Control of search parameters in evolutionary algorithms

João Pedro Pedroso

Riken Institute, Laboratory for Information Synthesis
Hirosawa 2-1, Wako-shi, Saitama 351-01, Japan
e-mail: jpp@brain.riken.go.jp

***Abstract*—** In this paper we present a strategy for automatically adapting the control parameters of an evolutionary algorithm. Its main features consist on its simplicity, and on providing total independence of the type of problem being solved.

I. Introduction

Evolutionary algorithms have many well-known applications in the solution of non linear optimisation problems. They are generally considered to be very reliable methods, although sometimes problems with premature convergence arise. In these cases, the choice of the algorithm's parameters can be very important for obtaining a good solution.

In this paper we introduce a strategy for automatic adaptation of the search parameters. One of the motivations for the implementation of such a strategy is to provide a completely autonomous system, which would be able to tackle any optimisation problem of its class without requiring the user to set up the control parameters manually. The other motivation concerns the ability to perform the evolutionary search for an unspecified amount of time, possibly with changes in the form of the objective function in the middle, always trying to keep appropriate search parameters.

The strategy proposed is largely empirical, as it is very difficult to accurately keep track of the whole status of the system, and predict exactly what action should be taken. It consists firstly on an exploitation of the search when the parameters are favourable to the evolution of the current population, and hence the objective is being ameliorated. This is done until we obtain no improvement in a generation of the evolutionary algorithm. When this arises new control parameters are randomly determined. An improvement to this basic technique is done by adding a memory to the control system, which would restore old values of the control parameters if the newly tested ones did not provide an improvement. Finally, a simple system for avoiding premature convergence is implemented, which forces an increase in the mutation parameters if too many identical individuals are found in the population.

We test the strategy proposed on a set of benchmark functions available in the literature, and compare the results obtained to the case where the parameters are fixed, either randomly or at their optimal value.

The results presented were obtained with an implementation based on floating point representation of the solution, and a particular shape of the mutation, recombination, selection, and scaling operators. The ideas that motivate the strategy proposed should, nevertheless, be valid for other implementation paradigms.

II. The genetic operators

The problems we deal with in this paper are nonlinear optimisation problems on a continuous domain. The genetic representation of solutions in our system is identical to the mathematical one: vectors of floating point numbers. So, for a problem of dimension n, the solution is a vector $x = (x_1 \dots x_n)$; we call each x_i a *chromosome*, and the vector x a *genome*.

The generation of a new individual from two parents is composed of three steps: meiosis, possibly complemented with crossover, possibly followed by mutation. The meiosis and crossover processes produce a linear combination of two individuals selected from the population. The mutation adds a random perturbation to the solution created this way. All these operators are characterised by two parameters: probability of occurrence and intensity of the operation.

We use the following notation: ν^p, χ^p, μ^p, are the probabilities of meiosis, crossover, and mutation, respectively; ν^s, χ^s, μ^s are their respective intensities. r is a random number uniformly distributed in $[0, 1]$, and $\delta(s) = 1 - r^{s^2}$ is the distribution of the perturbations, where s is the intensity.

For creating a new solution x from two parents y and z, the process of reproduction is presented in figure 1.

The meiosis operation creates an offspring genome from two parents. This is done through random selection of chromosomes from either the mother or the father, possibly complemented by crossover, until the genome has all its chromosomes.

Crossover takes on two chromosomes, and recombines them, leading to an intermediary value. The

```
if $r < \nu^p$                              Do the meiosis with probability $\nu^p$
    for $i = 1$ to $n$ do
        if $r < \nu^s$          (Note: $\nu^s$ measures the intensity of meiosis.)
            if $r < \chi^p$                 With some probability do crossover,
                set $x_i := y_i + (z_i - y_i)\,\chi^s r$     with intensity $\chi^s$
            else
                set $x_i := y_i$                no crossover, exact copy of $y_i$
        else
            . . .               do the same, swapping the roles of $y$ and $z$
    done
else                                         In this case, no meiosis occurs:
    set $x := y$, or $x := z$         copy exactly $y$ or $z$, with same prob.
for $i = 1$ to $n$ do       Now, do the mutation: for each element of $x$
    if $r < \mu^p$                                    with probability $\mu^p$
        $x_i := x_i \pm \delta(\mu^s)$               add mutation of intensity $\mu^s$.
done
```

Figure 1: Overview of the genetic operations.

smaller the "crossover intensity" parameter is, the closer the produced chromosome is to that of one of the parents.

Mutation adds, with some probability, a random perturbation to the value obtained that way. For each mutation, we randomly choose to add or subtract $\delta(s)$ to the value of the chromosome[1], where s gives the intensity, or magnitude of the mutation.

An additional parameter used in the algorithm is the selectivity, a factor that specifies how competitive an individual must be in relation to the average in order to have a favoured probability of being selected. This parameter, denoted by σ, acts on the scaling of the individuals' fitness prior to their selection for reproducing. If σ is close to 0, the differences between individuals are attenuated, and selection is close to random. When σ is close to 1, the better individuals are strongly favoured, and tend to be the only selected. (See [6] for more details on this scaling technique.) The selection scheme used is roulette wheel selection; for its description see, for example, [2].

