EVOLUTIONARY COMPUTATION

Th. Bäck
Leiden Institute of Advanced Computer Science, Leiden University,The Netherlands

Keywords: adaptation, evolution strategy, evolutionary programming, genetic algorithm, genetic programming, optimization, self-adaptation.

Contents

Summary

This contribution summarizes the field of evolutionary computation, i.e., computational methods for search and optimization gleaned from the model of organic evolution. The main classical branches of evolutionary computation, namely genetic algorithms, evolution strategies, evolutionary programming and genetic programming, are presented

in a unified way by discussing the structure of individuals and the typical evolutionary operators mutation, recombination, and selection for each of the methods. Furthermore, some of the most important theoretical results on genetic algorithms and evolution strategies are presented, and the application possibilities of evolutionary computation are outlined.

1. A General Evolutionary Algorithm

Evolutionary algorithms mimic the process of organic evolution, the driving process for the emergence of complex and well adapted organic structures. At a simplified level, evolution can be seen as the result of the interplay between the creation of new genetic information and its evaluation and selection. A single individual of a population is affected by other individuals of the population (e.g., by food competition, predators, and mating), as well as by the environment (e.g., by food supply and climate). The better an individual performs under these conditions the greater its chance to live for a longer while and generate offspring, which in turn inherit the (disturbed) parental genetic information. Over the course of evolution, this leads to a penetration of the population with the genetic information of individuals of above-average fitness. The non-deterministic nature of variation leads to a permanent production of novel genetic information and therefore to the creation of differing offspring. The following structure of a general evolutionary algorithm reflects on a high level of abstraction all essential components of standard implementations of evolutionary algorithms:

Algorithm 1

```
$t := 0$;
initialize $P(t)$;
evaluate $P(t)$;
while not terminate do
    $P'(t) := select1(P(t))$;
    $P''(t) := variation(P'(t))$;
    evaluate($P''(t)$);
    $P(t+1) := select2(P''(t) \cup Q)$;
    $t := t+1$;
od
```

The classical instances of evolutionary algorithms, namely genetic algorithms, evolution strategies, evolutionary programming and genetic programming, can all be described in the conceptual framework of the above pseudocode formulation. The following general features, however, are common to all evolutionary algorithms and can therefore be seen as the defining properties of evolutionary computation:

- $P(t)$ denotes a population (a multiset, i.e., multiple copies of individuals are possible) of a certain number μ of individuals (candidate solutions to a given problem) at generation (iteration) t of the algorithm. μ is called parent population size in the following.

- The initialization at $t = 0$ can be done randomly, or with known starting points obtained by any method.
- The evaluation of a population involves calculation of its members quality according to the given fitness function (i.e., a quality criterion such as an objective function f: $M \rightarrow \Re$ in case of an optimization task, assigning a quality value $f(\vec{x})$ to solution candidates $\vec{x} \in M$, where M denotes the search space of the optimization problem).
- The variation operators include the exchange of partial information between individuals (so-called recombination or crossover operators) and the typically small, random variation of single individuals (so-called mutation operators).
- By means of the variation operators, an offspring population $P'(t)$ of λ candidate solutions is generated. λ is called offspring population size in the following.
- Selection operators can be applied for selecting the intermediate population $P'(t)$ before any variational operators are applied, for selecting the new parent population from the offspring population $P''(t)$, or for both purposes. The operator select1 plays the role of a kind of mating selection, acting on the individuals prior to their involvement in recombination and mutation operators, while select2 can be interpreted as environmental selection, acting on the offspring of a population.
- Concerning the settings of μ and λ, no special assumptions are made except $\mu \geq 1$, $\lambda \geq 1$. If $\lambda = 1$ (only a single offspring is created, evaluated and substituted within $P(t)$ at each generation), the algorithm is sometimes called a steady-state algorithm. If $\lambda \leq \mu$, only the worst fraction $\gamma = \lambda/\mu$ of the parent population $P(t)$ is replaced at each generation. The fraction γ is usually called the generation gap. If $\lambda > \mu$, an offspring surplus is created and the environmental selection operator select2 is utilized to reduce the population size again to μ individuals.
- Q is a special set of individuals that might be considered by the select2 selection operator, e.g., $Q = P(t)$ if $\gamma < 1$ (but $Q = \emptyset$ is possible as well).
- The algorithm terminates if no more improvements are achieved over a number of subsequent iterations or if a given amount of time is exceeded.
- The algorithm returns the best (according to the quality criterion) individual ever found during its execution or the best individual from the last generation of the run.

