Evolutionary Algorithms: Some Very Old Strategies for Optimization and Adaptation

Thomas Bäck* Hans-Paul Schwefel†

University of Dortmund · Department of Computer Science
P.O. Box 50 05 00 · 4600 Dortmund 50 · Germany

Abstract

Genetic Algorithms and Evolution Strategies, the main representatives of a class of algorithms based on the model of natural evolution, are discussed w.r.t. their basic working mechanisms, differences, and application possibilities. The mechanism of self-adaptation of strategy parameters within Evolution Strategies is emphasized and turns out to be the major difference to Genetic Algorithms, since it allows for an on-line adaptation of strategy parameters without exogenous control.

1. Introduction

Within nearly 20 years of research it has been demonstrated that the search process of natural evolution can yield very robust search algorithms on a computer, although the imitations used are crude simplifications of the natural model. Several of these *Evolutionary Algorithms* are nowadays broadly accepted, the most important ones being the *Genetic Algorithm* (GA) and *Classifier System* (CFS) by Holland[7, 8], the *Evolution Strategy* (ES) by Rechenberg and Schwefel[11, 12], and the *Genetic Programming Paradigm* (GPP)[3, 10].

Each of these approaches is based upon the collective learning process within a population of individuals, each of which represents a search point in the space of potential solutions to a given problem. The start population is usually initialized at random. The population is able to adapt to a given environment by means of randomized processes of selection, recombination, and mutation. The environmental feedback delivers a quality information (*fitness*), and the selection process favours those individuals of higher quality to reproduce more often than worse individuals. Recombination allows for exchanging information between individuals, and mutation

*baeck@ls11.informatik.uni-dortmund.de
†uin005@ddohrz11.bitnet

introduces new information into the population. Altogether, during the evolution process the average quality of the population increases, hopefully leading to an optimal point.

This informal description leads to the rough outline of a general Evolutionary Algorithm given below:

$t = 0$;
initialize$(P(t))$;
evaluate$(P(t))$;
while not *terminate*$(P(t))$ **do**
    $t = t + 1$;
    $P(t) = $ *select*$(P(t-1))$;
    recombine$(P(t))$;
    mutate$(P(t))$;
    evaluate$(P(t))$;
od

Here t denotes the generation number, and $P(t) = (a_1, \ldots, a_\lambda)$ is the population at generation t, consisting of λ individuals $a_1, \ldots, a_\lambda$. At first glance, the main difference between the Evolutionary Algorithms is given by the complexity of the search space I the individuals are taken from ($a_i \in I$). GAs are working on binary strings, ESs on vectors of real values, CFSs on production system rules which are encoded over a ternary alphabet, and the GPP manipulates computer programs written in syntactically simple programming languages.

This paper focuses on GAs and ESs, which are the most frequently used and best understood instances of Evolutionary Algorithms.

2. Genetic Algorithms

Genetic Algorithms were mainly developed by John Holland at Ann Arbor, Michigan, during the 60ies and early 70ies. The basic theoretical investigations are given by Holland[7], and an implementation and application to parameter optimization was done in the same year by De Jong[9].

GAs work on binary strings of a fixed length l, i.e. $I = \{0,1\}^l$. The fitness function $f : I \to \mathbb{R}$ evaluates the quality of individuals. Although this way GAs seem to be restricted to optimization of pseudoboolean functions, there is a simple way to apply them also to continuous parameter optimization problems of the form $f' : \times_{i=1}^{n} [u_i, v_i] \to \mathbb{R}$. This is possible by using decoding functions $\Gamma^i_{l_x} : \{0,1\}^{l_x} \to [u_i, v_i]$, which map segments of length l_x to the intervals $[u_i, v_i]$. Typically, such a function is of the form $\Gamma^i_{l_x}(\alpha_1, \ldots, \alpha_{l_x}) = u_i + (v_i - u_i) \cdot \left(\sum_{j=1}^{l_x} \alpha_j 2^{j-1} \right) / (2^{l_x} - 1)$, and the total length of the individual is $l = n \cdot l_x$, where l_x determines the accuracy of the decoded value and may be different for the segments of an individual.

