Real-Coded Evolutionary Approaches to Unconstrained Numerical Optimization

Alexandre C. M. de OLIVEIRA

DEINF/UFMA
Av. dos Portugueses, s/n, Campus do Bacanga, S. Luíz MA, Brazil.
acmo@deinf.ufma.br

Luiz Antonio Nogueira LORENA

LAC/INPE
Caixa Postal 515
12.201-970 S. José dos Campos - SP Brazil
lorena@lac.inpe.br

Abstract. This paper describes an application of two evolutionary algorithms to unconstrained numerical optimization. The first is a steady-state genetic algorithm that combines some well-succeeded features of other genetic algorithms. The other is a heuristic training evolutionary approach that evolves a dynamic population to local optimal points. The algorithms are suitable to be applied to several applications in engineering and computer science. Computational tests are presented using available test functions taken from the literature.

Introduction

The basic idea of genetic algorithms is the control of a population of individuals (or chromosomes), representing candidate solutions fo r 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. Operators such as crossover and mutation base the reproduction on a process of individuals' selection and modification of the solutions that they represent.

The coding (or representation) of problem solutions in individuals is a general problem to all the approaches of evolutionary algorithms. Optimization problems can be modeled in several ways, where the variables can assume real, integer or binary code. The way as a problem is modeled strongly influences the representation and operators in genetic algorithms.

In function parameter optimization, real-coded genetic algorithms have been successively applied and authors 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. For example, the knowledge of bounds on variables bracketing the true optimal solution is required.

Several test functions can be found in literature and, frequently, many researchers have used them to study the performance of optimization algorithms. This work uses some of well-known test functions, such as michalewicz [1], rosembrock [2], schwefel [3], langerman [4], griewangk [5], and rastringin [6]. There exist many applications related to function parameter optimization well suitable to evolutionary algorithm application, such as neural network training, fuzzy set optimization, inverse problems, and others.

This work describes some experiments with two genetic algorithms with different architectures: a) a steady-state genetic algorithm with static size population ranked by individual fitness; b) a population training algorithm with dynamic size population ranked by an adaptation coefficient.

Population training approach consists in training a dynamic size population with respect to a specific-problem heuristic. In this case, the evolutionary process privileges well-adapted individuals in selection, improving its surviving time and obtaining promising results.

This paper is organized as follows. Section 1 presents an steady-state genetic algorithm and some experiments that shows its behavior. Section 2 presents the foundations of a genetic algorithm based on population training. Section 3 shows computational results using test functions taken from the literature. The main conclusions are presents at the end of this paper.

1. Steady-State Genetic Algorithm (SSGA)

The SSGA codes the function variables directly as real numbers in individuals of a static size population that evolves along the generations always preserving the best individual. It incorporates well known operators found in the literature, such as roulette wheel selection [7], blend crossover [8], and non-uniform mutation [1].

In order to speediness and simplicity, the method that updates the offspring into population was based on the steady-state method [9]. In this method, each descendant replaces one of the low-fitness individuals found in the previous updating, i.e., each crossover generates one (or more) descendant that will replace a lower-fitness individual in the population. Then, a new lower-fitness individual is found and replaced after the next crossover. One crossover means one generation so that the offspring directly compete with their parents for selections and crossover.

Steady-state method does not work with temporal gaps where a population is enough explored before be replaced by another entire one. Instead, the population contains individuals generated in all epochs. This method can be very slow because it usually ranks the population by the fitness to become the search easy. In this implementation, instead of ranking, a concept of worst neighbor is used to find out the next low-fitness individual nearest of the actual worst one that was replaced. The number of individuals that are verified to find out the next worst individual is defined as a percentage over the neighborhood. This method also maintains some diversified information in the population because it does not replace the really worse individual.

The first experiment with SSGA employs the 2-dimension Langerman's function. An initial population was generated and evolved along the generations. In the Fig1 is showed four images after 100, 500, 1000 and 5000 generations. Notice that a reasonable diversity is maintained in the first 500 evaluations, but quickly individuals are grouped around of the three promising regions (about 1000 generations). At the end, the population abandons one of these regions, staying around two local minimums, one of the which is the global minimum (about f(9.76,0.68) = -1,081) . One can see the best niche pointed out by lower arrow.

This experiment was repeated with the 2-dimension Schwefel's function that has several local minimums. Once again, the SSGA population converges quickly to lower value surfaces on search space. A snap image is showed in Fig2, after 1000 generations. One can concludes that, in spite its simplicity SSGA is able to achieve solutions in 2-dimension fashion and has a wanted behavior in these cases.