III. The benchmark test bed

For the evaluation of the strategies that we propose in this paper, we have relied on the set of benchmark tests proposed in [1]. The set of test functions is the following:

Problem 1: the sphere model. Value to reach: 10^{-6}.

$$f_1(x) = \sum_{i=1}^{N} (x_i - 1)^2$$
$$x_i \in [-5, 5] \quad i = 1, \ldots, N$$

Problem 2: Griewank's function. Value to reach: 10^{-4}.

$$f_2(x) = \frac{1}{d} \sum_{i=1}^{N} (x_i - 100)^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i - 100}{\sqrt{i}}\right) + 1$$
$$d = 4000$$
$$x_i \in [-600, 600] \quad i = 1, \ldots, N$$

[1]The value of the perturbation is scaled, so that it covers the whole region between the value x_i and its bounds.

Problem 3: Shekel's foxholes. Value to reach: -9. (c_i, $A(i)$ available in [1]).

$$f_3(x) = -\sum_{j=1}^{m} \frac{1}{||x - A(j)||^2 + c_j}$$
$$m = 30$$
$$x_i \in [0, 10] \quad i = 1, \ldots, N$$

Problem 4: Michalewicz' function. Value to reach: -9.66 (for $N = 10$).

$$f_4(x) = -\sum_{i=1}^{N} \sin(x_i) \cdot \sin^{2m}\left(\frac{i \cdot x_i^2}{\pi}\right)$$
$$m = 10$$
$$x_i \in [0, \pi] \quad i = 1, \ldots, N$$

Problem 5: Langerman's function. Value to reach: -1.4 (c_i, $A(i)$ available in [1]).

$$f_5(x) = -\sum_{j=1}^{m} c_j \cdot e^{-\frac{||x - A(j)||^2}{\pi}} \cdot \cos(\pi \cdot ||x - A(j)||^2)$$
$$m = 30$$
$$x_i \in [0, 10] \quad i = 1, \ldots, N$$

For all the results reported in this paper, we have used benchmark problems of dimension 10 (N=10). The performance indexes, measured on 100 independent runs, are the following:

- the best solution obtained for all runs.
- the average solution obtained.
- the number of successes, i.e., the number of times the value to reach was obtained.

In order to have a reference for comparing the performances for the different strategies employed, the evolutionary system used was set up with a population of 25 elements, evolving for 2500 generations. Under these circumstances, the difficult benchmark problems are virtually unsolvable, in a systematic way, by the evolutionary algorithm. It is, thus, a good basis where to observe effects of the control parameters in the evolution.

IV. Results with a standard algorithm

For the purpose of having a term of comparison for the strategy of adapting the control parameters that we propose in this paper, we have first made a series of runs with static parameters.

A. Random parameters

The first series of results were obtained for random control parameters. We have performed a series of 100 independent runs, each with exogenous, independent, randomly determined control parameters. Results are presented in table 1.

Table 1: Results with random parameters.

Problem	Best sol.	Mean best sol.	# successes
Sphere	4.8e-07	2.31	1
Griewank	0.0429	7.85	0
Shekel	-10.16	-1.15	1
Michalewicz	-9.93	-7.55	14
Langerman	-1.499	-0.513	2

Table 3: Results for optimised search parameters.

Problem	Best sol.	Mean best sol.	# successes
Sphere	4.4e-13	1.5e-11	100
Griewank	7.4e-03	5.6e-02	0
Shekel	-10.2	-1.71	2
Michalewicz	-9.98	-9.95	100
Langerman	-1.31	-0.615	0

B. Optimal parameters

We have next attempted to optimise the control parameters using another evolutionary algorithm. This method was inspired in the one described in [3], the multilevel genetic algorithm, where a meta-genetic algorithm is used to tune the control parameters of another genetic algorithm, making this last adaptive. In our case, we have set the meta problem as a completely independent one, whose objective is to find the control parameters which would lead to optimal, or more reliable, performance of the inner algorithm.

As it turned out, for the benchmark test bed used and for the limited number of generations and population size, this meta algorithm was rather unreliable. Its solution—the control parameters of the inner algorithm—did not stabilise. This was due to the nature of the meta objective functions, which are extremely noisy. For the same value of the meta solution (the control parameters), one could obtain many different values of the meta fitness (the best objective found by the inner algorithm, using those control parameters). In order to select more reliable parameters, we decided to take on the 1% best meta solutions obtained during the evolutionary process, and average their values. These results are presented in table 2. Note that, except for the case of the sphere model (the only unimodal benchmark), these values were quite disperse.

Table 2: Optimal parameters for each benchmark.

Problem	μ^p	μ^s	χ^p	χ^s	ν^p	ν^s	σ
Sphere	.0086	.021	.63	.18	.86	.52	.72
Griewank	.032	.13	.56	.37	.38	.61	.74
Shekel	.24	.63	.51	.33	.54	.57	.60
Michalewicz	.043	.68	.72	.35	.66	.48	.64
Langerman	.54	.20	.42	.45	.73	.69	.60

We have then used these control parameters to tune the evolutionary algorithm. The results obtained this way, for 100 independent runs, are reported in table 3.

