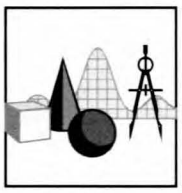

AN APPLICATION EXAMPLE OF THE BREEDER GENETIC ALGORITHM TO FUNCTION OPTIMIZATION

Lluís A. Belanche

Secció d'Intel·ligencia Artificial
Dept. de Llenguatges i Sistemes Informatics
Universitat Politecnica de Catalunya
belanche@lsi.upc.es

ABSTRACT

Evolutionary Algorithms (EA) have demonstrated their ability to solve optimization tasks in a wide range of applications. In this paper, after outlining the basics of such algorithms, the possibilities of one of the latest to emerge, the Breeder Genetic Algorithm (BGA) are exemplied by addressing a classical numerical optimization problem: the Fletcher-Powell pseudo-random function.

1. INTRODUCTION

Since the early days of computer science, function optimization has been one of the topics of more active research, because of the enormous potential of application to solve engineering or mathematical problems. For discrete solution spaces, traditional Articial Intelligence (AI) techniques for solving constrained combinatorial optimization tasks, such as the A* or IDA* algorithms and derivations [1] have proven to be useful for certain applications. However, these methods rely on heuristics to guide the search process that are not always easy to fi nd or to express. In addition, if the search space is big, the computational cost of the algorithm can become unacceptably high.

An alternative to these methods are Evolutionary Algorithms (EAs) [2], among which Genetic Algorithms (GAs) [3] have been extensively used for dis-crete as well as continuous optimization tasks, although they were not conceived for this [4]. Their main advantages over other methods are the coarsegrained *global search* mechanism, the neat tradeoff between *exploration* of the search space for new solutions and *exploitation* of the promising ones, the lack of an explicit heuristics, and the easiness of parallelization. In addition, no knowledge of the problem has to be coded apart from telling the algorithm how good (how *fit*, in EA jargon) a potential solution is.

Among the latest algorithms we find the Breeder Genetic Algorithm (BGA) [5]. It is characterized by a truncation selection procedure and direct representation of continuous variables. Truncation selection is a simple use of rank-based selection, proven to be very useful in traditional GAs [6]. The direct representation of variables eliminates the need for a coding scheme -that usually changes the search space- and permits to develop new, continuous genetic operators. The BGA has been shown to be superior to traditional GAs for classical continuous optimization test problems [7].

We show in this paper how a relatively simple algorithm like the BGA is able to cope with the Fletcher-Powell function, an extremely difficult optimization problem, often used for benchmarking. Our intention is not to solve this problem to a full, achieving the best attainable solution, because this would require a more thorough study and much higher computational demands than those used here. Rather, we use it to illustrate the possibilities of EAs as black-box and robust techniques for optimization.

In the next two sections, the basics of EAs are outlined, and in particular the BGA is briefly introduced. It then follows a description of the selected problem. Next, a small experimental setup is devised and the results on it are commented on, together with a comparison to other EAs. The paper ends with some conclusions and afterthoughts.

2. BASICS OF AN EVOLUTIONARY ALGORITHM

The term Evolutionary Algorithms refers to a big family of search methods based on concepts taken from Darwinian evolution of species and natural selection of the fittest. Some concepts from genetics are also present. Given a problem to be solved EAs maintain a population of *individuals* that represent potential solutions to it. Each individual in the population is represented by a *chromosome* consisting of a string of atomic elements called *genes*. Each gene contains (represents) a variable, either for the problem or for the algorithm itself. The possible values of a gene are called *alleles* and the gene's position in the chromosome is called *locus* (pl. loci). There is also a distinction between the *genotype*, the genetic material of an individual, and the *phenotype*, the individual result of genotype development (that is, the born living thing). In EAs the genotype coincides with the chromosome, and the phenotype is simulated via a *fitness function*, a scalar value -similar to a reinforcement- expressing how well and individual has come out of a given genotype.