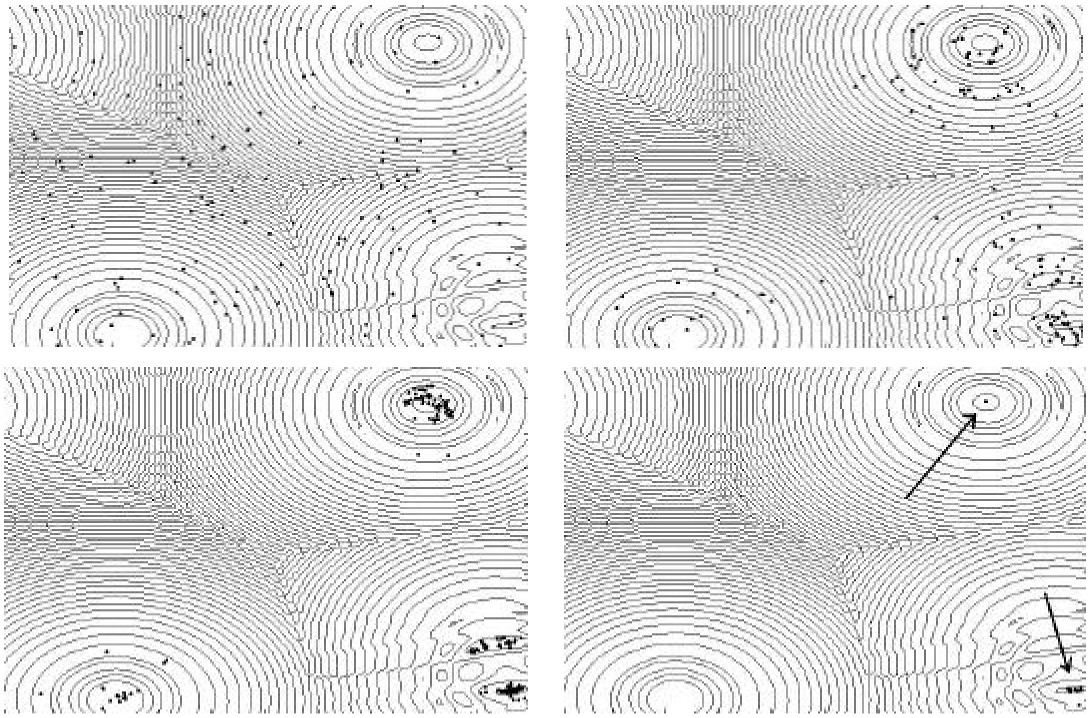
Fig 1. SSGA on 2-dimension contour map of Langerman's function after 100, 500 1000 and 5000 generations

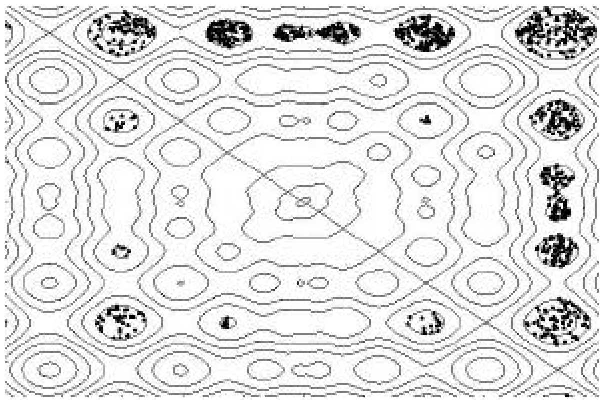
Fig 2. SSGA on 2-dimension contour map of Schwefel's function after 1000 function evaluations

2. Population Training Algorithm (PTA)

The PTA ranks a dynamic size population by a so-called adaptation coefficient. This new attribute considers both the objective function value and the error with respect to a specific training heuristic. The training heuristic is used to extract some information about the problem and guides the evolution process aiming performance improvement. A similar idea have been employed in Constructive Genetic Algorithm (CGA) proposed by Lorena and Furtado (2000) to clustering problems [10], and recently applied to permutation problems [11]. In CGA, the adaptation coefficient is also used to penalize schemata (individuals with incomplete information).

In this work, the adaptation coefficient of an individual s_k is calculated by:

$$\delta(s_k) = d\cdot(Gmax - g(s_k)) - (g(s_k) - f(s_k)) \qquad (1)$$

where Gmax is a constant upper bound to the objective function values; d is a proportionality constant with values in [0,1]; $g(s_k)$ is the objective function value for s_k; and $f(s_k)$ is the objective function value for a better neighbor of s_k, obtained by the training heuristic.