As these results clearly show, setting the appropriate values of the control parameters is a rather difficult task. The optimisation with another evolutionary algorithm does not lead to parameters providing a reliable final performance for all the problems, although it involves a huge amount of computational time for the solution of the meta problem. During the meta optimisation process, all the optimal values of each benchmark were obtained by the inner algorithm; but we could not find the parameters which would allow finding them in a systematic way. Furthermore, the meta solution (the control parameters of the inner optimisation) would alternate regions of good and poor performance; absolute convergence was not observed.

An additional drawback of this procedure is that the parameters determined are highly dependent on the benchmark. Good parameters for one problem are not, in general, still good for another one, as the results in table 2 show. Hence, if a new problem would be to be solved systematically with reasonable performance, a new meta optimisation would also be involved.

V. Adaptation strategies

A. Randomly perturbing parameters

In order to deal with the extreme complexity of the adaptation of control parameters problem, we could nor devise solutions other than those of, equally extreme, simplicity.

A first strategy consists on the following: we start the evolutionary algorithm with random parameters. We then create a new population, using these parameters for the generation of its individuals. Then, if the solution was improved, we keep the parameters; if not, we set new random parameters. We then perform another generation. This process is repeated until the termination condition is met. New parameters are determined with uniform random distribution in their domain, $[0, 1]$.

Results obtained with this strategy are reported in table 4. Although they are far from being satisfactory, this strategy already provides some desirable features: there is no need of external interference for the determination of the control parameters, and it is independent of the benchmark problem.

Table 4: Results for randomly adapted parameters.

Problem	Best sol.	Mean best sol.	# successes
Sphere	3.29e-08	1.89e-4	3
Griewank	0.025	0.184	0
Shekel	-10.2	-1.66	2
Michalewicz	-9.93	-9.75	89
Langerman	-1.5	-0.568	5

Table 5: Results for random adaptation with memory.

Problem	Best sol.	Mean best sol.	# successes
Sphere	3.93e-14	5.37e-06	69
Griewank	0.0148	0.0975	0
Shekel	-10.2	-1.76	3
Michalewicz	-9.98	-9.87	100
Langerman	-1.5	-0.617	4

B. Simple adaptation of parameters

A first improvement that can be made to this strategy concerns implementing a primitive memory: the system remembers what was the last action taken.

At each generation, we randomly select what parameter to "adapt". We then set it randomly, and keep in memory the old value that was being used. If the action was successful (i.e., the objective was ameliorated in the subsequent generation), then the parameter is kept; otherwise, we restore the previous value. This being done, we randomly choose another parameter to "adapt", and pursue this way until a termination criteria is met.

A second improvement consists on implementing a sentinel for checking premature convergence. In some situations, if the mutation intensity is too small, we may reach a population composed of individuals all around the same point; and, as there are still some small mutations being done, minimal improvements might be occurring, leading for no action in the parameter adaptation. In order to prevent this, we implemented a system for checking if all the individuals are similar. We define two individuals y and z as *not similar* iff $\exists i : ||y_i - z_i|| > 0.01 B_i$, where B_i is the difference between the upper and the lower bound for the chromosome i.

If at any given point of the evolution all individuals are similar, we have premature convergence; in this case, we force the mutation parameter to be increased. Note that this only provides a faster way for leaving premature convergence: a random perturbation of the mutation intensity parameter would sooner or later arise, with an equivalent effect. But this can avoid wasting too much time around a particular solution, doing only a very localised search.

Results obtained with this strategy are reported in table 5. An overall improvement of the performance for all the benchmark problems is observed, both in terms of reliability and in terms of quality of the solutions.

VI. Conclusion

In this paper we present a simple strategy for automatically adapting the control parameters of an evolutionary algorithm. It basically consists of randomly perturbing them whenever a new generation of the system does not lead to and improved solution.

Its nice characteristics, apart from its extreme simplicity, consist on providing total independence with respect to the type of problem being solved. There is no need of external interference for the determination of control parameters, independently of the problem being solved.

Some tests with benchmark problems available in the literature tend to indicate that this strategy increases both the reliability of the evolutionary system and the quality of the solutions found.

Future improvements that we foresee to this strategy consist on finding a more adequate distribution for the random perturbation of the parameters, which could potentially permit further improvements on the speed and on the quality of the solutions.

References

[1] H. Bersini, M. Dorigo, L. Gambardella, S. Langerman, and G. Seront. First international contest on evolutionary optimization, 1996. In IEEE International Conference on Evolutionary Computation.

[2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley, 1989.

[3] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions od Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[4] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[5] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, second, extended edition, 1994.

[6] J. P. Pedroso. Niche search: an evolutionary algorithm for global optimisation. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature IV*, volume 1141 of *Lecture Notes in Computer Science*, Berlin, Germany, 1996. Springer.