An EA can be formally described by the conceptual algorithm in g. 1, parameterized by a tuple:

$$< \text{EA-Setup} >=< \Pi_0, (\mu,\lambda), \Upsilon,\Omega, \Psi, \Theta, \Phi, \Xi >$$

where $\Pi_t = (i^t_1, i^t_2, \ldots i^t_\mu)$ is the population at time t and thus Π_0 is the, usually random, initial population, μ the population size, λ the offspring size (out of μ), Υ the selection operator, Ω the recombination operator, Ψ the mutation operator, Θ the termination criterion, Ξ the replacement criterion and Φ the fitness function. In this algorithm, operator sequencing on the popula-tion is as follows: Π_t represents the population at time (i.e., generation) t, Π^Υ_t the population after selection, Π^Ω_t after recombination and Π^Ψ_t after mutation, to end in a new population Π_{t+1}.

The search process usually starts with a randomly generated population Π_0 and evolves over time in a quest for better and better individuals where, from generation to generation, new populations are formed by application of three fundamental kinds of operators to the individuals of a population, forming a characteristic three-step procedure:

1. *Selection* of the fittest individuals, yielding the socalled *gene pool*;

2. *Recombination* of (some of) the previously se-lected individuals forming the gene pool, giving rise to an o spring of new individuals;

3. *Mutation* of (some of) the newly created individuals.

```
Procedure Evolutionary-Algorithm
{
        t:=0;
        create Π_t;
        evaluate Φ(i),∀i ∈ Π_t;
        while not (Θ(Π_t)) do
        {
                /* Create the gene pool Π^Υ_t */
                   select: Π^Υ_t := Υ(Π_t);

                /* Apply genetic operators */
                   recombine: Π^Ω_t := Ω(Π^Υ_t);
                   mutate: Π^Ψ_t := Ψ(Π^Ω_t);

                /* Evaluate their effect */
                   evaluate Φ(i), ∀i ∈ Π^Ψ_t;

                /* Form the new generation */
                   replace: Π_t+1 := Ξ(Π^Ψ_t ∪ Π^Υ_t);
                   t := t+1
        }
}
```

Figure 1: Evolutionary Algorithm.

By iterating this three-step mechanism, it is hoped that increasingly better individuals will be found (that is, will appear in the population). This reasoning is based on the following ideas:

1. The selection of the fittest individuals ensures that only the best ones[1] will be allowed to have o spring, driving the search towards good solutions, mimicking the natural process of selection, in which only the more adapted species are to survive.

2. By recombining the genetic material of these selected individuals, the possibility of obtaining an offspring where *at least* one child is better than any of its parents is high.

3. Mutation is meant to introduce new traits, not present in any of the parents. It is usually performed on freshly obtained individuals by slightly altering some of their genetic material.

Hence, an EA may be seen as a non-empty sequence of ordered operator applications: fitness evaluation, selection, recombination, mutation and replacement.

The entire process iterates until one of the following criteria is fullfiled:

1. *Convergence*: it happens because the individuals are too similar. Fresh and new ideas are needed, but recombination is incapable of providing them because the individuals are very close to one another, and mutation alone is not powerful enough to introduce the desired variability. Convergence can be monitored by on-line (average of the best individuals) and off-line (average of average individuals) throughout the generations;

2. *Problem solved*: the global optimum is found up to a satisfactory accuracy (if optimum known);

3. *End of resources*: the maximum number of function evaluations has been reached.

Evolutionary Algorithms are effective mainly because their search mechanism keeps a well-balanced tradeoff between *exploration* (trying to always drive the search to the discovery of new, more useful, genetic material) and *exploitation* (trying to fine-tune good already-found solutions). Exploration is mainly dealt with by the mutation operator. Exploitation is carried out by the selection process and the use of recombination operators, although mutation may also play a role in the netuning of solutions. The tness function is built out of the function to be optimized (called the *objective function*). All EAs represent the decision variables in the chromosome in one way or another, either directly as real values or resorting to a discrete coding, usually binary (like most GAs). The particular coding scheme is the classical knowledge representation problem in AI, and completely conditions the results. In addition, some algorithms (like the Evolution Strategies, ES) append their own variables to the representation in the form of auxiliary information that evolves with time like the other variables.