When all individuals of a population have been evaluated, the selection operator $s : I^\lambda \to I^\lambda$ selects a new population by taking λ probabilistically sampled copies from the old population. The sampling probabilities p_s of the individuals are given by their relative fitness, i.e. $p_s(a_i) = f(a_i)/\sum_{j=1}^{\lambda} f(a_j)$ (*proportional selection*).

This mechanism will obviously fail in case of negative fitness values or minimization tasks. Due to these reasons, proportional selection is typically used in combination with fitness *scaling* techniques, which map objective function values $f'(a_i)$ to fitness values $f(a_i)$. The most commonly used *linear scaling* technique adjusts the objective function values linearly according to the worst individual, resulting in the transformation $f(a_i) = -\min\{f'(a_j) \mid a_j \in P(t)\} + f'(a_i)$ in case of a maximization task (for minimization: $f(a_i) = \max\{f'(a_j) \mid a_j \in P(t)\} - f'(a_i)$)[4].

After selection, the frequencies of better individuals have increased at the expense of worse individuals, but no new search points are created so far. Recombination and mutation are used to achieve this.

For recombination $\omega_c : I \times I \to I$, which exchanges information between individuals, a large amount of *crossover* operators is described in the literature (e.g. see[4]). Here, we will only describe the *one point crossover* operator, which works on two individuals $a_1 = (\alpha_1 \ldots \alpha_l)$ and $a_2 = (\beta_1 \ldots \beta_l)$ by randomly choosing a crossover position $k \in \{1, \ldots, l-1\}$ and exchanging information behind that position, resulting in two new individuals $a_1' = (\alpha_1 \ldots \alpha_k \beta_{k+1} \ldots \beta_l)$ and $a_2' = (\beta_1 \ldots \beta_k \alpha_{k+1} \ldots \alpha_l)$. This operator is applied with a fixed, exogenously given probability p_c ($\approx 0.6 - 0.9$).

Finally, with a probability p_m (≈ 0.001) the mutation operator is applied to the bits of the individuals. If a bit position is selected for mutation, its value is simply inverted. This is usually seen as a 'background operator' which allows to recover from converged bits, i.e. bit positions having the same value throughout the population.

The essentials of GA-theory are derived by viewing at a GA as an algorithm which processes *schemata*. A schema $H \in \{0, 1, *\}^l$ is a description of a similarity template or hyperplane in l-dimensional bit space. *Instances* of a schema H are all bitstrings $a \in \{0, 1\}^l$ which are identical to H in all positions where H has a value of 0 or 1 (e.g. $H = (01*1*)$ has four instances: (01010), (01011), (01110), (01111)). Let $m(H^t)$ denote the number of instances of a schema H^t in the population $P(t)$. Furthermore, $\sigma(H)$ denotes the *order* of H, i.e. the number of fixed positions, and $\delta(H)$ denotes its *defining length*, i.e. the distance between its first and last fixed position (e.g. $\sigma(0**1*) = 2$, $\delta(0**1*) = 3$). Using the *average schema fitness* $f(H^t) = \frac{1}{m(H^t)} \sum_{a_i \in H^t} f(a_i)$ of schema H^t in population $P(t)$ and the average fitness $\bar{f}^t$ of $P(t)$, the schema growth equation for proportional selection turns out to be $m(H^{t+1}) = m(H^t) \cdot f(H^t)/\bar{f}^t$. Under the assumption that H^t is above average, i.e. $f(H^t) = \bar{f}^t + c\bar{f}^t$ with a constant value of $c > 0$, after t time steps starting with $t_0 = 0$, we obtain: $m(H^t) = m(H^0) \cdot (1+c)^t$. This means, that proportional selection allocates exponentially increasing (decreasing) numbers of trials to above (below) average schemata.