In (1) there are two components:

- Interval 1 (I1): $d \cdot (Gmax - g(s_k))$ contemplates the maximization of the difference between the upper bound Gmax and objective function value;
- Interval 2 (I2): $g(s_k)-f(s_k)$ measures the error obtained with respect a training heuristic

Before the initial population is generated, the upper bound Gmax is set, by the selection of an individual with higher objective function value, inside a small group of them, which was generated at random. The constant d is a parameter to be tuned.

Each individual that is generated receives a ranking value δ. Firstly, the function objective value $g(s_k)$ is calculated. Then, a training heuristic is applied to s_k, trying improve it. If it fails, s_k is well-adapted with respect to that heuristic. Otherwise, the amount I2 is non zero and is subtracted from I1, penalizing the individual. Thus, once Gmax is not changed along the generations, the higher rankings in minimization problems are obtained by individuals with lower values of $g(s_k)$ and $g(s_k)-f(s_k)$. In other words, the original problem is modeled as a new one: maximizing the interval I1 and minimizing the interval I2.

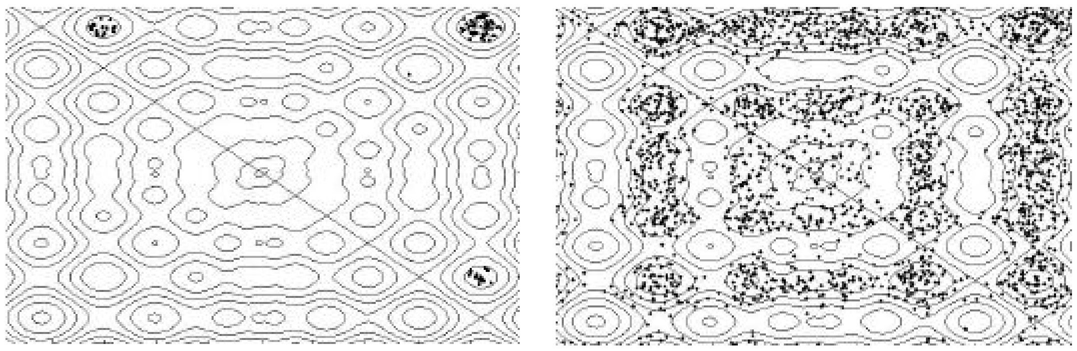

Fig 3. Interval minimization effect on 2-dimension contour map of Schwefel's function after 2000 generations: a) individuals with d.(Gmax-g) values about 80% of the best d ; b) individuals with (g-f) values equal to zero (well-trained by a heuristic)

In the Fig 3, the effect of each one of the two intervals, I1 and I2, is showed separately. The individuals with maximum I1 are positioned in most promising regions (Fig 3a), whilst the individual with minimum I2 are positioned in plateaus or regions from where they are likely to be a local minimum (Fig 3b). The entire population fills a meaningful piece of search space. The unfilled pieces are that where $g(s_k)$ closes to Gmax (high values) and the individuals initially generated in there are progressively eliminated by an adaptive rejection threshold, to be explained later.

2.1 The Training Heuristic

For better understanding the I2 interval is indispensable to explain the heuristic employed to train the population and the evolution process. In the performed experiments, the following well-known local optimizers (acting as heuristics) to numerical optimization are tested: the down-hill simplex [13], and the popular finite-difference gradient method. These methods are computationally expensive, because their complexity is a function of the dimension of the problem (number of variables on objective function). Besides, the training method shall to be applied to each individual when it is generated. For this reason a third method was built and tested, achieving good results: the in-line search method.

The in-line search method is based on the idea that it can be better to walk to well-adapted individuals. Thus, an individual s_r is chosen at random from the set of well-adapted ones. This set is defined through a percentage of population. For example, the 20% better ranked individuals form the set of individuals from where one will be selected as reference point s_r to in-line search method. Then, a fixed number of individuals (s_1, s_2, ..., s_n) are sampled along the line between s_k and s_r and the best objective function evaluation is held on as $f(s_k)$ value. In order to extrapolate the region between s_k and s_r, the method take other samples after s_r .

The Fig 3 shows a one-dimension example of the in-line search method being performed. In this case, two individuals are selected between s_k and s_r and a third one is selected after s_r, extrapolating the interval. Notice that the second sample, $f(s_2)$, had got the best F(x) (objective function value). Then, $f(s_2)$ is assigned to $f(s_k)$.