In the following, we will describe the mainstream instances of this general evolutionary algorithm in their standard forms. A large number of further variations have been developed in the past decade, especially by means of exchanging operators between the standard instances of evolutionary algorithms, by developing new operators, and by applying evolutionary algorithms to new search spaces. These variations cannot be discussed in this paper, and the interested reader is referred to the Handbook of Evolutionary Computation (see references) for further information.

2. Classical Genetic Algorithms

Referring to the general evolutionary algorithm outline as given in algorithm 1, the classical genetic algorithm is characterized by the following properties:

- Individuals are represented as binary vectors of fixed length l, i.e., $\vec{x} \in \{0,1\}^l$.

- In case of the so-called generational replacement, offspring and parent population sizes are identical ($\lambda = \mu$), $P(t+1) := P''(t)$ (there is no environmental selection), and select1 (mating selection) is the only selection operator.
- A generation gap $\gamma < 1$ including the steady-state case $\gamma = 1/\mu$ is sometimes used as an alternative to generational replacement.
- Classical genetic algorithms do not use $\lambda > \mu$. The main emphasis is put on mating selection.
- Crossover occurs in various instantiations and acts as main variation operator, while mutation is of secondary importance and acts as a background operator.

In the following sections, these components of classical genetic algorithms are discussed in detail.

2.1 The Structure of Individuals

Canonical genetic algorithms use a binary representation of individuals as fixed-length strings over the alphabet {0,1}, such that they are well suited to handle pseudoboolean optimization problems of the form

$$f: \{0,1\}^l \rightarrow \Re . \tag{1}$$

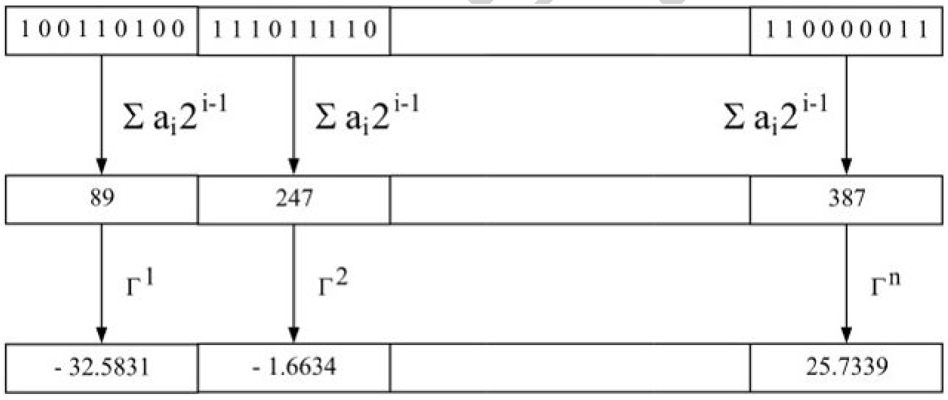

Figure 1: Decoding process used in canonical genetic algorithms for continuous search spaces. Γ^i denotes the linear mapping of an integer value $k \in \{0,\ldots,2^{l'}-1\}$ to the interval $[u_i,v_i]$, i.e.,

$$\Gamma^i = u_i + [(v_i - u_i)/(2^{l'}-1)] \cdot k$$

Sticking to the binary representation, genetic algorithms often enforce the utilization of encoding and decoding functions $h: M \rightarrow \{0,1\}^l$ and $h': \{0,1\}^l \rightarrow M$ that facilitate mapping solutions $\vec{x} \in M$ to binary strings $h(\vec{x}) \in \{0,1\}^l$ and vice versa, which sometimes requires rather complex mappings h and h'. In case of continuous parameter optimization problems, for instance, genetic algorithms typically represent a real-valued vector $\vec{x} \in \Re^n$ by a binary string $\vec{y} \in \{0,1\}^l$ as follows: the binary string is logically divided into n segments of equal length l' (i.e., $l = n \cdot l'$), each segment $(x_1 \ldots x_{l'})$ is decoded to yield the corresponding integer value $\sum_{i=1}^{l'} x_i 2^{i-1}$, and the integer value is in turn linearly mapped to the interval $[u_i, v_i] \subseteq \Re$ (corresponding to the i-th segment of the binary string) of real values. Figure 1 illustrates this decoding process for string segments of length $l' = 9$ (allowing for the representation of the integers $\{0,1,\ldots,511\}$), which are mapped by means of h' to the interval $[-50,50]$.