So far, recombination and mutation are not incorporated into the analysis. This is done by calculating the survival probabilities of a schema H^t under simple crossover (which is $1 - p_c \cdot \delta(H^t)/(l-1)$) and mutation ($(1-p_m)^{\sigma(H^t)}$), resulting in the *Schema*

Theorem of GAs:

$$m(H^{t+1}) \;\geq\; m(H^t) \cdot \frac{f(H^t)}{\bar{f}^t} \cdot \left(1 - p_c \frac{\delta(H^t)}{l-1}\right)(1-p_m)^{\sigma(H^t)} \tag{1}$$

The schema theorem states, that short, low-order, above-average schemata (*building blocks*) receive exponentially increasing trials in the following generations. The choice of a binary alphabet for encoding maximizes the number of schemata processed by a GA and therefore supports the hyperplane sampling process. Building blocks and their combination to form longer and longer useful substrings are the most important working mechanism of a GA. Therefore, consideration of the schema theorem and the building block hypothesis are the main design criteria for applying a GA to a certain problem[4].

3. Evolution Strategies

According to Rechenberg[11], the first efforts towards an Evolution Strategy took place in 1964 at the Technical University of Berlin. The experimental applications to parameter optimization at that time dealt with hydrodynamical problems like shape optimization of a bended pipe and of a flashing nozzle. The algorithm used was a simple mutation-selection scheme called *two membered* ES. An ES works on real-valued vectors $\vec{x} \in \mathbb{R}^n$, and in case of the two membered ES an offspring individual is created from a single parent individual by means of adding normally distributed random vectors with expectation zero and standard deviation σ (the same σ is used for all components of $\vec{x}$). The better (w.r.t. fitness) of both individuals becomes the ancestor for the next iteration. Such an algorithm is usually abbreviated as (1+1)–ES, indicating that selection chooses the best individual of (one) parent and (one) offspring to survive.

For this algorithm, which can be seen as a probabilistic gradient search technique, Rechenberg calculated the convergence rates for two different objective function topologies, leading to a theoretically supported rule for step size control[11]. These objective functions are the *linear corridor* of width b, i.e. $f_1(\vec{x}) = c_0 + c_1 x_1$ where $\forall i \in \{2, \ldots, n\} : -b/2 \leq x_i \leq b/2$, and the *sphere model* $f_2(\vec{x}) = \sum_{i=1}^{n} x_i^2$. For both functions the expressions derived for the maximum convergence rate φ_{max} and the corresponding optimal standard deviation σ_{opt} are of the form $\varphi_{max} \propto \rho/n$, $\sigma_{opt} \propto \rho/n$, where ρ is a topology parameter (corridor width b for f_1, actual distance r from the origin for f_2).

Calculation of the corresponding optimal success probabilities p_{opt} yields $p_{opt} \approx 0.184$ for f_1 and $p_{opt} \approx 0.270$ for f_2, which form the basis of Rechenberg's 1/5 *success rule*[11]: *The ratio of successful mutations to all mutations should be* 1/5. *If it is greater, increase; if it is less, decrease the standard deviation* σ.

Schwefel[12] gives reasons to use the multiplicative factors 0.82 and 1/0.82 for the adjustment of σ, which should take place every n mutations. The frequency of successful mutations should be measured e.g. over intervals of $10 \cdot n$ trials.

So far, the algorithm achieves linear convergence for the sphere model, and the 1/5 success rule increases efficiency at the cost of robustness[12]. The rule may lead the (1+1)–ES to premature termination even in the case of unimodal functions if there are sharp ridges or active restrictions. Furthermore, only one standard deviation σ is used for all object variables, i.e. no individual adaptation of the σ_i or scaling for the object variables x_i is possible this way.

The *multimembered* ES variants (μ+λ)–ES and (μ,λ)–ES incorporate both the idea of a population (and therefore recombination) as well as the idea of *self-adaptation* of strategy parameters. The notation (μ+λ)–ES indicates μ parents which create $\lambda \geq 1$ offspring individuals by means of recombination and mutation. The μ best individuals out of parents *and* offspring are selected to form the next population. For a (μ,λ)–ES with $\lambda > \mu$ the μ best individuals are selected from the λ offspring only. Although at first glance the possibility of a (μ,λ)–strategy to forget the currently best solution seems silly, it enables the algorithm to escape from local optima, to follow a moving optimum, to deal with noisy objective functions, and to self-adapt strategy parameters effectively (see below). Concerning the convergence rate and without recombination, the (1,λ)–ES–strategy was investigated theoretically for the corridor and sphere model, resulting in an optimal ratio of $\mu/\lambda \approx 1/5$ to achieve maximum convergence rates[12].