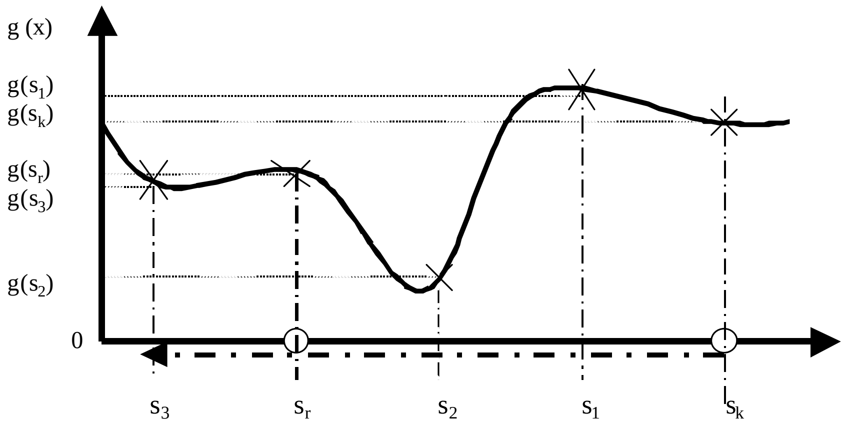

Fig 4. One -dimension example of the in-line search method

2.2 The Recombination process

The ranked individuals in expression (1) are maintained in descendent order, i.e., the first individuals are better adapted than the last ones. In this work, the selection operator was built to privileges the well-adapted individuals. Two individual are selected: one come from the population elite and the other come from the entire population. The elite group is the same used to choose the reference point in the in-line search method. A similar selection operator has been employed on Constructive Genetic Algorithms and is called base-guide selection [10]. After the selection of two individuals, the blend crossover (BLX-α, with α=0.25) is also applied to generate another individual, and this new individual can undergo a non-uniform mutation [1] with low probability (about 1%). Then, the new individual is evaluated (δ-calculation) and updated in the dynamic size population, according to its adaptation coefficient. The updating process, as well as the entire evolution process, is explained at following.

2.3 The Evolution Process

The updating process of PTA privileges the well-adapted individuals that are placed in the top of the ranked population. An adaptive rejection threshold, α, provides a progressive elimination of the ill-adapted individuals. It will be related to evolution time (generation) and also called in this form in the following.

The population at the evolution time α, denoted by P_α, is dynamic and its size varies accordingly the value of the adaptive parameter α, and can be emptied during the process. The parameter α is now compared to the ranks in expression (1), thus yielding the following expression:

$$\alpha \geq \delta(s_k) = d\cdot(Gmax - g(s_k)) - (g(s_k) - f(s_k)) \tag{2}$$

At the time they are created, individuals receive their corresponding rank $\delta(s_k)$. At each generation, these δ are compared with the current evolution parameter α. If the condition in (2) is satisfied, the individual s_k is eliminated.

Considering that the well-adapted individuals need to be preserved for recombination, the evolution parameter α is started from the lowest δ value (taken from the bottom of the ranked population), and then increases with step proportional to actual population size |P|.

$$\alpha = \alpha + k\cdot|P|\cdot\frac{\delta_{top} - \delta_{bot}}{Gr} + l \tag{3}$$

where k is a proportionality constant, l is the minimum increment allowed, Gr is the remaining number of generations, and $(\delta_{top} - \delta_{bot})$ is the actual range of values of δ.

One can observe that the adaptive increment of α is affected by the own environment (population size, best and worst δ's, etc). Thus, once the PTA achieves very good regions and does not get to improve the best rank, the parameter α goes eliminating the individuals until the population is emptied. The Fig 5 shows four snap images after 100, 500, 1000 and 5000 generations for 2-dimension Langerman's function.

