IEEE Computer Society, Aug. 1999, pp 88-92.

[4] A.C.M. Oliveira, L.A.N. Lorena, Population Training Approach to Unconstrained Numerical Optimization. In: II WorCAP, São José dos Campos - SP. 2002.

[5] De Jong, K.A, *An analysis of the behaviour of a class of genetic adaptive systems*. Ph.D. Dissertation, Univ. of Michigan, Ann Arbor.1975.

[6] J. Digalakis and K. Margaritis. *An experimental study of benchmarking functions for Genetic Algorithms*. IEEE Systems Transactions, 2000, pp. 3810-3815.