Presently, a Gray code interpretation of the binary string is often used for decoding purposes. The main advantage of a Gray code is seen in the fact that it maps Euclidean neighborhoods into Hamming neighborhoods due to the representation of adjacent integer values by binary strings with Hamming distance one (i.e., they are different by one bit only). Theoretical and experimental investigations strongly support this point of view.

2.2 Mutation

Mutation in genetic algorithms was introduced by Holland as a dedicated ``background operator" of small importance. Mutation works by inverting bits with very small probability such as $p_m = 0.001$, $p_m \in [0.005,0.01]$, or $p_m = 1/l$. Recent studies have impressively clarified, however, that much larger mutation rates, decreasing over the course of evolution, are often helpful with respect to the convergence reliability and velocity of a genetic algorithm, and that even so-called self-adaptive mutation rates are effective for pseudoboolean problems.

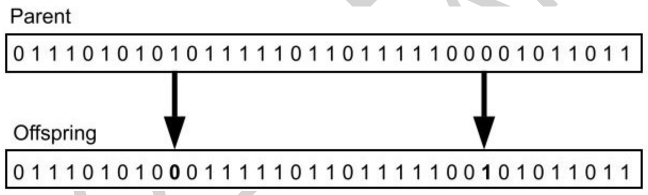

Figure 2: Mutation operator used in canonical genetic algorithms, with a bit inversion occurring at two random positions within the parent individual

The mutation operator is illustrated in figure 2. In this example, two bits of the parent individual are inverted by mutation. In general, for $p_m = 1/l$, one bit on average is expected to mutate per individual, but in principle also multiple mutations are possible (with exponentially decreasing probability, however). It should be noticed that this is an important property of the mutation operator, because changing multiple bits at the same time at least in principle facilitates the algorithm to escape from local optima - even if the probability of this to happen might be vanishingly small.

2.3 Recombination

The variation operators of canonical genetic algorithms, mutation and recombination, are typically applied with a strong emphasis on recombination. The standard algorithm performs a so-called one-point crossover, where two individuals are chosen randomly from the population, a position in the bitstrings is randomly determined as the crossover point, and an offspring is generated by concatenating the left substring of one parent and the right substring of the other parent.

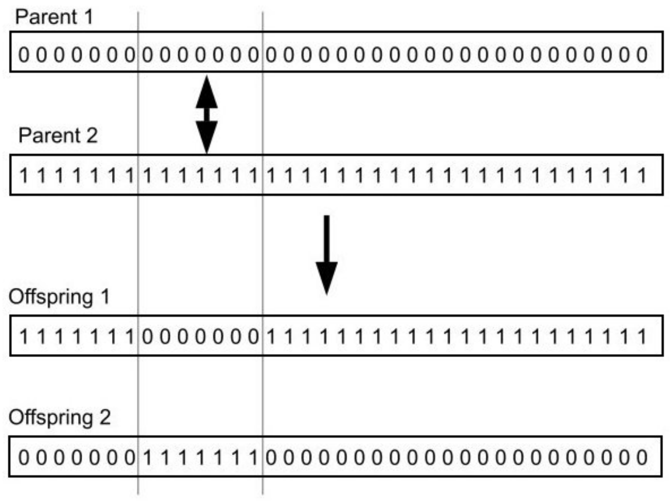

Figure 3: Two-point recombination operator used in canonical genetic algorithms. The two recombination positions are chosen randomly

Numerous extensions of this operator, such as increasing the number of crossover points, uniform crossover (each bit is chosen randomly from the corresponding parental bits), and others have been proposed, but similar to evolution strategies no generally useful recipe for the choice of a recombination operator can be given. The theoretical analysis of recombination is still to a large extent an open problem. Recent work on *multi-parent recombination* , where more than two individuals participate in generating a single offspring individual, clarifies that this generalization of recombination might yield a performance improvement in many application examples. Unlike evolution strategies, where it is either utilized for the creation of all members of the intermediate population (the default case) or not at all, the recombination operator in genetic algorithms is typically applied with a certain probability p_c, and commonly proposed settings of the crossover probability are $p_c = 0.6$, and $p_c \in [0.75,0.95]$.

An example of two-point crossover is given in figure 3. The recombination points marked by vertical lines are chosen randomly, and the offspring is created by exchanging the segment limited by the two crossover points between the parents, thus creating two offspring. Typically, one of the offspring individuals is randomly selected to be chosen for the next generation, while the other one is discarded.