An excellent state-of-the-art and review of EAs, and a useful departure point because of its rich set of references

[1] Or the luckiest in some EA instances, like most GAs.

is [8]. There is also a very complete FAQ with pointers to papers, books, software and the main groups working on EAs all over the world [9].

3. THE BREEDER GENETIC ALGORITHM

In traditional GAs, selection is stochastic and meant to mimic -to some degree- Darwinian evolution; instead, BGA selection (named *truncation* selection) is a deterministic and articial procedure driven by a *breeding* mechanism (as used in livestock), where only the best individuals -usually a fixed percentage of τ total population size μ- are selected and enter the gene pool to be recombined and mutated, as the basis to form a new generation. Genetic (recombination and mutation) operators are applied by randomly and uniformly selecting two parents until the number of offspring equals μ-q. Then, the former q best elements are re-inserted into the population, forming a new generation of μ individuals that replaces the previous one. This guaranteed survival of some of the best individuals is called *q-elitism.* For the BGA, the typical value is q = 1. The BGA selection mechanism is then deterministic (there are no probabilities), extinctive (the best elements are guaranteed to be selected and the worst are guaranteed *not* to be selected) and 1-elitist (the best element is always to survive from generation to generation).

The BGA chromosomes are potential solution vectors $\vec{x}$ of n components, where n is the problem size, the number of free variables of the function to be op-timized. The common aspect of BGAs with ordinary GAs is the fact that both are mainly driven by recombination, with mutation regarded as an important but background operator, in the double role of solution fine-tuner (for very small mutations) and as the main discovery force (for moderate ones). We will now briefly describe the different possibilities for the genetic operators. The reader is referred to [10] for a detailed description.

3.1 Recombination

Any operator Ω combining the genetic material of the parents is called a recombination operator. In BGAs, recombination is applied unconditionally, $Pr(\Omega) = 1$. Let $\vec{x} = (x_1, \ldots, x_n)$, $\vec{y} = (y_1, \ldots, y_n)$ be two selected gene-pool individuals $\vec{x}$, $\vec{y}$ such that $\vec{x} \neq \vec{y}$. Let $\vec{z} = (z_1, \ldots, z_n)$ be the result of recombination and $1 \leq i \leq n$. The following are some of the more common possibilities to obtain an o spring $\vec{z}$:

1. Discrete Recombination (DR).

$$z_i \in \{x_i, y_i\} \quad \text{(chosen with equal probability)}$$

2. Line Recombination (LR).

$$z_i = x_i + \alpha(y_i - x_i)$$

with a fixed $\alpha \in [0, 1]$. Typically, $\alpha = 0.5$.

3. Extended Intermediate Recombination (EIR).

$$z_i = x_i + \alpha_i(y_i - x_i)$$

with α_i $[-\delta, 1+\delta]$ chosen with uniform probability. The parameter δ expresses to what degree an offspring can be generated out of the parents's scope, the imaginary line that joins them in $\Re$. More precisely, it works by controlling the maximum fraction $a = \delta[y_i - x_i]$ of the distance between parents where the offspring can be placed, either left to the leftmost parent or right to the rightmost parent. A typical value for $\delta = 0.25$, although any non-negative real number not exceeding 0.5 is a potential value. The bigger the , the more the effect of the parents is diminished in creating offspring. A method for dynamically setting its value called range, was introduced in [10] and shown to have a remarkable effect in performance. It works as follows:

$$z_i = y_i + \alpha_i (x_i - y_i), \qquad \text{with } x_i \geq y_i$$

such that $\alpha_i \in [-\delta_i^-, 1+\delta_i^+]$ with uniform probability and,

$$\delta_i^- = \frac{y_i - r_i^-}{r_i^+ - r_i^-}$$

$$\delta_i^+ = \frac{r_i^+ - x_i}{r_i^+ - r_i^-}$$

This procedure assigns different values for the left (δ_i^-) and right (δ_i^+) limits of the interval from which α_i is to be selected, and does never generate a value outside the range $[r_i^-, r_i^+]$ for the variable i, an aspect not fullled by the other methods that otherwise has to be dealt with a posteriori.