An individual is now represented as a vector $a = (x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n) \in \mathbb{R}^n$, consisting of n object variables and their corresponding n standard deviations for individual mutations. For mutation, each x_i is mutated by adding an individual, $(0, \sigma_i)$–normally distributed random number. The σ_i themselves are also subject to mutation and recombination, and a complete mutation step $m(x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n) = (x'_1, \ldots, x'_n, \sigma'_1, \ldots, \sigma'_n)$ is formalized as follows:

$$\begin{aligned} s &= \exp(N(0, \tau)) \\ \sigma'_i &= \sigma_i \cdot \exp(N_i(0, \tau')) \cdot s \\ x'_i &= x_i + N_i(0, \sigma'_i) \end{aligned} \tag{2}$$

Mutation is performed on the σ_i by multiplication with two log-normally distributed factors, one individual factor, sampled anew for each σ_i ($\tau' = 1/\sqrt{2\sqrt{n}}$), and one common factor s ($\tau = 1/\sqrt{2n}$), sampled once per individual.

This way, a scaling of mutations along the coordinate axes can be learned by the algorithm itself, without an exogenous control of the σ_i. The objective function $f_3(\vec{x}) = \sum_{i=1}^{n} i \cdot x_i^2$ provides a test for the self-adaptation of standard deviations, since the optimal σ_i are all different and can be calculated in advance ($\sigma_i = c/\sqrt{i}$), thus allowing for an experiment which compares self-adaptation to optimal preadjusted settings of the σ_i relations. As a measure of progress, the expression $\log \sqrt{f^t/f^0}$ is used, where f^t denotes the best objective function value in generation t. For (μ,100)–ESs figure 1 shows the progress achieved by different strategies, depending on the selective pressure, i.e. the setting of μ.

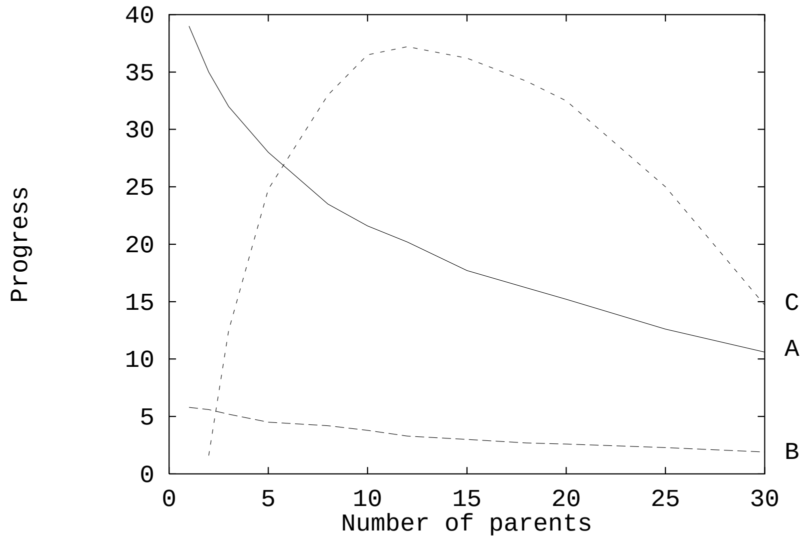

Figure 1: Comparison of convergence rates.

Curve A shows the behaviour of a strategy without self-scaling, i.e. with fixed σ_i relations such that only one common step size has to be learned. Here a (1,100)–ES fulfills the expectation to perform best. Curve B corresponds to non-optimal σ_i relations and otherwise identical conditions. When individual self-adaptation of the σ_i and recombination are allowed as shown in curve C, the progress of a (15,100)–ES comes very close to that of the perfect (1,100)–ES variant A and is surprisingly better than a (15,100)–ES with correctly preadjusted σ_i-relations. This is a striking example for a kind of *synergetic effect*, where 15 imperfect, but different individuals perform collectively better than the same number of cloned specialists.