2.4 Selection

In genetic algorithms the mating selection operator select1 is typically implemented as a probabilistic operator, using the individual selection probabilities

$$p(\vec{a}) = f(\vec{a}_i) / \sum_{j=1}^{\mu} f(\vec{a}_j) \qquad (2)$$

calculated as relative fitnesses of the individuals. This method - called proportional selection - requires positive fitness values and a maximization task, so that *scaling* *functions* are often utilized to transform the fitness values accordingly. Rather than using absolute fitness values, *rank-based selection* methods utilize the indices of individuals when ordered according to fitness values to calculate the corresponding selection probabilities. Linear as well as nonlinear mappings have been proposed for this type of seletion operator. Another alternative called *tournament selection* works by taking a random uniform sample of a certain size $q > 1$ from the population, selecting the single best of these q individuals to survive for the next generation, and repeating the process until the new population is filled. This method gains increasing popularity because it is easy to implement, computationally efficient, and allows for fine-tuning the selective pressure by increasing or decreasing the tournament size q. While most of these selection operators have been introduced in the framework of a generational genetic algorithm, they can also be used in combination with the steady-state and generation gap methods and of course in combination with the other branches of evolutionary computation.

3. Evolution Strategies

In terms of the general evolutionary algorithm, an evolution strategy is characterized by the following main distinguishing features:

- Individuals are represented as real-valued vectors, consisting of object variable vectors $\vec{x} \in \Re^n$ plus some additional information, the so-called strategy parameters.
- No mating selection is used, i.e., $P'(t) = P(t)$.
- Assuming $\lambda >> \mu$, the so-called (μ,λ)-selection operator (with $Q = \varnothing$) deterministically chooses the μ best solutions from $P''(t)$ to become $P(t+1)$. Alternatively, the $(\mu+\lambda)$-evolution strategy selects the μ best solutions from the union of $P'(t)$ and $P(t)$ (i.e., $Q = P(t)$) for creating $P(t+1)$.
- The mutation operator is implemented by means of normally distributed variations based on adaptable step sizes (i.e., variances and covariances of the normal distribution). Mutation is the main variation operator, while recombination plays only a secondary role.
- Variances and covariances of the mutation operator are strategy parameters which are part of the individuals. These strategy parameters are themselves optimized during the search according to a process called *self-adaptation* , a second-order learning process working on the parameters of the evolution strategy.

As before, the components of classical evolution strategies are discussed in the following sections.

3.1 The Structure of Individuals

For a given optimization problem

$$f : M \subseteq \Re^n \to \Re , \quad f(\vec{x}) \to \min \tag{3}$$

an individual of the evolution strategy contains the candidate solution $\underline{x} \in R^n$ as one part of its representation. Furthermore, there exist a variable amount (depending on the type of strategy used) of additional information, so-called *strategy parameters* , in the representation of individuals. These strategy parameters essentially encode the *n*-dimensional normal distribution which is to be used for the variation of the solution.

More formally, an individual $\vec{a} = (\vec{x}, \vec{\sigma}, \vec{\alpha})$ consists of up to three components $\vec{x} \in \Re^n$ (the solution), $\vec{\sigma} \in \Re^{n_\sigma}$ (a set of standard deviations of the normal distribution), and $\alpha \in [-\pi, \pi]^{n_\alpha}$ (a set of rotation angles representing the covariances of the *n*-dimensional normal distribution), where $n_\sigma \in \{1,\dots,n\}$ and $n_\alpha \in \{0,(2n-n_\sigma)\cdot(n_\sigma-1)/2\}$. The exact meaning of these components is described in more detail in section 3.2 .

-
-
-

Bibliography

Congress on Evolutionary Computation, CEC 1999, CEC 2000. Proceedings of the Congress on Evolutionary Computation. IEEE Press, Piscataway, NJ, 1999, 2000. [Proceedings volumes of a conference series that emerged 1999 from mainly the IEEE International Conference on Evolutionary Computation and the Conference on Evolutionary Programming. Held every year.]

D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995. [An overview of the three main paradigms of evolutionary algorithms, with some emphasis on machine intelligence.]

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, MA, 1989. [A textbook on classical genetic algorithms, focusing on the schema theory approach to explain the power of genetic algorithmns.]

G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kova\'cs, Hamburg, 1997. [A unified theoretical investigation of evolutionary algorithms, with a strong focus on convergence properties such as convergence reliability and convergence velocity.]