4. Fuzzy Recombination (FR). This operator, introduced in [11], basically replaces the uniform *pdf* (probability distribution function) by a bimodal one, where the two modes are located at x_i and y_i. The label "fuzzy" comes from the fact that the two parts $Pr_{xi}(t)$; $Pr_{yi}(t)$ of the probability distribution resemble triangular fuzzy numbers.

3.2 Mutation

A mutation operator Ψ is applied to each gene with some probability $Pr(\Psi) = 1/n$ so that, on average, one gene is mutated for each individual. Let $\vec{z} = (z_1, \ldots, z_n)$ denote the result of mutation of an individual $\vec{x}$. The elements of $\vec{z}$ are formed as follows:

1. Discrete Mutation (DM).

$$z_i = x_i + \text{sign} \cdot \text{range}_i \cdot \delta$$

with sign$\in \{-1, +1\}$ chosen with equal probability, $\text{range}_i = \rho(r_i^+ - r_i^-)$, $\rho \in [0.1, 0.5]$ and

$$\delta = \sum_{i=0}^{k-1} \varphi_i 2^{-i}$$

where $\varphi_i \in \{0,1\}$ from a Bernouilli probability distribution where $Pr(\varphi_i = 1) = 1/k$. In this setting $k \in N^+$ is a parameter originally related to the *precision* with which the optimum was to be located, a machine-dependent constant. In practice, however, the value of k is related to the *expected* value of mutation steps: the higher k is, the more fine-grained is the resultant mutation operator. The factor ρ is the *range ratio*, related to the *maximum* step that mutation is allowed to produce as a ratio of variable range.

2. Continuous Mutation (CM). Same as DM but with

$$\delta = 2^{-k\beta}$$

where $\beta \in [0, 1]$ with uniform probability.

4 THE FLETCHER-POWELL FUNCTION

This highly multimodal function was introduced in 1963 as a regression problem [12], where a collection of parameters have to be estimated such that a quadratic error term, depending on a non-linear expression, has to be minimized. Let us denote the n parameters as a vector $\vec{x}$. Let $A = (a_{ij})$, $B = (b_{ij})$ be two $n \times n$ real matrices and $\vec{\alpha}$ a fixed real vector of dimension n. Define the two terms A_i and B_i as:

$$A_i(\vec{\alpha}) = \sum_{j=1}^{n} (a_{ij} \sin\alpha_j + b_{ij} \cos\alpha_j)$$

$$B_i(\vec{x}) = \sum_{j=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos x_j)$$

Then, the function is defined as:

$$FP(\vec{x}) = \sum_{j=1}^{n} (A_i(\vec{\alpha}) - B_i(\vec{x}))^2$$

The solution is $FP(\vec{x}^*) = 0$, where $\vec{x}^* = \vec{\alpha}$. The function is clearly non-symmetric (which would make it easier) and non-separable (it cannot be optimized on each x_j separately). The sinusoidal factors lead to a high number of hills and valleys, making a gradient-based method unfeasible. There are up to 2^n extrema located in the subspace $|x_j| \leq \pi$, the one usually selected to constrain the solution space. The vector $\vec{\alpha}$ (the position of the absolute minimum) is chosen at random from this subspace $\alpha_j \in [-\pi, \pi]$. Moreover, the matrices A, B are also chosen at random with a_{ij}, $b_{ij} \in [-100, 100]$. This makes the suboptimal extrema to be randomly distributed over the search space. In our case, a value of n = 30 is chosen, so that the function includes a total of 1,830 random numbers.