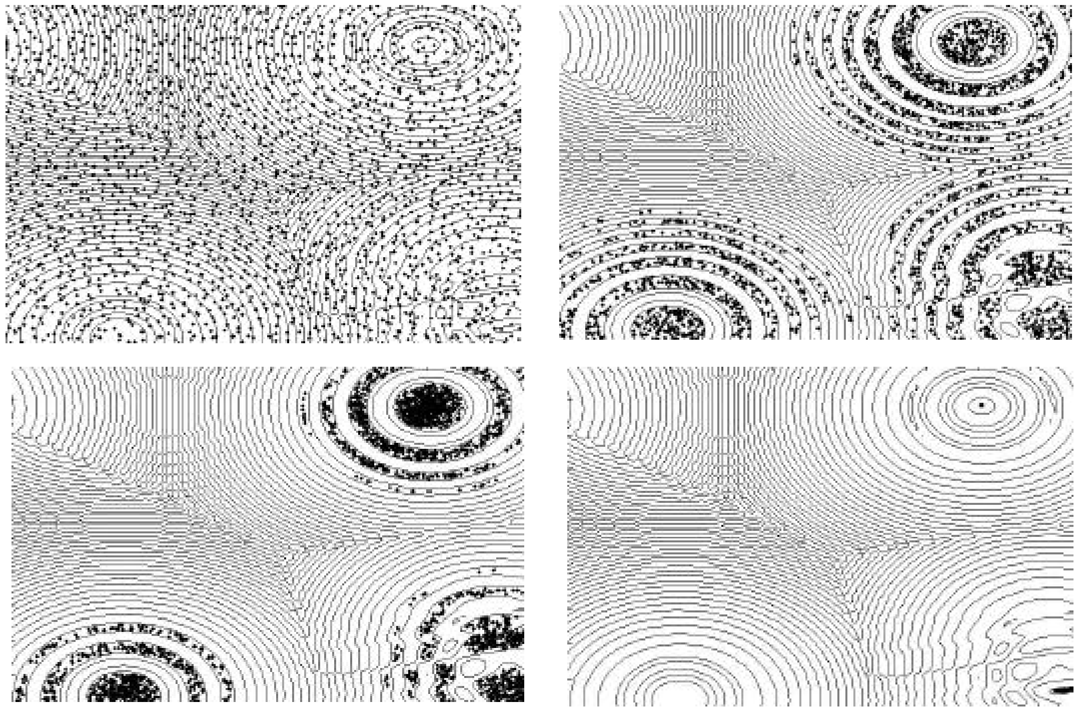

Fig 5. PTA on 2-dimension contour map of Langerman's function after 100, 500, 1000 and 5000 generations.

Notice that dynamic population fills great part of the search space and, along the generations, and the offspring are attracted to promising regions, like SSGA, but the density of individuals is reasonably greater. Perhaps this fact can contribute for an intensification of pressure over the local minimums as well as over the global one. At the end, about generation 5000, the population was practically emptied and few individuals are left in two minimums, including the same global minimum found by SSGA. The overall numerical results of all experiments that were made with both SSGA and PTA approaches are summarized and commented in next section.

3 Computational Tests

The SSGA and PTA were coded in ANSI C and run on Intel Pentium III (450Mhz) hardware. For the computational tests, some parameters were adjusted in both approaches. Once adjusted, 20 trials were performed with 6 test function in the 5-dimensions and 10-dimension cases, giving 240 trials with each approach.

The best results found with SSGA were obtained with non-uniform mutation and blend crossover probability equal to 5% and 100% respectively and a initial population of 300 up to 1000 individuals. The next worst individual is looked for in 10% of the neighborhood of the actual one. The blend crossover parameter α is equal to 0.25, as well as in PTA. The best results found in PTA were obtained with d parameter set to 0.10. This configures the proportionality between intervals I1 and I2. The k and *l* constants used at the step size of the adaptive rejection threshold α were set to $1 \cdot 10^{-2}$ and $1 \cdot 10^{-5}$ in most of the test functions. But in 10-dimension Langerman's function, it was not possible to find the right values to them. The results for this test function were under the expectation. The non-uniform mutation probability was set to 1%.

The stop criterion is: a) the expected best solution is found; or b) after 100 generations the best individual is not changed; or c) the limit of 751000 evaluations of the objective function is reached. The expected best solutions used on stop criteria are known in the literature and shown in Table I. Some executions of SSGA did not reach best solutions and had an excessive number of functions evaluations. These function evaluations are also shown in Table I.

For each test function, the average (M) of all the solutions found is calculated and the percentage of the error related to the expected solution can be calculated by [M-f(s*)], where f(s*) is the expected best solution.