Genetic and Evolutionary Computation Conference, GECCO 1999, GECCO 2000. Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, San Francisco, CA, 1999, 2000. [Proceedings volumes of a conference series that emerged 1999 from mainly the International Conference on Genetic Algorithms and the Genetic Programming Conference. Held every year.]

H.-G. Beyer. *The Theory of Evolution Strategies*. Natural Computing Series, Springer, Berlin, 2000. [An in-depth theoretical investigation of evolution strategies.]

H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series. Wiley, New York, 1995. [A textbook on modern evolution strategies, with an empirical comparison between evolution strategies and a large set of traditional optimization methods.]

H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26 of *Interdisciplinary Systems Research*. Birkhäuser, Basel, 1977. [The classical textbook from the inventor of (μ,λ)-evolution strategies.]

I. Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Frommann-Holzboog, Stuttgart, 1994. [A textbook on modern evolution strategies, including various issues such as nested evolution strategies, noisy environments and theoretical investigations. Contains also a reprint of Rechenberg 1973.]

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973. [The classical book on (1+1)-evolution strategies, from their inventor.]

J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975. [The classical book on genetic algorithms, also more generally called adaptive plans by the author.]

J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Complex Adaptive Systems. The MIT Press, Cambridge, MA, 1992. [The first book on genetic programming, by its inventor.]

L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966. [The classical book on evolutionary programming in its original form for evolving finite state machines.]

M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. The MIT Press, Cambridge, MA, 1999. [Presents an analysis of the classical genetic algorithm by means of dynamical systems theory.]

M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, MA, 1996. [A more recent textbook on genetic algorithms.]

Parallel Problem Solving from Nature Conference, PPSN 1990+2. Proceedings of the Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science vols. 496, 866, 1141, 1498. Springer, Berlin. [Proceedings volumes of the main European Conference on Evolutionary Computation. Held every second year.]

Th. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997. [The big reference handbook on evolutionary computation, with in-depth presentations of the algorithms, of theoretical results, and of practical examples.]

Th. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996. [An overview of the three main paradigms of evolutionary algorithms, using a unified formal presentation of the algorithms. Includes also some experimental comparison and theoretical results.]

W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers, San Francisco, CA, 1998. [An introduction to many variations and state-of-the-art technologies in genetic programming.]

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, 1996. [An overview of evolutionary computation, describing many different variations of the basic scheme. Widely used as a textbook.]

Biographical Sketch

Thomas B"ack received the Diploma degree in Computer Science in 1990 and the Ph.D. degree in Computer Science in 1994, both from the University of Dortmund, Germany. In 1995, he received the best dissertation award of the German Association for Computer Science (GI) for his Ph.D. thesis on evolutionary algorithms.

From 1990-1994, he worked as a Scientific Assistant at the Department of Computer Science of the University of Dortmund. From 1994-1996, he was Senior Research Fellow at the Center for Applied Systems Analysis within the Informatik Centrum Dortmund, and Department Director of the Center for Applied Systems Analysis from 1996 - 2000.

Since March 1999, he is also Managing Director of divis digital solutions GmbH, and since January 2000 CTO of NuTech Solutions, Inc. and Managing Director of the German branch of NuTech Solutions (former divis digital solutions GmbH). The company specializes in optimization and data mining applications of computational intelligence technology for industrial problems.

Since 1996, Dr. B"ack also serves as an Associate Professor in the Computer Science Department of Leiden University, The Netherlands, and teaches courses on evolutionary computation and systems analysis at Leiden University.

His current research interests are in the areas of theory and application of evolutionary computation and related areas of computational intelligence, as well as in DNA computing.

He is author of the book Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms New York, NY: Oxford University Press, 1996 , co-editor-in-chief of the Handbook of Evolutionary Computation (New York, NY: Oxford University Press Institute of Physics Publishing, 1997), member of the editorial board of Evolutionary Computation Cambridge, MA: The MIT Press , and associate editor of the IEEE Transactions on Evolutionary Computation. He is also a member of the editorial board of the Natural Computing Series.

Dr. B"ack is a member of the IEEE and the Dutch Association for Theoretical Computer Science (NVTI), serves on the IEEE Neural Networks Council's technical committee on evolutionary computation since 1995, was a co-program chair of the 1996 and 1997 IEEE International Conferences on Evolutionary Computation (ICEC) and the Fifth Annual Conference on Evolutionary Programming (EP '96), program chair of the Seventh International Conference on Genetic Algorithms and Their Applications (ICGA '97), and co-chair of the fifth International Conference on Parallel Problem Solving from Nature (PPSN V).