5. EXPERIMENTAL RESULTS

An execution with the BGA involves the choice of a mutation operator Ψ and its parameters ρ and k and of a recombination operator Ω and its parameter δ (only for EIR). The truncation threshold τ and population size μ are also to be set. These last two values are of great importance because they are strongly interelated. For the sake of clarity and simplicity we do not discuss them here. Instead, an educated guess is made just in order to obtain reasonable results within low computing resources. To this end, a value of $\tau =$ 15 is set, to ensure a high selective pressure so as to enforce the quick discovery of fairly good solutions. The low value of entails a moderate-to-high population size in correspondence, to account for enough diversity. We set then $\mu = 100$.

The choice of genetic operators is more elaborated. To begin with, ρ is set to 0.5, a value that stands for high average mutation steps, needed to broad-tune rather than to rene solutions. In addition, a low value of k (lower than 16) is in favour of this scheme, and known to have a much deeper impact on performance [10] than , so we carried out some preliminary runs with $k \in \{6, 8, 10, 12, 16\}$ that lead to the selection of $k = 10$. In addition, the continuous operator showed to be better.

All this tuning runs take only a few seconds (always less than a minute) in a shared SUN™ Ultra-60 System. The stopping criterion is based on the number of function evaluations permitted (given by the vari-able `FFEvals`). In particular, given a nite number of `FFEvals`, the algorithm will stop each run whenever [`FFEvals`/μ] generations are reached. This stopping criterion allows to compare different general settings in a fair way, since, for example, a smaller population would be allotted more generations, but always keeping the number of evaluations in similar values. For each conguration, a number of independent runs are performed - denoted by `NRuns`- keeping track of the mean and best solutions found. In these initial experiments, `FFEvals`=50,000 and `NRuns`=5.

Regarding the recombination operator, ten possibilities are tested from within the set {DR, LR with α = 0.5, EIR with δ= 0 to 0.30, EIR with $range_\delta$ and FR}. Among them, the EIR operator quickly stands out over the rest, with varying performance that depends on its parameter δ. The results for this last operator are

presented in a single table for ease of reading -Table 1- summarizing the information as a function of the sample values for tested. For each conguration, average and best solutions found are shown.

EIR ()	Average	Best
0.05	51,541	18,296
0.10	28,837	23,151
0.15	25,368	2,265
0.20	32,459	11,798
0.25	41,262	14,017
0.30	47,356	20,561
$range_\delta$	25,953	9,030

Table 1: Results for the EIR (δ) recombination operator. Each entry shows the average and best results across five runs.

As it can be seen, EIR (0.15) shows to be the best setting, both on average and in terms of the best solution found. Note that the curve of performance across is concave, with EIR (0.15) at the bottom. The operator with the modication range shows to have a performance almost equal than the best one achieved with a fixed δ. This is remarkable because it means that a comparable performance can be obtained without the need of a search along . Both settings (δ = 0.15 and $range_\delta$) are selected for further experiments. It has to be said that the already obtained results are quite good. The initial function evaluations are in the order of millions (between one and five, depending on the run).

Once the algorithm has been roughly tuned to the problem landscape, a series of runs are performed to assess its potential to a deeper degree. Operators are then set to continuous mutation with ρ= 0.5, k = 10 for mutation, and the two choices EIR (δ = 0.15), EIR ($range_\delta$) for recombination, using the knowledge gained so far; this time, however, `NRuns=100` runs of `FFEvals=100`, 000 are carried out. The results are presented collectively in Table 2, along with some other results present in the literature.

Bäck [2] (p. 157), for example, reports results on this function with a standard genetic algorithm (SGA), an evolutionary programming algorithm (EP) and the powerful evolution strategies (ES), reproduced in the table for

Algorithm	Average	Best
SGA	$4.581 \cdot 10^4$	$1.032 \cdot 10^4$
EP	$1.107 \cdot 10^5$	$2.997 \cdot 10^4$
BGA-1	$1.987 \cdot 10^4$	$8.165 \cdot 10^2$
BGA-2	$9.238 \cdot 10^3$	$8.649 \cdot 10^2$
ES	$1.749 \cdot 10^3$	$3.190 \cdot 10^{-1}$

Table 2: Comparative results found by some evolu-tionary algorithms. BGA-1 is with EIR (0.15), and BGA-2 with EIR (range)

[2] Crossover is a form of recombination used in discrete GAs.

convenience. The first notable point is the inferior performance of the SGA and EP compared to the other algorithms. Eiben and van Kemenade [14] report also some results for a GA, product of their study on diagonal and n-point crossover[2] operators. They show these operators to be generally superior to traditional 2-parent, 1- or 2-point crossover. Although they do not provide numerical results, this improvement seems to make the GA vary between 30,000 and 10,000 on average, occasionally going below this mark. Altogether then, the binary GA is not likely to go beyond this order of magnitude, unless heavily modied or tailored to the problem.