Learning of strategy parameters beyond the scaling along coordinate axes can be achieved by allowing the hyperellipsoids of equal probability density for placing an offspring to rotate in n-dimensional space, i.e. to adjust the orientation according to the local ascent direction. This can be achieved by introducing *correlated mutations* via incorporating the covariances of the standard deviations into the individuals. Then, up to $n(n-1)/2$ additional strategy parameters are added to the genotype, and mutation and recombination are extended w.r.t. the covariances[1].

Recombination in ESs with $\mu > 1$ is always done on the whole parent population. The standard mechanisms are *discrete* and *intermediate* recombination. In discrete recombination the components of two parents are selected at random from either the first or the second parent to form an offspring individual, while in intermediate recombination the components of an offspring have values somewhere between the corresponding components of the parents. Both operators can also be extended to their *global* form, where one parent is selected and fixed, and for each component of the offspring a second parent is chosen anew from the population to determine the components' values. Preliminary investigations led to best results for discrete recombination of the object variables and intermediate recombination for strategy parameters.

4. Conclusions

In the previous sections the basic working mechanism of both GAs and ESs turned out to be identical, while the main differences of the algorithms are identified in the representation of individuals (binary / real-valued), the selection mechanism (proportional / (μ,λ), $(\mu+\lambda)$), the role of mutation (background operator, bit inversion / main search operator, normally distributed random numbers), and the self-adaptation of strategy parameters (only within ESs).

The last topic in this list makes up a major difference, since it introduces learning on two different levels into the algorithm, i.e. on the usual level of the object variables and on the level of strategy parameters[6]. This is a promising way to overcome the difficulty of determining appropriate parameter settings of an Evolutionary Algorithm for each application task anew.

Evolutionary Algorithms have confirmed their robustness and effectivity in a great variety of real world applications as well as research problems, including such different tasks as optimization of neural networks (topology and weights), job shop scheduling and sequencing problems, dynamic control, game theory, data analysis, and computer aided optimal design of many technical products (for a list of EA-applications see[2]).

The importance of these algorithms will increase further with the dissemination of massively parallel hardware, since besides performing objective function evaluations in parallel by using a master-slave approach, the population can also easily be split up coarse-grained into parallel subpopulations or fine-grained into parallel individuals. Such parallel implementations are scalable concerning their total population size to arbitrary numbers of parallel processing elements and provide an elegant way to exploit the parallel hardware to solve increasingly harder problems[5].

5. References

1. T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications*, pages 2–9, University of California, San Diego, USA, 1991. Morgan Kaufmann Publishers.
2. T. Bäck, F. Hoffmeister, and H.-P. Schwefel. Applications of evolutionary algorithms. Report of the Systems Analysis Research Group (LS XI) SYS–2/92, University of Dortmund, Department of Computer Science, 1992. In print.
3. M. L. Cramer. A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 183–187, Hillsdale, New Jersey, 1985. Lawrence Erlbaum Associates.
4. D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, 1989.
5. F. Hoffmeister. Scalable parallelism by evolutionary algorithms. In M. Grauer and D. B. Pressmar, editors, *Parallel Computing and Mathematical Optimization*, volume 367 of *Lecture Notes in Economics and Mathematical Systems*, pages 177–198, Berlin, 1991. Springer.
6. F. Hoffmeister and T. Bäck. Genetic self–learning. In *Proceedings of the First European Conference on Artificial Life, December 11-13*, Paris, France, 1991. The MIT Press. In print.
7. J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.
8. J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems.* Academic Press, New York, 1978.
9. K. D. Jong. *An analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975. Diss. Abstr. Int. 36(10), 5140B, University Microfilms No. 76–9381.
10. J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan, editor, *Eleventh international joint conference on artificial intelligence*, pages 768–774. Morgan Kaufmann Publishers, August 1989.
11. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann–Holzboog Verlag, Stuttgart, 1973.
12. H.-P. Schwefel. *Numerical Optimization of Computer Models.* Wiley, Chichester, 1981.