Table I - Expected best solution

	Solution (avg)		Expected solution	Error		Evaluation (avg)	
	PTA	SSGA		PTA	SSGA	PTA	SSGA
Rosembrock(5)	0.012	0.351	0.001	0.011	0.350	486024.50	534000.00
Rastrigin(5)	0.001	0.001	0.001	0.000	0.000	178334.30	98992.70
Griewangk(5)	0.009	0.006	0.001	0.008	0.005	304628.80	540362.45
Langerman(5)	-0.963	-0.358	-1.400	0.437	1.042	103839.00	751024.00
Michalewicz(5)	-4.682	-4.687	-4.687	0.005	0.000	83140.80	92.533.80
Schwefel(5)	-2094.672	-2094.908	-2094.914	0.243	0.006	359142.70	404562.20
Rosembrock(10)	0.102	6.640	0.001	0.101	6.639	703015.50	751000.00
Rastrigin(10)	0.007	0.001	0.001	0.006	0.000	142098.40	243424.20
Griewangk(10)	0.009	0.009	0.001	0.008	0.008	353668.30	574628.80
Langerman(10)	-0.004	-0.003	-1.400	1.396	1.397	422314.40	751000.00
Michalewicz(10)	-9.555	-7.693	-9.660	0.105	1.967	691504.10	751000.00
Schwefel(10)	-3935.442	-3952.952	-4189.829	0.000	0.000	368686.50	544419.20
				2.319	**11.415**	**349699.775**	**503078.946**

Table I also shows the comparison between PTA and SSGA for 5-dimension and 10-dimension test functions. In 5-dimension, PTA approach presents a best overall performance, even without considering the poor performance of SSGA in rosenbrock's function (350% of average error). Thus, PTA obtains 7.66% of average error against 15,88% obtained by SSGA. In 10-dimension, once again SSGA runs poorly in rosenbrock's function (the best solution found was 6.10, i. e., above 6000% of error). Excluding the rosenbrock's result of both, there exist an equilibrium of performance: PTA with 23,95% of error and SSGA with 26,75%.

In 5-dimention, PTA had better performance in rosenbrock's, rastrigin's, and langerman's functions, although in this last, both approaches did not have any success. In 10-dimention, PTA was better than SSGA in rosembrock's, griewangk's , langerman's and michalewicz 's functions.

Conclusion

This work describes two real-coded evolutionary algorithms for numerical unconstrained optimization. The SSGA combines some successful features of other genetic algorithms, and the PTA proposes a new concept of population training, working with a dynamic population. The SSGA and PTA algorithms are tested with some well-known test instances (functions) obtaining comparable good solutions. Some feature figures confirm their good overall behavior seeking for local optimum. The PTA also has the reinforced effect of move away from unpromising solutions. Some further work is allowed on new heuristics for PTA and its adaptation to constrained problems.

Acknowledgements: The first author acknowledges the Programa Institucional de Capacitação Docente e Técnica -PICDT/CAPES for financial support. The second author acknowledges Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (proc. 300837/89-5) and Fundação para o Amparo a Pesquisa no Estado de S. Paulo - FAPESP (proc. 99/06954-7) for partial financial support.

References

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

[2] 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.

[3] Schwefel, H.-P.: Numerical optimization of computer models. Chichester: Wiley & Sons, 1981.

[4] H. Bersini, M. Dorigo, S. Langerman, G. Seront, and L. M. Gambardella, "Results of the first international contest on evolutionary optimisation (1st ICEO)," Proceedings of IEEE International Conference on Evolutionary Computation, IEEE-EC 96, 1996, IEEE Press, pp. 611--615.

[5] Griewangk, A.O., Generalized Descent for Global Optimization, JOTA, vol. 34, 1981, pp. 11 - 39

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

[7] Goldberg D. E., Genetic Algorithms in Search, Optimisation, and Machine Learning. Addison-Wesley Publishing Company, Inc. 1989.

[8] Eshelman, L.J. Schawer, J.D. Real-coded genetic algorithms and interval- schemata, in Foundation of Genetic Algorithms-2, L. Darrell Whitley (Eds.), Morgan Kaufmann Publishers: San Mateo, 1993, pp. 187-202.

[9] Davis, L., Handbook of Genetic Algorithms, Van nostrand Reinhold, 1991.

[10] L. A. N. Lorena. and J. C. Furtado. "Constructive genetic algorithm for clustering problems." Evolutionary Computation 9(3): 309-327. 2001.

[11] A. C. M. Oliveira and L. A. N. Lorena, "A Constructive Genetic Algorithm for Gate Matrix Layout Problems". Accepted to IEEE Transaction on Computer-Aided Designed of Integrated Circuits and Systems. 2002.

[12] Nelder, J. A. and Mead, R. A simplex method for function minimization. Computer Journal, vol. 7, pp. 308--313, 1965.