With respect to the BGAs, both have improved their average performance, due to the double number of evaluations they have been allowed. Also noteworthy is the superior average performance of EIR (range) over EIR (δ=0.15). We believe that, in the long run, the first method is likely to widen the gap because of its adaptive nature.

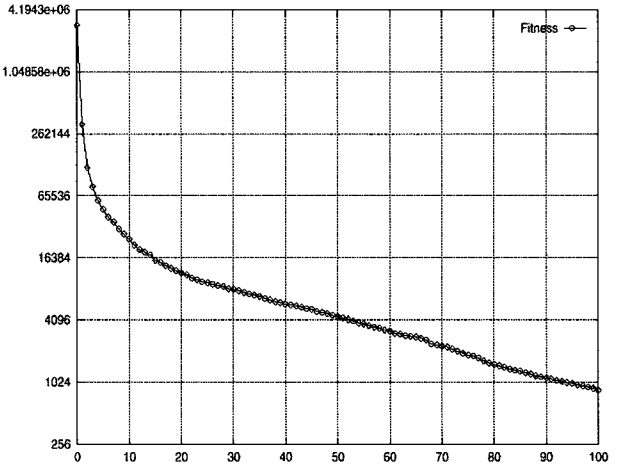

Figure 2: Performance curve for BGA-2 (EIR with range). The x-axis shows the number of generations (in multiples of ten). The y-axis shows the tness in a log2 scale. Note the quasilinear slope and the fact that further progress was still being achieved. The error decrease from the initial random guess to the final solution is in four orders of magnitude.

Note that, very roughly, both departing from similar average values, doubling the number of function evaluations has re ected in an average result for EIR ($range_\delta$) that is twice as good as that of EIR (δ = 0.15). However, the best absolute result is very similar for both, possibly indicating a limit in what can be achieved with the allowed resources and the selected μ, τ setting. A plot of the run nding the best solution for BGA-2 is shown in Figure 2.

The results reported by Bäck for the ES are by far the best, both on average and, especially, in the best solution found, which is excellent. This can be explained in several ways. First, ES [13] is a far more complicated algorithm that takes into account the interactions between variables more explicitly by selfoptimizing the amount of mutation

necessary. It is nonetheless very sensitive to its parameters so that tuning it is a more dicult task. However, once these are correctly set, ES performance is often superior to other algorithms. Second, it is interesting to note that the data for Bäck's experiments were obtained in sim-ilar conditions ($\mu = 100$, $\tau = 15$ and 100 runs) but resorting to 200,000 evaluations. The random initialization for the function is also a different one, which could well result in an overall easier or more complicated landscape. Bäck also reports the great variability found: of the 100 runs, only 44 were "success-ful runs", defined as those reaching a nal value under 400.0 (compare it with the best result obtained: 0.319). In our case, for example, the n deviation was computed to be 16792.8 for BGA-1, and 6563.7 for BGA-2. To see whether the BGA would be able to better its performance, we carried out a last experiment, a single run of BGA-2 with 200,000 evaluations. The result was 358.379, a value qualifying as successful.

6 CONCLUDING REMARKS

The quest for general-purpose search mechanisms is still an open and very active field. Since the 70's, new and powerful heuristic methods have emerged that are particularly well suited for function optimization -although this was not exactly their original purpose- mainly because of their generality, robustness, and conceptual (though not necessarily analytical) simplicity. Three of these methods are Simulated Annealing (SA), Tabu Search (TS) and Evolutionary Algorithms (EA). In the last decade, these properties and their remarkable successes have boosted their widespread use. The EA family is the biggest and subject of continuous improvement. In addition, many classical and modern problems are being reinstantiated and explored under the light of these methods. Articial neural networks are a good representative of this. In particular, the training process of a supervised feed-forward network can be easily cast as a function optimization problem.

In this paper we have given an impression of how different EAs can cope with a well-known task, one for which methods of non-linear optimization are prone to end up in local minima of the function. Among the former, the Breeder Genetic Algorithm (BGA) is relatively simple and yields reasonably good solutions in a very limited time. Note that although it has not been our intention to solve the task at hand, and only a small experiment setup has been devised for it, the quality of the solutions and the promise of better ones show the BGA (and EAs in general) as feasible alternatives for a great variety of tasks, ranging from the mentioned neural network training problem [15] to aerofoil design in Aerodynamics [16]. Work is in progress toward a thorough application of these methods in a principled way.

REFERENCES

[1] Pearl, J. Heuristics. Intelligent search strategies for computer problem solving. Addison-Wesley, 1984.

[2] Bäck, Th. Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York, 1996.

[3] Goldberg, D.E. Genetic Algorithms for Search, Optimization & Machine Learning. Addison-Wesley, 1989.

[4] Holland, J.H. Adaptation in natural and articial systems. The University of Michigan Press. Ann Arbor, MI, 1975.

[5] Mühlenbein, H., Schlierkamp-Voosen, D. Predictive Models for the Breeder Genetic Algorithm. Evolutionary Computation, 1 (1): 25-49, 1993.

[6] Whitley, D., The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In Procs. of the 3rd Intl. Conf. on Genetic Algorithms, (ed.) Scha er, J.D., Morgan Kaufmann: San Mateo, 116-121, 1989.

[7] De Falco, I., Del Balio, R., Della Cioppa, A., Tarantino, E. A Comparative Analysis of Evolutionary Algorithms for Function Optimisation. In Procs. of the Second Workshop on Evolutionary Computing (WEC2), Nagoya, JAPAN.

[8] Bäck, Th., Schwefel, H.P. An Overview of Evolutionary Algorithms for Parameter Optimization. Evolutionary Computation, 1 (1): 1-23, 1993.

[9] Heitkötter, J., Beasley, D. (Eds.). The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ). USENET: comp.ai.genetic. Available from anonymous FTP at rtfm.mit.edu, on /pub/usenet/news.answers/ai-faq/genetic.

[10] Belanche, Ll. A Study in Function Optimization with the Breeder Genetic Algorithm. LSI Research Report LSI-99-36-R. Dept. de Llenguatges i Sistemes Inform atics. Univ. Politecnica de Catalunya, 1999.

[11] Voigt, H.M., Mühlenbein, H., Cvetkovic, D. Fuzzy recombination for the continuous Breeder Genetic Algorithm. In Procs. of the 6th Intl. Conf. on Genetic Algorithms, (ed.) L. Eshelman, Morgan Kaufmann: San Mateo, 104-113, 1995.

[12] Fletcher, R., Powell, M.J.D. A rapidly conver-gent descent method for minimization. Computer Journal, 6:163-168, 1963.

[13] Bäck, Th. Evolution Strategies: An alternative Evolutionary Algorithm. Technical Report of the Informatik Centrum, Dortmund.

[14] Eiben, A.E., van Kemenade, C.H.M. Diagonal Crossover in Genetic Algorithms for Numerical Optimization. Journal of Control and Cybernetics, 26(3):447-465, 1997.

[15] Yao, X. A Review of Evolutionary Articial Networks. Intl. J. of Intelligent Systems, 8(4): 539-567, 1993.

[16] De Falco, I. An introduction to Evolutionary Algorithms and their application to the Aerofoil Design Problem - Part I: the Algorithms. Part II: the Results. Invited papers at the von Karman Lecture Series 1997, Bruxelles. Both pa-pers can be retrieved from the IRSIP Web Pages http://www.irsip.na.cnr.it/~hotg.